ON CHEMICAL AND SELF-HEALING NETWORKING PROTOCOLS

Inauguraldissertation

zur Erlangung der Würde eines Doktors der Philosophie
vorgelegt der Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel

von

Thomas Meyer

aus Muttenz BL, Schweiz

Basel, 2011

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag von

Prof. Dr. Christian Tschudin
Prof. Dr.-Ing. Markus Fidler
PD Dr. Peter Dittrich

Basel, den 14. Dezember 2010

Prof. Dr. Martin Spiess (Dekan)

Meinen Eltern, in Dankbarkeit und Liebe

Only those who attempt the absurd
will achieve the impossible.
I think it's in my basement…
let me go upstairs and check. [1]

M. C. Escher
(1898–1972)

Abstract

In this thesis, we look at networking protocols through the eyes of a chemist. By introducing an artificial chemistry for networking, we obtain an intrinsically dynamic, reaction-based view to packet processing. Our chemical virtual machine lets packets react with each other akin to chemical molecules. Simple reaction rules at the microscopic level translate to well-defined behavior on the macroscopic flow level. Because these two descriptions are linked by laws from chemical kinetics, we are able to apply tools from chemistry to predict the behavior of chemical networking protocols and even proof dynamic convergence properties.

Based on this principle, we develop an engineering model to design and analyze chemical protocols. We demonstrate the feasibility and usefulness of our approach with several new solutions to application scenarios ranging from a gossip-style aggregation protocol, over an enzymatic MAC protocol, to a chemical TCP-like congestion control algorithm, which ensures the coexistence and fairness among chemical and classical packet flows in the Internet.

The chemical reaction model has additional properties that are hard to achieve on traditional code execution platforms: We show how protocol software is able to continuously regenerate itself in order to exhibit intrinsically self-healing properties; our approach is based on self-replicating code and natural selection. We present a self-healing multipath routing protocol that is resilient to the removal of large parts of its own code.

With this work, we try to contribute to the future Internet by discovering the self-regulating capabilities of packet flows, which currently lie dormant.

Keywords: network protocol, artificial chemistry, emergent computation, packet flow balance, self-healing code

Funding: This work is supported by the Swiss National Science Foundation (SNF) through grant 2000201–109563, project "Self-Healing Protocols", which is thankfully acknowledged.

Acknowledgments

This thesis would not have been possible without the help and support of many people. I owe them my sincere gratitude.

First, I would like to thank my adviser *Prof. Christian Tschudin*, who had the original idea of using the chemical metaphor for networking and who made this work possible. I thank him for supervising this study with critical insights and appreciate the brainstorming sessions where he always surprised me with novel ideas.

Special thanks go to *Lidia Yamamoto* who co-supervised this project and from whom I learned a lot about natural and artificial chemistry and evolution. She was always generous in sharing her insights an discussing scientific and other topics.

I am grateful to *Prof. Markus Fidler* and *PD Peter Dittrich* for being in the thesis committee. Many thanks to the researchers in Jena for hosting us during a very pleasant week. I also thankfully acknowledge the contributions of *Prof. Wolfgang Banzhaf* to the energy model.

Philip Lüscher, Yvonne Mathis, and *Massimo Monti* also contributed to this work with their bachelor and master theses; I appreciate the effort they spent. I'm grateful also to the careful readings and the valuable comments the following people have given to drafts of this manuscript: *Ghazi Bouabene, Igor Talzi, Manolis Sifalakis, Marcel Lüthi,* and *Pierre Imai.*

I wish to thank all my colleagues at the department with whom I had a great time in the past years. I also thank my friends and former colleagues at Inalp – the best engineering team ever.

Finally, I wish to thank my parents, my brother and my sisters for their great confidence in me and their love; and many thanks to my brothers in law for their moral support.

Chapters

IV DISCUSSION

Contents

PART I INTRODUCTION, BACKGROUND & RELATED WORK

PART II
CHEMICAL NETWORKING PROTOCOLS (CNPS)

PART III
SELF-HEALING NETWORKING PROTOCOLS

PART IV
DISCUSSION

PART I

Introduction, Background & Related Work

CHAPTER 1

Introduction

1.1 Context and Motivation

The original aim of the Internet, namely providing a standardized way of reliably exchanging information between spatially distributed and architecturally different computers, has been fulfilled. We can assume that information sent from one application to the other is delivered reliably and on time. Hence, research interest shifted towards higher communication layers: the World Wide Web endowed users with location-independent linkage of information, the Web 2.0 enabled interactions with automatic services and with other users, Service Oriented Architectures (SOA) allow for the interaction of machines over well-defined interfaces.

Under the hood, the Internet is still based on the same principles and protocols as in the beginning: The Internet Protocol (IP) provides a global addressing scheme and structures information exchange into a sequence of packets; the Transmission Control Protocol (TCP) ensures that packet streams reliably cross the Internet and adapts the transmission rate to maximize throughput and ensure fairness; routing protocols automatically adapt to topological changes and support data packets in finding their way through the network.

However, the seemingly perfect service offered by the network's core is threatened by recent and future developments, including the ever-growing complexity and the potential decrease of reliability of future computing infrastructure.

The Complexity Problem: Due to the big success of the Internet, the overall traffic currently grows at an annual growth rate of 30 % to 40 %. Mobile data undergoes an even larger growth of up to 100 % per year (Cisco, 2010). With the advent of Internet-enabled mobile devices, more and more computers are and will be attached to the network. This growth is a challenge for the current, mostly manually administrated infrastructure. Autonomic protocols exist for certain areas, for example routing and address assignment, but infrastructure management is far from being autonomic: A lot of human intervention is still needed to maintain the overall service quality. The labor cost to maintain the IT infrastructure by far exceeds the overall cost for equipment.

The Reliability Problem: Another trend we observe is the complexity and miniaturization of computing machinery. Nowadays, ubiquitous mobile devices are full-fledged general-purpose computers requiring more and more processing power and energy. Integrated circuits are reaching a critical package density. When increasing the package density further, or when reducing the operating voltage to save energy, logic gates will occasionally fail to switch correctly. Thus, the computing service that future hardware platforms can provide will likely be unreliable.

These trends and the accompanied problems in networking led to our research objectives stated as

> *Organize information exchange and coordination tasks in packet networks such that autonomous protocols can be designed easily.*

and

> *Organize code execution such that communication software is still able to provide a reliable service on unreliable hardware.*

A method that addresses both objectives would be important for the future development of complex computer networks and their ubiquitous accessing devices. Especially the second objective is hard to address with traditional software engineering methods: Code is executed sequentially and is not able to validate its own correctness without relying on hardware mechanisms. This led us to the study of natural systems and how they achieve robustness and self-healing properties.

1.2 CONTRIBUTIONS

In this work we are looking at networking protocols through the eyes of a chemist. By transposing chemical concepts to computer networks we obtain a reaction and flow-based view to packet processing. The consistent implementation of chemical laws leads to protocols with similar emergent properties than found in chemical systems, such as for example robustness to perturbations and self-organization. In particular, this thesis makes the following original contributions:

- We introduce a novel **"chemical" execution engine** for networking protocols that treats packets as molecules and lets them react with code in virtual reaction vessels. The "molecules" span a distributed reaction network that may reach over the entire computer network and that performs a distributed computation.

- We put strong emphasis on the dynamical aspect of protocol execution and propose a **chemical packet scheduler**, which organizes packet processing according to laws from chemical reaction kinetics.

- This allows us to adopt analysis tools from chemistry to **predict the dynamics of protocols** by studying the distributed reaction network and come up with elegant **convergence proofs**.

- We provide sound **engineering methods** to design and analyze chemical networking protocols. In particular, we offer a couple of well-understood reaction network **motifs** that can be combined in a bottom-up approach to synthesize new protocols.

- In this chemical engineering framework, we also **model basic network components**, such as simple queues and links, and **implement common networking motifs**, such as source routing, traffic shaping, neighbor discovery, anycast, gossiping, and link load balancing.

- We show how chemistry suggests novel solutions for established protocols by introducing an **enzymatic** shared **medium access control** scheme, which allocates bandwidth more smoothly than existing MAC protocols.

- We demonstrate that chemical and classical protocols may coexist in the Internet. Therefore, we propose **a chemical congestion control algorithm (C3A)** that balances molecular packet flows and ensures fairness to TCP streams competing for bandwidth-limited links.

- We make protocols **more robust to packet loss** and achieve a **strong embodiment** to the network by exploiting rate-based information exchange, which results naturally from a reaction-centric view of communication. We demonstrate this concept for ***Disperser*, a gossip-style aggregation protocol**.

- We show how protocol software is able to continuously regenerate itself in order to exhibit intrinsically self-healing properties; our approach is based on self-replicating code and natural selection. We present a **self-healing multipath routing protocol** that is resilient to the removal of large parts of its own code.

- We **quantify the robustness** of protocol software by analytically computing its mean survival time and by carrying out empirical studies based on simulations. Our results show that self-maintaining molecule ensembles with as little as 20 replicas easily endure perturbations for billions of years.

- In order to increase the robustness of software running on unreliable hardware, we propose to integrate an **energy conservation scheme** to the chemical execution model.

1.3 APPROACH

Chemical reaction networks are similar to computer networks as both organize flows of matter or packets, which adapt to environmental changes. The main motivation for mimicking chemistry to structure communication protocols is the robustness of bio-chemical systems to all kinds of perturbations: Redundant pathways protect a metabolism from the knock-out of a participating enzyme; chemical reaction networks form organizations in which every part is at least produced by another part. In such a self-regulating reaction system, there is no dedicated controlling instance. Thus, "control" is an *emergent* phenomenon.

emergent control

1.3.1 INTRODUCTORY EXAMPLE: THE SELF-HEALING BOARD GAME

The seemingly complex but actually simple phenomenon of emergent self-organization is best introduced with the example of a board game. In their book *Das Spiel*[*] (1975), Eigen and Winkler used simple *board games* to illustrate how in nature, emergent behavior arises from simple rules. We follow Eigen by sketching the prototype of an intrinsically self-healing system as simple game:

Consider a checkerboard with two types of tokens, white and black, as depicted in Figure 1.1. In each round, apply a simple rule: Randomly select two places and compare the color of the tokens found there. If they are identical, do nothing. If the colors are different, create a copy of each token and let

[*] An english translation of the book is available (Eigen, Winkler, Kimber, & Kimber, 1993).

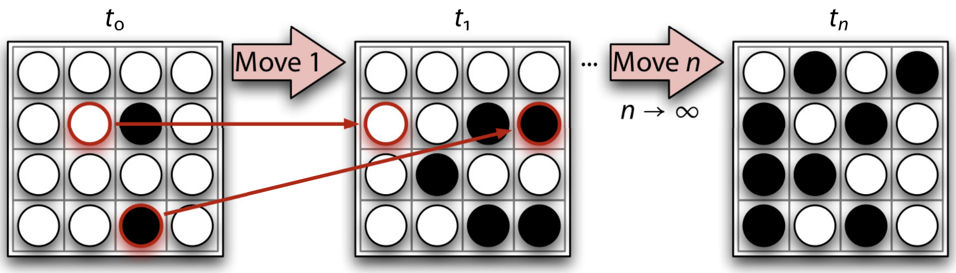


Figure 1.1 Mate-and-spread game: A checkerboard is initialized with only two black and 14 white tokens. After some rounds, the number of white and black tokens mutually approach each other.

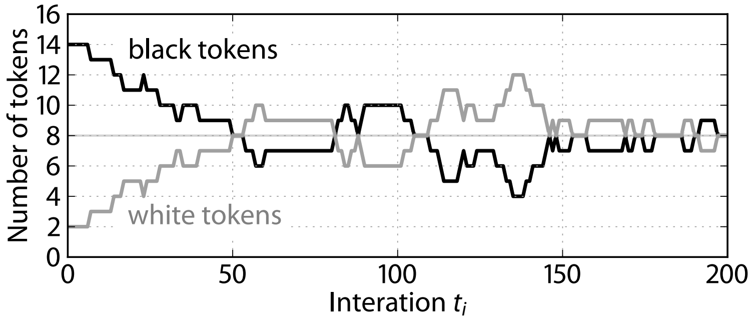


Figure 1.2 Example time trace of the mate-and-spread game: The game strives to equilibrium where black and white tokens are present with the same quantity.

them drop at random places, replacing any previously present token color. Then start the next round. We call this the "mate-and-spread game".

If each color is present at least once in the start configuration, the distribution of the tokens will most likely be uniform after a certain time: Independent on the initial distribution, this game strives to *equilibrium* where black and white tokens are present with the same quantity as shown in Figure 1.2.

equilibrium

The microscopic process of a move in this game is random (random selection and random replacement) and we cannot predict the outcome of the next round. However, at the macroscopic level we observe the emergence of an equilibrium.

Now imagine that the passive white and black tokens are instead two active software components. Instead of a central player picking tokens and replacing them if needed, tokens react with each other. Once they mate, they replicate and, if necessary, displace other software components in turn. At equilibrium, both parts will be present with an equal number of copies. If something happens to one instance of either part, for example, if a fault occurs such that it is not able to mate anymore, the erroneous copy will eventually be displaced by the remaining healthy parts: the overall software is *self-healing* without requiring a central controller.

self-healing

Such an intrinsically software-only self-healing mechanism is hardly possible in traditional software execution environments where instructions are executed sequentially. But if we put software into a chemical setting, where each code part is represented as a molecule that is able to react with other molecules in parallel, such self-organization of code suddenly becomes possible.

1.3.2 A CHEMICAL APPROACH TO COMMUNICATION PROTOCOLS

Self-healing is only one of the properties that result from such a chemical-like reaction model. If chemical reaction networks are arranged correctly, they strive to equilibrium where all forces are balanced, like in the board game, where eventually, replication and deletion forces are equal.

distributed chemical reaction networks

We aim at designing networking protocols as *distributed chemical reaction networks* that present their solutions at equilibrium. Perturbations in the network, such as packet loss or competing packet flows, will automatically bring the system back to the fixed point of optimal operation. The challenge is to express the problem chemically and to design a reaction network such that at equilibrium, the system behaves correctly according to the requirement specification. With the help of several examples we will demonstrate that this is feasible for most continuously running distributed algorithms, such as for example routing, load balancing, and consensus protocols.

Beyond the inherent robustness of chemical networking protocols, the chemical approach offers a unified model for protocol execution and analysis. That is, the dynamic behavior of a given protocol implementation can be analyzed mathematically. At the same time, a designed abstract reaction network can be transposed quickly to a concrete implementation in a chemical programming language.

In this thesis, we study both the use of a chemical model as an engineering framework to design, execute, and analyze robust communication protocols and the use of self-organizing reaction networks to obtain self-healing protocol code that is able to run on unreliable hardware.

1.4 STRUCTURE OF THE THESIS

As depicted in Figure 1.3, this thesis is organized in four parts: Part I (Chapters 1 to 3) gives background information in computer networks and artificial chemistry. Part II first introduces the fundamentals of our chemical networking theory in Chapters 4 and 5. The remaining chapters of the second part (Chapters 6 to 12) study the approach in more detail. In particular, we show how to analyze and design chemical networking protocols and discuss issues related to the integration of a chemical virtual machine into current hardware and computer networks. In Part III (Chapters 13 to 18), we indicate how the

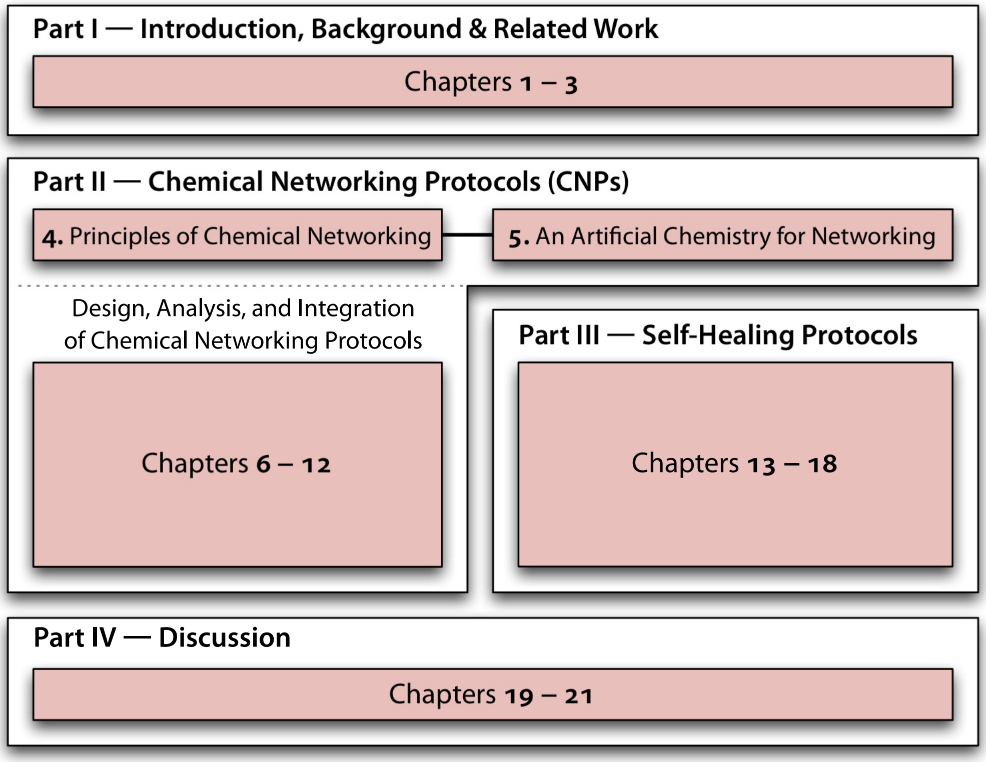

Part I — Introduction, Background & Related Work
Chapters 1 – 3
Part II — Chemical Networking Protocols (CNPs)
4. Principles of Chemical Networking
5. An Artificial Chemistry for Networking
Design, Analysis, and Integration of Chemical Networking Protocols
Chapters 6 – 12
Part III — Self-Healing Protocols
Chapters 13 – 18
Part IV — Discussion
Chapters 19 – 21

Figure 1.3 Structure of the thesis in four parts

chemical execution model gives rise to self-healing protocol software that tolerates deletion or mutation of parts of its own code base. Finally, part Part IV (Chapters 19 to 21) concludes this thesis by discussing the relevance and impact of our approach.

Audience (a)

This document is written with a broad audience in mind. We combine methods from two major research fields, namely formal and applied computer networking and (artificial) chemistry; but the thesis also touches areas in complex systems and evolutionary dynamics. We introduce all concepts thoroughly, having in mind that an expert in one area will need more background material in the other fields. Some chapters, written for a dedicated audience, contain skip marks in the introduction to guide the reader through the document. The first and fourth part is recommended to all readers. The second part mainly addresses researchers in computer networks whereas the third part on self-healing protocols may attract researchers from artificial life.

computer scientists

Computer scientists with a theoretical background are referred to Chapters 6 to 8, where we provide structural and dynamical analysis methods to prove properties of chemical protocols, and to Chapter 14 where we quantify the robustness of self-healing code.

Model / Example	Section	Page
Mate-and-spread game (self-healing board game)	Section 1.3.1	6
Catalytic bimolecular reaction (equilibrium)	Section 3.3.3(c)	34
Distributed equilibrium	Section 5.1.1(c)	54
Non-deterministic finite state machine	Section 5.3.1	66
Source routing	Section 5.3.2	67
***Disperser*: Gossip-style aggregation protocol**		
Simulation:	Section 5.4	68
Analysis:	Chapter 9	139–151
Design Variants:	Section 10.4	178–183
Packet forwarding	Section 7.1	85
Distributed arithmetic computation	Section 10.3.2	164–170
Enzymatic traffic shaping/limiter	Section 10.3.3(c)	173
Neighborhood discovery	Sections 10.3.4(a), 10.3.4(b)	175–176
Anycast	Section 10.3.4(d)	177
Chemical queue	Section 12.1.1	201
Chemical model of a bandwidth-limited link	Section 12.1.2	203
Enzymatic Medium Access Control (mac)	Section 12.1.3	206
C_3A: Chemical congestion control algorithm	Sections 12.3.2–12.3.4	230–237
C_3A^+: Congestion avoidance extension	Sections 12.3.5, 12.3.6	237–242
Self-healing distributed software updates	Section 15.3	291–294
Self-healing multipath routing protocol	Chapter 16	297–314

Table 1.1 **Examples and application cases**: Three major application cases (bold) are used to demonstrate properties of chemical protocols or the chemical engineering framework, the others illustrate discussed concepts.

computer network engineers

For *computer network engineers*, we provide engineering methods to design chemical protocols in Chapter 10; Chapter 12 illustrates how chemical and classical protocols cooperate in the Internet. There are a vast number of protocol examples to illustrate the principles of chemical program design and analysis: Table 1.1 lists all examples and application cases.

computer system architects

Computer system architects may be more interested in the architecture of the chemical virtual machine, which executes chemical protocols. Chapter 11 shows how a chemical execution engine can be integrated to existing computer and networking infrastructure.

natural scientists

Finally, *natural scientists* interested in the self-organizational properties of living systems may be attracted to see an application case for biological and chemical principles in computer networking. They are referred to the third part, where we study the behavior (Chapters 14 and 15) and robustness (Chapter 17) of self-healing code and adopt an energy model from nature to promote efficient and robust programs (Chapter 18).

NOTATIONAL CONVENTIONS 1.5

In this work, we used a few notational conventions listed below. An extensive list of all mathematical symbols used throughout this thesis can be found in the List of Symbols in the back matter on page 415.

Scalar variables are represented by lowercase Roman or Greek letters, $a \dots z$, or $\alpha, \dots, \omega$, vectors by lowercase Roman bold face letters, $\mathbf{a}, \dots, \mathbf{z}$, whereas we use uppercase Roman bold face letters for matrices, $\mathbf{A}, \dots, \mathbf{Z}$. Sets are typeset in a calligraphic font, $\mathcal{A}, \dots, \mathcal{Z}$, except for $\mathbb{N}$ (natural numbers), $\mathbb{Z}$ (integers), $\mathbb{R}$ (real numbers). Multisets over sets are written as $\mathcal{M}(\mathcal{A}), \dots, \mathcal{M}(\mathcal{Z})$, power sets (set of all subsets) as $\wp(\mathcal{A}), \dots, \wp(\mathcal{Z})$. The cardinality (number of elements) of a set $\mathcal{A}$ or a multiset $\mathcal{M}(\mathcal{A})$ is written as $|\mathcal{A}|$ and $\left|\mathcal{M}(\mathcal{A})\right|$, respectively. Molecule types (species) are represented by upright uppercase Roman letters, $\mathrm{A}, \dots, \mathrm{Z}$.

CHAPTER 2

Networking Paradigms

On the principles of current communication networks — an overview.

> Will the technologies of communication and culture help us to understand one another better, or will they deceive us and keep us apart? [2]
>
> Roger Waters

Electronic communication started with telegraph and telephone networks. Such connection-oriented networks establish a physical connection between the participants for the duration of a communication session. Later, Paul Baran and Donald Davies developed packet-switching networks in the 1960ies (Baran, 1964), which ultimately lead to the Internet, as we know it today. In the meantime, other communication patterns such as active networking appeared but never penetrated the world in the same way the data-packet-centric Internet did.

In this chapter we review the landscape of networking paradigms. We skip circuit-switched networks and immediately start by discussing the packet-switched paradigm in Section 2.1. Then, motivated by the possibility of our chemical model to exchange code on the fly, we give a short introduction to the principles of active networking in Section 2.2. Finally, also related to our approach, we discuss the benefit of stochastic protocols in Section 2.3.

2.1 PACKET-SWITCHED NETWORKS

There is no doubt that the Internet is the most successful packet-switched network. In packet-switched networks, there is no direct physical connection between two communicating endpoints. Information is split in chunks, which are transferred as data packets over shared physical links. But often, a *virtual circuit* hides the packet nature of the network from the user. Transport protocols such as the Transmission Control Protocol (TCP) (Postel, 1981) or the Stream Control Transmission Protocol (SCTP) (Ong & Yoakum, 2002; Stewart, 2007) establish a virtual connection between Internet applications. Such connection-oriented protocols also exist below the transport layer: X.25 (ITU, 1996) was developed as core protocol for the public data networks of the 1980ies (Compuserve, Euronet, etc.) and is still used for signaling in the ISDN D-channel. Frame Relay and ATM also use the concepts of virtual paths and virtual connections. Recently, virtual circuits below the network layer became popular again with MPLS (Rosen, Viswanathan, & Callon, 2001). Multi Protocol Label Switches classify packets into streams and manage them as such, for example by ensuring quality of service guarantees on the stream level.

virtual circuit

In this section, we review two aspects of packet-switched networks: In Section 2.1.2, we compare the role of the network's core to the role of end systems. In Section 2.1.2, we then discuss packet-switched networks from two different points of view, namely the microscopic view on the individual packets and the macroscopic view on packet flows through the network.

2.1.1 NETWORK CORE AND END SYSTEM

The core of a packet-switched network consists of several routers or switches. They forward a data packet to its destination and dynamically allocate the necessary bandwidth of a shared physical link by means of *statistical multiplexing*. Packets are sent asynchronously over the link in the order of their arrival. Since the link may be busy at the time a new packet arrives, packets are stored into FIFO (first in / first out) queues. This so called *store and forward* method afflicts the traversing packets with a delay, which is proportional to the current fill level of the queue.

statistical multiplexing

store and forward

FIFO ordering does not ensure fairness among different packet streams. Other scheduling disciplines such as fair queuing divide the arriving packets into individual packet streams, for example, one stream per source/destination address pair, and assign each stream an individual queue. The queues are then scheduled, for example, in a weighted round-robin fashion such that each stream obtains the same bandwidth share.

The Internet's design philosophy is to put most of the protocol's logic into the end stations whereas the core of the network shall operate stateless (*end-to-end principle*). Because the network itself does not provide fairness among the competing packet streams, transport protocols such as TCP have to control their own packet transmission rate to avoid congestion.

end-to-end principle

MICROSCOPIC STATE MACHINES AND MACROSCOPIC FLOWS 2.1.2

There are two levels of granularity at which we can look at protocols and the packets they exchange: a microscopic and a macroscopic level. At the *microscopic level*, a protocol implementation cares about the individual packets: the format of their header, the actions triggered by a received packet, timers that have to be set, etc. Protocols are implemented as state-machines where an asynchronous event such as an arriving packet or an elapsed timer triggers an action and a state transition.

microscopic level

At the *macroscopic level*, we look at packet streams or flows. The content of an individual packet is not important anymore; we focus on the large-scale dynamic behavior of a protocol. Often, a protocol is being developed at the functional, microscopic level first, ignoring its dynamical behavior, or analyzing it at a later time. An example of this approach is TCP: Only after a series of "congestion collapses" in 1986, it was recognized that TCP misbehaved on this dynamical level, because it did not throttle its transmission rate in order to ensure fairness to other packet streams competing for the same limited bandwidth resources. Jacobson (1988) then proposed a very successful congestion-control mechanism for TCP.

macroscopic level

Several analysis tools operate on this macroscopic level: *Queuing theory* was developed in the early 20th century by Erlang (1917) (see also Bose, 2002, and F. P. Kelly, 1979, for example) as a mathematical framework to determine the capacity requirements of circuit-switched telephone networks. Queuing theory models traffic (telephone calls, but also data packets) as stochastic processes and allows for the analysis of statistical properties, such as the expected waiting time or packet loss. A whole packet network is modeled as a *network* of queues where the output of one queue is fed as input to the next queue (Jackson, 1963; F. P. Kelly, 1976, 1979).

queuing theory

Instead of using a stochastic model, Chang (2000) as well as Le Boudec and Thiran (2001) developed a deterministic model of a queuing network based on min-plus algebra (Baccelli, Cohen, Olsder, & Quadrat, 1992). They use their *network calculus* to estimate the worst-case performance of packet flows through components, such as links with limited capacity, traffic shapers, etc. A deterministic treatment greatly simplifies the analysis of large queuing networks. However, the worst-case assumption ignores that statistical

network calculus

multiplexing can interleave packet streams and actually save bandwidth. Recently, Fidler (2006) developed a probabilistic network calculus, which takes statistical multiplexing into account (see also Jiang & Liu, 2008). It models packet streams and network components as stochastic processes and uses their moment-generating functions to derive performance characteristics of the network.

2.2 ACTIVE NETWORKING

Active networking refers to a communication paradigm in which packets carry instructions in addition to data; such active code is executed in the routers along the packet's path. This allows packets to dynamically modify the operation of the underlying core network.

messenger paradigm

Tschudin (1993) proposed his *messenger paradigm*, which "replaces the exchange of data values by the exchange of instructions". The goal is to speed-up protocol development and deployment: Protocols of traditional data packet networks have to be standardized. This is mandatory if two protocol implementations shall "understand" the syntax and semantics of the exchanged data packets. The messenger paradigm avoids the cumbersome standardization process by sending the code required to interpret information along with the information.

strong active networks

network processors

Tennenhouse and Wetherall (1996) describe active networks from an architectural level. They propose a distinction between two flavors: strong and moderate active networks (see also Sterbenz, 2002). *Strong active networks* is the flavor proposed by Tschudin (1993) where the *user* injects messengers (also called capsules) that are executed by the *network processors* along the packet's journey through the network. This paradigm causes performance and security concerns: Each capsule has to carry the code needed for interpreting the embedded data. Since many or all data values of a certain stream are treated in the same way, the same code is sent multiple times. The security problem results from the fact that code is remotely executed on distrusted hosts and that on the other hand the host should be protected from malicious user-supplied code. Sander and Tschudin (1998) proposed an encryption method that allows the hosts to execute encrypted code directly without needing a decoded form.

moderate active networks

Moderate active networks avoid the problems of strong active networks by only allowing the network provider to update its forwarding hardware with new code. This is a more controlled way of active networking, but does not offer the new flexibility to the end-user.

We advocate the paradigm of strong active networks and mobile code. Today, where everyone seeks to virtualize hardware, it is interesting that protocol execution is still static. Active networking would allow for separate sandboxes in which users run their own forwarding code, for example. We agree with Tennenhouse, Lampson, Gillett, and Klein (1996) that

> [...] active networks present an opportunity to change the structure of the networking industry, from a "mainframe" mind-set, in which hardware and software are bundled together, to "virtualized" approach in which hardware and software innovation are decoupled.

Our chemical networking paradigm extends the concept of active networking by a special, *stochastic* packet scheduling scheme. On the next pages we therefore address the purpose of randomness in communication protocols.

STOCHASTIC PROTOCOLS 2.3

Stochastic protocols differ from traditional protocols in the probabilistic nature of certain decision processes. Unlike stochastic modeling methods of deterministic protocols, such as queuing theory, stochastic protocols deliberately add some noise to their very *actions* in order to speed-up convergence time or to make the protocol robust to environmental noise.

Many stochastic approaches to communication have been proposed, from randomized consensus protocols, which are able to tolerate erroneous participants (Bracha & Toueg, 1985; Aspnes, 2003), to probabilistic algorithms that schedule access to a wireless medium in an energy-efficient way (Chlamtac, Petrioli, & Redi, 1997). We can only provide a limited overview of the field here: In Section 2.3.1 we discuss gossip protocols, which stochastically select their communication peers; many distributed problems have been solved within this framework. Later, in Section 2.3.2, we focus on a specific application case, namely packet routing, and review two different stochastic packet forwarding approaches.

GOSSIP PROTOCOLS 2.3.1

Gossip protocols mimic the epidemic spreading of gossip in a social network by periodically sending small pieces of information to one or multiple randomly selected neighbors.

Gossip protocols work according to the following simple scheme: Each node (1) periodically selects n peers from its set of neighbors, (2) sends its state to the selected neighbors and receives the state form the neighbors in turn. (3) Finally, an *update* function merges the local state with received states. All existing gossip protocols use this principle and vary in how (1) they select the node, (2) what the state represents, and (3) how the *update* function is implemented. Note that only the first step, namely the selection of the peers is usually randomized; the *update* function aggregates the states deterministically.

Gossip protocols have been proposed for different applications: The first gossip approach was an epidemic algorithm to maintain mutual consistency in a replicated database, proposed by Demers et al. (1987). Another application of gossiping is information *aggregation* in large networks. Aggregation refers to functions that provide the nodes access to global information as a summary of local information kept in the individual nodes (Jelasity, Montresor, & Babaoglu, 2005). This information can be related to the network itself, such as network size, average load, average uptime, or it can be an aggregation of environmental information such as the average, minimum, or maximum of measured temperatures in a Wireless Sensor Network (WSN) (Kempe, Dobra, & Gehrke, 2003), for example. In order to compute the distributed average with the above simple scheme, each node (1) selects one random peer, (2) sends its estimate s_1 to the peer and receives the peer's estimate s_2, and (3) computes a new estimate: $s_1 \leftarrow (s_1 + s_2)/2$.

aggregation

Jelasity, Montresor, and Babaoglu (2009) proposed a gossip protocol for *topology management*. Their T-MAN protocol builds overlay networks with regular topologies from scratch by randomly exchanging node descriptors to which the nodes shall connect.

topology management protocol

Gossip protocols are interesting because consensus is achieved based on the random exchange of messages. The system asymptotically computes the result and reaches *equilibrium*, where updates do not alter the state anymore. The number of messages needed to reach consensus is relatively small compared to a deterministic algorithm. That is, gossip protocols converge faster and require less messages, especially if the network is large.

2.3.2 STOCHASTIC ROUTING

From the various sub domains of stochastic protocols, we pick routing as an example, because we develop an own stochastic and chemical routing protocol in Chapter 16. A routing protocol has to collect topological information from the network and guides data packets towards their destination. Therefore, each router disseminates information about its local connectivity and collects and aggregates knowledge about the connectivity of its neighbors. Stochastic routing protocols usually start by sending real data or special probe packets randomly through the network. If a packet reaches the destination, an acknowledgment packet travels back to the sender and updates the states of all routers along the packet's path to reinforce the good path. The next data packet then randomly selects the next hop based on the obtained statistical model.

Routing Inspired by Ant Colony Optimization (a)

AntNet
AntHocNet

The most famous stochastic routing protocol is *AntNet* (Di Caro & Dorigo, 1998) and the Ad-Hoc network version *AntHocNet* (Di Caro, Ducatelle, & Gambardella, 2005). These protocols are inspired by the path optimization observed in the foraging task of ant colonies: Ants walk around randomly, searching for food. Once nutrition is found, the ants return to the nest and deposit a pheromone trail. These chemicals influence the search behavior of ants: if pheromones are absent, the ants search randomly, but if pheromones are present, the ants follow the previous trail with a high probability. The ant colony as a whole is able to optimize foraging as an emergent phenomenon.

AntNet mimics the behavior of real ants. Each network node periodically sends "ant" packets through the network to the known destinations. These packets collect information about their path through the network. On the way back, the "ants" increment the virtual pheromone concentration in each router. A statistical model evaluates the quality of a path based on these pheromone concentrations and sends data packets via the optimal path.

stochastic meta heuristics

Similar principles have been used in many ant colony inspired *stochastic meta heuristics* used for scheduling (Martens et al., 2007), image processing (Meshoul & Batouche, 2002), protein folding (Hu, Zhang, Xiao, & Li, 2008), and electronic circuit design (Zhang, Chung, Lo, & Huang, 2009), to name only a few.

Routing Inspired by Attractor Selection in Cells (b)

ARAS
stochastic attractor selection

Another stochastic routing approach is ARAS (Leibnitz, Wakamiya, & Murata, 2006). The protocol is inspired by *stochastic attractor selection* in cells that switch from one state to another depending on the availability of a nutrient (Kashiwagi, Urabe, Kaneko, & Yomo, 2006). ARAS' forwarding engine maintains several paths to the same destination. When the link quality of the primary path changes, another path is automatically selected. This switching phenomenon is emerging from the underlying differential equations steering the forwarding decision process. In order to let the system switch from one path to the other, randomness was intentionally added to the differential stability to environmental noise

equations. Interestingly, this augmented intrinsic noise made the system more *stable to the influence of environmental noise*.

2.4 TOWARDS CHEMICAL NETWORKING PROTOCOLS

Chemical concepts appear in a surprisingly large number of existing protocol descriptions: TCP's congestion control algorithm is based on the "conservation of packets principle" trying to bring a connection to "equilibrium" (Jacobson, 1988); AntNets stochastically routes packets proportional to "pheromone concentrations" (Di Caro & Dorigo, 1998); gossip protocols adopt the dynamics of epidemic spreading to disseminate information in a robust way.

Apparently, chemically inspired ideas are considered to be beneficial for the dynamic behavior of protocols. Furthermore, we recently observe a tendency towards stochastic protocols, because in large networks, deterministic service guarantees cannot be given anyway, and because inherent noise makes the protocols robust to environmental noise.

The chemical paradigm we introduce in this work formalizes chemical concepts for networking and provides a framework for the development of many chemically inspired protocols. The chemical paradigm extends the packet-based and active networks paradigm by an additional dimension: stochastic but predictable dynamic behavior. Whereas traditional protocols are packet- or data-oriented, active networks are code oriented. Chemical networks will be reaction-oriented and thus emphasize data and code *flows* and their dynamics. However, before we start to introduce our chemical paradigm for networking, we discuss existing models of chemistry.

CHAPTER

3

Artificial Chemistries

An introduction to abstract models of chemistry as unconventional ways of organizing computation.

Das Treffen zweier Persönlichkeiten ist wie der Kontakt zweier chemischer Substanzen: Wenn es eine Reaktion gibt, werden beide transformiert.	The meeting of two personalities is like the contact of two chemical substances: if there is any reaction, both are transformed. [3]
C.G. Jung	C.G. Jung

Artificial chemistries are abstract models of chemistry, simplified to simulate chemical reactions on a computer (Suzuki & Dittrich, 2009). Artificial chemistries are used as tools for biologists to simulate real chemistry and as unconventional paradigms to organize computation, among other usages. In this chapter, we provide an introduction to artificial chemistry and focus on the latter aspect: how the chemical reaction metaphor can be used in computer science to structure computation. Instead of listing all existing work related to artificial chemistries in this chapter, we only provide introductory material here and refer to the details in later chapters.

This chapter is organized as follows: In Section 3.1, we provide a definition of artificial chemistry before Section 3.2 elaborates on its history and the variety of chemical computing models existing today. In Section 3.3, we present a more formal definition of an important subset of artificial chemistries. We especially delve into the dynamical aspects of chemical computing and discuss existing algorithms to simulate chemical reactions *in silico*. This last section is important for our work, since our chemical packet-processing model extensively exploits the dynamics of chemical reaction systems.

3.1 DEFINITION

In the broadest sense, an artificial chemistry is a "man-made system that is similar to a chemical system" (Dittrich, Ziegler, & Banzhaf, 2001). In the design of such a system we usually recognize chemical entities such as molecules or reactions. There is not only *one* artificial chemistry: A scientist or program designer will craft its own chemical system depending on the problem to be solved; each problem usually requires another representation of molecules and reactions and a new definition of how they interact.

artificial chemistry

Dittrich et al. (2001) provided a generic definition that helps classify artificial chemistries: According to this definition an *Artificial Chemistry* is the triple $\mathcal{AC} = (\mathcal{S}, \mathcal{R}, \mathcal{A})$, where $\mathcal{S}$ is the set of molecular species, $\mathcal{R}$ is the set of reaction rules specifying how the molecules interact, and $\mathcal{A}$ is the reaction algorithm describing how and when the reaction rules are applied.

set of species

The *set of species* $\mathcal{S} = \{s_1, \ldots, s_n\}$ contains all molecule types that may appear in a certain chemistry. The species may either be defined explicitly by enumeration, for example, $\mathcal{S} = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$ or implicitly by providing rules how they are constructed. Artificial chemical models of biological reactions often define the molecules explicitly (Hjelmfelt, Weinberger, & Ross, 1991; Paŭn, 2000; Wolkenhauer, Ullah, Kolch, & Cho, 2004; Paladugu et al., 2006; Deckard, Bergmann, & Sauro, 2009). On the other hand, an implicit definition attribute the molecules a *structure* in form of, for example, mathematical objects (Banâtre & Le Métayer, 1993; Dittrich, 2001), binary strings (Banzhaf, 1993a, 1993b; Dittrich & Banzhaf, 1998), symbol strings (Holland, 1992; Decraene, 2006; Suzuki & Ono, 2002; Tschudin, 2003; Fernando & Rowe, 2007), or even graphs (Benkö, Flamm, & Stadler, 2003).

molecular structure

set of reaction rules

The *set of reaction rules* $\mathcal{R} = \{r_1, \ldots, r_m\}$ describes how the molecules interact. A reaction rule $r \in \mathcal{R}$ is usually given by the chemical reaction notation

$$s_{\mathrm{in},1} + \cdots + s_{\mathrm{in},n} \longrightarrow s_{\mathrm{out},1} + \cdots + s_{\mathrm{out},m} \qquad (3.1)$$

defining that the molecules on the left hand side are replaced by the molecules on the right hand side of the arrow. The reaction can be defined explicitly as in (3.1) or implicitly by providing rules how the structure of the reacting molecules leads to their interaction. Examples for implicit reaction rules are the interpretation of binary strings as code tapes acting on data strings as described by Dittrich and Banzhaf (1998), or the application of λ-expressions in AlChemy (Fontana, 1992).

reaction algorithm

The set of molecules together with the reaction rules describe the structural part of the chemistry. The *reaction algorithm* $\mathcal{A}$ completes the definition

1. Randomly select two tokens.
2. Try to apply reaction r_1, which is only effective if the color of the two tokens is different.
3. If the reaction is effective place the new tokens to random fields and remove the tokens previously occupying these fields.
4. Goto step 1.

Algorithm 3.1 Simple reaction algorithm for the mate-and-spread game: see also Figure 1.1 on page 7. This algorithm schedules chemical reactions by randomly picking two tokens and applying the reaction.

of the artificial chemistry by describing its dynamics, i.e. how and when which reaction rules are applied. The algorithm operates on a *reaction vessel*, i.e. an instance of an artificial chemistry $(\mathcal{S}, \mathcal{R}, \mathcal{A})$ hosting molecule instances. Note that we distinguish between molecule *species* and molecule *instances*: A *species* represents a molecular type whereas we use the term *molecule* when referring to an instance of a species.

reaction vessel

The algorithm $\mathcal{A}$ implicitly defines the structure of the vessel (Dittrich, 2001): For example, if the algorithm picks two random reactant molecules, the system simulates a well-stirred vessel. On the other hand, if the vessel shall have a spatial structure, the algorithm has to restrict itself to select reactants in a certain neighborhood.

MATE-AND-SPREAD GAME EXAMPLE 3.1.1

In Chapter 1, we introduced a simple board game to demonstrate a mechanism with emergent self-healing properties. Now we are able to associate the game with artificial chemistries:

The set of molecular species contains two explicitly defined species, namely black and white tokens: $\mathcal{S} = \{\mathrm{B}, \mathrm{W}\}$. The set of reaction rules comprises one reaction only: $\mathcal{R} = \{r_1\}$. It duplicates a pair of black and white tokens: r_1: $\mathrm{B} + \mathrm{W} \rightarrow 2\mathrm{B} + 2\mathrm{W}$. Finally, the reaction algorithm $\mathcal{A}$ continuously performs the steps listed in Algorithm 3.1.

Recall that the number of black and white tokens will strive to equilibrium where both tokens are present with an equal quantity. This equilibrium is achieved without being programmed explicitly. It is an emergent property of the small number of simple rules, described within the context of an artificial chemistry. Although we used a checkerboard to illustrate this example, the spatial distribution of the tokens does not matter for the current algorithm. Thus, the vessel can be seen as well stirred where molecules are floating around randomly. An implementation would probably store the molecules in a multiset or just remember the multiplicity of the two species.

3.2 HISTORY, PURPOSE AND VARIETY

In this section, we briefly sketch the history of artificial chemistries and their achievements, and highlight the vast diversity of artificial chemical computing models. For further information, the reader is referred to a couple of excellent reviews (Dittrich et al., 2001; Suzuki & Dittrich, 2009).

3.2.1 EMERGENCE OF REGULARITY AND ORGANIZATIONS

Turing patterns

Abstract models of chemistry describing emergent natural phenomena can be traced back to *Turing's patterns* (Turing, 1952): In his late work on mathematical biology, Turing studied reaction-diffusion systems in two and three dimensions and discovered the emergence of regular patterns, which he linked to morphogenetic phenomena in nature. In the following years, research on emergent computing focused on even more abstract and digital models, such as Cellular Automata (CA). We will refer to this branch of research later in Chapter 13, in the third part of the thesis.

string chemistry

Laing (1977) was probably the first to make a clear analogy between chemical reactions and computation: He constructed an artificial chemistry where strands of *strings* (molecules) come in contact and interact, in analogy to a Turing machine operating on a tape. Laing showed that self-replication by means of self-inspection is possible. Since then, researchers developed many other string chemistries, in which active molecules (tapes) operate on passive molecules (data) (e.g. Dittrich & Banzhaf, 1998; Ikegami, 1999).

AlChemy

One of the most influential artificial chemistries was Fontana's algorithmic chemistry *AlChemy* (Fontana, 1992; Fontana & Buss, 1994). In *AlChemy*, molecules are λ-expressions. The algorithm randomly picks two expressions from the vessel, applies the first to the second and performs β-reduction until the resulting expression is in normal form. This result then replaces another randomly selected expression in the vessel.

algorithmic chemistry

Today, as a generalization of Fontana's AlChemy, an *algorithmic chemistry* refers to an artificial chemistry where both, the species and the reaction rules are defined implicitly, i.e. where they are constructed algorithmically by a sub-molecular logic. Although there are a fixed number of rules defining how the species are structured and how they interact, the set of species and reaction rules is often infinite. The outcome of a reaction depends on the algorithmic logic and the sub-molecular structure of the participating molecules.

constructive

An algorithmic chemistry is *constructive* in the sense that species can be generated that were not present in the beginning. Fontana, Wagner, and Buss (1994) distinguish between weakly and strongly constructive systems:

In weakly constructive systems “new agents are constructed in an unspecific (essentially stochastic) fashion”. Algorithmic chemistries are strongly constructive, because “the encounter of two agents *implies* a *specific* third one” (Fontana et al., 1994).

no distinction between code and data

Another interesting aspect of AlChemy that holds for many artificial chemistries is that there is *no distinction between code and data*: In λ-calculus, functions and numbers are both represented by λ-expressions. The same molecule can either take the role of code (function) or data depending on the order the two molecules are selected. Thus, code may operate on code, which is the key requirement for self-replicating and self-modifying code and the basis for our own self-healing software approach.

emergence of organizations

Chemical Organization Theory

Fontana and Buss (1994) used AlChemy to demonstrate the spontaneous *emergence of organizations* from a “soup” of random λ-expressions. Similar self-replicating (or autocatalytic) structures were found in a lot of other chemistries (Farmer, Kauffman, & Packard, 1986a; Ikegami & Hashimoto, 1996; Dittrich & Banzhaf, 1998; Speroni di Fenizio & Banzhaf, 2000; Kvasnička & Pospichal, 2001; Suzuki & Ono, 2002; Hutton, 2002; Yamamoto, Schreckling, & Meyer, 2007). Dittrich and Speroni di Fenizio (2007) formally defined chemical organizations in their *Chemical Organization Theory* and recognized them as macro-states in which a chemical system stays for a long time. We will review the Chemical Organization Theory in more detail in Chapter 8.

ToyChem

energy model

One of the most realistic artificial chemistries is *ToyChem* (Benkö et al., 2003; Benkö, Flamm, & Stadler, 2005; Flamm et al., 2010). Molecules are represented as labeled graphs with basic properties derived from quantum mechanics. Chemical reactions are implemented as graph rewriting rules. The deep roots in quantum mechanics allow the chemistry to incorporate an *energy model* such that the macroscopic behavior follows the rules of chemical thermodynamics and kinetics. Modeling details down to the physical level however requires more processing power to simulate the reactions.

Artificial Life

Algorithmic chemistries are good frameworks for studying the emergence of organizations and, because of the remaining bonds to real chemistry, to model and study complex chemical phenomena related to life and its origins (Farmer et al., 1986a; Fontana & Buss, 1994; Dittrich et al., 2001). This is why artificial chemistry is a sub-branch of *Artificial Life*.

ARTIFICIAL CHEMICAL COMPUTING 3.2.2

Another purpose of artificial chemistries is to use reaction systems to organize computation in an unconventional way. In the 1980ies, Banâtre developed

Gamma

Gamma, an abstract multiset rewriting calculus (Banâtre & Le Métayer, 1986). In the γ-calculus, concurrent computation can be expressed quite naturally: An algorithm such as the computation of prime numbers is implemented as interaction among the entities, e.g. as simple arithmetic operations on number "molecules". "The result of a Gamma program is obtained when a stable state is reached that is to say when no more reactions can take place." (Banâtre, Fradet, & Radenac, 2005)

Note that this operation principle is quite different from the chemistries used to study organizations. In Gamma, the result is presented on the microscopic, symbolic level by the molecules remaining in the vessel after the chemical program terminated. On the other hand, a chemistry that runs continuously and settles in an organization presents its result at the macroscopic level, i.e. by the presence/absence or the concentration of the species in steady-state.

logical parallelism

In both representations, a chemical formulation of a problem automatically leads to a *logical parallelism*, which can be physically executed on one or multiple processors (Banâtre & Le Métayer, 1993). Berry and Boudol (1989) developed Gamma further to the Chemical Abstract Machine (CHAM).

P-systems

P-systems (Paŭn, 2000; Calude & Paŭn, 2001) is another computing model inspired by cell membranes. Each reaction vessel or "cell" contains a multiset of molecules and reactions operating on them. Additionally, P-systems allow hierarchies of multiset compartments to be constructed recursively and molecules to travel through the membranes. The result is presented microscopically in the outermost membrane when no reaction may take place anymore.

Fraglets

Fraglets is an artificial chemistry (Tschudin, 2003) in which multiset compartments are distributed over a network. The molecules actually represent packets traveling through the network and reacting with each other. The packets (=molecules) are structured as strings of symbols where each symbol represents an instruction that has to be executed in the target node. In this thesis, we use Fraglets as execution layer for our chemical networking protocols and show how both microscopic symbolic and macroscopic concentration information can be used together. We will introduce Fraglets in detail in Chapter 5.

applications

Artificial chemistries were considered for a variety of *applications* beyond the modeling of wet chemistry and the study of emergent organizations: Ziegler and Banzhaf (2001) used an artificial chemistry to control the movement of a robot. There are a few search and optimization algorithms based on artificial chemistries (Banzhaf, 1990; Kanada, 1995; Weeks & Stepney, 2005; Yamamoto, 2010; Yamamoto & Banzhaf, 2010). Tominaga and Setomoto (2008) even used an artificial chemistry for composing music. Although artificial chemistries were quite successful in solving concrete problems, a generic design and analysis method is still missing. There is no general recipe how to structure the microscopic rules such that the desired functionality emerges.

A FORMAL APPROACH TO ARTIFICIAL CHEMISTRIES 3.3

In this section, we re-iterate through the formal definition of an artificial chemistry. This time, we introduce more concepts and terms. We extensively discuss the dynamics of chemical reactions in well-stirred reaction vessels and their *in silico* simulation. By restricting the model to the well-stirred case, we will not cover all existing artificial chemistries. However, we now build the foundation upon which we later define our artificial chemical networking model.

We recall that according to Dittrich et al. (2001), an *artificial chemistry* is formally defined as the triple $\mathcal{AC} = (\mathcal{S}, \mathcal{R}, \mathcal{A})$. $\mathcal{S}$ denotes the set of molecular species, $\mathcal{R}$ the set of reaction rules, and $\mathcal{A}$ the algorithm that defines which reaction is executed next and when. We split the detailed description into three parts. Section 3.3.1 focuses on the structural aspects of the chemistry: the reaction network spanned over the set of species. In Section 3.3.2, we show how the reaction vessel and its contents are described. Finally, in the third and largest subsection, Section 3.3.3, we elaborate on the dynamical aspects of chemical reaction kinetics and discuss existing stochastic algorithms to simulate chemical reactions on a traditional computer.

artificial chemistry

THE STRUCTURE OF CHEMICAL REACTION NETWORKS 3.3.1

The structural part of an artificial chemistry describes the potential appearance of molecules and their interactions. A reaction rule describes how to replace molecules in the reaction vessel. A *reaction rule* $r \in \mathcal{R}$ is formally given as a pair of multisets and an assigned reaction coefficient

reaction rule

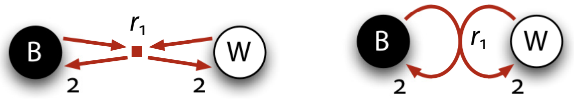
$$r\colon \quad \mathcal{M}_{\mathrm{in},r} \xrightarrow{k_r} \mathcal{M}_{\mathrm{out},r} \qquad \forall r \in \mathcal{R} \tag{3.2}$$

where $\mathcal{M}_{\mathrm{in},r} \in \mathcal{M}(\mathcal{S})$ are the *reactants* (or left-hand side) and $\mathcal{M}_{\mathrm{out},r} \in \mathcal{M}(\mathcal{S})$ the *products* (or right-hand side) of the reaction, and k_r is the *reaction coefficient* that determines the speed of the reaction; we will discuss the reaction coefficient later when studying the dynamics of a reaction network. The number of reactant molecules $\mathcal{O}(r) = |\mathcal{M}_{\mathrm{in},r}|$ is called the *order of the reaction*. As we mentioned before, reaction rule (3.2) can also be written in the equivalent chemical notation (compare to (3.1))

reaction coefficient

order of a reaction

Figure 3.1 Reaction network graphs of the mate-and-spread game: see also Figure 1.1 on page 7. The artificial chemistry is defined by the set of species $\mathcal{S} = \{B, W\}$ and the set of reaction rules $\mathcal{R} = \{r_1\colon B + W \longrightarrow 2B + 2W\}$. (a) depicts the arc-weighted bipartite directed graph whereas (b) shows the simplified graphical notation where the reaction vertex is implicitly represented by the joining arcs.



(a) Arc-weighted bipartite directed graph

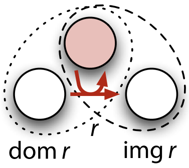
(b) Simplified graphical notation

$$r\colon \quad \sum_{s\in\mathcal{S}} \alpha_{s,r}\, s \xrightarrow{k_r} \sum_{s\in\mathcal{S}} \beta_{s,r}\, s \qquad \forall r \in \mathcal{R} \tag{3.3}$$

stoichiometric coefficient

In this equation, the positive integers $\alpha_{s,r}, \beta_{s,r} \in \mathbb{N}_0$ denote the number of molecules consumed and produced by the reaction. The net production of molecule s by reaction r is given by the *stoichiometric coefficient* $\gamma_{s,r} = \beta_{s,r} - \alpha_{s,r}$.

domain / image



Next to the reactant and product multiset we also define the set of reactant and product *species* according to Benkö et al. (2008): The *domain of a reaction* r is the set of those species that are in the reactant multiset: $\operatorname{dom} r = \{s \in \mathcal{S} \mid \alpha_{s,r} > 0\}$. Similarly, the *image of a reaction* r is the set of those species that are in the product multiset: $\operatorname{img} r = \{s \in \mathcal{S} \mid \beta_{s,r} > 0\}$.

The tuple $(\mathcal{S}, \mathcal{R})$ constitutes the structural part of the artificial chemistry. It can be represented by an arc-weighted bipartite directed graph with both, molecular species and reactions as vertices and edges $s \rightarrow r$ with weight $\alpha_{s,r}$ if $\alpha_{s,r} > 0$ and edges $r \rightarrow s$ with weight $\beta_{s,r}$ if $\beta_{s,r} > 0$ (Benkö et al., 2008). Figure 3.1(a) shows the bipartite graph of the mate-and-spread game. Instead of the bipartite graph, we usually draw a simplified graph, show in Figure 3.1(b) for the same reaction, where the reaction vertex is implicitly represented by the joining arcs.

3.3.2 REACTION VESSEL: INSTANTIATION OF A CHEMISTRY

chemical universe

scheduling algorithm

The structure of the reaction network is completely defined by the pair $(\mathcal{S}, \mathcal{R})$, which Benkö et al. (2008) call the *chemical universe*. The reaction algorithm $\mathcal{A}$ defines the dynamic behavior of this chemical universe. The reaction algorithm operates on a reaction vessel and schedules reaction events, i.e. it decides which reaction is executed next and when. Therefore, we also use the term *scheduling algorithm* synonymously for reaction algorithm. In this subsection, we discuss the structural aspects of the reaction algorithm, i.e. how it executes reactions in a reaction vessel. In the next subsection, we then focus on the dynamical aspects of reaction scheduling.

reaction vessel

species

A *reaction vessel* v is an instance (or realization) of an artificial chemistry. A vessel contains a multiset of molecules, denoted as $\mathcal{M}_v \in \mathcal{M}(\mathcal{S})$; each molecule in the reaction vessel is an instance of one of the molecular *species* $s \in \mathcal{S}$.

quantity

composition

We denote the *quantity* of species s in the vessel multiset (i.e. its multiplicity or copy number) as $N_s(t)$. The *composition* of reaction vessel v is then given by the vector of molecular quantities

$$\mathbf{N}_v(t) = \begin{pmatrix} N_{s_1}(t) \\ \vdots \\ N_{s_{|\mathcal{S}|}}(t) \end{pmatrix} \tag{3.4}$$

concentration

Chemists usually operate with the macroscopic *concentration* of species $s \in \mathcal{S}$, which is defined as $x_s = N_s/\Omega$. The scaling parameter Ω represents the size of the system. For molar concentrations, the system size is given as $\Omega = N_\mathrm{A} V$, where N_A is Avogadro's constant and V is the volume of the reaction vessel. The molar concentration is written as $[s] = N_s/N_\mathrm{A}V$. In an artificial chemical setting however, one may use a simplified notion of concentration. For example, the simplest way is to set the system size to $\Omega = 1$ such that the concentration becomes equivalent to the quantity of a species. Alternatively, we may define the concentration as the abundance, i.e. the relative quantity of a species. In this case, the system size is equal to the total number of molecules $\Omega = \sum_{s \in \mathcal{S}} N_s$.

active reaction

Often, more than one reaction rule is ready to be executed at the same time. The algorithm defines whether all possible reactions are applied in parallel, or, if not, which reaction is applied first and when. As we will see later, choosing the right algorithm is crucial for chemical networking protocols to behave nicely. We say that a reaction is *active*, if enough reactant molecules are present in the reaction vessel such that if the reaction is applied to the vessel, it can consume the required molecules. Formally, a reaction $r \in \mathcal{R}$ is active in vessel v iff

$$\mathcal{M}_v \cap \mathcal{M}_{\mathrm{in},r} = \mathcal{M}_{\mathrm{in},r} \tag{3.5}$$

inert

If none of the reactions is active, the artificial chemistry halts, and we say that the vessel is *inert*.

An alternative way of understanding the influence of reactions to a reaction vessel is to write the vessel multiset as a composition vector according to (3.4). The net effect of a reaction $r \in \mathcal{R}$ can then be represented by a state-change vector (Gillespie, 2002) of the same dimension

$$\boldsymbol{\gamma}_r = \begin{pmatrix} \gamma_{s_1,r} \\ \vdots \\ \gamma_{s_{|\mathcal{S}|},r} \end{pmatrix} \tag{3.6}$$

where $\gamma_{s_i,r}$, the stoichiometric coefficient, specifies the net change of the quantity of species s_i by reaction r. For a given initial vessel composition $\mathbf{N}_{v,0}$, the resulting configuration after applying reaction r is simply $\mathbf{N}_{v,1} = \mathbf{N}_{v,0} + \boldsymbol{\gamma}_r$.

3.3.3 CHEMICAL REACTION KINETICS

In a gas-phase chemical system that is kept well stirred and thermally equilibrated, molecules move around following Brownian motion (McQuarrie, 1997, Chap. 27). Because one usually does not want to keep track of all positions and moments of the individual molecules, the state of a chemical system can be reduced to the current number of molecules of each species. This is accompanied by a loss of information, resulting in a stochastic process that can be described by the *Chemical Master Equation* (McQuarrie, 1967; Gillespie, 1992). We will discuss the Chemical Master Equation in more detail in Chapter 8 where we show how to analyze the dynamics of chemical networking protocols. For now, we just note that the master equation describes the time evolution of the probability that the chemical system occupies one of the possible reaction vessel compositions.

Chemical Master Equation

(a) *The Law of Mass Action*

Even before having had a stochastic description that roots in statistical mechanics, researchers had observed macroscopic dynamical phenomena of chemical reactions. For example, the more molecules are located within the same volume, the more likely collisions and reactions become. This fundamental law of chemical kinetics was discovered in the 19th century and is known as the *law of mass action* (Waage & Guldberg, 1864; see also the English translation by Abrash, 1986). It states that the reaction rate is proportional to the concentration (quantity per volume) of each reactant. That is, the chemical reaction

law of mass action

$$\mathrm{X} + \mathrm{Y} \xrightarrow{k} \mathrm{Z} \tag{3.7}$$

consuming one molecule of species X and Y and producing a molecule of species Z reacts with an average rate of

$$r = k\,[\mathrm{X}]\,[\mathrm{Y}] \tag{3.8}$$

where k is the kinetic coefficient associated with the reaction, and $[X]$, $[Y]$ are the molar concentrations of reactants X and Y, respectively. The coefficient k can be expressed in terms of physical quantities like the temperature and the activation energy, but for the moment, we assume it is constant. We discuss the microphysical derivation of the reaction coefficient later in Chapter 18.

Such knowledge from textbook physical chemistry is a macroscopic description of the average behavior of large quantities of molecules. However, for an algorithmic chemistry we have to simulate the microscopic behavior of the system at the level of individual molecular collisions in order to perform the intended computations encoded within the molecules.

Exact Stochastic Reaction Algorithms *(b)*

A simulation algorithm for chemical reactions in a well-stirred vessel has to be correct and efficient: First, the algorithm has to simulate chemical reactions *stochastically correct*, i.e. it has to provide a single sample trajectory of the random process, described by the chemical master equation. Second, the algorithm shall be *efficient* in the sense that each iteration is guaranteed to execute a reaction. This requires a translation of the species-oriented chemical master equation to a reaction-oriented stochastic algorithm that generates and executes reaction events as a sequence of (r, t) tuples, recording which reaction r happens when (t).

stochastically correct

efficient

We could use a reaction algorithm similar to the one applied to our board game (see Algorithm 3.1), namely an algorithm that randomly selects two molecules and checks whether one of the reactions $r \in \mathcal{R}$ can be applied. This process akin to random molecular collisions indeed leads to a macroscopic behavior according to the law of mass action. However, this algorithm does not scale well for many species and few reaction channels when only a few collisions actually lead to reaction events.

Researchers proposed many *Monte Carlo algorithms* to provide an efficient and exact stochastic simulation of chemical reactions (Gillespie, 1976, 1977, 2001; Gibson & Bruck, 2000; Le Novère & Shimizu, 2001; Haseltine & Rawlings, 2002; Gillespie & Petzold, 2003; Rathinam, Petzold, Cao, & Gillespie, 2003; Tian & Burrage, 2004; Cao & Petzold, 2005; Cau, Gillespie, & Petzold, 2005, 2006; Chatterjee, Vlachos, & Katsoulakis, 2005; Samant & Vlachos, 2005; Di Liu & Vanden-Eijnden, 2007). An overview is provided by Gillespie (2007).

Monte Carlo algorithms

The two algorithms we considered for chemical program execution were *Gillespie's Direct Method* and the *Next Reaction Method*: In every iteration, *Gillespie's Direct Method* (Gillespie, 1977) draws two random numbers. The first one is used to determine which reaction shall be executed next whereas

Gillespie's Direct Method

Next Reaction Method

the second determines the next reaction time. The *Next Reaction Method* (Gibson & Bruck, 2000) calculates the next reaction time based on an exponentially distributed random variable for every reaction separately and sorts it into a priority queue from which the next reaction is determined.

Before we exemplarily discuss Gibson and Bruck's algorithm in more detail, we study how both algorithms compute the reaction interval such that the macroscopic behavior follows the law of mass action.

propensity

Reaction Propensities. Both algorithms make use of the concept of the reaction propensity. The *propensity* $a_r(\mathbf{N}(t))$ reflects the probability that reaction $r \in \mathcal{R}$ occurs within the next infinitesimal time interval $[t, t + \mathrm{d}t)$. The propensity is the product of the probability that a collision leads to a reaction times the frequency of a molecular collision:

$$a_r(\mathbf{N}(t)) = c_r \cdot h_r(\mathbf{N}(t)) \tag{3.9}$$

microscopic reaction coefficient

The *microscopic reaction coefficient* c_r is a stochastic rate constant depending on physical properties of the reactant molecules. It denotes the probability that a given combination of reactant molecules will collide in the next time interval $\mathrm{d}t$ multiplied by the probability that the colliding molecules will actually react (Gillespie, 1992, 2000). There is a direct mapping between the stochastic rate constant c_r and the macroscopic reaction rate constant k_r (see Wolkenhauer et al., 2004, Eq. (20)). We will delve more into the physical meaning of this coefficient in Chapter 18.

number of distinct combinations

The second factor in (3.9), $h_r(\mathbf{N}(t))$, denotes the *number of distinct combinations* of reactant molecules of reaction r (Wolkenhauer et al., 2004). The more reactants there are in the vessel, the more collision partners are available, i.e. the higher is h_r. This relation is expressed by the binomial coefficient

$$h_r(\mathbf{N}(t)) = \prod_{s \in \mathcal{S}} \binom{N_s(t)}{\alpha_{s,r}} \tag{3.10}$$

For example, for a reaction r: $2\mathrm{X} + \mathrm{Y} \longrightarrow \ldots$, the stoichiometric coefficients are $\alpha_{\mathrm{X},r} = 2$ and $\alpha_{\mathrm{Y},r} = 1$, and hence there are

$$
\begin{aligned}
h_r\big(\mathbf{N}(t)\big) &= \prod_{s\in\mathcal{S}} \binom{N_s(t)}{\alpha_{s,r}} \\
&= \binom{N_{\mathrm{X}}(t)}{2} \cdot \binom{N_{\mathrm{Y}}(t)}{1} \\
&= \frac{N_{\mathrm{X}}(t)\,\big(N_{\mathrm{X}}(t)-1\big)}{2} \cdot N_{\mathrm{Y}}(t)
\end{aligned}
\tag{3.11}
$$

reactant combinations.

The propensity expresses the stochastic equivalence of the deterministic reaction rate. Let us assume that the number of molecules is large ($N_s(t) \gg 1$) and that the chemistry only contains reactions where at most one instance of each reactant is consumed (i.e. $\alpha_{s,r} \in \{0,1\}$). In this case, the propensity can be simplified to

$$
a_r\big(\mathbf{N}(t)\big) = c_r \prod_{\{s\in\mathcal{S}|\alpha_{s,r}=1\}} N_s(t) \tag{3.12}
$$

For example, for a reaction r: $\mathrm{X} + \mathrm{Y} \longrightarrow \ldots$ the propensity is

$$
a_r\big(\mathbf{N}(t)\big) = c_r N_{\mathrm{X}}(t)\, N_{\mathrm{Y}}(t) \tag{3.13}
$$

By using our simplified meaning of concentrations ($x_s \equiv N_s$) we obtain an equation similar to the reaction rate equation (3.8). That is, as a first approximation, we can think of the propensity as the mean reaction rate. We will examine the relation between microscopic and macroscopic description in more detail in Chapter 8.

Gibson and Bruck's Next Reaction Method. The Next Reaction Algorithm by Gibson and Bruck (2000) computes the next reaction time of each reaction rule based on its propensity. The next event of each reaction rule is kept in a priority queue. In an endless loop, the algorithm picks the first event from the queue, executes the reaction, and updates the reaction times of all dependent reaction rules. Algorithm 3.2 shows the detailed algorithm.

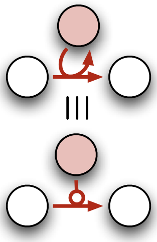
On average, the reaction rates are proportional to the corresponding propensity values and thus, to their reactant quantities. This means that the algorithm realizes the law of mass action on the macroscopic level. Therefore, we also refer to the Next Reaction Method algorithm as a particular instance of a *law of mass action scheduler*.

law of mass action scheduler

1. Initialization:
    - a) $t = 0$ s;
    - b) for each reaction rule $r_j \in \mathcal{R}$, calculate the propensity function a_j according to (3.10);
    - c) for each reaction rule $r_j \in \mathcal{R}$, draw the reaction interval from an exponential distribution: $\tau_j \sim \mathrm{Exp}(1/a_j)$;
    - d) store the putative reaction time values $t_j = t + \tau_j$ in an indexed priority queue Q (see Gibson and Bruck (2000)).
2. Let r_μ be the reaction rule whose putative reaction time, t_μ, stored in Q is least.
3. Advance the simulation time to the occurrence time of the reaction rule: $t = t_\mu$.
4. Execute reaction rule r_μ, i.e. rewrite the vessel's multiset according to the state-change vector (see (3.6) on page 30).
5. Update the next reaction time of all those reaction rules r_j that depend on the executed reaction. That is, all those reaction rules have to be adjusted that have reactants that were changed by reaction rule r_μ. Formally, for each reaction rule $r_j \in \left\{ r \in \mathcal{R} \mid \mathrm{dom}\, r \cap \left\{ s \in \mathcal{S} \mid \gamma_{s,r_\mu} \neq 0 \right\} \neq \varnothing \right\}$,
    - a) calculate the propensity function, a_j, according to (3.10);
    - b) if $j \neq \mu$, adjust the next reaction time without drawing a new random variable: $t_j \leftarrow t + (a_{j,\mathrm{old}}/a_{j,\mathrm{new}})(t_j - t)$;
    - c) if $j = \mu$, draw a new reaction interval from an exponential distribution, $\tau_j \sim \mathrm{Exp}(1/a_j)$, and set the next reaction time to $t_j \leftarrow t + \tau_j$;
    - d) replace the old value t_j in Q with the new value and re-sort Q.
6. Goto step 2.

Algorithm 3.2 Next Reaction Method: Schedules reactions according to the method proposed by Gibson and Bruck (2000).

(c) Example



Let us step through Gibson and Bruck's Next Reaction Algorithm while it drives a simple reaction network, depicted in Figure 3.2. It consists of a forward reaction rule r_f that converts X- into Y-molecules and a reverse reaction rule r_r doing the opposite. The two species C and D catalyze the reactions: A *catalyst* is a molecule that belongs to the domain and to the image of a reaction rule. For catalyzed reactions we often use the graphical short notation as shown in the margin.

The explicit artificial chemistry for this example is formally given as $\mathcal{AC} = (\mathcal{S}, \mathcal{R}, \mathcal{A})$ where the set of species is $\mathcal{S} = \{\mathrm{C}, \mathrm{D}, \mathrm{X}, \mathrm{Y}\}$ and the reactions are $\mathcal{R} = \left\{ r_f, r_r \right\}$, where

$$r_f\colon \quad \mathrm{C} + \mathrm{X} \longrightarrow \mathrm{C} + \mathrm{Y} \tag{3.14a}$$

$$r_r\colon \quad \mathrm{D} + \mathrm{Y} \longrightarrow \mathrm{D} + \mathrm{X} \tag{3.14b}$$

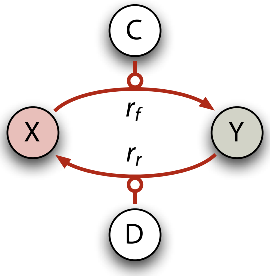


Figure 3.2 Reaction network of a catalyzed reversible reaction: A reversible reaction from species X to Y is represented by two non-reversible reactions, a forward and a reverse reaction. Both are catalyzed by species C and D, respectively.

We set the reaction coefficients of both reaction rules to $c_{r_f} = c_{r_r} = 1$. Let the algorithm $\mathcal{A}$ be an instance of the Next Reaction Algorithm by Gibson and Bruck (2000).

In the following, we will iterate through Algorithm 3.2 step by step. To illustrate the dynamic behavior of the algorithm, we display a series of snapshots of the vessel's state in Figure 3.3.

1.a) We start the experiment at time t_0 with ten instances of species X and no instances of species Y; each catalyst is present with one molecule:

$$\mathbf{N}(t_0) = \begin{pmatrix} N_C(t_0) \\ N_D(t_0) \\ N_X(t_0) \\ N_Y(t_0) \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 10 \\ 0 \end{pmatrix} \tag{3.15}$$

1.b) First, the algorithm computes the propensity of the two reaction rules according to (3.9). In other words, the algorithm determines the probability that reactions occur in the next infinitesimal time according to

$$a_{r_f}(t_0) = c_{r_f} N_C N_X = 10 \tag{3.16a}$$

$$a_{r_r}(t_0) = c_{r_f} N_D N_Y = 0 \tag{3.16b}$$

1.c) For each reaction rule, the algorithm draws an exponential random variable with the inverse of the corresponding propensity as mean. The resulting value reflects the reaction interval: The interval of the forward reaction rule is finite, $\tau_{r_f} \sim \mathrm{Exp}(1/10)$ – 0.1 s on average – whereas the second reaction rule never occurs, because there are no Y molecules to react with: $\tau_{r_r} \sim \mathrm{Exp}(1/0) = \infty$.

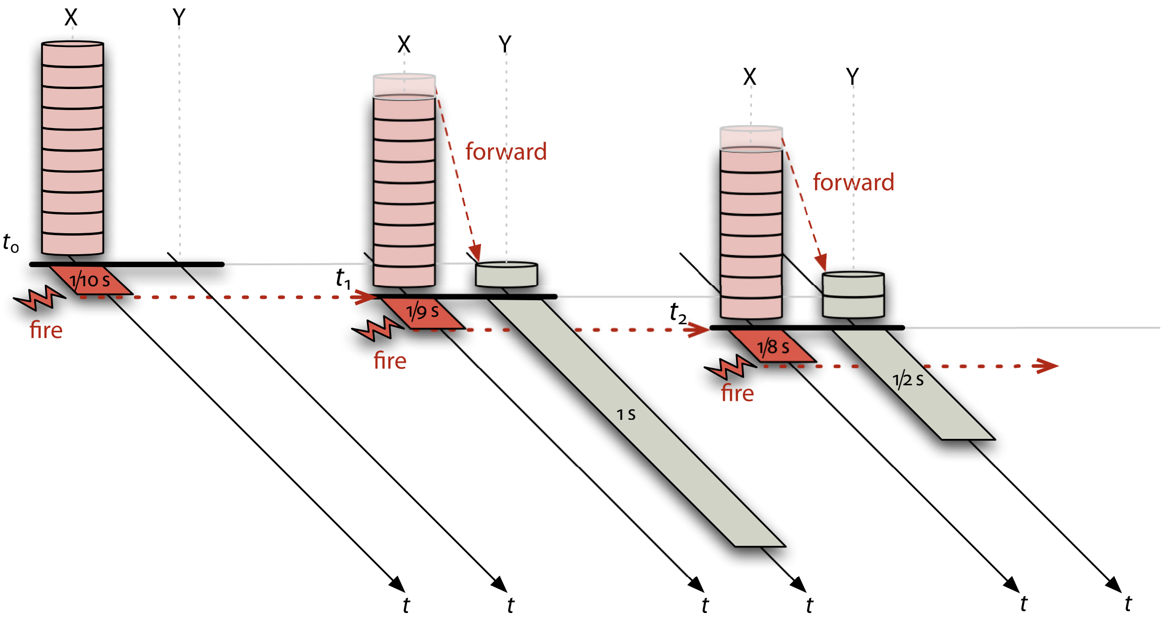


Figure 3.3 Next Reaction Algorithm driving a reversible reaction: Reactions are scheduled according to the Next Reaction Algorithm by Gibson and Bruck (2000). The rectangle along the time axes illustrate the putative reaction interval. The reaction rule with the earliest deadline is executed, and this causes all dependent reaction rules to be rescheduled.

1.d) The algorithm then computes the putative reaction time for each reaction rule: $t_{r_f} = t_0 + \tau_{r_f}$ and $t_{r_r} = \infty$. The initialization procedure is completed by adding the forward reaction rule before the reverse reaction rule to the priority queue.

2.-4. The forward reaction rule r_f, which is in front of the queue, is executed next: First, the simulation time is advanced to $t_1 = t_{r_f}$. Then, the forward reaction is executed according to the state-change vector (3.6) on page 30, meaning that an X-molecule is removed from the multiset and a Y-molecule is added instead. After this first reaction occurred, the composition vector reads

$$\mathbf{N}(t_1) = \mathbf{N}(t_0) + \mathbf{N}_{r_f} = \begin{pmatrix} 1 \\ 1 \\ 10 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -1 \\ +1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 9 \\ 1 \end{pmatrix} \tag{3.17}$$

5. After the reaction has been executed, the algorithm iterates over all reaction rules that potentially consume one of the species modified: In our example, reaction rule r_f modified the quantity of both species X and Y. That is, the combinatorial number of collision partners for

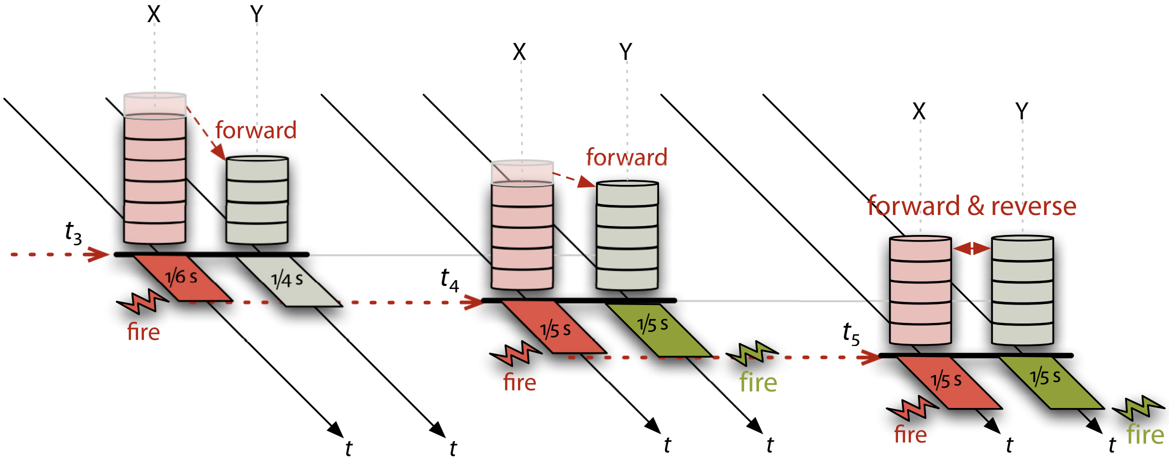


Figure 3.3 cont.: When both species are present in equal quantity ($N_X = N_Y$) the reaction intervals are equal, too. Consequently, the reaction network reached equilibrium where both X- and Y-species are present with equal quantity.

both reaction rules changed and the algorithm has to re-calculate their propensities:

$$a_{r_f}(t_1) = c_{r_f} N_C N_X = 9 \tag{3.18a}$$

$$a_{r_r}(t_1) = c_{r_f} N_D N_Y = 1 \tag{3.18b}$$

The algorithm again determines the reaction interval of both reaction rules based on their propensity values: $\tau_{r_f} \sim \text{Exp}(1/9)$ and $\tau_{r_r} \sim \text{Exp}(1)$. A forward reaction is very likely to happen first, but due to the randomness, there is a small chance that a reverse reaction is scheduled even before. Whatever reaction rule has a shorter reaction interval, it is moved in front of the priority queue and the reaction algorithm restarts with step 2.

Figure 3.3 shows the time evolution of the reaction vessel and at the same time the putatively scheduled reaction intervals for both reaction rules. In this illustration, we ignored the stochasticity of the algorithm and scheduled the reactions deterministically. In this case, the forward reaction is executed five times until the vessel contains the same number of X- and Y-molecules. Then,

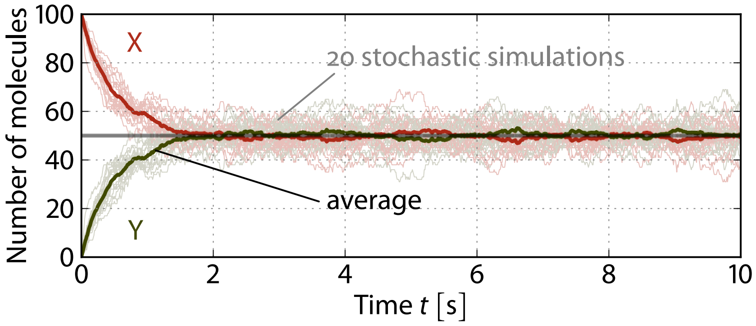


Figure 3.4 Stochastic simulation of a reversible reaction: Quantity of species X and Y in 20 independent simulation runs and their average.

both reactions have the same propensity, hence the same reaction interval, and are executed at the same time: the vessel's composition does not change anymore.

In reality, the algorithm determines the reaction intervals stochastically by drawing random variables form an exponential distribution. This still leads to equilibrium where both species contain the same number molecules *on average*. Figure 3.4 shows time traces of the quantities of X- and Y-molecules obtained by 20 simulation runs. Every run is different, but if the quantities are averaged over all runs, a deterministic trajectory can be observed.

3.4 SUMMARY

Artificial chemistries have been developed to study the behavior of chemical reactions. In the first place, artificial chemistries are modeling approaches to explain emergent phenomena in biochemistry, such as self-organization (Fontana & Buss, 1994; Dittrich & Speroni di Fenizio, 2007), pattern-formation (Turing, 1952), and the emergence of life (Fernando & Rowe, 2007).

In a second thrust, researchers started to use chemical models to organize computation (e.g. Laing, 1977; Banâtre & Le Métayer, 1986). Our chemical approach to networking protocols that we will present in the second part falls into this category: it uses the chemical reaction metaphor to structure communication in a computer network. Our self-healing approach discussed in the third part is then based on chemical self-organization.

In this chapter, we showed how an artificial chemistry is described as a reactive system among components (molecules) and discussed the dynamical aspects of chemical reactions. The *law of mass action* will play a central role in our own approach. We demonstrated that a stochastic algorithm can be used to schedule individual reaction events (Gillespie, 1977; Gibson & Bruck, 2000) and that the resulting system still follows a deterministic trajectory on average. We will come back to the relation between a stochastic and deterministic system description later in Chapter 8 when we discuss how the dynamics of a chemical networking protocol can be analyzed.

PART II

Chemical Networking Protocols (CNPs)

CHAPTER 4

Principles of Chemical Networking

Introduction to our novel method to design, execute, and analyze networking protocols, inspired by chemical reaction networks.

> The symbol and the metaphor
> are as necessary
> to science as to poetry. [4]
>
> *The Habit of Truth*
> Jacob Bronowski

This chapter introduces a novel approach to organize the exchange of information in packet networks, inspired by chemical reaction networks. In order to convey the principles of chemical networking, we are exploiting metaphors from chemistry. A "molecule" becomes a synonym for a data packet whereas a "reaction" refers to the interaction of a packet with code. Although the proposed model can be described algorithmically and mathematically without any reference to chemistry, the metaphor is helpful to intuitively understand the basic concepts of our model.

More importantly, the chemical metaphor has guided us in adapting tools and methods that are available for chemical reaction systems to man-made communication systems. One prevalent challenge for network engineers is that there is no satisfying theory that unifies the functional and the dynamical aspects of protocol design, execution and analysis. This could change with chemical networking protocols, because in chemistry, the interdependence between the functional reaction mechanism at the microscopic level and the reaction dynamics at the macroscopic level is well understood and can be described mathematically. But in order to apply results from chemical research to computer networks, the relations between chemistry and computer networks have to be much stronger than just metaphorical.

The goal of this chapter is to describe the big picture of our approach. The following chapters will then discuss each aspect and consequence in more detail: i.e. execution, analysis, and synthesis of chemical networking protocols. In this chapter, we first elaborate more on the chemical metaphor applied to networking protocols in Section 4.1. In Section 4.2, we propose an engineering model at two hierarchical levels: At the lower, microscopic level (Section 4.2.1) we provide an execution model for "chemical" networking protocols. Virtual "molecules" representing packets as well as fragments of code are floating around in a "reaction vessel". These molecules collide and, by reacting, they carry out computation and optionally send the result to a distant reaction vessel. At the upper, macroscopic level (Section 4.2.2), we abstract away from the concrete structure of code and data and focus on the dynamic behavior of the emergent distributed reaction networks. Because there is a correspondence between the microscopic execution engine and the macroscopic model, we are able to design and analyze chemical networking protocols based on the theory of chemical kinetics by treating information flows as distributed reaction networks.

4.1 CHEMISTRY AS A METAPHOR FOR NETWORKING PROTOCOLS

A chemical reaction can be described at two levels of detail: microscopically and macroscopically. At the microscopic level, two individual molecules collide; their structure or shape is modified by physical forces leading to one or more product molecules. If we know the physical laws we are able to comprehend and even predict the outcome of a molecular collision, decide whether or not the collision leads to a reaction and determine how the shape of the resulting products look. Identical reactants will always result in identical products.[*]

At the macroscopic level, a bimolecular reaction can be seen as a black box with two input streams and one or more output streams. In this sense, a stream is a flow of molecules of the same molecular type that can be characterized by the rate at which instances of a molecular type are consumed or produced by the chemical reaction. At this higher level, we are not interested in the microscopic rearrangement of atoms anymore. Instead, we treat the molecular types as abstract entities and try to understand the dynamics of chemical reactions. For instance, we are interested in how many molecules of a certain type are produced by a reaction per second. General statements

[*] This is a simplification, as physical chemistry also requires that the reactants collide with the same kinetic energy at the same angle.

such as the law of mass action are focal points at this level of description. The law of mass action connects the change of reactant concentrations to the resulting change of the reaction rate.

It is exactly the two different perspectives and scales of chemical reactions that make them interesting as a metaphor for information processing in packet switched computer networks. As we mentioned in Section 2.1.2, networking protocols are also modeled at two different levels of abstraction: At the microscopic level, packets are treated as individual objects, each instance having its own structure and content. This content is important to the end-user who wants to convey information over the network by encoding it inside the packet. The core network infrastructure also makes use of this content in order to decide how to treat the packet: For example, routers determine the outgoing interface of a packet by inspecting its header fields. Protocol software is commonly implemented as a collection of distributed state machines (see for example Postel (1981) for the Transmission Control Protocol (TCP) or Rekhter, Li, and Hares (2006) for the Boarder Gateway Protocol (BGP)). The state machine encodes what to do when receiving a packet or when triggered by another internal or external event.

When we take a step back to have a macroscopic view on data packets we see them as packet *streams*. Queuing theory (Cassandras, 1993, Chap. 6) and Network Calculus (Le Boudec & Thiran, 2001), as one of its analysis frameworks, abstract away from the content of data packets and make statistical assumptions about the properties of whole packet streams. On this abstraction level, network models focus on dynamic properties of the packet streams.

Despite the analogy between chemistry and chemically inspired computer networks, a fundamental difference remains: Whereas chemistry is a *model* of natural systems, a computer network is the system itself. Speaking about chemical models, Gibson and Bruck (2000) start their paper with the following statement:

> The process of creating a mechanistic, predictive model of a system can be broken into two steps: (a) creating a complete description of the chemical, physical, and biological processes involved; and (b) using mathematics to generate predictions.

In this sense, the goal of this thesis is to *create* a system *and* its corresponding model, for which the above statements hold. If we manage to develop a system for packet processing that is close enough to chemistry without dealing with unnecessary details, we hope to be able to build networking protocols

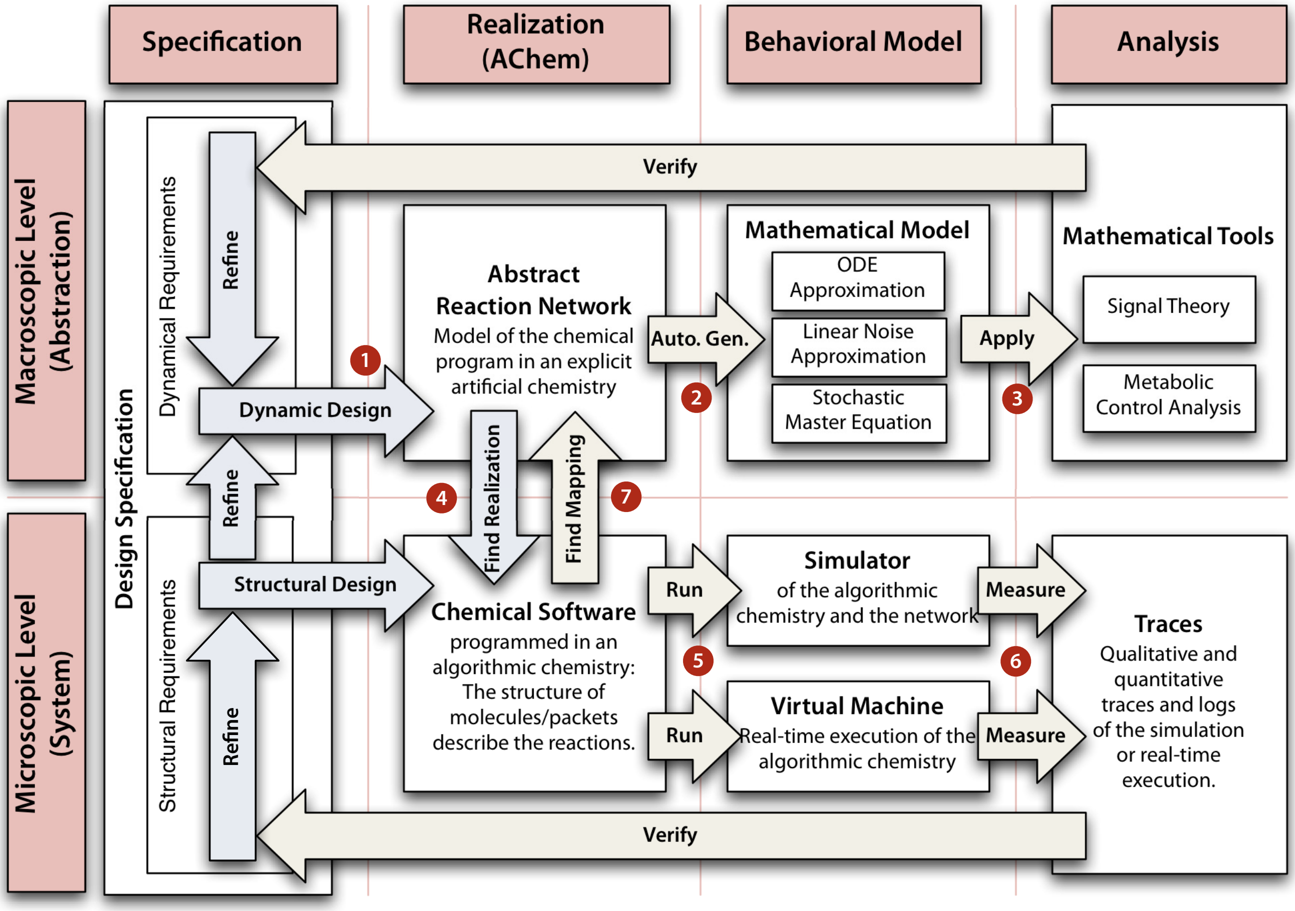

Specification
Realization (AChem)
Behavioral Model
Analysis
Macroscopic Level (Abstraction)
Microscopic Level (System)
Design Specification
Dynamical Requirements
Structural Requirements
Refine
Refine
Refine
Verify
Verify
Dynamic Design
Structural Design
Abstract Reaction Network
Model of the chemical program in an explicit artificial chemistry
Find Realization
Find Mapping
Chemical Software
programmed in an algorithmic chemistry: The structure of molecules/packets describe the reactions.
Auto. Gen.
Mathematical Model
ODE Approximation
Linear Noise Approximation
Stochastic Master Equation
Apply
Mathematical Tools
Signal Theory
Metabolic Control Analysis
Run
Run
Simulator
of the algorithmic chemistry and the network
Virtual Machine
Real-time execution of the algorithmic chemistry
Measure
Measure
Traces
Qualitative and quantitative traces and logs of the simulation or real-time execution.
1
2
3
4
5
6
7

Figure 4.1 Engineering model for chemical networking protocols: Two hierarchical levels: At the microscopic level, chemical protocol software is executed by a chemical virtual machine. The software is "written" in an algorithmic chemistry where the structure of the molecules/code strands implicitly defines the chemical reactions being executed. The macroscopic layer is for the design and analysis of dynamical protocol aspects. There, we look at chemical reactions from a high-level point of view. Design and analysis methods make use of the direct correspondence between the two layers.

with chemical methods, adopt mathematical tools from chemical kinetics for protocol engineering and bridge the current gap between microscopic execution and macroscopic flow analysis. That is, our goal is to use an artificial chemistry to organize computation and communication in a computer network.

4.2 A TWO-LEVEL ENGINEERING MODEL

Our proposed chemical networking paradigm already reflects the microscopic and macroscopic levels of chemistry in the engineering model. Figure 4.1 illustrates the process of designing and analyzing chemical networking protocols, which we will briefly introduce in this section.

1. Protocols are first and foremost designed at the macroscopic level where we envision the designer to combine reaction network patterns whose dynamic behavior are well known. By synthesizing such motifs in a bottom-up manner, the designer is able to anticipate the behavior of the assembled *abstract reaction network* that may span across several nodes of the computer network.

   abstract reaction network

2. One of the biggest advantages of the chemical engineering model compared to traditional protocol design is that from the abstract reaction network we can automatically generate a *mathematical model* of the protocol's behavior, for example in the form of the stochastic master equation, or as an approximation based on ordinary differential equations.

   mathematical model

3. This description can then be analyzed for properties such as convergence and stability by using *mathematical tools* and methods from chemistry.

   mathematical tools

4. Only in a second step, the protocol designer focuses on the algorithmic aspects of the protocol and tries to find a microscopic *realization* of the protocol in an algorithmic chemistry, in which the structure of the molecules implicitly span the aimed distributed reaction network.

   find realization

5. We provide an execution model for this *chemical software* by organizing computation and packet transmission as chemical reactions; we describe how virtual molecules are structured and how their reaction is simulated and executed on traditional computing infrastructure.

   chemical software

6. *Simulation* results can then be used to refine the algorithmic design on the microscopic level.

   simulation

7. The dynamic behavior of a protocol emerges from microscopic molecular interactions. Chemical reaction laws describe the rate of microscopic reactions, thereby bridging the microscopic to the macroscopic level. In other words, we can automatically *map* a given chemical program back to the abstract chemical reaction network and verify whether the software implements the designed behavior.

   find mapping

In the following, we will provide a quick walk through the main building blocks of our engineering model. Each of the remaining chapters of this part then focuses on one particular aspect.

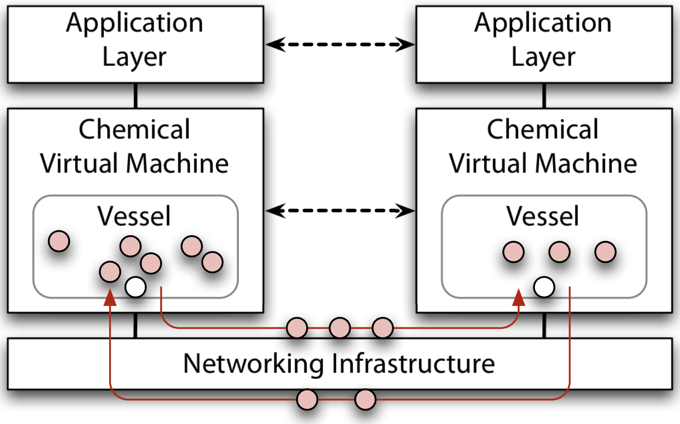


Figure 4.2 **Execution model**: A chemical virtual machine executes programs written in an algorithmic chemistry. Molecules are injected/absorbed by applications and may travel over traditional networking infrastructure to remote vessels.

4.2.1 MICROSCOPIC LEVEL — PROTOCOL EXECUTION

The execution model at the microscopic level (see lower part of Figure 4.1) comprises of an algorithmic chemistry that specifies the structure of the individual molecules (=packets) and how these molecules react. Our goal is to efficiently process data packets on traditional computing and networking infrastructure. Therefore the chemical virtual machine has to be integrated into today's computer architecture as depicted in Figure 4.2 and described below:

chemical virtual machine

The *chemical virtual machine* is a computer program, running on a standard CPU, that simulates chemical reactions at the microscopic level. In the future, we also envision a hardware implementation of the virtual machine. Each virtual machine maintains one or more reaction vessels, each containing a multiset of molecules. Each molecule is represented by a string of symbols. These symbols indicate what kind of computation the virtual machine shall carry out when two molecules (strings) "collide". Molecules (symbol strings) either represent data packets or active code that is executed when reacting with a packet. We extended an existing algorithmic chemistry – Fraglets (Tschudin, 2003) – designed for efficient packet processing. Unlike in traditional computing models where code is executed as fast as the CPU permits, in our chemical execution model, a reaction between two molecules is delayed for a well-defined time. As discussed in the previous chapter, an exact stochastic reaction algorithm schedules the reactions. Hence, the reaction rate follows the *law of mass action*, which is very helpful for the design and analysis of the protocols' dynamics.

law of mass action scheduling

networking infrastructure

In order to bring the communication aspect of protocols into our chemical system, molecules can be exchanged among distant vessels. Our current architecture relies on existing *networking infrastructure*. That is, virtual machines are interconnected by a traditional packet network, e.g. over a bare

Ethernet medium or via an overlay network using the Internet Protocol (IP). A reaction may result in one or more molecules being sent over a communication medium to a connected neighbor node. Hence, information that is stored inside a molecule can be passed from one network node to the next.

A chemical protocol offers its service to an *application layer* program. The application software may run in another chemical reaction vessel or may be programmed with traditional means. The application may at any time inject molecules into the protocol's reaction vessel. These molecules react with the molecules of the chemical network protocol in turn and eventually traverse from the source to the destination node where they are delivered to the remote application.

application layer

In this setting, a chemical protocol replaces parts of the network, the transport and optionally the application layer of a network stack.

MACROSCOPIC LEVEL — ABSTRACTION 4.2.2

When designing chemical networking protocols, we propose to first design the abstract reaction network and analyze its dynamics before trying to find the corresponding symbol strings that generate the required behavior microscopically. Often, we are looking for *equilibrium solutions*: That is, if we manage to find a chemical algorithm for a networking problem in which the solution is represented by a stable equilibrium of the global reaction network, then this solution is resilient to environmental perturbations.

equilibrium solutions

In Chapter 10 we will provide generic rules that can be used to build complex protocol behavior from simple, well-understood design patterns (reaction motifs). We will show, that with some experience, it is quite easy to develop distributed chemical reaction networks for a given problem statement, and that it is a straight forward task to find the corresponding molecule strings for the executable algorithmic chemistry.

It is also possible to obtain the macroscopic model from a given chemical program in order to analyze its dynamic behavior. In Chapter 10 we will show that this model conversion can be automated. At the macroscopic level, we do not care about the structure of molecules; the symbol strings that form the molecules are not important. We therefore map all strings with a similar structure (e.g. packets with different payload) to the same abstract species and model and analyze dynamical properties on this abstract reaction network.

STRUCTURE OF THIS PART 4.3

Our microscopic execution model is an *in silico* implementation of chemical reactions applied to data packets in a computer network. We adopt certain principles from chemistry, which we believe are beneficial for network protocol execution, design and analysis. While these principles come quite natural from the chemical metaphor they require a radical rethinking of network protocol design from the viewpoint of a computer scientist.

In the following chapters, we will delve into the details of those aspects: Chapter 5 describes the execution model, an artificial chemistry for networking. After this, we dedicate four chapters (Chapters 6 to 9) to the analysis of chemical networking protocols. After an introduction in Chapter 6, Chapter 7 studies structural aspects of chemical program analysis. Then, Chapter 8 demonstrates how analytical methods from "wet" chemistry can be adopted, leading to powerful instruments to analyze the dynamics of chemical networking protocols. Finally, in Chapter 9, we apply these methods to analyze a chemical gossip-style aggregation protocol. Chapter 10 then presents first chemical design patterns that allow for a bottom-up synthesis of chemical networking protocols. In Chapter 11, we discuss constraints of realistic computing infrastructure and the tools we used to simulate distributed reaction networks. Finally, Chapter 12 concludes this part with a study of how chemical networking protocols can be integrated into the current Internet.

CHAPTER 5

An Artificial Chemistry for Networking

On the development of an abstract model for distributed chemical computing and on Fraglets, an algorithmic chemistry for chemical protocols.

> The outstanding feature of behavior is that it is often quite easy to recognize but extremely difficult or impossible to describe with precision. [5]
>
> *An Essay of Mind*
> Anatol Rapoport

In this chapter we describe how chemical computation is organized in a distributed setting. As depicted in Figure 5.1, this involves both abstraction levels: the macroscopic and the microscopic level. At the macroscopic level, we extend the formal definition of artificial chemistry given in Chapter 3 for the networking context. The new description models spatially distributed reaction vessels, which exchange molecules over network links by executing reaction rules that generate remote products. At the microscopic level, we define the structure of the molecules, i.e. the syntax and semantics of the chemical programming language.

This chapter is structured into five sections. Section 5.1 defines an artificial chemistry for networking on the macroscopic level. This abstract definition is helpful during the design phase of chemical networking protocols and important for analyzing the dynamic properties of the resulting distributed system. In Section 5.2, we introduce Fraglets, a chemical programming language and algorithmic chemistry where the reactions are implicitly

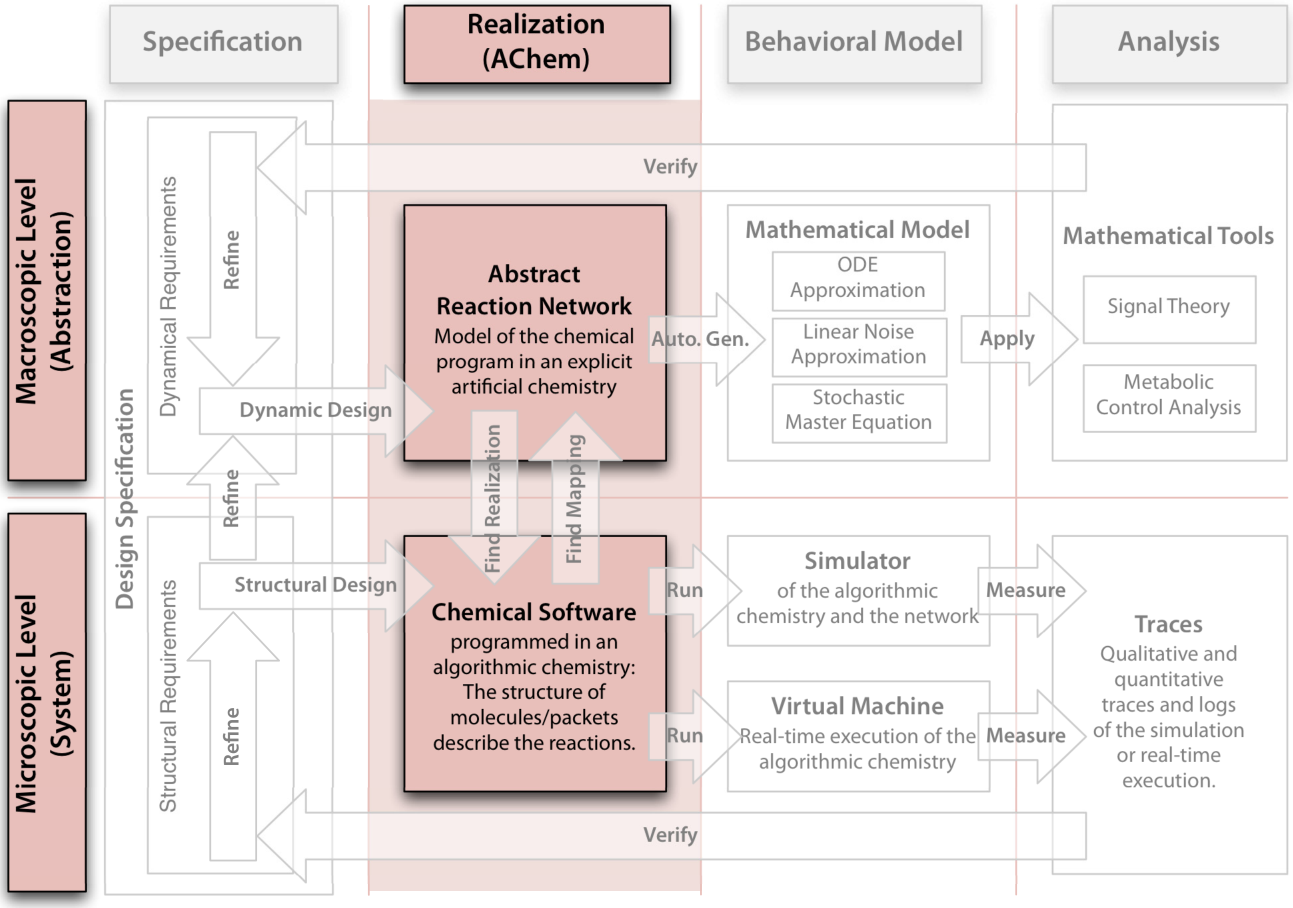

Specification
Realization (AChem)
Behavioral Model
Analysis
Macroscopic Level (Abstraction)
Microscopic Level (System)
Design Specification
Dynamical Requirements
Structural Requirements
Refine
Refine
Refine
Verify
Verify
Dynamic Design
Structural Design
Abstract Reaction Network
Model of the chemical program in an explicit artificial chemistry
Find Realization
Find Mapping
Chemical Software
programmed in an algorithmic chemistry: The structure of molecules/packets describe the reactions.
Auto. Gen.
Mathematical Model
ODE Approximation
Linear Noise Approximation
Stochastic Master Equation
Apply
Mathematical Tools
Signal Theory
Metabolic Control Analysis
Run
Run
Simulator
of the algorithmic chemistry and the network
Virtual Machine
Real-time execution of the algorithmic chemistry
Measure
Measure
Traces
Qualitative and quantitative traces and logs of the simulation or real-time execution.

Figure 5.1 **Artificial chemistries in the engineering model**: The concept of an artificial chemistry is extended for networking. This requires adaptations on the macroscopic as well as on the microscopic level.

specified by the structure of the molecules. All examples and application cases in this thesis are written in Fraglets. In Section 5.3, we show how simple tasks can be solved in Fraglets. Section 5.4 contains the master example of this chapter. We implement and discuss *Diserser*, a gossip-style aggregation protocol that calculates the average of distributed values. This example will also escort us through the remaining chapters of this part. Finally, Section 5.5 summarizes this chapter.

5.1 AN ABSTRACT CHEMICAL MODEL OF COMMUNICATION

In this section, we develop an abstract model for "chemical" communication in a computer network. The goal is to first provide an implementation-agnostic model by defining molecules as abstract species and reactions as generic interaction rules among those molecules. In this sense, we are talking about a "molecule" and mean "packet", but we don't define its internal structure or purpose; we use the term "reaction" to specify how molecules interact without defining the underlying reason for a particular reaction.

Our model is based on the formal definition of artificial chemistries by Dittrich et al. (2001) as introduced in Section 3.1 and formalized in Section 3.3. In the networking context, we have to address several issues that are not prevalent in an isolated, well-stirred vessel: A computer network consists of multiple interconnected nodes. In the language of chemistry, we say that there are spatially distributed reaction vessels. Thus, we first have to represent data transmission in a chemical way. Second, we have to come up with a *distributed* reaction algorithm, which specifies how and when molecules react and are sent from one vessel to another.

We start with a formal definition of our distributed artificial chemistry before we give a simple protocol example within this abstract model.

DEFINITION OF A DISTRIBUTED ARTIFICIAL CHEMISTRY 5.1.1

A distributed artificial chemistry is an artificial chemistry extended by a network graph. It is defined as quadruple $\mathcal{DAC} = (\mathcal{G}, \mathcal{S}, \mathcal{R}, \mathcal{A})$, where $\mathcal{G}$ is the computer network graph, $\mathcal{S}$ is the set of all molecular species in the network, $\mathcal{R}$ is the set of all reaction rules, and $\mathcal{A}$ is the global view on the distributed reaction algorithm. These four components deserve some more discussion.

Network of Distributed Nodes/Vessels (a)

computer network

network link

adjacency matrix

neighborhood

We define the *computer network* as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$ is the set of all nodes in the network. The edges $\mathcal{E} = \{e_1, \ldots, e_{|\mathcal{E}|}\}$ are unidirectional *network links* and connect neighbor nodes. Node v_j is a neighbor of node v_i iff there exists an edge $(v_i, v_j) \in \mathcal{E}$. In this case, we define $\text{adj}(i, j) = 1$, otherwise, if node v_i is not connected to node v_j, $\text{adj}(i, j) = 0$. The *adjacency matrix* is the square $|\mathcal{V}| \times |\mathcal{V}|$-matrix $A = [a_{ij}]$ where $a_{ij} = \text{adj}(i, j)$. We further define the *neighborhood* of a network node v_i as the set of its neighbor vessels:

$$\mathcal{N}_i = \{v_j \in \mathcal{V} \mid \text{adj}(i, j) = 1\} \tag{5.1}$$

The communication network that interconnects the nodes describes a high-level structure, conceptually one topological layer above the reaction network inside a node.

(b) The Structure of Distributed Chemical Reaction Networks

network node

local artificial chemistry

Inside each *network node* $v_i \in \mathcal{V}$, an algorithm $\mathcal{A}_i$ updates a local multiset of molecules according to a set of local reaction rules. That is, each node v_i defines a *local artificial chemistry* as the triple $(\mathcal{S}_i \cup \mathcal{S}_i^{(j)}, \mathcal{R}_i, \mathcal{A}_i)$. The set $\mathcal{S}_i$ defines the species of all molecules that can possibly be found in the local multiset. The set $\mathcal{S}_i^{(j)} = \bigcup_{j \in \mathcal{N}_i} \mathcal{S}_j$ contains all local species of v_i's neighbors. Each node also defines its own set of reaction rules $\mathcal{R}_i$ where a reaction rule $r_i \in \mathcal{R}_i$ is specified by a pair of multisets (reactants, products) and an assigned reaction coefficient

$$r_i\colon \quad \mathcal{M}_{\text{in},r_i} \xrightarrow{k_i} \mathcal{M}_{\text{out},r_i} \qquad \forall r_i \in \mathcal{R}_i \tag{5.2}$$

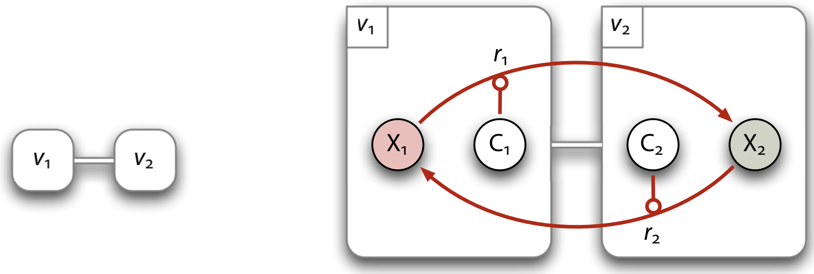
transmission

A node only has access to its local multiset. Consequently, we require that all reactants are local species: $\mathcal{M}_{\text{in},r_i} \in \mathcal{M}(\mathcal{S}_i)$. On the other hand, a reaction may produce molecules in neighbor reactors: $\mathcal{M}_{\text{out},r_i} \in \mathcal{M}(\mathcal{S}_i \cup \mathcal{S}_i^{(j)})$. This is the way we model *transmission* in an abstract chemical way: by allowing a reaction to create products in a neighbor vessel. Thus, in addition to a strictly local reaction network, each node also defines extended reaction rules that send molecules to other nodes in its neighborhood.

We require that all nodes run the same algorithm, $\mathcal{A}$, in order to allow the protocols to reach consensus. As mentioned before, the algorithm operates on the local multiset $\mathcal{M}_{v_i} \in \mathcal{M}(\mathcal{S}_i)$. The multiset is either updated by a local reaction, driven by the local reaction algorithm, or by receiving a molecule from a neighbor node as a consequence of a reaction rule executed there.

(c) Example

In Chapter 3, we studied a simple reversible reaction (see Figure 3.2 on page 35). Let us now install such a reversible reaction among two nodes with the distributed artificial chemistry $\mathcal{DAC} = \{\mathcal{G}, \mathcal{S}, \mathcal{R}, \mathcal{A}\}$ with network graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ as depicted in Figure 5.2(a) where $\mathcal{V} = \{v_1, v_2\}$ and $\mathcal{E} = \{(v_1, v_2), (v_2, v_1)\}$. Each node v_i defines two molecular species, C_i and X_i. Thus, the overall set of species is $\mathcal{S} = \{\mathrm{C}_1, \mathrm{X}_1, \mathrm{C}_2, \mathrm{X}_2\}$. Either node defines a single reaction rule that sends one instance of the local X-molecule to its neighbor, catalyzed by the local species C. The overall distributed reaction



(a) Network graph (b) Distributed reaction network graph

Figure 5.2 Reaction network of a simple distributed artificial chemistry: The distributed reaction network is spanned over a network topology by a set of local molecular species and a set of local and/or local-to-neighbor reactions.

network is described by the set $\mathcal{R} = \{r_1, r_2\}$, shown in 5.2(b) and as chemical reaction equation below.

$$r_1\colon \quad C_1 + X_1 \longrightarrow C_1 + X_2 \tag{5.3a}$$

$$r_2\colon \quad C_2 + X_2 \longrightarrow C_2 + X_1 \tag{5.3b}$$

The local algorithm in node v_1 sends an X-molecule to neighbor v_2 by executing reaction r_1. Similarly, node v_2 sends the X-molecule back by executing reaction r_2. In order to complete our definition of a distributed artificial chemistry we have to specify how the local algorithm decides when to execute which reaction such that distributed reversible reaction exhibits the same dynamic behavior as depicted in Figure 3.4 on page 38.

5.1.2 A DISTRIBUTED EXACT STOCHASTIC REACTION ALGORITHM

In traditional queuing networks, packets are sent whenever the link is idle. Here, we aim for a controlled scheduling of reactions based on results from chemical kinetics. The algorithm shall induce an emergent dynamic behavior that is as close to the dynamics of real chemical reactions as possible, but simple enough to enable an efficient implementation on a traditional *von Neumann* machine.

In Section 3.3.3, we studied such an algorithm in detail – the Next Reaction Algorithm by Gibson and Bruck (2000). However, in our distributed artificial chemistry, there is *no centralized algorithm* that globally decides which reaction in which network node is executed next and when. Instead, each node has to run its own algorithm. The distributed algorithms must work together such that the behavior of the global reaction network is coherent.

no centralized algorithm

1. Initialization:
   a) the current physical time is t_{now};
   b) for each reaction rule $r_j \in \mathcal{R}_i$, calculate the propensity function a_j according to (5.4);
   c) for each reaction rule $r_j \in \mathcal{R}_i$, draw the reaction interval from an exponential distribution: $\tau_j \sim \text{Exp}(1/a_j)$;
   d) store the putative reaction time values $t_j = t + \tau_j$ in an indexed priority queue Q_i (see Gibson and Bruck (2000)).
2. Let r_μ be the reaction rule whose putative reaction time, t_μ, stored in Q_i is least.
3. Wait as long as $t_{\text{now}} < t_\mu$.
4. Execute reaction rule r_μ, i.e. rewrite the vessel's multiset according to the reaction vector.
5. Update the next reaction time of all those reaction rules r_j that depend on the executed reaction. That is, all those reaction rules have to be adjusted that have reactants that were changed by reaction r_μ. Formally, for each reaction rule $r_j \in \left\{ r \in \mathcal{R}_i \mid \operatorname{dom} r \cap \left\{ s \in \mathcal{S}_i \mid \gamma_{s,r_\mu} \neq 0 \right\} \neq \varnothing \right\}$,
   a) calculate the propensity function, a_j, according to (5.4);
   b) if $j \neq \mu$, adjust the next reaction time without drawing a new random variable: $t_j \leftarrow t_{\text{now}} + (a_{j,\text{old}}/a_{j,\text{new}})(t_j - t_{\text{now}})$;
   c) if $j = \mu$, draw a new reaction interval from an exponential distribution, $\tau_j \sim \text{Exp}(1/a_j)$, and set the next reaction time to $t_j \leftarrow t_{\text{now}} + \tau_j$;
   d) replace the old value t_j in Q_i with the new value and re-sort Q.
6. Goto step 2.

Algorithm 5.1 Real-time Next Reaction Method: Schedules and triggers reaction execution within a network node $v_i \in \mathcal{V}$. The algorithm has a notion of physical time, t_{now}, which is continuously updated outside of the algorithm, e.g. by the hardware.

(a) Our Extended Real-time Next Reaction Method

We adapted the exact stochastic *Next Reaction Method* by Gibson and Bruck (2000) (see Algorithm 3.2) for the distributed context by the changes highlighted in Algorithm 5.1:

real-time scheduling

First, since our purpose is not to simulate real chemical reactions as fast as possible, but to use the algorithm to drive execution *online*, we schedule the reactions in *real-time*. This is, we assume that there is a physical clock that is continuously incremented by the hardware. Step 3 of the algorithm synchronizes the simulation time to the physical time by sleeping until the physical time reaches the next reaction time.

simplified propensity function

The second modification is to use a simpler method to calculate the *propensities*. The occurrence probability of reaction r is now given as

$$a_r\big(\mathbf{N}(t)\big) = k_r \prod_{\{s \in \mathcal{S}\}} N_s(t)^{\alpha_{s,r}} \tag{5.4}$$

We directly use the macroscopic reaction coefficient k_r and to ignore the factorial denominator in (3.10), which originally stems form the combinatorial choice of molecule instances.

packet reception

Third, we have to define the influence of a received molecule to the scheduled reactions. Unlike Gillespie's algorithm, the Next Reaction Method tracks the occurrence time of each reaction separately. Thus, the reception handler just calls step 5 of the algorithm upon receiving a new packet. This automatically and correctly adjust the occurrence time of those reactions that depend on the arrived species.

Discussion (b)

time synchronization

one-to-one mapping

In the networking context each virtual reaction vessel interacts with the environment. Thus, each device has to map the calculated virtual time to the physical time in a coherent way to present consistent reaction times. This is the reason for introducing step 3 to the original algorithm. This delay is used to *synchronize the virtual time* dictated by the law of mass action to the physical time in which communication with neighbor nodes take place. The simplest way of doing this is to assume a *one-to-one mapping* (the virtual time *is* the physical time), and to just sleep for the time remaining until the next reaction occurs. Other proportional mappings are possible, too, as long as all nodes of the network agree. For example, one virtual second could be mapped to one physical millisecond, speeding up the reactions by a factor of 1000.

upper bound

However, virtual time steps can become arbitrarily small as the reactant concentrations grow. Our simple time mapping algorithm obviously requires that there is an *upper bound* in the number of molecules in the reactor, above which the CPU is unable to keep the pace of the simulated chemistry. For now, we assume that the CPU is fast enough to drive the reactions in time. In Chapter 11, we will provide a method to limit the number of molecules in a dynamic way.

Another difference to the original algorithm is to use a simplified propensity function. In fact, we loose microphysical plausibility for reactions among multiple instances of the same reactants. However, we gain a more direct mapping from the microscopic world of reaction probabilities and stochastic reaction coefficients to the macroscopic and deterministic description of law of mass action dynamics. Since we are not simulating real chemistry, we have the freedom to design both, the system *and* its model, such that we can afford this small deviation from natural behavior.

The abstract execution model described so far may be implemented in a traditional programming language for an explicitly defined set of abstract molecules and reactions. But instead, in the next section, we present an execution model based on an algorithmic chemistry, where molecular species and reactions are implicitly defined by the structure of symbol strings. This allows the system to define the reactions at run-time.

5.2 FRAGLETS — A PROGRAMMING LANGUAGE FOR CNPS

Fraglets (Tschudin, 2003) is an execution model for communication protocols inspired by chemical reactions and designed for fast packet header processing. In this section, we present Fraglets from different angles: First, in Section 5.2.1 we follow the path of Tschudin (2003) and introduce Fraglets as a string rewriting system. Section 5.2.2 then defines Fraglets as an artificial chemistry in the sense of Dittrich et al. (2001) and makes the connection to the previous section by pointing out that in Fraglets, the reaction rules are defined implicitly by the structure of the symbol strings currently present in the vessel multiset. Section 5.2.3 highlights how the characteristics of the Fraglets reaction algorithm influence the convenience of the protocol design process. Finally, Section 5.2.4 considers Fraglets an assembler-like programming language and discusses its instruction set.

5.2.1 STRING REWRITING IN A MULTISET CONTEXT

In computer networking, the most frequently executed action on a data packet is the rewriting of header fields. For example, on each leg of a packet's route through a sequence of Ethernets, the packet must obtain a new destination field to reach the next hop. A *fraglet* is a small fragment of code or data, represented by a string of symbols, e.g. [exch a b c d]. Fraglets interact among each other akin to molecules. This interaction is driven by the header symbols of the packets.

fraglet

Fraglets is a combination of multiset rewriting system (Banâtre & Le Métayer, 1986) and tag system (Post, 1943; Minsky, 1967), a string rewriting system in which the leftmost symbol identifies the rule to apply. For example, the rule

$$[\mathtt{exch}\ \sigma_1\ \sigma_2\ \sigma_3\ \sigma_4\ \ldots\ \sigma_n] \Rightarrow [\sigma_1\ \sigma_3\ \sigma_2\ \sigma_4\ \ldots\ \sigma_n] \tag{5.5}$$

when applied to [exch a b c d] will result in [a c b d] — that is, two symbols are swapped. The exch symbol acted as a prefix command for the rest of

Tag	Production Rule
nul	[nul Φ] $\Rightarrow \varnothing$ (fraglet is removed)
exch	[exch σ α β Φ] $\Rightarrow$ [σ β α Φ]
dup	[dup σ χ α Φ] $\Rightarrow$ [σ α α Φ]
fork	[fork σ τ Φ] $\Rightarrow$ [σ Φ] + [τ Φ]
split	[split Φ_* $*$ Ψ] $\Rightarrow$ [Φ_*] + [Ψ]
send	$_{v_i}$[send v_j Φ] $\Rightarrow$ $_{v_j}$[Φ] (send to neighbor v_j)

$\alpha, \beta, \sigma, \tau, \chi \in \Sigma$ (symbols), $\Phi, \Psi \in \Sigma^*$, (symbol string of arbitrary length), $\Omega_* \in \{\Sigma \backslash *\}^*$ (symbol string of arbitrary length not containing an asterisk symbol), $v_i, v_j \in \mathcal{V}$ (network node identifiers).
Our dup-rule differs from the original specification in Tschudin (2003) in order to avoid possible infinite loops when executing the fraglet [dup dup dup].

Table 5.1 Subset of elementary Fraglets transformation rules: They are applied to single fraglets matching the first symbol

Tag	Production Rule
match	[match σ Φ] + [σ Ψ] $\Rightarrow$ [Φ Ψ]
matchp	[matchp σ Φ] + [σ Ψ] $\Rightarrow$ [matchp σ Φ] + [Φ Ψ]

$\sigma \in \Sigma$ (symbol), $\Phi, \Psi \in \Sigma^*$, (symbol string of arbitrary length)

Table 5.2 Subset of Fraglets synchronization rules: They are applied to a pair of fraglets matching their headers

the word whereas the leftmost symbol “a” serves as a continuation pointer for further processing the result.

string rewriting system

alphabet

production rule

Formally, a *string rewriting system* is a pair $(\Sigma, \mathcal{P})$, where Σ is a finite *alphabet* of symbols and $\mathcal{P}$ is a set of production rules. A *production rule*, $p \in \mathcal{P}$, $p\colon \Sigma^* \times \Sigma^*$, is a string substitution pattern that operates on words $w \in \Sigma^*$. Unlike Post's original tag system (1943), which operates on one initial word and asks about the system's expansion, we place ourselves in a multiset context where the production rules are applied to all words in a multiset. Each network node (reaction vessel) implements a multiset of fraglets. The node's scheduling algorithm continuously examines the fraglets in the multiset and selects the fraglets to be processed.

transformation rule

synchronization rule

There are two types of rewriting rules ($\mathcal{P} = \mathcal{P}_\mathrm{T} \cup \mathcal{P}_\mathrm{S}$): (i) *Transformation rules* ($\mathcal{P}_\mathrm{T}$) rewrite a single fraglet into one or more fraglets, potentially sending them to neighbor nodes; (ii) *synchronization rules* ($\mathcal{P}_\mathrm{S}$) combine two fraglets by concatenating them together. Analogous to chemistry, transformations implement unimolecular reactions whereas synchronizations can be treated a molecular collisions. A subset of the elementary transformation rules is shown in Table 5.1 whereas Table 5.2 lists some synchronization rules.

Note the difference between a production rule and the resulting transformation of a concrete fraglet instance. The fraglet instance [fork a b c] is transformed to the fraglets [a c] and [b c] by the fork production rule. We use the double arrow ($\Rightarrow$) to denote a prototypical production rule and the single arrow (→) for its application to a fraglet instance. Because the first symbol of the fraglet uniquely identifies the production rule being applied, we usually don't mention the rule explicitly. Hence, the following two notations are used synonymously:

$$\texttt{[fork a b c]} \xrightarrow{\texttt{[fork } \alpha\ \beta\ \Phi\texttt{]}\Rightarrow\texttt{[}\alpha\ \Phi\texttt{]+[}\beta\ \Phi\texttt{]}} \texttt{[a c]} + \texttt{[b c]} \qquad (5.6a)$$

$$\texttt{[fork a b c]} \longrightarrow \texttt{[a c]} + \texttt{[b c]} \qquad (5.6b)$$

tag matching

Synchronization rules pick the molecules based on *tag matching*. The second symbol, σ, of the fraglet [match σ ...] must be identical to the first symbol of the fraglet to synchronize with, [σ ...]. By restricting ourselves to exact tag matching instead of using more complex schemes such as pattern matching, we are able to limit the computational complexity to find corresponding synchronization partners.

5.2.2 FRAGLETS AS DISTRIBUTED ARTIFICIAL CHEMISTRY

Apart from seeing Fraglets as a string rewriting system, we can make the link to our distributed artificial chemistry: Fraglets is a realization of a distributed artificial chemistry $\mathcal{DAC} = (\mathcal{G}, \mathcal{S}, \mathcal{R}, \mathcal{A})$ where $\mathcal{G}$ is the graph of network nodes running the Fraglets rewriting system. The set of species $\mathcal{S}$ is defined implicitly as the set of all possible words w of arbitrary length over the symbol alphabet: $\mathcal{S} = \left\{ w \mid w \in \Sigma^* \right\}$. Note that every distinct string is a separate species even if two fraglets start with the same symbol. For example, fraglet [match X Y] is a different species than fraglet [match X Z], because the first one produces [Y...] whereas the latter generates a fraglet [Z...].

The set of reaction rules, $\mathcal{R}$, is also defined implicitly by the finite set of production rules of Fraglets, $\mathcal{P}$, together with the current set of fraglets in the vessel multiset. Because both, the molecules as well as the reactions are defined implicitly, the chemistry is able to construct new molecules and is therefore an *algorithmic chemistry* (see Section 3.2)

algorithmic chemistry

Our extended *Next Reaction Method* is used as a reaction algorithm $\mathcal{A}$. Thus, the dynamics of rewriting steps (reactions) is a random process, on average governed by the law of mass action.

5.2.3 DYNAMIC BEHAVIOR OF FRAGLETS PROGRAMS

In this work we put strong emphasis on the dynamic behavior of networking protocols. An engineer always designs the global dynamics of the distributed reaction network first. Afterwards, in a second step, he/she translates this macroscopic description to the microscopic level by finding the corresponding fraglets, which generate the desired reaction network. We will elaborate on this in Chapter 10. In addition to the dynamic behavior, network protocols also have to perform functional tasks on the microscopic layer, such as inspecting and modifying header fields.

We would like to disentangle the functional from the dynamical behavior of protocols, meaning that an engineer has to be able to change the functional operation of the fraglets on the microscopic, symbolic level without affecting the previously designed macroscopic dynamic behavior. This is a challenging requirement since the two layers are intertwined as follows:

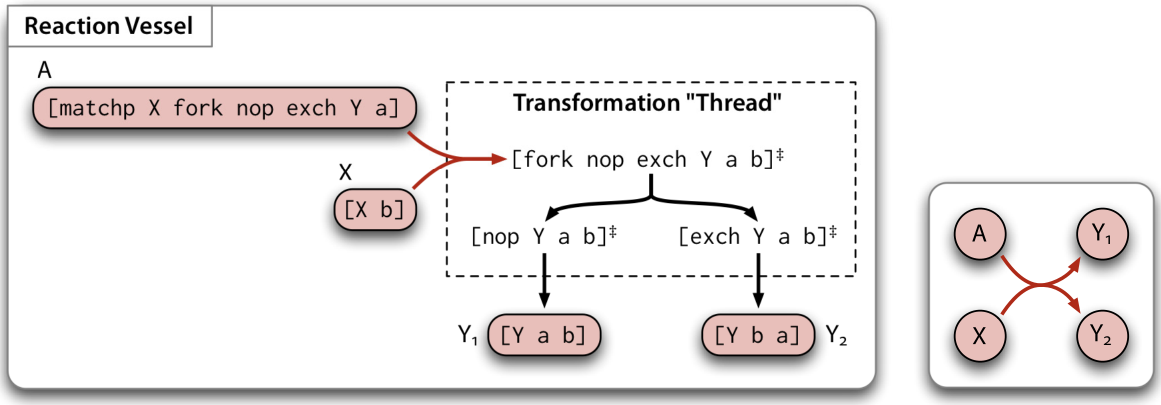
Protocol behavior emerges from the execution of Fraglets in a bottom-up manner: The global distributed reaction network results from the structure of the constituting fraglets. By scheduling reactions according to the law of mass action, the structure of the reaction network is directly linked to its dynamic behavior. The change of a single fraglet may therefore lead to a completely different dynamic behavior.

Thus there is a conflict between top-down engineering requirements and the bottom-up stiffness caused by the imperative effect of microscopic changes to the macroscopic behavior. Our experiments with the original Fraglets instruction set and dynamic behavior revealed that is was very hard to find a Fraglets program that brings about the requested dynamics *and* performs a certain function on the microscopic, symbolic level.

With the goal to decouple the dynamics of the reaction network from the underlying implementation as much as possible, we decided that transformation rules such as `exch`, `dup`, `split`, and `send` are applied immediately. This shall give the designer the freedom to change the microscopic function without affecting the macroscopic dynamics.

Immediate Execution of Transformations *(a)*

Fraglet strings starting with transformation symbols are not scheduled by the law of mass action scheduler but are rather treated as transient molecules that undergo immediate transformation until the first symbol of a fraglet is a synchronization instruction (such as `match`) or a non-instruction tag (e.g. `x`). Figure 5.3 shows an example where a reaction among the fraglets `[match X...]` and `[X b]` is scheduled according to the law of mass action, producing a transient fraglet starting with symbol `fork`. This transient fraglet and all its successors are immediately transformed to their "normal forms", meaning that they are iteratively transformed until no transformation rules



(a) Fraglets rewriting steps

(b) Reaction network

‡: transient fraglets, reduced immediately

Figure 5.3 **Transformations are executed immediately**: (a) Two fraglets are joined by a synchronization rule; the resulting fraglet is eventually reduced by transformation rules to several fraglets in normal form. (b) On the macroscopic level, we can ignore all transformations to obtain the reduced reaction network, which is dynamically equivalent to the Fraglets rewriting system in (a).

transient fraglet

normal form

can be applied anymore. Formally, a fraglet $w \in \Sigma^*$ is *transient* iff $\exists p \in \mathcal{P}_\mathrm{T}$ such that $w \xrightarrow{p} v$; otherwise the fraglet is in its *normal form*.

The normalized products [Y a b] and [Y b a] are returned to the reaction vessel multiset where the law of mass action scheduler re-calculates the next reaction time of all dependent reactions.

AlChemy

Fontana (1992) used a similar approach (see also Fontana & Buss, 1994): In *AlChemy*, molecules are represented by λ-expressions that, when colliding, are applied to each other and undergo as many β-reduction steps as necessary to reach another λ-expression in normal form.

Note however, that with the immediate application of transformation rules, we are facing another problem: We have to avoid endless transformation loops; that is, we have to make sure that a sequence of transformations always terminates in either an empty fraglet or a synchronization symbol at the fraglets' heads. This requirement has a direct impact on the design of the Fraglets instruction set as discussed below.

5.2.4 THE FRAGLETS INSTRUCTION SET

instruction

thread

A fraglet can be seen as a passive object on which active rewriting rules – rooted in the Fraglets system – operate. An alternative point of view attributes the symbols an active role by regarding the header symbol as assembler-like *instruction*. Hence, a fraglet is a fragment of code (a list of transformation instructions), executed in its own "thread", independently from the other fraglets in the reactor. As soon as the first symbol becomes a synchronization symbol (such as `match`) the thread asks for synchronization with another thread, which starts with the requested tag.

