Université de Grenoble

HABILITATION A DIRIGER DES RECHERCHES

présentée par

Gregor GÖSSLER

Formal Techniques for Component-based Design of Embedded Systems

Date de soutenance : 25 septembre 2014

Composition du jury :

Rapporteurs :	Farhad Arbab	U. Leiden / CWI
	Thomas A. Henzinger	IST Austria
	Claude Jard	U. Nantes
Examinateurs :	Jean-Claude Fernandez	U. Grenoble
	Oded Maler	CNRS

Résumé

Le cadre à composants Bip

Après mon arrivée à l'Inria j'ai commencé à développer, avec Joseph Sifakis, le cadre à composants Bip. Nos objectifs principaux lors de la conception de Bip étaient de développer un cadre à composants général et expressif qui supporte des modèles *hétérogènes* en mettant en œuvre une *séparation des préoccupations* [154].

L'hétérogénéité signifie que les composants interagissent à travers différents *modèles d'interaction* (comme la diffusion ou le rendez-vous n-aire) et *modèles d'exécution* (ou politiques d'ordonnancement). L'hétérogénéité est essentielle pour construire des systèmes en utilisant ou en raffinant des composants à différents niveaux d'abstraction ou sur différentes plates-formes. Par exemple, à un niveau abstrait, des composants peuvent communiquer à travers un rendez-vous qui est ensuite raffiné en un protocole de communication.

La séparation des préoccupations est un prérequis pour que le cadre supporte l'hétérogénéité, en rendant explicites la description et la composition des différents aspects du modèle.

Chaque composant Bip consiste en trois couches qui décrivent séparément son *comportement* (**B**ehavior), le *modèle d'interaction* entre sous composants (**I**nteraction model) [154], et le *modèle d'exécution* spécifiant des contraintes d'ordonnancement (**P**riorities) [153]. Bip supporte deux opérations sur les composants : la *composition* et la *restriction*. La composition de composants Bip est à nouveau un composant Bip, de même que la restriction d'un composant par un modèle d'exécution. Des introductions plus détaillées se trouvent dans [138] et [24].

Contrairement à d'autres cadres à composants avec un modèle d'interactions fixe, la modélisation directe d'interactions non triviales dans un formalisme déclaratif de haut niveau permet de modéliser de manière concise des schémas de coordination complexes [25, 23]. A partir du modèle d'interaction une implémentation correcte peut être générée automatiquement [22], ce qui réduit le risque d'erreurs.

J'ai montré que le modèle d'exécution de Bip est un moyen simple de modéliser des schémas de coordination non triviaux comme les réseaux de régulation géniques [141].

Coordination de systèmes à composants

La construction à base de composants — motivée par l'objectif de construire des systèmes complexes à partir de composants plus simples — nécessite que le concepteur soit capable de garantir la correction du système à partir de la correction de ses composants et de la manière dont ils sont composés. Cet axe de mon travail aborde la question de comment vérifier et coordonner le comportement de composants, dans différents cadres, de manière à garantir la correction globale du système par rapport à une spécification.

Plus concrètement, j'ai développé des techniques pour l'analyse compositionnelle de l'absence de blocages [151, 144], du progrès [144], de la vivacité [143] et de l'atteignabilité [141] en Bip, et je les ai implémentées dans l'outil Prometheus [138].

Ensuite j'ai étudié le problème de synthétiser des composants adaptateurs afin de faire communiquer un ensemble donné de composants — potentiellement incompatibles — qui communiquent de manière asynchrone à travers des tampons FIFO, de manière à éviter les blocages et à garantir des propriétés temps-réel et l'utilisation finie des tampons [257]. Ce travail a été réalisé dans le cadre du projet Européen Artist 2.

Finalement je me suis intéressé au raffinement compositionnel d'un contrôleur discret C pour un système Bip S en un contrôleur C' pour un système cible S' tel que S' contrôlé par C' se comporte comme S contrôlé par C. La notion de *mapping de stratégies* permet de réutiliser automatiquement un contrôleur lorsque le système contrôlé doit être raffiné [140].

Réalisabilité de chorégraphies pour systèmes asynchrones. Les langages de spécification de chorégraphies décrivent d'un point de vue global les interactions parmi un ensemble de services dans un système à concevoir. Etant donné une spécification de chorégraphie, l'objectif est d'obtenir une implémentation distribuée de la chorégraphie sous la forme d'un ensemble de composants appelés *pairs* qui communiquent. Ces pairs peuvent être donnés en entrée ou générés automatiquement par projection de la chorégraphie. Vérifier la *réalisabilité* consiste à prouver qu'un ensemble de pairs implémente la chorégraphie. Dans un cadre de communication asynchrone où les pairs interagissent par l'envoi de messages, cette vérification est en général indécidable.

Dans [149] nous nous sommes appuyés sur un résultat de décidabilité récent pour vérifier automatiquement la réalisabilité d'une chorégraphie par un ensemble de pairs dans un modèle de communication asynchrone avec des tampons non bornés *a priori*.

Application : modélisation et analyse de réseaux géniques. J'ai montré comment les principes de construction de Bip peuvent être utilisés pour modéliser et analyser des réseaux géniques complexes, en exploitant la structure de ces derniers. En particulier, j'ai proposé un algorithme d'analyse d'atteignabilité basé sur une simulation dirigée par l'objectif d'atteignabilité, pour des modèles biologiques à dimension élevée. J'ai appliqué cette approche à plusieurs réseaux de signalisation. Ces études de cas ont montré que le nouvel algorithme passe à l'échelle pour analyser des modèles de l'ordre de 600 gènes et de 10^{200} états, ce qui est hors de portée des techniques non compositionnelles [141].

En outre nous avons utilisé la vérification formelle pour l'estimation des paramètres d'un réseau de régulation génique, de manière à approcher soit un comportement observé expérimentalement, soit un comportement désiré [27].

Contrats pour l'ingénierie de besoins hétérogènes

Les contrats ont d'abord été introduits comme un système de typage pour classes [211] : une méthode garantit une postcondition pourvu que sa précondition soit satisfaite. Dans la communauté de programmation par composants, les contrats sont de plus en plus considérés comme un moyen d'atteindre l'un des principaux objectifs du paradigme de la programmation par composants, à savoir la (ré-)utilisation de composants dans des contextes différents, *a priori* inconnus.

Puisqu'un composant peut interagir sous des modèles de communication divers, la notion de contrat a été généralisée des pré- et postconditions sous forme de prédicats à des contrats *comportementaux* qui permettent de raisonner sur le comportement temporel des environnements avec lesquels le composant peut être composé. Dans la littérature, les contrats ont été utilisés à différents niveaux dans le processus de conception et avec des objectifs différents, comme l'ingénierie des besoins [36] et la vérification à l'exécution [21]. Pour le développement de programmes la thèse a été soutenue que la programmation par contrats devait remplacer la programmation défensive [211]. Cette approche gagne actuellement en popularité avec l'arrivée de multi-processeurs-sur-puce (MPSoC) hautement reconfigurables où la correction de chaque bloc fonctionnel — par exemple, la cohérence de la mémoire partagée — sous toutes les utilisations possibles est abandonnée pour une correction sous des préconditions, au profit de performances accrues.

Composition de contrats modaux et probabilistes. J'ai proposé un cadre à contrats pour Bip qui supporte différentes opérations dans le processus de conception à base de composants. En particulier, j'ai introduit une théorie de contrats hypothèse – garantie modaux et étudié les différentes opérations de composition requises pour supporter une construction ascendante (par assemblage de composants existants) et descendante (par composition de besoins) dans le flot de conception [148]. J'ai en outre développé une théorie de spécification pour des contrats probabilistes, permettant de raisonner sur la construction de systèmes avec des incertitudes sur les comportements [269]. Ce travail a été effectué dans le cadre du projet Européen Combest.

Analyse de causalité logique. Dans le cadre du projet Lise [243] sur les *problèmes de responsabilité en génie logiciel* qui réunissait des chercheurs en informatique et en droit, nous avons développé des techniques qui permettent à des partenaires contractuels (au sens juridique) de définir et d'établir les responsabilités en cas de dysfonctionnements logiciels. En particulier, j'ai défini plusieurs notions de *causalité logique* entre les violations de contrats informatiques. Ces définitions nous permettent d'analyser des logs d'exécution observés afin d'établir quels composants étaient responsables d'une violation de la spécification du système [147, 145, 146].

Synthèse de contrôleurs multi-échelles

Le contrôle des systèmes hybrides (systèmes exhibant à la fois des comportements discrets et continus) requiert généralement le développement de techniques spécifiques combinant des idées des théories des systèmes dynamiques discrets et continus. La complexité induite par l'interaction entre les dynamiques discrètes et continues rend bien souvent difficile l'obtention de résultats analytiques. Pour cette raison, la synthèse de contrôleurs pour les systèmes hybrides est souvent abordée par des méthodes d'abstraction où la dynamique continue est approchée par un modèle symbolique discret. En utilisant la notion de bisimulation approchée [131], on peut calculer des modèles symboliques de précision arbitraire pour une classe de systèmes hybrides. Le calcul des modèles symboliques implique une discrétisation du temps et de l'espace au moyen de grilles. Pour obtenir un modèle plus précis ou utiliser un paramètre de discrétisation en temps plus petit, il faut choisir un paramètre de discrétisation en espace plus petit [223, 132]. L'utilisation de grilles uniformes limite l'application de cette approche à des systèmes de taille modeste.

Dans le cadre du projet Vedecy [244], nous avons cherché à dépasser cette limitation en introduisant des grilles adaptatives multi-échelles [66, 65]. L'idée principale est de travailler avec un modèle symbolique de précision uniforme mais avec des paramètres de discrétisation en temps et en espace adaptatifs. Dans certaines régions de l'espace d'états, le contrôleur a besoin de prendre des décisions rapides, ceci nécessite des échantillonnages en temps et en espace plus fins. Dans d'autres régions de l'espace d'états, l'attention du contrôleur peut être réduite, ceci nous permet d'utiliser des paramètres d'échantillonnage plus grossiers. Pour que la méthode soit efficace, l'échelle locale de la grille doit être choisie pendant la synthèse du contrôleur discret. Ceci nécessite le développement de nouveaux algorithmes de synthèse de contrôleurs exploitant les spécificités des modèles symboliques multi-échelles. En utilisant les grilles multi-échelles sur l'étude de cas d'un convertisseur de tension, nos résultats montrent une réduction dramatique de la complexité du contrôleur [65, 217].

Plus récemment nous avons proposé une approche de synthèse multi-échelles où l'abstraction de l'état continu est la séquence bornée des derniers modes appliqués [194]. Nous avons montré qu'une précision arbitraire peut être obtenue en considérant des séquences suffisamment longues. Cette abstraction a un double avantage : d'abord, le système de transition permet une représentation compacte ; ensuite, l'approche n'utilise pas de grille dont la complexité croît exponentiellement avec le nombre de dimensions. Nous avons appliqué l'approche à un modèle de circulation routière pour lequel nous avons synthétisé un ordonnanceur pour la coordination des feux tricolores sous des contraintes de sûreté et d'équité.

Remerciement

Ce document résume mes travaux de recherche depuis la fin de ma thèse. Je tiens à remercier tous ceux qui ont, d'une manière ou d'une autre, contribué à, influencé, ou supporté ces travaux.

Contents

Chapter 1

Introduction

Embedded systems have become ubiquitous — from avionics, automotive, mobile communication devices to medical devices. In safety critical application domains, failures of embedded systems may produce material damage or compromise safety of human beings. Although many application domains of embedded systems are not safety critical, failures may still result in financial penalties or loss of customer satisfaction and market shares. At the same time, economic pressure towards shorter time-to-market, together with ever growing complexity of the systems to be designed, create a tremendous need for formal and semi-formal design techniques.

The appealing goal of component-based design [251] is to build complex systems from simpler components — similar to standard practice in other engineering disciplines — that are well understood and can be reused, so as to simplify the design process while improving the confidence in the artifact. In contrast to disciplines such as mechanical or electrical engineering, however, the theory of how to compose components is still immature.

Constructing a system from simpler components so as to preserve properties of the components and ensure new properties of the composition, is compromised by the multitude of ways in which components may interact (and interfere). To some extent this problem can be limited by restricting possible interactions (functional languages, actor-oriented formalisms), hiding internal behavior, and making abstraction from low-level behavior that can be automatically generated (synchronous languages). Nevertheless, existing design approaches do not provide satisfactory solutions for the fast growing applicative needs.

1.1 Challenges

Component-based design techniques have to take up the following challenges in order to meet the needs of system designers. For each point we provide references to the sections addressing it.

1.1.1 Heterogeneity

Heterogeneity of the models of computation and communication (MOC) has long been identified as a major challenge in component-based design, see e.g. [246]. As an important effort towards this direction, component-based design supporting heterogeneity has been addressed by the European project COMBEST [88].

Model of interaction. Interactions can involve two or more components. For each of the participating components they may be blocking or non-blocking. For instance, a handshake is a binary blocking interaction; writing to a buffer may be non-blocking if the oldest buffered element is overwritten when the buffer is full.

Expressing interaction schemes such as n-ary rendez-vous in terms of binary interactions is expensive (as a whole protocol must be modeled in terms of low-level synchronization primitives) and intrusive (as existing components must be re-engineered to achieve atomicity of the interaction). Therefore, a general-purpose component framework should provide high-level interaction primitives that separate interaction from the behaviors of the components.

In practice, complex models are constructed incrementally by progressively composing sub-systems. The interaction model should support this incremental construction. (§2.3)

Model of execution. In concurrent systems, two extreme cases of execution are the synchronous [160] and the asynchronous model of computation. In the former, all components progress jointly at each time step. In the latter, components must obtain resources (e.g., CPU time, memory access) and may progress individually as long as they have all needed resources to do so. Usually a scheduler is used to ensure fair access to shared resources in the asynchronous model. Looking more closely, it becomes clear that the execution model does not need to be global, but may be relative to each resource. For instance, progress (CPU access) may be synchronous while access to other resources is asynchronous, or vice versa [35]. Therefore, the execution model should support flexible modeling of scheduling policies, and composition of components using different models of execution. (§2.4)

Discrete vs. continuous behavior. A strong trend in embedded systems design is the tight interaction between discrete computations and the physical environment in a closed loop. Examples for such *cyber-physical systems* are multi-processor systems-on-chip where for each computing element voltage and frequency are controlled in a closed loop according to the current system charge, temperature, and available power [250]; and closed-loop control of (discrete) computing systems for load balancing and reconfiguration [56]. Not surprisingly, verification and design of cyber-physical systems are particularly challenging as they involve proving, on an infinite state space, the interdependent correctness of both the discrete and the continuous behavior. (§6)

1.1.2 Design Flow

The need for rigorous system design has been put forward for a long time (see [247] for a recent article).

Part of the difficulty software engineering is facing, compared to other engineering disciplines, in scaling up systems stems from the lack of physical constraints on software designs. For instance, many mechanical and electrical systems are robust to small disturbances due to energy dissipation, in contrast to most software systems whose discontinuous behavior may propagate and amplify unintended behaviors[1]. Similarly, the ways to compose physical components are strongly constrained, compared to software components.

Accordingly, there is a strong need for a rigorous design flow grounded on formally defined operations such as composition and refinement, supporting different needs during the design process. Composing readily implemented or off-the-shelf components on the one hand, and composing different requirements on the same component on the other hand, are two distinct operations with different properties. A framework supporting the entire design process is crucial for constructive formal methods to fully deploy their benefits.

[1]Robustness of sequential circuits has been studied in [106].

Compositionality. Formal verification and design techniques have made considerable progress since the introduction of model checking [84, 226]. However, in spite of a considerable research effort, the development of formal results supporting the construction of complex systems remains a hard problem. In particular, there is a lack of modular techniques supporting a component-based design flow based on incremental construction, and compositional verification exploiting the component structure.

A current trend in systems design is to design the components of a system such that they behave correctly when used according to some specification [250]. For instance, a memory management component ensures coherence of shared memory only when used according to some protocol, rather than enforcing correctness under all circumstances. The expected benefits of this approach are less overhead for correctness enforcement at run-time and thus, improved efficiency, at the price of verifying correctness depending on how a component is used. (§3.2, §3.4, §3.6, §4, §6.1.2)

Composability. In practice, the desired behavior of a component (or sub-system) is usually not defined by a single specification but rather, by the composition of several requirements, or *aspects*. A composition operation for aspects should ensure that the result refines its arguments, and signal possible inconsistencies. Often different aspects (say, safety and quality of service) do not have the same importance. In this case, a prioritized composition should refine the composed aspects such that in case of inconsistencies among them, an aspect of higher priority "overrides" a lower-priority specification. (§2.4, §3.7, §4)

Refinement. Formal support for the whole design flow from specification to implementation requires taking into account different levels of abstraction, and the refinement relation between them, for functional behavior as well as quantities such as different time scales and use of resources. Refinement of functionality and time scales is well understood for simple frameworks. For instance, the formalism of Reactive Modules [10] encompasses operations for both spatial and temporal abstraction in order to combine synchronous and asynchronous execution. However, in more complex frameworks such as embedded systems in charge of controlling a continuous physical process, linking different levels of abstraction is less obvious. Yet, being able to reason on different levels of abstraction is crucial for correct and efficient design and implementation. (§3.4, §4, §6.2)

Dealing with uncertainty and faults. Embedded and distributed systems often encompass unreliable software or hardware components, as it may be technically or economically impossible to make a system entirely reliable. As a result, system designers have to deal with uncertain behaviors, e.g., using probabilistic specifications such as "the probability that this component fails at this point of its behavior is less than or equal to 10^{-4}", or fault recovery techniques. (§3.6, §4.2, §5)

1.1.3 Understanding Logical Causality

Many applications are heterogeneous not only regarding their models of computation and communication, but also their origin. In the vast majority of complex systems, such as telecommunication networks, automotive control, and even application software, different components are provided by different vendors. When correctness by construction cannot be achieved — e.g., due to the workflow, the presence of unreliable components, or the sheer complexity of the system —, determining which component(s) effectively caused incorrect behavior, in order to establish liability of the component vendors for a resulting damage, is as inevitable as it is challenging [5]. (§5)

1.2 Contribution

This report summarizes contributions to the construction of embedded systems:

- the BIP component framework (§2), published in [152, 151, 153, 154];
- approaches for ensuring correctness by construction:
  - techniques for compositional analysis and validation of BIP models with respect to deadlock freedom (§3.2.2), progress (§3.2.3), liveness (§3.2.4), and reachability (§6.1.2) [143, 144, 139, 141];
  - coordination of components by means of adapter synthesis (§3.3) [257, 238];
  - compositional strategy mapping (§3.4) [140];
  - conformance checking and buffer dimensioning for components interacting asynchronously (§3.5) [149];
  - a theory of fault recovery in BIP (§3.6) [53, 54];
  - verification of weakly-hard requirements on quasi-synchronous systems (§3.7) [249].
- contract-based design: we present specification theories for both modal (§4.1) and probabilistic (§4.2) contracts [148, 269, 155].
- We introduce notions of logical causality in component-based systems, and study their analysis (§5) [147, 263, 145, 146, 142].
- We study symbolic abstractions of two classes of continuous systems:
  - modeling of genetic signalling networks in BIP and compositional analysis of reachability properties (§6.1.2) [139, 141]; and determination of parameters for a parametric model of a genetic network by model-checking (§6.1.3) [27];
  - we propose an approach for controller synthesis on multi-scale discrete abstractions of switched continuous systems (§6.2) [66, 65, 217, 194].

1.3 Notations

For a set of variables X, let $V(X)$ denote the set of valuations of X, let $\mathcal{P}(X) = 2^{V(X)}$ be the set of predicates on $V(X)$. Given a predicate $P \in \mathcal{P}(X)$, we interpret P in the straight-forward way on valuations of $X' \supseteq X$. Given a relation $R \subseteq X \times Y$, let $dom(R) = \{x \in X \mid \exists y \in Y : (x, y) \in R\}$ and $ran(R) = \{y \in Y \mid \exists x \in X : (x, y) \in R\}$ be the domain and the range of R, respectively. For $X' \subseteq X$, let $R(X') = \{y \in Y \mid \exists x \in X' \,.\, (x, y) \in R\}$. We write $i...j$ for the set $\{i, ..., j\}$.

symbol	meaning	definition
Σ	alphabet (of ports), switched system	2.1, 5.2, 6.9
B	symbolic transition system (behavior), behavioral model	2.1, 5.2
pre	precondition	2.6
C	set of connectors, component signature	2.10, 5.1
$I(C)$	set of interactions spanned by C	2.10
$\cdot\|\cdot$	interaction	notation 2.2
$Comp$	set of complete interactions	2.10
IM	interaction model $(C, Comp)$	2.10, notation 2.1
IM_{gl}	glue interaction model $(C, Comp)$	2.12, notation 2.1
K	component (B, IM, P)	2.11, 2.17
$\|_{IM}$	composition of interaction models, components, unfoldings, specifications, and contracts	2.13, 2.28, 3.27, 4.11, 4.31
$beh(K)$	semantic STS of component K	2.15, 2.18
$sem(K)$	semantic LTS of component K	2.15, 2.18
$\rightarrow_K$	semantic transition relation of component K	2.15, 2.18
P	execution model	2.16
$(B, IM, true)$	component with trivial execution model	notation 2.3
$/$	restriction of BIP components and unfoldings	2.19, 3.28
$\oplus$	composition of priorities	2.22, 2.27
$\cdot[K']$	projection of BIP interactions, interaction models, and components	2.29
enabled	predicate enabling transition	3.24
σ	unfolding, strategy, set of counterfactuals	3.26, 3.29, 5.5
π	projection of unfoldings, contracts, and traces	3.30, 4.30, notation 5.1
ϕ	focal point	6.3
$Q(M)$	qualitative model	6.6
$C(M)$	component model	6.7
$\wedge$	greatest lower bound of modal specifications and probabilistic contracts	4.7, 4.34
$\vee$	least upper bound of modal specifications	4.8
$\mathcal{C}$	contract, cone of influence	4.13, 4.22, 5.4
$\otimes_{IM}$	composition of modal contracts	4.17
$\oplus$	aspect conjunction of modal contracts	4.18
$\mathcal{S}$	specification	5.1
ρ	information flow relation	5.2
tr, $tr[i...]$	trace, prefix	notation 5.1

Table 1.1: Frequently used symbols.

Chapter 2

The Bip Component Framework

This chapter summarizes the publications [152, 151, 153, 154] introducing the initial version of the component framework baptized Bip in 2005. From then, Bip has been further developed at Verimag.

2.1 Motivation

Bip is a general, formal component framework built upon the idea of a simple yet expressive notion of "glue" to compose components. Two of our main goals in the initial design of the Bip component framework were to support *incremental* construction of *heterogeneous* systems. The two goals are achieved by making use of *separation of concerns* and introducing expressive models of interaction and execution. Let us discuss these design goals before formally introducing the framework.

Incrementality means that a system can be constructed and validated by gradually integrating additional components. In Bip, there is no restriction operation as in CCS [212]: the system remains open, such that any component can be seen (and verified) as a complete system, and be further integrated if desired. Therefore, two sets are required to describe the interactions of each component, so as to be able to distinguish *complete* interactions that are intended to persist when the (sub)system is further integrated, and *maximal* interactions that may not be complete, and are intended to interact with other components once the latter are integrated.

Heterogeneity means that components may interact through different *interaction models* (such as broadcast or n-ary rendez-vous) and *execution models* (such as asynchronous execution or run-to-completion). Heterogeneity is crucial to seamlessly construct systems by using or refining components on different levels of abstraction and on different platforms. For instance, on an abstract level, components may communicate through rendez-vous, which is then refined into a communication protocol using shared variables (see also the example in §3.4.5). The *execution model* is, in its general form, expressive enough to disable arbitrary interactions depending on the current system state; however, the introduction of *extended priorities* (Definition 2.24) has allowed us to significantly reduce the cost of checking whether a given restriction does not introduce any deadlock, while retaining sufficient expressiveness to model a large class of scheduling policies. Heterogeneous system models are also becoming an issue of growing importance in the design of *systems on chip*, where components of various origin, using different models of interaction, need to be integrated (see also §3.3 on converter synthesis).

Separation of concerns is a key requirement for a framework to support incremental construction of complex systems, by enforcing a separate and explicit description and composition of different aspects of the model. Any Bip component consists of three layers: *behavior*, *interaction model*, and *execution model* (initially formalized by priorities), hence the acronym Bip. Bip supports two operations: composition and restriction. The composition of Bip components is a component again, and so is the restriction of a component with an execution model. In the remainder of this chapter we outline only the basic building blocks of Bip. More complete overviews can be found in [138, 24].

2.2 Behavior

The behavior of a component is defined in terms of a *transition system*[1].

Definition 2.1 (Symbolic transition system) *A symbolic transition system (STS) is a tuple* $(X, \Sigma, G, F, init)$ *where*

- $X = \{x_1, ..., x_m\}$ *is a finite set of variables over arbitrary domains* $\mathbb{D}_{x_1}, ..., \mathbb{D}_{x_m}$*;*
- Σ *is a finite alphabet of actions, also called* ports*;*
- $G : \Sigma \to \mathcal{P}(X)$ *associates with every action its* guard *specifying when the action can occur;*
- F *is a function associating with each action* $a \in \Sigma$ *a partial function* $F(a) : X \to \big(V(X) \to \mathbb{D}_1 \cup ... \cup \mathbb{D}_m\big)$, $F(a)(x) \mapsto \big(V(X) \to \mathbb{D}_x\big)$ *defining an* assignment *to variables, depending on the current state. The assignment function is total (defined on all valuations* $V(X)$*);*
- $init \in \mathcal{P}(X)$ *is a predicate characterizing the set of initial states.*

Given a valuation $v = (v_1, ..., v_m) \in V(X)$ and $a \in \Sigma$ we write $F(a)(v)$ for the valuation $\big(F(a)(v_1), ..., F(a)(v_m)\big)$.

Definition 2.2 (LTS) *A* labeled transition system *(LTS) over an alphabet* Σ *is a tuple* $B = (Q, \Sigma, \to, Q_0)$ *with* Q *a set of states,* $\to \subseteq Q \times \Sigma \times Q$ *is transition relation, and* $Q_0 \subseteq Q$ *a set of initial states. As usual we write* $q \xrightarrow{a} q'$ *for* $(q, a, q') \in \to$*.* B *is* deterministic *if* $\forall q, a, q', q'' : q \xrightarrow{a} q' \wedge q \xrightarrow{a} q'') \implies q' = q''$*.*

When Q_0 is a singleton set $\{q_0\}$ we also write $(Q, \Sigma, \to, q_0)$.

Definition 2.3 (Semantics of an STS) *The semantics of an STS* $B = (X, \Sigma, G, F, init)$ *is an LTS* $sem(B) = (Q, \Sigma, \to, Q_0)$ *where* $Q = V(X)$, $\to = \{(q, a, q') \in Q \times \Sigma \times Q \mid \big(G(a)\big)(q) \wedge q' = F(a)(q)\}$*, and* $Q_0 = \{q \in Q \mid init(q)\}$*.*

In figures, we will draw the semantic LTS instead of STS.

Example 2.1 *Consider an STS modeling an application that cyclically requests a resource, gets access to it, and returns to an idle state:* $A = (X_A, \Sigma_A, G_A, F_A, init_A)$ *with*

- $X_A = \{waiting, using\}$*;*
- $\Sigma_A = \{req, p, v\}$*;*
- $G_A(req) = \neg waiting \wedge \neg using$, $G_A(p) = waiting$, $G_A(v) = using$*;*
- $F_A(req) = \{waiting := true\}$, $F_A(p) = \{waiting := false, using := true\}$, $F_A(v) = \{waiting := false, using := false\}$*;*

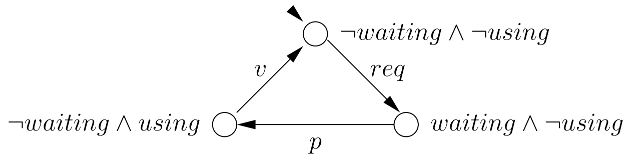


Figure 2.1: Behavior of the STS A.

- $init_A = \neg waiting \wedge \neg using$.

The reachable part of the LTS $sem(A)$ is shown in Figure 2.1.

Definition 2.4 (Sink state) *A sink state of an LTS $(Q, \Sigma, \rightarrow, Q_0)$ is a state $q \in Q$ such that for any $a \in \Sigma$ and $q' \in Q$, $(q, a, q') \notin \rightarrow$.*

Definition 2.5 (Deadend freedom) *An LTS is deadend free if it does not have any sink state.*

Definition 2.6 (*pre*) *Given an STS $B = (X, \Sigma, G, F, init)$ with $sem(B) = (Q, \Sigma, \rightarrow, Q_0)$ and a predicate $P \in \mathcal{P}(X)$, we define the predicate $pre_a(P)$ such that $\forall q \in Q : pre_a(P)(q) \iff \exists q' \in Q : q \overset{a}{\rightarrow} q'$ with $P(q')$.*

Definition 2.7 (Invariant) *Given an STS $B = (X, \Sigma, G, F, init)$ and a predicate $P \in \mathcal{P}(X)$, P is an* invariant *of B if $P \implies \bigwedge_{\alpha\in\Sigma} \neg pre_a(\neg P)$.*

2.3 Interaction Model

The main characteristic of Bip is the modeling of coordination and communication between components by an *interaction model* as introduced in [152, 154].

Definition 2.8 (Interaction) *An interaction over a set of ports Σ is a set $\alpha \subseteq \Sigma$.*

Definition 2.9 (Closure) *Given sets A, B of interactions with $A \subseteq B$, A is closed with respect to B if for any $\alpha \in A$, A contains all supersets of α in B.*

Definition 2.10 (Interaction model) *An interaction model over a vocabulary of ports Σ is a tuple $IM = (C, Comp)$ where*

- *$C \subseteq 2^\Sigma$ is a set of interactions called* connectors *such that (1) $\forall c, c' \in C : c \subseteq c' \implies c = c'$, and (2) $\bigcup_{c\in C} c = \Sigma$;*
- *$Comp$ is a set of* complete interactions *such that each interaction in $Comp$ is included in some connector of C, and $Comp$ is closed with respect to $I(C)$.*

where $I(C) = \{\alpha \mid \exists c \in C : \alpha \subseteq c\} \setminus \{\emptyset\}$ denotes the set of interactions spanned by C.

The set of connectors defines the set of maximal interactions between components. Notice that connectors need not be complete interactions. A connector in $C \setminus Comp$ is a maximal interaction in the component under consideration, but is supposed to be part of a larger complete interaction once the component is further integrated.

Notation 2.1 *Given $IM = (C \cup Comp)$, we sometimes write — by abuse of notation — IM for the union $C \cup Comp$.*

[1]More recently, other behavioral models such as Petri nets and timed automata have been used [2].

The *interaction model IM* specifies how components interact with each other. Each connector in C is a maximal set of component actions that can be executed jointly. The set of connectors C uniquely defines the set of interactions $I(C)$. $Comp \subseteq I(C)$ is the set of *complete interactions*. Complete or maximal interactions (that is, connectors) do not need to synchronize with other actions to take place, whereas non-maximal incomplete interactions cannot occur alone. Communication between component only takes place through interactions, so as to separate communication from computation.

Notation 2.2 *As usual we write* $a_1|...|a_n$ *for an interaction* $\alpha = \{a_1, ..., a_n\}$. *For instance, the set of connectors* $\{\{a, b\}, \{c\}\}$ *is simply written* $\{a|b, c\}$.

Example 2.2 *Consider a simple tokenring protocol consisting of two types of components, application layer* (A_i) *and network layer* (N_i)*, connected under the interaction model* $IM = (C, Comp)$ *with*

$$C = \bigcup_{i \in 1...3} \{req_i, p_i|grant_i, v_i|free_i, pass\_tok_i|get\_tok_{(i \mod 3)+1}\}$$

and

$$Comp = \bigcup_{i \in 1...3} \{p_i|grant_i, v_i|free_i, pass\_tok_i|get\_tok_{(i \mod 3)+1}\}$$

The interaction req_i *stands for a request to access the network,* $p_i|grant_i$ *for access being granted,* $v_i|free_i$ *to the end of network access, and* $pass\_tok_i|get\_tok_{(i \mod 3)+1}$ *for passing on the token to the next node in the ring. A system composed of three instances of the application layer and three instances of the network layer, interacting under IM, is shown in Figure 2.2 (composition is formally introduced in Definition 2.13). Incomplete actions (that do not form a connector by themselves) are indicated with* $\bullet$. *The singleton interactions* req_i *are maximal but not complete, as they are supposed to interact with some application software that is not part of the model.*

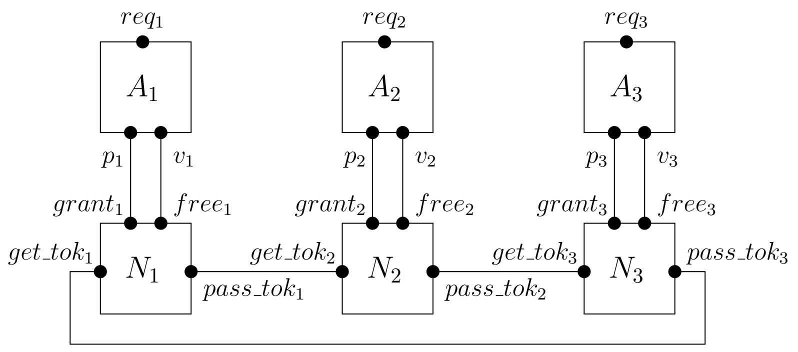


Figure 2.2: Architecture of the tokenring.

Example 2.3 *Consider the interaction model* $IM_1 = (C_1, Comp_1)$ *over the port alphabet* $\{a, b, c\}$ *with* $C_1 = \{a|b|c\}$ *and* $Comp_1 = \{a|b, a|b|c\}$*, graphically depicted in Figure 2.3, where the triangle stands for a complete sub-interaction, also called* trigger[2]. *IM models the fact that c can only take place together with a and b, that is, the rendez-vous between a and b may "trigger" c.* IM_1 *can be used, for instance, to model the fact that if in an optimistic resource sharing protocol a conflict is detected between two components waiting for each other's resource (a and b enabled), then a reset mechanism is triggered (c).*

[2]An algebra of BIP connectors is studied in [49].

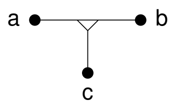


Figure 2.3: A simple interaction model consisting of a single connector $a|b|c$. Bullets and triangles stand for incomplete and complete interactions (triggers), respectively: a, b, and c are incomplete whereas $a|b$ and $a|b|c$ are complete.

The interaction model is fundamental for incremental construction: while specifying which interactions can occur at any stage of construction, it also provides information about still incomplete interactions that require synchronization with further components. For instance, a non-maximal and incomplete interaction $\alpha \in IC \setminus IM$, although it may not occur alone, may be visible in order to allow for an interaction $\alpha \cup \alpha'$ once the component is integrated in an environment proposing α'.

We first define atomic components and components consisting of a behavior and an interaction model, and add the execution model in §2.4.

Definition 2.11 (Atomic component) *An atomic component is a tuple* (B, IM) *of a behavior* $B = (X, \Sigma, G, F, init)$ *and an interaction model* $IM = (C, Comp)$ *where* $C = \big\{\{a\} \mid a \in \Sigma\big\}$*, and* $Comp \subseteq C$*.*

In the sequel let $\mathbb{I}$ be some index set over components.

Definition 2.12 (Glue interaction model) *A* glue interaction model *over a set of components* $K_i = (B_i, IM_i)$ *over port alphabets* Σ_i *with* $IM_i = (C_i, Comp_i)$*,* $i \in \mathbb{I}$*, is a tuple* $IM_{gl} = (C_{gl}, Comp_{gl})$ *where*

- C_{gl} *is a set of interactions called* glue connectors *over* $\Sigma = \bigcup_i \Sigma_i$ *such that* $\forall c, c' \in C_{gl} : c \subseteq c' \implies c = c'$*; and*
- $Comp_{gl}$ *denotes a set of complete interactions spanned by* C_{gl} *that is closed with respect to* $I(C_{gl})$*.*

such that such that for any interaction $\alpha \in C_{gl} \cup Comp_{gl}$ *there exist two components* $i \neq j$ *such that* $\alpha \cap \Sigma_i \neq \emptyset$ *and* $\alpha \cap \Sigma_j \neq \emptyset$*.*

The first item requires the glue connectors to be incomparable. The second item means that every interaction including a complete interaction is also complete.

Example 2.4 *The glue interaction model can be used to represent the extreme cases of asynchronous composition (with* $IM_{gl} = (\emptyset, \emptyset)$*) and synchronous composition (take* $IM_{gl} = (C_{gl}, \emptyset)$ *with* $C_{gl} = \{c_1 \cup ... \cup c_n \mid \forall i \in 1...n : c_i \in C_i\}$*). In the first case, the glue interaction model does not add any coordination, whereas in the second case, the connectors of the composed model are the interactions where each component participates with one of its connectors.*

Definition 2.13 (Composition of interaction models) *The composition of interaction models* $IM_i = (C_i, Comp_i)$ *over disjoint port alphabets* Σ_i*,* $i \in \mathbb{I}$*, with respect to a glue interaction model* $IM_{gl} = (C_{gl}, Comp_{gl})$ *over* $\Sigma = \bigcup_i \Sigma_i$ *is the interaction model* $\|_{IM_{gl}}\{IM_i \mid i \in \mathbb{I}\} = (C, Comp)$ *over* Σ *where*

- $C = \max(C_{gl} \cup \bigcup_i C_i)$*, the set of maximal connectors with respect to set inclusion, and*
- $Comp = Comp_{gl} \cup \bigcup_i Comp_i$*.*

Definition 2.14 (Composition of functions) *Given functions $f_1 : X_1 \to Y_1$ and $f_2 : X_2 \to Y_2$ with $X_1 \cap X_2 = \emptyset$, let $f_1 \cup f_2$ be their composition such that for any $x \in X_1 \cup X_2$, $(f_1 \cup f_2)(x) = f_1(x)$ if $x \in X_1$, and $(f_1 \cup f_2)(x) = f_2(x)$ otherwise.*

Definition 2.15 (Semantics) *The semantics of a component $K = (B, IM)$ with $B = (X, \Sigma, G, F, init)$ and $IM = (C, Comp)$ over Σ is $sem(K)$ of the STS $beh(K) = (X, C \cup Comp, G', F', init)$ where for each $\alpha \in C \cup Comp$,*

- $G'(\alpha) = \bigwedge_{a \in \alpha} G(a)$;
- $F'(\alpha) = \bigcup_{a \in \alpha} F(a)$.

Remark 2.1 *A connector $c = \{a_1, ...a_l\}$ specifies a degree of cooperation. For this connector to be executed in the global system, all l partners have to cooperate. As different connectors may have different size and involve different components, the degree of cooperation and the involved partners vary in the system. For instance, in one global state two components cooperate, whereas in another one, it may be three or more. One port may cooperate in different connectors with different partners and different degrees of cooperation. This is a powerful feature of the model that allows for great flexibility and distinguishes the* Bip *framework from others, for example process algebras or I/O-automata [203]. In process calculi such flexibility is either not realizable or can be achieved only in a clumsy way.*

The interaction model has been extended to encompass value passing and hiding [138]. These extensions are omitted here for the sake of conciseness.

2.4 Execution Model

The third layer of a Bip component is given by an execution model, as introduced in [152, 153].

Definition 2.16 (Execution model) *An execution model over a component (B, IM) with $B = (X, \Sigma, G, F, init)$ and $IM = (C, Comp)$ is a tuple $P = \big(V^X, (V^\alpha)_{\alpha \in C \cup Comp}\big)$ where $V^X \in \mathcal{P}(X)$ is a predicate characterizing the set of legal states, and for each $\alpha \in C \cup Comp$, $V^\alpha \in \mathcal{P}(X)$ is a constraint on the execution of α.*

We can now give the full definition of a Bip component.

Definition 2.17 (Component) *A component over an alphabet of ports Σ is a tuple $K = (B, IM, P)$ of a behavior B, an interaction model IM, and an execution model P over Σ.*

Definition 2.18 (Semantics) *The semantics $sem(K)$ of a component $K = (B, IM, P)$ with $B = (X, \Sigma, G, F, init)$, $IM = (C, Comp)$, and $P = \big(P^X, (P^\alpha)_{\alpha \in C \cup Comp}\big)$ is defined by $sem(B) = (Q, C \cup Comp, \to, Q_0)$ of the STS $B = beh(K) = (X, C \cup Comp, G'', F', init)$ where for any $\alpha \in C \cup Comp$,*

$$G''(\alpha) = G'(\alpha) \wedge P^\alpha \wedge pre_\alpha(P^X)$$

and $beh\big((B, IM)\big) = (X, C \cup Comp, G', F', init)$ is the semantic STS of K without the execution model.

The set of states *of K is Q.*

Notation 2.3 *We write $(B, IM, true)$ for a component with the trivial execution model $P = \big(true, (true)_{\alpha \in IM}\big)$.*

Definition 2.19 (Restriction) *The restriction of component $K = (B, IM, P)$ with $IM = (C, Comp)$ and $P = \big(V^X,\ (V^\alpha)_{\alpha \in C \cup Comp}\big)$, with an execution model $P_2 = \big(V_2^X,\ (V_2^\alpha)_{\alpha \in C \cup Comp}\big)$ is the component $K/P = (B, IM, P)$ where $P = \big(V^X \wedge V_2^X,\ (V^\alpha \wedge V_2^\alpha)_{\alpha \in C \cup Comp}\big)$.*

Definition 2.20 (Deadend freedom) *A component $K = (B, IM, P)$ is called* deadend-free *if $sem(K)$ is deadend-free.*

2.4.1 Priorities

Priorities are a particular case of an execution model.

Definition 2.21 (Priority order) *A priority order (priority, for short) over an interaction model $IM = (C, Comp)$ is a strict partial order over $C \cup Comp$.*

Definition 2.22 (Composition of priority orders) *The composition of two priority orders $\prec_1$ and $\prec_2$, written $\prec_1 \oplus \prec_2$, is the least priority order including $\prec_1 \cup \prec_2$.*

In other words, $\prec_1 \oplus \prec_2$ is the transitive closure of $\prec_1 \cup \prec_2$ if the closure is acyclic, and otherwise undefined.

Definition 2.23 (Induced execution model) *A priority order $\prec$ over a component (B, IM) with $B = (X, \Sigma, G, F, init)$ and $IM = (C, Comp)$ induces an execution model $P = \big(true, (V^\alpha)_{\alpha \in C \cup Comp}\big)$ where for any $\alpha \in C \cup Comp$ and any valuation $q \in V(X)$,*

$$V^\alpha = \neg \bigvee_{\substack{\beta \in C \cup Comp \\ \alpha \prec \beta}} G(\beta)$$

We write $(B, IM, \prec)$ for the particular case where the execution model of a component is defined by a priority order.

Example 2.5 *Coming back to Example 2.2, suppose that each component A_i behaves as shown in Figure 2.4 (left) and each component N_i as shown in Figure 2.4 (right).*

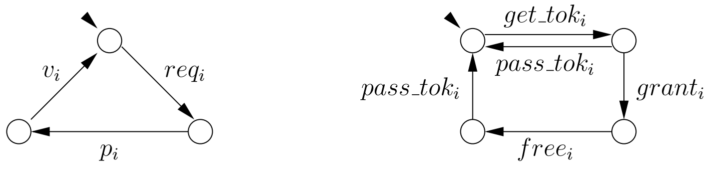


Figure 2.4: Behaviors of the tokenring components A_i (left) and N_i (right).

In $TR = \|_{IM}\{A_1, A_2, A_3, N_1, N_2, N_3\}$, requests req_i may never be served: when a request, say req_1, has been issued and the network component N_1 has acquired the token, both $p_1|grant_1$ and $pass\_tok_1|get\_tok_2$ are enabled. In order to give priority to serving pending requests over passing on the token, we restrict TR with the priority order

$$\begin{aligned}\prec\ = \{&pass\_tok_1|get\_tok_2 \prec p_1|grant_1,\ pass\_tok_2|get\_tok_3 \prec p_2|grant_2,\\ &pass\_tok_3|get\_tok_1 \prec p_3|grant_3\}.\end{aligned}$$

In $TR' = TR/\prec$, pending requests are served before the token is passed over. If no request has been issued, the token may be passed on immediately.

A useful property of priorities is that they preserve deadend freedom.

Proposition 2.1 (Preservation of deadend freedom) *If for a component $K = (B, IM, \prec)$ the semantics $sem(K)$ is deadend-free, and $\prec'$ is a priority order over K such that $\prec \oplus \prec'$ is defined, then $K/\prec' = (B, IM, \prec \oplus \prec')$ is deadend-free [153].*

Thus, priorities are an elegant way to ensure both composability, and preservation of deadend freedom.

As priority orders are not fine-grained enough for applications where the decision whether to disable some interaction depends on several interactions being enabled simultaneously — this is, for instance, the case for the generalized mutual exclusion properties as in Example 2.6 below —, we generalize them as follows.

Definition 2.24 (Extended priority) *An extended priority over an interaction model $IM = (C, Comp)$ is a relation $\prec \subseteq (C \cup Comp) \times 2^{C \cup Comp}$.*

We write $\alpha \prec \beta_1\beta_2 \cdots \beta_k$ for $(\alpha, \{\beta_1, \beta_2, ..., \beta_k\}) \in \prec$.

Definition 2.25 (Induced execution model) *An extended priority $\prec$ over a component (B, IM) with $B = (X, \Sigma, G, F, init)$ and $IM = (C, Comp)$ induces an execution model $P = \big(true, (V^\alpha_{\alpha \in C \cup Comp}\big)$ where for any $\alpha \in C \cup Comp$,*

$$V^\alpha = \neg \bigvee_{\substack{m \\ \alpha \prec m}} \bigwedge_{\beta \in m} G(\beta)$$

It would be nice to have a similar definition for the composition of extended priorities as for priority orders, that yields a simple criterion for the composition to preserve deadend freedom. To this end, we introduce a new definition of closure.

Proposition 2.2 (Safety) *Given a component $K = (B, (C, Comp), \prec)$, an interaction $\alpha \in C \cup Comp$, and a state q of K, α is disabled in q if there exists a set of interactions $B \subseteq C \cup Comp$ such that $\alpha \prec B$ and all interactions in B are enabled in q [153].*

Remark 2.2 *In [154] the semantics of a component has been defined in a more permissive way: all (even non-maximal and incomplete) interactions whose participating ports are enabled, may occur. This approach guarantees that the composition of deadend-free components is deadend-free. Safety with respect to the non-occurrence of non-maximal incomplete interactions, called* interaction safety *in [154], relies on the fact that transitions violating the property are disabled by complete or maximal interactions of higher priority.*

> ***Definition 2.26 (Interaction safety)*** *Consider a system S with interaction model $IM = (C, Comp)$. Define the priority order $\prec$ on incomplete interactions such that $\alpha_1 \prec \alpha_2$ if $\alpha_1 \in I(C) \setminus (C \cup Comp)$ and $\alpha_2 \in C \cup Comp$. S is called* interaction safe *if its restriction by $\prec$ can perform only complete or maximal interactions.*

In this setting, interaction safety can be checked in the same way as deadend freedom in the usual interaction semantics, e.g. by means of a dependency graph (see §3.2.2).

Definition 2.27 (Composition of extended priorities) *Given two extended priorities $\prec_1$ and $\prec_2$, their composition $\prec_1 \oplus \prec_2$ is the least extended priority $\prec$ such that $\prec_1 \cup \prec_2 \subseteq \prec$, and*

- *if $a \prec B$, $b \prec C$, and $b \in B$, then $a \prec B \cup C$;*
- *if $a \prec A$ and $a \in A$ then $a \prec A \setminus \{a\}$.*

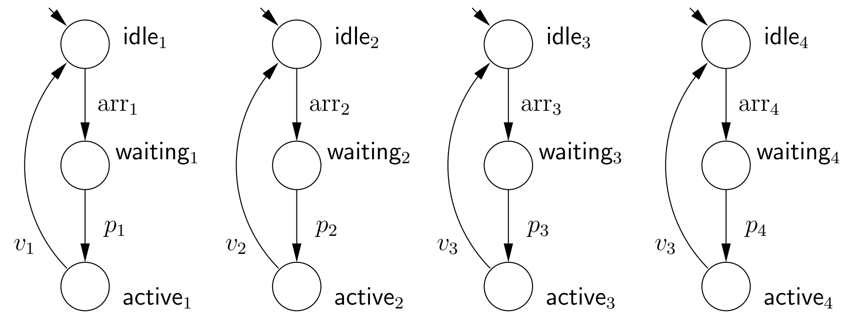


Figure 2.5: Behaviors B_i, $i \in 1...4$, of four processes contending for shared resources.

Proposition 2.3 (Preservation of deadend freedom) *If for a component $K = (B, IM, \prec)$ where $\prec$ is an extended priority, the semantics $sem(K)$ is deadend-free, and $\prec'$ is an extended priority over K such that $\prec \oplus \prec'$ is irreflexive, then $K/\prec' = (B, IM, \prec \oplus \prec')$ is deadend-free [153].*

Example 2.6 (Modular Construction of Safety Properties) *We illustrate the composability of extended priorities to guarantee safety and deadend freedom, by the example of four simple processes modeled by the systems shown in Figure 2.5. The processes share two identical resources. Each of the processes P_i has three states* idle_i, $\mathsf{waiting}_i$, *and* active_i. *Transition p_i enters state* active_i *where one of the resources is used. Transition v_i frees the resource and returns to state* idle_i.

Mutual exclusion "2-out-of-4" on the use of the resources by at most two processes is ensured by the priority

$$\prec_{\mathsf{mutex}} = \{p_i \prec_{\mathsf{mutex}} v_j v_k \mid i, j, k \in 1...4 \land i \neq j \land i \neq k \land j \neq k\}$$

Intuitively, the p_i action of an idle process is disabled whenever two other processes are using the resource.

In order to schedule the processes such that processes should have increasing priority from left to right when competing for resources, we define the fixed-priority scheduling policy $\prec_{\mathsf{pol}}$ such that $p_1 \prec_{\mathsf{pol}} p_2$, $p_2 \prec_{\mathsf{pol}} p_3$, and $p_3 \prec_{\mathsf{pol}} p_4$.

In order to enforce both requirements, we compose the extended priorities. The obtained extended priority $\prec_{\mathsf{mutex}} \oplus \prec_{\mathsf{pol}}$ is irreflexive. By Proposition 2.3, the constructed system $\|_{(\emptyset,\emptyset)}\{B_1, B_2, B_3, B_4\}/\prec_{\mathsf{mutex}} \oplus \prec_{\mathsf{pol}}$ is deadend-free (where $\|_{(\emptyset,\emptyset)}$ denotes asynchronous composition, see Definition 2.28 below).

Priorities have been shown to be a convenient way to model and compose even non-trivial scheduling policies [7].

2.5 Component Composition

Now we can extend Definition 2.13 to three-layered components.

Definition 2.28 (Composition) *Let $K_i = (B_i, IM_i, P_i)$ be components with $B_i = (X_i, \Sigma_i, G_i, F_i, init_i)$ over disjoint port alphabets Σ_i, $i \in \mathbb{I}$, and let $IM_{gl} = (C_{gl}, Comp_{gl})$ be a glue interaction model over $\Sigma = \bigcup_i \Sigma_i$. The composition of components K_i under IM_{gl} is the component $\|_{IM_{gl}}\{K_i \mid i \in \mathbb{I}\} = (B, IM, P)$ where*

- $B = (\bigcup_i X_i,\ \Sigma,\ \bigcup_i G_i,\ \bigcup_i F_i,\ \bigwedge_i init_i)$;
- $IM = (C, Comp) = \|_{IM_{gl}}\{IM_i \mid i \in \mathbb{I}\}$;

- $P = \left(\bigwedge_i V_i^X,\ (V^\alpha)_{\alpha \in C \cup Comp}\right)$ *where for any* $\alpha \in C \cup Comp$,

$$V^\alpha = \begin{cases} V_i^\alpha & \text{if } \alpha \in C_i \cup Comp_i \text{ for some } i \\ V_{gl}^\alpha & \text{if } \alpha \in C_{gl} \cup Comp_{gl} \\ true & \text{otherwise} \end{cases}$$

We often call *component system* or *system* a component obtained by composition.

Definition 2.29 (Projection) *Let K_i be components over disjoint port alphabets Σ_i, $i \in \mathbb{I}$, and let $IM_{gl} = (C_{gl}, Comp_{gl})$ be a glue interaction model over $\Sigma = \bigcup_i \Sigma_i$. Let $S = \|_{IM_{gl}}\{K_i \mid i \in \mathbb{I}\}$, $K' \subseteq \{K_i \mid i \in \mathbb{I}\}$, and $\Sigma[K'] = \bigcup_{K_i \in K'} \Sigma_i$.*

Given an interaction α over Σ, let $\alpha[K'] = \alpha \cap \Sigma[K']$ be its projection on K'.

The projection of S on components K' is $S[K'] = \|_{IM_{gl}[K']}\{K_i \mid K_i \in K'\}$ where $IM_{gl}[K'] = (C_{gl}[K'], Comp_{gl}[K'])$ with

- $C_{gl}[K'] = \max\left\{c[K'] \mid c \in C_{gl} \ \wedge\ \forall i \in \mathbb{I} : \neg(c[K'] \subseteq \Sigma_i)\right\}$, *and*
- $Comp_{gl}[K'] = \{\alpha \in Comp_{gl} \mid \alpha \subseteq \Sigma[K']\}$.

When projecting on a singleton set $K' = \{k\}$ we also write $\cdot[k]$ instead of $\cdot[\{k\}]$.

Intuitively, the set of projected glue connectors $C_{gl}[K']$ is the projection of all connectors in C_{gl} on $\Sigma[K']$ from which all connectors that are local to a single component — and that are by Definition 2.12 already part of its set of connectors — are removed. The set $Comp_{gl}[K']$ is the set of complete interactions that belong to K'.

Proposition 2.4 *Given a system $S = \|_{IM_{gl}}\{K_i \mid i \in \mathbb{I}\}$ of components and $K' \subseteq \{K_i \mid i \in \mathbb{I}\}$, we have*

$$S[K'] = \|_{IM_{gl}[K']}\{K_i \mid K_i \in K'\}$$

Proposition 2.5 *Composition is associative [154]: given a system $S = \|_{IM_{gl}}\{K_1, K_2, K_3\}$,*

$$S = \|_{IM_{gl}}\left\{\left(\|_{IM_{gl}[K_1,K_2]}\{K_1, K_2\}\right), K_3\right\} = \|_{IM_{gl}}\left\{K_1, \left(\|_{IM_{gl}[K_2,K_3]}\{K_2, K_3\}\right)\right\}$$

2.6 Prometheus

Prometheus [138] was the first implementation of the Bip framework with bounded domains. It uses a BDD-based symbolic representation of the components, and has been used as a platform to experiment many of the techniques described in the sequel. Independently, the Bip framework has been implemented in a tool platform also called Bip [22] and used for verification in various industrial case studies, see e.g. [25].

2.7 Discussion

We have presented the Bip component framework. Its three-layered components and two basic operations — composition and restriction — support the modeling of systems combining heterogeneous models of interaction and execution.

In contrast to component frameworks supporting a fixed model of interaction, the direct modeling of non-trivial interactions in a high-level formalism allows us to concisely model complex coordination patterns and protocols [25, 23]. This *structural expressiveness* of the Bip interaction model has been compared in [50] with other frameworks such as CCS [212] and CSP [168]. From the interaction model a correct implementations can then be derived automatically [22], hence reducing the risk of errors.

The interaction model presented here is non-causal, in the sense that there is no distinction — apart from the completeness of interactions — between ports initiating an interaction, and ports passively waiting for an interaction to happen. Such a notion of causality has been introduced in [51].

The execution model of BIP has proven to be a convenient way to model such non-trivial coordination schemes as for instance genetic regulatory networks; this application is discussed in more depth in §6.1.2.

Besides its application to industrial case studies [25], BIP has become a platform for implementing and experimenting new results. Recent extensions of BIP include the dynamic variant *Dy*-BIP [57], *S*BIP [33] for statistical model-checking, and *sec*BIP [240] for reasoning about information flow security.

2.7.1 Related Work

In the following we shortly discuss formal component frameworks with related interaction and coordination mechanisms.

The ALTARICA [15] framework is based on principles that are closely related to the layers in BIP. Interactions of components are modeled by a set of *broadcast synchronization vectors* generalizing the Arnold-Nivat synchronized product. Similarly to BIP, a priority order on actions can be defined. A bisimulation relation defines equivalence of components; replacing a sub-component with an equivalent sub-component yields an equivalent component.

The channel-based coordination model REO [14] is focusing on connectors and their composition. Basic connectors — including synchronous lossy and non-lossy connectors, one-place and unbounded FIFO connectors, fork, and merge — can be composed to form more complex coordination schemes.

An algebra of stateless connectors is developed in [60], building on a set of basic connectors, namely, symmetry, synchronization, mutual exclusion, hiding, and inaction. The framework is shown to be expressive enough to model the stateless connectors of the categorical framework COMMUNITY [116] and of REO.

Another component framework with expressive, reusable connectors is the architectural specification language WRIGHT [6] where connectors consist of *roles* describing the behaviors of participants, and *glue* coordinating and constraining the interactions. Roles and component protocols are specified in CSP [168].

The RAPIDE language [201] for prototyping distributed systems uses partially ordered event sets to model dependency in synchronization, concurrency, data flow, and timing aspects. Rapide distinguishes the component interface and and executable model of the component behavior.

FIACRE [46] is an intermediate format consisting of behavioral models in the form of state machines, and a layer modeling communication between the state machines, and specifying real-time constraints and priorities.

FRACTAL [59] is a component framework based on an extension of the π-calculus [214]. Each component consists of a *membrane* and a *content*. The membrane consists of a set of controllers in charge of introspection and filtering of messages. Components interact through reconfigurable communication channels.

The formalism of reactive modules [10] has been designed as a formalism to combine synchronous and asynchronous execution. Two notions of abstraction are defined: temporal abstraction, collapsing several rounds of execution into a single step, and spatial abstraction (hiding).

PTOLEMY [195] is an actor-based framework for modeling and composing different models of computation and communication called *domains*. Some domains have a formal semantics.

Java/A [30] is a component language for programming software architectures that integrates the concept of connectors in Java. Connectors are binary. The abstract component model has a formal semantics.

Besides, a number of component frameworks have been proposed that lack a formal semantics, but support analysis of certain aspects, among them SaveCCM [69] and Rubus [163] for embedded real-time system.

Chapter 3

Coordination of Component-based Systems

3.1 Motivation

Component-based construction — motivated by the goal of constructing complex systems from simpler components — requires the designer to be able to ensure correctness of the system from the correctness of its components and the way they are composed. This section tackles the question of how to verify and coordinate the behavior of components, in different settings, so as to ensure overall correctness of a system with respect to some specification.

In §3.2 techniques for compositional analysis of deadlock freedom, progress, and liveness are presented.

In §3.3 we study the problem of synthesizing adaptor components to make a set of given — possibly incompatible — components work together so as to avoid deadlocks and ensure boundedness for components exchanging items through FIFO buffers.

In §3.4 we are interested in compositionally refining a discrete controller C for a Bip system S into a controller C' for a target system S' such that S' controlled by C' behaves like S controlled by C. This notion of *strategy mapping* allows us to automatically reuse a controller as the system to be controlled is refined.

In §3.5 we study conformance of a set of components interacting over an asynchronous communication model with a priori unbounded buffers, with a system-level specification.

3.2 Compositional Analysis

Building systems that satisfy given specifications is a central problem in systems engineering. Standard engineering practice consists in decomposing a complex system to be designed into a set of components. A pivotal question is how to compose components in order to construct a correct system, or dually, how to verify whether the global behavior of the system satisfies given properties. In some cases, it is possible to solve the composition problem by synthesizing a controller or supervisor that restricts the behavior of the components [205] so that the overall system behaves correctly by construction or is amenable to formal analysis. However, both verification at the global system level and synthesis techniques have well-known limitations due to their inherent complexity or undecidability, and cannot be applied to arbitrarily complex systems. As an alternative to cope with complexity, *compositional* verification and design techniques have been studied for a long time [102]. The goal of compositional construction is to model and compose the components of a system in such a way that important properties such as progress and liveness are preserved when a new component is added to the system.

3.2.1 Compositional Analysis in Metropolis

Metropolis [20] is a design environment for embedded systems. It supports a methodology that favors the reusability of components by explicitly decoupling the specification of orthogonal aspects over a set of abstraction levels. Metropolis proposes a formalism called *meta-model* that is designed so that various computation and communication semantics can be specified using common building blocks. The meta-model supports progressive refinement of components, their communication and coordination. The Metropolis architecture encompasses a compiler front-end to translate a meta-model specification into an intermediate representation, and a set of back-end tools to support tasks such as synthesis, refinement, analysis, and verification of the model.

In [150] we have presented a framework and tool support for compositional modeling and analysis of Metropolis models. The basic question addressed in this paper is how to link compositional modeling methodologies and tools with a general framework like Metropolis. More precisely, we were interested in constructing, from a given Metropolis model, a component model consisting of a set of components, interactions, and priority functions[1], and verifying (1) consistency of the priority functions (Definition 2.22), (2) interaction safety (Definition 2.26), (3) deadlock freedom, and (4) structural liveness [55]. These verification techniques were implemented in the Prometheus I tool [137].

For the verification of models given in an expressive modeling language like the meta-model of Metropolis, it is in general necessary to represent high-level constructs in a more basic formalism on which verification can be carried out. To this end we have developed the following work flow, implemented within the Metropolis platform:

1. The system designer provides a restricted meta-model specification that is compatible with the Prometheus I formalism, and optionally a high-level description of a scheduling policy to be applied to the system.

2. The meta-model specifications are transformed in the Prometheus I formalism and the model is analyzed by Prometheus I.

For the interfacing with Metropolis, Prometheus I has been equipped with a parser front-end for the Metropolis meta-model. Prometheus I constructs the model incrementally by first analyzing the behavior of the components, then taking into account their synchronization on shared resources, and finally applying the specified scheduling policy. The resulting model is analyzed compositionally by verifying properties of the components and deriving properties of the system.

We have presented a set of results on a non-trivial example, the micro-kernel operating system TinyOS [92]. TinyOS is an extremely small (less than a kilobyte) foot-print operating system for embedded systems that provides basic functionality such as task scheduling, message passing, and interrupt handling, and supports a modular structure of the application. TinyOS has been designed to run on small, inexpensive embedded computers called nodes, which can be distributed over a building, measure parameters such as temperature, and communicate with each other over a short-range low-power wireless device.

A TinyOS application consists of a set of modules that interact through two types of communication: commands and signaling of events. Both are non-blocking; command

[1]This work was done before the development of Bip, but can be cast into the real-time extension of the latter.

invocation only initiates the command execution and returns control to the caller. A TinyOS application therefore has a high degree of logical concurrency.

We verified a sensing and routing application running on each node that is in charge of periodically requesting data from a sensor, transmitting this data, and routing incoming messages towards their destination. The application was modeled in METROPOLIS and automatically translated into 15 interacting components and 43 priority functions modeling mutual exclusion constraints between critical sections using shared resources, and the scheduling policy. The priority functions were checked for consistency and composed. By applying the compositionality results, PROMETHEUS I determines the model to be deadlock-free and safe, and the actors to be live.

The TinyOS case study has shown the power of compositionally verifying the liveness and soundness of a non-trivial example, but also current limitations. In particular, this work has motivated the development of techniques for compositional analysis of properties of components in a system: sufficient conditions for individual deadlock freedom and liveness of components will be presented in §3.2.3 and §3.2.4.

3.2.2 Deadlock Freedom

Throughout this section we consider a BIP model $S = \|_{IM}\{K_i \mid K_i \in K\}$ with semantics $sem(S) = (Q, IM, \rightarrow)$, K countable, and components $K_i = (B_i, IM_i, true)$ with $B_i = (X_i, IM[K_i], G_i, F_i, init_i)$ and semantics $sem(K_i) = (Q_i, IM[K_i], \rightarrow_i, Q_0)$.

Let P be a predicate on the global state space. We assume here that P is an *inductive invariant*, i.e. $\forall q, q' \in Q\ \forall \alpha \in C \cup Comp\ (P(q) \land q \xrightarrow{\alpha} q' \Rightarrow P(q'))$. As an example we consider the predicate $P_{reach(Q_0)}$ describing all global states that are *reachable* from the initial states.

A system is considered to be P-deadlock-free if in every global state satisfying P it may perform a maximal or complete interaction.

Definition 3.1 (P-deadlock-free) *Given an inductive invariant P, a component system S with $sem(S) = (Q, IM, \rightarrow)$ is called P-deadlock-free if for every state $q \in Q$ satisfying P there is a transition $q \xrightarrow{\alpha} q'$ with $\alpha \in C \cup Comp$. S is called deadlock-free if it is P-deadlock-free for $P = true$.*

Definition 3.2 (Complete state) *Consider components $K' \subseteq K$ with $sem(S[K']) = (Q', IM[K'], \rightarrow')$. A state $q \in Q'$ of K' is called* complete *in S if there is some interaction $\alpha \in C \cup Comp$ and some q' with $q \xrightarrow{\alpha}{}' q'$. Otherwise it is called* incomplete.