In this sub-section, we review the original design decision of the Fraglets *instruction set*, i.e. the set of rewriting rules, and introduce extensions that were necessary for this thesis: more powerful instructions were added and a new variant of instructions, stack instructions, operate on the tail of the fraglet. As we will see later, stack instructions are very helpful to decouple the dynamic design of protocols from their concrete symbolic implementation. Appendix B lists all Fraglets instructions.

instruction set

History and Development (a)

The original Fraglets instruction set was designed for the typical tasks of efficient packet header processing (Tschudin, 2003). In order to be able to process packets at wire-speed the effort to determine the rewriting rule to apply must be bound; we cannot afford a complex pattern-matching scheme. This is the reason why only the first few header symbols are considered and why synchronization between two fraglets is based on exact tag matching.

The initial instruction set defined by Tschudin (2003) mainly contains instructions for packet header processing, i.e. moving symbols around in the header, splitting (`split`) and joining (`match`, `matchp`) packets; the `send` instruction is used to send fraglets to distant nodes.

Later, Yamamoto et al. (2007) added integer symbol types, accompanied by *arithmetic instructions* that operate on them (`sum`, `diff`, `mult`, `div`), *conditional operations* to branch the execution (`eq`, `lt`) and *inspection*-instructions that return contextual information, such as the length of a fraglet (`length`) or the identifier of the network node (`node`).

arithmetic, conditional, and inspection instructions

In the present work, we recognized the potential of using the dynamics of chemical reaction networks to perform networking tasks. This has consequences to the design of the instruction set. For example, the decision to execute transformation instructions immediately requires that there must be no transformation loops. Therefore we reviewed all existing instructions and modified and extended some of them to obey some general rules:

Transformations Reduce the Fraglet Length (b)

We redesigned the instruction set such that every transformation reduces the fraglet by at least one symbol. That is, a transient fraglet of length *l* will be in its normal form after at most *l* transformation steps. Consequently, we modified instructions that produce new symbols such that they now require

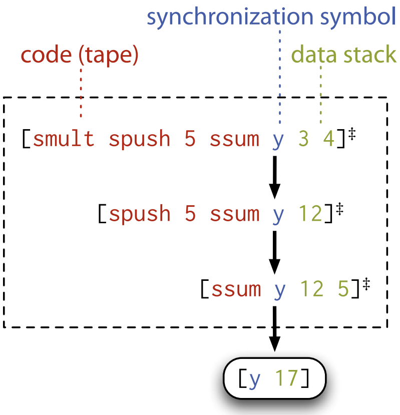


Figure 5.4 Stack instructions operate on the fraglet tail: A fraglet can be treated as virtually separated into code tape (head) and data stack (tail). Stack instructions, executed from the top operate on the tail of the fraglet.

‡: transient fraglets, reduced immediately

a dummy operand that is just deleted. For instance the dup instruction that duplicates a symbol (α) additionally deletes the third symbol (χ):

$$[\texttt{dup}\ \ \sigma\ \ \chi\ \ \alpha\ \ \Phi] \Rightarrow [\sigma\ \ \alpha\ \ \alpha\ \ \Phi] \tag{5.7}$$

(c) Stack Instructions

Above, we came to the conclusion that if it shall be possible to construct a predesigned reaction network by the structure of fraglets *and* if the fraglets shall also do useful computation on the microscopic, symbolic level, we must have the possibility to execute several transformations in one reaction step. Thus, the transformations must have enough expressive power to perform complex operations, *nota bene* without the possibility of looping. This is hardly possible with the original instruction set, because both, the instructions as well as their operands were located in the header of the fraglet. We decided to introduce stack instructions where the head of the fraglet is treated as the code "tape" whereas the tail is treated as data stack.

Push programming language

Figure 5.4 shows such a Fraglet starting with a chain of multiple stack instructions; the fraglet computes the arithmetic expression $y = f(x, z) = xz + 5$. The values for $x = 3$ and $z = 4$, located in the fraglet's tail, are multiplied by the smult instruction: The operands are popped from the stack and the result, 12, is pushed back in turn. Since the operands and the result do not clutter up the head of the fraglet, the next stack transformation instruction may directly follow the first one and consequently, the stack instructions do not need a continuation symbol (such as σ in Table 5.1). Table 5.3 shows an excerpt of the provided stack transformation rules. They are closely inspired by the instruction set of the *Push programming language* (Spector, Perry, Klein, & Keijzer, 2004). Appendix B provides a list of all Fraglets instructions.

Tag	Production Rule
sexch	$[\texttt{sexch}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \alpha\ \beta]$
sdup	$[\texttt{sdup}\ \chi\ \Phi\ \alpha] \Rightarrow [\Phi\ \alpha\ \alpha]$
ssum	$[\texttt{ssum}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ (\alpha+\beta)]$
smult	$[\texttt{smult}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ (\alpha*\beta)]$
seq	$[\texttt{seq}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ (\alpha=\beta\ ?\ 1:0)]$
sif	$[\texttt{sif}\ \sigma\ \tau\ \Phi\ \alpha] \Rightarrow [(\alpha\neq 0\ ?\ \sigma:\tau)\ \Phi]$

$\alpha, \beta, \sigma, \tau \in \Sigma$ (symbols), $\Phi \in \Sigma^*$, (symbol string of arbitrary length).

Table 5.3 Subset of Fraglets stack transformation rules: Operands are popped from the tail, results are pushed to the tail by stack instructions

Summary (d)

The continuous evolution of the Fraglets instruction set has been driven by practical needs so far. Each time we struggled with the current instructions while developing a protocol implementation we added missing instructions that simplified the solution. The original basic instruction set is believed to be Turing complete* but lacks the expressiveness needed to perform complex symbolic operations in protocols with ease.

We introduced stack instructions in order to chain several transformations without the need to synchronize the fraglet with another one. Furthermore, we identified a simple rule for the design of new instructions in order to avoid endless transformation loop: each transformation must reduce the length of the fraglet.

In the future, we have to consolidate the instruction set. Several instructions could be replaced by more general ones doing the same. At the moment, Fraglets runs in a simulator that handles instructions as abstract symbols. We could envision a hardware implementation of the Fraglets virtual machine. In this case we have to develop an instruction set architecture and define a binary encoding scheme for Fraglets. In Chapter 18, we will delve more into such an encoding scheme when studying the mutational robustness of Fraglets programs.

PROGRAMMING IN FRAGLETS 5.3

In this section, we give a few examples how Fraglets may be used to realize traditional networking tasks, such as implementing finite state machines, or transmit packets to a distant node by an active networking approach to source routing. These examples do not make use of the dynamical properties of the law of mass action scheduling but only show the algorithmic aspect of parallel and rule-based computing in Fraglets. In the next section, we will present an example that fully exploits the dynamics of chemical reaction kinetics.

*The proof for the Turing completeness of Fraglets has not been published yet.

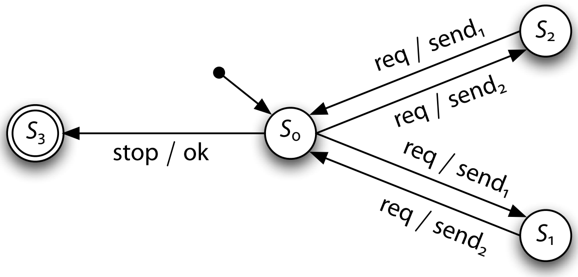


Figure 5.5 Non-deterministic finite state machine: The FSM randomly decides to which of the two nodes it sends a message.

req: transmission request from the application; send$_j$: send message to neighbor node v_j; ok: termination request from the application

5.3.1 FINITE STATE MACHINE

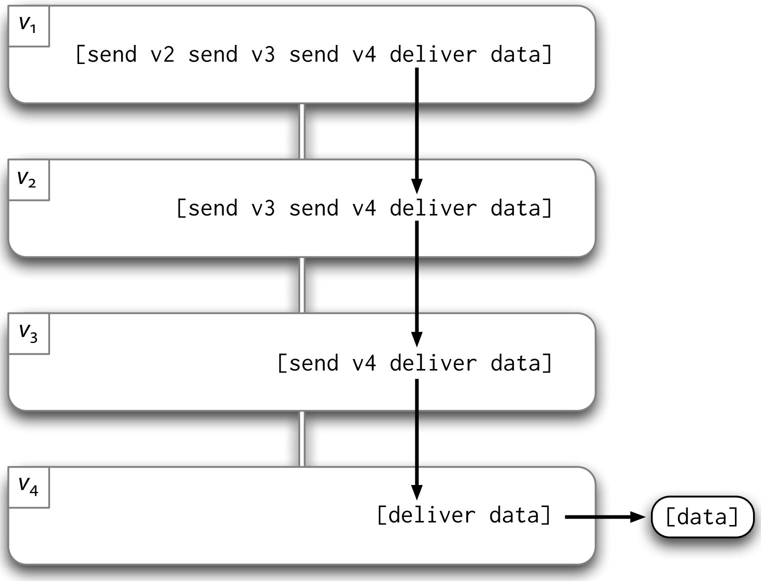
Finite state machines (FSM) are integral parts of today's networking protocols. Often, the logic of protocols is realized as a FSM that is triggered by events (see for example the specification of TCP (Postel, 1981) or BGP (Rekhter et al., 2006)). Events materialize as incoming packets, service requests from the application, or internally generated events, such as elapsed timers.

Let us implement a state machine for a simple protocol that distributes messages from the application equally to one of two nodes. Figure 5.5 shows a non-deterministic FSM that complies with this specification.

In Fraglets, we may represent the current state by the presence of one instance of molecular species [S0], [S1], [S2], or [S3]. Likewise, each event is represented by an individual molecular species, [req] and [stop], respectively. For each transition, we install an active species that atomically reacts with two molecules, a state and an event molecule, generates the molecule for the destination state, and performs the action, i.e. sends the consumed request molecule to the corresponding node. Generating the Fraglets program according to this recipe is straightforward:

```
[mmatchp 2 S0 req split S1 * send v1 req]
[mmatchp 2 S0 req split S2 * send v2 req]
[mmatchp 2 S0 stop S3]
[mmatchp 2 S1 req split S0 * send v2 req]
[mmatchp 2 S2 req split S0 * send v1 req]
[S0]      (initial state)
```

The nondeterministic transition from state S_0 to either S_1 or S_2 is achieved by having two mmatchp-molecules that compete for the same state/event pair. The reaction algorithm randomly decides which of those molecules it picks.



v1
[send v2 send v3 send v4 deliver data]
v2
[send v3 send v4 deliver data]
v3
[send v4 deliver data]
v4
[deliver data]
[data]

Figure 5.6 Source Routing in Fraglets: A concatenation of send instructions guides the fraglet through the network.

According to the law of mass action, the reaction probability distribution follows the distribution of their multiplicities.

SOURCE ROUTING WITH AN ACTIVE NETWORKING APPROACH 5.3.2

Source routing allows the sender of a packet to determine the packet's path through the network. In Fraglets, the tail of a fraglet starting with the send symbol will be sent to the specified neighbor node. Since the tail may also contain instructions, those instructions will be executed in the neighbor node, enabling a lightweight dissemination method for code. Such an *active networking* approach does not require any code to be pre-installed in the intermediate nodes.

active networking

The following fraglet sends data to a destination node by concatenating send instructions.

```
[send v2 send v3 send v4 deliver data]
```

Figure 5.6 shows a simple network topology and the fraglet's transformation on its way through the network. When reaching the destination node the remaining fraglet delivers its data tail to the application using the `deliver` instruction.

5.4 APPLICATION CASE: DISPERSER — A GOSSIP-STYLE AGGREGATION PROTOCOL

In this section, we present *Disperser*, a chemical networking protocol that calculates the average of distributed values. Unlike the previous Fraglets examples, this protocol implementation highlights one of the key benefits of our chemical system: the ability to outsource the computation of the average value to the dynamics of the distributed reaction system, which is enabled by the law of mass action scheduling.

5.4.1 THE GOSSIP-STYLE PUSH-SUM PROTOCOL

Push-Sum

Recently, gossip-based or epidemic protocols gained attention because of their potential to disseminate information in a robust way. For example, the *Push-Sum* protocol (Kempe et al., 2003) averages out locally stored values by means of a simple local algorithm: Node v_i stores sum and weight as tuple (s_i, w_i) starting with $(y_i(0), 1)$ where $y_i(0)$ is the node's initial value. In each round, i.e. after a fixed time interval, each node v_i sends the tuple $\left(\frac{1}{2}s_i, \frac{1}{2}w_i\right)$ to a randomly chosen neighbor and, at the same time, to itself. During the next interval, each node v_i collects the tuples $\left\{(s_{i,j}, w_{i,j})\right\}$ received from its neighbors v_j and sums them up, $s_i = \sum_j s_{i,j}$ and $w_i = \sum_j w_{i,j}$. Hence, the fraction $y_i = s_i/w_i$ *asymptotically approaches the average* of the distributed initial values.

asymptotically approaches the average

5.4.2 A CHEMICAL DISPERSER PROTOCOL

For a chemical implementation of the averaging protocol, we make use of the distributed reversible reaction introduced previously (see Figure 5.2 on page 55). This equilibrium reaction balances the number of molecules between two nodes. Our goal is to expand this mechanism to multiple nodes in order to reach a global equilibrium where each node contains the same (the average) number of molecules.

Figure 5.7 shows the chemical *Disperser* protocol in a simple but non-trivial network topology of 4 nodes. Generally, each node $v_i \in \mathcal{V}$ defines the molecular species $\mathcal{S}_i = \left\{\mathrm{C}_{i,j}, \mathrm{X}_i\right\}$. The quantity of X_i-molecules represents the computed average, $y_i(t) = N_{\mathrm{X}_i}(t)$, which is initially set to the local value $y_i(0)$. For each link $(i,j) \in \mathcal{E}$ to v_i's neighbors there is a single instance of molecule $\mathrm{C}_{i,j}$ that reacts with the local X_i-molecule and, by doing so, sends

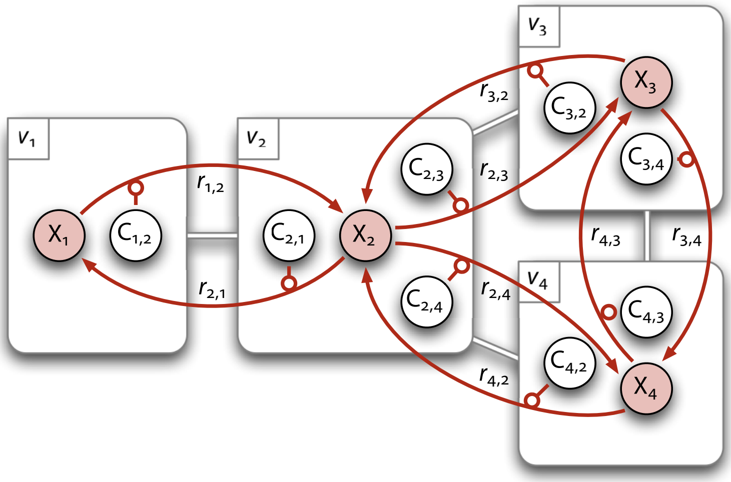


Figure 5.7 Reaction network of *Disperser*: Computes the average of X_i-molecules by letting the catalysts, $C_{i,j}$, send them to the corresponding neighbors. The result of the computation is an effect of the law of mass action scheduling.

it to the corresponding neighbor node v_j. The set of reactions, $\mathcal{R}$, is thus formed by the chemical reaction equations

$$\mathcal{R} = \left\{ r_{i,j} \mid (i,j) \in \mathcal{E} \right\} \tag{5.8a}$$

$$r_{i,j}\colon\ \mathrm{C}_{i,j} + \mathrm{X}_i \longrightarrow \mathrm{C}_{i,j} + \mathrm{X}_j \tag{5.8b}$$

One can intuitively grasp that the global reaction network strives to equilibrium. The more neighbors a node has, the greater is the outflow of X, since there are more C-molecules to react with. On the other hand, a node with higher degree also receives X-molecules from more neighbors.

Fraglets implementation

It is easy to come up with a *Fraglets implementation* that is equivalent to the abstract reaction network described above. The catalytic behavior of $C_{i,j}$ is represented by the persistent fraglet $_{v_i}$[matchp X send v_j X], which reacts with an X-molecule [X] and produces a transient fraglet

$$\begin{aligned}&{}_{v_i}[\texttt{matchp X send } v_j \texttt{ X}] + {}_{v_i}[\texttt{X}]\\ \longrightarrow\ &{}_{v_i}[\texttt{matchp X send } v_j \texttt{ X}] + {}_{v_i}[\texttt{send } v_j \texttt{ X}]\end{aligned}$$

which sends itself to the neighbor:

$${}_{v_i}[\texttt{send } v_j \texttt{ X}] \longrightarrow {}_{v_j}[\texttt{X}]$$

Simulation result and protocol comparison 5.4.3

In order to illustrate the principle of the *Disperser* protocol and to compare it to the existing *Push-Sum* protocol, we carried out OMNET++ (Varga, 2009)

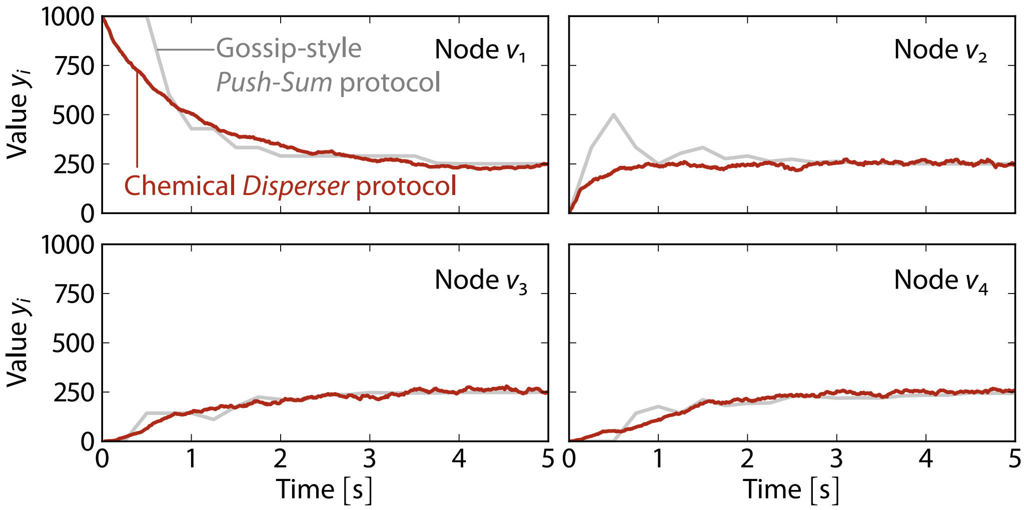


Figure 5.8

OMNeT++ simulation of *Push-Sum* and *Disperser*: The value of each node asymptotically converges to the average of 250 for both protocols.

simulations in various network topologies. In the following, we discuss *Disperser*'s behavior in two exemplary networks.

We first simulate the protocol in the four node topology presented earlier in Figure 5.7. For *Push-Sum*, we used an update interval of 250 ms. Both protocols start with an initial value of $y_1(0) = 1000$ in node v_1 while all other nodes are initialized with value $y_i(0) = 0$. Figure 5.8 shows how the value of each node asymptotically converges to the average of $\hat{y}_i = 250$ for both protocols.

Figure 5.9 shows the convergence behavior in a larger network of one hundred nodes, which are arranged in a toroidal topology. At time $t = 0$ s, 10^4 molecules are injected into node at coordinate (4,4). The figure shows a series of snapshots of the deviation of the molecular quantity from the expected average of 100 molecules; red indicates a value that is too low, green a value that is too high, whereas white is displayed when the node has reached the expected average.

mass conservation

Both protocols *Push-Sum* as well as *Disperser* rely on a kind of *mass conservation*. In *Push-Sum*, half of a node's sum is sent whereas the remainder is kept, but the overall sum remains constant. For *Disperser*, the conservation principle is obvious: The total number of X-molecules is conserved by all reactions. The two protocols differ in how they asymptotically approach the equilibrium: While *Push-Sum*'s code is executed isochronously, moving half of the value to a neighbor, *Disperser* transfers only one molecule per reaction, this rate being controlled by the inter-reaction time interval, which is inversely proportional to the concentration.

convergence time

The *convergence time* of both protocols can be lowered: In *Push-Sum* this is achieved by choosing a shorter update interval, whereas in *Disperser*, we may speed up the reactions by increasing the number of C-molecules. While

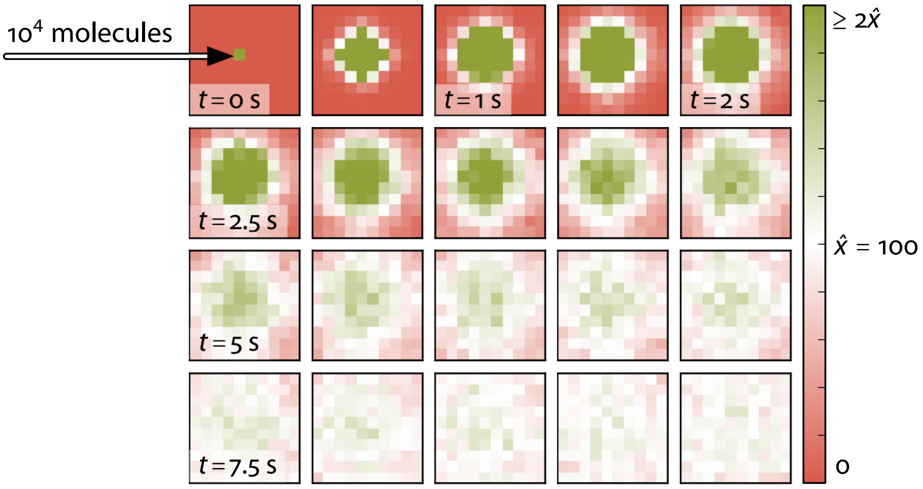


Figure 5.9 OMNeT++ simulation of *Disperser*: Series of snapshots showing the deviation from the average in a toroidal topology of 100 nodes.

Push-Sum calculates the exact value of the average, *Disperser* only provides an approximation. On the other hand, the *Push-Sum* protocol shows a tendency to overshoot in nodes with high degree, even if the transmission links do not delay packets. This is not the case for *Disperser*, as visualized in the subplot for node v_2 in Figure 5.8.

robustness to packet loss

Note that *neither of the two protocols is robust to packet loss*. However, while a lost packet in *Push-Sum* results in the loss of half of a node's value, a packet loss in *Disperser* only decreases the value by one. This higher robustness of *Disperser* is enabled by conveying less information per packet resulting in a higher message complexity of the chemical protocol.

elegant convergence proof

Last but not least, it is easier and *elegant to prove the convergence* of the *Disperser* protocol. We will show the details of this proof in Chapter 9 after reviewing the necessary mathematical foundations.

SUMMARY 5.5

In this chapter, we introduced the execution model for our chemical network architecture. This execution model is based on the Fraglets language, an efficient packet rewriting system. We extended the original Fraglets system in two respects: First, a reaction scheduler derived from the Next Reaction Method by Gibson and Bruck (2000) mimics the law of mass action behavior of molecular reactions. Second we extended the instruction set with stack instructions and harmonized the instruction set in order to make it easier to design chemical networking protocols.

The virtual machine executing Fraglets reactions can be regarded as the *system* while the related abstract description of the distributed artificial chemistry is the *model* of that system. This abstract model is based on the concept of artificial chemistry formalized by Dittrich et al. (2001), and here extended by a network of reaction vessels in order to build distributed reaction networks.

With the example of the *Disperser* protocol, we saw a prototypical showcase for a chemical networking protocol that resorts to a representation-free information encoding: Traditional protocols store their local state symbolically in variables, such as integers or flags, ultimately encoded as bit patterns. Such symbolic information is then piggybacked to packets in order to send information to distant nodes. Chemical networking protocols often encode protocol states in the quantity of molecules in the multiset.

Due to the law of mass action scheduling, the packet rate reflects the concentration of its originating chemicals. Thus, the packet rate itself, not the symbolic information inside the packet, is used to communicate state information among nodes. This rate-based encoding scheme akin to the nervous system (Dayan & Abbott, 2001) results in a higher resilience to the loss of packets. The law of mass action plays an important role: It mediates between the local and global world by proportionally mapping molecule concentrations to packet rates and vice-versa.

CHAPTER 6

Introduction to CNP Analysis

An overview of existing protocol analysis methods and an introduction to the next three chapters on the analysis of chemical networking protocols.

The epistemological value
of probability theory
is based on the fact
that chance phenomena,
considered collectively
and on a grand scale,
create non-random regularity. [6]

Limit Distributions for Sums of
Independent Random Variables
Andrey N. Kolmogorov

One prevalent problem in the research field of communication networks is the lack of a satisfying and easy-to-use modeling discipline. Existing protocol analysis and verification methods are often too complex and only applicable to very simple protocols or very restricted network topologies. Hence, researchers often tend to first build protocols and then measure them rather than first model, verify and then build their implementation (Jonsson, Kreiker, & Kwiatkowska, 2010).

In this and the following three chapters, we demonstrate how to apply existing analysis methods – originally used to study chemical reaction networks – to chemical networking protocols. Figure 6.1 shows the scope of these chapters and their positioning in our chemical engineering model. In general, in order to carry out formal verification, a researcher has to establish a correspondence between the system itself and its abstract mathematical model,

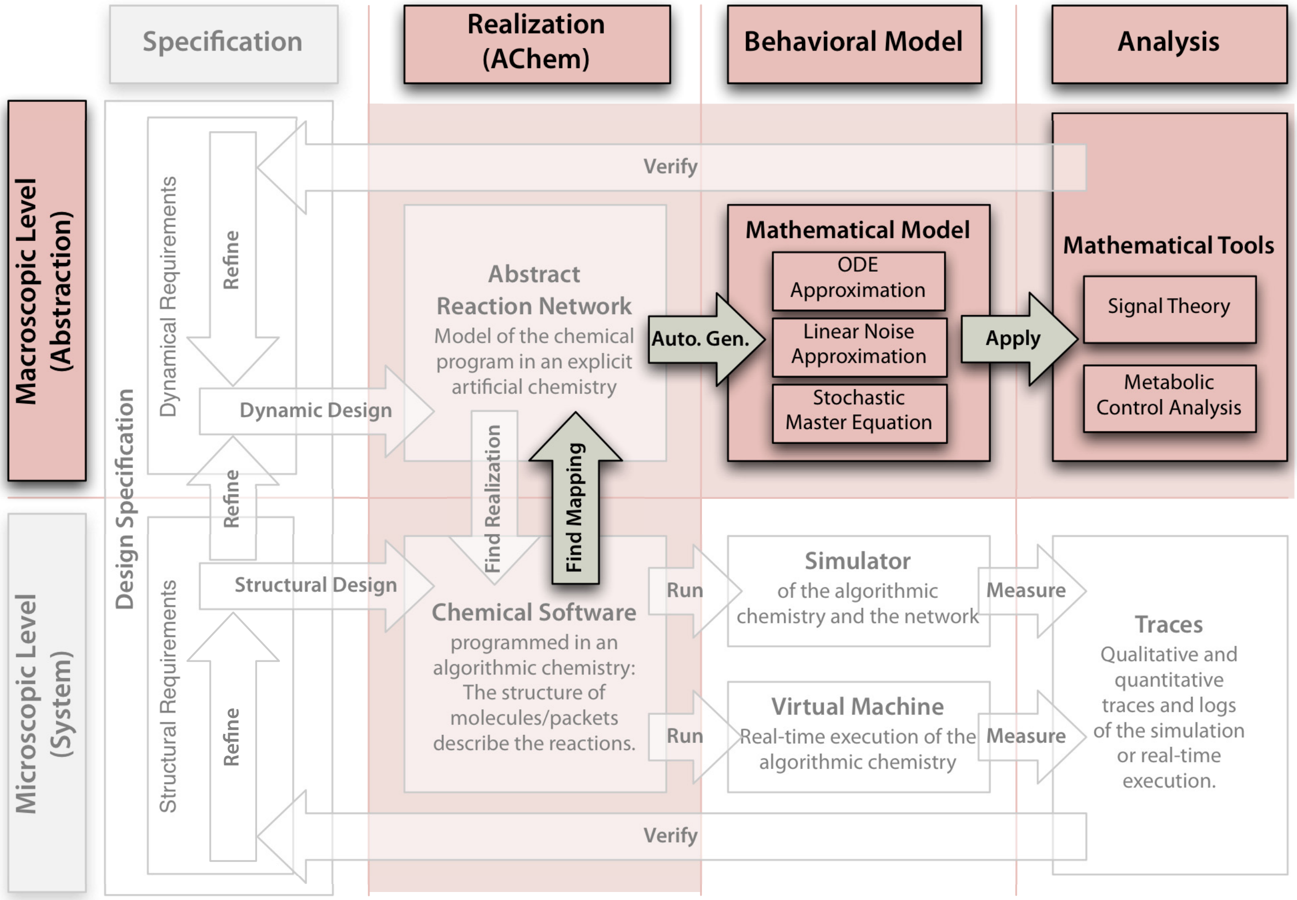

Specification
Realization (AChem)
Behavioral Model
Analysis
Macroscopic Level (Abstraction)
Microscopic Level (System)
Design Specification
Dynamical Requirements
Structural Requirements
Refine
Refine
Refine
Verify
Dynamic Design
Structural Design
Abstract Reaction Network
Model of the chemical program in an explicit artificial chemistry
Find Realization
Find Mapping
Auto. Gen.
Mathematical Model
ODE Approximation
Linear Noise Approximation
Stochastic Master Equation
Apply
Mathematical Tools
Signal Theory
Metabolic Control Analysis
Chemical Software
programmed in an algorithmic chemistry: The structure of molecules/packets describe the reactions.
Run
Run
Simulator
of the algorithmic chemistry and the network
Virtual Machine
Real-time execution of the algorithmic chemistry
Measure
Measure
Traces
Qualitative and quantitative traces and logs of the simulation or real-time execution.
Verify

Figure 6.1 **Analytical methods in the engineering model**: Chapters 6 to 9 discuss the direct correspondence between the behavior of chemical software, its abstract model, and the mathematical model of the abstract reaction network on which rigorous mathematical analysis can be carried out.

which is the basis for formal proofs. Usually, this is an intricate task, but for chemical networking protocols (CNPS), there is a direct relation between the execution model (the chemical software: bottom left), an intermediate abstract description (the abstract reaction network: top left), and its formal mathematical model (top center) for which analytical tools exist. But before delving into these details, this chapter first reviews existing protocol verification methods in Section 6.1 and gives an overview of the proposed analysis and verification procedures for CNPS in Section 6.2.

6.1 RELATED WORK

The goal of protocol analysis methods is to support the researcher in designing networking protocols that comply with the requirement specification. Analysis is embedded into the design process as feedback mechanism: analytical results are used iteratively, for example to detect and fix possible errors or to optimize the protocol's performance, robustness, or other measures.

There are a vast number of existing methods for protocol analysis at different levels of precision and generality. The spectrum ranges from experimental gathering of statistical data over simulation to formal verification. Formal verification itself can be approached from two opposing extremes: state-based and proof-based approaches (Amjad, 2004). Model checking exhaustively examines all possible states of the system together with all possible stimuli from the environment and checks that only the desired properties hold. Proof-based approaches describe the system in logic clauses and use logical inference to show that the system exhibits the required properties.

EMPIRICAL ANALYSIS 6.1.1

The most frequently used method in practice is empirical data gathering of a *concrete* protocol implementation in a *given* network topology. Traces of packet sequences reveal possible qualitative bugs whereas statistical data is collected to analyze the protocol's performance in a given network environment. Online data evaluation becomes difficult when multiple locations in the network have to be inspected simultaneously in order to understand the behavior of a distributed algorithm.

Network simulators

Network simulators simplify empirical analysis by simulating a whole network on a single machine or a cluster. This not only mitigates the cost for an equivalent test bed, but also enables the engineer to collect synchronized traces from virtually distributed, but now physically concentrated nodes. Many network simulators are available today; next to many commercial tools used in industry, open source simulators like NS2 (Fall & Vardhan, 2010), its successor NS3 (Henderson, Roy, Floyd, & Riley, 2006), or OMNET++ (Varga & Hornig, 2008; Varga, 2009) are predominantly used in research.

Yet, network simulators are only able to analyze a specific protocol implementation in a limited set of simulated network environments. They do not analyze protocols for arbitrary network topologies formally or prove that the requirement specification is satisfied.

MODEL CHECKING 6.1.2

Model checking is based on an abstract specification of the system, which is used to formally verify a concrete implementation. During the design phase, the engineer has to come up with an abstract model of his/her protocol implementation. The design specification itself has also to be expressed formally, for example, by expressions in Linear Temporal Logic (LTL) (Pnueli, 1977). This specification contains a set of properties the protocol must satisfy. Model checking then exhaustively expands all possible states of the system and checks whether the properties are satisfied in a fully-automatic way, resulting either in success, meaning that the model of the implementation complies with the formal specification, or in a counter-example, given as an execution trace for which the specification is violated (Clarke, 2008).

tools

Early protocol verification started with tools to verify Petri Nets in the 1970ies, Finite State Machines (FSM) approaches that systematically explore the reachable global state-space of a protocol (Bochmann, 1976), and first languages to express models, such as PAN (Protocol ANalyzer) (Holzmann, 1981). Probably the most famous model checking software for networking protocols is SPIN (Simple Promela INterpreter), a successor of PAN. SPIN is a generic model checker for asynchronous processes that focuses on proving the correctness of process interactions (Holzmann, 1997). Design specifications are written in the PROMELA language (PROcess MEta LAnguage) (Holzmann, 1991) and correctness claims are accepted in LTL. Model checking has been successfully applied to qualitative problems, such as checking the Dynamic Host Configuration Protocol (DHCP) (Islam, Sqalli, & Kahn, 2006), the Lightweight Underlay Network Ad hoc Routing protocol (LUNAR) (Wibling, Parrow, & Pears, 2004), and security protocols (Maggi & Sisto, 2002), among others.

problems

Traditional model checking aims at proving the absolute correctness of systems or protocols. In practice, such rigid claims are hard or even impossible to guarantee for the following reasons: First, although checking the model towards the specification is fully automatic, building the abstract model and the formal specification are manual processes. A model that successfully passes the check is no guarantee that the actual implementation complies with the informal requirement specification. Second, because the checker exhaustively expands all possible states, modeling large networks suffers from state explosion: the number of states grows exponentially with the number of network nodes. This may be no problem for security protocols that usually operate among a limited number of nodes, but distributed algorithms aiming to reach a consensus among a vast number of nodes cannot be treated with traditional state expansion. Some solutions to this problem have been proposed, such as partial order reduction (Peled, 1993), model abstraction (Hsieh & Levitan, 1998), and compositional reasoning (Berezin, Campos, & Clarke, 1998). The third challenge for model checking are stochastic protocols, such as gossip protocols. They use stochastic decision processes to disseminate information to random neighbors. Such stochastic processes cannot be modeled by deterministic FSMs.

Probabilistic model checking addresses the stochastic nature of recent protocols and at the same time captures the randomness of the environment, for instance the irregularity of user interactions. Probabilistic model checking first constructs a probabilistic model of the system that exhaustively explores the states space, i.e. enumerates all possible and probable states, followed by a quantitative analysis of the stochastic state trajectories. Several models have been proposed based on discrete or continuous time Markov chains, Markov decision processes, and Probabilistic Timed Automata (PTA) (Kwiatkowska, Norman, Segala, & Sproston, 2002). The system's specification is usually formalized in Probabilistic Temporal Logic PCTL or Linear Temporal Logic (LTL) (Pnueli, 1977). PRISM (Hinton, Kwiatkowska, Norman, & Parker, 2006) is a well-established tool to automatically verify probabilistic systems. Probabilistic model checking has been successfully applied to networking protocol analysis, such as the IEEE 802.3 (CSMA/CD) protocol (Duflot et al., 2010), but also to chemical reaction networks (Barbuti, Cataudella, Maggiolo-Schettini, Milazzo, & Troina, 2005; Heath, Kwiatkowska, Norman, Parker, & Tymchyshyn, 2008; Ballarini, Mardare, & Mura, 2009).

probabilistic model checking

PROCESS CALCULUS 6.1.3

Process calculi are algebraic descriptions of concurrent processes and their (asynchronous) interactions. Existing process calculi are, among others, Communicating Sequential Processes (CSP), originally proposed by Hoare (1978), Milner's Calculus of Communicating Systems (CCS) (1980), and its successors, π-calculus, which allows processes to change their configuration over time, and ambient calculus (Cardelli & Gordon, 1998), which enables the description of mobile agents.

Formal reasoning in process calculi is performed by algebraic rules and bi-simulation: The distributed algorithms as well as their desired properties are modeled within the algebraic framework. The two formalisms are then manipulated by algebraic reasoning until they are minimal in the number of states. Then it can be shown that they are behavioral equivalent, which proves the asserted properties.

THEOREM PROVING 6.1.4

Yet another formal approach to analyze networking protocols is logical inference. The (distributed) system is described by a set of definitions in mathematical logic. The properties of the system are then derived as theorems that follow from these definitions (Amjad, 2004). Unlike in model checking, where all reachable states are exhaustively checked, theorem proving uses logical reasoning.

The main problems of proving theorems for practical protocols are the following two requirements that are hard to fulfill at the same time: The logic formulation needs to be expressive such that a given protocol implementation can easily be translated. On the other hand, the more expressive the logic, the harder it is to prove the theorems in that logic (Halpern & Vardi, 1991). Therefore, human understanding of the system is usually needed to assist the proof. Manual investigation is also needed if a certain assertion cannot be proved: this is because theorem provers, unlike model checkers, are not able to generate a counter-example.

Many theorem-proving tools are available today, such as Isabelle/HOL (L. C. Paulsson, 1988; Nipkow, Paulson, & Wenzel, 2002), PVS (Owre, Rajan, Rushby, Shankar, & Srivas, 1996), and Coq (Bertot & Castéran, 2004). Theorem-proving successfully proved security protocols (L. C. Paulsson, 1998; Bella, 2007), for example Transport Layer Security (TLS) (L. C. Paulsson, 1998), and routing protocols such a the Routing Information Protocol (RIP) (Bhargavan, Gunter, & Obradovic, 2000). Hasan and Tahar (2009) applied theorem proving to probabilistic systems by formalizing and verifying random variables in higher-order logic, a method originally proposed by Hurd (2002).

6.1.5 RIGOROUS MATHEMATICAL ANALYSIS

A completely different way of approaching the analysis of networking protocol is to rely solely on "pen and paper" to proof certain dynamic properties of the protocol. This only works if the dynamic behavior of a networking protocol is well understood and can be described numerically (for example using calculus or probability theory). Then, however, there is a chance to come up with simple proofs that are expressive and conclusive. In the past, this method has often been applied to nature inspired protocols, where the native process is already well understood and mathematically captured. For example, Eugster, Guerraoui, Kermarrec, and Massoulié (2004) used the dynamics of epidemic spreading to model gossip-style protocols. But also for traditional protocols, such as the window size behavior in the Transmission Control Protocols (TCP), Budhiraja, Hernández-Campos, Kulkarni, and Smith (2004) found a mathematical description.

continuous approximations

As indicated in Kolmogorov's quote preceding this chapter, probabilistic systems tend to behave deterministically in the limit. This often allows for *continuous approximations* of their behavior. In other words, if the system's dynamics can be described mathematically it is often possible to approximate its behavior to arrive at a simpler, more tractable description, which can be solved for large or even for arbitrary network topologies. For example, Bakhshi, Cloth, Fokkink, and Haverkort (2009) approximated the behavior of large-scale gossip networks by a mean-field theoretic formulation.

problems

The main problem of this method is the lack of numerical mathematical models for most established networking protocols. For example, it does not make sense to model a single database query via quantitative analysis. Rigorous mathematical models (e.g. queuing theory or network calculi) are only appropriate for the quantitative analysis of the large time scale behavior, especially for continuously running processes whose dynamic behavior is crucial for the correct operation of the protocol or the entire network.

CHEMICAL PROTOCOL ANALYSIS OVERVIEW 6.2

Chemical networking protocols (CNPs) outsource computation to the dynamic behavior of chemical reaction networks. Because mathematical models for chemical reaction networks already exist, CNPs are good candidates for a rigorous mathematical analysis. Ultimately, we would like to prove that a CNP is effective, efficient, and robust. That is, we want to prove that a certain CNP converges to the desired solution within reasonable time, and that this solution is stable even if components of the system or the network environment are perturbed. In the following three chapters, we propose an analysis method and procedure for CNPs. We show that such proofs are feasible even for arbitrary network topologies, and that those proofs are easy to understand and often render automatic theorem provers unnecessary.

Figure 6.2 shows an overview of the proposed analysis procedure, which leads from a distributed Fraglets program (bottom left) via the construction of an abstract artificial chemistry (top left) to a rigorous mathematical analysis of its behavior (top right).

structural analysis

Before we are able carry out a dynamical analysis, we have to identify the components of the protocol and their interactions. As mentioned in Section 5.2.2, Fraglets is an implicit and constructive artificial chemistry, meaning that a reaction among molecules may form new types of molecules. The reaction rules are also defined implicitly by the structure of the molecules; hence, new molecules may induce new reaction channels, yielding another set of new molecule types, and so on.

Thus, the first step of our analysis procedure, described in Chapter 7, exhaustively expands the reaction network by systematically and iteratively executing all possible rewriting rules among the installed "program" molecules and all imaginable input molecules until we have unfolded the complete chem-

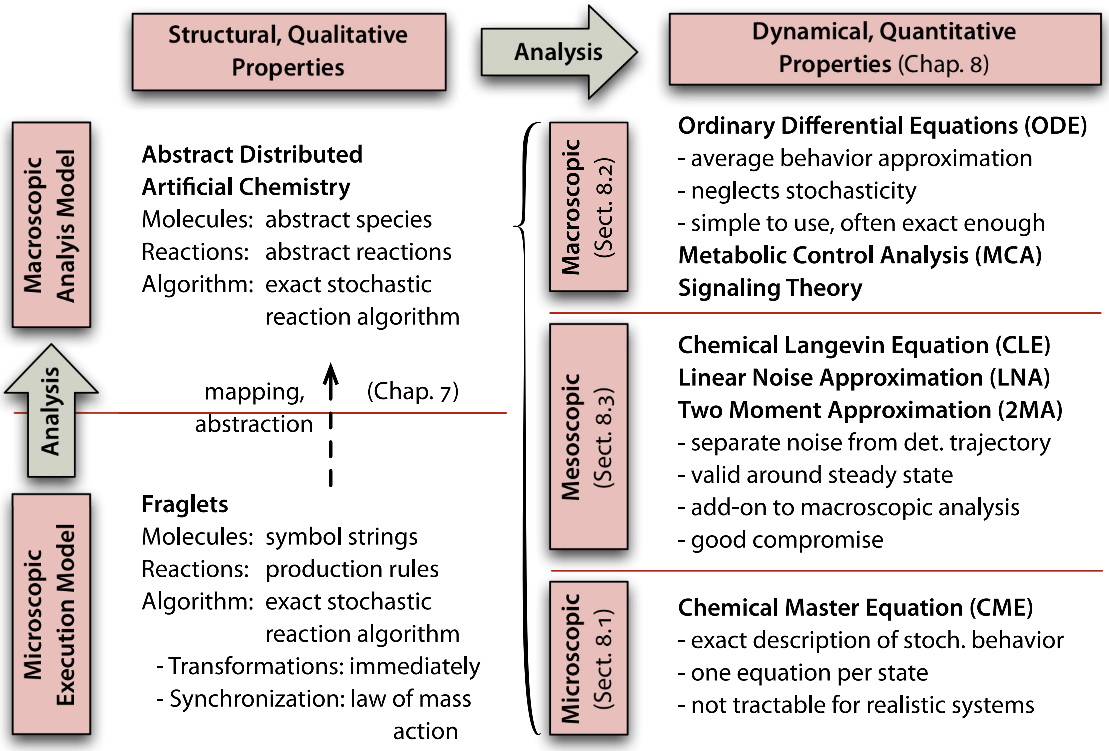

Structural, Qualitative Properties
Analysis
Dynamical, Quantitative Properties (Chap. 8)
Macroscopic Analysis Model
Abstract Distributed Artificial Chemistry
Molecules: abstract species
Reactions: abstract reactions
Algorithm: exact stochastic reaction algorithm
Analysis
mapping, abstraction
(Chap. 7)
Microscopic Execution Model
Fraglets
Molecules: symbol strings
Reactions: production rules
Algorithm: exact stochastic reaction algorithm
- Transformations: immediately
- Synchronization: law of mass action
Macroscopic (Sect. 8.2)
Ordinary Differential Equations (ODE)
- average behavior approximation
- neglects stochasticity
- simple to use, often exact enough
Metabolic Control Analysis (MCA)
Signaling Theory
Mesoscopic (Sect. 8.3)
Chemical Langevin Equation (CLE)
Linear Noise Approximation (LNA)
Two Moment Approximation (2MA)
- separate noise from det. trajectory
- valid around steady state
- add-on to macroscopic analysis
- good compromise
Microscopic (Sect. 8.1)
Chemical Master Equation (CME)
- exact description of stoch. behavior
- one equation per state
- not tractable for realistic systems

Figure 6.2 **Chemical protocol analysis overview**: We map the Fraglets program to an abstract artificial chemistry on which dynamical analysis is carried out at different levels of detail.

ical universe that is relevant for the protocol. This expansion has to be done carefully, because a separate treatment of every distinct fraglet string often leads to infinite reaction networks. Therefore, we propose a mapping from Fraglets to an abstract artificial chemistry that maps multiple (potentially infinitely many) similar fraglet strings into the same abstract molecular species. The resulting abstract reaction network shall be equivalent to the Fraglets system with respect to its dynamic behavior.

dynamical analysis

Once all molecule types and reactions comprising the protocol are identified and the abstract reaction network is constructed, we are in a position to analyze its behavior based on a mathematical description of its dynamics; this is captured by Chapter 8. Fortunately, we can profit from many mathematical descriptions that have been developed for real chemical reaction networks in the last two centuries.

As illustrated on the right side of Figure 6.2 there are three different levels at which the dynamic behavior of chemical reaction networks can be analyzed: the (1) microscopic, the (2) macroscopic, and the (3) mesoscopic level.

1. At the **microscopic level** (Section 8.1), the stochastic nature of the reaction system is fully and correctly described by the so called Chemical

Master Equation (CME). However, an analytical or numerical treatment of the CME is usually too complicated, even for simple systems.

2. At the **macroscopic level**, the stochastic behavior is approximated by deterministic Ordinary Differential Equations (ODEs), which describe the expected trajectory, averaged over several trials. Section 8.2 reviews how to derive this approximation from the microscopic CME and demonstrates how ODEs can be used to prove the convergence of CNPs. Analysis methods such as Metabolic Control Analysis (MCA) and signal theory use this approximation to study the transient behavior of reaction networks.

3. With the deterministic model, we certainly lose information about the stochastic nature of the system. **Mesoscopic** dynamical analysis methods (Section 8.3) try to estimate the inherent "noise" that is added by the system around the steady state. We quickly discuss the easy-to-obtain Chemical Langevin Equation (CLE) and the more precise Two Moment Approximation (2MA), but finally use the Linear Noise Approximation (LNA) to quantify the stochastic fluctuations in CNPs.

In Section 8.4 we also mention two generic theorems that are helpful to prove the convergence of CNPs: the Deficiency Zero Theorem and the Chemical Organization Theory. This catalog of methods covers many aspects of protocol analysis, ranging from detailed exact analysis of simple problems to good estimations that can be applied very quickly to complex protocols in order to estimate their behavior.

For illustration purposes, the description of all analytical methods is complemented with their application to a simplified version of the *Disperser* protocol. Chapter 9 then provides a complete convergence proof for *Disperser* for arbitrary network topologies and estimates the accuracy of its calculated result.

CHAPTER 7

Structural CNP Analysis

On the exploration of the potentially infinite sequence space of a Fraglets program and its mapping to a finite abstract model.

> An abstraction is one thing
> that represents several real things
> equally well. [7]
>
> Edgser W. Dijkstra

As a prerequisite for the dynamical analysis, the structural analysis of a CNP identifies all components (molecules) and their interactions (reactions) that are taking part in a certain implementation. This is done by exploring the chemical universe that is spanned by a protocol. In this chapter, we demonstrate how such a structural analysis is carried out for Fraglets implementations. This chapter is addressed to an audience with a background or interest in formal analysis. Readers that are more interested in the dynamical analysis are referred to the next chapter.

As indicated by Figure 7.1, we propose to build an abstract model of a protocol implementation by mapping sets of fraglet strings onto species of an explicit artificial chemistry, before analyzing the protocol's dynamic behavior. Such a mapping is required for the following reasons:

Reduction: Subsuming a set of distinct fraglets with similar properties under a single abstract species helps reducing the number of components of the system. Often, we are able to reduce an infinite set of Fraglet strings to a finite set of abstract species.

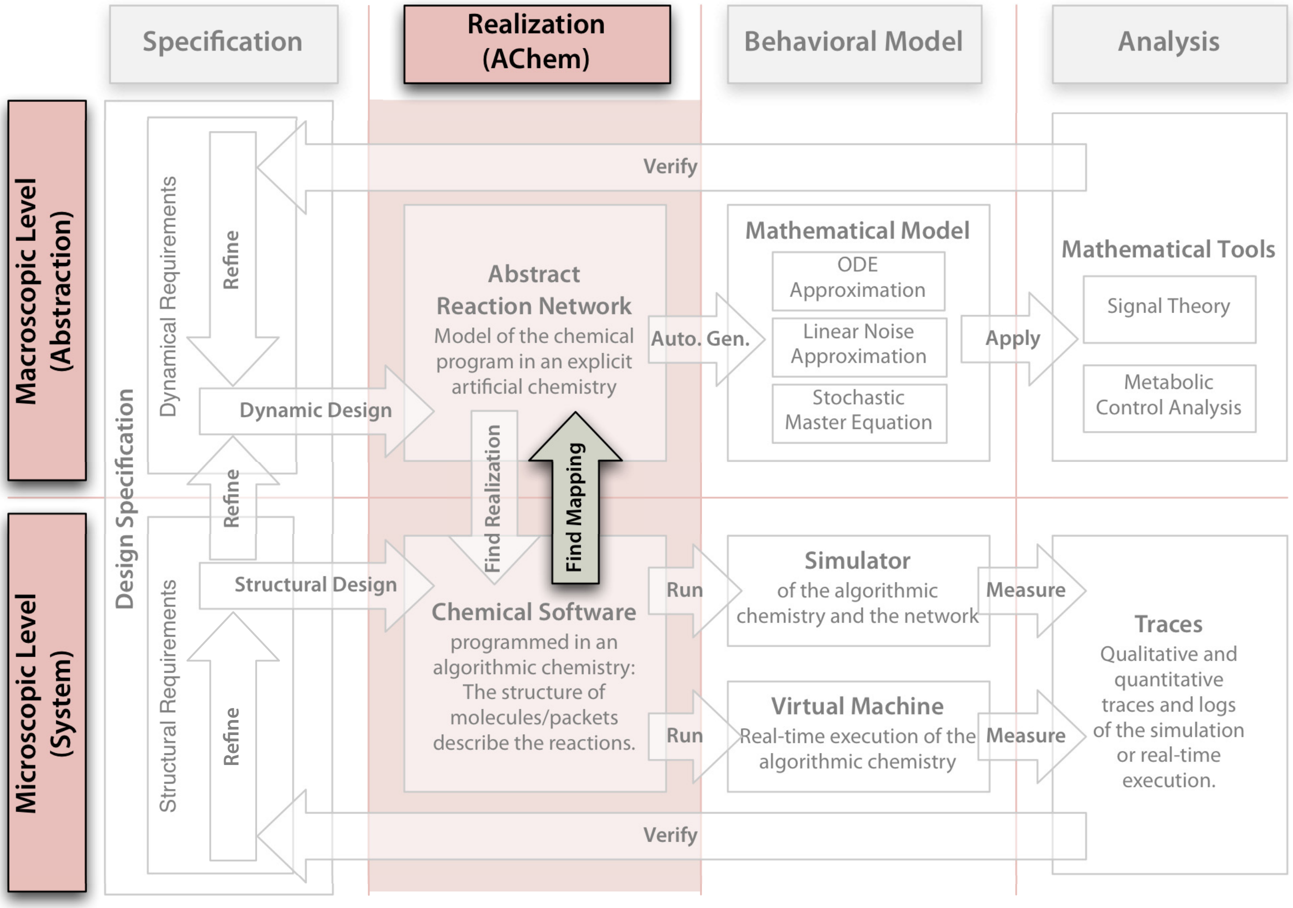

Specification
Realization (AChem)
Behavioral Model
Analysis
Macroscopic Level (Abstraction)
Microscopic Level (System)
Design Specification
Dynamical Requirements
Structural Requirements
Refine
Refine
Refine
Verify
Verify
Dynamic Design
Structural Design
Abstract Reaction Network
Model of the chemical program in an explicit artificial chemistry
Find Realization
Find Mapping
Chemical Software
programmed in an algorithmic chemistry: The structure of molecules/packets describe the reactions.
Auto. Gen.
Mathematical Model
ODE Approximation
Linear Noise Approximation
Stochastic Master Equation
Apply
Mathematical Tools
Signal Theory
Metabolic Control Analysis
Run
Run
Simulator
of the algorithmic chemistry and the network
Virtual Machine
Real-time execution of the algorithmic chemistry
Measure
Measure
Traces
Qualitative and quantitative traces and logs of the simulation or real-time execution.

Figure 7.1 **Structural analysis in the engineering model**: This chapter discusses the mapping of fraglet strings to an abstract model, which can later be analyzed dynamically.

Abstraction: With this mapping we abstract away from Fraglets. Thus, the subsequent dynamical analysis is made available not only for Fraglets, but for any chemical programming language and execution model that can be mapped to the explicit artificial chemistry.

Exploration: In order to come up with such a mapping, we have to consider all possibly appearing molecules, which forces us to perform an exhaustive exploration of the reaction network. This procedure is similar to the systematic state-space exploration of traditional formal program verification techniques. It allows us to gain insights into some qualitative features of the protocol, for example, whether the sequence space is bounded or infinite, or how a modification of a fraglet string affects the global reaction network.

This chapter is organized as follows: Section 7.1 explains why it is not trivial to find the right mapping function. Next, in Section 7.2 we demonstrate how to

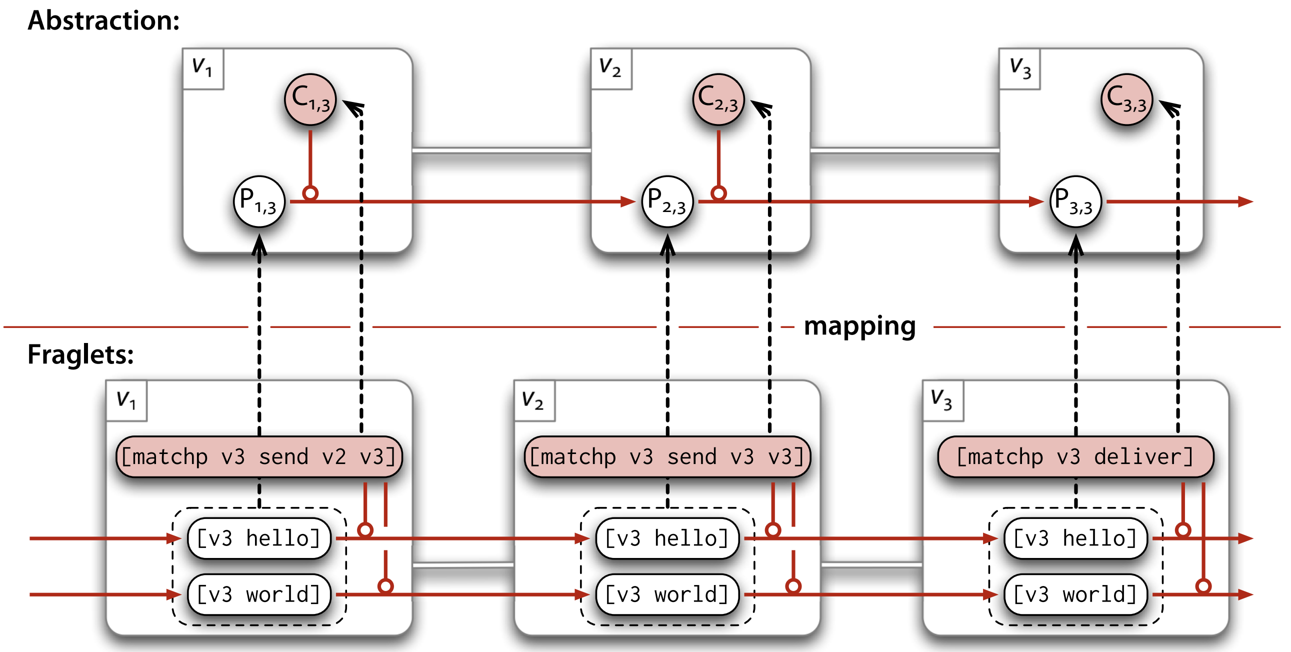


Figure 7.2 Mapping fraglet sets to abstract species: Simple forwarding: The name of the destination node is prepended to the payload to form a data packet. Corresponding persistent `matchp`-fraglets forward data packets to the next hop. All packets with the same header tag can be mapped to the same abstract species.

explore the sequence space of a protocol implementation in Fraglets and relate this operation to the closure term appearing in different research fields. We then propose two different algorithms to map Fraglets CNP implementations to an explicit artificial chemistry: The first algorithm, presented in Section 7.3, is based on Fraglets expression rewriting and needs human assistance, whereas Section 7.4 introduces another algorithm that operates fully automatic. Finally, Section 7.5 discusses the presented structural analysis methods.

PROBLEM DESCRIPTION 7.1

Every distinct fraglet string can be considered as a separate molecular species. However, this valid distinction often leads to an *explosion of the sequence space*. Consider, for example, a simple forwarding protocol as depicted in Figure 7.2 (bottom). Data packets carrying payload ("hello", "world") are prepended with the name of the destination node, v_3. Each node contains a persistent `matchp`-fraglet that forwards those packets to the next hop. If we treat all data strings as distinct species, we would have to deal with infinitely many molecular species, and hence with an infinite reaction network.

sequence space explosion

In this example, we would rather map all `[v3...]`-fraglets in node v_i to the same species $P_{i,3}$ and only distinguish between data packets in different nodes to different destinations, independent on their payload (see Figure 7.2,

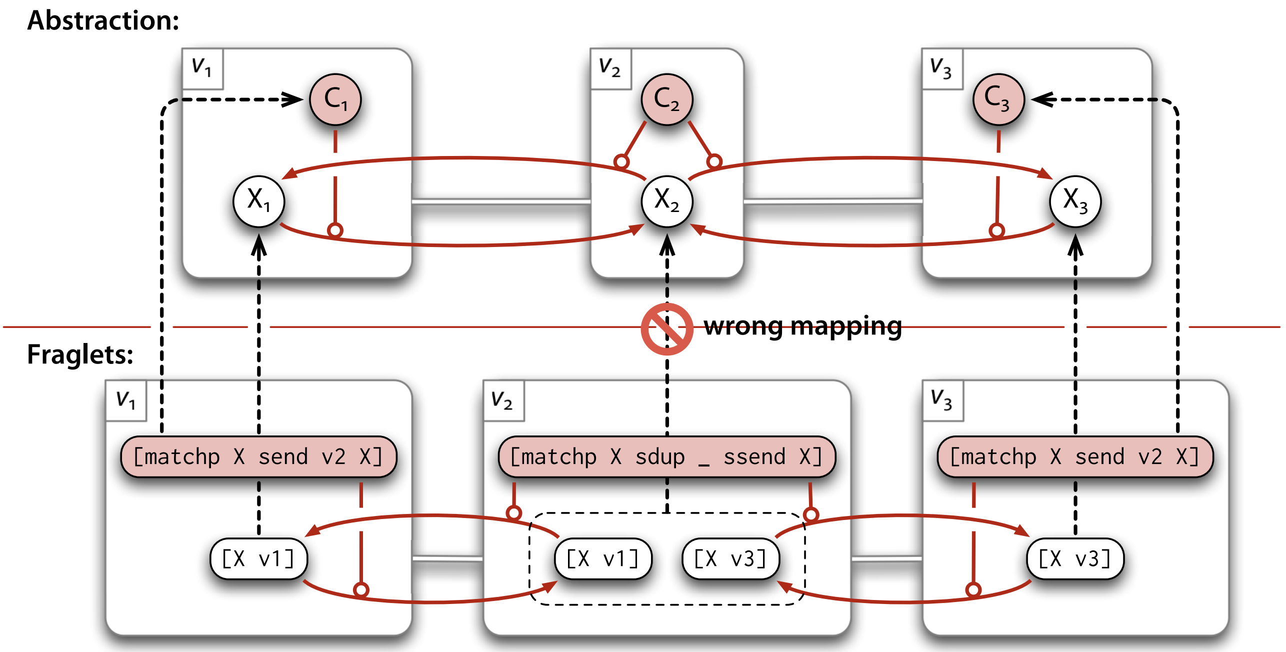


Figure 7.3 **Wrong mapping**: In Fraglets, the two distinct $_{v_2}$[X . . .]-fraglets in node v_2 are involved in separate reactions. The abstract reaction network does not reflect this distinction.

top). This simple mapping is valid, because Fraglets only performs a perfect match of the header symbol when carrying out a bimolecular reaction.

However, the simple strategy of mapping sets of fraglet strings with the same header tag to the same abstract species not always leads to an abstract reaction network that is dynamically equivalent to its implementation in Fraglets. As a counter example, consider the Fraglets program depicted in Figure 7.3 — a modified *Disperser* protocol. Indeed, the two distinct $_{v_2}$[X . . .]-fraglets in node v_2 react with the same catalyst, as they start with the same header symbol. But the two reactions yield different products, because the ssend-instruction treats the second symbol as next hop address. Therefore, the two fraglets $_{v_2}$[X v1] and $_{v_2}$[X v3] must *not* be mapped to the same abstract species X_2. The wrong abstract reaction network shown in Figure 7.3 (top) mixes molecules of two distinct reaction sub-networks whereas the Fraglets implementation maintains a separate population of [X v1]- and [X v3]-molecules.

Hence, the problem is to devise an algorithm that determines the granularity of the mapping, somewhere between a one-to-one and a identical-header-tags-to-one mapping. Formally, we are given the set of seed fraglets $\mathcal{W}_{\text{seed}} \subseteq \Sigma^*$. This seed set contains those strings that are pre-installed in the vessels as "program" and all fraglet strings we must expect from the user or the environment as input. The algorithm now has to solve the following two sub-problems:

1. Find the set of fraglets relevant to the system $\mathcal{W}_{\text{rel}} \subseteq \Sigma^*$, i.e. all strings that may eventually appear during the operation of the protocol. As we will see in 7.2, this set may be very large or even infinite.

2. Find a mapping from fraglet strings to abstract species, $\mathcal{F}\colon \mathcal{W}_{\text{rel}} \rightarrow \mathcal{S}$, such that the generated abstract chemical universe, $(\mathcal{S}, \mathcal{R})$, is behaviorally equivalent (Milner, 1980) to the Fraglets implementation, $(\mathcal{W}_{\text{rel}}, \mathcal{P})$, when driven with an exact stochastic reaction algorithm $\mathcal{A}$: $(\mathcal{S}, \mathcal{R}) \sim_{\mathcal{A}} (\mathcal{W}_{\text{rel}}, \mathcal{P})$. We will propose two algorithms for this problem in Sections 7.3 and 7.4.

EXPLORING THE SEQUENCE SPACE OF PROTOCOL IMPLEMENTATIONS IN FRAGLETS 7.2

In order to find the set of relevant fraglets $\mathcal{W}_{\text{rel}}$ from the seed set $\mathcal{W}_{\text{seed}}$, we have to "expand" the Fraglets implementation of the protocol by executing every possible reaction path. We therefore construct the structural reachability graph of the seed set by carrying out all production rules $\mathcal{P}$ possible for any initial and its consecutive fraglet string sets. We call $\mathcal{W}_{\text{rel}}$ the closure of $\mathcal{W}_{\text{seed}}$.

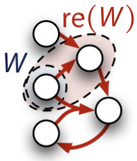
closure

The term "closure" appears in many fields of natural science with the similar meaning of reaching a bounded set by applying the closure operation. We use the term closure in the sense of Dittrich and Speroni di Fenizio (2007), who introduced it formally for chemical reaction systems in their Chemical Organization Theory. It is loosely related to the closure term in general topology (Benkö et al., 2008), the transitive closure on binary relations in set theory (Huet, 1980) or the ε-closure in automata and compiler theory (Aho, Lam, Sethi, & Ullman, 2006). In the context of Fraglets, we define a set of strings $\mathcal{W}$ as closed if all strings that can be generated by rewriting rules applied to strings in this set are already in $\mathcal{W}$. The closure of a set $\mathcal{W}$ is the minimal set that is closed and contains $\mathcal{W}$.

rewriting expansion function

In order to find the closure of a set $\mathcal{W}$ of fraglet strings we roughly follow Benkö et al. (2008) by first defining a *rewriting expansion function* $\text{re}\colon \wp(\Sigma^*) \rightarrow \wp(\Sigma^*)$, which gives the union of the set of fraglets, $\mathcal{W} \subseteq \Sigma^*$, with the set of fraglet strings that can be reached from that set within one rewriting step. Formally,

$$\text{re}(\mathcal{W}) = \mathcal{W} \cup \bigcup_{p \in \mathcal{P}_{\mathcal{W}}} \text{img}\, p \tag{7.1}$$



Algorithm 7.1 Closure of a set of fraglets: Execute all rewriting rules possible among the fraglet strings in the seed set as long as new strings are generated.

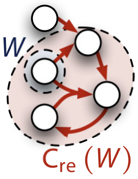
```
Input: $\mathcal{W}_{\text{seed}}$ (set of seed fraglets)
Output: $\mathcal{W}_{\text{rel}} = \mathrm{C}_{\text{re}}(\mathcal{W}_{\text{seed}})$ {closure of $\mathcal{W}_{\text{seed}}$}
$\mathcal{W}_{\text{rel}} \leftarrow \varnothing$
$\mathcal{Y} \leftarrow \mathcal{W}_{\text{seed}}$
while $\mathcal{Y} \neq \varnothing$ do
    $\mathcal{W}_{\text{rel}} \leftarrow \mathcal{W}_{\text{rel}} \cup \mathcal{Y}$
    $\mathcal{Y} \leftarrow \varnothing$
    for all $\mathcal{X}_i \in \wp(\mathcal{W}_{\text{rel}})$ {iterate over subsets} do
        if $\mathcal{X}_i$ forms the left-hand side of a production rule then
            $\mathcal{X}_i \Rightarrow \mathcal{X}_o$ {execute the production rule}
            $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{X}_o$
        end if
    end for
end while
return $\mathcal{W}_{\text{rel}}$
```

where

$$\mathcal{P}_{\mathcal{W}} = \left\{ p \in P \mid \operatorname{dom} p \subseteq \mathcal{W} \right\} \tag{7.2}$$

is the set of all production rules that can be applied to fraglet strings in $\mathcal{W} \subseteq \Sigma^*$.

abstract closure operator



The *abstract closure operator* $\mathrm{C}_f(\mathcal{W})$ generates the smallest subset closed under the set-valued set function $f : \wp(\mathcal{W}) \rightarrow \wp(\mathcal{W})$ that contains $\mathcal{W}$ as a subset. The abstract closure operator has the properties of being extensive ($\mathrm{C}_f(\mathcal{W}) \supseteq \mathcal{W}$), idempotent ($\mathrm{C}_f\big(\mathrm{C}_f(\mathcal{W})\big) = \mathrm{C}_f(\mathcal{W})$), and monotone ($\mathcal{W}_1 \subseteq \mathcal{W}_2$ implies $\mathrm{C}_f(\mathcal{W}_1) \subseteq \mathrm{C}_f(\mathcal{W}_2)$). The abstract closure operator under rewriting expansion gives the set of fraglet strings, $\mathcal{W}_{\text{rel}}$, that is potentially reachable from the seed set $\mathcal{W}_{\text{seed}}$:

$$\mathcal{W}_{\text{rel}} = \mathrm{C}_{\text{re}}(\mathcal{W}_{\text{seed}}) = \lim_{n \to \infty} \mathrm{re}^n(\mathcal{W}_{\text{seed}}) \tag{7.3}$$

That is, the closure operator, or the corresponding Algorithm 7.1, explores the sequence space of a protocol implementation in Fraglets.

7.2.1 FINITE AND INFINITE CLOSURES

finite closure

The closure of a set of fraglets $\mathcal{W}_{\text{seed}} \subseteq \Sigma^*$ is finite if it contains a finite number of fraglet strings. One would probably expect that the closure of a finite seed is finite, too. However, this does not hold in general. Because of the properties of the closure operator, the closure under rewriting expansion is *finite* if the seed set, $\mathcal{W}_{\text{seed}}$, is finite *and* if there is an integer n such that $\mathrm{re}^n(\mathcal{W}) = \mathrm{re}^{n+1}(\mathcal{W})$; otherwise, if the algorithm does not terminate, the closure is *infinite*. Three different situations in Fraglets protocol implementations may result in an infinite closure: (1) arithmetic explosion, (2) structural explosion, and (3) infinite seeds.

infinite closure

1. **Arithmetic Explosion:** Arithmetic operations may produce fraglets starting with the same header tag but continuously different values in the tail. For example, consider the initial set

$$\mathcal{W}_{\text{seed}_1} = \{\texttt{[matchp X ssum X 1]}, \texttt{[X 0]}\} \tag{7.4}$$

   The induced reaction increments the tail symbol of the second fraglet, producing `[X 1]` in a first reaction step, `[X 2]` in the subsequent reaction step, an so on. Thus, the closure of this set is infinite:

$$\mathrm{C}_{\text{re}}\left(\mathcal{W}_{\text{seed}_1}\right) = \{\texttt{[matchp X ssum X 1]}\} \cup \left\{\texttt{[X } i\texttt{]} \mid i \in \mathbb{N}_0\right\} \tag{7.5}$$

   In this example we allowed arbitrary natural numbers. A realistic implementation of the Fraglets virtual machine will however limit the range of integer values, for example, to the integers representable by 32 bits. This leads to a huge, but still finite closure.

2. **Structural Explosion:** Structural explosion leads to an infinite variety of strings that goes beyond altering a single symbol like for the arithmetic explosion. Consider the initial set

$$\mathcal{W}_{\text{seed}_2} = \{\texttt{[matchp X X X]}, \texttt{[X]}\} \tag{7.6}$$

   The first reaction step adds another `X`-symbol to the second fraglet, converting `[X]` to `[X X]`. The next reaction step takes `[X X]` to produce `[X X X]` and so on, yielding longer and longer fraglets. The infinite closure of this set can be summarized as

$$\mathrm{C}_{\text{re}}\left(\mathcal{W}_{\text{seed}_2}\right) = \{\texttt{[matchp X X X]}\} \cup \left\{\texttt{[X X}^{i}\texttt{]} \mid i \in \mathbb{N}_0\right\} \tag{7.7}$$

   Structural explosion is more serious than arithmetic explosion, since the computer running the Fraglets virtual machine has to provide memory for the infinitely growing fraglet. Unlike in wet chemistry, fraglet reactions do not conserve mass (i.e. symbols). *Mass conservation* would partly solve the problem of structural explosion as we will see in

mass conservation

Chapter 18. However, mass conservation would require that all fraglet instructions are mass conserving, which would make its applicability for manually engineered programs and networking protocols more cumbersome.

3. **Infinite Seed:** An obvious but frequent case in which the closure of a CNP is infinite is when the seed is already infinite. This is the case for our packet forwarding example (see Figure 7.2), where we have to consider a potentially infinite number of distinct but similar fraglet strings representing data packets with arbitrary payload.

An infinite closure does not imply the lack of a stable dynamical fixed point. The two examples used to demonstrate arithmetic and structural explosion exhibit an infinite closure, but the number of molecules in the system does not grow unbounded. Both systems maintain a set of only two fraglet instances. Thus, an infinite closure indicates a sequence space explosion and not necessarily an unstable system with respect to the system's dynamics. However, all available mathematical tools to analyze the dynamic behavior of reaction networks require a finite sequence space, because they describe the time evolution of each distinct component separately.

undecidable

Related to the halting problem, there is *no general method to decide* whether the closure of a given set of fraglet strings is finite or infinite. The only way of generating the closure is to execute the Fraglets rewriting rules on fraglet strings (of potentially infinite length). In practice we define a threshold of N iterations after which Algorithm 7.1 stops and declares the closure infinite. This method may lead to false negatives. However, our experience showed that this rarely happens if N is approximately ten times the size of the initial set.

Infinite closures suffering arithmetic or structural explosion are usually the result of design errors. Because of the halting problem, the Fraglets virtual machine is not able to detect infinite closures in an early stage. Thus, when designing CNPs, the off-line closure check is an important verification method to make sure that the virtual machine does not suffer from memory shortage caused by structural explosion. Later, in Part III (Chapter 18), we will introduce another, energy based reaction algorithm that is able to cope with such sequence space explosions.

For most practical cases, an infinite closure arises because the seed is already infinite. In the next section, we propose a method to map potentially infinite sets of fraglet strings with similar properties to a single abstract molecule. Our hope is to obtain a finite closure in the equivalent abstract artificial chemistry.

ASSISTED ALGORITHM TO MAP FRAGLET STRINGS TO AN ABSTRACT ARTIFICIAL CHEMISTRY 7.3

In Chapter 5, we introduced the execution model by first coming up with an abstract artificial chemistry before we showed a possible realization in Fraglets. Here we go in the opposite direction by demonstrating how to construct an abstract model for a given protocol implementation in Fraglets. We present a method to obtain a function $\mathcal{F}\colon \mathcal{W}_{\text{rel}} \to \mathcal{S}$ that maps the potentially infinite closure of a set of fraglet strings, $\mathcal{W}_{\text{rel}}$, to a finite set of abstract molecular species, $\mathcal{S}$. By subsuming an infinite set under a single element we often escape the impossibility of analyzing infinite sequence spaces dynamically.

Our first mapping algorithm requires assistance from the protocol designer and is carried out as follows:

1. The designer has to describe similar fraglets with fraglet expressions. Fraglet expressions are patterns akin to regular expressions that describe a set of symbol strings. The designer's goal is that all fraglet strings that match the same expression can be mapped to the same abstract species.
2. The closure algorithm, Algorithm 7.1, is fed with these expressions instead of concrete strings. Thereby it
   a) finds the closure of the set of fraglets matching the expressions, and
   b) verifies that all fraglet strings matching the same expression follow the same reaction path. If this is the case,
3. all fraglet strings that match an expression are mapped to the same species in the explicit artificial chemistry, which is dynamically equivalent to the protocol implementation in Fraglets.

FRAGLET EXPRESSIONS 7.3.1

A *fraglet expression* is a string $\mathcal{E}^*$ over the alphabet $\mathcal{E} = \Sigma \cup \Sigma_{\text{ext}}$ where Σ is the Fraglets symbol alphabet and Σ_{ext} contains the following extension symbols:

fraglet expression

Symbol Wildcard: The *symbol wildcard*, $\diamond$, matches any single symbol from the Fraglets alphabet Σ.

symbol wildcard

Symbol Restriction: The above wildcard may optionally be subscripted with a symbol exclusion set. The symbol wildcard $\diamond_{\mathcal{X}}$ matches any single symbol from the alphabet $\Sigma \backslash \mathcal{X}$ where $\mathcal{X} \subset \Sigma$.

Kleene Star: Either of the above wildcards may be superscripted by the Kleene star in order to indicate that the wildcard matches from zero to an infinite number of symbols. Thus $\diamond^*$ matches any number of any symbol from the whole Fraglets symbol alphabet whereas the expression $\diamond^*_{\mathcal{X}}$ matches any number of symbols from the alphabet $\Sigma \backslash \mathcal{X}$.

expression matching

The following fraglet expressions on the left hand side match the indicated concrete symbol strings on the right hand side, indicated by the binary relation $\simeq$:

$$\begin{aligned}
\texttt{[match X } \diamond^* \texttt{]} &\simeq \texttt{[match X A B C]}\\
\texttt{[match X } \diamond^* \texttt{]} &\simeq \texttt{[match X]}\\
\texttt{[match X ssend } \diamond \texttt{ X]} &\simeq \texttt{[match X ssend v2 X]}\\
\texttt{[match X ssend } \diamond \texttt{ X]} &\simeq \texttt{[match X ssend v3 X]}\\
\texttt{[split } \diamond^*_{\{*\}} \texttt{ * } \diamond^* \texttt{]} &\simeq \texttt{[split a b c * d e f]}\\
\texttt{[split } \diamond^*_{\{*\}} \texttt{ * } \diamond^* \texttt{]} &\simeq \texttt{[split * g h i]}
\end{aligned}$$

The definition of fraglet expressions intentionally differs from regular expressions. Regular expressions over the alphabet Σ are usually defined inductively as $R(\Sigma) = R(\Sigma) + R(\Sigma) \mid R(\Sigma) \cdot R(\Sigma) \mid R(\Sigma)^* \mid \Sigma \mid \varepsilon$, allowing arbitrary combinations of sub-expressions. For example, the regular expression $(ab)^*$ is built by combining the concatenation of symbols with the Kleene-star. Such combinations of rules are currently not allowed for fraglet expressions in order to make the following algorithms simpler.

7.3.2 EXPRESSION REWRITING

The production rules of Fraglets, $\mathcal{P}$, are not only able to rewrite fraglet strings, but are also capable of rewriting fraglet expressions. For example, the production rule

$$\texttt{[exch } \sigma\ \alpha\ \beta\ \Phi\texttt{]} \Rightarrow \texttt{[}\sigma\ \beta\ \alpha\ \Phi\texttt{]} \tag{7.8}$$

can be applied to the expression `[exch` $\diamond$ `a` $\diamond$ `b c` $\diamond^*$`]` resulting in the following transformation

$$\texttt{[exch } \diamond \texttt{ a } \diamond \texttt{ b c } \diamond^* \texttt{]} \longrightarrow \texttt{[} \diamond\ \diamond \texttt{ a b c } \diamond^* \texttt{]} \tag{7.9}$$

expression verification

By rewriting an expression, the production rule also confirms that the expression is well formed and meets the rule's constraints. Otherwise, the production rule throws an exception, which happens in the following situations:

Single Symbol Matches Kleene-Star: If the production rule expects a symbol, indicated by a lower-case Greek letter, such as $\alpha, \beta, \ldots$, but the corresponding symbol in the left-hand side expression is decorated with the Kleene-star, the production rule throws an exception, because the Kleene-star also includes the case that the represented substring is empty. For example, the following fraglet expression will be rejected:

$$\texttt{[exch}\ \diamond^*\texttt{]}$$

The correct expression for describing all valid fraglets transformable by the `exch` transformation is

$$\texttt{[exch}\ \diamond\ \diamond\ \diamond\ \diamond^*\texttt{]}$$

Active Symbol Matches Kleene-Star: A few production rules require that a certain symbol in the left-hand side expression must not be a wildcard. These *structurally active symbols* have a crucial impact on the structure of the reaction network. For example the second symbol of the `send` production rule determines the next hop node to which the right-hand side fraglet is sent. A symbol wildcard at that position would lead to an ambiguous reaction network and therefore an exception is thrown for the following fraglet expression:

structurally active symbols

$$\texttt{[send}\ \diamond\ \diamond^*\texttt{]}$$

Symbol Scan Matches Unrestricted Wildcard: If the production rule scans a left-hand side string for the next occurrence of a certain symbol α and if, instead of α, it finds a wildcard symbol (e.g. $\diamond$ or $\diamond^*$), which does not exclude α from the alphabet, then the production rule throws an exception, because the wildcard symbol may represent the searched symbol α. For example, the following fraglet expression will be rejected, because the string wildcard before the asterisk must not contain asterisks:

$$\texttt{[split}\ \diamond^*\ \star\ \diamond^*\texttt{]}$$

The correct expression for describing all valid fraglets transformable by the `split` transformation is

$$\texttt{[split}\ \diamond^*_{\{\star\}}\ \star\ \diamond^*\texttt{]}$$

Expression Starts with Wildcard: If an expression starts with a wildcard symbol, the virtual machine cannot determine which production rule to apply and also throws an exception.

7.3.3 INFORMAL EQUIVALENCE PROOF

mapping hypothesis

This method requires the protocol designer to come up with a *hypothesis* of which fraglet strings can be summarized to an expression. Then, algorithm Algorithm 7.1 determines the closure of this set of seed expressions. This is straightforward; no modifications of Algorithm 7.1 are required. The closure algorithm iteratively enlarges the candidate expression set by executing all possible production rules on this seed set. We just require that all production rules are able to rewrite fraglet expressions as shown before and that the algorithm aborts when one of the production rules throws an exception.

If the algorithm is able to find a closure without throwing an exception, this is the proof that none of the expressions contain wildcards that lead to an ambiguous reaction network. That is, all fraglets strings that match a given expression will be processed by the same production steps and hence have an equivalent dynamic behavior. A certain fraglet expression then describes which (potentially infinite) set of fraglets can be mapped to the same abstract species. Otherwise, if the closure algorithm throws an exception, the hypothesis is invalid, meaning that more than one reaction network is possible, different fraglets matching the same rule may take different paths through the reaction network, and hence the dynamic behavior is not the same for all fraglets summarized by one of the expressions. In this case, the engineer has to come up with another hypothesis and run the closure algorithm again.

7.3.4 EXPRESSION MAPPING EXAMPLE

Consider the data packet forwarding example once more. Figure 7.4 illustrates our hypothesis, which states that all injected fraglets matching the expression ${}_{v_1}$[v3 $\diamond^*$] can be mapped to the same abstract species $P_{1,3}$. Indeed, when generating the closure, the fraglets' fate never depends on symbols represented by the wildcard $\diamond^*$. The closure algorithm terminates correctly, and hence the hypothesis is correct.

On the other hand, the closure algorithm immediately throws an exception if the expression ${}_{v_1}$[$\diamond^*$] is in the seed set. This hypothesis tries to map data packets to any destination to the same abstract species, where in our example, only packets to v_3 are forwarded.

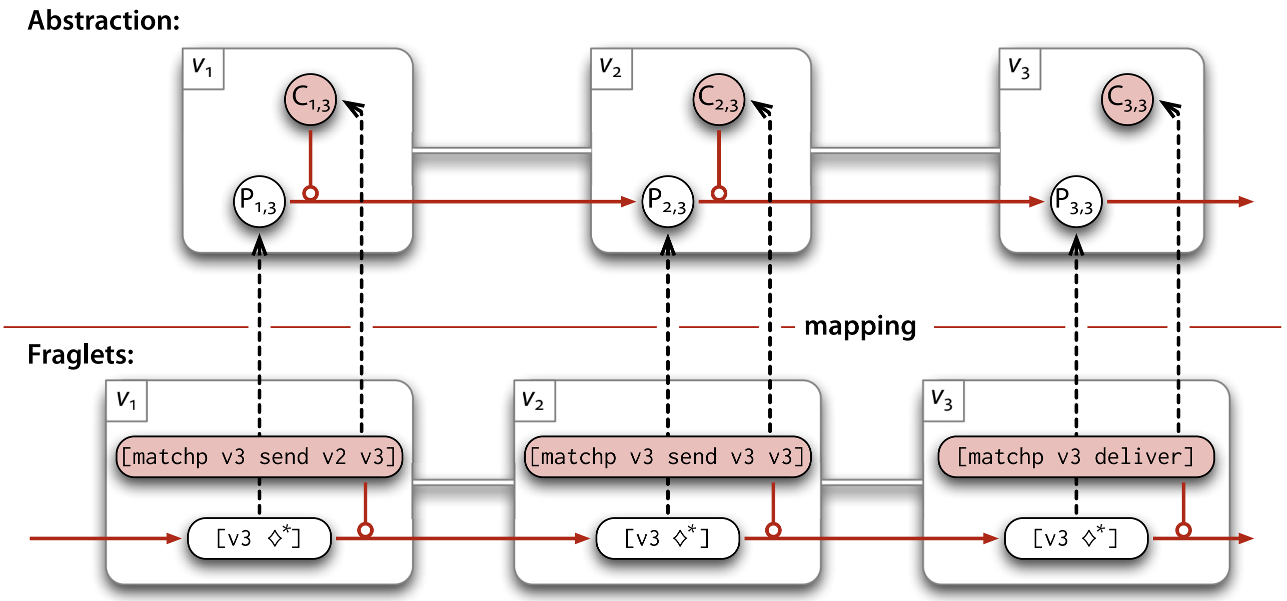


Figure 7.4 **Mapping fraglet expressions to abstract species:** All fraglets that match the expression [v3 ◇*] in the same node v_i are mapped to the same abstract species $P_{i,3}$.

AUTOMATIC ALGORITHM TO MAP FRAGLET STRINGS TO AN ABSTRACT ARTIFICIAL CHEMISTRY 7.4

Usually, a CNP designer has a good intuition about which parts of the fraglet strings can be represented by wildcards. For cases where hypotheses repeatedly fail or when the analysis is not carried out by the original protocol designer, we provide an algorithm to automatically generate valid fraglet expressions for a given protocol implementation in Fraglets. Those expressions are guaranteed to be patterns describing fraglet sets that can be subsumed under the same abstract species.

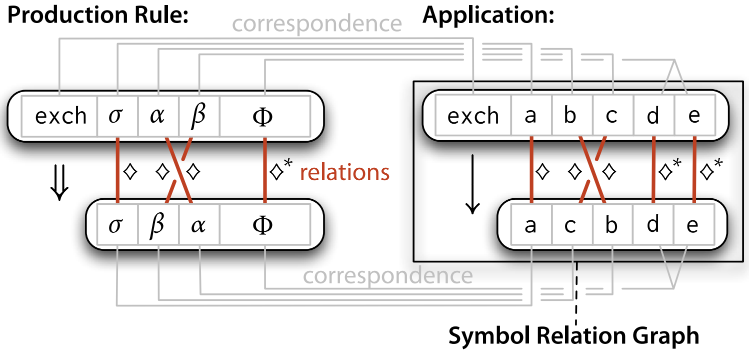
The proposed method needs samples of input fraglets together with the "program" fraglet set. It generates the closure of those concrete fraglets and tracks the fate of individual symbols by building a symbol relation graph. Based on a graph-coloring algorithm, we are able to identify those symbols in the sample fraglet strings that are not structurally important. Those free symbols can then be replaced by symbol wildcards.

SYMBOL RELATION GRAPH 7.4.1

symbol relation graph

The *symbol relation graph* connects those symbols across the fraglet strings of a closure set that are related through a rewriting step. In Fraglets, a rewriting step is implicitly defined by a production rule (see Tables 5.1 and 5.2, and Appendix B), for example

Figure 7.5 **Symbol relation graph**: The production rule relates symbols before and after the rewriting step. This symbol relation has a correspondence in the spanned reaction network. A symbol relation graph connects (edges) related symbols (nodes) and is orthogonal to the fraglet strings.



$$[\texttt{exch}\ \ \sigma\ \ \alpha\ \ \beta\ \ \Phi] \Rightarrow [\sigma\ \ \beta\ \ \alpha\ \ \Phi] \qquad (7.10)$$

substitution expression

Such a production rule describes the rewriting step with a *substitution expression*. The first fraglet on the left-hand side starts with the instruction symbol, which triggers the rule, and continues with single symbols wildcards ($\sigma, \alpha, \beta = \Sigma$) and a Kleene-star wildcard ($\Phi = \Sigma^*$). Those Greek letters indicate which symbols on the left-hand side of the production rule correspond to which symbols on the right-hand side as depicted in Figure 7.5 (left). By using this description we are able to derive a corresponding relation between the symbols of the concrete fraglets that are rewritten by the production rule (Figure 7.5, right).

The undirected labeled symbol graph is constructed while generating the closure of the seed set. For each production rule applied, symbols in the reactant and product fraglets are linked. Thereby, each edge is labeled with the corresponding wildcard symbol from the extension alphabet Σ_{ext} (see Section 7.3), for example ◇, or ◇*. If an edge is unlabeled in one of our figures, we assume the default labeling ◇*, meaning that there is neither a restriction for the alphabet nor for the length of the substring.

Figure 7.6 shows the symbol relation graph for a reaction network of a bimolecular match-synchronization followed by three subsequent transformations steps. Note that the graph consists of 10 disconnected subgraphs,* of which two arbitrary subgraphs are colored. A symbol may have many neighbors. For example, the symbols 1 and 2 are joined to symbol 3 by the ssum transformation whereas the fork instruction distributes the symbol Z to two distinct product fraglet types. The symbol relation graph captures how changes of a symbol's value propagate in the reaction network.

We have to distinguish two qualitatively different types of symbol modifications: When altering a symbol, such as 2, all connected symbols in the symbol relation graph will also change, but the shape of the reaction network

*In fact, the isolated symbols match and the two distinct M-symbols are subgraphs, too.

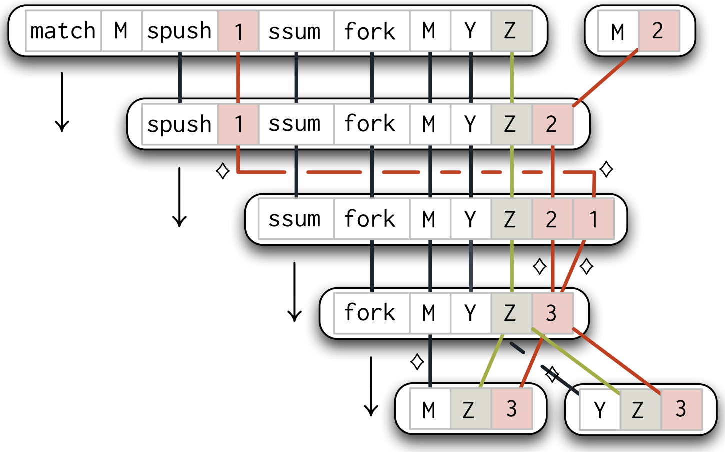


Figure 7.6 Symbol relation graph example: Several disconnected subgraphs connect the related symbols in a reaction network. Two arbitrary subgraphs are colored in this three-step transformation example.

does not. However, for example, by changing the symbol `fork` to `nop` we change the very structure of the reaction network.

Let us go back to our original plan: We would like to find out what fraglet strings can be subsumed under the same abstract species. In other words, we would like to know what symbols can be changed in a certain fraglet string without changing the dynamics, i.e. the structure of the reaction network. Thus, as a next step, we have to find out which are those dangerous symbols that modify the reaction network and which are the free symbols we are allowed to change.

FINDING STRUCTURALLY ACTIVE SYMBOLS IN THE SYMBOL RELATION GRAPH 7.4.2

The next step in finding those fraglets that can be subsumed under the same abstract species is to find those symbols that change the structure of the reaction network when altered. We call them *structurally active symbols*.

structurally active symbols

Each production rule knows its own structurally active symbols. For example, all instruction- or identifier-symbols in the header of the reactants are always structurally active. By changing the instruction symbol, we would also change the applied production rule, which will most certainly change the structure of the reaction network. In fact, for most transformation rules, the instruction symbol is the only structurally active symbol. In the definition of rewriting rules, we *underline* structurally active symbols, for example

underline

$$[\underline{\texttt{exch}}\ \sigma\ \alpha\ \beta\ \Phi] \Rightarrow [\sigma\ \beta\ \alpha\ \Phi] \tag{7.11}$$

Usually, symbols that are just moved around in the fraglet string without its value being modified are not considered active. Even arithmetic transformations, such as `ssum`, that alter symbols only mark the instruction symbol as structurally active. This is perhaps surprising since such a production rule computes an arithmetic function of two symbols, which could therefore be seen as active. But changing an operand does not change the structure of the product fraglet, so operand- and result-symbols of the `ssum` and all other arithmetic production rules are not considered structurally active. In contrast, the outcome of reaction rules such as `match` structurally depends on the matching tag, too:

$$[\underline{\texttt{match}}\ \underline{\sigma}\ \ \Phi] + [\underline{\sigma}\ \ \Psi] \Rightarrow [\Phi\ \ \Psi] \tag{7.12}$$

As a rule of thumb: those symbols that cannot be connected between the left-hand side and the right-hand side of a production rule in the symbol relation graph (see Figure 7.5, left) are usually structurally active. The comprehensive list of instructions, given in Appendix B, also indicates structurally active symbols for each Fraglets instruction.

7.4.3 ALGORITHM TO SUBSUME SETS OF FRAGLETS UNDER A SINGLE ABSTRACT SPECIES

We assume that the closure, $\mathcal{W}_{\text{rel}}$, for the set of concrete seed Fraglets, $\mathcal{W}_{\text{seed}}$, has been generated. The following algorithm maps each fraglet string in $\mathcal{W}_{\text{rel}}$ to an abstract molecular species. By building the symbol relation graph, the algorithm is able to exploit information from fraglet expressions embedded in the production rules. Hence, the algorithm is able to come up with a valid mapping hypothesis that tries to subsume an infinite set of fraglet string to a single abstract species.

1. Generate the symbol relation graph of the closure as shown in Figure 7.6 for the reaction network.

2. For each rewriting step, color the structurally active symbols in the symbol relation graph. Figure 7.7 depicts this operation for the same example (red).

3. Color those symbols that are used to capture the result of the reaction network (Figure 7.7, green). These symbols are not inspected by the reaction network itself, but rather by an external observer that expects to read the result from symbols at certain positions within certain fraglets. In our example, we expect results in fraglets starting with the symbol `X`.

4. Create additional edges for all result fraglets and label them according to the expected structure. In our example, we expect the result fraglet

matching the fraglet expression $[\mathtt{X}\ \diamond^{*}\ \diamond]$, the last symbol containing the result value.

5. For each connected subgraph of the symbol relation graph that contains at least one colored symbol, color all connected nodes as depicted in Figure 7.8. The resulting coloring marks all structurally dependent symbols, meaning that any uncolored symbol can be changed arbitrarily without changing the structure of the reaction network.

6. For each connected subgraph of the symbol relation graph that remained uncolored, build the *intersection of the edge labels*. The intersection of two wildcards computes the maximal restriction. It is formally defined as

   edge label intersection

   $$\diamond_{\mathcal{X}}^{n} \cap \diamond_{\mathcal{Y}}^{m} = \diamond_{\mathcal{X}\cap\mathcal{Y}}^{n\wedge m} \tag{7.13}$$

   where n, m are Boolean variables indicating the presence of a Kleene-star. This intersection makes sure that a symbol restriction that appears at a later rewriting step is propagated back to the precursor strings.

7. For each (non-transient) fraglet f in normal form, generate the corresponding fraglet expression e as follows:

   a) Initialize the fraglet expression to the empty string $e = \varepsilon$.

   b) Iterate over all symbols of the fraglet $f = [\sigma_1, \ldots, \sigma_l]$. For each symbol σ_i do:

      i. If σ_i is colored (structurally active symbol), append the symbol to the fraglet expression;

      ii. otherwise (free symbol), append the symbol wildcard of the connected subgraph to the fraglet expression e.

   The resulting fraglet expression e is a valid pattern for all those fraglet strings that follow the same reaction path.

8. Map every distinct fraglet expression to a separate abstract molecule as shown in Figure 7.9.

DISCUSSION 7.5

false positives

The assisted algorithm sometimes generates *false positives*, meaning that it wrongly refuses to accept a certain fraglet expression as a valid mapping hypothesis, even though all matching strings follow the same reaction path

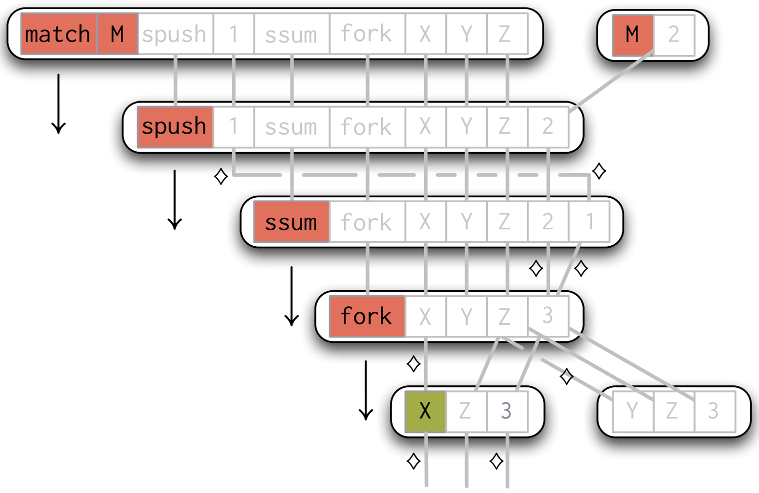


Figure 7.7 Coloring of structurally active symbols: From the Fraglets production rules we obtain the symbols that lead to a structural change of the reaction network when altered. We color them in the Fraglet strings that undergo such a rewriting step. We additionally color those symbols that constitute the result of the operation (inspected by an external observer).

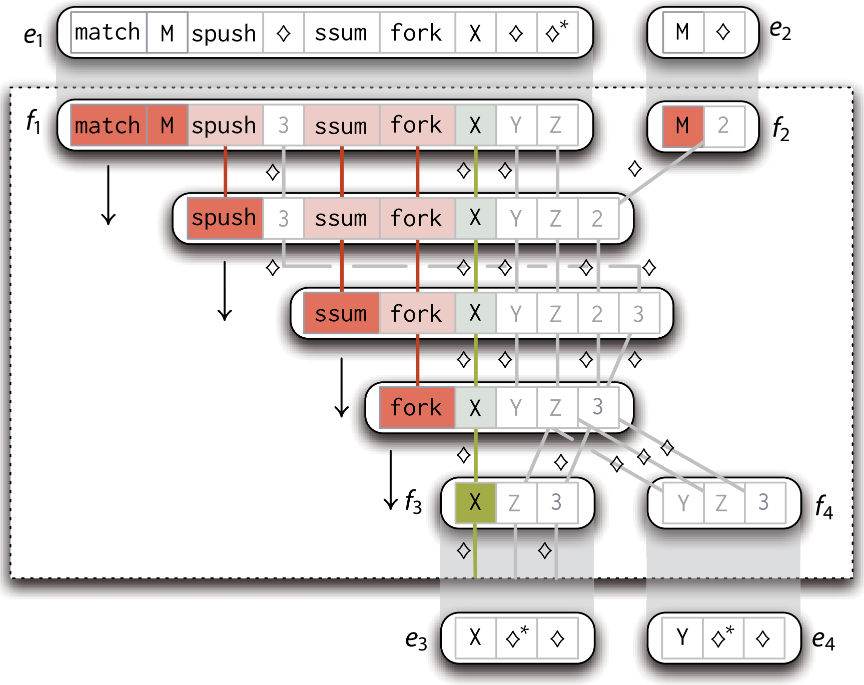


Figure 7.8 Expansion of the coloring along the subgraphs: If one node of a symbol relation subgraph is colored we color the whole connected subgraph to globally obtain all structurally active symbols in the reaction network.

and can be regarded as dynamically equivalent. For example, consider the following fraglet expression

$$[\mathtt{exch}\ \diamond^{*}\ \diamond\ \diamond\ \diamond]$$

which is refused because the exch production rule requires a single symbol at the second position. However, the above expression is equivalent to the following expression, which is accepted.

$$[\mathtt{exch}\ \diamond\ \diamond\ \diamond\ \diamond^{*}]$$

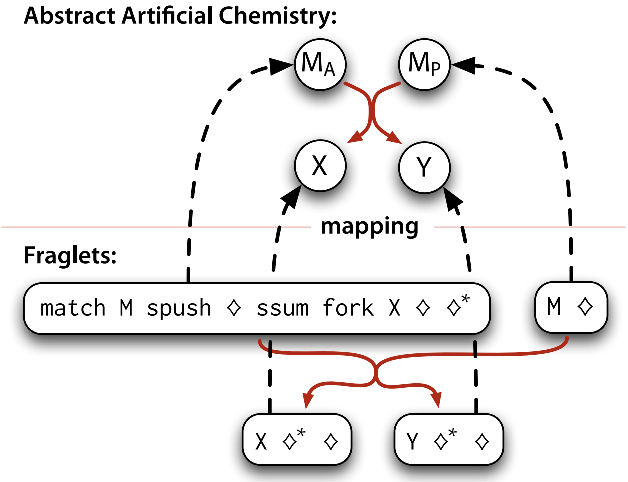

Abstract Artificial Chemistry:
M_A
M_P
X
Y
mapping
Fraglets:
match M spush ◇ ssum fork X ◇ ◇*
M ◇
X ◇* ◇
Y ◇* ◇

Figure 7.9 Mapping fraglet expressions to abstract species: All fraglet strings that match a fraglet expressions can be mapped to the same abstract species. The resulting abstract chemical universe is dynamically equivalent to the corresponding reaction system in Fraglets.

We accept this behavior for the sake of a simple and efficient closure-generating algorithm. Instead of using fraglet expressions we could *use standard regular expressions*. However, as a consequence, each production rule would have to find the equivalence between the subexpressions specified in the left-hand side of the production rule and the subexpression in the reactant fraglet. Meyer and Stockmeyer (1972) showed that the problem of determining whether two regular expressions describe the same set of strings requires exponential space (and hence exponential time).

using regular expressions instead

We see a potential to analyze the structure of Fraglets rewriting networks further using process calculi. For example, with the help of CCS, or its probabilistic variant WSCCS (Tofts, 1994; McCraig, 2007), we could perhaps prove that certain string sequences are reachable while other, harmful sequences are not. Cardelli (2008) already made the connection between chemical reaction systems and process calculi by expressing reactions in π-calculus.

SUMMARY 7.6

In this chapter, we suggested to analyze the chemical universe spanned by a protocol implementation structurally before studying the induced reaction network dynamically. The chemical universe is obtained by computing the closure of a seed set. The closure-generating algorithm can be applied to both, concrete fraglet strings as well as to fraglet expressions.

Infinite closures hinder a detailed dynamic analysis due to the sequence space explosion. For chemical networking protocols, where packets carry an arbitrary payload, infinite seed sets are the main reason for infinite closures. We proposed two methods of mapping infinite string sets to finite sets of species in an equivalent explicit artificial chemistry: For the first, assisted method, the protocol developer has to come up with hypothetical fraglet expressions describing the pattern of the strings to be combined. The closure algorithm is then able to verify the validity of this hypothesis. The second method automatically provides valid fraglet expressions for a given set of seed fraglet strings.

Such an abstraction should be possible for any algorithmic chemistry, not only for Fraglets. Thus the dynamic analysis method discussed in the next chapter can be applied to all chemical programming languages, because these methods operate on the abstract model.

CHAPTER 8

Dynamical CNP Analysis

How to exploit analysis tools from chemistry to study the dynamics of chemical networking protocols.

> One has then to develop a dynamics for such a string like structure. [8]
>
> *Bombay Lectures*
> Paul Dirac

The previous chapter revealed how an implicit artificial chemistry (Fraglets in our case) can be mapped to a dynamically equivalent abstract chemical reaction network in an explicit artificial chemistry. In this chapter, we model the behavior of this abstract model. As indicated by Figure 8.1, we review and assess existing mathematical models that describe the dynamic behavior of chemical reaction networks at different levels of accuracy and efficiency. We demonstrate how these models can be used to prove quantitative properties of CNPs.

In Section 8.1, we start by studying a rigorous mathematical description of the stochastic reaction algorithm leading to the Chemical Master Equation (CME). Because the microscopic stochastic description is often too complex to analyze, we show in Section 8.2 how to derive a macroscopic, deterministic approximation. This approximation, however, lacks a description of the stochastic noise that may be relevant for the quality of networking protocols. Therefore, in Section 8.3, we review mesoscopic descriptions where the deterministic approximation is augmented by a noise term that captures the inherent stochastic fluctuations of the random process. We illustrate all

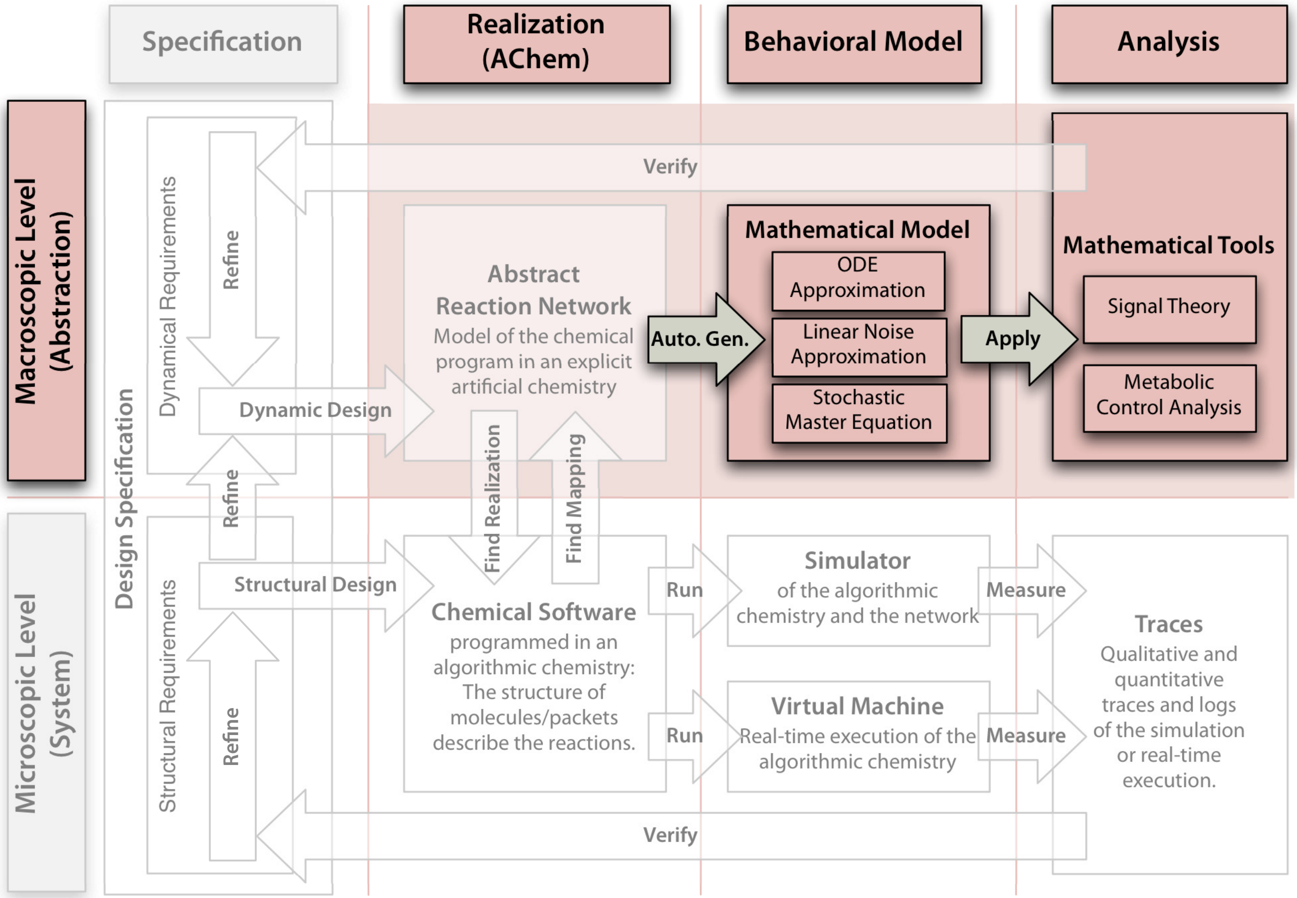

Specification
Realization (AChem)
Behavioral Model
Analysis
Macroscopic Level (Abstraction)
Microscopic Level (System)
Design Specification
Dynamical Requirements
Structural Requirements
Refine
Refine
Refine
Verify
Verify
Dynamic Design
Structural Design
Abstract Reaction Network
Model of the chemical program in an explicit artificial chemistry
Find Realization
Find Mapping
Auto. Gen.
Mathematical Model
ODE Approximation
Linear Noise Approximation
Stochastic Master Equation
Apply
Mathematical Tools
Signal Theory
Metabolic Control Analysis
Chemical Software
programmed in an algorithmic chemistry: The structure of molecules/packets describe the reactions.
Run
Run
Simulator
of the algorithmic chemistry and the network
Virtual Machine
Real-time execution of the algorithmic chemistry
Measure
Measure
Traces
Qualitative and quantitative traces and logs of the simulation or real-time execution.

Figure 8.1 **Dynamical analysis in the engineering model**: This chapter discusses mathematical models describing the dynamics of abstract reaction networks on which rigorous mathematical analysis can be carried out.

mathematical descriptions and analysis methods by applying them to a simple toy example — the *Disperser* protocol between two nodes. Readers familiar with the analysis methods of chemical reaction kinetics may skip this chapter and go directly to Chapter 9 where we prove the convergence of *Disperser* for arbitrary topologies.

8.1 MICROSCOPIC STOCHASTIC MODEL

The microscopic stochastic model provides a detailed description of the behavior of chemical reactions at the molecular level. Gillespie (1977) showed that his scheduling algorithm for artificial chemistries simulates real chemical reactions stochastically correct. Reactions are modeled by a Markov process, which leads to the Chemical Master Equation (CME). We briefly show this argument in Section 8.1.1. An analysis based on this exact stochastic description is only feasible for very simple protocols. However, because it serves as the foundation for useful approximations, it is worth sketching it here. In Section 8.1.2 we show a simplified analysis approach for CNPs with a finite number of total molecule instances (states). Sections 8.1.3 and 8.1.4 review phase-type distributions, which are used to determine the time a system needs to reach a given state. Finally, Section 8.1.5 concludes this section by referring to related work on stochastic analysis methods for chemical reaction networks.

MARKOV PROCESSES AND THE CHEMICAL MASTER EQUATION 8.1.1

Our distributed artificial chemistry, driven by an exact stochastic reaction algorithm, such as the ones described by Gillespie (1977) or Gibson and Bruck (2000), can be modeled as a continuous time discrete space *Markov jump process* on the *state space* $\mathcal{X}$, a subset of the non-negative $|\mathcal{S}|$-dimensional integer lattice $\mathbb{N}_0^{|\mathcal{S}|}$. The *state* of the system is given by the current composition, i.e. by the current number of molecules of each species. The stochastic process is a vector of random variables

Markov jump process

state space

system state

$$\mathbf{N}(t) = \left(N_{s_1}(t) \quad \dots \quad N_{s_{|\mathcal{S}|}}(t)\right)^T \tag{8.1}$$

where each random variable $N_s(t)$ refers to the number of molecules of species $s \in \mathcal{S}$ at time t. The *state probability*

conditional state probability

$$\mathbb{P}(\mathbf{n}, t) = \mathbb{P}\left[\mathbf{N}(t) = \mathbf{n} \mid \mathbf{N}(t_0) = \mathbf{n}_0\right] \tag{8.2}$$

denotes the conditional probability that at time t, the process is in state $\mathbf{n} \in \mathcal{X}$ when started at state $\mathbf{n}_0$ at t_0.

A chemical reaction rule $r \in \mathcal{R}$ is reflected as a state transition $\mathbf{n} \overset{r}{\mapsto} \mathbf{n} + \boldsymbol{\gamma}_r$ where the *stoichiometric vector* or state-change vector (Gillespie, 2002)

stoichiometric vector

$$\boldsymbol{\gamma}_r = \left(\gamma_{s_1,r} \quad \dots \quad \gamma_{s_{|\mathcal{S}|},r}\right)^T \tag{8.3}$$

corresponds to the number of molecules added and removed by the reaction rule r (see the definition for $\gamma_{s,r}$ in Section 5.1.1 and the stoichiometric vector (3.6) on page 30). Note that in the context of CNPs, a reaction can represent both, a recomposition of the local multiset or a transmission of molecules to a neighbor vessel. Hence, we are always considering the *global state*, which comprises of the compositions of all reaction vessels in the network.

global state

The state-space describes the structure of the Markov process. Additionally, we also have to define the dynamics of the process, which is given in terms of transition rates between states. In general, the transition rate for moving from state $\mathbf{n}_i$ to state $\mathbf{n}_j$ denotes the mean rate of a process with exponentially distributed event intervals, yielding a time-homogeneous process. Here we are considering exact stochastic reaction algorithms, such as those by Gillespie (1977) or Gibson and Bruck (2000), which have this property. Each reaction channel defines a field of state transitions from state $\mathbf{n}$ to state $\mathbf{n} + \boldsymbol{\gamma}_r$, whose transition rates are given by the propensity function of the reaction rule, a_r, evaluated at the origin state $\mathbf{n}$. As stated in (5.4) on page 56, the propensity function $a_r(\mathbf{n})$ defines the probability for a reaction r to occur in the next infinitesimal time interval $[t, t + \mathrm{d}t)$ and is given as

$$a_r(\mathbf{n}) = k_r \prod_{\{s \in \mathcal{S}\}} n_s^{\alpha_{s,r}} \tag{8.4}$$

Chemical Master Equation

The *Chemical Master Equation* (CME) (McQuarrie, 1967) summarizes and fully describes the stochastic dynamics of the state transitions. It can be derived directly from the propensity function (Gillespie, 1992) and is given as

$$\frac{\partial \mathbb{P}(\mathbf{n}, t)}{\partial t} = \sum_{r \in \mathcal{R}} \left(\overbrace{a_r(\mathbf{n} - \boldsymbol{\gamma}_\mathbf{r})\, \mathbb{P}(\mathbf{n} - \boldsymbol{\gamma}_\mathbf{r}, t)}^{\text{inflow}} - \overbrace{a_r(\mathbf{n})\, \mathbb{P}(\mathbf{n}, \mathbf{t})}^{\text{outflow}} \right) \tag{8.5}$$

The Chemical Master Equation states that the change of the probability of being in state $\mathbf{n}$ at time t is equal to the probability of arriving at $\mathbf{n}$ by executing reaction r, subtracted by the probability of leaving $\mathbf{n}$ by executing r (Sunkara, 2009).

8.1.2 FINITE STATE MARKOV PROCESSES

If the state-space of a continuous time Markov process is finite it is possible to simplify the Chemical Master Equation and solve it for a given state-space size. For this purpose, we first index all m states $\{\mathbf{n}_1, \dots, \mathbf{n}_m\}$ using an arbitrary scheme and write the probability distribution as row vector

$$\mathbf{p}(t) = \begin{pmatrix} p_1(t) & p_2(t) & \dots & p_m(t) \end{pmatrix} \tag{8.6}$$

where p_i denotes the state probability of state $\mathbf{n}_i$

$$p_i(t) = \mathbb{P}(\mathbf{n}_i, t) = \mathbb{P}\left[\mathbf{N}(t) = \mathbf{n}_i \mid \mathbf{N}(t_0) = \mathbf{n}_0\right]. \tag{8.7}$$

The Chemical Master Equation then simplifies to

$$\frac{\partial \mathbf{p}(t)}{\partial t} = \mathbf{p}(t)\, \mathbf{Q} \tag{8.8}$$

The $m \times m$ *transition rate matrix* $\mathbf{Q} = [q_{ij}]$, also called infinitesimal generator matrix, contains the mean transition rates from state i to state j in its off-diagonal elements:

transition rate matrix

$$q_{i,j} = \sum_{\{r \in \mathcal{R} | \mathbf{n}_i + \boldsymbol{\gamma}_\mathbf{r} = \mathbf{n}_j\}} a_r(\mathbf{n}_i) \tag{8.9}$$

That is, the $q_{i,j}$-th off-diagonal element ($i \neq j$) is obtained by summing up the propensity functions (evaluated at state $\mathbf{n}_i$) of those reactions leading from state $\mathbf{n}_i$ to state $\mathbf{n}_j$. $\mathbf{Q}$ is a *stochastic matrix*, meaning that its row sums are zero. Thus, the diagonal entries are given by $q_{i,i} = -\sum_{j \neq i} q_{i,j}$. They denote the rate at which the probability of being in state $\mathbf{n}_i$ decreases, or, said differently, $1/q_{i,i}$ denotes the expected time of resting in state $\mathbf{n}_i$. Afterwards the system moves to state $\mathbf{n}_j$ with probability $p_{i,j} = -q_{i,j}/q_{i,i}$.

stochastic matrix

By integrating (8.8) we obtain the solution for the state probability evolution, $\mathbf{p}(t)$:

$$\mathbf{p}(t) = \mathbf{p}(0) \cdot e^{\mathbf{Q}t} \tag{8.10}$$

A property we are often interested in is the *stationary probability distribution*, i.e. the state probability distribution of the system at equilibrium: $\hat{\mathbf{p}} = \mathbf{p}(t \to \infty)$. The stationary probability distribution $\hat{\mathbf{p}}$ is obtained by setting the left-hand side of (8.8) to zero: it satisfies

stationary probability distribution

$$\hat{\mathbf{p}}\mathbf{Q} = \mathbf{0} \quad \text{under the constraint} \quad \sum_i \hat{p}_i = 1,\ \hat{p}_i \geq 0 \tag{8.11}$$

This is known as the *total balance condition*, meaning that the probability flux into a state is equal to the probability flux out of that state.

total balance condition

Example: Convergence of the Disperser *Protocol Between Two Nodes* *(a)*

Some CNPs can be described by such a finite state Markov process. For example, the *Disperser* protocol maintains a constant population of molecules; hence its state-space is finite. In the following, we will calculate the stationary probability distribution of the *Disperser* protocol between two network nodes.

The abstract reaction network is spanned by the set of species $\mathcal{S} = \{\mathrm{C}_1, \mathrm{X}_1, \mathrm{C}_2, \mathrm{X}_2\}$ and the reaction rules as depicted in Figure 8.2. The state-space can be reduced from four dimensions to one because the number of C-molecules is constant (given as c_1 and c_2, respectively) and the total number of X-molecules, x_T, is conserved. We model the reaction system as a finite Markov birth-death process over the state space $\mathcal{X} = \{\mathbf{n}_0, \ldots, \mathbf{n}_{x_\mathrm{T}}\}$, where state $\mathbf{n}_i$ represents the composition of i X_1-molecules and $x_\mathrm{T} - i$ instances of type X_2. Figure 8.3

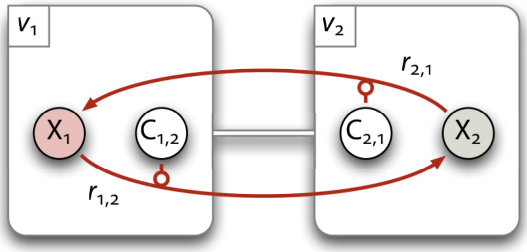


Figure 8.2 Reaction network of *Disperser* between two nodes: A finite number of X-molecules are sent forth and back between the two nodes.

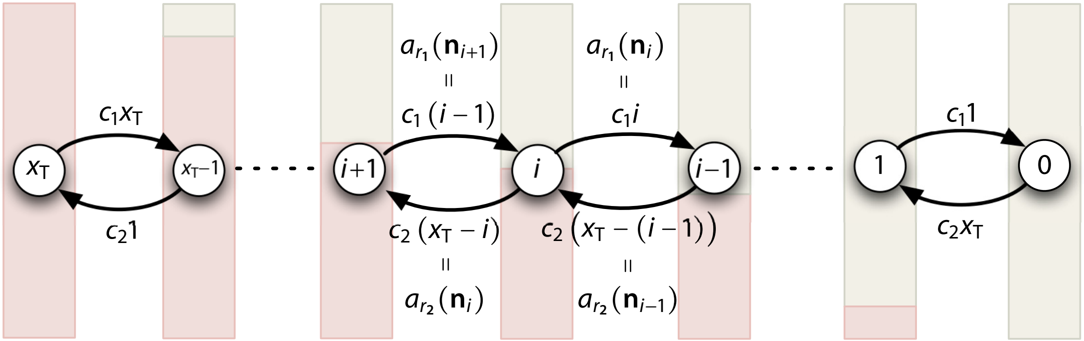


Figure 8.3 Markov chain of *Disperser*: in a network of 2 nodes for a total number of x_T X-molecules, c_1 and c_2 catalyst molecule in either node, respectively.

The state number i corresponds to the number of X_1-molecules, $N_{X_1}(t)$. The edges are labeled by the transition rates, i.e. by the propensity function of the two reactions evaluated at the current state.

depicts the transition graph of the corresponding birth-death process. Because each reaction event only adds or removes one molecule instance at a time, the elements of the $(x_T+1)\times(x_T+1)$ transition rate matrix $\mathbf{Q}$ are zero except for

$$q_{i,i-1} = a_{r_1}(\mathbf{n}_i) = c_1 i \tag{8.12a}$$

$$q_{i,i+1} = a_{r_2}(\mathbf{n}_i) = c_2\,(x_T-i) \tag{8.12b}$$

$$q_{i,i} = -\left(c_1 i + c_2\,(x_T-i)\right) \tag{8.12c}$$

In order to obtain the stationary distribution, we calculate the left null-space of the transition rate matrix $\mathbf{Q}$. This problem has already been solved for the isomerization reaction: a reversible reaction that converts a molecule into another form by rearranging its atoms:

$$X_1 \underset{c_2}{\overset{c_1}{\rightleftharpoons}} X_2 \tag{8.13}$$

binomial distribution

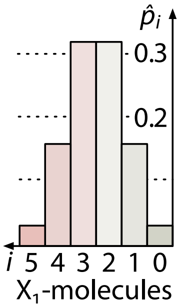
For this analogous reaction network, Gillespie (2002) showed that the stationary distribution $\hat{\mathbf{p}}$ is a *binomial distribution* (see also van Kampen, 2007):

$$\hat{p}_i = \binom{x_T}{i} q^i\,(1-q)^{x_T-i} \quad \text{where} \quad q = \frac{c_2}{c_1+c_2} \tag{8.14}$$

The stochastic model for $c_1 = c_2$ is equal to that of the Ehrenfest model (Ehrenfest & Ehrenfest, 1907; see also Takács, 1979; F. P. Kelly, 1979, Sect. 1.4), which is famous for showing *ergodic* behavior: That is, independent on the probability distribution from which the system starts, it eventually reaches the equilibrium distribution, in which those states are most likely where the particles (molecules) are equally distributed.

ergodic

For a concrete case with a total number of $x_T = 5$ molecules, and one control molecule in each node ($c_1 = c_2 = 1$), the stationary probability distribution yields



$$\hat{\mathbf{p}} = \tfrac{1}{32}\begin{pmatrix} 1 & 5 & 10 & 10 & 5 & 1 \end{pmatrix} \tag{8.15}$$

showing indeed that for $c_1 = c_2$, the most likely configuration is given when the X-molecules are equally distributed between the two nodes.

For this simple reversible reaction system, there is even a closed-form expression for the mean

$$\mathbb{E}\left[N_{X_1}\right] = x_T \frac{c_2}{c_1 + c_2} \qquad \mathbb{E}\left[N_{X_2}\right] = x_T \frac{c_1}{c_1 + c_2} \tag{8.16}$$

stating that on average, the X_1- and X_2-molecules are distributed inversely proportional to the local number of C-molecules. The variance is proportional to the total number of molecules x_T:

$$\mathrm{Var}\left[N_{X_1}\right] = \mathrm{Var}\left[N_{X_2}\right] = x_T \frac{c_1 c_2}{\left(c_1 + c_2\right)^2} \tag{8.17}$$

An interesting quantity derivable from mean and variance is the accuracy of the calculated result, i.e. the *signal-to-noise ratio* (SNR) of the calculated average, presented as steady-state quantity of the X-molecules:

signal-to-noise ratio

$$\mathrm{SNR}\left(N_{X_1}\right) = \frac{\mathbb{E}\left[N_{X_1}\right]}{\sqrt{\mathrm{Var}\left[N_{X_1}\right]}} = \sqrt{x_T \frac{c_1}{c_2}} \tag{8.18a}$$

$$\mathrm{SNR}\left(N_{X_2}\right) = \frac{\mathbb{E}\left[N_{X_2}\right]}{\sqrt{\mathrm{Var}\left[N_{X_2}\right]}} = \sqrt{x_T \frac{c_2}{c_1}} \tag{8.18b}$$

That is, the accuracy of the result increases for larger numbers of X-molecules in the system, or, said differently, the resulting average is noisier for small values.

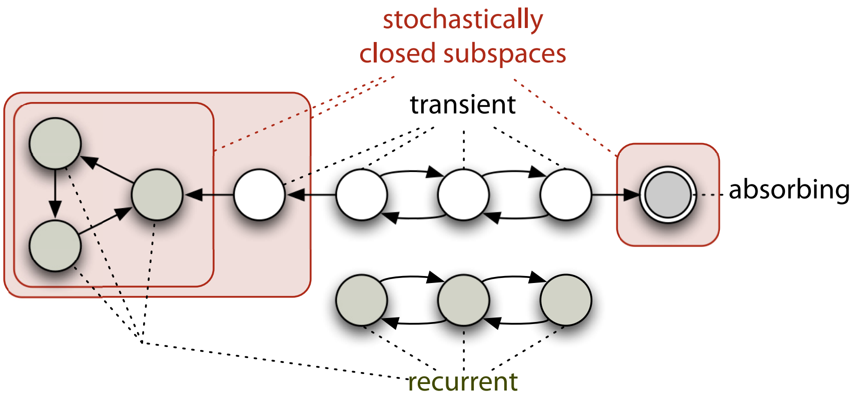


Figure 8.4 Classification of states: The transition graph depicts transitions among states in a finite Markov process.

8.1.3 CLASSIFICATION OF STATES

transition graph

A Markov process is often visualized by a *transition graph*, where the vertices represent the states and the directed and weighted edges connect those states for which a non-zero transition rate exists. Figure 8.4 depicts such a transition graph. In the following, we quickly review the usual classification of states, which is needed for the further analysis:

reachable

irreducible

stochastically closed subspace

absorbing

A state is *reachable* when there is a path to this state from any other state. The Markov process is *irreducible* if every state is reachable from every other state in a finite number of steps. If the process is not irreducible then it contains at least one *stochastically closed subspace*, in which the process stays, once it reaches any state belonging to this subspace. Note that this notion of closure is also related to the closure term from Section 7.2, here applied to the transition graph. Such a closed subspace may consist of one or more *absorbing* states, which are states in which the process stays with probability one. Additionally, the process may cycle around in different states of the closed subspace with probability one.

transient

A state is *transient* if there is a non-zero probability that the system never returns to that state, for example, when there is a reachable absorbing state. A state that is not transient is called *recurrent*.

8.1.4 CALCULATING THE FIRST-PASSAGE TIME TO ABSORPTION

Absorbing states are often undesired deadlocks and, in chemical reaction networks, result from the non-reversible extinction of a molecular species. If a reaction network yields such absorbing states we would like to know the probability of reaching them and the transition time to such a state.

The reachability question is easy to answer: Whenever an absorbing state is reachable from the initial state, the probability of eventually ending in absorption is one; in other words, a deadlock situation is guaranteed.

An interesting class of chemical reaction systems, *quasi-steady state* systems, remain near a local attractor for a very long time until they finally end in a stochastically closed subspace. For such systems, we are interested in knowing how long they survive in transient states. This first-passage time to absorption can be calculated using *phase-type distributions* according to Neuts (1981):

quasi-steady state

phase-type distribution

We first identify all absorbing states by finding the largest stochastically closed proper subset of the state space, $\mathcal{X}_0 \subset \mathcal{X}$. Since we are currently not interested in where the system ends, we may summarize all states in $\mathcal{X}_0$ into a single absorbing state Z. All other states, $\mathcal{X}_\mathrm{T} = \mathcal{X} \setminus \mathcal{X}_0$, are transient states. We define a new random variable, representing the first-passage time to Z:

$$T_{\mathcal{X}_0} = \inf\left\{t \geq 0 \colon N(t) \in \mathcal{X}_0\right\} \tag{8.19}$$

This random variable is phase-type distributed: $T_{\mathcal{X}_0} \sim \mathrm{PH}(\mathbf{Q}, \mathbf{p}_\mathrm{ini})$. The time to absorption depends on where the system starts. Thus, $\mathbf{p}_\mathrm{ini}$ denotes the initial probability distribution over the state space. The transition rate matrix $\mathbf{Q}$ is re-sorted such that the absorbing state Z is at the bottom:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_\mathrm{T} & \mathbf{Q}_0 \\ \mathbf{0} & 0 \end{pmatrix} \tag{8.20}$$

$\mathbf{Q}_\mathrm{T}$ is the nonsingular transition matrix among the transient states and $\mathbf{Q}_0 = -\mathbf{Q}_\mathrm{T} \cdot \mathbf{1}$ is the vector of transition rates to the absorbing state Z. ($\mathbf{1}$ is a column vector of all ones.)

Neuts (1981) proved that the *cumulative distribution function* of the first passage time $T_{\mathcal{X}_0}$ is given by

cumulative distribution function

$$F(t) = \mathbb{P}\left[T_{\mathcal{X}_0} \leq t\right] = 1 - \mathbf{p}_\mathrm{ini} e^{\mathbf{Q}_\mathrm{T} t} \tag{8.21}$$

the *probability density function* is

probability density function

$$f(t) = \mathbf{p}_\mathrm{ini} e^{\mathbf{Q}_\mathrm{T} t} \mathbf{Q}_0 \tag{8.22}$$

and the first moment, i.e. the *mean time to absorption* is

mean time to absorption

$$\mathbb{E}\left[T_{\mathcal{X}_0}\right] = -\mathbf{p}_\mathrm{ini} \mathbf{Q}_\mathrm{T}^{-1} \mathbf{1} \tag{8.23}$$

Neuts' method works for any finite continuous time Markov jump process, not only for birth-death chains. Note however, that the computational complexity rapidly increases with the number of states, since we have to compute the inverse of a $|\mathcal{X}_\mathrm{T}|$-dimensional sparse matrix.

For finite birth-death processes with two absorbing states at the opposite ends of the Markov chain, Glaz (1979) provided an explicit formula for the mean of the first absorption time. The transition rates among the neighbor states of the space $\mathcal{X} = \{0, 1, \ldots, N\}$ are given as birth rate λ_i and death rate μ_i with $\lambda_0 = \lambda_N = \mu_0 = \mu_N = 0$. The mean time to absorption when starting in state k with probability one is given by Glaz as

$$\mathbb{E}\left[T_{\{0,N\}}\right] = \frac{\sum_{s=1}^{N-1} u_-(s)\, u_+(s)}{u_{\text{tot}}} \tag{8.24}$$

where

$$u_-(s) = \sum_{i=1}^{\min(k,s)} \left(\prod_{j=1}^{i-1} \mu_j \cdot \prod_{j=i}^{s-1} \lambda_j \right) \tag{8.25a}$$

$$u_+(s) = \sum_{i=1}^{N-s-\max(k-s,0)} \left(\prod_{j=s+1}^{N-i} \mu_j \cdot \prod_{j=N-i+1}^{N-1} \lambda_j \right) \tag{8.25b}$$

$$u_{\text{tot}} = \sum_{i=1}^{N} \left(\prod_{j=1}^{N-i} \mu_j \cdot \prod_{j=N-i+1}^{N-1} \lambda_j \right) \tag{8.25c}$$

Even though the above equation is not a closed-form expression, it reduces the complexity of the calculation tremendously compared to the matrix inversion method.

(a) *Example: Convergence Time of the* Disperser *Protocol Between Two Nodes*

Neuts' phase-type distribution can also be used to calculate the first-passage time to non-absorbing states. In this example we demonstrate how to compute the convergence time of *Disperser* between two nodes.

We assume that x_T X-molecules are initially placed in node 1 while node 2 starts without X-molecules, and that both nodes contain the same amount of catalysts. We previously learned that for $c_1 = c_2$ the most likely state is $\mathbf{n}_{x_T/2}$ where the X-molecules are equally distributed between the two nodes. We are interested in the first passage time from the initial state to the most likely state.

Note that we do not care what happens after the most likely state is reached. Thus, we simply define this state as absorbing by removing all arrows leaving it (see the original Markov chain in Figure 8.3). We then calculate the mean first passage time to this artificially introduced absorbing state according to (8.23).

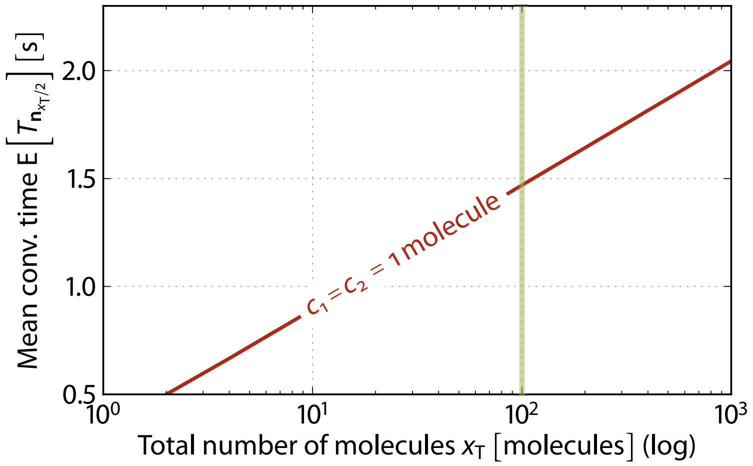


Figure 8.5 Convergence time of *Disperser* (1): First passage time to the most likely state, $\mathbf{n}_{x_T/2}$, for $c_1 = c_2$. The first passage time is plotted with respect to the total number of molecules x_T, which are initially placed in node 1.