Definition 3.3 (Dependency graph) *The* dependency graph *for S is a labeled directed graph $\mathcal{G}_S = (V, E)$ where the set of nodes is $V = K$ and the set of labels is $L = L_1 \cup L_2$, with $L_1 = \{c \in C \mid \nexists \alpha \in Comp : \alpha \subseteq c\}$ and $L_2 = \{(c, \alpha) \mid c \in C, \alpha \in Comp, \alpha \subseteq c$, and $\nexists \beta \in Comp : \beta \subsetneq \alpha\}$ and the set of edges is $E \subseteq V \times L \times V$ such that*

1. *$(i, c, j) \in E$, where $c \in L_1$, iff $\exists q_i \in Q_i$, q_i incomplete, $\exists q_i' \in Q_i$ such that $q_i \overset{c[i]}{\rightarrow}_i q_i'$ and $c[j] \neq \emptyset$;*
2. *$(i, (c, \alpha), j) \in E$, where $(c, \alpha) \in L_2$, iff $\exists q_i \in Q_i$, q_i incomplete, $\exists q_i' \in Q_i$ such that $q_i \overset{c[i]}{\rightarrow}_i q_i'$ and $\alpha[j] \neq \emptyset$.*

We define the snapshot *of $\mathcal{G}_S$, resp. of a subgraph $\mathcal{G}$ of $\mathcal{G}_S$, with respect to a global state $q = (q_1, q_2, \ldots, q_i, \ldots) \in Q$ as $\mathcal{G}_S(q) = (V, E(q))$ where $E(q) \subseteq E$ such that*

1. *$(i, c, j) \in E(q)$, where $c \in L_1$, iff q_i is incomplete and $\exists q_i' \in Q_i$ such that $q_i \overset{c[i]}{\rightarrow}_i q_i'$;*

2. *$(i,(c,\alpha),j) \in E(q)$, where $(c,\alpha) \in L_2$, iff q_i is incomplete and $\exists q_i' \in Q_i$ such that $q_i \xrightarrow{c[i]}_i q_i'$.*

Intuitively, the nodes of the graphs are the components of S. An edge (i,c,j) means that there is an incomplete state q_i such that $c[i]$ is enabled in q_i, and j participates in the incomplete connector c. An edge $(i,(c,\alpha),j)$ means that there is an incomplete state q_i such that $c[i]$ is enabled in q_i, and j participates in the complete interaction α.

Remark 3.1 (Complexity) *Note that for the construction of the graph we inspect each component K_i separately and hence avoid the combinatorial complexity of global state analysis. The graph $\mathcal{G}_S$ can be constructed in $O\big(|C| \cdot |K| \cdot (|Comp| + \sum_i | \rightarrow_i |)\big)$.*

In the following we will use predicates on global states that are conjunctions of predicates on local states.

Definition 3.4 *Let $\mathcal{G} = (V, E)$ be a dependency graph according to Definition 3.3.*
For $e = (i,c,j) \in E$ let $cond(e) = G_i(c[i]) \wedge \neg G(c)$.
For $e = (i,(c,\alpha),j) \in E$ let $cond(e) = G_i(c[i]) \wedge \neg G(\alpha)$.
For $K_i \in K$ let $inc(K_i) = \{q_i \in Q_i \mid q_i$ is incomplete$\}$.
If $p = e_1, ..., e_r$ is a path in $\mathcal{G}_S$, then we put $cond(p) = \bigwedge_{i=1}^{r} cond(e_i)$.

In the next definition the notion of P-critical path is introduced. A critical cycle describes a situation where cyclic waiting of components could arise.

Definition 3.5 (P-critical path) *Given an inductive invariant P, a path p in $\mathcal{G}_S$ is called P-critical if $P \wedge (cond(p) \wedge \bigwedge_{K_i \in p} inc(K_i)) \not\equiv false$, where $K_i \in p$ means that node K_i is the start of some edge of p. A path p in $\mathcal{G}_S(q)$ is called* critical *if $(cond(p) \wedge \bigwedge_{K_i \in p} inc(K_i))(q) \neq false$. A path that is not critical is called non-critical.*

Certain paths can immediately be singled out as non-critical.

Lemma 3.1 (Non-critical path) *If $c \in L_1$, or $(c,\alpha) \in L_2$, occurs $|c|$ times as a label on $p = e_1, ..., e_r$ with $e_i \neq e_j$ for $i \neq j$ in $\mathcal{G}_S$, then $cond(p) \equiv false$.*

Definition 3.6 (P-refutable) *Let P be an inductive invariant and p be a P-critical cycle in a finite successor-closed subgraph $\mathcal{G}_f$ of $\mathcal{G}_S$, and $q = (q_1, q_2, ...)$ a global state with $P(q)$. p is said to be P-*refutable*, if, whenever p lies in $\mathcal{G}_f(q)$, where q_i is incomplete for every i, then there is a non-critical path $\hat{p}$ in $\mathcal{G}_f(q)$ such that for every edge $e = (i,c,j)$, resp. $e = (i,(c,\alpha),j)$, on that path $G_i(c[i])(q_i)$ holds.*

As the next theorem shows, a system is deadlock-free if there is a successor-closed subgraph of $\mathcal{G}_S$ that does not contain any critical cycle. It is not hard to see that this condition can be considered equivalent to the one given in [154]. In addition, the theorem states that, if there is no such subgraph, we have the option to check if there is a subgraph in which the critical cycles can be *refuted.*

Theorem 3.1 (P-deadlock freedom) *Given an inductive invariant P, if there is a finite non-empty successor-closed subgraph $\mathcal{G}_f$ of $\mathcal{G}_S$ such that every P-critical cycle in $\mathcal{G}_f$ is P-refutable, then S is P-deadlock-free [144].*

Example 3.1 *Figure 3.1 shows the dependency graph $\mathcal{G}_S$ of the tokenring from Example 2.5. The only successor-closed subgraph is $\mathcal{G}_S$ itself. There are 8 minimal critical cycles:*

$$A_i \xrightarrow{p_i|grant_i} N_i \xrightarrow{v_i|free_i} A_i, \quad i = 1,2,3$$

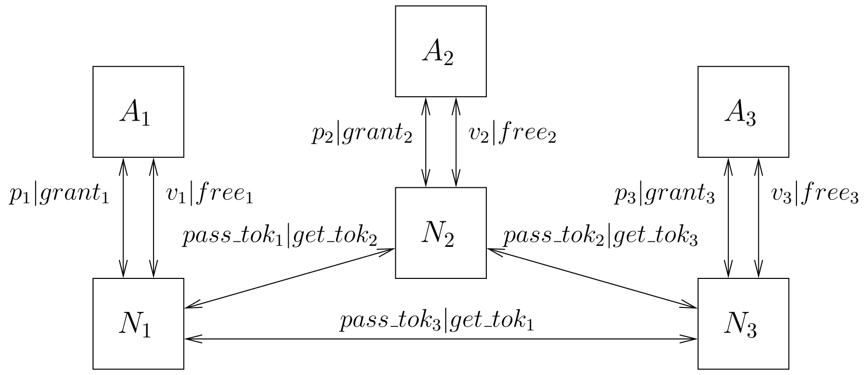


Figure 3.1: Dependency graph $\mathcal{G}_{TR}$ of the tokenring system. All labels c of the edges stand for (c, c). All connectors are complete interactions.

$A_i \overset{v_i|free_i}{\longrightarrow} N_i \overset{p_i|grant_i}{\longrightarrow} A_i, \quad i = 1, 2, 3$
$N_1 \overset{pass\_tok_1|get\_tok_2}{\longrightarrow} N_2 \overset{pass\_tok_2|get\_tok_3}{\longrightarrow} N_3 \overset{pass\_tok_3|get\_tok_1}{\longrightarrow} N_1$
$N_3 \overset{pass\_tok_2|get\_tok_3}{\longrightarrow} N_2 \overset{pass\_tok_1|get\_tok_2}{\longrightarrow} N_1 \overset{pass\_tok_3|get\_tok_1}{\longrightarrow} N_3$
None of them is refutable. However, we can show that the predicate

$$P = \Big(\bigvee_{i\in 1\dots 3} token_i \wedge \neg \bigvee_{j\neq i} token_j\Big) \wedge \Big(\bigwedge_{i\in 1\dots 3} (using_i \iff busy_i) \wedge (busy_i \implies token_i)\Big)$$

is an inductive invariant of TR. None of the cycles above is P-critical. By Theorem 3.1, TR is P-deadlock-free.

The existence of a successor-closed subgraph of $\mathcal{G}_S$, that has no critical cycles or only such that can be refuted, can be used in a straightforward way when composing systems. If the new connectors do not refer to the components of the subgraph then the composed system is deadlock-free.

Proposition 3.1 (Incremental deadlock freedom) *Let $K_i = (B_i, IM_i, true)$ be component systems with $IM_i = (C_i, Comp_i)$, $i = 1, 2$. Let $\mathcal{G}_f$ be a successor-closed subgraph of $\mathcal{G}_{K_1}$ in which every critical cycle can be refuted. Let w.l.o.g the nodes of $\mathcal{G}_f$ be $\{1, \dots l\}$. Let $I = \{x \in I(C_1) \mid x[j] = \emptyset$ for all $j \in \{1, \dots l\}\}$. Let $IM_{gl} = (C_{gl}, Comp_{gl})$ be a glue interaction model over K_1 and K_2. If $C_{gl}[K_1] \subseteq I$ then $\|_{IM_{gl}}\{K_1, K_2\}$ is deadlock-free [144].*

Robustness of deadlock freedom with respect to the failure of a set of ports has been studied in [144].

3.2.3 Progress

Throughout this section we consider a component system $S = \|_{IM}\{K_i \mid K_i \in K\}$ with semantics $sem(S) = (Q, IM, \rightarrow)$, K countable, and components $K_i = (B_i, IM_i, true)$ with $B_i = (X_i, IM[K_i], G_i, F_i, init_i)$ and semantics $sem(K_i) = (Q_i, IM[K_i], \rightarrow_i, Q_i^0)$.

Deadlock freedom is an important property of a system. But it does not provide any information about the progress an individual component $K_i \in K$ may achieve. Hence, it is interesting to consider the property of (individual) progress of component i, i.e. the property that at any point of any run of the system, there is an option to proceed in such a way that i will eventually participate in some interaction, which means that a clever scheduler can achieve progress of component i (see [154]).

Definition 3.7 (**P-run**) *Let S be P-deadlock-free. A P-run of S is an infinite sequence $\sigma = q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_1} q_2 \dots$ where $q_0 \in Q_0$ and $q_l \in Q$, $P(q_l) = true$, and $\alpha_l \in C \cup Comp$ for any l. For $n \in \mathbb{N}$, σ_n denotes the prefix $q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_1} q_2 \dots \xrightarrow{\alpha_{n-1}} q_n$.*

Definition 3.8 (**P-progress**) *Let S be P-deadlock-free. Let $K' \subseteq K$.*

- K' may P-progress in S *if for any P-run σ of S and for any $n \in \mathbb{N}$ there exists σ' such that $\sigma_n\sigma'$ is a P-run of S and for some $K_i \in K'$, K_i participates in some interaction α of σ'.*
- K' may strongly P-progress in S *if for any P-run σ of S and for any $n \in \mathbb{N}$ there exists σ' such that $\sigma_n\sigma'$ is a P-run of S such that every $K_i \in K'$ participates in some interaction α of σ'.*

In the first case, we just guarantee that we may always proceed in such a way that some component of K' participates in some interaction. In the second case, we may proceed in such a way that every component $K_i \in K'$ participates in some interaction.

If a set K' of components *may progress* in S then a clever scheduler can guarantee that a *run is chosen* where infinitely often some interaction with participation of the subsystem K' is performed.

The graph $\mathcal{G}_S$ that we use to determine deadlock-freedom of a system can also be used to determine if a subsystem of components $K' \subseteq K$ may progress. For this we construct the restriction $S[K']$ of the system S to K', and define when K' is controllable with respect to an interaction $\alpha \in I(C[K'])$ that is a potential partner for an interaction in $I(C) \setminus Comp$, which means that we may ensure that the subsystem defined by K' will be able to provide α when it is needed.

Definition 3.9 (*PRE*) *Let $K' \subseteq K$ and $S' = S[K']$ the projection on K' with semantics $sem(S[K']) = (Q', I(C[K']), \rightarrow')$. For $X \subseteq Q'$ define $pre(X) \subseteq Q'$ such that $q = (q_1, ..., q_{|K'|}) \in pre(X)$ if*

1. *$q \in Q'$ is complete in S' then $\exists \alpha \in (C[K'] \cap C[K]) \cup Comp[K']\ \exists q' : q \xrightarrow{\alpha}{}' q' \wedge q' \in X$*
2. *$q \in Q'$ is incomplete in S' then $\forall q' \in Q'\ \forall \alpha \in I(C[K']) \setminus Comp : q \xrightarrow{\alpha}{}' q' \implies q' \in X$.*

For $Q_0 \subseteq Q'$, we denote by $PRE(Q_0)$ the predicate characterizing the least solution of $X = Q_0 \cup pre(X)$. $PRE(Q_0)$ characterizes the set of states from which we can reach a state in Q_0 along a path in $sem(S[K'])$ by always performing complete interactions whenever a state is complete in S'.

Definition 3.10 (**P-controllable**) *Let S be a P-deadlock-free component system. Then, K is* P-controllable with respect to *$\alpha \in I(C)$ if $P \implies PRE(G(\alpha))$.*

$K' \subseteq K$ *is* P-controllable with respect to *IM if $\forall \alpha' \in I(C[K'])$:*

$$\big(\exists \alpha \in I(C) \setminus Comp : \alpha \cup \alpha' \in I(C)\big) \implies K' \text{ is } P\text{-controllable with respect to } \alpha' \text{ in } S[K']$$

That is, K' is controllable with respect to *IM* if it is controllable with respect to all of its interactions α' that participate in some interaction $\alpha \cup \alpha'$ such that $\alpha \neq \emptyset$ is incomplete.

We can now present a condition ensuring that a subsystem induced by K' may strongly progress, that is, for every run σ we may at any point continue with a run σ' such that every component of K' will participate at some time in the run σ'.

Theorem 3.2 (Strong P-progress) *Let S be a P-deadlock-free component system. $K' \subseteq K$ may strongly P-progress in S if the following two conditions hold [144]:*

1. *$\forall K_i \in K' \ \exists$ a finite successor-closed subgraph $\mathcal{G}_{f,i}$ of $\mathcal{G}_S$ that contains i and does not contain any P-critical cycle.*

2. *$\forall K_i \in K' \ \forall \alpha \in C[K''] : owners(\alpha)$ is P-controllable with respect to IM, where K'' is the set of components of $\mathcal{G}_{f,i}$, and $owners(\alpha) = \{K_j \in K \mid K_j(\alpha) \neq \emptyset\}$.*

The property of progress can be treated in an analogous way.

Example 3.2 *We want to check whether in the tokenring TR of Example 2.5, the application component A_1 may (strongly) P-progress, where P is the inductive invariant defined in Example 3.1. We reuse the graph $\mathcal{G}_{TR}$ of Figure 3.1. The first condition of Theorem 3.2 is satisfied, as we have already seen in Example 3.1.*

In order to verify the second condition of Theorem 3.2 we have to check whether for all connectors $c \in C$ of TR, $owners(c)$ is P-controllable with respect to IM:

- *req_i: $owners(req_i) = \{A_i\}$, $I(C[\{A_i\}]) = \{req_i, p_i, v_i\}$. We have to check P-controllability of $\{A_i\}$ with respect to p_i and v_i; this condition is satisfied.*

- *$p_i|grant_i$: $owners(p_i|grant_i) = \{A_i, N_i\}$, $I(C[\{A_i, N_i\}]) = \{req_i, p_i|grant_i, p_i, grant_i, v_i|free_i, v_i, free_i, get\_tok_i, pass\_tok_i\}$. We have to check P-controllability of $\{A_i, N_i\}$ with respect to get_tok_i and $pass\_tok_i$; this condition is also satisfied.*

- *$v_i|free_i$: $owners(v_i|free_i) = \{A_i, N_i\}$, as above.*

- *$pass\_tok_i|get\_tok_{(i \bmod 3)+1}$: $owners(pass\_tok_i|get\_tok_{(i \bmod 3)+1}) = \{N_i, N_{(i \bmod 3)+1}\}, I(C[\{N_i, N_{(i \bmod 3)+1}\}]) = \{grant_i, grant_{(i \bmod 3)+1}, free_i, free_{(i \bmod 3)+1}, pass\_tok_i|get\_tok_{(i \bmod 3)+1}, get\_tok_i, pass\_tok_{(i \bmod 3)+1}\}$. We have to check P-controllability of $\{N_i, N_{(i \bmod 3)+1}\}$ with respect to $grant_i$, $free_i$, $grant_{(i \bmod 3)+1}$, $free_{(i \bmod 3)+1}$, get_tok_i, and $pass\_tok_{(i \bmod 3)+1}$. Figure 3.2 shows the behavior of $TR[\{N_i, N_{(i \bmod 3)+1}\}]$. It can be checked that $\{N_i, N_{(i \bmod 3)+1}\}$ is P-controllable with respect to get_tok_i, $grant_i$, $free_i$, $grant_{(i \bmod 3)+1}$, and $pass\_tok_{(i \bmod 3)+1}$. However, $\{N_i, N_{(i \bmod 3)+1}\}$ is not P-controllable with respect to $free_{(i \bmod 3)+1}$. This is because once $N_{(i \bmod 3)+1}$ has received the token from N_i, the sub-system cannot control which of $grant_{(i \bmod 3)+1}$ or $pass\_tok_{(i \bmod 3)+1}$ to choose.*

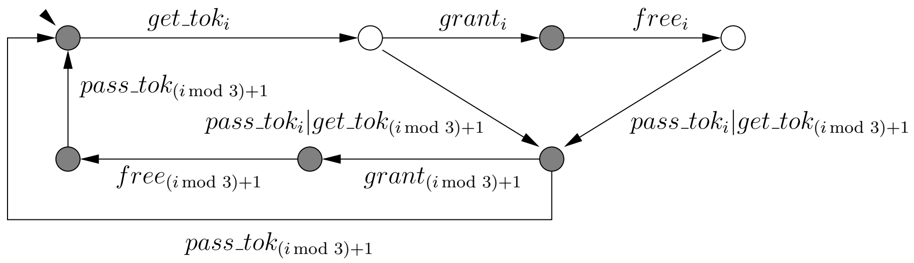

get_tok_i
$grant_i$
$free_i$
$pass\_tok_{(i \bmod 3)+1}$
$pass\_tok_i|get\_tok_{(i \bmod 3)+1}$
$pass\_tok_i|get\_tok_{(i \bmod 3)+1}$
$free_{(i \bmod 3)+1}$
$grant_{(i \bmod 3)+1}$
$pass\_tok_{(i \bmod 3)+1}$

Figure 3.2: Behavior of $TR[\{N_i, N_{(i \bmod 3)+1}\}]$. Grey states are incomplete.

Therefore, progress of A_1 cannot be established by Theorem 3.2. This shortcoming could be resolved by examining controllability of supersets of $owners(\cdot)$ to establish strong progress.

3.2.4 Liveness

In this section we recall the results from [143], and then generalize them to components with arbitrary execution models.

Liveness in systems with execution model *true*

Deciding liveness is NP-hard for component-based systems [204]. However, one may define (stronger) conditions that are easier to test and entail the desired properties. Here we present a condition that can be tested in polynomial time and entails liveness for a component $K_i \in K$. In what follows, we assume for simplicity that the local transition systems K_i have the property that they offer *at least one interaction* in every state. The general case can be reduced to this case by introducing idle actions or by adapting the definitions and results below to include this situation. To test for liveness we construct a graph, where the nodes are the components, and, intuitively, an edge $K_i \to K_j$ means "K_j needs K_i", in the sense that K_i eventually has to participate in an interaction involving K_j when K_j progresses.

Throughout this section we consider a component system $S = \|_{IM}\{K_i \mid K_i \in K\}$ with $IM = (C, Comp)$ and K finite. A subset K' of components is said to be P-live if K' participates infinitely often in every P-run.

Definition 3.11 **(P-liveness)** *Let $\emptyset \neq K' \subseteq K$. K' is P-live in S if S is P-deadlock-free and every P-run of S encompasses an infinite number of transitions where some $K_i \in K'$ participates, i.e. for every P-run σ and for all $n \in \mathbb{N}$ there is an m with $m \geq n$ and there is $K_i \in K'$ with $\alpha_m[K_i] \neq \emptyset$.*

Definition 3.12 **(P-inevitable)** *A set of interactions $A \subseteq I(C)$ is P-inevitable in S if every P-run of S encompasses an infinite number of transitions labeled with interactions in A.*

Proposition 3.2 *A set of components $K' \subseteq K$ is P-live in S if $I(C[K'])$ is P-inevitable in S, and S is P-deadlock-free.*

In order to check P-liveness of a component system, we define a sufficient condition that is easy to verify.

Definition 3.13 **($\mathcal{G}_{live}$)** *The graph $\mathcal{G}_{live}(S)$ is given by $(K, \to)$ where*

$$K_i \to K_j \qquad \textit{if} \qquad I(C[K_j]) \setminus excl(K_i)[K_j] \textit{ is } P\textit{-inevitable in } K_j$$

and $excl(K_i)[K_j] = \{\alpha[K_j] \mid \alpha \in C \cup Comp \wedge \alpha[K_i] = \emptyset\}$.

An edge $K_i \to K_j$ means "K_i is needed by K_j", in the sense that K_j cannot progress without K_i eventually making a step.

Theorem 3.3 **(P-liveness)** *Let S be P-deadlock-free, and let $K_k \in K$. We put $R_0(K_k) = \{K_j \mid K_k \to^* K_j\}$ and $R_{i+1}(K_k) = \{K_l \in K \setminus R_i(K_k) \mid \forall \alpha \in C \cup Comp : \alpha[K_l] \neq \emptyset \implies \exists K_j \in R_i(K_k) : \alpha[K_j] \neq \emptyset\} \cup R_i(K_k)$.*

If $\bigcup_{i \geq 0} R_i(K_k) = K$ then K_k is P-live in S [143].

Example 3.3 *We want to check P-liveness of N_1 in the tokenring $TR' = TR/\prec$ for the inductive invariant*

$$P = \Big(\bigvee_{i \in 1...3} token_i \wedge \neg \bigvee_{j \neq i} token_j\Big) \wedge \Big(\bigwedge_{i \in 1...3} (using_i \iff busy_i) \wedge (busy_i \implies token_i)\Big)$$

We first check P-liveness in TR. Figure 3.3 shows the graph $\mathcal{G}_{live}(TR)$. We compute $R_0(N_1) = \{A_1, A_2, A_3, N_1, N_2, N_3\} = K$. By Theorem 3.3, N_1 (and similarly, N_2 and N_3) is P-live in TR. P-deadlock-freedom of $TR' = TR/\prec$ (see Example 2.5) follows from P-deadlock-freedom of TR (see Example 3.1) and Proposition 2.3. It follows that N_i, $i \in 1...3$ is P-live in TR'.

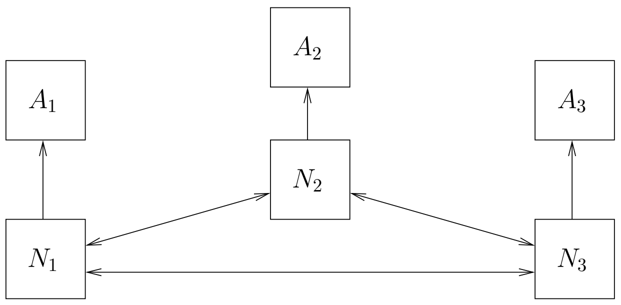


Figure 3.3: Inevitability dependency graph $\mathcal{G}_{live}(TR)$ of the tokenring system.

Proposition 3.3 (Complexity) *Testing the condition of Theorem 3.3 can be done in polynomial time in the sum of the sizes of the $sem(K_i)$ and the size of $C \cup Comp$ [143].*

The condition given in the above theorem can easily be adapted to establish the P-liveness of a set $K' \subseteq K$ of components.

Liveness in systems with arbitrary execution model

Theorem 3.3 uses a criterion on the interaction model as a sufficient condition for inevitability. This may, however, be overly pessimistic for components systems where the execution model helps to ensure inevitability, such as a fair scheduler. We therefore generalize the result to take into account the execution model. To this end, we introduce the notion of *maximal l-fixed invariant.*

Throughout this section we consider a component system $S = \big(\|_{IM}\{K_i \mid K_i \in K\}\big)/P$ with $IM = (C, Comp)$, semantics $sem(S) = (Q, IM, \to, Q_0)$ and a finite number of components $K_i = (B_i, IM_i, P_i)$ with $B_i = (X_i, IM[K_i], G_i, F_i, init_i)$ and semantics $sem(K_i) = (Q_i, IM[K_i], \to_i, Q_i^0)$.

Definition 3.14 (Existential abstraction) *Let X and X' be sets of variables such that $X' \subseteq X$, and let P be a predicate on X. We write $P|^{\exists}_{X'}$ for the* existential abstraction of P on X', *that is, the strongest predicate on X' implied by P.*

Definition 3.15 (Maximal l-fixed invariant) *Given two components $K_k, K_l \in K$, we define the* maximal l-fixed invariant of k*:*

$$inv(k,l) = \nu Y : Y = \big(Y \wedge pre_{k,l}(Y)\big)|^{\exists}_{X_k \cup X_l}$$

where $\nu Y.P$ denotes the greatest fixed point Y satisfying P, and

$$pre_{k,l}(Y) = \bigvee_{\substack{\alpha \in C \cup Comp \\ \alpha[K_k] \neq \emptyset \wedge \alpha[K_l] = \emptyset}} G'(\alpha) \wedge pre_\alpha(Y)$$

Intuitively, $pre_{k,l}(Y)$ characterizes the set of states from which Y can be reached in one step by an interaction involving K_k but not K_l. The predicate $inv(k,l)$ is an upper bound on the state space within which k is deadlock-free for some fixed state of l. That is, $inv(k,l) = false$ means that for any state of l, k will eventually block such that it cannot become enabled without having l progress.

Definition 3.16 (Inevitability dependency graph) *The inevitability dependency graph $idg(K,S)$ is the graph $idg(K,S) = (K, \to)$ with edges*

$$\big\{K_i \to K_j \mid inv(j,i) = false\big\}$$

The inevitability dependency graph generalizes $\mathcal{G}_{live}$ by taking the execution model into account; for systems with the trivial execution model *true*, both coincide.

Theorem 3.4 (Liveness) *Let S be a deadlock-free system. A non-empty subset $K' \subseteq K$ is live in S if all components in $K \setminus K'$ are reachable from K' in $idg(K, S)$.*

Proof 3.1 *Suppose that all components in $K \setminus K'$ are reachable from K' in $idg(K, S)$, and consider some component $K_i \in K \setminus K'$. By construction of $idg(K, B)$, component K_i can progress only a finite number of steps without any of K_i's predecessors making some action. By the hypothesis on reachability of all components from K', this means that none of the components in K can execute an infinite number of actions without some component in K' progressing as well. Since B is deadlock-free, this implies liveness of K'.* ■

Example 3.4 *The inevitability dependency graph of the tokenring example with the priorities defined in Example 2.5 is the same as the dependency graph of Figure 3.3. Liveness of each component N_i follows from Theorem 3.4.*

3.2.5 Related Work

Compositional model-checking. Compositional model checking for CTL^* by the use of interface processes has been proposed in [86]. The problem of compositional minimization of synchronized LTSs is tackled in [157] based on user-provided interface specifications modeling the behavioral constraints imposed on each LTS by synchronization with other LTS. A method for automated generation of such interfaces is described in [188]. Using a similar approach, compositional state space generation by abstracting component behaviors under different equivalences has been discussed in [260]. [3] formalizes a component calculus with different styles of composition based on a variant of the π-calculus [214]. The calculus supports reasoning about congruence but there are no results for verification of specific properties.

Deadlock freedom. Our condition on deadlock freedom has been further improved in [184]. [18] presents algorithms for verifying deadlock freedom of concurrent programs with shared variables in polynomial time, provided that the programs have a particular form. An efficiently verifiable sufficient condition for deadlock freedom in Bip similar to ours has been presented recently in [19].

In the component framework Wright [6], connectors consist of *roles* describing the behaviors of participants, and *glue* coordinating and constraining the interactions. The composition of compatible components under a deadlock-free connector is deadlock-free if the connector is conservative, in the sense that it does not introduce new behaviors.

In the abstract component model of Java/A [30], the composition of deadlock-free components is deadlock-free provided that the components are compatible. This verification is preformed by the compiler.

A method for compositional deadlock detection based on invariants has been implemented in the D-Finder tool [34] for Bip.

Liveness. The notion of *synchronic distance* as a measure for the independence of two components with respect to synchronization with each other has been introduced in [135]. Enforcing a maximal synchronic distance is proposed as a means to enforce fairness.

[9] provides compositional and assume/guarantee rules for the refinement of *live reactive systems*. Verification of liveness properties using compositional reachability analysis has been studied in [76].

3.3 Adapter Synthesis

Due to their increasing complexity, embedded systems and systems-on-chip (SoC) are nowadays often designed by (re)using already implemented components called COTS (Commercial Off-The-Shelf) components or IPs (intellectual property blocks). This approach helps to save development effort, reduce time-to-market, and achieve better quality thanks to already validated components.

However, while a set of components may meet the functional requirements, their protocols may not be consistent, leading to several kinds of mismatches such as *control*, *data*, and *timing* mismatches [52, 123, 170, 175, 251]. Control mismatches happen when the sequencing of control signals between protocols is inconsistent. Data mismatches happen when the data-widths of the two protocols differ and additional buffers are needed to manage loss-less data communication. Timing mismatches arise from inconsistent timing constraints of the protocols.

Adapter synthesis — also called protocol conversion in the literature — deals with the automatic synthesis of an additional component, often referred to as an *adapter* or a *converter*, to coordinate the interaction behavior of the components in order to avoid mismatches and guarantee that the obtained system satisfies a given specification. Adapter synthesis allows the developer to automatically build systems that are correct by construction from third-party components, hence improving reusability.

The requirements for a converter — and thus, the specifications for a converter synthesis algorithm — depend heavily on the considered models of computation and communication and on the properties to be preserved. For instance, consider a framework where components are synchronous sequential circuits and communications are electric wires, with the requirement that the protocol should refine a specified behavior. The role of an adapter would be to pass on, delay (latch), or withhold signals between components, or even to generate missing signals so as to ensure that the requirements are met. For this framework, converter synthesis based on a refinement relation between the protocols and a specification has been studied in [238].

On the other hand, consider an instance of Bip where components are connected to a common clock, and data items are communicated by binary rendez-vous, with the requirement to guarantee deadend freedom and causality of communications (each data item must have been produced before it is consumed, that is, data items cannot be "guessed" by the adapter). Here, the role of the adapter will be to desynchronize time progress of the components, store data items until they are consumed, and withhold data items whose consumption at the current state could lead to a violation of the requirements. An approach for converter synthesis in this setting will be presented in the next section.

3.3.1 Real-time Adapters

In [257] we have studied adapter synthesis for black-box components communicating through a data-flow interaction model. Each component is equipped with an interface that specifies the interaction behavior with the expected environment through input and output actions. In addition, the component interface specifies the component's *activation frequency* and timing constraints — namely, *latency* and *duration* of actions with respect to its activations — and controllability of the component actions.

The output ports of a component may be connected to input ports of other components through synchronous interactions. In order to deal with incompatible components (e.g., clock inconsistency, read/write latency/duration inconsistency, mismatching interaction protocols) we synthesize component adapters interposed between two or more interacting components. We have formalized the adapter synthesis algorithm by using Petri nets [221] theory. We have implemented the results in a tool called *SynthesisRT*, which we have used to validate the approach on a case study concerning a remote medical care system.

Although the work in [257] was originally presented using a timed process algebra, we formalize it here in the BIP framework for the sake of consistency. In contrast to [257] where the progress of each component was subject to a periodic activation clock, we do not consider activation clocks here in order to simplify the presentation; however, it is straight-forward to model the effect of activation clocks by the component behaviors.

Definition 3.17 (Timed protocol) *A* timed protocol *is a* BIP *model $S = \|_{IM_{gl}}\{K_i \mid i \in \mathbb{I}\} = (B, IM, true)$ with $\mathbb{I}$ a finite index set and $IM = (C, Comp)$ satisfying the following requirements.*

- *The port alphabet Σ of S is partitioned into* input *ports Σ^{in},* output *ports Σ^{out},* internal *ports Σ^{int}, and clock ports $\Sigma^{clk} = \{tick_i \mid i \in \mathbb{I}\}$. Furthermore, Σ is partitioned into* controllable *ports Σ^c and* uncontrollable *ports Σ^u such that $\Sigma^{out} \cup \Sigma^{clk} \subseteq \Sigma^c$ and $\Sigma^{int} \subseteq \Sigma^u$; input ports may be either controllable or uncontrollable.*
- *$K_i = (B_i, IM_i, true)$ are finite components with $B_i = (X_i, \Sigma_i, G_i, F_i, init_i)$ with initial states $init_i \in V(X_i)$, and $IM_i = (C_i, \emptyset)$ with $C_i = \big\{\{a\} \mid a \in \Sigma_i\big\}$. Moreover, in $sem(B_i)$ there is no cycle consisting only of uncontrollable actions.*
- *$IM_{gl} = (C_{gl}, \emptyset)$ with $C_{gl} \subseteq \{a|b \mid a \in \Sigma^{out} \wedge b \in \Sigma^{in} \cap \Sigma^c\} \cup \{tick_1|...|tick_n\}$ such that $\forall c, c' \in C_{gl} : c \cap c' \neq \emptyset \implies c = c'$.*

Uncontrollable actions model internal choices or open inputs, and can only interleave. Glue connectors either connect an output port with an input port, or all clock ports; each input or output port is part of at most one connector. All interactions are supposed to be incomplete; this will allow the converter to disable them, by not offering a required interaction, in order to ensure deadlock freedom and boundedness of buffers.

In order to enforce safety requirements, part of the components may play the roles of observers that move into a sink state whenever some undesired behavior is observed.

Example 3.5 *Consider the timed protocol of Figure 3.4 with $\Sigma^{in} = \{go, data?, ack?\}$, $\Sigma^u = \{go\}$, $\Sigma^{out} = \{data!, ack!\}$, $\Sigma^{int} = \emptyset$, $\Sigma^{clk} = \{tick_1, tick_2\}$, and interaction model $IM = (C, \emptyset)$ where $C = \{go, data!|data?, ack!|ack?, tick_1|tick_2\}$. The protocol suffers from two kinds of mismatches. First, the producer emits two data items ($data!$) and then waits for an acknowledgment ($ack?$), whereas the consumer sends an acknowledgment after each $data?$. This means that the second data item and the first acknowledgment need to be buffered in order to avoid a deadlock. Second, the producer lets time elapse after its two transmissions, whereas the consumer needs a $tick_2$ transition after each acknowledgment. Therefore, in order to avoid a deadlock (or rather, a timelock), time progress of both components has to be desynchronized.*

As an input, we consider a timed protocol $S = \|_{IM_{gl}}\{K_i \mid i \in \mathbb{I}\} = (B, IM, true)$ with $IM = (C, \emptyset)$. In general, S may be deadlocking. Our adapter synthesis approach consists of three main steps.

1. First, the BIP model is translated into a Petri net in order to desynchronize communications between output and input ports through a buffer place. This translation is compositional.
2. Next, we compute a *saturated marking graph* as a finite sub-behavior of the (in general infinite) marking graph, and perform discrete controller synthesis to ensure deadend freedom, to obtain a *controlled marking graph.*

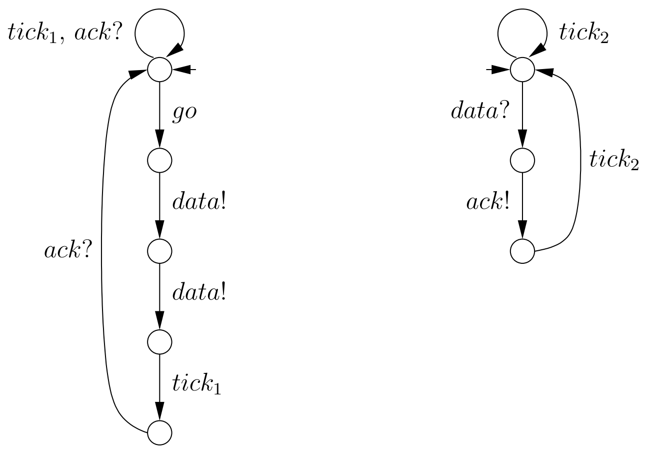


Figure 3.4: Timed protocol: producer (left) and consumer (right).

3. Finally, we extract an adapter component and compute a new interaction model to compose the adapter with the original components.

In the sequel we will describe each of the steps above more in detail.

Translation into a Petri net

The goal of the first step is to desynchronize communications by buffering outputs until they are consumed by an input action, while preserving the synchronization of clock ports. In [257] this was achieved by translating the behavior B_i of each component into a Petri net $N(B_i)$, and synchronizing the transitions standing for occurrences of $tick_i$ actions of the obtained Petri nets. However, this approach leads to an exponential number of common transitions modeling time progress. Here we present a variant where time progress of the components is D-synchronous [87], that is, any component may have a difference of up to D time units with respect to synchronous time progress, where $D \geq 1$ is a parameter.

Definition 3.18 (Petri net) *A Petri net is a bipartite graph (P, T, A, M_0) with nodes $P \cup T$ and directed edges A, where P is a set of places, T is a set of transitions partitioned into controllable transitions T^c and uncontrollable transitions T^u, $A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, and $M_0 : P \to \mathbb{N}$ is a set of initial markings.*

The first step consists in translating the behavior B_i of each component into a safe Petri net $N(B_i)$ where each transition represents the occurrence of an action in $sem(B_i)$, and each place represents a state of $sem(B_i)$. Thus, the semantics of $N(B_i)$ is identical to $sem(B_i)$. We name the transitions of $N(B_i)$ corresponding to the occurrences of an action a with $a^1, ..., a^{n_a}$.

Next, we have to take communications and synchronization on a common time base into account. In order to model communications, we add for each connector $a|b \in C$ with $a \in \Sigma^{out}$ and $b \in \Sigma^{in}$ a fresh place $p_{a|b}$ and arcs $a^i \to p_{a|b}$ from each transition a^i modeling an occurrence of a to $p_{a|b}$, and similarly arcs $p_{a|b} \to b^j$ to transitions standing for occurrences of b.

In order to synchronize time progress of the component Petri nets, we add, as shown in Figure 3.5, a new global $tick$ transition, and for each component K_i two places $pre - tick_i$ and $post - tick_i$ and arcs $pre - tick_i \to tick \to post - tick_i$, and $post - tick_i \to tick_i^j \to pre - tick_i$ for each component transition $tick_i^j$.

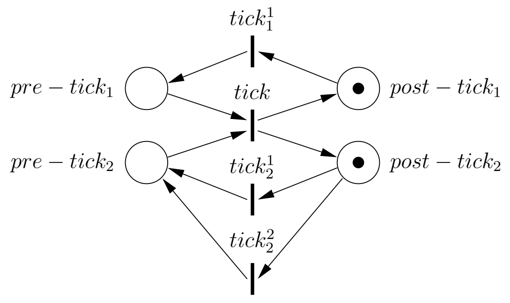


Figure 3.5: Decoupling time progress of two components K_1 and K_2 for $D = 1$.

A transition t of the obtained Petri net $N_D(S)$ is uncontrollable if and only if $t = a^i$ with $a \in \Sigma^u$ and $i \in 1...n_a$. Let T^u denote the set of uncontrollable transitions.

Finally, the initial marking m_0 of the obtained Petri net $N_D(S)$ is defined as follows: for each place p of $N(B_i)$ standing for the initial state of B_i, let $m_0(p) = 1$; for each component K_i let $m_0(post - tick_i) = D$; for all other places $p \in P$ let $m_0(p) = 0$.

In $N_D(S)$, local time progress of the components is loosely coupled, in the sense that any two components can be desynchronized by at most $D \geq 1$ time units.

Computing the Controlled Marking Graph

Definition 3.19 (Marking graph) *The* marking graph *of a Petri net $N = (P, T, A, M_0)$ is an LTS $(Q, T, \rightarrow, M_0)$ where $Q = 2^{P \rightarrow \mathbb{N}}$ is the set of* markings*, and*

$$\rightarrow = \Big\{(q, t, q') \mid q, q' \in Q \wedge t \in T \wedge \forall p \in P : \big((p, t) \in A \implies q(p) \geq 1\big) \wedge$$
$$q'(p) = \begin{cases} q(p) - 1 & \textit{if } (p,t) \in A \wedge (t,p) \notin A \\ q(p) + 1 & \textit{if } (p,t) \notin A \wedge (t,p) \in A \\ q(p) & \textit{otherwise} \end{cases} \Big\}$$

We use a *saturated marking graph* (called "extended coverability graph" in [257]) to obtain a finite abstraction of the behavior of $N_D(S)$.

Definition 3.20 (Saturated marking graph) *Let $N = (P, T, M_0)$ be a Petri net with marking graph $G = (M, \Sigma, \rightarrow, M_0)$. A sub-graph $H = (M_H, \Sigma, \rightarrow_H, M_0)$ of G (with $M_H \subseteq M_G$ and $\rightarrow_H \subseteq \rightarrow_G$) is a* saturated marking graph *of N if $M_0 \in M_H$ and for any $m \in M_H$ and $m' \in M$ with $m \xrightarrow{t} m'$,*

1. *if $t \in T^u$ then $m' \in M_H$ and $(m, a, m') \in \rightarrow_H$;*
2. *if $t \in T^c$ then either $m' \in M_H$ and $(m, a, m') \in \rightarrow_H$, or there exists $m'' \in M_H$ such that*
   (a) $M_0 \rightarrow_H^* m'' \rightarrow^* m'$, and
   (b) $\forall p \in P$: $m''(p) \leq m'(p) \wedge \big(m''(p) < m'(p) \implies m''(p) \geq \max\{1, M_0(p)\}\big)$.

The first condition ensures closure under uncontrollable transitions. The second condition means that H contains all markings m' of G unless there exists some "smaller" marking m'' in H from which the same transitions are enabled and from which m' is reachable. For a given Petri net, the minimal saturated marking graph (with respect to graph inclusion) can be constructed by exploring the marking graph and "cutting" transitions leading to a state m' for which some m'' satisfying conditions *2.(a)* and *(b)* exists. This graph is finite. The rationale behind condition *2.* is that markings are added as long as they enable new transitions. A similar criterion has been used in [91] to identify *irrelevant markings* and conjectured to be complete, meaning that if a bounded and non-blocking execution exists, it will be represented in the saturated marking graph. In contrast to the *coverability graph* [119] that over-approximates the behaviors of a Petri net, the saturated marking graph is an under-approximation.

Example 3.6 *The marking graph of the timed protocol of Example 3.5 is infinite, since more $ack!$ than $ack?$ actions may take place if the $ack?$-self-loop of the producer is never taken. In the saturated marking graph, only a finite prefix of the execution path is explored.*

Let H be a saturated marking graph of $N_D(S)$. We use discrete controller synthesis [232] to compute a maximal controller ensuring deadend freedom on H.

Definition 3.21 (Maximally permissive controller for deadend freedom) *Given an LTS $B = (Q, \Sigma, \rightarrow, Q_0)$ with Σ partitioned into controllable actions Σ^c and uncontrollable actions Σ^u, the* maximally permissive controller ensuring deadend freedom *is $C(B) = (Q', \Sigma, \rightarrow', Q'_0)$ where*

- *Q' is the greatest fixpoint of $X = Q \cap \widetilde{pre}(X)$ where $\widetilde{pre}(X) = \{q \in Q \mid \exists a \in \Sigma^c\ \exists q' \in X : q \xrightarrow{a} q' \wedge \forall b \in \Sigma^u\ \forall q'' : (q \xrightarrow{b} q'' \implies q'' \in X)\}$;*
- *$\rightarrow' = \rightarrow \cap (Q' \times \Sigma \times Q')$;*
- *$Q'_0 = Q_0 \cap Q'$.*

Thus, $C(H)$ is the maximally permissive controller ensuring deadend freedom on H. By construction, $C(H)$ can be used as a controller ensuring deadend freedom on the marking graph of $N_D(S)$. We call $C(H)$ the *controlled marking graph.*

Extracting the Adapter

In order to extract an adapter from the controlled marking graph and compose it with the timed protocol S, we have to define an interaction model for the adapter component, and construct an appropriate glue interaction model.

Definition 3.22 (Adapter) *Given the component system S and a controlled marking graph $\mathcal{G} = (Q, \Sigma_{\mathcal{G}}, \rightarrow, M_0)$ of $N_D(S)$, we construct a component $A(\mathcal{G}) = (B_A, IM_A, true)$ such that $sem(B_A) = \mathcal{G}'$, where $\mathcal{G}'$ is obtained from $\mathcal{G}$ by renaming, for each action $a \in \Sigma^{in} \cup \Sigma^{out}$ of S, each copy $a^i \in \Sigma$ introduced in the Petri net construction, with the fresh symbol a'. We put $IM_A = (C_A, \emptyset)$ with $C_A = \big\{\{a\} \mid a \in \Sigma_{\mathcal{G}}\big\}$.*

Each primed symbols a' of the adapter mirrors an input or output action a of the timed protocol S.

Definition 3.23 (Adapted timed protocol) *Given a timed protocol $S = \|_{IM}\{K_i \mid i \in \mathbb{I}\}$ with $IM = (C, \emptyset)$ and an adapter $A = (B_A, IM_A, true)$ for $N_D(S)$, let $IM_{gl} = (C_{gl}, Comp_{gl})$ with $C_{gl} = Comp_{gl} = \{a|a' \mid a \in \Sigma\}$. Then, $S' = \|_{IM_{gl}}\{K_i \mid i \in \mathbb{I}\} \cup \{A\}$ is the adapted timed protocol.*

That is, each input, internal, output, and clock port of the components in the timed protocol is composed with the corresponding ports of the adapter. Uncontrollable ports are supposed to be observable, but by construction of the adapter they cannot be disabled.

Example 3.7 *Figure 3.6 shows the architecture of the adapted timed protocol from Example 3.5.*

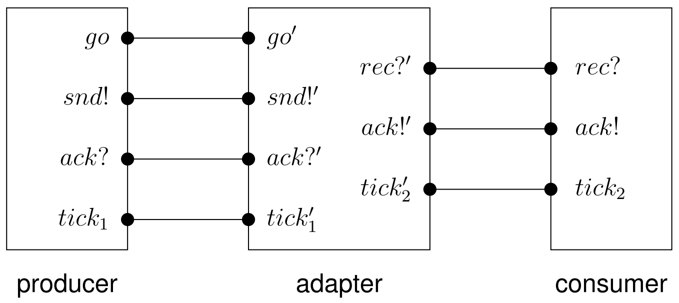


Figure 3.6: Architecture of the adapted timed producer/consumer protocol.

Properties. The adapted timed protocol S' has the following properties:

- If the adapter $A(\mathcal{G})$ has a non-empty behavior then S' is deadend-free.
- S' is a D-synchronous [87] composition of the components of S: synchronous interactions of S are desynchronized with upper bound D on the buffer sizes.

3.3.2 Discussion

The first approach to demonstrate the problem and some informal steps for a solution were proposed in [158]. By now there is a rich body of formal approaches to adapter synthesis in different settings, see e.g. [219] for a game-theoretical approach, [108] for a refinement-based solution, and [67] for an approach based on Petri nets. In contrast to the solution presented here, many of these approaches ignore real-time. Moreover, for most of them, adapter synthesis boils down to composing the component LTSs with a specification LTS and applying discrete controller synthesis to *disable* transitions so as to ensure deadend freedom and possibly fairness; solutions where the adapter decouples the components by buffering more than one instance of each data item, are not explored.

Closely linked to the problem of adapter synthesis is that of quasi-static scheduling — see e.g. [245, 91] — where the goal is to schedule a set of communicating processes so as to ensure boundedness of buffers, by choosing at run-time from a set of pre-computed schedules depending on uncontrollable (data-dependent) internal choices of the processes. The existence of a quasi-static scheduler has recently been proven to be undecidable in general [95]; this means that the minimal saturated marking graph may not represent the full behavior of $N_D(S)$; therefore the approach sketched above may not be complete.

Some interesting directions of future work are a generalization to partial observability of component ports by the adapter, and incremental adapter synthesis where an adapter for a set of components K can be obtained by first constructing an adapter for a set $K' \subseteq K$, using an abstraction of the behavior of its context $K \setminus K'$. Finally, it would be interesting to study dynamic adapter synthesis, possibly reusing ideas from quasi-static scheduling, in order to repair inconsistencies that may arise at run-time in dynamically reconfigurable frameworks such as Fractal [59].

3.4 Strategy Mapping

In this section we consider the port alphabet Σ_k of each component to be partitioned into a set of *controllable* ports Σ_k^c and a set of *uncontrollable* ports Σ_k^u.

3.4.1 Motivation

With the increasing complexity of embedded systems, coupled with the need for faster time-to-market and high confidence in the reliability of the product, design methods that ensure correctness by construction are, when available, the solution of choice. When dealing with reactive systems, which interact with their environment, the behavior of the system to be designed has to be considered in terms of *strategies*: can some desired behavior be enforced in spite of the — potentially non cooperative — environment?

Computing a strategy satisfying some property is expensive, and although decentralized discrete controller synthesis has been studied for some decades, compositionality remains a hard problem. In particular, progress properties are notoriously more difficult to tackle compositionally than safety properties. Therefore, most modular approaches focus on safety properties, e.g., [268, 125], or consider special cases. The interesting approach of [74], based on secure equilibria, is limited to two components. More closely related to our work, [96] defines a strong notion of modular refinement between "sociable" interface automata — allowing for interaction through shared actions and variables —, such that the refinement relation between a pair of components is independent of the environment. [58] defines refining contexts ensuring compositional refinement in a framework of components communicating through streams.

In [140], we are interested in a design flow supporting the refinement of strategies, rather than in discrete controller synthesis performed on some given level of abstraction. We consider a platform-based design process consisting of successive mapping steps [177]. The goal of each step is to map a strategy σ constructed so far, which guarantees a desired behavior on a system S, onto a system S'. The approach is constructive: if the mapping succeeds, a refinement relation is automatically constructed, showing how σ can be implemented using the primitives provided by S'. The mapping is performed component-wise, using an abstraction of the behavior of the environment of each component. We provide compositionality results ensuring that the refinement carries over to the global strategy.

More precisely, given a strategy σ over a system S of components C_i, and a target platform S' of components C'_i — where we assume that each component C_i is associated with exactly one component C'_i —, our goal is to implement σ on S' as follows. First, we compute a refinement relation $\preceq_i$ between each pair (C_i, C'_i). Each of these relations is then implemented by a local strategy σ'_i that can be seen as a control logic constraining the behavior of C'_i, obtained as an image of σ under $\preceq_i$.

Mapping is particularly interesting in a heterogeneous framework supporting different models of interaction and communication, and thus, different ways of implementing a desired behavior. We therefore formulate the results in a subset of BIP.

To our knowledge, this is the first work on compositional mapping of strategies. In contrast to many approaches limited to safety, strategy refinement and mapping support arbitrary strategies. We expect compositionally verifying refinement of a strategy that ensures some property to be less pessimistic than compositionally synthesizing a strategy ensuring the same property "from scratch", especially for properties other than safety.

3.4.2 Strategies

We consider action vocabularies A partitioned into *controllable* actions A^c and *uncontrollable* actions A^u. Uncontrollable actions are used to represent input events that cannot be triggered nor prevented by the modeled system.

For the sake of simplicity, we suppose that for any interaction α, $\alpha \cap A^u \neq \emptyset \implies |\alpha| = 1$, that is, uncontrollable actions can only interleave. According to the presence of uncontrollable actions, we partition IM into IM^c and IM^u.

Definition 3.24 (enabled) *Given an LTS $B = (Q, IM, \to, Q_0, init)$, $\alpha \in IM$, and $q \in Q$, let* $\mathsf{enabled}_B(\alpha)(q) \iff \exists q' \in Q \,.\, q \xrightarrow{\alpha} q'$.

Definition 3.25 (Interface interaction) *Given an interaction α and a behavior B_k with port alphabet A_k, let $IF_k(\alpha) = \alpha \setminus A_k$ be the* interface interaction *of α.*

We first introduce some notions used to reason about strategies.

Definition 3.26 (Unfolding) *An unfolding of a finite behavior B with $sem(B) = (Q_B, IM, \to_B, Q_B^0)$ is a tuple $\sigma = (Q, IM, \to, l, Q_0)$ such that $(Q, IM, \to, Q_0)$ is an LTS, $l : Q \to Q_B$ is a total labeling function such that $\forall q, q' \in Q\ \forall \alpha \in IM \,.\, q \xrightarrow{\alpha} q' \implies l(q) \xrightarrow{\alpha}_B l(q')$, and $Q_0 \subseteq Q$ is a set of* initial states *such that $\forall q \in Q_0 : l(q) \in Q_B^0$.*

By abuse of notation, we sometimes identify B with its trivial unfolding $(Q_B, IM, \to_B, id_{Q_B}, Q_B^0)$.

Example 3.8 *Consider the system $mutex = \|_{IM}\{P_1, P_2, S\}$ consisting of the three components shown in Fig. 3.7 with $IM = \{a_1, p_1|p, v_1|v, a_2, p_2|p, v_2|v\}$. Two processes contend for the use of a shared resource, with a semaphore ensuring mutual exclusion. All actions except for a_1 and a_2 are controllable.*

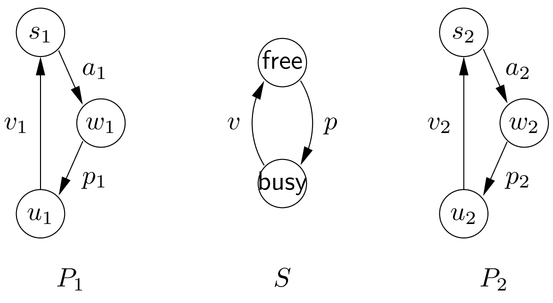


Figure 3.7: Two components P_1 and P_2 coordinated by a semaphore S.

Fig. 3.8 shows an unfolding σ of mutex modeling a first-come-first-served policy for fair resource use. σ has two states labeled with the same product state w_1w_2 where both components are waiting for access to the critical section. Depending on the order in which the access has been requested ($a_1; a_2$ or $a_2; a_1$), the component having made the request first is granted access.

We extend the definitions of composition and restriction to unfoldings.

Definition 3.27 (Composition) *Given an interaction model IM and unfoldings $\sigma_i = (Q_i, IM[i], \to_i, l_i, Q_i^0)$, $i \in 1...n$, let $\|_{IM}\{\sigma_i \mid i \in 1...n\} = (Q, IM, \to, l, Q_0)$ where*

- $Q = Q_1 \times ... \times Q_n$, $Q_0 = Q_1^0 \times ... \times Q_n^0$;
- $q \xrightarrow{\alpha} q'$ *if* $\alpha \in IM$ *and for any* $i \in 1...n$, *either* $q_i \xrightarrow{\alpha[i]}_i q_i'$, *or* $\alpha[i] = \emptyset$ *and* $q_i' = q_i$.
- $l(q_1, ..., q_n) = (l_1(q_1), ..., l_n(q_n))$.

Definition 3.28 (Restriction) *The* restriction *of an unfolding $\sigma = (Q, IM, \to, l, Q_0)$ with an execution model $P = \big(true, (V^a)_{a \in IM^c}\big)$ is the unfolding $\sigma/P = (Q, IM, \to', l, Q_0)$ where $\to' = \{(q, \alpha, q') \in \to \mid \alpha \in IM^u \vee V^\alpha(q)\}$.*

Definition 3.29 (Strategy) *A strategy σ over a behavior B with $sem(B) = (Q_B, IM, \to_B, Q_B^0)$ is an unfolding $\sigma = (Q, IM, \to, l, Q_0)$ of B that is closed under uncontrollable transitions, that is, if $q \in Q$ and $\exists \alpha \in IM^u\ \exists y \,.\, l(q) \xrightarrow{\alpha}_B y$, then $\exists q' \in Q$ s.t. $q \xrightarrow{\alpha} q'$ and $l(q') = y$.*

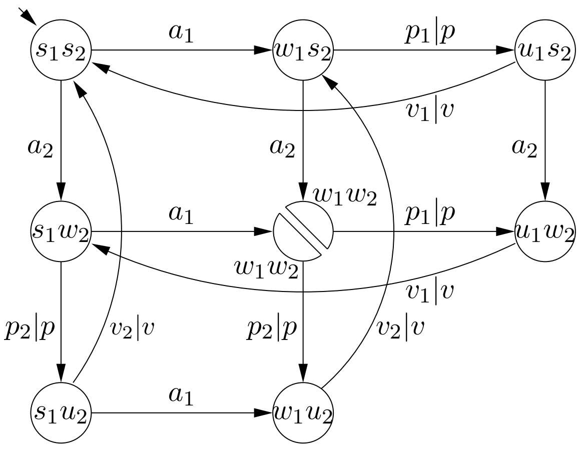


Figure 3.8: Unfolding σ (the names of the semaphore states are omitted).

Example 3.9 *The unfolding σ of mutex from Fig. 3.8 is a strategy.*

Definition 3.30 (Projection) *Given an unfolding $\sigma = (Q, IM, \rightarrow, l, Q_0)$ over $\|_{IM}\{C_i \mid i \in 1...n\}_i/P$ and a component C_k, let $\sigma' = (Q, IM[k], \rightarrow', l, Q_0)$ where $\rightarrow' = \{(q, \alpha[k], q') \mid (q, \alpha, q') \in \rightarrow\}$. The* projection *$\pi_k(\sigma)$ of σ on C_k is the unfolding $(Q', IM[k], \rightarrow'', l', Q'_0)$ where $(Q', IM[k], \rightarrow'', Q'_0)$ is the determinization (τ-elimination, see e.g. [4]) of $(Q, IM[k], \rightarrow', Q_0)$. For a state $q = \{q^1, ..., q^m\} \in Q'$ with $q^i = (q^i_1, ..., q^i_n)$ (where $\forall i, j \in 1...m : q^i_k = q^j_k$) let $l'(q) := l(q^1_k)$.*

Example 3.10 *The projection $\sigma_i = \pi_i(\sigma)$ of σ on each of the components P_1, P_2, and S, of Example 3.8 is equal to the trivial unfolding of the components.*

3.4.3 Strategy Refinement

Let $\mathbb{I}$ be a finite set of component indices. Given a system $B = \|_{IM}\{B_i \mid i \in \mathbb{I}\}/P = (Q, IM, \rightarrow, Q_0)$, an unfolding $\sigma = (Q_\sigma, IM, \rightarrow_\sigma, l, Q^0_\sigma)$ over B, $\alpha \in IM$, and $q \in Q_\sigma$, let $\mathsf{enabled}_\sigma(\alpha)(q) \iff \exists q' \in Q_\sigma \,.\, q \xrightarrow{\alpha} q'$. For $k \in \mathbb{I}$, let $\mathsf{disabled}_{k,\sigma}(\alpha)$ be a function associating with each interaction α a predicate over $Q \times Q_\sigma$ characterizing the states in which α is locally enabled in all involved components except for k, but disabled in σ:

$$\mathsf{disabled}_{k,\sigma}(\alpha) = \neg\mathsf{enabled}_\sigma(\alpha) \wedge \bigwedge_{\substack{i \neq k \\ \alpha[i] \neq \emptyset}} \mathsf{enabled}_{B_i}(\alpha[i])$$

We first define (non-compositional) strategy refinement.

Definition 3.31 *Given behaviors B_1, B_2 with $sem(B_i) = (Q_1, IM, \rightarrow_1, Q^0_1)$ and $sem(B_2) = (Q_2, IM', \rightarrow_2, Q^0_2)$, unfoldings $\sigma = (Q, IM, \rightarrow, l, Q_0)$ and $\sigma' = (Q', IM', \rightarrow', l', Q'_0)$ over B_1 and B_2, respectively, and a relation $\sqsubseteq\, \subseteq Q_2 \times Q_1$, we define $\leq_{B_1,B_2} (\sigma, \sigma') \subseteq Q' \times Q$ as the greatest fixpoint of*

$$\begin{aligned}
\prec\; = \;\sqsubseteq \cap \big\{(y, x) \mid \forall \alpha \in IM^c \;\; s.t. \;\; x \xrightarrow{\alpha} x' \\
&\exists m \geq 0 \; \exists b_0, ..., b_m \in (IM')^c \; \exists y_1, ..., y_m, y' \in Q' \,. \\
&y_0 = y \xrightarrow{b_0}{}' \, ... \xrightarrow{b_{m-1}}{}' \, y_m \xrightarrow{b_m}{}' \, y' \wedge \forall i \in 1...m \,.\, y_i \prec x \wedge y' \prec x' &\quad (3.1)\\
&\wedge \forall \beta \in (IM')^u \,.\, y \xrightarrow{\beta}_2 y'' \implies \\
&\quad \big(y'' \prec x \;\vee\; \exists \alpha \in IM^u \,.\, x \xrightarrow{\alpha} x' \wedge y'' \prec x'\big)\big\} &\quad (3.2)
\end{aligned}$$

σ' refines σ if $Q_0 \subseteq\, \leq_{B_1,B_2} (\sigma, \sigma')(Q'_0)$.

Intuitively, line (3.1) defines a simulation relation: from any pair of states x, y of the local strategies with $y \prec x$, whenever σ can make a transition $x \xrightarrow{\alpha} x'$, σ' can make a sequence of transitions such that both target states again satisfy the relation, and all intermediate states visited by σ' are related with x. Line (3.2) ensures refinement to be contravariant: simulation is preserved by all uncontrollable transitions of σ'. Unlike the usual definition of refinement with respect to safety properties, an unfolding σ' refines σ if it simulates σ (the underlying behaviors may be incomparable). This ensures that the behavior offered by σ — which may be used when the component B_1 is deployed — is also provided by σ'. On the other hand, σ' may have additional controllable behavior: refinement ensures that σ' is *sufficient* to implement σ on B_2. The refinement relation is a form of *alternating simulation* [97]. A weaker definition of refinement such as (alternating) trace inclusion would not be sufficient to ensure compositionality.

In [140] we have defined the compositional refinement of a strategy σ_G over S onto a platform S', by adding sufficient conditions to Definition 3.31 so as make refinement compositional. Intuitively, the moves of σ' are required to be enabled in the global system S' whenever the corresponding move of σ is enabled in S, and the effect of σ' on S' must not be not more restrictive than the effect of σ on S. More precisely, consider

- an unfolding σ_G with state space Q_G over $S = \big(\|_{IM}\{C_i \mid i \in \mathbb{I}\}\big)/P$ with state space Q_S and component state spaces Q_i;
- a target platform $S' = \big(\|_{IM'}\{C'_i \mid i \in \mathbb{I}\}\big)/P'$ with state space $Q_{S'}$ and component state spaces Q'_i;
- an unfolding σ'_k over C'_k;
- a refinement invariant inv over $Q_S \times Q_G \times Q_{S'}$.

We define the predicate $\sqsubseteq$ ("less restrictive than") over $Q_G \times Q_S \times Q_{S'}$ such that for any $x \in Q_k$, $y \in Q'_k$,

$$y \sqsubseteq x \iff inv \wedge \bigwedge_{\gamma \in IM^c \setminus IM[k]} \big(\mathsf{enabled}_{\sigma_G}(\gamma)[l(x)/Q_k] \implies \neg\mathsf{disabled}_{k,S'}(\gamma)[l'(y)/Q'_k]\big)$$

This predicate characterizes the states for which the behavior of other components than k is not more restricted in S' than in σ_G, both due to interactions that are not offered by k, or due to the execution model.

Let σ_k be the projection of σ_G on C_k. The refinement of σ_k by σ'_k under inv, written $\sigma'_k \preceq^{inv}_{S,S'} \sigma_k$, is defined in [140], using the predicate $\sqsubseteq$ as an inductive invariant. The relation $\preceq^{inv}_{S,S'}$ depends on S and S'. This is because strategy refinement takes into account an abstraction of the environment, even if the actual simulation relation is local to a pair of behaviors. The abstraction distinguishes states according to the interactions they disable; transitions are distinguished according to the set of interactions they can participate in. The global unfolding σ_G allows us to provide information about the desired global behavior, if available, to make the definitions of $\mathsf{enabled}$ and $\sqsubseteq$ less pessimistic. Whenever the global strategy σ_G is not explicitly specified, we conservatively assume $\sigma_G = \big(\|_{IM}(\{\sigma\} \cup \{C_i \mid i \neq k\})\big)/P$. The role of the refinement invariant is to provide information about related states in both systems S and S', e.g., to require equivalence of the states of two observers.