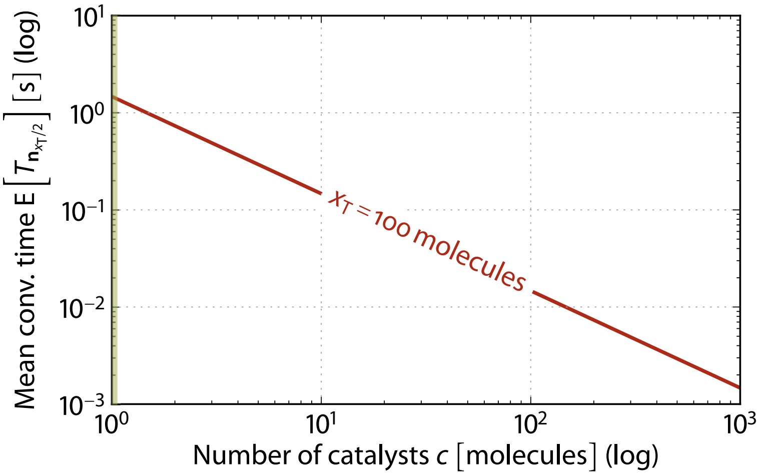


Figure 8.6 Convergence time of *Disperser* (2): First passage time to the most likely state, $\mathbf{n}_{x_T/2}$, for $c_1 = c_2$. The first passage time is plotted with respect to the number of catalysts in either node $c = c_1 = c_2$. The total amount of $x_T = 100$ molecules are initially placed in node 1.

Figure 8.5 shows how fast *Disperser* converges to equilibrium for one catalyst molecule in each node, plotted with respect to the total number of X-molecules, x_T. The convergence time logarithmically increases with the total number of molecules. Although the protocol has to send more molecules to the neighbor node for increasing x_T, the dependency is not linear. This is due to the law of mass action scheduling, which causes the non-equilibrium migration rate to grow with the number of reactants.

For a fixed total number of $x_T = 100$ molecules, Figure 8.6 plots the convergence time with respect to the number of catalyst molecules. In the region of only a few catalysts, the convergence time can be drastically lowered by just adding a few more C-molecules to all nodes. As we will see later, this comes along with an increased message complexity.

8.1.5 DISCUSSION

A detailed analysis based on the Chemical Master Equation (CME) is only feasible for simple CNPs, because the CME contains a separate equation for each state $\mathbf{n}_i$. That is, the state-space and thus the dynamic description of the system grows exponentially with the number of species (Ferm, Lötstedt, & Hellander, 2008). This is why it is hard to solve the CME analytically or numerically for most practical systems.

closed reaction systems

In *closed reaction systems* the total number of molecules is finite, and the system can be described by a finite state Markov process: A finite number of linear differential equations describe the transient behavior and a finite number of linear equations (8.11) yield the stationary probability distribution.

open reaction systems

If the total number of molecules is not bounded by a conservation relation the state-space is infinite. This is the case for *open reaction systems* for which there is an inflow of molecule instances. Even if an equilibrium distribution exists there is no theoretical limit in the number of molecules. CNPs are open reaction systems as soon as user-generated molecules are injected, which is a common scenario. In this respect, the *Disperser* protocol, being closed after initialization, is a counter-example.

For first-order (linear) reaction networks, the following generic solutions have been obtained by Gadgil, Lee, and Othmer (2005) and Jahnke and Huisinga (2007): The steady-state probability distribution of a linear closed reaction system is a multinomial distribution. If the linear reaction system has additional inflow and outflow reactions, the steady-state probability distribution is a Poisson distribution. The latter even holds for open bimolecular reaction systems (Gelenbe, 2008). Gadgil et al. (2005) provided closed-form expressions for the first two moments (mean and variance) of open and closed linear reaction systems.

The research field on Markov jump processes is still very active. Several results have been published recently and will be published in the future that may also be applied to the analysis of CNPs. For example, the transition path theory (Metzner, Schütte, & Vanden-Eijnden, 2009) provides a method to calculate the statistical properties of a transition between two selected subsets of the state-space.

However, for the majority of practical CNPs, a detailed microscopic dynamic analysis is too complex. This is why in the next section, we review how the inherently stochastic reaction process can be approximated by deterministic differential equations.

MACROSCOPIC DETERMINISTIC APPROXIMATION 8.2

One method to reduce the complexity of the stochastic system is to examine the mean time evolution of the system, i.e. the average dynamic behavior of the same system for different trials. In Section 8.2.1, we derive the deterministic Reaction Rate Equation (RRE), a set of Ordinary Differential Equations (ODEs) describing the trajectories of molecular concentrations. Then, we are focusing on two analysis methods for chemical reaction networks that are basing on this macroscopic description: Section 8.2.2 first demonstrates how to prove the existence of equilibria using linear stability analysis. This is especially useful to prove the convergence of CNPs. In Section 8.2.3, we then use results from Metabolic Control Analysis (MCA) and control theory to analyze the transient behavior of chemical reaction networks.

DERIVATION OF THE REACTION RATE EQUATION 8.2.1

Instead of keeping track of the full state distribution, here we are only interested in the time evolution of the *quantity of species*

quantity of species

$$x_s(t) = \mathbb{E}\left[N_s(t)\right] \tag{8.26}$$

for every molecule $s \in \mathcal{S}$. The following equation captures the change of those expected quantities. It can be derived from the Chemical Master Equation (8.5) (Gillespie, 2000; van Kampen, 2007; Ullah & Wolkenhauer, 2009a).

$$\frac{\partial x_s(t)}{\partial t} = \sum_{r \in \mathcal{R}} \gamma_{s,r} \cdot \mathbb{E}\left[a_r\big(\mathbf{N}(t)\big)\right] \qquad \forall i \in \mathcal{S} \tag{8.27}$$

If the propensity function $a_r\big(\mathbf{N}(t)\big)$ is a linear function, which is the case if reaction rule r is unimolecular, such as $\mathrm{A} \rightarrow \mathrm{B} + \mathrm{C}$, then $\mathbb{E}\left[a_r(\mathbf{N})\right] = a_r\big(\mathbb{E}\left[\mathbf{N}\right]\big) = a_r(\mathbf{x})$ and the above equation can be expressed in terms of the macroscopic, continuous variable $\mathbf{x}(t)$:

$$\frac{\partial x_s(t)}{\partial t} = \sum_{r \in \mathcal{R}} \gamma_{s,r} a_r\big(\mathbf{x}(t)\big) \qquad \forall i \in \mathcal{S} \tag{8.28}$$

or, in vector notation

$$\dot{\mathbf{x}} = \mathbf{S} \cdot \mathbf{a}(\mathbf{x}) \tag{8.29}$$

where $\mathbf{S} = [\gamma_{s,r}]$ is the $|\mathcal{S}| \times |\mathcal{R}|$ stoichiometric matrix and $\mathbf{a}\big(\mathbf{x}(t)\big)$ is the $|\mathcal{R}|$-dimensional propensity vector evaluated with the macroscopic, continuous variable.

However, if r is a multi-molecular reaction rule, and hence $a_r\big(\mathbf{N}(t)\big)$ is a nonlinear function, then the above assumption is only an approximation, since higher-order moments enter as well (van Kampen, 2007):

$$\mathbb{E}\left[a_r(\mathbf{N})\right] = a_r\big(\mathbb{E}\left[\mathbf{N}\right]\big) + \frac{1}{2}\mathbb{E}\left[\big(\mathbf{N} - \mathbb{E}\left[\mathbf{N}\right]\big)^2\right] a_r''\big(\mathbb{E}\left[\mathbf{N}\right]\big) \tag{8.30}$$

That is, (8.28) only approximates the average behavior by ignoring the higher order moments; this approximation is adequate if the fluctuations are small. Later, in Section 8.3.2 on Linear Noise Approximation, we will show how to estimate these fluctuations.

If we are only interested in an approximated average behavior of our virtual chemical reaction system, and if we ignore the fact that for multi-molecular reactions the average behavior can be influenced by stochastic variation, we may substitute $x_s(t) = \mathbb{E}\left[N_s(t)\right]$ yielding the deterministic macroscopic *reaction rate equation*

reaction rate equation

$$\frac{\partial \mathbf{x}(t)}{\partial t} = \mathbf{S} \cdot \mathbf{a}\big(\mathbf{x}(t)\big), \tag{8.31}$$

a set of Ordinary Differential Equations (ODE), in vector notation

$$\dot{\mathbf{x}} = \mathbf{S} \cdot \mathbf{a} \tag{8.32}$$

Note the difference between n_s, $N_s(t)$, and $x_s(t)$: n_s is a realization of the discrete random variable $N_s(t)$, which exists for each molecular species $s \in \mathcal{S}$. This random variable is stochastically changed according to the state transitions (propensity functions). Finally, $x_s(t)$ is the continuous variable that approximates the expected number of species s at time t. Thus $x_s(t)$ is a macroscopic variable that ignores the stochasticity of the random process. At the macroscopic level, the propensity vector $\mathbf{a}$ can be seen as the reaction rate vector; it denotes how frequently reactions occur on average.

8.2.2 LINEAR STABILITY ANALYSIS

Linear stability analysis determines whether an arbitrary reaction network has a stable fixed point based on the deterministic description of the system. In order to cope with non-linear (i.e. multi-molecular) reaction systems it linearizes the equations around the fixed points (Strogatz, 1994; Otto & Day, 2007).

From the ODE system (8.32), which approximates the dynamic behavior of a chemical reaction network, the *fixed points* can be obtained by setting

fixed point

the time derivatives of the quantity variables to zero, $\dot{\mathbf{x}} = \mathbf{0}$, and by solving the resulting equation system $\mathbf{Sa}(\hat{\mathbf{x}}) = \mathbf{0}$ with respect to the macroscopic steady-state quantity vector $\hat{\mathbf{x}}$.

Not all fixed points are stable: A stable fixed point is a state to which the system returns after a small *perturbation*. Let's assume that $\hat{\mathbf{x}}$ is a fixed point and $\tilde{\mathbf{x}}(t) = \mathbf{x}(t) - \hat{\mathbf{x}}$ is a small perturbation relative to it. The ODE for the perturbation variable, $\tilde{\mathbf{x}}$, is identical to (8.32):

perturbation

$$\frac{\partial \mathbf{x}}{\partial t} = \frac{\partial\left(\hat{\mathbf{x}} + \tilde{\mathbf{x}}\right)}{\partial t} = \overbrace{\frac{\partial \hat{\mathbf{x}}}{\partial t}}^{=0} + \frac{\partial \tilde{\mathbf{x}}}{\partial t} = \mathbf{Sa}(\hat{\mathbf{x}} + \tilde{\mathbf{x}}) \tag{8.33}$$

In order to efficiently treat multi-molecular reactions, the reaction rate function is linearized around the fixed point. That is, $\mathbf{a}(\mathbf{x})$ is Taylor expanded around $\mathbf{x} = \hat{\mathbf{x}}$:

$$\frac{\partial \tilde{\mathbf{x}}}{\partial t} = \mathbf{S}\left(\mathbf{a}(\hat{\mathbf{x}}) + \left.\frac{\partial \mathbf{a}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}} \cdot \tilde{\mathbf{x}} + \cdots\right) \tag{8.34}$$

where $\frac{\partial \mathbf{a}(\mathbf{x})}{\partial \mathbf{x}}$ is the $|\mathcal{R}| \times |\mathcal{S}|$ matrix

$$\frac{\partial \mathbf{a}(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial a_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial a_1(\mathbf{x})}{\partial x_{|\mathcal{S}|}} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_{|\mathcal{R}|}(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial a_{|\mathcal{R}|}(\mathbf{x})}{\partial x_{|\mathcal{S}|}} \end{pmatrix} \tag{8.35}$$

The matrix

$$\mathbf{J}(\mathbf{x}) = \mathbf{S}\,\frac{\partial \mathbf{a}(\mathbf{x})}{\partial \mathbf{x}} \tag{8.36}$$

is called the *Jacobian matrix*. It contains all information about whether and how fast the system returns to the fixed point once perturbed along each molecular quantity. We drop the second and higher order derivatives in (8.34), and by recognizing that $\mathbf{Sa}(\hat{\mathbf{x}}) = \mathbf{0}$, we obtain the linear approximation for the perturbation variable, a linear differential equation system.

Jacobian matrix

$$\dot{\tilde{\mathbf{x}}} = \left.\mathbf{J}(\mathbf{x})\right|_{\mathbf{x}=\hat{\mathbf{x}}} \cdot \tilde{\mathbf{x}} \tag{8.37}$$

The fixed point is stable if the perturbations decay over time. For a linear system, this holds if the real part of all eigenvalues of the matrix $\mathbf{J}(\mathbf{x})|_{\mathbf{x}=\hat{\mathbf{x}}}$ are negative. If the real part of the leading eigenvalue is zero, then the stability analysis is inconclusive and higher order terms have to be considered. If the eigenvalues have complex parts, the system spirals around the fixed point, but eventually reaches it.

convergence proof

Linear stability analysis is a powerful tool to *prove the convergence* of CNPs, which often present their result at equilibrium. Thus, by showing that the system has the desired fixed point and that this fixed point is stable, we can prove that the protocol behaves correctly. In the following example, we prove *Disperser's* stability in a trivial network of two nodes. The full proof for arbitrary topologies follows later in Chapter 9.

(a) *Example: Linear Stability Analysis Applied to the* Disperser *Protocol Between Two Nodes*

The *Disperser* protocol consists of unimolecular reactions only. Thus, a linearization is actually not required. Anyway, we apply it here for pedagogical reason to the *Disperser* protocol between two nodes (see Figure 8.2 on page 108).

We already mentioned before that the number of catalysts remains constant and hence we don't model the change of the C-molecules and set $c_1 = x_{C_1} = \text{const}$ and $c_2 = x_{C_2} = \text{const}$. The ODE system that approximates the two-node *Disperser* protocol is given as $\dot{\mathbf{x}} = \mathbf{S}\mathbf{a}(\mathbf{x})$

$$\overbrace{\begin{pmatrix} \dot{x}_{X_1} \\ \dot{x}_{X_2} \end{pmatrix}}^{\dot{\mathbf{x}}} = \overbrace{\begin{pmatrix} -1 & +1 \\ +1 & -1 \end{pmatrix}}^{\mathbf{S}} \cdot \overbrace{\begin{pmatrix} c_1 x_{X_1} \\ c_2 x_{X_2} \end{pmatrix}}^{\mathbf{a}(\mathbf{x})} \tag{8.38}$$

The fixed point satisfies the expression $c_1 \hat{x}_{X_1} = c_2 \hat{x}_{X_2}$. We also showed before that the total number of X-molecules, x_T, is conserved. Hence, we substitute $x_{X_2} = x_T - x_{X_1}$ yielding the fixed point

$$\hat{x}_{X_1} = \frac{c_2}{c_1 + c_2} x_T \qquad \hat{x}_{X_2} = \frac{c_1}{c_1 + c_2} x_T \tag{8.39}$$

Next, we linearize the system by calculating the Jacobian matrix

$$\mathbf{J}(\mathbf{x}) = \overbrace{\begin{pmatrix} -1 & +1 \\ +1 & -1 \end{pmatrix}}^{\mathbf{S}} \overbrace{\begin{pmatrix} c_1 & 0 \\ 0 & c_2 \end{pmatrix}}^{\frac{\partial \mathbf{a}(\mathbf{x})}{\partial \mathbf{x}}} = \begin{pmatrix} -c_1 & c_2 \\ c_1 & -c_2 \end{pmatrix} \tag{8.40}$$

The fixed point is stable if the eigenvalues of $\mathbf{J}(\mathbf{x})|_{\mathbf{x}=\hat{\mathbf{x}}}$ are negative. The eigenvalues are calculated from $\det\big(\mathbf{J}(\mathbf{x})|_{\mathbf{x}=\hat{\mathbf{x}}} - \lambda\mathbf{I}\big) = 0$, which results in the characteristic equation $(-c_1 - \lambda)\,(-c_2 - \lambda) - c_1 c_2 = 0$, yielding the eigenvalues $\lambda_1 = 0$, and $\lambda_2 = -\,(c_1 + c_2)$. In this example, one of the eigenvalues is zero, meaning that a perturbation in the number of X-molecules is persistent. Actually, this is the goal of the *Disperser* protocol: to continuously react on changes and re-calculate the average number of molecules. In this respect, the fixed point of the *Disperser* protocol between two nodes is indifferent.

METABOLIC CONTROL ANALYSIS 8.2.3

Unlike perturbation analysis, which studies equilibria, Metabolic Control Analysis (MCA) offers a framework to analyze the *transient response* of reaction networks to various external and internal perturbation channels. With this method, we are able to calculate the sensitivity of a reaction network to sudden changes of molecule concentrations, changes of the reaction coefficient, and fluctuations of a packet stream.

transient response

Metabolic Control Analysis (MCA) was proposed in the 1970ies by Kacser and Burns (1973) and Heinrich, Rapoport, and Rapoport (1977) as a framework to analyze metabolic pathways and genetic networks. The theory aims to link perturbation around the steady state in individual network components to steady-state changes in the systematic behavior of the network (Rao, Sauro, & Arkin, 2004). MCA has further matured since its first description (Heinrich & Schuster, 1996; Fell, 1997). A good step-by-step introduction is given by Reder (1988) whereas Hofmeyr (2001) provides a distilled reference. From the huge number of publications and approaches we focus on a description by Ingalls (2004), where the dynamic behavior of chemical reaction networks is analyzed in the frequency domain.

The analysis procedure follows the following steps: First, the system has to be described as a linear time-invariant (LTI) state-space model. For this purpose, we have to linearize the system around its fixed points. Next, we define what parameters we want to perturb, for example the quantity of species or reaction coefficients. Those parameters are the input signals of the model whereas the quantity of species form the state-space. Then, by applying standard methods from control theory, we are able to calculate the frequency response of the system, that is, the effect of a perturbation of one of the parameters to one of the observed outputs. In the following, we elaborate more on this procedure and apply it to the two-node *Disperser* example.

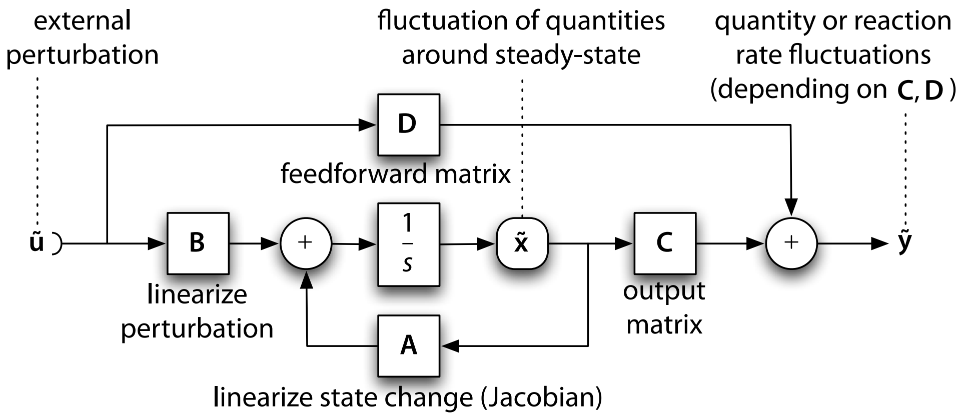

external perturbation
fluctuation of quantities around steady-state
quantity or reaction rate fluctuations (depending on C, D)
D
feedforward matrix
ũ
B
+
1/s
x̃
C
+
ỹ
linearize perturbation
output matrix
A
linearize state change (Jacobian)

Figure 8.7 **Linearized perturbation state-space model**: A traditional Linear Time-Invariant System analysis approach is enabled by linearizing the feedback and the perturbation effect around the fixed point.

(a) State-Description of the Linearized Reaction System

As introduced before, a chemical reaction system can be described by a set of deterministic ODEs

$$\dot{\mathbf{x}}(t) = \mathbf{Sa}\big(\mathbf{x}(t), \mathbf{u}(t)\big) \tag{8.41}$$

where $\mathbf{x}(t)$ denotes the vector of approximated number of molecules, $\mathbf{S}$ is the stoichiometric matrix, and $\mathbf{a}\big(\mathbf{x}(t), \mathbf{u}(t)\big)$ is the vector of reaction rate functions. We assume that the reactions obey the law of mass action; therefore, the reaction rate depends on the number of reactant molecules, indicated by $\mathbf{x}$. MCA additionally models the fact that the reaction coefficient may be disturbed by a vector of *external perturbations*, $\mathbf{u}(t)$.

external perturbations

For a perturbation analysis we are only interested in the fluctuations around the reaction system's steady state $\hat{\mathbf{x}}$. Thus, like for the linear stability analysis before, we introduce the following perturbation variables, which denote the offset from the fixed point:

$$\tilde{\mathbf{x}}(t) = \mathbf{x}(t) - \hat{\mathbf{x}} \tag{8.42a}$$
$$\tilde{\mathbf{u}}(t) = \mathbf{u}(t) - \hat{\mathbf{u}} \tag{8.42b}$$

The linearized state-space model of the perturbation, also depicted in Figure 8.7, takes the form

$$\dot{\tilde{\mathbf{x}}}(t) = \mathbf{A}\tilde{\mathbf{x}}(t) + \mathbf{B}\tilde{\mathbf{u}}(t) \tag{8.43a}$$
$$\tilde{\mathbf{y}}(t) = \mathbf{C}\tilde{\mathbf{x}}(t) + \mathbf{D}\tilde{\mathbf{u}}(t) \tag{8.43b}$$

state matrix

The *state matrix* $\mathbf{A}$ defines how a change of the quantities (state) affects their future change. This matrix is equal to the Jacobian matrix evaluated at the fixed point

$$\mathbf{A} = \mathbf{J}(\mathbf{x})\Big|_{\substack{\mathbf{x}=\hat{\mathbf{x}}\\ \mathbf{u}=\hat{\mathbf{u}}}} = \mathbf{S}\,\frac{\partial \mathbf{a}}{\partial \mathbf{x}}\Big|_{\substack{\mathbf{x}=\hat{\mathbf{x}}\\ \mathbf{u}=\hat{\mathbf{u}}}} \tag{8.44}$$

The *input matrix* **B** indicates how external perturbations affect the fluctuations of the state, also evaluated at the fixed point

input matrix

$$\mathbf{B} = \mathbf{S}\,\frac{\partial \mathbf{a}}{\partial \mathbf{u}}\Big|_{\substack{\mathbf{x}=\hat{\mathbf{x}}\\ \mathbf{u}=\hat{\mathbf{u}}}} \tag{8.45}$$

The *output matrix* **C** and the *feedthrough matrix* **D** define what output measures we are interested in. We have to chose them appropriately depending on the response we want to analyze. There are two typical cases for which we give **C** and **D**: (1) Often, we want to study the fluctuation of molecular quantities with respect to perturbations of the input parameters. In this case we use

output and feedthrough matrix

$$\mathbf{C} = \mathbf{I}_{|\mathcal{S}|} \qquad \mathbf{D} = \mathbf{0} \tag{8.46}$$

yielding $\tilde{\mathbf{y}}(t) = \tilde{\mathbf{x}}(t)$. (2) For the other frequent case, where we want to examine the fluctuation of the reaction rates with respect to perturbations of the input parameters, we use

$$\mathbf{C} = \frac{\partial \mathbf{a}}{\partial \mathbf{x}}\Big|_{\substack{\mathbf{x}=\hat{\mathbf{x}}\\ \mathbf{u}=\hat{\mathbf{u}}}} \qquad \mathbf{D} = \frac{\partial \mathbf{a}}{\partial \mathbf{u}}\Big|_{\substack{\mathbf{x}=\hat{\mathbf{x}}\\ \mathbf{u}=\hat{\mathbf{u}}}} \tag{8.47}$$

Frequency and Step Response (b)

An LTI system, such as the linearized perturbation description, can be analyzed not only in time domain, but also in frequency domain. The *Laplace transform* converts a function in the time domain, $f(t)$, to its corresponding image function in the frequency domain, $F(s)$:

Laplace transform

$$F(s) = \mathcal{L}\left\{f(t)\right\} = \int_0^\infty e^{-st} f(t)\,\mathrm{d}t \tag{8.48}$$

The *frequency response* $H(s)$ of an LTI system is given by its transfer function. It describes the frequency-dependent gain of the output with respect to the input. The frequency response of the linearized perturbation system around the steady state, i.e. the transient behavior of system's deviation from the fixed point $\tilde{\mathbf{Y}}(s)$ with respect to a perturbation on the input $\tilde{\mathbf{U}}(s)$, is given as

frequency response

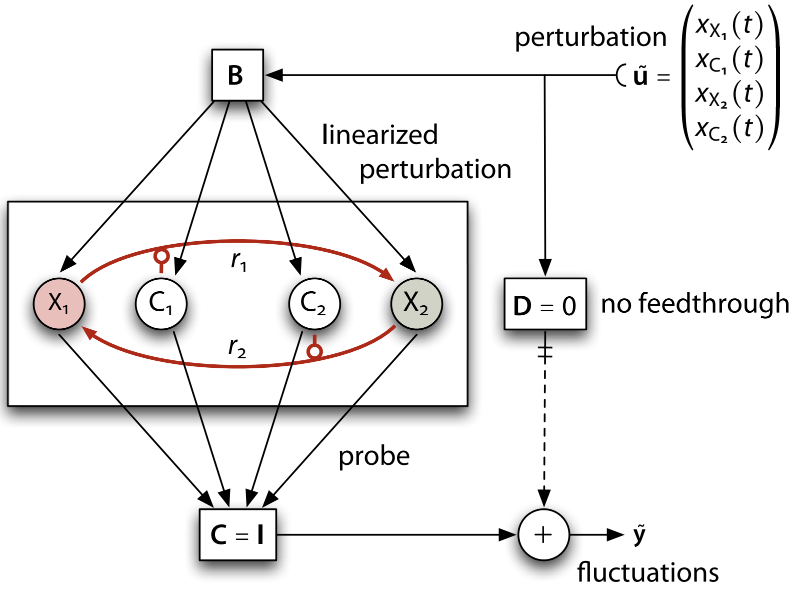


Figure 8.8 Linearized perturbation state-space model of *Disperser*: Our aim is to analyze the cross-effect of a perturbation of the molecule's quantity.

$$\mathbf{H}(s) = \frac{\tilde{\mathbf{Y}}(s)}{\tilde{\mathbf{U}}(s)} = \mathbf{C}\left(s\mathbf{I}_{|\mathcal{S}|} - \mathbf{A}\right)^{-1}\mathbf{B} + \mathbf{D} \tag{8.49}$$

step response

This matrix-valued function will return the gain of each output (column) with respect to each perturbation channel (row). The corresponding *step response* is obtained by dividing **H** by s and transforming the resulting function back from frequency to time domain using the inverse Laplace transform by calculating the contour integral ($s = \sigma + i\omega$):

$$\mathbf{Y}(t) = \mathcal{L}^{-1}\left\{\frac{\mathbf{H}(s)}{s}\right\} = \frac{1}{2\pi i}\int_{\sigma-i\omega}^{\sigma+i\omega}\frac{\mathbf{H}(s)}{s}e^{st}\,\mathrm{d}s \tag{8.50}$$

(c) *Example: Perturbation Analysis Applied to the* Disperser *Protocol Between Two Nodes*

As an example, we perform a sensitivity analysis of the *Disperser* protocol in a simple network topology of two nodes using the presented MCA method.

As shown in Figure 8.8, our LTI system is represented by a chemical reaction network, for which we have already given the ODEs in (8.38) and calculated the fixed point in (8.39). We define the perturbation (input) vector as

$$\mathbf{u}(t) = \begin{pmatrix} x_{X_1}(t) & c_1(t) & x_{X_2}(t) & c_2(t) \end{pmatrix}^T \tag{8.51}$$

That is, we would like to determine the sensitivity of the system to a small concentration perturbation $\tilde{\mathbf{u}}(t)$, around the fixed point $\hat{\mathbf{u}} = \mathbf{0}$. The linearized system matrix, **A**, is given by the Jacobian in (8.40), and the linearized input matrix, **B**, is

$$\begin{aligned}\mathbf{B} = \mathbf{S}\,\frac{\partial \mathbf{a}}{\partial \mathbf{p}}\bigg|_{\substack{\mathbf{x}=\hat{\mathbf{x}}\\ \mathbf{u}=\hat{\mathbf{u}}}} &= \begin{pmatrix} -1 & +1 \\ +1 & -1 \end{pmatrix} \begin{pmatrix} c_1 & \hat{x}_1 & 0 & 0 \\ 0 & 0 & c_2 & \hat{x}_2 \end{pmatrix} \\ &= \begin{pmatrix} -c_1 & -\hat{x}_{X_1} & +c_2 & +\hat{x}_{X_2} \\ +c_1 & +\hat{x}_{X_1} & -c_2 & -\hat{x}_{X_2} \end{pmatrix}\end{aligned} \tag{8.52}$$

According to (8.49), the frequency response matrix is

$$\mathbf{H}(s) = \left(s\mathbf{I}_{|\mathcal{S}|} - \mathbf{A}\right)^{-1}\mathbf{B} = \frac{1}{s + c_1 + c_2} \begin{pmatrix} -c_1 & -\hat{x}_{X_1} & +c_2 & +\hat{x}_{X_2} \\ +c_1 & +\hat{x}_{X_1} & -c_2 & -\hat{x}_{X_2} \end{pmatrix} \tag{8.53}$$

We obtain the impulse response in the time domain by applying the inverse Laplace transform to the frequency response matrix:

$$\begin{aligned}\mathbf{y}(t) = \mathcal{L}^{-1}\{\mathbf{H}\} &= \begin{pmatrix} \frac{\partial \tilde{x}_1}{\partial \mathbf{u}} \\ \frac{\partial \tilde{x}_2}{\partial \mathbf{u}} \end{pmatrix} \\ &= \begin{pmatrix} -c_1 & -\hat{x}_{X_1} & +c_2 & +\hat{x}_{X_2} \\ +c_1 & +\hat{x}_{X_1} & -c_2 & -\hat{x}_{X_2} \end{pmatrix} e^{-(c_1+c_2)t}\end{aligned} \tag{8.54}$$

Figure 8.9 shows the impulse response of $\tilde{x}_1$, the offset of the number of X_1-molecules from the fixed point, when perturbed by a unit impulse along the four defined perturbation channels. This figure illustrates that the system returns to equilibrium when the perturbations are gone. Because the number of control molecules is small, perturbations of x_{C_1} or x_{C_2} have a large effect on the concentrations of the X-molecules.

The *Disperser* protocol can be made more robust to changes of control molecules by increasing their quantity equally in all nodes. This example shows that MCA gives useful insights about the sensitivity of CNPS to external and internal changes.

MESOSCOPIC DYNAMICAL MODEL — RE-AUGMENTING THE INTRINSIC NOISE 8.3

This section reviews existing methods to characterize the stochastic fluctuations in chemical reaction networks. In Section 8.1 we introduced the exact

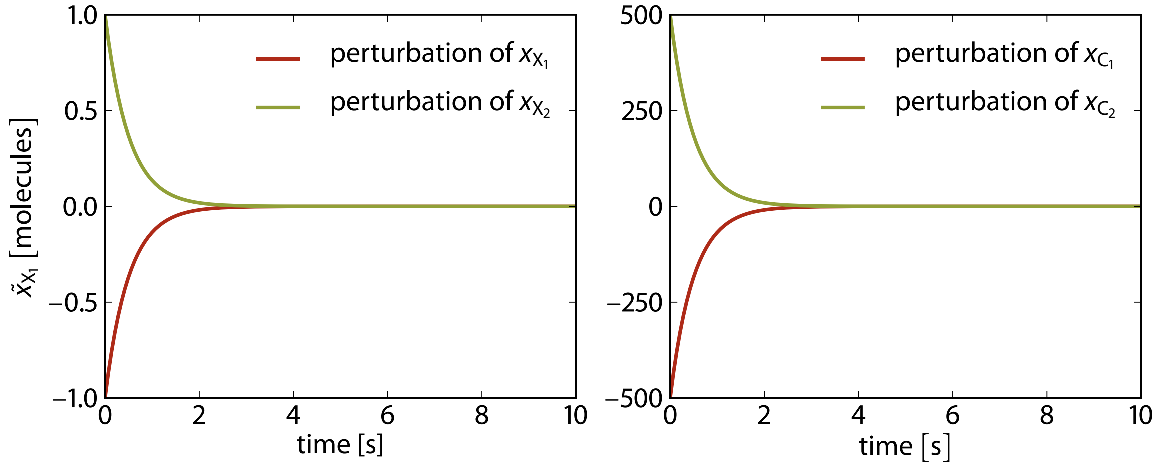


Figure 8.9 Perturbation impulse response: The quantities of species X_1, X_2 as well as C_1, and C_2 are perturbed with a unit impulse; both graphs show the response of $\tilde{x}_1$, the offset of the number of X_1-molecules from its steady-state quantity. The total number of X-molecules is $x_T = 1000$, i.e. the fixed point is at $x_{X_1} = x_{X_2} =$ 500 molecules.

mathematical description of CNPs, scheduled by a stochastic reaction algorithm. In the previous section, we then introduced the deterministic ODE approximation to describe the average trajectories of CNPs. An alternative view is to regard the average trajectory as the *systemic* part of the "signal" that is "polluted" by noise. The term "noise" is actually not really appropriate in this context (Ullah & Wolkenhauer, 2009b): The fluctuations are not necessarily unwanted data without meaning and are not imposed by an external source but stem from the inherent randomness of molecular collisions on the microscopic level.

systemic trajectory

Recently, researchers came up with several approaches to separate the mathematical description of a stochastic process into a systemic part, and noise. The ODE approximation, which is easy to obtain, can be used as systemic trajectory whereas the noise is characterized and *re-augmented* to this deterministic signal.

augmented noise

In this section, we discuss three of those mesoscopic mathematical descriptions for chemical reaction networks. Section 8.3.1 quickly mentions the Chemical Langevin Equation (CLE), which, despite its pervasiveness in natural science, points out to be too complex to apply to CNPs. Instead, we propose to use the Linear Noise Approximation (LNA), which we review in Section 8.3.2. Finally, Section 8.3.3 mentions the Two Moment Approximation (2MA), an enhancement of the LNA.

THE CHEMICAL LANGEVIN EQUATION 8.3.1

In natural science, instead of using the cumbersome stochastic Chemical Master Equation (CME), a stochastic process is often described by a system of *Langevin equations* — a set of stochastic differential equations that separates the change of any state in a systemic and a noisy part.

Langevin equation

$$\dot{x}_i = \overbrace{f_i(\mathbf{x})}^{\text{systemic}} + \overbrace{\sum_{j=1}^{m} g_{i,j}(\mathbf{x})\,\eta_j(t)}^{\text{noise}} \tag{8.55}$$

f_i and g_i are deterministic functions whereas the η_js are random functions in time with zero mean. The system is much easier to analyze if all g_is are constants; in this case the system is subject to *additive noise*. Otherwise, if the amplitude of the noise depends on the actual state of the system, the noise is said to be *multiplicative* and a mathematical treatment is more intricate.

additive noise

multiplicative noise

Like the deterministic ODE approximation, the Langevin equation operates on continuous variables and is therefore an approximate model for the discrete valued CME. Gillespie (2000) provided a derivation of the Langevin equation for the stochastic chemical jump type Markov process by translating the CME (8.5) to a *Chemical Langevin Equation* (CLE)

Chemical Langevin Equation

$$\dot{x}_s = \overbrace{\sum_{r\in\mathcal{R}} \gamma_r a_r(\mathbf{x})}^{\text{systemic}} + \overbrace{\sum_{r\in\mathcal{R}} \gamma_r \sqrt{a_r(\mathbf{x})}\,\Gamma_r}^{\text{noise}} \qquad \forall s \in \mathcal{S} \tag{8.56}$$

This equation system describes the change of the approximated continuous quantity x_s of species $s \in \mathcal{S}$ in terms of the stoichiometric vector γ_r and the reaction rates $a_r(\mathbf{x})$. The term $\Gamma_r \sim \mathcal{N}(0,1)$ is temporally uncorrelated Gaussian white noise, which is also statistically uncorrelated for each reaction. Hence, the CLE is a stochastic differential equation with multiplicative Gaussian white noise.

To arrive at this solution, Gillespie (2000) assumed that the quantities are large. In fact, the CLE is precise only if the following two conditions hold: First, within a small time interval $[t, t + \mathrm{d}t)$ the state change has to be small such that the propensity function does not change much: $a_r\big(\mathbf{x}(t)\big) \approx a_r\big(\mathbf{x}(x + \mathrm{d}t)\big)$. Second, within the same time interval each reaction must be executed more than once. Whereas the first condition defines a lower limit on dt, the second defines an upper limit. Both conditions usually hold if the number of involved molecules is large.

An additional approximation was done by characterizing the noise as white noise. In reality, the noise exhibits small and often negligible non-

Gaussian parts (Zwanzig, 2001). The noise is of the simpler additive type only for a limited subset of very simple unimolecular reactions operating on a finite state-space (Hanggi & Shuler, 1981), as for example the *Disperser* protocol (see the example in Section 8.1). The Langevin equation can therefore only be taken as a first approximation of the expected noise a CNP exhibits.

(a) Application to Chemical Networking Protocol Analysis

Although writing down the CLE for chemical reaction networks is straightforward, solving the system of stochastic differential equations is not. Thus CLEs are of limited use for the analysis of CNPs.

The CLE for the *Disperser* protocol between two nodes (see Figure 8.2) for $c_1 = x_{C_1}$ and $c_2 = x_{C_1}$ is

$$\dot{x}_{X_1} = -c_1 x_{X_1} + c_2 x_{X_2} - \sqrt{c_1 x_{X_1}} \cdot \Gamma_1 + \sqrt{c_2 x_{X_2}} \cdot \Gamma_2 \tag{8.57a}$$

$$\dot{x}_{X_2} = +c_1 x_{X_1} - c_2 x_{X_2} + \sqrt{c_1 x_{X_1}} \cdot \Gamma_1 - \sqrt{c_2 x_{X_2}} \cdot \Gamma_2 \tag{8.57b}$$

The usual method to study a system of CLEs is to use numerical methods to find sample trajectories for the state variables. This however does not provide us with more insights compared to executing the CNP in a stochastic simulation. In theory, it is possible to solve stochastic differential equations using Itô calculus (Øksendal, 2003), but this is only feasible for very simple systems. Furthermore, there is no straightforward way to quantify the variance (noise) of a certain variable. This is because the CLEs are interdependent. For example, for our two node *Disperser* example, we know from (8.17) that the variance of the quantities is $\text{Var}\left[N_{X_1}\right] = \text{Var}\left[N_{X_2}\right] = x_T \frac{c_1 c_2}{(c_1+c_2)}$, which we cannot directly deduce from the two equations above. Fortunately, there are other methods to describe the noise of chemical reaction systems; we will review two of them below.

8.3.2 LINEAR NOISE APPROXIMATION

Like the Chemical Langevin Equation (CLE), the Linear Noise Approximation (LNA) separates the description of the chemical reaction behavior into two distinct parts: the systemic trajectory is given by the macroscopic reaction rate equation (8.32) whereas the intrinsic noise is modeled as a Fokker-Planck equation, which is analytically tractable. Hence, the LNA provides a first order approximation of the system's dynamics described by the Chemical Master Equation (8.5). For linear reaction networks (first order reactions) it even gives exact solutions (Bruggeman, Blüthgen, & Westerhoff, 2009).

LNA has been introduced by van Kampen (1976) (see also van Kampen, 2007, Chap. 10) and was further studied by Elf and Ehrenberg (2003), Elf (2004), Tomioka, Kimura, Kobayashi, and Aihara (2004). LNA has been successfully applied to the study of enzymatic reaction networks (Hayot & Jayaprakash, 2004) and gene regulatory networks (GRN) (J. Paulsson, 2004; Scott, Ingalls, & Kærn, 2006).

Derivation from the Chemical Master Equation *(a)*

LNA is based on simplifications of the Chemical Master Equation (CME) and makes use of the observation that in real chemical reaction vessels, stochastic fluctuations become smaller for large system volumes. The macroscopic *concentration* of molecules $s \in \mathcal{S}$ is expressed as the average number of molecules per volume Ω:

concentration

$$x_s(t) = \frac{\mathbb{E}\left[N_s(t)\right]}{\Omega} \tag{8.58}$$

Next, similar than for the CLE, the random variable that describes the quantity of a molecule s, $N_s(t)$, is separated into a continuous, deterministic concentration variable $x_s(t)$, which captures the systemic change, and a new random variable $\eta(t)$ that describes the microscopic fluctuations. Because we expect the fluctuations to be of order $\sqrt{\Omega}$ the ansatz

$$N_s = \Omega x_s + \sqrt{\Omega}\eta_s \tag{8.59}$$

is used; the conditional probability terms in the CME are expressed with respect to the new variables.

Then, all terms of the CME are Taylor-expanded near the macroscopic concentration $\mathbf{x}(t)$ using the Ω-expansion method (van Kampen, 1976, 2007) (see also Elf & Ehrenberg, 2003, sup. mat.). Ω-expansion expands the virtual volume of the reaction vessel Ω in powers of $1/\Omega$. For example, the mesoscopic reaction rate of reaction r, i.e. the number of molecules processed per volume is then given by

$$a_r\left(\mathbf{x} + \frac{1}{\sqrt{\Omega}}\boldsymbol{\eta}\right) = a_r(\mathbf{x}) + \frac{1}{\sqrt{\Omega}}\sum_{s\in\mathcal{S}}\left(\frac{\partial a_r(\mathbf{x})}{\partial x_s}\eta_s\right) + \mathcal{O}\left(\frac{1}{\Omega}\right) \tag{8.60}$$

The higher order terms $\mathcal{O}(1/\Omega)$ are dropped. Collecting terms of order $\sqrt{\Omega}$ yields the macroscopic rate equation (8.28)

$$\frac{\partial x_s(t)}{\partial t} = \sum_{r\in\mathcal{R}} \gamma_{s,r} a_r\big(\mathbf{x}(t)\big) \qquad \forall i \in \mathcal{S} \tag{8.61}$$

describing the systemic trajectory of the reaction system according to the law of mass action, whereas collecting terms of order Ω gives a Fokker-Planck equation for the noise components only — the *Linear Noise Approximation* (LNA) (Elf & Ehrenberg, 2003; Wallace, 2010):

Linear Noise Approximation

$$\frac{\partial \mathbb{P}(\boldsymbol{\eta},t)}{\partial t} = \overbrace{-\sum_{i,j\in\mathcal{S}}\left(\mathbf{J}_{i,j}(\mathbf{x})\,\frac{\partial \eta_k \mathbb{P}(\boldsymbol{\eta},t)}{\partial \eta_i}\right)}^{\text{drift}} \underbrace{+\frac{1}{2}\sum_{i,j\in\mathcal{S}}\left(\mathbf{D}_{i,j}(\mathbf{x})\,\frac{\partial^2 \mathbb{P}(\boldsymbol{\eta},t)}{\partial \eta_i \partial \eta_k}\right)}_{\text{diffusion}} \tag{8.62}$$

Fokker-Planck equation

drift

Jacobian matrix

diffusion matrix

This *Fokker-Planck equation* is a continuous approximation of the probability distribution of the noise expressed in a drift and a diffusion term. The *drift* term characterizes the force of the process to return to its mean. It is no surprise that this force is given in terms of the *Jacobian matrix* **J**, which expresses the speed of the system to return to the fixed point when perturbed – for each molecule concentration separately and linearized around the fixed point. The diffusion term is given in terms of the *diffusion matrix*[*].

$$\mathbf{D}(\mathbf{x}) = \mathbf{S}\,\mathrm{diag}\big(\mathbf{a}(\mathbf{x})\big)\,\mathbf{S}^T \tag{8.63}$$

The diffusion matrix characterizes the "smearing" force, i.e. the tendency of the system to deviate from the mean.

It is well known that the stationary solution of any Fokker-Planck equation is a multivariate Gaussian distribution. Thus, the stationary solution for (8.62) is

$$\mathbb{P}(\boldsymbol{\eta},t) = \frac{\exp\left(-\frac{\boldsymbol{\eta}^T\,\Xi(\mathbf{x})\,\boldsymbol{\eta}}{2}\right)}{\sqrt{(2\pi)^{|\mathcal{S}|}\cdot\det\Xi(\mathbf{x})}} \tag{8.64}$$

with zero mean and a covariance matrix $\Xi(\mathbf{x}) = \mathbb{E}\left[\boldsymbol{\eta}\boldsymbol{\eta}^T\right]$ to find. That is, all molecular fluctuations (and all linear combinations thereof) are approximated by Gaussian distributions.

[*] The diffusion matrix is also called dissipation matrix, from the fluctuation-dissipation theorem in statistical thermodynamics (Kaizer, 1987).

In (8.59) we introduced the ansatz $N_s = \Omega x_s + \sqrt{\Omega}\eta_s$. Hence, the unknown covariance matrix of the noise $\boldsymbol{\eta}$ is identical to the covariance matrix of the random variables $\mathbf{N}$ of molecular quantities (the volume Ω cancels out). The covariance matrix contains the variances (diagonal elements) and covariances of the molecular quantities with respect to each other, e.g. the variance of the molecular quantity X_i is

$$\mathrm{Var}(N_i) = \left[\Xi(\mathbf{x})\right]_{i,i} \tag{8.65}$$

Recipe to Calculate the Covariance Matrix *(b)*

Elf and Ehrenberg (2003) provided the following simple recipe of how to find the *covariance matrix* $\hat{\Xi}$, which describes the noise around the fixed point.

covariance matrix

1. Provide the macroscopic reaction rate equation for the reaction system under study (according to Section 8.2):

$$\dot{\mathbf{x}} = \mathbf{Sa}(\mathbf{x})\,; \tag{8.66}$$

2. Find the fixed point $\hat{\mathbf{x}}$ by solving $\mathbf{Sa}(\hat{\mathbf{x}}) = \mathbf{0}$ (see Section 8.2.1);

3. Calculate the *Jacobian matrix*, evaluated at the fixed point: $\hat{\mathbf{J}} = \mathbf{J}(\mathbf{x})|_{\mathbf{x}=\hat{\mathbf{x}}}$ (see Section 8.2.2);

Jacobian matrix

4. Calculate the *diffusion matrix*, also evaluated at the fixed point: $\hat{\mathbf{D}} = \mathbf{D}(\hat{\mathbf{x}})$, according to (8.63);

diffusion matrix

5. Find the steady-state covariance matrix $\hat{\Xi}$ by solving the following *Lyapunov matrix equation*

Lyapunov matrix equation

$$\hat{\mathbf{J}}\hat{\Xi} + \hat{\Xi}\hat{\mathbf{J}}^T + \hat{\mathbf{D}} = \mathbf{0}, \tag{8.67}$$

meaning that at equilibrium, the relaxation towards the fixed point must be equal to the diffusion due to the randomness of the intrinsic noise.

Note that the LNA prescribes a multivariate Gaussian distribution for the probability density function of the molecular quantities at steady state. That is, it provides an exact description of the noise for unimolecular reaction networks, and an accurate approximation for higher-order reaction networks at equilibrium.

(c) *Example: Linear Noise Approximation Applied to the* Disperser *Protocol Between Two Nodes*

The *Disperser* protocol consists of unimolecular reactions only. Thus, the LNA should provide exact values for the variance of the molecular quantities. Figure 8.2 on page 108 shows the reaction network under consideration for which we apply the LNA recipe step by step:

1. The ODEs of the two-node *Disperser* protocol have already been provided in (8.38). The LNA only works if the ODEs are linearly independent. Since the two reactions are dependent, we make use of the fact that the X-molecules are conserved. We substitute $x_{X_2} = x_T - x_{X_1}$ and obtain the single ODE

$$\overbrace{\dot{x}_{X_1}}^{\dot{\mathbf{x}}} = \overbrace{\begin{pmatrix} -1 & 1 \end{pmatrix}}^{\mathbf{S}} \overbrace{\begin{pmatrix} c_1 x_{X_1} \\ c_2 \left(x_T - x_{X_1} \right) \end{pmatrix}}^{\mathbf{a}(\mathbf{x})} \tag{8.68}$$

2. The fixed point is obtained by solving $\mathbf{Sa}(\hat{\mathbf{x}}) = \mathbf{0}$ (see (8.39) on page 118):

$$\hat{x}_{X_1} = \frac{c_2}{c_1 + c_2} x_T \tag{8.69}$$

3. The Jacobian matrix for (8.68), evaluated at the fixed point is

$$\hat{\mathbf{J}} = \mathbf{J}(\mathbf{x})|_{\mathbf{x}=\hat{\mathbf{x}}} = \mathbf{S} \, \frac{\partial \mathbf{a}}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\hat{\mathbf{x}}} = -(c_1 + c_2) \tag{8.70}$$

4. The diffusion matrix, evaluated at the fixed point is

$$\begin{aligned} \hat{\mathbf{D}} &= \mathbf{S}\,\mathrm{diag}\big(\mathbf{a}(\hat{\mathbf{x}})\big)\,\mathbf{S}^T \\ &= \begin{pmatrix} -1 & 1 \end{pmatrix} \begin{pmatrix} \frac{c_1 c_2}{c_1+c_2} x_T & 0 \\ 0 & \frac{c_1 c_2}{c_1+c_2} x_T \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} \\ &= 2 \frac{c_1 c_2}{c_1 + c_2} x_T \end{aligned} \tag{8.71}$$

5. Finally, we obtain the covariance matrix, $\hat{\Xi} = \hat{\xi}_{1,1}$, by solving the Lyapunov equation

$$\begin{aligned} \mathbf{0} &= \hat{\mathbf{J}}\hat{\Xi} + \hat{\Xi}\hat{\mathbf{J}}^T + \hat{\mathbf{D}} \\ &= \big(-\left(c_1 + c_2\right)\big)\,\hat{\xi}_{1,1} + \hat{\xi}_{1,1}\,\big(-\left(c_1 + c_2\right)\big) + 2\frac{c_1 c_2}{c_1 + c_2} x_T \end{aligned} \tag{8.72}$$

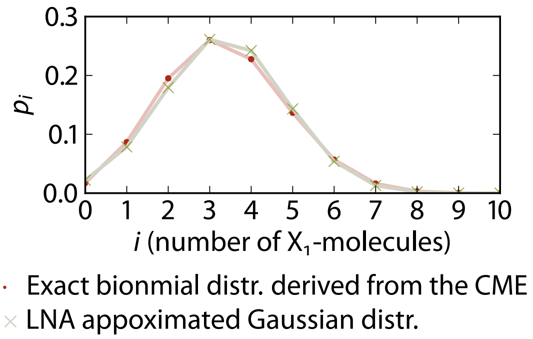


Figure 8.10 Effective/Approximated probability distribution of *Disperser*: The LNA approximates the binomial distribution with a Gaussian distribution. This approximation is more accurate for large number of molecules and when the reaction coefficients c_1 and c_2 are similar. Here: $c_1 = 2$, $c_2 = 1$, $x_T = 10$

Hence, the variance of the number of species X_1 at equilibrium is

$$\operatorname{Var}\left(\hat{N}_{X_1}\right) = \hat{\xi}_{1,1} = \frac{c_1 c_2}{(c_1 + c_2)^2} x_T \tag{8.73}$$

In this example, which consists of first order reactions only, the LNA is able to produce the same expression for the variance as the exact calculation from the CME (compare (8.73) on page 131 to (8.17) on page 109). However, we have to keep in mind that the LNA approximates the probability distribution as a continuous Gaussian distribution, which differs from the actual binomial distribution. Figure 8.10 shows a comparison of the two distributions for $c_1 = 2$, $c_2 = 1$, and a very small total number of molecules of $x_T = 10$. The approximation becomes better for larger quantities, or when $c_1 \approx c_2$.

TWO MOMENT APPROXIMATION 8.3.3

The Coupled Mean-Variance Approximation (Gómez-Uribe & Verghese, 2007) or Two Moment Approximation (2MA) (Ullah & Wolkenhauer, 2009a, 2009b) is an extension of the Linear Noise Approximation (LNA). Like the other mesoscopic descriptions, the 2MA separates the intrinsic noise from the systemic trajectories of the system. Additionally, the 2MA models the fact that the *noise actually influences the systemic trajectory*. Because we are not applying this method to CNPs, we just briefly sketch the ideas behind the 2MA and leave the reader to the referred articles for further information.

noise influences systemic trajectory

Unlike the LNA, which focuses on determining the covariances of the molecular quantities around the steady state, the 2MA describes the evolution of the mean $\mathbf{x}(t)$ *and* the covariance matrix $\Xi(t) = \left[\xi_{i,j}(t)\right]$ over time by separate differential equations. They can be derived from the Chemical Master Equation (CME) when third and higher order moments are ignored. (see Ullah and Wolkenhauer (2009b, Appx. A1) for further details and proofs). The ODE for the effective molecular concentration is

$$\frac{\partial \mathbf{x}(t)}{\partial t} = \overbrace{\mathbf{Sa}(\mathbf{x})}^{\text{deterministic flux}} \quad \overbrace{+\frac{1}{2}\mathbf{S}\sum_{i,j\in\mathcal{S}}\left(\frac{\partial^2 \mathbf{a}(\mathbf{x})}{\partial x_i \partial x_j}\,\xi_{i,j}(\mathbf{x})\right)}^{\text{stochastic flux}} \tag{8.74}$$

whereas the ODE for the covariance matrix is

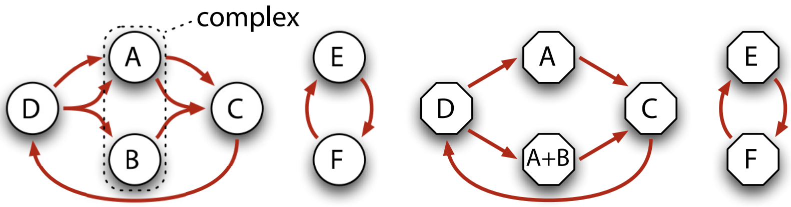
$$\frac{\partial \Xi(\mathbf{x})}{\partial t} = \mathbf{J}(\mathbf{x})\,\Xi(\mathbf{x}) + \Xi(\mathbf{x})\,\mathbf{J}^T(t) + \mathbf{D}(\mathbf{x}) + \frac{1}{2}\sum_{i,j\in\mathcal{S}}\left(\frac{\partial^2 \mathbf{D}(\mathbf{x})}{\partial x_i \partial x_j}\,\xi_{i,j}(\mathbf{x})\right) \tag{8.75}$$

From those $|\mathcal{S}|\left(|\mathcal{S}|+3\right)/2$ ODEs, the effective trajectories of molecular concentrations as well as the corresponding covariances can be calculated, for example by numerical integration.

Recall from Section 8.2.1 how the deterministic reaction rate equation (8.32) can be derived from the stochastic CME (8.5). There, we assumed that the expected value of the propensity function is equal to the propensity function taken at the mean quantity vector, $\mathbb{E}\left[\mathbf{a}(\mathbf{N})\right] = \mathbf{a}\big(\mathbb{E}\left[\mathbf{N}\right]\big) = \mathbf{a}(\mathbf{x})$, which is only correct for first order reactions. After introducing the mesoscopic models, we recognize that for second- and higher-order reactions, we have to consider the second- or higher-order moments of the stochastic process, because the propensity function combines dependent random variables of molecular quantities in a non-linear fashion. The 2MA considers the second moment (covariances) but ignores third and higher order moments of the probability distributions. Hence, the 2MA provides an exact solution for first- and second-order reactions and an accurate approximation for third- and higher-order reactions.

8.4 EXISTENCE THEOREMS FOR EQUILIBRIA

In this section, we give a short introduction to two existing theorems that infer from the structure of a chemical reaction network on the existence of fixed points. Such generic theorems provide us with information about the dynamic behavior of a CNP without forcing us to give a microscopic or macroscopic description of the dynamic behavior and consequently, without solving ODEs or the CME. The two theorems we review in this section are the Deficiency Zero Theorem in Section 8.4.1 and the Chemical Organization Theory in Section 8.4.2.



(a) Reaction network graph

(b) Graph of complexes

Figure 8.11 Reaction network among species complexes: The Deficiency Zero Theorem defines (a) a reaction network as (b) a graph of complexes. A complex is a reactant or product multiset of species that appears in the reaction network.

Deficiency Zero Theorem 8.4.1

The Deficiency Zero Theorem (Feinberg, 1972; Horn & Jackson, 1972; Horn, 1973a, 1973b, 1973c) provides a general result about the stability of a certain class of chemical reaction networks. The theorem is particularly interesting because it allows for deciding whether or not a reaction network is stable, solely based on its structure. A dynamical analysis is not required as long as the reaction rates obey the law of mass action. The Deficiency Zero Theorem states that a weakly reversible reaction network with a deficiency value of zero has exactly one fixed point, which is asymptotically stable.

In the following, we briefly introduce the terms "weakly reversible" and "deficiency" and illustrate the theorem on a simple example, depicted in Figure 8.11(a). Feinberg and Horn introduced a new reaction network graph, the directed graph of complexes. *Complexes* are those multiset of species that appear on the left- and the right-hand side of a reaction. In our example network, the molecules A and B appear together on the left- and right-hand side of the reactions $A + B \rightarrow C$ and $D \rightarrow A + B$, respectively. Thus, the multiset $\{A, B\}$ is a complex as well as $\{A\}$, which appears alone in the reactant/product multiset of other reactions. The graph of complexes for our simple example is shown in Figure 8.11(b).

complex

A chemical reaction network is *weakly reversible* if for every reaction leading from complex C_i to complex C_j, there is also a chain of reactions leading from C_j back to C_i. Our example reaction network is weakly reversible, because there is no complex that is produced but not consumed. We also have to define the term *linkage class*, which denotes a connected subgraph in the graph of complexes. Our example consists of two linkage classes.

weakly reversible reaction network

linkage class

The *deficiency* δ of a chemical reaction network is a positive integer, defined as

deficiency

$$\delta = |\mathcal{C}| - l - \operatorname{rank}(\mathbf{S}) \tag{8.76}$$

where $|\mathcal{C}|$ is the number of components, l is the number of linkage classes, and $\operatorname{rank}(\mathbf{S})$ is the *rank of the stoichiometric matrix*. Our example has a deficiency of

rank of the stoichiometric matrix

$$\delta_{\text{Figure 8.11}} = \overbrace{6}^{|\mathcal{C}|} - \overbrace{2}^{l} - \overbrace{4}^{\text{rank}(\mathbf{S})} = 0 \qquad (8.77)$$

The Deficiency Zero Theorem states that because our example reaction network is weakly reversible and its deficiency is zero, the reaction network has a single fixed point, which is asymptotically stable.

Note that the theorem only provides stability proofs for a certain class of chemical reaction networks. If the network is neither weakly reversible nor is its deficiency zero, we are not able to draw any conclusions about its stability. Thus, the applicability of the theorem to the analysis of CNPs depends on the concrete case.

A good introduction to the dynamics of chemical reaction networks in general and a proof of the Deficiency Zero Theorem can be found in Feinberg's lecture notes (1979). There is an extension of the theorem to stochastic processes (Anderson, Craciun, & Kurtz, 2010), which is used by Mairesse and Nguyen (2009) to show that a reaction network of deficiency zero, when translated to a stochastic Petri Net model, results in a stationary probability distribution (equilibrium) which is in product form (F. P. Kelly, 1979).

8.4.2 CHEMICAL ORGANIZATION THEORY

The Chemical Organization Theory (COT) also provides general statements about the existence of fixed points. But it goes much further than the Deficiency Zero Theorem: The Chemical Organization Theory was developed by Dittrich and Speroni di Fenizio (2007) as a continuation of the work by Fontana and Buss (1994) on constructive reaction systems. In such systems, new types of species and reactions are dynamically added or removed. The original aim of the Chemical Organization Theory was to capture the dynamic behavior of constructive reaction systems by relating fixed points to the changing structure of the reaction network. We will cover such volatile reaction network structures in more detail in Part III; in this section, we highlight that the COT – by only analyzing the structure of the reaction network – is able to predict the existence of dynamic fixed points.

One of the main results of the COT is that every dynamical fixed point is an instance of an organization. Organizations are subsets of the chemical universe, i.e. subsets of $\wp(\mathcal{S})$. An *organization* is a closed and self-maintaining set of molecules. Dittrich proved that all dynamic fixed points of the reaction network must be instances of such organizations. He further showed that the organizations form a lattice hierarchy. Due to stochastic fluctuations, the

organization

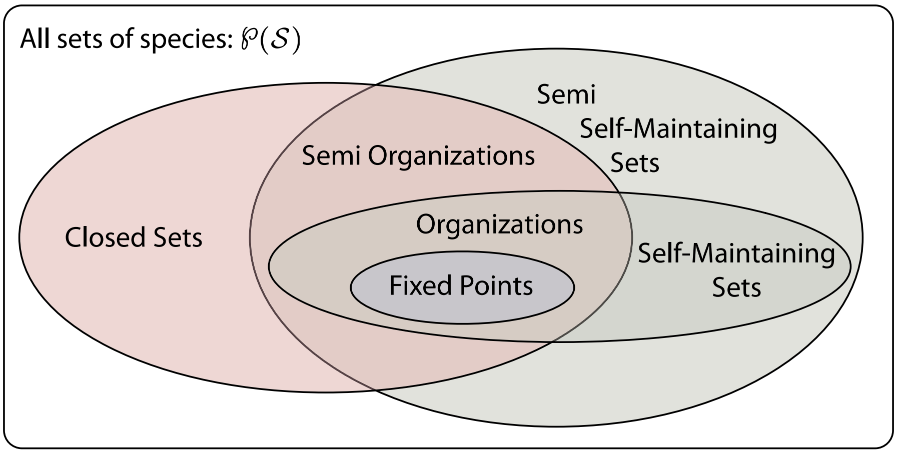


Figure 8.12 Venn diagram of the Chemical Organization Theory: Every fixed point is an instance of an organization.

system may completely lose some molecular species, which corresponds to a down-movement in the lattice of organizations whereas the appearance of new, before unknown species, may result in an up-movement.

To find the organizations and other important power sets over the set of molecular species (see Figure 8.12) the reaction network is analyzed structurally first, without considering its dynamics. The structure of the reaction network can be mapped to a set of *semi-organizations*. Semi-organizations are those sets of molecules that are closed and semi self-maintaining. We already introduced the concept of closure in Section 7.2 to determine the sequence space of Fraglets programs. Generally, a set of molecular species is *closed* if no new species can be generated by reactions among any multiset of this set. Formally, a set $\mathcal{X} \subseteq \mathcal{S}$ is closed iff

semi-organization

closed

$$\operatorname{img} r \subseteq \mathcal{X} \qquad \forall \left\{ r \in \mathcal{R} \mid \operatorname{dom} r \subseteq \mathcal{X} \right\} \tag{8.78}$$

A set $\mathcal{X} \subseteq \mathcal{S}$ is *semi self-maintaining* if every molecule that is consumed within this set is also produced within this set. A molecule $s \in \mathcal{X}$ is consumed within the set $\mathcal{X}$ iff $\exists \left\{ r \in \mathcal{R} \mid \operatorname{dom} r \subseteq \mathcal{X} \wedge \alpha_{s,r} \geq 0 \right\}$; it is produced within the set $\mathcal{X}$ iff $\exists \left\{ r \in \mathcal{R} \mid \operatorname{dom} r \subseteq \mathcal{X} \wedge \beta_{s,r} \geq 0 \right\}$.

semi self-maintaining

All sets $\mathcal{X}$ that are both closed and semi self-maintaining are called semi-organizations. Being a semi-organization is a necessary requirement for being an organization, which is in turn required to host a fixed point.

A set of species is an *organization* if the set is closed and self-maintaining.

self-maintaining

A set of molecule $\mathcal{X} \subseteq \mathcal{S}$ is *self-maintaining* if there exists a reaction rate vector $\mathbf{a}\big(\mathbf{N}(t)\big)$ such that the following conditions apply:

$$a_r\big(\mathbf{N}(t)\big) > 0 \qquad \forall \left\{r \in \mathcal{R} \mid \operatorname{dom} X \subseteq \mathcal{S}\right\} \tag{8.79a}$$

$$a_r\big(\mathbf{N}(t)\big) = 0 \qquad \forall \left\{r \in \mathcal{R} \mid \operatorname{dom} X \nsubseteq \mathcal{S}\right\} \tag{8.79b}$$

$$\dot{x}_i = \Big[\mathbf{Sa}\big(\mathbf{N}(t)\big)\Big]_i \geq 0 \qquad \forall s_i \in \{X\} \tag{8.79c}$$

Note that none of the conditions require that the reactions obey the law of mass action; the reaction rate vector $\mathbf{a}\big(\mathbf{N}(t)\big)$ can be any function, not necessarily dependent on the number of reactant species. This is why not all organizations contain stable fixed points. An organization is a more general precondition for a sustainable dynamical system independent on its concrete dynamical realization.

We will make use of the COT in Part III where it is particularly helpful to analyze mutations of Fraglets strings, leading to novel species that were not present in the system before.

8.4.3 DISCUSSION

Both the Deficiency Zero Theorem as well as the Chemical Organization Theory make predictions on the presence of dynamic fixed points by merely analyzing the structure of the reaction network. They use a different approach and yield different results: The Deficiency Zero Theorem only provides a result for a certain class of reaction networks, those whose dynamic behavior follow the law of mass action and whose deficiency value is zero. If this is the case, the theorem asserts that a non-trivial stable fixed point exists; otherwise, the theorem does not provide any result. The Chemical Organization Theory, on the other hand, is able to analyze any reaction network but only gives us hints where the dynamic fixed points are located; they must be instances of organizations.

Both theorems have their application areas in the analysis of CNPS: If we are not interested in the location of the steady state, the Deficiency Zero Theorem may be used to quickly prove the convergence of a protocol. The Chemical Organization Theory is useful for finding the location of steady-states by transforming the sets of molecules forming a semi-organization into constraints and feeding them to a symbolic equation solver.

SUMMARY AND DISCUSSION 8.5

Most of the dynamical analysis carried out in this thesis makes use of deterministic approximations, because protocol proofs are getting easier and more understandable while still being valid for most reaction systems, especially for large number of molecules (Gillespie, 2000, 2002). However, for certain networks, such an approximation may hide certain quantitative effects that only appear in the detailed stochastic model (Ullah & Wolkenhauer, 2009b):

value-dependent signal-quality

First, the amount of stochastic fluctuations depends on the molecular quantities as demonstrated for the *Disperser* protocol by calculating the signal-to-noise ratio in the previous section. Second, in the stochastic model, molecules may go *extinct* when their quantity drops from one to zero whereas the deterministic approximation tracks the molecular quantities in continuous variables that can become arbitrarily small. Third, in some situations, the actual mean can be larger than the deterministic prediction, leading to an enhanced sensitivity of the network (Pedraza & van Oudenaarden, 2005), known as *stochastic focusing* (Paulsson, Berg, & Ehrenberg, 2000). Finally, if the reaction network has more than one stable steady state (bi-stability), either steady state may be reached regardless of the initial concentration; this is known as *stochastic switching*.

extinction

stochastic focusing

stochastic switching

Generally, the deterministic approximation only predicts the system correctly if the macroscopic rate equations have a globally stable solution, independent on the initial condition. In this case the stochastic trajectory converges to the deterministic path (Mansour, Van Den Broeck, Nicolis, & Turner, 1981).

Chemical networking protocols differ from traditional protocols in that they exploit the dynamic behavior of code interactions as an additional design dimension. Traditional protocol analysis methods, such as model checking, process calculi, or theorem proving, which have their strengths in proving the qualitative correctness of protocols, are not able to capture the quantitative aspect of CNPs. Recent developments, such as probabilistic model checking, come much closer to our requirements.

On the other hand, classical quantitative protocol analysis could potentially benefit from the mathematical tools developped for chemistry: So far, using probabilistic model checking, it has not been possible to precisely reason about statistical quantities beyond expectation values. For example, measures such as the variance have not made accessible (Hasan & Tahar, 2009). For CNPs, this is now possible by using the recently established method of mesoscopic dynamical analysis for chemical reaction networks using LNA, or 2MA, for example.

CHAPTER 9

Analysis of a Chemical Gossip Protocol

Application of the chemical analysis tools to the study of a distributed data aggregation protocol.

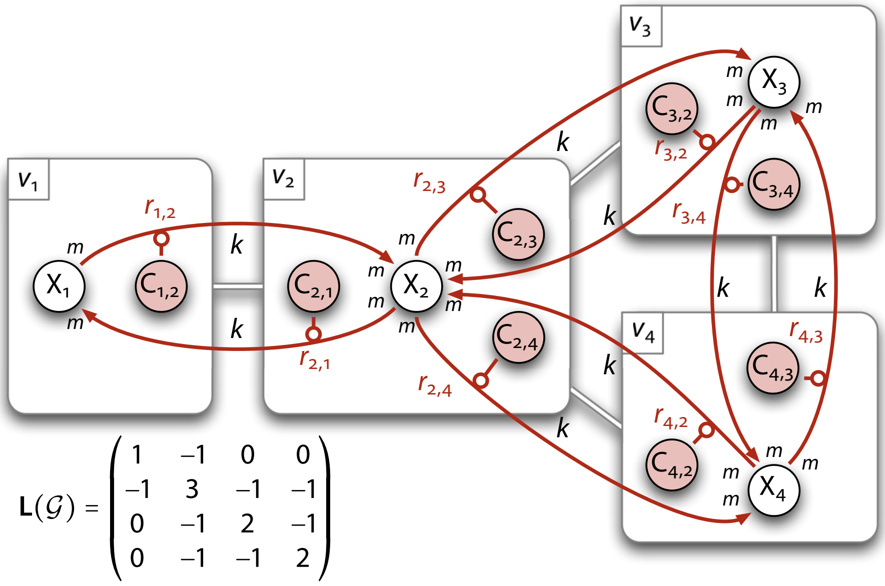
> Someone who gossips to you
> will gossip about you. [9]
>
> *English proverb*

The Disperser protocol, introduced as an application case in Section 5.4, is a chemical gossip aggregation protocol. It calculates an average value in a distributed way. Unlike in traditional gossip protocols, the calculation is not achieved by exchanging symbolic values, but by relying on the dynamic behavior of the spanned distributed reaction network such that at equilibrium, the result is presented to each node in the number of molecules.

In this chapter, we first model the dynamic behavior of the generalized *Disperser* protocol by deterministic ODEs (Section 9.1). Within this model, we prove that the system asymptotically converges to a stable fixed point by using linear stability analysis (Section 9.2). In Section 9.3 we apply the Linear Noise Approximation (LNA) to calculate the signal-to-noise ratio of the presented result before we discuss our findings in Section 9.4.

Abstract model and mathematical description 9.1

Formally, the distributed reaction network of *Disperser* in a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be represented as an abstract explicit artificial chemistry $(\mathcal{S}, \mathcal{R}, \mathcal{A})$. The set of species is given explicitly by the set



$$\mathbf{L}(\mathcal{G}) = \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}$$

Figure 9.1 Reaction network and graph Laplacian of *Disperser*: The protocol computes the average of X_i-molecules by letting the catalysts, $C_{i,j}$, send them to the corresponding neighbors. The Laplacian of the network graph describes the topology of the network of nodes.

$$\mathcal{S} = \left\{ X_i \mid i \in \mathcal{V} \right\} \cup \left\{ C_{i,j} \mid \left(i,j\right) \in \mathcal{E} \right\} \tag{9.1}$$

It consists of an X-species per node and a molecule of type C per node and link. In node v_i, the X_i-molecules are relocated to one of the neighbor nodes v_j by reacting with the corresponding catalyst $C_{i,j}$.

higher order reactions

In this section, we generalize the *Disperser* protocol first presented in Section 5.4: instead of sending only one single molecule, we use *higher order reactions* that send multiple molecules at once. The m molecules that are removed from the originating node are combined in a single "packet" to the destination node, where the m molecules are re-injected. This reduces the message complexity of the protocol. We also introduce a reaction coefficient, k, which has to be identical for all reactions. The corresponding distributed reaction network is explicitly described by the following m^{th}-order reactions, depicted for a concrete network topology of four nodes in Figure 9.1:*

$$\mathcal{R} = \left\{ r_{i,j} \mid \left(i,j\right) \in \mathcal{E} \right\} \tag{9.2a}$$

$$r_{i,j}\colon\ C_{i,j} + mX_i \xrightarrow{k} C_{i,j} + mX_j \tag{9.2b}$$

The algorithm of our abstract artificial chemistry, $\mathcal{A}$, is an implementation of an exact stochastic reaction algorithm.

We restrict the problem to the case where the graph $\mathcal{G}$ is fully *connected and symmetric*, meaning that network links are bidirectional. We still consider ideal links without delay, packet loss or bandwidth limitations. We also require

*The number of $C_{i,j}$-molecules remains constant, which is why the reaction order is m instead of $(m+1)$.

that each catalyst species $C_{i,j}$ is present with the same number of instances: $x_{C_{i,j}} = c$ for all links $(i,j) \in \mathcal{E}$.

FORMAL CONVERGENCE PROOF 9.2

We use the dynamic approximation, presented in Section 8.2, to prove that the *Disperser* protocol converges. We have to find the fixed point of the distributed reaction network and show that it is stable.

DETERMINING THE FIXED POINT 9.2.1

We first write down the differential equations that describe the time evolution of the number of X_i-molecules:

$$\dot{x}_{X_i} = \overbrace{\sum_{j \in \mathcal{V}} \operatorname{adj}(j,i)\, k x_{C_{j,i}} x_{X_j}^m}^{\text{inflow}} - \overbrace{\sum_{j \in \mathcal{V}} \operatorname{adj}(i,j)\, k x_{C_{i,j}} x_{X_i}^m}^{\text{outflow}} \qquad \forall i \in \mathcal{V} \qquad (9.3)$$

Then, in order to find the global fixed point, we set $\dot{x}_{X_i} = 0$. Because there are c instances of each catalyst, we substitute $x_{C_{i,j}} = c \cdot \operatorname{adj}(i,j)$, and because the adjacency function, $\operatorname{adj}(\cdot)$, only returns values 0 and 1, we use $\operatorname{adj}(i,j)^2 = \operatorname{adj}(i,j)$. Hence, from (9.3) we obtain the following equation, which captures the *balance of inflow and outflow* at equilibrium in each node.

flow balance

$$\overbrace{kc \sum_{j \in \mathcal{V}} \operatorname{adj}(j,i)\, \hat{x}_{X_j}^m}^{\text{inflow}} = \overbrace{kc \hat{x}_{X_i}^m \underbrace{\sum_{j \in \mathcal{V}} \operatorname{adj}(i,j)}_{\deg_{\text{out}}(i)}}^{\text{outflow}} \qquad \forall i \in \mathcal{V} \qquad (9.4)$$

Solving (9.4) with respect to $\hat{x}_{X_i}$ yields

$$\hat{x}_{X_i} = \sqrt[m]{\frac{\sum_{j \in \mathcal{V}} \operatorname{adj}(j,i)\, \hat{x}_{X_j}^m}{\deg_{\text{out}}(i)}} \qquad \forall i \in \mathcal{V} \qquad (9.5)$$

where $\deg_{\text{out}}(i) = \sum_{j \in \mathcal{V}} \operatorname{adj}(i,j)$ is the *out-degree* of node v_i, i.e. the number of links leaving v_i. This equation states that the number of X-molecules in node v_i at equilibrium, $\hat{x}_{X_i}$, is equal to the *power mean* with exponent m of the number of X-molecules in v_i's neighbors.

out-degree

power mean

By assuming that at equilibrium the X-molecules are evenly distributed among all nodes, we use the ansatz

$$\hat{x}_{\mathrm{X}} = \hat{x}_{\mathrm{X}_i} \qquad \forall i \in \mathcal{V} \tag{9.6}$$

in (9.5) and obtain

$$\hat{x}_{\mathrm{X}_i} = \sqrt[m]{\frac{\deg_{\mathrm{in}}(i)}{\deg_{\mathrm{out}}(i)}\hat{x}_{\mathrm{X}}^m} \qquad \forall i \in \mathcal{V} \tag{9.7}$$

We are allowed to replace $\deg_{\mathrm{in}}$ by $\deg_{\mathrm{out}}$, because we required that the network graph is symmetric. Our ansatz $\hat{x}_{\mathrm{X}_i} = \hat{x}_{\mathrm{X}}$ immediately follows from (9.7), which proves that the ansatz was correct.

Finally, by using the assumption that the network graph is fully connected, and by observing that the total number of X-molecules, x_{T}, is conserved by all reactions, formally described by

$$\sum_{i \in \mathcal{V}} x_{\mathrm{X_i}}(t) = x_{\mathrm{T}}, \tag{9.8}$$

we show, with the help of (9.6), that

$$\sum_{i \in \mathcal{V}} \hat{x}_{\mathrm{X_i}} = |\mathcal{V}|\,\hat{x}_{\mathrm{X}} = x_{\mathrm{T}} \tag{9.9}$$

and therefore

$$\hat{x}_{\mathrm{X}} = \hat{x}_{\mathrm{X}_i} = \frac{x_{\mathrm{T}}}{|\mathcal{V}|} \qquad \forall i \in \mathcal{V} \tag{9.10}$$

Consequently, at chemical equilibrium the X-molecules are equally distributed over the network. □

9.2.2 PROVING FIXED POINT STABILITY

linearize

Jacobian matrix

We now have to show that this fixed point is stable. For this purpose, we use a linear stability analysis to *linearize* the system around the fixed point and show that it returns to the fixed point after a small perturbation. We linearize the system by calculating the $|\mathcal{V}| \times |\mathcal{V}|$ *Jacobian matrix* of (9.3):

$$\mathbf{J}(\mathbf{x}) = \left[j_{i,j}(\mathbf{x})\right] = \left[\frac{\partial \dot{x}_{\mathrm{X}_i}}{\partial x_{\mathrm{X}_k}}\right] = -kcm^2\mathbf{L}(\mathcal{G})\begin{pmatrix} x_{\mathrm{X}_1}^{m-1} \\ \vdots \\ x_{\mathrm{X}_{|\mathcal{V}|}}^{m-1} \end{pmatrix} \tag{9.11}$$

Interestingly, the Jacobian is proportional to the negative *Laplacian* of the network graph. The Laplacian matrix is defined as $\mathbf{L} = \mathbf{\Delta}_{\text{out}} - \mathbf{A}$, where $\mathbf{\Delta}_{\text{out}} = \text{diag}\big(\text{deg}_{\text{out}}(i)\big)$ is the *out-degree matrix* of the graph and $[\mathbf{A}]_{i,j} = \text{adj}\big(i, j\big)$ is the *adjacency matrix* of the graph. We evaluate the Jacobian at the fixed point.

Laplacian

out-degree matrix

adjacency matrix

$$\hat{\mathbf{J}} = \mathbf{J}(\mathbf{x})\Big|_{\mathbf{x}=\hat{\mathbf{x}}} = -kcm^2\hat{x}_{\text{X}}^{m-1}\mathbf{L}(\mathcal{G}) \tag{9.12}$$

It is well known (Saerens, Fouss, Yen, & Dupont, 2004) that all eigenvalues of the Laplacian of any symmetric graph are positive and hence, because the factor $kcm^2\hat{x}_{\text{X}}^{m-1} > 0$, the eigenvalues of $\mathbf{J}$ are all negative and the fixed point is stable. □

SIGNAL-TO-NOISE RATIO 9.3

Disperser presents its result – the average number of X-molecules – to every node in the network. Obviously, this result "signal" is afflicted with the inherent noise resulting from the stochastic scheduling algorithm. In this section, we use the Linear Noise Approximation (LNA) to demonstrate that the Signal-to-Noise Ratio (SNR) of *Disperser* actually drops for increasing networks, but that the worst-case SNR is only marginally smaller than for the simplest two-node topology.

CONJECTURE OF A CLOSED-FORM VARIANCE EXPRESSION FOR THE DISPERSER PROTOCOL 9.3.1

In order to calculate the SNR, we need to obtain the mean and the variance of the number of X-molecules in each node. The mean is equal to the steady-state quantity at the fixed point; the variance is obtained by calculating the covariance matrix using the LNA. So far we did not manage to obtain a closed-form expression for the covariance matrix Ξ for generic network topologies. Instead we studied the covariance matrices of some simple networks (up to 5 nodes) and were able to extract some patterns, which lead us to a conjecture for a closed-form expression for arbitrary topologies.

We already calculated the variance of the two-node *Disperser* protocol in (8.73). For $c_1 = c_2$, the variance resulted in $x_{\text{T}}/4$. The covariance matrix for

m^{th}-order reactions in a chain and a star network topology with three and four nodes is

$$\Xi_{\text{chain3}} = \Xi_{\text{star3}} = \frac{x_{\text{T}}}{9}\begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix} \tag{9.13a}$$

$$\Xi_{\text{chain4}} = \Xi_{\text{star4}} = \frac{x_{\text{T}}}{16}\begin{pmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{pmatrix} \tag{9.13b}$$

We also calculated the covariance matrix of the four node topology depicted in Figure 9.1:

$$\Xi_{\text{Figure 9.1}} = \frac{x_{\text{T}}}{16}\begin{pmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{pmatrix} \tag{9.14}$$

The covariance matrix seems to be identical for networks with the same number of nodes, i.e. independent on how the nodes are connected. We conjecture that the covariance matrix for the *Disperser* protocol for an arbitrary network topology is

$$\Xi = \frac{\hat{x}_{\text{X}}}{|\mathcal{V}|}\left(|\mathcal{V}|\,\mathbf{I}_{|\mathcal{V}|} - \mathbf{U}_{|\mathcal{V}|}\right) \tag{9.15}$$

where $\mathbf{U}_{|\mathcal{V}|}$ is the $|\mathcal{V}| \times |\mathcal{V}|$-dimensional unit matrix in which each element is one. Hence, the variances are

$$\operatorname{Var}\left[N_{\text{X}_i}\right] = [\Xi]_{i,i} = \hat{x}_{\text{X}}\left(1 - \frac{1}{|\mathcal{V}|}\right) \qquad \forall v_i \in \mathcal{V} \tag{9.16}$$

and the covariances

$$\operatorname{Cov}\left[N_{\text{X}_i}, N_{\text{X}_j}\right] = [\Xi]_{i,j} = -\frac{\hat{x}_{\text{X}}}{|\mathcal{V}|} \qquad \forall \left\{v_i, v_j \in \mathcal{V} \mid i \neq j\right\} \tag{9.17}$$

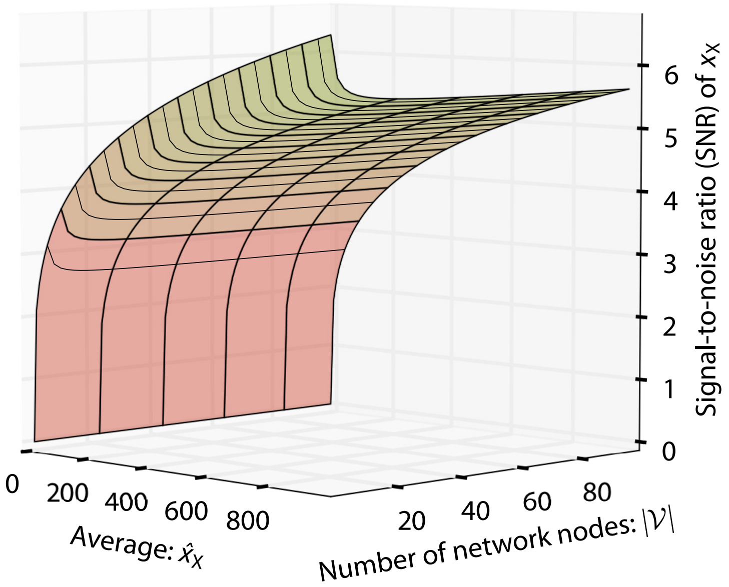


Figure 9.2 Signal-to-noise ratio of *Disperser*: According to the Linear Noise Approximation, the signal becomes more accurate for larger result values and noisier for larger networks.

where $\hat{x}_X = x_T / |\mathcal{V}|$ is the mean, i.e. the average number of X-molecules without intrinsic noise. The SNR is then given as

$$\mathrm{SNR}\left(N_{X_i}\right) = \frac{\mathbb{E}\left[N_{X_i}\right]}{\sqrt{\mathrm{Var}\left[N_{X_i}\right]}} = \sqrt{\frac{\hat{x}_X}{1 - \frac{1}{|\mathcal{V}|}}} \qquad \forall v_i \in \mathcal{V} \qquad (9.18)$$

We verified our conjecture for about 30 different random graphs by calculating their covariance matrix; the conjecture was satisfied by all of them. For the first-order reaction network case ($m = 1$), the steady-state distribution of *Disperser* is a multinomial distribution (Gadgil et al., 2005; Jahnke & Huisinga, 2007), for which the variances and covariances are known and agree with our conjecture.

Interestingly, the mean, the (co-)variances, and hence the SNRs are independent on the reaction order m, the number of catalysts c, and the reaction rate coefficient k. As indicated by (9.18) and depicted in Figure 9.2, the presented result (i.e. the calculated average) becomes more accurate for larger values of the mean. On the other hand, the result becomes more noisy for larger networks. For $|\mathcal{V}| \to \infty$ the SNR is $\sqrt{\hat{x}_X}$, whereas the SNR is maximal ($\sqrt{2\hat{x}_X}$) for a two-node topology.

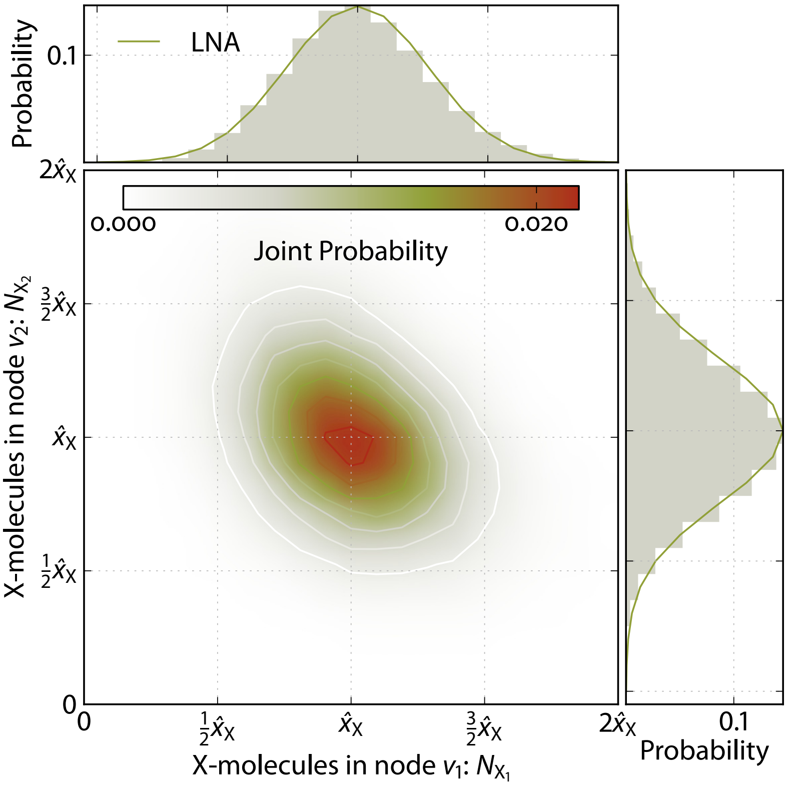


Figure 9.3 Noise of *Disperser* with little molecules: Marginal and joint stationary probability distributions of the number of X-molecules between node v_1 (horizontal axis) and node v_2 (vertical axis) in the network topology depicted in Figure 9.1. The total number of molecules is $x_T = 40$ molecules, resulting in a steady-state quantity of $\hat{x}_X = 10$ molecules; the standard deviation is $\hat{\sigma}_i \approx 2.74$ molecules, the coefficient of variation $\hat{\varsigma}_i \approx 0.274$.

9.3.2 CORRESPONDENCE BETWEEN ANALYTICAL AND SIMULATION RESULTS

To complement this analysis, we carried out simulation runs in Fraglets for different parameters and compared the empirically obtained stationary probability distributions to the Gaussian distributions predicted by the LNA.

Figure 9.3 shows the stationary joint probability distribution of the quantity of X-molecules in node v_1 and v_2 together with their corresponding marginal probability distributions for the network topology depicted in Figure 9.1. We started with a total number of $x_T = 40$ molecules, which results in a steady-state quantity of $\hat{x}_X = \mathbb{E}\left[x_{X_i}\right] = 10$ molecules in each node. According to the LNA, the standard deviation is $\hat{\sigma}_i = \sqrt{\mathrm{Var}\left[N_{X_i}\right]}$, which is approximately 2.74 molecules in this scenario. The coefficient of variation, i.e. the standard deviation normalized with respect to the mean, is quite high at $\hat{\varsigma}_i = \hat{\sigma}_i / \hat{x}_X \approx 0.274$. The bell-shaped curves denote the calculated normal distributions with the mentioned first two moments; they accurately predict the noise of *Disperser* as expected.

For the second scenario, we multiplied the total number of molecules by a factor of 10, resulting in a steady-state concentration of $\hat{x}_X = 100$ molecules, a standard deviation of $\hat{\sigma}_i \approx 8.66$ molecules, and a drastically decreased coefficient of variation of $\hat{\varsigma}_i \approx 0.0866$, which is clearly visible in Figure 9.4.

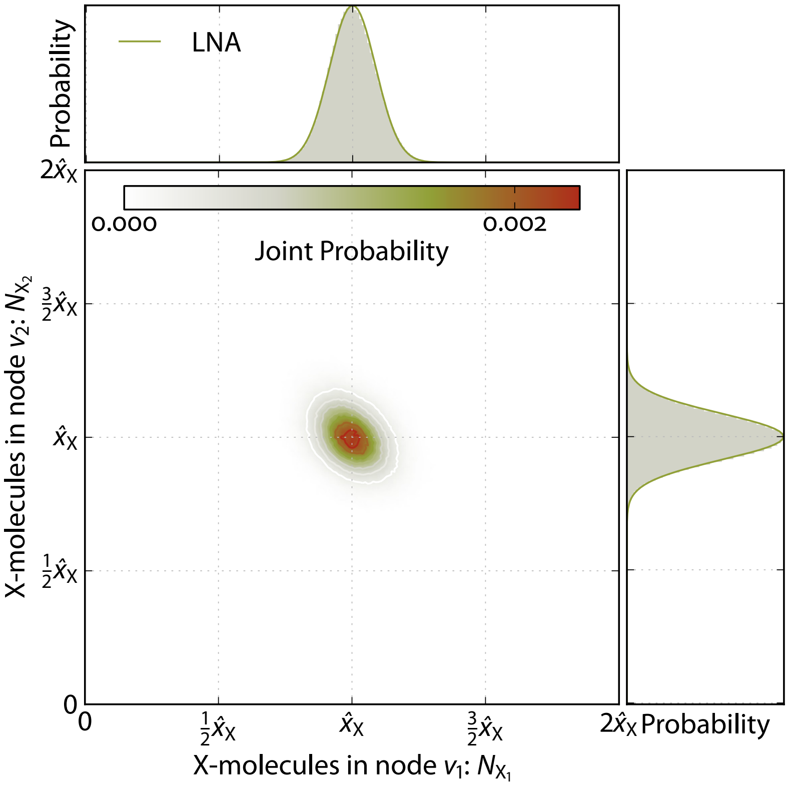


Figure 9.4 Noise of *Disperser* with more molecules: Marginal and joint stationary probability distributions of the number of X-molecules between node v_1 (horizontal axis) and node v_2 (vertical axis) in the network topology depicted in Figure 9.1. The total number of molecules is $x_T = 400$ molecules, resulting in a steady-state quantity of $\hat{x}_X = 100$ molecules; the standard deviation is $\hat{\sigma}_i \approx 8.66$ molecules, the coefficient of variation $\hat{\varsigma}_i \approx 0.0866$.

For the third scenario, we implemented *Disperser* with bimolecular reactions ($m = 2$). That is, each reaction changes the vessel's composition in discrete steps of two molecules. Consequently, the state space only consists of even molecular quantities when we start with an even total number of molecules ($x_T = 40$ molecules). Interestingly, as indicated by Figure 9.5, the first two moments (mean and variance) of the stationary probability distributions for bimolecular reactions are identical to those for unimolecular reactions (compare to Figure 9.3). The void left by the unoccupied states is compensated by the neighboring states, which exhibit an m-fold probability. This supports our analytical result, suggesting that the first two moments are independent of the reaction order m.

DISCUSSION 9.4

We proved that the *Disperser* protocol asymptotically converges to a stable fixed point where each node contains the average number of X-molecules in the network. The position and stability of the fixed point is not affected by the choice of the reaction order, m, the number of catalysts, c, or the reaction coefficient, k, as long as those parameters are identical among all network

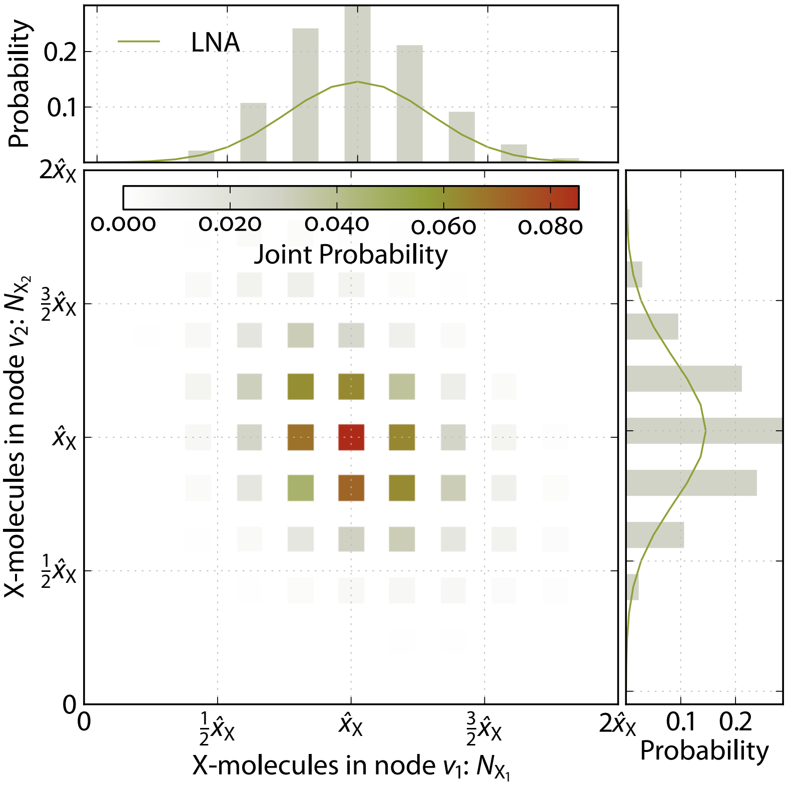


Figure 9.5 Noise of *Disperser* driven by bimolecular reactions: Marginal and joint stationary probability distributions of the number of X-molecules between node v_1 (horizontal axis) and node v_2 (vertical axis) at steady-state in the network topology depicted in Figure 9.1. Bimolecular reactions exchange molecules between the nodes ($m = 2$). The total number of molecules is $x_T = 40$ molecules, resulting in a steady-state quantity of $\hat{x}_X = 10$ molecules; the standard deviation is $\hat{\sigma}_i \approx 2.74$ molecules, the coefficient of variation $\hat{\varsigma}_i \approx 0.274$.

nodes. This *a priori* agreement on the parameters is required for the protocol to reach consensus.

There are probably a lot of other methods to prove *Disperser's* convergence. Cremean and Murray (2003), for example, came up with a very general approach that also applies for the *Disperser* protocol with linear reactions ($m=1$): A global system, composed of local components, each described by ODEs and locally stable, is still globally stable if the components are *linearly coupled*. It is possible to map chemical reaction networks such as the one of *Disperser* to the description of Cremean and Murray (2003). However, like the Deficiency Zero Theorem, Cremean and Murray's method is only able to prove the *existence* of a stable fixpoint without providing its location.

linearly coupled systems are stable

Several parameters influence the efficiency and effectiveness of *Disperser*. Unlike traditional protocols that convey information symbolically, chemical protocols exploit distributed stochastic processes and use randomly fluctuating molecule concentrations and packet rates to communicate. As a consequence, the quality of Disperser's result "signals" as well as the message complexity of the protocol depends on the value range of the expected result: For example, if *Disperser* is used to calculate the average temperature in a sensor network, it is important to decide how the values are encoded, i.e. how a molecular quantity has to be interpreted: degrees celsius, centi-degrees, de-

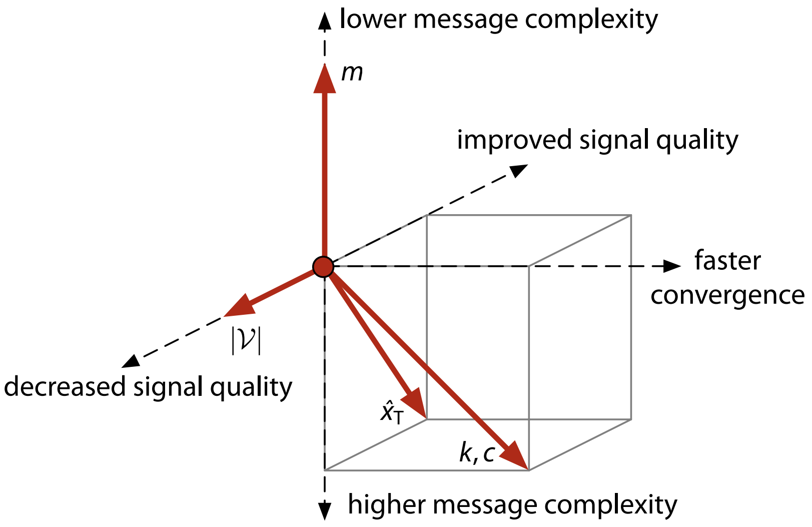


Figure 9.6 Performance effects of *Disperser's* parameters: From a given set of parameters, indicated with the black dot, the quality, message complexity, and convergence time of *Disperser* can be influenced by changing the following parameters: number of network nodes $|\mathcal{V}|$, average number of X-molecules $\hat{x}_\mathrm{T}$, number of catalysts $c = x_{\mathrm{C}_i}$, reaction coefficients k, and the reaction order m.

grees Kelvin, Fahrenheit, or any other encoding. Figure 9.6 relates parameter changes to their qualitative effect on the protocol's performance.

message complexity

The *message complexity*, i.e. the packet rate, is proportional to the average number of molecules; because of the law of mass action scheduling, higher molecule concentrations increase the packet rate. While an increase of the number of C-molecules improves the convergence time, it has a negative effect on the message complexity. On the other hand, the message complexity can be reduced by introducing higher-order reactions that collect multiple molecules in a single packet: m is inversely proportional to the message complexity. However, this recipe has to be handled with care, since the LNA only provides an *approximation* of the noise. Obviously, the reaction order has to be much smaller than the minimum average expected: $m \ll \hat{x}_\mathrm{X}$, because the m^th-order reaction is not able to transmit less than m molecules at a time. As a consequence, no reaction would occur anymore. Also note that multi-molecular reactions are much faster, which usually requires to adapt the reaction coefficients k accordingly.

noise grows weakly with the network size

Our analysis clearly indicates that the inherent noise, with which the result is presented, weakly grows with the network size. It is surprising that the SNR can be calculated from local information only and that it is independent on the degree of the nodes. That is, the topological arrangement of the same number of network nodes has no influence on the noise, as indicated by the identical covariances. Furthermore, the number of C-molecules and the reaction coefficient k has no influence on the SNR.

9.5 SUMMARY

A detailed analysis of the behavior of Disperser was quite simple by resorting to existing mathematical descriptions of chemical reaction dynamics and well-established tools from chemistry to analyze reaction networks. In the previous four chapters, we reviewed some of these methods and showed how they can be applied to CNPs.

We propose to analyze CNPs in the classical way of systems theory: by first identifying all components and their interactions before analyzing the resulting state-space. Translated to CNPs, this means that we first analyze the protocol structurally by mapping a concrete Fraglets program onto its dynamically equivalent abstract macroscopic model. We introduced two algorithms that are capable of converting the rewriting rules of the implicit artificial chemistry Fraglets into a set of explicit species and reaction rules. Thereby, these algorithms explore the sequence space of the Fraglets program by finding its closure and automatically summarize equivalent molecule strings. Note that the sequence space is not equivalent to the state-space: The sequence space defines the dimensionality of the state space and spans its possible realizations.

Once having a description of the protocol in an explicit artificial chemistry, we are able to analyze its dynamics and come up with quantitative proofs. Because we use an exact stochastic reaction algorithm, which brings the behavior of program execution dynamically close to the reaction dynamics of real chemical reactions, we can apply mathematical descriptions and tools that were originally developed for "wet" chemistry. We reviewed three mathematical description for the dynamic behavior of chemical reactions: the exact microscopic stochastic description using the Chemical Master Equation (CME), a macroscopic deterministic approximation using Ordinary Differential Equations (ODEs), and several mesoscopic descriptions, which augment the intrinsic noise to the deterministic trajectories (e.g. the Chemical Langevin Equation (CLE), the Linear Noise Approximation (LNA), and the Two Moment Approximation (2MA)).

In this work we put emphasis on the rigorous mathematical analysis of CNPs. We believe that handcrafted proofs are easier to understand than automated proofs.This approach is usually not taken into account because manual proofs are too complicated. However, in the chemical model, this is no longer the case.

CHAPTER 10

Design and Synthesis of CNPs

On the art of designing chemical reaction networks from which the desired distributed algorithms shall emerge.

> Complexity must be grown
> from simple systems
> that already work. [10]
>
> *Out of Control*
> Kevin Kelly

The design of distributed algorithms is hard to automate. The common problem of finding a software implementation for a given problem is accentuated in a distributed context: In a network, the collective system behavior results from the actions of the participating nodes, which have to be programmed individually. The design of distributed algorithms is inherently bottom-up and is thus more intuitive art than rock-solid science. Therefore, it is virtually impossible to come up with general recipes that guide a developer from the problem statement to the final solution. But what we can and will provide is a set of simple rules and design patterns for common networking motifs, from which complex distributed algorithms can be grown.

Figure 10.1 shows the focus of this chapter in our chemical protocol engineering landscape. This chapter is mainly addressed to engineers. It provides guidelines how to develop chemical protocols. The main design principle is to determine the dynamic behavior of protocols first by constructing the distributed reaction network. Later in the design process, we have to find an implementation of this abstract reaction network that also satisfies structural, microscopic requirements. One common treatment on the microscopic level

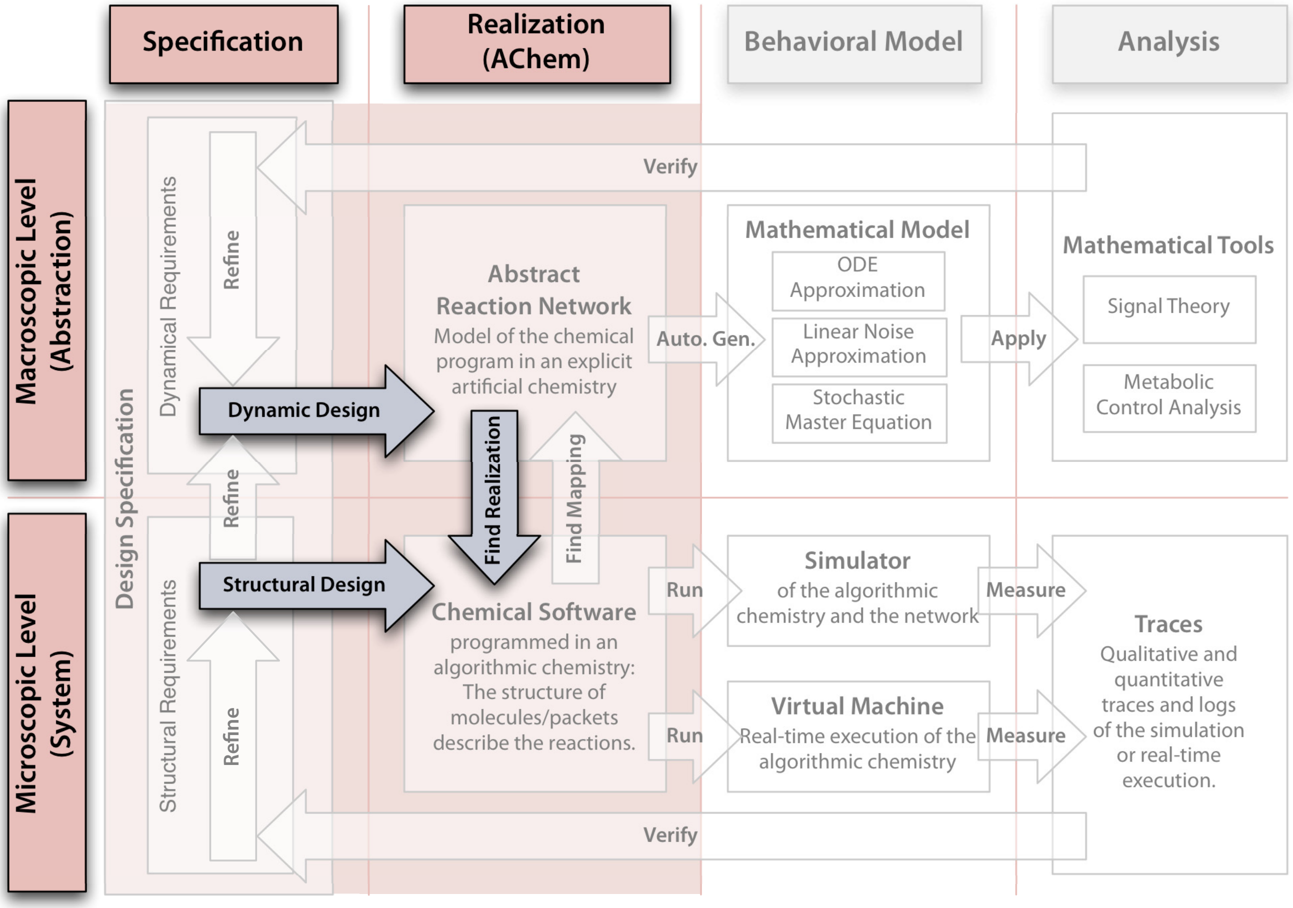

Specification
Realization (AChem)
Behavioral Model
Analysis
Macroscopic Level (Abstraction)
Microscopic Level (System)
Design Specification
Dynamical Requirements
Structural Requirements
Refine
Refine
Refine
Verify
Verify
Dynamic Design
Structural Design
Find Realization
Find Mapping
Abstract Reaction Network
Model of the chemical program in an explicit artificial chemistry
Chemical Software
programmed in an algorithmic chemistry: The structure of molecules/packets describe the reactions.
Auto. Gen.
Mathematical Model
ODE Approximation
Linear Noise Approximation
Stochastic Master Equation
Apply
Mathematical Tools
Signal Theory
Metabolic Control Analysis
Run
Run
Simulator
of the algorithmic chemistry and the network
Virtual Machine
Real-time execution of the algorithmic chemistry
Measure
Measure
Traces
Qualitative and quantitative traces and logs of the simulation or real-time execution.

Figure 10.1 **Design methods in the engineering model**: This chapter discusses how to synthesize well-known reaction motifs to protocols.

is to decorate molecules by payload, which has to be carried between two distant applications.

This chapter is structured as follows: We first expose the chemical protocol design paradigm in Section 10.1 before providing practical tools for protocol designers in Sections 10.2 and 10.3. Essentially, these are rules and patterns of well-understood chemical reaction motifs that proofed useful to synthesize chemical protocols. Finally, in Section 10.4, we demonstrate an application for our design methods by extending the *Disperser* protocol by an automatic neighborhood discovery mechanism. The chapter concludes with a summary in Section 10.5.

10.1 CHEMICAL PROTOCOL DESIGN PARADIGMS

After we developed our first chemical protocols, we recognized that already a small number of molecules and reactions exhibit the desired dynamic behavior whereas in the traditional programming style a complex program structure of event handlers, timers, and so forth is needed. We also observed that the dynamics of the drafted chemical reaction network was mainly responsible for the protocol's function.

Based on these observations, we derived the most important design principle for CNPs: *dynamics precedes structure*. We encourage the designer to first determine and analyze the dynamic behavior of a reaction network before focusing on its implementation in an algorithmic chemistry. This *paradigm shift* of emphasizing the dynamics may be similar to the transition from functional to object-oriented design, when suddenly everything became an object. In CNPs, all problems are reduced to dynamic packet flows. The *flow-based design* process tries to balance packet flows by exploiting the law of mass action behavior of chemical reactions.

dynamics precedes structure

paradigm shift

flow-based design

The advantage of the chemical approach is that we can *outsource parts of the protocol logic to the reaction network dynamics*. For example, there is no need to check each lost packet individually; if packets are lost, the stream of acknowledgments will be smaller than the data packet stream, the concentration of acknowledgments drop, and consequently, the dynamic behavior of reactions consuming the ACKs changes. There is no need to program this functionality symbolically; it is sufficient to arrange the molecules and reactions correctly such that the requested behavior emerges.

outsource protocol logic to the reaction network dynamics

The chemical design process itself is not trivial. A paradigm shift in software development is always accompanied with new tools and methods to plan and implement code. In this thesis, we give first ideas how such a design process may look like. We think that the following aspects have to be studied in detail in order to come up with a proper design for a chemical networking protocol solving a given problem: (1) mapping the problem to an artificial chemistry, (2) choosing between symbolic and representation-free information, and (3) designing dynamic and functional behavior.

MAPPING THE PROBLEM TO AN ARTIFICIAL CHEMISTRY 10.1.1

The first step in developing a chemically inspired distributed algorithm is to get an idea how to map the problem to the chemical model. This requires a good understanding of the problem and the properties of the abstract chemical reaction model such as the law of mass action); experience with previous designs is also helpful.

Often, the *solution* to a problem manifests at a dynamic *equilibrium* of the chemical reaction system. Thus, the designer has to fit the problem to a dynamic description for which an equilibrium exists. Unfortunately, there

solution in equilibrium

are no simple recipes for this process. However, a good staring point is to think about how information shall be represented by the algorithm.

10.1.2 CHOOSING BETWEEN SYMBOLIC AND REPRESENTATION-FREE INFORMATION

A networking protocol conveys information between spatially or virtually separated nodes (computers, or processors). A piece of data shall either be transmitted from one node to another (e.g. client-server model), shared among multiple nodes (e.g. distributed hash-tables), or the network participants shall come to a consensus (e.g. gossiping, distributed aggregation). In traditional networking, a piece of information is encoded and processed *symbolically*: That is, information is encoded as a sequence of binary numbers stored at a certain memory location, and information is conveyed by transmitting binary data *in* a packet, i.e. via the structure of the packet.

symbolic information

Distributed chemical reaction networks allow and even suggest encoding information *representation-free*. Instead of encoding a state symbolically, it is represented by the quantity of a molecular species, and instead of sending binary data, information is conveyed by the rate of distributed reactions. Quantity and rate signals are connected by the law of mass action: By changing the state, i.e. by changing the molecular concentration, the rate of a reaction consuming these molecules will be modulated accordingly.

representation-free information

Matsumaru, Lenser, Hinze, and Dittrich (2007) demonstrated another method of representing information in chemical reaction systems. They encoded a state by the *chemical organization* in which its current attractor is located and demonstrated the usefulness of this approach by solving the maximal independent set problem with a distributed chemistry. Although this approach is promising for binary decision problems, we are more interested in dynamic and "analog" solutions here.

organizational information

The designer first has to decide whether data shall be encoded symbolically, i.e. inside the structure of the molecule, and/or representation-free, i.e. as molecular quantities and reaction rates. As a rule of thumb that often leads to good designs (in our opinion), end-to-end user-information should be encoded symbolically and attached to molecules as payload whereas protocol control loops should be embedded into the distributed chemical reaction networks. Note that *Disperser*, where the aggregated sensor values are encoded as molecular quantities, is a counter-example to this rule.

DESIGNING DYNAMIC BEFORE FUNCTIONAL BEHAVIOR 10.1.3

Unlike in traditional protocol design, where the dynamic behavior is often studied after the functional solution is found, we encourage the designer to design and analyze the dynamic behavior first, because there is a direct and easy graspable relation between the reaction network topology and its dynamic behavior. We promote the *reaction network graph* as an instrument to visualize the behavior of distributed reaction networks on a high level. The goal of such a visual representation is to bring the design up to the level of intuitive reasoning. At the same time, we provide simple *rules* that help verify the intuition, and simple patterns or *motifs* that can be combined in order to assemble complex protocols.

reaction network graph

rules

motifs

Once the abstract reaction network is found that solves the problem dynamically, we move one conceptual layer down to the chemical programming language. For Fraglets, this means that we have to find the concrete strings that induce the designed reaction network. In this design phase, the molecule strings are additionally decorated by symbolically encoded user-information if necessary.

In the following two sections, we present conservation rules in chemical reaction networks (Section 10.2) and propose simple motifs or design patterns for commonly occurring sub-problems in protocol design (Section 10.3). This toolbox shall support the designer in synthesizing complex reaction networks from the well-understood simple motifs, as indicated in K. Kelly's quote preceding this chapter.

CHEMICAL CONSERVATION RULES 10.2

The design rules discussed in this section are directly related to information obtainable from the reaction graph. They serve the protocol designer to decide in an early design stage how the behavior of a reaction network changes if the reaction network topology is altered. The designer shall be able to predict what happens if he/she draws a new arrow between two molecules in the reaction graph or deletes an existing one.

KIRCHHOFF'S CONSERVATION LAWS 10.2.1

Kirchhoff's laws are conservation rules that apply to electronic circuits. Derived from Maxwell's equations they describe how energy is conserved in electronic circuits. These laws can also be applied to chemical reaction networks (Perelson & Oster, 1974; Fishtik, Callaghan, & Datta, 2004a, 2004b, 2005; Fehribach, 2009).

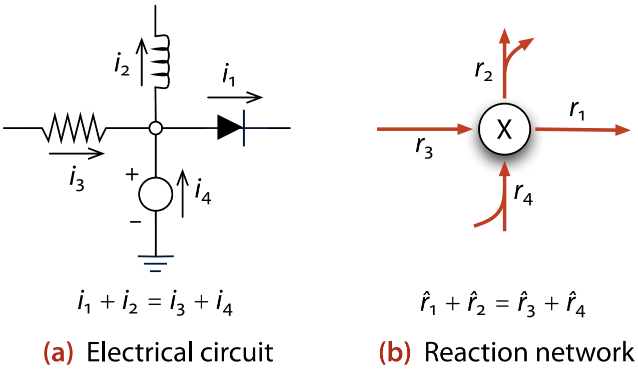


Figure 10.2 Kirchhoff's current law: (a) The total current flowing into a node is equal to the total current flowing out of that node. (b) At equilibrium the total production rate of a molecule is equal to the total consumption rate.

(a) Kirchhoff's Current Law

Kirchhoff's current law (or point rule) uses the fact that energy is conserved, and hence the sum of current flowing into a node of an electrical circuit is equal to the sum of current flowing out of that node (see Figure 10.2(a)). A similar conservation law exists for arbitrary chemical reaction networks at equilibrium: the total production rate of each species is equal to its total consumption rate. This simple rule can be proven with the help of the deterministic ODE equation. Consider, for example, the species X in Figure 10.2(b). Its quantity changes over time according to the ODE

$$\dot{x}_X = -r_1 - r_2 + r_3 + r_4 \tag{10.1}$$

When reaching equilibrium the quantity does not change ($\dot{x}_X = 0$), and hence

$$\overbrace{\hat{r}_1 + \hat{r}_2}^{\text{influx}} = \overbrace{\hat{r}_3 + \hat{r}_4}^{\text{efflux}} \tag{10.2}$$

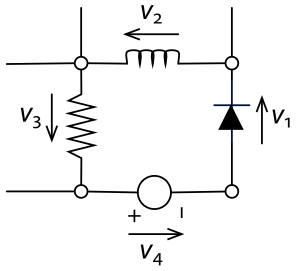
Nodes in electronic circuits cannot store electrons, but chemical species may temporarily buffer molecule instances. A chemical species acts more like a capacitor than an ideal wire. But like the capacitor, which exhibits equal current in- and efflux if the circuit reaches equilibrium, the in- and efflux of a molecular species is equal at equilibrium, too. Therefore, Kirchhoff's current law is *only valid* for reaction networks *at equilibrium*.

only valid at equilibrium

(b) Kirchhoff's Voltage Law

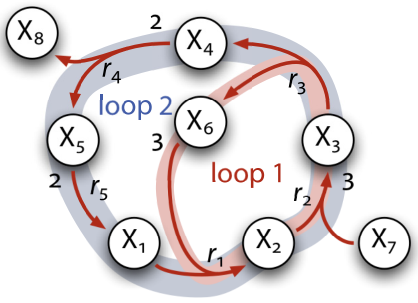
The second law by Kirchhoff – the voltage law (or loop rule) – states that the directed sum of the potential differences (voltages) around a closed circuit must be zero (see Figure 10.3).

Unfortunately, there is no corresponding law for chemical reaction networks that is directly extractable from the reaction network graph. This is



$v_1 + v_2 + v_3 + v_4 = 0$

Figure 10.3 Kirchhoff's voltage law: The directed sum of the potential differences (voltages) around a closed circuit must be zero. There is no direct equivalence for reaction network graphs.



$$x_{X_1} + x_{X_2} + x_{X_3} + x_{X_4} + x_{X_5} = \text{const.}$$
$$x_{X_2} + x_{X_4} + x_{X_6} = \text{const.}$$

(a) Reaction Network

$$\begin{array}{c|ccccc}
 & r_1 & r_2 & r_3 & r_4 & r_5 \\
\hline
X_1 & -1 & 0 & 0 & 0 & 1 \\
X_2 & 1 & -1 & 0 & 0 & 0 \\
X_3 & 0 & 3 & -1 & 0 & 0 \\
X_4 & 0 & 0 & 1 & -2 & 0 \\
X_5 & 0 & 0 & 0 & 1 & -2 \\
X_6 & -3 & 0 & 1 & 0 & 0 \\
X_7 & 0 & -1 & 0 & 0 & 0 \\
X_8 & 0 & 0 & 0 & 1 & 0
\end{array}$$

loop 1   loop 2

(b) Stoichiometric Matrix

Figure 10.4 Molecule conservation loops: (a) If the number of molecules consumed are equal to the number of molecules produced along a loop of a reaction network, and if every species along this loop is only produced by one reaction and consumed by one reaction, then, the number of molecules along the species on that loop is constant. **(b)** The first criterion is met if the stoichiometric coefficients of all reactions participating a loop sum up to zero.

because a (multi-molecular) reaction links more than two species and because reactions are generally not reversible in our artificial setting. Perelson and Oster (1974) presented a method how the reaction network graph can be converted into a linear reaction route graph to which the voltage law can be applied. Fishtik et al. (2004a, 2004b, 2005) recently studied the properties of reaction route graphs in detail, followed by Fehribach (2009), who calls them Kirchhoff graphs.

We do not review these methods in detail, because the necessary translation of the reaction network graph to a Kirchhoff graph complicates the design process. Recall that we aim at methods that help the designer of CNPs to iteratively design reaction networks with feedback from simple and intuitively graspable rules on the visual level.

MOLECULE CONSERVATION LOOPS 10.2.2

A reaction network graph may contain several loops over a certain set of species. In molecule conservation loops, the summed quantity of the traversed species is constant. Figure 10.4(a) shows a reaction network with two molecule conservation loops.

Generally, the total number of molecules in a loop is constant if the following conditions are met:

1. The total number of molecules consumed by reactions along the loop must be equal to the total number of molecules produced. This is equivalent to the requirement that the sum of all stoichiometric coefficients belonging to the reactions and the species of the loop are zero (see Figure 10.4(b)).

2. Every species along the loop must only be altered (consumed or produced) by reactions in this or another conservation loop.

The loop conservation law not only holds at equilibrium but also in the transient phase. The simplest conservation loop is spanned by a catalytic reaction such as $X + \ldots \longrightarrow X + \ldots$: The first requirement is met because the same reaction that consumes the catalyst X re-creates it afterwards. The second requirement is only met if no other reactions consume or produce the catalyst unless they belong to another conservation loop.

Another example is the *Disperser* protocol (see Section 5.4). In a two-node network topology, there is only one reaction loop in which the molecules are sent between the two nodes. By attaching further nodes, the loop is extended by an additional reversible reaction, each consuming a molecule and producing it in another node.

In the next section, we will illustrate several reaction motifs and prove them by coming back to the conservation rules introduced in this section. With their help, we hope to build complex chemical networking protocols.

10.3 MOTIFS — COMMON DESIGN PATTERNS FOR CNPS

Design patterns provide simple and elegant recipes for common problems. They were introduced for object-oriented programming by Beck and Cunningham (1987) and gained popularity with the book *Design Patterns: Elements of Reusable Object-Oriented Design* (Gamma, Helm, Johnson, & Vlissides, 1995). In this section, we suggest a similar pattern-guided approach to design CNPs. We present a non-exhaustive list of chemical design patterns – which we call *motifs* – and study their behavior and application cases.

motifs

In the object-oriented paradigm, design patterns show the relationships among classes and the interaction between objects. Our chemical motifs are small reaction networks performing a certain function on the macroscopic, dynamic level. The idea to study simple reaction networks is not new: Deckard et al. (2009) enumerated all reaction networks consisting of up to ten species and seven bimolecular reactions and classified the resulting 47 million motifs. Paladugu et al. (2006) found motifs for bi-stable switches using evolutionary algorithms. Deckard and Sauro (2004) evolved arithmetic operation motifs using unimolecular and bimolecular reactions whereas Buisman, ten Eikelder, Hilbers, and Liekens (2008) came up with arithmetic operation motifs based on basic enzymatic reactions.

We do not aim at exhaustively enumerating all reaction networks possible. Our selection of motifs shall be helpful for designing chemical protocol software. That is, the motifs have to perform a useful function, their dynamic behavior must be well understood, and they have to be combined easily to larger reaction networks. This latter aspect is important, as the dynamics of reaction networks may be nonlinear, which invalidates the superposition principle.

We classified the motifs into the following groups: Section 10.3.1 first provides programming language dependent motifs by showing how basic reactions are implemented in Fraglets. Section 10.3.2 introduces reaction networks that are able to compute arithmetic functions. Section 10.3.3 then illustrates how representation-free state information is conveyed across the network. Finally, Section 10.3.4 provides neighborhood discovery motifs.

FRAGLETS LANGUAGE MOTIFS 10.3.1

In the following, we present *idiomatic motifs* for Fraglets. Like in object-oriented design, where a certain set of patterns is dedicated for C++-specific design issues (Coplien, 1991), some chemical reaction motifs describe Fraglets' handling of reactions. In particular, we demonstrate how bimolecular, unimolecular, and multi-molecular reactions are realized in Fraglets. By first demonstrating how basic reactions can be implemented in Fraglets, we can discuss the remaining motifs independent on the chemical programming language.

idiomatic motifs

Bimolecular Reactions in Fraglets (a)

As discussed in Section 5.2, a bimolecular reaction in Fraglets, such as $X + Y \longrightarrow \ldots$, is induced by a synchronization rule triggered by a pair of fraglets, one starting with `[match` σ Φ`]` (we call it the *active molecule*) and the other with the corresponding header symbol `[`σ Ψ`]` (the *passive molecule*). The induced bimolecular reaction

active molecule

passive molecule

$$\underbrace{[\,\overbrace{\texttt{match}\ \sigma}^{\text{cons.}}\ \ \overbrace{\Phi}^{\text{prod.}}\,]}_{\text{active molecule}} + \underbrace{[\ \overbrace{\sigma}^{\text{cons.}}\ \ \overbrace{\Psi}^{\text{prod.}}\ ]}_{\text{passive molecule}} \xrightarrow{k=1} [\Phi\ \ \Psi] \qquad (10.3)$$

is implemented by two different regions in the participating fraglet strings: the consumption (matching) part and the production part. Whereas the *consumption part* in the header of the two fraglets identifies the reaction partners, the *production part* defines what transformations are applied to the concatenated transient molecule. Here, we mainly focus on the consumption part, that is, the left hand side of a reaction, as the right hand side may contain arbitrary `split` or `fork` instructions to produce multiple products.

consumption/ production parts

The consumption part of each reactant may optionally be prepended by the `weight`-instruction, which changes the reaction coefficient of the induced reaction. The effective reaction coefficient is the product of the weights of all reactants, for example

$$[\texttt{weight}\ w_1\ \texttt{match}\ \sigma\ \ \Phi] + [\texttt{weight}\ w_2\ \sigma\ \ \Psi] \xrightarrow{k=w_1 \cdot w_2} [\Phi\ \ \Psi] \qquad (10.4)$$

where w_1 and w_2 are floating point numbers.

(b) Multi-Molecular Reactions in Fraglets

Multi-molecular reactions, such as $2X + Y + Z \longrightarrow \ldots$, are induced by an active fraglet starting with a symbol of the `mmatch`-family:

$$[\texttt{mmatch}\ n\ \sigma_1\ \ \ldots\ \ \sigma_n\ \ \Phi] \qquad (10.5)$$

Such an active fraglet reacts with n passive fraglets, each starting with the corresponding header symbol σ_i $(1 \leq i \leq n)$ and concatenates their tails. Consider, for example, the following tri-molecular reaction:

$$[\texttt{mmatch}\ 2\ \sigma_1\ \ \sigma_2\ \ \Phi] + [\sigma_1\ \ \Psi_1] + [\sigma_2\ \ \Psi_2] \longrightarrow [\Phi\ \ \Psi_1\ \ \Psi_2] \qquad (10.6)$$

Its reaction rate is governed by the law of mass action, meaning that the rate is proportional to the product of the number of reactants. If the same tag σ_i appears more than once in the active fraglet, say m times, m instances of species starting with this tag are joined. Consequently, the reaction rate is proportional to the number of $[\sigma_i\ \ \ldots]$-molecules, raised to the m-th power. Like for bimolecular reactions, the `mmatch`-instruction may be prepended by the `weight`-instruction in order to modify the reaction coefficient.

Unimolecular Reactions in Fraglets *(c)*

An unimolecular reaction, such as $X \longrightarrow \ldots$, is a *special case of a multi-molecular reaction*, in which the active `mmatch`-fraglet reacts with zero passive fraglets. Indeed, the following unimolecular reaction leads to the expected result:

special case of a multi-molecular reaction

$$\texttt{[mmatch 0 }\Phi\texttt{]} \longrightarrow \texttt{[}\Phi\texttt{]} \tag{10.7}$$

Note the difference to the `nop`-instruction. The `mmatch`-symbol triggers a synchronization rule, which is scheduled according to the law of mass action, whereas all transformations (such as `nop`) are executed immediately.

Assisted Reactions in Fraglets *(d)*

As shown above, a multi-molecular reaction of order m is implemented in Fraglets by one active fraglet and $m-1$ passive fraglets (including the special bimolecular and unimolecular cases). In fact, the active fraglet determines the tags of the passive fraglets it reacts with. Thus, the active fraglet takes a special role by fully determining the consumption part of the reaction.

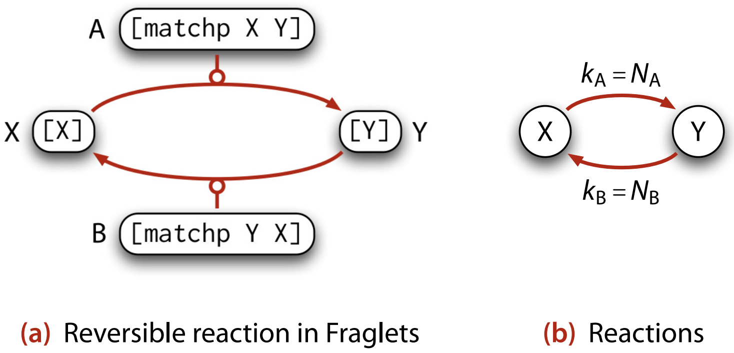
This asymmetry in the Fraglets language is one of the reasons why not all reaction networks can be implemented: We say that *Fraglets is not chemically complete*. For example, two active Fraglets will never react, nor will two passive fraglets. A yet simpler example causing problems is the reversible reaction $X \longleftrightarrow Y$. The molecules X and Y cannot be mapped directly into Fraglet strings. Our first guess would be to map the abstract species X to the fraglet `[mmatch 0 Y]`, inducing the forward reaction, but the result would be Y = `[Y]`, which is a passive fraglet that cannot react autonomically. Replacing the symbol `Y` by another first-order reaction `mmatch 0` would lead to infinite regression. This is a consequence of the general Fraglets paradigm according to which every instruction must reduce the length of a fraglet string in order to avoid infinite transformation loops (see Section 5.2.4 on page 62).

Fraglets is not chemically complete

A work-around for this situation is to introduce new species – active catalyst fraglets A and B – and let the main participants X and Y be represented by passive fraglets. The above reversible reaction can be implemented in Fraglets as depicted in Figure 10.5(a); note that we already used this concept in the *Disperser* protocol. The number of persistent catalysts does not change, as the catalysts regenerate themselves after being consumed. Fraglets provides the `matchp` synchronization rule for this purpose.

Figure 10.5(b) shows the chemical reaction network that is dynamically equivalent to Figure 10.5(a). In Figure 10.5(b), we replaced the catalysts by reaction coefficients. This is allowed because according to the law of mass

Figure 10.5 **Reversible reaction in Fraglets**: (a) An assisted unimolecular reaction is implemented by a bimolecular Fraglets reaction where an active catalyst defines reactant and product tag. (b) This schematic illustration shows the corresponding chemical reaction network that is dynamically equivalent to the Fraglets program in (a).



(a) Reversible reaction in Fraglets

(b) Reactions

action, the reaction rate is proportional to both the reaction coefficient *and* the number of reactants.

This method of using a catalyst to define a reaction in Fraglets is what we call *assisted reaction*. In fact, the active catalyst fraglet completely defines the reaction: The catalyst fraglet is not mapped to an abstract *molecule* like [X] is mapped to X; instead the catalyst is mapped to and represents the abstract *reaction* X ⟶ Y. Every multi-molecular reaction of order m can be implemented as an assisted reaction, where an active catalyst represents the reaction itself and processes m passive fraglets. Thus, the assisted method increments the reaction order by one and, at the same time, allows an (integer) reaction coefficient to be specified via the initial number of catalysts.

assisted reaction

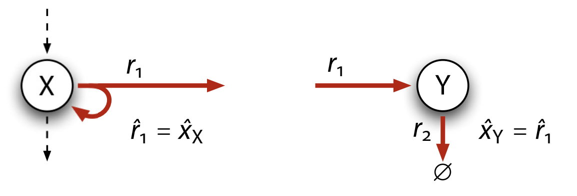
fraglet represents reaction, not molecule

We often resort to this method when implementing reaction networks in Fraglets, because it is much simpler to find catalysts that rewrite passive fraglets than finding active fraglets that represent individual molecules. But this method fails in an active networking context: once installed, a persistent catalyst can never be removed again. Thus, we have to find other methods to dynamically deploy chemical code in the network. We will come back to active code deployment in the third part of this thesis.

For the generic rules and motifs discussed in the following sections, we do not consider a concrete Fraglets implementation and assume that any abstract reaction network can be mapped to Fraglets. The example in Section 10.4 then demonstrates how the assisted reaction method is used to map a given chemical reaction network down to the corresponding Fraglets implementation.

10.3.2 ARITHMETIC MOTIFS

Chemical networking protocols often use implicit information such as the multiplicity of molecules and the rate of reactions to store protocol states and convey them across the network. The *Disperser* protocol is an example



(a) Quantity to rate **(b)** Rate to quantity

Figure 10.6 Motif to map a molecular quantity to a rate and vice-versa: **(a)** The rate of the unimolecular reaction r_1 is equal to the number of reactants X, x_X. **(b)** The influx and efflux rates are balanced. If the only efflux of a species X is a unimolecular reaction, the steady-state concentration equals its influx rate r_1.

that is completely based on representation-free encoding, where even user-information is encoded in a non-symbolic way. This requires methods to combine and process "molecular concentrations".

In the following, we show how a molecular quantities can be converted to a reaction rate and vice-versa and present several methods to compute arithmetic functions of such signals. After a settling time – at equilibrium – the reaction network presents the result as number of molecules of a result species. We show the correctness of the motifs by using a deterministic steady-state analysis when needed (see Section 8.2) or by referring to one of the conservation rules discussed in Section 10.2.

Conversion Between Molecular Quantity and Rate *(a)*

The two very simple motifs shown in Figure 10.6 convert the quantity of a species into a reaction rate and vice-versa by exploiting the law of mass action. As depicted in Figure 10.6(a), a single unimolecular reaction is sufficient to convert the quantity of species X to a reaction rate r_1. Note that other reactions (dashed arrows) may change the quantity of X, but this is out of our focus here. The measurement reaction r_1 intentionally does not alter the quantity of its reactant species X by regenerating the consumed molecules. On average, according to the law of mass action, the rate $\hat{r}_1$ is equal to the number of X-molecules:

$$\hat{r}_1 = \hat{x}_X \tag{10.8}$$

A reaction rate can be converted back to an quantity value as depicted in Figure 10.6(b). If a molecule (such as Y) is product and reactant of two different reactions at the same time, Kirchhoff's current law states that their reaction rates must have the same value at equilibrium ($\hat{r}_1 = \hat{r}_2$). Furthermore, according to the law of mass action, the efflux rate r_2 is equal to the number of reactants ($\hat{r}_2 = \hat{x}_Y$). Hence, the number of Y-molecules is equal to the influx rate.

$$\hat{x}_X = \hat{r}_1 \tag{10.9}$$

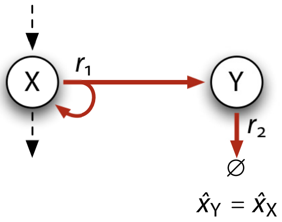


Figure 10.7 **Quantity-mirroring motif**: X molecules are copied to Y. Because at equilibrium the inflow rate must be equal to the outflow rate, the quantity of Y follows the quantity of X.