Strategy refinement is compositional:

Theorem 3.5 (Strategy refinement) *Let $S = \big(\|_{IM}\{C_i \mid i \in \mathbb{I}\}\big)/P$ and $S' = \big(\|_{IM'}\{C'_i \mid i \in \mathbb{I}\}\big)/P'$. Given unfoldings σ_i over C_i, σ'_i over C'_i, $i \in 1...n$, and σ_G over S, if $\forall i \,.\, \sigma'_i \preceq^{inv}_{S,S'} \sigma_i$ with respect to σ_G then $\sigma' \preceq^{inv}_{S,S'} \sigma_G$, where $\sigma' = \|_{IM'}\{\sigma'_i \mid i \in \mathbb{I}\}/P'$.*

That is, the composition of locally refining strategies is a refining strategy.

$$\begin{tikzcd}
 & \text{component } C_i & & \text{component } C'_i & & \text{component } C''_i \\
\textbf{local} & \text{unfolding } \sigma_i \arrow[rr, "\succeq"] & & \text{unfolding } \sigma'_i \arrow[rr, "\succeq"] & & \text{strategy } \sigma''_i \arrow[d, "\|"] \\
\textbf{global} & \text{unfolding } \sigma \arrow[u, "\pi_i"] \arrow[rr, "\succeq", "\text{Theorem 3.5}"'] & & \text{unfolding } \sigma' \arrow[u, "\pi_i"] \arrow[rr, "\succeq", "\text{Corollary 3.1}"'] & & \text{strategy } \sigma'' \\
 & \text{system } S & & \text{system } S' & & \text{system } S''
\end{tikzcd}$$

Figure 3.9: Design flow using strategy refinement.

Remark 3.2 *The definition of compositional refinement and Theorem 3.5 assume one-to-one component refinement, in order to keep the syntactic simplicity of one-to-one correspondence. It is still possible to reason about refinement where components are removed or added, by representing a component C_k that exists only in S (resp. S'), by a "neutral" component $\bar{C}_k$ in S' (resp. S) with the same controllable interactions $IM[k]^c$ that are always enabled.*

Definition 3.32 (Stability) *An unfolding p is* stable *in S if there exists a strategy σ and a refinement invariant inv such that $\sigma \preceq_{S,S}^{inv} p$.*

Corollary 3.1 (Stability) *Given an unfolding p over $S = \|_{IM}\{C_i \mid i \in \mathbb{I}\}/P$ and a refinement invariant inv, if for each component C_k, $p_k = \pi_k(p)$ is stable — say, $\sigma_k \preceq_{S,S}^{inv} p_k$ for some strategy σ_k — then p is stable, and $\|_{IM}\{\sigma_i \mid i \in \mathbb{I}\}/P \preceq_{S,S}^{inv} p$.*

The scheme of Fig. 3.9 summarizes a possible design flow supported by the results above.

3.4.4 Strategy Mapping

Theorem 3.5 provides a means to check for refinement between a pair of unfoldings. A strategy can be automatically and compositionally mapped on another system. The following definition and proposition allow us to map a strategy σ on an unfolding σ', that is, construct the maximal sub-strategy $\sigma'' \subseteq \sigma'$ that effectively refines σ.

In [140] we have defined the compositional mapping of a strategy σ_G over S onto a platform S'. More precisely, consider

- an unfolding σ_G with state space Q_G over $S = \big(\|_{IM}\{C_i \mid i \in \mathbb{I}\}\big)/P$ with state space Q_S;
- a target platform $S' = \big(\|_{IM'}\{C'_i \mid i \in \mathbb{I}\}\big)/P'$ with state space $Q_{S'}$;
- an unfolding σ'_k over C'_k;
- a refinement invariant *inv* over $Q_S \times Q_G \times Q_{S'}$.

Let σ_k be the projection of σ_G on C_k. The *mapping* of σ_k on σ'_k, written $\sigma_k \searrow_{inv} \sigma'_k$, is defined in [140] as an unfolding σ''_k over C'_k. Mapping uses our notion of strategy refinement, and restricts the target unfoldings to the largest strategies such that their composition refines the composition of the source strategies. Strategy mapping satisfies the following properties.

Proposition 3.4 *If σ'_k is a strategy then $\sigma_k \searrow_{inv} \sigma'_k$ is a strategy.*

Mapping an unfolding σ on a target unfolding σ', rather than directly on the underlying behavior B, helps to improve precision, as mapped states may be distinguished in a mapping on σ' that may be confused when mapping on B.

Proposition 3.5 (Correctness) *Let* $S = \big(\|_{IM}\{C_i \mid i \in \mathbb{I}\}\big)/P$, $S' = \big(\|_{IM'}\{C'_i \mid i \in \mathbb{I}\}\big)/P'$, $\sigma_i = (Q_i, IM[i], \rightarrow_i, l_i, Q^0_i)$ *be an unfolding on* C_i, $i = 1,...,n$, *and inv be a refinement invariant. If* $\forall i \,.\, C'_i \preceq^{inv}_{S,S'} \sigma_i$, *then*

- $\forall k \,.\, \sigma_k \searrow_{inv} C'_k \preceq^{inv}_{S,S'} \sigma_k$, *and*
- $\|_{IM'}\{\sigma_i \searrow_{inv} C'_i \mid i \in \mathbb{I}\}/P' \preceq^{inv}_{S,S'} \|_{IM}\{\sigma_i \mid i \in \mathbb{I}\}/P$.

Proposition 3.6 (Completeness) *If* $(\sigma \searrow_{inv} C') \npreceq^{inv}_{S,S'} \sigma$, *then there is no strategy* σ' *over* C' *refining* σ *under inv.*

3.4.5 Case Study: Distributed Mutual Exclusion

An interesting application domain of strategy mapping is to solve the following problem: how to deploy a centralized component-based system, whose components are coordinated through synchronization primitives, on a distributed platform where only lower-level communication primitives are available? We illustrate the principle with the mapping of a simple strategy ensuring mutual exclusion and fairness, on Kessels' distributed mutual exclusion algorithm [176].

Consider the centralized system $mutex = \|_{IM}\{P_1, P_2, S\}$ of Fig. 3.7 with $IM = \{a_1, p_1|p, v_1|v, a_2, p_2|p, v_2|v\}$ and the first-come-first-served strategy σ of Fig. 3.8. We call ww_1 (resp. ww_2) the state labeled with w_1w_2 reached with $a_1; a_2$ (resp. $a_2; a_1$) in which P_1 (resp. P_2) is served first. All other states of σ are uniquely identified by their label. Let σ_{P_1}, σ_{P_2}, and σ_S be the projections of σ on P_1, P_2, and S, respectively. We want to map strategy σ on the target platform given by two components implementing Kessels' mutual exclusion algorithm [176]:

```
K1:                                     K2:
req[1] = 1;                             req[2] = 1;
turn[1] = turn[2];                      turn[2] = !turn[2];
await (!req[2] or turn[1]!=turn[2]);    await (!req[1] or turn[1]=turn[2]);
// critical section                     // critical section
req[1] = 0;                             req[2] = 0;
```

According to Remark 3.2, we model Kessels' algorithm with two components and one dummy component $obs = \big(\{q_\perp\}, \{p, v\}, \{(q_\perp, \{p\}, q_\perp), (q_\perp, \{v\}, q_\perp)\}\big)$ "refining" the semaphore. *obs* has only one state and is always ready to make a p or v action. We will use this component as an observer to identify transitions in which one of the components K_1, K_2 enters or leaves its critical section.

Kessels' algorithm is modeled as component model $ks = \|_{IM'}\{K_1, K_2, obs\}/P$ with K_1 and K_2 as shown in Fig. 3.10, $IM' = \{t_1, t'_1, a_1, p_1|p, v_1|v, t_2, t'_2, a_2, p_2|p, v_2|v\}$ and $P = \big(true, (V^\alpha)_{\alpha \in IM}\big)$ defined as follows:

α	V^α
t_1	$turn_2$
t'_1	$\neg turn_2$
$p_1\|p$	$\neg req_2 \lor turn_1 \neq turn_2$
t_2	$\neg turn_1$
t'_2	$turn_1$
$p_2\|p$	$\neg req_1 \lor turn_2 = turn_1$

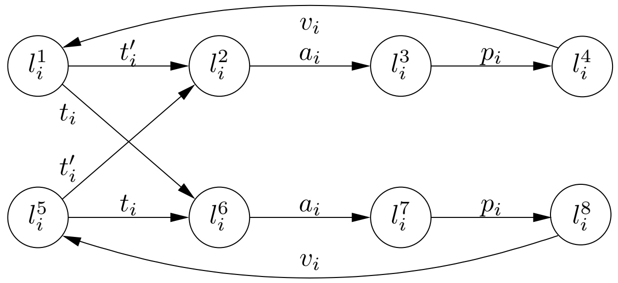


Figure 3.10: Behavior of component K_i.

and $V^\alpha = \mathit{true}$ for all other $\alpha \in IM'$, where $req_i = l_i^2 \vee l_i^3 \vee l_i^4 \vee l_i^6 \vee l_i^7 \vee l_i^8$, and $turn_i = l_i^5 \vee l_i^6 \vee l_i^7 \vee l_i^8$. While the Bip framework supports variable assignment between components [24, 138], the component states are explicit in the simplified presentation adopted here. We therefore represent the reading with a pair of transitions t_i and t_i' and the execution model shown above.

Let us now map strategy σ on ks, that is, implement mutual exclusion with the fairness constraint modeled by σ, in a distributed setting where the components K_1, K_2 communicate only through shared variables. According to Kessels' algorithm where in case of conflict, K_1 is given priority over K_2 if and only if $turn_1 \neq turn_2$, we fix the refinement invariant

$$inv = (ww_1 \implies turn_1 \neq turn_2) \wedge (ww_2 \implies turn_1 = turn_2)$$

and compute

$$\begin{aligned}\sqsubseteq_{P_1} = \mathit{inv}\ \wedge\ &\big(\mathsf{enabled}_{P_1,\sigma}(a_2) \implies \neg\mathsf{disabled}_{K_1,ks}(a_2)\big)\wedge\\ &\big(\mathsf{enabled}_{P_1,\sigma}(p_2|p) \implies \neg\mathsf{disabled}_{K_1,ks}(p_2|p)\big)\wedge\\ &\big(\mathsf{enabled}_{P_1,\sigma}(v_2|v) \implies \neg\mathsf{disabled}_{K_1,ks}(v_2|v)\big)\\ = \mathit{inv} \wedge &\big(\neg w_2 \vee ww_1 \vee u_1\ \vee\ \neg(l_2^3 \vee l_2^7)\ \vee\ \neg req_1 \vee turn_1 = turn_2\big)\end{aligned}$$

for component P_1/K_1, and similarly for components P_2/K_2 and S/obs. The refinement relation $\leq = \leq^{inv}_{mutex,ks} (\sigma_{P_1}, K_1)$ is shown in Table 3.1.

	l_1^1	l_1^2	l_1^3	l_1^4	l_1^5	l_1^6	l_1^7	l_1^8
s_1	●	●	-	-	●	●	-	-
w_1	●	●	●	-	●	●	●	-
u_1	-	-	-	●	-	-	-	●

Table 3.1: $\prec$ after 1 iteration and greatest fixpoint $\leq^{inv}_{mutex,ks} (\sigma_{P_1}, K_1)$ of $\prec$.

In order to prove refinement of P_1 by K_1, we have to show according to Definition 3.31 that (1) the initial state s_1 of P_1 is refined by some initial state of K_1 (all of whose states are by definition initial states in its trivial unfolding): this condition is obviously satisfied; and (2) for any state q' among the states l_1^1, l_1^2, l_1^5, l_1^6 simulating the initial state s_1 of the projection of σ, we have $\sqsubseteq_{P_1} [(s_1, s_2, idle)/Q_{mutex}, q'/Q_{K_1}] = \mathit{true}$). This condition is satisfied, too. It follows that $K_1 \preceq^{inv}_{mutex,ks} \pi_{P_1}(\sigma)$. Similarly, it can be shown that $K_2 \preceq^{inv}_{mutex,ks} \pi_{P_2}(\sigma)$ and $obs \preceq^{inv}_{mutex,ks} \pi_S(\sigma)$. The mappings $\sigma_{K_1} := \sigma_{P_1} \searrow_{inv} K_1$, $\sigma_{K_2} := \sigma_{P_2} \searrow_{inv} K_2$, and $\sigma_{obs} := \sigma_S \searrow_{inv} obs$ are equal to the target components K_1, K_2, and obs, that is, their full behavior of the target components is used to implement σ. With Theorem 3.5, global refinement, that is, $ks \preceq^{inv}_{mutex,ks} \sigma$, follows.

In this example, we have mapped σ on a system whose behavior is very close to that of σ. This does not need to be the case, however, as long as the available behavior of the target platform is "expressive enough" to refine σ.

3.4.6 Discussion

We have presented an approach to compositionally cope with strategies, by way of refinement and mapping, in a platform-based design process consisting of successive mapping steps. The latter are performed component-wise; compositionality results ensure that the refinement carries over to the global strategy. The results are formulated in a subset of the heterogeneous component framework Bip.

The complexity of the definitions of strategy refinement and strategy mapping — only sketched here — comes, on one hand, from the experience of case studies — like the one detailed above — that showed that former versions of strategy refinement were too strict and hence, often were not satisfied by concrete examples. This led to the introduction of a refinement invariant, and the use of $\sqsubseteq$ as an inductive invariant. The second source of complexity is the expressiveness of the Bip interaction model.

Related work. Our notion of strong refinement from [140] uses a particular form of bisimulation to ensure that the controllable behavior of the abstract strategy is implemented by the lower-level strategy, and that the latter, in turn, does not have any behavior not consistent with the former. A similar approach has been adopted to verify action contraction [235] and non-atomic refinement [105], and prove linearizability [166] of concurrent CSP processes [104]. The existence of refinement mappings with respect to both safety and liveness properties has been studied in [1]. More recently, compositional refinement of reachability properties has been studied in [64] in a framework of modal specifications.

Future work. This work opens several interesting research directions.

First, the results should be reformulated in a modeling framework with a less expressive notion of glue — thus allowing for more straight-forward definitions of strategy refinement —, for instance synchronous data-flow frameworks such as [258].

It seems particularly promising to apply the approach to the verification and design of highly concurrent systems of loosely interacting components, such as sensor networks or genetic networks.

Furthermore it would be interesting to study whether our definition of refinement can be relaxed — e.g., using the notion of *coupled simulation* from [105] which allows a *sequence of actions* to be refined by another sequence — while preserving compositionality.

Considering unfoldings as programs, uncontrollable events as faults, and strategies as fault-tolerant programs, an interesting application of this work would be a compiler dedicated to, or parametrized with, source and target platforms S and S', that compiles a program σ over S into an equivalent fault-tolerant program σ' running on S'.

3.5 Conformance Checking for Choreographies of Components Interacting Asynchronously

Choreography specification languages describe from a global point of view interactions among a set of services in a system to be designed. Given a choreography specification, the goal is to obtain a distributed implementation of the choreography as a system of communicating components (also called *peers*). The protocols of these peers can be given as input or automatically generated by projection from the choreography. Verifying whether some set of peers implements a choreography specification is called *conformance checking*. This check is in general undecidable if asynchronous communication is considered, that is, services interact through message buffers.

In [149] we have leveraged a recent decidability result on quasi-static scheduling [95] to check automatically the conformance of a set of peers under an asynchronous communication model with a priori unbounded buffers, and compute the minimal required buffer size. The verification relies on exploring only a subset called the *canonical schedule* of the reachable state space. Finiteness of the canonical schedule implies conformance for sufficiently large but finite buffers. The use of a dominance order on states allows us to decide whether the canonical schedule is infinite.

3.6 Fault Recovery in Bip

In [53, 54] we have introduced a generic formal framework for specifying and reasoning about fault recovery (also called *non-masking* fault-tolerance) for component-based models. We characterize component-based models based on Bip. However, our method is not limited to Bip. Unlike the approaches in [17, 186, 200] where a monolithic model is analyzed or components are defined in terms of properties of sets of computations, our method is based on observational behavior of a model in the presence of faults.

We have defined what it means for component models to provide a recovery mechanism, so that the model converges to its normal behavior in the presence of faults (e.g., in self-stabilizing systems), and presented a sufficient condition for incrementally constructing non-masking models. We have further defined what it means for a component to be a *corrector* and shown that non-masking models must contain corrector components; these components correct the observational behavior of a faulty model. We have shown how they can be constructed as distinct components interacting with components that provide functional tasks, so as to separate recovery from functional concerns: when a normal execution phase is interrupted by the occurrence of faults, control is transferred from the impacted functional components to corrector components in charge of fault handling and recovery, and handed back to the functional components once normal behavior is reestablished. Finally, we have illustrated that any non-masking model can be transformed into an equivalent model, where functional and recovery tasks are modularized in different components.

3.7 Verification of Weakly-Hard Requirements on Quasi-Synchronous Systems

Synchronous Languages [38] such as Lustre [161], Esterel [45], and Signal [41], are based on clock calculi as formalisms to express and reason about (Boolean) constraints on the presence of signals, e.g. to define a sub-clock relation. These formalisms fit well their intended purpose: to reason about synchronous systems where time is abstracted to a sequence of discrete instants.

As maintaining the hypothesis of a synchronous clock may become expensive or infeasible for distributed implementations, distribution has been the focus of many publications since the early days of synchronous programming. Existing distribution methods maintain the synchronous semantics in the distributed implementation by means of protocols [259, 71, 70] or a combination of protocols and assumptions on the clock drifts [39, 40]. These assumptions, and the reasoning about the properties they ensure, are made *outside* of the model in the form of theorems, each of which has to be proven “by hand”. In contrast, in many systems such as distributed control systems, occasional deviations of the behavior of the distributed implementation from the synchronous model may be acceptable as long as the frequency of such deviations is bounded [179]. The goal of Gideon Smeding's PhD thesis [248] was to propose a quasi-synchronous framework encompassing constraints on the relative speed of clocks, together with a formalism for reasoning about clock-dependent properties within the model, in order to provide a seamless link between synchronous models and their asynchronous implementation.

The quasi-synchronous approach developed in [248] considers independently clocked, locally synchronous components that interact via communication-by-sampling or FIFO channels. We have defined relative drift bounds on pairs of recurring events such as clock ticks or the arrival of a message. Drift bounds express constraints on the stability of clocks, e.g., clock x ticks at least twice within any three consecutive ticks of clock y. We can thus move from total synchrony, where all clocks tick simultaneously, to global asynchrony by relaxing the drift bounds. As constraints are more relaxed, behavior diverges more and more from the synchronous system behavior. The approach of [248] takes as inputs a program written in a Lustre-like language extended with asynchronous communication by sampling, application requirements on the distribution in the form of weakly-hard constraints [42] bounding e.g. the tolerated loss of data tokens, and platform assertions (e.g., relative clock speeds, available communication resources), and verifies whether the program meets the requirements under the platform assertions.

A second contribution of [248] is the use of drift bounds for performance analysis of stream processing systems. This clock calculus captures correlated variations of streams. In the common case of a data-flow system that splits a stream for separate treatment, and joins them afterwards, the proposed analysis yields more precise results than comparable methods, especially in the case of bursty behavior [249].

The targeted application domains of [248] include contract-based design of real-time systems — where the guaranteed behavior of a component will depend on the timing guarantees provided by its context —, and formal reasoning about non strictly semantics-preserving desynchronization, for instance to support synchronous programming of sensor networks.

Chapter 4

Contract-based Design

Contracts have first been introduced as a type system for classes [211]: a method guarantees some post-condition under the assumption that its pre-condition is satisfied. In the component-based programming community, contracts are moving in the focus of research as a means to achieve one of the main goals of the component paradigm, namely the deployment and reuse of components in different, a priori unknown contexts. As components may interact under various models of communication, the notion of contract has been generalized from pre- and post-conditions in the form of predicates to *behavioral contracts*, allowing the designer to reason about the temporal behavior of environments with which a component can be composed. In the literature, contracts have been used at different stages of the design process and with different goals, including requirements engineering [36] and runtime verification [21]. For program development it has been argued [211] that contract-based design should replace defensive programming. This approach is currently gaining popularity with the advent of highly reconfigurable multi-processor-on-chip (MPSoC) designs where correctness of each functional block — for instance, coherence of shared memory — with respect to all possible uses is traded against correctness only under some assumptions, and better efficiency.

In this chapter we study contract frameworks over Bip. In §4.1, we introduce a theory of modal assume/guarantee-contracts, and study different composition operations supporting bottom-up and top-down construction in a design flow. In §4.2, we study a specification theory for probabilistic contracts.

4.1 Modal Assume/Guarantee Contracts

4.1.1 Motivation

In contrast to a *specification* defining how a component *must* behave, contracts can be seen as implications, providing a guarantee depending on an assumption on the context. Accordingly, different semantics of contract composition are conceivable, with the two special cases of *conjunction of implications* yielding a *lazy* composition, and *implication of a conjunction* for an *eager* composition. The latter approach is adopted by [190], where the assumption of the composed contract is defined as the weakest assumption ensuring the conjunction of both guarantees. In the present work we choose the former approach: a component satisfying the composition of two contracts must satisfy each guarantee if and only if the corresponding assumption holds. This notion of composition is consistent with the component paradigm mentioned above, enabling the component to offer different guarantees depending on the context.

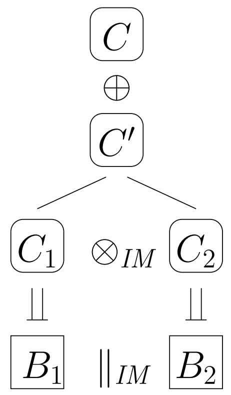


Figure 4.1: Example of a contract-based design flow.

In a component-based design flow, contracts and their composition may be used with different goals in the design flow: on the one hand, a contract may be used to describe the guarantees a component is able to give, depending on its environment. We call this a *component contract*. Component contracts can be used top-down to successively refine an abstract component, or bottom-up to build a system from previously constructed components. On the other hand, a contract may specify a *requirement* as a guarantee that must be ensured under some hypothesis. We call this a requirement contract, or *aspect*; sometimes they are also referred to as *use cases*. Aspects are usually implemented top-down. Therefore, contracts are an elegant way to combine bottom-up and top-down design. Although syntactically there is no difference between both kinds of contracts, the difference comes from the way they are composed.

For instance, suppose that we want to design a controller for a mobile video player satisfying the requirements $\mathcal{C}$ ("the incoming video stream is decoded in high quality when sufficient power is available, and with degraded quality of service otherwise") and $\mathcal{C}'$ ("the sound volume is limited if headphones are plugged in"). We can design our controller by constructing $\mathcal{C} \oplus \mathcal{C}'$, decomposing the resulting contract (manually or by quotienting with the modal specification of a legacy component [229]) into simpler contracts $\mathcal{C}_1$ and $\mathcal{C}_2$, and separately implementing both. The controller is then obtained by composing the implementations B_1 and B_2. The design flow is shown in Figure 4.1.

For component contracts over disjoint components we define a "best effort" composition operation that is parametrized by an interaction model. The composition ensures each guarantee depending on the satisfaction of its assumption, provided that the guarantee is feasible under the specified interaction model. We show that this operation satisfies the property of *independent implementability*.

For aspects on the same component or sub-system we define a composition operation based on modal conjunction to ensure that the composition refines both contracts. This is motivated by the fact that different aspects express different requirements whose conjunctions is to be satisfied. The same operation serves to compose an aspect with a component. It is shown to be *sound* and, under some conditions, complete. To our knowledge this is the first work formalizing and allowing us to effectively combine both types of contract composition.

Furthermore, we define a prioritized composition of aspects whose result refines the composed aspects, such that in case of inconsistencies among them, an aspect of higher priority "overrides" a lower-priority contract.

In [148] we define contracts in terms of *modal automata* [189] extending automata with a modality that indicates for each transition whether it *may* or *must* be implemented. This more expressive framework has the advantage of keeping a larger design space, whereas a premature choice of implementing or not a given transition would prematurely narrow the design space, and rule out possible implementations.

Related work. Modal specifications benefit from a well-established theory and a set of results that we build upon, in particular work on modal residuation [229]. A detailed discussion of benefits of modalities for interface theories, in particular for fitting together contracts over different action vocabularies, can be found in [231]. Since we introduce two distinct operations for composing contracts over disjoint interaction models and a common interaction model, respectively, we do not encounter this issue here.

Verification based on modal contracts in Bip is studied in [227], where a decomposition of contracts is used to define compositional refinement of component contracts. [36] discusses a contract-based design flow for a rich component framework. Aspects are formalized in terms of pairs (assumption, guarantee) of sets of traces. Conjunction of non-modal specifications has been introduced in [202] with the goal of enabling heterogeneous specifications mixing operational and logical parts.

Outline. In §4.1.2 the lattice of modal specifications is defined. We introduce two new operations on modal specification called *weak implication* and *priority composition*, and discuss their properties. §4.1.3 introduces contracts as pairs of modal specifications. In §4.1.4 we use the operations on modal specifications to define several composition operations on contracts.

4.1.2 Modal Specifications

Automata enriched with modalities on transitions have been introduced in [189] two decades ago (see [13] for a complete survey). Basically, modal specifications possess two types of transitions: *may*-transitions that are optional, as opposed to *must*-transitions that are mandatory. We fix Σ° as the *universal alphabet.*

Definition 4.1 (Modal specification) *A* modal specification *is a tuple $\mathcal{S} = \langle Q, q^0, \Sigma, \Delta^m, \Delta^M \rangle$ where Q is a finite set of states, $q^0 \in Q$ is the unique initial state, $\Sigma \subseteq \Sigma^\circ$ is a finite set of actions (or ports), and $\Delta^m, \Delta^M \subseteq Q \times \Sigma \times Q$ are respectively the set of may-transitions and of must-transitions. We require that Δ^m is deterministic and that $\Delta^M \subseteq \Delta^m$.*

The condition $\Delta^M \subseteq \Delta^m$ naturally imposes that every required transition (i.e., in Δ^M) is also allowed (i.e., is also in Δ^m). In the sequel we will refer to this as the *consistency condition.*

For short, we shall write or draw: $q \xrightarrow{a} q'$ when $(q, a, q') \in \Delta^M$; $q \overset{a}{\dashrightarrow} q'$ when $(q, a, q') \in \Delta^m \setminus \Delta^M$; $q \overset{a}{\nrightarrow}$ when $\forall q' : \ (q, a, q') \notin \Delta^m$.

Example 4.1 *Consider a communication channel whose alphabet of actions includes* msg *for a sending request and two kinds of acknowledgment for transmission:* ack *in case of success and* nack *in case of failure. The modal specification in Fig. 4.2 specifies that every message sent must be acknowledged.*

Definition 4.2 (Behavior) *We call* behavior *a modal specification $\langle Q, q^0, \Sigma, \Delta^m, \Delta^M \rangle$ where $\Delta^M = \Delta^m$.*

Thus, a behavior does not have any optional transition and can be seen as a standard LTS.

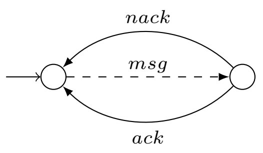


Figure 4.2: A modal specification

When composing specifications (several composition operations will be introduced later on), inconsistencies between the modalities may appear. We thus consider the following extension of modal specification called *pseudo-modal specifications* in which we relax the consistency condition in order to model the possible discrepancies:

Definition 4.3 (Pseudo-modal specification) *A* pseudo-modal specification *is a tuple* $p\mathcal{S} = \langle Q, q^0, \Sigma, \Delta^m, \Delta^M \rangle$ *similar to a modal specification, except that the consistency condition is relaxed i.e.,* $\Delta^M \not\subseteq \Delta^m$ *is possible.*

Modal specifications correspond to the subclass of pseudo-modal specifications for which $\Delta^M \subseteq \Delta^m$. As a consequence, the definitions below for pseudo-modal specifications also apply to modal specifications. For $q \in Q$, we denote:

- $\mathit{may}(q) = \{a \in \Sigma \mid \exists q' \in Q : (q, a, q') \in \Delta^m\}$ the set of *allowed* actions;
- $\mathit{must}(q) = \{a \in \Sigma \mid \exists q' \in Q : (q, a, q') \in \Delta^M\}$ the set of *required* actions;
- $\mathit{mustnot}(q) = \Sigma \setminus \mathit{may}(q)$ the set of *forbidden* actions.

A state q such that $\mathit{must}(q) \not\subseteq \mathit{may}(q)$ is said *inconsistent*, denoted $\lightning q$, and *consistent* otherwise. We write $\lightning Q \subseteq Q$ for the set of inconsistent states.

The Lattice of Modal Specifications

We now define the semantics of modal and pseudo-modal specifications.

Refinement. Refining a pseudo-modal specification boils down to preserve the *must*-transitions and to possibly transform some optional transitions into *must*-transitions or to take them off the specification.

Definition 4.4 (Refinement) *A pseudo-modal specification* $p\mathcal{S}_1 = \langle Q_1, q_1^0, \Sigma_1, \Delta_1^m, \Delta_1^M \rangle$ *refines a pseudo-modal specification* $p\mathcal{S}_2 = \langle Q_2, q_2^0, \Sigma_2, \Delta_2^m, \Delta_2^M \rangle$ *with* $\Sigma_2 \subseteq \Sigma_1$*, written* $p\mathcal{S}_1 \preceq p\mathcal{S}_2$*, if there exists a simulation relation* $\theta \subseteq Q_1 \times Q_2$ *such that:* $(q_1^0, q_2^0) \in \theta$*, and for all* $(q_1, q_2) \in \theta$*, the following holds:*

- *for every* $(q_2, a, q_2') \in \Delta_2^M$ *there exists* $(q_1, a, q_1') \in \Delta_1^M$ *and* $(q_1', q_2') \in \theta$*;*
- *for every* $(q_1, a, q_1') \in \Delta_1^m$ *with* $a \in \Sigma_2$ *there exists* $(q_2, a, q_2') \in \Delta_2^m$ *and* $(q_1', q_2') \in \theta$*;*
- *if* $(q_1, a, q_1') \in \Delta_1^m$ *with* $a \notin \Sigma_2$ *then* $(q_1', q_2) \in \theta$*.*

For any pair $(q_1, q_2) \in \theta$ we have by definition:

$$\left\{ \begin{array}{rcl} \mathit{may}_1(q_1) & \subseteq & \mathit{may}_2(q_2) \cup (\Sigma_1 \setminus \Sigma_2) \\ \mathit{must}_1(q_1) & \supseteq & \mathit{must}_2(q_2) \end{array} \right.$$

Models of a pseudo-modal specification are an ultimate refinement:

Definition 4.5 (Model relation) *A behavior $\mathcal{B}$ is a* model *of a pseudo-modal specification $p\mathcal{S}$, denoted $\mathcal{B} \models p\mathcal{S}$, if $\mathcal{B} \preceq p\mathcal{S}$. The set of models of $p\mathcal{S}$ is denoted by $\mathsf{Mod}(p\mathcal{S})$.*

We write $\mathcal{S}_\top$ for the modal specification $\langle \{q^0\}, q^0, \emptyset, \emptyset, \emptyset \rangle$; $\mathcal{S}_\top$ is greater for $\preceq$ than all modal specifications and it admits every behavior as a model. On the other hand, let $\mathcal{S}_\bot$ be the modal specification $\langle \{q^0\}, q^0, \Sigma^\circ, \emptyset, \Delta^M \rangle$ with $\Delta^M = \{(q^0, a, q^0) \mid a \in \Sigma^\circ\}$. $\mathcal{S}_\bot$ refines all modal specifications, and its set of models is empty.

It is possible to transform a pseudo-modal specification into a modal specification without changing its set of models [229]. We call this operation *reduction* and denote $\rho(p\mathcal{S})$ the reduction of $p\mathcal{S}$. It consists in pruning away the inconsistent states of $p\mathcal{S}$. It follows the observation that when $\mathcal{B}_1 \models p\mathcal{S}_2$, for any $(q_1, q_2) \in \theta$:

$$must_2(q_2) \subseteq must_1(q_1) = may_1(q_1) \subseteq may_2(q_2)$$

and thus q_2 is consistent.

For $U \subseteq Q$, let $pre^M(U) = \{q \in Q \mid (q, a, q') \in \Delta^M \wedge q' \in U\}$ and let

$$\begin{array}{rcll} pre_0^M(U) & = & U & \\ pre_{k+1}^M(U) & = & pre^M(pre_k^M(U)) & \text{for } k \geq 0 \\ pre_*^M(U) & = & \bigcup_k pre_k^M(U) & \end{array}$$

Definition 4.6 (Reduction ρ) *The reduction $\rho(p\mathcal{S})$ of a pseudo-modal specification $p\mathcal{S} = \langle Q, q^0, \Sigma, \Delta^m, \Delta^M \rangle$ is defined as $\mathcal{S}_\bot$ if $q^0 \in pre_*^M(\lightning Q)$, and as the modal specification $\langle Q \setminus pre_*^M(\lightning Q), q^0, \Sigma, \Delta_\rho^m, \Delta_\rho^M \rangle$ otherwise, where $\Delta_\rho^m \subseteq \Delta^m$ and $\Delta_\rho^M \subseteq \Delta^M$ are the sets of transitions of $p\mathcal{S}$ whose source state and destination state do not belong to $pre_*^M(\lightning Q)$.*

This construction is similar to the synthesis of a most permissive controller [232] with states from $\lightning Q$ interpreted as the bad states and with must-transitions as transitions labeled by uncontrollable events.

In the sequel, pseudo-modal specifications may serve as an artifice when composing modal specifications; conflicts between the modalities of the composed modal specifications may generate an inconsistent state. By then applying the reduction operation, a semantically equivalent modal specification is obtained.

Remark 4.1 *The main reason for assuming determinism in the may-transition relation for modal specifications is that the modal refinement then coincides with the inclusion of sets of models [229]: let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two modal specifications:*

$$\mathcal{S}_1 \preceq \mathcal{S}_2 \iff \mathsf{Mod}(\mathcal{S}_1) \subseteq \mathsf{Mod}(\mathcal{S}_2)$$

If non-determinism is allowed in $\mathcal{S}_1$ or $\mathcal{S}_2$, modal refinement is not complete [191].

Modal specifications equipped with modal refinement form a complete lattice. The construction of their infimum and their supremum are introduced next.

Definition 4.7 (Greatest lower bound $\wedge$) *The* greatest lower bound *of two modal specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ is $\mathcal{S}_1 \wedge \mathcal{S}_2 = \rho(\mathcal{S}_1 \& \mathcal{S}_2)$ where $\mathcal{S}_1 \& \mathcal{S}_2 = \langle Q_1 \times Q_2, (q_1^0, q_2^0), \Sigma_1 \cup \Sigma_2, \Delta^m, \Delta^M \rangle$ where*

$$\Delta^m = \left\{ \big((q_1, q_2), a, r\big) \mid r = \left\{ \begin{array}{ll} (q_1', q_2') & \textit{if } (q_1, a, q_1') \in \Delta_1^m \textit{ and } (q_2, a, q_2') \in \Delta_2^m \\ (q_1', q_2) & \textit{if } (q_1, a, q_1') \in \Delta_1^m \textit{ and } a \notin \Sigma_2 \\ (q_1, q_2') & \textit{if } (q_2, a, q_2') \in \Delta_2^m \textit{ and } a \notin \Sigma_1 \end{array} \right. \right\}$$

$\wedge$	$q_2 \xrightarrow{a}_2 q_2'$	$q_2 \overset{a}{\dashrightarrow}_2 q_2'$	$q_2 \overset{a}{\not\rightarrow}_2$	$a \notin \Sigma_2$
$q_1 \xrightarrow{a}_1 q_1'$	$(q_1, q_2) \xrightarrow{a} (q_1', q_2')$	$(q_1, q_2) \xrightarrow{a} (q_1', q_2')$	$\lightning(q_1, q_2)$	$(q_1, q_2) \xrightarrow{a} (q_1', q_2)$
$q_1 \overset{a}{\dashrightarrow}_1 q_1'$	$(q_1, q_2) \xrightarrow{a} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\not\rightarrow}$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1', q_2)$
$q_1 \overset{a}{\not\rightarrow}_1$	$\lightning(q_1, q_2)$	$(q_1, q_2) \overset{a}{\not\rightarrow}$	$(q_1, q_2) \overset{a}{\not\rightarrow}$	$(q_1, q_2) \overset{a}{\not\rightarrow}$
$a \notin \Sigma_1$	$(q_1, q_2) \xrightarrow{a} (q_1, q_2')$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1, q_2')$	$(q_1, q_2) \overset{a}{\not\rightarrow}$	N/A

Table 4.1: Transition relation of $\mathcal{S}_1 \wedge \mathcal{S}_2$

$\vee$	$q_2 \xrightarrow{a}_2 q_2'$	$q_2 \overset{a}{\dashrightarrow}_2 q_2'$	$q_2 \overset{a}{\not\rightarrow}_2$
$q_1 \xrightarrow{a}_1 q_1'$	$(q_1, q_2) \xrightarrow{a} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\dashrightarrow} q_1'$
$q_1 \overset{a}{\dashrightarrow}_1 q_1'$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\dashrightarrow} q_1'$
$q_1 \overset{a}{\not\rightarrow}_1$	$(q_1, q_2) \overset{a}{\dashrightarrow} q_2'$	$(q_1, q_2) \overset{a}{\dashrightarrow} q_2'$	$(q_1, q_2) \overset{a}{\not\rightarrow}$

Table 4.2: Transition relation of $\mathcal{S}_1 \vee \mathcal{S}_2$

$$\Delta^M = \Big\{ \big((q_1, q_2), a, r\big) \mid r = \begin{cases} (q_1', q_2') & \textit{if } (q_1, a, q_1') \in \Delta_1^M \textit{ and } (q_2, a, q_2') \in \Delta_2^m, \\ & \textit{or } (q_1, a, q_1') \in \Delta_1^m \textit{ and } (q_2, a, q_2') \in \Delta_2^M \\ (q_1', q_2) & \textit{if } (q_1, a, q_1') \in \Delta_1^M \textit{ and} \\ & \big(a \notin \Sigma_2 \textit{ or } \forall q_2' : (q_2, a, q_2') \notin \Delta_2^m\big) \\ (q_1, q_2') & \textit{if } (q_2, a, q_2') \in \Delta_2^M \textit{ and} \\ & \big(a \notin \Sigma_1 \textit{ or } \forall q_1' : (q_1, a, q_1') \notin \Delta_1^m\big) \end{cases} \Big\}$$

The resulting transitions are explicitly enumerated in Table 4.1. The greatest lower bound operation coincides with a logical conjunction [228]: for any modal specifications $\mathcal{S}_1$ and $\mathcal{S}_2$,

$$\mathsf{Mod}(\mathcal{S}_1 \wedge \mathcal{S}_2) = \mathsf{Mod}(\mathcal{S}_1) \cap \mathsf{Mod}(\mathcal{S}_2)$$

Definition 4.8 (Least upper bound $\vee$) *The* least upper bound $\mathcal{S}_1 \vee \mathcal{S}_2$ *of two modal specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ is the tuple $\langle (Q_1 \times Q_2) \cup Q_1 \cup Q_2, (q_1^0, q_2^0), \Sigma_1 \cap \Sigma_2, \Delta^m, \Delta^M \rangle$, where $\Delta^m = \Delta_1^m \cup \Delta_2^m \cup \Delta_\vee^m$, $\Delta^M = \Delta_1^M \cup \Delta_2^M \cup \Delta_\vee^M$, and*

$$\Delta_\vee^m = \Big\{ \big((q_1, q_2), a, r\big) \mid r = \begin{cases} (q_1', q_2') & \textit{if } (q_1, a, q_1') \in \Delta_1^m \textit{ and } (q_2, a, q_2') \in \Delta_2^m \\ q_1' & \textit{if } (q_1, a, q_1') \in \Delta_1^m \textit{ and } \nexists q_2' : (q_2, a, q_2') \in \Delta_2^m \\ q_2' & \textit{if } (q_2, a, q_2') \in \Delta_2^m \textit{ and } \nexists q_1' : (q_1, a, q_1') \in \Delta_1^m \end{cases} \Big\}$$

$$\Delta_\vee^M = \big\{ \big((q_1, q_2), a, (q_1', q_2')\big) \mid (q_1, a, q_1') \in \Delta_1^M \wedge (q_2, a, q_2') \in \Delta_2^M \big\}$$

The resulting transitions are explicitly enumerated in Table 4.2.

Remark 4.2 *The least upper bound of modal specification does not match with a logical disjunction. In fact, modal specification are not closed under disjunction [114]. However, $\mathcal{S}_1 \vee \mathcal{S}_2$ is the least specification for the refinement preorder whose set of models contains $\mathsf{Mod}(\mathcal{S}_1) \cup \mathsf{Mod}(\mathcal{S}_2)$.*

Weak Implication

This section introduces a new operation on modal specification called *weak implication* which is a partial adjoint of the conjunction. In other words, the weak implication $\mathcal{S} \div \mathcal{S}_1$ characterizes the modal specifications $\mathcal{X}$ solving the equation $\mathcal{S}_1 \wedge \mathcal{X} \leq \mathcal{S}$.

$\div$	$q_2 \xrightarrow{a}_2 q_2'$	$q_2 \overset{a}{\dashrightarrow}_2 q_2'$	$q_2 \overset{a}{\not\rightarrow}_2$	$a \notin \Sigma_2$
$q_1 \xrightarrow{a}_1 q_1'$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1', q_2')$	$(q_1, q_2) \xrightarrow{a} (q_1', q_2')$	$(q_1, q_2) \xrightarrow{a} \top$	$(q_1, q_2) \xrightarrow{a} (q_1', q_2)$
$q_1 \overset{a}{\dashrightarrow}_1 q_1'$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\dashrightarrow} \top$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1', q_2)$
$q_1 \overset{a}{\not\rightarrow}_1$	$(q_1, q_2) \overset{a}{\not\rightarrow}$	$(q_1, q_2) \overset{a}{\not\rightarrow}$	$(q_1, q_2) \overset{a}{\dashrightarrow} \top$	$(q_1, q_2) \overset{a}{\not\rightarrow}$

Table 4.3: Transition relations of $\mathcal{S}_1 \div \mathcal{S}_2$

Definition 4.9 (Weak implication $\div$) *The* weak implication *of two modal specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ is $\mathcal{S}_1 \div \mathcal{S}_2 = \langle (Q_1 \times Q_2) \cup \{\top\}, (q_1^0, q_2^0), \Sigma_1, \Delta^m, \Delta^M \rangle$ where*

$$\Delta^m = \Big\{ \big((q_1, q_2), a, r\big) \mid r = \begin{cases} (q_1', q_2') & \text{if } (q_1, a, q_1') \in \Delta_1^m \text{ and } (q_2, a, q_2') \in \Delta_2^m \\ (q_1', q_2) & \text{if } (q_1, a, q_1') \in \Delta_1^m \text{ and } a \notin \Sigma_2 \\ \top & \text{if } \forall\, q_2' : (q_2, a, q_2') \notin \Delta_2^m \end{cases} \Big\}$$
$$\cup \{(\top, a, \top) \mid a \in \Sigma_1\}$$

$$\Delta^M = \Big\{ \big((q_1, q_2), a, r\big) \mid r = \begin{cases} (q_1', q_2') & \text{if } (q_1, a, q_1') \in \Delta_1^M \text{ and } (q_2, a, q_2') \in \Delta_2^m \setminus \Delta_2^M \\ (q_1', q_2) & \text{if } (q_1, a, q_1') \in \Delta_1^M \text{ and } a \notin \Sigma_2 \end{cases} \Big\}$$

Let $\mathcal{S}_\perp \div \mathcal{S} = \mathcal{S}_\perp$ when $\mathcal{S} \neq \mathcal{S}_\perp$ and $\mathcal{S} \div \mathcal{S}_\perp = \mathcal{S}_\top$.

The resulting possible transitions are enumerated in Table 4.3. Intuitively, for a given pair of states, the modality of each action is defined as the weakest modality such that the conjunction with $\mathcal{S}_2$ refines $\mathcal{S}_1$. Observe that if $\Sigma_1 \cap \Sigma_2 = \emptyset$ then $\mathcal{S}_1 \div \mathcal{S}_2 = \mathcal{S}_1$.

Proposition 4.1 *Given three modal specifications $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}$, the following implication holds:*

$$\mathcal{S}_2 \preceq \mathcal{S} \div \mathcal{S}_1 \implies \mathcal{S}_1 \wedge \mathcal{S}_2 \preceq \mathcal{S}.$$

The converse does not hold in general. Consider the modal specifications in Fig. 4.3 defined over the same alphabet $\{a, b\}$, we have $\mathcal{S}_1 \wedge \mathcal{S}_2 \preceq \mathcal{S}$ but $\mathcal{S}_2 \not\preceq \mathcal{S} \div \mathcal{S}_1$ as after a first occurrence of a, the action a is required in $\mathcal{S} \div \mathcal{S}_1$ whereas, in $\mathcal{S}_2$, a is forbidden.

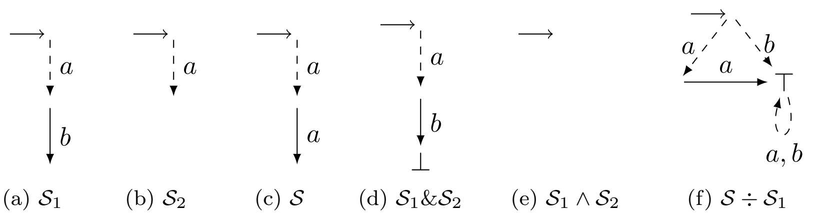

a
b
(a) $\mathcal{S}_1$
a
(b) $\mathcal{S}_2$
a
a
(c) $\mathcal{S}$
a
b
$\perp$
(d) $\mathcal{S}_1 \& \mathcal{S}_2$
(e) $\mathcal{S}_1 \wedge \mathcal{S}_2$
a
b
a
$\top$
a, b
(f) $\mathcal{S} \div \mathcal{S}_1$

Figure 4.3: Counterexample showing the incompleteness of weak implication $\div$

We now define a relation between modal specifications called *non-conflicting* under which completeness of weak implication is ensured. Intuitively two modal specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ are *non-conflicting* if any shared action required by one is not forbidden by the other. In [192] this relation is called *independence*.

Definition 4.10 (Non-conflicting) *Two modal specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ are* non-conflicting *if there exists a relation $\Gamma \subseteq Q_1 \times Q_2$ such that $(q_1^0, q_2^0) \in \Gamma$ and for all pairs $(q_1, q_2) \in \Gamma$:*

- *For all $a \in \Sigma_1 \cap \Sigma_2$:*

   – *for every* $(q_1, a, q_1') \in \Delta_1^M$ *there exists* $(q_2, a, q_2') \in \Delta_2^m$ *with* $(q_1', q_2') \in \Gamma$*;*
   – *for every* $(q_2, a, q_2') \in \Delta_2^M$ *there exists* $(q_1, a, q_1') \in \Delta_1^m$ *with* $(q_1', q_2') \in \Gamma$*;*
   – *if* $(q_1, a, q_1') \in \Delta_1^m$ *and* $(q_2, a, q_2') \in \Delta_2^m$ *then* $(q_1', q_2') \in \Gamma$*.*

- *for all* $a \in \Sigma_1 \setminus \Sigma_2$*, if* $(q_1, a, q_1') \in \Delta_1^m$ *then* $(q_1', q_2) \in \Gamma$*;*
- *for all* $a \in \Sigma_2 \setminus \Sigma_1$*, if* $(q_2, a, q_2') \in \Delta_2^m$ *then* $(q_1, q_2') \in \Gamma$*.*

Example 4.2 *The modal specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ in Fig. 4.3 are not non-conflicting as, after the occurrence of a, the action b is required in $\mathcal{S}_1$ and forbidden in $\mathcal{S}_2$.*

If $\mathcal{S}_1$ and $\mathcal{S}_2$ are non-conflicting then the computation of the greatest lower bound $\mathcal{S}_1 \wedge \mathcal{S}_2$ does not produce inconsistency, as the rule $q_1 \xrightarrow{a} q_1'$ and $q_2 \overset{a}{\nrightarrow}$ in Table 4.1, which entails that (q_1, q_2) is inconsistent, is never applied.

Proposition 4.2 *Given three modal specifications $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}$ such that $\mathcal{S}_1$ and $\mathcal{S}_2$ are non-conflicting:*

$$\mathcal{S}_1 \wedge \mathcal{S}_2 \preceq \mathcal{S} \implies \mathcal{S}_2 \preceq \mathcal{S} \div \mathcal{S}_1.$$

Weak implication is called a *partial* adjoint of conjunction as it is correct (Prop. 4.1) but complete only under a certain assumption (Prop. 4.2). A correct and complete construction which would then be an adjoint of conjunction does not exist for modal specifications.

Theorem 4.1 *Conjunction for modal specification does not have an adjoint [148].*

Parallel Composition

We now generalize parallel composition of Bɪᴘ component behaviors (Definition 2.28) to modal transition systems.

Definition 4.11 (Composition $\|_{IM}$) *The composition of two modal specifications $\mathcal{S}_i = \langle Q_i, q_i^0, \Sigma_i, \Delta_i^m, \Delta_i^M \rangle$, $i = 1, 2$ with disjoint alphabets, under an interaction model IM over $\Sigma_1 \uplus \Sigma_2$ is $\mathcal{S}_1 \|_{IM} \mathcal{S}_2 = \langle Q_1 \times Q_2, (q_1^0, q_2^0), IM, \Delta^m, \Delta^M \rangle$, where the transition relations Δ^m and Δ^M are obtained by synchronizing respectively Δ_i^m and Δ_i^M: for any $\alpha \in IM$,*

- $((q_1, q_2), \alpha, (q_1', q_2')) \in \Delta^m$ *if* $(q_i, \alpha[i], q_i') \in \Delta_i^m$ *when* $\alpha[i] \neq \emptyset$*, and* $q_i = q_i'$ *otherwise,* $i = 1, 2$*;*
- $((q_1, q_2), \alpha, (q_1', q_2')) \in \Delta^M$ *if* $(q_i, \alpha[i], q_i') \in \Delta_i^M$ *when* $\alpha[i] \neq \emptyset$*, and* $q_i = q_i'$ *otherwise,* $i = 1, 2$*.*

Let $\mathcal{S} \|_{IM} \mathcal{S}_\perp = \mathcal{S}_\perp \|_{IM} \mathcal{S} = \mathcal{S}_\perp$*.*

Example 4.3 *Fig. 4.4 shows the modal specifications $\mathcal{S}_1$ and $\mathcal{S}_2$, defined over the alphabets $\{a, b\}$ and $\{c, d\}$, respectively, and their composition $\mathcal{S}_1 \|_{IM} \mathcal{S}_2$ under the interaction model $IM = \{b, a|c, b|d\}$.*

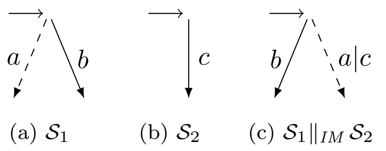


Figure 4.4: Example of composition $\|_{IM}$

$\olessthan$	$q_2 \overset{a}{\rightarrow}_2 q_2'$	$q_2 \overset{a}{\dashrightarrow}_2 q_2'$	$q_2 \overset{a}{\not\rightarrow}_2$	$a \notin \Sigma_2$
$q_1 \overset{a}{\rightarrow}_1 q_1'$	$(q_1, q_2) \overset{a}{\rightarrow} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\rightarrow} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\not\rightarrow}$	$(q_1, q_2) \overset{a}{\rightarrow} (q_1', q_2)$
$q_1 \overset{a}{\dashrightarrow}_1 q_1'$	$(q_1, q_2) \overset{a}{\rightarrow} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1', q_2')$	$(q_1, q_2) \overset{a}{\not\rightarrow}$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1', q_2)$
$q_1 \overset{a}{\not\rightarrow}_1$	$(q_1, q_2) \overset{a}{\rightarrow} q_2'$	$(q_1, q_2) \overset{a}{\not\rightarrow}$	$(q_1, q_2) \overset{a}{\not\rightarrow}$	$(q_1, q_2) \overset{a}{\not\rightarrow}$
$a \notin \Sigma_1$	$(q_1, q_2) \overset{a}{\rightarrow} (q_1, q_2')$	$(q_1, q_2) \overset{a}{\dashrightarrow} (q_1, q_2')$	$(q_1, q_2) \overset{a}{\not\rightarrow}$	N/A

Table 4.4: Transition relation of $\mathcal{S}_1 \olessthan \mathcal{S}_2$

Refinement is a congruence with respect to composition (this property is also called *stepwise refinement* of modal specifications):

Proposition 4.3 *For modal specifications $\mathcal{S}$ over Σ, $\mathcal{S}_1, \mathcal{S}_2$ over Σ', and IM over $\Sigma \uplus \Sigma'$,*

$$\mathcal{S}_1 \preceq \mathcal{S}_2 \implies \mathcal{S}_1 \|_{IM} \mathcal{S} \preceq \mathcal{S}_2 \|_{IM} \mathcal{S}$$

Priority Composition

Last, we introduce a priority composition. Intuitively, $\mathcal{S}_1 \olessthan \mathcal{S}_2$ behaves, for each action, like $\mathcal{S}_2$ whenever the modality in $\mathcal{S}_2$ is different from *may* (undetermined), otherwise it behaves like $\mathcal{S}_1$.

Definition 4.12 (Priority composition) *Let $\mathcal{S}_i = \langle Q_i, q_i^0, \Sigma_i, \Delta_i^m, \Delta_i^M \rangle$, $i = 1, 2$ be two modal specifications, $\mathcal{S}_1 \olessthan \mathcal{S}_2$ is the tuple $\langle (Q_1 \times Q_2) \cup Q_2, (q_1^0, q_2^0), \Sigma_1 \cup \Sigma_2, \Delta^m, \Delta^M \rangle$ with $\Delta^m = \Delta_2^m \cup (\Delta^m)'$, $\Delta^M = \Delta_2^M \cup (\Delta^M)'$, and $(\Delta^m)'$ and $(\Delta^M)'$ are defined by Table 4.4.*

Proposition 4.4 *Given three modal specifications $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$:*

- $\mathcal{S}_1 \olessthan \mathcal{S}_2 \preceq \mathcal{S}_2$;
- $(\mathcal{S}_1 \olessthan \mathcal{S}_2) \olessthan \mathcal{S}_3 = \mathcal{S}_1 \olessthan (\mathcal{S}_2 \olessthan \mathcal{S}_3)$.

4.1.3 Modal Assume/Guarantee Contracts

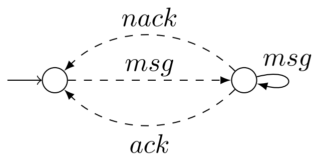
We now enrich our component-based framework with a notion of contracts. As briefly indicated in the introduction, a contract is a pair of specifications: one describes an assumption on the usage of the component made by its environment; the other one corresponds to a guarantee offered by the component as long as the assumption is satisfied.

Definition 4.13 (Contract) *A* contract *$\mathcal{C}$ is a pair $(\mathcal{A}, \mathcal{G})$ of modal specifications where $\mathcal{A}$ is called* assumption *and $\mathcal{G}$* guarantee*.*

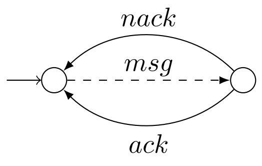
We do not make restriction on the alphabet of $\mathcal{A}$ and $\mathcal{G}$; their scope can thus be different. A contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ on a component with alphabet Σ guarantees any implementation to satisfy $\mathcal{G}$, provided that $\mathcal{A}$ is satisfied. Since the context in which the component will be deployed is unknown at design time, $\mathcal{A}$ can only make assumptions about the locally observable behavior of the component when integrated in its environment, not about the behavior of the environment itself.

Example 4.4 *The contract $(\mathcal{A}_{Fun}, \mathcal{G}_{Fun})$ consisting of the pair of modal specifications in Fig. 4.5 specifies that every message sent must be acknowledged, provided that a sent message is re-emitted as long as it has not been acknowledged.*

The semantics of contracts is defined as follows.



(a) Assumption $\mathcal{A}_{Fun}$



(b) Guarantee $\mathcal{G}_{Fun}$

Figure 4.5: Communication channel – functional aspect $\mathcal{C}_{Fun}$

Definition 4.14 (Implementation) *A modal specification $\mathcal{S}$ satisfies a contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ if $\mathcal{A} \wedge \mathcal{S} \preceq \mathcal{G}$. Models $\mathcal{B}$ of $\mathcal{S}$ are then called implementations of $\mathcal{C}$, written $\mathcal{B} \models \mathcal{C}$.*

A contract represents early requirements that may be strengthened during design time. We thus introduce a refinement relation on contracts.

Refinement

Refining a contract means weakening assumptions and strengthening guarantees when the initial assumption is met:

Definition 4.15 (Refinement) *A contract $(\mathcal{A}', \mathcal{G}')$ refines $(\mathcal{A}, \mathcal{G})$, written $(\mathcal{A}', \mathcal{G}') \preceq (\mathcal{A}, \mathcal{G})$ if $\mathcal{A} \preceq \mathcal{A}'$ and $\mathcal{A} \wedge \mathcal{G}' \preceq \mathcal{G}$.*

We denote $\equiv$ the equivalence relation induced by $\preceq$. Contracts can also be ordered by comparing their sets of implementations:

Definition 4.16 (Model inclusion) *We write $(\mathcal{A}', \mathcal{G}') \sqsubseteq (\mathcal{A}, \mathcal{G})$ if every implementation of $(\mathcal{A}', \mathcal{G}')$ is also an implementation of $(\mathcal{A}, \mathcal{G})$. We write $(\mathcal{A}', \mathcal{G}') \equiv (\mathcal{A}, \mathcal{G})$ if $(\mathcal{A}', \mathcal{G}') \sqsubseteq (\mathcal{A}, \mathcal{G})$ and $(\mathcal{A}, \mathcal{G}) \sqsubseteq (\mathcal{A}', \mathcal{G}')$.*

Proposition 4.5 $(\mathcal{A}', \mathcal{G}') \preceq (\mathcal{A}, \mathcal{G}) \implies (\mathcal{A}', \mathcal{G}') \sqsubseteq (\mathcal{A}, \mathcal{G})$. *The converse is not true.*

Let us distinguish the two following particular contracts: $\mathcal{C}_\perp = (\mathcal{S}_\top, \mathcal{S}_\perp)$ and $\mathcal{C}_\top = (\mathcal{S}_\perp, \mathcal{S}_\top)$. For any contract $\mathcal{C}$ we have $\mathcal{C}_\perp \preceq \mathcal{C} \preceq \mathcal{C}_\top$. Moreover, $\mathcal{C}_\perp$ has no implementation and $\mathcal{C}_\top$ admits any behavior as implementation.

Implicit Form

A fundamental question following Definition 4.14 is then: can we compute a modal specification having the same set of implementations as a given contract $\mathcal{C}$? This question is addressed by the *implicit form* of $\mathcal{C}$. For a given contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$, the weak implication $\mathcal{G} \div \mathcal{A}$ is called the *implicit form* of $\mathcal{C}$. It is refined by all specification $\mathcal{S}$ satisfying $\mathcal{C}$ and such that $\mathcal{S}$ and $\mathcal{A}$ are non-conflicting.

Example 4.5 *The implicit form of the contract $\mathcal{C}_{Fun}$ in Fig. 4.5 is depicted in Fig. 4.6. While* ack *and* nack *are forbidden in the initial state of the assumption $\mathcal{A}_{Fun}$, the implicit form of $\mathcal{C}_{Fun}$ has a may-transition from the initial state to a $\top$-state labelled by* ack *and* nack*. This represents the consequence of the violation of the assumption: if* ack *or* nack *occur then the contract is relaxed and no more guarantee is provided.*

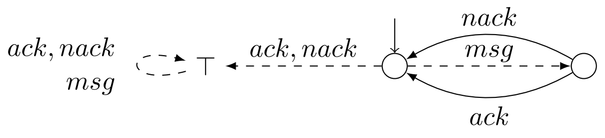


Figure 4.6: Implicit form of the contract $\mathcal{C}_{Fun}$

4.1.4 Composition of Contracts

Composition operations for contracts may be used with different goals in a design flow, and hence, be given different semantics: on the one hand, a contract may describe the guarantees a component is able to give, depending on its environment. Accordingly, the *component composition* of two contracts is the *strongest* contract satisfied by the composition of any pair of implementations of both contracts.

On the other hand, a contract may be used to specify a *requirement* as a guarantee that must be ensured under some hypothesis. We call this a requirement contract, or *aspect*. We define two operations for *aspect composition* of two contracts: one with a conjunctive semantics, defined as the *weakest* common refinement of both contracts, and *priority composition*, allowing a contract of higher priority to override a lower-priority contract in case of conflict.

Component Composition

For disjoint component contracts we define a "best effort" composition operation that is parametrized by a Bip interaction model. This composition ensures each guarantee depending on the satisfaction of its assumption, provided that the guarantee is feasible under the specified interaction model.

Consider two behaviors $\mathcal{B}_1$ and $\mathcal{B}_2$ that are respectively implementations of contracts $\mathcal{C}_1$ and $\mathcal{C}_2$. When a system is built bottom-up by composing $\mathcal{B}_1$ and $\mathcal{B}_2$ under an interaction model *IM*, one obvious question is: what can be inferred on $\mathcal{B}_1 \|_{IM} \mathcal{B}_2$ from $\mathcal{C}_1$ and $\mathcal{C}_2$? To this end, we define the composition operation $\otimes_{IM}$ on contracts.

Definition 4.17 (Composition $\otimes_{IM}$) *Given contracts $\mathcal{C}_1 = (\mathcal{A}_1, \mathcal{G}_1)$ and $\mathcal{C}_2 = (\mathcal{A}_2, \mathcal{G}_2)$ on disjoint alphabets and an interaction model IM, we define $\mathcal{C}_1 \otimes_{IM} \mathcal{C}_2 = (\mathcal{A}, \mathcal{G})$ where:*

- $\mathcal{G} = (\mathcal{G}_1 \div \mathcal{A}_1) \|_{IM} (\mathcal{G}_2 \div \mathcal{A}_2)$
- $\mathcal{A} = (\mathcal{A}_1 \|_{IM} \mathcal{A}_2) \div \mathcal{G}$

The composition $\otimes_{IM}$ is commutative and associative. The property of *independent implementability* (also referred to as *constructivity* in [165]) allows us to obtain an implementation of the composition of contracts as the composition of their implementations.

Theorem 4.2 (Independent implementability) *Let $\mathcal{C}_1, \mathcal{C}_2$ be two contracts and IM an interaction model, if $\mathcal{B}_1 \models \mathcal{C}_1$ and $\mathcal{B}_2 \models \mathcal{C}_2$ then $\mathcal{B}_1 \|_{IM} \mathcal{B}_2 \models \mathcal{C}_1 \otimes_{IM} \mathcal{C}_2$.*

Since the contract $\mathcal{C}_1 \otimes_{IM} \mathcal{C}_2$ is satisfied by the composition under *IM* of any pair of implementations of $\mathcal{C}_1$ and $\mathcal{C}_2$, the composition operation $\otimes_{IM}$ allows us to reason about contracts in a bottom-up manner. Similarly, for contracts the composition $\otimes_{IM}$ is monotonic with respect to the refinement preorder:

Proposition 4.6 (Stepwise refinement) *Given three contracts $\mathcal{C}$, $\mathcal{C}_1$, $\mathcal{C}_2$, the following holds:*

$$\mathcal{C}_1 \preceq \mathcal{C}_2 \implies \mathcal{C}_1 \otimes_{IM} \mathcal{C} \preceq \mathcal{C}_2 \otimes_{IM} \mathcal{C}$$

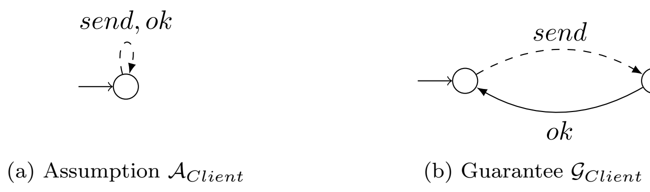

send, ok
send
ok
(a) Assumption $\mathcal{A}_{Client}$
(b) Guarantee $\mathcal{G}_{Client}$

Figure 4.7: A contract $\mathcal{C}_{Client}$ for a client component

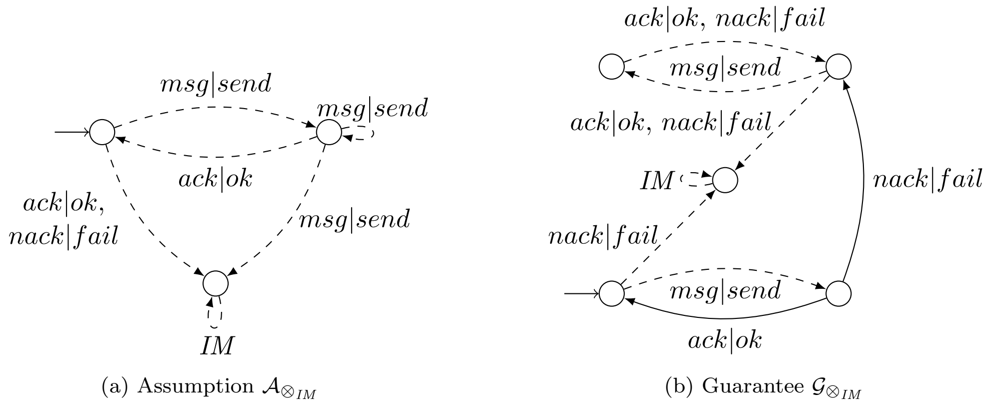


Figure 4.8: The contract $\mathcal{C}_{Fun} \otimes_{IM} \mathcal{C}_{Client}$

One may argue that there are other ways to compose $\mathcal{C}_1$ and $\mathcal{C}_2$ such that the independent implementability property is satisfied. We however now show that $\mathcal{C}_1 \otimes_{IM} \mathcal{C}_2$ is the minimal contract with respect to the model inclusion preorder satisfying the independent implementability property:

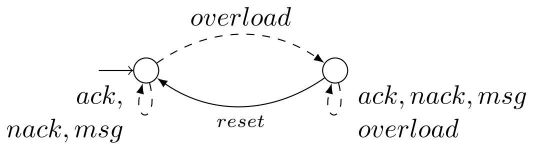
Theorem 4.3 (Minimality) *If for all $\mathcal{B}_1$ and $\mathcal{B}_2$ such that $\mathcal{B}_1 \models \mathcal{C}_1$ and $\mathcal{B}_2 \models \mathcal{C}_2$ we have $\mathcal{B}_1 \|_{IM} \mathcal{B}_2 \models \mathcal{C}$, then $\mathcal{C}_1 \otimes_{IM} \mathcal{C}_2 \sqsubseteq \mathcal{C}$.*

Example 4.6 *Consider a client whose alphabet consists of the three actions:* send *for a message to be transmitted, and two kinds of responses:* ok *if the message has been received, and* fail *otherwise. The contract in Fig. 4.7 states that, under the hypothesis that* fail *never occurs, every transmitted message is well received (i.e.,* send *is acknowledged by* ok*). The interactions between the client and the communication channel are given by the interaction model $IM = \{msg|send, ack|ok, nack|fail\}$. The result of composing $\mathcal{C}_{Fun}$ of Fig. 4.5 and $\mathcal{C}_{Client}$ of Fig. 4.7 is depicted in Fig. 4.8. In the obtained guarantee,* msg|send *is followed by* ack|ok *unless* nack|fail *occurs, which constitutes a violation of $\mathcal{A}_{Client}$. From then on,* msg|send *may be followed by* ack|ok *or* nack|fail.