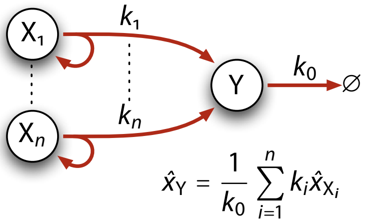


Figure 10.8 **Linear combination motif**: The reaction network computes the sum of all X_i-molecule quantities, weighted by the coefficients k_i, and presents the result in the number of Y-molecules.

In Figure 10.6(b), reaction r_2 is a decay reaction that destroys Y-molecules in order to keep the balance. This reaction may be redirected to other species in order to process the quantity signal further. Note however, that when additional reactions are added to or from Y, its quantity is not equal to the rate r_1 anymore. If additional reactions have to be attached to Y they need to preserve its quantity as shown in Figure 10.6(a).

(b) Quantity-Mirroring Motif

The two previous motifs can be combined in order to "copy" or mirror the quantity of one species to the quantity of another species as depicted in Figure 10.7. Reaction r_1 sends copies of X to species Y where reaction r_2 decays the instances according to the law of mass action. The quantity of Y follows the quantity of X:

$$\hat{x}_Y = \hat{x}_X \tag{10.10}$$

(c) Linear Combination Motif

By extending the previous motif, we obtain a reaction network that calculates the weighted sum of molecular quantities as depicted in Figure 10.8. The total influx rate of Y is $\sum_{i=1}^{n} k_i x_{X_i}$, the efflux rate is $k_0 x_Y$. At equilibrium the influx rate is equal to the efflux rate and hence, the steady-state quantity of Y is

$$\hat{x}_Y = \frac{1}{k_0}\sum_{i=1}^{n} k_i \hat{x}_{X_i} \tag{10.11}$$

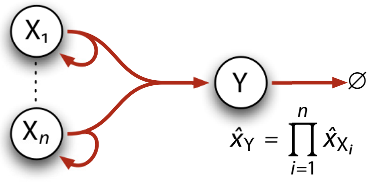


Figure 10.9 Product motif: The reaction network computes the product of the quantities of *n* different species, X_i. The result is presented in the number of Y-molecules.

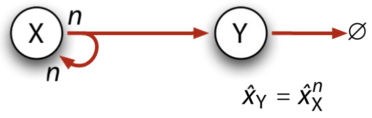


Figure 10.10 Exponentiation motif (1): The reaction network computes the *n*-th power of the number of X-molecules and presents the result in the number of Y-molecules.

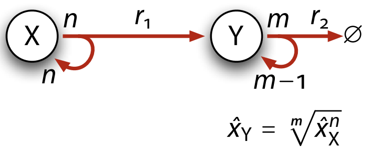


Figure 10.11 Exponentiation motif (2): The reaction network computes the *n*/*m*-th power of the number of X-molecules and presents the result in the number of Y-molecules.

Product Motif (d)

Calculating the product of two quantities is straightforward, as the law of mass action always yields reaction rates that are equal to the product of the number of reactant molecules. Thus, in order to multiply the quantity of *n* different species, we have to install a multi-molecular reaction of order *n* among them as depicted in Figure 10.9. The steady-state quantity of the result species Y is

$$\hat{x}_Y = \prod_{i=1}^{n} \hat{x}_{X_i} \tag{10.12}$$

Exponentiation Motif (e)

Raising a variable to the *n*-th power is a special case of calculating the product of *n* variables, i.e. *n* times the same variable. This observation suggests how to design a reaction network to *raise the quantity of a species to the n-th power* ($n \in \mathbb{N}$). The resulting reaction network is depicted in Figure 10.10.

raise to the *n*-th power

By letting the decay reaction of Y be a multi-molecular reaction, too, we extend this motif such that it raises the quantity of X to a fraction of integers n/m as depicted in Figure 10.11. Note that the decay reaction has to regenerate $m - 1$ instances of the m consumed Y-molecules in order to only consume one instance per reaction event. The reaction network computes the function

$$\hat{x}_Y = \sqrt[m]{\hat{x}_Y^n} \tag{10.13}$$

For example, to calculate the *square root* of x_X we let the first reaction be unimolecular ($n = 1$) and we chose the decay reaction to be bimolecular

square root

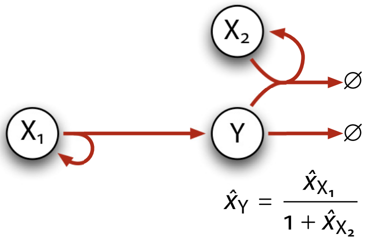


Figure 10.12 Division motif (1): The reaction network computes the quotient of the quantity of species X_1 divided by X_2 and presents the result in the number of Y-molecules.

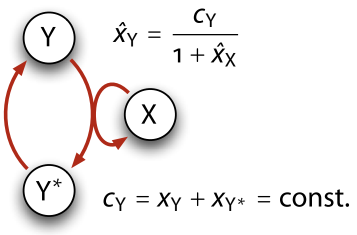

Y
X
Y*
$\hat{x}_Y = \frac{c_Y}{1+\hat{x}_X}$
$c_Y = x_Y + x_{Y^*} = \text{const.}$

Figure 10.13 Division motif (2): The reaction network computes the quotient of the total number of Y- and Y*-molecules divided by the quantity of X and presents the result in the number of Y-molecules.

polynomial functions

$(m = 2)$. A combination of the exponentiation motif (Section 10.3.2(e)) and the linear combination motif (Section 10.3.2(c)) allows for calculating *polynomial functions*.

(f) Division Motif

It is more difficult to design a motif that divides the quantity of one species by the quantity of another species. For this purpose, we have to find a reaction network that lowers the number of result molecules when the number of another molecule increases. That is, we have to drain molecules from the result (quotient) species in proportion to the quantity of the divisor species. Figure 10.12 shows a reaction network that has this property. By applying Kirchhoff's current law to the quotient species Y we find out that at equilibrium, the influx $\hat{x}_{X_1}$ is equal to the efflux $\hat{x}_Y + \hat{x}_{X_2}\hat{x}_Y$, and hence

$$\hat{x}_Y = \frac{\hat{x}_{X_1}}{1+\hat{x}_{X_2}} \tag{10.14}$$

An alternative implementation is depicted in Figure 10.13. It consists of a molecule conservation loop Y–Y* in which the total number of molecules is constant: $c_Y = x_Y + x_{Y^*} = \text{const.}$ The dividend is represented by the total number of molecules in this loop, c_Y. The application of Kirchhoff's current law yields a steady-state quantity of

$$\hat{x}_Y = \frac{c_Y}{1+\hat{x}_X} = \frac{x_Y + x_{Y^*}}{1+\hat{x}_X} \tag{10.15}$$

Difference Motif (g)

Like for the division operation, it is not straightforward how to construct a reaction network that computes the difference between two molecular quantities. We study this motif in more detail and reveal how one can find similar arithmetic motifs: Our plan is to start with the target equation, to which we apply arithmetic operations until the equation can be converted to a valid reaction network:

At equilibrium, the network shall compute the difference between the molecular quantities of two species X_1 and X_2, and present the result in the number of Y-molecules:

$$\hat{x}_{\mathrm{Y}} = \hat{x}_{\mathrm{X}_1} - \hat{x}_{\mathrm{X}_2} \tag{10.16}$$

We previously recognized that Kirchhoff's current law yields an equation containing sums. So let us rewrite this equation to

$$\overbrace{\hat{x}_{\mathrm{X}_1}}^{\text{influx}} = \overbrace{\hat{x}_{\mathrm{Y}} + \hat{x}_{\mathrm{X}_2}}^{\text{efflux}} \tag{10.17}$$

and regard the left-hand side as influx of species Y and the right-hand side as its efflux. The problem with this equation is that according to the law of mass action, the efflux must always be proportional to the number of reactants. This is violated by the term $\hat{x}_{\mathrm{X}_2}$. By multiplying (10.17) by $\hat{x}_{\mathrm{Y}}$ we obtain equation

$$\underbrace{\overbrace{\hat{x}_{\mathrm{X}_1}\hat{x}_{\mathrm{Y}}}^{\text{influx}}}_{r_1} = \overbrace{\underbrace{\hat{x}_{\mathrm{Y}}^2}_{r_2} + \underbrace{\hat{x}_{\mathrm{X}_2}\hat{x}_{\mathrm{Y}}}_{r_3}}^{\text{efflux}} \tag{10.18}$$

which can easily be converted to the corresponding reaction network depicted in Figure 10.14. The first reaction is responsible for the influx of Y: It adds one molecule at rate $r_1 = x_{\mathrm{X}_1} x_{\mathrm{Y}}$ and, at the same time, maintains the population of input molecules. The second and third reactions together form the efflux of Y: The second reaction removes one Y-molecule at rate $r_2 = x_{\mathrm{Y}}^2$ whereas the third reaction removes one instance at rate $r_3 = x_{\mathrm{X}_2} x_{\mathrm{Y}}$.

Such a reverse-engineering process often leads to the reaction network that calculates the desired arithmetic function.

Constant Quantity Motif (h)

The motif depicted in Figure 10.15 forces the concentration of the target species Y to a certain steady-state value, regardless of its initial quantity:

Figure 10.14 Difference motif: The reaction network subtracts the number of X_2-molecules from the number of X_1-molecules and presents the result in the number of Y-molecules. If $x_{X_2} > x_{X_1}$, the presented quantity is $x_Y = 0$ (a quantity can never be negative).

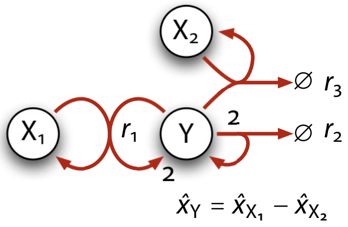
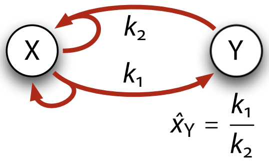


Figure 10.15 Constant quantity motif: This reaction network forces the number of Y-molecules to be k_1/k_2, independent on its initial quantity.



$$\hat{x}_Y = \frac{k_1}{k_2} \tag{10.19}$$

We need a helper species X to force Y to this value. Neither of the two reactions alter the quantity of the helper molecule. Interestingly, the quantity of the helper molecule does not influence the result. According to Kirchhoff's current law, the influx must be equal to the efflux of the target molecule Y at equilibrium; that is, $k_1\hat{x}_X = k_2\hat{x}_X\hat{x}_Y$. The number of X-molecules cancels out, yielding the result in (10.19).

In reality, the quantity cannot take an arbitrary floating-point value as suggested by (10.19), as molecule Y is present with a integer multiplicity. However, the average number of Y-molecules over time may very well be a fraction of an integer. For example, if $k_1 = 1$ and $k_2 = 2$ a Y-molecule is only present half of the time. Another reaction may use Y as input and receives molecules with the correct rate of 0.5 molecules/s. Anyway, we have to keep in mind that such low concentrations are afflicted with heavy noise.

10.3.3 TRANSMISSION MOTIFS

All motifs discussed so far operate in a single vessel. In this subsection, we introduce motifs that can be used to convey information from one vessel to a distant vessel in a network, establishing a communication between the two nodes.

(a) Remote Quantity-Mirroring Motif

The quantity-mirroring motif discussed before (see Section 10.3.2(b)) is commonly used to copy the state (molecular quantity) of one node to another node as depicted in Figure 10.16. Node v_1 sends molecules to node v_2 with a

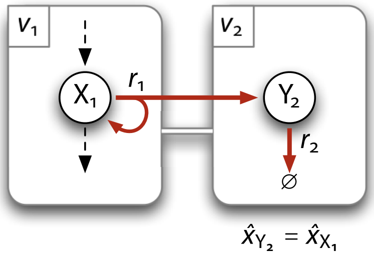


Figure 10.16 Remote quantity-mirroring motif: X_1-molecules in node v_1 are copied to node v_2 with a rate equal to the number of X-molecules. Because at equilibrium, the inflow rate must be equal to the outflow rate, and if we assume that no packets are lost, the number of Y-molecules follows the number of X-molecules.

rate equal to the quantity of the local species X_1: $r_1 = x_{X_1}$. If we assume that no packets are lost, the molecules arrive at the target node with the same rate. There, this rate is converted back to an quantity signal. At equilibrium, the number of Y_2-molecules is equal to the number of X_1-molecules in the originating node:

$$\hat{x}_{Y_2} = \hat{x}_{X_1} \tag{10.20}$$

If the link between the two nodes is afflicted with *packet loss*, the number of Y_2-molecules in the target node is lowered in proportion to the packet loss probability. Thus, a chemical rate signal behaves more like an analog than a digital signal: it gets perturbed in face of noise (packet loss).

packet loss

All motifs discussed so far may be extended to the distributed case. The chemical execution model only requires that all reactants of a given reaction reside in the same node. Thus, the protocol designer is free to choose whether a computation is carried out locally or whether it is *distributed* among several network nodes.

distributed computation

Echo and Packet Loss Motifs (b)

The echo motif can be used to determine whether a certain network node is reachable as depicted in Figure 10.17(a). It is based on the remote quantity-mirroring motif (Section 10.3.3(a)), but redirects the decay reaction of E_2 back to the originating node. At equilibrium, the quantity of X_1 is copied to E_2, which is copied back to Y_1. If the destination node is not reachable, there will be no Y_1-molecules.

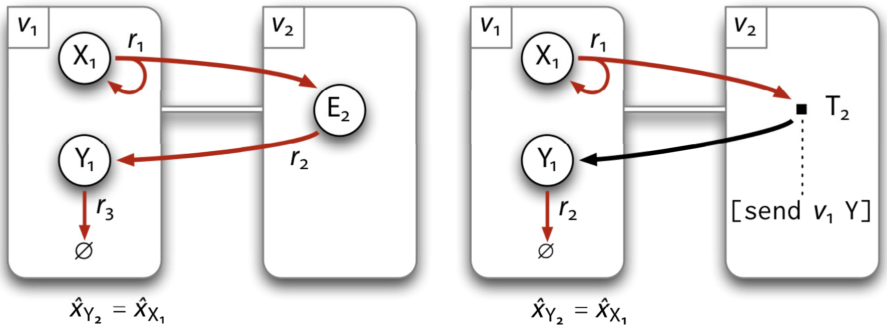
Figure 10.17(b) shows an equivalent *implementation in Fraglets* where the remote molecule is not scheduled according to the law of mass action. Instead, the fraglet sent to the remote node is a transient molecule, starting with the transformation instruction `send`, which immediately sends its tail back to the originating node. We usually prefer this alternative, active networking approach where the echoing code in node v_2 is sent with the echo request.

implementation in Fraglets

The same motif can be used to compute the *round-trip packet loss probability* to a certain node. We previously recognized that the distributed quantity-

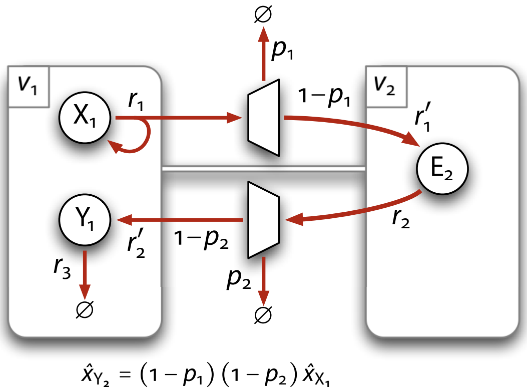
round-trip packet loss probability

Figure 10.17 **Echo motif**: **(a)** Two remote mirroring motifs in series. Molecules sent to a distant node are mirrored back to the originating node. **(b)** An equivalent implementation in Fraglets where the remote species is not scheduled according to the law of mass action, but echoes itself back immediately; this is realized by executing a send transformation instruction in the remote node.



(a) Via Echo-Molecule **(b)** Via Fraglets Transformation

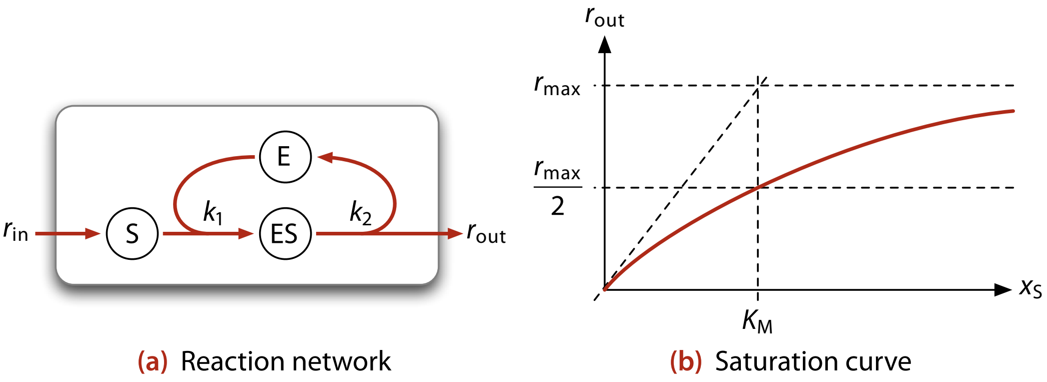
Figure 10.18 **Packet loss motif**: The path from source to destination loses packets with a rate p_1; the reverse path exhibits a packet loss probability of p_2. Some of the transmitted molecules are lost, such that the number of Y_1-molecules is lower than the number of X_1-molecules in proportion to the round-trip packet loss probability.



mirroring motif (Section 10.3.3(a)) copies the quantity signal with an error that is proportional to the packet loss probability. Figure 10.18 shows how this can be exploited. We assume that packets from the source node v_1 to the destination node v_2 are lost with probability p_1, whereas packets on the reverse path are lost with probability p_2. The packet rate arriving at the destination node is $r_2' = \left(1-p\right) x_{X_1}$. According to Kirchhoff's current law, the influx of species E_2 is equal to its efflux ($r_2 = r_1'$), from which a fraction is lost, such that the packet rate arriving back in the source node is $r_2' = \left(1-p_2\right) r_2$. Using Kirchhoff's current law again for species Y_1 yields

$$\hat{x}_{Y_1} = \left(1-p_1\right)\left(1-p_2\right)\hat{x}_{X_1} \tag{10.21}$$

This quantity signal actually reflects the packet yield probability. It can be used, for example, to down-regulate the transmission rate. Or, if required, the subtraction motif (Section 10.3.2(g)) can be used to compute the packet loss probability.



(a) Reaction network

(b) Saturation curve

Figure 10.19 Rate limitation motif: The total number of enzymes (E and ES together) limit the rate at which the substrate S is sent.

Rate Limitation Motif *(c)*

In chemistry, there is a well-known reaction motif that exhibits a maximum reaction rate: the enzymatic reaction discovered by Michaelis and Menten (1913) (see also the English translation by Teich, 1992) and formalized by Briggs and Haldane (1925). Here, we use a simplified model, the so-called Van Slyke-Cullen (VC) scheme (van Slyke & Cullen, 1914) depicted in Figure 10.19(a). Unlike in the Michaelis-Menten model, reaction r_1 is irreversible in the VC model.

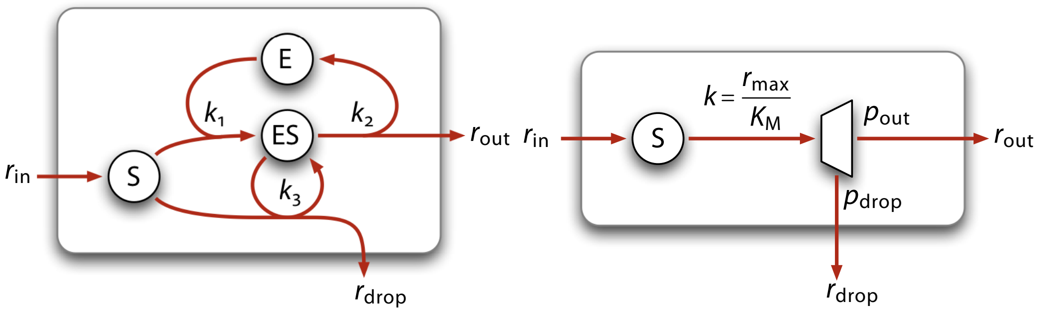
This chemical reaction motif limits the output flow rate r_{out}. This maximum rate is determined by the number of enzyme molecules present; they are either in a free form (E) or bound in a enzyme-substrate complex (ES). The output rate grows with the quantity of the substance S while more and more enzymes are bound (see Figure 10.19(b)). The rate is maximal if all enzymes are bound to a substrate molecule.

Michaelis-Menten equation

According to Kirchhoff's current rule, the influx equals to the efflux of the enzyme-substrate complex ES: $r_1 = r_2$ or $k_1 x_S x_E = k_2 x_{ES}$. We also note that the loop E–ES is a molecule conservation loop, and hence the total number of enzymes is constant: $x_E + x_{ES} = e_0 = \text{const.}$. By combining the two equations we obtain the *Michaelis-Menten equation*, which in the distributed context gives the transmission rate with respect to the substrate quantity:

$$r_{out} = r_{max} \frac{x_S}{K_M + x_S} \tag{10.22}$$

Figure 10.19(b) depicts the resulting hyperbolic saturation curve of the output rate with respect to the quantity of the substrate. The coefficient $K_M = k_2/k_1$ specifies the number of S-molecules at which half of the maximal rate $r_{max} = k_2 e_0$ is reached. Also note that for a low substrate quantity ($x_S \to 0$) the enzymatic reaction behaves like a unimolecular reaction with rate r_{max}/K_M whereas for a high substrate quantity ($x_S \to \infty$) the transmission rate asymptotically converges to the maximum rate r_{max}.



(a) Reaction Network

(b) Short Notation

Figure 10.20 Rate-limiting motif with excessive drop: By adding a third reaction, the total substrate outflow is equivalent to a first-order reaction; excessive substrate molecules are dropped.

The number of substrate molecules is usually not held at a constant value, but rather generated by a precursor reaction r_{in}, as indicated in Figure 10.19(a). At equilibrium, the output rate r_{out} will be equal to the input rate r_{in} as long as this rate does not exceed the saturation rate r_{max}. Let us denote the *load factor* as

load factor

$$\rho = \frac{r_{in}}{r_{max}} \tag{10.23}$$

The steady-state substrate concentration grows hyperbolically with the load factor and reaches infinity for $\rho \to 1$:

$$\hat{x}_S = K_M \frac{\rho}{1-\rho} \tag{10.24}$$

The enzymatic rate-limiting motif can be used whenever the packet rate shall be throttled to a certain rate, for example, to limit the transmission rate over a link with well-known bandwidth characteristics.

(d) Rate Limitation Motif With Excessive Drop

If the substrate is produced with a rate r_{in} that is higher than the saturation rate r_{max}, i.e. if the load factor is greater than one ($\rho > 1$), the number of substrates continuously increases. This can be avoided by the motif depicted in Figure 10.20(a), which extends the enzymatic reaction by an additional decay reaction r_3. At equilibrium, the substrate quantity settles at

$$\hat{x}_S = K_M \rho \tag{10.25}$$

With the decay reaction, the number of substrate molecules grows linearly instead of hyperbolically with respect to the load factor.

The total efflux rate (r_{out} and r_{drop} together) corresponds to the rate

$$r_{\text{out}} + r_{\text{drop}} = \frac{r_{\max}}{K_{\text{M}}} x_{\text{S}} \tag{10.26}$$

which is a linear function with respect to the number of substrate molecules. Hence, we can regard the two substrate-consuming reactions together as a single unimolecular reaction with coefficient $k = r_{\max}/K_{\text{M}}$ as depicted in Figure 10.20(b). One part of the molecules drained from S is sent while the remaining part is dropped. According to the saturation curve of the enzymatic reaction, the transmission probability is

$$p_{\text{out}} = \frac{1}{1+\rho} \tag{10.27}$$

whereas the drop probability is

$$p_{\text{drop}} = \frac{\rho}{1+\rho} \tag{10.28}$$

Like the previous motif (Section 10.3.3(c)), this extended motif may be used to limit the transmission rate over a link with well-known bandwidth characteristics. If substrate molecules arrive at a higher rate than this bandwidth, this motif should be preferred over the previous one. We will discuss the similarity between the enzymatic reaction and a FIFO-ordered packet queue later in Chapter 12.

NETWORK NEIGHBORHOOD DISCOVERY MOTIFS 10.3.4

The transmission motifs seen so far manage packet communication between two nodes. In this subsection, we discuss motifs that inspect the network neighborhood, for example, in order to determine the number of neighbor nodes.

Motif to Determine the Number of Neighbor Nodes (a)

If all network nodes *broadcast* a packet stream with a well-known packet rate, each node receives a cumulative packet stream with a rate proportional to the number of neighbors. This simple fact is exploited to determine the number of neighbor nodes as depicted in Figure 10.21. We modified the quantity-mirroring motif (Section 10.3.2(b)) by using *broadcast* instead of unicast transmission primitives, and the result is summed up as in the linear combination motif (Section 10.3.2(c)). The resulting steady-state concentration of Y_i is

broadcast

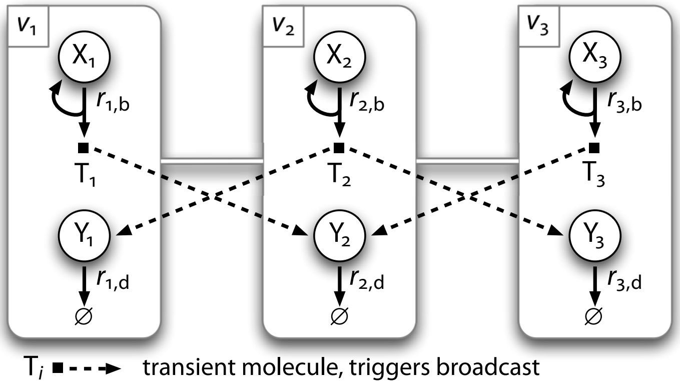


Figure 10.21 Neighbor quantification motif: This reaction network computes the number of neighbor nodes by summing up in species Y_i of each node v_i the rate of packets received from this node's neighbors.

$$\hat{x}_{Y_i} = \sum_{j \in \mathcal{N}_i} \hat{x}_{X_j} \qquad \forall i \in \mathcal{V} \tag{10.29}$$

Given that all nodes have an identical number of X-molecules, $\hat{x}_X$, the concentration of Y in node v_i is equal to the number of v_i's neighbors ($\mathcal{N}_i$) times the number of X-molecules.

$$\hat{x}_{Y_i} = \mathcal{N}_i \hat{x}_X \qquad \forall i \in \mathcal{V} \tag{10.30}$$

Note that this motif can only be used if all nodes run the same preinstalled program that implements the two reactions.

(b) *Motif to Determine the Number of Neighbor Nodes (Alternative Fraglets Implementation)*

The alternative implementation depicted in Figure 10.22 does not require the neighbor nodes to run preinstalled software. It makes use of Fraglets' active networking capability through which code can be sent along with the transmitted packet. The alternative motif is based on the echo motif discussed before (Section 10.3.3(b)), but instead of broadcasting the Y-molecules to all neighbor nodes, each node broadcasts an *echo fraglet* (`[send` v_2 `Y]`), which immediately sends back a Y-molecule to the source node. There, the Y-molecules are decayed in order to obtain a steady-state quantity that reflects the inflow of Y-molecules. This inflow is k times the original broadcast rate, where k is the number of neighbor nodes. Thus, a node actually probes how many Fraglets vessels are in its neighborhood.

echo fraglet

(c) *Motif to Determine the Number of Connected Neighbor Nodes*

The above motif can even be pushed further by broadcasting a packet twice in sequence before sending the reply back to the originator node, as shown in

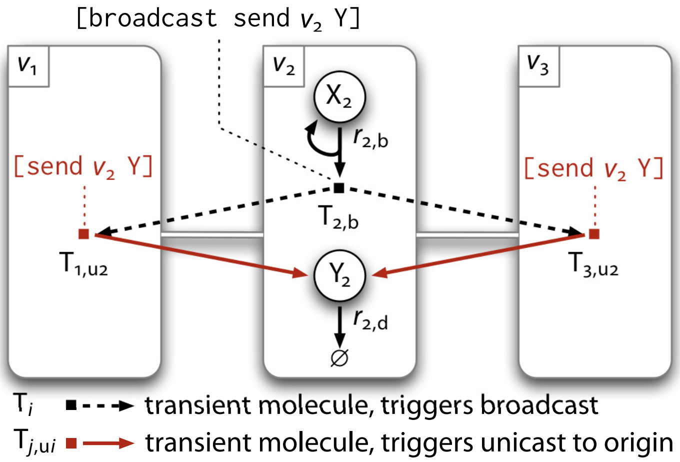


Figure 10.22 Neighbor quantification motif (2): This alternative implementation works despite there is no preinstalled program in the neighbor nodes. An *echo fraglet* is sent to all neighbors; the sum of the rate of all returning fraglets is proportional to the number of neighbor nodes.

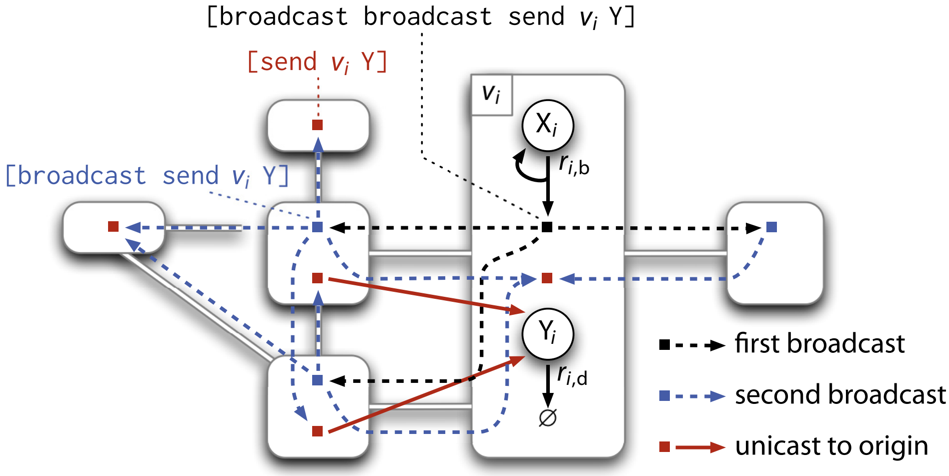


Figure 10.23 Connected neighbor quantification motif: The originator node v_i broadcasts a fraglet, which broadcast a unicast message back to the originator. The quantity of Y is proportional to the number of connected neighbors of node v_i.

Figure 10.23. The originator node v_i sends a broadcast message to all neighbors. This message contains a fraglet that broadcasts itself to the neighbor's neighbors. There, yet another fraglet unfolds that tries to send an indicator molecule Y back to the originating node. This last message only returns to the originating node if the packet traversed two directly connected neighbors, and thus, the number of Y-molecules reflects the number of connected neighbor nodes:

$$\hat{x}_{Y_i} = \hat{x}_{X_i} \left|\left\{\left(j,k\right)\right\}\right| \qquad \forall i \in \mathcal{V}, v_j, v_k \in \mathcal{N}_i, \left(j,k\right) \in \mathcal{E} \qquad (10.31)$$

If there are no Y-molecules, all neighbors are *independent*. independent

Anycast Motif *(d)*

Fraglets provides an anycast-instruction, which sends the fraglet's tail to a randomly chosen neighbor node. However, the anycast-instruction only

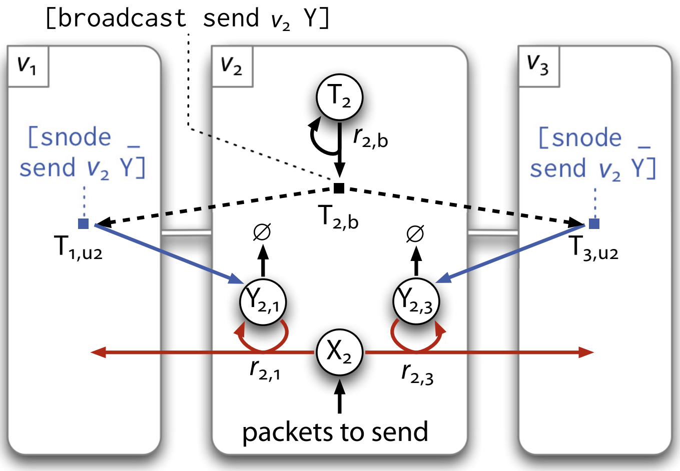


Figure 10.24 Anycast motif: The originator node v_i broadcasts a fraglet, which appends the local node identifier to the fraglet and unicasts itself back to the original node. There the fraglet competes with similar fraglets returning from other neighbors for reacting with data packets X and drag them to the corresponding neighbor.

works if the underlying transmission channel provides information about the identities of the neighbor nodes. The following motif spans a reaction network that simulates anycast transmission using broadcast and unicast transmission primitives only.

Figure 10.24 illustrates the spanned reaction network: Each node v_i maintains a population of trigger molecules T_i that determines the pace of sending broadcast messages. Once arrived in a neighbor node v_j, the broadcasted fraglet inspects the identifier of the local node by executing the snode-instruction and sends itself back to the originating node v_i. The returned fraglet has the format $Y_{i,j}$ = [Y v_j]. With the help of an assisted reaction, such a $Y_{i,j}$-molecule reacts with a data packet X_i and sends it to the neighbor node v_j from which the broadcasted fraglet returned.

Note that the reaction network from T via the neighbors to the corresponding Y-molecules and their decay, is an application of the remote quantity-mirroring motif (Section 10.3.3(a)). That is, the concentrations of all $Y_{i,j}$ $\left(j \in \mathcal{N}_i\right)$ are equal, and hence the probability of a data packet being sent to either node is equal. As a further consequence, the rate at which the broadcast is sent, which is controlled by the quantity of T-molecules, also determines how fast data packets X are sent.

10.4 APPLICATION CASE: DISPERSER IN A NETWORK WITH UNKNOWN TOPOLOGY

In this section, we demonstrate an extension of the *Disperser* protocol to illustrate how the various motifs introduced in this chapter can be combined. We introduced the *Disperser* protocol in Section 5.4 and analyzed its behavior in Chapter 9; for reference, see Figure 10.25.

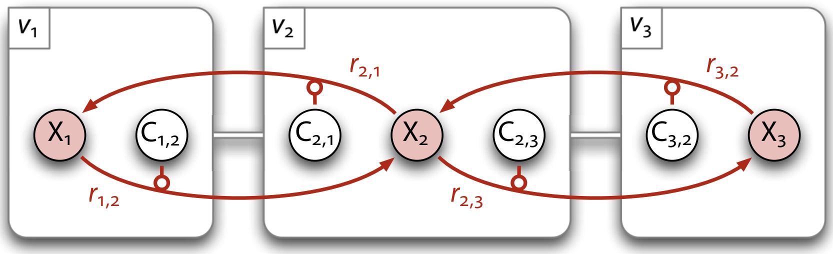


Figure 10.25 Original reaction network of *Disperser*: This protocol design is problematic, because the control molecules $C_{i,j}$ must be pre-installed manually; there must be one control molecule per link $(i,j) \in \mathcal{E}$.

The original design of the protocol is quite static: A control molecule $C_{i,j}$ must be manually installed for each network link $\left(i, j\right) \in \mathcal{E}$. That is, there must be an external process – either human or automatic, but out of the scope of the chemical execution model – that knows the node's neighborhood and creates or removes the control molecules as the network topology changes.

In this section, we aim at improving *Disperser*'s design such that the protocol autonomically discovers its network neighborhood and installs the required control molecules. We propose two different design variants in Sections 10.4.1 and 10.4.2 and compare their behavior.

FIRST DESIGN VARIANT 10.4.1

We follow our design paradigm by first finding the chemical reaction network that solves the problem dynamically and, in a second step, to look for an implementation of the resulting reaction network in Fraglets.

First Step: Reaction Network Design (a)

The first design variant relies on the remote mirroring motif (Section 10.3.3(a)) as depicted in Figure 10.26. Each node v_i broadcasts control molecules to its neighbors. There these $C_{j,i}$-molecules drag X_j-molecules to the originator of the broadcast. This implements the actual exchange of data-molecules as indicated by the red, curved arrows. In order to maintain a constant population of control-molecules, a first-order reaction decays them. The reactions highlighted with the blue background are an instantiation of the remote mirroring motif, according to which the number of control molecules $C_{j,i}$ is equal to the number of trigger molecules in the originating node, T_i. If all nodes start with the same number of triggers, the draining force for each link is equal, and *Disperser* computes the average number of X-molecules as required.

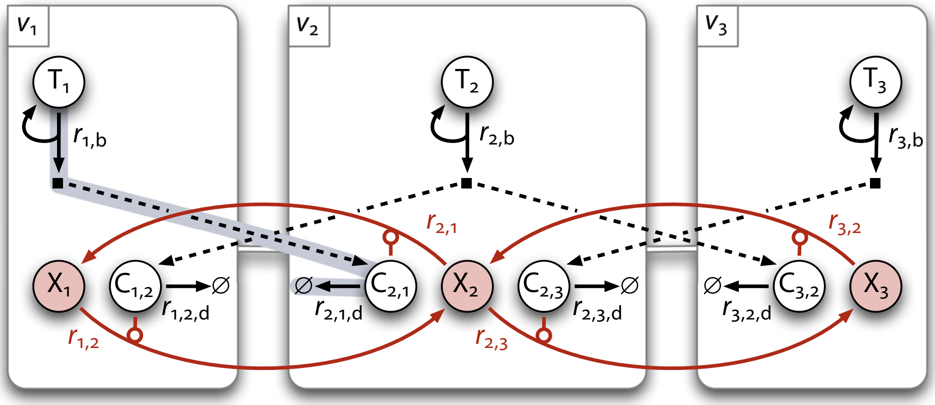


Figure 10.26 Reaction network of the extended *Disperser*: Each node broadcasts control molecules to its neighbors. There, the enzyme is used as a catalyst to send X-molecules back to the originating node. An additional decay reaction makes sure the control molecules are kept a constant number.

(b) Second Step: Fraglets Implementation

The next step is to come up with an implementation of this distributed reaction network in Fraglets. Ideally, we would map each species to a fraglet string. But, because each Fraglets reaction requires an active and a passive string, we have to resort to assisted reactions (see Section 10.3.1(d)), where reactions rather than molecules are represented by fraglet strings. In particular, the following conflict arises: Control species $C_{i,j}$ is consumed by the first-order decay reaction $r_{i,j,\mathrm{d}}$, suggesting that the control species is represented by an active `[mmatch 0]`-fraglet (see also Section 10.3.1(c)). But at the same time, the same control species must react with X_i-molecules; a dual matching tag would be required, which is unfortunately not possible in Fraglets yet. Thus, we are forced to fall back to assisted reactions as shown below.

Each node $v_i \in \mathcal{V}$ is initialized with the following set of fraglets:

$$T_i\colon \quad {}_{v_i}\texttt{[mmatchp 0 broadcast C } v_i\texttt{]}t_0 \tag{10.32a}$$

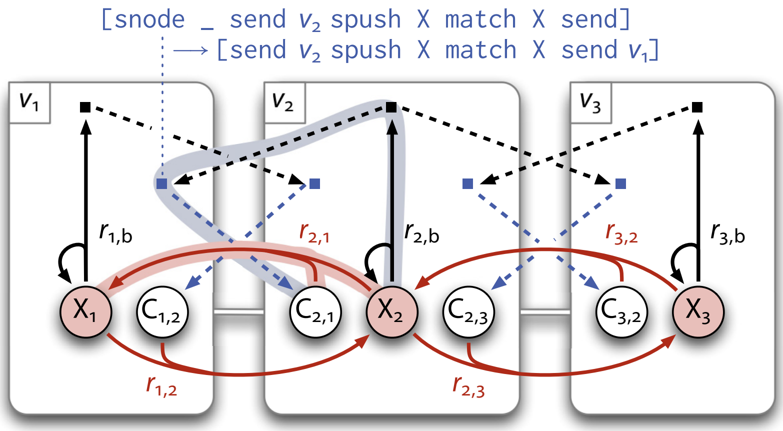
$$r_{i,j,\mathrm{d}}\colon \quad {}_{v_i}\texttt{[matchp C nul]} \tag{10.32b}$$

$$r_{i,j}\colon \quad {}_{v_i}\texttt{[mmatchp 2 C X mfork 3 spush X send nop nop C]} \tag{10.32c}$$

$$X_i\colon \quad {}_{v_i}\texttt{[X]}x_{i,0} \tag{10.32d}$$

The strings representing the efflux reactions of X, (10.32b) and (10.32c), are only present once, whereas the quantity of X_i, $x_{i,0}$, represents the node's initial value. The number of trigger molecules T_i, t_0, must be identical in each node. The latter quantity signal is mirrored to the quantity of the neighbor's control molecules $C_{j,i}$.

As a remainder from Section 9.3 we note that the SNR increases for a high molecular quantity. We expect and show later that for lower values of t_0 the calculated result becomes more noisy.



[snode _ send v_2 spush X match X send]
⟶ [send v_2 spush X match X send v_1]

Figure 10.27 Alternative reaction network of the extended *Disperser*: Each node broadcast echo fraglets with a rate equal to the current number of X-molecules. The returned echo is a control molecule that sends X-molecules to the corresponding neighbor and gets consumed. According to the constant quantity motif (Section 10.3.2(h)), the average number of control molecules is kept constant at one.

SECOND DESIGN VARIANT 10.4.2

The *Disperser* protocol only works if the number of control molecules is equal for each link in the network. The first design variant therefore copied a well-known quantity via broadcasts to ensure this equality. The second design variant presented here makes use of the constant concentration motif (Section 10.3.2(h)) instead. The resulting distributed reaction network is depicted in Figure 10.27.

First Step: Reaction Network Design (a)

Unlike in the previous variant, each network node now installs its *own* control molecules wherefore we modified the neighbor quantification motif Section 10.3.4(b) such that the echoed molecules received from different neighbors are treated as separate species – the control molecules $C_{i,j}$. Consequently, the reaction network yields a constant number of one control molecule per link, which is a consequence of the fact that the control molecule is produced by a unimolecular reaction from another molecule X_i (highlighted in blue) and consumed in a bimolecular reaction with the same other molecule (highlighted in red), as we discussed in Section 10.3.2(h).

Second Step: Fraglets Implementation (b)

The following initial set of Fraglets is installed in each node v_i:

$$r_{i,\mathrm{b}}: \quad {}_{v_i}[\texttt{matchps X broadcast} \\ \quad \texttt{snode \_ send } v_i \texttt{ spush X match X send}] \tag{10.33a}$$

$$\mathrm{X}_i: \quad {}_{v_i}[\texttt{X}]x_{i,0} \tag{10.33b}$$

The reaction network unfolds from these two fraglets. They react with each other and finally result in the production of the control molecules through the following steps: First, the reaction product broadcasts itself to all neighbors:

$$
\begin{align}
& {}_{v_i}[\texttt{matchps X broadcast} \nonumber\\
& \quad \texttt{snode \_ send } v_i \texttt{ spush X match X send}] + {}_{v_i}[\texttt{X}] \nonumber\\
\longrightarrow\ & {}_{v_i}[\texttt{broadcast} \tag{10.34a}\\
& \quad \texttt{snode \_ send } v_i \texttt{ spush X match X send}] + \ldots \nonumber\\
\longrightarrow\ & {}_{v_j \in \mathcal{N}_i}[\texttt{snode \_ send } v_i \texttt{ spush X match X send}] \tag{10.34b}
\end{align}
$$

Every neighbor $v_j \in \mathcal{N}_i$ executes the received string, which appends the local node identifier to its tail and sends itself back to the originating node:

$$
\begin{align}
& {}_{v_j \in \mathcal{N}_i}[\texttt{snode \_ send } v_i \texttt{ spush X match X send}] \nonumber\\
\longrightarrow\ & {}_{v_j \in \mathcal{N}_i}[\texttt{send } v_i \texttt{ spush X match X send } v_j] \tag{10.34c}\\
\longrightarrow\ & {}_{v_i}[\texttt{spush X match X send } v_j] \qquad \forall v_j \in \mathcal{N}_i \tag{10.34d}
\end{align}
$$

The third step consists of restructuring the returned molecule such that it becomes the control molecule:

$$
\begin{align}
& {}_{v_i}[\texttt{spush X match X send } v_j] \nonumber\\
\longrightarrow\ & {}_{v_i}[\texttt{match X send } v_j \texttt{ X}] \qquad \forall v_j \in \mathcal{N}_i \tag{10.34e}
\end{align}
$$

It reacts with a passive [X]-fraglet and sends it to the neighbor over which the control fraglet was echoed.

10.4.3 SIMULATION RESULTS

Both design variants replace the persistent control molecules in the original design by a flow of dynamically generated control molecules. Each of the additional reactions potentially increases the noise, especially for low molecular concentrations. To illustrate this problem, we empirically measured the SNR of the computed result at equilibrium for the original *Disperser* design and the two variants. For this purpose, we ran an OMNET++ simulation for the three-node topology depicted in Figures 10.25, 10.26, and 10.27, for a steady state result of $\hat{x}_\mathrm{X} = 100$ molecules.

Figure 10.28 shows that the SNR is about 12.2 for the original design variant. For a steady-state result of 100 molecules this means a standard

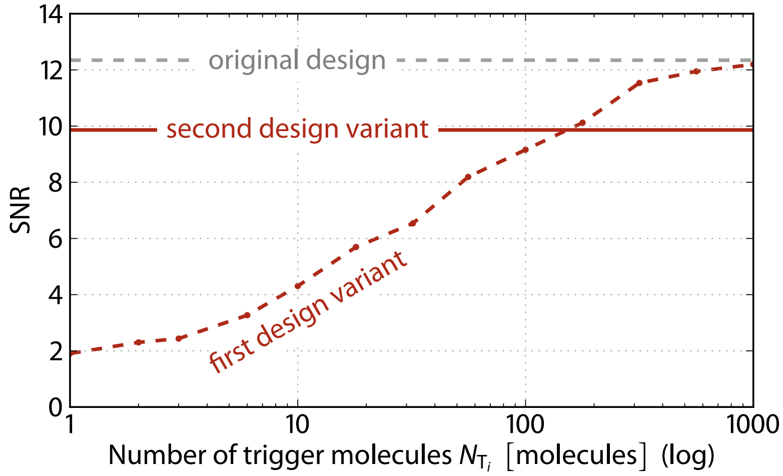


Figure 10.28 Signal-to-noise ratio of different *Disperser* design variants: Empirically obtained results from a Fraglets simulation of the three-node network topology depicted in Figures 10.25, 10.26, and 10.27, for a steady-state result of $\hat{x}_X = 100$ molecules. The original version is most accurate. The first design variant mirrors trigger molecules to neighbor nodes; its accuracy increases for a larger number of trigger molecules. The second design variant uses the constant quantity motif. Its accuracy is independent on an additional parameter.

deviation of 8.1 molecules. The figure also shows that the SNR for the first design variant using the remote mirroring motif increases for a higher number of trigger molecules. Note that the noise of the trigger molecule (and hence of the copied control molecules) is actually added on top of the original variant's noise where the number of control molecules is held constant. Finally, the second design variant, which is based on the constant quantity motif, exhibits a slightly lower accuracy than the original design.

We favor the second design variant, because it automatically adapts to network topology changes, and, compared to the first variant, it does not require all nodes to run preinstalled software.

SUMMARY 10.5

In the previous section, we enhanced the design of our chemical gossiping protocol *Disperser* such that it discovers its neighbor nodes and installs the corresponding control molecules autonomously. The design process highlighted that the motifs presented earlier in this chapter are welcome building blocks that can be combined and inserted into an existing chemical reaction network. In a second step, we translated the abstract reaction network to an implementation in Fraglets. We demonstrated that even though Fraglets is not chemically complete there exist work-arounds that allow us to implement any reasonable reaction network.

The motifs presented in this chapter heavily exploit the dynamics of chemical reaction networks. The result is not presented symbolically in a single molecule instance but manifests as a steady-state quantity or reaction rate. We demonstrated the well known result that chemical reaction networks are able to process such *analog* signals, for example by calculating arithmetic functions, and we applied this result to CNPs. Such computation can now be organized in a distributed environment. Beyond mere arithmetic calculations, our reaction networks use concentrations signals to control the protocol's dynamics. In Chapter 12, we will introduce a chemical networking protocol that automatically adapts to the available bandwidth and even provides fairness among different packet streams akin TCP's congestion control mechanism.

It became apparent that for a given problem, multiple design variants exist in the solution space. It is therefore impossible to provide a general recipe that generates the "right" result. We rather provided general solutions for very basic problems and leave it to the engineer to combine them in an innovative way in order to solve a more complex task.

CHAPTER 11

CNP Simulation and Runtime Engine

A more detailed description of the Fraglets Virtual Machine, its embedding into a network simulator, and on the limiting effects of real computing infrastructure.

> Time is not bought ready-made at the watchmaker's. [11]
>
> *The Habit of Truth*
> Jacob Bronowski

Our chemical execution model introduced in Chapter 5 left several implementation details unspecified. In this chapter, we discuss the architecture of the current Fraglets Virtual Machine implementation and its integration into the OMNET++ network simulator in more detail. We also address physical constraints; in particular, we examine the consequences of limited memory and limited computing power for chemical software. Thus, this chapter is mainly addressed to computer system architects, to whom it provides a guideline for implementing a chemical virtual machine. In the next chapter, we will then focus on restrictions in the networking infrastructure, i.e. limited bandwidth and packet loss. Figure 11.1 shows where these topics are located in our chemical engineering landscape.

This chapter is structured as follows: Section 11.1 describes architectural details of the Fraglets Virtual Machine. In Section 11.2, we show how the virtual machine is integrated into the OMNET++ simulator. Section 11.3 then discusses additional memory constraints and time-synchronization problems if the virtual machine is embedded into a real network and operates on-line.

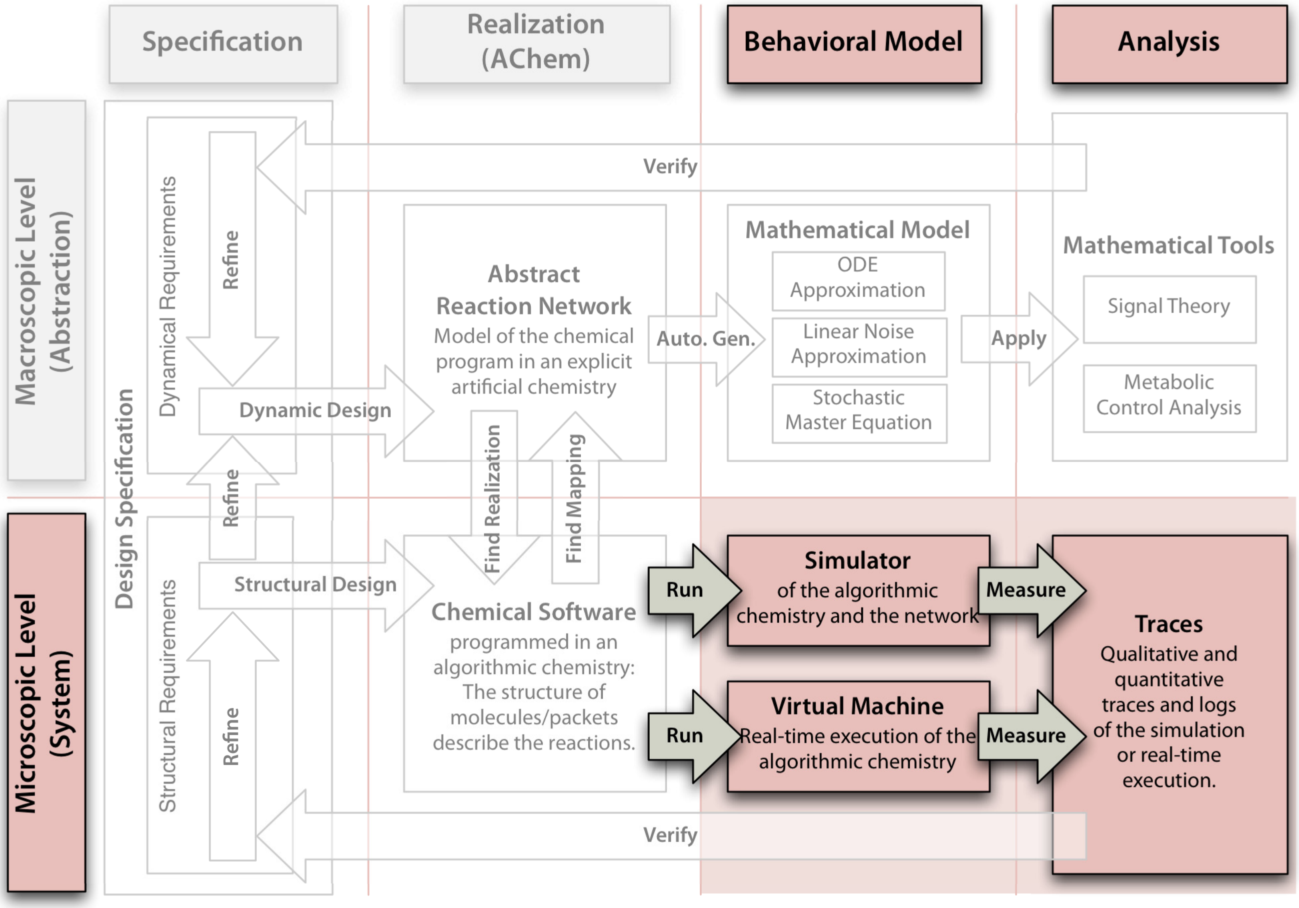

Specification
Realization (AChem)
Behavioral Model
Analysis
Macroscopic Level (Abstraction)
Microscopic Level (System)
Design Specification
Dynamical Requirements
Structural Requirements
Refine
Refine
Refine
Verify
Dynamic Design
Structural Design
Abstract Reaction Network
Model of the chemical program in an explicit artificial chemistry
Find Realization
Find Mapping
Chemical Software
programmed in an algorithmic chemistry: The structure of molecules/packets describe the reactions.
Auto. Gen.
Mathematical Model
ODE Approximation
Linear Noise Approximation
Stochastic Master Equation
Apply
Mathematical Tools
Signal Theory
Metabolic Control Analysis
Run
Run
Simulator
of the algorithmic chemistry and the network
Virtual Machine
Real-time execution of the algorithmic chemistry
Measure
Measure
Traces
Qualitative and quantitative traces and logs of the simulation or real-time execution.
Verify

Figure 11.1 **Implementation issues in the engineering model**: This chapter focuses on the implementation of the Fraglets Virtual Machine and its integration into the OMNeT++ network simulator.

11.1 VIRTUAL MACHINE (VM) ARCHITECTURE

The Fraglets Virtual Machine (VM) is the core of our Fraglets execution environment. It is implemented as a C++ library that is linkable to any program running on a traditional computer with a POSIX-compliant operating system interface (Eissfeldt, 1997). Currently, the VM is embedded into two programs: a standalone executable, which parses a Fraglets program file and executes it in turn, and a plug-in library for the OMNET++ network simulator.

In this work, we use Fraglets to implement network services. As shown in Figure 11.2, this positions the virtual machine between an application layer program and the existing network infrastructure; the VM offers network services in the form of chemical protocols to the application.

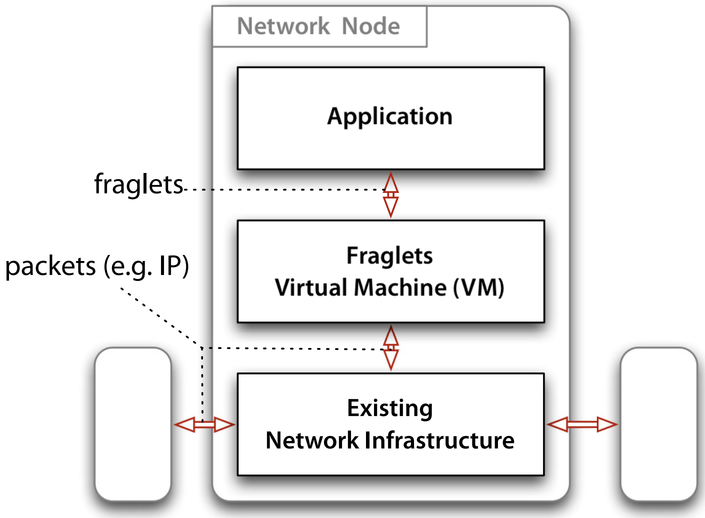


Figure 11.2 Conceptual Fraglets network node: Each network node consist of an application and an existing network infrastructure such as a simulated IP network. In between, the Fraglets Virtual Machine implements the network protocols.

INTERNAL STRUCTURE OF THE FRAGLETS VM 11.1.1

The internal structure of the VM-implementation is depicted in Figure 11.3 and described below:

Multiset Observer: The multiset observer is notified by the multiset whenever a fraglet is extracted or injected into the "soup". The observer maintains a *hash-table* for all currently active reaction channels. In Fraglets, there is one reaction channel per tag symbol (X, Y, Z, etc.). The observer computes the propensity of each reaction channel according to (5.4) on page 56, determines the reaction's next occurrence time and (re-)inserts a reaction event into the priority queue.

hash-table

Reaction Priority Queue: This queue contains one event for each reaction channel, ordered with respect to the planned occurrence time. A reaction event is relocated within the queue whenever reactant molecules are added or removed to or from the multiset, according to the updated propensity function.

Reaction Arbiter: The arbiter places the earliest reaction event to an external *event scheduler*. The scheduler notifies the arbiter as soon as the next reaction shall be executed, which causes the arbiter to trigger the Fraglets processor.

event scheduler

Fraglets Processor: The Fraglets processor implements the Fraglets language. It selects and extracts reactants from the multiset and *executes the reaction* according to the Fraglets production rules (see Appendix B). The product strings are either placed back into the multiset, or, if some product strings

executes the reaction

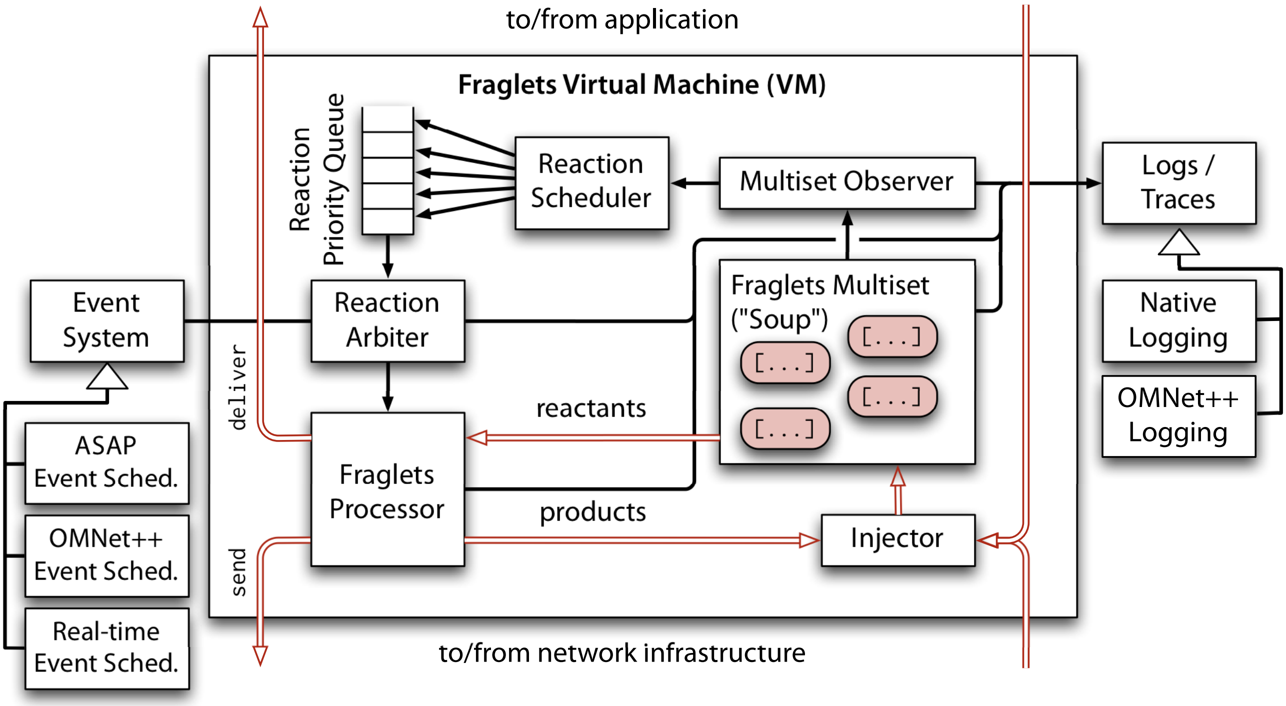


Figure 11.3 Fraglets Virtual Machine (VM): The observer continuously tracks changes in the multiset and updates the reaction priority queue. The reaction arbiter is integrated into the simulation environment's event scheduler and triggers the Fraglets Processor for the next reaction. The processor applies rewriting rules by selecting and extracting reactants from the multiset, by executing the fraglets reaction and subsequent transformation steps, and by injecting the products back to the soup, or sending them to the application or over the network.

start with the deliver- or the send-instruction, these strings are sent to the application or to the network infrastructure, respectively.

Injector: The injector module is responsible for adding new molecules to the multiset. It is called likewise for locally produced and received molecules.

11.1.2 DESIGN OBJECTIVES OF THE FRAGLETS VM

The current VM was designed with flexibility and not performance in mind. It is more an engineering instrument than a productive system. Nevertheless, we integrated an efficient memory management system, which uses smart pointers and a copy-on-write strategy to avoid unnecessary copy operations. On the other hand, the software is very modular, which is useful during development and debugging, but which decreases the overall performance of the VM. In the following, we discuss two modular interfaces in more detail: the logging infrastructure and the event system. They are responsible for the seamless integration of the Fraglets VM into other programs such as the OMNET++ simulator.

Logging Subsystem *(a)*

As shown on the right of Figure 11.3, each component of the VM (the multiset, the Fraglets processor, the reaction arbiter, etc.) provides a logging interface to which observers may be attached. Loggers dynamically register with these components and obtain state updates in turn. Each program, into which the VM is embedded, will usually provide its own loggers. Our stand-alone executable, for example, writes state information to a file in CSV (comma-separated value) format whereas the OMNET++ plug-in forwards them to the logging infrastructure of the simulator. The following description lists some of the loggers available in the current implementation:

State Logger: The state logger writes – on the occurrence of an event – a snapshot of the multiset to a file. Typically, this happens at the end of a simulation run to extract the computed result of a chemical program.

Multiset Inspection: This logger continuously tracks multiset information and writes it into a CSV file. Such information includes the quantity of species, the length distribution of fraglet strings, statistics about a numeric value at a certain symbol position in a fraglet string, etc. Many graphs in this thesis were drawn with Matplotlib (Hunter, 2007)[*] based on such inspection files.

Reaction Traces: The trace logger prints a textual representation of each executed reaction like the trace shown by (10.34b)–(10.34e) on page 182, for example.

Reaction Network Graphs: The grapher module tracks all executed reactions and internally builds a reaction network graph, which is printed in the *dot* file format. Graphviz (Ellson, Gansner, Koutsofios, North, & Woodhull, 2002)[†] parses this file and draws the reaction graph.

Timers: The timer module is used to inject one or more fraglets at a certain simulation time.

As all loggers use the same interfaces, it is easy for programmers to write their own loggers and integrate them into the VM.

[*] The Matplotlib python library is freely available form the "Matplotlib Web Page" (2010).

[†] The Graphviz software is available from the "Graphviz Web Page" (2010).

Event Subsystem *(b)*

The event subsystem is responsible for scheduling reaction events and executing reactions on time. As depicted in Figure 11.3, the Reaction Arbiter places an event to an external scheduler via the VM's abstract event system interface. Thus, the VM delegates the exact timing of its reactions.

There are different implementations of the event scheduler, one for each program into which the VM is embedded: Our stand-alone Fraglets interpreter provides an ASAP-scheduler, which does not defer the reaction but calls the reaction arbiter immediately. As a consequence, reactions are executed back-to-back and the simulated time diverges from the physical time, which in a simulation is not a problem.

There is another scheduler for the OMNET++ integration. It converts our reaction events into OMNET++ events and schedules them using OMNeT's native event scheduler. This automatically synchronizes Fraglets' notion of time to OMNeT's simulation time.

A third scheduler would have to be provided if Fraglets is used in a productive system where the VM's are distributed and driven by separate CPUs. We will address such timing issues later in Section 11.3.2, where we discuss computing power limitations.

11.2 INTEGRATION OF THE FRAGLETS VM TO THE OMNET++ SIMULATOR

During the development and design phase of chemical protocols, controlled protocol simulations are preferred over extensive empirical experiments in network test beds or productive networks. Such simulations, carried out in the OMNET++ framework, complement our analytical studies and provide a convenient rapid prototyping method to incrementally refine the protocols when needed (see Figure 11.1).

OMNET++ (Györgi, 1993; Varga, 2001; Varga & Hornig, 2008) is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. It features a set of ready-made simulation models including INET, an Internet protocol suite containing implementations of IP, TCP, UDP, PPP, Ethernet, etc., and a mobility framework for the simulation of wireless and mobile networks.

components

In OMNET++, a network is designed by combining *components*, i.e. abstract wrappers of functionality. There are two types of components: *simple components* are C++ plug-ins providing the actual functionality whereas *compound components* are used to group other components together to a manageable entity of its own, e.g. network nodes. Both types communicate by exchanging message objects through so called *gates*, which are linked by *connectors*. Each time a simple module sends a packet over one of its gates, the framework routes the packet along the connectors and calls the event handler of the destination module. The OMNET++ framework provides a

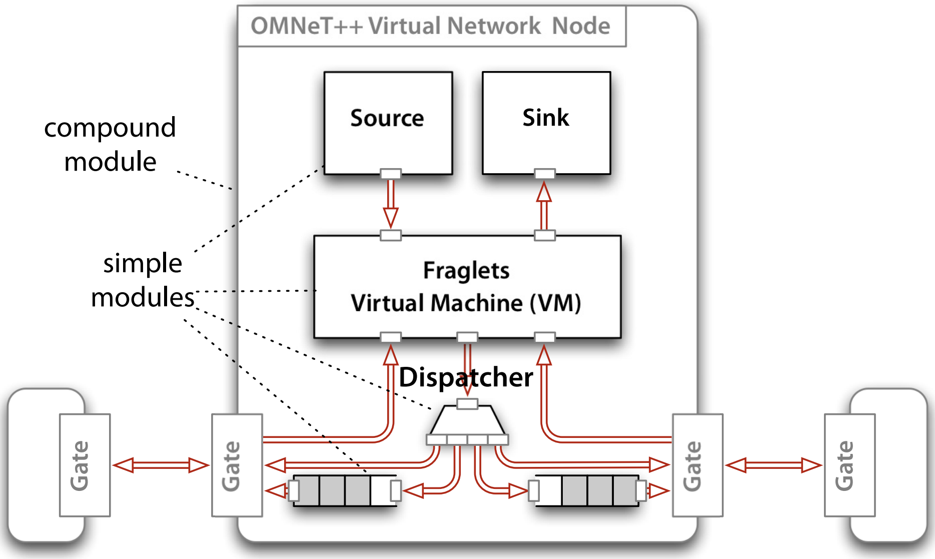


Figure 11.4 **Embedding of Fraglets into OMNeT++:** Each virtual network node instantiates a Fraglets virtual machine. User-data is generated by a source module whereas the sink module accepts data destined for this node. The Dispatcher sends fraglet strings to one or multiple neighbors based on the identifier following the send-instruction.

central event scheduler for all virtual network nodes and hence assures a consistent simulation time.

Our integration is based on previous work by Lüscher (2009): A virtual network node running Fraglets is represented by a compound module, depicted in Figure 11.4. A node has a unique name, serving as its address. It contains a number of simple modules, implementing the actual functionality. Each node contains an own instance of the *Fraglets VM*. That is, our Fraglets plug-in, written in C++, provides an OMNET++-wrapper for the Fraglets VM.

Fraglets VM

A *source* and a *sink module* is connected to the VM. They simulate application requests and are configured to terminate a stream of Fraglet molecules sent from one node to another node. In Chapter 12, we occasionally replace them by TCP sockets.

source, sink

Whenever the Fraglets VM executes a send-instruction, it passes the fraglet to the *Dispatcher* component, which locates the neighbor node to which the Fraglet is addressed, and dispatches the fraglet as an OMNET++ message to one of the node's gates. The Dispatcher can be configured to either send the packet directly over the attached connector or to insert it into a FIFO queue. The use of FIFO queues is discussed in detail in the next chapter.

Dispatcher

As mentioned before, *synchronization* of the Fraglets reaction time to the OMNET++ simulation time is straightforward: Our plug-in implements a wrapper to the OMNET++ event system; reactions are actually scheduled by the network simulator and use the same notion of time as other simulated elements such as queues, protocol timers, etc.

synchronization

11.3 INTEGRATION OF THE FRAGLETS VM TO A REAL NETWORK

In this section, we study the effects of using Fraglets in a productive system, i.e. in a real distributed network environment. If the Fraglets VM goes on-line, physical constraints limit several aspects of the ideal chemical model and thus deserve some more attention. We address two particular limitations, namely limited memory in Section 11.3.1 and limited computing power in Section 11.3.2: Finite memory resources limit the maximum number of molecules in a vessel and limited computing power implies a maximum reaction rate. We show several strategies of how to cope with such constraints during the design phase and at run-time.

11.3.1 MEMORY LIMITATIONS

Each computer comes with limited memory and consequently, the representable state-space is finite. For an artificial chemistry such as Fraglets this implies that only a finite number of molecules can be stored in the multiset. We first discuss two different ways of storing a multiset in memory and discuss their strengths and weaknesses, before we present strategies to cope with limited memory.

(a) *Storage Models for the Multiset*

There are at least two methods how a multiset of molecules can be stored in memory: (1) The simplest method is to store each molecule *instance* separately in its own memory region. This requires the VM to allocate memory in proportion to the total number of symbols in the multiset, i.e. the number of molecule instances times their lengths. (2) A second method is to store each molecule *species* in a separate memory region together with a counter that remembers the number of its instances.

memory efficiency

The second method is more *memory efficient* because it only requires memory in proportion to the number of species times their lengths plus some memory for the instance counters. However, this comes along with an increased *computational overhead* for the insert operation, where the multiset has to find the corresponding species before incrementing its multiplicity.

computational overhead

Another drawback of the second method is the high vulnerability to spontaneous memory alterations. We study this phenomenon in detail in Part III. Here, we just note that a flipped bit changes a single instance in the first method but changes all instances of a species in the second method.

Thus, in terms of robustness, it is beneficial to exploit the memory by storing different instances of the same species in separate memory regions.

Our current VM implementation resorts to the second method and stores the fraglets in a hash-table. Each fraglet provides an efficient hashing function, which allows the multiset to quickly locate the species. However in the following, we anticipate results from Part III and assume that instances are stored separately. We discuss the consequence of limited memory and develop strategies cope with this constraint.

Strategies to Cope with a Limited Vessel Capacity (b)

Imagine a VM with limited memory, such that not more than a certain number of symbols can be stored. We implicitly assume that each symbol is encoded by a fixed length binary code; a possible encoding scheme is discussed later in Chapter 18. We also ignore memory fragmentation and assume that all memory can be exploited to store fraglets. The question we would like to answer here is: What happens if new molecules are produced or injected while the memory is full? We currently envision two possible strategies:

Tail-Drop: An obvious method is to throw away new products if the vessel is saturated. This mimics the tail-drop behavior of limited FIFO queues.

Random Dilution: Another strategy is to let a new molecule instance displace one or more randomly selected molecules in the vessel. Consequently, each species is subject to a dilution flux, imposed by the overall productivity of the reaction system.

Both methods have their strengths and weaknesses. The *tail-drop* method does not overwrite existing molecules and thus, does not threaten the integrity of chemical software. However, there are cases where tail-drop behavior leads to a *stalled system*: Imagine a situation in which the vessel is full but the reaction system is inert, for example, waiting for input. A molecule received from the application layer or from a distant vessel may reactivate the local reaction system. However, the tail-drop behavior does not allow these molecules to enter the vessel – the system remains inert forever.

tail-drop

stalled system

By applying a *random dilution flux*, the system does not run into this problem, because it removes molecules to make room for new ones. Note however, that any type of molecule is subject to dilution. Although the protocol may tolerate the loss of molecules containing user-information in form of payload, the dilution flux will also remove molecules comprising the essential code of the chemical software. If catalysts (for example persistent `matchp`-fraglets) are removed, the functionality of the software is disrupted.

random dilution flux

One method to protect software is to artificially protect persistent fraglets against the dilution flux. This is fine if static software is installed permanently, but does not work if code is dynamic and mobile. Another method is to antagonize the dilution flux by providing multiple redundant code replicas. However, all replicas will eventually be diluted. The only method to protect dynamic code from the dilution flux is to come up with software that continuously rewrites itself and regulates its own redundancy. We will propose such an approach in Part III.

(c) Mathematical Description of the Dilution Flux Dynamics

flow reactor

The concept of a random dilution flux appears in a lot of artificial chemistries (Fontana, 1992; Banzhaf, 1993a, 1993b; Dittrich & Banzhaf, 1997; Speroni di Fenizio & Banzhaf, 2000; Kvasnička & Pospichal, 2001; Suzuki & Ono, 2002). It has its analogy in the chemical flow reactor: In a *flow reactor* of finite volume, continuous input of substrate builds up a pressure that causes molecules (usually products) to be squeezed out. In an artificial chemistry, a flow reactor is efficiently simulated by defining a constant vessel capacity in terms of the maximum number of molecules or symbols and by randomly destroying excessive molecules whenever new molecules are produced or injected.

In the following, we provide a deterministic mathematical description for two different limits: A first, simpler variant assumes that all molecules require the same memory, in which case it is sufficient to restrict the total number of *molecules*. In reality, molecules are of different length and have different memory requirements. We treat this scenario separately.

A deterministic mathematical description of a vessel with a limited capacity of N molecule instances is given by the following differential equation system:

$$\dot{\mathbf{x}} = \overbrace{\mathbf{S}\cdot\mathbf{a}(\mathbf{x})}^{\text{reaction rate equation}} - \overbrace{\frac{\mathbf{x}}{N}f(\mathbf{x})}^{\text{dilution flux}} \tag{11.1}$$

The first part of (11.1) is the reaction rate equation (see (8.31) on page 116), capturing the change of the number of instances of each species without memory constraints. The second part of (11.1) describes the dilution flux applied to each species $s \in \mathcal{S}$ in proportion to its current concentration x_s/N. The function $f(\mathbf{x})$ denotes the overall dilution flux, i.e. the rate at which the *Injector* (see Figure 11.3) deletes arbitrary molecules to make room for new ones. This dilution flux is only applied if the vessel is saturated and amounts to the net production rate of the system:

$$f(\mathbf{x}) = \begin{cases} 0 & \text{for } \sum_{s\in\mathcal{S}} x_s < N; \\ \sum_{s\in\mathcal{S}} \mathbf{S}_s \cdot \mathbf{a}(\mathbf{x}) & \text{for } \sum_{s\in\mathcal{S}} x_s = N. \end{cases} \tag{11.2}$$

If the molecules are of different length, we cannot simply restrict the number of molecules. Instead, we have to limit the vessel capacity to C *symbols*. Let $\mathbf{l} = (|s_1| \ \cdots \ |s_{|\mathcal{S}|}|)$ be the row vector of species lengths. The ODE system can now be expressed as

$$\dot{\mathbf{x}} = \overbrace{\mathbf{S} \cdot \mathbf{a}(\mathbf{x})}^{\text{reaction rate equation}} - \overbrace{\frac{\mathbf{xl}}{C} f(\mathbf{x})}^{\text{dilution flux}} \tag{11.3}$$

where

$$f(\mathbf{x}) = \begin{cases} 0 & \text{for } \sum_{s\in\mathcal{S}} x_s l_s < C; \\ \sum_{s\in\mathcal{S}} \mathbf{S}_s \cdot \mathbf{a}(\mathbf{x}) & \text{for } \sum_{s\in\mathcal{S}} x_s l_s = C. \end{cases} \tag{11.4}$$

We will make use of these ODEs later in Part III to calculate equilibrium properties of self-healing software.

LIMITED COMPUTING POWER 11.3.2

If we are going to embed the Fraglets VM into a real network, there is no global scheduler anymore. Each VM is driven by a separate CPU. Since chemical protocols also convey information via the packet *rate*, it is important that both the sender and the receiver have the same notion of time. This is achieved by coupling the chemical time (=simulation time) to the physical time. This means that a computer must send packets at the rate dictated by the law of mass action, although a small clock drift in the network is tolerable.

Chemical protocols only operate dynamically correct if all reactions are executed at the (physical) time for which they are scheduled. Consider the *Disperser* protocol as an example: If the CPU of a certain node is not able to send data-molecules in proportion to their local quantity anymore, but still receives molecules from its network neighbors, the molecules will accumulate in the overloaded node and distort the result.

Thus, if the Fraglets VM is driven by a real CPU with limited computing power, the VM is only capable to drive reactions up to a maximum reaction rate. This maximum rate first depends on the number of reactants: According to the law of mass action, the more reactants there are, the higher is the reaction rate. Second, the Fraglets transformations have to be executed immediately and as fast as possible. That is, the more transformations a reaction product contains in its header, the longer this reaction occupies the CPU and the lower is the reaction rate the CPU is able to ensure.

(a) *Worst and Best Case Scenarios*

maximum reaction rate

We computed the *maximum reaction rate* demanded by the law of mass action and compared it to the maximum reaction rate a CPU is able to deliver for two different scenarios: a best-case and a worst-case scenario (see details in Section A.1). The CPU reaches a maximal reaction rate if the reaction vessel only contains unimolecular reactions processing short molecules: Short molecules require only a few transformations to be executed per reaction and for unimolecular reactions, the rate only grows linearly with the number of molecules. Because the *Disperser* protocol matches these criteria, we refer to it as the best-case scenario. For the worst-case scenario we assume that there are bimolecular reactions, and that each reaction product only contains transformations. That is, the processing time is proportional to the length of the reaction products.

limiting the vessel capacity

It turns out, that by *limiting the vessel capacity* in terms of the number of symbols, we can effectively restrict the reaction rate such that it does not exceed the speed of the CPU. For the *Disperser* protocol, the vessel capacity has to be limited to

$$C < \frac{1}{2Tk_{\max}} \tag{11.5}$$

whereas for the bimolecular worst-case scenario, the vessel capacity has to be restricted to

$$C < \left(\frac{2}{Tk_{\max}}\right)^{-3/2} + \sqrt{\frac{2}{Tk_{\max}}} \tag{11.6}$$

In these equations, C denotes the symbol capacity of the reaction vessel, which is limited by the available memory and, based on our calculations, should be further restricted to let the CPU cope with the law of mass action requirements. T denotes the CPU time required to execute one Fraglets production rule – either a reaction or a transformation – and $k_{\max}$ is the reaction coefficient of the fastest reaction.

Figure 11.5 plots these equations with respect to the term $Tk_{\max}$. The maximum tolerable capacity drops if the CPU either needs more time to transform a fraglet, or if the reaction coefficients in the chemical model require faster reactions. On a computer with an Intel Core Duo 1.8 GHz processor, our Fraglets VM is able to perform about 10^5 reactions/s‡, meaning that the VM is able to correctly drive *Disperser* for up to 10^5 X-molecules.

‡The VM implementation is not multi-threaded. Thus only one processor core is used.

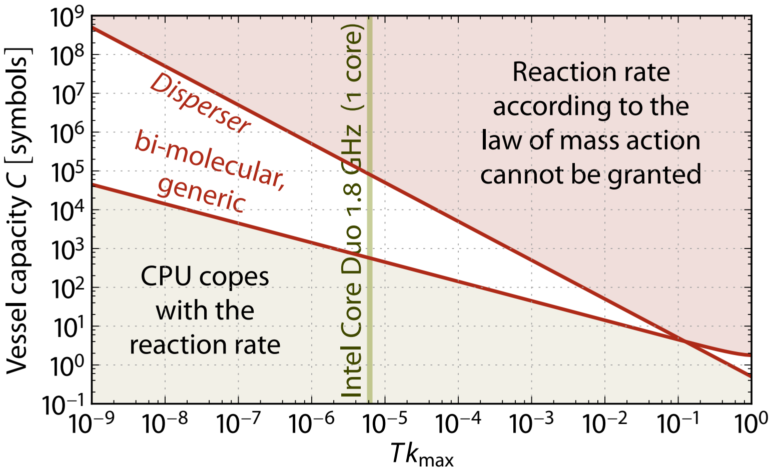


Figure 11.5 Realistic reaction vessel capacity limits: The maximal vessel capacity tolerable, C, is lower for slow CPUs and for faster reaction coefficients. In reality, the vessel must be dimensioned somewhere in between the best-case (Disperser) and the worst-case (bi-molecular, generic) scenario.

Strategies to Cope with Limited Computing Power (b)

There are at least three possible strategies to cope with the limited power a computing infrastructure offers: optimization of the reaction coefficients during the design phase of a chemical protocol, adaptive dilution at run-time, and optimization of the reaction algorithm.

Reaction coefficient optimization: One possibility to ensure a dynamically correct execution is to avoid the problem by choosing reaction coefficients such that the reaction rate required by the law of mass action never exceeds the physically obtainable rate. The lower the reaction coefficients are, the more can the vessel capacity be increased on the CPU, which in fact also lowers the noise (see Section 9.3). On the other hand, lower reaction coefficients lead to a slower protocol convergence time. Hence, the protocol designer has to find a good balance between fast convergence and a high signal-to-noise ratio, subject to the constraints of limited memory and computing power.

Adaptive dilution: Another possibility is to adjust the vessel capacity dynamically: The reaction algorithm could randomly destroy molecules in the vessel as soon as the CPU does not cope with the chemical reaction model, i.e. as soon as reaction channels are scheduled before the current physical time. The system would then use the same *random dilution* strategy for limited computing power as for limited memory.

Optimization of the reaction algorithm: A third possibility is to increase the maximum reaction rate of a CPU by using a more efficient reaction algorithm that supports, for example, *τ-leaping* (Gillespie, 2001; Rathinam et al., 2003; Tian & Burrage, 2004; Chatterjee et al., 2005; Cao &

Petzold, 2005): Instead of treating each molecule separately, a reaction updates multiple instances of the same reactant and product species. The system advances in pre-selected time steps τ during which more than one reaction event may occur. For this to be still stochastically exact, the time leap τ has to be chosen small enough such that none of the propensity functions change significantly during this time. With this method, a frequent reaction only occurs occasionally and rewrites several molecule instances instead. This method can also be used to reduce the message complexity of chemical protocols. If a leaping reaction generates several instances of a send-fraglet, the fraglet is only sent once together with a multiplicity counter. However, this obviously comes at the price of a lower robustness to packet loss.

In addition to τ-leaping, which reduces the computational overhead for fast reactions, the reaction algorithm could provide a *reaction cache* to reduce the CPU time of a single reaction. Some frequent reactions could store the tuple of reactant and transformed product species in a cache. The next time such a cached reaction occurs, the reaction algorithm injects the cached products instead of computing them again.

11.4 SUMMARY

In this chapter, we completed the description of our execution model. We provided implementation details of the Fraglets Virtual Machine (VM) and demonstrated that its modular design allows the VM library to be integrated into other programs and simulators. We use the OMNET++ network simulator framework to carry out all examples in this thesis.

In the second part of this chapter, we studied the influence of constraints of a realistic computing infrastructure. Limited memory and processing power requires us to restrict the number of molecules (or symbols) in the reaction vessel. In the next chapter we further study the effect of a realistic networking infrastructure where the bandwidth of links is limited and packets may be lost.

CHAPTER 12

CNPs in the Internet Context

On the challenges and promises to merge chemical networking protocols with the Internet — on competing and cooperating packet streams.

> Of course, there isn't any
> "God of the Internet."
> The Internet works
> because a lot of people cooperate
> to do things together.[12]
>
> Jon Postel

So far, we modeled the network as a set of reaction vessels interconnected by ideal links. In this chapter, we study the behavior of chemical networking protocols (CNPs) in more realistic network scenarios where the bandwidth of links is limited and where molecules sent to another vessel are delivered deferred. In the previous chapter, we discussed physical limitations of the executing machinery. In this chapter, we focus on the limits of the networking infrastructure.

Our second aim is to determine in what extent the chemical model is compatible with the Internet and its protocols. In particular, we are interested whether a stream of molecules cooperates with TCP streams. TCP's congestion control algorithms are the key methods for ensuring fair bandwidth allocation and are responsible for the big success of the Internet. In this respect, we study whether TCP is able to operate over a chemical transport medium and whether chemical molecule streams are fair to competing TCP packet streams.

We structure this chapter into two parts: realistic network modeling and Internet compatibility. Section 12.1 demonstrates how realistic network infrastructure is modeled within the chemical framework and proposes a medium access control scheme that reflects the behavior of enzymatic reactions. The goal of this section is to provide a realistic model for a link with limited bandwidth over which chemical protocols such as *Disperser* are still able to reach consensus.

The second part of this chapter examines whether chemical and traditional protocols may coexist in the Internet. Section 12.2 discusses the *chemistry in the core* scenario, where we envision that some of the Internet's forwarding engines may be replaced by chemical reaction vessels. We study whether TCP streams are still capable to operate over such a chemical substrate. Simulations show that TCP only performs well if we abandon most of the randomness in the chemical execution engine.

chemistry in the core

chemistry at the edge

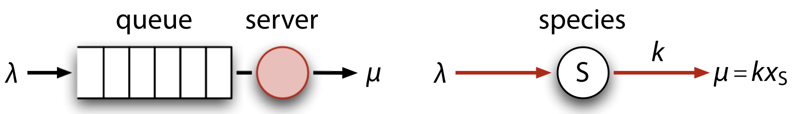
Section 12.3 discusses the contrary *chemistry at the edge* scenario, where reaction vessels communicate over the Internet and where molecule streams compete with TCP streams. We implement C3A, a chemical congestion control algorithm and demonstrate that its design and analysis is simple and straightforward. Simulation results indicate that under the regime of this algorithm, chemical streams nicely cooperate with TCP streams.

This chapter closes the second part of this thesis. Section 12.4 therefore not only summarizes this chapter, but the whole part on chemical networking protocols.

12.1 CHEMICAL MODELS OF REALISTIC NETWORKING INFRASTRUCTURE

In our chemical model, molecular species are similar to traditional packet queues as they buffer packets for a certain time. But the two packet buffering methods differ in their dynamic behavior. In this section, we have a look at components of a realistic networking infrastructure such as queues or links with limited bandwidth and delay. We present and analyze chemical reaction networks that model the behavior of such building blocks. Their chemical interpretation allows us to analyze the networking infrastructure with the chemical analysis methods discussed in Chapters 6 to 8.

In Section 12.1.1, we start with the simplest chemical building block: the chemical species. We study the buffering capability of a molecular species and compare it to a traditional FIFO queue. Section 12.1.2 then introduces a first chemical model of a link with limited bandwidth and delay. In Section 12.1.3, we develop a novel link allocation scheme based on enzymatic reactions: It combines the models of a link, a preceding queue, and a scheduling policy that sends enqueued packets over the link. Finally, in Section 12.1.4, we analyze how the now familiar *Disperser* protocol runs over realistic links.



(a) Traditional Queue (b) Chemical Queue

Figure 12.1 **Traditional vs. chemical queue**: Customers (packets or molecules) enter both queues at rate λ, modeled as a Poisson process. In the traditional queue (a), the server regularly drains the queue with rate μ whereas in the chemical queue (b), the rate depends on the fill level (number of S-molecules).

A CHEMICAL MODEL OF PACKET QUEUES 12.1.1

In order to study the dynamics of a protocol in the Internet, one usually models the core network as a network of interconnected queues (e.g. see Bose (2002)). Queues are necessary to buffer irregularly arriving data packet before they are being transmitted over a medium with limited bandwidth. Queues are ubiquitous in the Internet and the design of Internet protocols have been largely influenced by their FIFO ordering and tail-drop behavior. In the following, we demonstrate that already a simple chemical species buffers molecules like a queue buffers packets, and we discuss the differences between the chemical buffer and the traditional FIFO queue.

The Chemical Species as a Packet Buffer (a)

Each molecular species can be regarded as a simple packet buffer: Reactions producing instances of that species enter "customers" (=molecules) to the buffer. These customers stay in the buffer until another egress reaction consumes them and converts them to another species, i.e. enters them into another buffer. Figure 12.1(b) depicts such a chemical buffer in comparison to a traditional M/M/1/K queue, shown in Figure 12.1(a). This four-letter classification of queues is attributed to Kendall (1953). Here, we are comparing the chemical buffer to a queue with limited buffering capacity K, operated by one (1) server; the arrival and service processes are (M)arkov, i.e. Poisson processes for which the event intervals are exponentially distributed.

The two buffers treat the packets differently in terms of scheduling policy, tail-drop behavior, and the latency imposed to traversing packets.

Scheduling Policy (b)

Traditional queues apply a FIFO (first in / first out) policy to packet streams. FIFO
That is, the packet order is maintained from input to output, unless for the special case of priority queues. On the contrary, the chemical buffer regularly reorders packets: Molecules entering the chemical buffer are not enqueued

but put into an unordered multiset. The egress reaction then randomly selects a molecule instance from this multiset to remove. That is, information about the packets' arriving order is lost.

Note that reordering only matters if the queued customers/molecules can be distinguished. This is the case, for example, if symbolic user-information is attached to packets. CNPS that represent user-information by molecule quantities and reaction/packet rates are not sensitive to packet reordering. One example for the latter case is *Disperser*.

(c) Drop Behavior

Memory is limited, and hence the capacity of a realistic queue is limited, too. If a new packet arrives while the queue is full, traditional FIFO queues discard the new packet, which is known as *tail-drop* behavior.

tail-drop

As we discussed in the previous chapter, the related chemical concept is the flow reactor where produced or injected molecules lead to the random dilution of other molecules in the vessel. The drop behavior of a chemical buffer in a flow reactor differs from a traditional M/M/1/K queue in two aspects: First, a chemical buffer in a flow reactor exhibits *random-drop behavior*, meaning that not the latest but an arbitrary molecule instance is dropped. Second, the vessel's *memory is shared* among all species: A new molecule of one species may result in an instance of another species being destroyed. Hence, there is a certain amount of dynamic crosstalk between chemical buffers in the same vessel.

random-drop behavior

shared memory

Both, the traditional FIFO queue as well as the chemical buffer *does not ensure fairness* to different packet streams: In saturation, packets belonging to different streams are destroyed in proportion to their arrival rate. Thus, law of mass action scheduling together with random packet reordering does not guarantee a fair allocation of resources.

no fairness

(d) Latency

The latency of data packets through a traditional FIFO queue depends on the current fill-level. In contrast, the expected delay of a chemical buffer is constant, which is a direct consequence of the scheduling policy.

A packet in a FIFO queue has to wait until all previously arrived packets are served. The expected waiting time of a packet in an M/M/1 queue (including the service time) is

$$d_{\mathrm{M/M/1}} = \frac{1}{\mu - \lambda} \tag{12.1}$$

The waiting time is zero if the queue is empty and strives to infinity if the arrival rate λ approaches the constant service rate μ.

In contrast, the service rate of the chemical reaction system depicted in Figure 12.1(b) is proportional to the fill level of the chemical buffer in accordance to the law of mass action: $\mu = kx_S$. The probability that a certain molecule leaves the buffer with the next egress reaction event is inversely proportional to the number of S-molecules: $p_{\text{serv}} = 1/x_S$. On average, a molecule has to wait x_S times until it is picked up by the reaction. If we assume that the reaction system is at equilibrium, the arrival and the service rates are equal according to Kirchhoff's current law ($\lambda = \mu$, see Section 10.2.1(a)) and the steady-state fill level is $x_S = \lambda/k$. Hence, the expected waiting time – the latency – of a chemical buffer is constant.

$$d_S = \frac{x_S}{\mu} = \frac{1}{k} \tag{12.2}$$

A higher coefficient of the egress reaction results in a shorter waiting time.

If we assume that all forwarding engines are implemented as chemical first-order reactions, then, the average *end-to-end latency* of packets across the network *remains constant* even if the network load increases.

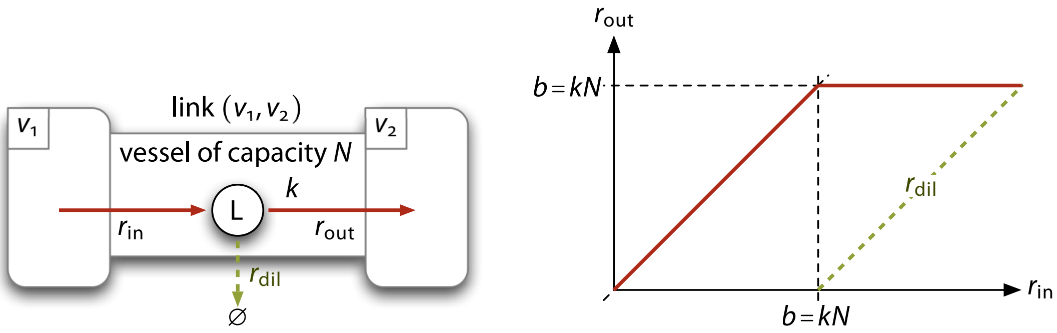
constant end-to-end latency

A CHEMICAL MODEL OF NETWORK LINKS WITH LIMITED BANDWIDTH 12.1.2

After having discussed the properties of a chemical buffer, we now look for a model of a link with limited bandwidth. The characteristics of a traditional unidirectional network link connecting two nodes are usually given by the tuple (b, d) where b is the bandwidth and d is the delay of the link. As depicted in Figure 12.2(a), such a realistic link can be modeled in the chemical framework by a reaction vessel of limited capacity containing a single buffer species L. Packets sent over this link stay in the imaginary chemical buffer until the egress reaction delivers them to the other end. By changing the reaction coefficient k of this egress reaction, we parameterize the delay afflicted to the molecules on transit.

We limit the capacity of the virtual link vessel to N molecules in order to model the finite bandwidth of the link: If the link receives a new packet from the source node while there are already N packets "on the wire" (i.e. in the virtual link vessel), an arbitrary L-molecule is dropped. The rate of this imposed dilution flux depends on the packet injection rate.

Hence, the model has two parameters that need to be specified: the egress reaction coefficient k and the reaction vessel capacity N. In the following we



(a) Reaction network **(b)** Saturation graph

Figure 12.2 Chemical link model: A link can be modeled as a reaction vessel of limited capacity N, containing a single buffer species L.

show how these parameters can be derived from the link characteristics to be modeled, i.e. from the bandwidth b and the delay d.

(a) Mathematical Analysis of the Chemical Link Model

Mathematically, the chemical link model is described by the following differential equation:

$$\dot{x}_{\mathrm{L}} = r_{\mathrm{in}} - \overbrace{k x_{\mathrm{L}}}^{r_{\mathrm{out}}} - \overbrace{r_{\mathrm{dil}}}^{\text{dilution}} \tag{12.3}$$

The dilution flux is only applied if the vessel is saturated, in which case it destroys molecules such that the number of L-molecules never exceeds the vessel capacity N:

$$r_{\mathrm{dil}}(x_{\mathrm{L}}) = \begin{cases} 0 & \text{for } x_{\mathrm{L}} \leq N; \\ r_{\mathrm{in}} - kN & \text{for } x_{\mathrm{L}} = N. \end{cases} \tag{12.4}$$

At equilibrium, the link hosts $\hat{x}_{\mathrm{L}} = \min\left(\hat{r}_{\mathrm{in}}/k, N\right)$ molecules. According to the law of mass action, the packet rate at the output of the link yields the expected saturation curve as depicted in Figure 12.2(b):

$$\hat{r}_{\mathrm{out}} = k\hat{x}_{\mathrm{L}} = k\min(r_{\mathrm{in}}, kN) \tag{12.5}$$

bandwidth

delay

Note that the term $b = kN$ appearing in the $\min(\cdot)$ function represents the *bandwidth* of the link. A second equation is found by remembering that a first-order reaction imposes a *delay* of $d = 1/k$ to the molecules in transit. Thus, the egress reaction coefficient has to be dimensioned as the reciprocal

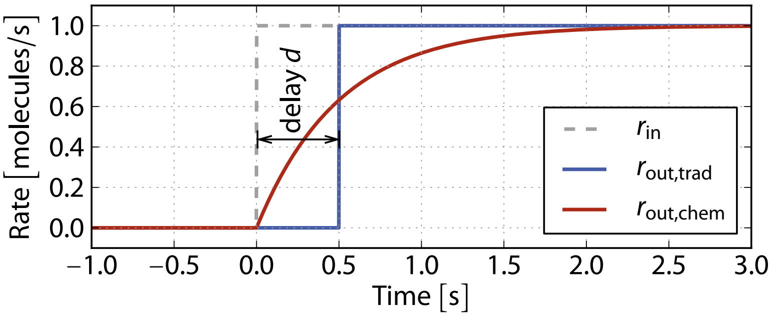


Figure 12.3 Step response of a real link and its chemical model: The output rate of a link follows exactly the input rate after a delay d. The chemical model uses a buffer molecule to simulate the delay, which exhibits a low-pass filtering character to the transmission rate.

of the delay, and it is no surprise that the vessel capacity has to be set to the *bandwidth delay product*, i.e. the maximum number of packets on the link:

bandwidth delay product

$$k = \frac{1}{d} \tag{12.6a}$$

$$N = bd \tag{12.6b}$$

While a steady-state analysis of the chemical link model correctly predicts the behavior of a real link, the model's transient behavior deviates from reality. The reason for this deviation is the *low-pass behavior* of a chemical reaction: Figure 12.3 shows the step response of the output rate r_{out} when the input rate is suddenly changed. Since the output rate is proportional to the current number of buffered molecules it only slowly adapts to the input rate.

low-pass filter

Hence, a limited artificial flow reactor containing a chemical buffer species can only be used to model a realistic network link if we are only interested in the steady-state behavior of the distributed reaction system.

No Fairness Guarantees for Competing Packet Streams (b)

The chemical link model correctly predicts that bandwidth is not fairly allocated to competing packet streams. Multiple packet streams through the same link can be modeled by an individual buffer species L_i for each stream i ($1 \le i \le n$), as shown in Figure 12.4. Their dynamic behavior is captured by the ODEs in (11.1) on page 194, now applied to the link model:

$$\dot{x}_{\mathrm{L}_i} = r_{\text{in},i} - \overbrace{k x_{\mathrm{L}_i}}^{r_{\text{out},i}} - \overbrace{\frac{x_{\mathrm{L}_i}}{N} f(\mathbf{x})}^{r_{\text{dil},i}} \tag{12.7}$$

where the total dilution flux $f(\mathbf{x})$ is non-zero if the vessel capacity is reached. Note that each buffer species is diluted in proportion to its quantity. Let us denote the total transmission rate as $r_{\text{in}} = \sum_{i=1}^{n} r_{\text{in},i}$ and the current link load (=vessel fill level) as $x_{\mathrm{L}} = \sum_{i=1}^{n} x_{\mathrm{L}_i}$. The total dilution flux is given as

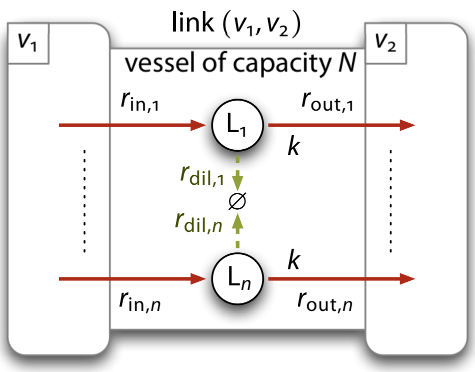


Figure 12.4 **Chemical link model for competing packet streams:** Each packet stream i is modeled with its own buffer molecule L_i. The dilution rate is proportional to the corresponding quantities.

$$f(\mathbf{x}) = \begin{cases} 0 & \text{for } x_{\mathrm{L}} \leq N; \\ r_{\mathrm{in}} - kN & \text{for } x_{\mathrm{L}} = N. \end{cases} \tag{12.8}$$

where again, $b = kN$ represents the bandwidth of the link. At equilibrium, this results in an output rate for each packet stream of

$$r_{\mathrm{out},i} = \begin{cases} r_{\mathrm{in},i} & \text{for } r_{\mathrm{in}} \leq b; \\ \frac{r_{\mathrm{in},i}}{r_{\mathrm{in}}} b & \text{for } r_{\mathrm{in}} \geq b. \end{cases} \tag{12.9}$$

As expected, the saturated link is shared *pro-rata* with the offered total load, meaning that the link does not provide fairness among the different packet streams.

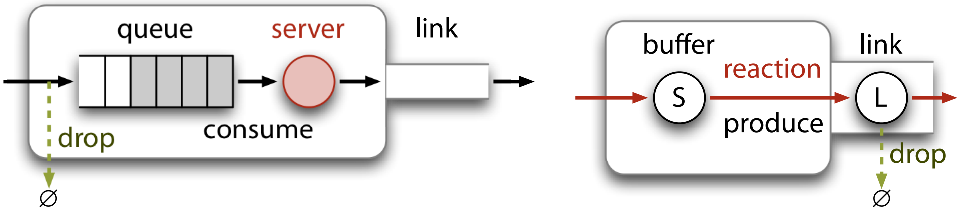
12.1.3 A NOVEL ENZYMATIC LINK ALLOCATION SCHEME

Modeling a link in isolation does not make a lot of sense. Usually, we also want to model how the limited bandwidth of a link is allocated, that is, we study the link in combination with a preceding queue and its server. In the following, we compare the traditional and the chemical way of allocating a link and propose a chemical link allocation scheme based on enzymatic reactions.

(a) Consumer and Producer Allocation Modes

consumer mode

The traditional queue-server-link triple (see Figure 12.5(a)) is one way of modeling and implementing such a resource allocation problem: Arriving packets are buffered in the queue. As soon as the link is idle, the server sends the next packet. The queue is operated in *consumer mode*, meaning that the server is the active part that drains the passive queue based on the remaining bandwidth available on the link. If the queue is never empty, this consumer



(a) Traditional Consumer Mode

(b) Chemical Producer Mode

Figure 12.5 **Consumer vs. producer link allocation modes**: In the traditional model (a), the queue is consumed by the active server, which "knows" when the link is idle. In the chemical model (b), the buffer is self-triggered, i.e. it actively sends molecules over the link according to the law of mass action.

mode of operation leads to a back-to-back transmission of data packets and a full exploitation of the link's bandwidth.

A first approach of modeling link allocation chemically is to put a chemical buffer (see Section 12.1.1) and a saturated vessel (see Section 12.1.2) in sequence. In this model, depicted in Figure 12.5(b), there is no explicit server that allocates the link. The chemical buffer operates in a self-triggered *producer mode*: its egress reaction is the active entity that sends data packets over the link. The transmission rate of this simple chemical model grows with increasing number of buffered molecules, eventually reaching the bandwidth of the link. But the chemical buffer does not reduce its transmission rate at that point. This leads to packet loss *within* the link. The chemical model discussed so far lacks a feedback mechanism that enables the reaction to control its production rate.

producer mode

We thus need an alternative chemical link model, which allows the link allocation reaction to "inspect" the link's current load and to throttle its transmission rate accordingly.

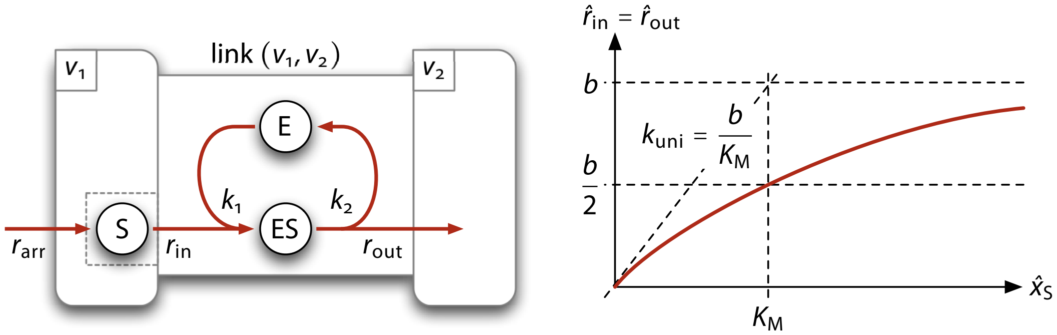
(b) *Enzymatic Link Allocation*

We already discussed that in chemistry, *enzymatic reactions* exhibit rate-limiting properties, and we proposed a corresponding motif in Section 10.3.3(c). Figure 12.6(a) depicts a novel link model that is based on such an enzymatic reaction. Here, the limited bandwidth is not modeled by capping the capacity of the virtual link vessel, but by providing a limited number (e_0) of enzyme "tokens" per link. Those enzymes are either free (E) or bound in the enzyme-substrate complex (ES). The rate at which molecules are sent, r_{in}, is proportional to the number of packets waiting in the source node and the number of free enzymes. The link *throughput* is given as

enzymatic reactions

throughput

$$\hat{r}_{\text{in}} = \hat{r}_{\text{out}} = b\frac{\hat{x}_{\text{S}}}{K_{\text{M}} + \hat{x}_{\text{S}}} \qquad (12.10)$$



(a) Reaction Network (b) Saturation curve

Figure 12.6 Enzymatic link model: The link's bandwidth limitation is modeled by an enzymatic reaction.

Recall from the rate limitation motif in Section 10.3.3(c) on page 173 that the coefficient $K_M = k_2/k_1$ specifies the number of substrate S_1-molecules at which half of the bandwidth $b = k_2 e_0$ is reached.

Unlike the traditional queue-server-link model, the enzymatic link model does not transmit packets as fast as possible (i.e. back-to-back) with rate b. The rate r_{in} rather continuously increases with the number of buffered substrate molecules as depicted in Figure 12.6(b). For a few buffered molecules ($\hat{x}_S \to 0$) the transmission rate linearly increases with the number of S-molecules, a behavior comparable to a unimolecular reaction with coefficient $k_{uni} = b/K_M$. If many molecules are buffered ($\hat{x}_S \to \infty$) the transmission rate asymptotically converges to the bandwidth of the link b.

The enzymatic link model is parameterized by the reaction coefficients k_1 and k_2, and the total number of enzymes in the link e_0. These parameters have to be chosen such that the model exhibits a maximum transmission rate b and a certain delay d. In addition, the steepness of the saturation curve can be specified by k_{uni}. The three parameters can be derived according to the following equations:

$$k_2 = \frac{1}{d} \tag{12.11a}$$

$$e_0 = bd \tag{12.11b}$$

$$k_1 = \frac{k_{uni}}{bd} \tag{12.11c}$$

Compared to the previous chemical link model, packets are not lost within the link. Instead they are kept in the substrate buffer S until the link is idle. This is a huge advantage, because in the source vessel, they alternatively can be processed by other reactions. For example, if there is an additional link to the same destination, the enzymatic reaction of the second link can attach to the same substrate. The law of mass action then automatically balances the load over the two links, resulting in a very simple link *load-balancing* implementation.

load-balancing

(c) *Comparison of the Enzymatic Link Model to a Traditional M/M/1 Queue*

The structure of the enzymatic reaction network suggests a comparison to an M/M/c-queue, a queue processed by c servers: The substrate species S can be compared to a queue where users (molecules) arrive a rate r_{arr}. The queue is processed by c servers (i.e. e_0 enzymes) that become occupied while processing a user (the enzyme is bound to the substrate forming the complex ES). Finally, users depart from each server with a constant average rate, which in the enzymatic reaction is represented by the dissociation of the enzyme with rate k_2.

Despite this similarity, the dynamics of the two systems is quite different: First, the chemical reaction network reorders molecules whereas the queue maintains the packet order. Second, once a bound enzyme delivers a product and becomes free, it does not react immediately with the next substrate molecule, but rather waits for a certain time, which is determined by the law of mass action. A server in a queuing system, on the other hand, immediately processes the next customer. In the following, we analyze the dynamic behavior of the enzymatic reaction and compare it phenomenologically to a traditional queuing model.

At equilibrium, the input and output rates of the enzymatic link will be equal to the arrival rate ($r_{\text{out}} = r_{\text{in}} = r_{\text{arr}}$) as long as the arrival rate does not exceed the bandwidth b. Let us define the *load factor* as

load factor

$$\rho = \frac{r_{\text{arr}}}{b} \tag{12.12}$$

denoting the fraction of the bandwidth in use. As expected, the steady-state substrate concentration grows hyperbolically with the load factor and reaches infinity for $\rho \to 1$:

$$\hat{x}_S = K_M \frac{\rho}{1 - \rho} \tag{12.13}$$

Interestingly, such a behavior is phenomenologically similar to that of a simple *M/M/1-queue* – a traditional packet queue of unlimited buffering capacity, operated by *one* server. Table 12.1 shows some reference values of the M/M/1-queue and the enzymatic link model in comparison. The most striking similarity is that the total number of users in an M/M/1-queue is

comparison to the M/M/1-queue

M/M/1-Queue Property	(average)	Enzymatic Link Property	(average)
User arrival rate	r_{arr}	Substrate arrival rate	r_{arr}
Packet reordering	no	Molecule reordering	yes
Server action	immediately	Enzyme binding	delayed (LOMA)
Number of servers	1	Number of enzymes	e_0
Service rate	b	Max. transmission rate	b
Users in queue	$\frac{\rho^2}{1-\rho}$	Unbound subst. molecules $\hat{x}_S$	$K_M \frac{\rho}{1-\rho}$
Users in server/link	ρ	Bound enzymes ($\hat{x}_{ES}$)	$\frac{r_{\text{arr}}}{k_2}$
Total Users in System	$\frac{\rho}{1-\rho}$	Substrate molecules ($\hat{x}_S + \hat{x}_{ES}$)	$K_M \frac{\rho}{1-\rho} + \frac{r_{\text{arr}}}{k_2}$
Waiting Time in Queue	$\frac{\rho}{b-r_{\text{arr}}}$	Waiting Time in S	$K_M \frac{1}{b-r_{\text{arr}}}$
Total Waiting Time	$\frac{1}{b-r_{\text{arr}}}$	Total Waiting Time	$K_M \frac{1}{b-r_{\text{arr}}} + \frac{1}{k_2}$

r_{arr}: arrival rate; b: service rate / link bandwidth; $\rho = r_{\text{arr}}/b$: load factor

Table 12.1 **Comparison of the enzymatic link model to an M/M/1-queue**: Despite the fact that the M/M/1 queue only has one server whereas the enzymatic reaction contains e_0 enzymes, the average measures are remarkably similar.

equal to the number of molecules in the substrate buffer S for $K_M = 1$ and that the waiting time of a packet in the queue is equal to the waiting time of a molecule in the substrate buffer.

(d) *Towards an Enzymatic Medium Access Control Protocol*

So far, we only used enzymatic reactions as a *model* for realistic links. However, some chemical protocols require such a smooth saturation behavior, as we will see in the next subsection for the case of *Disperser*. Only then they reach a consensus based on distributed quantities and rate gradients. Thus, we envision a Medium Access Control (MAC) regime that exhibits the dynamic behavior of enzymatic reactions on the macroscopic scale. Currently, there is no implementation of such an access control protocol yet, but in the following, we provide some requirements for such a hypothetical algorithm.

In each vessel attached to the medium, the enzymatic MAC protocol would have to inspect the number of substrate molecules and find a way to schedule their transmission in proportion to the substrate quantities by respecting the bandwidth limitation and the delay of the medium. If multiple vessels have access to the same medium, the dynamic behavior of the enzymatic MAC protocol has to obey the following model depicted in Figure 12.7: The enzyme “tokens” are shared among all vessels accessing the link. The vessels actually compete for the enzymes according to the law of mass action, meaning that bandwidth is assigned to the vessels in proportion to their buffered substrates.

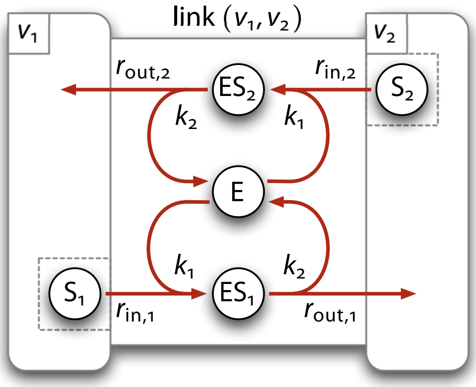


Figure 12.7 **Enzymatic shared medium link model**: The enzyme is virtually shared among all nodes connected via this link.

Let us assume that there are n nodes accessing the link, each of them is buffering molecules for transmission in substrate species S_i $(1 \leq i \leq n)$. Let us denote the total number of substrate molecule instances as $x_\text{S} = \sum_{i=1}^{n} x_{\text{S}_i}$ and the total throughput as $r_\text{in} = \sum_{i=1}^{n} r_{\text{in},i}$. The transmission rate of each individual node at equilibrium is

$$\hat{r}_{\text{in},i} = \hat{r}_{\text{out},i} = b \frac{x_{\text{S}_i}}{K_\text{M} + x_\text{S}} \tag{12.14}$$

Each node obtains the same share of

$$\frac{\hat{r}_{\text{in},i}}{\hat{r}_\text{in}} = \frac{\hat{x}_{\text{S}_i}}{\hat{x}_\text{S}} \tag{12.15}$$

Note that even though the saturation curve is smooth, the enzymatic link model does not ensure intrinsic stream-level fairness.

Whether or not an enzymatic MAC protocol implementation is possible, has to be verified in the future. One problem that could arise is, that the more substrate is present in the nodes, the higher will probably be the coordination overhead to perform the distributed scheduling.

12.1.4 APPLICATION CASE: DISPERSER OVER A LINK WITH LIMITED BANDWIDTH

What happens if we run *Disperser* over a network with limited bandwidth b and delay d? In the last part of this section we analytically study *Disperser's* behavior in a network in which all nodes are connected by links with the same characteristics. We examine two different scenarios: In the first scenario, we use traditional FIFO queues to allocate the link bandwidth. In the second scenario, we make use of the enzymatic link model to allocate the bandwidth.

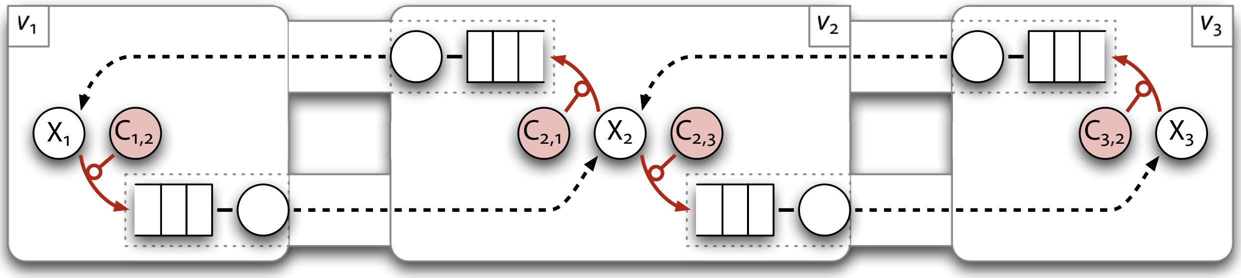


Figure 12.8 **Disperser over a traditionally allocated link**: Control molecules C put packets to the corresponding transmission queues. Each queue is scheduled by its server as soon as the link is idle.

Traditional Packet Queue Allocation. *Disperser* relies on the conservation of molecules principle, which means that packets must not be dropped. One way to achieve this is to put traditional FIFO queues in front of each link. After a control molecule $C_{i,j}$ reacted with a data molecule X_i, it puts the product molecule to be sent to neighbor node v_j to the corresponding packet queue as depicted in Figure 12.8. Note that the red solid arrows are chemical reactions, scheduled according to the law of mass action, whereas the black dashed arrows are executed as fast as possible. That is, as soon as the link is idle, the "server" dequeues the next packet and sends it over the link.

This method does not always lead to an equilibrium in which the presented number of X-molecules is the expected average: As soon as the concentration in one node is higher than the bandwidth, the queues are filled faster than they can be drained. If the average value is higher than the bandwidth, the nodes send each other molecules at the maximum rate b, meaning that the influx is equal to the efflux in each node. Thus, the presented result is bounded.

$$\hat{x}_{\mathrm{X}} = \hat{x}_{\mathrm{X}_i} = \min\left(b, \frac{x_{\mathrm{T}}}{|\mathcal{V}|}\right) \qquad \forall i \in \mathcal{V} \tag{12.16}$$

The remaining molecules got stuck in the queues. Thus, saturated traditional links, even when allocated by a queue, do not allow the distributed chemical reactions to exploit their dynamic behavior to reach a consensus.

Enzymatic Link Allocation. Instead of using traditional queues, we now make use of our enzymatic link model and prove that *Disperser* converges to an acceptable solution, which however slightly deviates from the correct average value. As shown in Figure 12.9, the data molecules X_i in each node v_i are now drained to a neighbor node by the control molecule (enzyme) $C_{i,j}$ *in* the corresponding link $(i, j) \in \mathcal{E}$. The resulting control-data complex $CX_{i,j}$

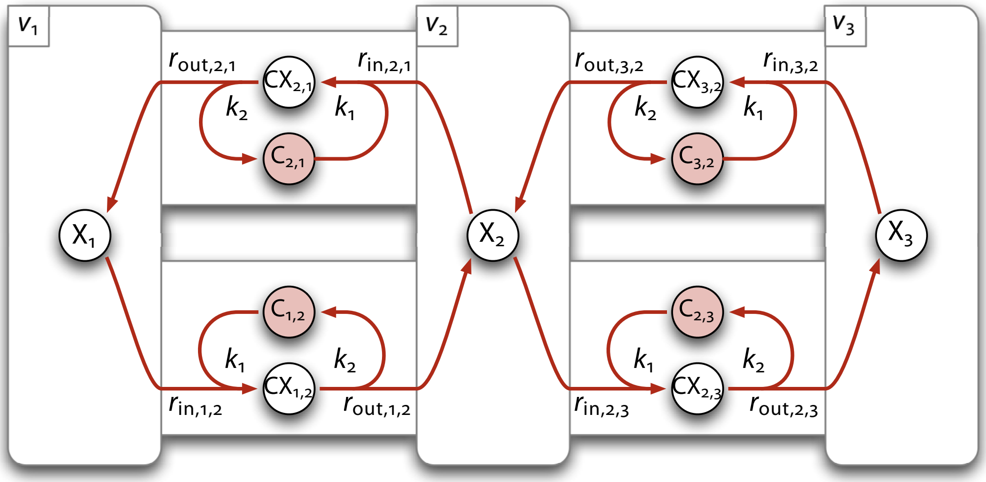


Figure 12.9

Disperser over enzymatic links: Control molecules C virtually reside inside the link. The enzymatic reaction network in the link makes sure the transmission rate does not exceed the link's bandwidth.

delivers the data molecule after the link delay $d = 1/k_2$ and becomes free for other data molecules to shuttle.

In the following, we prove that this variant of *Disperser* still converges to a fixed point for an arbitrary network topology if all nodes are connected and if for each link $(i, j) \in \mathcal{E}$ there is also a link in the opposite direction $(j, i) \in \mathcal{E}$. In order to simplify the notation, let us use the following *variable substitutions*:

variable substitutions

$$x_i := x_{\text{X}_i} \qquad \text{number of data molecules in node } v_i \tag{12.17}$$

$$y_{i,j} := x_{\text{CX}_{i,j}} \qquad \text{number of data molecules bound in link } \big(i, j\big) \tag{12.18}$$

$$c_{i,j} := x_{\text{C}_{i,j}} \qquad \text{number of free control molecules in link } \big(i, j\big) \tag{12.19}$$

The ODE system of the network comprises of equations for the data species in free (X_i) and bound form ($\text{CX}_{i,j}$):

$$\dot{x}_i = \overbrace{\sum_{j \in \mathcal{N}_i} k_2 y_{j,i}}^{\text{from link } (j,i)} - \overbrace{\sum_{j \in \mathcal{N}_i} k_1 c_{i,j} x_i}^{\text{to link } (i,j)} \qquad \forall i \in \mathcal{V} \tag{12.20a}$$

$$\dot{y}_{i,j} = \underbrace{k_1 c_{i,j} x_i}_{\text{from node } v_i} - \underbrace{k_2 y_{i,j}}_{\text{to node } v_j} \qquad \forall \big(i, j\big) \in \mathcal{E} \tag{12.20b}$$

subject to the conservation relations

$$e_0 = \overbrace{c_{i,j}}^{\text{free}} + \overbrace{y_{i,j}}^{\text{bound}} \qquad \forall\,(i,j) \in \mathcal{E} \tag{12.20c}$$

$$x_{\mathrm{T}} = \underbrace{\sum_{i \in \mathcal{V}} x_i}_{\text{in nodes}} + \underbrace{\sum_{(i,j) \in \mathcal{E}} y_{i,j}}_{\text{in links}} \tag{12.20d}$$

which represent the two types of conservation loops in this reaction network: the number of enzymes in each link – free or bound – are kept constant at the value e_0 whereas the total number of data molecules – in nodes and links – is also conserved at x_{T}.

Setting the time derivative in (12.20b) to zero and using the enzyme conservation relation (12.20c) yields a steady-state expression for the bound data molecules within a link, which resembles the Michaelis-Menten equation (see (12.10) on page 207):

$$\hat{y}_{i,j} = e_0 \frac{\hat{x}_i}{K_{\mathrm{M}} + \hat{x}_i} \qquad \forall\,(i,j) \in \mathcal{E} \tag{12.21}$$

From (12.20a) we obtain a preliminary steady-state expression for the number of data-molecules in each node:

$$\hat{x}_i = K_{\mathrm{M}} \frac{\sum_{j \in \mathcal{N}_i} \hat{y}_{j,i}}{\sum_{j \in \mathcal{N}_i} \underbrace{\hat{c}_{i,j}}_{e_0 - \hat{y}_{i,j}}} \qquad \forall i \in \mathcal{V} \tag{12.22}$$

By substituting $\hat{c}_{i,j} = e_0 - \hat{y}_{i,j}$ from (12.20c) as indicated, plugging in (12.21) for $\hat{y}_{i,j}$, and bringing all $\hat{x}_i$-terms to the left hand side we obtain

$$\underbrace{e_0 \frac{\hat{x}_i}{K_{\mathrm{M}} + \hat{x}_i}}_{\hat{y}_{i,j}} = \frac{\sum_{j \in \mathcal{N}_i} \hat{y}_{j,i}}{\deg(i)} \qquad \forall i \in \mathcal{V} \tag{12.23}$$

At first sight, we recognize that all links leaving a particular node v_i contain the same number of data molecules. Second, the number of bound data molecules in link (i,j) is equal to the average number of data molecules in all links entering node v_i. This only holds if all links contain the same amount of data molecules:

$$\hat{y} := \hat{y}_{i,j} = \frac{\sum_{i \in \mathcal{V}} \hat{y}_{i,j}}{|\mathcal{V}|} \qquad \forall\,(i,j) \in \mathcal{E} \tag{12.24}$$

Since according to (12.21) the equilibrium number of bound data molecules $\hat{y}_{i,j} = f(\hat{x}_i)$ is a monotonic function, the inverse function $\hat{x}_i = f^{-1}\left(\hat{y}_{i,j}\right)$ is monotonic, too. Consequently, since all links contain the same number of bound data molecules, the equilibrium number of molecules is the same in all nodes, too:

$$\hat{x} := \hat{x}_i = \frac{\sum_{i \in \mathcal{V}} \hat{x}_i}{|\mathcal{V}|} \qquad \forall v_i \in \mathcal{V} \tag{12.25}$$

Hence, the system converges to a fixed point where all nodes present the same number of data molecules. We find the resulting quantity by plugging in the previous two equilibrium equations (12.24) and (12.25) into the conservation relation for the data molecules (12.20d):

$$x_{\mathrm{T}} = |\mathcal{V}|\,\hat{x} + |\mathcal{E}|\,\hat{y} \tag{12.26}$$

Instead of the number of edges $|\mathcal{E}|$ we use the *average node degree* in the network $\delta = |\mathcal{V}| \,/\, |\mathcal{E}|$ and obtain

average node degree

$$x_{\mathrm{ok}} = \hat{x} + \delta \hat{y} \tag{12.27}$$

The left-hand side value $x_{\mathrm{ok}} = x_{\mathrm{T}} / |\mathcal{V}|$ is the value we would like the network to calculate – the average number of data molecules in the network. Note that at equilibrium, this expected result is split into one part presented as data molecules in the nodes ($\hat{x}$) and another part that is stored within the connecting links ($\delta \hat{y}$). Expanding $\hat{y}$ according to (12.21) yields

$$x_{\mathrm{ok}} = \hat{x} + \delta e_0 \frac{\hat{x}}{K_{\mathrm{M}} + \hat{x}} \tag{12.28}$$

The term $\delta e_0 = \delta b d$ is the average total outgoing bandwidth delay product of a node, i.e. the maximal number of molecules that can be buffered within the outgoing links of an average node. Finally, the coefficient $K_M = b$ denotes the bandwidth of the links (if we assume $k_{\mathrm{uni}} = 1$). As a next step, in order to eliminate one parameter, we normalize the equation with respect to the expected result x_{ok}: the normalized obtained result is $\hat{\chi} = \hat{x} / x_{\mathrm{ok}}$, the normalized link bandwidth $\beta = b / x_{\mathrm{ok}}$. This leads to the following quadratic equation

$$\hat{\chi}^2 - \hat{\chi}\left(1 - \beta\,(1 + \delta d)\right) - \beta = 0 \tag{12.29}$$

yielding

$$\hat{\chi}_{1,2} = \frac{1}{2}\left[1 - \beta\,(1 + \delta d) \pm \sqrt{\left[1 - \beta\,(1 + \delta d)\right]^2 + 4\beta}\right] \tag{12.30}$$

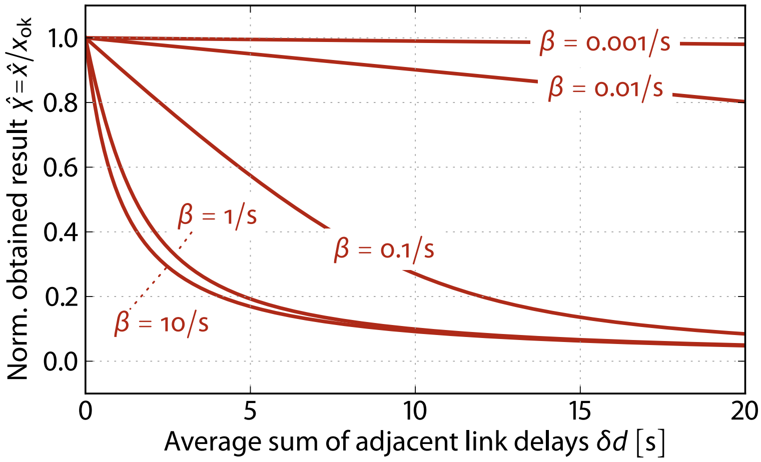


Figure 12.10 Accuracy of *Disperser* in a network with enzymatic links (1): Plot of the presented result normalized with respect to the expected correct average value plotted with respect to the sum of an average node's outgoing link delays. The result is accurate if the system is operated below the link bandwidth ($\hat{x} \ll b$). Otherwise, the result is reduced for increasing link delays.

The obtained value approximates the correct average result ($\chi \to 1$) for $\delta d \to 0$, i.e. if the links either have no delay or if the network consists of a single node.*

Figure 12.10 shows the accuracy of the result (i.e. the presented result normalized with respect to the expected result) plotted with respect to the average total outgoing delay of a node. The value on the x-axis increases for a larger delay and for a higher network connectivity. The five curves represent different normalized bandwidth values β. The bandwidth is normalized with respect to the expected result; remember that *Disperser* over traditional queues failed for $\beta > 1$. Similarly, when running over enzymatic links, *Disperser* is most accurate when the network is operated below the bandwidth, i.e. when the bandwidth is smaller than the expected result, but the protocol also works well if the link delay is small. The reason is that for smaller delays, less data packets are bound to enzymes within a link.

Figure 12.11 again shows the accuracy of the result, but now plotted with respect to the normalized bandwidth while the four curves represent different values of the average total outgoing delay. As we calculated before, the result is accurate if the link has no delays. For pathologically large delays around or above 1 s† the obtained result is much lower than the expected result when *Disperser* operates at or above the link bandwidth.

12.1.5 SUMMARY

In this section, we compared the scheduling algorithm of the chemical reaction model to the traditional packet processing policy in the Internet and identified two main differences: (1) Chemical reactions are scheduled according to the law of mass action whereas packet queues are drained as fast as possible. (2) Molecules are frequently reordered whereas a queuing network

*Note that we required a connected network with bidirectional links. With these constraints, the average outdegree is one iff the network contains only one node.

†Typical Internet link delays are below 100 ms.

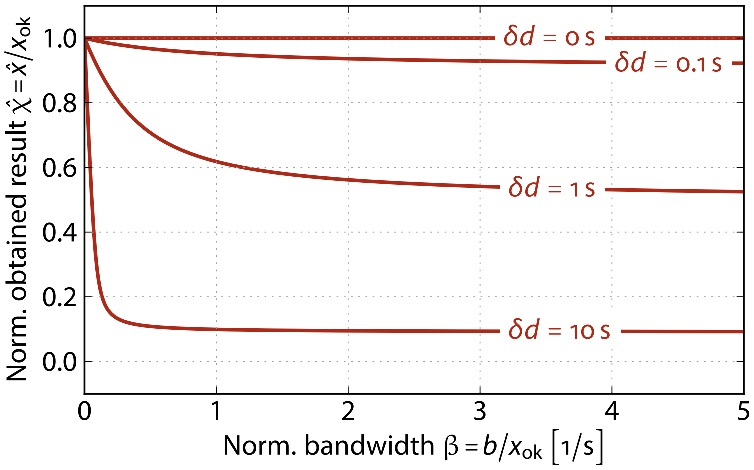


Figure 12.11 Accuracy of *Disperser* in a network with enzymatic links (2): Plot of the presented result normalized with respect to the expected correct average value plotted with respect to the bandwidth normalized with respect to the expected correct average value. The result is accurate if the delay is small. Otherwise, the result is reduced if the bandwidth is around or above the expected result.

maintains the packet order. As we will see in the next chapter, especially the random reaction order causes problem together with TCP.

After comparing the two approaches, we obtained a chemical model of a realistic link based on an enzymatic reaction. Unlike traditional link allocation methods, the enzymatic reaction smoothly increases the transmission rate in proportion to the fill level of the chemical buffer. We analytically proofed that *Disperser* requires such a link allocation scheme in order to work correctly in a realistic network.

CHEMISTRY IN THE CORE: TCP OVER CHEMICAL NETWORKS 12.2

In this section, we study whether it is possible to exchange parts of the Internet's forwarding functionality by chemical protocols. We look at a hypothetical scenario where the network's core is replaced by reaction vessels running chemical packet forwarding processes while applications at the network's edge are still implemented in the classical style. In particular, we study the behavior of TCP streams forwarded by chemical reactions.

Our initial OMNET++ simulations in Section 12.2.1 reveal that standard TCP does not perform well over chemical reactions due to frequent packet reordering, which is caused by the stochastic chemical reaction-scheduling algorithm. In Section 12.2.2, we therefore propose a modification to the scheduling algorithm in order to approximate the behavior of traditional queues. Based on further simulations, we then compare in Section 12.2.3 TCP's behavior over queue-allocated links versus enzymatic links and discover no significant differences.

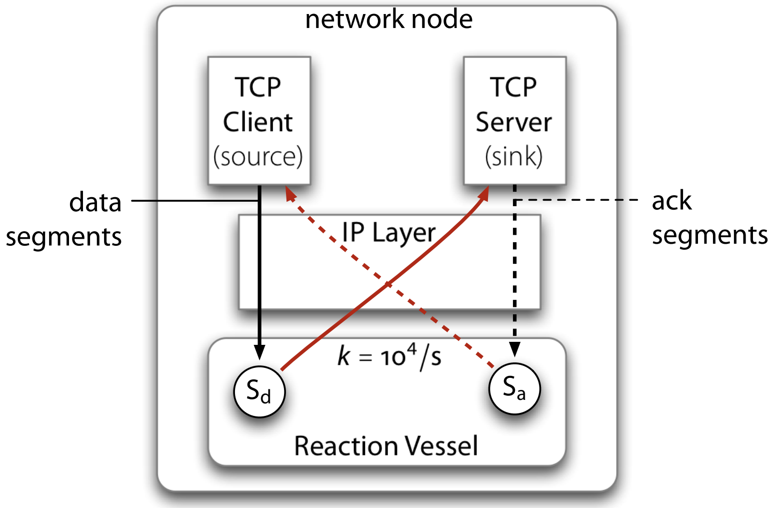


Figure 12.12 TCP loopback through a reaction vessel: The link layer below TCP/IP is replaced by a chemical reaction vessel. Packets from the network layer are looped back by a unimolecular reaction after being relayed in a chemical species: species S_d buffers data packets whereas S_a buffers acknowledgments.

12.2.1 INITIAL EXPERIMENT: THE BEHAVIOR OF A TCP STREAM THROUGH A CHEMICAL REACTION VESSEL

We start with a simple OMNET++ simulation experiment to illustrate the problems of a TCP stream traversing a chemical reaction vessel. As depicted in Figure 12.12, we replaced the link layer of a network node with a chemical reaction vessel. A packet arriving from the network layer is converted into a fraglet. In our simple scenario, a unimolecular reaction sends the fraglet back to the network layer. This reaction is scheduled by the law of mass action scheduler (see Section 5.1.2), which is indicated by the red arrows pointing upwards in Figure 12.12.

(a) Representation of IP-Packets in Fraglets

Let us discuss in more detail, how IP packets are converted into fraglet strings and how reactions operate on them. An IP packet from source node *a.b.c.d* to destination node *w.x.y.z* containing a binary TCP segment *s* is converted to fraglet

$$\texttt{[IP\_}w\texttt{\_}x\texttt{\_}y\texttt{\_}z\texttt{ srcAddr IP\_}a\texttt{\_}b\texttt{\_}c\texttt{\_}d\texttt{ data }s\texttt{]} \qquad (12.31)$$

The fraglet starts with a passive tag that represents the packet's destination IP address, followed by the source address and a single symbol containing the TCP segment.

Such a fraglet version of a TCP/IP packet reacts with any active fraglet that matches for the corresponding IP address tag. A packet is sent back to the network layer by invoking the `deliver`-instruction. In fact, the unimolecular reaction that we installed in the vessel for this scenario is implemented by 1000 copies of the fraglet

```
[matchp IP_127_0_0_1 deliver IP_127_0_0_1]
```
(12.32)

This persistent active fraglet reacts with a passive IP-fraglet destined to the node's loopback address and delivers it back to the network layer after regenerating the consumed destination address.

Simulation Setup (b)

The TCP client sends a block of data to the TCP server; TCP segments are tunneled through the local reaction vessel: At time $t = 0.1$ s, the client initiates a connection to the server; this triggers TCP's three-way handshake (SYN/SYN-ACK/ACK). At time $t = 1$ s, the client tries to send a block of 10 MB data to the server. After completion, the client and the server close the connection.

We use the TCP-Reno implementation of OMNET++ with the following parameters: The Maximum Segment Size (MSS) is set to 1000 B, delayed ACKS are disabled, Nagle's algorithm (1984) is enabled, selective ACK (SACK) support is disabled (Mathis, Mahdavi, Floyd, & Romanow, 1996), the advertised window is set to the maximum value of 64 kB.

Simulation Results and Discussion (c)

The OMNET++ simulation reaches a throughput of 1.67 MB/s, or expressed as a packet rate, $\hat{r}_{\mathrm{tp}} = 1670$ pkt/s. This is a very low throughput when considering that there is no bandwidth limitation and no packet loss. The reason for such a bad performance is twofold: The most important problem is that packets buffered in the chemical reaction vessel are *reordered* as the chemical buffer does not provide FIFO ordering. TCP on the other hand is very sensitive to packet reordering. The second problem is that the reaction interval is exponentially distributed, which introduces a huge *jitter* to the packet stream.

reordering

jitter

To better understand the consequences of chemical packet forwarding we discuss the measurements captured in OMNET++ of a typical simulation run depicted in Figure 12.13. The figure on the top shows the number of *buffered data and acknowledgment molecules* in the reaction vessel: $x_{\mathrm{S}} = x_{\mathrm{S_d}} + x_{\mathrm{S_a}}$. Remember that packets are forwarded by a unimolecular reaction with reaction coefficient $k = 1000$/s. According to Kirchhoff's current law the influx of a chemical species at equilibrium is equal to its efflux:

buffered molecules

$$\hat{r}_{\mathrm{tp}} = k\hat{x}_{\mathrm{S_d}} \qquad \hat{r}_{\mathrm{tp}} = k\hat{x}_{\mathrm{S_a}} \tag{12.33}$$

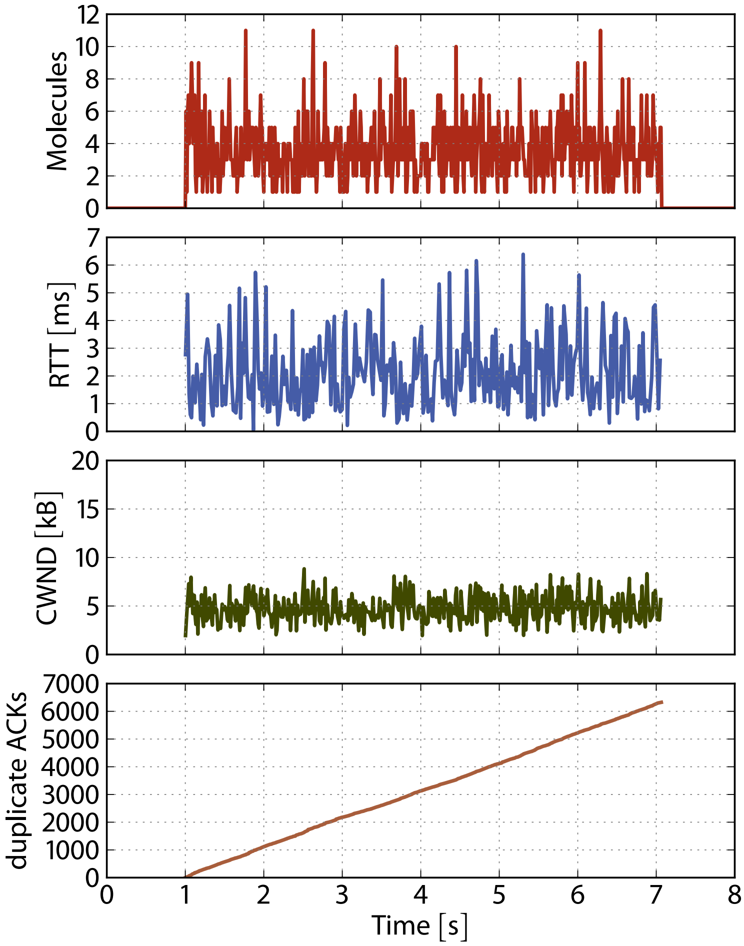


Figure 12.13 TCP loopback through a reaction vessel — measurements: 10 MB of data is sent from client to server between time $t = 1$ s and $t = 7$ s. **Molecules:** Number of IP fraglets buffered in the reaction vessel (both data and acknowledgment packets together). The steady-state quantity is between three and four molecules. **RTT:** Round trip time in milliseconds measured by the TCP sender. Large fluctuations are observed; the average is at 2 ms. **CWND:** Congestion window of the sender in kilobytes. The congestion window is never able to grow above a few packets due to frequent packet reordering in the reaction vessel. **Duplicate ACKs:** Cumulative number of duplicate acknowledgments arriving at the client; duplicate ACKs are a consequence of reordered packets; a sequence of three or more consecutive duplicate ACKs is interpreted as packet loss.

yielding a steady-state quantity of $\hat{x}_{S_d} = \hat{x}_{S_a} = \hat{r}_{tp}/k$ for both the data and the acknowledgment packets. The total number of molecules thus is

$$\hat{x}_S = \hat{x}_{S_d} + \hat{x}_{S_a} = 2\frac{\hat{r}_{tp}}{k} = 2\frac{1670\,\text{pkt/s}}{1000/\text{s}} = 3.34\,\text{pkt} \qquad (12.34)$$

which is about the mean observed in Figure 12.13.

round-trip time

The second subfigure form the top shows the *round-trip time* (RTT) measured by the TCP client. On average, the RTT seems to be around 2 ms. This delay is in fact the delay imposed by the chemical reaction vessel. As discussed in Section 12.1.1, a chemical buffer afflicts each packet with a delay of $d = 1/k$. Since both data packets as well as acknowledgments are passed through a chemical buffer, the overall RTT is expected to be

$$\text{RTT} = 2\frac{1}{k} = \frac{2}{1000\,/\text{s}} = 2\,\text{ms} \qquad (12.35)$$

The third curve from the top in Figure 12.13 shows the size of the *congestion window* (CWND) of the TCP client. This window is part of the slow start mechanism of TCP (Jacobson, 1988; Stevens, 1997; Allman, Paxson, & Stevens, 1999; Allman, Paxson, & Blanton, 2009) and controls how many unacknowledged packets are allowed to be "on the wire". The window is initially set to one segment and is incremented each time an ACK is received, such that the window is opened exponentially. In our simulation, the congestion window is not able to grow, because packets are reordered frequently in the reaction vessel. This causes repeated duplicate ACKS.

congestion window

The subfigure at the bottom of Figure 12.13 shows the cumulative number of duplicate ACKS observed by the client. If segments arrive out of sequence the server generates an immediate acknowledgment potentially leading to the observation of *duplicate ACKS* at the client side. Note that in our simulation the 10000 segments transmitted resulted in the huge amount of 6000 duplicate ACKS. If three or more duplicate ACKS are observed in a row, the sender assumes that a segment has been lost, closes the congestion window and retransmits the lost packet.

duplicate ACKs

The maximum throughput reachable in theory is limited by the maximum window size of 64 kB. This is the maximal bandwidth delay product that can be exploited by a standard TCP implementation without the window scale option (Jacobson, Braden, & Borman, 1992), which we do not consider here. For a delay of 2 ms the maximum throughput is

$$TP_{\max} = \frac{\mathrm{WND}_{\max}}{\mathrm{RTT}} = \frac{64\,\mathrm{kB}}{2\,\mathrm{ms}} = 31.25\,\frac{\mathrm{MB}}{\mathrm{s}} \tag{12.36}$$

The achieved throughput of 1.67 MB/s is only 5 % of the maximal theoretical throughput.

12.2.2 ADDING DETERMINISM TO THE CHEMICAL TRANSPORT LAYER

The simulation of a TCP stream through a chemical reaction vessel revealed TCPs brittleness to segment reordering. Although there is a TCP variant that copes with reordered segments (Bohacek, Hespanha, Lee, Lim, & Obraczka, 2003) most hosts in the Internet still use TCP Reno. In order to use chemical processes in the Internet's forwarding core we have to avoid that packets are brought out of sequence. We therefore modify the reaction algorithm with respect to two aspects, aiming at making chemistry more deterministic: The first modification adds FIFO-order to chemical species whereas the second removes the randomness of reaction intervals. We thereby depart further

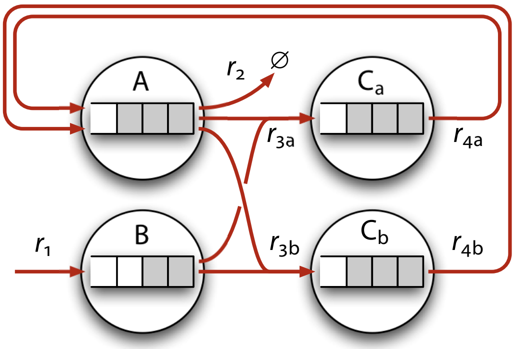


Figure 12.14 Chemcial reaction network with FIFO-order: The reactor hosts a FIFO queue for each species in the vessel. Produced molecule instances are enqueued to the corresponding product species. Reactions that consume molecules remove the oldest molecule of the corresponding reactant species.

from chemical plausibility and converge towards more traditional packet processing paradigms.

(a) A Modification to the Reaction Algorithm to Achieve FIFO-Order and Tail-Drop Behavior

We change the data structure of the chemical reaction vessel such that instead of adding new molecule instances to a common multiset, the vessel now maintains a FIFO queue for each molecular species. A reaction dequeues the oldest molecules from the corresponding reactant species and enqueues the produced molecules to the product species. Figure 12.14 shows a toy reaction network where each species "contains" the queue for its instances.

The difference to a traditional queuing network is in how the queues are scheduled. While traditional queues are drained as fast as possible, chemical queues are still operated by a law of mass action scheduler. That is, the more elements a queue contains, the faster those elements are removed. If the reaction is multi-molecular, the reaction rate also depends on the fill-level of the other reactant queues. For example, in the toy example depicted in Figure 12.14, reaction r_{3a} (with an assumed coefficient of $k_3 = 1/(\text{pkt s})$) is executed at rate $a(x_A, x_B) = 1/(\text{pkt s}) \cdot 3\,\text{pkt} \cdot 2\,\text{pkt} = 6\,\text{pkt/s}$ while r_2 occurs three times a second.

drawbacks

There are two *drawbacks* of a FIFO queuing discipline. First, the latency of molecules through chemical species changes. In Section 12.1.1(d), we demonstrated that each molecule instance that passes a species, which is consumed by a unimolecular egress reaction with coefficient k, is afflicted with a constant average delay of $\hat{d} = 1/k$. This still holds if the species is operated in FIFO-mode, but the delay is not exponentially distributed anymore but a sum of exponentials: The system maintains more information now, each molecule instance is guaranteed to wait until the previously entered molecules have been consumed, and each reaction step occurs with an exponentially distributed time interval.

The second drawback is that we cannot group arbitrary species together anymore as postulated in Chapter 7. Consider Figure 12.14 once more. Species C_a and C_b are produced by reactions r_{3a} and r_{3b}, respectively, which consume molecules from the same species. The two C-species also produce the same products via reactions r_{4a} and r_{4b}. If the molecule order does not matter C_a and C_b can be summarized into one species, together having the same dynamics as a when separated. This does not hold anymore for the FIFO reaction discipline as two parallel queues show a different behavior in terms of molecule instance reordering than a single queue.

Fraglets VM implementation of queues

Thus, when chemical programs are executed by a Fraglets interpreter, it is important to define *how the Fraglets Virtual Machine implements the molecule queues*: In Fraglets, there is one queue instance for each unique tag, i.e. all fraglets starting with the same tag are put into the same queue. For example, the fraglets `[x a b c]` and `[x d e f]` end up in the same queue even though they differ in their tails. In addition, there is one queue instance for each matching signature: The active fraglets `[match x nop]` and `[match x spush 2]` are considered the same species, because they have the same signature (`x`). They are therefore put into the same queue. The active fraglet inducing a multi-molecular reaction `[mmatch 2 x y nop]` however has the matching signature (`x`, `y`) and is therefore put into a separate queue.

tail-drop behavior

In accordance to the common practice in the Internet, we would also like to implement *tail-drop behavior*. TCP relies on tail-drop unless the selective ACK (SACK) mechanism is used (Mathis et al., 1996). We implement the tail-drop behavior in Fraglets as follows: If the capacity of a reaction vessel is reached, the reactor does not overwrite a random fraglet anymore. It rather drops the newly created or injected molecule.

Deterministic Reaction Intervals *(b)*

Our initial experiment also revealed (and we already knew it from previous examples, e.g. in Section 9.3) that chemical reactions add noise to the rate signal of packet streams. The stochastic reaction interval originally has been introduced to artificial chemical reaction algorithms because molecules undergo Brownian motion and collide randomly in real chemistry. Gillespie (1977) actually appraises that his algorithm is more accurate than a previous one by Bunker, Garrett, Kleindienst, and Long (1974), which calculates the next reaction interval deterministically.

With the aim of making the scheduling of reactions more deterministic, we now follow Bunker et al. (1974). That is, we determine the next reaction interval for each reaction $r_j \in \mathcal{R}$ by using the reciprocal of its propensity value

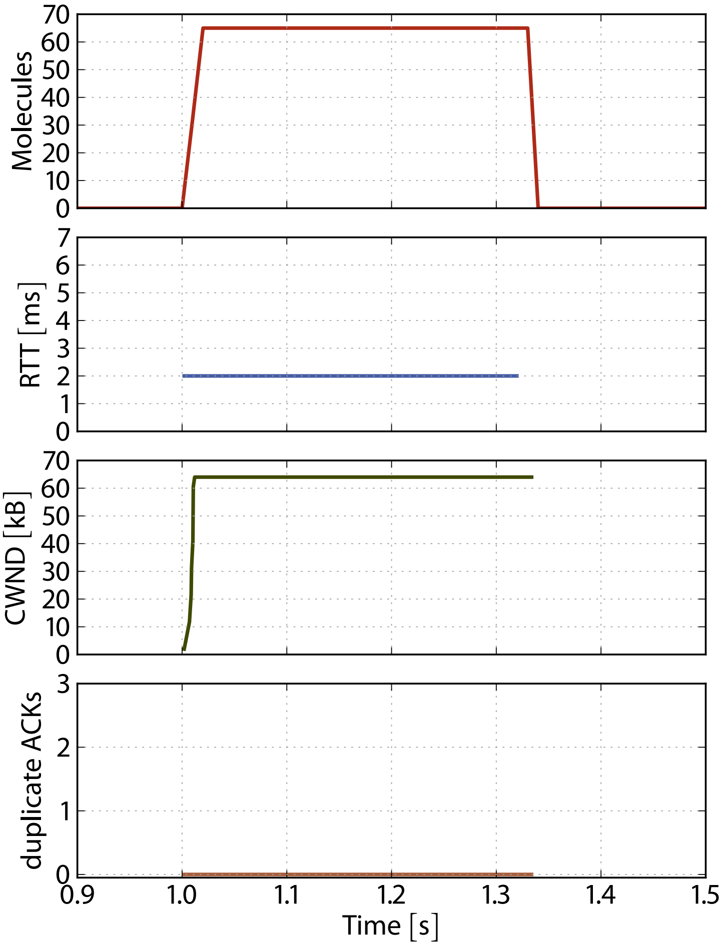


Figure 12.15 TCP loopback through a deterministic vessel — measurements: 10 MB of data is sent from client to server between time $t = 1$ s and $t = 1.338$ s. **Molecules:** Number of IP fraglets buffered in the reaction vessel (both data and acknowledgment packets together). The steady-state quantity is 53 molecules. **RTT:** Round trip time in milliseconds measured by the TCP sender. No fluctuations are observed anymore; the average is at 2 ms as before. **CWND:** Congestion window of the sender in kilobytes. The congestion window grows exponentially in fast start phase. **Duplicate ACKs:** Cumulative number of duplicate acknowledgments arriving at the client; duplicate ACKs are a consequence of reordered packets; no duplicate ACKs are observed anymore.

$\tau_j \sim 1/a_j$ and not $\tau_j \sim \mathrm{Exp}(1/a_j)$ anymore (compare to our original reaction algorithm, Algorithm 3.2 on page 34).

(c) Initial Simulation With the Modified Reaction Algorithm

We now repeat the initial experiment with the same parameters but use FIFO-queues for each species in the reaction vessel and a strictly deterministic reaction scheduler. Compare the new measurements in Figure 12.15 with those for the original algorithm in Figure 12.13.

TCP now achieves a throughput of 29.6 MB/s, which is 95 % of the theoretical maximum throughput for a delay of 2 ms. The full theoretical throughput cannot be achieved due to of the ramp-up behavior of the congestion window (slow start). Overall, our algorithmic modifications of preserving the packet order in chemical reactions and avoiding jitter by deterministically scheduling the reactions led to an improvement of the throughput by a factor of 19. These modifications are mandatory if we want to use chemical protocols in the Internet core.

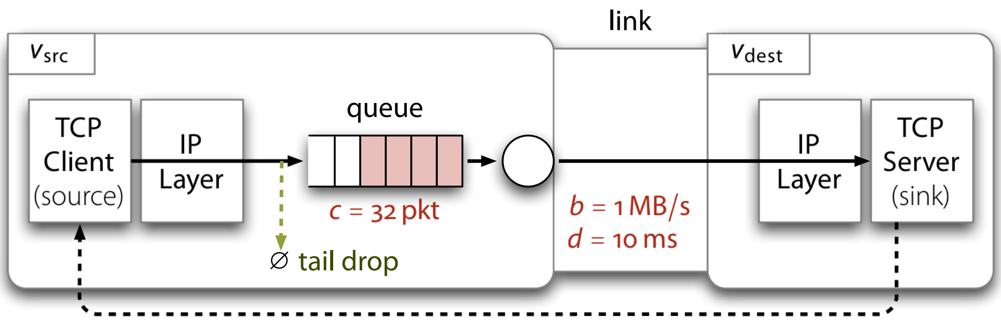


Figure 12.16 TCP over a FIFO queue allocated link — scenario: A TCP stream is sent over a bandwidth limited link, which is allocated by a traditional FIFO queue with capacity $c = 32$ pkt. The queue is serviced as soon as the link is idle. The link has a bandwidth of $b = 1$ MB/s and a delay of $d = 10$ ms. The ACK segments are sent back to the client out-of-band (no bandwidth limitation, no delay).

COMPARING TCP OVER QUEUE-ALLOCATED LINKS TO TCP OVER ENZYMATIC LINKS 12.2.3

We are now ready to directly compare the two link allocation disciplines, traditional queue-based and chemical enzymatic allocation, by using TCP throughput as a benchmark criterion. These simulation results may give more confidence whether in the future, parts of the Internet core could be replaced by artificial chemical reaction vessels that run, for example, chemical routing and forwarding protocols.

Simulation Scenarios (a)

We performed OMNET++ simulations in a simple network topology with two nodes as depicted in Figure 12.16. The TCP client creates a connection to the distant server and sends a block of data. The data stream passes a link with a bandwidth of $b = 1$ MB/s and a delay of $d = 10$ ms. The link bandwidth is allocated by a traditional FIFO queue with a capacity of $c = 32$ pkt. The reverse traffic, which only consists of the acknowledgments in this scenario, is sent back to the client out-of-band. That is, no delay and no bandwidth constraints apply for the ACKs.

This scenario is compared to the chemical counterpart shown in Figure 12.17. Instead of using a queue, the data packets accumulate in the vessel as substrate S of the enzymatic link. We use (12.11a)–(12.11c) on page 208 to determine the three parameters k_1, k_2, and e_0, such that the enzymatic

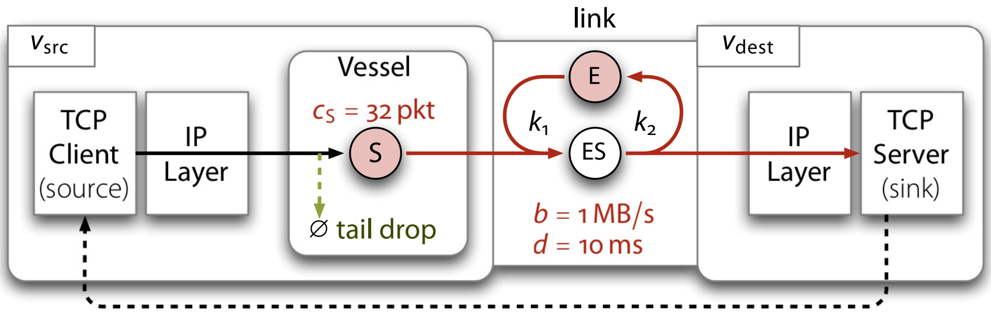


Figure 12.17 TCP over an "enzymatic link" — scenario: A TCP stream is sent via a chemical reaction vessel over a link, which is modeled as enzymatic reaction. The reaction algorithm operates in FIFO mode with the new deterministic reaction scheduler. Packets are tail-dropped if there are more than $c_S = 32$ pkt. To obtain a link bandwidth of $b = 1$ MB/s and a delay of $d = 10$ ms we chose the following parameters: the reaction coefficients are set to $k_1 = 100/(\text{pkt s})$ and $k_2 = 100/\text{s}$, and we initially place $e_0 = 10$ enzymes into the hypothetical link vessel. The ACK segments are sent back to the client out-of-band (not via a chemical reaction vessel, no bandwidth limitation, no delay).

reaction models the given bandwidth limitation and delay, and in order to obtain a reaction rate of $k_{\text{uni}} = 10^4/\text{s}$ if the link is not saturated:

$$k_2 = \frac{1}{d} = 100\,\frac{1}{\text{s}} \tag{12.37a}$$

$$e_0 = bd = 10\,\text{pkt} \tag{12.37b}$$

$$k_1 = \frac{k_{\text{uni}}}{bd} = 100\,\frac{1}{\text{pkt s}} \tag{12.37c}$$

(b) Simulation Results

Figures 12.18 and 12.19 show the measurements of a typical simulation of the traditional and the chemical scenario, respectively. TCP performs slightly better over traditional queues than over enzymatic links: The overall throughput is 0.93 MB/s for the traditional scenario and 0.89 MB/s for the chemical model.

Both figures show the typical saw-tooth pattern of TCP: Since the slots of the queue (and of the substrate molecule) are limited, a packet is eventually dropped if the queue is full. As a consequence, the congestion window (CWND) is divided by two and starts to linearly increase again.

In the traditional scenario, if the queue is empty the round trip time (RTT) is minimal and equal to the link delay (10 ms). For the enzymatic link scenario, the minimal RTT is 11 ms. The additional delay of 1 ms is the reciprocal of the reaction coefficient $k_{\text{uni}} = 10^4/\text{s}$. The difference in the RTT

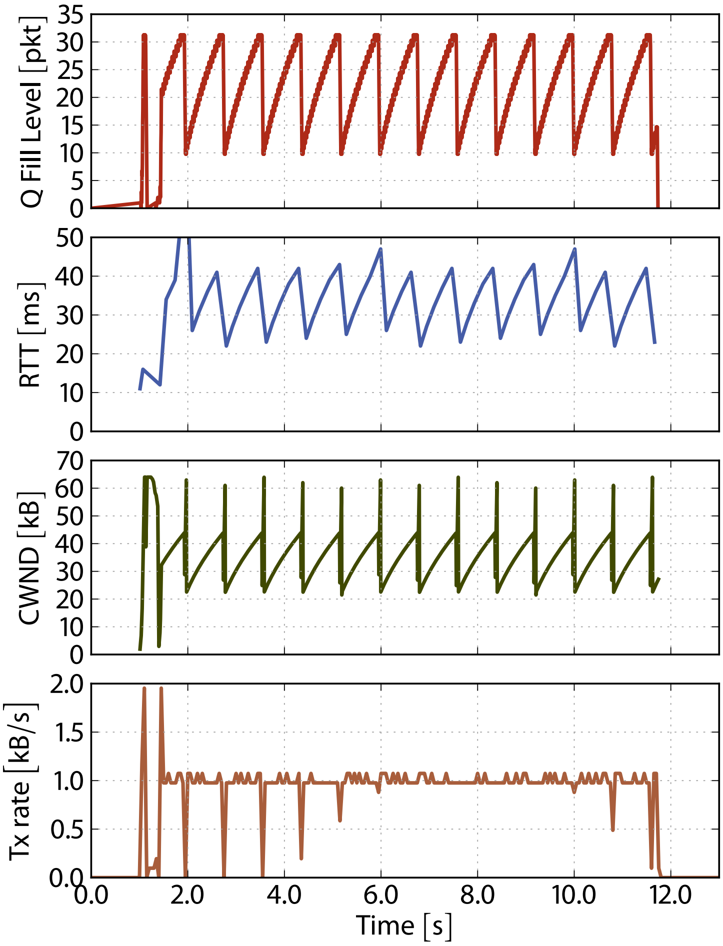


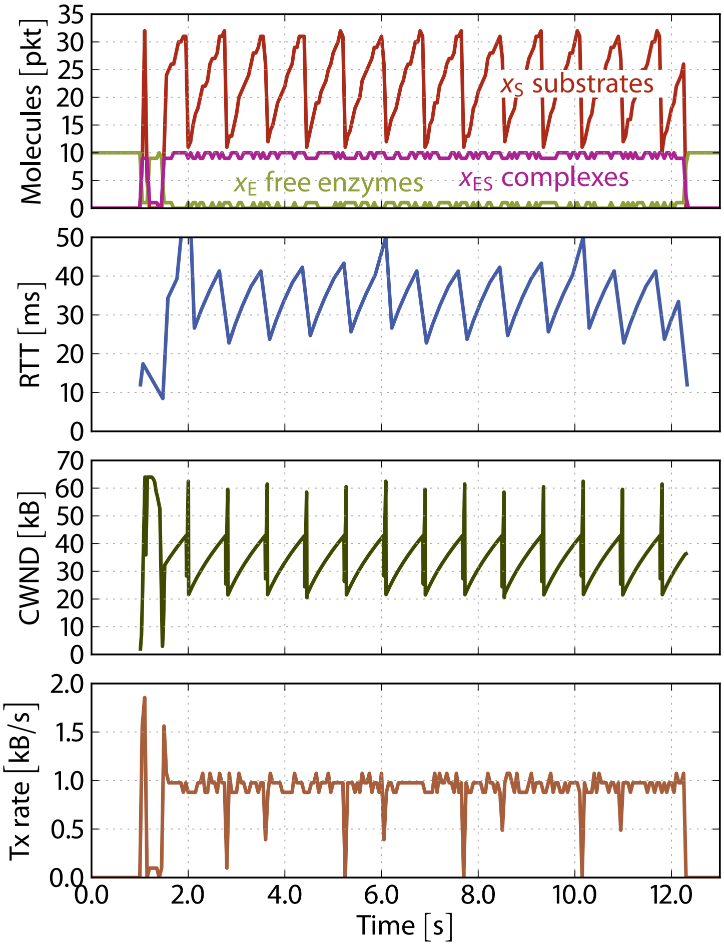
Figure 12.18 TCP over a FIFO queue allocated link — measurements: 10 MB of data is sent from client to server between time $t = 1$ s and $t = 11.73$ s. **Q Fill Level:** The current number of packets in the queue. A packet that arrives while the queue is full (contains 32 packets) is dropped. **RTT:** Round trip time in milliseconds, measured by the TCP sender. The RTT is 10 ms if the queue is empty and linearly increases with the fill level. **CWND:** Congestion window of the sender in kilobytes. The congestion window grows exponentially in the faststart phase and linearly in the congestion avoidance phase. **Tx Rate:** The transmission rate of the data stream, measured before the queue. TCP correctly controls its transmission rate down to the available path bandwidth.

explains why the enzymatic link has some performance drawbacks compared to the traditional queue allocated link.

DISCUSSION 12.2.4

Form these experiments we conclude that TCP does not perform well over a stochastically scheduled chemical forwarding engine. We were forced to bend the chemical metaphor towards traditional packet processing in order to increase the throughput to an acceptable value. The difference between FIFO-queue operation and chemical reaction scheduling with the modified, deterministic algorithm is marginal. The longer RTT of the chemical forwarding engine could even be removed by letting the saturation curve of the enzymatic reaction become very steep. For $k_{\mathrm{uni}} \rightarrow \infty$, i.e. for $K_{\mathrm{M}} \rightarrow 0$, the saturation curve approaches the curve of the traditional queue. That is, packets are then drained from the queue with the maximum rate (link bandwidth) as soon as there is a single substrate molecule. Our modified deterministic

Figure 12.19 TCP over an "enzymatic link" — measurements: 10 MB of data is sent from client to server between time $t = 1$ s and $t = 12.27$ s. **Molecules:** Number of substrate molecules S, free enzymes E, and bound enzyme-substrate complexes ES. If the link is in saturation, all enzymes are bound. **RTT:** Round trip time in milliseconds measured by the TCP sender. The RTT can be calculated according to Table 12.1 on page 210. **CWND:** Congestion window of the sender in kilobytes. The congestion window grows exponentially in the faststart phase and linearly in the congestion avoidance phase. **Tx Rate:** The transmission rate of the data stream, measured before the queue. TCP also correctly controls its transmission rate down to the available path bandwidth when operated over an "enzymatic link".



scheduling algorithm was very successful and must be considered for using chemical networking protocols in the Internet core.

12.3 CHEMISTRY AT THE EDGE: C3A — A TCP-FRIENDLY CHEMICAL CONGESTION CONTROL PROTOCOL

Instead of replacing parts of the network's core with chemical protocols, a more likely scenario is that some network services at the edge are implemented using the chemical model. We envision Internet nodes that autonomically find each other and build an overlay network. Their communicating molecules would be encapsulated into IP packets and sent over the Internet core.

One problem that arises when sending traffic streams over the Internet is congestion. In the previous section, we already reviewed the approach of TCP Reno to control its transmission rate in order to avoid congestion. In this section, we develop C3A, a Chemical Congestion Control Algorithm aimed at controlling its own transmission rate such that TCP-fairness is achieved. It does not perform symbolic computation on packet loss probabilities or round trip times, but rather builds a reaction network in which the control loop emerges, i.e. by relying on the dynamic behavior of the spanned chemical reaction network.

We first discuss existing congestion control approaches in Section 12.3.1 before we present our chemical algorithm (C3A) in Section 12.3.2. In Section 12.3.3, we analyze its fairness theoretically, and we verify these results empirically with OMNET++ simulations in Section 12.3.4. We study the efficiency and fairness of our algorithm while it competes for congested link resources against another chemical streams or against TCP streams. In Section 12.3.5, we propose C3A$^+$, an improvement to our congestion control algorithm, which avoids congestion by observing variations of the round trip time; Section 12.3.6 finally discusses the performance measurements of the new algorithm.

RELATED WORK 12.3.1

The Internet is an inherently unfair communication medium. Its core only provides state-less best-effort packet delivery whereas all protocol logic is located at the edge, i.e. provided by the application. This end-to-end philosophy makes it mandatory for transport protocols to cooperate with each other when transmitting packet streams. TCP, the transmission control protocol of the Internet, contains a congestion control algorithm, which ensures global fairness by performing local control. The basic idea of congestion control is to throttle the transmission rate such that each virtual connection obtains the same bandwidth share.

F. P. Kelly (1997) provided a thorough mathematical analysis of flow rate *max-min fairness*, which is achieved when: (1) the minimum data rate that a data flow achieves is maximized, (2) the second lowest level data rate that a data flow achieves is maximized. Kelly formulated fairness as an optimization problem and showed that there are two different approaches to solve it: primal and dual algorithms (Kelly, Maulloo, & Tan, 1998; F. P. Kelly, 2003).

max-min fairness

End-to-end primal algorithms either make use of explicit congestion signals such as the Explicit Congestion Notification (ECN) flag (Ramakrishnan, Floyd, & Black, 2001) provided by the network, or they use implicit congestion indications such as the detection of lost packets. TCP Reno uses the latter approach, initially proposed by Jacobson (1988): The sender maintains a transmission window, which determines the amount of unacknowledged transmitted data in the network. This window is linearly increased in order to probe for the available bandwidth of the path. If a packet is lost the window additive increase / multiplicative decrease

is divided by two and the transmission rate is therefore reduced drastically. Chiu and Jain (1989) proved that such an *additive increase / multiplicative decrease* mechanism is required to reach stability and fairness among streams operating under the same regime; Sastry and Lam (2005) studied the effect of other increase / decrease functions.

Dual algorithms, on the other hand, make use of changes in the propagation delay to control the transmission rate. The basic idea behind this algorithm is the observation that queuing delays increase before the links are congested. A dual algorithm, such as the one used in TCP Vegas (Brakmo & Peterson, 1995), measures the minimum round trip time (RTT) and continuously observes the RTT during transmission. The algorithm reduces the transmission rate as soon as the delay increases. TCP Vegas has been studied extensively by several modeling approaches (Samios & Vernon, 2003), analysis (Low, Peterson, & Wang, 2002), and empirical validation (Athuraliya & Low, 2001).

The end-to-end paradigm of the Internet also requires non TCP-streams to control their transmission rate in order to behave "TCP-friendly". Applications level protocols may, for example, use the Datagram Congestion Control Protocol (DCCP) (Kohler, Handley, & Floyd, 2006), which makes use of criteria for TCP-friendly Rate Control (TFRC) defined by Handley, Floyd, Padhye, and Widmer (2003), Floyd, Handley, Padhye, and Widmer (2008). This requires the application to measure the round trip time and to use it together with the packet loss frequency to compute the transmission rate.

12.3.2 C3A — A CHEMICAL CONGESTION CONTROL ALGORITHM

The aim of our Chemical Congestion Control Algorithm (C3A) is to regulate the transmission rate of molecules over the Internet. We do not re-implement all features of TCP. For example, since the chemical world is random by nature, we are not interested in reliable transmission and hence do not care about retransmitting lost or reordered packets. The only goal is that a stream of molecules sent over the Internet is reasonably fair to TCP streams when competing for the same bandwidth resources.

Inspired by the additive increase and multiplicative decrease behavior of TCP Reno (Jacobson, 1988; Chiu & Jain, 1989), we came up with a reaction network that regulates the sender's transmission rate based on lost packet signals. Figure 12.20 shows the reaction network of our algorithm. The source node sends data packets D over the Internet to the destination node, which acknowledges each received molecule by an ACK packet. The source node controls the transmission rate r_{tx} based on this feedback. Let us study the components of the reaction network in more detail:

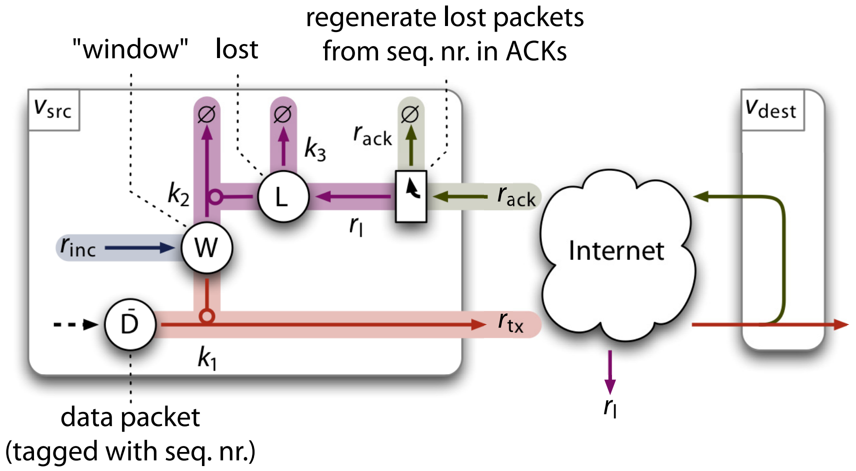
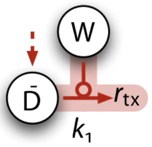
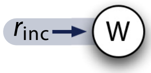


Figure 12.20 Reaction network of C3A: Window molecules W are continuously increased. The transmission rate r_{tx} is proportional to this quantity. Data packets and acknowledgments are tagged with a sequence number such that the lost packets, represented by molecules L, can be regenerated from the stream of acknowledgments. An L-molecule catalyzes the decay of window molecules (r_2) until it is eventually decayed itself (r_3).

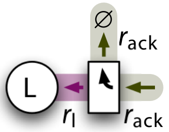
The application layer (not shown in the figure) places one data packet to the reaction vessel as molecule D. After it has been sent, the application immediately places the next data packet from its transmit queue. In chemical terms, the data species D is buffered: it contains a constant amount of molecules: $x_D = 1\,\text{pkt}$. Additionally, the application tags each data packet with a continuously increasing sequence number.



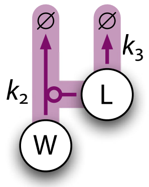
TCP Reno controls the packet transmission rate via the size of the congestion window of the sender together with the receive window advertised by the receiver. In our chemical algorithm, we do not limit the amount of unacknowledged bytes in the network, but directly regulate the transmission rate by the multiplicity of "window"-molecules W. According to the law of mass action, the transmission rate is proportional to the number of window molecules: $r_{tx} = k_1 x_W x_D$. Note that the window molecules are not consumed by the transmit reaction, meaning that data packets are sent at a constant rate if the number of W-molecules does not change.



A zero-order reaction continuously increases the number of window molecules W at rate r_{inc}. Consequently, the transmission rate increases linearly over time; this is the process responsible for the additive increase.



The source node inspects the sequence number of the received acknowledgment packets and generates an L-molecule for each missing sequence number. That is, the number of L-molecules reflects the number of lost packets.



The last reaction motif decreases the transmission rate based on the packet loss feedback: A fast reaction (r_2) removes window molecules W in proportion to the number of packets lost. At the same time, lost packets decay from L via reaction r_3. That is, each lost packet drains window molecules for a certain duration until it is decayed itself. This leads to an exponential decrease of the transmission rate. After some time, when the L molecules are gone, the transmission rate linearly increases again.

We implemented this algorithm in Fraglets, studied its behavior analytically and performed network simulations in OMNET++.

12.3.3 FORMAL ANALYSIS OF C3A

We first analyze the dynamic behavior of C3A using deterministic ODEs in order to find out whether our algorithm is TCP-friendly. According to Floyd and Fall (1999) (see also Floyd et al., 2008), the transmission rate of a TCP-fair protocol should be proportional to $1/\sqrt{p_l}$ where p_l denotes the packet loss rate.

Let us therefore have a look at the reaction rates of the algorithm: Packets are sent at rate $r_{tx} = k_1 x_W x_D$. During their round-trip journey through the Internet they are being lost with probability p_l. Thus, the source node receives acknowledgments at rate $r_{ack} = (1-p_l) r_{tx}$. By inspecting the sequence number in the ACK packets the source node is able to regenerate a stream of lost packets with rate $r_l = p_l r_{tx}$.

Having the important rates defined, we are able to write down the differential equation system of this transmission control loop:

$$\dot{x}_W = r_{inc} - k_2 x_W x_L \qquad (12.38a)$$

$$\dot{x}_L = \underbrace{p_l k_1 x_W x_D}_{r_l} - k_3 x_L \qquad (12.38b)$$

At equilibrium ($\dot{x}_W = \dot{x}_L = 0$) the transmission rate settles at

$$\hat{r}_{tx} = \sqrt{\frac{k_1 k_3 x_D r_{inc}}{k_2 p_l}} \qquad (12.39)$$

Indeed, we note that the equilibrium transmission rate is proportional to the reciprocal of the square root of the loss probability.

parameterization

The chemical reaction network contains four *parameters* to be specified: the injection rate of window molecules, r_{inc}, and the three reaction coefficients k_1, k_2, and k_3. We studied the model using the signal theoretic methods reviewed in Section 8.2.3 (Monti, 2010), and verified candidate parameter sets in OMNET++ simulations. The objective of this optimization was fast convergence without suffering too heavy oscillations. The following streamlined parameter set performed well:

$$r_{\mathrm{inc}} = 10^3\,\frac{\mathrm{pkt}}{\mathrm{s}} \tag{12.40a}$$

$$k_1 = 1\,\frac{1}{\mathrm{pkt\,s}} \tag{12.40b}$$

$$k_2 = 10^4\,\frac{1}{\mathrm{pkt\,s}} \tag{12.40c}$$

$$k_3 = 10^5\,\frac{1}{\mathrm{s}} \tag{12.40d}$$

yielding an equilibrium transmission rate of

$$\hat{r}_{\mathrm{tx}} = \frac{100\,\mathrm{pkt/s}}{\sqrt{p_1}} \tag{12.41}$$

FAIRNESS AND TCP-FRIENDLINESS OF C3A IN SIMULATIONS 12.3.4

In the following, we present and discuss some of the performed network simulation results of C3A and show how it performs when sharing a bandwidth limited link with another chemical molecule stream or with a TCP packet stream.

Simulated Network Topology *(a)*

All simulations discussed in this section were performed in OMNET++, which simulates the network topology depicted in Figure 12.21. Depending on the scenario, up to three source nodes send data packets to the corresponding destination nodes at the other end of the network. All data streams have to traverse the shared link between router v_{A} and router v_{B}. This link has a bandwidth of $b = 1\,\mathrm{MB/s}$ and a delay of $d = 10\,\mathrm{ms}$. A traditional FIFO queue in front of this link buffers up to $c = 32\,\mathrm{pkt}$; all other links are ideal. The acknowledgments are sent back to the source nodes out of band, meaning that they are neither lost nor afflicted with a delay.

The edge nodes 1 and 2 contain Fraglets reaction vessels, which are scheduled by our modified deterministic reaction algorithm that maintains the packet order. Source nodes $v_{\mathrm{src},1}$ and $v_{\mathrm{src},2}$ contain an implementation of C3A. Node $v_{\mathrm{src},3}$ and $v_{\mathrm{dest},3}$ run a TCP Reno client and server, respectively.

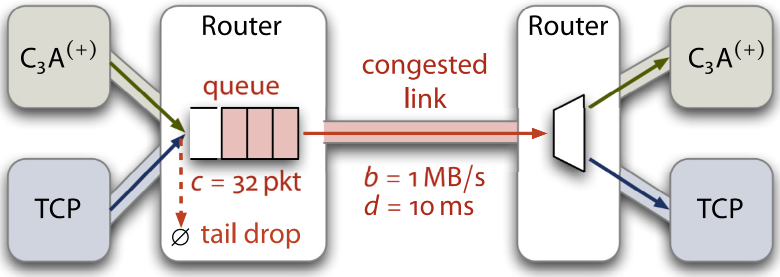
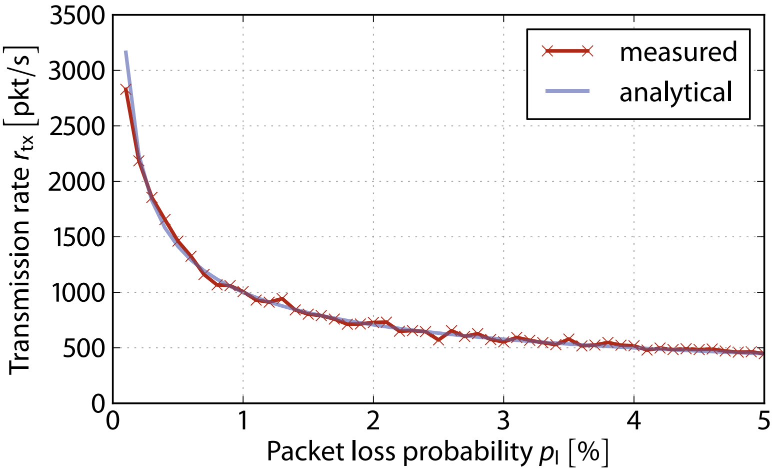


Figure 12.21 **Network topology for congestion control simulations**: Three source nodes send packet streams to the corresponding destination nodes over the same link with limited bandwidth $b = 1\,\mathrm{MB/s}$ and delay $d = 10\,\mathrm{ms}$ between router v_A and router v_B. Router v_A contains a queue with capacity $c = 32\,\mathrm{pkt}$. The acknowledgment packets are sent back out-of-band.

Figure 12.22 **Packet-loss dependent transmission rate in C_3A**: The analytical curve is a plot of (12.39), which predicts that the transmission rate drops with respect to the packet loss probability with a square root relation. Complementary OMNeT++ measurements demonstrate that our Fraglets implementation of C_3A exhibits the designed dynamic behavior.



(b) *Verification of the Analytical Transmission Rate Prediction*

Before we demonstrate how our congestion control algorithm performs in detail, with Figure 12.22, we highlight that our Fraglets implementation of C3A exhibits the analytically predicted transmission rate. For this simulation, only edge nodes $v_{\mathrm{src},1}$ and $v_{\mathrm{dest},1}$ were active. We parameterized the network with the packet loss ratio between router v_A and v_B rather than with the bandwidth. Starting with a packet loss of $p_l = 0.1\,\%$ we continuously increased it in steps of 0.1 % up to a value of $p_l = 5\,\%$. In each simulation run, we waited until the reaction network reached equilibrium before we measured and averaged the transmission rate over 10 s. As depicted in Figure 12.22, the obtained measurements accurately follow the square root dependency predicted by (12.39).

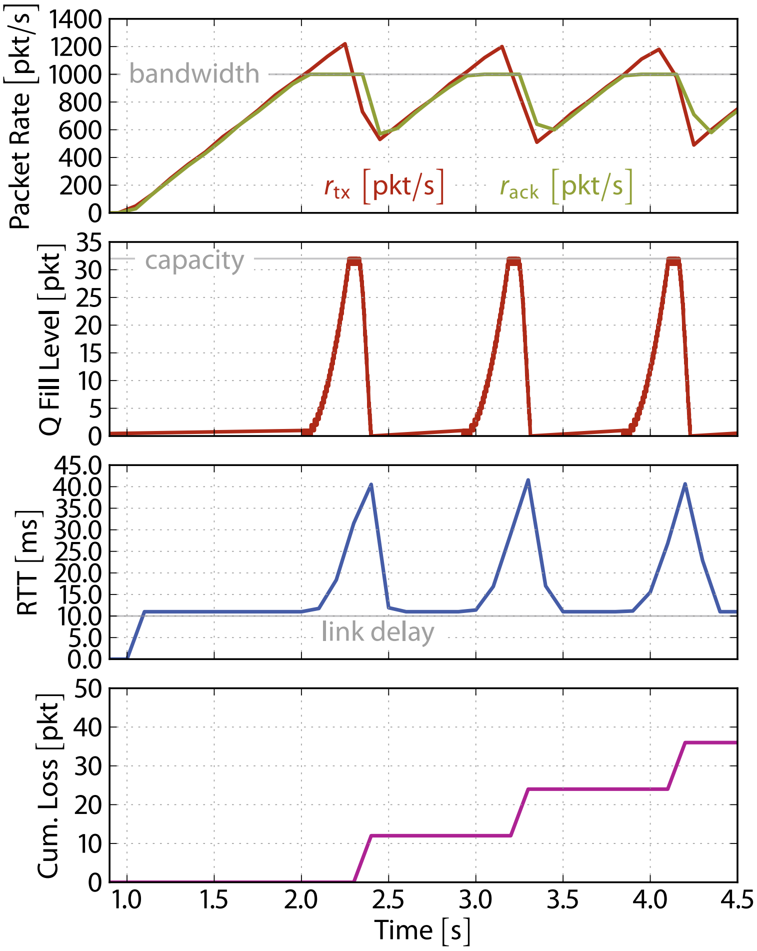


Figure 12.23 Single molecule stream under congestion control: C3A sends data from source node $v_{src,1}$ to destination node $v_{dest,1}$. **Packet Rate:** Rate at which data molecules are sent (r_{tx}) and acknowledgments are received (r_{ack}) in node $v_{src,1}$. Obviously, the acknowledgment rate cannot exceed the link bandwidth $b = 1$ MB/s. **Q Fill Level:** Number of packets buffered in the FIFO queue in router v_A. If the transmission rate r_{tx} is larger than the bandwidth, the queue is filled and starts to tail-drop packets. **RTT:** Round trip time in milliseconds measured by the source. The RTT linearly increases with the fill level of the queue. **Cum. Loss:** Cumulative number of lost packets. Packets arriving at the queue when it is full are lost. Because of the non-zero RTT, the algorithm's response to a packet loss is deferred, which results in a sequence of 12 packets being lost at a time.

Signature of a Single Molecule Stream under Congestion Control (c)

Figure 12.23 shows the time evolution of some important measures for a single data stream between source node $v_{src,1}$ and destination node $v_{dest,1}$. The other edge nodes in the network depicted in Figure 12.21 are deactivated. The link bandwidth between router v_A and router v_B is set to $b = 1$ MB/s and it delay to $d = 10$ ms.

The top subfigure illustrates how the transmission rate r_{tx} linearly increases as a consequence of the continuous injection of window molecules. The rate of received acknowledgments r_{ack} cannot rise above the bandwidth. However, this cap of the acknowledgment rate does not mean that the remaining packets are lost: First, the queue in router v_A is filled as indicated in the second subfigure. As a side effect, the round trip time (RTT) is increased as shown in the third subfigure. As soon as the queue is full, packets are tail-dropped. The sender continues to increase the transmission rate until it receives an ACK packet with a sequence number jump. Then, as indicated in

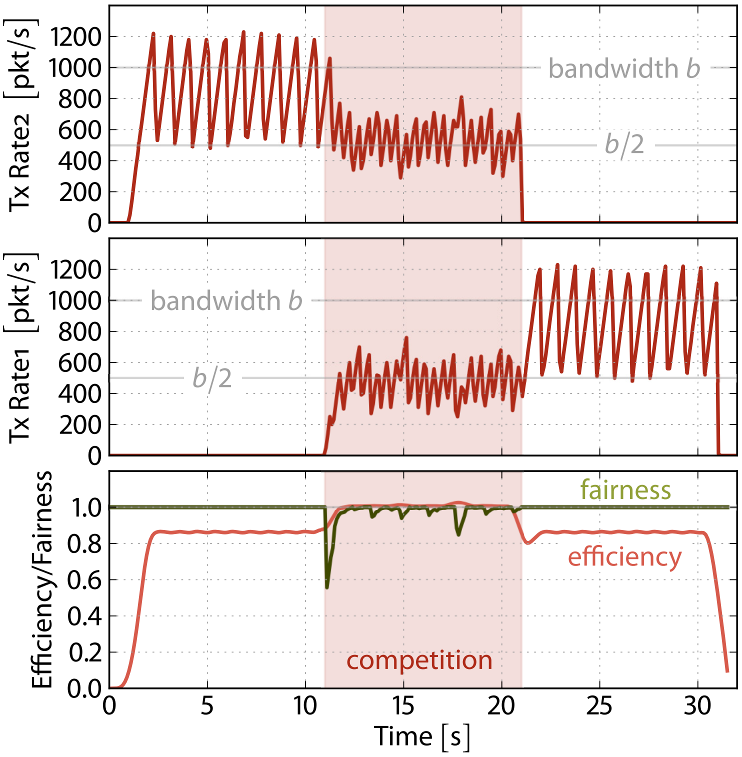


Figure 12.24 Efficiency/Fairness of two C3A-controlled streams: Two source nodes, both controlled by C3A, send data over the congested link. The first source node, $v_{\mathrm{src},1}$, is active between time $t = 1$ s and $t = 21$ s while the second source node, $v_{\mathrm{src},2}$ sends data molecules between time $t = 11$ s and $t = 31$ s. For ten seconds the two data streams compete for bandwidth resources. The bottom subfigure shows the efficiency and fairness of this bandwidth allocation, calculated by using Jain's fairness index (Jain, Chiu, & Hawe, 1984). The efficiency curve has been smoothed using a Hann window of size 2 s.

the top subfigure, it immediately drains about half of the window molecules and thus lowers the transmission rate drastically.

The behavior of the chemical congestion control reaction network shows a similar additive increase / multiplicative decrease signature as TCP Reno in congestion avoidance mode. Unfortunately, C3A loses about 12 molecules each time it runs the link into congestion. This is due to the delay in the control loop. This behavior cannot be avoided by just using different reaction coefficients. We will later introduce an improved version with an improved yield.

(d) *Competition among Two C3A-Controlled Molecule Streams*

For the next simulation, we let two sources send data over the same bandwidth limited link and compute efficiency and fairness measures. We define the *efficiency* of an allocation for n streams, each with a transmission rate of $r_{\mathrm{tx},i}$ $(1 \leq i \leq n)$, as the total utilization of the bandwidth:

efficiency

$$e(\mathbf{r}_{\mathrm{tx}}) = \frac{\sum_{i=1}^{n} r_{\mathrm{tx},i}}{b} \qquad (12.42)$$

fairness

The *fairness* is computed according to Jain, Chiu, and Hawe (1984), measuring the "equality" of allocation:

$$f(\mathbf{r}_{\text{tx}}) = \frac{\left(\sum_{i=1}^{n} r_{\text{tx},i}\right)^2}{\sum_{i=1}^{n} r_{\text{tx},i}^2} \qquad (12.43)$$

If all streams are sent with the same rate, then the fairness index is one, i.e. the allocation is 100 % fair. As the disparity increases, fairness decreases and approaches zero if a few streams get the full bandwidth.

In the network topology depicted in Figure 12.21, we activated source $v_{\text{src},1}$ at time $t = 1$ s for the duration of 20 s, and ten seconds later, at time $t = 11$ s, we activated source $v_{\text{src},2}$ for the same duration. That is, for ten seconds the two packet streams compete for the limited link bandwidth between router v_{A} and v_{B}.

Figure 12.24 shows the transmission rates of the two streams and, in the subfigure at the bottom, the efficiency and fairness of the bandwidth allocation. The first stream correctly decreases its average transmission rate such that the fairness is close to the optimum. Note that a single stream is not able to fully exploit the path bandwidth alone due to the saw-tooth pattern of the transmission rate. A second stream however, is able to fill the remaining bandwidth such that in a competitive situation the maximum efficiency is reached.

Competition among Chemical and TCP Streams (e)

As a next step, we would like to make sure that a molecule stream, controlled by C3A, is fair to a competing TCP packet stream. We therefore perform the same simulation as before, but this time, we enable TCP client $v_{\text{src},3}$ instead of $v_{\text{src},2}$ and let it compete with the molecule stream from source $v_{\text{src},1}$ (see the network topology in Figure 12.21 on page 234).

Figure 12.25 shows the transmission rates of the two streams together with the efficiency and fairness of the bandwidth allocation. The fairness is good despite some fluctuations and the two streams together fully exploit the link bandwidth.

C3A$^+$ — A CONGESTION AVOIDANCE EXTENSION FOR C3A 12.3.5

The congestion control algorithm presented so far uses packet loss signals as indications for a congested path. It is actually the ACK that is received *after* a packet is lost that causes the algorithm to reduce its transmission rate. Even worse, the non-zero round trip time together with this indirect signaling mode adds a delay into the congestion control loop. Consequently, the reaction network lowers the transmission rate only after a dozen of packets

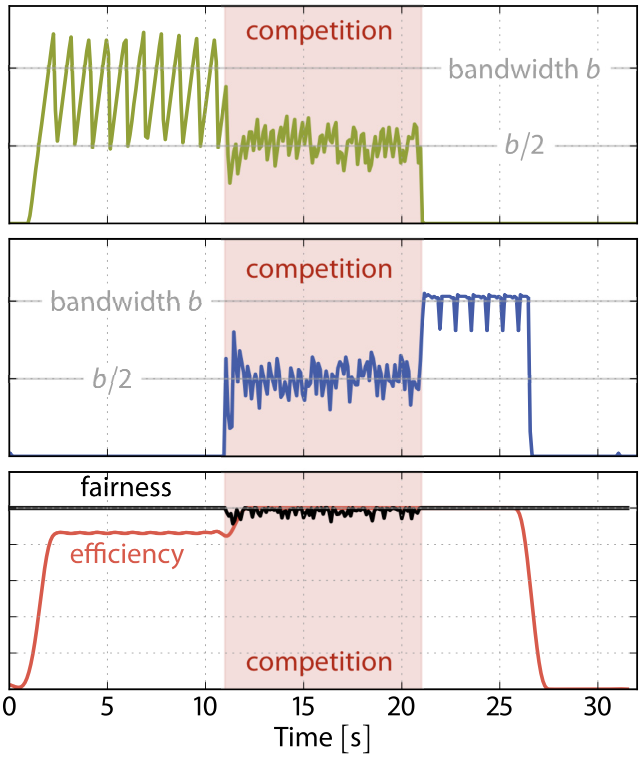


Figure 12.25 Efficiency/Fairness of a C_3A-controlled stream in competition with a TCP stream: Source node $v_{src,1}$, controlled by C_3A, and TCP client $v_{src,3}$ send data over the congested link. The first node is active between time $t = 1$ s and $t = 21$ s while the TCP client sends 10 MB of data starting from $t = 11$ s. For ten seconds the two data streams compete for bandwidth resources. The bottom subfigure shows the efficiency and fairness of this bandwidth allocation, calculated by using Jain's fairness index (Jain et al., 1984). The efficiency curve has been smoothed using a Hann window of size 2 s.

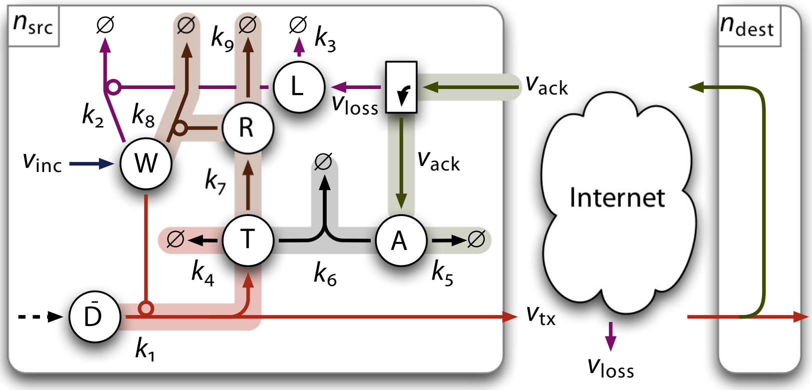


Figure 12.26 Reaction network of C_3A^+: ACK molecules are now stored in species A whereas a copy of each transmitted packet is stored in T. The fast reaction r_6 eliminates pairs of A- and T-molecules such that T reflects the difference between r_{tx} and r_{ack}. The window W is decreased in proportion to this value similar as for a lost packet, but with a smaller rate.

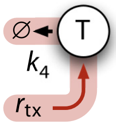
have already been lost. Here, we present C3A$^+$, an extension to our chemical congestion control algorithm that senses the round trip time variation of a packet stream and avoids congestion by throttling its transmission rate early.

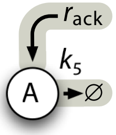
TCP Vegas performs congestion avoidance by measuring the RTT variation above the minimum RTT and by regulating down its transmission rate early, as soon as the RTT rises. In a chemical setting, we cannot and do not want to measure the minimum RTT explicitly. Instead of collecting and computing path statistics, we would like to use rates and molecular quantities to determine and react to sudden changes in the RTT.

Consider again the packet rate curves in Figure 12.23 on page 235. When the path bandwidth is fully exploited we observe a sudden cap of the rate of received acknowledgments while the transmission rate continues to increase. This difference between transmission rate and acknowledgment rate either indicates a dramatic loss of packets or a sudden increase of the RTT due to a queue that is being filled. In both cases we would like to lower the transmission rate.

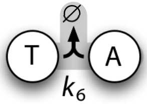
Figure 12.26 depicts the extended reaction network that takes the variation of the RTT into account; added reactions are highlighted. In the following paragraphs, we demonstrate how we designed this extension step by step with the help of the motifs introduced in Section 10.3.

We first convert the transmission rate r_{tx} into a molecular quantity by using the motif discussed in Section 10.3.2(a): Each data packet sent to the destination is at the same time copied (without payload) to an instance of species T where it is decayed by a unimolecular reaction with coefficient k_4. The steady-state number of T-molecules is $\hat{x}_{\text{T}} = \hat{r}_{\text{tx}}/k_4$. We install a similar reaction for the received acknowledgments. The number of A-molecules is $\hat{x}_{\text{A}} = \hat{r}_{\text{ack}}/k_5$ at equilibrium.

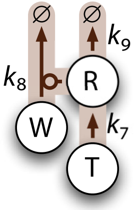




In a next step, we install a motif that continuously computes the difference between the two rates: The bimolecular reaction r_6 removes T- and A-molecules pair wise. We chose a high reaction coefficient k_6 such that two instances of the two species immediately annihilate each other, such that either T or A reflects the difference of the two rates, depending on which rate is higher.



Finally, we use the resulting number of T-molecules ($\hat{x}_{\text{T}} \propto |\hat{r}_{\text{tx}} - \hat{r}_{\text{ack}}|$) to lower the transmission rate: Reaction r_7 creates R-molecules as long as the transmission rate is higher than the rate of received acknowledgments. These molecules destroy window molecules and therefore reduce the transmission rate. This decay of window molecules has to be done moderately: When the algorithm starts sending data packets there are no acknowledgments yet which obviously results in a higher transmission rate than acknowledgment rate. Thus the effect of the decay reaction has to be smaller than the rate of injected window molecules. The algorithm should probe for the RTT by slowly increasing the transmission rate and should retract when observing a sudden difference in the rates.



12.3.6 SIMULATIONS OF C3A$^+$

We performed the same OMNET++ simulations again for the extended algorithm with the following reaction coefficients for the newly introduced reactions:

$$k_4 = k_5 = 10\,\frac{1}{\mathrm{s}} \tag{12.44a}$$

$$k_6 = 10^3\,\frac{1}{\mathrm{pkt\,s}} \tag{12.44b}$$

$$k_7 = 100\,\frac{1}{\mathrm{s}} \tag{12.44c}$$

$$k_8 = 50\,\frac{1}{\mathrm{pkt\,s}} \tag{12.44d}$$

$$k_9 = 2 \times 10^3\,\frac{1}{\mathrm{s}} \tag{12.44e}$$

(a) Signature of a Single Molecule Stream under Congestion Control

Figure 12.27 shows the time evolution of some important measures for a single data stream between source node $v_{\mathrm{src},1}$ and destination node $v_{\mathrm{dest},1}$ while the other sources are deactivated.

The extended version of our congestion control algorithm increases the transmission rate slower compared to the original version. As mentioned before, the minimum RTT causes an initial difference between transmission and acknowledgment rate at the source node, which partly inhibits the increase of window molecules.

congestion avoidance mode

In the next phase, starting from $t = 4$ s, the algorithm is in *congestion avoidance mode*: The transmission rate reached the link bandwidth and the fill level of the queue grows. This is accompanied by an increase of the RTT, which causes the algorithm to slow down the increase of its transmission rate.

congestion control mode

However, unlike TCP Vegas, the algorithm does not know the minimal RTT as a reference. Thus, the transmission rate never stops to increase; the queue eventually gets filled and tail-drops a packet. This is when the algorithm switches to *congestion control mode*, meaning that the reactions already present in the original version divide the number of window molecules by approximately two. Consequently, the transmission rate is reduced drastically and the system starts to oscillate. It seems that congestion avoidance and congestion control phases alternate in turn.

lower packet loss rate

In our simulations of C3A$^+$, the *packet loss rate* was 5 to 10 times *lower* than in comparable simulations of C3A, which is a consequence of the atten-

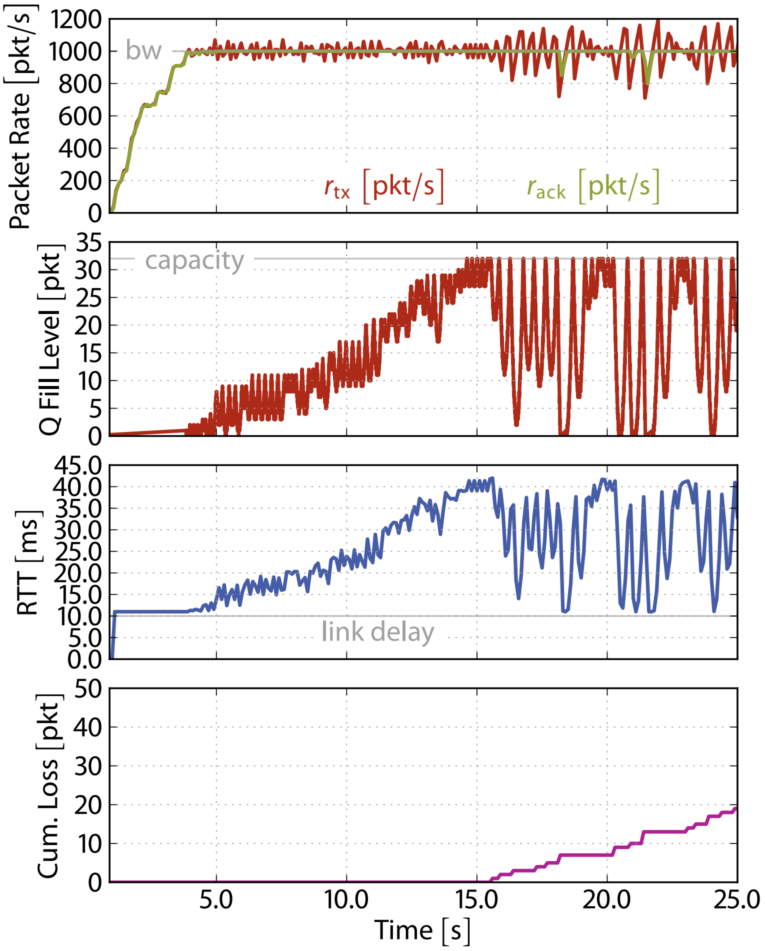


Figure 12.27 Single C_3A^+-controlled molecule stream: The extended chemical congestion control algorithm sends data from source node $v_{src,1}$ to destination node $v_{dest,1}$. **Packet Rate:** Rate at which data molecules are sent (r_{tx}) and acknowledgments are received (r_{ack}) in node $v_{src,1}$. The convergence time is longer in the extended algorithm, but the sawtooth like fluctuations are dampened. **Q Fill Level:** Number of packets buffered in the FIFO queue in router v_A. If the transmission rate r_{tx} is larger than the bandwidth the queue is filled and then starts to tail-drop packets. **RTT:** Round trip time in milliseconds measured by the source. The RTT linearly increases with the fill level of the queue. The extended algorithm slowly exploits the queue's capacity. **Cum. Loss:** Cumulative number of lost packets. Packets arriving at the queue when it is full are lost. The extended algorithm has lower packet loss.

uated increase of the transmission rate. However, this comes along with a longer convergence time.

Competing Streams (b)

We again simulated two streams competing for link bandwidth resources in two scenarios: A molecule stream, controlled by C3A$^+$, is once in competition with a similar, chemically controlled molecule stream, and in the next scenario, it is in competition with a TCP Reno packet stream (compare to the results for the original algorithm discussed in Section 12.3.4(d) and Section 12.3.4(e), respectively).

Figure 12.28 shows transmission rates, allocation efficiency and fairness of two competing chemically controlled streams whereas Figure 12.29 depicts the same results for a competing TCP packet stream. The transmission rate fluctuations are much smaller than before. Unlike for C3A, one single stream controlled by C3A$^+$ is able to exploit the full link bandwidth. The steady-state fairness is slightly worse for the extended algorithm, meaning that the streams

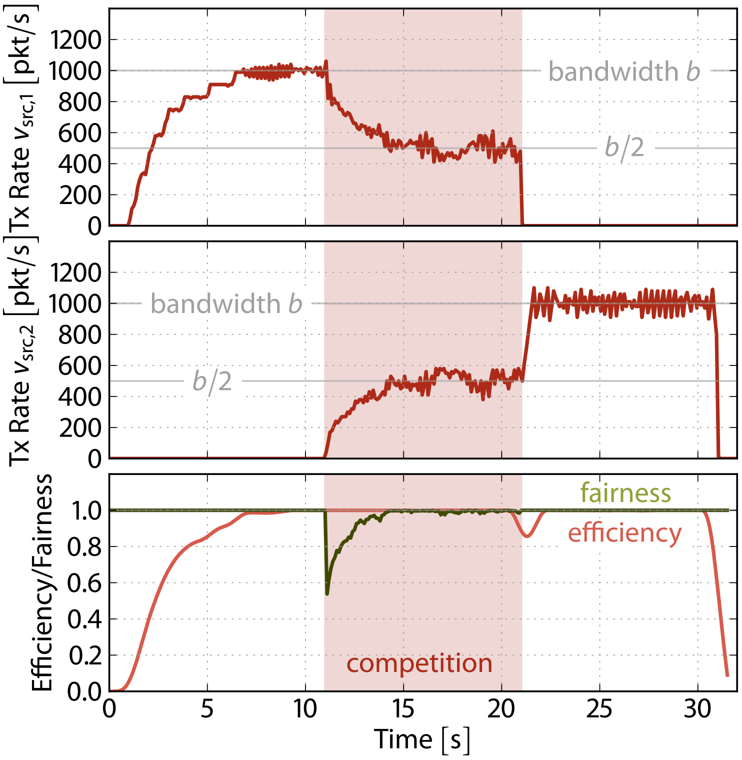


Figure 12.28 Efficiency/Fairness of two C_3A^+-controlled streams: Two source nodes, both controlled by the extended chemical congestion control algorithm, send data over the congested link. The first source node $v_{src,1}$ is active between time $t = 1$ s and $t = 21$ s while the second source node $v_{src,2}$ sends data molecules between time $t = 11$ s and $t = 31$ s. For ten seconds the two data streams compete for bandwidth resources. The bottom subfigure shows the efficiency and fairness of this bandwidth allocation, calculated by using Jain's fairness index (Jain et al., 1984). The efficiency curve has been smoothed using a Hann window of size 2 s.

show larger fluctuations around the fairness equilibrium. The extended algorithm also needs longer to converge. Table 12.2 shows the properties of the two algorithm versions in comparison.

12.4 SUMMARY

In this chapter we assessed the operation principle of chemical networking protocols in a real network environment. First, we came up with chemical models for realistic network components such as queues and bandwidth limited links. Such chemical models of traditional components enable us to use our engineering framework to analyze and proof chemical protocols that are operating in a realistic network. We illustrated this by proofing *Disperser's* convergence in a network with bandwidth-limited links.

The second goal of this chapter was to bridge the gap between the chemical and traditional networking by investigating whether chemistry should be used in the core or at the edge of the Internet. We see a potential for both scenarios:

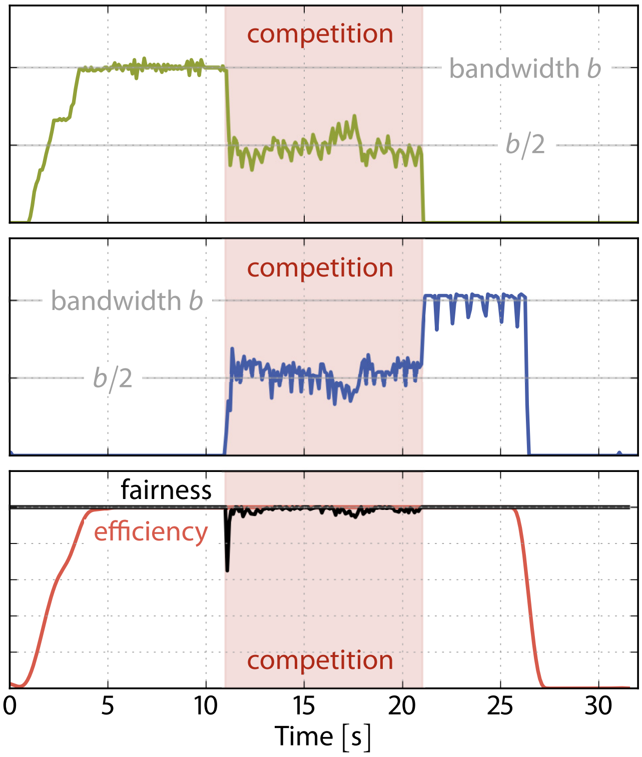


Figure 12.29 Efficiency/Fairness of a C_3A^+-controlled stream in competition with a TCP stream: The source node $v_{src,1}$, controlled by C_3A^+, and the TCP client $v_{src,3}$ send data over the congested link. The first node is active between time $t = 1$ s and $t = 21$ s while the TCP client sends 10 MB of data starting from $t = 11$ s. For ten seconds the two data streams compete for bandwidth resources. The bottom subfigure shows the efficiency and fairness of this bandwidth allocation, calculated by using Jain's fairness index (Jain et al., 1984). The efficiency curve has been smoothed using a Hann window of size 2 s.

Property	C_3A	C_3A^+
Use packet loss signal	strong feedback	strong feedback
Use RTT variation signal	no	weak feedback
Convergence time	faster	slower
Rate fluctuation	strong	weak
Fairness fluctuations	weak	moderate
Single stream efficiency	85 %	100 %
Multiple stream efficiency	100 %	100 %

Table 12.2 Comparison of the two chemical congestion algorithm versions: The original version (C_3A) controls the transmission rate based on singular feedback from packet loss signals whereas the extended version (C_3A^+) additionally uses the round trip time variation to regulate the transmission rate.

CHEMISTRY IN THE CORE 12.4.1

We discovered that a stochastic packet-scheduling algorithm in forwarding engines is poisonous for TCP streams that are tunneled over such a chemical substrate. We were able to mitigate this problem by a modified scheduling algorithm, which behaves more deterministically and does not reorder packets.

absorb bursts

Chemical reactions are also helpful to *absorb bursts* of uncooperative packet streams: Most chemical reactions are low-pass filters. Thus, a chemical forwarding engine will temporarily store a burst of packets locally and only increase its forwarding rate slowly and steadily as depicted in Figure 12.2 on page 204. That is, a packet stream is automatically smoothed early when entering the network, which allows other streams to adapt. This makes the whole system less chaotic.

Our enzymatic MAC protocol exhibits a similar behavior. It does not exploit all available bandwidth for low traffic, but continuously increases the load factor for additional pressure imposed by traversing packet flows. The continuous saturation curve enables end-to-end rate control algorithms to react more smoothly to congestion.

12.4.2 CHEMISTRY AT THE EDGE

We demonstrated the ease of designing a chemical congestion control algorithm, C3A, which exploits the available bandwidth but is fair to molecule streams competing for the same bandwidth and even to TCP streams. Our simulations indicate that chemical and traditional networking protocols are able to cooperate and coexist in the same network. Thus, it is possible to gradually replace existing protocols by chemical protocols, if their advantages are required.

Congestion is the consequence of the best-effort statistical multiplexing service offered by the Internet's core. The Internet is only fair to its users if the users cooperate by using congestion-control algorithms. Such an environment requires protocols to have a macroscopic view to data flows rather than treating each packet individually. Designers of classical protocols spend a lot of time to reconstruct the dynamic flow behavior from the discrete and isochronous runtime behavior that their execution machinery offers.

Unlike other congestion control algorithms, our chemical variant does not compute statistics about the network environment explicitly in order to adapt its transmission rate. It is rather embodied to the environment by its distributed reaction network and senses the network condition implicitly by reacting to slight variations in the packet rates. This behavior is enabled by the law of mass action scheduling, which also makes it easy to design equilibrium solutions.

12.4.3 CLOSING REMARKS

The strength of chemical protocols is to have an engineering framework for organizing packet flows from macroscopic design to microscopic execution and back to macroscopic flow analysis. The strength of this engineering method became apparent while designing and analyzing a complex chemical networking protocol such as C3A. With the help of our reaction motifs, it was quite easy to come up with an acceptable solution, and its mathematical model based on deterministic ODEs to determine the equilibrium solution was applied quickly and easily.

In the previous chapters, we used *Disperser* as a showcase of a typical chemical protocol. *Disperser* is a *pure* chemical protocol in the sense that it uses rate-based encoding to convey user-information. In the Internet context, we rather promote *hybrid* protocols, where user information is transferred symbolically in the payload of data packets, but where the protocol's state is represented by the number of molecules and where "control" emerges by the interaction of the rate-encoded state information along the distributed reaction network with the computer network environment.

pure

hybrid

This chapter closes the second part of this thesis. In the next part we push the chemical metaphor even further by trying to structure chemical execution such that reaction networks emerge that maintain their stability even in presence of execution errors or code mutations, a behavior that is barely possible under the traditional code execution regime.

PART III

Self-Healing Networking Protocols

CHAPTER 13

Self-Healing Software

From related work on self-healing computing systems to requirements for intrinsically self-healing software.

The art of healing
comes from nature,
not from the physician.
Therefore the physician
must start from nature,
with an open mind. [13]

Paracelsus

In the previous part, we showed how chemical networking protocols seek an equilibrium in which they provide the solution to a distributed consensus problem. When perturbed by changes in the network environment, they eventually return to the fixed point. In this part, we apply the same principle to disturbances of the execution platform that runs the protocol software. This results in code that is able to run on unreliable hardware – self-healing software.

This chapter introduces the third part of this thesis, which focuses on self-organizational aspects of chemical code. It may be interesting for researchers in the area of artificial life to see a concrete application case of life-like properties in computer networking. On the other hand, computer scientists find a description of an innovative method to obtain intrinsic code-level robustness. After motivating our venue in Section 13.1, we show in Section 13.2 how research on natural and artificial self-replicating systems inspired our approach. In Section 13.3, we relate our method to existing work before we provide an overview of this part in Section 13.4.

13.1 MOTIVATION

Today's computer programs are human-engineered artifacts. A program builds an abstract model of the real world and implements reactions to this modeled world (Simon, 1996). It is no surprise that programs often fail in real-world environments, because engineers either wrote erroneous code or simply because they could not foresee all situations the program encounters. For example, the program may be exposed to malicious attacks or may run in an unreliable execution environment.

self-adaption / self-optimization

The long-term goal of our research is to come up with networking software that *adapts itself* to unpredictable situations and *optimizes itself* in a new environment. These ultimate goals are currently out of reach. In this thesis, we restrict ourselves to the self-healing aspect of networking software, meaning that originally correct software shall be able to detect and repair itself when faults occur. We believe that such robustness is a key requirement for future self-adaptation and optimization properties.

probabilistic chips

We especially target faults on the code level that may result from an unreliable execution environment or from spontaneous alterations of memory bits, caused by cosmic radiation, for example. Such errors already affect computer systems today: Sun Microsystems acknowledged that cosmic rays interfered with cache memories and caused crashes in server systems at major customer sites, including America Online and eBay (Baumann, 2002; Reis, Chang, Vachharajani, Rangan, & August, 2005). This problem will become even more prominent in the future, because the influence of electrical noise and cosmic radiation increases with the higher package density of future chips. Systems will likely become more susceptible to spurious execution errors. This will not only lower the reliability of symbolic computation, it will also lead to disruptions of the program flow and to unrecoverable program exceptions. The promise, to reduce the production cost and energy consumption by *probabilistic chips* (Chakrapani, Korkmaz, Akgul, & Palem, 2007), leads to a further accentuation of the problem.

redundancy

central observer

The typical answer of computer scientists to cope with unreliable systems is to manually build up *redundancy* in order to mask such errors (Johnson, 1996; Pradhan, 1996; Wilfredo, 2000). In a redundant ensemble, multiple identical or similar systems are performing the same task. A *central observer* compares the states and/or the output of the redundant systems and votes for the most common result. When the observer detects a participant that deviates from the ensemble majority, the erroneous component is restarted or the system is reset to the last checkpoint. This architecture around a central observer has the inevitable flaw that the central decision maker may also be error prone. We would require an observer of the observer, and so on, which ultimately leads to infinite regression. Hence, we rather need a solution where the central observer that steers the redundancy is redundant too and is blended into the system.

Rather than asking for redundant hardware, we are aiming at software that it able to run on unreliable hardware and that organizes its own health intrinsically. Most imperative programming paradigms and languages that are available today hardly allow for monitoring the program's health and for recovering from a perturbation away from the expected behavior. Moreover, in order to avoid having a central observer, we require that the program monitors and repairs *itself*. Repairing implies that the program is able to modify its own code, for which a sound theoretical framework is still missing (Anckaert, Madou, & de Bosschere, 2007).

INSPIRATION FROM NATURE AND EARLY COMPUTER SCIENCE 13.2

Natural systems are able to dynamically construct redundancy by assembling and reproducing their components. Often, components exist in several copies (flocks, but also blood or nerve cells), exploiting parallelism and minimizing the impact of the loss of a single item. For singular components (e.g. bones) and in order to fight the problem of aging, redundancy is achieved over time through procreation, yielding a new and possibly modified copy. In computer science however, software is considered to be static and without wear.

This view is recent: Back in the 1940s, von Neumann (1966) developed a theory of self-reproducing automata. He described a *universal constructor*, a machine able to produce a copy of any other machine whose soft- and hardware blueprint is provided as input. Being universal, the constructor is also able to generate a copy of itself.

universal constructor

Considerable research on self-replication was carried out on the framework of *Cellular Automata* (CA), in which remarkable results were achieved, also in terms of robustness and self-repair (Tempesti, Mange, & Stauffer, 1998). However, these results are hard to transfer from CAs to the world of today's computer software. In the 1960s, with the desire to understand the fundamental information-processing principles and algorithms involved in self-replication, researchers started to focus on *self-replicating code*, i.e. how textual computer programs are able to replicate independent from their physical realization.

Cellular Automata

self-replicating code

The existence of self-replicating programs is a consequence of *Kleene's second recursion theorem* (1938), which states that for any program P there exists a program P', which generates its own encoding and passes it to P along

Kleene's second recursion theorem

with the original input. The simplest form of a self-replicating program is a *Quine*, named after the philosopher and logician Willard van Orman Quine, and made popular by Hofstadter (1979): A Quine is a program that prints its own code. Quines exist for any programming language that is Turing complete, and it is a common challenge for students to come up with a Quine in their language of choice. The *Quine Page* provides a comprehensive list of such programs for various programming languages (Thompson, 2010).

Quine

In this third part of the thesis, we put Quines in the parallel execution environment of an artificial chemistry, permitting an ensemble of Quine copies to achieve surprising robustness with respect to code and packet loss and even execution errors. Our contribution consists in the demonstration of an operational system based on Quines that runs highly reliable network services with provable dynamic properties. Our method is to use the chemical execution model introduced in Part II, in which we place carefully crafted self-replicating programs – Quines. Useful computation is then piggybacked to the Quine structures in order to implement network services.

13.3 RELATED WORK

In this section, we reference the relevant corner stones for our work where we could draw important insights, namely self-healing systems and software, self-reproduction, fault tolerance, artificial chemistries and their dynamics, the dynamics of competing populations and finally cooperation patterns.

13.3.1 SELF-HEALING COMPUTER SYSTEMS

Research on self-healing computer systems is a sub-branch of autonomic systems, which seeks for human-made artifacts that nevertheless adapt themselves autonomously and without human intervention to different environments. A big driver of this line of research was IBM's *Autonomic Computing Manifesto* (IBM, 2001; see also Kephart & Chess, 2003), in which an autonomic computer system is defined as a system which

autonomic computing manifesto

- knows itself and comprise components that also posses a system identity,
- knows its environment and the context surrounding its activity, and acts accordingly,
- configures and reconfigures itself,
- optimizes itself,

self-configuration
self-optimization

- performs something akin to healing,

self-healing

- protects itself, and

self-protection

- anticipates but hides its complexity.

These life-like so called "self-*" properties were challenges for many researches, but were anticipated as necessary requirements to master the complexity of future computer systems.

Self-healing is one of these requirements and asks a system "to be able to recover from routine and extraordinary events that might cause some of its parts to malfunction" (IBM, 2001) whereas Ghosh, Sharman, Rao, and Upadhyaya (2007) defined self-healing as

> [...] property that enables a system to perceive that it is not operating correctly and, without human intervention, make the necessary adjustments to restore itself to normalcy.

Keromytis (2007) stated that a self-healing architecture is

> [...] composed of two high-level elements: the software service whose integrity and availability we are interested in improving, and the elements of the system that perform the monitoring, diagnosis and healing.

These definitions require the system to be aware of what health is and to intentionally react to perturbations. Such a view subversively suggests an implementation around a central "mind".

Instead, we better like the idea that the system does not "know" its state explicitly, but that its state space is structured by large basins of attraction such that a perturbation always leads the system back to the fixed point of optimal operation. This is along the line of Shaw (2002) who propagated *homeostasis* for computing systems. The term homeostasis was introduced by Cannon (1929) and refers to "a mechanism through which the system acts to maintain a stable internal environment despite external variations" (Shaw, 2002). In their survey on self-healing systems, Ghosh et al. (2007) also distinguished between the two approaches of maintenance of health and active detection of failures and recovery.

homeostasis

Gangadhar (2005) proposed a self-healing framework within the context of Boolean networks: Similar to Dittrich, who identified chemical organization as high-level states of a chemical reaction systems (Dittrich & Speroni di Fenizio, 2007), Gangadhar (2005) identified the basins of attraction in meta dynamic states

Boolean networks as *Meta Dynamic States* (MDS). A healer component is notified if the system leaves the healthy basin of attraction and plans the necessary microstate changes in order to bring the system back to the healthy fixed point.

artificial immune system

Another high-level method to maintain the health of systems is an *Artificial Immune System* (AIS) (Timmis, Knight, De Castro, & Hart, 2004). AIS are able to distinguished between own and foreign states and objects based on the characteristics of the natural innate or adaptive immune system. Recently, Schreckling and Marktscheffel (2010) applied an AIS approach to Quines in Fraglets to protect chemical programs from various external attacks.

It is surprising how many of the published self-healing approaches still require a central decision maker. As a representative of the truly distributed approaches we mention the self-assembly of components in Amorphous Computing (Clement & Nagpal, 2003; Nagpal, Kondacs, & Chang, 2003).

13.3.2 SELF-HEALING SOFTWARE

The goal of self-healing software is to provide reliability techniques to protect program code against transient faults without the overhead of hardware techniques (Reis, Chang, & August, 2007). The promise of software-only methods is the ability to control and adapt the level of reliability required for a given application.

software redundancy

There are a number of proposed *software redundancy* techniques: Shirvani, Saxena, and McCluskey (2000) implemented an error correcting code to protect memory in software. Oh, Shirvani, and McCluskey (2002) proposed an error detection mechanism based on instructions that were duplicated during compile-time. These instruction replicas are executed by different registers and operate on different memory regions. Inserted checkpoints continuously validate the state of the computation. A similar technique to instrument the compiler was proposed by Reis et al. (2005).

probabilistic inference

Wong and Horowitz (2006) studied the soft error resilience of *probabilistic inference* algorithms. Programs based on probabilistic reasoning produce non-exact results but are still useful for many applications, for example in computer vision (Felzenszwalb & Huttenlocher, 2004), speech recognition (Huang et al., 1993), or robotics (Thrun, Fox, Burgard, & Dellaert, 2000). Wong and Horowitz (2006) showed that simple software modifications and checkpointing can reduce the number of program crashes drastically.

Based on our review of existing approaches, we postulate that self-healing software has to meet the following two requirements: First, software components have to be redundant: Only the duplication of information allows the program to re-activate damaged code. Second, this redundancy has to be managed dynamically: The software has to detect itself whether a Single Event Upset (SEU) occurred and has to access the redundant information in order to repair the damage. Thus, we define self-healing software as program code that is able to *organize its own redundancy*.

organize own redundancy

We propose to use the "analog" signals of molecular concentrations as feedback signals for the system's health. Analog signals are more resilient than digital codes. Stepney (2010) analyzed classical computation systems from a dynamical systems perspective. She argues that the attractor of a computer program is its halting state where the result is presented to the output. A small perturbation in the program's transition path usually leads to a different attractor, i.e. a different result of the computation. Dynamical systems on the other hand exhibit smoother and larger basins of attraction. Already earlier, similar observations led to the development of *analog computation models* (Rubel, 1993; Mills, 2008) or reaction-diffusion computers (Adamatzky, De Lacy Costello, & Asai, 2005), which exhibit better robustness than traditional computing paradigms.

analog computing

SELF-REPRODUCTION 13.3.3

In this thesis, we demonstrate how self-healing code naturally emerges in an artificial chemical setting by continuously replicating code, and by dynamically controlling this rewriting process.

Since the work of von Neumann (1966), many variants of *universal constructors* for self-reproduction have been proposed and elaborated. For an overview, see Freitas Jr. And Merkle (2004) or Sipper (1998). Langton (1984) argued that natural systems are lacking a universal constructor and relaxed the requirement that self-replicating structures must be equipped with a universal constructor. Instead, self-replication may arise from *dynamic loops* instead of static tapes; the information necessary to replicate the structure may be distributed in this loop and may not be present in explicit and distinct entities of a passive, un-interpreted blueprint and its active version of interpreted instructions. This observation lead to a new surge of research on such self-replicating structures (Perrier, Sipper, & Zahnd, 1996; Sipper, 1998).

universal constructors

dynamic loops

Some related work also focus on the structure of self-replicating sets of molecules: Bagley et al. (1989), Farmer, Kauffman, and Packard (1986b), Kauffman (1993), and later Mossel and Steel (2005) examined the properties of *random autocatalytic sets*, Fontana and Buss (1994) as well as Speroni di Fenizio and Banzhaf (2000) studied the formation of organizations in a random reaction vessel of λ-expressions and combinator algebra, respectively. Most of them focus on the structure of self-replicating sets. Here we additionally aim at performing microscopic computation (Dittrich, 2005), i.e.

random autocatalytic sets

computation that is carried out on the individual *instances* of molecules. This is a central requirement for being able to symbolically convey payload in packet networks.

There is little work on self-replicating programs in traditional languages: McKay and Essam (2001) investigated self-replicating programs in a functional programming language whereas Larkin and Stocks (2004) studied self-replicating λ-expressions. Self-replicating and self-modifying code also barely appears in today's software: *Self-modifying code* (SMC) was traditionally used as an optimization method (for example to implement state-dependent loops in assembly languages), for genetic programming (Koza, 1992), as camouflage for computer viruses, or as copy protection mechanism. Cai, Shao, and Vaynberg (2007) provide a framework to analyze self-modifying code in Hoare logic.

self-modifying code

13.3.4 CHEMICAL COMPUTING

In Chapter 3, we already introduced Artificial Chemistry as a branch of *Artificial Life* (ALife). Here, we complement this literature review by research related to self-replication:

Artificial Life

Artificial chemical computing models (Banâtre, Fradet, & Radenac, 2006; Calude & Paŭn, 2001; Dittrich, 2005; Holland, 1992; Paŭn, 2000) express computations as chemical reactions that consume and produce objects (data or code). In the same way as ALife seeks to understand life by building artificial systems with simplified life-like properties, Artificial Chemistry builds simplified abstract chemical models that nevertheless exhibit properties that may lead to emergent phenomena, such as the spontaneous organization of molecules into self-maintaining structures (Dittrich & Speroni di Fenizio, 2007; Fontana & Buss, 1994). The applications of artificial chemistries go beyond ALife, reaching biology, information processing (in the form of natural and artificial chemical computing models) and evolutionary algorithms for optimization, among other domains. Chemical models have also been used to express replication, reproduction and variation mechanisms (Dittrich & Banzhaf, 1998; Dittrich et al., 2001; Hutton, 2002; Teuscher, 2007; Yamamoto et al., 2007).

13.3.5 POPULATION DYNAMICS

The dynamics of natural chemical reactions is governed by the law of mass action (Abrash, 1986), which states that the reaction rate is proportional to the reactant concentration. In our work, we implement hard limits to an artificial chemical vessel's capacity. Environments with limited resources that host replicating entities lead to *natural selection*. This has been shown in research on population dynamics, for example by Szathmáry (1991), Stadler, Fontana, and Miller (1993), Fernando and Rowe (2007). In our case, the population consists of software components: Healthy software survives whereas errors are displaced. This naturally leads to *software homeostasis* – the intrinsic self-regulation of code in order to maintain a stable, healthy state.

natural selection

software homeostasis

Natural selection inevitably leads to a competitive environment where software instances fight for resources and where this struggle may lead to the extinction of healthy but inefficient or rarely used code. There are, however, well-known methods that show the emergence of *cooperation* (K. Wagner, 2000) in a competitive environment, such as the theory of *hypercycles* (Eigen & Schuster, 1979). Furthermore, in computer network research, mechanisms to control redundancy on the level of data-packets are well known. Transmission control protocols such as TCP (Postel, 1981) are not only able to recover from packet losses, but also to adapt the transmission rate to the limited bandwidth of the network (Jacobson, 1988), providing fairness for the competitive environment of the underlying IP network. Within our setting, we are able to transpose these methods, currently only used for data stream control, down to the code execution level, ensuring fairness among the software parts that fight for limited (memory) resources.

cooperation

Hypercycle

STRUCTURE OF THIS PART 13.4

We structure this part on self-healing protocols as follows: In this chapter, we reviewed related work and briefly introduced our concept of dynamic code replication (Quines), which yields self-healing code in limited memory. In Chapter 14, we show how our ideas are realizable in Fraglets and study the behavior of manually designed Quines for self-healing code in detail. Then, in Chapter 15, we discuss the phenomenon that multiple Quines residing in the same vessel compete against each other and propose methods to cooperatively link the Quines. As an application case, Chapter 16 demonstrates a multi-path routing protocol that makes use of continuous code replication in order to exhibit self-healing properties. In Chapter 17, we then take a closer look to the effects of code mutations in chemical software. Finally, in Chapter 18, we provide a first approach to cope with spontaneous alterations of memory by proposing a selection mechanism on the level of redundant reaction vessels.

CHAPTER 14

Self-Healing Software by Dynamic Code Replication

How self-replicating code – Quines – put into an environment with limited resources, exhibits self-healing properties.

Was war also das Leben?
Es war Wärme,
das Wärmeprodukt
formerhaltender Bestandlosigkeit,
ein Fieber der Materie,
von welchem der Prozess
unaufhörlicher Zersetzung
und Wiederherstellung
unhaltbar verwickelt,
unhaltbar kunstreich aufgebauter
Eiweissmolekel begleitet war. [14]

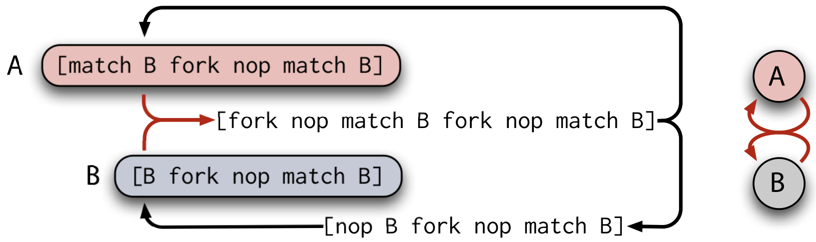
Der Zauberberg
Thomas Mann

What was life?
It was warmth,
the warmth generated
by a form-preserving instability,
a fever of matter,
which accompanied the process
of ceaseless decay
and repair
of protein molecules
that were too impossibly
ingenious in structure.

The Magic Mountain
Thomas Mann

In this chapter we first demonstrate how to write Quines in Fraglets, i.e. how sets of fraglet strings can preserve their own structure by continuously rewriting themselves (Section 14.1). Then, by limiting the memory resources of the reaction vessel and decaying excessive molecules, we impose an instability to the system, turning these Quines into software elements that steer their own redundancy and by this way become intrinsically self-healing. We also quantify the Quines' robustness by calculating their average survival time using phase-type distributions (Section 14.2) and show how to increase

Figure 14.1 Chemical Quine in Fraglets: A set of molecules (here: fraglet strings) regenerates itself. The blueprint molecule B reacts with its active variant A. The consecutive rewriting steps regenerate the two molecules.


A
[match B fork nop match B]
[fork nop match B fork nop match B]
B
[B fork nop match B]
[nop B fork nop match B]
A
B

(a) Rewriting loop in Fraglets

(b) Reactions

the robustness by distributing the Quines over the network (Section 14.3). Section 14.4 then introduces a modified Quine that, beyond replicating, performs some useful computation. This data-processing Quine can be used as a generic building block for self-healing programs, as we will demonstrate in the next chapter.

14.1 THE CHEMICAL QUINE

Quine

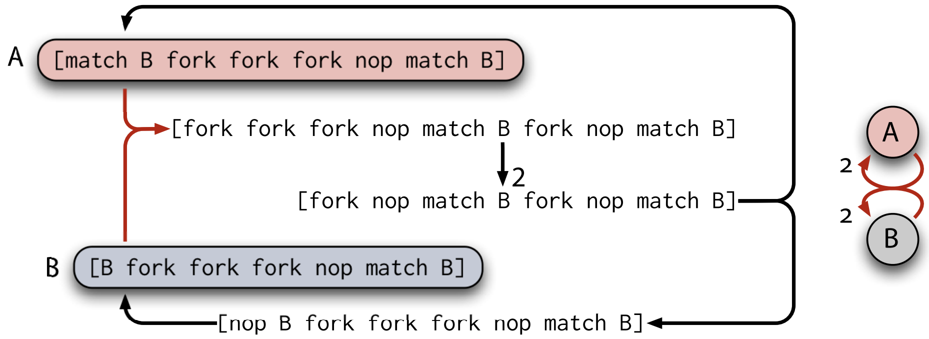
blueprint molecule

In ordinary sequential programming languages, a *Quine* is a single piece of code outputting its own source code. In the parallel world of an artificial chemistry like Fraglets, a Quine becomes a set of molecules that is able to regenerate itself. An example that illustrates this concept is the combination of a *blueprint molecule* B = `[B fork nop match B]` and its active variant A = `[match B fork nop match B]` (Yamamoto et al., 2007). The two molecules react with each other and, according to the Fraglets rewriting rules, regenerate themselves as shown in Figure 14.1. The schematic illustration in Figure 14.1(b) shows the corresponding chemical reaction network that is dynamically equivalent to the Fraglets rewriting loop in Figure 14.1(a). Note that only bimolecular reactions are scheduled according to the law of mass actions; unimolecular rewriting rules such as `fork` are immediately executed, as discussed in detail in Section 5.2.3. Hence, these intermediate steps (molecules) are omitted in the schematic notation.

14.1.1 REPLICATING QUINE AND LIMITED RESOURCES

replicating Quine

By repeating the `fork` instruction three times, the above Quine can be converted into a *replicating Quine* as shown in Figure 14.2. The replicating Quine generates two copies of itself in each round while consuming the original copy. Because the reactions are scheduled according to the law of mass action, the overall production rate increases with the growing number of Quine instances. Consequently, the population of Quines grows hyperbolically (Szathmáry, 1991), meaning that it theoretically reaches an infinite quantity in finite time.


A
[match B fork fork fork nop match B]
[fork fork fork nop match B fork nop match B]
2
[fork nop match B fork nop match B]
B
[B fork fork fork nop match B]
[nop B fork fork fork nop match B]
A
2
2
B

(a) Rewriting loop in Fraglets

(b) Reactions

Figure 14.2 Replicating Quine: The replicating Quine increases its population size by generating two replicas while the original copy is consumed.

As a limit to this unbounded growth, we introduce a *non-selective dilution flux* to the reaction vessel, which destroys arbitrary molecules as long as the total number of molecules exceeds a pre-defined vessel capacity. This leads to a selective pressure: Only molecules that are part of a self-replicating set have a chance to persist – all other molecules will eventually be displaced.

non-selective dilution flux

Dynamic Behavior and Deterministic Fixed Points 14.1.2

The dynamic behavior of the replicating Quine in a vessel of limited capacity is described by the *Catalytic Network Equation* (Stadler et al., 1993), a deterministic approximation expressed by Ordinary Differential Equations (ODEs) where x_A is the number of A-molecules and where x_B denotes the number of blueprints B

Catalytic Network Equation

$$\dot{x}_A = \overbrace{x_A x_B}^{\text{growth}} - \overbrace{\frac{x_A}{N} f(\mathbf{x})}^{\text{death}} \tag{14.1a}$$

$$\dot{x}_B = \underbrace{x_A x_B}_{\text{growth}} - \underbrace{\frac{x_B}{N} f(\mathbf{x})}_{\text{death}} \tag{14.1b}$$

subject to the conservation relation $x_A + x_B = N$, where N is the vessel capacity. The two molecules react according to the law of mass action with rate $r = x_A x_B$; each reaction event leads to an additional pair of molecules. The dilution flux applied to the vessel is equal to the net production rate $f(\mathbf{x}) = 2x_A x_B$, thus satisfying the conservation relation. The dilution flux is non-selective, i.e. each species is diluted with a rate proportional to its relative concentration (see also Section 11.3.1(c)).

Molecular quantities in a limited vessel are often expressed in (relative) concentrations. The *concentration* of molecule $s \in \mathcal{S}$, χ_s, denotes the fre-

concentration

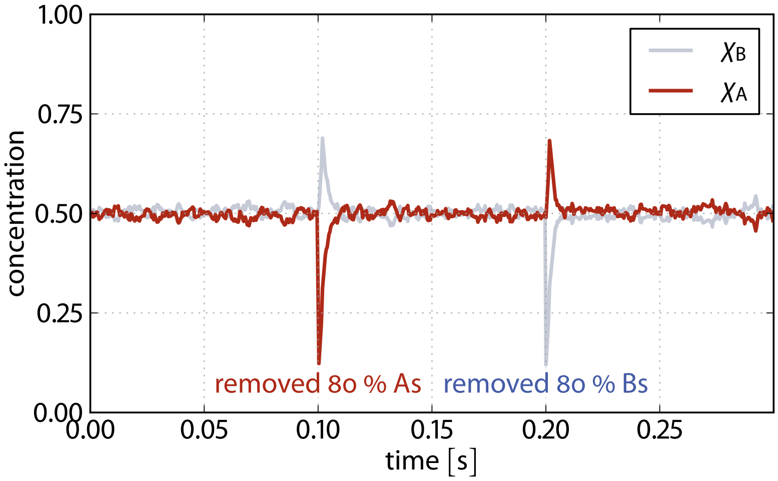


Figure 14.3 Replicating Quine subject to deletion attacks: Fraglets simulation of the replicating Quine in a vessel of capacity $N = 1000$ molecules. At time $t = 0.1$ s we removed 80 % of the active molecules A, whereas the same amount of B molecules is removed at time $t = 0.2$ s. Shortly after the attack, the remaining Quine instances refill the vessel.

quency or abundance of s in the vessel multiset, hence $\chi_s = x_s/N$. Expressed in concentrations, the above equations become

$$\dot{\chi}_A = N\chi_A\chi_B - \chi_A\Phi(\chi) \qquad (14.2a)$$

$$\dot{\chi}_B = N\chi_A\chi_B - \chi_B\Phi(\chi) \qquad (14.2b)$$

where $\Phi(\chi) = 2N\chi_A\chi_B$.

fixed points

The system exhibits three dynamic *fixed points*, one at $\hat{\chi}_A = \hat{\chi}_B = 1/2$ and two pathological cases at $\hat{\chi}_A = 1$, $\hat{\chi}_B = 0$, and $\hat{\chi}_A = 0$, $\hat{\chi}_B = 1$. The first fixed point is locally stable according to a standard perturbation analysis (see Section 8.2.2) and is characterized by both molecules – the blueprint and its active variant – being present with the same quantity.

Figure 14.3 demonstrates the robustness of the replicating Quine to perturbations: In a Fraglets vessel of capacity $N = 1000$ molecules, both molecule types are present with 500 instances each. At time $t = 0.1$ s we forcefully removed 80 % of the active molecules A from the vessel. The remaining Quines continue to produce replicas, which quickly repopulate the vessel. Once the vessel reaches saturation again, there are more blueprints than active molecules, which cause the dilution flux to remove the first more frequently than the latter until equilibrium is reached. At time $t = 0.2$ s we performed the same attack to the blueprints B, from which the system recovers likewise.

The stability property essentially means that the system returns to equilibrium condition: Even if we perturb the system by removing some instances of either species, the opponent forces of hyperbolic growth and non-selective dilution flux let the system autonomically find back to this fixed point. In other words, the system intrinsically maintains its own redundancy without an external controller!

14.2 ROBUSTNESS OF THE QUINE

In this section we quantify the robustness of the replicating Quine in the presence of faults: execution errors and random alteration of memory. But even when no faults occur, the lifetime of the Quine is finite due to stochastic fluctuations.

14.2.1 BASELINE ROBUSTNESS

The two pathological fixed points from the analysis above deserve some more attention. The deterministic ODE model predicts that they are not locally stable, which may lead to the conclusion that these states are not reachable or not persistent. However, in our stochastic execution environment, these fixed points will eventually be reached and represent states where one of the species is completely absent such that the system becomes deadlocked and finds itself in a so called *absorbing state* (see Section 8.1.3). As a consequence, the lifetime of a chemical Quine is limited, even in the absence of faults.

absorbing state

(a) *Quantification of the Robustness Using Phase-Type Distributions*

In order to quantify the baseline robustness of the replicating Quine, we now calculate the mean first-passage time to either absorbing state.

As we have seen in Section 8.1, a chemical reaction network governed by the law of mass action is stochastically modeled by a continuous time discrete space Markov jump process, whose dynamic behavior is described by the Chemical Master Equation (CME) (Gillespie, 1992). Our simple replicating Quine only consist of two species and the total number of molecules is fixed to N. Like for the example in Section 8.1.2, this allows us to model the system as a *finite birth-death Markov chain* where the state $N_A(t) \in [0, N]$ is a random variable denoting the number of A-molecules, whereas the number of blueprints B is given by $N_B(t) = N - N_A(t)$.

finite birth-death Markov chain

Figure 14.4 depicts the corresponding birth-death Markov chain, in which we labeled the states according to the number of A-molecules present. In saturation, a reaction yields two additional molecules and brings the system to a hypothetical state with $N+2$ molecules (red area in the figure's upper right corner), such that the dilution flux has to remove two arbitrary molecules. The effective transitions move the system along the saturation line; their rate is calculated as product of the reaction rate times the sum of all dilution path probabilities to one of the neighbor states. In this context, the birth rate λ_i represents the average rate of gaining an A-molecule and losing a B-molecule

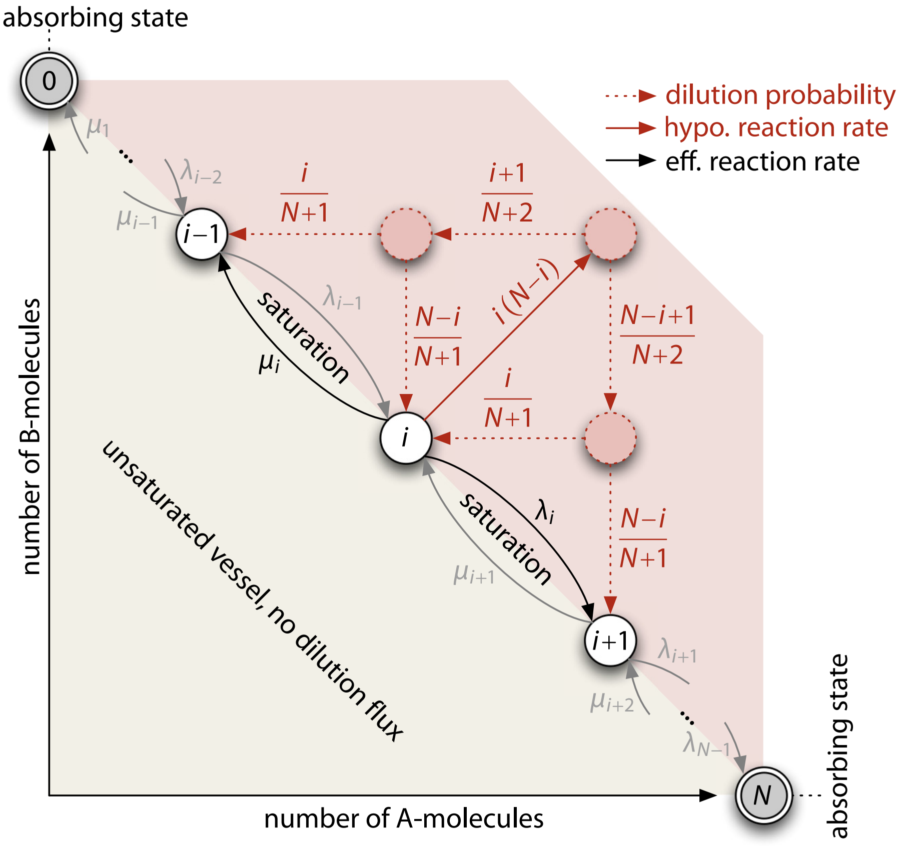


Figure 14.4 Markov chain of the replicating Quine: The replicating Quine in a saturated vessel is modeled as a birth-death Markov chain. The states are labeled according to the number of A-molecules present. Reactions from a saturated state leads to a hypothetical state (dashed circles). The dilution flux (dashed arrows) brings the system back to a valid state along the saturation line. The effective transition rates (black arrows) are calculated as product of the reaction rate times the sum of all dilution paths to the target state.

in state i whereas the death rate μ_i describes a movement in the opposite direction:

$$\lambda_i = \overbrace{i(N-i)}^{\text{reaction rate}} \overbrace{\frac{N-i+1}{N+2}\frac{N-i}{N+1}}^{\text{dilution prob.}} \tag{14.3a}$$

$$\mu_i = \underbrace{i(N-i)}_{\text{reaction rate}} \underbrace{\frac{i+1}{N+2}\frac{i}{N+1}}_{\text{dilution prob.}} \tag{14.3b}$$

We are interested in the survival time of the replicating Quine, i.e. the time until the Markov process hits one of the two absorption states ($\{0, N\}$).

phase-type distribution

In Section 8.1.4 we showed that the first passage time to absorption is described by a *phase-type distribution* according to Neuts (1981). We developed a tool that, given the reaction network and the vessel capacity N, automatically builds the corresponding Markov chain and its transition matrix. We then used Scilab (Campbell, Chancelier, & Nikoukhah, 2006) to calculate the mean time to absorption based on the generated transition matrix. This

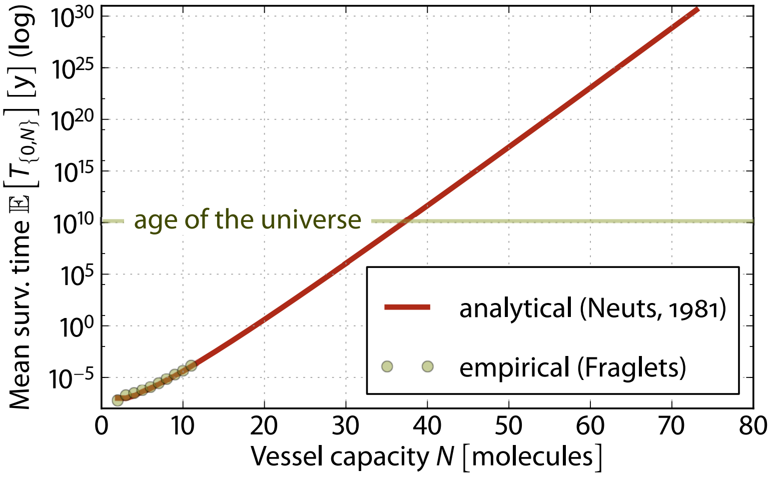


Figure 14.5 Baseline robustness of the replicating Quine: Based on calculation methods by Glaz (1979) and Neuts (1981), the mean survival time exponentially increases with the vessel capacity N and reaches the current age of the universe for about $N = 37$ molecules. These predictions match empirical measurements, i.e. the survival times of Fraglets simulations, averaged over 1000 runs.

method is feasible for vessel capacities $N \leq 30$. For larger vessels, the transition matrices become very large and Scilab produces erratic values due to the limited precision of the floating-point representation. Therefore, for vessel capacities $N > 30$ we used Glaz' method (1979) (see Section 8.1.4).

Figure 14.5 shows the *mean time to absorption* for the replicating Quine in vessels with different capacities. The figure illustrates that the mean time to absorption exponentially increases with the vessel capacity N. In fact, already for $N = 37$ molecules the mean survival time exceeds the current age of the universe. For small Ns we complemented these analytical calculations with 1000 simulation runs in Fraglets; the empirically measured average survival time accurately matches the predicted survival time.

mean time to absorption

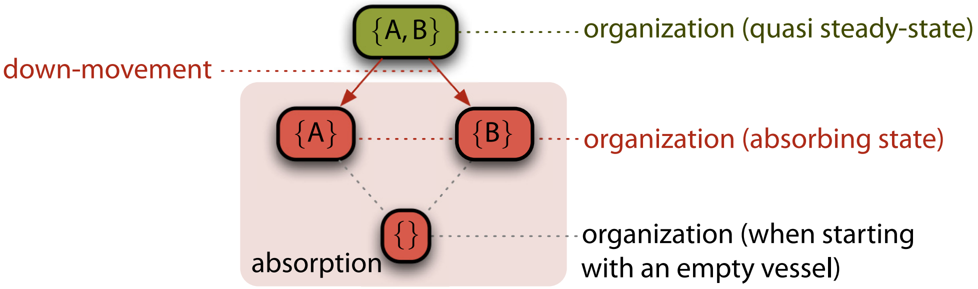
Identification of Absorption States by Using the Chemical Organization Theory (b)

For large reaction networks, it becomes unfeasible to analyze the corresponding Markov chain, and thus a detailed dynamic analysis is not possible anymore. As discussed in Section 8.4.2, the *Chemical Organization Theory* (COT) (Dittrich & Speroni di Fenizio, 2007) analyzes a reaction network structurally rather than analyzing the system's dynamics; it not even requires the reactions to follow the law of mass action. Instead of operating on multisets, the COT identifies those *sets* of species – *organizations* – that "are likely to be observed in a large reaction vessel on the long run" (Dittrich & Speroni di Fenizio, 2007). Such organizations are either desired quasi steady-states, unwanted absorption states, or stochastically closed subspaces. In the following, we demonstrate how to use the COT to at least identify organizations of large reaction systems, for which a detailed stochastic analysis would be too expensive or even impossible.

organizations

Figure 14.6

Organizations of the replicating Quine: The system may lose one of the species due to the stochastic dilution flux (down-movement). The system is dead for species sets in the red area (absorption).


{A, B}
organization (quasi steady-state)
down-movement
{A}
{B}
organization (absorbing state)
{}
absorption
organization (when starting with an empty vessel)

reactive flow system

The COT models the dilution flux by explicit first-order decay reactions. Such a *reactive flow system* (Dittrich & Speroni di Fenizio, 2007) for the replicating Quine is modeled by the following reactions:

$$A + B \longrightarrow 2A + 2B \tag{14.4a}$$

$$A \longrightarrow \varnothing \tag{14.4b}$$

$$B \longrightarrow \varnothing \tag{14.4c}$$

For this reaction system, the COT predicts two organizations: the set where both species are present hosting the quasi steady-state ($\{A, B\}$) and the pathological empty set $\{\}$ as consequence of the dilution reactions.

Note however that the COT does not predict the two absorption states we previously identified in our stochastic dynamical analysis: we know that two absorption states exist where either A- or B-molecules are absent. The reason for this discrepancy roots in the way we modeled the dilution flux: In a reaction vessel with limited capacity, dilution is always linked to an excessive production of molecules; the dilution rate is actually proportional to the net production rate of any molecule in the vessel. For the special case where no reaction occurs anymore, i.e. when the system becomes inert, the dilution rate drops to zero and the reactions (14.4b) and (14.4c) become void. This changes the type of the reaction system from a consistent reaction system to a non-consistent system, for which additional organizations may appear. Thus, in addition to finding the organization for the above reaction system, we have to determine the organizations of the reaction network without a dilution flux. Then, the union of both organization sets yields the organizations relevant for our execution model (P. Dittrich, personal communication, July 8, 2010).

The resulting lattice of organizations for the replicating Quine is depicted in Figure 14.6. Organization $\{A, B\}$ contains all species and represents the healthy quasi steady-state, in which the system stays for a long time.

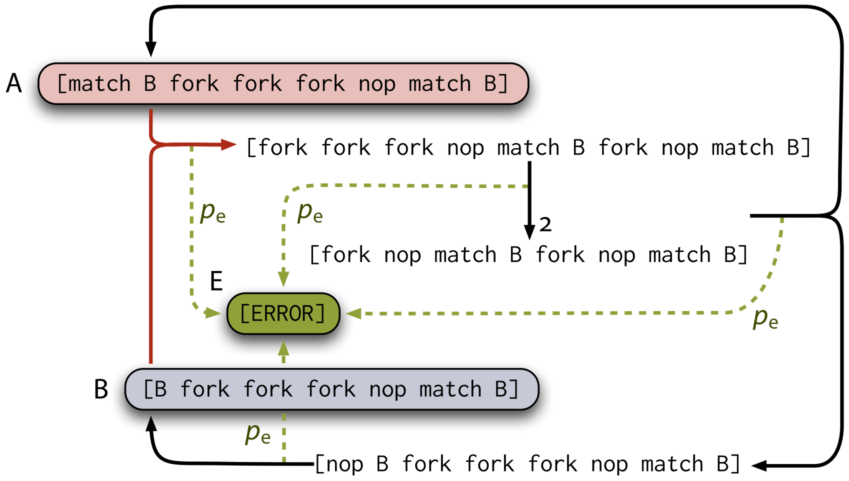

A
[match B fork fork fork nop match B]
[fork fork fork nop match B fork nop match B]
p_e
p_e
2
[fork nop match B fork nop match B]
E
[ERROR]
p_e
B
[B fork fork fork nop match B]
p_e
[nop B fork fork fork nop match B]

Figure 14.7 Replicating Quine subject to execution errors: Each rewriting step may be subject to an execution error (dashed lines) with probability p_e. We assume that the error product E is not able to react with the other molecules.

A down-movement in the lattice of organizations to one of the absorption organizations $\{A\}$ or $\{B\}$ may happen because the reaction vessel loses one of the species due to the random dilution flux. But from there, the system is never able to reach the pathological organization $\{\}$. This organization is only reachable if the vessel already starts empty. From a lower-level organization such as $\{A\}$ or $\{B\}$ the system is not able to intrinsically move up to a higher organization. Thus, all states within the two single-species organizations are stochastically closed and hence absorption states.

The COT provides us a qualitative picture of the possible fate of a complex reaction system. In this chapter, we will complement the robustness calculations of all reaction networks with the corresponding lattice of organizations.

ROBUSTNESS TO EXECUTION ERRORS 14.2.2

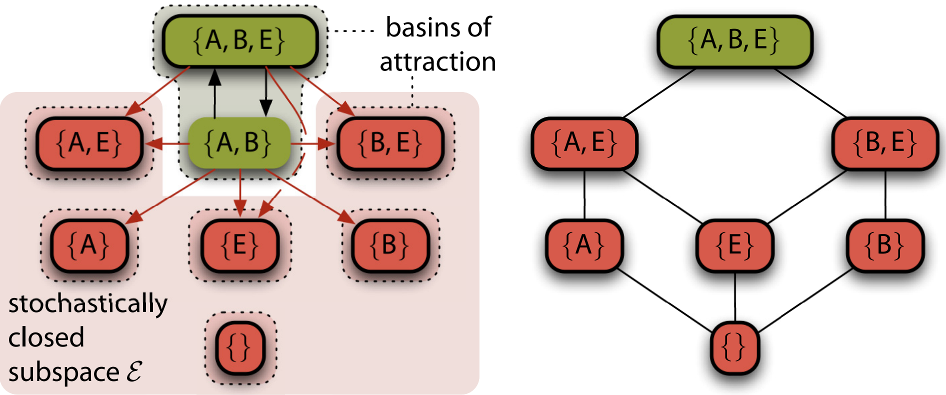
We expect the robustness of a Quine to decrease in the presence of execution errors, which we model as follows: With probability p_e, each Fraglets instruction fails to produce the right result; instead, a *neutral fraglet* is generated that is unable to replicate or to react with another molecule.

neutral fraglet

Figure 14.7 shows the rewriting steps of the replicating Quine that is subject to execution errors. The dashed arrows indicate that instead of the

Figure 14.8

Organizations of the replicating Quine (execution errors): The system may lose one of the species due to the stochastic dilution flux. The system is dead for species sets in the red area. Organizations have a black border; they form a lattice as shown in (b). Note that $\{A, B\}$ is not an organization because errors eventually appear.



(a) Molecule set transitions (b) Lattice of organizations

intended rewriting result, a error product E is generated. The corresponding reaction network is

$$A + B \xrightarrow{p_e(2-p_e)} E \tag{14.5a}$$

$$A + B \xrightarrow{(1-p_e)^2 p_e} 2E \tag{14.5b}$$

$$A + B \xrightarrow{(1-p_e)^3 p_e^2} 2E + 2B \tag{14.5c}$$

$$A + B \xrightarrow{(1-p_e)^3 p_e(2-p_e)} A + E + 2B \tag{14.5d}$$

$$A + B \xrightarrow{(1-p_e)^5} 2A + 2B \tag{14.5e}$$

Before calculating the survival time, we analyze the reaction network structurally. Figure 14.8(a) provides the set transition diagram, a simplified illustration of the underlying Markov process where we mapped all states containing the same species to the same *meta-state* ($\mathcal{M}(\mathcal{S}) \rightarrow \wp(\mathcal{S})$). An even simpler view is provided by the lattice of organizations in Figure 14.8(b), which depicts the possible (quasi) steady-states of the system.

meta-state

In order to determine the absorption state we recall that the system is alive if the vital reaction among A and B molecules is still possible, and hence those molecules must be present. Note that the set $\{A, B\}$ itself is not an organization. These states belong to the attractor or *generator of the organization* $\{A, B, E\}$, because errors will eventually be produced while the two molecules react. Thus the organization $\{A, B, E\}$ and its generator sets host the healthy quasi steady-state of the system. Figure 14.8(a) helps us

generator of the organization

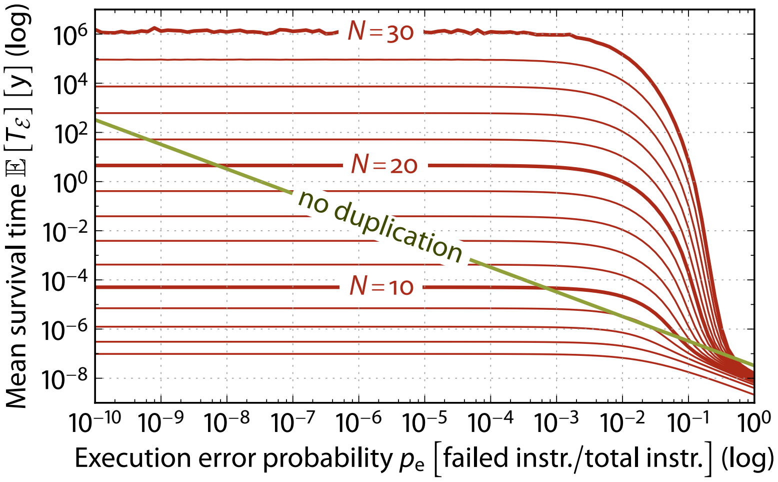


Figure 14.9 Robustness of the replicating Quine (execution errors): Mean survival time in years, in a vessel of capacity N, plotted with respect to the execution errors probability p_e. The fluctuations for $N = 30$ molecules are due to floating point precision limits in Scilab. The diagonal line illustrates the survival time of a single non-replicating and thus not self-healing Quine instance, representing a traditional piece of software, which realistically fails at the first occurrence of an execution error.

in identifying the *stochastically closed subspace* $\mathcal{E}$, from which no transition leads back to the healthy organization. Absorption states are instances of any organization in this subspace.

stochastically closed subspace

Now that we identified the subspace $\mathcal{E}$, in which the system became inert, we calculate the *mean first-passage time* to this subspace from the healthy quasi steady-state. Glaz' method does not work, because the system cannot be modeled by a simple birth-death chain anymore, but Neuts' generic method based on phase-type distribution can still be applied. The survival time, i.e. the first-passage time to an absorption state, can be calculated from the transition matrix of the corresponding two-dimensional Markov state array.

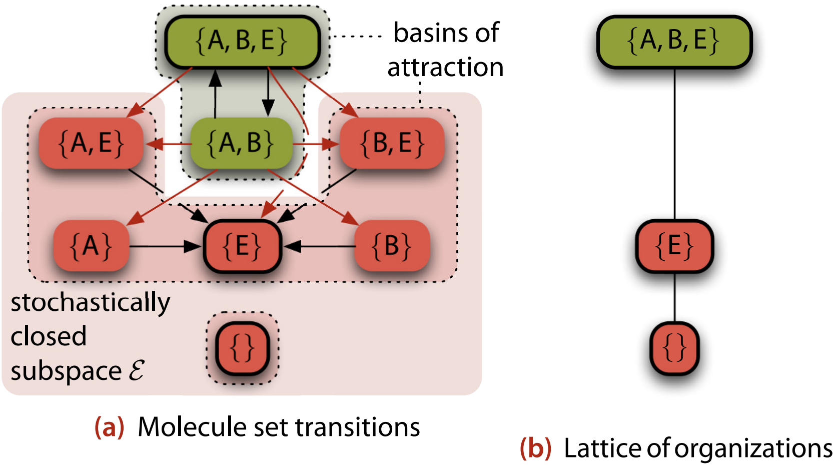
mean first-passage time

Figure 14.9 shows the mean survival time of a replicating Quine in vessels of different capacities plotted with respect to different execution error probabilities. The robustness of the Quine barely decreases for realistic error probabilities and only sharply drops for probabilities above 1 %. The diagonal line in Figure 14.9 illustrates the survival rate of a single Quine instance that just rewrites itself but does not replicate, and thus, is not able to heal itself. This situation is comparable to traditional sequential software, which realistically fails at the first occurrence of an execution error. Although the self-healing Quine eventually dies even in the absence of execution errors, it lives much longer than un-instrumented code in the presence of execution errors, even in small reaction vessels.

ROBUSTNESS TO MEMORY ALTERATIONS 14.2.3

The second type of error we discuss in this section are spontaneous alterations of memory bits with rate δ in $/(\text{bit s})$. Such *Single Event Upsets* (SEUs) can be caused, for example, by cosmic radiation (Normand, 1996). The rate at which a fraglet is hit depends on its length l in symbols and the bit-width of

Single Event Upset (SEU)



(a) Molecule set transitions

(b) Lattice of organizations

Figure 14.10 Organizations of the replicating Quine (memory alterations): The system may lose one of the species due to the stochastic dilution flux. The system is dead for species sets in the stochastically closed subspace, indicated by the red background in (a). Organizations have a black border; they form a lattice as shown in (b).

the symbol encoding scheme, b; thus a fraglet is hit $lb\delta$ times per second. We model the spontaneous mutations that turn a fraglet into a neutral molecule E with the reaction network

$$\mathrm{A} + \mathrm{B} \xrightarrow{1} 2\mathrm{A} + 2\mathrm{B} \tag{14.6a}$$

$$\mathrm{A} \xrightarrow{8b\delta} \mathrm{E} \tag{14.6b}$$

$$\mathrm{B} \xrightarrow{7b\delta} \mathrm{E} \tag{14.6c}$$

As for execution errors, the lattice of organizations for memory alterations depicted in Figure 14.10 indicates that the quasi steady-state includes mutation products. The stochastic dilution flux may bring the system to absorption, i.e. to the species set $\{\mathrm{A}, \mathrm{E}\}$ or $\{\mathrm{B}, \mathrm{E}\}$. Note that the memory alteration "reactions" (14.6b) and (14.6c) are still active in those states. They eventually but very slowly turn all A- and B-molecules to error products E, which is the reason why the species set $\{\mathrm{E}\}$ is the only non-pathological absorption organization.

mean survival time

Figure 14.11 shows the *mean survival time* of our replicating Quine, now plotted with respect to different symbol mutation rates (we assumed that each fraglets symbol is encoded by eight bits). To appraise these curves, we recall that, according to Normand (1996), the Single Event Upset (SEU) rate caused in microelectronic devices by radiation is around 5×10^{-16} /(bit s). In this region, the Quine is almost unaffected by mutations.

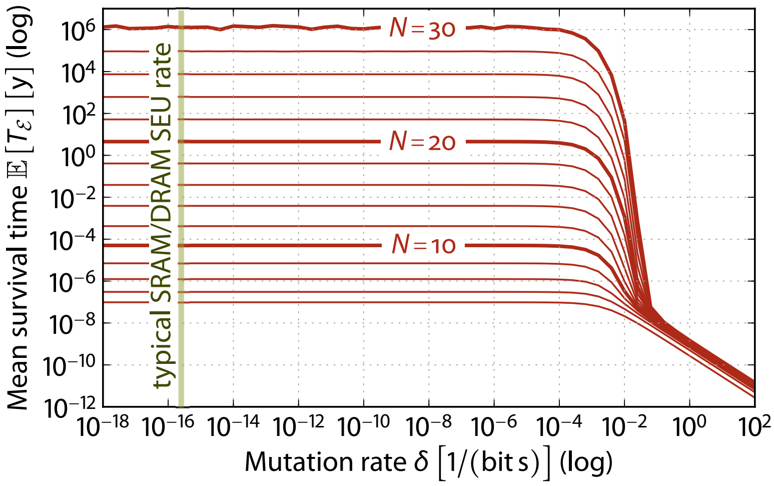


Figure 14.11 Robustness of the replicating Quine (memory alterations): Mean survival time in years, in a vessel of capacity *N*, plotted with respect to the bit mutation rate δ. A Fraglet symbol is encoded by 8 bits. The fluctuations for $N = 30$ molecules are due to floating point precision limits in Scilab. The Single Event Upset (SEU) rate caused in microelectronic devices by radiation is around 5×10^{-15} /(bit s). In this region, the Quine is almost unaffected by mutations.

DISTRIBUTED QUINES 14.3

In this section, we study whether the survival time of a Quine is higher in a distributed context. Intuitively, we expect that the system is able to survive for a longer time when backup molecules exist in neighbor vessels: When one vessel hits an absorption state, a neighbor vessel could be still alive and would be able to heal the inert node by sending it the seed to restart the Quine's loop. However, we found out that the strategy of spreading seeds is important for this scheme to work.

We analyzed the robustness of two different types of distributed Quines. Common to both types is that each node v_i contains a reaction between the Quine's active molecule A_i and the corresponding blueprint B_i. The difference is that in the first case we produce the seed replicas locally and send them to a random neighbor node (anycast), as is shown in Figure 14.12. In Fraglets, this is realized by encoding the two species as

type 1

$$\mathrm{A}_i = \texttt{[match B anycast fork fork fork nop match B]} \qquad (14.7a)$$

$$\mathrm{B}_i = \texttt{[B anycast fork fork fork nop match B]} \qquad (14.7b)$$

Once arrived, the seeds unfold and generate two instances of either species. The second distributed Quine type is depicted in Figure 14.13 and the Fraglets code is shown below.

type 2

$$\mathrm{A}_i = \texttt{[match B fork nop anycast fork nop match B]} \qquad (14.8a)$$

$$\mathrm{B}_i = \texttt{[B fork nop anycast fork nop match B]} \qquad (14.8b)$$

Their product only sends one seed to a random neighbor node whereas the other seed is used to regenerate the Quine locally.

Figure 14.12 Distributed Quine type 1: The product of a reaction among A and B is sent to an arbitrary neighbor node where two replicas of those molecules are produced. We consider memory bit mutations at rate δ and packet loss during transmission with probability p_l.

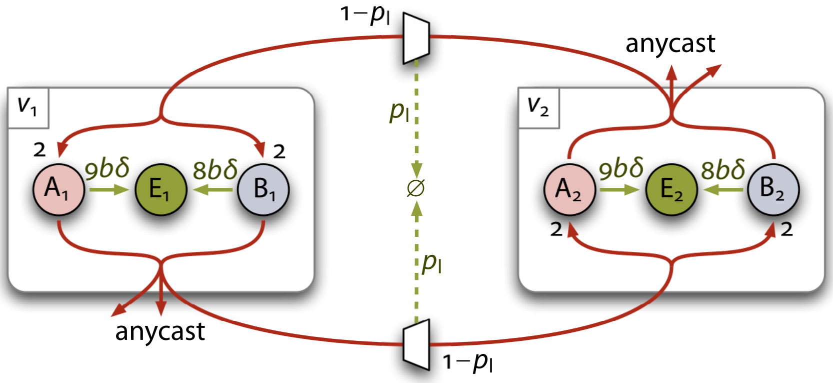


Figure 14.13 Distributed Quine type 2: The product of a reaction among A and B is sent to an arbitrary neighbor node *and* is delivered locally. In total, one additional instance of both molecules is produced. We consider memory bit mutations at rate δ and packet loss during transmission with probability p_l.