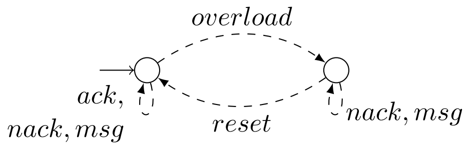
By independent implementability, the composition of any pair of implementations of $\mathcal{C}_{Fun}$ and $\mathcal{C}_{Client}$ satisfies $\mathcal{C}_{Fun} \otimes_{IM} \mathcal{C}_{Client}$. This may look surprising, since the assumption $\mathcal{A}_{\otimes_{IM}}$ allows ack|ok *to take place from the initial state, whereas* ack *was not allowed to happen from the initial state of $\mathcal{A}_{Fun}$. This is because this behavior is ruled out by $\mathcal{C}_{Client}$, which guarantees only* send *to take place. In other words, the information of $\mathcal{C}_{Client}$ is used to weaken the assumption $\mathcal{A}_{\otimes_{IM}}$.*

Aspect Composition

It is current engineering practice to model different aspects of a specification separately. In terms of contracts, this amounts to attach several contracts to a single component. A



(a) Assumption $\mathcal{A}_{Rel}$



(b) Guarantee $\mathcal{G}_{Rel}$

Figure 4.9: Communication channel – reliability aspect $\mathcal{C}_{Rel}$

central question is whether a set of contracts is consistent, and how to compute a common implementation, or *shared refinement* [107]. We define the composition of contracts $\mathcal{C}_1$ and $\mathcal{C}_2$ as the *weakest* contract C refining each aspect, that is, making the guarantee of each contract provided that the corresponding assumption is met.

Definition 4.18 (Composition $\oplus$) *Given two contracts $\mathcal{C}_1 = (\mathcal{A}_1, \mathcal{G}_1)$ and $\mathcal{C}_2 = (\mathcal{A}_2, \mathcal{G}_2)$, let*

$$\mathcal{C}_1 \oplus \mathcal{C}_2 = \big(\mathcal{A}_1 \vee \mathcal{A}_2,\ (\mathcal{G}_1 \div \mathcal{A}_1) \wedge (\mathcal{G}_2 \div \mathcal{A}_2)\big)$$

This notion of composition is consistent with the component paradigm mentioned above, enabling the component to offer different guarantees depending on the context. The composition $\oplus$ is commutative and, for contracts sharing the same alphabet, associative.

Proposition 4.7 (Stepwise refinement) *For any contracts $\mathcal{C}$, $\mathcal{C}_1$, and $\mathcal{C}_2$,*

$$\mathcal{C}_1 \preceq \mathcal{C}_2 \implies \mathcal{C}_1 \oplus \mathcal{C} \preceq \mathcal{C}_2 \oplus \mathcal{C}$$

Example 4.7 *Consider now a second contract $\mathcal{C}_{Rel}$ in Fig. 4.9 dealing with the reliability of a communication channel. Two new actions are introduced:* overload *which occurs when the maximal capacity of the communication channel is reached and* reset *for the re-initialization of the system. The contract $\mathcal{C}_{Rel}$ specifies the following:*

- *assumption $\mathcal{A}_{Rel}$: on* overload*, the system must be reset.*
- *guarantee $\mathcal{G}_{Rel}$: on* overload*, the communication channel only produces* nack*; or, equivalently, messages can be positively acknowledged only when the system is not overloaded.*

The guarantee obtained by composing the two aspects $\mathcal{C}_{Fun}$ and $\mathcal{C}_{Rel}$ is depicted in Fig. 4.10. It can be decomposed in four blocks:

- *the upper left block corresponds to the situation where none of the assumptions made in the two contracts is violated and thus, both guarantees are verified;*
- *the upper right block corresponds to the case where the assumptions of $\mathcal{C}_{Fun}$ has been violated. As a consequence, the contract $\mathcal{C}_{Fun}$ is disabled and only $\mathcal{G}_{Rel}$ is ensured;*
- *similarly, in the bottom left block, the assumption of $\mathcal{C}_{Rel}$ have been violated and only $\mathcal{G}_{Fun}$ is ensured;*

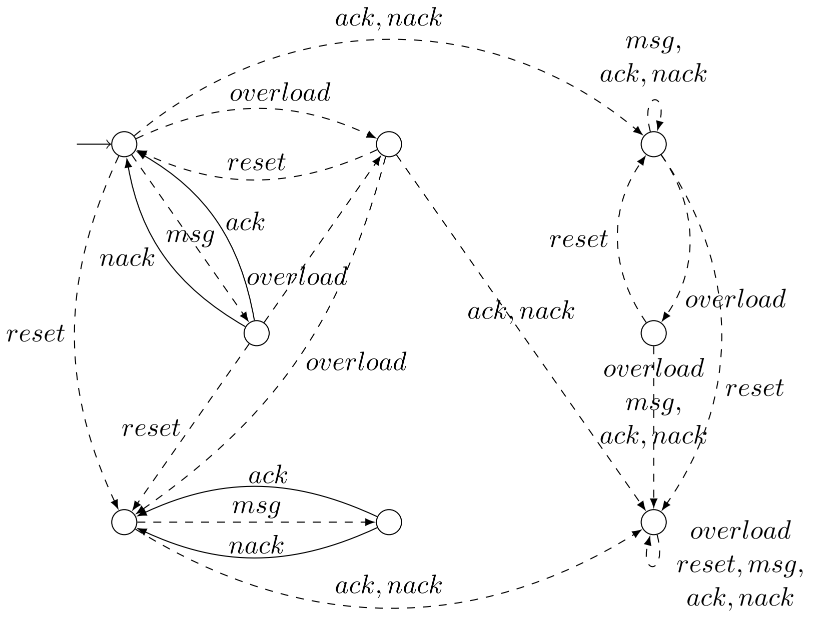


Figure 4.10: The guarantee of $\mathcal{C}_{Fun} \oplus \mathcal{C}_{Rel}$.

- *last, the bottom right block is a $\top$-state corresponding to the situation where both assumptions of $\mathcal{C}_{Fun}$ and $\mathcal{C}_{Rel}$ have been violated and both contracts are disabled.*

Theorem 4.4 (Correctness of $\oplus$) *The aspect composition operation $\oplus$ is correct with respect to model inclusion: for any contracts $\mathcal{C}_i$,*

$$\mathcal{C}_1 \oplus \mathcal{C}_2 \sqsubseteq \mathcal{C}_i, i = 1, 2.$$

Theorem 4.5 (Partial completeness of $\oplus$) *Consider two contracts $\mathcal{C}_1 = (\mathcal{A}_1, \mathcal{G}_1)$ and $\mathcal{C}_2 = (\mathcal{A}_2, \mathcal{G}_2)$. If $\mathcal{S}$ is a modal specification that satisfies $\mathcal{C}_1$ and $\mathcal{C}_2$ and is non-conflicting with $\mathcal{A}_1$ and $\mathcal{A}_2$, then $\mathcal{S}$ satisfies $\mathcal{C}_1 \oplus \mathcal{C}_2$.*

Because of the non-conflicting assumption here, we only talk about *partial* completeness. We conjecture that due to the non-existence of an adjoint to the greatest lower bound of modal specifications, aspect composition cannot be correct and complete; this is still an open question.

Prioritized Aspect Composition

In practice, aspects are not equally important. For instance, an aspect “safety” may be chosen to override an aspect “quality of service”. The operation of *priority* composes aspects in a hierarchical order, such that in case of inconsistency, an aspects of higher priority overrides a lower-priority contract. A similar operation for the four-valued Belnap logic has been introduced in [61] to compose access control policies.

Definition 4.19 (Priority $\oslash$ on contracts) *Given two contracts $\mathcal{C}_1 = (\mathcal{A}_1, \mathcal{G}_1)$ and $\mathcal{C}_2 = (\mathcal{A}_2, \mathcal{G}_2)$, we define:*

$$\mathcal{C}_1 \oslash \mathcal{C}_2 = \big(\ \mathcal{A}_1 \vee \mathcal{A}_2,\ (\mathcal{G}_1 \div \mathcal{A}_1) \oslash (\mathcal{G}_2 \div \mathcal{A}_2)\ \big)$$

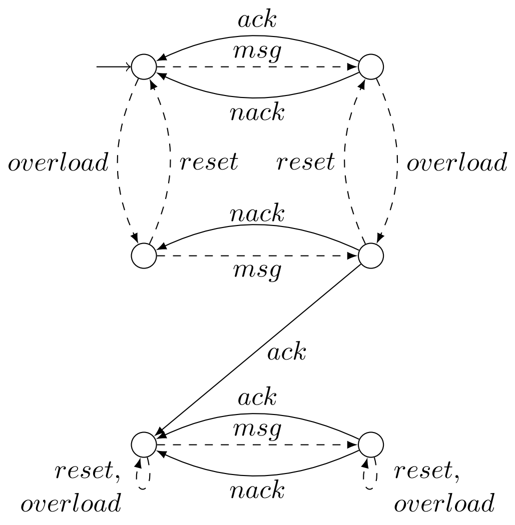


Figure 4.11: The guarantee of $\mathcal{C}_{Rel} \otimes \mathcal{C}_{Fun}$

Example 4.8 *The guarantee obtained by composing $\mathcal{C}_{Rel} \otimes \mathcal{C}_{Fun}$ is depicted in Fig. 4.11. When the communication channel is on overload, a positive acknowledgment* ack *may occur as specified in the higher-priority contract $\mathcal{C}_{Fun}$, which was impossible in $\mathcal{C}_{Rel}$. The remaining guarantee is $\mathcal{G}'_{Fun}$ (lower block).*

Proposition 4.8 *For any contract $\mathcal{C}$, $\mathcal{C}_1$ and $\mathcal{C}_2$, the priority composition operation on contracts satisfies the following properties:*

1. $\mathcal{C}_1 \otimes \mathcal{C}_2 \sqsubseteq \mathcal{C}_2$;
2. $\mathcal{C} \otimes (\mathcal{S}_\perp, \mathcal{S}_\top) = \mathcal{C}$.

That is, all models of the priority composition satisfy the contract of higher priority. Whenever the latter does not make a choice, the models must satisfy the choice made by $\mathcal{C}_1$.

For any contracts $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ over the same alphabet, the priority composition operation is associative. We can therefore extend priority composition to a hierarchy of an arbitrary number of aspects and compose them in any order.

4.1.5 Discussion

We have defined modal contracts as a pair of modal specifications, and introduced a new operation on modal specification called *weak implication.* Based on this operation, we have introduced three composition operations between modal contracts, responding to different requirements in the design flow and satisfying different properties:

- *component composition* $\otimes_{IM}$ of contracts over disjoint sub-systems, parametrized with an interaction model *IM.* The composition is defined as the *strongest* contract satisfying the property of independent implementability;
- *aspect composition* $\oplus$ of contracts over the same sub-system, defined as the *weakest* contract refining both arguments;
- *priority composition* $\otimes$ of contracts over the same sub-system, enforcing a hierarchy of importance among the contracts.

In contrast to modal interfaces [230] and our implicit form $G \div A$ using a distinct state $\top$ representing the fact that the assumption has been violated, modal assume/guarantee contracts explicitly distinguish assumption and guarantee. This has several practical consequences. First, in assume/guarantee contracts, assumption and guarantee can be refined separately, whereas in the implicit form, weakening an assumption coincides with a strengthening of the guarantee.

Second, assume/guarantee contracts enable the designer to make assumptions on some desired behavior, in contrast to the implicit form where assumptions (in the form of transitions leading to $\top$) can only express that some behavior is *not* expected.

Finally, and more importantly, it is frequent that assumption and guarantee cover different aspects of a design. In this case, assume/guarantee contracts enable separation of concerns and compositional reasoning about different views, whereas the complexity of the implicit form would blow up with the product of the size of the assumption and the guarantee.

As we have seen, these advantages of modal assume/guarantee contracts come at the price of aspect composition being only partially complete (Theorem 4.5).

Related work. A theory of assume/guarantee interfaces is presented in [183], along with acyclic assume/guarantee rules for computing guarantees of composed components.

In [190], assume/guarantee interfaces are seen as pairs of input-enabled I/O-automata. These component contracts are then composed by synchronized product where the assumption of the composed contract is defined as the weakest assumption ensuring the conjunction of both guarantees. The monolithic form of these contracts corresponds to interface automata [97], which jointly capture input assumptions and output guarantees, whereas the approach developed in this article can be mapped onto modal interfaces which are more expressive and more pertinent for interface-based design [193, 230].

More recently, a general framework defining a contract theory from a specification theory has been proposed in [29] and instantiated with the theory of modal specifications. However, aspect composition is not considered.

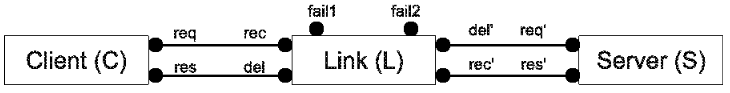
[75] develops a (non-modal) compositional specification theory including parallel composition, conjunction, and quotient operations, for both a declarative and an operational component theory where I/O actions synchronize. This work is extended in [79] to assume/guarantee specifications.

The approach developed in [37, 36] aims at being the most general possible and considers assume/guarantee contracts as pairs of specifications, without fixing their formalism *a priori*. A general composition operation called *fusion* is then defined that separately composes assumptions and guarantees. This generality entails the strong restriction of the specification formalism and the guarantees being closed under $\wedge$, $\vee$, and negation.

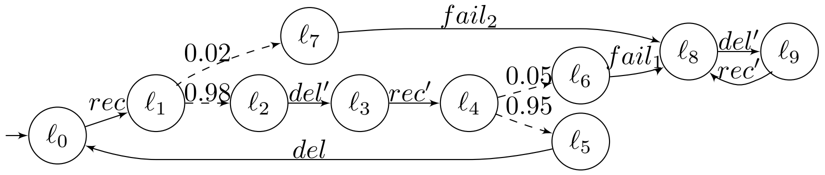
4.2 Probabilistic Contracts

In this section we present the framework of probabilistic contracts introduced in [269, 155] for constructing component-based embedded systems, based on the formalism of discrete-time Interactive Markov Chains. A contract specifies the assumptions a component makes on its context and the guarantees it provides. Probabilistic transitions represent allowed uncertainty in the component behavior, for instance, to model internal choice or reliability. Action transitions are used to model non-deterministic behavior and communication between components. An interaction model specifies how components interact with each other.

We provide the ingredients for a component-based design flow, including (1) contract satisfaction and refinement, (2) parallel composition of contracts over disjoint, interacting



(a) Client – Link – Server.



(b) The IMC M_ℓ of the Link.

Figure 4.12: An example of IMC: a Client-Link-Server.

components, and (3) conjunction of contracts describing different requirements over the same component. Compositional design is enabled by congruence of refinement.

4.2.1 Motivation

Embedded and distributed systems often encompass unreliable software or hardware components, as it may be technically or economically impossible to make a system entirely reliable. As a result, system designers have to deal with probabilistic specifications such as "the probability that this component fails at this point of its behavior is less than or equal to 10^{-6}". More generally, uncertainty in the observed behavior is introduced by abstraction of black-box behavior of components, the environment, or the execution platform. In this work, we introduce a framework for the design of correct systems from probabilistic, interacting components.

Figure 4.12(a) shows a Link system that transmits data between a Client and a Server. The Link receives a request from the Client and encodes the request before sending it to the Server. The encoding process fails with probability 0.02. After receiving a response from the Server, it decodes the data before delivering it to the Client. To model components, we use a variant of Interactive Markov Chain (IMC) framework [167] with discrete time semantics, which combines labeled transition systems (LTS) and Markov chains. Figure 4.12(b) shows an IMC describing the Link component of Figure 4.12(a). From its initial state ℓ_0, the Link goes to state ℓ_1 as soon as it receives (rec) a request from a Client; the probability that it delivers (del') this request to the Server is 0.98 and the probability that it fails to deliver it to the Server is 0.02. The Link goes to state ℓ_4 immediately after receiving a response (rec') from the Server; the probability that it delivers (del) the response to the Client is 0.95 and the probability of failing to do so is 0.05. In state ℓ_8, the Link may still communicate with the Server regarding other services, but will not deliver any response to the Client.

Components communicate through interactions, that is, synchronized action transitions. Interactions are essential in component frameworks because they allow the modeling of how components cooperate and communicate. We use BIP to model interactions between components. Since the deploying context of a component is not known at design time, we use *probabilistic contracts* to specify and reason about the correct behaviors of a component. Contracts were first introduced in [210]. They allow the designer to specify what a component can expect from its context, what it must guarantee, and explicitly limit the responsibilities of both.

The framework we propose here allows us to model components, their interactions, and the uncertainty in their observed behavior (§4.2.2). It supports different steps in a design flow: refinement, satisfaction, and projection (§4.2.3), parallel composition (§4.2.4), and conjunction (shared refinement) (§4.2.4). These operations satisfy the following properties:

- Refinement is compositional, that is, the parallel composition of refining contracts refines the composition of the refined contracts.
- Several contracts over the same component may be used to independently specify different requirements, possibly over different subsets of the component interactions. The conjunction is a common refinement of all contracts. It is the greatest common refinement if all contracts have the same alphabet.

As pointed out in [63], conjunction of probabilistic specifications is non trivial, since a straight-forward approach would introduce spurious behaviors.

4.2.2 Components and Contracts

We use Interactive Markov Chains [167] with discrete-time semantics to model the behavior of components.

Definition 4.20 (Probability distribution) *A* probability distribution *over a finite set X is a function $f : X \to [0,1]$ such that $\sum_{x\in X} f(x) = 1$.*

Definition 4.21 (Interactive Markov Chain (IMC)) *An IMC is a tuple $(\mathcal{Q}, \mathcal{A}, \to, \Delta, s_0)$ where:*

- *$\mathcal{Q}$ is a nonempty finite set of states, partitioned into $\mathcal{Q}^p$, the set of probabilistic states, and $\mathcal{Q}^a$, the set of action states;*
- *A is a finite alphabet of actions;*
- *$\to \subseteq \mathcal{Q}^a \times \mathcal{A} \times \mathcal{Q}$ is an action transition relation;*
- *$\Delta : \mathcal{Q}^p \to (\mathcal{Q} \to [0,1])$ is a transition probability function such that, for each $s \in \mathcal{Q}^p$, $\Delta(s)$ is a probability distribution over $\mathcal{Q}$;*
- *s_0 is the initial state.*

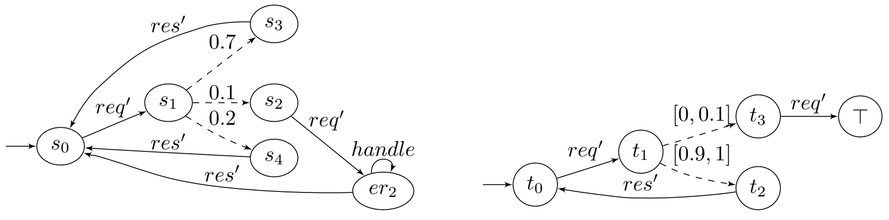
Each action state in $\mathcal{Q}^a$ may have outgoing action transitions — also called *non-deterministic transitions* in the literature — like those in a labeled transition system (LTS). Each probabilistic state in $\mathcal{Q}^p$ has outgoing probabilistic transitions like those in a Markov chain. Probability distributions on states are memoryless, i.e., the future of an IMC depends only on the current state, not on past choices. For example, in Figure 4.12(b), the probabilistic choice that the Link delivers the response to the Client (i.e., $\Delta(\ell_4)(\ell_5) = 0.95$) is independent from the probabilistic choice of delivering a request to the Server (i.e., $\Delta(\ell_1)(\ell_2) = 0.98$).

Notation: For convenience, we sometimes write the transition probability function Δ as a transition relation $\dashrightarrow \subseteq \mathcal{Q}^p \times [0,1] \times \mathcal{Q}$ such that:

$$\dashrightarrow = \{(s, p, s') \mid s \in \mathcal{Q}^p \wedge s' \in \mathcal{Q} \wedge p = \Delta(s)(s')\}$$

Graphically, we only depict the $\dashrightarrow$ transitions labeled with a non null probability (see Figure 4.13(a)).

We introduce *contracts* as a finite specification for a possibly infinite number of components modeled by IMCs. In contrast to IMCs, the probabilistic transitions of a contract are labeled with probability *intervals*, similar to the formalism of [173, 270]. Moreover, two distinct states $\top$ and $\bot$ are used to distinguish the assumptions on the use of the component from the guarantees it provides.



(a) IMC M_s for Server (b) Contract C_s for Server

Figure 4.13: Contract Examples

Definition 4.22 (Contract) *A contract is a tuple* $(\mathcal{Q}, \mathcal{A}, \rightarrow, \sigma, t_0)$ *where:*

- $\mathcal{Q}$ *is a nonempty finite set of states, partitioned into* $\mathcal{Q} = \mathcal{Q}^p \cup \mathcal{Q}^a \cup \{\top, \bot\}$*, where* $\mathcal{Q}^p$ *is the set of probabilistic states,* $\mathcal{Q}^a$ *is the set of action states, and* $\top$ *and* $\bot$ *are distinct states without any outgoing transitions;*
- A *is a finite alphabet of actions;*
- $\rightarrow \subseteq \mathcal{Q}^a \times \mathcal{A} \times \mathcal{Q}$ *is the action transition relation;*
- $\sigma : \mathcal{Q}^p \rightarrow (\mathcal{Q} \rightarrow 2^{[0,1]})$ *is a transition probability predicate, associating with each pair of states in* $\mathcal{Q}^p \times \mathcal{Q}$ *an interval of probabilities;*
- t_0 *is the initial state.*

Definitions: We also write σ as a transition relation $\dashrightarrow \subseteq \mathcal{Q}^p \times 2^{[0,1]} \times \mathcal{Q}$ such that $\dashrightarrow = \{(s, P, s') \mid s \in \mathcal{Q}^p \wedge s' \in \mathcal{Q} \wedge P = \sigma(s)(s')\}$. We write $q \overset{>0}{\dashrightarrow} q'$ if $\exists p > 0 : p \in \sigma(q, q')$ and denote by $\overset{>0}{\dashrightarrow}^{+}$ the transitive closure of $\overset{>0}{\dashrightarrow}$. Let $\rightsquigarrow = \rightarrow \cup \overset{>0}{\dashrightarrow}$, and let $\rightsquigarrow^*$ be the reflexive and transitive closure of $\rightsquigarrow$. A state $q \in \mathcal{Q}$ is *reachable* if $t_0 \rightsquigarrow^* q$. A contract is *consistent* if $\bot$ is not reachable.

The meaning of a contract C over a component M is the following:

- a transition $s \xrightarrow{a} \top$ specifies the *assumption* of the component M that an interaction involving action a does not occur in state s;
- in an action state s, an action a labeling a transition not leading to $\top$ specifies the *guarantee* of the component M that a is enabled in s; conversely, the absence of any outgoing transition labeled with a specifies the *guarantee* that an interaction involving a will not occur;
- the $\top$ state represents the fact that the *assumption* has been violated, and henceforth, the component M can show arbitrary, uncontrollable behavior;
- the $\bot$ state stands for "inconsistent" and means that M cannot satisfy the contract C any more;
- a transition $s \overset{[a,b]}{\dashrightarrow} t$ specifies an interval of allowed transition probabilities.

Hypothesis 4.1 *We require that the target states of probabilistic transitions are action or probabilistic states: if* $q \overset{>0}{\dashrightarrow} q'$ *then* $q' \notin \{\top, \bot\}$.

Example 4.9 *The contract C_s in Figure 4.13(b) specifies that, after the Server receives a request req′, the probability that it reaches state t_3 is within* $[0, 0.1]$*; in state t_3, it* assumes *that the environment does not provide req′; if this occurs, its implementation is not bound by C_s any more; the probability that it reaches t_2 from t_1 is within* $[0.9, 1]$*; in state t_2, it* guarantees *to send a response (res′). In §4.2.3, we show how to check that the IMC M_s (in Figure 4.13(a)) satisfies the contract C_s.*

From the definitions of IMC and contract, we can see that an IMC can be trivially converted into a contract. For this, we define a lifting operator $\lfloor . \rfloor$:

$$\begin{aligned} \lfloor s_1 \xrightarrow{\alpha} s_2 \rfloor &= s_1 \xrightarrow{\alpha} s_2 \\ \lfloor s_1 \overset{p}{\dashrightarrow} s_2 \rfloor &= s_1 \overset{[p,p]}{\dashrightarrow} s_2 \end{aligned}$$

For the sake of simplicity, we use the same notation $\dashrightarrow$ to represent both kinds of probabilistic transitions (i.e., those in an IMC and in a contract).

In Figure 4.14 we define some useful operations related to probability intervals.

$$\begin{aligned} \lceil n \rceil &= \text{if } n > 1 \text{ then } 1 \text{ else } n & \\ [\ell_1, u_1] + [\ell_2, u_2] &= [\ell_1 + \ell_2, \lceil u_1 + u_2 \rceil] & \text{[F1]} \\ [\ell_1, u_1] * [\ell_2, u_2] &= [\ell_1 * \ell_2, u_1 * u_2] & \text{[F2]} \\ k * [\ell, u] &= [k * \ell, k * u] \text{ for } k \in [0, 1] & \text{[F3]} \end{aligned}$$

Figure 4.14: Operations on probability intervals.

The following definition, borrowed from [111], states that, for any probability chosen in any probabilistic transition's interval, it is always possible to choose probabilities in the intervals of all the remaining transitions outgoing from the same state such that the sum is 1.

Definition 4.23 (Delimited contract) *A contract $C = (\mathcal{Q}, \mathcal{A}, \rightarrow, \sigma, t_0)$ is delimited [111] iff $\forall s \in \mathcal{Q}^p \; \forall s' \in \mathcal{Q} \; \forall p \in \sigma(s)(s') : 1 - p \in \sum_{s'' \in \mathcal{Q} \setminus \{s'\}} \sigma(s)(s'')$.*

4.2.3 Contract Refinement

System synthesis involves refining a contract until an implementation is obtained. We therefore define formally the notion of contract refinement.

Refinement and Satisfaction

We first define contract refinement, and give thereafter some explanations.

Definition 4.24 (Contract refinement) *Let $C_1 = (\mathcal{Q}_1, \mathcal{A}, \rightarrow_1, \sigma_1, s_0)$ and $C_2 = (\mathcal{Q}_2, \mathcal{A}, \rightarrow_2, \sigma_2, t_0)$ be two contracts. A relation $\preceq \subseteq \mathcal{Q}_1 \times \mathcal{Q}_2$ is a* simulation *if for all $s \preceq t$ we have:*

1. $s = \top \implies t = \top$;
2. $t = \bot \implies s = \bot$;
3. *If $(s, t) \in \mathcal{Q}_1^a \times (\mathcal{Q}_2^a \cup \{\top\})$ then*
   - *(a)* $\forall t' \neq \top \in \mathcal{Q}_2, \; (t \xrightarrow{\alpha}_2 t') \implies (\exists s' \in \mathcal{Q}_1, \; s \xrightarrow{\alpha}_1 s' \wedge s' \preceq t')$;
   - *(b)* $\forall s' \in \mathcal{Q}_1, \; (s \xrightarrow{\alpha}_1 s') \implies (t = \top \vee \exists t' \in \mathcal{Q}_2, \; t \xrightarrow{\alpha}_2 t' \wedge s' \preceq t')$.

4. *If $(s,t) \in \mathcal{Q}_1^p \times \mathcal{Q}_2^p$ then there exists a function $\delta : \mathcal{Q}_1 \times \mathcal{Q}_2 \to [0,1]$, which, for each $s' \in \mathcal{Q}_1$, gives a probability distribution $\delta(s')$ over $\mathcal{Q}_2$, such that for every probability distribution f over $\mathcal{Q}_1$ with $f(s') \in \sigma_1(s)(s')$ and $\forall t' \in \mathcal{Q}_2$,*

$$\sum_{s' \in \mathcal{Q}_1} f(s') * \delta(s')(t') \in \sigma_2(t)(t') \quad \textit{and} \quad \forall s' \in \mathcal{Q}_1 : \big(\delta(s')(t') > 0 \implies s' \preceq t'\big)$$

5. *If $(s,t) \in \mathcal{Q}_1^a \times \mathcal{Q}_2^p$ then $\exists t^a \in \mathcal{Q}_2^a : t \overset{>0}{\dashrightarrow}_2^{+} t^a \wedge s \preceq t^a$ and $\forall t' \in \mathcal{Q}_2$, $\big(t \overset{>0}{\dashrightarrow}_2 t' \implies s \preceq t'\big)$.*

6. *If $(s,t) \in \mathcal{Q}_1^p \times \mathcal{Q}_2^a$ then $\exists s^a \in \mathcal{Q}_1^a : s \overset{>0}{\dashrightarrow}_1^{+} s^a \wedge s^a \preceq t$ and $\forall s' \in \mathcal{Q}_1$, $\big(s \overset{>0}{\dashrightarrow}_1 s' \implies s' \preceq t\big)$.*

It can be shown that a greatest simulation relation, called refinement and noted $\leq$, exists. C_1 refines C_2 (written $C_1 \leq C_2$) iff $s_0 \leq t_0$.

In Definition 4.24, conditions (1) and (2) ensure that C_1 makes no stronger assumptions on the context than C_2, and that the inconsistent state $\bot$ is only refined by itself. Since Definition 4.24 defines $\leq$ as the greatest relation, this implies that for any state s, $\bot \leq s$ and $s \leq \top$.

Condition (3a) says that any action transition accepted by C_2 must also be accepted by C_1. In contrast, action transitions leading to $\top$ (i.e., violating the assumption) do not need to be present in the refinement C_1. This is why we have $\forall t' \neq \top$ in condition (3a). On the other hand, condition (3b) says that each action transition of C_1 must also be enabled in C_2, unless C_2 is in the $\top$ state. Condition (4), adapted from [173], deals with refinement among probabilistic states. Intuitively, $s \preceq t$ if there exists a function δ that distributes the probabilities of transitions from s to all successor states s' onto the transitions from t to its successors t', such that the sum of the probability fractions (i.e., $f(s') * \delta(s')(t')$) is in the range $\sigma_2(t)(t')$; this is illustrated in Example 4.11. Condition (5) says that an action state s refines a probabilistic state t if it refines all action states reachable with a path of positive probabilities from t. Finally, condition (6) is symmetrical to condition (5).