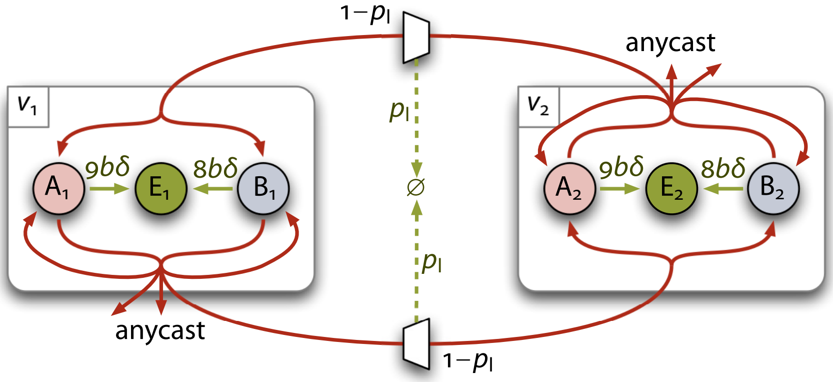


Figure 14.14 Robustness of distributed Quines (memory alterations): Mean survival time in years, in two vessels with a total capacity of $N = 14$ molecules in comparison to a single replicating Quine with the same total capacity of $N=14$ molecules, plotted with respect to the bit mutation rate δ. A Fraglet symbol is encoded by 8 bits. An absorption state is reached when both vessels either lack A_i or B_i molecules.

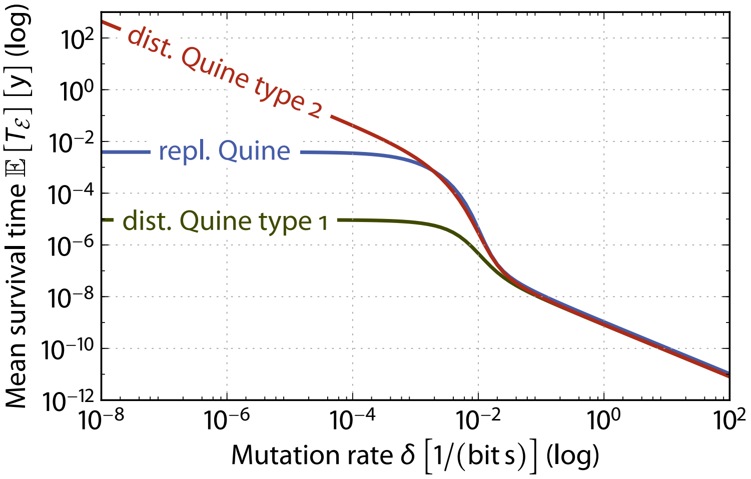


Figure 14.14 shows the mean survival time against different mutation rates of the two distributed Quine types in a two-node network topology, as well as the curve of a one-node replicating Quine for comparison purpose. The total capacity of the reaction vessel(s) is the same for all three cases, namely $N = 14$ molecules: That is, for the one-node Quine, the single vessel provides the whole capacity, whereas for the two distributed cases, the capacity is evenly distributed between the two participating nodes.

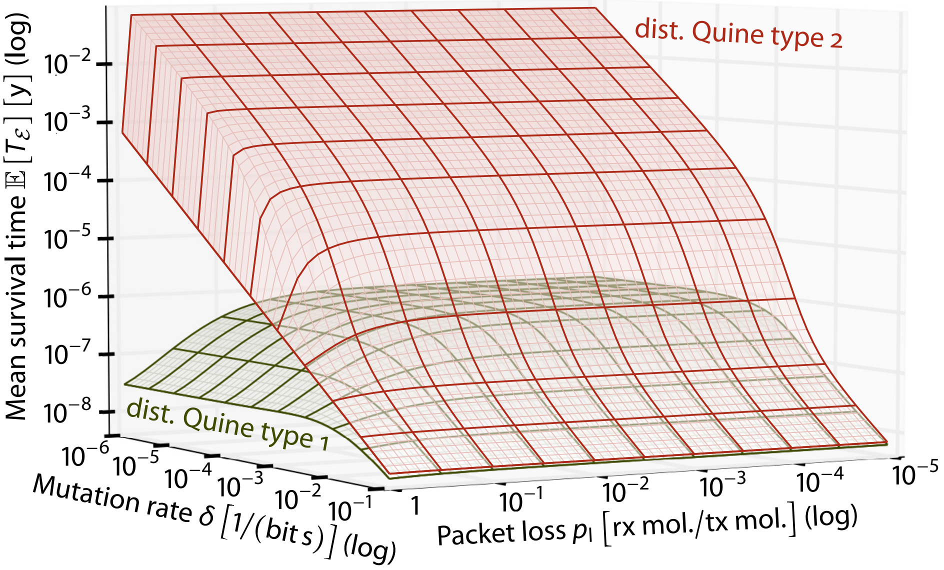
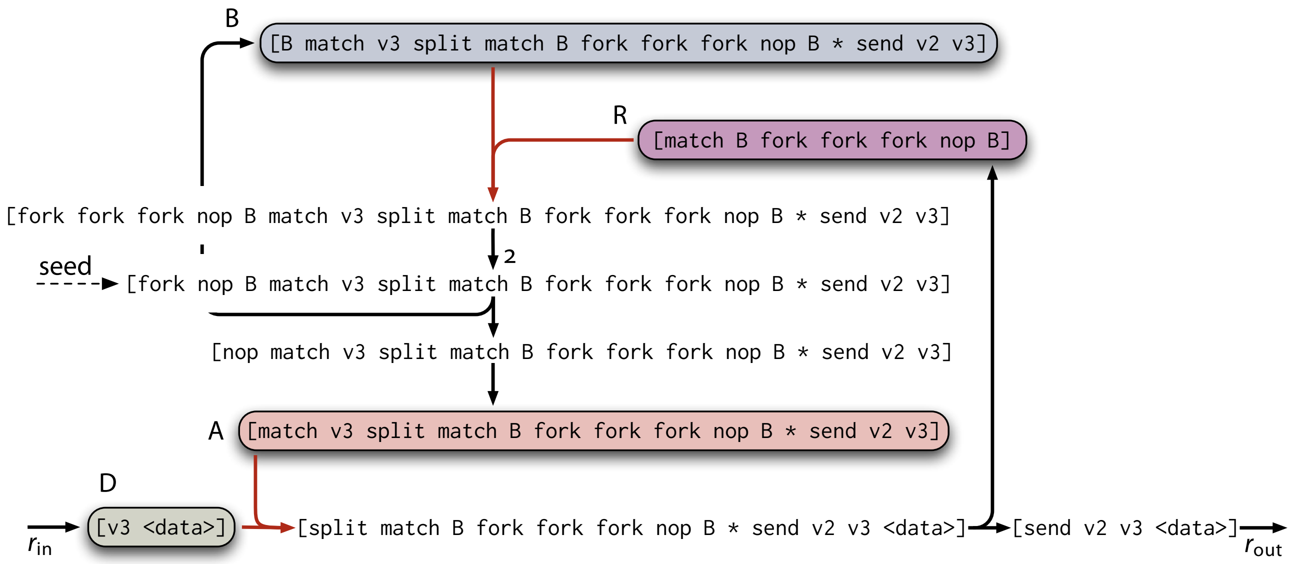


Figure 14.15 Robustness of distributed Quines (packet loss): Mean survival time in years, in two vessels with a total capacity of $N = 10$ molecules, plotted with respect to the bit mutation rate δ and the inter-vessel packet loss probability p_l. A Fraglet symbol is encoded by 8 bits. The absorption state is an instance of the species set $\mathcal{E}$ where both vessels lack A_i or B_i.

We observe that the single Quine always outperforms the distributed Quine of type 1, which sends both seed copies to the neighbor. The distributed Quine of type 2 however is the most robust variant for moderate bit alteration rates. This stems from the fact that the second type has to digest only two received molecules at once, whereas the first type receives four instances at a time. The more excessive molecules the dilution flux has to remove, the higher is the fluctuation around the quasi steady-state and, consequently, the more likely one of the absorption states is reached. Thus, the good recipe for robust distributed Quines is to replicate locally in order to maintain the population and to send only one copy to a neighbor node for the case a neighbor deviates from the fixed point or even reached a local absorption state.

Figure 14.15 shows the robustness surface of the two distributed Quine types for different symbol alteration rates and packet loss probabilities. Packet loss only has a marginal effect on the robustness, especially for the second type, which replicates locally. The second type only needs the received Quine seeds to bootstrap the local Quine when accidentally hitting a local absorption state.

Our calculations show that a carefully designed distributed Quine exhibits a better robustness than an isolated Quine even when comparing them with the same *total* vessel capacity. In a typical networking scenario, the total memory resource available grows with the number of nodes in the network. In this case, where additional memory comes for free, the distributed Quine is always by far more robust than a network of isolated Quines. Its persistence even makes it hard to replace the Quine intentionally with a new version of itself, as we will see in the application case of the next chapter.


B
[B match v3 split match B fork fork fork nop B * send v2 v3]
R
[match B fork fork fork nop B]
[fork fork fork nop B match v3 split match B fork fork fork nop B * send v2 v3]
2
seed
[fork nop B match v3 split match B fork fork fork nop B * send v2 v3]
[nop match v3 split match B fork fork fork nop B * send v2 v3]
A
[match v3 split match B fork fork fork nop B * send v2 v3]
D
r_{in}
[v3 <data>]
[split match B fork fork fork nop B * send v2 v3 <data>]
[send v2 v3 <data>]
r_{out}

(a) Rewriting loop in Fraglets

Figure 14.16 Data-processing Quine: The active molecule A processes a data molecule D resulting in a reward R. The blueprint B contains all information necessary to generate two copies of A and B when reacting with R.

14.4 A GENERIC BUILDING BLOCK FOR SELF-HEALING SOFTWARE

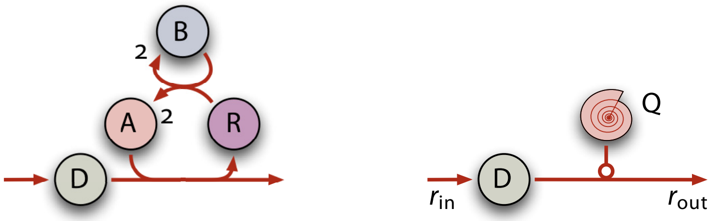
data-processing Quine

design pattern

The Quine studied so far just spends CPU cycles replicating itself. In this section, we demonstrate how our self-healing Quines can be enriched to perform an actual and useful computation. The resulting *data-processing Quine* can be used as a software building block for various tasks, serving as a *design pattern* to engraft the self-healing property to an arbitrary piece of code.

14.4.1 THE DATA-PROCESSING QUINE

A small modification to the replicating Quine's structure leads to a data-processing Quine, as is depicted in Figure 14.16. With the new structure, the set of A- and B-molecules does not directly replicate anymore. Instead, the active molecule A reacts with a data molecule (or "data packet") D, computes some product and also generates an additional reward molecule R. The reward molecule reacts with and consumes a blueprint molecule B, which contains the necessary information to re-create the active molecule and its blueprint. As usual with replicating Quines, the reaction between R and B results in two instances of A and B.



(b) Reaction network

(c) Short notation

Figure 14.16 cont.: We make use of two equivalent short-notations for the data-processing Quines.

A GENERIC TEMPLATE TO QUINIFY CODE 14.4.2

Every Fraglets rule can be converted to a data-processing Quine, which is intrinsically robust to the discussed faults. In the following, we introduce a simple recipe to "quinify" code.

quinification

The data-processing Quine depicted in Figure 14.16 is the self-healing implementation of a packet-forwarding reaction that we introduced earlier in Figure 7.4 on page 95:

$$\begin{aligned} & {}_{v_1}\overbrace{[\texttt{matchp v3 send v2 v3}]}^{f} + {}_{v_1}[\texttt{v3 }\Omega] \\ \longrightarrow\ & {}_{v_1}[\texttt{matchp v3 send v2 v3}] + {}_{v_2}[\texttt{v3 }\Omega] \end{aligned} \tag{14.9}$$

The matchp-fraglet residing in node v_1 consumes local data packets destined to node v_3 and sends them to the next hop v_2. In fact, any persistent Fraglets rule f like the one in (14.9) can be converted to a data-processing Quine by using the template

$$[\Psi_{\text{spawn}}\ \ \Psi_{\text{consume}}(f)\ \ \Psi_{\text{replicate}}\ \ \Psi_{\text{produce}}(f)\,] \tag{14.10a}$$

where generally the symbol strings Ψ_{spawn} and $\Psi_{\text{replicate}}$ are defined as

$$\Psi_{\text{spawn}} := \texttt{fork nop B} \tag{14.10b}$$

$$\Psi_{\text{replicate}} := \texttt{split match B fork fork } \Psi_{\text{spawn}}\ \texttt{*} \tag{14.10c}$$

To generate the Quine replacement for fraglet $f = [\texttt{matchp v3 send v2 v3}]$, we define the consumption and production part of of the data-processing Quine as

$$\Psi_{\text{consume}}(f) := \texttt{match v3} \tag{14.10d}$$

$$\Psi_{\text{produce}}(f) := \texttt{send v2 v3} \tag{14.10e}$$

which results in the seed

```
[fork nop match v3 split match B
                              fork fork fork nop B * send v2 v3]  (14.11)
```

that bootstraps the data-processing Quine (compare to Figure 14.16). Complex software, consisting of many matchp-rules, can be quinified by following this recipe. There are, however, some circumstances in which those Quines compete against each other for the limited memory rather than cooperate to accomplish a common task. We will dedicate the next chapter to the study of this competitive behavior.

14.4.3 INTRINSIC PACKET LOSS

replication rate

Unlike the original Quine, the data-processing Quine only replicates upon processing a data packet. The *replication rate* is given by the production rate of R, which is approximately equal to the packet injection rate r_{in}. Thus, in order to exhibit self-healing properties, the data-processing Quine has to be continuously fed with input molecules.

Ideally, the rate at which the Quine produces results, r_{out}, is equal to the data molecule injection rate r_{in}. In this case the system does not intrinsically lose packets. However, we have to expect some of the packets D being diluted. In fact, the more buffered packets are present in the vessel, the higher is the probability for intrinsic packet loss due to the non-selective dilution flux. In the following, we determine the packet loss probability p_{l}, given as the fraction of injected packets that does not arrive at the output:

$$p_{\text{l}} = 1 - \frac{r_{\text{out}}}{r_{\text{in}}} \tag{14.12}$$

deterministic fixed point

In order to quantify this packet loss, we present a closed-form expression for the *deterministic fixed point* of the data-processing Quine. Therefore, we write down the corresponding ODE system in terms of relative concentrations

$$\dot{\chi}_{\text{D}} = \frac{r_{\text{in}}}{N} - N\chi_{\text{A}}\chi_{\text{D}} - \chi_{\text{D}}\Phi(\chi) \tag{14.13a}$$

$$\dot{\chi}_{\text{A}} = -N\chi_{\text{A}}\chi_{\text{D}} + 2N\chi_{\text{B}}\chi_{\text{R}} - \chi_{\text{A}}\Phi(\chi) \tag{14.13b}$$

$$\dot{\chi}_{\text{R}} = N\chi_{\text{A}}\chi_{\text{D}} - N\chi_{\text{B}}\chi_{\text{R}} - \chi_{\text{R}}\Phi(\chi) \tag{14.13c}$$

$$\dot{\chi}_{\text{B}} = N\chi_{\text{B}}\chi_{\text{R}} - \chi_{\text{B}}\Phi(\chi) \tag{14.13d}$$

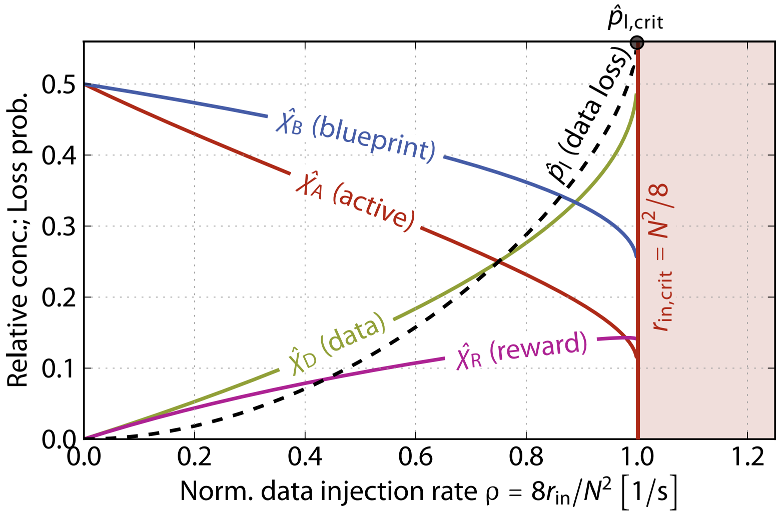


Figure 14.17 Deterministic fixed point of the data processing Quine: The concentrations are plotted with respect to ρ, the data injection rate normalized to the vessel capacity. For increasing data injection rates, attended by a growing steady-state concentration of the data molecule $\hat{\chi}_D$, the packet loss probability $\hat{p}_l$ increases due to the dilution flux.

with the total dilution flux being

$$\Phi(\chi) = \frac{r_{\mathrm{in}}}{N} - N\chi_A\chi_D + 2N\chi_B\chi_R \tag{14.13e}$$

subject to the constraint that all concentrations denote proper molecule frequencies, $\chi_D, \chi_A, \chi_R, \chi_B \in [0,1]$, and that the total number of molecules is conserved, $\chi_D + \chi_A + \chi_R + \chi_B = 1$. This ODE system depends on two parameters, the vessel capacity N and the packet injection rate r_{in}. In order to eliminate one parameter, we normalize the packet injection rate: For the *normalized injection rate* $\rho(N) = 8r_{\mathrm{in}}/N^2$ the above equation system yields the following parametric fixed point:

normalized injection rate

$$\hat{\chi}_D(\rho) = \frac{1}{2}\left(1 - \sqrt{1-\rho}\right) \tag{14.14a}$$

$$\hat{\chi}_A(\rho) = \frac{1}{4}\left(3 - \hat{\chi}_D - \sqrt{1 + 2\rho + 2\hat{\chi}_D + \hat{\chi}_D^2}\right) \tag{14.14b}$$

$$\hat{\chi}_B(\rho) = \frac{1}{2}\left(1 - \hat{\chi}_D\right) \tag{14.14c}$$

$$\hat{\chi}_R(\rho) = \frac{1}{2}\left(1 - 2\hat{\chi}_A - \hat{\chi}_D\right) \tag{14.14d}$$

Figure 14.17 shows the steady-state concentrations of all species with respect to the normalized packet injection rate ρ. If packets are injected at a slow pace, they are immediately processed by the Quine's active molecules while the vessel's capacity is equally shared among the active molecules A and their blueprints B. Consequently, the vast majority of injected packets are converted to products. For an increasing injection rate, data molecules start to accumulate, building-up a non-zero steady-state concentration $\hat{\chi}_D$. Since the vessel's capacity is shared among all molecules, this partly squeezes out active molecules and blueprints. On the other hand, the replicating Quine also displaces packets. In order to calculate the *probability of packet loss* we recall that the output rate follows the law of mass action (LOMA): $r_\text{out} = x_\text{A} x_\text{D}$. Hence, the steady-state data loss probability $\hat{p}_\text{l}$, shown as dashed line in Figure 14.17, is given as

probability of packet loss

$$\hat{p}_\text{l}(\rho) = 1 - \frac{r_\text{out}}{r_\text{in}} = 1 - \frac{\overbrace{\hat{x}_\text{A}\hat{x}_\text{D}}^{\text{LOMA}}}{r_\text{in}} = 1 - 8\frac{\hat{\chi}_\text{A}\hat{\chi}_\text{D}}{\rho} \tag{14.15}$$

The steady-state packet loss rises for increasing data injection rates and reaches $\hat{p}_{l,\text{crit}} \approx 0.56$ for $\rho = 1$, i.e. for a critical packet injection rate of $r_\text{in,crit} = N^2/8$.

14.4.4 BIFURCATION AT THE CRITICAL DATA INJECTION RATE

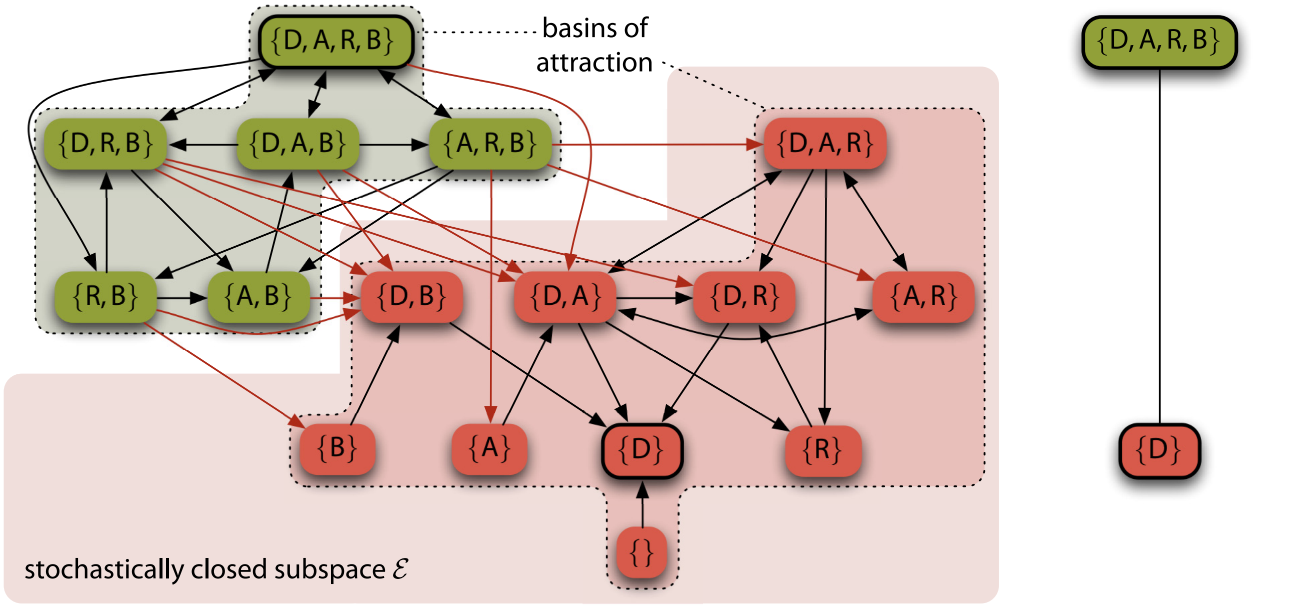
If the data-processing Quine has to digest data molecules that are injected faster than with the critical rate of $r_\text{in,crit} = N^2/8$, there is no stable fixed point anymore where all species are present. For higher rates, the active molecules and blueprints are completely squeezed out by the packet stream. As a result, the Quine is not able to replicate anymore and the system ends in the pathological fixed point $\hat{\chi}_\text{D} = 1$, $\hat{\chi}_\text{A} = \hat{\chi}_\text{R} = \hat{\chi}_\text{B} = 0$.

We expect (and will show in the next section) that the robustness of the data-processing Quine drops for an increasing injection rate. This essentially implies that the data-processing Quine must be fed with a low-rate packet stream ($r_\text{in} \ll N^2/8$) or, said differently, the vessel capacity must be dimensioned such that the critical rate never occurs, on which we will focus later in Section 14.4.6. Since the critical rate is proportional to N^2, the system scales well, and it is feasible to find a reasonable capacity for arbitrary packet rates.

14.4.5 BASELINE ROBUSTNESS

The deterministic fixed point calculated above coincides with the quasi steady-state of the stochastic model. As we showed for the replicating Quine, the stochastic process of the data-processing Quine also contains a stochastically closed subspace, in which the system eventually ends due to the stochastic dilution flux. In the following, we compare the baseline robustness of the data-processing Quine to that of the replicating Quine.

Figure 14.18(a) depicts the set transition diagram, i.e. the simplified illustration of the underlying Markov process. An even simpler view is provided by the Chemical Organization Theory (COT), which attributes the system two



(a) Molecule set transitions (b) Lattice of org.

Figure 14.18 Organizations of the data-processing Quine: The system may lose one of the species due to the stochastic dilution flux. The system is inert for species sets in the stochastically closed subspace, indicated by the red background in (a). Organizations have a black border; they form a lattice as shown in (b). Note that {} is not an organization as long as data molecules are continuously injected.

organizations, shown in Figure 14.18(b): one is the quasi steady-state where all species are present ($\{D, A, R, B\}$) whereas the other is the single absorption steady-state in which the whole vessel only contains data packets ($\{D\}$). Note that the empty set is no organization as long as packets are continuously injected.

stochastically closed subset

The states in the red area of Figure 14.18(a) span a *stochastically closed subset* $\mathcal{E} \subset \wp(\mathcal{S})$, from which the quasi steady-state is unreachable (see Section 8.1.3). Instead, the system eventually ends in the absorption organization $\{D\}$. In $\mathcal{E}$ the system became inert, meaning that the Quine is not able to replicate anymore and the remaining molecules are eventually displaced by the continuously arriving packets.

baseline robustness

Once more, we use Neuts' method (1981) to calculate the *baseline robustness*, i.e. the survival time, of the data-processing Quine. More explicitly, we calculate the mean first-passage time to the stochastically closed set $\mathcal{E}$ when starting in the quasi steady-state.

Figure 14.19 depicts the mean survival time of the data-processing Quine for different vessel capacities N with respect to the normalized data injection rate ρ. The lifetime of the Quine without injected packets and without execution errors or memory alterations is infinite, but its expected lifetime continuously decreases with an increasing packet injection rate. As suspected

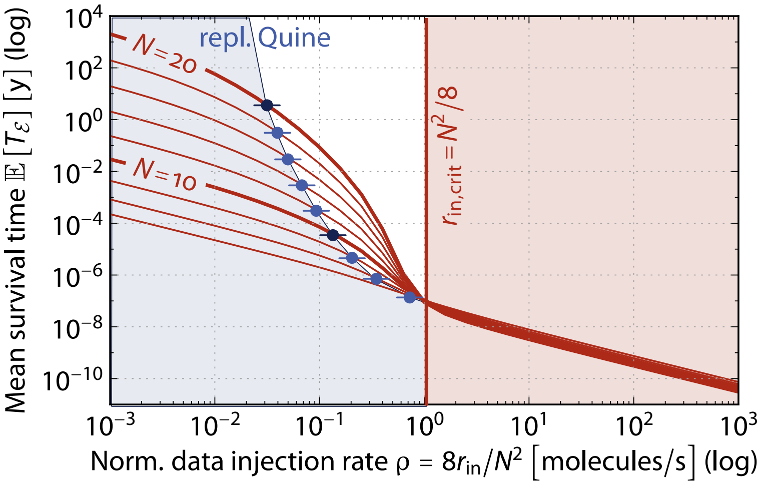


Figure 14.19 Robustness of the data-processing Quine: Mean survival time in years, in a vessel of capacity N, plotted with respect to the normalized data injection rate $\rho = 8r_{\text{in}}/N^2$. The blue dots indicate the corresponding mean survival time of the idle-looping replicating Quine for comparison. The lifetime drastically drops around and above the critical injection rate (red area). The blue area denotes the packet injection rate for which the lifetime of the data-processing Quine is longer than the lifetime of the replicating Quine in a vessel with the same capacity.

before based on the deterministic ODE description, the critical injection rate $r_{\text{in}} = N^2/8$, above which there is no deterministic fixed point, sharply limits the survival time for any vessel capacity. That is, for a practical usage the system has to artificially limit the injection rate or the vessel capacity has to be dimensioned for the fastest injection rate possible.

14.4.6 RESOURCE REQUIREMENTS

Compared to a persistent reaction rule, a quinified operation needs more memory and processor time. In terms of CPU time, each data processing Quine not only processes the data molecule, but performs seven additional reaction or transformation steps.

A single instance of a data-processing Quine also needs more memory than the comparable persistent rule of length l: The active molecule as well as the blueprint coexists in the reaction vessel, leading to a memory requirement of $2l+19$ symbols. However, because we need redundancy in case of a fault, not a single Quine instance, but a population of Quine replicas must be present. The total memory needed depends on the level of robustness the application requires. In a short thought experiment, we demonstrate how to dimension the vessel capacity for the worst-case scenario. That is, we show how to choose the vessel capacity for the maximal expected packet injection rate, such that the packet loss is small and the lifetime of the data-processing Quine is long enough.

Consider a packet-forwarding engine in which a data-processing Quine's job is to send incoming packets to a dedicated next hop, such as for example the Quine in Figure 14.16. We aim at dimensioning the vessel capacity N for the case where packets arrive at wire-speed from a Gigabit Ethernet link.

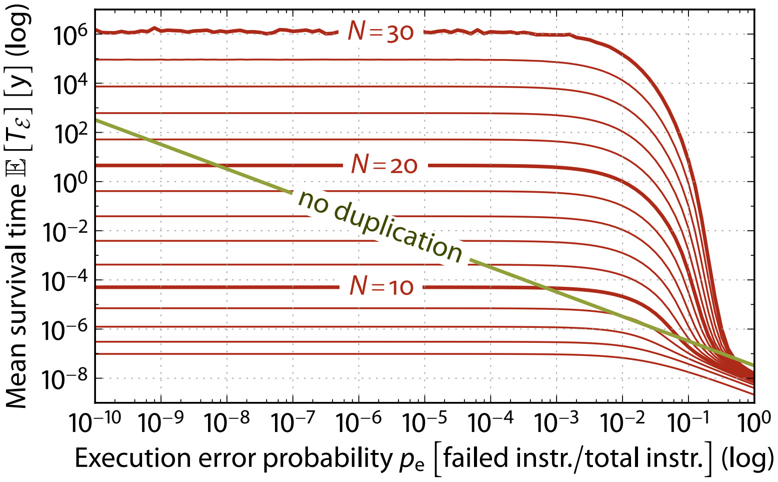


Figure 14.20 Robustness of the data-processing Quine: Mean survival time for different normalized data injection rates $\rho = 8r_\mathrm{in}/N^2$, plotted with respect to the vessel capacity N in comparison to the mean survival time of the replicating Quine.

A Gigabit Ethernet is able to transfer 81 275 packets/s if all of them have the maximal size of 1500 bytes (not considering jumbo frames). The vessel capacity for which this is the critical rate is $N = \sqrt{8r_{\mathrm{in,crit}}} = \sqrt{8 \cdot 81275} \approx 800$ molecules. We definitely need more capacity because, according to Figure 14.19, the survival time of the Quine for the critical rate is only a few seconds (10^{-7} years).

Let us consider a ten-fold capacity of $N = 8000$ molecules, resulting in a normalized injection rate of $\rho \approx 0.01$ at wire-speed. We cannot calculate the baseline robustness of such large vessels with reasonable effort anymore as the state space roughly contains 3^{8000} states. Instead, we approximate the survival time: Figure 14.20 depicts the survival time of the data-processing Quine for different packet injection rates. Already for small Ns we observe that the survival time increases exponentially with the vessel capacity. A rough linear regression on the lin-log plot (dashed lines) indicates that the *survival time* for the normalized injection rate $\rho = 0.01$ reaches the current age of the universe already for $N \approx 40$ molecules. For the same injection rate the packet loss probability is about 0.5 % according to the parametric fixed-point calculation in the previous section. This loss probability is tolerable for most applications.

survival time

Such a worst-case calculation is helpful to optimize the system with respect to maximum lifetime, minimum packet loss, and minimum capacity needed. Usually we can carry out this optimization only on the packet loss vs. capacity axis and ignore the system's lifetime, because the expected lifetime increases in order of $\exp(N)$ and reaches a sufficiently high value soon, whereas the packet loss only decreases in the order of N^2.

14.5 SUMMARY

Unlike traditional self-healing engineering approaches, where a central observer monitors and repairs the system, we propose a method based on an algorithmic chemistry that is intrinsically robust to code perturbations. Our method is based on a subset of the necessary ingredients for evolution: self-replication and a limited lifetime for the individuals due to limited resources. These ingredients yield self-healing code – code that autonomically organizes its own redundancy.

For an implicit artificial chemistry like Fraglets, the challenge is to find molecular structures that lead to self-replicating loops — Quines. We will dedicate Chapter 17 on the search for such structures. As we demonstrated, once we have found a set of self-replicating molecules, we are able to embed symbolic computation to the loop, letting the Quine perform useful computation such as symbolic packet processing in networking protocols.

So far, we assumed that all execution errors and memory bit alterations finally lead to neutral and harmless Fraglet strings. In reality, the error products interact with the program. We will delve more into this topic in Chapter 18.

The introduction of the random dilution flux makes some of the design patterns introduced in Chapter 10 impossible: In a vessel with limited capacity, some molecules of a population always fall victim to the dilution flux. A non-zero concentration is not sustainable when the population is not refreshed by an external or intrinsic process. In such an open system, conservation rules are void and protocols basing on them, such as *Disperser*, are not functional anymore. New rules and patterns to design networking protocols are required.

One such pattern is the data-processing Quine, which allows an arbitrary symbolic operation to be rendered self-healing. A protocol designer may use this motif as a generic building block to construct larger chemical programs. Unfortunately, this turns out to be non-trivial, since uncoupled Quines fight against each other for existence. In the next chapter, we will delve into this phenomenon and demonstrate how Quines can be forced to cooperate with each other.

CHAPTER 15

Competition and Cooperation of Replicating Code

Competing Quines struggle for existence, but they can be symbiotically linked such that they cooperatively achieve a common task.

We will now discuss
in a little more detail
the Struggle for Existence. [15]

The Origin of Species
Charles Darwin

By applying an excessive dilution flux, we created a competitive environment in which software components have to rewrite themselves continuously. A single Quine does so and this enables it to survive. But protocol software rarely consists of a single Quine. In this chapter, we study whether multiple Quines are able to coexist in the same vessel under the selective pressure imposed by the dilution flux. As expected, they naturally compete for the limited resource, leading to the extinction of all but one. Only a well-designed symbiotic strategy yields to the survival of all Quines that belong to a cooperative set.

Section 15.1 demonstrates that only one out of many uncoupled Quines survives in a single, well-stirred vessel, but also shows that this trivial outcome does not hold for the distributed case. Then, in Section 15.2, two topics come together: We show how to couple Quines such that they cooperate, and, at the same time, we demonstrate that the previously introduced data-processing Quine serves as a generic building-block to construct larger programs. Finally,

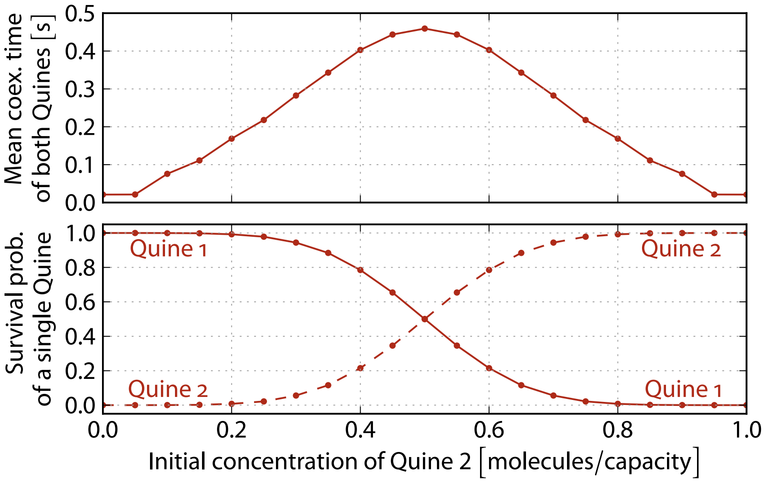


Figure 15.1 Coexistence time and survival probability of competing Quines: The two replicating Quines do not coexist for a long time; the one with a higher initial concentration most likely prevails. Top: Mean coexistence time in a vessel of capacity $N = 20$ molecules, plotted with respect to the relative initial concentration of the second Quine. Bottom: Survival probability of either Quine in the same vessel.

in Section 15.3, we present a first self-healing protocol made of Quines that allows for a controlled distributed update of self-healing software.

15.1 COMPETITION OF QUINES IN LOCAL AND DISTRIBUTED CONTEXTS

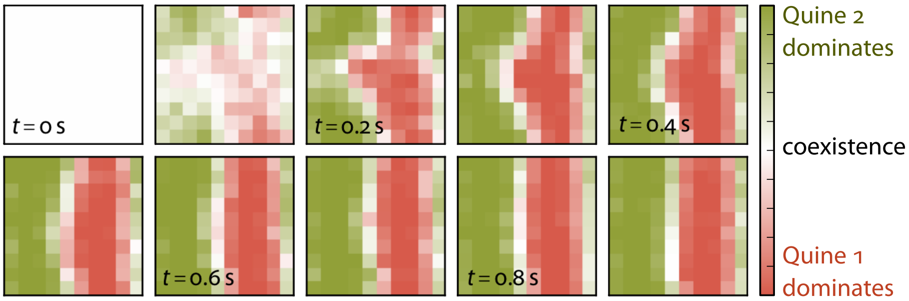
As we have shown in the previous chapter, the aggressive growth of the Quine is instrumental for letting it self-heal. However, in a vessel with limited resources this leads to a very competitive environment where two different Quine types cannot coexist. In a distributed setting, however, Quines may find their topological niche in which they are able to survive.

15.1.1 COMPETITION IN A WELL-STIRRED VESSEL — THE WINNER TAKES ALL

survival of the common

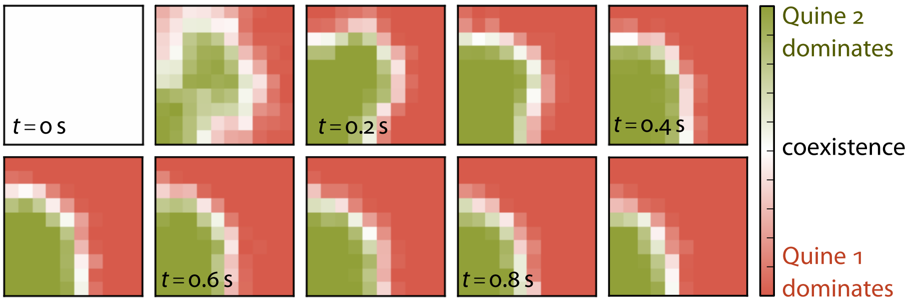
When placing multiple replicating Quine species into a common reaction vessel, only one will survive while the others will be literally squeezed out. This is due to the finding that independently and hyperbolically growing populations with finite resources lead to the *survival of the common* (Szathmáry, 1991): The first Quine that reaches a sufficiently high concentration will dominate the others and lead to their extinction even if their replication rate is higher.

Figure 15.1 illustrates the struggle between two replicating Quines in the same vessel for different initial concentrations but for a constant total vessel capacity of $N = 20$ molecules. We observe that the robustness of the Quines heavily depend on their initial concentration – the (initial) allocation of memory slots for Quine 2 grows linearly from left to right – and that the non-selective dilution flux does not guarantee fairness among the Quines *per*



(a) Toroidal topology (horizontal and vertical wrap-around)

Figure 15.2 **The emergence of islands of dominance**: Series of snapshots over time in a toroidal topology of 100 vessels. Each vessel has a capacity of 100 molecules. In each vessel, two competing distributed Quines of type 2 start with an equal concentration. In some simulation trials, we observed the emergence of horizontal or vertical stripes.



(b) Grid topology (neither horizontal nor vertical wrap-around)

Figure 15.2 **cont.:** In a grid topology (no horizontal or vertical wrap-around) different patterns emerge. We regularly observed isolated island of either Quine in one of the four corners of the grid.

se, as one of the Quines is squeezed out by the other within a fraction of a second.

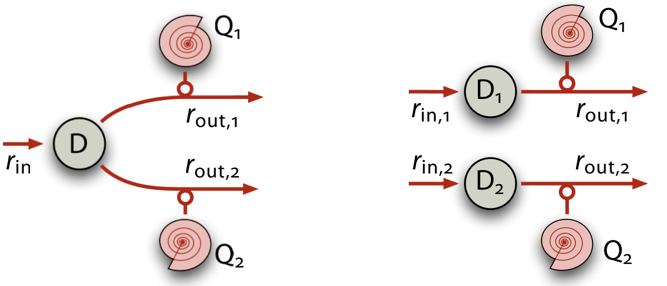
COMPETITION IN THE NETWORK — SEPARATED ISLANDS 15.1.2

In a distributed setting, the outcome is different: We studied the distributed Quine type 2 (see Figure 14.13 on page 272) in different network topologies and found out that the network fosters phase islands, in which different Quines may *coexist* in spatially distant locations.

coexistence in spatially distant locations

Figure 15.2 shows the results of a typical Fraglets simulation run in a network of 100 vessels arranged in a two-dimensional grid. Each vessel has a capacity of $N = 100$ molecules and is initially populated with an equal quantity of the two Quines. The two figures show a series of spatial snapshots of the network over time. Each pixel represents the vessel at the corresponding coordinates: red and green colors indicate that either Quine 1 or Quine 2 dominates whereas a coexistence of both Quines is represented by white pixels.

For the first example, we used a toroidal network topology, meaning that vessels at the border of the grid are connected to the vessels at the opposite border. Figure 15.2(a) depicts a pattern that we frequently observed in this



(a) Same data stream

(b) Different streams

Figure 15.3 Competing data-processing Quines: There are two situations in which data-processing Quines compete against each other: (a) If several Quines process the same packet, a random one survives. (b) The concentration of Quines processing different packets is proportional to the corresponding packet injection rate; a Quine may die out if its relative packet injection rate is too small.

stripes of identical Quines

topology: stationary horizontal or vertical *stripes* of either Quine type. A vessel inside such a stripe is completely surrounded by vessels hosting the same Quine type. At the border of the stripe, the two Quines coexist in the same vessel, because the inflow of either type from opposite directions is approximately equal.

network corners of identical Quines

If we slightly reconfigure the network topology by disabling the wrap-around connections at the border we observe additional patterns, such as the one depicted in Figure 15.2(b): One Quine retracts and is able resist the dominance of the other Quine in one of the *corners* of the network.

fixation probability

Note that in these experiments, we initialized all nodes with the same amount of either Quine, and therefore we gave both Quines the same chance to prevail. Once the network only hosts one Quine, the probability that the network is invaded by another similar Quine is very low. In biology, this is called the *fixation probability*, which denotes the probability that a population of wild-type instances is replaced by a single instance of a mutant (Wright, 1931; Moran, 1958; Nowak, 2006). Hence, once installed, it is very hard to get rid of the Quine or to replace it with a newer version. In Section 15.3, we demonstrate how two versions cooperate in order to allow a distributed software update.

evolutionary graph theory

It is already well known that bi-stable systems become stable in a spatial environment (Kapral & Oppo, 1986; Károlyi, Péntek, Scheuring, Tel, & Toroczkai, 2000) or are able to form more complex organizations (Dittrich & Banzhaf, 1997). Additionally, there are results from *evolutionary graph theory* (Lieberman, Hauert, & Nowak, 2005) indicating that the topology of the network graph has a strong influence on the selective pressure. A more detailed analysis of the spatial behavior of Quines is left for future work.

Competing Data-Processing Quines 15.1.3

We return to the analysis of competition in a well-stirred vessel. The struggle for existence between different data-processing Quines is behaviorally similar to the replicating Quines although they need data packets to replicate.

For moderate packet injection rates we distinguish two different scenarios depending on whether or not two Quines process the same packet (see Figure 15.3): (a) When consuming the same molecular species (for example by matching the same header tag in Fraglets), the two Quines actually fight for the same "food" molecule D, which is required for the Quines to replicate. Due to the law of mass action the Quine that exhibits a larger multiplicity will react more often with the food, yielding even more offspring. Consequently, the qualitative outcome is the same as for the simple replicating Quine: survival of the common. (b) Even if the two Quines are "digesting" distinct food molecules, D_1 and D_2, respectively, they still live in the same habitat and fight for the same memory resource. The Quines only coexist if their food molecules are injected with a rate in the same order of magnitude, which causes their replication rate to be similar. In fact, the steady-state concentration of the Quines is proportional to the corresponding food injection rates. Otherwise, if the injection rates diverge too much, the concentration of the slower Quine will very likely drop to zero and the Quine becomes extinct.

processing the same data stream

processing different data streams

This competing behavior is problematic, because we would like to compose multiple self-healing Quines, each working on some part of a problem to solve. As we will show in the next section, this is in fact possible if the Quines cooperate.

Cooperative Linkage of Quines 15.2

In this section, we introduce four design patterns to couple data-processing Quines. Cooperating *Quines must mutually control their replication rate* in order to achieve fairness with respect to their expected survival time. Ideally, we want the survival time to be independent of the packet-injection rate. In the following, we analyze various coupling methods for data-processing Quines.

Quines must mutually control their replication rate

String of Quines 15.2.1

Frequently, a computation requires multiple data-processing *operations in sequence*. Figure 15.4 shows such a scenario for *n* operations, each using our data-processing Quine structure. In such a reaction network topology, the common packet stream generates a symbiotic relationship among the Quines:

sequential operations

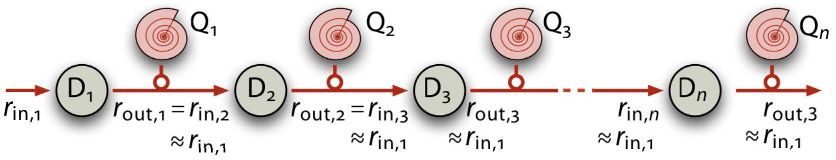


Figure 15.4 String of Quines: Multiple data-processing Quines are indirectly linked by sequentially processing the same data stream.

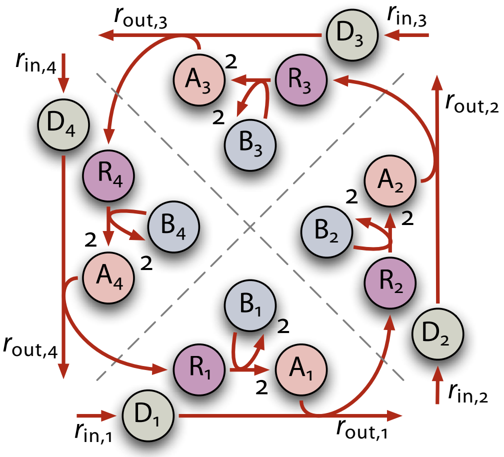


Figure 15.5 Hypercycle Quine: Multiple data-processing Quines are symbiotically linked by mutually letting them generate their reward to replicate in a cyclic fashion.

The second Quine is only able to replicate if the first Quine generated a product molecule and replicated in turn. The third Quine only replicates after the second replicated before. Thus, if the packet injection rate is much lower than the critical margin of $N^2/8$, the growth rate of all Quines is approximately equal to the data injection rate.

15.2.2 HYPERCYCLE

If Quines do not process the same data stream sequentially, we need another method of symbiotically coupling the Quines. Eigen and Schuster (1979) proposed a *cyclic linkage* of reactions as an explanation of self-organization of prebiotic systems in which RNA strands and enzymes cooperate. We translate this idea to the world of Quines in Fraglets: As shown in Figure 15.5, a cycle of Quines consists of several data-processing Quines, which do not generate their own replication reward R_i. Instead, each Quine cyclically generates the reward for another Quine.

cyclic linkage

The advantage of such a cyclic dependence is that none of the Quines is able to replicate much faster than the others, which leads to their sustained coexistence in the reaction vessel. The disadvantage comes from the fact that an active molecule A_i is consumed when a data packet is processed and is not immediately available for the next data packet. Thus, on the long run, Quine i cannot process a data stream faster than twice the data processing rate of the predecessor Quine. That is, the hypercyclic Quines impose strong

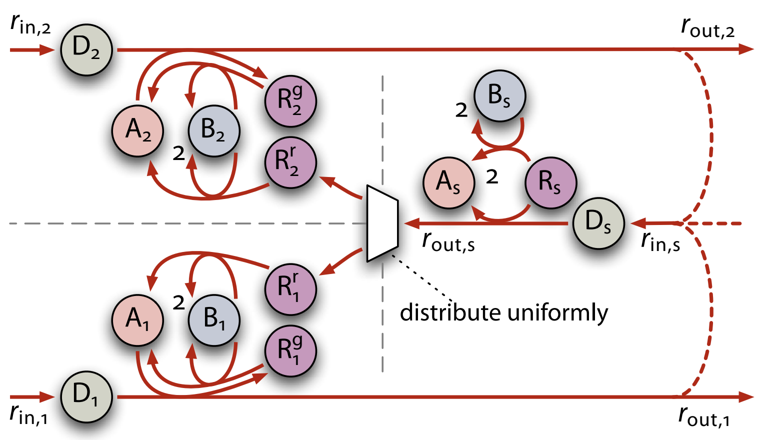


Figure 15.6 Separation of regeneration and replication: The two data-processing Quines only regenerate themselves; a replication feed is provided by a separate supporting Quine on the right.

restrictions on the range of data streams they are able to process as the most active element is doomed to starve.

DATA-RATE INDEPENDENT REPLICATION FEED 15.2.3

The main disadvantage of the hypercylic Quine, starvation of the most active molecules, arises from the fact that these molecules are produced by one specific other Quine in the cycle. This dependency can be broken by separating regeneration from replication: When processing a data packet, the Quine shall immediately regenerate the consumed active molecule. This maintains a constant number of active molecules, provided that there is no dilution flux and no mutation or execution error. To cope with the relatively rare error events we have to provide a separate stream of replication rewards for the fraglet; these replication rewards increase the number of active molecules and blueprints. Because we want all coupled Quines to survive, we have to guarantee that they all receive their replication rewards with the same rate, i.e. they are fairly fed.

replication feed pattern

Figure 15.6 shows such a *replication feed pattern*. Two Quines process independent data streams and regenerate themselves individually by producing a local regeneration reward molecule R_i^g for each data molecule processed. A separate support Quine generates replication rewards R_i^r and distributes them uniformly to the involved Quines. The support Quine is self-replicating, paced by an externally provided stream of trigger molecules. Ideally, its "feeding rate" would be adjusted to the expected error rate. If the error rate cannot be estimated, the data processing Quines could be reprogrammed in order to automatically produce this trigger, although the latter method will probably result in a feeding rate that is higher than needed.

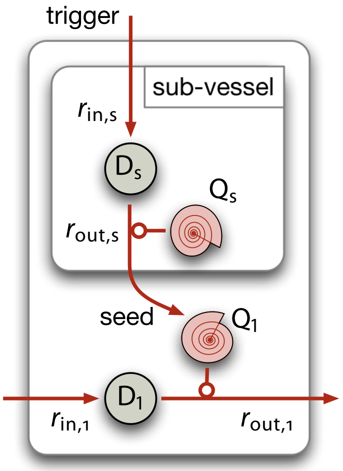


Figure 15.7 Additional replication feed from a separate vessel: The additional support Quine is not affected by the packet stream. It recovers the data-processing Quine by continuously expelling the seed consisting of a blueprint and its active molecule to the outer vessel.

15.2.4 COMPARTMENTATION — DATA-RATE INDEPENDENT BASELINE ROBUSTNESS

Another way to switch-off competition between two individuals is to put them in different habitats. If two Quines are located in different vessels, each with separate memory resources, their growth do not affect each other. This is a simple yet very effective method to separate Quines operating on different packet streams.

Compartmentation is also helpful to increase the survival time of data-processing Quines in general. In Section 14.4.5, we realized that the baseline robustness decreases for increasing packet rates because packets eventually displace all other molecules. We suggested to increase the vessel capacity, which shifted the critical packet injection rate towards higher values.

Instead of adding more resources in order to increase the baseline robustness of the data-processing Quine, we may use the pattern depicted in Figure 15.7. A supporting Quine, similar to the one seen previously, now lives in a separate reaction vessel. The supporting Quine is fed by an external *trigger*, which initiates the replication of the supporting Quine and additionally generates a *seed* that is required to bootstrap the data-processing Quine. The latter Quine lives in a harsh environment subject to an uncontrollable packet stream, which may cause the death of the Quine. However, the isolated supporting Quine is not affected by the packet stream and eventually replenishes the A_1- and B_1-molecules of the data-processing Quine.

trigger

seed

If we study the data-processing Quine in the outer vessel of Figure 15.7 with the tools from Chemical Organization Theory (COT) we recognize that all species, except the reward R_1, are injected from externally. The reward molecule itself is produced in a reaction among A_1- and D_1-molecules. It is no surprise that the outer reaction vessel has one single organization – the steady state in which all molecules are present: $\{D_1, A_1, R_1, B_1\}$. That is, the outer vessel does not exhibit an absorption state anymore since it is able to recover from the loss of every and all species. Consequently, the survival time of the overall system is equal to the survival time of the supporting Quine in the inner vessel. Since the supporting Quine is not affected by the packet stream, its survival time only depends on the trigger molecule injection rate $r_{\text{in,s}}$ and the vessel capacity of the inner vessel N_s. We already quantified the time to absorption of such a single data-processing Quine in Section 14.4.5.

APPLICATION CASE: SELF-HEALING DISTRIBUTED SOFTWARE UPDATES 15.3

The distributed Quine, introduced in Section 14.3, can be used to deploy software in a network: If we put one instance in a single vessel it quickly spreads over the whole network. But we also mentioned before that a distributed Quine is very robust to the invasion of a competitor. Once installed, distributed software built with this pattern is hardly exchangeable. A network operator – or in the future, the system itself – often has to install software updates. We need a method to replace all or certain building blocks of self-healing software. In this section, we provide a method to construct distributed, updatable software, which is still inherently self-healing.

Thereby we revisit three ideas mentioned so far and combine them to obtain a robust software deployment pattern: First, we assume that all networking software consists of data-processing Quines and is therefore intrinsically robust to certain code-level perturbations. Second, we use distributed Quines of type 2, introduced in Section 14.3, which increases the overall robustness of the software by distributing replicas over the network. Third, all Quines are tagged with a version number; Quines with a higher number shall replace Quines with a lower number.* If we manage to unify these three elements in a single Quine structure, the resulting distributed software autonomically updates itself: The distributed replication mechanism propagates old and new software versions while locally interacting old and new Quines destroy the first and proliferate the latter.

In Section 15.3.1, we present our self-healing Quine variant that performs some useful function, autonomically disseminates itself over the network, and updates itself with a coexisting newer version. Then, in Section 15.3.2 we demonstrate the feasibility of this approach with the help of Fraglets simulation results.

*If the networking software comprises of several cooperative Quines, each of them has its own version number and can be updated separately.

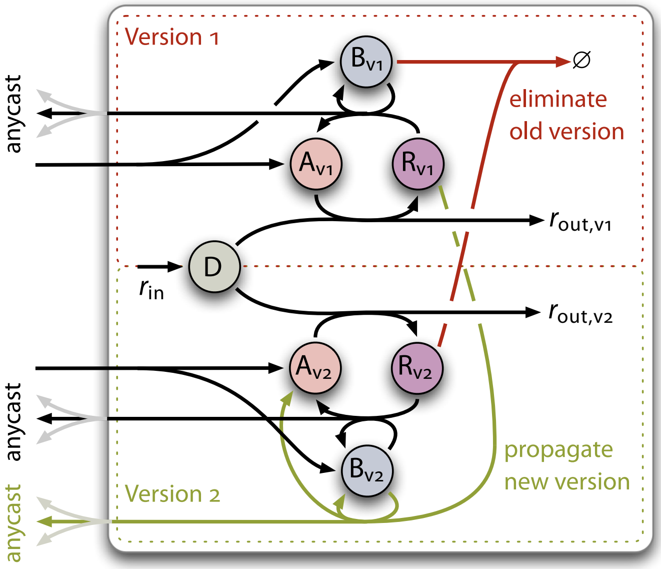


Figure 15.8 Versioned Quine: Multiple versions of the same Quine compete for the same data molecule D. Unlike in Figure 15.3(a) (parallel Quines), their rewards R_{vi} and blueprints B_{vj} interact. Rewards eliminate old blueprints but replicate its own or newer blueprints.

15.3.1 A VERSIONED QUINE

Figure 15.8 depicts the reaction network of the versioned Quine. We assume that multiple versions i of the Quine may coexist in the same vessel for a short time. They process the same data molecule D but, because of their different blueprints B_{vi}, their active variants A_{vi} produce different results.

At first glance this pattern looks like the competing parallel Quines discussed in Section 15.1 (see Figure 15.3(a) on page 286). We mentioned that parallel Quines competing for the same “food” molecule inevitably lead to the extinction of all but one. This is in fact what we are aiming at, but we want to control which of the Quines is able to survive, namely the most recent one. Unlike the parallel Quines studied before, the molecules of the versioned Quine interact. More precisely, the reward R_{vi} only replicates newer blueprints B_{vj} and their active variants A_{vj} $(j \geq i)$. Otherwise, if a reward molecule reacts with an older blueprint, the two molecules annihilate each other. This mechanism provides newer software a selective advantage over the older version.

15.3.2 FRAGLETS SIMULATION RESULTS

We implemented the versioned Quine in Fraglets and performed extensive simulations to study its behavior. A detailed description of the Fraglets implementation is provided in Section A.2.

One particular topology we studied in more detail is the toroidal network of 100 nodes: a regular, fully connected network where each node has four neighbors and a capacity of $N = 100$ fraglets. Starting with empty nodes we

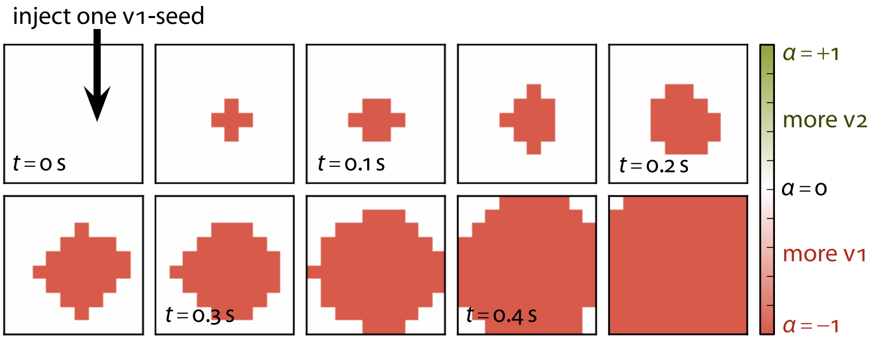


Figure 15.9 Initial spreading of the versioned Quine: Series of snapshots over time in a toroidal topology of 100 nodes for a Fraglets simulation of the versioned Quine. The Quine quickly populates all vessels in the network if no software is installed yet.

injected a single Quine of version 1 into one of the nodes. As depicted in Figure 15.9 the Quine quickly distributes itself over the network. Each image in the figure shows the mixture of Quine versions in the two-dimensional network topology. That is, each pixel represents the node at the corresponding coordinates. We chose a color scheme based on the normalized number of blueprints α, defined as

$$\alpha = \frac{N_{B_{v2}} - N_{B_{v1}}}{N_{B_{v1}} + N_{B_{v2}}} \qquad (-1 \le \alpha \le 1) \qquad (15.1)$$

If a node only contains version 1 Quines, the value α is −1, indicated by a green pixel in Figure 15.9, if it is completely filled with version 2 Quines, α is +1 (red), and if the two Quines are present with equal concentration, α has the value 0 (white).

Once all vessels were populated by Quine version 1 we flushed an arbitrary vessel with version 2 instances by injecting N seeds with the new code. Figure 15.10 shows how the new Quine populates the network: In a first phase ($t < 1$ s), the new Quine spreads to the neighborhood of the source node. While annihilating version 1 blueprints, the new Quine is diminished by version 1 Quines that surge in from the neighbor nodes. In a second phase ($t < 7$ s), the new Quine spreads over the whole network but still has a very small concentration compared to the old version. Suddenly, around $t \approx 7$ s, a region distant from the original source node exhibits a substantial concentration of the new Quine. From this location the new Quine quickly takes over the network and eventually prevails.

Finally, as a crosscheck, we flushed the same source vessel with version 1 instances once the network only hosted version 2 Quines. As shown in Figure 15.11, older Quines are not able to displace more recent software.

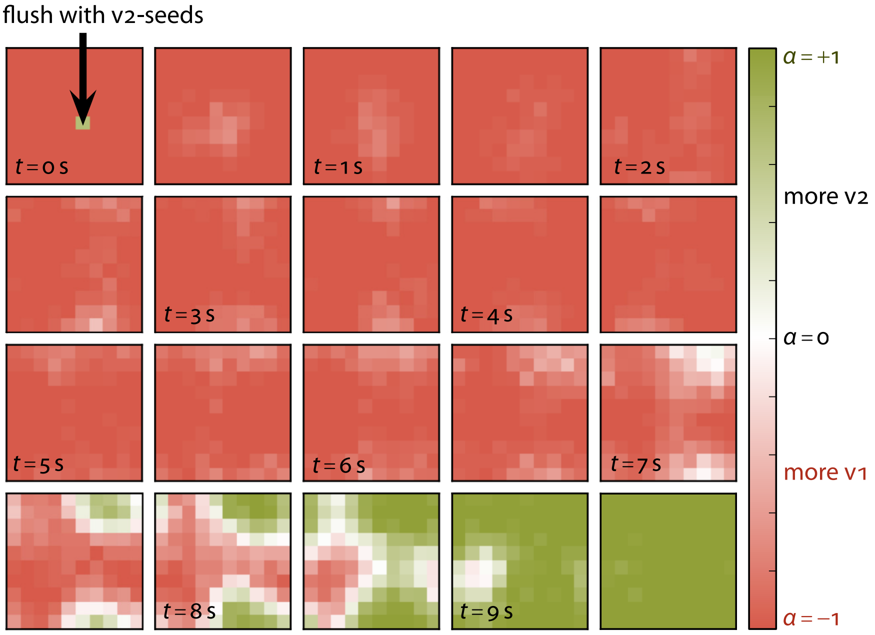


Figure 15.10 Infection with version 2 Quines and their pervasion: Series of snapshots over time in a toroidal topology of 100 nodes for a Fraglets simulation of the versioned Quine. A newer version is spread but attacked at the same time by the inflow of older versions from neighbor vessels. However, eventually the version 2 Quine prevails.

(a) The Update Resistance of Large Networks

The versioned Quine behaves like a distributed chemical switch (Ramakrishnan & Bhalla, 2008). If a newer version of the same Quine is present with a certain threshold concentration, the whole network flips to the new software. Our experiments revealed that this threshold depends on the size of the network. The more old-versioned Quines put pressure against the injected update, the less likely it is that the software update is successful. We showed before that the update succeeds in a network of 100 nodes when completely replacing the older versions by new version instances in a single node. However, already in a network of 400 nodes, the same strategy surprisingly leads to the extinction of the *new* software. The new Quines diffuse too fast over the network and are not able to build-up the critical concentration that is needed for triggering the transition.

One solution that worked for all (tested) network sizes was to continuously inject the new Quine for a certain period of time with a rate that is approximately equal to the vessel size (see Figure 15.12). By continuously seeding the new Quine, we help it getting over the critical threshold and forming a spatially growing topological island of new versions. At the center of the island, all neighbors are populated with new Quines: the island maintains itself. At the border, the inflow of old Quine versions is limited due to the neighborhood to nodes in which the new Quine dominates. Consequently, the island expands and eventually covers the whole network.

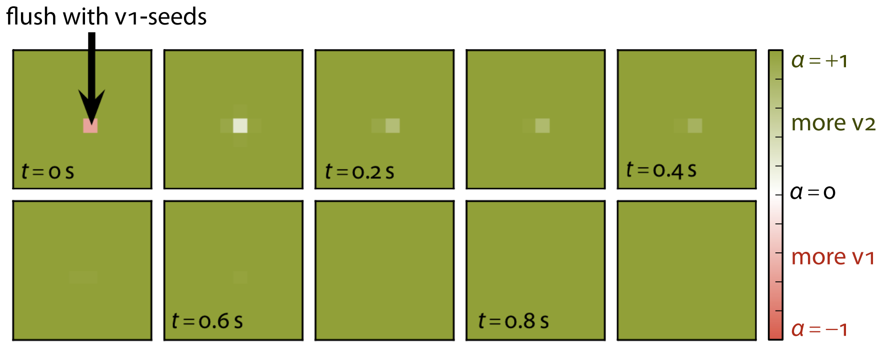


Figure 15.11 Infection with version 1 Quines has no chance: Series of snapshots over time in a toroidal topology of 100 nodes for a Fraglets simulation of the versioned Quine. Even if a vessel is completely filled with an old version, the old Quine has no chance to prevail.

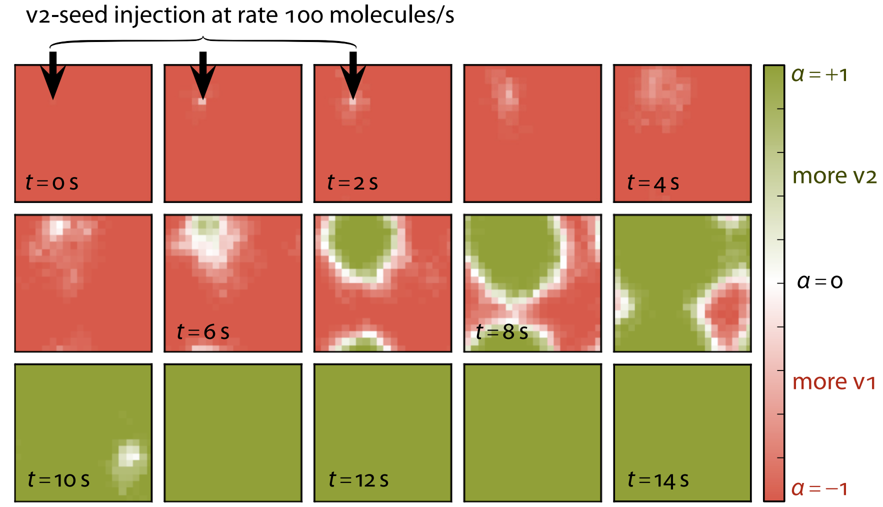


Figure 15.12 Continuous injection of version 2 Quines for a certain time: Series of snapshots over time in a toroidal topology of 400 nodes with a capacity of 100 molecules for a Fraglets simulation of the versioned Quine. A newer version is injected in an arbitrary vessel at rate 100 molecules/s. An island of newer version Quines builds-up and eventually populates the whole network.

SUMMARY 15.4

In the previous chapter, we demonstrated the high robustness of Quines to faults on the execution level, which is the consequence of natural selection. However in this chapter, we highlighted that code may become too robust: once it populates the memory, it can barely be replaced. Furthermore, different software elements often struggle for the same memory resources and compete against each other rather than collaborate. A similar behavior was observed and became famous in Core War (Dewdney, 1984), where hostile programs engage in a battle of memory bits.

Thus, it seems that the adoption of natural mechanisms comes along with natural forces, which we have to tame. We proposed several methods how selfish code can be turned into symbiotic software that still exhibits the desired high level of robustness.

CHAPTER 16

A Self-Healing Multipath Routing Protocol

Design, implementation, and analysis of a chemical routing protocol based on distributed self-replicating Quines.

Ceux qui passent toujours
par les mêmes chemins,
voyent ordinairement toujours
les mêmes objets;
il est rare qu'à force
de suivre différentes routes,
on ne découvre
de nouveaux sujets
dignes de nos attentions
les plus sérieuses. [16]

Expériences de physique
Pierre Polinière

Those who always
take the same paths,
usually see
the same objects;
it is rare that upon
following different routes,
one won't discover
new topics
worthy of our
most serious attention.

Expériences de physique
Pierre Polinière

This chapter demonstrates how self-replicating Quines can be used as building blocks to synthesize a more complex networking protocol – a self-healing multipath routing protocol. Our approach relies on the concentration of routing table entries to stochastically decide which path a packet should take through the network. A reinforcement mechanism rewards successful forwarding rules that compete against each other for delivering data packets over alternative paths. This application case demonstrates the design principles of chemical networking protocols once more. In order to make the

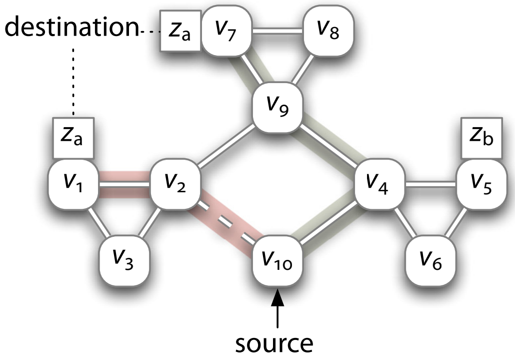


Figure 16.1 Network topology for multipath routing protocol simulations: The network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ consists of 10 nodes $\mathcal{V} = \{v_1, \ldots, v_{10}\}$ and two services: Service z_a is announced by nodes v_1 and v_7, whereas service z_b is only hosted by node v_5. Our simulations send a data stream from node v_{10} to service z_a. The optimal path is (v_{10}, v_2, v_1). If link (v_2, v_{10}) is dropped the best alternative path to the destination service is (v_{10}, v_4, v_9, v_7).

protocol robust to execution and mutation errors, we combine the design motifs presented in the second part (Chapter 10) with the cooperative strategies for Quines in this part (Chapter 15).

This chapter is structured as follows: Section 16.1 first describes the problem of packet routing. In Section 16.2, we provide a short overview of our chemical and self-healing multipath routing approach. Sections 16.3 to 16.5 then explore the protocol in more detail. In Section 16.6, we simulate the protocol in OMNET++ and discuss the results. Finally, in Section 16.7 we assess our protocol and relate it to existing stochastic routing approaches.

16.1 PROBLEM DESCRIPTION

The purpose of a routing protocol is to setup forwarding state in network nodes such that data packets can be forwarded to their destination. At the same time, the protocol should maximize throughput and minimize packet loss. Here we additionally require the routing software to be able to tolerate disruptions in its own code base, meaning that the protocol continues to operate and heals itself from any molecule deletion attack as soon as possible. Once more, we assume that spontaneous memory alterations or execution errors lead to neutral fraglets that do not interact with the protocol software.

We assess the protocol with the help of OMNET++ simulations of the network topology depicted in Figure 16.1, which consists of 10 nodes that are connected by bidirectional links. Selected nodes offer *services*, i.e. possible destinations for data packets. Each service, as well as each node, has a globally unique identifier, i.e. a string representing the *address* of the node or service. Service addresses may differ from the addresses of the hosting nodes. This allows a given service to be replicated and offered at different locations. In this case, data packets destined to the service can be sent to any node that announces the service. For example, service z_a is offered by nodes v_1 and v_7. This scheme implements a data centric network model (Krishnamachari,

services

addressing

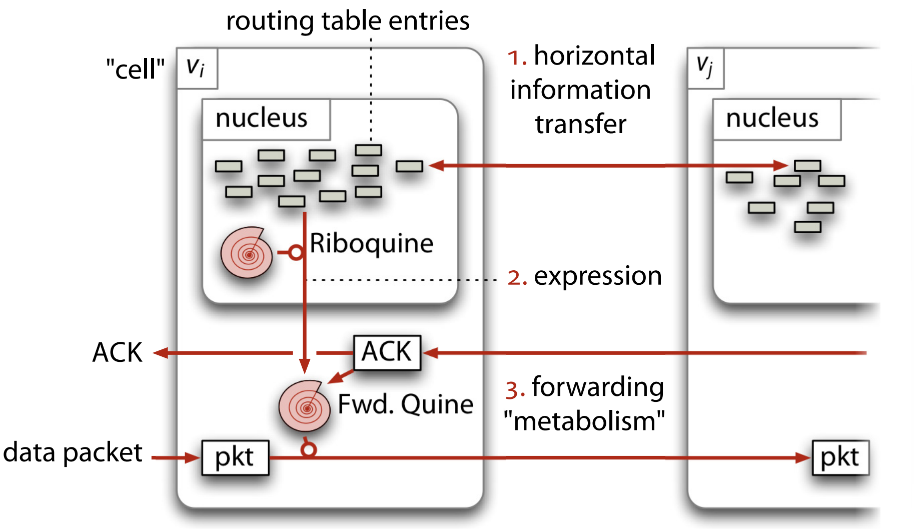


Figure 16.2 Cell metaphor for the multipath routing protocol: Each node (=cell) contains two vessels: the main vessel in which the forwarding logic sends packets to the next hop and the "nucleus", which host the routing table entries (RTE). The RTEs are exchanged among the nuclei in a diffusion-like process. The "Riboquine" "expresses" RTEs, i.e. it uses them as a blueprint for creating active forwarding rules (Quines).

Estrin, & Wicker, 2002) where packets for a given destination service are forwarded to any (preferentially to the closest) node that hosts the service. In this context, a routing protocol is responsible for announcing the availability of services in nodes and to disseminate this information over the network.

Formally, we extend the network graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ (see Section 5.1.1) by a set of services $\mathcal{Z} = \{z_a, z_b\}$ and a set of associations between services and nodes (*bindings*) $\mathcal{B} = \{(z_a, v_1), (z_a, v_7), (z_b, v_5)\}$. We define a function $\text{bnd}\colon \mathcal{B} \to \{0, 1\}$, which yields 1 if the service is bound to the node, or 0 otherwise. For example, $\text{bnd}(z_a, v_7) = 1$.

bindings

16.2 PROTOCOL OVERVIEW

As depicted in Figure 16.2, the design of our protocol is inspired by the structure of an eukaryotic cell (Martin & Russell, 2003) featuring a nucleus. That is, each network node has two nested reaction vessels: The outer vessel serves as forwarding engine that sends incoming data packets to one of the neighbor nodes. A second vessel – the *nucleus* – contains the "genome", which in our case accumulates information about the topology of the network in form of Routing Table Entries (RTEs). Linking both vessels, there is the "Riboquine", inspired by ribosomes in cells. The Riboquine is responsible for "expressing" the nucleus' routing table entries as active forwarding rules and expelling them to the main vessel.

nucleus

Riboquine

The first task of our proactive routing protocol is to *gather information about the network topology*: Each node has to obtain knowledge about which service can be reached over which neighbor node(s) as for traditional distance vector routing protocols. Unlike in traditional routing protocols, we do not

gather information about the network topology aim at immediately find the best path to a service; instead, the transmission paths are later reinforced by the forwarding engine.

path reinforcement

Path reinforcement is based on a competition and reward mechanism for forwarding rules. Active fraglets represent forwarding rules, i.e. tuples (destination service, next hop). A forwarding rule reacts with a passive data packet destined to a given service and sends it to the next hop. The stochastic reaction algorithm makes sure that one of the competing rules for the same destination is picked at random but in proportion to their multiplicity. When a packet finally reaches the destination service, an acknowledgment (ACK) is sent back along the reverse path. This ACK packet reacts with all corresponding rules that were responsible for forwarding the data packet in the first place and triggers their replication. Thus, whenever a forwarding rule contributed to the successful delivery of a packet, it is replicated, i.e. its multiplicity increases, and hence the probability of being chosen for a next forwarding task rises.

This reward mechanism results in the most efficient path being reinforced; less efficient paths quickly vanish due to the dilution flux. This means that if there is a preferential path to a certain destination the corresponding forwarding rules may completely displace rules for a suboptimal alternative path. This would be a problem if the forwarding rules would be the only place where we keep topological information about the network's structure. However, this information is also kept inside the nucleus by routing table entries, which are effectively shielded from the fierce competition of the path optimization process. The role of the nucleus is therefore to *cultivate a variety of alternative paths* and to continuously inject the entries to the forwarding engine. Even if rules for suboptimal paths are squeezed out when another path suddenly becomes attractive, the nucleus maintains the blueprints of the rules and is able to re-instantiate them later. This is an application of the compartmentation strategy of cooperation, as introduced in Section 15.2.4.

cultivate a variety of alternative paths

The remaining chapter is structured as follows: In Section 16.3, we first show how routing information is disseminated among the nuclei to build up a multiset of routing table entries in each node. Then, in Section 16.4, we discuss the Riboquine before we analyze the self-optimizing forwarding engine in Section 16.5. In Section 16.6, we present and discuss our simulation results. Finally, in Section 16.7, we assess our chemical self-healing routing protocol and relate it to existing stochastic routing protocol approaches.

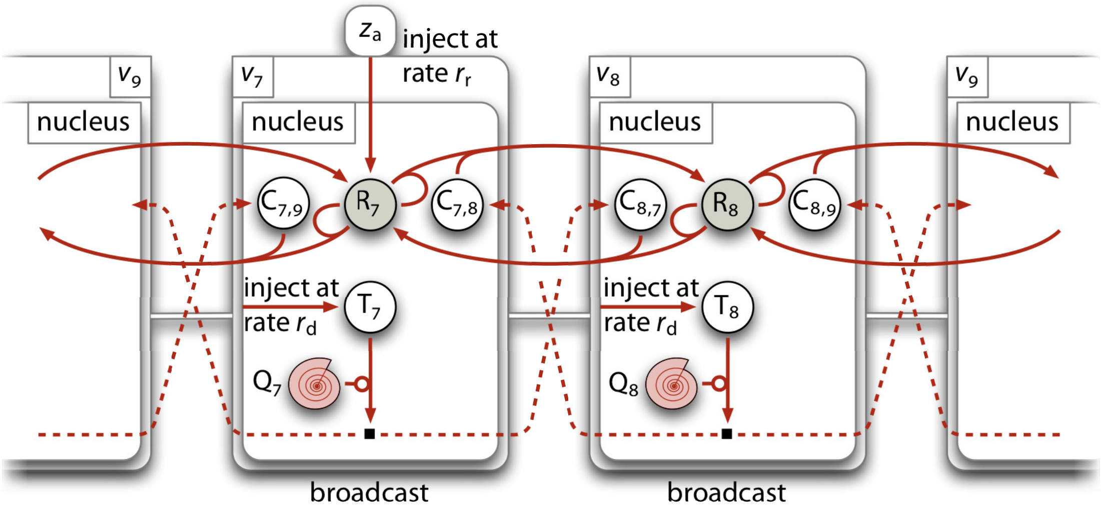


Figure 16.3 Reaction network to disseminate routing table entries: Each service injects routing table entries R with rate r_r to the nucleus of the hosting node. Periodically injected (rate r_t) trigger molecules T invoke the Quine Q, which broadcasts a fetch-molecule C to its neighbors. There, it reacts with a random routing table entry and fetches a copy of it.

DISSEMINATION OF ROUTING TABLE ENTRIES 16.3

In the chemical model, it is natural to represent a *routing table* as a multiset of molecules. Each routing table entry (=molecule instance) contains a tuple $\left(z_m, \text{path}(z_m)\right)$, where $\text{path}(z_m)$ is a list of nodes along a *path* to the destination service. In order to keep a high variety of alternative paths to the same destination, the dissemination protocol periodically obtains copies of routing table entries from all neighbors and at the same time distributes its own software.

routing table

routing path

Figure 16.3 shows the resulting distributed reaction network for a part of our topology. The dissemination protocol is formally described by the following reactions for each node $v_i \in \mathcal{V}$ (a verbose description follows below):

Injection of service announcements:

$$\varnothing \xrightarrow{r_r} \mathrm{R}_{i,j} \qquad \forall \left\{z_j \in \mathcal{Z} \mid \text{bnd}(z_j, v_i) = 1\right\} \tag{16.1a}$$

Injection of distribution triggers:

$$\varnothing \xrightarrow{r_t} \mathrm{T}_i \tag{16.1b}$$

Broadcast of a fetch-entry molecule:

$$\mathrm{T}_i + \mathrm{QA}_i \longrightarrow \mathrm{QR}_i + \sum_{v_j \in \mathcal{N}_i} \mathrm{C}_{j,i} \tag{16.1c}$$

Distributed replication of the Quine:

$$\mathrm{QR}_i + \mathrm{QB}_i \longrightarrow \mathrm{QA}_i + \mathrm{QB}_i + \sum_{v_j \in \mathcal{N}_i} \Big(\mathrm{QA}_j + \mathrm{QB}_j\Big) \tag{16.1d}$$

Fetching a random remote routing table entry:

$$\mathrm{C}_{i,j} + \mathrm{T}_i \longrightarrow \mathrm{T}_i + \mathrm{T}_j \tag{16.1e}$$

Let us discuss the five reactions in more detail: (16.1a) Each service periodically injects routing table entries into its host's nucleus. For example, in the topology depicted in Figure 16.1, service z_a injects such *service announcement* fraglets with rate r_r into the nuclei of nodes v_1 and v_7; in particular, these are the fraglets $_{v_1}$`[RTE za]` and $_{v_7}$`[RTE za]`, respectively.

service announcement

The dissemination protocol works similar to the first design variant of *Disperser* in a network with unknown topology (see Section 10.4.1) where we collected information about the neighborhood by using broadcast echo requests. Here, we aim at receiving routing table entries from all neighbors at a given rate, and we use a distributed Quine in order to make the protocol self-healing: The data-processing Quine Q_i consists of three molecules: the active molecule QA_i, the replication reward QR_i, and the blueprint QB_i (see also Section 14.4).

(16.1b) A trigger fraglet T_i injected at rate r_t invokes the replication of the Quine. (16.1d) We use the strategy presented in Section 14.3 to distribute a Quine in the network: Rather than creating two copies locally, one of them is broadcasted to the neighbors. Like this, a new node containing no software is automatically *bootstrapped* by its neighbors. (16.1c) At the same time, the Quine broadcasts an active fraglet to its neighbors $v_j \in \mathcal{N}_i$. (16.1e) There, this active fraglet reacts with a random routing table entry and sends a copy back to the originating node v_i after it appended the neighbor's node identifier v_j. The following reaction trace illustrates a simplified version of this last reaction (16.1e) and the symbolic operation that is applied to the routing table entry:

bootstrap

$$
\begin{aligned}
& {}_{v_j}[\texttt{matchs RTE snode \_ send } v_i \texttt{ RTE}] + {}_{v_j}[\texttt{RTE za}] \\
\longrightarrow\ & {}_{v_j}[\texttt{RTE za}] + {}_{v_j}[\texttt{snode \_ send } v_i \texttt{ RTE za}] \\
& {}_{v_j}[\texttt{snode \_ send } v_i \texttt{ RTE za}] \\
\longrightarrow\ & {}_{v_j}[\texttt{send } v_i \texttt{ RTE za } v_j] \\
\longrightarrow\ & {}_{v_i}[\texttt{RTE za } v_j]
\end{aligned}
$$

Reaction (16.1e) additionally *checks for routing loops*. That is, node v_i drops the routing table entry $\big(z_m, \text{path}(z_m)\big)$ if v_i is already part of $\text{path}(z_m)$.

avoid routing loops

The last line illustrates that node v_i has collected an RTE-entry, which states that service z_a is reachable via node v_j. Multiple RTE-entries will accumulate in the nucleus and will be subject to a *dilution flux*: This eventually removes entries for services that no longer exist.

dilution flux

Note that the routing table entries are passive molecules, i.e. they cannot perform actions themselves. The next task for our protocol is to generate active forwarding rules based on these passive entries.

EXPRESSION OF ROUTING TABLE ENTRIES 16.4

We again use a Quine to "express" the collected passive routing table entries and to expel their active version to the outer vessel where they constitute the forwarding algorithm. We call this Quine the *Riboquine* in analogy to the ribosomes in eukaryotic cells that translate genetic information (mRNA) into proteins (Lafontaine & Tollervey, 2001). The Riboquine is triggered periodically with rate r_e. The Riboquine also distributes itself to the neighbor nodes.

Riboquine

Unlike in biological cells where ribosomes are located in the rough endoplasmic reticulum within the cytoplasm, our Riboquine is located in the nucleus. This location is important because it protects the molecules involved in the dissemination and expression of routing table entries from the competition for resources in the outer vessel, as we will see later. Note that in steady-state, and if there are no data packets to be forwarded in the outer vessel, the concentration of forwarding rules is proportional to the concentration of the corresponding routing table entries in the nucleus.

The Riboquine generates two different kinds of forwarding rules: one for the case where the target service is locally present and another rule if the packet has to be forwarded to a neighbor node. Technically, the first case is recognized by the fact that the corresponding RTE-fraglet is of length two, for example [RTE za] for a local service z_a, for which the Riboquine installs

local delivery

forwarding

a local delivery rule [match sa deliver]. For the other case, if the routing table entry is longer than two symbols (e.g. [RTE za v1 v2]), the Riboquine produces a forwarding rule of the type [match za send v2 za]. The exact structure and operation of the forwarding rules will be discussed in the next section.

16.5 FORWARDING PATH REINFORCEMENT

As we have shown in Section 7.1, a natural way to implement a forwarding engine in Fraglets is to use active molecules ([match za send ...]) as forwarding rules: A data packet of the form [za ...] will react with such a rule, leading to its transmission to the corresponding next hop. The forwarding state in node v_{10} could for example look as follows:

$$
\begin{aligned}
&{}_{v_{10}}\texttt{[match za send v2 za]}60\\
&{}_{v_{10}}\texttt{[match za send v4 za]}20\\
&{}_{v_{10}}\texttt{[match zb send v4 zb]}40
\end{aligned}
$$

In this example, there are two competing rules for a data packet destined to service z_a. According to the law of mass action, the reaction probability is proportional to the concentration of the reaction rules. In our example, 75 % of the packets to z_a will be sent to neighbor v_2 while 25 % travel via v_4.

A forwarding rule is consumed when it is used to send a data packet. The rate of replenishment by the Riboquine is much smaller than the rate at which data packets arrive. Therefore, we need a mechanism that regulates the concentration of forwarding rules by selectively replicating those that are performing well.

16.5.1 REINFORCEMENT OF THE OPTIMAL PATH BY SELF-REPLICATING FORWARDING RULES

forwarding Quine

Instead of producing simple forwarding rules, the Riboquine actually expresses *forwarding Quines*, i.e. packet processing Quines that replicate and are able to maintain a certain concentration. But unlike the traditional data-processing Quine (see Section 14.4), which generates its replication reward immediately, the replication reward is generated by the acknowledgment packet that is received upon the successful delivery of a data packet.

reward

That is, as sketched in Figure 16.4, we install a stream of feedback fraglets that *reward* those rules which participated in the successful delivery of data packets. Because the reward has to replicate all forwarding Quines along the

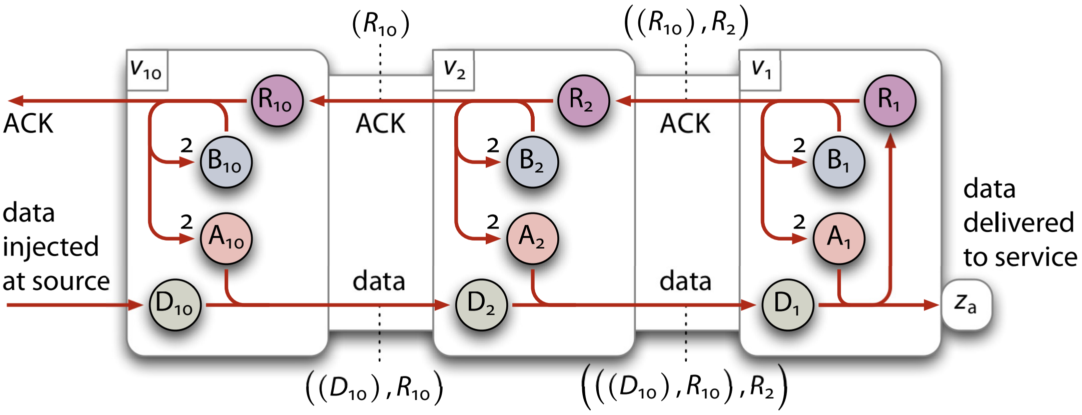


Figure 16.4 **Reaction network of a single forwarding path:** A chain of Quines sends a data packet over the path (v_{10}, v_2, v_1) to the destination service z_a. On the forward path, each Quine attaches its own reward R_i to the data packet. The Quine in the destination node replicates itself and sends back the accumulated ACKs using source routing. On the reverse path, each reward R_i replicates the Quine responsible for the successful delivery of the data packet.

path, we construct the reverse route by appending the identifier of each node along the packet's journey to the data packet.

At the end of the forwarding chain, a delivery Quine creates an acknowledgment packet and embeds the collected rewards for all forwarding Quines along the packet's path. This ACK-fraglet traverses the path in reverse order and, by successively delivering the rewards, it triggers the replication of all those Quines that were responsible in the successful delivery of the packet. Consequently, these Quines increase their relative concentration with respect to other forwarding Quines in the same node, resulting in a higher probability that a packet to the same destination will take the same path again.

FORMAL CONVERGENCE PROOF 16.5.2

This scheme leads to a perfect packet balance among two paths in a situation depicted in Figure 16.5. If the source node's reaction vessel is saturated, its molecules either belong to Quine 1 or 2, as other molecules have been squeezed out. Let us denote the relative concentrations of the two Quines by $x_{\text{src},1}$ and $x_{\text{src},2}$, respectively, satisfying $x_{\text{src},1} + x_{\text{src},2} = 1$. Since replication is triggered by received acknowledgments, these concentrations are

$$x_{\text{src},1} = \frac{r'_1}{r'_1 + r'_2} \tag{16.2a}$$

$$x_{\text{src},2} = \frac{r'_2}{r'_1 + r'_2} \tag{16.2b}$$

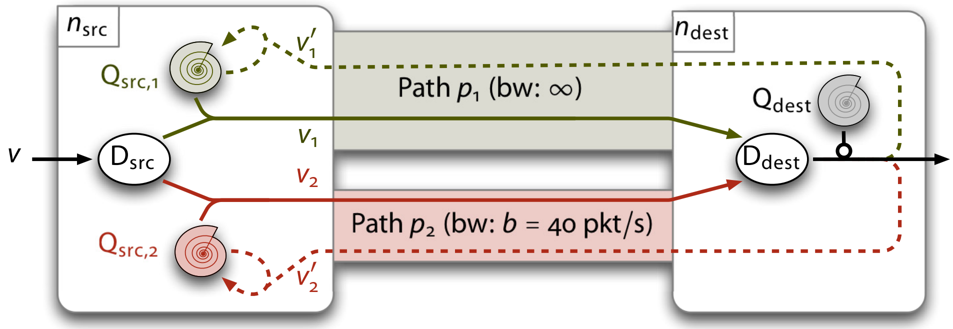


Figure 16.5 Competition among two forwarding Quines: The Quines replicate with a rate that is equal to the rate of received ACKs. Consequently, the "fitness" of a path is reflected by the concentration of the Quines. Because to the law of mass action scheduling, the transmission rate adapts to the available bandwidth.

where r'_i is the rate of acknowledgments received over path p_i.

Let us assume that the bandwidth of p_1 is infinite whereas p_2 drops packets exceeding b. We examine the overload situation where the total rate $r > 2b$. Consequently, the rate of acknowledgments is $r'_1 = r_1$ and $r'_2 = \min(r_2, b)$. Due to the law of mass action, the fraction of packets sent over p_1 is proportional to the concentration of Quine 1:

$$r_1 = x_{\mathrm{src},1} \cdot r = \frac{r_1}{r_1 + \min(r_2, b)} \cdot r \tag{16.3}$$

Hence,

$$r_1 = r - b \tag{16.4a}$$

$$r_2 = r - r_1 = b \tag{16.4b}$$

Quine 2 reduced its concentration so as to only forward packets up to the bandwidth limitation of path p_2, as was to be proved. □

16.6 RESULTS

This section illustrates how our protocol responds to topological changes in the network and how it responds to link delay and packet loss as well as to code deletion attacks. We will always use the network topology depicted in Figure 16.1, for which we perform OMNET++ simulations. We study the impact of simulated perturbations on a data stream from a source in node v_{10} to the destination service z_a. The vessel capacity of all nuclei is set to 500 molecules, whereas the outer vessels, which host the forwarding Quines, have a capacity of 1000 molecules.

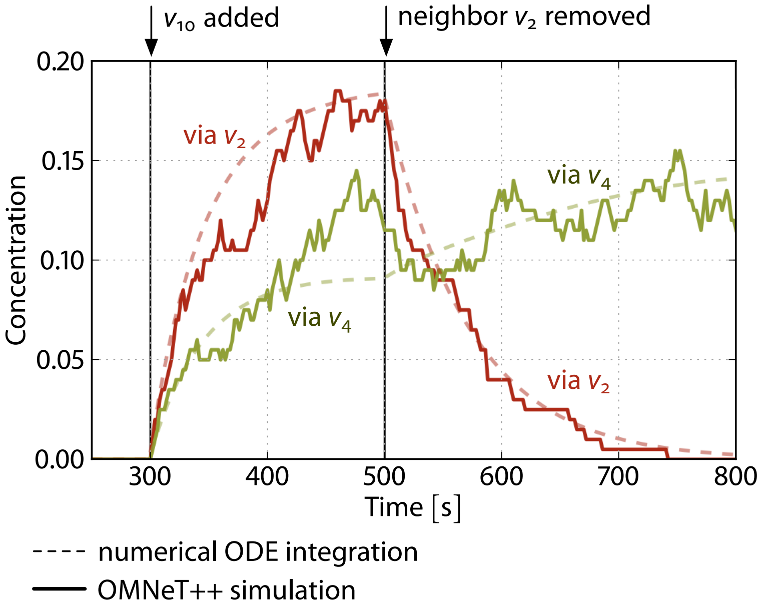


Figure 16.6 Slow diffusion of routing table entries: Concentration (relative number) of routing table entries in the nucleus of source node v_{10} for service z_a (see topology in Figure 16.1). The vessel capacity of the nucleus is 500 molecules and routing table entries are exchanged once a second. The fluctuating curves were obtained from an omnet++ simulation whereas the dashed curves from numerically integrating the ODEs of the reaction network. At time $t = 300$ s, the source node is added to the network. At time $t = 500$ s, its neighbor v_2 is removed.

First, in Section 16.6.1, we examine the speed of the diffusion process of routing table entries. In Section 16.6.2, we then demonstrate that after a link outage, the data stream is automatically deviated over an alternative link. In Section 16.6.3, we show that packets are preferably sent over links without packet loss with little impact on the data stream itself. Section 16.6.4 reveals a rather counter-intuitive phenomenon: delayed paths seem to have a small benefit compared to ideal ones. Finally, in Section 16.6.5 we highlight the robustness of the protocol software against code and data deletion attacks.

DIFFUSION OF ROUTING TABLE ENTRIES 16.6.1

We simulated the dissemination of routing table entries in OMNET++ and complemented our measurements with expectations obtained by numerically integrating the ODEs derived from the reactions (16.1a)–(16.1e). Figure 16.6 shows the concentration of routing table entries in the source node v_{10} for service z_a. The source node joins the network at time $t = 300$ s, its neighbor v_2 leaves it at time $t = 500$ s. After v_{10} joined, the number of routing table entries for service z_a starts to rise as a consequence of the broadcasted fetch-fraglets. Because of the limited vessel capacity and the resulting dilution flux, the entry distribution reaches a steady state in which there are more entries received from neighbor v_2 than from v_4. After neighbor v_2 disconnected at time $t = 500$ s, the multiplicity of routing table entries slowly adapts to favor the link via the remaining neighbor v_4.

As shown, the routing table only slowly adapts to topological changes. For instance, it takes about 300 s to forget about link (v_{10}, v_2) after node v_2 has

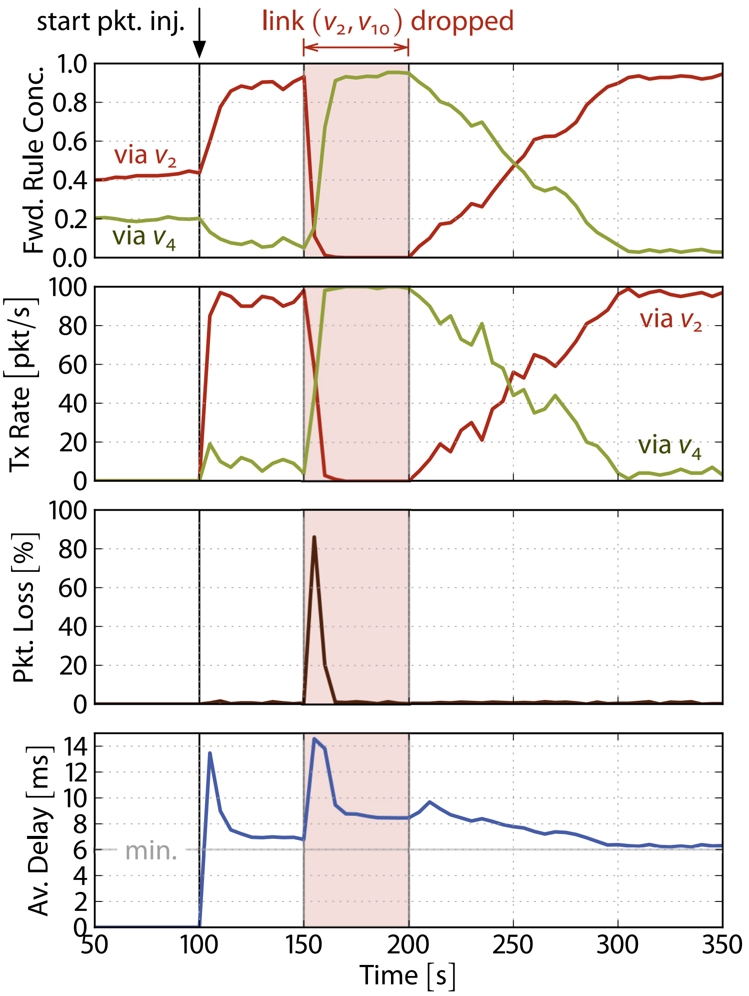


Figure 16.7 Forwarding performance with link outage: OMNeT++ simulation of the multipath routing protocol. Data packets for destination service z_a are injected to source node v_{10} at rate 100 pkt/s (see topology in Figure 16.1). At time $t = 100$ s, data transmission starts, and the packets are mainly sent over the shortest path via neighbor v_2. The link to node v_2 is dropped at time $t = 150$ s for 50 s. All measurements were collected in node v_{10}: **Fwd. Rule Conc.:** Relative number of forwarding rules to either neighbor v_2 or v_4. **Tx Rate:** Rate of data packets transmitted to either neighbor. **Pkt. Loss:** Percentage of packets lost between source node v_{10} and destination service z_a. **Av. Delay:** Average packet delay from source to destination.

been switched off. But remember that we are only interested in the diversity of alternative paths. We expect the reinforcement algorithm in the forwarding engine to react faster to topological changes, and demonstrate this in the remainder of this section.

16.6.2 ADAPTION TO TOPOLOGICAL CHANGES

The most typical topological change in a network is the temporary outage of a link or node. Here we simulate an outage of link (v_2, v_{10}). Apart from this disruption, all links are ideal: they neither lose packets nor deliver them deferred. The results are summarized in Figure 16.7:

Before $t = 100$ s, as depicted in the top subfigure, the concentration of forwarding rules reflects the concentration of the corresponding routing table entries in the nucleus, which currently favors the shorter path over neighbor v_2. At time $t = 100$ s, we start injecting data packets with a rate of 100 pkt/s. Consequently, as shown in the subfigure just below, most of the traffic is sent via v_2 and the forwarding rule is able to replicate faster than the competing rule, which sends packets via v_4.

At time $t = 150$ s, the link (v_2, v_{10}) is disconnected. In the source node there is still a stock of rules sending packets via v_2, but they don't receive acknowledgments anymore and are not able to replicate; the packet loss tentatively rises to 80 % (see the third subfigure in Figure 16.7). The dilution of these rules is to the credit of rules that send packets via v_4. They are able to gain concentration and are more likely to be chosen for packet forwarding in turn. Hence, the data stream is redirected over the alternative link.

Later, at time $t = 200$ s, we reconnect link (v_2, v_{10}). The forwarding Quine over v_2 only has a small advantage over its competitor. This is because the transmission paths over both neighbors exhibit a similar packet loss rate and end-to-end delay. In other words, the system only shows a small attraction towards the optimal path.

As noted before, chemical reactions afflict packets with a delay, which is inversely proportional to the number of forwarding rules. In the bottom of Figure 16.7 this is clearly visible. The higher the overall number of forwarding rules the shorter is the delay. The minimal end-to-end delay in our setup is 6 ms: In the optimal case, a vessel's capacity of 1000 molecules is shared between blueprints and forwarding rules for the same destination. That is, the maximum number of rules is 500, yielding a delay of 2 ms per vessel. The shortest path from source node v_{10} to destination service z_a traverses three vessels: v_{10}, v_2, and v_1. Thus, the average overall end-to-end delay cannot be shorter than 6 ms.

PACKET LOSS: STRONG ATTRACTION OF THE OPTIMAL PATH 16.6.3

Next, we examine the situation where the link (v_4, v_{10}) exhibits a packet loss probability of 10 % while the remaining network is still ideal. In this case, the optimal path exhibits a stronger attraction. Figure 16.8 shows the detailed results (also compare to Figure 16.7):

After we start to inject data packet at time $t = 100$ s, the forwarding rules over the ideal link receive all ACKs and therefore "win the competition", meaning that rules over v_4 become extinct. In other words: there is no alternative forwarding rule when the primary link goes down at $t = 150$ s. In the next 20 s, packets are only forwarded by the few forwarding Quines re-generated by the Riboquine. The remaining data packets are accumulated in the vessel unless the ACKS received from v_4 replicate the alternative Quine in sufficient quantity. After this obstacle is overcome, the alternative rule quickly becomes stronger, because there is no competition anymore. The long response time of the alternative rule can be reduced by increasing the rate

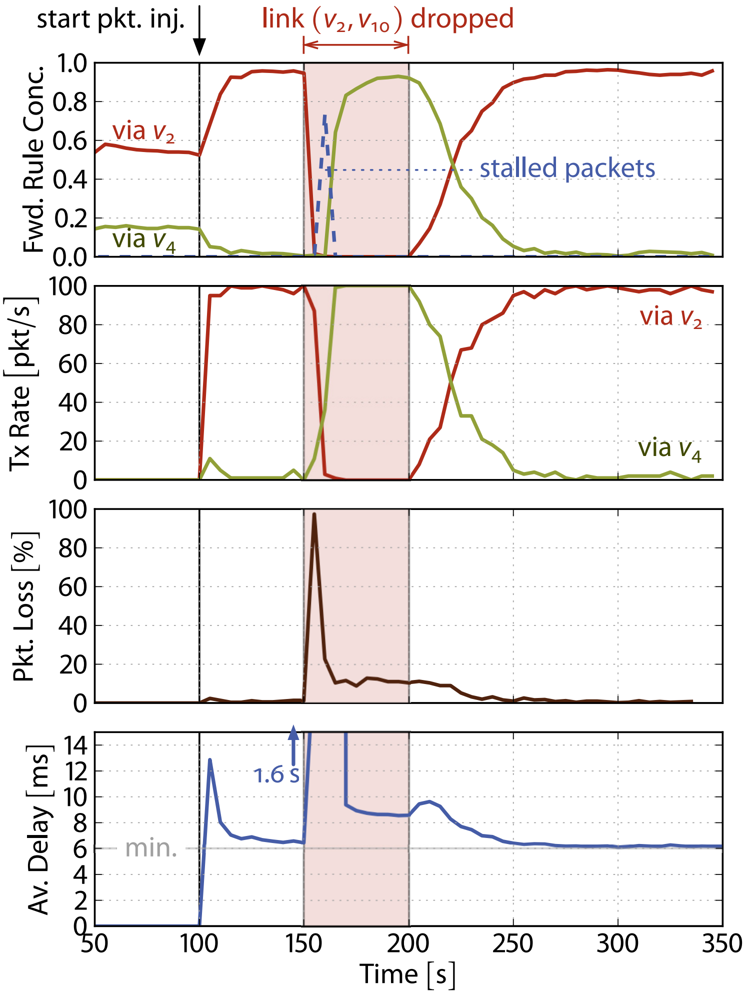


Figure 16.8 Forwarding performance with link outage and packet loss: OMNeT++ simulation of the multipath routing protocol. Data packets for destination service z_a are injected to source node v_{10} at rate 100 pkt/s (see topology in Figure 16.1). The packet loss probability of link (v_4, v_{10}) is 10 %. At time $t = 100$ s, data transmission starts, and the packets are mainly sent over the shortest path via neighbor v_2. The link to node v_2 is dropped at time $t = 150$ s for 50 s. All measurements were collected in node v_{10}: **Fwd. Rule Conc.:** Relative number of forwarding rules to either neighbor v_2 or v_4. Additionally, the concentration of data packets waiting for transmission is plotted with a dashed curve. **Tx Rate:** Rate of data packets transmitted to either neighbor. **Pkt. Loss:** Percentage of packets lost between source node v_{10} and destination service z_a. **Av. Delay:** Average packet delay from source to destination.

r_e, at which the Riboquine expresses routing table entries. Once more, this highlights the importance of the separate nucleus, which maintains diversity in alternative paths despite the violent dynamics of the outer vessel. Finally, at time $t = 200$ s, when we reconnected link (v_2, v_{10}), the loss-less path quickly outperforms the alternative path over v_4 as expected. A direct comparison of Figure 16.8 to Figure 16.7 shows that the attraction of the optimal path is much stronger if the two paths exhibit a different packet loss rate.

16.6.4 DELAY: WEAK ATTRACTION OF THE OPTIMAL PATH

Finally, we examine the behavior of the protocol when links deliver packets deferred. We again start with an ideal network, but link (v_4, v_{10}) afflicts traversing packets with a delay of 1 s. A simulation trace of this scenario is shown in Figure 16.9.

After the optimal link is dropped at time $t = 150$ s, all rules over the alternative but delayed path are quickly consumed. It lasts one second until

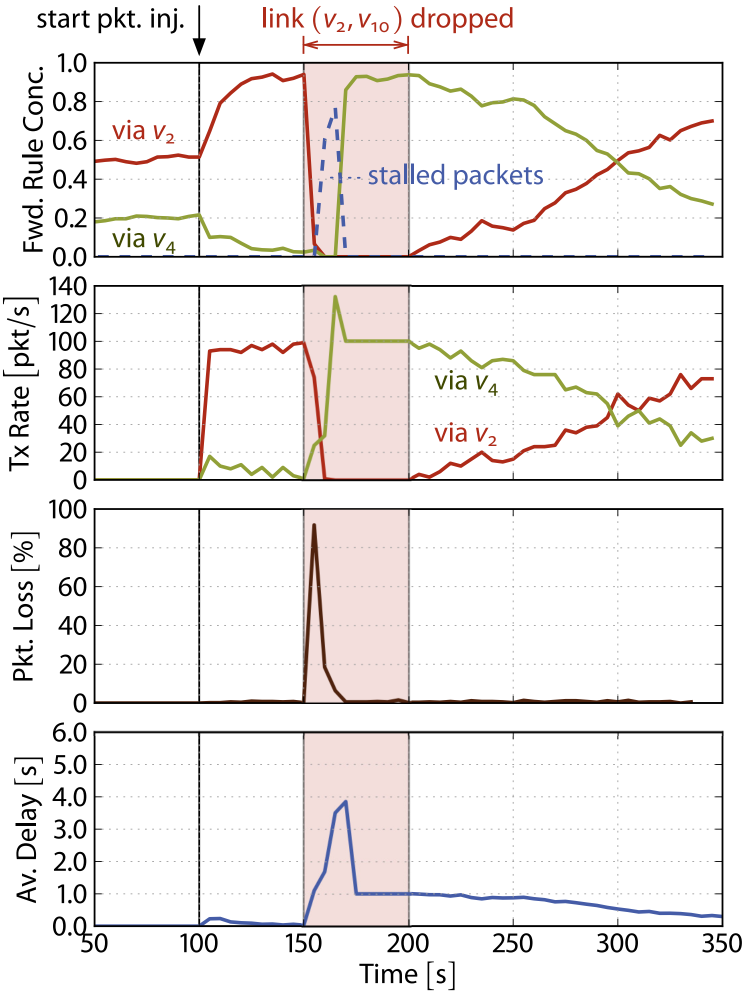


Figure 16.9 Forwarding performance with link outage and delay: OMNeT++ simulation of the multipath routing protocol. Data packets for destination service z_a are injected into source node v_{10} at rate 100 pkt/s (see topology in Figure 16.1). The delay of link (v_4, v_{10}) is 1 s. At time $t = 100$ s, data transmission starts, and the packets are mainly sent over the shortest path via neighbor v_2. The link to node v_2 is dropped at time $t = 150$ s for 50 s. All measurements were collected in node v_{10}: **Fwd. Rule Conc.:** Relative number of forwarding rules to either neighbor v_2 or v_4. Additionally, the concentration of data packets waiting for transmission is plotted with a dashed curve. **Tx Rate:** Rate of data packets transmitted to either neighbor. **Pkt. Loss:** Percentage of packets lost between source node v_{10} and destination service z_a. **Av. Delay:** Average packet delay from source to destination.

the replication rewards return. Consequently, the injected data packets accumulate as shown in the top subfigure. After the optimal link (v_2, v_{10}) is reconnected at time $t = 200$ s, surprisingly, re-adaption is much slower compared to the scenario without delays (compare to Figure 16.7). A delayed path seems to have a competitive advantage compared to an ideal one:

Even if the round-trip time is longer over a delayed path, the rate at which the ACKs return is the same. After the initial round-trip time, the forwarding Quine replicates with the same rate as its competitor. Even if the latter had a head start in gaining weight, the force to displace other molecules is the same for both Quines. Additionally, the buffering capacity of a link rises with its delay: more packets are "on the wire" at the same time, and these packets are not subject to the dilution flux of any reaction vessel.* If a data stream is able to "store" packets in the non-diluted environment of a delayed link, it has a small competitive advantage. This explains why the reaction network does not react to delays as intuitively suggested.

*Remember that data packets in a reaction vessel are also subject to dilution, since they have to wait a short time before being picked up by a forwarding rule.

The reason that the path over the ideal link finally wins the competition is due to the fact that in our topology, the optimal path traverses less nodes: The system is more sensitive to the number of hops than to the actual round-trip time.

16.6.5 SELF-HEALING ASPECTS

The design of our routing and forwarding protocol makes rigorous use of Quines. In order to let them cooperate, we applied the following strategies:

protected environment

In the nucleus, we foster a *protected environment* in which every node is able to control its net production rate; there is no uncontrolled inflow of molecules. The reference clocks, generated by the injection rates of the trigger molecules (r_r and r_e), control the rate at which routing table entries are imported from neighbors and therefore the growth rate of the molecules in the nucleus; the Riboquine expels its products to the main vessel. Thus the dynamic behavior of the nucleus is predictable and independent of the forwarding traffic, and since all the code in the nucleus is made of Quines, which are replicated with the same rate, the whole dissemination and expression process is robust to the loss of any of its molecules. The nucleus may even lose all molecules at once, in which case the neighbor nodes eventually regenerate the protocol software by the distributed replication process.

The Quines in the forwarding engine are not able to reproduce themselves immediately, because the required reward is deferred by the acknowledgment mechanism. The forwarding Quine may therefore become extinct, for example if data or ACK packets are lost. However, this extinction of code is desired, because it leads to the reinforcement mechanism. As a backup strategy, the Riboquine periodically repopulates the forwarding engine.

purge attacks

Figure 16.10 shows a simulation trace where molecules suffer *purge attacks*. At time $t = 150$ s, we removed all molecules from both reaction vessels in the observed source node v_{10} while a data packet stream is flowing through the vessel. Immediately after the attack, the nucleus is bootstrapped by the neighbors, it quickly learns the network topology from the neighbors and generates new forwarding rules. Even though the concentration drop is clearly visible in Figure 16.10, its effect to the forwarding rate and packet loss is barely noticeable whereas the delay temporally increases. During a longer lasting attack, data packets may accumulate due to the reduced number of forwarding rules and the end-to-end delay will increase even more. However, as soon as the attack is over, the accumulated data packets will temporarily increase the reaction rate of the forwarding Quine due to the law of mass action, which will compensate for the reduced transmission rate during the attack.

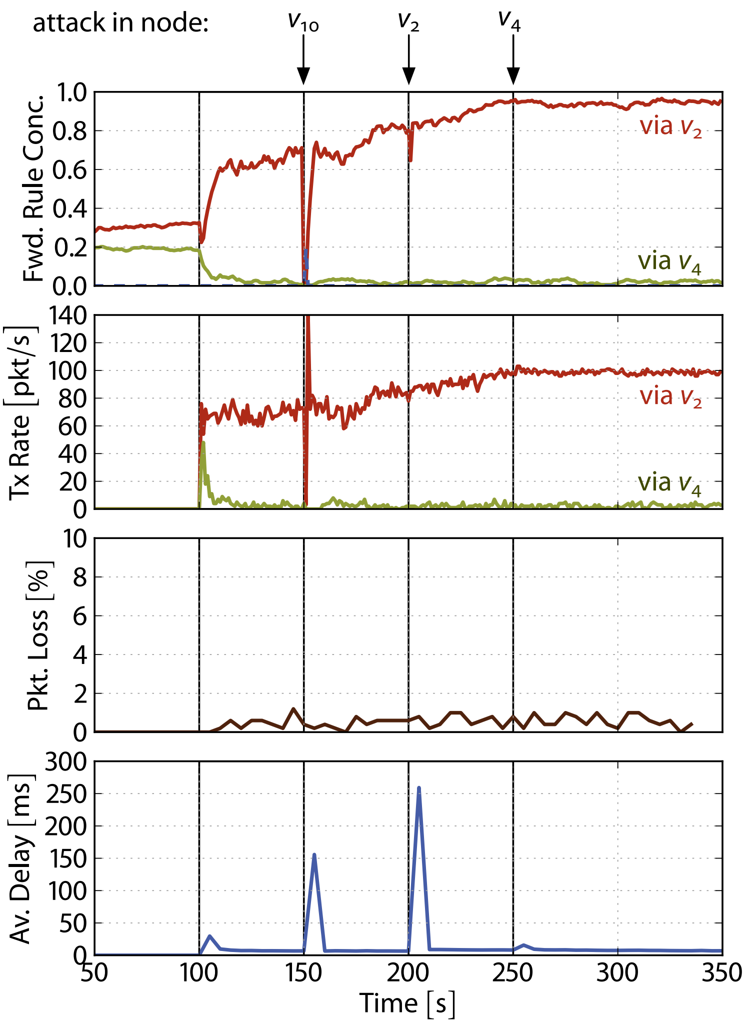


Figure 16.10 Forwarding performance while the program code is deleted: OMNeT++ simulation of the multipath routing protocol. Data packets for destination service z_a are injected to source node v_{10} at rate 100 pkt/s (see topology in Figure 16.1). At time $t = 100$ s, data transmission starts. At time $t = 150$ s, all molecules in node v_{10} are forcefully destroyed. The same attack is carried out to neighbor v_2 at time $t = 200$ s and to neighbor v_4 at time $t = 250$ s. All measurements were collected in node v_{10}: **Fwd. Rule Conc.:** Relative number of forwarding rules to either neighbor v_2 or v_4. **Tx Rate:** Rate of data packets transmitted to either neighbor. **Pkt. Loss:** Percentage of packets lost between source node v_{10} and destination service z_a. **Av. Delay:** Average packet delay from source to destination.

RELATED WORK AND ASSESSMENT 16.7

We compare our protocol to two other biologically inspired routing protocols, *AntNet* (Di Caro & Dorigo, 1998) and ARAS (Leibnitz et al., 2006), and critically assess our protocol in this context.

AntNet

In *AntNet* (Di Caro & Dorigo, 1998) and *AntHocNet* (Di Caro et al., 2005), each node periodically sends forward ants to known destinations. The path of the forward ant is stochastically determined by the established routing table. The forward ant records the number of hops and the latency during its journey. When reaching the destination, they generate a backward ant that travels back to the source node and deposits pheromones in the nodes according to the quality of the path.

ARAS

Leibnitz et al. (2006) assumed that the location of the destination service can be approximated using geographical information and therefore did not need to disseminate location information. ARAS' forwarding engine also probabilistically selects one out of multiple possible next hops. When a packet reaches its destination, the quality of the path is evaluated and this information is sent back along the path to update the routing tables.

reactive

proactive

In both *AntNet* and our protocol, the concentration of chemicals is used to indicate the quality of a path. *AntNet* uses a *reactive* approach where discovery ants are sent out only in response to a data packet. Our protocol, on the other hand, uses a *proactive* approach: There is no feedback from the forwarding engine to the routing table. The nucleus broadcasts unspecific fetch requests to its neighbors to learn about the existence of services in advance. We are aware that this approach does not scale well with the size of the network. ARAS avoids this step by using geographical information to approximately find next hops towards the destination service.

stochastic path selection

no explicit calculations

robustness

All the three algorithms use feedback information from the destination service to update their forwarding logic, which then *stochastically selects a path*. While the update frequency in *AntNet* is determined by the independent rate at which discovery ants are sent, the adaption speed of ARAS and our algorithm depends on the rate of the forwarded traffic. The main difference between our and the other algorithms is that we *do not explicitly calculate* the quality of a certain path. We never store a metric, delay or quality value symbolically in one of the molecules. Instead, as suggested by our chemical design philosophy, the concentration of molecules together with the law of mass action scheduling algorithm leads to the emergent phenomenon of optimal path reinforcement. This mechanism is more *robust* when facing destruction attacks: a deleted entry does not distort the probability of choosing a certain path.

self-healing

The presented protocol is not superior to comparable multipath routing protocols but is able to *heal itself* from the loss of any or all code parts in a network node. One observed problem is that links with shorter delays are not preferred. In addition, there is a small portion of data packets that might be lost by dilution. However, the presented protocol correctly switches to alternative paths to avoid dropped links or nodes and tries to optimize itself for high throughput.

CHAPTER 17

Code Robustness Analysis

On the frequency of self-replicating sets in Fraglets, their robustness subject to memory mutations and execution errors, and on the link between robustness and the instruction set.

> Things do not turn out
> the way you think they will. [17]
>
> *Prey*
> Michael Crichton

So far, we assumed that all execution errors and bit mutations lead to neutral fraglets being unable to react with other parts of the chemical program. That is, we only considered mutations on the molecular level and studied their dynamics. In this chapter, we take a look at mutations on the structural, symbolic level. We show that most of the mutations are indeed harmless, but that one out of ten random symbol alterations lead to an infinite closure, i.e. to an ever-growing set of fraglet strings. We then methodologically search for a Quine that is robust to mutations and execution errors and realize that such ultimate robust replicators most probably do not exist in Fraglets.

This chapter contains a detailed description of the problem of structural robustness. We cannot provide a good solution for this problem yet. But the problem itself deserves a thorough understanding. In Section 17.1 we study the effect of mutations to a chemical program comprised of data processing Quines and develop a scheme to classify the mutants. In Section 17.2, we report on our search for Quines that are intrinsically robust to execution errors. We did not find the ultimate Quine though, but evaluated the robustness of thousands of Quines. This allows us to draw conclusions about the vulnerability of the Fraglets instruction set to mutations and execution errors and the tendency of fraglet strings to suffer from uncontrollable elongation. Section 17.3 then proposes a redundant instruction-encoding scheme to mitigate the problem. In the next chapter, we resort to confining the uncontrolled elongation rather than preventing it.

17.1 EFFECTS OF SYMBOL MUTATIONS

Fraglets is an algorithmic chemistry (see Section 3.1), meaning that the symbolic structure of the molecules is responsible for the outcome of a molecular reaction. Hence, Fraglets is constructive, i.e. capable of continuously producing new molecular species. This is in contrast to many simulations of real chemistry, where the molecular species and their reactions are known beforehand, and where a deterministic approach based on ODEs (Ordinary Differential Equations) is often sufficient. Hence, by altering a single symbol in one of the molecules participating in the reaction network we actually modify the reaction network itself: When mutated molecules react with other molecules, their immediate and future products are likely to be different.

17.1.1 MUTATED PROGRAM

In this section, we study the effect of symbol mutations to three versions of a typical Fraglets program that symbolically computes the arithmetic expression $y = x^3 + 2x + 1$: (1) The first program version uses seven *persistent* `matchp`-*rules*, each computing a partial result:

persistent `matchp`-rules

```
[matchp X dup T1 nop]
[matchp T1 exch T2 3]
[matchp T2 pow T3]
[matchp T3 exch T4]
[matchp T4 mult T5 2]
[matchp T5 sum T6]
[matchp T6 sum Y 1]
```

Figure 17.1 depicts the reaction network of the seven operations while processing the input molecule `[X 5]` and producing the result `[Y 136]`. (2) The second version of the program instantiates 100 copies of the same `matchp`-rules and thus makes the rules *redundant*. (3) The third version uses Quines: We use the "quinification" template introduced in Section 14.4.2 in order to turn each of the seven operations into a Quine. The resulting string of Quines

redundant

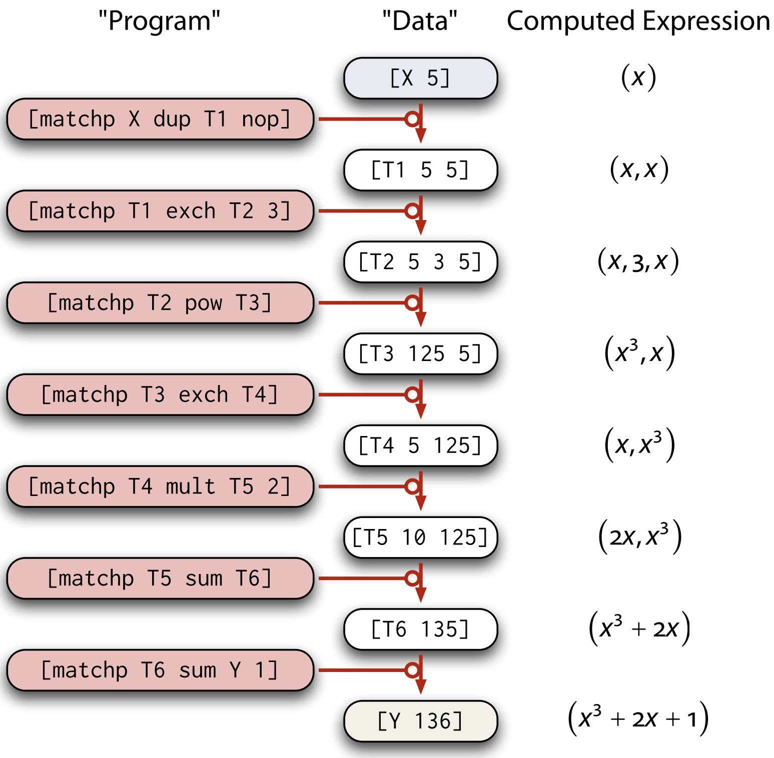


Figure 17.1 Wild-type of the program for a symbol mutation analysis: The program consists of seven consecutive persistent `matchp`-rules, each producing the reactant for the next. The program's wild-type (un-mutated form) computes the arithmetic expression $y = x^3 + 2x + 1$.

cooperates and is collectively *self-healing* as discussed in Section 15.2.1. The capacity of the reaction vessel is limited to 1400 molecules in order to have 100 instances of active and blueprint molecules for each of the seven operations on average.

self-healing

17.1.2 MUTATION PROCEDURE

The *wild-type* (i.e. the original un-mutated form) of the program computes the correct arithmetic expression by design. We simulated the three program versions from above in the Fraglets interpreter for each possible one-symbol mutation. In other words, we took the original programs, mutated a given symbol, initialized the reaction vessel with the mutant and 10^4 input molecules, and waited until the program produced the result molecules.

wild-type

Each symbol locus (position) in the program was mutated $|\Sigma| - 1$ times, where Σ is the considered symbol alphabet. It consists of $|\Sigma| = 99$ symbols: a selection of 33 Fraglets instructions (`match`, `fork`, `nop`, etc.), 33 passive tags such as `X`, `Y`, `T1`, `T2`, and 33 integers numbers ranging from -16 to 16.

The first and second version of the program contains $l_{1,2} = 32$ loci in total, which results in an overall number of

$$n_{\text{mut},1,2} = l_{1,2}\left(|\Sigma| - 1\right) = 3136 \qquad (17.1)$$

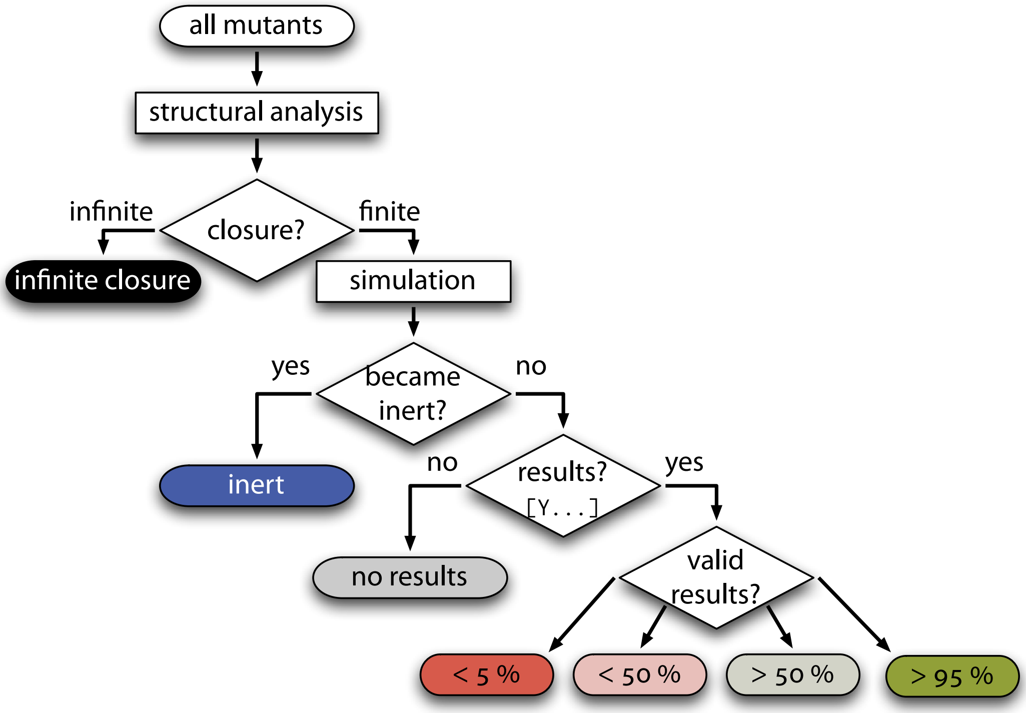


Figure 17.2

Mutant classification scheme: A structural analysis of the reaction network sorts out infinite closures. Simulation runs reveal if a result is produced and if it is correct.

mutants in the one symbol neighborhood of the wild-type. The operations of the third, quinified version contain two copies of each symbol, one in the active `match`-rule and another in the Quine's blueprint. Together with additional symbols needed for replication the total number of loci is $l_3 = 2l_{1,2} + 7 \cdot 19 = 197$ resulting in

$$n_{\text{mut},1,2} = l_3 \left(|\Sigma| - 1\right) = 19306 \tag{17.2}$$

mutants.

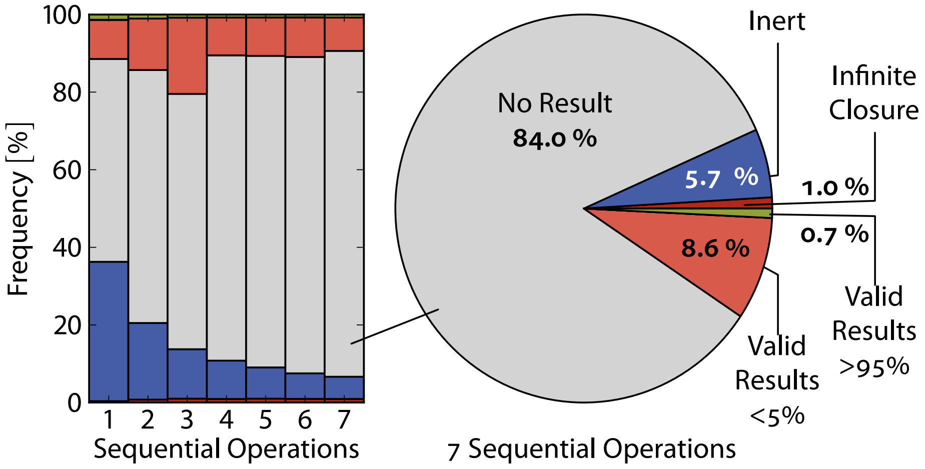
17.1.3 CLASSIFICATION OF THE MUTANTS

We classified each mutant by analyzing it structurally and based on the quality of results produced in a simulation run. Figure 17.2 illustrates the classification scheme we applied: We first analyzed the reaction network of each mutant structurally in order to find out whether the closure is finite or infinite (see Section 7.2). If the closure of a reaction network is infinite it inevitably produces longer and longer strings because the symbol alphabet is finite. We therefore sorted out networks with *infinite closure* as they either become inert or eventually exhaust all CPU and memory resources.

infinite closure

We simulated the remaining mutants until the 10^4 input molecules were processed. If the reaction vessel lost all reactive molecules before the inputs were consumed we classified the mutant as non-reactive or *inert*. For those

inert



Valid Result >*x* %: Percentage of result molecules carrying the correct value. **No Result:** The mutated reaction network does not produce any result molecules. **Inert:** The mutated reaction network is not reactive anymore. **Infinite Closure:** The closure of the mutated reaction network is infinite.

Figure 17.3 **Effect of one-symbol mutations — persistent rules:** The program consists of 1–7 consecutive persistent `matchp`-rules, each producing the reactant for the next. The alphabet consists of a subset of 33 of the Fraglets instructions, 33 passive tags, and 33 integer values from -16 to +16.

mutants that stayed reactive, we examined whether any result molecules of the form `[Y ...]` was generated and sorted out mutants producing *no results*. Finally, we counted how many of the result molecules contained the expected value and further classified the mutants into four quality groups: (1) those where the results was correct for more than 95 % of the input values, (2) those where the result was correct for the majority of the cases, (3) those where the results was incorrect for the majority of the inputs, and finally (4) those that only delivered 5 % of the result molecules with the expected value.

no results

RESULTS 17.1.4

The pie graph in Figure 17.3 illustrates the mutational robustness of the first version of the program, which is based on single `matchp`-rules. About 84 % of the mutants in the one symbol neighborhood of the wild-type are not able to produce a `[Y ...]`-fraglet anymore.

Another 5.7 % of the mutants were inert and only 0.7 % of the mutants maintained their functionality and delivered the correct result. Finally, one percent of the mutations resulted in an infinite reaction network potentially producing longer and longer strings.

The bar graph in Figure 17.3 shows the distribution of the mutant's classification if the program contains less than seven sequential operations. This provides an idea how the robustness depends on the size of a program. According to the observed trend, the percentage of programs delivering no results – either because they are inert (blue) or because the result fraglet is not

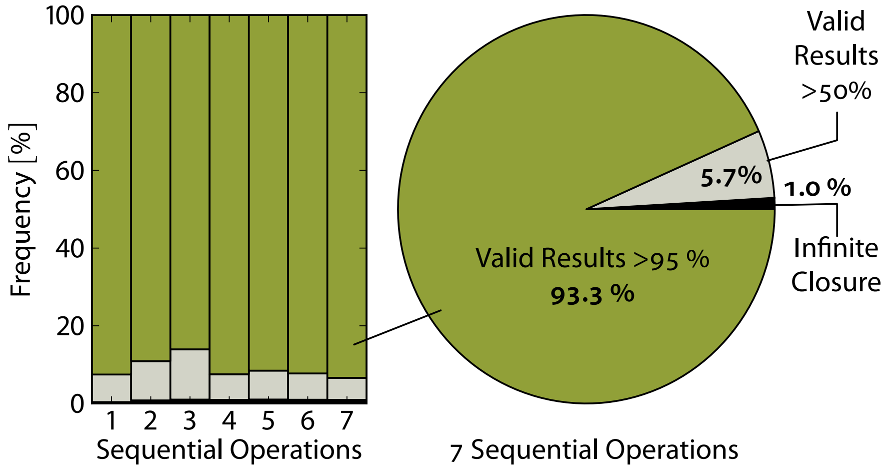


Figure 17.4 Effect of one-symbol mutations — redundant persistent rules: The program consists of 100 copies of 1–7 consecutive persistent `matchp`-rules, each producing the reactant for the next. The alphabet consists of a subset of 33 of the Fraglets instructions, 33 passive tags, and 33 integer values from -16 to +16.

Valid Result >*x* **%:** Percentage of result molecules carrying the correct value. **Infinite Closure:** The closure of the mutated reaction network is infinite.

produced (gray) – remains more or less constant with increasing program size.

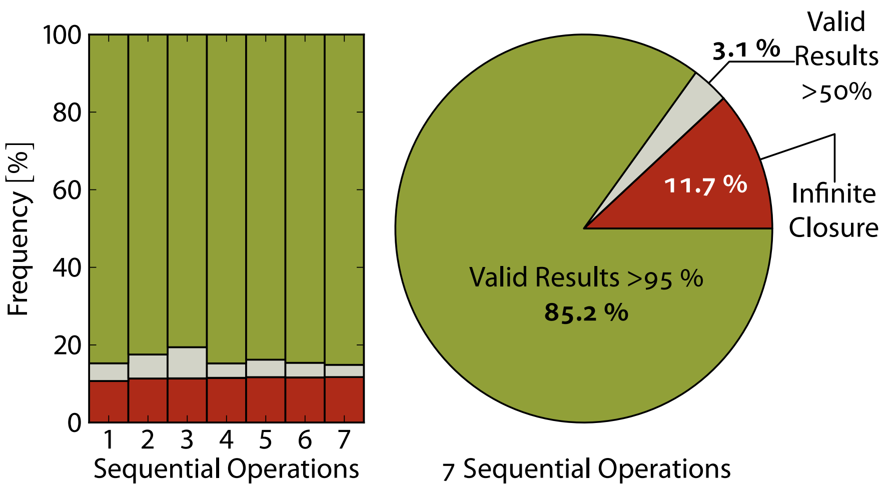
The robustness of the program increases dramatically by having redundant copies of each rule: Figure 17.4 shows the statistics of the second version of our program where each rule is present with 100 copies, of which only one is mutated. Even if a mutated rule is not able to produce any or the right result anymore the other 99 copies are still healthy. However, this does not change the percentage of infinite closures, as a single mutant producing longer strings is sufficient to infect the program. Note that even though redundancy is able to mask the error, the system is not aware of it and does not heal itself. When multiple mutations occur in sequence, the system continuously degrades.

The third version of the program based on Quines is able to recover from mutants producing the wrong result. A single erroneous rule will very likely be displaced by the majority of correct Quine replicas. Figure 17.5 illustrates the mutational robustness for the quinified program version: Due to the redundancy, the Quines exhibit the same behavior of masking singular errors by the majority of healthy replicas. Perhaps surprisingly, compared to the second version, the percentage of infinite closures increases from 1 % to 11.7 %.

(a) *The Quine: A Balancing Act Between Loss and Gain*

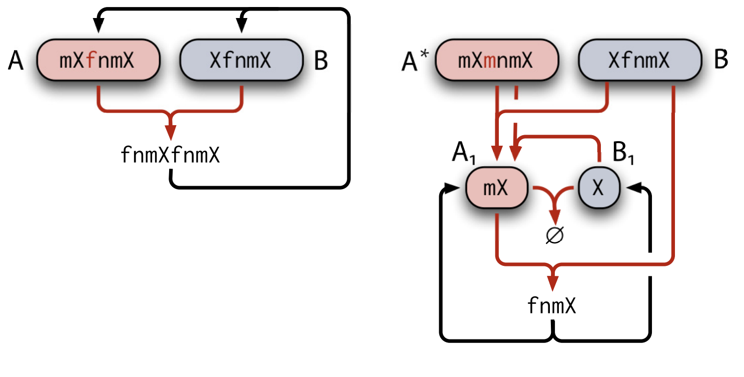
self-replicating spiral

The reason for this common explosion of diversity is the fact that the Quine is a self-replicating *circle*, which produces exactly the same species that reacted in the first place. A deviation from this loop caused by a symbolic perturbation often leads to a *self-replicating spiral* in which the copy products are not identical replicas anymore. Instead, the spiral produces similar molecules



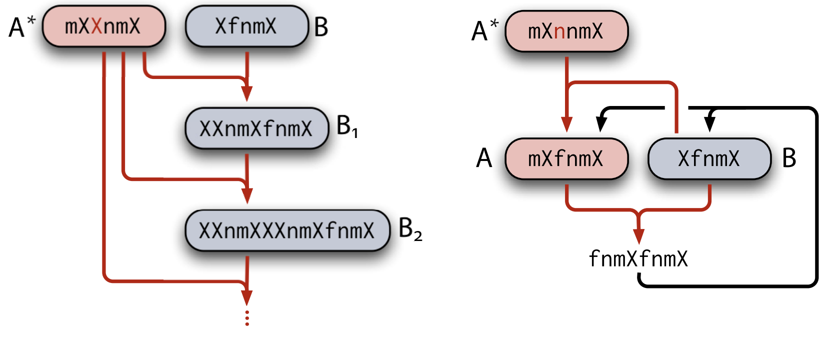
Valid Result >*x* **%:** Percentage of result molecules carrying the correct value.
Infinite Closure: The closure of the mutated reaction network is infinite.

Figure 17.5 Effect of one-symbol mutations — redundant Quines: The program consists of $n = 1$ to $n = 7$ consecutive data processing Quines, each producing the reactant for the next. The capacity of the reaction vessel is limited to 200*n* molecules. The alphabet consists of a subset of 33 of the Fraglets instructions, 33 passive tags, and 33 integer values from -16 to +16.



(a) Wild-type **(b)** Reduction Spiral

Figure 17.6 Structural damage of Quines: The mutation of the same locus of the original replicating Quine **(a)** may lead to a spiral towards shorter fraglets with the same prefix **(b)**.
`m ≡ match` `f ≡ fork` `n ≡ nop`



(c) Elongation Spiral **(d)** Fallback to wild-type

Figure 17.6 cont.: The mutation of the same locus may also lead to a spiral towards longer fraglets with the same prefix **(c)**, or in rare cases, to the identical fraglets **(d)**.
`m ≡ match` `f ≡ fork` `n ≡ nop`

that are shorter or longer than the original reactants; similar in the sense of having the same prefix such that they still react with the same strings.

A typical example is depicted in Figure 17.6. For illustration purpose, we chose the non-replicating Quine that just rewrites itself as the wild-type. We mutated the third symbol of the active fraglet: once from `fork` to `match`, once from `fork` to `X`, and finally from `fork` to `nop`: As shown in Figure 17.6(b), reduction spiral

a mutation from `fork` to `match` leads to a *reduction spiral*, meaning that the products of a reaction among the mutated molecule A^* and the blueprint B only produces a shorter version A_1 of the active molecule.[*] If there are more instances of the two longer species A^* and B they will also react with the new species and again produce shorter versions. Finally, the two short molecules annihilate each other. Note that the reduction happens on the symbolic as well as on the dynamic level: molecules are continuously drained from the long to the short versions where they decay.

elongation spiral

When mutating the third locus of the active fraglet from `fork` to `X`, the loop flips to the other side, producing longer and longer strings. This *elongation spiral* is depicted in Figure 17.6(c). The closure of this reaction network is infinite but not self-maintaining, meaning that the system does not survive for a long time because no active molecules are produced.

fallback to wild-type

Figure 17.6(d) finally shows a case where the mutation eventually leads to the *original wild-type*. The mutated active fraglet A^* reacts with the blueprint and just instantiates an active copy of the blueprint, which has not been mutated. Unfortunately, such harmless mutations are rare.

structural fixed point

This example shows that a healthy Quine balances between growth and death in two respects: First, it is a dynamic fixed point that maintains its population by regenerating the molecule instances that were consumed. Second, the Quine can be seen as a *structural fixed point* that only produces the species it stems from – not more and not less. In the Chemical Organization Theory, the structural fixed point is called a semi organization, i.e. a set that is closed (does not generate new species) and semi self-maintaining (re-generates all consumed species). Dittrich and Speroni di Fenizio (2007) proved that a structural fixed point is a requirement for a dynamic fixed point.

Note that the structural and dynamic fixed points of the Quine are not stable: The simple Quine does not recover when we remove or add molecule instances, and it structurally decomposes into a spiral when subject to symbol mutations. In Section 14.1, we demonstrated that the dynamic fixed point can be stabilized by letting the Quine replicate. The growing number of replicas together with a random dilution flux applied to the vessel leads to a self-regulating population. Ultimately, we would also like to find a stable *structural* fixed point, where a small mutational perturbation automatically leads the reaction system back to the core replicator set. In the following section, we describe our fruitless search for such a structurally stable Quine.

[*] Note that a fraglet starting with a `match`-instruction requires an identifier at the second position, which is not case for the product of A^* and B. In this case Fraglets just ignores all non-identifier symbols such that `[mnmXfnmx]` ≡ `[mXfnmX]`.

THE SEARCH FOR (ROBUST) REPLICATORS 17.2

The Quines presented and analyzed so far were manually designed. In the previous section, we recognized that this self-replicator is not robust to all mutations. In this section, we methodologically search for other replicators in the vast sequence space of Fraglets and determine their robustness to execution errors. We start by elaborating our search method in Section 17.2.1. Then, in Section 17.2.2, we show how likely we find a Quine in a random Fraglets multiset. In Section 17.2.3, we classify the found Quines and measure their robustness to execution errors. Finally, in Section 17.2.4, we highlight the link between structural robustness and the Fraglets instruction set.

SEARCH METHOD 17.2.1

Our aim is to find and assess self-replicating sets of molecules exhibiting a structural and dynamic fixed point. That is, the candidates must fulfill the following two criteria: First, the reaction network spanned by the molecules must be an organization, i.e. closed and self-maintaining according to Dittrich and Speroni di Fenizio (2007). Second, the reaction network must exhibit a non-trivial dynamic fixed point when scheduled according to the law of mass action. The first criterion makes sure that the reaction network is structurally stable. The second criterion requires the first to hold but is stronger, since there are organizations that do not exhibit a non-trivial dynamic fixed point.

The potential search space is infinite: it contains all possible multisets over all fraglet strings. This means that we cannot exhaustively test all possible multisets. We therefore limit our search subspace to single fraglet strings of a certain length and perform exhaustive and random search over this subspace. The restriction to single strings is feasible because every multiset of fraglet strings can be produced by a single *seed* fraglet, i.e. by using `split`- and `fork`-instructions. For example, as depicted in Figure 17.6(a), our simple Quine can be bootstrapped by the seed `[fork nop match X fork nop match X]`.

seed

For each candidate seed we performed a structural and dynamical analysis as depicted in Figure 17.7 in order to find out whether the seed is a Quine or not. We first computed the biggest semi-organization by finding the closure of the seed and removed those species from the closure set that are not semi self-maintaining. If this yields a non-trivial semi-organization we then injected ten instances of the seed string into an empty vessel and simulated 10^4 reaction steps. If the system did not become inert and if the vessel capacity of 1000 molecules was not reached the candidate string is very likely the seed for a Quine.

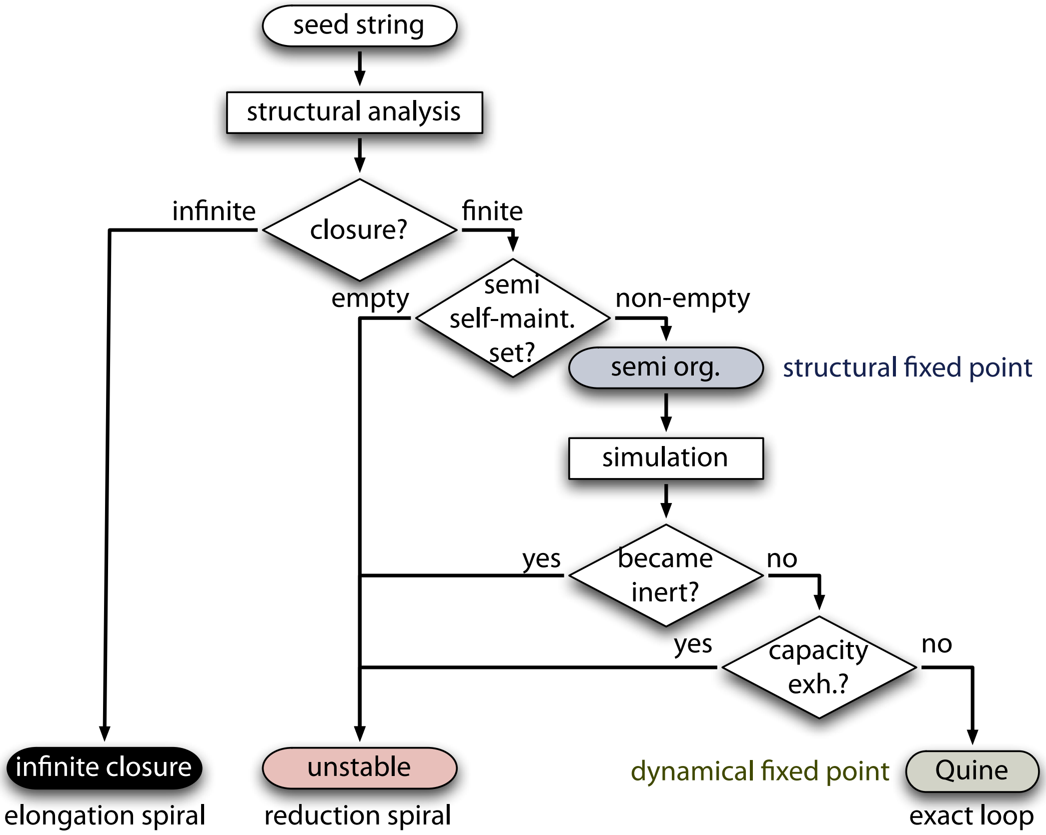


Figure 17.7 Quine detection method: A structural analysis of the reaction network that is spanned by the seed determines if the reaction network contains a structural fixed point; infinite closures and non-semi-self-maintaining sets are sorted out. The subsequent simulation run identifies a dynamic fixed point – a Quine.

17.2.2 THE FREQUENCY OF QUINES

exhaustive search

Before we study the robustness of Quines we illustrate how frequent Quines are in the vast search space. We performed an *exhaustive search* for Quines over all fraglet seeds of length $l = 10$ over the reduced alphabet

$$\Sigma = \{\texttt{X}, \texttt{Y}, \texttt{match}, \texttt{fork}, \texttt{nop}, \texttt{exch}, \texttt{pop}, \texttt{rot}, \texttt{split}, *\} \tag{17.3}$$

The size of the search space is $n = l^{|\Sigma|} = 10^{10}$ seeds. The exhaustive search over this space revealed that the majority of the seeds is not self-maintaining. Only a fraction of 3.63×10^{-6} seeds bootstrap Quines. In contrast, 2088 out of a million seeds lead to infinite closures. Thus, it is more likely that a random fraglet generates a reduction or elongation spiral than a self-maintaining Quine.

random search

Such an exhaustive search is only feasible for very short strings because the search duration grows exponentially with the length. We also explored the search space *randomly* for different seed lengths. A random search is not able to find all Quines for a given seed length, but we can gain statistical insights

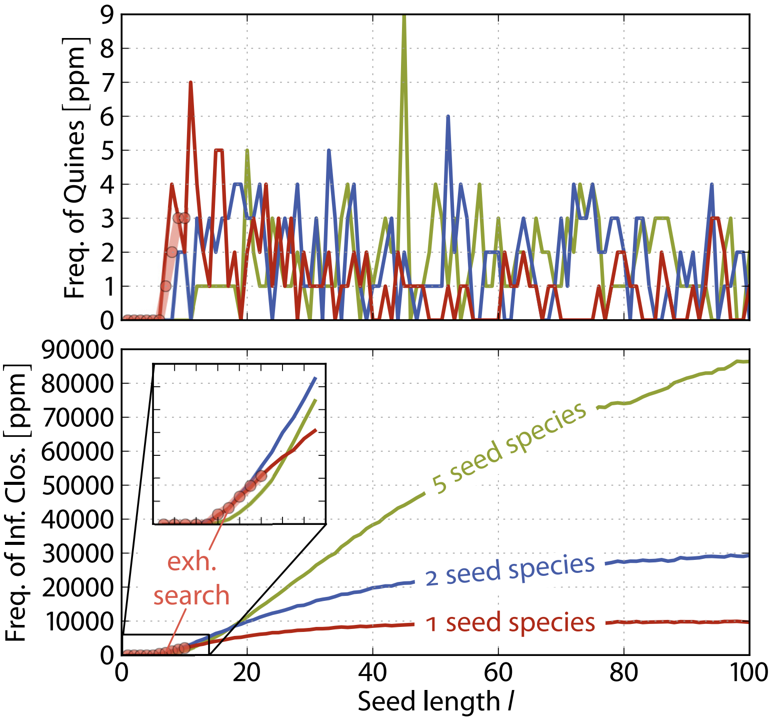


Figure 17.8 Frequency of Quine appearance with respect to the seed length: Fraction of Quines (top) and infinite closures (bottom) in parts per million (ppm). For each length in the range from 1 to 100, 10^7 random seeds were tested. We additionally tested the seeds when randomly split into 2 or 5 parts. The small box shows a magnification of the lower lengths where we complement the results of the random search with reference values from the exhaustive search to show that the random search yields representative results.

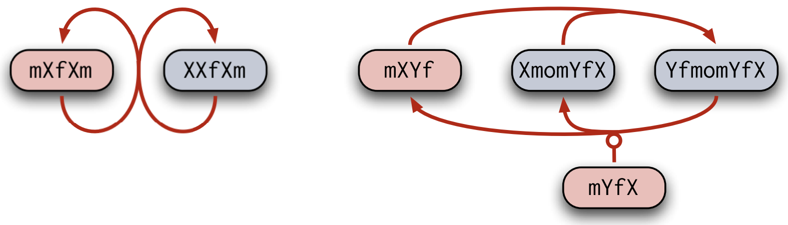
such as how the frequency of Quines changes with respect to the length of the seed. We generated 10^7 random seeds for each length in the range $l = [1, 100]$ and additionally split them into 1, 2, or 5 fraglets.

Figure 17.8 shows the frequency of Quines and infinite closures with respect to the initial number of symbols. For seed lengths up to 10, we complemented the random search results with the exact results from the exhaustive search. The frequency of Quines does not increase above a few per million if we either increase the length or the number of seed splitters. On the other hand, the frequency of infinite closures grows for more complex seeds. This is a strong indication that the Fraglets instruction set has a *tendency to elongate*. We will discuss this in more detail later.

tendency to elongate

Classification of Quines and Assessment of Their Robustness to Execution Errors 17.2.3

As planned originally, we now search for Quines that are intrinsically robust to execution errors. That is, their structural fixed point shall be robust if the underlying virtual machine sporadically executes wrong instructions. We performed a random search over the seed space of length $l = 15$, now with an alphabet extended to 14 symbols (see Appendix B for details on the instructions):



(a) Function 1 (92.8 %)

(b) Function 2 (1.4 %)

Figure 17.9 Most frequent Quine functions: Most (92.8 %) Quines span a reaction network of type 1 **(a)** followed (1.4 %) by the reaction network of type 2 **(b)**.
`m` ≡ `match` `f` ≡ `fork` `o` ≡ `spop`

$$\Sigma = \{\texttt{X}, \texttt{Y}, \texttt{Z}, \texttt{match}, \texttt{nop}, \texttt{exch}, \texttt{spop}, \texttt{spush}, \texttt{rev}, \texttt{cross}, \\ \texttt{fork}, \texttt{splitat}, \texttt{divide}, \texttt{shuffle}\} \tag{17.4}$$

execution errors

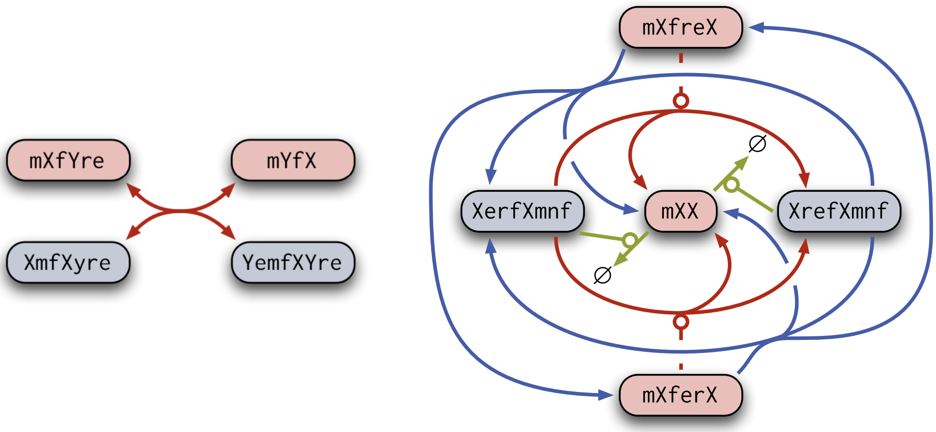
Unlike before, where we mutated the fraglets, we now simulate *execution errors*: That is, in each simulation run we assume that the virtual machine sometimes misinterprets a certain instruction symbol s_{ori} as another symbol s_{err}, which is executed in turn. Hence, for each Quine we performed $|\Sigma|^2$ simulations for all possible symbol pairs $(s_{\text{ori}}, s_{\text{err}})$ and again analyzed all systems structurally and dynamically. We declare a Quine robust to a certain execution error $(s_{\text{ori}}, s_{\text{err}})$ if the resulting reaction network is still a Quine. Note that we do not require that the reaction network subject to execution errors is identical to the reaction network executed on an ideal virtual machine, but it still has to be a self-replicating set. Ultimately, we are looking for Quines that replicate independent of symbol misinterpretations of the virtual machine.

(a) Classification of Quines Into Functional Groups

homomorphic graph

We tested 120×10^9 random seeds of length $l = 15$ on the ideal virtual machine and found 48376 Quines. The density of Quines in this search space can thus be approximated to 0.401×10^{-6}, lower than before due to the extended alphabet. We grouped together all Quines that are functionally identical. That is, if the reaction network graphs of two Quines are *homomorphic* (Hell & Nešetřil, 2004) we say that they perform the same *function* and assign them to the same functional group. We found 164 different groups and ranked them according to their relative frequency. Figure 17.9 depicts four functional groups.

The majority of the Quines (93 %) span a reaction network graph of the first type, which is at the same time the simplest reaction network. Our manually designed Quine is of this type, too, but there are other fraglet strings that generate the same reaction network. For example, the fraglets shown



(c) Function 3 (0.5 %)

(d) Function 151 (once)

Figure 17.9 cont.: The third most frequent reaction network (0.5 %) is of type 3 (c). Only one Quine spanned the interesting reaction network shown in (d). `e ≡ exch n ≡ nop r ≡ rot`

in Figure 17.9(a) are different yet shorter than the strings of the manually designed Quine.

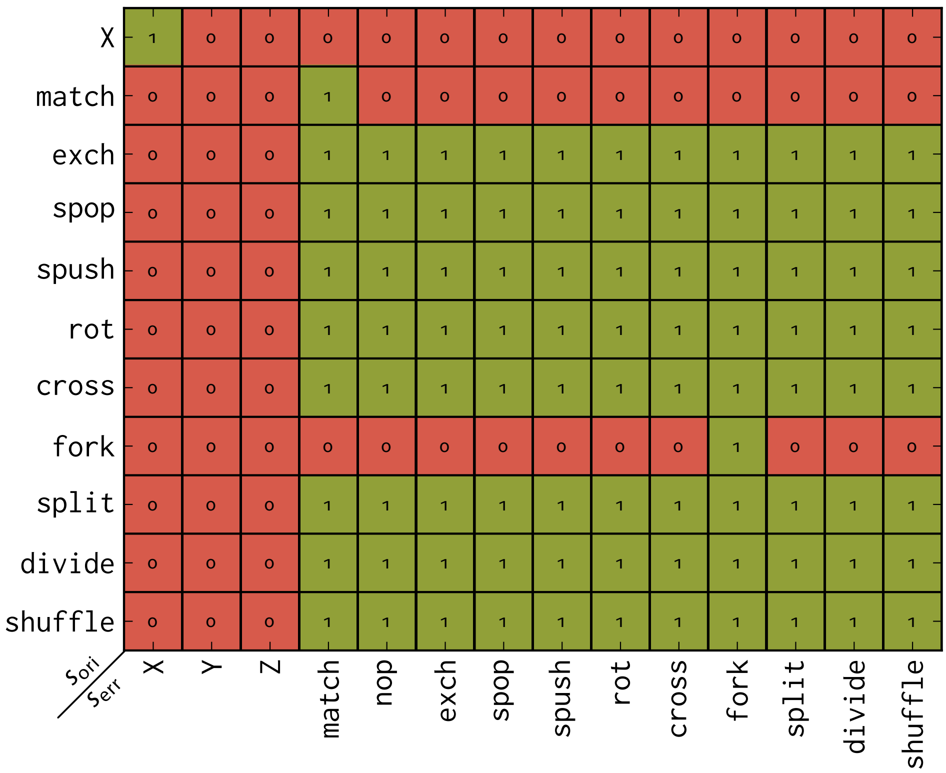
The second and third most common function, shown in Figure 17.9(b) and (c) respectively, are more complex as four species are involved. The latter reaction network actually implements a reversible reaction for which two tags `X` and `Y` are required. We cannot print the reaction networks of all functions here but want to highlight a rare but interesting function: The reaction network shown in Figure 17.9(d) involves five different species, although only one tag, `X`, is involved. That is, all active fraglet species (red) react with all blueprint species (blue). At equilibrium, the central species `[match X X]` is present with double the quantity of the other species. As we will see below, a more complex reaction network does not mean that the Quine is more robust to execution errors.

Quantification of the Quines' Robustness to Execution Errors *(b)*

We define the *robustness to execution errors* q_{exe} of a Quine as the number of tuples (s_{ori}, s_{err}) yielding another Quine divided by the total number of essential misinterpretation tuples. Trivial cases such as (`match`, `match`) are not considered.

robustness to execution errors

We simulated all of the 48376 Quines that we found in an ideal environment, now on a virtual machine with execution errors as described before. We did not find the ultimate Quine that is robust to all instruction glitches, i.e. where $q_{err} = 1$. Figure 17.10 depicts the robustness matrix of the most robust Quine found. It is seeded by the fraglet



s_{ori} / s_{err}	X	Y	Z	match	nop	exch	spop	spush	rot	cross	fork	split	divide	shuffle
X	1	0	0	0	0	0	0	0	0	0	0	0	0	0
match	0	0	0	1	0	0	0	0	0	0	0	0	0	0
exch	0	0	0	1	1	1	1	1	1	1	1	1	1	1
spop	0	0	0	1	1	1	1	1	1	1	1	1	1	1
spush	0	0	0	1	1	1	1	1	1	1	1	1	1	1
rot	0	0	0	1	1	1	1	1	1	1	1	1	1	1
cross	0	0	0	1	1	1	1	1	1	1	1	1	1	1
fork	0	0	0	0	0	0	0	0	0	0	1	0	0	0
split	0	0	0	1	1	1	1	1	1	1	1	1	1	1
divide	0	0	0	1	1	1	1	1	1	1	1	1	1	1
shuffle	0	0	0	1	1	1	1	1	1	1	1	1	1	1

Figure 17.10 Robustness to execution errors of the most robust Quine found: A value of one (green) in this matrix indicates that the Quine is robust to an execution error where instruction s_{ori} is misinterpreted by the virtual machine as instruction s_{err}. Even the most robust Quine does not tolerate misinterpretations of the instructions `match`, `fork` and tags.

```
[fork match X cross exch spop spush divide
          split rot shuffle X fork match Y]
```

and exhibits a robustess to execution errors of $q_{exe} = 80/143 = 0.56$.

The reaction network spanned by this seed implements function 1 (see Figure 17.9(a)). Our manually designed Quine, which has the same reaction network graph, exhibits a robustness of $q_{exe} = 0$. This highlights that the robustness to execution errors is not tied to the macroscopic behavior of the reaction network but is rather linked to the way the reaction network is generated microscopically. By a clever arrangement of symbol sequences, we could in theory improve the robustness of chemical programs. However currently, there is no method of how to exploit this observation in order to build more robust programs.

The robustness of Quines obviously depends on the instruction set. Even though the most robust Quine is robust to most execution errors, it is brittle to the misinterpretation of the instructions `match`, `fork` and to tag glitches. In the following, we examine the influence of the instruction set to the robustness in more detail.

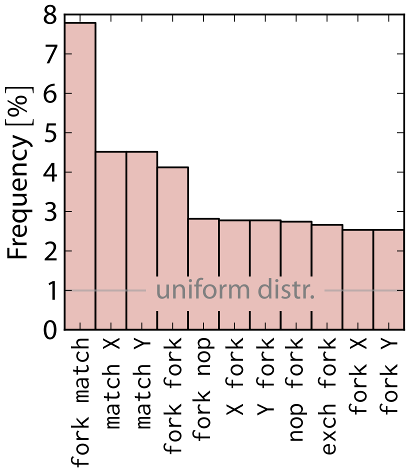


Figure 17.11 Symbol sequence histogram as found in Quines: Frequency of consecutive symbol sequences in the closure of the Quines found by an exhaustive search over seed length $l = 10$ with the alphabet (17.3). The 1 % line illustrates the expected frequency when all sequences would be distributed uniformly.

IMPACT OF THE FRAGLETS INSTRUCTION SET TO THE ROBUSTNESS OF QUINES 17.2.4

We further processed the information collected from the analysis of the 48376 Quines in order to draw statistical conclusions regarding the dependency between the robustness to execution errors and the Fraglets instruction set.

essential instructions

The instructions `fork` and `match` are *essential instructions* for building Quines: without them or similar instructions, it is virtually impossible to build self-replicating loops in Fraglets. We can demonstrate this with two statistical evaluations:

First, independent of which instruction set we used, about 25 % of the symbols found in Quines are `fork`-instructions, followed by `match` and tags. Figure 17.11 even shows that some sequence snippets including these instructions are very likely to be observed in the species that span the Quines' reaction network.

Second, we examined the minimal alphabet needed for realizing the 164 functional Quine groups. Figure 17.12 shows which instructions are required for how many functions: All the Quines we found made use of the `fork`-instruction, the `match` synchronization symbol and at least one tag.

We could have expected such a result: As mentioned in Section 5.2.4, we intentionally force all transformation instructions to reduce the length of the fraglet string in order to avoid infinite transformation loops. This comes at the cost that a self-replicator must join two molecules in order to compensate for the instructions that were consumed during execution. The only way to elongate a molecule is to combine it with another molecule via a synchronization instruction of the `match` family, which also requires tags. Then the Quine must eventually use an instruction that splits-up the joined molecule into pieces in order reproduce the original reactants. The `fork`

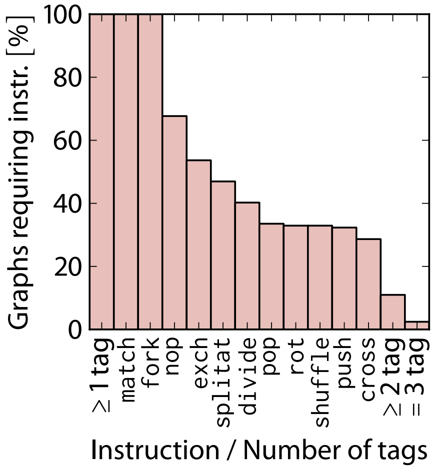


Figure 17.12 Instructions required for the diversity of graph types: Fraction of graph types requiring a certain instruction. None of the found Quines would exist without the instructions `match`, `fork`, and at least one tag `X`, `Y`, or `Z`. Graphs requiring 2 or more tags are very seldom.

instruction seems to be more powerful than the `split` reaction. Thus, a Quine has to perpetually execute alternating join and split instructions in order to be structurally stable: A set of molecules only regenerates itself if *join and split operations are well balanced* and used in coordination.

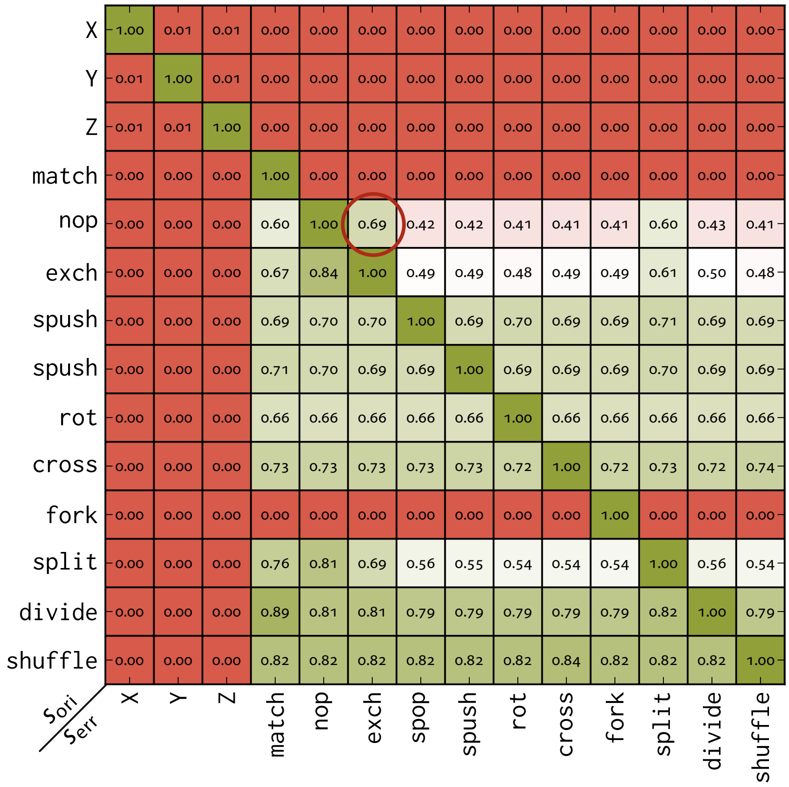
balance of join and split operations

Unfortunately, those essential instructions are the ones most vulnerable to execution errors. Figure 17.13 statistically combines the execution error matrices of all Quines examined. The matrix shows the instruction dependent robustness of Quines to execution errors. Each element of this matrix represents a misinterpretation of symbol $s_{\mathtt{ori}}$ as symbol $s_{\mathtt{err}}$. The value in the corresponding matrix element denotes the fraction of Quines that were still Quines when executed subject to the given execution error. For example, the value highlighted by the circle indicates that 69 % of all mutants of the 48376 Quines where a `nop`-instruction was interpreted as an `exch`-instruction still replicated themselves.

The matrix demonstrates that the instructions `fork`, `match`, and the tags are very vulnerable to execution errors. But these are exactly the instructions needed for self-replication. Thus we conjecture that with the current Fraglets instruction set, there is most likely no ultimate Quine that is robust to all execution errors.

17.3 INDIRECT INSTRUCTION ENCODING

One possible way of mitigating the effect of bit mutations (but not execution errors) is to provide redundancy already on the bit level. So far, we assumed that each memory alteration leads to a symbol mutation. Instead, we can resort to an indirect and redundant encoding of symbols that masks bit mutations.



s_{ori} / s_{err}	X	Y	Z	match	nop	exch	spop	spush	rot	cross	fork	split	divide	shuffle
X	1.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Y	0.01	1.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Z	0.01	0.01	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
match	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
nop	0.00	0.00	0.00	0.60	1.00	0.69	0.42	0.42	0.41	0.41	0.41	0.60	0.43	0.41
exch	0.00	0.00	0.00	0.67	0.84	1.00	0.49	0.49	0.48	0.49	0.49	0.61	0.50	0.48
spush	0.00	0.00	0.00	0.69	0.70	0.70	1.00	0.69	0.70	0.69	0.69	0.71	0.69	0.69
spush	0.00	0.00	0.00	0.71	0.70	0.69	0.69	1.00	0.69	0.69	0.69	0.70	0.69	0.69
rot	0.00	0.00	0.00	0.66	0.66	0.66	0.66	0.66	1.00	0.66	0.66	0.66	0.66	0.66
cross	0.00	0.00	0.00	0.73	0.73	0.73	0.73	0.73	0.72	1.00	0.72	0.73	0.72	0.74
fork	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
split	0.00	0.00	0.00	0.76	0.81	0.69	0.56	0.55	0.54	0.54	0.54	1.00	0.56	0.54
divide	0.00	0.00	0.00	0.89	0.81	0.81	0.79	0.79	0.79	0.79	0.79	0.82	1.00	0.79
shuffle	0.00	0.00	0.00	0.82	0.82	0.82	0.82	0.82	0.82	0.84	0.82	0.82	0.82	1.00

Figure 17.13 Instruction dependent robustness of Quines to mutations: The values in this matrix represent the fraction of Quines that still regenerate themselves when symbol s_{ori} is misinterpreted as symbol s_{orr} by the virtual machine. The matrix identifies the instructions `fork`, `match`, and the tags `X`, `Y`, and `Z` as vulnerable symbols.

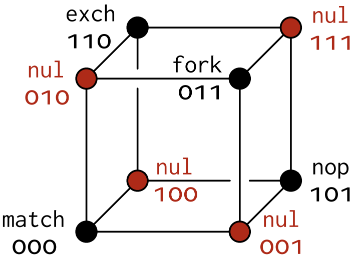


Figure 17.14 Parity-bit encoding scheme for Fraglets instructions: Bit strings with an odd number of ones are mapped to the `nul`-instruction. Any one-mutant neighbor of an even-coded instruction is `nul`, which causes the fraglet to commit suicide.

Nature makes use of a similar redundancy mechanism: The genetic code maps $4^3 = 64$ nucleotide triples to only 20 amino acids. Some one-nucleotide mutations are masked to prevent, for example, copying errors during DNA replication (Griffiths, Miller, Suzuki, Lewontin, & Gelbart, 2000, Chap. 17). In computer science, error-detecting codes are used to protect memory content from bit mutations caused by cosmic radiation (Slayman, 2005) or voltage instability. The same mechanisms are used to protect information exchanged over unreliable communication channels (Tanenbaum, 2002, Sect. 3.2).

parity bit

Here, we discuss a very simple *parity bit* protection mechanism for fraglet strings. Let us assume that each Fraglets symbol is encoded by a string of $n-1$ bits. We introduce an n-th parity bit and arrange the coding scheme such that bit strings with an odd number of ones are mapped to the `nul`-instruction. Figure 17.14 shows an example for $n=3$. With such an encoding scheme, the one-mutant bit neighbors of all symbols are interpreted as `nul`-

instruction, which leads to the self-destruction of the fraglet as soon as this instruction is executed. Since the software continuously rewrites itself when based on Quines, each part of the software will eventually verify itself by being executed.

map code mutation to code destruction

With the parity-encoding scheme we actually *map code mutation to code destruction*. In the traditional sequentially executed program, we cannot simply throw away or ignore code like this. However, in the chemical model, and because we are able to construct software based on self-healing Quines, our code is resilient to code deletion. Thus, we do not need an error correction mechanism; a simple error detection mechanism is sufficient.

Mutations of instructions can be effectively contained by such a parity-encoding scheme. Other symbols, such as numbers or payload, may be altered unrecognized by the system. However, there is a high probability that such mutations are of transient nature and do not affect the long-term health of the system. We envision that the instruction robustness matrix in Figure 17.13 could further influence the development of more efficient bit encoding schemes in the future.

17.4 SUMMARY AND OUTLOOK

There are several reasons why we should care about the mutational robustness and the robustness to execution errors of distributed artificial chemistries: First, not only data packets but active code is transmitted over unreliable links, which leads to the alteration of code if the molecules are not protected by an error detection mechanism. Second, unreliable hardware or external disturbances such as cosmic radiation may result in the mutation of local code. Third, if the underlying execution machinery (i.e. CPU) is unreliable, a bit-level redundancy encoding scheme does not even help, because correctly encoded instructions may be misinterpreted by the CPU after being decoded.

We have two options to continue our research to increase the robustness to execution errors: First, we could continue the search for robust instructions, which enable Quines exhibiting a structural fixed point. Each wrongly executed product would lead to a side-product that eventually converges back to the replicator set. As we demonstrated, it is implausible that the current Fraglets instruction set bears such replicators.

probabilistic instruction set

A second approach for the future is to develop a *probabilistic instruction set*, in which a symbol represents a probability distribution over instructions. In this regime, a Quine can be considered as a gas of molecules reacting in a cloud around the expected cyclic execution trajectory. However, such a probabilistic instruction set will most likely complicate the design process of human-engineered networking protocols and is more suitable as an approach for evolved protocols.

One problem we observed is that mutations and execution errors cause the uncontrollable elongation of fraglet strings. The *elongation catastrophe* phenomenon was reported recently by Decraene, Mitchell, and McMullin (2008) in a string-based chemistry based on Holland's broadcast language (Holland, 1992; Decraene, 2006; Decraene, Mitchell, McMullin, & Kelly, 2007): When the strings are placed in a reactor subject to random mutations and dilution, fitter molecules emerge which are also longer and bind with a higher specificity. Reactions then progressively slow down until they stop, and the system is considered dead.

elongation catastrophe

As demonstrated in this chapter, we also observe a similar phenomenon in Fraglets. The question is whether we can use the elongation catastrophe as a signal that something went wrong and whether the chemical system could intrinsically react and correct the phenomenon. In the next chapter, we show how to confine rather than to prevent such elongation by introducing a notion of energy to the chemical reaction algorithm. The competition among reaction vessels for energy then automatically sorts out spiraling Quines – again without the need of a central observer.

CHAPTER 18

Structural Robustness by Energy Conservation

Introduction of a modified reaction scheduler, which conserves virtual energy to prevent mutated code from consuming all resources.

> It is important to realize that in physics today, we have no knowledge what energy is.[18]
>
> *The Feynman Lectures on Physics*
> Richard Feynman

Our experiments in the previous chapter revealed a strong tendency of Fraglets soups to suffer from the elongation catastrophe. Random symbol mutations also resulted in reaction networks that produce ever-growing strings. In this last chapter of the third part, we propose an extension of the law of mass action scheduling algorithm that is able to confine unbounded elongation.

energy

The new scheduling algorithm introduces the notion of *energy* to an artificial chemistry. The basic idea is simple: Each virtual molecule has a certain potential energy, which is proportional to its mass. The mass of a fraglet string is defined as its length in number of symbols. A reaction that elongates a fraglet cannot "create" the additional energy, but has to take it from a pool of kinetic energy, which is part of the reaction vessel. The total energy is constant. That is, the total number of symbols in a Fraglets reactor is finite. Reactions convert free kinetic into bounded potential energy and vice-versa, depending on whether fraglet strings become longer or shorter.

Such an energy conservation mechanism makes sure that the molecules do not grow unbounded. However, as we observed in our experiments, a random soup or mutated program still has the tendency to grow and yield a small number of very long strings. Such a system usually terminates in a single string, which is not able to react and replicate anymore – the system became inert. In order to avoid this concentration of mass in a single string we have to carefully design the reaction algorithm to economically convert free kinetic energy into potential energy (symbols). We therefore mimic the energy signature of real chemical reactions. As we will show, our resulting extended Fraglets reaction algorithm is able to prevent the elongation catastrophe.

Even though elongation is now under control, mutated programs may still be error prone, resulting, for example, in all available energy being spent for the production of useless garbage molecules. We therefore let multiple reaction vessels (or cells) run the same program in parallel. The cells grow and divide akin to natural cells. Random selection is not only applied to molecules within the cells, but the environment also maintains a constant population of cells, which naturally fight for not being displaced by other growing and dividing cells. In other words, we put the system in an evolutionary context. We show that such a *multi-level selection* scheme is robust to erroneous mutations in one of the cells.

multi-level selection

This chapter is organized as follows: In Section 18.1, we provide background information about energy conservation in chemical reactions and derive a simplified model that operates on abstract molecules. Section 18.2 then presents our generic energy framework for artificial chemistries that ensures energy conservation. In Section 18.3, we demonstrate how the energy framework can be applied to Fraglets in order to prevent the string elongation problem. In Section 18.4, we present a multi-level selection method and discuss initial evolution experiments, which show that a chemical program is robust to mutations in this new regime and even evolves to more efficient cooperating clusters. Finally, after discussing out approach in Section 18.5, we summarize this chapter and the third part of the thesis in Section 18.6.

18.1 THE ROLE OF ENERGY IN CHEMICAL REACTIONS

In this section, we review the dynamic behavior of a chemical reaction and focus on the role of energy. Figure 18.1 depicts the energy diagram of a typical microscopic reaction event, as it proceeds from reactants (left side) to products (right side). The total energy is conserved (1st law of thermodynamics),

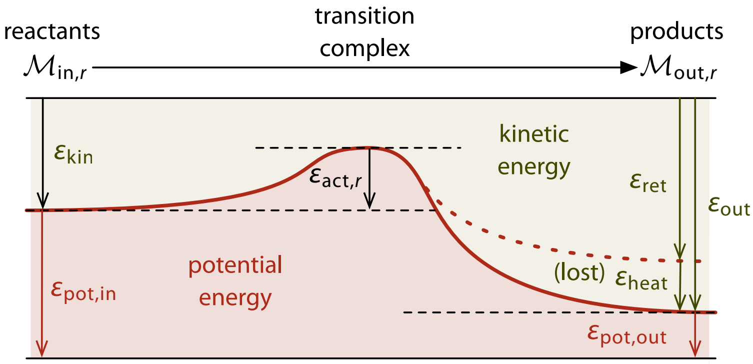

reactants
$\mathcal{M}_{\text{in},r}$
transition complex
products
$\mathcal{M}_{\text{out},r}$
ε_{kin}
$\varepsilon_{\text{act},r}$
kinetic energy
ε_{ret}
ε_{out}
$\varepsilon_{\text{pot,in}}$
potential energy
(lost) $\varepsilon_{\text{heat}}$
$\varepsilon_{\text{pot,out}}$

Figure 18.1 Energy diagram of an exothermic reaction: The reaction $r \in \mathcal{R}$ converts reactants (left side) given by the multiset $\mathcal{M}_{\text{in}}$ into products (right side) given by $\mathcal{M}_{\text{out}}$. The total energy is conserved; the collision energy ε_{kin} has to overcome the activation energy $\varepsilon_{\text{act},r}$. Some kinetic energy (heat) may be lost for further reactions: $\varepsilon_{\text{heat}}$. The remaining kinetic energy is returned to the vessel: ε_{ret}.

meaning that the sum of potential and kinetic energy of the reactants is equal to the total energy of the products. The collision energy ε_{kin} has to overcome the activation energy ε_{act}. In this example, the reaction is exothermic: the potential energy, i.e. the binding energy between atoms of the products, is lower than that of the reactants. The remaining energy returns to the vessel as kinetic energy, but part of it may be dissipated as heat ($\varepsilon_{\text{heat}}$), contributing to the entropy of the system (2^{nd} law of thermodynamics).

This exothermic reaction converts one part of the potential energy that is bound in the molecules into kinetic energy. But the reaction only occurs if enough kinetic energy is available, i.e. in macroscopic terms, if the temperature is high enough. In the following, we study this energy-dependent reaction rate on the macroscopic scale before we make the connection to the microscopic scale. The transition to the microscopic, molecular level is necessary in order to derive a reaction algorithm for our algorithmic chemistry.

MACROSCOPIC SCALE 18.1.1

As shown in Chapter 3, the law of mass action states that a chemical reaction such as $\mathrm{X} + \mathrm{Y} \xrightarrow{k} \mathrm{Z}$ happens with an average speed of $r = k\,[\mathrm{X}]\,[\mathrm{Y}]$, where k is the kinetic coefficient associated with the reaction, and $[\mathrm{X}]$, $[\mathrm{Y}]$ are the concentrations of reactants X and Y, respectively. The coefficient k is usually given by the *Arrhenius equation* (Arrhenius, 1884; McQuarrie, 1997):

Arrhenius equation

$$k = A \exp\left(-\frac{E_{\text{act}}}{RT}\right) \tag{18.1}$$

where A is the pre-exponential factor for the reaction, E_{act} is the activation energy barrier that must be overcome for the reaction to occur, R is the gas constant ($R = k_{\text{B}} N_{\text{A}}$, where k_{B} is the Boltzmann constant, and N_{A} is the Avogadro constant), and T is the absolute temperature. Such knowledge from textbook physical chemistry is a macroscopic description of the average behavior of large quantities of molecules. In an algorithmic chemistry however, it is necessary to simulate the microscopic behavior of the system at the level of individual molecular collisions in order to perform the intended computations encoded within such molecules. Therefore we use results from statistical mechanics for bimolecular gas phase reactions, which permit to derive the Arrhenius equation from microscopic laws (Upadhyay, 2006).

18.1.2 MICROSCOPIC SCALE

First of all, a change in scale must happen, resulting in a microscopic kinetic coefficient $k' = k/(N_A V)$ and a microscopic reaction rate of $r' = k' N_X N_Y = r N_A V$ for a bimolecular* reaction involving two reactants X and Y, when there are N_X instances of species X and N_Y instances of species Y inside volume V, i.e. $N_X = [X]\, N_A V$ and $N_Y = [Y]\, N_A V$. The activation energy must also be rescaled to the microscopic level by taking ε_{act} as the activation energy per reaction, given by $\varepsilon_{act} = E_{act}/N_A$. Note that this does not change the exponential term of the Arrhenius equation, since

$$\exp\left(-\frac{E_{act}}{RT}\right) = \exp\left(-\frac{\varepsilon_{act}}{k_B T}\right) \tag{18.2}$$

microscopic reaction rate

The equation for the *microscopic reaction rate* then becomes:

$$r' = \underbrace{\overbrace{\frac{A}{N_A V}}^{F_{const}} \cdot \overbrace{\exp\left(-\frac{\varepsilon_{act}}{k_B T}\right)}^{\mathbb{P}[\text{reaction}|\text{collision}]}}_{\text{microscopic reaction coefficient } k'} \cdot \overbrace{N_A N_B}^{F_{col}} \tag{18.3}$$

collision frequency

This equation is the product of three terms: (1) The *collision frequency* $F_{col} = N_A N_B$ determines how often a collision occurs. The collision frequency is proportional to the multiplicity of each reactant and hence reflects the familiar law of mass action. (2) A constant factor $F_{const} = A/N_A V$ captures the physical details such as the mass of the molecules or the orientation in which they have to collide. We will abstract away from this factor for our artificial chemical algorithm and set it to $F_{const} = 1$. (3) The third factor is the exponential term attributed to Arrhenius. It actually expresses the conditional *probability that a reaction is effective* (i.e. happens) provided that the collision occurred:

reaction probability

$$\mathbb{P}\left[\text{reaction} \mid \text{collision}\right] = \exp\left(-\frac{\varepsilon_{act}}{k_B T}\right) \tag{18.4}$$

* A similar treatment applies for reactions other than bimolecular; see Gillespie (1977) for details.

It is important to note that we separated the reaction rate into three factors, one that contains all the physical details, a second that reflects the law of mass action, and a third that captures the energy-dependence of the reaction. The product of the constant factor and the energy-dependent exponential term is the *microscopic reaction coefficient* k'.

microscopic reaction coefficient

DERIVATION OF THE ENERGY-DEPENDENT REACTION PROBABILITY 18.1.3

Let us now focus on the energy term, which is given by the reaction probability $\mathbb{P}\left[\text{reaction} \mid \text{collision}\right]$. We would like to determine the "algorithmic procedure" that a single reaction has to perform such that the reaction probability in (18.4) emerges on the macroscopic scale.

According to the collision theory, the kinetic energy of two colliding molecules follows a chi-square distribution with three degrees of freedom resulting from the Maxwell-Boltzmann distributed velocity components of the molecules. However, the full kinetic energy can only be invested into the reaction if the molecules come in touch by a frontal collision. In general, only a fraction of the kinetic energy – the energy along the line of the molecule's centers – is actually exploitable by a reaction. This effective collision energy ε_{kin} is stochastically described by an exponential distribution with mean $k_\text{B}T$ (Upadhyay, 2006).

$$\varepsilon_{\text{kin}} \sim \text{Exp}(k_\text{B}T) \tag{18.5}$$

We can regard a collision event as the following algorithmic procedure: Two molecules collide with an effective energy of ε_{kin}, distributed according to (18.5). If the effective collision energy is greater than the activation energy of the reaction ($\varepsilon_{\text{kin}} \geq \varepsilon_{\text{act}}$) then the reaction happens. Otherwise, the collision is elastic and the reaction is not carried out. Thus, according to this simple algorithm, the probability that a reaction is effective is given as

$$\begin{aligned}\mathbb{P}\left[\text{reaction} \mid \text{collision}\right] &= \mathbb{P}\left[\varepsilon_{\text{kin}} > \varepsilon_{\text{act}}\right] = 1 - \mathbb{P}\left[\varepsilon_{\text{kin}} \leq \varepsilon_{\text{act}}\right] \\ &= 1 - \left(1 - \exp\left(-\frac{\varepsilon_{\text{act}}}{k_\text{B}T}\right)\right) \\ &= \exp\left(-\frac{\varepsilon_{\text{act}}}{k_\text{B}T}\right)\end{aligned} \tag{18.6}$$

which is exactly the exponential term of the Arrhenius equation. We conclude that if the collision energies are exponentially distributed with mean $k_B T$ then the macroscopic behavior follows the Arrhenius equation.

This simple algorithm of (1) drawing a collision energy form an exponential distribution, (2) comparing it to the activation energy of the reaction, and (3) executing the reaction if the collision energy is greater, can efficiently be integrated to an artificial reaction algorithm. It remains to be defined how we should choose the parameter for the exponential distribution, i.e. how the Boltzmann constant k_B and the temperature T shall be interpreted in an abstract setting. Scafetta and West (2007) stated that, since the term $k_B T$ is the mean kinetic energy of a molecule, the term is equal to the total kinetic energy in the reaction vessel divided by the current number of molecules $|\mathcal{M}_v|$:

$$k_B T \equiv \frac{\sum_{s \in \mathcal{M}_v} \varepsilon_{\text{kin},s}}{|\mathcal{M}_v|} \tag{18.7}$$

We will use these observations to derive our energy-aware algorithm in the next section.

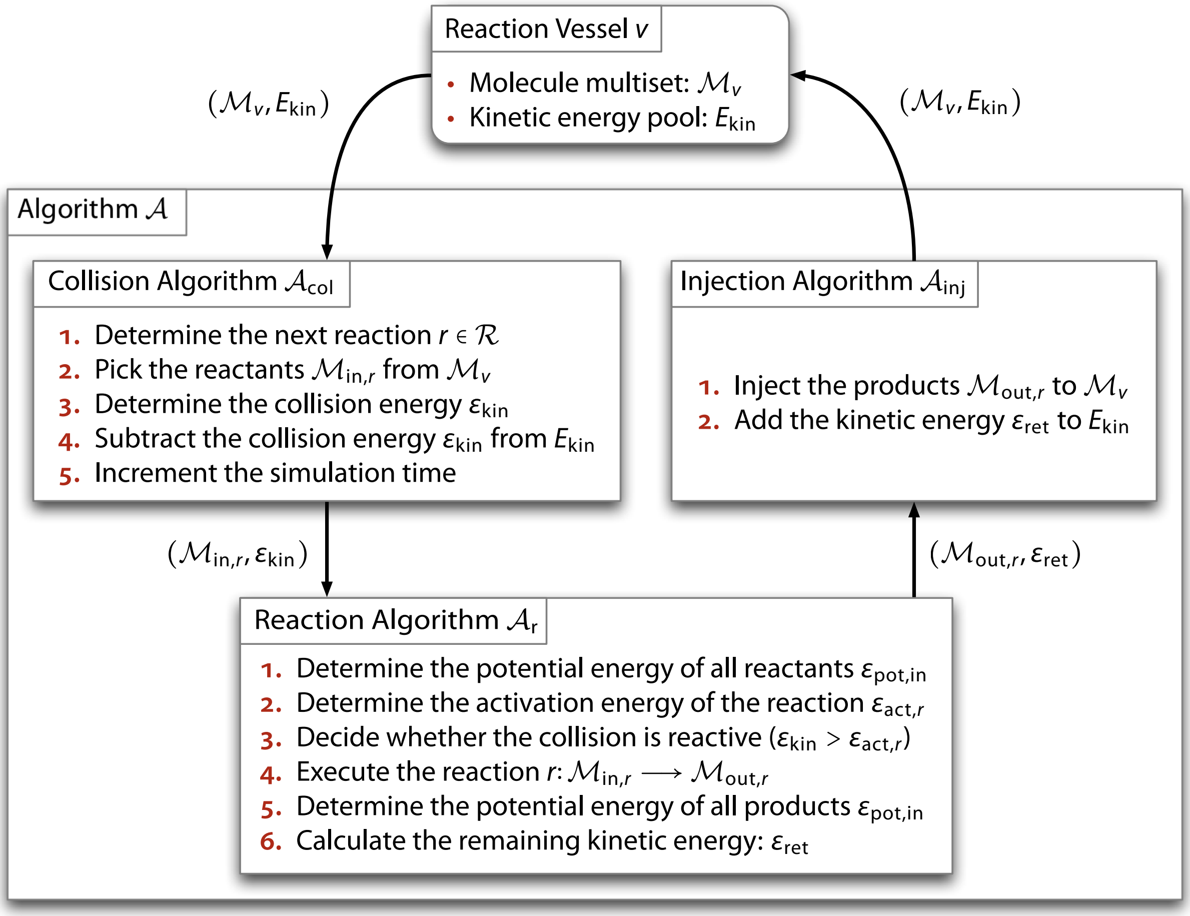
18.2 A GENERIC ENERGY FRAMEWORK FOR ARTIFICIAL CHEMISTRIES

The goal of our extended reaction algorithm is to mimic energy conservation of real chemical reactions, although in an abstracted form. The macroscopic dynamic behavior shall follow the Arrhenius equation, which allows for a mathematical analysis of a chemical program under the regime of the new algorithm. In this section, we present an energy framework that fulfills these requirements. Like any framework, it can be customized and targeted for the needs of a particular application case, i.e. for a particular artificial chemistry.

18.2.1 OUTLINE OF THE ENERGY FRAMEWORK

reaction vessel

Figure 18.2 shows an overview of the energy framework. As before, the *reaction vessel* contains a multiset in which the molecules (e.g. fraglet strings) are stored. We now require that the reaction vessel also keeps track of the total kinetic energy in the vessel E_{kin}. Note that for efficiency reason, we do not require the vessel to keep track of each molecule's kinetic energy or velocity vector. Instead, the vessel just has to store a single variable containing the total kinetic energy.



Reaction Vessel v
Molecule multiset: $\mathcal{M}_v$
Kinetic energy pool: E_{kin}
$(\mathcal{M}_v, E_{kin})$
$(\mathcal{M}_v, E_{kin})$
Algorithm $\mathcal{A}$
Collision Algorithm $\mathcal{A}_{col}$
1. Determine the next reaction $r \in \mathcal{R}$
2. Pick the reactants $\mathcal{M}_{in,r}$ from $\mathcal{M}_v$
3. Determine the collision energy ε_{kin}
4. Subtract the collision energy ε_{kin} from E_{kin}
5. Increment the simulation time
Injection Algorithm $\mathcal{A}_{inj}$
1. Inject the products $\mathcal{M}_{out,r}$ to $\mathcal{M}_v$
2. Add the kinetic energy ε_{ret} to E_{kin}
$(\mathcal{M}_{in,r}, \varepsilon_{kin})$
$(\mathcal{M}_{out,r}, \varepsilon_{ret})$
Reaction Algorithm $\mathcal{A}_r$
1. Determine the potential energy of all reactants $\varepsilon_{pot,in}$
2. Determine the activation energy of the reaction $\varepsilon_{act,r}$
3. Decide whether the collision is reactive ($\varepsilon_{kin} > \varepsilon_{act,r}$)
4. Execute the reaction r: $\mathcal{M}_{in,r} \longrightarrow \mathcal{M}_{out,r}$
5. Determine the potential energy of all products $\varepsilon_{pot,in}$
6. Calculate the remaining kinetic energy: ε_{ret}

Figure 18.2 **Block diagram of the generic energy-aware reaction algorithm**: A kinetic energy pool is added to the reaction vessel. The algorithm is split into three parts: (1) The collision algorithm collides molecules according to the law of mass action and determines the collision energy. (2) The reaction algorithm decides whether the reaction is effective or elastic. In the first case the reaction is executed. (3) The injection algorithm integrates the reaction products to the vessel and updates the vessel's energy pool.

The reaction algorithm $\mathcal{A}$ of the artificial chemistry $(\mathcal{S}, \mathcal{R}, \mathcal{A})$ is now split into *three functionally different blocks*: the collision algorithm, the reaction algorithm and the injection algorithm.

three sub-algorithms

Collision Algorithm *(a)*

The collision algorithm $\mathcal{A}_{col}$ inspects the vessel's multiset and schedules the next reaction $r \in \mathcal{R}$. Actually it schedules the next collision, because it is not certain whether the reaction is effective (happens) or elastic (not enough collision energy). The collision frequency is determined according to the law of mass action, i.e. it is given by the product of the number of reactant molecules. Thus the collision algorithm can be implemented by simply drawing molecules at random from the vessel, as it was done by Fontana and

Buss (1994) or Decraene et al. (2008), which is especially useful when most molecules can react with each other. If there are many different reactions we better use a well-known algorithm such as the ones by Gillespie (1977) or Gibson and Bruck (2000) (when the number of possible reactions is small).

collision energy

The collision algorithm also determines the *collision energy*. In reality, this energy can be derived form the velocity vectors of the colliding molecules. However, keeping track of each molecule's kinetic energy (motion) individually would have been computationally prohibitive. A stochastic approach is used instead. The collision energy ε_{kin} is determined based on the available free kinetic energy in the reaction vessel, E_{kin}, by drawing a random variable from the exponential distribution

$$\varepsilon_{\text{kin}} \sim \text{Exp}\left(\frac{E_{\text{kin}}}{|\mathcal{M}_v|}\right) \tag{18.8}$$

were $|\mathcal{M}_v|$ is the current number of molecules in the reaction vessel. This distribution stems from (18.5), but uses the average kinetic energy $E_{\text{kin}}/|\mathcal{M}_v|$ for the term $k_B T$ as proposed by Scafetta and West (2007).

When a collision happens, the collision algorithm selects the reactants $\mathcal{M}_{\text{in},r}$ from the vessel's multiset and calls the reaction algorithm with the reactants and the collision energy as arguments.

(b) Reaction Algorithm

determine if the collision is reactive

The reaction algorithm $\mathcal{A}_\text{r}$ has to *determine if the collision is reactive*. It therefore determines the potential energy of the reactants, $\varepsilon_{\text{pot,in}}$, as well as the activation energy of the reaction, $\varepsilon_{\text{act},r}$, and executes the reaction if the collision energy is greater than the activation energy ($\varepsilon_{\text{kin}} \geq \varepsilon_{\text{act},r}$). Otherwise, the collision is elastic, in which case the collision algorithm re-injects the reactants and the collision energy back to the vessel. If the reaction is effective the reaction algorithm determines the potential energy of the products and the kinetic energy that is left, ε_{ret}. As shown in Figure 18.1, energy must be conserved. In other words, the total input energy ($\varepsilon_{\text{pot,in}} + \varepsilon_{\text{kin}}$) must be equal to the total output energy ($\varepsilon_{\text{pot,out}} + \varepsilon_{\text{ret}} + \varepsilon_{\text{heat}}$) (some energy may be lost as heat). Finally, the reaction algorithm calls the injection algorithm with the products and the remaining kinetic energy as arguments.

(c) Injection Algorithm

The injection algorithm $\mathcal{A}_{\text{inj}}$ injects the products of the reaction, $\mathcal{M}_{\text{out},r}$, to the vessel's multiset $\mathcal{M}_v$ and adds the collision energy that was not converted into potential energy, ε_{ret}, to the energy pool E_{kin}.

In addition to rewriting the multiset, the overall algorithm also modifies the total kinetic energy variable. The two data structures influence each other: Endothermic reactions will not happen if the kinetic energy is too low, while the kinetic energy pool is increased or decreased based on the reactions executed.

CUSTOMIZATION OF THE FRAMEWORK 18.2.2

The algorithm and schema introduced above leaves several computational aspects underspecified as it only provides an energy framework that has to be instantiated for a specific artificial chemistry. The strength of an algorithmic chemistry is that the reaction rules are implicitly defined by the structure of the molecules (Dittrich et al., 2001) such that they can be redefined at run-time. We also require that the potential energy of a molecule is implicitly defined by its structure and that the activation energy of a reaction is defined by the structure of its reactants and products.

The framework therefore contains hooks in form of four parametric energy-mapping functions $f(\cdot) \to \mathbb{R}$. By changing these functions, the notion of energy can be redefined.

Potential energy function $\varepsilon_{\text{pot}} = f_{\text{pot}}(\mathcal{M})$: The potential energy function assigns a potential energy to each molecule in a multiset $\mathcal{M}(\mathcal{S})$, either in a predefined way or as a function of the molecules' structures.

Activation energy function $\varepsilon_{\text{act}} = f_{\text{act}}(r, \mathcal{M}_{\text{in},r}, \mathcal{M}_{\text{out},r})$: The activation energy function assigns each reaction $r \in \mathcal{R}$ an activation energy, either as a constant or as a function of the interactions between the involved molecules.

Energy return function $\varepsilon_{\text{ret}} = f_{\text{ret}}(\varepsilon_{\text{kin}}, \varepsilon_{\text{act}}, \mathcal{M}_{\text{in},r}, \mathcal{M}_{\text{out},r})$: The energy return function determines how much of the collision energy that has not been transformed to potential energy by a reaction is returned to the vessel's kinetic energy pool after the reaction. The remaining energy is lost to the environment as heat. To simulate a thermodynamically closed system one would set $f_{\text{ret}}(\varepsilon_{\text{kin}}, \varepsilon_{\text{act}}, \mathcal{M}_{\text{in},r}, \mathcal{M}_{\text{out},r}) = \varepsilon_{\text{kin}} - \big(f_{\text{pot}}(\mathcal{M}_{\text{out},r}) - f_{\text{pot}}(\mathcal{M}_{\text{in},r})\big)$.

Energy flux rate $r_{\text{e}}(t)$: The energy flux rate determines the amount of kinetic energy that is injected ($r_{\text{e}} > 0$) or drained ($r_{\text{e}} < 0$) from the reaction vessel. To simulate a thermodynamically closed system one would set $r_{\text{e}} = 0$.

18.2.3 ASSESSMENT AND RELATED WORK

The algorithmic framework proposed in this section mimics energy conservation of chemical reactions. Reaction rates are no longer defined explicitly, but derived from the structure of the colliding molecules. We designed the algorithm such that the macroscopic behavior follows the Arrhenius equation. However, it is left open to the user of the framework to define what energy represents. For example, the potential energy could represent the binding strength of the atoms constituting the molecule. The activation energy would then be the kinetic energy needed to break these bonds. This freedom of being able to arbitrarily define the meaning of energy is provided by the parametric energy functions. They offer the designer the possibility to arbitrary map molecules and reactions to energy values while the framework still operates according to the laws of thermodynamics. Note that we are using an abstracted dimensionless form of energy, i.e. not specified in Joules. In the context of an artificial chemistry, the energy term is just a placeholder for a resource, which is needed to build up complexity.

An obvious alternative way to take energy into account is to apply existing exact stochastic reaction algorithms (Gillespie, 1977; Gibson & Bruck, 2000) directly, using the full equation (18.3) as the weight of the reaction. A kinetic energy pool E_{kin} can still be used by substituting $k_{\mathrm{B}} T$ in (18.3) with its equivalent $E_{\mathrm{kin}}/|\mathcal{M}_v|$. This leads to a sum of exponentials (over all possible reaction weights), which is not necessarily faster than drawing a random number according to (18.5), especially when the number of possible reactions is large (which is often the case in algorithmic chemistries).

The advantage of our method is two fold: first, it simulates the microscopic level more accurately with little or no extra computational penalty; second, it provides a clear separation of concerns between the collision method and the energy-aware reaction part.

Related energy models aim mostly at the study of real chemistry or biology, especially the origin of life. These existing models do not seem directly appropriate in an algorithmic chemistry context. Some are too complex (the ToyChem approach by Benkö et al. (2005) defines reaction rules down to the quantum level), others too simplified for this purpose (for example, Lancet, Sadovsky, and Seidemann (1993) focus on equilibrium states whereas Pereira (2005) on catalytic reactions only). Most are unable to handle constructive systems: Many require the designer to specify rate and energy parameters for each species and reaction exhaustively (Gordon-Smith, 2007); or assign kinetic coefficients at random (Bagley & Farmer, 1992; Fernando & Rowe, 2007), without considering the reactants' composition or shape. Some handle shape explicitly (Lancet et al., 1993; Takeuchi & Hogeweg, 2008) but not activation energy.

A REACTION ALGORITHM TO PREVENT UNBOUNDED STRING ELONGATION IN FRAGLETS 18.3

In this section, we show how the energy framework can be parameterized to effectively prevent unbounded string elongation in an artificial string-based chemistry such as Fraglets.

CONFIGURATION OF THE ENERGY FRAMEWORK 18.3.1

To limit string growth, instead of defining an arbitrary length threshold, we limit the total energy ($E_{\text{tot}} = E_{\text{kin}} + E_{\text{pot}}$) and keep it at a constant level. That is, we simulate a thermodynamically closed system. Each string binds potential energy in proportion to its length. Unlike in nature, we allow energy to be freely converted between kinetic and potential energy. In order to produce longer string, the required potential energy must be taken from the common kinetic energy pool. In order to achieve this mechanism, we define the parametric functions of the energy framework as follows:

Potential energy: The potential energy of a molecule $s \in \mathcal{S}$ is set to its string length, and the potential energy of a multiset is the sum of the potential energy of its members:

$$f_{\text{pot}}(s) = |s| \tag{18.9a}$$

$$f_{\text{pot}}(\mathcal{M}) = \sum_{s \in \mathcal{M}} f_{\text{pot}}(s) \tag{18.9b}$$

Activation energy: Reactions that generate additional symbols require energy from the pool. The collision energy for these endothermic reactions has to overcome the energy required for the additional symbols. Exothermic reactions, i.e. reactions where the products contain less symbols than the reactants, are always effective:

$$f_{\text{act}}(r, \mathcal{M}_{\text{in},r}, \mathcal{M}_{\text{out},r}) = \max\left[0, f_{\text{pot}}(\mathcal{M}_{\text{out},r}) - f_{\text{pot}}(\mathcal{M}_{\text{in},r})\right] \tag{18.10}$$

Energy return: An exothermic reaction returns the energy that corresponds to the deleted symbols back to the kinetic energy pool; no energy is lost as heat:

$$\begin{aligned} f_{\text{ret}}(\varepsilon_{\text{kin}}, \varepsilon_{\text{act}}, \mathcal{M}_{\text{in},r}, \mathcal{M}_{\text{out},r}) &= \varepsilon_{\text{kin}} \\ &- \left(f_{\text{pot}}(\mathcal{M}_{\text{out},r}) - f_{\text{pot}}(\mathcal{M}_{\text{in},r})\right) \end{aligned} \tag{18.11}$$

Energy flux: The system is thermodynamically closed:

$$r_e = 0 \tag{18.12}$$

The total energy is fully conserved by returning the whole potential energy difference back to the kinetic energy pool. If a reaction vessel tends to produce longer strings, the temperature (i.e. the average kinetic energy $E_{\text{kin}}/|\mathcal{M}_\nu|$) drops and consequently, elongation becomes less frequent. This does not mean that elongation reactions are totally stopped. But it is less likely to draw a collision energy high enough to overcome the activation energy for an elongation reaction. Macroscopically, this may be interpreted as a reduction of the reaction coefficient depending on the temperature as characterized by Arrhenius. Competing reactions that shorten strings are spontaneous in the sense that they do not require any collision energy in order to be executed, because they do not build up mass. The system automatically finds a balance between string elongation and reduction reactions and makes the most efficient use of the available energy.

18.3.2 FRAGLETS SIMULATION OF A POLYMERIZATION REACTION

In order to illustrate the working principle of the energy-aware reaction algorithm we consider the following example, which mimics a simple polymer grow-shrink process akin to chain-growth polymerization. A single type of monomer X can form polymer chains of arbitrary length i, represented as X_i. The polymerization reaction adds one monomer to the polymer, while depolymerization removes one monomer.

$$X_i \xrightarrow{k_{\text{pol}}} X_{i+1} \tag{18.13a}$$

$$X_i \xrightarrow{k_{\text{dep}}} X_{i-1} \quad \text{for } i > 1 \tag{18.13b}$$

According to the energy mapping functions, each molecule is assigned a potential energy equal to its length. Furthermore, the activation energy of the polymerization reaction is $\varepsilon_{\text{act,pol}} = 1$, and for the depolymerization reaction no activation energy is required: $\varepsilon_{\text{act,dep}} = 0$. We assign the polymerization process a higher pre-exponential factor $A_{\text{pol}} = 2$ whereas $A_{\text{dep}} = 1$ in order to favor polymerization, just to show afterwards how such elongation can

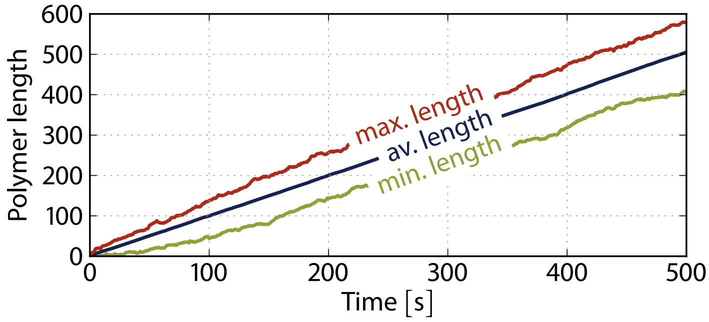


Figure 18.3 Unbounded polymer elongation without energy constraints: Minimum, average, and maximum polymer length development over time. The reaction vessel is initialized with 100 polymers of length 1; there is no energy conservation. The average polymer length grows linearly, because the polymerization coefficient is larger.

be controlled via the energy conservation mechanism. This results in the reaction coefficients

$$k_{\text{dep}}(t) = A_{\text{dep}} \exp\left(-\varepsilon_{\text{act,dep}} \frac{|\mathcal{M}_v|}{E_{\text{kin}}(t)}\right) = 1 \tag{18.14a}$$

$$k_{\text{pol}}(t) = A_{\text{pol}} \exp\left(-\varepsilon_{\text{act,pol}} \frac{|\mathcal{M}_v|}{E_{\text{kin}}(t)}\right) = 2e^{-|\mathcal{M}_v|/E_{\text{kin}}(t)} \tag{18.14b}$$

Note that the kinetic energy $E_{\text{kin}}(t)$ and thus the reaction coefficient of the polymerization reaction $k_{\text{pol}}(t)$ varies over time. On the other hand, for this reaction network only, the total number of molecules $|\mathcal{M}_v|$ remains unchanged.

Without Energy Conservation (a)

We first simulate the system without energy control, using a simple law of mass action algorithm. This is equivalent to choosing an activation energy of zero ($\varepsilon_{\text{act}} = 0$) for all reactions, leading to $k_{\text{pol}} = 2$. Figure 18.3 shows the time evolution of a typical simulation run. The reaction vessel is initially equipped with 100 polymers all of length 1. Hence, the initial total potential energy is $E_{\text{pot}} = 10^3$. The overall concentration remains constant whereas the polymer lengths grow linearly in time. The reason for this elongation is the higher reaction coefficient of $k_{\text{pol}} = 2$ compared to the depolymerization coefficient of $k_{\text{dep}} = 1$.

With Energy Conservation (b)

We now repeat the simulation using the energy framework with an initial kinetic energy pool of $E_{\text{kin}}(0) = 9900$. The system is thermodynamically closed, meaning that the total energy is always conserved at $E_{\text{tot}} = E_{\text{pot}} + E_{\text{kin}} = 10^4$. Figure 18.4 shows the distribution of polymer lengths over time. There is

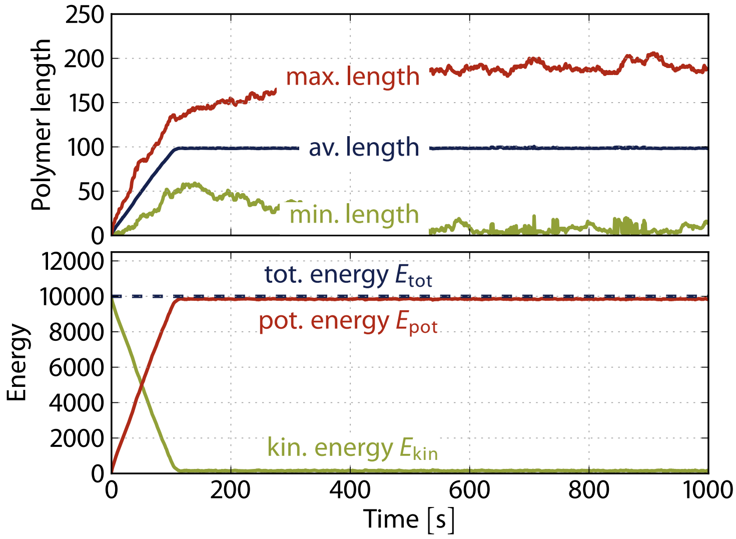


Figure 18.4 Bounded polymer elongation with energy constraints: Top: Minimum, average, and maximum polymer length development over time. Bottom: Free kinetic energy, potential energy bound in the polymers, and total energy. The reaction vessel is initialized with 100 polymers of length 1. The total energy is kept constant at 10^4 units. The average polymer length grows linearly until the kinetic energy of the vessel is exhausted. Then, a stable length equilibrium establishes.

still the same drift towards longer polymers as without energy constraints. However, after the majority of the kinetic energy is converted into potential energy, the average polymer length settles at a stable equilibrium where both, the energy-dependent polymerization and the constant depolymerization reaction reached equal coefficients. A diffusion-like behavior reshuffles the polymers around that average length. At $t \to \infty$, the polymer lengths will be distributed exponentially, with many monomers and a few long polymers

We can even determine the average polymer length analytically. At equilibrium the two reaction coefficients must be equal:

$$\hat{k}_{\text{dep}}(t) = \hat{k}_{\text{pol}}(t) \tag{18.15}$$

$$1 = \exp\left(-\frac{|\mathcal{M}_v|}{\hat{E}_{\text{kin}}}\right) \tag{18.16}$$

This leads to the equilibrium value for the free kinetic energy.

$$\hat{E}_{\text{kin}} = \frac{|\mathcal{M}_v|}{\ln 2} \tag{18.17}$$

The potential energy is equal to the number of symbols in the vessel.

$$E_{\text{pot}} = E_{\text{tot}} - E_{\text{kin}} = \sum_{s \in \mathcal{M}_v} |s| = |\mathcal{M}_v| \left\langle |s| \right\rangle \tag{18.18}$$

Hence, the mean string length at equilibrium is

$$\left\langle |s| \right\rangle = \frac{\hat{E}_{\text{pot}}}{|\mathcal{M}_v|} = \frac{E_{\text{tot}}}{|\mathcal{M}_v|} - \frac{1}{\ln 2} = \frac{10^4}{10^2} - \frac{1}{\ln 2} = 98.56 \tag{18.19}$$

This matches the value observed in Figure 18.4. Note that not all kinetic energy has been converted to potential energy (symbols), because some polymers always depolymerize.

This is an efficient yet flexible way of restricting length explosion in an artificial chemistry. Unlike other approaches where the length of polymers is restricted by throwing away molecules longer than an arbitrarily chosen maximum length, here, this limit is fathomed dynamically based on the available virtual energy. In the next section, we will review hard length restrictions and show that they lead to undesired dynamic behavior. By using the energy framework, we can inject or drain kinetic energy to leave the system more or less freedom to explore the state space.

MUTATIONAL ROBUSTNESS BY CELL-GROWTH AND MULTI-LEVEL SELECTION 18.4

The energy-aware reaction algorithm is able to confine unbounded elongation, but it cannot prevent a mutation from changing a healthy reaction network into one that does not produce the right result anymore and starts to accumulate waste. As we realized in the previous chapter, a single harmful symbol mutation irrecoverably pushes a stable organization into an infinite closure.

In order to solve the elongation problem in their chemistry, Decraene et al. (2008) proposed to introduce multi-level selection where there is competition for resources among multiple vessels in addition to the selective pressure within the reaction vessel. In this section, we apply a similar method such that a reaction vessel suffering the elongation problem is going to be replaced by a viable mutant of another vessel.

OUTLINE OF THE MULTI-LEVEL SELECTION ALGORITHM 18.4.1

In Section 14.1, we showed how replicating molecules in a limited environment result in a selective pressure that eventually dilutes non-viable, i.e. non-replicating molecules. In the same way, we now install a selective pressure among the reaction vessels (or *cells*). Vessels shall be able to replicate themselves when being healthy akin to *cell division*. That is, several cells run in parallel and perform the same task. The total number of cells is limited. Thus a dividing vessel displaces another randomly selected one. Consequently, cells that divide faster are naturally selected against the defective ones.

cell

cell division

We regard the production rate of membrane molecules (to be defined) as the fitness of a vessel. The ultimate goal is to constrain the system such that membrane production is linked to the efficiency of the cell with respect to cell division mechanism

the expected result. Once a cell decides to divide, the *cell division mechanism* atomically executes the following steps:

1. A new empty offspring cell (reaction vessel) is generated.
2. Half of the molecules of the dividing cell are selected randomly to move to the offspring cell.
3. One of the migrated molecules is randomly mutated. (This step is optional in order to allow for intentional program evolution.)
4. The new cell displaces one of the already existing cells; this maintains a constant population of cells in the environment.

The third step intentionally mutates one molecule of the offspring. Since we expect the cell population to be robust to mutations we now even use mutation to introduce novelty in order to optimize the overall system. Natural selection arises because the faster a cell is able to generate membrane molecules, i.e. the fitter it is, the sooner it divides and thus has a selective advantage over the other cells. A cell that is not able to produce membranes anymore will eventually be displaced.

18.4.2 A SIMPLISTIC VERSION OF FRAGLETS

For our evolution experiments we use a simplified version of Fraglets, designed on purpose to show a very aggressive elongation behavior. The chemistry consists of polymers strings $s = \Sigma^*$ of arbitrary length over an alphabet of only 4 symbols $\Sigma = \{\mathtt{A}, \mathtt{M}, \mathtt{f}, \mathtt{n}\}$. Like in traditional Fraglets, the first symbol of a string implicitly defines the string rewriting operation applied to this molecule as follows

$$\begin{aligned} \mathtt{A}\Psi + \mathtt{M}\Omega &\longrightarrow \Psi\Omega && \text{(match and join)} \\ \mathtt{f}\alpha\beta\Omega &\longrightarrow \alpha\Omega + \beta\Omega && \text{(fork)} \\ \mathtt{n}\Omega &\longrightarrow \Omega && \text{(nop)} \end{aligned}$$

where α, β are arbitrary symbols and Ψ, Ω are strings. A simple Quine also exists in this chemistry:

$$\mathtt{AfAM} + \mathtt{MfAM} \longrightarrow \mathtt{fAMfAM} \longrightarrow \mathtt{AfAM} + \mathtt{MfAM}$$

Like Fraglets, this chemistry tends to produce strings of increasing length since the join reaction almost doubles the length on average while the two transformations only reduce the length by 3 (fork) or 1 (nop) symbols, respectively. Therefore it is no surprise that most random soups lead to unlimited elongation.

SIMULATION SETUP 18.4.3

In this section, we study the evolutionary behavior of a manually designed replicating Quine: {`AfffAM`, `MfffAM`}. These molecules react and, during the reduction steps, generate two copies of themselves. Thus the concentration within a reaction vessel grows hyperbolically when being simulated with a law of mass action algorithm.

We put $N_v = 100$ vessels running the initial self-replicating program in an evolutionary context by forcing the cell to divide whenever the number of *membrane molecules* (here: molecules starting with symbol `M`) reaches a threshold of 1000 instances.

membrane molecules

The experiment should demonstrate whether the initial self-replicator is able to persist despite symbol mutation events.

SIMULATION RESULTS 18.4.4

We carried out three different simulation scenarios: one where there is no length restriction, one where we delete or truncate long molecules, and finally we used our energy framework to constrain the vessel's uptake of energy.

No Length Restriction (a)

A significant number of all possible mutations on the initial program lead to a set of molecules with infinite closure: the molecules of such a set have the potential to reach an infinite sequence space when reacting among each other. Consequently, without any length restriction, such a mutation may easily result in the accumulation of ever-growing strings. The resulting hyperbolically rising CPU and memory requirements prevent the simulation of such systems to be carried out on real-world computers. None of the 20 simulation runs we carried out survived 10 generations without exhausting our machine's resources.

Hard Length Restrictions (b)

We then tried two simple methods to prevent strings from growing infinitely: First, we destroyed molecules longer than a certain threshold l (arbitrarily chosen), and second, we truncated molecules longer than this threshold. Both methods prevent elongation but have an undesired side effect as discussed below. We performed several experiments for thresholds $l = 10, 20, 30, 100$ with a population size of $N_v = 100$ vessels:

After several generations, we always observed systems where the number of A- and M- molecules were no longer balanced. This is a consequence of the stochastic distribution of molecules during cell division. Both deviations from the symbolic equilibrium may result in a higher reproductive ratio of the cell:

A cell that generates more M- than A-molecules has a selective advantage by replicating faster since the membrane threshold is reached sooner. However, if the cell carries this strategy to excess it will cease producing A-molecules necessary for sustained replication.

A cell that instead speeds up the production of A-molecules indirectly produces more membranes, too, due to the law of mass action. The top subfigure of Figure 18.5 shows that the reproductive ratio of such systems is much higher than the one of the original program. Such mutants quickly take over the population after some generations. However, this high productivity comes along with a lower efficiency (see Figure 18.5, middle). We measure

efficiency

the *efficiency* as the surface to volume ratio where the surface is the number of membrane molecules and the volume is defined as the total number of symbols in the cell, i.e. the sum over all molecule lengths. The vessels fully exploit the length restriction: the average molecule length almost reaches the length threshold l as depicted in the bottom subfigure of Figure 18.5. A typical resulting cell multiset that speeds up production and favors A- over M-molecules is as follows:

$$\begin{aligned}\{&\mathtt{MfffffffffffffffffAM}_{278},\\ &\mathtt{MfffffffffffffffffAA}_{241},\\ &\mathtt{AfffffffffffffffffAM}_{1974},\\ &\mathtt{AfffffffffffffffffAA}_{553}\}\end{aligned}$$

In spite of applying simple length constraints, a population of cells under evolutionary pressure easily finds a way to increase the average reproductive ratio without respecting any notion of efficiency with respect to virtual or physical resources like the length of strings (virtual mass) or the CPU cycles needed to simulate the reactions. The chemistry is only able to respect such constraints when we integrate a notion of them into the microscopic rules of the artificial chemistry as described below.

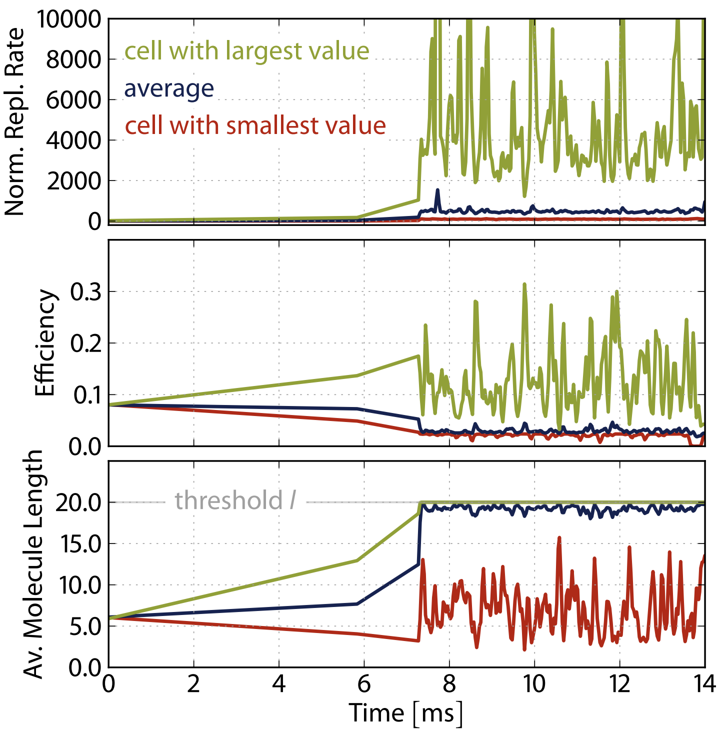


Figure 18.5 Length-exploiting replicators for hard length restriction: Evolution in a population of $N_v = 100$ cells, all initialized with the replicating Quine. Molecules longer than $l = 20$ symbols are deleted. The three subplots show the average measures together with the measures of those cells with the smallest and largest value in the population: **Norm. Repl. Rate:** Replication rate of the cell, normalized with respect to the replication rate of the initial Quine. A sudden discovery of a fast replicator can be observed at time $t = 7.3$ s. **Efficiency:** Measures how efficient a cell produces membranes needed for its division; i.e. fraction of membrane molecules divided by the total mass of the cell (total number of symbols). **Av. Molecule Length:** Average molecule length in the cell, i.e. total number of symbols divided by the number of molecules.

Energy Conservation (c)

A better method of preventing string elongation is to gradually slow down the production of new symbols on resource shortage. We aim at steering the growth by controlling the system's virtual energy inflow. By using a stochastic approach, our energy framework simulates the microscopic energy barrier of a reaction. This method turns out to be very effective in restricting the length of evolved solutions.

We use the same energy mapping functions $f(\cdot)$ as in the polymerization example (see Section 18.3.1). However, unlike the chain-growth polymerization example where the total energy was kept constant we now continuously inject kinetic energy in order to allow the cells to grow.

Moderate Energy Injection Rate. Again we carried out simulations in a population of $N_v = 100$ vessels and started with a moderate energy injection rate of $r_e = 10^5$/s. The shortage of kinetic energy results in linear growth of the concentration of the membrane molecule within a cell running the initial program. After 100 generations, most of the vessels still run the initial program; no better solution could be found. Unlike before, the system now cannot increase the reaction rate by excessively producing A-molecules. Any

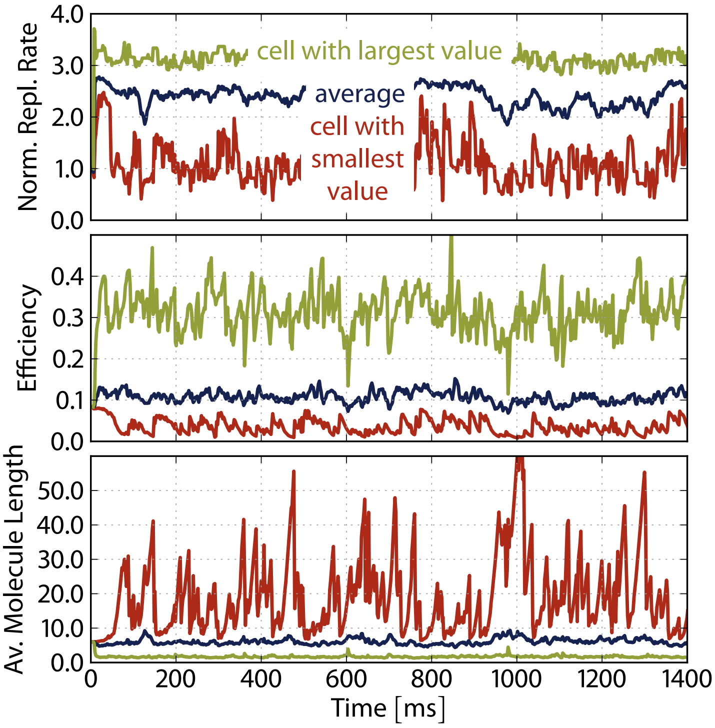


Figure 18.6 Maintained replicators within the energy framework: Evolution in a population of $N_V = 100$ cells, all initialized with the replicating Quine. Kinetic energy is limited and injected at rate $r_e = 10^7$/s. The three subplots show the average measures together with the measures of those cells with the smallest and largest value in the population: **Norm. Repl. Rate:** Replication rate of the cell, normalized with respect to the replication rate of the initial Quine. **Efficiency:** Measures how efficient a cell produces membranes needed for its division; i.e. fraction of membrane molecules divided by the total mass of the cell (total number of symbols). **Av. Molecule Length:** Average molecule length in the cell, i.e. total number of symbols divided by the number of molecules.

production of molecules decreases the temperature and makes endothermic reactions less frequent.

Critical Energy Injection Rate. A moderate energy injection rate successfully eliminates mutants with infinite closures. However, if we increase the energy injection by two orders of magnitude to a rate of $r_e = 10^7$/s we observe a qualitatively different phenomenon: The effect of injecting more energy to an initial program is that it is able to grow hyperbolically after each cell division because sufficient energy is available. The probability of an effective reaction is almost one in this phase. During cell growth, the kinetic energy is shared by an increasing number of molecules. This “cools down” the vessel, which gradually returns back to linear growth. Arising mutants with infinite closures start to explore the sequence space by generating longer molecules, but due to the energy restriction the cells are limited in doing so extensively. The existence of longer strings leads to viable mutants that incorporate these new molecules while still being able to survive, i.e. they still have comparable reproduction ratios; the average reproductive ratio remains more or less constant over the whole simulation run as illustrated in the top subfigure of Figure 18.6.

One of the reoccurring inventions we observed in cells after 100 generations is a cluster of molecules of different lengths that all react among each

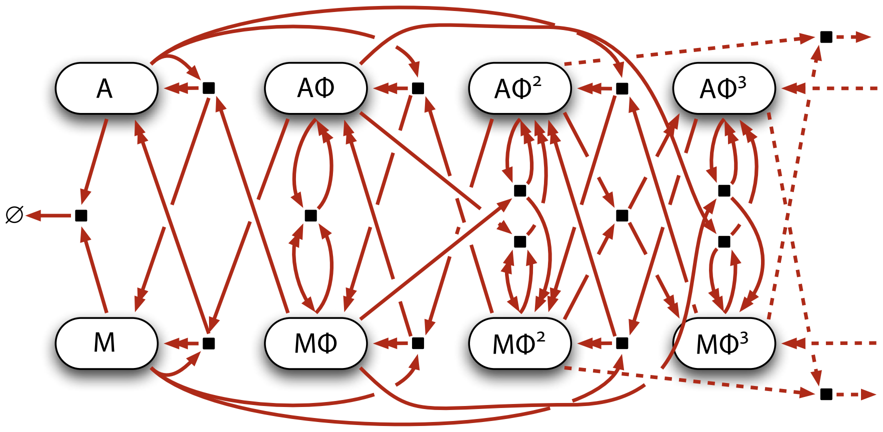


Figure 18.7 Reaction network of the emerging self-replicating cluster: All opponent molecules within the cluster react with each other and either annihilate each other (A + M), replicate each other (AΦ + MΦ), or generate two copies of longer molecules ($A\Phi^i + A\Phi^i$ where $i \geq 2$). The double arrow head means that two instances of the product are being generated.

other (see Figure 18.7). The evolved reaction network can be described by the following reaction equations

$$\begin{array}{ll} \mathtt{A}\Phi^i + \mathtt{M}\Phi^j \rightarrow \Phi^{i+j} \rightarrow 2\mathtt{fAM}\Phi^{i+j-1} \rightarrow 2\mathtt{A}\Phi^{i+j-1} + 2\mathtt{M}\Phi^{i+j-1} & \text{for } i+j \geq 0, \\ \mathtt{A} + \mathtt{M} \rightarrow \varnothing & \text{otherwise.} \end{array}$$

where Φ^n denotes the n-fold concatenation of the copy and split pattern $\Phi = \mathtt{fffAM}$. This motif reoccurs in all molecules. The resulting reactions generate two instances of shorter, longer, or equal sized molecules. The cluster contains a lot of very short molecules (A and M). These molecules do not contain the necessary information to replicate themselves. However, they react with a small number of larger molecules that contain multiple copies of the copy and split motif. These longer molecules are maintained at a lower concentration, which exponentially decreases with their length.

Even though the cells start to produce larger strings, the average molecule size remains constant as depicted at the bottom of Figure 18.6 and the population maintains its efficiency as shown in the subfigure above. Finally, the subfigure at the bottom of Figure 18.8 shows the length distribution of a typical simulation run after 100 generations. Vessels that contain the mentioned cluster are prominently present in the population. The cluster consists of molecules of size 1, 6, 11, 16, 21, …, which is clearly visible in the figure.

High Energy Injection Rate. When we further increase the inflow of kinetic energy to $r_e = 10^8$/s the vessels grow hyperbolically without energy shortage. Mutants start to explore the sequence space more aggressively and without hindrance, which leads to the same problems as discussed for the simple

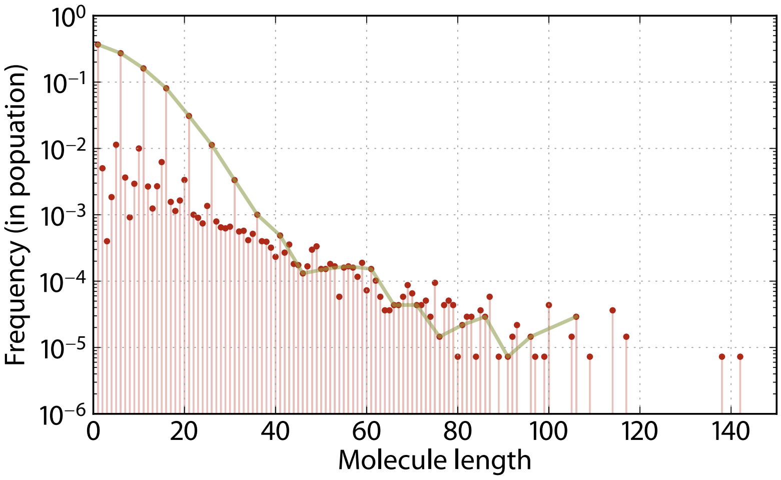


Figure 18.8 Molecule length histogram of an energy-controlled population: Distribution of molecule lengths. The green curve connects molecule lengths 1, 6, 11, 16, 21, …, suspected to belong to the described cluster.

constraints: The number of A- and M-molecules becomes unbalanced, which leads to a sudden increase of the replication ratio of some cells that take over the whole population in turn, followed by the sudden death of the whole infected population due to the extinction of either A- or M-molecules. Thus there exists a range of energy injection rate for which the system is able to survive *and* explore. A lower injection rate leaves no energy for exploration while a higher injection rate leads to a outbreak of fast-burning vessels that inevitably cause the death of the whole population.

18.5 DISCUSSION

In this chapter, we introduced a generic energy framework for algorithmic chemistries that stochastically simulates the energy exchange of a reaction on the microscopic level at which artificial chemical computing systems transform information.

18.5.1 EQUIVALENCE TO WET CHEMISTRY

One can expect the artificial chemistry to behave thermodynamically and kinetically similar to an equivalent real chemistry, yet in a simplified form. The resulting macroscopic dynamic behavior follows the temperature dependent Arrhenius equation known from real chemistry. By adapting four parametric energy functions, our framework allows the experimenter to simulate different thermodynamic scenarios for different chemistries.

EFFICIENCY 18.5.2

The method suggested avoids counting on hardware-dependent parameters such as CPU run time to control and constrain the consumption of resources like the approach by Decraene et al. (2008). It rather extends existing exact stochastic simulation algorithms by an additional energy-processing step, which requires us to draw an additional random number and compare it to the activation energy of the reaction. When the kinetic energy in the vessel is high, i.e. when all reactions are effective, we do not observe significant performance drawbacks compared to traditional law of mass action algorithms (Gillespie, 1977; Gibson & Bruck, 2000). When lowering the kinetic energy, the collisions become more elastic, which results in less reaction steps performed per physical time. However the virtual time unit that a CPU is able to simulate per physical time unit does not decrease in this case.

EFFECTIVENESS WITH RESPECT TO THE ELONGATION PROBLEM 18.5.3

As an application case, we presented a solution for the well-known elongation problem in artificial polymer chemistries. We constructed a rather aggressive instruction set in terms of length control and were still able to *keep the elongation process under control*. Even if there is no mass conservation, and even if growth is largely favored, an external constraint on energy inflow is able to keep the string length within reasonable bounds. Furthermore, the obvious but not optimal solution of a *hard length restriction is not needed*. Such a restriction may reveal undesired side effects like a symbolic imbalance that only becomes apparent in stochastic as opposed to deterministic simulations (e.g. ODEs).

keep the elongation process under control

hard length restriction is not needed

We noted that the amount of energy that we inject influences the exploratory capability of a population of artificial cells: If the injected energy flow is too low, the cells show linear growth and the population does not find better solutions for the initial self-replicator. For energy injection rates that are very high, our cells exhibit unbounded growth which leads to the same result as for hard length restrictions: the symbols get unbalanced resulting in a very high reproductive ratio of the affected cells. This is followed by the sudden death of the whole population. Even if the mechanisms behind this behavior are not comparable with those that trigger the elongation catastrophe in the work of Decraene et al. (2008), interestingly, the resulting effect of the "rise and fall of the fittest" is the same.

For a narrow range of medium energy injection rates, the system survives while exploration of the sequence space is moderate, keeping molecule lengths emerging "clusters"

within acceptable bounds. We observed *emerging "clusters"* of molecules, which all together form an autocatalytic set. Even though the closure of the set is infinite, the dynamics of the energy-aware algorithm makes reactions that form long strings more unlikely. This nicely reflects the nature of biochemical reaction system where more and more complex molecules evolve over time in the presence of enough energy.

18.5.4 POTENTIAL OF THE ENERGY FRAMEWORK

However, the elongation examples do not fully make use of the capability of our energy framework. For example, we used the smallest possible value as activation energy, i.e. the difference in potential energy between product and reactant molecules. In the future, we would like to have a more realistic mapping from molecule structure to activation energy in order to simulate enzymatic reactions, which lower the activation energy with respect to the uncatalyzed reactions. This will hopefully lead to metabolic networks in a constructive artificial chemistry.

We already linked the virtual potential energy of molecules to the memory resources they need. Another promising approach would be to link the virtual kinetic energy to real electrical energy or computational power, respectively. Reactions need CPU time to be executed. The activation energy and kinetic energy gradient between products and reactants could be chosen such that the virtual heat generated by the reaction is equivalent to the electrical energy dissipated by the underlying hardware while executing this reaction. The chemical program would then shrink and grow in correspondence to the real energy invested.

18.5.5 APPLICATION TO NETWORKING PROTOCOLS

evolving protocols

By allowing or at least tolerating mutations we now talk about *evolving protocols*. Evolution promises to optimize a system, but has the drawback that this optimization strategy is often hard to control. In order to use the energy framework to run and evolve networking protocols, protocol software must be able to reliably evaluate its own fitness: The protocol's reaction network must generate membrane molecules when it is operating correctly; the rate of membrane production, and hence the rate of cell division being an indirect measure of the software's fitness. This raises the problem of *distributed fitness evaluation*: How can we determine whether a protocol is operating correctly with local knowledge only? How can we make sure a reaction network does not just exploit all energy without actually performing the desired functionality? Hence, how can we map a desired functionality to energy constraints? We

distributed fitness evaluation

cannot give an answer to those questions yet, and it probably needs another research project to attack them.

SUMMARY 18.6

This chapter closes the third part of this thesis. In Chapters 14 and 15, we introduced Quines as a method to make software intrinsically self-healing and studied their robustness analytically on the macroscopic, molecular level. In the previous chapter, we then examined the effect of memory mutations and execution errors on the microscopic, symbolic level and recognized that they are hard to mask with the current Fraglets instruction set. In this chapter, we tried another approach by tolerating mutations. We introduced the notion of energy and were able to successfully stop uncontrolled elongation of molecules in infinite reaction networks. Our multi-level selection mechanism is a potential framework for future protocol evolution.

MACROSCOPIC TREATMENT 18.6.1

We started with the hypothesis that faults break the self-replicating capabilities of replicating Quines. In this case, the system can be analyzed mathematically by the Catalytic Network Equation (Stadler et al., 1993). This equation captures the essence of the dynamics of chemical replicators and has strong links to similar equations in biological population dynamics (Lotka-Volterra equation (Lotka, 1910; Volterra, 1926)) and evolutionary game theory (Maynard Smith & Price, 1973; Nowak, 2006, Chap. 4). Indeed, our system showed homeostatic behavior, meaning that a perturbation of the code on the macroscopic level (by removing parts of the code) does not disrupt the function of the protocol. The remaining code even automatically regenerates the missing parts without actually being “aware” of the fault. This is possible because we forced the healthy software to have a large basin of attraction. Thus, we used the same principles for code-level robustness than for robustness against perturbation in the network environment.

MICROSCOPIC TREATMENT 18.6.2

We identified two important causes for run-time software faults, namely spontaneous memory bit mutations and unreliable instruction execution. These faults affect the microscopic structure of the chemical program. Experiments revealed that our hypothesis is only valid for ninety percent of these faults, and that one out of ten microscopic faults lead to a replication spiral, which produces longer and longer molecules. This indicates that on the microscopic level, the system is still vulnerable to code-perturbations. We recognized that with the current Fraglets instruction set, it is not possible to come up with a structural fixed point, because the perturbation of one symbol sometimes moves the system far away in state space. We hope to solve this problem in the future with a probabilistic instruction set.

18.6.3 TOWARDS EVOLVING PROTOCOLS

Mutations can be masked by a redundant binary encoding scheme. However, unreliable execution is not easy to capture and cure. We proposed a multi-level selection model that does not try to prevent mutations but reshape the state space such that epidemic mutants are forced to die. Our solution is based on the observation that the majority of infectious mutants either die out or consume a lot of memory and CPU resources. By introducing a notion of energy and by implicitly linking the memory requirements of a reaction system to the speed of its computation, we were able to stop the uncontrolled growth and elongation of replication spirals.

However, we believe that an intrinsically self-healing system will always be an evolving system. Ingredients for evolutionary processes are replication, selection, and variation. Variation automatically comes from the environment in form of unreliable hardware. Replication is needed to continuously refresh healthy software or at least to verify its integrity. Finally, selection is required to sort out erroneous code and to proliferate the correct program.

Whether the automatic synthesis of protocols, or even their evolution, will be required to master the complexity of the future Internet is not clear right now. Because the chemical networking approach provides both, a formal framework to analyze protocols, as well as a baseline robustness that is needed for evolution (A. Wagner, 2007), we see the potential of CNPs for both engineered and evolved protocols.

PART IV

Discussion

CHAPTER 19

Discussion in Relation to Existing Work

The chemical networking paradigm related to other networking principles, programming methods, and execution models.

> That simple principle
> predicts almost everything
> that's happening. [19]
>
> Noam Chomsky

This chapter puts our chemical networking paradigm into the context of other work and illustrates common links and differences. We structure the discussion around the illustration in Figure 19.1, which situates several programming paradigms and networking principles in a three-dimensional space.

Most classical networking protocols are executed by immobile code, programmed in an imperative language, and are focusing on the structural aspects of information exchange. They reside in the bottom-left-front corner of the cube. In contrast, chemical protocol code is mobile, rule-based, and intrinsically dynamic. The arrows in Figure 19.1 show how the chemical networking principle was inspired by advances along each of the three axes and from natural chemical reaction systems. However, the advantages of being in the top-right-back corner of the cube come at the price of increased randomness and higher resource requirements. We will discuss this additional cost and show that intrinsic noise can be beneficial and that the increased memory and computing power requirements are justifiable for the benefits of the chemical model.

The first three sections of this chapter discuss the advantages of the three paradigmatic changes and relate CNPs to other work along these axes: In

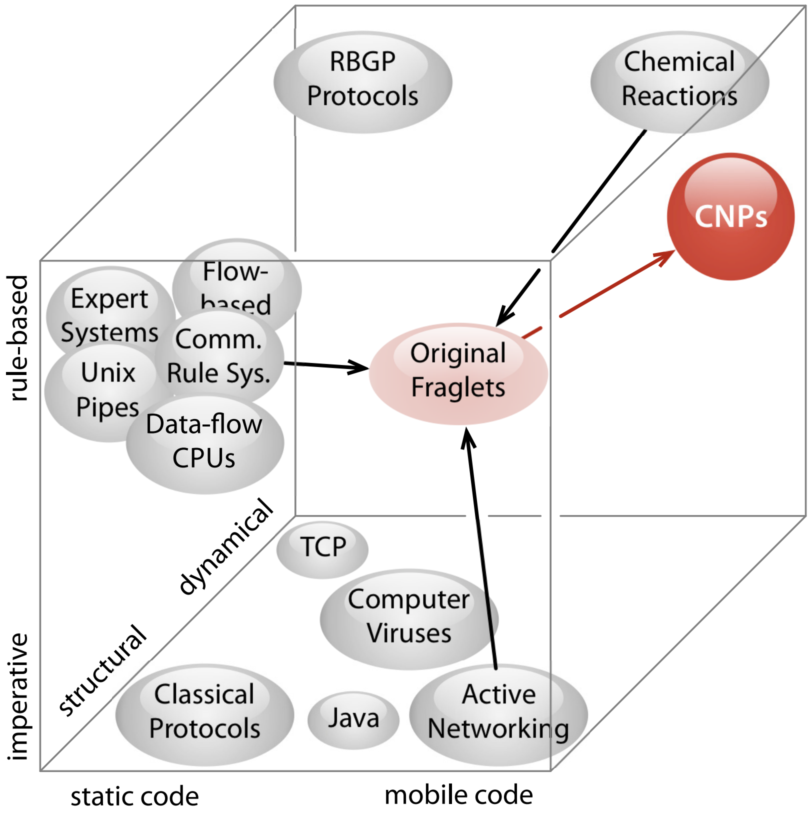


Figure 19.1 Chemical networking in relation to other paradigms: Classical programs and protocol implementations are static pieces of code, programmed in an imperative language, focusing on the structural aspects of (distributed) computation. Chemical protocol code is mobile, rule-based, and intrinsically dynamic. Fraglets was inspired by advances along the three axes.

Section 19.1, we look at the difference between imperative and rule-based programming languages and locate Fraglets in between the two. In Section 19.2, we show how chemical protocols programmed in Fraglets can freely chose between using mobile or static code. Section 19.3 recapitulates how the structure of chemical protocols is linked to their behavior and how chemical protocols are embodied in the environment of a computer network. In Section 19.4, we then debate whether the intrinsic noise of chemical protocols is benefit or drawback. Finally, in Section 19.5, we show that the higher robustness of CNPs comes at the price of higher resource demands.

19.1 RULE-BASED PROGRAMMING MODELS

imperative, procedural

Today, most networking protocols are programmed in a *imperative, procedural* programming language. Such implementations group together sequences of statements in procedures that change the state of the program. The chemical metaphor suggests departing from the sequential execution and from the state-centric view. Reacting molecules can be captured better by a rule-based approach.

RULE-BASED PROGRAMMING 19.1.1

rule-based

Rule-based systems organize computation as a sequence of rewriting steps, operating on different structures such as strings, terms, graphs, or theorems. A production rule performs two operations: matching and rewriting. When a match is found the rule manipulates (i.e. rewrites) the structure. The rules together with the data structures form a graph along which data elements (e.g. packets) are processed. From the point of view of a data packet, computation is sequential. But this does not impose that a single packet is processed by all rules to termination before the next packet enters the system. It is possible to match each rule to the data in parallel (Gupta, Forgy, Newell, & Wedig, 1986), which is a huge benefit compared to procedural programs.

flow-based

Flow-based programming is a similar approach where a process is described as a flow graph in which basic tasks are connected together in a way that indicates the data dependencies. In contrast to rule-based systems, flow-based programs explicitly wire the flow graph. Flow-based programming goes back to Conway's coroutines (1963) and is still present in the Unix pipe system (McIlroy, 1964; The Open Group, 2008). Dennis (1980) proposed a data-flow-oriented computer architecture where multiple microprocessors are ready to process an operation as soon as the operands are available.

EXISTING RULE-BASED APPROACHES 19.1.2

expert systems

So far, rule-based systems were mostly used in *expert systems* (Giarratano & Riley, 2004), for example for VLSI routing (Joobbani & Siewiorek, 1985) or medical consultations (Shortliffe, 1976). However, the idea of using rewriting rules to structure communication in computer networks is not new: Mackert and Neumeier-Mackert (1987) proposed their *Communicating Rule Systems* (CRS) where distributed protocol entities are represented as rule systems that communicate via a set of connecting gates (see also Schneider, Mackert, Zörntlein, Velthuys, & Bär, 1992). Mackert's system was one of the original inspirations for Fraglets. Braden, Faber, and Handley (2002) proposed a "role-based architecture", which also uses a set of rules as an implementation basis. Dressler, Dietrich, German, and Krüger (2009) describe a rule-based system, inspired by cellular signaling, for programming self-organized sensor and actor networks.

Communicating Rule Systems

Recently, Weise (2009) demonstrated how to evolve distributed algorithms by his *Rule-Based Genetic Programming* (RGBP) approach (see also Weise, Zapf, & Geihs, 2007). He showed that the evolvability of protocols is better if the program representation is based on rules. In particular, a depar-

Rule-Based Genetic Programming

ture from the sequential execution order lowers epistasis, i.e. the dependency between code positions of a solution program.

19.1.3 FRAGLET RULES

Fraglets is a rule-based system. Reactions, induced by `match`-rules, recognize a corresponding passive fraglet in the multiset and perform some rewriting steps. In contrast to most rule-based systems, the matching mechanism of Fraglets is a simple exact match between a single symbol of the active fraglet (the rule) and the passive fraglet (the data). This enables a very efficient implementation of the matching part, which is usually the biggest performance problem of rule-based systems. For the subsequent rewriting steps, the programming model is that of a stack-based programming language, similar to Push (Spector et al., 2004). Since both the matching and rewriting code is represented in the same string format as data, it is quite natural to manipulate existing rules or generate new rules. Such self-reference is one of the key requirements for self-healing code.

19.2 MOBILE CODE

The flat code and data representation also allows Fraglets to send active code to distant reaction vessel via the `send`-instruction. Thus, a chemical program may use the principles of active networks. We already introduced the active network approach in Section 2.2. As mentioned there, by sending code along with the data, the remote note does not need to know how to interpret the data. This makes standardization beyond the definition of the virtual machine obsolete and leverages on-the-fly deployment of protocol code.

Active networks have been criticized for the additional overhead that is required when each packet carries its interpreting code. In Fraglets, we can decide on a case-to-case basis whether only data or both code and data shall be exchanged: The *Disperser* example, introduced in Section 5.4 and analyzed in Chapter 9, only sends passive data molecules, whereas the extended design variants, discussed in Section 10.4, broadcasts an active molecule that sends itself back to the origin. We also made use of active code deployment in the distributed software update example in Section 15.3, where we increased the system's robustness by letting the protocol continuously distribute its own code. Finally, in the multipath protocol example, the piggybacked rewards were active molecules that replicated the "good" forwarding rules (see Section 16.5). Thus, the active networking motif is not only used to quickly adapt the execution substrate to new protocols, but also to increase the robustness of a distributed system by continuously refreshing the protocol software.

DYNAMICS 19.3

If a traditional protocol such as TCP aims at controlling its own dynamic behavior it has to start and stop hardware-assisted timers and integrate these events into its state machine; large parts of the state machine have to deal with timing and race-conditions. The requirement to fully control each microscopic state change leads to a fundamentally *interval-based perspective* on dynamic processes, where the protocol has to manually compute the next time interval. Because the functional aspects of a protocol are often the prime and sole concern in the design phase, dynamics is treated as a side effect for later consideration or discovery.

interval-based perspective

In contrast, designers of CNPS can adopt the more natural *flow-based perspective* on communication processes and packet streams. Protocols are first designed on the *macroscopic* level where they are regarded as distributed chemical reaction networks. Designers are actually *forced* to care about the dynamic behavior, because reaction networks follow the law of mass action, a law that derives the dynamics from the structure of the reaction network. On the microscopic level, unlike traditional protocols, chemical protocols do no have to care about timing issues. Once the reaction network is designed, the stochastic reaction algorithm correctly schedules the program that spans this reaction network and the programmer never has to handle time intervals him/herself.

flow-based perspective

THE COMPUTER NETWORK AS DYNAMIC ENVIRONMENT 19.3.1

A packet-based communication network is a very fluctuating environment. The changes of the network characteristics are not only due to topological changes caused by administrative interventions but mainly due to the cross talk between different packet streams. That is, each packet sent over the network, changes the environment for other participants. This is usually considered to be a nuisance, but it has its benefits: Weise (2009) pointed out the positive effect of the network's dynamics while evolving distributed algorithms: One evolved solution of a leader election protocol exploited the changes in message latencies to break the symmetry of the state transitions and converge to a global solution.

A packet network is also an inherently competitive environment because it usually offers only best-effort delivery. Cooperation has to be enforced by explicit congestion control algorithms, for example. This is also true for CNPS where we recognized in Section 12.1.3 that a chemical replacement of the core transport services does not ensure fairness among different packet streams either. Therefore, we proposed C3A, a chemical variant of a congestion control algorithm that is even compatible and fair to TCP streams.

19.3.2 EMBODIMENT

While exploiting resources, CNPs at the same time explore and dynamically adapt to the environment. Our multipath routing protocol, for example, is able to obtain bandwidth information *because* it fills the link with packets; the Quines occupy the available memory but survive when reducing it. Hence, CNPs achieve a high degree of embodiment (Quick, Dautenhahn, Nehaniv, & Roberts, 2000): computation may be outsourced to the environment, a concept already employed in robotics (Pfeifer, Lungarella, & Iida, 2007).

19.4 STOCHASTIC EXECUTION VS. DETERMINISTIC STATE CHANGES

The shift towards mobile, dynamic, rule-based systems is not for free. For example, with rule-based software execution we lose a certain level of determinism, and rate-encoded information leads to a higher message complexity. This section shows the drawbacks and advantages of the intrinsic noise due to random execution whereas the next section focuses on the conflict between robustness and resource requirements.

stochastic execution model

The chemical execution model is *inherently stochastic* in two different aspects: First, once a reaction is executed, reactant instances are chosen randomly from the reactant species, and second, the reaction algorithm determines a random reaction interval.

19.4.1 DRAWBACKS OF INTRINSIC NOISE

higher packet loss

We realized that a randomized reaction interval leads to noisy packet rates and hence to a *higher packet loss* over bandwidth limited links. We therefore proposed a deterministic scheduler instead, which lowered the noise dramatically (see Section 12.2.2). The drawback of this approach is that the system can no longer be modeled as a Markov jump process. However, a stochastic analysis is only feasible for very small systems anyway. Another drawback is that the advantages of noise (see below) cannot be exploited with deterministic reaction intervals.

packet reordering

The other source of randomness, the random selection of reactants, is responsible for *reordering packets*. If chemical protocols are to be used in the core of the Internet, chemical species should exhibit FIFO and tail-drop behavior. Otherwise, the performance of TCP streams flowing through chemical reaction vessels would drop drastically. Although TCP's data streaming method is efficient, there may be other strategies to transmit documents over a network. A more chemically inspired vision is to consider a document as a set of loosely coupled molecules that may be sent individually and in arbitrary order to a remote node, where they re-gather. In this model, either form of randomness is tolerable. This can be further improved by adding a little redundancy through fountain codes, for example the raptor forward correction scheme (Luby, Shokrollahi, Watson, & Stockhammer, 2007), which allows a receiver to reconstruct the conveyed information from a subset of the received stream of code symbols.

ADVANTAGES OF INTRINSIC NOISE 19.4.2

Intrinsic random noise is an advantageous in many cases. In nature, cells are able to *control the amount of noise*: When undesired, noise may be suppressed for robust behavior, for example, by negative feedback (Ullah & Wolkenhauer, 2009b; El-Samad & Khammash, 2004; Paulsson & Ehrenberg, 2000). However, some noise may be desirable to maintain variability (Shibata & Ueda, 2008; Blomberg, 2006). This has also influenced the design of networking protocols: Several protocols use a randomly initialized timer to avoid regular traffic patterns leading to collisions. For example, the Address Resolution Protocol (ARP) (Plummer, 1982) waits for a random interval before probing whether an address is already in use. This *intentional jitter* "helps ensure that a large number of hosts powered on at the same time do not all send their initial probe packets simultaneously" (Cheshire, 2008). Another example is the *random exponential back-off* feature in Carrier Sense Multiple Access (CSMA) protocols such as Ethernet ("IEEE Standard 802.3," 2008), which wait a random time before sending the same data frame again after detecting a collision.

noise control

intentional jitter

random exponential back-off

Another protocol example where noise is exploited is the ARAS protocol by Leibnitz et al. (2006), which mimics *attractor selection* of biochemical switches. Noise is intentionally added to the system's behavior in order to let the protocol explore different routing paths. In Chapter 16, we demonstrated the same principle with our chemical multipath routing protocol. Stochastic noise is also desired for gossip protocols. They exploit the randomness to *break the symmetry* of the network topology in order to converge faster.

attractor selection

symmetry breaking

19.4.3 DETERMINISTIC APPROXIMATION OF STOCHASTIC CNPS

deterministic approximation

The advantage of the chemical model is that the stochastic dynamic behavior can be *approximated by deterministic* ODEs, which are automatically derived from the structure of the reaction network. We introduced this analysis method in Section 8.2 and used it extensively to study the behavior of our protocols (e.g. in Chapter 9, Section 11.3.1, Chapter 12, Sections 14.1, and 16.5). Compared to traditional state-machine based implementation, this simplifies the behavioral analysis.

19.5 ROBUSTNESS VS. RESOURCE REQUIREMENTS

Robustness always comes along with redundancy, and redundancy must be paid with resources. For chemical networking protocols, this means that robustness to packet loss comes at the price of a higher message complexity, and that robustness to unreliable execution requires more memory and computing power. In this section, we discuss the resource requirements of chemical networking protocols in terms of bandwidth, memory and computing power.

19.5.1 ROBUSTNESS OF CHEMICAL NETWORKING PROTOCOLS

We motivated the shift towards dynamical, more "analog" distributed algorithms by the inherent robustness of bio-chemical reaction systems to perturbations. Why this is the case is explained by Stepney (2010). She looked at computation from a dynamical systems point of view, in which case robustness can be seen as the ability of the system to return to the same fixed point when perturbed. Classical programs do not "recognize" small perturbations, meaning that a small perturbation leads the system into another fixed point, which computes another (and wrong) result. The reason for this is the small size of the basins of attraction that classical state-based systems exhibit.

large basin of attraction

The macroscopic solutions of CNPs are presented in a fixed point with a *larger basin of attraction*. A perturbation moves the system a small distance from the original state. In chemical reaction networks, stoichiometric rules force the system to move only in small steps within the neighborhood of the current state. Therefore a perturbation usually does not lead to the computation of a completely different result.

19.5.2 HIGHER MESSAGE COMPLEXITY

Pure chemical networking protocols such as *Disperser* (see Section 5.4) use the packet rate to convey information. The packet rate is only slightly disturbed by a single lost packet. But this robustness to packet loss is paid by a *higher message complexity*, which depends on the value being conveyed. In general, the bandwidth requirement of pure chemical protocols is higher than classical protocols, which encode their information symbolically.

higher message complexity

On the other hand, no additional bandwidth is required for hybrid chemical protocols. For continuously running algorithms that send packets anyway, such as routing or transport protocols, additional information can be conveyed in the packet *interval*. That is, rate-based information is modulated onto the packet stream, orthogonally to the symbolic payload. This is the operating principle of our C3A congestion control algorithm (see Section 12.3.2) as well as for the forwarding path reinforcement mechanism of the self-healing routing protocol (see Section 16.5). For such hybrid protocols that exchange representation-free information in addition to symbolic payload, no additional bandwidth is required.

MEMORY REQUIREMENT: COMPACT VS. REDUNDANT STATE INFORMATION AND CODE 19.5.3

Chemical protocols do not require more memory *per se* because the protocol logic can usually be expressed in a *compact* way: Compared to traditional programs, they outsource computation to the dynamics of the reaction network. For example, chemical programs do not have to care about setting timers or gathering and processing statistical information about the network. This comes for free by interacting with the network through chemical reactions and thus by being embodied into the network.

compact program representation

But despite their compactness, chemical programs usually require *more memory* because information is encoded in a macro-state by the *multiplicity of molecules*. Again, this is not an issue for hybrid protocols where the vessel hosts multiple similar packets anyway.

redundant state representation

To obtain self-healing software, next to redundant state information, we have to keep redundant copies of code. In Chapter 14, we demonstrated that at least 20 replicas of each code molecule are required to achieve a sufficient level of robustness and determinism.

COMPUTING POWER REQUIREMENTS 19.5.4

The computing power requirements of CNPs depend on the momentary reaction rates and the complexity of the microscopic transformation that each reaction triggers. Thus, for pure chemical protocols such as *Disperser*, the required computing power increases with the value stored and conveyed.

Self-healing programs also require more computing power because every operation is implemented as a Quine. For each processed packet, a data-processing Quine needs seven additional instruction steps to also replicate itself. That is, a self-healing program is expected to consume seven times the computing power of comparable persistent code. Such a constant factor scales well with the complexity of programs and is usually tolerable.

19.6 SUMMARY

In this chapter, we showed how the chemical paradigm pushes communication protocols towards (a) mobile code, (b) rule-based and (c) dynamic execution, and we discussed several other approaches along these three axes. We were surprised by the small number of existing concepts to master the dynamic behavior of protocols, particularly with regard to the complexity problem of the Internet.

There are other dimensions that open up at the top-right-back corner of the cube in Figure 19.1. One additional dimension is evolvability: If future hardware will be more unreliable, variation of code is for granted. CNPS together with the limited resources available can be used to support program evolution. In fact, an intrinsically self-healing system, as the one that we introduced, will become an evolving system. On the other hand, evolvability requires a certain level of robustness (A. Wagner, 2007). The investigations carried out in the third part of this thesis may therefore also be helpful to ultimately find robust evolving protocols.

CHAPTER 20

Relevance and Future Directions

On a possible development from static towards dynamic, flow-based networking and how the chemical metaphor may act as a signpost.

> I believe in intuition and inspiration. Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution. [20]
>
> Albert Einstein

Shall the future Internet be controllable or autonomic? This seems to be a major conflict as automatisms usually lead to complex unpredictable phenomena. In this chapter, we argue that both is possible if we take a step back and look at the large-scale behavior of macroscopic flows instead of trying to predict microscopic packet interactions.

The tension between predictability and automation arises from the different needs of Internet users and operators. If we ask the users about their demands, they require reliable and deterministic services to exchange, store and process their data. Network operation engineers have another view. Overwhelmed by the numerous changes in the network and suffering from the effort required to manually monitor and optimize the network, they ask for new tools to automate network management and to delegate resource allocation to the system itself. Their fear, however, is that autonomic processes will not be controllable anymore.

On the long run, we cannot avoid to automate certain networking tasks, as we currently observe the end of feasible micro-management. The same happens on the formal side: Computer scientists recognized that deterministic proofs of functional protocol aspects are not possible for complex distributed phenomena and started to develop probabilistic approaches. We need a similar paradigm shift as in the physics of the 18th and 19th century where the theory of classical object-centric mechanics had to be extended by a wave-based concept to capture phenomena like electrodynamics and optics. We conjecture that a similar paradigm shift in networking allows us to develop better solutions for the future Internet. We have to move one step back, disregard the individual packets, and rather focus on the dynamical aspects of packet flows in order to be able to come up with new predictions on this macro-level.

In this chapter, we sketch a possible path towards a future Internet where not only the exchange of data (Section 20.1), but also their storage (Section 20.2) and processing (Section 20.3) are treated as predictable flows. That is, we are favoring enriched network services that go beyond data transfer. We argue that despite the loss of predictability on the packet-level, large-scale guarantees on the flow-level are still possible thanks to the chemical metaphor and sound engineering practice.

20.1 FLOW-BASED FORWARDING

With the congestion collapse in the 1980ies, network engineers became aware that the dynamics of protocols essentially contribute to the success or failure of a network. This led to a set of congestion control algorithms to which we also added a chemical variant in Section 12.3. It seems that the choice of an end-to-end congestion control algorithm is global and permanent. The incompatibility between the TCP Reno and Vegas variants (Ahn, Danzig, Liu, & Yan, 1995) has showed that coexistence between congestion algorithms is not guaranteed, which makes a further development difficult.

We conjecture that the (a) mobile, (b) rule-based, and (c) dynamic approach of chemical protocols will help to develop new and fair forwarding and traffic shaping algorithms. (a) The active networking approach allows installing forwarding code on the fly to the network's core. Note that there is – unlike in the Internet – no default forwarding machinery in CNPs. We have seen in Section 12.1.2 that chemical protocols can be used to dampen bursts in the network. Similarly, CNPs can be used to actively block, shape and regulate packet streams in the forwarding engine. (b) The rule-based approach enables the coexistence of competing or cooperating solutions. Instead of finding the optimal congestion control algorithm for the whole Internet, each user or operator may install its own solution, which optimizes throughput and fairness in the local environment. (c) The tight relation between the structure of the reaction network in some network region and the flow dynamics enables predictions about the behavior of coupled flows and flow systems.

Such a flow-centric view requires more than ever that the dynamic processes are understood and that the influences among the algorithms can be designed, and their interaction studied. In this thesis, we have focused on (chemical) forwarding, but in the future we should also consider including storage and processing.

FLOW-BASED STORAGE 20.2

Storage usually comes with the notion of stiffness and persistence. In CNPS, storage (in a vessel) is natural, but also ephemeral. This makes sense: A private collection of photographs, for example, is a continuous stream of new snapshots. Another example is the stream of scientific documents: Researchers carry out new experiments, publish new results, ideas and insights. These documents have to be redundantly stored and made accessible to other researchers. One burning issue is how to organize long-time archiving. Today, it has become virtually impossible to manually manage our own photo collection or to decide what should be preserved for the future.

We also have to take a step back from the individual objects and see databases as a temporal flow of documents. Each single document will have a certain half-life period. When running out of storage, it is inevitable to delete some artifacts. Perhaps it is sufficient to keep only some pictures from our holidays twenty years ago, or the empirical details of an ancient scientific experiment can be deleted, but not the aggregated results and assessments.

We think that a chemical view to such a *temporal stream of documents* opens interesting approaches. As for transport processes, it is important to understand the flow dynamics of such streams. Document streams intend to exploit resources, but as memory is limited, a certain level of fairness must be ensured among those flows, which brings us back to congestion control in shape space.

FIRST STEPS — CONTENT-CENTRIC NETWORKS 20.2.1

Recently, Van Jacobson started to promote his content-centric view of networking (Jacobson, 2006; Jacobson et al., 2009), which is a first step towards the unification of storage and transmission processes. Rather than addressing a document by the address of its host, documents are mobile and have their own unique name. This approach blurs the clear distinction between data storage and data delivery, as documents may relocate to the node where they are being used. We think that such a data-centric view is an important first step towards self-maintaining documents.

20.2.2 A CHEMICAL APPROACH TO ORGANIZE DISTRIBUTED STORAGE

If we consider documents or parts of the documents as molecules, we could use an artificial chemistry to organize their links by bonds, express their conversion into another format by chemical reactions, and control their location by equilibria stemming from simple interaction rules.

For example, the storage location of documents could be determined by an artificial chemistry. Repulsion forces ensure that identical replicas of a document are stored spatially distributed whereas attraction forces pull a document near to where it is used. If memory runs short in one node, documents are automatically pushed to neighbor nodes. If the pressure of novel documents is too large, some documents even have to be deleted.

Such a system is subject to a continuous influx of new documents leading to an automatic reorganization according to simple rules. Our hope is that we can transpose the insights gained from controlling transport flows to documents flows and provide a similar level or analyzability that helps to predict the behavior of such complex dynamic systems.

20.3 FLOW-BASED PROCESSING

Another problem of today's information society and knowledge economy is the huge amount of data available at our fingertips with a myriad of ways of extracting information.

If we project our flow-based vision even further, a next consequent step would be to automatically process and aggregate the content of documents. For example, photographs could be grouped according to the pictured persons, or related research information could be grouped or even summarized in a continuous stream. Thus, based on existing artifacts, human-assisted or fully automatic agents continuously produce *new* information-, idea-, and insight-flows.

Note that software agents are themselves streams, namely an endless sequence of software versions, updated through more or less regular software patches. Overall, this leads to a system with open-ended development, which is constantly fed with new findings or annotations, and information extraction logic, able to produce aggregated knowledge and leading to a higher-level organization of documents and traffic flows. As a virtual image of today's research process, such an automated or assisted system may assists research and private life.

In such a vision of process flows beyond physical boundaries, privacy and trust are of utter importance. But trust itself is also dynamic. Our trust to other persons, institutions, and ideas constantly change during our life. Thus, security mechanisms may also be regarded as flows of processes that interact with us and with the data they protect.

In such a highly dynamic environment, where everything is volatile *per se* and nothing persists forever, it is more than ever important to understand the dynamic processes. We don't believe that our methods presented in this thesis can be applied directly to these future complex systems. But we believe that for things to endure they have to be dynamic and – at the end – are flows.

> Nur was sich ändert bleibt bestehen! [21]
>
> *Sprichwort*

CHAPTER 21

Conclusions

What one concludes to see
depends on the chosen model
of reality. [22]

H. Dieter Zeh

Stimulated by the latest advances in systems biology and artificial life, we ventured on a new approach to organize packet processing in computer networks in a chemically inspired way. At the center of our approach is a novel packet-scheduling algorithm stemming from chemical reaction kinetics that treats packets as virtual molecules and intentionally defers packet-processing time. Starting from the simple principles of rule-based, dynamic packet processing, paired with mobile code, we developed a theory of chemical networking protocols (CNPs) that helps network engineers design robust protocols easily and analyze them thoroughly. We recapitulate and assess our three major findings as follows:

1. In a distributed system that runs within a chemical scheduling regime, formal analysis of the dynamics of packet flows on the macroscopic level becomes feasible. The behavior of chemical networking protocols is described by stochastic processes or – as an approximation – by a deterministic mathematical model based on ordinary differential equations. This enables and simplifies **formal analysis of protocol dynamics**. There are a vast number of analysis tools, originally developed for natural chemical reaction systems, which we adopted to CNPs. While traditionally, only the packet flows have been modeled as stochastic processes (e.g. in queuing theory), we also apply and impose

this to the packet processing. On the analysis side, this does not change much: the same mathematical framework can be applied, but thanks to the chemical execution engine, we are now able to derive a mathematical description of the protocol's dynamics directly from the chemical program. We provided chemical models for basic network components, such as simple queues and links, and demonstrated the power of the analysis methods by proving the convergence of a gossip-based aggregation protocol (*Disperser*), the fairness of our chemical congestion control algorithm (C3A), and the stable equilibrium solution of a link load-balancing algorithm. Proofs for a protocol's dynamics are rare in networking, so we are pleased that ours turned out to be quite elegant.

2. Our environment forces protocol designers to come up with "fluid" protocols that continuously track the environment and adapt to it. This might seem to be a challenging task. But, in addition to a sound mathematical foundation, a chemical networking approach potentially promotes good system-wide properties when composing such protocols. To support the development process, we came up with **design methods** to engineer CNPs: In contrast to traditional protocol design, the dynamic behavior of CNPs is specified and analyzed before the microscopic aspects are fixed. The reaction network is synthesized in a bottom-up approach from simple reaction motifs that exhibit a dynamic behavior that is well understood. We offered a couple of useful motifs for distributed arithmetic computation, source routing, traffic shaping, neighbor discovery, anycast, gossiping, and link load balancing. These motifs are combined in the reaction graph, which serves as a visual design instrument that makes the dynamic behavior of protocols intuitively graspable and communicable to other engineers. We were surprised how easy and quickly new protocol ideas came to live after having had some experience with the chemical mindset.

3. This thesis also addressed the question whether protocols are able to offer **dynamic robustness**, when faced with competition, and **code stability**, when faced with resource constraints or deletion attacks. We departed from explicit rate computation and relied on the embodiment of key information in the reaction rates. Our chemical congestion control algorithm (C3A) is able to mimic TCP, or said differently, TCP is a chemical protocol in disguise! Code stability is obtained through continuous rewriting and code replication. We showed that such protocol software recovers from the loss of major code parts, and we were even able to quantify the robustness by computing the mean survival time. An open problem is that on an unreliable execution machinery, a small

> percentage of errors lead to self-replicating spirals that quickly consume all resources. We sketched a possible solution based on an energy conservation scheme paired with multi-level selection. Whether the automatic synthesis of protocols, or even their evolution, is desirable, needed or just inevitable also remains open. From our point of view, CNPS are an attractive and well-formalizable basis for such an endeavor.

In this work we examined how chemical concepts can be transposed to network design in order to obtain the same provable emergent properties that we find in chemical systems such as stable equilibria. This requires a paradigmatic shift from designing local state machines to weaving global reaction networks. We hope that our work will contribute to a robust and future-proof Internet.

Appendices

APPENDIX A

Supplementary Material

This appendix contains supplementary material that did not fit in the main part.

A.1 Limiting the Vessel Capacity to Restrict the Reaction Rate

In the following, we compute and compare the maximum reaction rate demanded by the law of mass action to the maximum reaction rate the CPU is able to deliver. This is additional material to Section 11.3.2. We provide an equation for the dimensioning of the reaction vessel for a worst-case scenario. This result is only valid for bimolecular reactions. We base this calculation on the following parameters that characterize our computing infrastructure:

T: denotes the CPU time required to execute one Fraglets production rule, either a reaction or a transformation.

C: denotes the symbol capacity of the reaction vessel, which is limited by the memory available and, based on our calculations, should be further restricted to let the CPU cope with the law of mass action requirements.

Let us first calculate the maximum reaction rate required by the law of mass action ($r_{\max,\text{LOMA}}$) and the maximum reaction rate the CPU is able to achieve ($r_{\max,\text{CPU}}$). Then we have to make sure that at any time

$$r_{\max,\text{LOMA}} < r_{\max,\text{CPU}} \tag{A.1}$$

A.1.1 WORST-CASE SCENARIO FOR ARBITRARY BIMOLECULAR REACTIONS

(a) *Maximum Reaction Rate Required by the Law of Mass Action*

In general, the symbol capacity C of a saturated vessel is distributed over N_s instances of length $|s|$ of species $s \in \mathcal{S}$:

$$C = \sum_{s \in \mathcal{S}} N_s \, |s| \tag{A.2}$$

The rates of all reactions $r \in \mathcal{R}$ are given by the propensity function (see (5.4) on page 56)

$$a_r\big(\mathbf{N}(t)\big) = k_r \prod_{\{s \in \mathcal{S}\}} N_s(t)^{\alpha_{s,r}} \tag{A.3}$$

The maximum reaction rate required by the law of mass action is the maximum value returned by one of the propensity functions:

$$r_{\max,\text{LOMA}} = \sup \left\{ a_r(\mathbf{n}) : r \in \mathcal{R} \right\} \tag{A.4}$$

Since we are interested in the worst-case behavior, we are looking for the vessel composition that yields the maximum reaction rate. For bimolecular reactions the fastest rate can be achieved if the molecules are evenly distributed between the two reactants of the fastest reaction:

$$r_{\max,\text{LOMA}} = k_{\max} N_{s_1} N_{s_2} = k_{\max} \frac{N}{2} \frac{N}{2} = k_{\max} \frac{N^2}{4} \tag{A.5}$$

In this case, the vessel only contains two species, which allows us to compute their cumulative length based on (A.2) as

$$|s_1| + |s_2| = 2 \frac{C}{N} \tag{A.6}$$

(b) *Maximum Reaction Rate Obtainable from the CPU*

Next, we compute the maximum reaction rate the CPU is able to cope with. As shown previously in Section 5.2, a Fraglets reaction first concatenates two molecules and then executes the transformation rules in the product's head until the fraglet is in its normal form. The worst-case scenario in terms of required CPU time is that the product of the reaction only contains transformation instructions, which must be executed to completion. The maximum processing time required is thus

$$T_{\text{max,CPU}}(s_1, s_2) = \overbrace{T}^{\text{reaction}} + \overbrace{(|s_1| + |s_2| - 3)\, T}^{\text{transformations}} \tag{A.7}$$

We previously recognized that only two species are present in the worst-case scenario, and we also computed their length in A.6. Hence, the maximum reaction rate yields

$$r_{\text{max,CPU}} = \frac{1}{T_{\text{max,CPU}}(s_1, s_2)} = \frac{N}{2T\,(C - N)} \tag{A.8}$$

Derived Restriction of the Vessel Capacity (c)

As stated in (A.1) the worst-case reaction rate required by the law of mass action must never exceed the rate the CPU is able to perform reactions:

$$\overbrace{k_{\text{max}} \frac{N^2}{4}}^{r_{\text{max,LOMA}}} < \overbrace{\frac{N}{2T\,(C - N)}}^{r_{\text{max,CPU}}} \tag{A.9}$$

We resolve this equation with respect to the vessel capacity C and obtain a new restriction of

$$C < \frac{1}{T k_{\text{max}} N} + N \tag{A.10}$$

This equation still depends on the number of molecule instances N. This parameter defines whether the symbol capacity of the vessel is distributed to many short molecules or a few long molecules. The above equation has a minimum at $N = \sqrt{2/(T k_{\text{max}})}$, and hence the vessel capacity shall be restricted to

$$C < \left(\frac{2}{T k_{\text{max}}}\right)^{-3/2} + \sqrt{\frac{2}{T k_{\text{max}}}} \tag{A.11}$$

BEST-CASE SCENARIO FOR UNIMOLECULAR REACTIONS (DISPERSER) A.1.2

Although the *Disperser* protocol (see Section 5.4 and Chapter 9) is based on bimolecular reactions, one reactant is always a catalyst (C), which does not change its quantity. The reaction network can therefore be approximated by unimolecular reactions among the data molecules X. Here, we assume that each reaction vessel only contains N_X data molecules of size $|X| = 1$ symbol. This means that the vessel capacity in terms of symbols is equal to the vessel capacity in terms of molecules:

$$C = \sum_{s \in \mathcal{S}} N_s \, |s| = N_X \tag{A.12}$$

Thus, each network node v_i has to trigger $\deg(v_i)$ reactions with a total reaction rate of

$$r_{\max,\text{LOMA}} = \deg(v_i)\, k_{\max} C \tag{A.13}$$

Each reaction additionally executes a `send`-transformation. Thus, the maximum reaction rate the CPU can cope with is

$$r_{\max,\text{CPU}}(s_1, s_2) = \frac{1}{2Tk_{\max}} \tag{A.14}$$

In order to be sure that the CPU is able to deliver the reaction rate required by the model, the capacity has to be limited to

$$C < \frac{1}{2Tk_{\max}} \tag{A.15}$$

A.2 VERSIONED QUINE

This section provides a detailed description of the Fraglets implementation of the versioned Quine, which we used to realize autonomic and self-healing software updates. The basic idea behind this Quine is described in Section 15.3.

In Fraglets, this reaction network is implemented by appending a version number to the blueprint.

$$\text{B}_{vi} = [\texttt{B} \;\; \ldots \;\; i] \tag{A.16}$$

The reward verifies whether its own version number is less than the version number of the blueprint. If yes, the product is destroyed using a `nul`-instruction, otherwise the replication process is engaged. The detailed Fraglets rewriting steps after a reaction among a reward $\mathrm{R}_{\mathrm{v}i}$ and a blueprint $\mathrm{B}_{\mathrm{v}j}$ are shown below:

$$\overbrace{\texttt{[match B sdup dummy ...]}}^{\mathrm{R}_{\mathrm{v}i}} + \overbrace{\texttt{[B ... }j\texttt{]}}^{\mathrm{B}_{\mathrm{v}j}} \tag{A.17a}$$

$$\Rightarrow \texttt{[sdup dummy spush }i\texttt{ ... }j\texttt{]} \tag{A.17b}$$

$$\Rightarrow \texttt{[spush }i\texttt{ slt ... }j\ j\texttt{]} \tag{A.17c}$$

$$\Rightarrow \texttt{[slt sif nul mfork ... }j\ j\ i\texttt{]} \tag{A.17d}$$

For $j < i$ (new reward reacts with an old blueprint) the product destroys itself.

$$\Rightarrow \texttt{[sif nul mfork ... }j\texttt{ 1]} \tag{A.18a}$$

$$\Rightarrow \texttt{[nul ... }j\texttt{]} \tag{A.18b}$$

$$\Rightarrow \texttt{[]} \tag{A.18c}$$

Otherwise, for $j \geq i$ (reward reacts with blueprint of same or newer generation) the replication continues.

$$\Rightarrow \texttt{[sif nul mfork ... }j\texttt{ 0]} \tag{A.19a}$$

$$\Rightarrow \texttt{[mfork 2 send any nop nop fork ... }j\texttt{]} \tag{A.19b}$$

$$\Rightarrow \underbrace{\texttt{[send any fork ... }j\texttt{]}}_{\text{remote replica}} + \underbrace{\texttt{[nop nop fork ... }j\texttt{]}}_{\text{local replica}} \tag{A.19c}$$

$$\underbrace{\texttt{[nop nop fork ... }j\texttt{]}}_{\text{local replica}}$$

$$\Rightarrow \texttt{[fork sdel B ... }j\texttt{]} \tag{A.19d}$$

$$\Rightarrow \texttt{[sdel ... }j\texttt{]} + \texttt{[B ... }j\texttt{]} \tag{A.19e}$$

$$\Rightarrow \underbrace{\texttt{[...]}}_{\mathrm{A}_{\mathrm{v}j}} + \underbrace{\texttt{[B ... }j\texttt{]}}_{\mathrm{B}_{\mathrm{v}j}} \tag{A.19f}$$

APPENDIX B

Fraglets Instruction Set

This appendix lists the complete Fraglets instruction set provided by Fraglets version 0.5.1. Section B.1 introduces a meta expression notation, which is used to describe the instruction set. In Section B.2, we show the general format of *immediate*- an *stack*-instructions. Finally, Section B.3 lists all Fraglets instructions grouped by their function. Further information about the Fraglets programming language can be found in Section 5.2.

Fraglets meta expressions B.1

Fraglet production rules can be described by regular expression substitution rules. For example, the `exch` rule that swaps two symbols is represented by

$$\texttt{[exch (.) (.) (.) (.*)]} \Rightarrow \texttt{[\textbackslash 1 \textbackslash 3 \textbackslash 2 \textbackslash 4]} \tag{B.1a}$$

Instead of using this traditional notation, we prefer to use Greek letters as placeholders for substrings in the matching part (left-hand side) and reuse the same letter in the substitution part (right-hand side). The same production rule is written as

$$\texttt{[exch}\ \sigma\ \alpha\ \beta\ \Phi\texttt{]} \Rightarrow \texttt{[}\sigma\ \beta\ \alpha\ \Phi\texttt{]} \tag{B.1b}$$

We use the following placeholders:

$\alpha, \beta, \ldots, \omega$: Greek lowercase letters denote symbols of the finite alphabet Σ: $\alpha, \beta \in \Sigma$,

$\Gamma, \Delta, \ldots, \Omega$: Greek uppercase letters denote possibly empty symbol strings of arbitrary length over the alphabet Σ: $\Phi, \Psi \in \Sigma^*$,

$\Gamma^l, \Delta^l, \ldots, \Omega^l$: Superscripted Greek uppercase letters denote symbol strings of length l over the alphabet Σ: $\Phi^l, \Psi^l \in \Sigma^l$, $|\Phi^l| = |\Psi^l| = l$

$\Gamma_{\mathcal{X}}, \Delta_{\mathcal{X}}, \ldots, \Omega_{\mathcal{X}}$: Subscripted Greek uppercase letters denote possibly empty symbol strings of arbitrary length that must not contain symbols from the exclusion set $\mathcal{X}$: $\Phi_{\mathcal{X}}, \Psi_{\mathcal{X}} \in \left\{\Sigma \backslash \mathcal{X}\right\}^*$

$\Gamma^l_{\mathcal{X}}, \Delta^l_{\mathcal{X}}, \ldots, \Omega^l_{\mathcal{X}}$: The previous two notations can be combined to denote possibly empty symbol strings of length l that do not contain symbols from the exclusion set $\mathcal{X}$: $\Phi^l_{\mathcal{X}}, \Psi^l_{\mathcal{X}} \in \left\{\Sigma \backslash \mathcal{X}\right\}^l$, $|\Phi^l_{\mathcal{X}}| = |\Psi^l_{\mathcal{X}}| = l$

v_i, v_j : Network node identifiers: $v_i, v_j \in \mathcal{V}$.

In the following instruction tables, we typeset the instruction identifiers in blue, the operand symbols in red, and the result symbols in green.

B.2 GENERAL FORMAT OF IMMEDIATE AND STACK INSTRUCTIONS

Most instructions come in two different variants: an immediate instruction that uses operands embedded in the code (head of the fraglet) whereas the corresponding stack instruction accesses its operands in the tail of the fraglet. The general format for immediate instructions is

$$[\underline{\texttt{iinstr}}\ \sigma\ \alpha\ \ldots\ \omega\ \Phi] \Rightarrow [\sigma\ \mathrm{op}(\alpha\ \ldots\ \omega)\ \Phi] \tag{B.2a}$$

That is, the instruction `iinstr` uses the 3rd and further symbols as operands, keeps the 2nd symbol as continuation pointer (usually a tag) and replaces the operands with the result of the operation. The general format for stack instructions is

$$[\underline{\texttt{sinstr}}\ \Phi\ \omega\ \ldots\ \alpha] \Rightarrow [\Phi\ \mathrm{op}(\alpha\ \ldots\ \omega)] \tag{B.2b}$$

The stack instruction `sinstr` uses the last and previous symbols as operands and replaces them with the result of the operation. No special continuation symbol is required: The first symbol of the string Φ determines the next production rule applied to the product fraglet.

An underlined symbol denotes a structurally active symbol (see Section 7.4 for details). In the following we list both, the immediate and the stack variant next to each other if both exist.

FRAGLETS INSTRUCTIONS B.3

In this section, we list all Fraglets instructions, grouped by their function.

SYNCHRONIZATION INSTRUCTIONS B.3.1

Synchronization instructions concatenate two or more fraglets and are scheduled according to the law of mass action (see Section 5.2.3).

Immediate Instructions (a)

Tag	Production Rule
match	$[\underline{\texttt{match}}\ \underline{\sigma}\ \ \Phi] + [\underline{\sigma}\ \ \Psi] \Rightarrow [\Phi\ \ \Psi]$
matchp	$[\underline{\texttt{matchp}}\ \underline{\sigma}\ \ \Phi] + [\underline{\sigma}\ \ \Psi] \Rightarrow [\underline{\texttt{matchp}}\ \underline{\sigma}\ \ \Phi] + [\Phi\ \ \Psi]$
matchs	$[\underline{\texttt{matchs}}\ \underline{\sigma}\ \ \Phi] + [\underline{\sigma}\ \ \Psi] \Rightarrow [\underline{\sigma}\ \ \Psi] + [\Phi\ \ \Psi]$
matchps	$[\underline{\texttt{matchps}}\ \underline{\sigma}\ \ \Phi] + [\underline{\sigma}\ \ \Psi]$ $\Rightarrow [\underline{\texttt{matchps}}\ \underline{\sigma}\ \ \Phi] + [\underline{\sigma}\ \ \Psi] + [\Phi\ \ \Psi]$
mmatch	$[\underline{\texttt{mmatch}}\ \underline{\alpha}\ \underline{\sigma_1}\ \ldots\ \underline{\sigma_\alpha}\ \ \Phi] + [\underline{\sigma_1}\ \ \Psi_1] + \cdots + [\underline{\sigma_\alpha}\ \ \Psi_\alpha]$ $\Rightarrow [\Phi\ \ \Psi_1\ \ldots\ \Psi_\alpha]$
mmatchp	$[\underline{\texttt{mmatchp}}\ \underline{\alpha}\ \underline{\sigma_1}\ \ldots\ \underline{\sigma_\alpha}\ \ \Phi] + [\underline{\sigma_1}\ \ \Psi_1] + \cdots + [\underline{\sigma_\alpha}\ \ \Psi_\alpha]$ $\Rightarrow [\underline{\texttt{mmatchp}}\ \underline{\alpha}\ \underline{\sigma_1}\ \ldots\ \underline{\sigma_\alpha}\ \ \Phi] + [\underline{\sigma_1}\ \ \Psi_1] + \cdots + [\underline{\sigma_\alpha}\ \ \Psi_\alpha]$ $+[\Phi\ \ \Psi_1\ \ldots\ \Psi_\alpha]$
mmatchs	$[\underline{\texttt{mmatchs}}\ \underline{\alpha}\ \underline{\sigma_1}\ \ldots\ \underline{\sigma_\alpha}\ \ \Phi] + [\underline{\sigma_1}\ \ \Psi_1] + \cdots + [\underline{\sigma_\alpha}\ \ \Psi_\alpha]$ $\Rightarrow [\underline{\sigma_1}\ \ \Psi_1] + \cdots + [\underline{\sigma_\alpha}\ \ \Psi_\alpha] + [\Phi\ \ \Psi_1\ \ldots\ \Psi_\alpha]$
mmatchps	$[\underline{\texttt{mmatchps}}\ \underline{\alpha}\ \underline{\sigma_1}\ \ldots\ \underline{\sigma_\alpha}\ \ \Phi] + [\underline{\sigma_1}\ \ \Psi_1] + \cdots + [\underline{\sigma_\alpha}\ \ \Psi_\alpha]$ $\Rightarrow [\underline{\texttt{mmatchps}}\ \underline{\alpha}\ \underline{\sigma_1}\ \ldots\ \underline{\sigma_\alpha}\ \ \Phi] + [\underline{\sigma_1}\ \ \Psi_1] + \cdots + [\underline{\sigma_\alpha}\ \ \Psi_\alpha]$ $+[\underline{\sigma_1}\ \ \Psi_1] + \cdots + [\underline{\sigma_\alpha}\ \ \Psi_\alpha] + [\Phi\ \ \Psi_1\ \ldots\ \Psi_\alpha]$

Examples (b)

$$[\texttt{mmatch 2 X Y a}] + [\texttt{X b}] + [\texttt{Y c}] \longrightarrow [\texttt{a b c}] \tag{B.3a}$$

TRIVIAL INSTRUCTIONS B.3.2

The `nop`-instruction consumes no operands and performs no operation. The `nul`-instruction removes the current fraglet from the vessel. Since both instructions consume no operands there are no stack variants.

(a) Immediate Instructions

Tag	Production Rule
nop, n	$[\texttt{nop}\ \ \Phi] \Rightarrow [\Phi]$
nul	$[\texttt{nul}\ \ \Phi] \Rightarrow \varnothing$ (fraglet is removed)

(b) Examples

$$[\texttt{nop match x ssum y 1}] \longrightarrow [\texttt{match x ssum y 1}] \tag{B.4a}$$

$$[\texttt{nul match x ssum y 1}] \longrightarrow \varnothing \tag{B.4b}$$

B.3.3 SIMPLE SYMBOL SHUFFLING INSTRUCTIONS

The exch-instruction swaps two symbols, the dup-instruction duplicates a symbol and destroys another one to make sure the product is shorter than the reactant. The del-instruction deletes a symbol.

(a) Immediate Instructions

Tag	Production Rule
exch, e	$[\texttt{exch}\ \ \sigma\ \ \alpha\ \ \beta\ \ \Phi] \Rightarrow [\sigma\ \ \beta\ \ \alpha\ \ \Phi]$
dup	$[\texttt{dup}\ \ \sigma\ \ \chi\ \ \alpha\ \ \Phi] \Rightarrow [\sigma\ \ \alpha\ \ \alpha\ \ \Phi]$
del	$[\texttt{del}\ \ \sigma\ \ \chi\ \ \Phi] \Rightarrow [\sigma\ \ \Phi]$

(b) Stack Instructions

Tag	Production Rule
sexch	$[\texttt{sexch}\ \ \Phi\ \ \beta\ \ \alpha] \Rightarrow [\Phi\ \ \alpha\ \ \beta]$
sdup	$[\texttt{sdup}\ \ \chi\ \ \Phi\ \ \alpha] \Rightarrow [\Phi\ \ \alpha\ \ \alpha]$
sdel	$[\texttt{sdel}\ \ \Phi\ \ \chi] \Rightarrow [\Phi]$

(c) Examples

$$[\texttt{sexch a b c d}] \longrightarrow [\texttt{a b d c}] \tag{B.5a}$$

$$[\texttt{sdup x a b c d}] \longrightarrow [\texttt{a b c d d}] \tag{B.5b}$$

Complex Symbol Reordering Instructions B.3.4

The rot-instruction exchanges the head and the tail at a given separation symbol. The repl-instruction replaces all occurrences of a given symbol. The rev-instruction reverses the order of all symbols whereas the cross-instruction moves all symbols at odd positions before the symbols at even positions.

Immediate Instructions (a)

Tag	Production Rule
rot	$[\texttt{rot}\ \sigma\ \underline{\alpha}\ \Phi_{\{\alpha\}}\ \underline{\alpha}\ \Psi] \Rightarrow [\sigma\ \Psi\ \Phi_{\{\alpha\}}]$
repl	$[\texttt{repl}\ \sigma\ \alpha\ \beta\ \Phi_{\{\alpha\}}\ \alpha\ \ldots\ \Psi_{\{\alpha\}}]$ $\Rightarrow [\sigma\ \Phi_{\{\alpha\}}\ \beta\ \ldots\ \Psi_{\{\alpha\}}]$
rev, r	$[\texttt{rev}\ \alpha\ \beta\ \ldots\omega] \Rightarrow [\omega\ \ldots\ \beta\ \alpha]$
cross, c	$[\texttt{cross}\ \alpha\ \beta\ \gamma\ \delta\ \ldots\ \pi\ \omega]$ $\Rightarrow [\alpha\ \gamma\ \ldots\ \pi\ \beta\ \delta\ \ldots\ \omega]$

Stack Instructions (b)

Tag	Production Rule
srot	$[\texttt{srot}\ \Phi_{\{\alpha\}}\ \underline{\alpha}\ \Psi\ \underline{\alpha}] \Rightarrow [\Psi\ \Phi_{\{\alpha\}}]$
srepl	$[\texttt{srepl}\ \Phi_{\{\alpha\}}\ \alpha\ \ldots\ \Psi_{\{\alpha\}}\ \beta\ \alpha]$ $\Rightarrow [\Phi_{\{\alpha\}}\ \beta\ \ldots\ \Psi_{\{\alpha\}}]$

Examples (c)

$$[\texttt{rot a X b c d X e f g}] \longrightarrow [\texttt{e f g b c d}] \tag{B.6a}$$

$$[\texttt{srepl a b X c d X e f Y X}] \longrightarrow [\texttt{a b Y c d Y e f}] \tag{B.6b}$$

$$[\texttt{rev a b c d e}] \longrightarrow [\texttt{e d c b a}] \tag{B.6c}$$

$$[\texttt{cross a b c d e}] \longrightarrow [\texttt{a c e b d}] \tag{B.6d}$$

Move Between Code and Data Stack B.3.5

The push/spush-instructions consume the first operand from the code section (head of the fraglet) and push it to the data stack (tail of the fraglet). The pop/spop-instructions do the opposite.

(a) Immediate Instructions

Tag	Production Rule
push	$[\underline{\texttt{push}}\ \sigma\ \alpha\ \Phi] \Rightarrow [\sigma\ \Phi\ \alpha]$
pop	$[\underline{\texttt{pop}}\ \sigma\ \Phi\ \alpha] \Rightarrow [\sigma\ \alpha\ \Phi]$

(b) Stack Instructions

Tag	Production Rule
spush, u	$[\underline{\texttt{spush}}\ \alpha\ \Phi] \Rightarrow [\Phi\ \alpha]$
spop, o	$[\underline{\texttt{spop}}\ \Phi\ \alpha] \Rightarrow [\alpha\ \Phi]$

(c) Examples

$$[\texttt{spush 1 ssum y 5}] \longrightarrow [\texttt{ssum y 5 1}] \tag{B.7a}$$

$$[\texttt{spop y 5 1 ssum}] \longrightarrow [\texttt{ssum y 5 1}] \tag{B.7b}$$

B.3.6 COMPLEX STACK INSTRUCTIONS

These instructions are used to reshuffle the data stack (tail of the fraglet). sshove pushes the current tail symbol deep into the stack whereas syank pulls a symbol from deep in the stack to the tail. The position of the deep stack symbol is given by its relative position from the tail. The duplication variant syankdup creates a copy of the symbol. The sshoveat and syankat instructions don't use a number to identify the position but search for a certain symbol instead. Finally, the syankdupat variant searches for a certain symbol within the stack and copies its right neighbor symbol to the tail. Immediate variants are provided but rarely used in practice.

Immediate Instructions *(a)*

Tag	Production Rule
shove	$[\underline{\texttt{sshove}}\ \sigma\ \underline{\alpha}\ \beta\ \Phi\ \Psi^{\alpha}\ ] \Rightarrow [\sigma\ \Phi\ \beta\ \Psi^{\alpha}]$
syank	$[\underline{\texttt{syank}}\ \sigma\ \underline{\alpha}\ \Phi\ \beta\ \Psi^{\alpha}] \Rightarrow [\sigma\ \Phi\ \Psi^{\alpha}\ \beta]$
syankdup	$[\underline{\texttt{syankdup}}\ \sigma\ \underline{\alpha}\ \Phi\ \beta\ \Psi^{\alpha}] \Rightarrow [\sigma\ \Phi\ \beta\ \Psi^{\alpha}\ \beta]$
sshoveat	$[\underline{\texttt{sshoveat}}\ \sigma\ \underline{\alpha}\ \underline{\beta}\ \gamma\ \Phi\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}]$ $\Rightarrow [\sigma\ \Phi\ \gamma\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}]$
syankat	$[\underline{\texttt{syankat}}\ \sigma\ \underline{\alpha}\ \underline{\beta}\ \Phi\ \gamma\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}]$ $\Rightarrow [\sigma\ \Phi\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}\ \gamma]$
syankdupat	$[\underline{\texttt{syankdupat}}\ \sigma\ \underline{\alpha}\ \underline{\beta}\ \Phi\ \gamma\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}]$ $\Rightarrow [\sigma\ \Phi\ \gamma\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}\ \gamma]$

Stack Instructions *(b)*

Tag	Production Rule
sshove	$[\underline{\texttt{sshove}}\ \Phi\ \Psi^{\alpha}\ \beta\ \underline{\alpha}] \Rightarrow [\Phi\ \beta\ \Psi^{\alpha}]$
syank	$[\underline{\texttt{syank}}\ \Phi\ \beta\ \Psi^{\alpha}\ \underline{\alpha}] \Rightarrow [\Phi\ \Psi^{\alpha}\ \beta]$
syankdup	$[\underline{\texttt{syankdup}}\ \Phi\ \beta\ \Psi^{\alpha}\ \underline{\alpha}] \Rightarrow [\Phi\ \beta\ \Psi^{\alpha}\ \beta]$
sshoveat	$[\underline{\texttt{sshoveat}}\ \Phi\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}\ \gamma\ \underline{\beta}\ \underline{\alpha}] \Rightarrow [\Phi\ \gamma\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}]$
syankat	$[\underline{\texttt{syankat}}\ \Phi\ \gamma\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}\ \underline{\beta}\ \underline{\alpha}] \Rightarrow [\Phi\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}\ \gamma]$
syankdupat	$[\underline{\texttt{syankdupat}}\ \Phi\ \gamma\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}\ \underline{\beta}\ \underline{\alpha}]$ $\Rightarrow [\Phi\ \gamma\ \Psi^{\alpha}\ \underline{\beta}\ \Upsilon_{\{\beta\}}\ \gamma]$

Examples *(c)*

$$[\texttt{sshove this is test a 1}] \longrightarrow [\texttt{this is a test}] \tag{B.8a}$$

$$[\texttt{syankdupat 3 2 1 a b c a 2}] \longrightarrow [\texttt{3 2 1 a b c 2}] \tag{B.8b}$$

$$[\texttt{syankat 3 2 1 a b c a -1}] \longrightarrow [\texttt{3 2 1 a c b}] \tag{B.8c}$$

ARITHMETIC INSTRUCTIONS B.3.7

The following instructions treat the two operators as numbers (integer or floating point) and compute an arithmetic function. The instructions `slt`, `seq`, and `sgt` compare the two tail symbols and are mainly used in combination with the conditional instruction `sif`.

(a) *Immediate Instructions*

Tag	Production Rule
sum	$[\texttt{sum}\ \sigma\ \alpha\ \beta\ \Phi] \Rightarrow [\sigma\ \left(\alpha+\beta\right)\ \Phi]$
diff	$[\texttt{diff}\ \sigma\ \alpha\ \beta\ \Phi] \Rightarrow [\sigma\ \left(\alpha-\beta\right)\ \Phi]$
mult	$[\texttt{mult}\ \sigma\ \alpha\ \beta\ \Phi] \Rightarrow [\sigma\ \left(\alpha*\beta\right)\ \Phi]$
div	$[\texttt{div}\ \sigma\ \alpha\ \beta\ \Phi] \Rightarrow [\sigma\ \left(\alpha/\beta\right)\ \Phi]$
mod	$[\texttt{mod}\ \sigma\ \alpha\ \beta\ \Phi] \Rightarrow [\sigma\ \left(\alpha \bmod \beta\right)\ \Phi]$
pow	$[\texttt{pow}\ \sigma\ \alpha\ \beta\ \Phi] \Rightarrow [\sigma\ \left(\alpha^{\beta}\right)\ \Phi]$
min	$[\texttt{min}\ \sigma\ \alpha\ \beta\ \Phi] \Rightarrow [\sigma\ \min\left(\alpha,\beta\right)\ \Phi]$
max	$[\texttt{max}\ \sigma\ \alpha\ \beta\ \Phi] \Rightarrow [\sigma\ \max\left(\alpha,\beta\right)\ \Phi]$

(b) *Stack Instructions*

Tag	Production Rule
ssum	$[\texttt{ssum}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \left(\alpha+\beta\right)]$
sdiff	$[\texttt{sdiff}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \left(\alpha-\beta\right)]$
smult	$[\texttt{smult}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \left(\alpha*\beta\right)]$
sdiv	$[\texttt{sdiv}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \left(\alpha/\beta\right)]$
smod	$[\texttt{smod}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \left(\alpha \bmod \beta\right)]$
spow	$[\texttt{spow}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \left(\alpha^{\beta}\right)]$
smin	$[\texttt{smin}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \min\left(\alpha,\beta\right)]$
smax	$[\texttt{smax}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \max\left(\alpha,\beta\right)]$
slt	$[\texttt{slt}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \left(\beta<\alpha\,?\,1:0\right)]$
seq	$[\texttt{seq}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \left(\beta=\alpha\,?\,1:0\right)]$
sgt	$[\texttt{sgt}\ \Phi\ \beta\ \alpha] \Rightarrow [\Phi\ \left(\beta>\alpha\,?\,1:0\right)]$

(c) *Examples*

The following example computes the expression $y=2x+5$ for $x=12$:

$$\begin{array}{l}
[\texttt{match x spush 2 smult spush 5 ssum y}] + [\texttt{x 12}] \\
\longrightarrow [\texttt{spush 2 smult spush 5 ssum y 12}] \\
\longrightarrow [\texttt{smult spush 5 ssum y 12 2}] \\
\longrightarrow [\texttt{spush 5 ssum y 24}] \\
\longrightarrow [\texttt{ssum y 24 5}] \\
\longrightarrow [\texttt{y 29}]
\end{array} \tag{B.9}$$

CONDITIONAL INSTRUCTIONS B.3.8

The immediate instructions eq, and lt compare two numeric symbols in the head of the fraglet and prepend one or another symbol to the result. The stack instruction sif is used in combination with the comparison instructions slt, seq, sgt, or any other arithmetic instruction (see above).

Immediate Instructions (a)

Tag	Production Rule
eq	$[\underline{\texttt{eq}}\ \sigma\ \tau\ \alpha\ \beta\ \Phi] \Rightarrow [\left(\alpha=\beta\ ?\ \sigma : \tau\right)\ \Phi]$
lt	$[\underline{\texttt{lt}}\ \sigma\ \tau\ \alpha\ \beta\ \Phi] \Rightarrow [\left(\alpha<\beta\ ?\ \sigma : \tau\right)\ \Phi]$
empty	$[\underline{\texttt{empty}}\ \sigma\ \tau\ \Phi] \Rightarrow [\left(\lvert\Phi\rvert=0\ ?\ \sigma : \tau\right)\ \Phi]$

Stack Instructions (b)

Tag	Production Rule
sif	$[\underline{\texttt{sif}}\ \sigma\ \tau\ \Phi\ \alpha] \Rightarrow [(\alpha\neq 0\ ?\ \sigma : \tau)\ \Phi]$

Examples (c)

$$[\texttt{sif a b c d 0}] \longrightarrow [\texttt{b c d}] \tag{B.10a}$$

$$[\texttt{sif a b c d 3}] \longrightarrow [\texttt{a c d}] \tag{B.10b}$$

$$[\texttt{seq sif a b 2 3}] \longrightarrow [\texttt{sif a b 0}] \longrightarrow [\texttt{b}] \tag{B.10c}$$

SPLITTING INSTRUCTIONS B.3.9

These instructions make two out of one fraglet. The fork-instruction creates a copy of the fraglet and leaves both copies with a different header symbol. The mfork-instruction leaves multiple header symbols separate. The pop2-instruction splits off the header symbol; this instruction is deprecated and only provided for backwards compatibility. The split-instruction splits the fraglet at the first occurrence of the asterisk (*) symbol whereas the splitat-instruction does the same at a specified symbol. The release-instruction extracts a string of symbols between the first two occurrences of the asterisk (*) symbol as a separate fraglet. Again, there is a releaseat-instruction where the splitting point can be specified.

(a) *Immediate Instructions*

Tag	Production Rule
fork, f	$[\underline{\texttt{fork}}\ \sigma\ \tau\ \Phi] \Rightarrow [\sigma\ \Phi] + [\tau\ \Phi]$
mfork	$[\underline{\texttt{mfork}}\ \underline{\alpha}\ \Phi^{\alpha}\ \Psi^{\alpha}\ \Upsilon] \Rightarrow [\Phi^{\alpha}\ \Upsilon] + [\Psi^{\alpha}\ \Upsilon]$
pop2	$[\underline{\texttt{pop2}}\ \sigma\ \tau\ \alpha\ \Phi] \Rightarrow [\sigma\ \alpha] + [\tau\ \Phi]$
split, s	$[\underline{\texttt{split}}\ \Phi_{\{*\}}\ \underline{*}\ \Psi] \Rightarrow [\Phi_{\{*\}}] + [\Psi]$
splitat	$[\underline{\texttt{splitat}}\ \underline{\alpha}\ \Phi_{\{\alpha\}}\ \underline{\alpha}\ \Psi] \Rightarrow [\Phi_{\{\alpha\}}] + [\Psi]$
release	$[\underline{\texttt{release}}\ \Phi_{\{*\}}\ \underline{*}\ \Psi_{\{*\}}\ \underline{*}\ \Upsilon]$ $\Rightarrow [\Phi_{\{*\}}\ \Upsilon] + [\Psi_{\{*\}}]$
releaseat	$[\underline{\texttt{releaseat}}\ \underline{\alpha}\ \Phi_{\{\alpha\}}\ \underline{\alpha}\ \Psi_{\{\alpha\}}\ \underline{\alpha}\ \Upsilon]$ $\Rightarrow [\Phi_{\{\alpha\}}\ \Upsilon] + [\Psi_{\{\alpha\}}]$
divide, d	$[\underline{\texttt{divide}}\ \Phi\ \Psi]$ $\Rightarrow [\Phi] + [\Psi] \quad \lvert\Phi\rvert = \lvert\Psi\rvert \text{ or } \lvert\Phi\rvert = \lvert\Psi\rvert + 1)$
shuffle, h	$[\underline{\texttt{shuffle}}\ \alpha\ \beta\ \gamma\ \delta\ \ldots\ \pi\ \omega]$ $\Rightarrow [\alpha\ \gamma\ \ldots\ \pi] + [\beta\ \delta\ \ldots\ \omega]$

(b) *Stack Instructions*

Tag	Production Rule
sfork	$[\underline{\texttt{sfork}}\ \Phi\ \sigma\ \tau] \Rightarrow [\sigma\ \Phi] + [\tau\ \Phi]$
smfork	$[\underline{\texttt{smfork}}\ \Upsilon\ \Phi^{\alpha}\ \Psi^{\alpha}\ \underline{\alpha}] \Rightarrow [\Phi^{\alpha}\ \Upsilon] + [\Psi^{\alpha}\ \Upsilon]$
ssplitat	$[\underline{\texttt{ssplitat}}\ \Phi_{\{\alpha\}}\ \underline{\alpha}\ \Psi\ \underline{\alpha}] \Rightarrow [\Phi_{\{\alpha\}}] + [\Psi]$
sreleaseat	$[\underline{\texttt{sreleaseat}}\ \Phi_{\{\alpha\}}\ \underline{\alpha}\ \Psi_{\{\alpha\}}\ \underline{\alpha}\ \Upsilon\ \underline{\alpha}]$ $\Rightarrow [\Phi_{\{\alpha\}}\ \Upsilon] + [\Psi_{\{\alpha\}}]$

(c) *Examples*

$$[\underline{\texttt{split}}\ \texttt{a b c * d e f}] \longrightarrow [\texttt{a b c}] + [\texttt{d e f}] \tag{B.11a}$$

$$[\underline{\texttt{releaseat}}\ \texttt{X a b X c d X e f}] \longrightarrow [\texttt{a b e f}] + [\texttt{c d}] \tag{B.11b}$$

B.3.10 TRANSMISSION INSTRUCTIONS

Transmission instructions send fraglets to a neighbor vessel (`send`), to all neighbor vessels (`broadcast`), to any neighbor (`anycast`), inject it into a sub-vessel (`inject`) or expel it to the outer vessel (`expel`). The newnode-instruction is an experimental instruction, which creates a new sub-vessel on the fly.

Immediate Instructions *(a)*

Tag	Production Rule
send	${}_{v_i}[\underline{\mathtt{send}}\ \underline{v_j}\ \ \Phi] \Rightarrow {}_{v_j}[\Phi]$ if $(v_i, v_j) \in \mathcal{E}$
broadcast	${}_{v_i}[\underline{\mathtt{broadcast}}\ \ \Phi] \Rightarrow {}_{v_j}[\Phi]$ for all $(v_i, v_j) \in \mathcal{E}$
anycast	${}_{v_i}[\underline{\mathtt{anycast}}\ \ \Phi] \Rightarrow {}_{v_j}[\Phi]$ for one random $(v_i, v_j) \in \mathcal{E}$
inject	${}_{v_i}[\underline{\mathtt{inject}}\ \underline{v_j}\ \ \Phi] \Rightarrow {}_{v_j}[\Phi]$ if v_j a sub-vessel of v_i
newnode	${}_{v_i}[\underline{\mathtt{newnode}}\ \underline{v_j}\ \ \Phi] \Rightarrow {}_{v_j}[\Phi]$ create v_j as sub-vessel of v_i
expel	${}_{v_i}[\underline{\mathtt{expel}}\ \ \Phi] \Rightarrow {}_{v_j}[\Phi]$ if v_i a sub-vessel of v_j

Stack Instructions *(b)*

Tag	Production Rule
ssend	${}_{v_i}[\underline{\mathtt{ssend}}\ \ \Phi\ \underline{v_j}] \Rightarrow {}_{v_j}[\Phi]$ if $(v_i, v_j) \in \mathcal{E}$
sinject	${}_{v_i}[\underline{\mathtt{sinject}}\ \ \Phi\ \underline{v_j}] \Rightarrow {}_{v_j}[\Phi]$ if v_j a sub-vessel of v_i
snewnode	${}_{v_i}[\underline{\mathtt{snewnode}}\ \ \Phi\ \underline{v_j}] \Rightarrow {}_{v_j}[\Phi]$ create v_j as sub-vessel of v_i

Examples *(c)*

$${}_{\mathtt{local}}[\mathtt{send\ remote\ a\ b\ c}] \longrightarrow {}_{\mathtt{remote}}[\mathtt{a\ b\ c}] \tag{B.12a}$$

INSPECTION INSTRUCTIONS B.3.11

Inspection instructions obtain information from the environment: node returns the identifier of the reaction vessel, i.e. the address of the network node if the vessel is the outermost vessel. The instruction length returns the length of the fraglet not including the length and the subsequent dummy symbol, which is deleted. The instruction hash computes a hash value of the fraglet. The rnd-instruction obtains a random number. These instructions consume a dummy symbol χ in order to reduce the length of the fraglet by at least one symbol (see Section 5.2.4). The newname-instruction constructs a new tag symbol by combining the strings of two other symbols. Finally, the delay-instruction removes the fraglet from the vessel and re-injects it later.

(a) Immediate Instructions

Tag	Production Rule
node	${}_{v_i}[\underline{\texttt{node}}\ \ \sigma\ \ \chi\ \ \Phi] \Rightarrow {}_{v_i}[\sigma\ \ \underline{v_i}\ \ \Phi]$
length	$[\underline{\texttt{length}}\ \ \sigma\ \ \chi\ \ \underline{\Phi}] \Rightarrow [\sigma\ \ \underline{\lvert\Phi\rvert}\ \ \underline{\Phi}]$
hash	$[\underline{\texttt{hash}}\ \ \sigma\ \ \chi\ \ \underline{\Phi}] \Rightarrow [\sigma\ \ \underline{\mathrm{hash}(\Phi)}\ \ \underline{\Phi}]$
rnd	$[\underline{\texttt{rnd}}\ \ \sigma\ \ \chi\ \ \Phi] \Rightarrow [\sigma\ \ \underline{\left(\sim \mathrm{U}\left(-2^{31}, 2^{31}-1\right)\right)}\ \ \Phi]$
newname	$[\underline{\texttt{newname}}\ \ \sigma\ \ \underline{\alpha}\ \ \underline{\beta}\ \ \Phi] \Rightarrow [\sigma\ \ \underline{\text{“}\alpha\beta\text{”}}\ \ \Phi]$
delay	$[\underline{\texttt{delay}}\ \ \sigma\ \ \underline{\alpha}\ \ \Phi] \Rightarrow \cdots\tau = \alpha\cdots \Rightarrow [\sigma\ \ \Phi]$

(b) Stack Instructions

Tag	Production Rule
snode	${}_{v_i}[\underline{\texttt{snode}}\ \ \chi\ \ \Phi] \Rightarrow {}_{v_i}[\Phi\ \ \underline{v_i}]$
slength	$[\underline{\texttt{slength}}\ \ \chi\ \ \underline{\Phi}] \Rightarrow [\underline{\Phi}\ \ \underline{\lvert\Phi\rvert}]$
shash	$[\underline{\texttt{shash}}\ \ \chi\ \ \underline{\Phi}] \Rightarrow [\underline{\Phi}\ \ \underline{\mathrm{hash}(\Phi)}]$
srnd	$[\underline{\texttt{srnd}}\ \ \chi\ \ \Phi] \Rightarrow [\Phi\ \ \underline{\left(\sim \mathrm{U}\left(-2^{31}, 2^{31}-1\right)\right)}]$
snewname	$[\underline{\texttt{snewname}}\ \ \Phi\ \ \underline{\beta}\ \ \underline{\alpha}] \Rightarrow [\Phi\ \ \underline{\text{“}\beta\alpha\text{”}}]$
sdelay	$[\underline{\texttt{sdelay}}\ \ \Phi\ \ \underline{\alpha}] \Rightarrow \cdots\tau = \alpha\cdots \Rightarrow [\Phi]$

(c) Examples

$${}_{\texttt{local}}[\texttt{snode X a b c}] \longrightarrow {}_{\texttt{local}}[\texttt{a b c local}] \tag{B.13a}$$

$$[\texttt{slength X a b c}] \longrightarrow [\texttt{a b c 3}] \tag{B.13b}$$

$$[\texttt{shash X a b c}] \longrightarrow [\texttt{a b c 100384}] \tag{B.13c}$$

$$[\texttt{srnd X a b c}] \longrightarrow [\texttt{a b c 311937467}] \tag{B.13d}$$

$$[\texttt{newname X a b c}] \longrightarrow [\texttt{X ab c}] \tag{B.13e}$$

List of Figures

List of Tables

List of Algorithms

List of Symbols

Functions

$\operatorname{adj}(i,j)$	Adjacency of nodes v_i and v_j	51
$a_r(\mathbf{N}(t))$	Propensity of each reaction $r \in \mathcal{R}$ at a given vessel composition $\mathbf{N}(t)$	31
$\operatorname{bnd}(z,v)$	Binding of network service z to node v	297
$\mathrm{C}_f(\mathcal{W})$	Abstract closure operator on set $\mathcal{W}$ under function f	86
$\mathrm{C}_{\mathrm{re}}(\mathcal{W})$	Closure of fraglet set $\mathcal{W}$ under rewriting expansion	86
$\deg_{\mathrm{in}}(v_i)$	In-degree of node v_i, i.e. number of links arriving at v_i	140
$\deg_{\mathrm{out}}(v_i)$	Out-degree of node v_i, i.e. number of links leaving v_i	139
$\operatorname{dom} r$	Domain (set of reactant species) of reaction r	27
$\mathbb{E}[X(t)]$	Expected value of random variable $X(t)$	113
$f(\mathbf{x})$	Net production rate; total dilution flux	192
$h_r(\mathbf{N}(t))$	Number of distinct combinations of reactant molecules of reaction r at composition $\mathbf{N}(t)$	32
$H(s)$	Frequency response / transfer function	119
$\operatorname{img} r$	Image (set of product species) of reaction r	28
$\mathcal{L}\{f(t)\}$	Laplace transform of function $f(t)$	119
$\mathcal{L}^{-1}\{F(s)\}$	Inverse Laplace transform of function $F(S)$	120
$\mathcal{O}(r)$	Order of reaction rule r	27
$\operatorname{path}(z)$	List of network nodes along a routing path to service z	299
$\Phi(\chi)$	Relative net production rate; total relative dilution flux	260
$P(\mathbf{n},t)$	Conditional state probability of composition $\mathbf{n}$ at time t	103
$\operatorname{rank}(\mathbf{S})$	Rank of matrix $\mathbf{S}$	131
$\operatorname{re}(\mathcal{W})$	Rewriting expansion function of the set of fraglets $\mathcal{W}$	85
$\operatorname{Var}[X(t)]$	Variance of the random variable $X(t)$	126

MATRICES

$\mathbf{A}$	Adjacency matrix	141
$\mathbf{A}$	State matrix of an LTI system	118
$\mathbf{B}$	Input matrix of an LTI system	118
$\mathbf{C}$	Output matrix of an LTI system	119
$\mathbf{D}$	Diffusion matrix	126
$\mathbf{D}$	Feedthrough matrix of an LTI system	119
$\mathbf{I}$	Identity matrix	116
$\mathbf{J}$	Jacobian matrix	115
$\mathbf{Q}$	Transition rate matrix	105
$\mathbf{S}$	Stoichiometric matrix	113
$\mathbf{U}$	Unit matrix	142
$\boldsymbol{\Xi}$	Covariance matrix	126

SCALARS

A	Pre-exponential factor of the Arrhenius equation	335
$\alpha_{s,r}$	Number of s-molecules consumed by reaction r	27
b	Link bandwidth	201
$\beta_{s,r}$	Number of s-molecules produced by reaction r	27
Γ	Gaussian white noise	123
$\gamma_{s,r}$	Stoichiometric coefficient of species s in reaction r	27
C	Vessel capacity (number of symbols)	193
$\chi_s(t)$	Concentration of molecule s	260
c_r	Stochastic rate constant of reaction r	31
d	Delay	201
δ	Bit mutation rate	267
δ	Deficiency	131
E_{act}	Activation energy (macroscopic)	335
ε_{act}	Activation energy (microscopic)	335
$\varepsilon_{\text{heat}}$	Heat	340
E_{kin}	Kinetic energy (macroscopic)	338
ε_{kin}	Kinetic energy (microscopic)	335
E_{pot}	Potential energy (macroscopic)	343
ε_{pot}	Potential energy (microscopic)	341
ε_{ret}	Energy that is returned to the vessel	340
E_{tot}	Total energy	343
$\eta_s(t)$	Noise random variable of species s	125
k_{B}	Boltzmann constant	335

$k_{\max}$	Reaction coefficient of the fastest reaction	194
k_r	Reaction coefficient of reaction rule r	27
λ	Birth rate	110
μ	Death rate	110
N	Vessel capacity (number of molecules)	192
N_A	Avogadro constant: number of molecules per mole	29
$N_s(t)$	Random variable for the quantity of species s	28
Ω	Volume of the reaction vessel	29
ρ	Load factor	171
T	Max. CPU time required to execute one production rule	194
T	Temperature	335
$x_s(t)$	Mean quantity of molecule s	113

OTHER SYMBOLS

$\mathcal{A}$	Reaction algorithm	22
(i, j)	Network link (edge) between nodes (vertices) v_i and v_j	51
v_i	Network node	52
$\diamond$	Fraglets symbol wildcard	89
w	Word: string of symbols over an alphabet	57

SETS, MULTISETS, & TUPLES

$\mathcal{AC}$	Artificial chemistry	22
$\mathcal{B}$	Set of bindings between network services and nodes	297
$\mathcal{DAC}$	Distributed artificial chemistry	51
$\mathcal{E}$	Set of edges (network links)	51
$\mathcal{E}$	Stochastically closed subspace	267
$\mathcal{G}$	Network graph	51
$\mathcal{M}(\mathcal{S})$	Multiset over a set $\mathcal{S}$	28
$\mathcal{M}_{\mathrm{in},r}$	Reactant multiset of reaction rule r	27
$\mathcal{M}_{\mathrm{out},r}$	Product multiset of reaction rule r	27
$\mathcal{M}_v$	Multiset of species in the reaction vessel	28
$\mathcal{M}_{v_i}$	Multiset of species in network node v_i	52
$\mathcal{N}_i$	Set of neighbor nodes of network node v_i	51
$\mathcal{P}$	Set of production rules	57
$\mathcal{P}_{\mathcal{W}}$	Set of production rules among fraglets in set $\mathcal{W}$	85
$\mathcal{R}$	Set of reaction rules	22
$\mathcal{R}_i$	Set of reaction rules among molecules of network node v_i	52
$\mathcal{S}$	Set of molecular species	22
$\mathcal{S}_i$	Set of molecular species in network node v_i	52

Σ	Alphabet: set of symbols	57
$\mathcal{V}$	Set of network nodes (vertices)	51
$\mathcal{X}$	State space	103
$\mathcal{Z}$	Set of network services	297

VECTORS

$\mathbf{1}$	Column vector of all ones	109
$\boldsymbol{\gamma}_r$	Stoichiometric vector of reaction r	103
$\boldsymbol{\eta}(t)$	Vector of noise random variables	125
$\mathbf{n}$	Concrete composition: instance of $\mathbf{N}$	103
$\mathbf{N}(t)$	Composition: vector of molecular quantities	103
$\mathbf{N}_v(t)$	Molecular composition of vessel v	28
$\hat{\mathbf{p}}$	Stationary probability distribution	105
$\mathbf{u}(t)$	Input signal vector of an LTI system	118
$\hat{\mathbf{x}}$	Steady-state quantity vector	114
$\mathbf{x}(t)$	Average composition: vector of average quantities	114
$\tilde{\mathbf{x}}(t)$	Perturbation vector; offset from the fixed point	115

List of Quotes

1. "Only those who attempt the absurd will achieve the impossible. I think it's in my basement... let me go upstairs and check." Maurits Cornelis Escher (1898–1972), Dutch artist. (p. vii)
2. "Will the technologies of communication and culture help us to understand one another better, or will they deceive us and keep us apart?" In the October issue of the *Penthouse Magazine* (1988). Roger Waters (*1943), British musician, bass player and primary lyricist for the rock band Pink Floyd. (p. 13)
3. "The meeting of two personalities is like the contact of two chemical substances: if there is any reaction, both are transformed." In Jung (1933) (revised edition: Jung, 2005, p. 49). Carl Gustav Jung (1875–1961), Swiss psychiatrist and founder of analytical psychology. (p. 21)
4. "The symbol and the metaphor are as necessary to science as to poetry." In Bronowski (1958, Part 2, § 6, p. 36) (revised edition: Bronowski, 1990). Jacob Bronowski (1908–1974), British mathematician, biologist, and science historian of Polish origin. (p. 43)
5. "The outstanding feature of behavior is that it is often quite easy to recognize but extremely difficult or impossible to describe with precision." In Rapoport (1962, p. 92). Anatol Rapoport (1911–2007), Russian-born American Jewish mathematical psychologist, co-founder of the general systems theory. (p. 51)
6. "The epistemological value of probability theory is based on the fact that chance phenomena, considered collectively and on a grand scale, create non-random regularity." In Kolmogorov (1954). Andrey Nikolaevich Kolmogorov (1903–1987), Soviet Russian mathematician. (p. 73)

7. "An abstraction is one thing that represents several real things equally well." Quoted in Parnas (2007). Edgser Wybe Dijkstra (1930–2002), Duch computer scientist, winner of the 1972 Turing Award. (p. 83)
8. "One has then to develop a dynamics for such a string like structure." In Dirac (1955). Paul Adrien Maurice Dirac (1902–1984), British theoretical physicist. (p. 103)
9. "Someone who gossips to you will gossip about you." (English proverb) (p. 139)
10. "Complexity must be grown from simple systems that already work." In K. Kelly (1995). Kevin Kelly (*1952), founding editor of *Wired* magazine. (p. 153)
11. "Time is not bought ready-made at the watchmaker's." In Bronowski (1958, Part 2, § 5, p. 35) (revised edition: Bronowski, 1990). Jacob Bronowski (1908–1974), British mathematician, biologist, and science historian of Polish origin. (p. 185)
12. "Of course, there isn't any 'God of the Internet.' The Internet works because a lot of people cooperate to do things together." In "Heavenly Father of the NET" (1997). Jon Postel (1943–1998), American computer scientist. (p. 199)
13. "The art of healing comes from nature, not from the physician. Therefore the physician must start from nature, with an open mind." Philip von Hohenheim (a.k.a. Paracelsus) (1493–1541), alchemist, physician, astrologer, and general occultist. (p. 249)
14. "Was war also das Leben? Es war Wärme, das Wärmeprodukt formerhaltender Bestandlosigkeit, ein Fieber der Materie, von welchem der Prozess unaufhörlicher Zersetzung und Wiederherstellung unhaltbar verwickelt, unhaltbar kunstreich aufgebauter Eiweissmolekel begleitet war." In Mann (1924, Chap. 5). Paul Thomas Mann (1875–1955), German novelist and 1929 Nobel Prize laureate. (p. 259)
15. "We will now discuss in a little more detail the Struggle for Existence." In Darwin (1995, Chap. 3). Charles Darwin (1809–1882), British naturalist outlining the theory of evolution. (p. 283)
16. "Ceux qui passent toujours par les mêmes chemins, voyent ordinairement toujours les mêmes objets; il est rare qu'à force de suivre différentes routes, on ne découvre de nouveaux sujets dignes de nos attentions les plus sérieuses." In Polinière (1728, p. vii). Pierre Polinière (1671–1734), early investigator of electricity and electrical phenomena. (p. 297)
17. "Things do not turn out the way you think they will." In Crichton (2002). John Michael Crichton (1942–2008), American author, film producer and television producer. (p. 315)

18. “It is important to realize that in physics today, we have no knowledge what energy is.” In Feynman, Leighton, and Sands (1963, Sect. 4.1). Richard Feynman (1918–1988), American physicist. (p. 335)
19. “That simple principle predicts almost everything that’s happening.” In a radio interview on KUOW, Seattle (Chomsky, 2005). Noam Chomsky (*1928), American professor of linguistics, anarchist, human rights activist, and political analyst. (p. 363)
20. “I believe in intuition and inspiration. Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution.” In Einstein (1931, p. 97). Albert Einstein (1879–1955), German-Swiss-Austrian-American physicist. (p. 373)
21. “Nur was sich ändert bleibt bestehen!” (Sprichwort) (p. 377)
22. “What one concludes to see depends on the chosen model of reality.” In Zeh (1980). Heinz-Dieter Zeh (*1932), German theoretical physicist. (p. 379)

Bibliography

A

Abrash, H. I. (1986). Studies Concerning Affinity. *Journal of Chemical Education*, *63*, 1044–1047. English translation of Waage & Guldberg (1864). doi:10.1021/ed063p1044

Adamatzky, A., De Lacy Costello, B., & Asai, T. (2005). *Reaction-Diffusion Computers*. Elsevier.

Ahn, J. S., Danzig, P. B., Liu, Z., & Yan, L. (1995). Evaluation of TCP Vegas: emulation and experiment. *SIGCOMM Computer Communication Review*, *25* (4), 185–195. doi:10.1145/217391.217431

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers, Principles, Techniques & Tools*. Pearson Education, Inc.

Allman, M., Paxson, V., & Blanton, E. (2009). TCP Congestion Control. RFC 5681 (Draft Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc5681.txt

Allman, M., Paxson, V., & Stevens, W. (1999). TCP Congestion Control. RFC 2581 (Proposed Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc2581.txt

Amjad, H. (2004). *Combining Model Checking and Theorem Proving*. Cambridge, UK.

Anckaert, B., Madou, M., & de Bosschere, K. (2007). A Model for Self-Modifying Code. In J. Camenisch, C. Collberg, N. Johnson & P. Sallee (Eds.), *Proc. 8th International Conference on Information Hiding* (Vol. 4437, pp. 232–248). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. doi:10.1007/978-3-540-74124-4_16

Anderson, D., Craciun, G., & Kurtz, T. (2010). Product-Form Stationary Distributions for Deficiency Zero Chemical Reaction Networks. *Bulletin of Mathematical Biology*, *72*, 1947–1970. doi:10.1007/s11538-010-9517-4

Arrhenius, S. (1884). *Kongliga Svenska Vetenskaps-Akademiens Handlingar*, *8*, 66–93.

Aspnes, J. (2003). Randomized Protocols for Asynchronous Consensus. *Distributed Computing*, *16* (2–3), 165–175. doi:10.1007/s00446-002-0081-5

Athuraliya, S., & Low, S. H. (2001). An Empirical Validation of a Duality Model of TCP and Queue Management Algorithms. In *Proc. 2001 Winter Simulation Conference* (Vol. 2, pp. 1269–1274). doi:10.1109/WSC.2001.977445

B

Baccelli, F., Cohen, G., Olsder, G. J., & Quadrat, J.-P. (1992). *Synchronization and Linearity : An Algebra for Discrete Event Systems*. Wiley Series in Probability and Mathematical Statistics. Chichester, UK: John Wiley & Sons Ltd. Retrieved 29 September 2010, from http://www-rocq.inria.fr/metalau/cohen/documents/BCOQ-book.pdf

Bagley, R. J., & Farmer, J. D. (1992). Spontaneous Emergence of a Metabolism. In C. G. Langton, C. E. Taylor, D. J. Farmer & S. Rasmussen (Eds.), *Proc. 2nd Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems (Artificial Life II)* (pp. 93–140). Redwood City, CA, USA: Addison-Wesley.

Bagley, R. J., Farmer, J. D., Kauffman, S. A., Packard, N. H., Perelson, A., & Stadnyk, I. (1989). Modeling Adaptive Biological Systems. *Biosystems*, *23* (2–3), 113–137. Artificial Worlds Modeling Of Biological Systems. doi:10.1016/0303-2647(89)90016-6

Bakhshi, R., Cloth, L., Fokkink, W., & Haverkort, B. (2009). Mean-Field Analysis for the Evaluation of Gossip Protocols. *Proc. 6th International Conference on Quantitative Evaluation of Systems*, 247–256. doi:10.1109/QEST.2009.38

Ballarini, P., Mardare, R., & Mura, I. (2009). Analysing Biochemical Oscillation through Probabilistic Model Checking. *Electronic Notes in Theoretical Computer Science*, *229* (1), 3–19. Proceedings of the Second Workshop From Biology to Concurrency and Back (FBTC 2008). doi:10.1016/j.entcs.2009.02.002

Banâtre, J. P., Fradet, P., & Radenac, Y. (2006). A Generalized Higher-Order Chemical Computation Model. *Electronic Notes in Theoretical Computer Science*, *135* (3), 3–13. doi:10.1016/j.entcs.2005.09.016

Banâtre, J.-P., Fradet, P., & Radenac, Y. (2005). Principles of Chemical Programming. In *Proc. 5th International Workshop on Rule-Based Programming* (Vol. 124, *1*, pp. 133–147). Elsevier. doi:10.1016/j.entcs.2004.07.019

Banâtre, J.-P., & Le Métayer, D. (1986). *A New Computational Model and its Discipline of Programming* (Research Report No. 566). Retrieved 19 October 2010, from http://hal.inria.fr/docs/00/07/59/88/PDF/RR-0566.pdf

Banâtre, J.-P., & Le Métayer, D. (1993). Programming by Multiset Transformation. *Communications of the ACM*, *36* (1), 98–111. doi:10.1145/151233.151242

Banzhaf, W. (1990). The "molecular" traveling salesman. *Biological Cybernetics*, *64* (1), 7–14. doi:10.1007/BF00203625

Banzhaf, W. (1993a). Self-replicating Sequences of Binary Numbers. Foundations I: General. *Biological Cybernetics*, *69*, 269–274. doi:10.1007/BF00203123

Banzhaf, W. (1993b). Self-replicating Sequences of Binary Numbers. Foundations II: Strings of length N=4. *Biological Cybernetics*, *69*, 275–281. doi:10.1007/BF00203124

Baran, P. (1964). On Distributed Communications Networks. *IEEE Transactions on Communication Systems*, *12* (1), 1–9. doi:10.1109/TCOM.1964.1088883

Barbuti, R., Cataudella, S., Maggiolo-Schettini, A., Milazzo, P., & Troina, A. (2005). A Probabilistic Model for Molecular Systems. *Fundamena Informaticae*, *67* (1–3), 13–27.

Baumann, R. C. (2002). Soft Errors in Commercial Semiconductor Technology: Overview and Scaling Trends. *IEEE 2002 Reliabiliby Physics Tutorial Notes*.

Beck, K., & Cunningham, W. (1987). Using Pattern Languages for Object-Oriented Programs. In *OOPSLA '87 Workshop on Specification and Design for Object-Oriented Programming*.

Bella, G. (2007). *Formal Correctness of Security Protocols*. Springer.

Benkö, Flamm, C., & Stadler, P. F. (2005). Explicit Collision Simulation of Chemical Reactions in a Graph Based Artificial Chemistry. In M. S. Capcarrère, A. A. Freitas, P. J. Bentley, C. G. Johnson & J. Timmis (Eds.), *Proc. 8th European Conference on Advances in Artificial Life (ECAL 2005)* (pp. 725–733). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. doi:10.1007/11553090_73

Benkö, G., Centler, F., Dittrich, P., Flamm, C., Stadler, B., & Stadler, P. F. (2008). A Topological Approach to Chemical Organizations. *Artificial Life*, *15* (1), 71–88. doi:10.1162/artl.2009.15.1.15105

Benkö, G., Flamm, C., & Stadler, P. F. (2003). A Graph-Based Toy Model of Chemistry. *Journal of Chemical Information and Computer Science*, *43* (3), 1085–1093. doi:10.1021/ci0200570

Berezin, S., Campos, S., & Clarke, E. M. (1998). Compositional Reasoning in Model Checking. In *Proc. International Symposium on Compositionality (COMPOS'97)* (Vol. 1536, pp. 81–102). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. doi:10.1007/3-540-49213-5_4

Berry, G., & Boudol, G. (1989). The Chemical Abstract Machine. In *Proc. 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 81–94). New York, NY, USA: ACM. doi:10.1145/96709.96717

Bertot, Y., & Castéran, P. (2004). Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions. In *Texts in Theoretical Computer Science* (Vol. XXV). EATCS. Springer.

Bhargavan, K., Gunter, C. A., & Obradovic, D. (2000). Routing Information Protocol in HOL/SPIN. In *Proc. 13th International Conference on Theorem Proving in Higher Order Logics (TPHOLs '00)* (pp. 53–72). Berlin / Heidelberg: Springer. doi:10.1007/3-540-44659-1_4

IBM. (2001). Autonomic Computing: IBM's Perspective on the State of Information Technology. Retrieved 30 July 2010, from http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

IEEE Standard 802.3. (2008). IEEE. Retrieved 16 October 2010, from IEEE: http://standards.ieee.org/getieee802/download/802.3-2008_section1.pdf

ITU. (1996). X.25: Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit. International Telecommunication Union.

Blomberg, C. (2006). Fluctuations for Good and Bad: The Role of Noise in Living Systems. *Physics of Life Reviews*, *3* (3), 133–161. doi:10.1016/j.plrev.2006.06.001

Bochmann, G. V. (1976). Finite State Description of Communication Protocols. *Computer Networks*, *2* (4–5), 361–372. doi:10.1016/0376-5075(78)90015-6

Bohacek, S., Hespanha, J., Lee, J., Lim, C., & Obraczka, K. (2003). TCP-PR: TCP for persistent packet reordering. In *Proc. 23rd Interational Conference on Distributed Computing Systems* (pp. 222–231). doi:10.1109/ICDCS.2003.1203469

Bose, S. K. (2002). *An Introduction to Queueing Systems*. Springer.

Bracha, G., & Toueg, S. (1985). Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM*, *32* (4), 824–840. doi:10.1145/4221.214134

Braden, R., Faber, T., & Handley, M. (2002). From Protocol Stack to Protocol Heap — Role-Based Architecture. In *Proc. Hot Topics in Computer Networks (HotNets-I)*. ACM.

Brakmo, L., & Peterson, L. (1995). TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, *13* (8), 1465–1480. doi:10.1109/49.464716

Briggs, G. E., & Haldane, J. B. S. (1925). A Note on the Kinetics of Enzyme Action. *Biochemical Journal*, *19* (2), 338–339. PMID: PMC1259181

Bronowski, J. (1958). *Science and Human Values*. Harper & Row.

Bronowski, J. (1990). *Science and Human Values* (Revised ed.). Harper Perennial.

Bruggeman, F. J., Blüthgen, N., & Westerhoff, H. V. (2009). Noise Management by Molecular Networks. *PLOS Computational Biology*, *5* (9), e1000506. doi:10.1371/journal.pcbi.1000506

Budhiraja, A., Hernández-Campos, F., Kulkarni, V. G., & Smith, F. D. (2004). Stochastic Differential Equation for TCP Window Size: Analysis and Experimental Validation. *Probability in the Engineering and Informational Sciences*, *18* (1), 111–140. doi:10.1017/S0269964804181084

Buisman, H. J., ten Eikelder, H. M. M., Hilbers, P. A. J., & Liekens, A. M. L. (2008). Computing Algebraic Functions with Biochemical Reaction Networks. *Artificial Life*, *15* (1), 5–19. doi:10.1162/artl.2009.15.1.15101

Bunker, D. L., Garrett, B., Kleindienst, T., & Long, G. S. (1974). Discrete simulation methods in combustion kinetics. *Combustion and Flame*, *23* (3), 373–379. doi:10.1016/0010-2180(74)90120-5

C

Cai, H., Shao, Z., & Vaynberg, A. (2007). Certified Self-Modifying Code. In *Proc. 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'07)* (pp. 66–77). San Diego, CA, USA: ACM. doi:10.1145/1273442.1250743

Calude, C. S., & Paŭn, G. (2001). *Computing with Cells and Atoms: An Introduction to Quantum, DNA and Membrane Computing*. CRC Press.

Campbell, S., Chancelier, J.-P., & Nikoukhah, R. (2006). *Modeling and Simulation in Scilab/Scicos*. New York, NY, USA: Springer.

Cannon, W. B. (1929). Organization For Physiological Homeostasis. *Physiological Review*, *9*, 399–431. Retrieved from http://physrev.physiology.org/cgi/reprint/9/3/399.pdf

Cao, Y., & Petzold, L. R. (2005). Trapezoidal Tau-Leaping Formula for the Stochastic Simulation of Chemically Reacting Systems. In *Proc. Foundations in Systems Biology in Engeneering (FOSBE 2005)* (pp. 149–152).

Cardelli, L. (2008). From Processes to ODEs by Chemistry. In G. Ausiello, G. Karhumäki Juhani Mauri & C.-H. L. Ong (Eds.), *Proc. 5th IFIP International Conference On Theoretical Computer Science (TCS 2008)* (Vol. 273, pp. 261–281). IFIP. Boston, MA, USA: Springer. doi:10.1007/978-0-387-09680-3_18

Cardelli, L., & Gordon, A. D. (1998). Mobile Ambients. In M. Nivat (Ed.), *Proc. 1st International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'98)* (Vol. 1378, pp. 140–155). Berlin / Heidelberg: Springer. Retrieved from http://www.springerlink.com/content/x2rka7bb8qk5hlf3

Cassandras, C. G. (1993). *Discrete Event Systems.* Burr Ridge, IL, USA: Richard D. Irwin, Inc., and Asken Associates, Inc.

Cau, Y., Gillespie, D. T., & Petzold, L. R. (2005). The Slow-Scale Stochastic Simulation Algorithm. *Journal of Chemical Physics*, *122* (1). doi:10.1063/1.1824902

Cau, Y., Gillespie, D. T., & Petzold, L. R. (2006). Efficient Step Size Selection for the Tau-Leaping Simulation Method. *Journal of Chemical Physics*, *124* (4). doi:10.1063/1.2159468

Chakrapani, L. N., Korkmaz, P., Akgul, B. E. S., & Palem, K. V. (2007). Probabilistic System-on-a-Chip architectures. *ACM Transactions on Design Automation of Electronic Systems*, *12* (3), 1–28. doi:10.1145/1255456.1255466

Chang, C.-S. (2000). *Performance Guarantees in Communications Networks.* Berlin / Heidelberg: Springer.

Chatterjee, A., Vlachos, D. G., & Katsoulakis, M. A. (2005). Binomial Distribution Based Tau-Leap Accelerated Stochastic Simulation. *Journal of Chemical Physics*, *122* (2). doi:10.1063/1.1833357

Cheshire, S. (2008). IPV4 Address Conflict Detection. RFC 5227 (Proposed Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc5227.txt

Chiu, D.-M., & Jain, R. (1989). Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, *17* (1), 1–14. doi:10.1016/0169-7552(89)90019-6

Chlamtac, I., Petrioli, C., & Redi, J. (1997). An Energy-Conserving Access Protocol for Wireless Communication. In *Proc. IEEE International Conference on Communications (ICC 97)* (Vols. 2, pp. 1059–1062). doi:10.1109/ICC.1997.610041

Chomsky, N. (2005). Interview by Steve Scher on KUOW. Radio broadcast. Seattle, WA, USA. Retrieved 6 October 2010, from http://www.kuow.org/program.php?id=8683

Cisco. (2010). Cisco Visual Networking Index: Forecast and Methodology, 2009—2014. Cisco Systems Inc. San Jose, CA, USA. Retrieved 28 September 2010, from Cisco Systems Inc.: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

Clarke, E. M. (2008). The Birth of Model Checking: History, Achievements, Perspectives. In O. Grumberg & H. Veith (Eds.), *25 Years of Model Checking — History, Achievements, Perspectives* (Vol. 5000, pp. 1–26). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer.

Clement, L., & Nagpal, R. (2003). Self-Assembly and Self-Repairing Topologies. In *Proc. Workshop on Adaptability in Multi-Agent Systems.*

Conway, M. E. (1963). Design of a Separable Transition-Diagram Compiler. *Communications of the ACM*, *6* (7), 396–408. doi:10.1145/366663.366704

Coplien, J. O. (1991). *Advanced C++ Programming Styles and Idioms.* Addison-Wesley.

Cremean, L. B., & Murray, R. M. (2003). Stability Analysis of Interconnected Nonlinear Systems Under Matrix Feedback. In *Proc. 42th IEEE Conference on Decision and Control* (pp. 3078–3083). doi:10.1109/CDC.2003.1273096

Crichton, M. (2002). *Prey.* HarperCollins.

D

Darwin, C. (1995). *The Origin of Species.* Gramercy.

Dayan, P., & Abbott, L. F. (2001). *Theoretical Neuroscience.* MIT Press.

Deckard, A., & Sauro, H. M. (2004). Preliminary Studies on the In Silico Evolution of Biochemical Networks. *ChemBioChem*, *5* (10), 1423–1431. doi:10.1002/cbic.200400178

Deckard, A. C., Bergmann, F. T., & Sauro, H. M. (2009). Enumeration and Online Library of Mass-Action Reaction Networks. *arXiv.org q-bio.NM.*

Decraene, J. (2006). *The Holland Broadcast Language.*

Decraene, J., Mitchell, G., & McMullin, B. (2008). Unexpected Evolutionary Dynamics in a String Based Artificial Chemistry. In S. Bullock, J. Noble, R. A. Watson & M. A. Bedau (Eds.), *Proc. 11th International Conference on the Simulation and Synthesis of Living Systems (Artificial Life XI)* (pp. 158–165). Cambridge, MA, USA: MIT Press.

Decraene, J., Mitchell, G., McMullin, B., & Kelly, C. (2007). The Holland Broadcast Language and the Modeling of Biochemical Networks. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi & A. Esparcia-Alc'azar (Eds.), *Genetic Programming* (Vol. 4445, pp. 361–370). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. doi:10.1007/978-3-540-71605-1_34

Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., ... Terry, D. (1987). Epidemic Algorithms for Replicated Database Maintenance. In *Proc. 6th Annual ACM Symposium on Principles of Distributed Computing (PODC '87)* (pp. 1–12). Vancouver, British Columbia, Canada: ACM. doi:10.1145/41840.41841

Dennis, J. B. (1980). Data Flow Supercomputers. *Computer*, *13* (11), 48–56. doi:10.1109/MC.1980.1653418

Dewdney, A. K. (1984). In the Game Called Core War Hostile Programs Engage in a Battle of Bits. *Scientific American*.

Di Caro, G., & Dorigo, M. (1998). AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, *9*, 317–365.

Di Caro, G., Ducatelle, F., & Gambardella, L. M. (2005). AntHocNet: An Adaptive Nature-Inspired Algorithm for Routing in Mobile Ad Hoc Networks. *European Transactions on Telecommunications* (16), 443–455. doi:10.1002/ett.1062

Di Liu, W. E., & Vanden-Eijnden, E. (2007). Nested Stochastic Simulation Algorithms for Chemical Kinetic Systems with Multiple Time Scales. *Journal of Computational Physics*, *221* (1), 158–180. doi:10.1016/j.jcp.2006.06.019

Dirac, P. (1955). Lectures on Quantum Mechanics and Relativistic Field Theory: Tata Institute of Fundamental Research, Bombay.

Dittrich, P. (2001). *On Artificial Chemistries.* (Doctoral Thesis, University of Dortmund).

Dittrich, P. (2005). Chemical Computing. In J.-P. Banâtre, P. Fradet, J.-L. Giavitto & O. Michel (Eds.), *Unconventional Programming Paradigms* (Vol. 3566, pp. 19–32). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. doi:10.1007/11527800_2

Dittrich, P., & Banzhaf, W. (1997). A Topological Structure Based on Hashing — Emergence of a "Spatial" Organization. In P. Husbands & I. Harvey (Eds.), *Proc. 4th European Conference on Advances in Artificial Life (ECAL 1997)*. MIT Press.

Dittrich, P., & Banzhaf, W. (1998). Self-Evolution in a Constructive Binary String System. *Artificial Life*, *4* (2), 203–220. doi:10.1162/106454698568521

Dittrich, P., & Speroni di Fenizio, P. (2007). Chemical Organization Theory. *Bulletin of Mathematical Biology*, *69* (4), 1199–1231. doi:10.1007/s11538-006-9130-8

Dittrich, P., Ziegler, J., & Banzhaf, W. (2001). Artificial Chemistries - A Review. *Artificial Life*, *7* (3), 225–275. doi:10.1162/106454601753238636

Dressler, F., Dietrich, I., German, R., & Krüger, B. (2009). A Rule-Based System for Programming Self-Organized Sensor and Actor Networks. *Computer Networks*, *53* (10), 1737–1750. doi:10.1016/j.comnet.2008.09.007

Duflot, M., Kwiatkowska, M., Norman, G., Parker, D., Peyronnet, S., Picaronny, C., & Sproston, J. (2010). Practical Applications of Probabilistic Model Checking to Communication Protocols. In S. Gnesi & T. Margaria (Eds.), *FMICS Handbook on Industrial Critical Systems*. IEEE Computer Society.

E

Ehrenfest, P., & Ehrenfest, T. (1907). Über zwei bekannte Einwände gegen das Boltzmannsche H-Theorem. *Physikalische Zeitschrift*, *8*, 311.

Eigen, M., & Schuster, P. (1979). *The Hypercycle: A Principle of Natural Self-Organization*. Springer.

Eigen, M., & Winkler, R. (1975). *Das Spiel*. English translation: Eigen et al. (1993). Piper.

Eigen, M., Winkler, R., Kimber, R. B., & Kimber, R. (1993). *Laws of the Game: How the Principles of Nature Govern Chance*. English translation of Eigen & Winkler (1975). Princeton University Press.

Einstein, A. (1931). *Cosmic Religion: With Other Opinions and Aphorisms*. Covici-Friede.

Eissfeldt, H. (1997). POSIX: a Developer's View of Standards. In *Proc. Annual Conference on USENIX Annual Technical Conference (ATEC '97)* (pp. 24–24). Anaheim, CA, USA: USENIX Association.

El-Samad, H., & Khammash, M. (2004). Intrinsic Noise Rejection in Gene Networks by Regulation of Stability. In *Proc. 1st International Symposium on Control, Communications and Signal Processing* (pp. 187–190). Hammamet, Tunisia. doi:10.1109/ISCCSP.2004.1296252

Elf, J. (2004). *Intracellular Flows and Fluctuations.* (Doctoral Thesis, Uppsala University).

Elf, J., & Ehrenberg, M. (2003). Fast Evaluation of Fluctuations in Biochemical Networks With the Linear Noise Approximation. *Genome Research*, *13* (11), 2475–2484. doi:10.1101/gr.1196503

Ellson, J., Gansner, E., Koutsofios, L., North, S., & Woodhull, G. (2002). Graphviz — Open Source Graph Drawing Tools. In P. Mutzel, M. Jünger & S. Leipert (Eds.), *Graph Drawing* (Vol. 2265, pp. 594–597). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. doi:10.1007/3-540-45848-4_57

Erlang, A. K. (1917). Solution of some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges. *Elektrotkeknikeren*, *13*.

Eugster, P. T., Guerraoui, R., Kermarrec, A.-M., & Massoulié, L. (2004). From Epidemics to Distributed Computing. *IEEE Computer*, *37* (5), 60–67.

F

Fall, K., & Vardhan, K. (2010). The Network Simulator (NS-2). Retrieved 29 June 2010, from http://www.isi.edu/nsnam/ns

Farmer, J. D., Kauffman, S. A., & Packard, N. H. (1986a). Autocatalytic Replication of Polymers. *Physica D*, *2* (1–3), 50–67.

Farmer, J. D., Kauffman, S. A., & Packard, N. H. (1986b). Autocatalytic Replication of Polymers. *Physica D*, *2* (1–3), 50–67.

Fehribach, J. D. (2009). Vector-Space Methods and Kirchhoff Graphs for Reaction Networks. *SIAM Journal on Applied Mathematics*, *70* (2), 543–562. doi:10.1137/080720115

Feinberg, M. (1972). Complex Balancing in General Kinetic Systems. *Archive for Rational Mechanics and Analysis*, *49* (3). doi:10.1007/BF00255665

Feinberg, M. (1979). Lectures on Chemical Reaction Networks. Delivered at University of Wisconsin, Madison. Retrieved 2 July 2010, from http://www.che.eng.ohio-state.edu/~feinberg/LecturesOnReactionNetworks

Fell, D. A. (1997). *Understanding the Control of Metabolism.* London: Portland Press.

Felzenszwalb, P., & Huttenlocher, D. (2004). Efficient Belief Propagation for Early Vision. (Vol. 1, pp. 261–268). doi:10.1109/CVPR.2004.1315041

Ferm, L., Lötstedt, P., & Hellander, A. (2008). A Hierarchy of Approximations of the Master Equation Scaled by a Size Parameter. *Journal of Scientific Computing*, *34* (2), 127–151. doi:10.1007/s10915-007-9179-z

Fernando, C., & Rowe, J. (2007). Natural Selection in Chemical Evolution. *Journal of Theoretical Biology*, *247* (1), 152–167. doi:10.1016/j.jtbi.2007.01.028

Feynman, R., Leighton, R., & Sands, M. (1963). *The Feynman Lectures on Physics: Volume 1* (2nd ed., Vols. 1). The Feynman Lectures on Phsyics. Boston, MA, USA: Addison-Wesley.

Fidler, M. (2006). An End-to-End Probabilistic Network Calculus with Moment Generating Functions. In *14th IEEE International Workshop on Quality of Service (IWQOS 2006)* (pp. 261–270). doi:10.1109/IWQOS.2006.250477

Fishtik, I., Callaghan, C. A., & Datta, R. (2004a). Reaction Route Graphs. I. Theory and Algorithm. *Journal of Physical Chemistry B*, *108*, 5671–5682. doi:10.1021/jp0374004

Fishtik, I., Callaghan, C. A., & Datta, R. (2004b). Reaction Route Graphs. II. Examples of Enzyme- and Surface-Catalyzed Single Overall Reactions. *Journal of Physical Chemistry B*, *108*, 5683–5697. doi:10.1021/jp037401w

Fishtik, I., Callaghan, C. A., & Datta, R. (2005). Reaction Route Graphs. III. Non-Minimal Kinetic Mechanisms. *Journal of Physical Chemistry B*, *109*, 2710–2722. doi:10.1021/jp046115x

Flamm, C., Ullrich, A., Ekker, H., Mann, M., Högerl, D., Rohrschneider, M., . . . Stadler, P. F. (2010). Evolution of Metabolic Networks: A Computational Framework. *Journal of Systems Chemistry*, *1* (4), 1–14. doi:10.1186/1759-2208-1-4

Floyd, S., Handley, M., Padhye, J., & Widmer, J. (2008). TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 5348 (Proposed Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc5348.txt

Floyd, S., & Fall, K. (1999). Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, *7* (4), 458–472. doi:10.1109/90.793002

Fontana, W. (1992). Algorithmic Chemistry. In C. G. Langton, C. E. Taylor, J. D. Farmer & S. Rasmussen (Eds.), *Proc. 2nd Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems (Artificial Life II)* (pp. 159–210). Redwood City, CA, USA: Addison-Wesley.

Fontana, W., & Buss, L. W. (1994). "The Arrival of the Fittest": Toward a Theory of Biological Organization. *Bulletin of Mathematical Biology*, *56* (1), 1–64. doi:10.1016/S0092-8240(05)80205-8

Fontana, W., Wagner, G. P., & Buss, L. W. (1994). Beyond Digital Naturalism. *Artificial Life*, *1* (1–2), 211–227.

Freitas Jr., R. A., & Merkle, R. C. (2004). *Kinematic Self-Replicating Machines*. Georgetown, TX, USA: Landes Bioscience. Retrieved from http://www.molecularassembler.com/KSRM.htm

G

Gadgil, C., Lee, C. H., & Othmer, H. G. (2005). A Stochastic Analysis of First-Order Reaction Networks. *Bulletin of Mathematical Biology*, *67* (5), 901–946. doi:doi:10.1016/j.bulm.2004.09.009

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Design*. Addison-Wesley.

Gangadhar, D. K. (2005). Meta Dynamic States for Self Healing Autonomic Computing Systems. In *2005 IEEE International Conference on Systems, Man and Cybernetics* (Vol. 1, pp. 39–46). doi:10.1109/ICSMC.2005.1571119

Gelenbe, E. (2008). Network of Interacting Synthetic Molecules in Steady State. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, *464* (2096), 2219–2228. doi:10.1098/rspa.2008.0001

Ghosh, D., Sharman, R., Rao, H. R., & Upadhyaya, S. (2007). Self-Healing Systems - Survey and Synthesis. *Decision Support Systems*, *42* (4), 2164–2185. doi:10.1016/j.dss.2006.06.011

Giarratano, J. C., & Riley, G. (2004). *Expert Systems, Principles and Programming* (4th ed.). Course Technology.

Gibson, M. A., & Bruck, J. (2000). Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. *Journal of Physical Chemistry A*, *104* (9), 1876–1889. doi:10.1021/jp993732q

Gillespie, D. T. (1976). A General Method for Numerically Simulating the Stochastic Time evolution of Coupled Chemical Reactions. *Journal of Computational Physics*, *22* (4), 403–434. doi:10.1016/0021-9991(76)90041-3

Gillespie, D. T. (1977). Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry*, *81* (25), 2340–2361. doi:10.1021/j100540a008

Gillespie, D. T. (1992). A Rigorous Derivation of the Chemical Master Equation. *Physica A: Statistical Mechanics and its Applications*, *188* (1–3), 404–425. doi:10.1016/0378-4371(92)90283-V

Gillespie, D. T. (2000). The Chemical Langevin Equation. *Journal of Chemical Physics*, *113* (1). doi:10.1063/1.481811

Gillespie, D. T. (2001). Approximate Accelerated Stochastic Simulation of Chemically Reacting Systems. *Journal of Chemical Physics*, *115* (4), 1716–1733. doi:10.1063/1.1378322

Gillespie, D. T. (2002). The Chemical Langevin and Fokker-Planck Equations for the Reversible Isomerization Reaction. *Journal of Physical Chemistry A*, *106* (20), 5063–5071. doi:10.1021/jp0128832

Gillespie, D. T. (2007). Stochastic Simulation of Chemical Kinetics. *Annual Review of Physical Chemistry*, *58*, 35–55. doi:10.1146/annurev.physchem.58.032806.104637

Gillespie, D. T., & Petzold, L. R. (2003). Improved Leap-Size Selection for Accelerated Stochastic Simulation. *Journal of Physical Chemistry*, *119* (16), 8229–8234. doi:10.1063/1.1613254

Glaz, J. (1979). Probabilities and Moments for Absorption in Finite Homogeneous Birth-Death Processes. *Biometrics*, *35* (4), 813–816. JSTOR: 2530113. Retrieved from http://www.jstor.org/stable/2530113

Gómez-Uribe, C. A., & Verghese, G. C. (2007). Mass Fluctuation Kinetics: Capturing Stochastic Effects in Systems of Chemical Reactions Through Coupled Mean-Variance Computations. *Journal of Chemical Physics*, *126* (2). doi:10.1063/1.2408422

Gordon-Smith, C. (2007). Evolution Without Smart Molecules. In *Extending the Darwinian Framework: New levels of selection and inheritance, workshop at the 9th European Conference on Artificial Life (ECAL 2007).*

Graphviz Web Page. (2010). Retrieved 3 October 2010, from http://www.graphviz.org

Griffiths, A. J. F., Miller, J. H., Suzuki, D. T., Lewontin, R. C., & Gelbart, W. M. (2000). *An Introduction to Genetic Analysis* (7th ed.). New York, NY, USA: W. H. Freeman.

Gupta, A., Forgy, C., Newell, A., & Wedig, R. (1986). Parallel Algorithms and Architectures for Rule-Based Systems. *SIGARCH Computer Architecture News*, *14* (2), 28–37. doi:10.1145/17356.17360

Györgi, P. (1993). OMNET++: Objective Modular Network Testbed. In *Proc. International Workshop on Modeling, Analysis, and Simulation on Computer and Telecommunication Systems (MASCOTS '93)* (pp. 323–326).

H

Halpern, J. Y., & Vardi, M. Y. (1991). Model Checking vs. Theorem Proving: a Manifesto. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, 151–176.

Handley, M., Floyd, S., Padhye, J., & Widmer, J. (2003). TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448 (Proposed Standard). Obsoleted by RFC 5348. Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc3448.txt

Hanggi, P., & Shuler, K. E. (1981). On the Relations Between Markovian Master Equations and Stochastic Differential Equations. *Physica A*, *107* (1), 143–157. doi:10.1016/0378-4371(81)90028-5

Hasan, O., & Tahar, S. (2009). Probabilistic Analysis of Wireless Systems Using Theorem Proving. *Electronic Notes in Theoretical Computer Science*, *242* (2), 43–58. Proc. 1st Workshop on Formal Methods for Wireless Systems (FMWS 2008). doi:10.1016/j.entcs.2009.06.022

Haseltine, E. L., & Rawlings, J. B. (2002). Approximate Simulation of Coupled Fast and Slow Reactions for Stochastic Chemical Kinetics. *Journal of Chemical Physics*, *117* (15), 6959–6969. doi:10.1063/1.1505860

Hayot, F., & Jayaprakash, C. (2004). The Linear Noise Approximation for Molecular Fluctuations Within Cells. *Physical Biology*, *1* (4), 205–210. doi:10.1088/1478-3967/1/4/002

Heath, J., Kwiatkowska, M., Norman, G., Parker, D., & Tymchyshyn, O. (2008). Probabilistic Model Checking of Complex Biological Pathways. *Theoretical Computer Science*, *391* (3), 239–257. Converging Sciences: Informatics and Biology. doi:10.1016/j.tcs.2007.11.013

Heavenly Father of the NET. **(1997).** *NetWorker*.

Heinrich, R., Rapoport, S. M., & Rapoport, T. A. (1977). Metabolic Regulation and Mathematical Models. *Progress in Biophysics and Molecular Biology*, *32* (1), 1–82.

Heinrich, R., & Schuster, S. (1996). *The Regulation of Cellular Systems*. Springer.

Hell, P., & Nešetřil, J. (2004). *Graphs and Homomorphisms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press.

Henderson, T. R., Roy, S., Floyd, S., & Riley, G. F. (2006). NS-3 project goals. In *Proc. 2006 Workshop on NS-2: The IP Network Simulator (WNS2 '06)*. Pisa, Italy: ACM. doi:10.1145/1190455.1190468

Hinton, A., Kwiatkowska, M., Norman, G., & Parker, D. (2006). PRISM: A Tool for Automatic Verification of Probabilistic Systems. In H. Hermanns & J. Palsberg (Eds.), *Proc. 12th International Conference on Tools* *and Algorithms for the Construction and Analysis of Systems (TACAS'06)* (Vol. 3920, pp. 441–444). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. doi:10.1007/11691372_29

Hjelmfelt, A., Weinberger, E. D., & Ross, J. (1991). Chemical Implementation of Neural Networks and Turing Machines. *Proc. National Academy of Sciences USA*, *88*, 10983–10987. Retrieved from http://www.pnas.org/content/88/24/10983.full.pdf+html

Hoare, C. A. R. (1978). Communicating Sequential Processes. *Communications of the ACM*, *21* (8), 666–677. doi:10.1145/359576.359585

Hofmeyr, J. H. S. (2001). Metabolic control analysis in a nutshell. In T. M. Yi, M. Hucka, M. Morohashi & H. Kitano (Eds.), *Proc. 2nd International Conference on Systems Biology* (pp. 291–300). Madison, WI, USA: Omnipress.

Hofstadter, D. (1979). *Gödel, Escher, Bach: An Eternal Golden Braid.* Basic Books.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems.* Cambridge, MA, USA: MIT Press.

Holzmann, G. J. (1981). *PAN - a Protocol Specification Analyzer* (Technical Memorandum No. TM81-11271-5). Bell Laboratries. Retrieved 20 October 2010, from http://spinroot.com/spin/Doc/pan81.pdf

Holzmann, G. J. (1991). *Design and Validation of Computer Protocols.* Englewood Cliffs, NJ, USA: Prentice-Hall.

Holzmann, G. J. (1997). The Model Checker SPIN. *IEEE Transaction on Software Engineering*, *23* (5), 1–17. doi:10.1109/32.588521

Horn, F. (1973a). On a Connexion between Stability and Graphs in Chemical Kinetics. I. Stability and the Reaction Diagram. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, *334* (1598), 299–312. doi:10.1098/rspa.1973.0093

Horn, F. (1973b). On a Connexion between Stability and Graphs in Chemical Kinetics. II. Stability and the Complex Graph. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, *334* (1598), 313–330. doi:10.1098/rspa.1973.0094

Horn, F. (1973c). Stability and Complex Balancing in Mass-Action Systems with Three Short Complexes. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, *334* (1598), 331–342. doi:10.1098/rspa.1973.0095

Horn, F., & Jackson, R. (1972). General Mass Action Kinetics. *Archive for Rational Mechanics and Analysis*, *47* (2). doi:10.1007/BF00251225

Hsieh, Y.-W., & Levitan, S. P. (1998). Model Abstraction for Formal Verification. *Proc. European Conference on Design, Automation and Test.* doi:10.1109/DATE.1998.655848

Hu, X.-m., Zhang, J., Xiao, J., & Li, Y. (2008). Protein Folding in Hydrophobic-Polar Lattice Model: A Flexible Ant-Colony Optimization Approach. *Protein and Peptide Letters*, *15*, 469–477. doi:10.2174/092986608784567465

Huang, X., Alleva, F., Hon, H.-W., Hwang, M.-Y., Lee, K.-F., & Rosenfeld, R. (1993). The SPHINX-II Speech Recognition System: An Overview. *Computer Speech and Language*, *7* (2), 137–148. doi:10.1006/csla.1993.1007

Huet, G. (1980). Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *Journal of the ACM*, *27* (4), 797–821. doi:10.1145/322217.322230

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science and Engineering*, *9*, 90–95. doi:10.1109/MCSE.2007.55

Hurd, J. (2002). *Formal Verification of Probabilistic Algorithms.* (PhD Thesis, University of Cambridge, Cambridge, UK).

Hutton, T. J. (2002). Evolvable Self-Replicating Molecules in an Artificial Chemistry. *Artificial Life*, *8* (4), 341–356. doi:10.1162/106454602321202417

I

Ikegami, T. (1999). Evolvability of Machines and Tapes. *Artificial Life and Robotics*, *3*, 242–245. 10.1007/BF02481188.

Ikegami, T., & Hashimoto, T. (1996). Replication and Diversity in Machine-Tape Coevolutionary Systems. In C. G. Langton (Ed.), *Proc. 5th International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life V).*

Ingalls, B. (2004). A Frequency Domain Approach to Sensitivity Analysis of Biochemical Networks. *Journal of Physical Chemistry B*, *108* (3), 1143–1152.

Islam, S. M. S., Sqalli, M. S., & Kahn, S. (2006). Modeling and Formal Verification of DHCP Using SPIN. *International Journal of Computer Science and Applications*, *3* (6), 145–159.

J

Jackson, J. R. (1963). Jobshop-like Queueing Systems. *Management Science*, *10* (1), 131–142.

Jacobson, V. (1988). Congestion Avoidance and Control. In *Proc. Symposium on Communications Architectures and Protocols* (pp. 314–329). Stanford, CA, USA: ACM. doi:10.1145/52324.52356

Jacobson, V. (2006). A New Way to look at Networking. Retrieved 8 October 2010, from http://video.google.com/videoplay?docid=-6972678839686672840#

Jacobson, V., Braden, R., & Borman, D. (1992). TCP Extensions for High Performance. RFC 1323 (Proposed Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc1323.txt

Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N., & Braynard, R. (2009). Networking named content. In *Proc. 5th ACM International Conference on Emerging Networking Experiments and Technologies (CONEXT 2009)* (pp. 1–12).

Jahnke, T., & Huisinga, W. (2007). Solving the Chemical Master Equation for Monomolecular Reaction Systems Analytically. *Journal of Mathematical Biology*, *54*, 1–26.

Jain, R. K., Chiu, D.-M. W., & Hawe, W. R. (1984). *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems* (Research Report No. TR-301).

Jelasity, M., Montresor, A., & Babaoglu, O. (2005). Gossip-Based Aggregation in Large Dynamic Networks. *ACM Transactions on Computer Systems*, *23* (3), 219–252. doi:10.1145/1082469.1082470

Jelasity, M., Montresor, A., & Babaoglu, O. (2009). T-Man: Gossip-Based Fast Overlay Topology Construction. *Computer Networks*, *53* (13), 2321–2339. doi:10.1016/j.comnet.2009.03.013

Jiang, Y., & Liu, Y. (2008). *Stochastic Network Calculus*. Berlin / Heidelberg: Springer.

Johnson, B. W. (1996). An Introduction to the Design and Analysis of Fault-Tolerant Systems. In *Fault-Tolerant Computer System Design* (pp. 1–87). Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Jonsson, B., Kreiker, J., & Kwiatkowska, M. (2010). Executive Summary. In *Quantitative and Qualitative Analysis of Network Protocols* (10051). Dagstuhl Seminar Proceedings. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Joobbani, R., & Siewiorek, D. (1985). WEAVER: A Knowledge-Based Routing Expert. (pp. 266–272). doi:10.1109/DAC.1985.1585951

Jung, C. G. (1933). *Modern Man in Search of a Soul* (W. S. Dell & C. F. Baynes, Trans.). London, UK: Kegan Paul.

Jung, C. G. (2005). *Modern Man in Search of a Soul* (2nd ed.). Routledge.

K

Kacser, H., & Burns, J. A. (1973). The Control of Flux. *Symposia of the Society for Experimental Biology*, *27*, 65–104.

Kaizer, J. (1987). *Statistical Thermodynamics of Nonequilibrium Processes*. New York, NY, USA: Springer-Verlag.

Kanada, Y. (1995). Combinatorial Problem Solving Using Randomized Dynamic Composition of Production Rules. (Vol. 1, p. 467). doi:10.1109/ICEC.1995.489193

Kapral, R., & Oppo, G.-L. (1986). Competition Between Stable States in Spatially-Distributed Systems. *Physica D: Nonlinear Phenomena*, *23* (1–3), 455–463. doi:10.1016/0167-2789(86)90151-X

Károlyi, G., Péntek, Á., Scheuring, I., Tel, T., & Toroczkai, Z. (2000). Chaotic Flow: The Physics of Species Coexistence. *Proc. National Academy of Sciences USA*, *97* (25). JSTOR: 2666426. Retrieved from http://www.jstor.org/stable/2666426

Kashiwagi, A., Urabe, I., Kaneko, K., & Yomo, T. (2006). Adaptive Response of a Gene Network to Environmental Changes by Fitness-Induced Attractor Selection. *PLOS ONE*, *1* (1), e49. doi:10.1371/journal.pone.0000049

Kauffman, S. A. (1993). *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press.

Kelly, F. P. (1976). Networks of Queues. *Advances in Applied Probability*, *8* (2), 416–432.

Kelly, F. P. (1979). *Reversibility and Stochastic Networks*. John Wiley and Sons Ltd. Retrieved from http://www.statslab.cam.ac.uk/~frank/BOOKS/kelly_book.html

Kelly, F. P. (1997). Charging and Rate Control for Elastic Traffic. *European Transactions on Telecommunications*, *8*, 33–37.

Kelly, F. P. (2003). Fairness and Stability of End-to-End Congestion Control. *European Journal of Control*, *9*, 159–176.

Kelly, F. P., Maulloo, A. K., & Tan, D. K. H. (1998). Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability. *The Journal of the Operational Research Society*, *49* (3). JSTOR: 3010473. Retrieved from http://www.jstor.org/stable/3010473

Kelly, K. (1995). *Out of Control: The New Biology of Machines, Social Systems & the Economic World*. Basic Books.

Kempe, D., Dobra, A., & Gehrke, J. (2003). Gossip-based Computation of Aggregate Information. In *Proc. 44th Annual IEEE Symposium on Foundations of Computer Science* (pp. 482–491).

Kendall, D. G. (1953). Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics*, *24* (3), 338–354.

Kephart, J. O., & Chess, D. M. (2003). The Vision of Autonomic Computing. *Computer*, *36* (1), 41–50.

Keromytis, A. D. (2007). Characterizing Self-Healing Systems. In *Proc. 4th International Conference on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS)*.

Kleene, S. C. (1938). On Notation for Ordinal Numbers. *Journal of Symbolic Logic*, *3* (4), 150–155.

Kohler, E., Handley, M., & Floyd, S. (2006). Datagram Congestion Control Protocol DCCP. RFC 4340 (Proposed Standard). Updated by RFCS 5595, 5596. Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc4340.txt

Kolmogorov, A. (1954). *Limit Distributions for Sums of Independent Random Variables*. Addison-Wesley.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Krishnamachari, B., Estrin, D., & Wicker, S. (2002). *Modelling Data Centric Routing in Wireless Sensor Networks* (Technical Report No. CENG 02-14).

Kvasnička, V., & Pospichal, J. (2001). Autoreplicators and Hypercycles in Typogenetics. *Journal of Molecular Structure: THEOCHEM*, *547* (1–3), 119–138. doi:10.1016/S0166-1280(01)00464-X

Kwiatkowska, M., Norman, G., Segala, R., & Sproston, J. (2002). Automatic Verification of Real-Time Systems with Discrete Probability Distributions. *Theoretical Computer Science*, *282*, 101–150.

L

Lafontaine, D. L. J., & Tollervey, D. (2001). The Function and Synthesis of Ribosomes. *Nature Reviews. Molecular Cell Biology*, *2* (7), 514–520.

Laing, R. (1977). Automaton Models of Reproduction by Self-Inspection. *Journal of Theoretical Biology*, *66* (3), 437–456. doi:10.1016/0022-5193(77)90294-6

Lancet, D., Sadovsky, E., & Seidemann, E. (1993). Probability Model for Molecular Recognition in Biological Receptor Repertoires: Significance to the Olfactory System. *Proc. National Academy of Sciences USA*, *90*, 3715–3719.

Langton, C. G. (1984). Self-Reproduction in Cellular Automata. *Physica D: Nonlinear Phenomena*, *10* (1–2), 135–144.

Larkin, J., & Stocks, P. (2004). Self-Replicating Expressions in the Lambda Calculus. In *Proc. 27th Australian Conference on Computer Science* (Vol. 26, pp. 167–173). Dunedin, New Zealand: Australian Computer Society, Inc.

Le Boudec, J.-Y., & Thiran, P. (2001). *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet.* Lecture Notes in Computer Science. Berlin / Heidelberg: Springer.

Le Novère, N., & Shimizu, T. S. (2001). STOCHSIM: Modelling of Stochastic Biomolecular Processes. *Bioinformatics*, *17* (6), 575–576. doi:10.1093/bioinformatics/17.6.575

Leibnitz, K., Wakamiya, N., & Murata, M. (2006). Resilient Multi-path Routing Based on a Biological Attractor Selection Scheme. In *Proc. 2nd International Workshop on Biologically Inspired Approaches to Advanced Information Technology (BIOADIT 2006)* (Vol. 3853, pp. 48–63). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer.

Lieberman, E., Hauert, C., & Nowak, M. A. (2005). Evolutionary Dynamics on Graphs. *Nature*, *433*, 312–316.

Lotka, A. J. (1910). Contribution to the Theory of Periodic Reaction. *Journal of Physical Chemistry*, *14* (3), 271–274.

Low, S. H., Peterson, L. L., & Wang, L. (2002). Understanding TCP Vegas: A Duality Model. *Journal of the ACM*, *49* (2), 207–235. doi:10.1145/506147.506152

Luby, M., Shokrollahi, A., Watson, M., & Stockhammer, T. (2007). Raptor Forward Error Correction Scheme for Object Delivery. RFC 5053 (Proposed Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc5053.txt

Lüscher, P. (2009). *Applying Ant Colony Optimization Methods in an Artificial Chemistry Context to Routing Problems.* (Master Thesis, University of Basel). Retrieved 3 October 2010, from http://cn.cs.unibas.ch/pub/doc/2009-msthLuescher.pdf

M

Mackert, L. F., & Neumeier-Mackert, I. B. (1987). Communicating Rule Systems. In *Proc. IFIP WG6.1 7th International Conference on Protocol Specification, Testing and Verification* (pp. 77–88). Amsterdam: North-Holland Publishing Co.

Maggi, P., & Sisto, R. (2002). Using SPIN to Verify Security Properties of Cryptographic Protocols. In *Proc. 9th Inernational Workshop on Model Checking Software* (Vol. 2318, pp. 85–87). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer.

Mairesse, J., & Nguyen, H.-T. (2009). Deficiency Zero Petri Nets and Product Form. *arXiv.org cs.DM, 0905.3158v2.*

Mann, T. (1924). *Der Zauberberg.* S. Fischer.

Mansour, M. M., Van Den Broeck, C., Nicolis, G., & Turner, J. W. (1981). Asymptotic Properties of Markovian Master Equations. *Annals of Physics*, *131* (2), 283–313. doi:10.1016/0003-4916(81)90033-6

Martens, D., De Backer, M., Haesen, R., Vanthienen, J., Snoeck, M., & Baesens, B. (2007). Classification With Ant Colony Optimization. *IEEE Transactions on Evolutionary Computation*, *11* (5), 651–665. doi:10.1109/TEVC.2006.890229

Martin, W., & Russell, M. J. (2003). On the Origins of Cells. *Philosophical Transaction of the Royal Society of London. Series B, Biological Sciences Trans. R. Soc. Lond. B. Biol. Sci. 358* (1429). doi:10.1098/rstb.2002.1183

Mathis, M., Mahdavi, J., Floyd, S., & Romanow, A. (1996). TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc2018.txt

Matplotlib Web Page. (2010). Retrieved 3 October 2010, from http://matplotlib.sourceforge.net

Matsumaru, N., Lenser, T., Hinze, T., & Dittrich, P. (2007). Toward Organization-Oriented Chemical Programming: A Case Study with the Maximal Independent Set Problem. In F. Dressler & I. Carreras (Eds.), *Advances in Biologically Inspired Information Systems* (Vol. 69, pp. 149–165). Studies in Computational Intelligence. Berlin / Heidelberg: Springer. doi:10.1007/978-3-540-72693-7_8

Maynard Smith, J, & Price, G. R. (1973). The Logic of Animal Conflict. *Nature*, *246*, 15–18.

McCraig, C. (2007). *From Individuals to Populations: Changing Scale in Process Algebra Models of Biological Systems.* (PhD Thesis, University of Stirling).

McIlroy, M. D. (1964). The Origin of Unix Pipes. Memorandum. Retrieved 6 August 2010, from http://doc.cat-v.org/unix/pipes/

McKay, R. I., & Essam, D. (2001). Evolving Self-Reproducing Programs. In *Proc. Inaugural Workshop on Artificial Life (AL'01)*.

McQuarrie, D. A. (1967). Stochastic Approach to Chemical Kinetics. *Journal of Applied Probability*, *4* (3), 413–478.

McQuarrie, D. A. (1997). *Physical Chemistry*. University Science Books. Retrieved from http://books.google.com/books?id=f-bje0-DEYUC

Meshoul, S., & Batouche, M. (2002). Ant Colony System with Extremal Dynamics for Point Matching and Pose Estimation. In *Proc. 16th International Conference on Pattern Recognition (ICPR'02)* (Vol. 3, p. 30823). Washington, DC, USA: IEEE Computer Society.

Metzner, P., Schütte, C., & Vanden-Eijnden, E. (2009). Transition Path Theory for Markov Jump Processes. *Multiscale Modeling & Simulation*, *7* (3), 1192–1219. doi:10.1137/070699500

Meyer, A. R., & Stockmeyer, L. J. (1972). The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. In *Proc. 13th on Switching and Automata Theory (SWAT '72)* (pp. 125–129). Washington, DC, USA: IEEE Computer Society. doi:10.1109/SWAT.1972.29

Michaelis, L., & Menten, M. (1913). Die Kinetik der Invertinwirkung. *Biochemische Zeitschrift*, *49*, 333–369. Retrieved 13 August 2010, from http://web.lemoyne.edu/~giunta/menten.html

Mills, J. W. (2008). The Nature of the Extended Analog Computer. *Physica D: Nonlinear Phenomena*, *237* (9), 1235–1256. doi:10.1016/j.physd.2008.03.041

Milner, R. (1980). *A Calculus of Communicating Systems*. Springer.

Minsky, M. (1967). *Computation: Finite and Infinite Machines*. Englewood Cliffs, NJ, USA: Prentice-Hall.

Monti, M. (2010). *A Signaling Processing Approach to the Analysis of Chemical Networking Protocols*. (Master Thesis, University of Basel and University of Pisa). Retrieved from http://cn.cs.unibas.ch/pub/doc/2010-msthMonti.pdf

Moran, P. A. P. (1958). Random Processes in Genetics. *Mathematical Proceedings of the Cambridge Philosophical Society*, *54*, 60–71. doi:10.1017/S0305004100033193

Mossel, E., & Steel, M. (2005). Random Biochemical Networks: The Probability of Self-Sustaining Autocatalysis. *Journal of Theoretical Biology*, *233* (3), 327–336. doi:10.1016/j.jtbi.2004.10.011

N

Nagle, J. (1984). Congestion Control in IP/TCP Internetworks. RFC 896. Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc896.txt

Nagpal, R., Kondacs, A., & Chang, C. (2003). Programming Methodology for Biologically-Inspired Self-Assembling Systems. In *AAAI Spring Symposium on Computational Synthesis*.

Neuts, M. F. (1981). *Matrix-Geometric Solutions in Stochastic Models.* New York, NY, USA: Dover Publications Inc.

Nipkow, T., Paulson, L. C., & Wenzel, M. (2002). *Isabelle/HOL — A Proof Assistant for Higher-Order Logic.* Lecture Notes in Computer Science. Springer.

Normand, E. (1996). Single Event Upset at Ground Level. *IEEE Transactions on Nuclear Science*, *43* (6), 2742–2750.

Nowak, M. A. (2006). *Evolutionary Dynamics: Exploring the Equations of Life.* Cambridge, MA, USA: Belknap.

O

Oh, N., Shirvani, P. P., & McCluskey, E. J. (2002). Error detection by duplicated instructions in super-scalar processors. *IEEE Transactions on Reliability*, *51* (1), 63–75. doi:10.1109/24.994913

Øksendal, B. K. (2003). *Stochastic Differential Equations: An Introduction with Applications.* Berlin: Springer.

Ong, L., & Yoakum, J. (2002). An Introduction to the Stream Control Transmission Protocol (SCTP). RFC 3286 (Informational). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc3286.txt

Otto, S. P., & Day, T. (2007). *A Biologist's Guide to Mathematical Modeling in Ecology and Evolution.* Princeton, NJ, USA: Princeton University Press.

Owre, S., Rajan, S., Rushby, J. M., Shankar, N., & Srivas, M. K. (1996). PVS: Combining Specification, Proof Checking, and Model Checking. In *Proc. 8th International Conference on Computer Aided Checking (PAV'96)* (Vol. 1102, pp. 411–414). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer.

P

Paladugu, S., Chickarmane, V., Deckard, A., Frumkin, J., McCormack, M., & Sauro, H. (2006). In Silico Evolution of Functional Modules in Biochemical Networks. *IEEE Proc. Systems Biology*, *153* (4), 223–235. doi:10.1049/ip-syb:20050096

Parnas, D. L. (2007). Use the Simplest Model, But Not Too Simple: Forum. *Communications of the ACM*, *50* (6), 7–9. doi:10.1145/1247001.1247014

Paulsson, J. (2004). Summing Up the Noise in Gene Networks. *Nature*, *427* (6973), 415–418.

Paulsson, J., Berg, O. G., & Ehrenberg, M. (2000). Stochastic Focusing: Fluctuation-enhanced Sensitivity of Intracellular Regulation. *Proc. National Academy of Sciences USA*, *97* (13), 7148–7153.

Paulsson, J., & Ehrenberg, M. (2000). Random Signal Fluctuations Can Reduce Random Fluctuations in Regulated Components of Chemical Regulatory Networks. *Physical Review Letters*, *84* (23), 5447–5450. doi:10.1103/PhysRevLett.84.5447

Paulsson, L. C. (1988). Isabelle: The Next Seven Hundred Theorem Provers. In E. L. Lusk & R. A. Overbeek (Eds.), *Proc. 9th International Conference on Automated Deduction* (Vol. 310, pp. 772–773). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer.

Paulsson, L. C. (1998). The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, *6*, 85–128.

Paŭn, G. (2000). Computing with Membranes. *Journal of Computer and System Sciences*, *61* (1), 108–143.

Pedraza, J. M., & van Oudenaarden, A. (2005). Noise Propagation in Gene Networks. *Science*, *307* (5717), 1965–1969.

Peled, D. A. (1993). All from One, One for All: On Model Checking Using Representatives. In *Proc. 5th International Conference on Computer Aided Verification (CAV '93)* (Vol. 697, pp. 409–423). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer.

Pereira, J. (2005). A Biochemistry-Inspired Artificial Chemistry: LAC. In *Proc. Portuguese Conference on Artificial Intelligence (EPIA 2005)* (pp. 79–84). doi:10.1109/EPIA.2005.341269

Perelson, A. S., & Oster, G. F. (1974). Chemical Reaction Dynamics Part II: Reaction Networks. *Archive for Rational Mechanics and Analysis*, *57* (1), 31–98. doi:10.1007/BF00287096

Perrier, J.-Y., Sipper, M., & Zahnd, J. (1996). Toward a Viable, Self-Reproducing Universal Computer. *Physica D*, *97*, 335–352. doi:10.1016/0167-2789(96)00091-7

Pfeifer, R., Lungarella, M., & Iida, F. (2007). Self-Organization, Embodiment, and Biologically Inspired Robotics. *Science*, *317* (5853), 1088–1093.

Plummer, D. (1982). Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826 (Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc826.txt

Pnueli, A. (1977). The Temporal Logic of Programs. In *Proc. 18th IEEE Symposium on Foundations of Computer Science* (pp. 46–57). doi:10.1109/SFCS.1977.32

Polinière, P. (1728). *Expériences de physique* (3rd ed.). Paris, France: Charles Moette.

Post, E. (1943). Formal Reductions of the Combinatorial Decision Problem. *American Journal of Mathematics*, *65*, 197–215.

Postel, J. (1981). Transmission Control Protocol. RFC 793 (Standard). Updated by RFCS 1122, 3168. Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc793.txt

Pradhan, D. K. (1996). *Fault-Tolerant Computer System Design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Q

Quick, T., Dautenhahn, K., Nehaniv, C. L., & Roberts, G. (2000). The Essence of Embodiment: A Framework for Understanding and Exploiting Structural Coupling Between System and Environment. In D. M. Dubois (Ed.), *Proc. 3rd International Conference on Computing Anticipatory Systems (CASYS'99)* (Vol. 517, *1*, pp. 649–660). AIP. doi:10.1063/1.1291299

R

Ramakrishnan, K., Floyd, S., & Black, D. (2001). The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc3168.txt

Ramakrishnan, N., & Bhalla, U. S. (2008). Memory Switches in Chemical Reaction Space. *PLOS Computational Biology*, *4* (7), e1000122. doi:10.1371/journal.pcbi.1000122

Rao, C. V., Sauro, H. M., & Arkin, A. P. (2004). Putting the "Control" in Metabolic Control Analysis. In *Proc. 7th International Symposium on Dynamics and Control of Process Systems (DYCOPS 7)*.

Rapoport, A. (1962). *An Essay on Mind*. Free Press.

Rathinam, M., Petzold, L. R., Cao, Y., & Gillespie, D. T. (2003). Stiffness in Stochastic Chemically Reacting Systems: The Implicit Tau-Leaping Method. *Journal of Chemical Physics*, *119* (24), 12784–12794. doi:10.1063/1.1627296

Reder, C. (1988). Metabolic Control Theory: A Structural Approach. *Journal of Theoretical Biology*, *135* (2), 175–201. doi:10.1016/S0022-5193(88)80073-0

Reis, G., Chang, J., Vachharajani, N., Rangan, R., & August, D. (2005). SWIFT: Software Implemented Fault Tolerance. (pp. 243–254). doi:10.1109/CGO.2005.34

Reis, G. A., Chang, J., & August, D. I. (2007). Automatic Instruction-Level Software-Only Recovery. *IEEE Micro*, *27* (1), 36–47. doi:10.1109/MM.2007.4

Rekhter, Y., Li, T., & Hares, S. (2006). A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc4271.txt

Rosen, E., Viswanathan, A., & Callon, R. (2001). Multiprotocol Label Switching Architecture. RFC 3031 (Proposed Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc3031.txt

Rubel, L. A. (1993). The Extended Analog Computer. *Advances in Applied Mathematics*, *14* (1), 39–50. doi:10.1006/aama.1993.1003

S

Saerens, M., Fouss, F., Yen, L., & Dupont, P. (2004). The Principal Components Analysis of a Graph, and Its Relationships to Spectral Clustering. In J.-F. Boulicaut, F. Esposito, F. Giannotti & D. Pedreschi (Eds.), *Proc. 15th European Conference on Machine Learning (ECML 2004)* (Vol. 3201, pp. 371–383). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer.

Samant, A., & Vlachos, D. G. (2005). Overcoming Stiffness in Stochastic Simulation Stemming from Partial Equilibrium: A Multiscale Monte Carlo Algorithm. *Journal of Chemical Physics*, *123* (14). doi:10.1063/1.2046628

Samios, C. B., & Vernon, M. K. (2003). Modeling the Throughput of TCP Vegas. In *Proc 2003 International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS 2003)* (pp. 71–81). San Diego, CA, USA: ACM. doi:10.1145/781027.781037

Sander, T., & Tschudin, C. (1998). Towards Mobile Cryptography. (pp. 215–224). doi:10.1109/SECPRI.1998.674837

Sastry, N. R., & Lam, S. S. (2005). CYRF: A Theory of Window-Based Unicast Congestion Control. *IEEE/ACM Transactions on Networking*, *13* (2), 330–342. doi:10.1109/TNET.2005.845545

Scafetta, N., & West, B. J. (2007). Probability Distributions in Conservative Energy Exchange Models of Multiple Interacting Agents. *Journal of Physics: Condensed Matter*, *19* (6), 065138 (18pp). Retrieved from http://stacks.iop.org/0953-8984/19/065138

Schneider, J. M., Mackert, L. F., Zörntlein, G., Velthuys, R. J., & Bär, U. (1992). An Integrated Environment for Developing Communication Protocols. *Computer Networks and ISDN Systems*, *25* (1), 43–61. Formal Description Techinique (FDT) Concepts and Tools. doi:10.1016/0169-7552(92)90123-8

Schreckling, D., & Marktscheffel, T. (2010). An Artificial Immune System Approach for Artificial Chemistries Based on Set Rewriting. In E. Hart, C. McEwan, J. Timmis & A. Hone (Eds.), *Artificial Immune Systems* (Vol. 6209, pp. 250–263). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. doi:10.1007/978-3-642-14547-6_20

Scott, M., Ingalls, B., & Kærn, M. (2006). Estimations of Intrinsic and Extrinsic Noise in Models of Nonlinear Genetic Networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *16* (2), 026107. doi:10.1063/1.2211787

Shaw, M. (2002). "Self-Healing": Softening Precision to Avoid Brittleness: Position Paper. In *Proc. 1st Workshop on Self-Healing Systems (WOSS '02)* (pp. 111–114). Charleston, SC, USA: ACM. doi:10.1145/582128.582152

Shibata, T., & Ueda, M. (2008). Noise Generation, Amplification and Propagation in Chemotactic Signaling Systmes of Living Cells. *Biosystems*, *93* (1–2), 126–132.

Shirvani, P., Saxena, N., & McCluskey, E. J. (2000). Software-implemented EDAC protection against SEUs. *IEEE Transactions on Reliability*, *49* (3), 273–284. doi:10.1109/24.914544

Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: MYCIN*. New York, NY, USA: America Elsevier Publishing Company, Inc.

Simon, H. A. (1996). *The Sciences of the Artificial* (3rd ed.). Cambridge, MA, USA: MIT Press.

Sipper, M. (1998). Fifty Years of Research on Self-Replication: An Overview. *Artificial Life*, *4*, 237–257. doi:10.1162/106454698568576

Slayman, C. W. (2005). Cache and Memory Error Detection, Correction, and Reduction Techniques for Terrestrial Servers and Workstations. *IEEE Transactions on Device and Materials Reliability*, *5* (3), 397–404. doi:10.1109/TDMR.2005.856487

Spector, L., Perry, C., Klein, J., & Keijzer, M. (2004). *Push 3.0 Programming Language Description* (Technical Report No. HC-CSTR-2004-02). Amherst, MA, USA.

Speroni di Fenizio, P., & Banzhaf, W. (2000). A Less Abstract Aritifical Chemistry. In M. A. Bedau, J. S. McCaskill, N. H. Packard & S. Rasmussen (Eds.), *Proc. 7th International Conference on Artificial Life (Artificial Life VII)*. Cambridge, MA, USA: MIT Press.

Stadler, P. F., Fontana, W., & Miller, J. H. (1993). Random catalytic reaction networks. *Physica D*, *63* (3–4), 378–392. doi:10.1016/0167-2789(93)90118-K

Stepney, S. (2010). Non-Classical Computation: A Dynamical Systems Perspective. In G. Rozenberg, T. Bäck & J. N. Kok (Eds.), *Handbook of Natural Computing, Volume II* (Chap. 52). Berlin / Heidelberg: Springer.

Sterbenz, J. (2002). Intelligence in Future Broadband Networks: Challenges and Opportunities in High-Speed Active Networking. (pp. 2–1–2–7). doi:10.1109/IZSBC.2002.991742

Stevens, W. (1997). TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001 (Proposed Standard). Obsoleted by RFC 2581. Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc2001.txt

Stewart, R. (2007). Stream Control Transmission Protocol. RFC 4960 (Proposed Standard). Internet Engineering Task Force. IETF. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc4960.txt

Strogatz, S. H. (1994). *Nonlinear Dynamics and Chaos*. Studies in Nonlinearity. Westview Press.

Sunkara, V. (2009). The Chemical Master Equation with Respect to Reaction Counts. In *Proc. 18th World IMACS/MODSIM Congress*.

Suzuki, H., & Dittrich, P. (2009). Artificial Chemistry. *Artificial Life*, *15* (1), 1–3. doi:10.1162/artl.2009.15.1.15100

Suzuki, H., & Ono, N. (2002). String Rewriter that Allows the Maintenance of Different Types of Self-Replicators. In *Proc. 5th International Conference on Humans and Computers (HC-2002)* (pp. 171–178).

Szathmáry, E. (1991). Simple Growth Laws and Selection Consequences. *Trends in Ecology and Evolution*, *6* (11), 366–370. doi:10.1016/0169-5347(91)90228-P

T

Takács, L. (1979). On an Urn Problem of Paul and Tatiana Ehrenfest. *Mathematical Proceedings of the Cambridge Philosophical Society*, *86*, 127–130.

Takeuchi, N., & Hogeweg, P. (2008). Evolution of Complexity in RNA-like Replicator Systems. *Biology Direct*, *3*. doi:10.1186/1745-6150-3-11

Tanenbaum, A. (2002). *Computer Networks*. Prentice Hall.

Teich, M. (1992). *Documentary History of Biochemistry, 1770–1940*. Rutherford, NJ, USA: Fairleigh Dickinson University Press.

Tempesti, G., Mange, D., & Stauffer, A. (1998). Self-Replicating and Self-Repairing Multicellular Automata. *Artificial Life*, *4*, 259–282. doi:10.1162/106454698568585

Tennenhouse, D., Lampson, B., Gillett, S. E., & Klein, J. S. (1996). Virtual Infrastructure: Putting Information Infrastructure on the Technology Curve. *Computer Networks and ISDN Systems*, *28* (13), 1769–1790. doi:10.1016/0169-7552(96)00009-8

Tennenhouse, D. L., & Wetherall, D. J. (1996). Towards an Active Network Architecture. *SIGCOMM Computer Communication Review*, *26* (2), 5–17. doi:10.1145/231699.231701

Teuscher, C. (2007). From Membranes to Systems: Self-Configuration and Self-Replication in Membrane Systems. *Biosystems*, *87* (2–3), 101–110. Papers presented at the 6th International Workshop on Information Processing in Cells and Tissues, York, UK, 2005. doi:10.1016/j.biosystems.2006.09.002

The Open Group. (2008). pipe — create an interprocess channel. The Open Group. Retrieved 6 October 2010, from The Open Group: http://www.opengroup.org/onlinepubs/9699919799/functions/pipe.html

Thompson, G. P. (2010). The Quine Page (Self-reproducing Code). Retrieved 18 June 2010, from http://www.nyx.net/~gthompso/quine.htm

Thrun, S., Fox, D., Burgard, W., & Dellaert, F. (2000). Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence*, *128* (1–2), 99–141.

Tian, T., & Burrage, K. (2004). Binomial Leap Methods for Simulating Stochastic Chemical Kinetics. *Journal of Chemical Physics*, *121* (21). doi:10.1063/1.1810475

Timmis, J., Knight, T., De Castro, L. N., & Hart, E. (2004). An Overview of Artificial Immune Systems. In R. Paton, H. Bolouri, M. Holcombe,

J. H. Parish & R. Tateson (Eds.), *Computation in Cells and Tissues: Perspectives and Tools for Thought* (pp. 51–86). Natural Computation Series. Springer.

Tofts, C. (1994). Processes with Probabilities, Priority and Time. *Formal Aspects in Computing*, *6* (5), 536–564.

Tominaga, K., & Setomoto, M. (2008). An Artificial-Chemistry Approach to Generating Polyphonic Musical Phrases. In M. Giacobini, A. Brabazon, S. Cagnoni, G. Di Caro, R. Drechsler, A. Ekárt, . . . S. Yang (Eds.), *Applications of Evolutionary Computing* (Vol. 4974, pp. 463–472). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. doi:10.1007/978-3-540-78761-7_49

Tomioka, R., Kimura, H., Kobayashi, T. J., & Aihara, K. (2004). Multivariate Analysis of Noise in Genetic Regulatory Networks. *Journal of Theoretical Biology*, *229* (4), 501–521. doi:10.1016/j.jtbi.2004.04.034

Tschudin, C. (1993). *On the Structuring of Computer Communictions.* (PhD Thesis, University of Geneva). Retrieved 29 September 2010, from http://netresearch.ics.uci.edu/Previous_research_projects/agentos/related/messengers/phd-1.ps

Tschudin, C. (2003). Fraglets - a Metabolistic Execution Model for Communication Protocols. In *Proc. 2nd Annual Symposium on Autonomous Intelligent Networks and Systems (AINS)*.

Turing, A. M. (1952). A Chemical Basis for Morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B. Biological Sciences*, *237*, 37–72.

U

Ullah, M., & Wolkenhauer, O. (2009a). Investigating the Two-Moment Characterisation of Subcellular Biochemical Networks. *arXiv.org q-bio.SC*, *0809.0773v3*.

Ullah, M., & Wolkenhauer, O. (2009b). Stochastic Approaches in Systems Biology. *Wiley Interdisicplinary Reviews: Systems Biology and Medicine*.

Upadhyay, S. K. (2006). *Chemical Kinetics and Reaction Dynamics*. Springer.

V

van Kampen, N. G. (1976). The Expansion of the Master Equation. *Advances in Chemical Physics*, *34*.

van Kampen, N. G. (2007). *Stochastic Processes in Physics and Chemistry* (3rd ed.). Elsevier.

van Slyke, D. D., & Cullen, G. E. (1914). The Mode of Action of Urease and of Enzymes in General. *Journal of Biological Chemistry*, *19*, 141–180.

Varga, A. (2001). The OMNET++ Discrete Event Simulation System. In *Proc. European Simulation Multiconference (ESM 2001)*.

Varga, A. (2009). OMNeT++ Community Site. Retrieved 12 March 2010, from http://www.omnetpp.org

Varga, A., & Hornig, R. (2008). An Overview of the OMNET++ Simulation Environment. In *Proc. 1st Internation Conference on Simulation Tools and Techniques for Communications (SIMUTOOLS '08)* (pp. 1–10).

Volterra, V. (1926). Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Mem. Acad. Lincei Roma*, *2*, 31–113.

von Neumann, J. (1966). *Theory of Self-Reproducing Automata*. Champaign, IL, USA: University of Illinois Press.

W

Waage, P., & Guldberg, C. M. (1864). Studies Concerning Affinity. *Forhandlinger: Videnskabs - Selskabet i Christiania*, *35*.

Wagner, A. (2007). *Robustness and Evolvability in Living Systems* (S. A. Levin & S. H. Strogatz, Eds.). Princeton Studies in Complexity. Princeton University Press.

Wagner, K. (2000). Cooperative Strategies and the Evolution of Communication. *Artificial Life*, *6* (2), 149–179. doi:10.1162/106454600568384

Wallace, E. W. J. (2010). A Simplified Derivation of Van Kampen's System Size Expansion. *arXiv.org q-bio.QM*, *1004.4280v2*.

Weeks, A., & Stepney, S. (2005). Artificial Catalysed Reaction Networks for Search. In *ECAL Workshop on Artificial Chemistry*.

Weise, T., Zapf, M., & Geihs, K. (2007). Rule-based Genetic Programming. In *Proc. 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems (BIONETICS 2007)* (pp. 8–15). doi:10.1109/BIMNICS.2007.4610073

Weise, T. (2009). *Evolving Distributed Algorithms with Genetic Programming*. (Doctoral dissertation, Distributed Systems Group, FB 16, University of Kassel, Wilhelmshöher Allee 73, 34121 Kassel, Germany).

Wibling, O., Parrow, J., & Pears, A. (2004). Automatized Verification of Ad Hoc Routing Protocols. In *Proc. 24th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems*

(FORTE 2004) (Vol. 3235, pp. 343–358). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer.

Wilfredo, T.-P. (2000). *Software Fault Tolerance: A Tutorial.*

Wolkenhauer, O., Ullah, M., Kolch, W., & Cho, K.-H. (2004). Modeling and Simulation of Intracellular Dynamics: Choosing an Appropriate Framework. *IEEE Transactions on Nanobioscience*, *3* (3), 200–207.

Wong, V., & Horowitz, M. (2006). Soft Error Resilience of Probabilistic Inference Applications. In *Proc. 2nd Workshop on System Effects of Logic Soft Errors (SELSE)*.

Wright, S. (1931). Evolution in Mendelian Populations. *Genetics*, *16* (2), 97–159.

Y

Yamamoto, L. (2010). Evaluation of a Catalytic Search Algorithm. In J. González, D. Pelta, C. Cruz, G. Terrazas & N. Krasnogor (Eds.), *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)* (Vol. 284, pp. 75–87). Studies in Computational Intelligence. Berlin / Heidelberg: Springer. doi:10.1007/978-3-642-12538-6_7

Yamamoto, L., & Banzhaf, W. (2010). Catalytic Search in Dynamic Environments. In H. Fellermann, M. Dörr, M. M. Hanczyc, L. L. Laursen, S. Maurer, D. Merkle, . . . S. Rasmussen (Eds.), *Proc. 12th International Conference on the Simulation and Synthesis of Living Systems (Artificial Life XII)* (pp. 277–284). Retrieved 24 September 2010, from http://mitpress.mit.edu/books/chapters/0262290758chap53.pdf

Yamamoto, L., Schreckling, D., & Meyer, T. (2007). Self-Replicating and Self-Modifying Programs in Fraglets. In *Proc. 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems (BIONETICS 2007)*.

Z

Zeh, H.-D. (1980). *Information and Determinism: Epist. Letters.* Ferdinand Gonseth Association.

Zhang, J., Chung, H.-H., Lo, A.-L., & Huang, T. (2009). Extended Ant Colony Optimization Algorithm for Power Electronic Circuit Design. *IEEE Transactions on Power Electronics*, *24* (1), 147 –162. doi:10.1109/TPEL.2008.2006175

Ziegler, J., & Banzhaf, W. (2001). Evolving Control Metabolisms for a Robot. *Artificial Life*, *7* (2), 171–190. doi:10.1162/106454601753138998

Zwanzig, R. (2001). A Chemical Langevin Equation with Non-Gaussian Noise. *Journal of Physical Chemistry*, *105*, 6472–6473.

Index