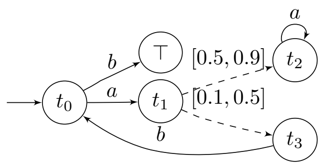
In §4.2.2, we gave an intuitive explanation of contracts: transitions leading to $\top$ model the violation of the assumption, whereas action transitions not leading to $\top$ model the guarantee that the transition has to be offered. The following example shows that Definition 4.24 is consistent with the usual contravariant notion of contract refinement requiring that the refining contract has a weaker assumption and a stronger guarantee.

Example 4.10 *In Figure 4.15 (a), the contract C_2 says that, in the state t_0, the action b is assumed not to happen; if an interaction involving b occurs (and the environment violates the assumption of C_2), then the component implementing C_2 is no longer bound by C_2; i.e., it can do anything after the action b is synchronized. The contract C_2 also says that, in the state t_0, the action a is guaranteed to be offered. Thus, a contract can refine C_2 in different ways, as shown in Figure 4.15:*

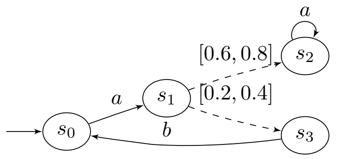
(1) $C_{1a} \leq C_2$: the contract C_{1a} does not offer action b in state s_0.

(2) $C_{1b} \leq C_2$: the contract C_{1b} offers action b in state u_0. If the b is synchronized with its environment, it reaches state u_4, from which C_{1b} can perform any action.

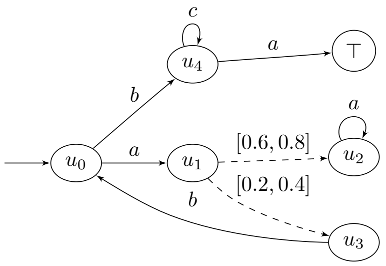
Both in C_{1a} and C_{1b}, the action a is guaranteed in state s_0 and u_0 respectively. It is also easy to check that $s_1 \leq t_1$ as the probabilistic transition leading to s_2 has a tighter *interval* *and $s_2 \leq t_2$, and similarly for the transition leading to s_3. This means that both C_{1a} and C_{1b} have stronger guarantees than C_2. At the same time, the states of both C_{1a} and C_{1b} have fewer transitions leading to $\top$ than the states of C_2 they are refining. For instance, contract C_{1a} guarantees not to offer action b in state s_0, whereas C_2 assumes b not to occur. This means that u_0 accepts more behaviors by the environment than t_0 without reaching $\top$.*



(a) Contract C_2



(b) Contract C_{1a}



(c) Contract C_{1b}

Figure 4.15: Stronger guarantee and weaker assumption

We define the satisfaction of a contract by an IMC as the refinement of the contract by the lifted IMC (i.e., written in the form of a contract).

Definition 4.25 (Contract satisfaction) *An IMC M satisfies a contract C (written $M \models C$) iff $\lfloor M \rfloor \leq C$.*

Example 4.11 *We illustrate in Figure 4.16 how to check that the contracts of Figure 4.13 are such that $\lfloor M_s \rfloor \leq C_s$, in particular, $s_1 \leq t_1$. It is easy to check that $s_3 \leq t_2$, $s_4 \leq t_2$, and $s_2 \leq t_3$. According to Condition (4) in Definition 4.24, we must find for each $s_i \in \{s_2, s_3, s_4\}$ a probability distribution $\delta(s_i)$ over $\{t_2, t_3\}$ such that $\sum_{i\in\{2,3,4\},j\in\{2,3\}} f(s_i) * \delta(s_i)(t_j) \in \sigma_2(t_1)(t_j)$ — where f is the probability distribution over $\{s_2, s_3, s_4\}$ with $f(s_2) = 0.1$, $f(s_3) = 0.7$, and $f(s_4) = 0.2$ —, and $\delta(s_i)(t_j) = 0$ if $s_i \not\leq t_j$. In Figure 4.16, $\delta(s_3)(t_2) = d_1$, $\delta(s_4)(t_2) = d_2$, $\delta(s_2)(t_3) = d_3$ (all three represented by dotted lines), and $\delta(s_i)(t_j) = 0$ for all other pairs of states. We must thus check that for each tuple (p_2, p_3, p_4) satisfying the constraints (1) to (4) in Figure 4.16, the constraints (5) and (6) are implied. As each $\delta(s_i)$ is a probability distribution, we obtain for our example $d_1 = d_2 = d_3 = 1$. (Note that if we had $s_2 \leq t_2$ as well with weight d_4 from s_2 to t_2, we would have another constraint $d_3+d_4 = 1$, and (5) would become $p_3*d_1+p_4*d_2+p_2*d_4 \in [0.9, 1]$.) Condition (4) can be checked efficiently by requiring the set inclusion to hold for the bounds of interval $\sigma(s)(s')$, using a linear programming solver.*

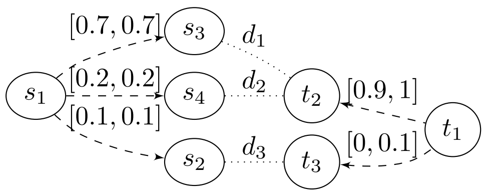
Definition 4.26 (Models of contracts) *The set of models of a contract C — written $\mathcal{M}(C)$ — is the set of IMCs that satisfy C: $\mathcal{M}(C) = \{M \mid M \models C\}$.*

It can be checked that the inconsistent contract $C_\perp$, consisting only of the state $\perp$, does not have any model.

In [155] we have shown refinement to be reflexive and transitive.

Bisimulation

We adapt the usual notion of bisimulation to contracts, and define reduction of a contract with respect to bisimulation.



(1) $p_2 \in [0.1, 0.1]$
(2) $p_3 \in [0.7, 0.7]$
(3) $p_4 \in [0.2, 0.2]$
(4) $p_2 + p_3 + p_4 = 1$
(5) $p_3 * d_1 + p_4 * d_2 \in [0.9, 1]$
(6) $p_2 * d_3 \in [0, 0.1]$

Figure 4.16: Left: Contract refinement $s_1 \leq t_1$. Right: Constraints to be checked.

Definition 4.27 (Bisimulation $\simeq$) *Given two contracts $C_1 = (\mathcal{Q}_1, \mathcal{A}, \rightarrow_1, \sigma_1, s_0)$ and $C_2 = (\mathcal{Q}_2, \mathcal{A}, \rightarrow_2, \sigma_2, t_0)$, a relation $\simeq \subseteq \mathcal{Q}_1 \times \mathcal{Q}_2$ is a* bisimulation *if both $\simeq$ and $\simeq^{-1} = \{(t, s) \mid s \simeq t\}$ are simulations.*

C_1 and C_2 are bisimilar (written $C_1 \simeq C_2$) iff $s_0 \simeq t_0$, where $\simeq$ is the greatest bisimulation.

Definition 4.28 (Reduction modulo $\simeq$ and reduced contract $\overline{C}$) *Let $C = (\mathcal{Q}, \mathcal{A}, \rightarrow, \sigma, s_0)$ be a contract and $\simeq$ be a bisimulation over $\mathcal{Q}$. For all $s \in \mathcal{Q}$, let $\mathcal{C}_s = \{q \in \mathcal{Q} \mid s \simeq q\}$ be the equivalence class of s. Let $\mathcal{C} = \{\mathcal{C}_s \mid s \in \mathcal{Q}\}$. The reduced contract, written $C_{/\simeq}$, is $(\mathcal{C}, \mathcal{A}, \rightarrow_{\simeq}, \sigma_{\simeq}, \mathcal{C}_{s_0})$ with $\mathcal{C}^p = \{c \in \mathcal{C} \mid \forall s \in c : s \in \mathcal{Q}^p\}$ and $\mathcal{C}^a = \mathcal{C} \setminus (\mathcal{C}^p \cup \{\top, \bot\})$ such that, $\forall s = \{s_1, \ldots, s_m\}, t = \{t_1, \ldots, t_n\} \in \mathcal{C}$, we have:*

- *$s \overset{\alpha}{\rightarrow}_{\simeq} t$ iff $\exists i, j : s_i \overset{\alpha}{\rightarrow} t_j$, and*
- *$\sigma_{\simeq}(s, t) = \sum_{1 \leq j \leq n} \sigma(s_1, t_j)$ iff $s \in \mathcal{C}^p$.*

If $\simeq$ is the greatest bisimulation then we write $\overline{C}$ for $C_{/\simeq}$.

Notice that an equivalence class may contain both action and probabilistic states. For each probabilistic state $s_i \in s$, the probabilities of transitions to states $t_j \in t$ are summed up (it does not matter which of the transitions is taken since all the successors t_j are equivalent). This sum is the transition probability from s_i to some state in t. By definition of $\simeq$, the sum is the same for all $s_i \in s$, thus we pick $\sigma(s_1, t_j)$.

Lemma 4.1 (Bisimilarity of reduction) *For any contract C, we have $\overline{C} \simeq C$.*

Definition 4.29 (Deadend freedom) *A delimited contract $C = (\mathcal{Q}, \mathcal{A}, \rightarrow, \sigma, s_0)$ is deadend-free if any reachable state has an outgoing transition in $(\mathcal{Q} \setminus \{\top\}, \mathcal{A}, \rightarrow', \sigma, s_0)$ where $\rightarrow' = \{(q, a, q') \in \rightarrow \mid q' \neq \top\}$.*

In other words, C is deadend-free if all reachable action states have a successor state other than $\top$. In particular, $\bot$ is unreachable in any deadend-free contract since $\bot$ has no successor at all.

Theorem 4.6 (Refinement preserves deadend-freedom) *Let $C = (\mathcal{Q}, \mathcal{A}, \rightarrow, \sigma, s_0)$ and $C' = (\mathcal{Q}', \mathcal{A}, \rightarrow', \sigma', s'_0)$ be two contracts such that $C' \leq C$, and C' is delimited and consistent. If C is deadend-free then so is C'.*

Contract Projection

The need of projection arises naturally in contract frameworks. $\mathcal{A}$ and $\mathcal{B}$ being two alphabets of actions such that $\mathcal{B} \subseteq \mathcal{A}$, we abstract from actions in $\mathcal{A} \setminus \mathcal{B}$ that are not relevant by renaming them into internal τ actions. The contract over the alphabet $\mathcal{B} \cup \{\tau\}$ is then projected on the sub-alphabet $\mathcal{B}$ by using the standard determinization algorithm by subset construction (see e.g. [4]).

Definition 4.30 (Projection) *Let $C = (\mathcal{Q}, \mathcal{A}, \rightarrow_1, \sigma, s_0)$ be a contract and $\mathcal{B} \subseteq \mathcal{A}$ such that for any $q \in \mathcal{Q}^a$ and $\alpha \in \mathcal{A}$, if $q \xrightarrow{\alpha}_1 \top$ or $q \xrightarrow{\alpha}_1 \bot$ then $\alpha \in \mathcal{B}$. Let $C' = (\mathcal{Q}, \mathcal{B} \cup \{\tau\}, \rightarrow_2, \sigma, s_0)$ be the contract where all transition labels in $\mathcal{A} \setminus \mathcal{B}$ are replaced with a new label τ. We require that C is such that $act \cap prob = \emptyset$ where*

$$
\begin{aligned}
act = \big\{q \in \mathcal{Q} \mid \exists q' \in \mathcal{Q} : q \xrightarrow{\tau^*}_2 q' \wedge \\
&\big((\exists \alpha \in \mathcal{B}\ \exists q'' \in \mathcal{Q} : q' \xrightarrow{\alpha}_2 q'') \vee (\forall q'' : q' \xrightarrow{\tau^*}_2 q'' \implies q'' \in \mathcal{Q}^a)\big)\big\} \\
prob = \{q \in \mathcal{Q} \mid \exists q' \in \mathcal{Q}^p : q \xrightarrow{\tau^*}_2 q'\}
\end{aligned}
$$

and $\xrightarrow{\tau^*}_2$ is the transitive and reflexive closure of $\xrightarrow{\tau}_2$.

The projection of C on $\mathcal{B}$ (written $\pi_{\mathcal{B}}(C)$) is obtained by τ-elimination (determinization) of C'.

The requirement that action transitions immediately leading to $\top$ or $\bot$ be kept in the projection ensures that Hypothesis 4.1 is preserved. The second requirement ensures that the states of $\pi_{\mathcal{B}}(C)$ are partitioned into action states, probabilistic states, $\{\top\}$, and $\{\bot\}$. More precisely, *act* is the set of states q from which a state q' is reachable by taking only τ transitions, such that either a transition with an action label in $\mathcal{B}$ is enabled in q', or no more probabilistic state is reachable. Conversely, *prob* is the set of states from where a probabilistic state can be reached. Disjointness of both sets ensures that every state of $\pi_{\mathcal{B}}(C)$ is uniquely typed, such that $\pi_{\mathcal{B}}(C)$ is a contract again.

Lemma 4.2 (Projection and refinement) *For all contracts $C_1 = (\mathcal{Q}_1, \mathcal{A}, \rightarrow_1, \dashrightarrow_1, s_0)$ and $C_2 = (\mathcal{Q}_2, \mathcal{A}, \rightarrow_2, \dashrightarrow_2, t_0)$ and for all $\mathcal{B} \subseteq \mathcal{A}$ such that $\pi_{\mathcal{B}}(C_1)$ and $\pi_{\mathcal{B}}(C_2)$ are defined, if $C_1 \leq C_2$ then $\pi_{\mathcal{B}}(C_1) \leq \pi_{\mathcal{B}}(C_2)$.*

Example 4.12 *In Figure 4.13, if we do not care how the implementation handles failure cases, we can check that $\pi_{\mathcal{A}_s \setminus \{handle\}}(M_s) \models C_s$, where $\mathcal{A}_s$ is the action alphabet of C_s.*

4.2.4 Contract Composition

We introduce two composition operations for contracts: parallel composition $\|$ parametrized with an interaction model, and conjunction $\wedge$ (also called shared refinement).

Parallel Composition of Contracts

Parallel composition allows the designer to build complex models from simpler components in a stepwise and hierarchical manner. In order to reason about the composition of components at the contract level, we extend the parallel composition of BIP component behaviors (Definition 2.28) to probabilistic contracts.

Definition 4.31 (Parallel composition of contracts) *Let $C_1 = (\mathcal{Q}_1, \mathcal{A}_1, \rightarrow_1, \dashrightarrow_1, s_0)$ and $C_2 = (\mathcal{Q}_2, \mathcal{A}_2, \rightarrow_2, \dashrightarrow_2, t_0)$ be two contracts and IM an interaction model over $\mathcal{A}_1 \cup \mathcal{A}_2$. The parallel composition of C_1 and C_2 with respect to IM – written $C_1 \|_{IM} C_2$ – is the contract $\big(\mathcal{Q}, \mathcal{I}, \rightarrow', \dashrightarrow, (s_0, t_0)\big)$ where:*

1. *$\mathcal{Q} = (\mathcal{Q}_1' \times \mathcal{Q}_2') \cup \{\top, \bot\}$ with $Q_1' = Q_1 \setminus \{\top_1, \bot_1\}$, $Q_2' = Q_2 \setminus \{\top_2, \bot_2\}$, $\mathcal{Q}^a = \mathcal{Q}_1^a \times \mathcal{Q}_2^a$, and $\mathcal{Q}^p = \mathcal{Q} \setminus (\mathcal{Q}^a \cup \{\top, \bot\})$;*

2.
$$
\begin{aligned}
\rightarrow' = \{(q, a, q') \in \rightarrow \mid q' \notin \mathcal{Q}^\top \cup \mathcal{Q}^\bot\} \cup \\
\{(q, a, \top) \mid \exists q' \in \mathcal{Q}^\top : (q, a, q') \in \rightarrow\} \cup \\
\{(q, a, \bot) \mid \exists q' \in \mathcal{Q}^\bot : (q, a, q') \in \rightarrow\}
\end{aligned}
$$

where $\rightarrow$ is the least relation satisfying the rules [R1]–[R3] *in Figure 4.17; and*

3. $\dashrightarrow$ *is the least relation satisfying the rules* [R4]–[R6] *in Figure 4.17*

where $\mathcal{Q}^\top = (\mathcal{Q}_1 \times \{\top_2\}) \cup (\{\top_1\} \times \mathcal{Q}_2)$ and $\mathcal{Q}^\bot = (\mathcal{Q}_1 \times \{\bot_2\}) \cup (\{\bot_1\} \times \mathcal{Q}_2)$.

In other words, $\top$ (resp. $\bot$) is reached in $C_1 \|_{IM} C_2$ as soon as one of C_1 or C_2 reaches its $\top_i$ (resp. $\bot_i$) state.

$$\frac{q_1 \xrightarrow{\alpha}_1 q_1' \quad \alpha \in \mathcal{I} \qquad q_2 \in \mathcal{Q}_2^a}{(q_1, q_2) \xrightarrow{\alpha} (q_1', q_2)} \ [\mathrm{R1}] \qquad \frac{q_2 \xrightarrow{\alpha}_2 q_2' \quad \alpha \in \mathcal{I} \qquad q_1 \in \mathcal{Q}_1^a}{(q_1, q_2) \xrightarrow{\alpha} (q_1, q_2')} \ [\mathrm{R2}]$$

$$\frac{q_1 \xrightarrow{\alpha}_1 q_1' \quad q_2 \xrightarrow{\beta}_2 q_2' \quad \alpha|\beta \in \mathcal{I}}{(q_1, q_2) \xrightarrow{\alpha|\beta} (q_1', q_2')} \ [\mathrm{R3}] \qquad \frac{q_1 \overset{[p1,p2]}{\dashrightarrow}_1 q_1' \quad q_2 \overset{[p3,p4]}{\dashrightarrow}_2 q_2'}{(q_1, q_2) \overset{[p_1 * p_3, p_2 * p_4]}{\dashrightarrow} (q_1', q_2')} \ [\mathrm{R4}]$$

$$\frac{q_1 \overset{P}{\dashrightarrow}_1 q_1' \qquad q_2 \in \mathcal{Q}_2^a}{(q_1, q_2) \overset{P}{\dashrightarrow} (q_1', q_2)} \ [\mathrm{R5}] \qquad \frac{q_2 \overset{P}{\dashrightarrow}_2 q_2' \qquad q_1 \in \mathcal{Q}_1^a}{(q_1, q_2) \overset{P}{\dashrightarrow} (q_1, q_2')} \ [\mathrm{R6}]$$

Figure 4.17: Rules for the parallel composition of contracts.

Rules [R1] to [R3] are the usual parallel composition rules for LTS, while Rule [R4] is similar to the typical parallel composition for Markov chains but on probability intervals. Finally, Rules [R5] and [R6] state that probabilistic transitions, usually modeling hidden internal behavior, have priority over action transitions. Parallel composition is commutative since the rules are symmetrically defined.

Example 4.13 *Figure 4.18 illustrates the parallel composition of contracts C_s (from Figure 4.13(b)) and $C_\ell = \lfloor M_\ell \rfloor$ (where M_ℓ is given in Figure 4.12(b)), with $IM = \{rec, del, req'|del', res'|rec', fail_1, fail_2\}$. The composed contract $C_s \|_{IM} C_\ell$ states that a failure in the Link component does not prevent it from continuing to deliver the request req' to the Server, and receiving the response res' from the Server, but the failure prevents it from delivering the response res' back to the Client.*

We end the section on parallel composition with two useful theorems.

Theorem 4.7 (Congruence of refinement for $\|_{IM}$) *For all contracts C_1, C_2, C_3, C_4 and an interaction model IM, if $C_1 \leq C_2$ and $C_3 \leq C_4$, then $C_1 \|_{IM} C_3 \leq C_2 \|_{IM} C_4$.*

Theorem 4.8 (Independent implementability) *For all IMCs M, N, contracts C_1, C_2, and interaction model IM, if $M \models C_1$ and $N \models C_2$, then $M \|_{IM} N \models C_1 \|_{IM} C_2$.*

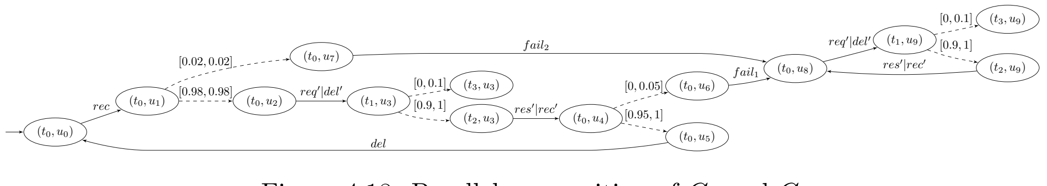


Figure 4.18: Parallel composition of C_s and C_ℓ.

Conjunction of contracts

A single component may have to satisfy several contracts that are specified independently, each of them specifying different requirements on the component, such as safety, reliability, or quality of service. Therefore, the contracts may use different, possibly overlapping, sub-alphabets of the component. The *conjunction* of contracts computes a common refinement of all contracts. Prior to conjunction, we define *similarity* of contracts as a test whether a common refinement exists.

Definition 4.32 (Similarity ($\sim$)) *Let $C_1 = (\mathcal{Q}_1, \mathcal{A}_1, \rightarrow_1, \dashrightarrow_1, s_0)$ and $C_2 = (\mathcal{Q}_2, \mathcal{A}_2, \rightarrow_2, \dashrightarrow_2, t_0)$ be two contracts. $\sim \subseteq (\mathcal{Q}_1 \setminus \{\bot\}) \times (\mathcal{Q}_2 \setminus \{\bot\})$ is the largest relation such that $\forall (s,t) \in (\mathcal{Q}_1 \setminus \{\bot\}) \times (\mathcal{Q}_2 \setminus \{\bot\})$, $s \sim t$ iff $(s = \top \vee t = \top)$ or conditions (1) to (4) below hold:*

1. *If $(s,t) \in \mathcal{Q}_1^a \times \mathcal{Q}_2^a$ then*
   (a) *for all $s' \in \mathcal{Q}_1$, if $s \xrightarrow{\alpha}_1 s'$, then either*
      i. *$\alpha \notin \mathcal{A}_2$, or*
      ii. *$\alpha \in \mathcal{A}_2$ and $\exists m \geq 0$, $\exists \beta_1, ..., \beta_m \in \mathcal{A}_2 \setminus \mathcal{A}_1$, $\exists t_1, ..., t_m, t' \in \mathcal{Q}_2 : t \xrightarrow{\beta_1}_2 t_1 \xrightarrow{\beta_2}_2 \dots \xrightarrow{\beta_m}_2 t_m \xrightarrow{\alpha}_2 t' \wedge \forall i \in 1...m : s \sim t_i$;*

   (b) *for all $t' \in \mathcal{Q}_2$, if $t \xrightarrow{\alpha}_2 t'$, then either*
      i. *$\alpha \notin \mathcal{A}_1$, or*
      ii. *$\alpha \in \mathcal{A}_1$ and $\exists m \geq 0$, $\exists \beta_1, ..., \beta_m \in \mathcal{A}_1 \setminus \mathcal{A}_2$, $\exists s_1, ..., s_m, s' \in \mathcal{Q}_1 : s \xrightarrow{\beta_1}_1 s_1 \xrightarrow{\beta_2}_1 \dots \xrightarrow{\beta_m}_1 s_m \xrightarrow{\alpha}_1 s' \wedge \forall i \in 1...m : s_i \sim t$;*

2. *If $(s,t) \in \mathcal{Q}_1^p \times \mathcal{Q}_2^p$ then*
   (a) *for all $s' \in \mathcal{Q}_1$, if $s \overset{P_1}{\dashrightarrow} s'$, then $t \overset{P_2}{\dashrightarrow} t'$ for some $t' \in \mathcal{Q}_2$ with $P_1 \cap P_2 \neq \emptyset$ and $s' \sim t'$; and*
   (b) *for all $t' \in \mathcal{Q}_2$, if $t \overset{P_2}{\dashrightarrow} t'$, then $s \overset{P_1}{\dashrightarrow} s'$ for some $s' \in \mathcal{Q}_1$ with $P_1 \cap P_2 \neq \emptyset$ and $s' \sim t'$;*

3. *If $(s,t) \in \mathcal{Q}_1^a \times \mathcal{Q}_2^p$ then for all $t' \in \mathcal{Q}_2$ with $t \overset{P}{\dashrightarrow}_2 t'$, $s \sim t'$;*

4. *If $(s,t) \in \mathcal{Q}_1^p \times \mathcal{Q}_2^a$ then for all $s' \in \mathcal{Q}_1$ with $s \overset{P}{\dashrightarrow}_1 s'$, $s' \sim t$.*

Finally, C_1 and C_2 are similar, written $C_1 \sim C_2$, iff $s_0 \sim t_0$.

Each P_i in Definition 4.32 refers to a probabilistic interval in the form of $[\ell_i, u_i]$. Any state is similar to a top state $\top_i$ (where the contract does not constrain the implementation in any way). The bottom states $\bot_i$ are not similar to any state. Two action states are similar if they agree on the enabled actions in the shared alphabet $\mathcal{A}_1 \cap \mathcal{A}_2$. The successor states are not required to be similar again, as they may be made unreachable in a subsequent parallel composition. Two probabilistic states are similar if the probabilistic transitions can be matched such that the intervals overlap ($P_1 \cap P_2 = \emptyset$) and the successor states are similar. Overall, two states are similar if they agree on the behavior up to and including the next reachable action transition in the shared alphabet.

Definition 4.33 (Unambiguous contract) *A contract $C = (\mathcal{Q}, \mathcal{A}, \rightarrow, \dashrightarrow, s_0)$ is unambiguous w.r.t $\mathcal{B} \subseteq \mathcal{A}$ iff for all r, s, and $t \in \mathcal{Q}$ such that:*

$$\left(r \overset{>0}{\dashrightarrow} s \;\wedge\; r \overset{>0}{\dashrightarrow} t\right) \;\vee\; \left(\exists \alpha, \beta \in (\mathcal{A} \setminus \mathcal{B}) \cup \{\emptyset\} : r \xrightarrow{\alpha} s \;\wedge\; r \xrightarrow{\beta} t\right)$$

we have: if $s \sim t$ then $s = t$, where $q \xrightarrow{\emptyset} q$ for all $q \in \mathcal{Q}$, .
C is unambiguous *if it is unambiguous w.r.t $\mathcal{A}$.*

In other words, a contract is *unambiguous* if the reachable successor states of any probabilistic state are pairwise non-similar.

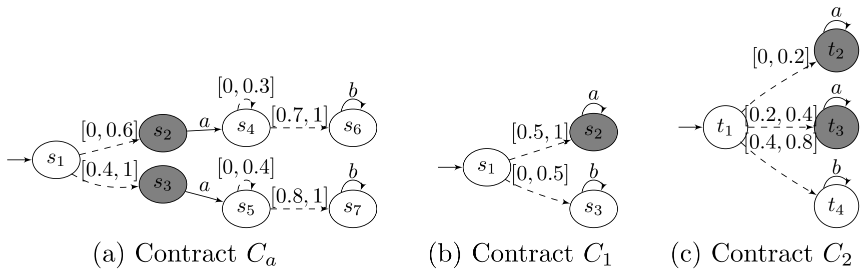


Figure 4.19: (a) An ambiguous contract C_a; (b,c) Two non-similar contracts C_1 and C_2.

Example 4.14 *In Figure 4.19(a), the contract C_a is ambiguous because $s_2 \sim s_3$ (highlighted in gray) but $s_2 \neq s_3$.*

We are now ready to define the conjunction of two contracts. The two contracts may refer to different, not necessarily disjoint alphabets. Therefore, the contracts can be used to specify requirements on two (not necessarily disjoint) aspects of a component.

Definition 4.34 (Conjunction of contracts ($\wedge$)) *For contracts $C_1 = (\mathcal{Q}_1, \mathcal{A}_1, \rightarrow_1, \dashrightarrow_1, s_0)$ and $C_2 = (\mathcal{Q}_2, \mathcal{A}_2, \rightarrow_2, \dashrightarrow_2, t_0)$ such that C_1 and C_2 are unambiguous w.r.t $\mathcal{A}_1 \cap \mathcal{A}_2$, let $C_1 \wedge C_2$ be the contract $\big(\mathcal{Q}, \mathcal{A}_1 \cup \mathcal{A}_2, \rightarrow', \dashrightarrow, (s_0, t_0)\big)$ where:*

1. *$\mathcal{Q} = \{(q_1, q_2) \in \mathcal{Q}_1 \times \mathcal{Q}_2 \mid q_1 \sim q_2 \wedge (q_1 \neq \top_1 \vee q_2 \neq \top_2)\} \cup \{\top, \bot\}$, $\mathcal{Q}^p = \mathcal{Q} \cap \big((\mathcal{Q}_1^p \times \mathcal{Q}_2) \cup (\mathcal{Q}_1 \times \mathcal{Q}_2^p)\big)$, and $\mathcal{Q}^a = \mathcal{Q} \setminus (\mathcal{Q}^p \cup \{\top, \bot\})$;*

2. $$\begin{aligned} \rightarrow' = \; & \{(q, a, q') \in \rightarrow \mid q' \in \mathcal{Q}\} \cup \\ & \{(q, a, \top) \mid \big(q, a, (\top_1, \top_2)\big) \in \rightarrow\} \cup \\ & \{(q, a, \bot) \mid \exists q' = (q_1', q_2') \in \mathcal{Q}_1 \times \mathcal{Q}_2 : \neg(q_1' \sim q_2') \wedge (q, a, q') \in \rightarrow\} \end{aligned}$$

   where $\rightarrow$ is the least relation satisfying the rules [C1] – [LiftR] *in Figure 4.20, and*

3. *$\dashrightarrow$ is the least relation satisfying the rules* [C3] – [C4R] *in Figure 4.20 (where for all other probabilistic transitions $(q_1, q_2) \overset{P}{\dashrightarrow} (q_1', q_2')$, $P = [0, 0]$).*

The $\bot$ state is entered in the contract $C_1 \wedge C_2$ as soon as a pair of non-similar states (including, by definition, pairs with at least one $\bot$ state) is reached.

Rule [C1] requires the contracts to agree on action transitions over their common alphabet. According to rule [C2L] (resp. [C2R]), the conjunction behaves like the first (resp. second) contract as soon as the other contract is in $\top$. Rules [LiftL] and [LiftR] allow the interleaving of action transitions that are not in the common alphabet. Rules [C3] – [C4R] define probabilistic transitions whose successor states are similar.

Example 4.15 *Figure 4.21 shows three contracts for the Link component: $C_{\ell 1}$ specifies that the implementation should receive a request (rec) from the Client and deliver it to the Server (del′); $C_{\ell 2}$ specifies that the implementation should receive a response (rec′) from*

$$\frac{q_1 \xrightarrow{\alpha}_1 q_1' \quad q_2 \xrightarrow{\alpha}_2 q_2'}{(q_1, q_2) \xrightarrow{\alpha} (q_1', q_2')} \quad [\text{C1}]$$

$$\frac{q_1 \xrightarrow{\alpha}_1 q_1'}{(q_1, \top_2) \xrightarrow{\alpha} (q_1', \top_2)} \quad [\text{C2L}] \qquad \frac{q_2 \xrightarrow{\alpha}_2 q_2'}{(\top_1, q_2) \xrightarrow{\alpha} (\top_1, q_2')} \quad [\text{C2R}]$$

$$\frac{q_1 \xrightarrow{\alpha}_1 q_1' \quad q_2 \in \mathcal{Q}_2^a}{(q_1, q_2) \xrightarrow{\alpha} (q_1', q_2)} \quad \alpha \notin \mathcal{A}_2 \quad [\textsc{LiftL}] \qquad \frac{q_2 \xrightarrow{\alpha}_2 q_2' \quad q_1 \in \mathcal{Q}_1^a}{(q_1, q_2) \xrightarrow{\alpha} (q_1, q_2')} \quad \alpha \notin \mathcal{A}_1 \quad [\textsc{LiftR}]$$

$$\frac{q_1 \overset{P_1}{\dashrightarrow}_1 q_1' \quad q_2 \overset{P_2}{\dashrightarrow}_2 q_2'}{(q_1, q_2) \overset{P_1 \cap P_2}{\dashrightarrow} (q_1', q_2')} \quad q_1' \sim q_2' \quad [\text{C3}]$$

$$\frac{q_1 \overset{P}{\dashrightarrow}_1 q_1' \quad q_2 \in \mathcal{Q}_2^a \cup \{\top_2\}}{(q_1, q_2) \overset{P}{\dashrightarrow} (q_1', q_2)} \quad q_1' \sim q_2 \quad [\text{C4L}]$$

$$\frac{q_2 \overset{P}{\dashrightarrow}_2 q_2' \quad q_1 \in \mathcal{Q}_1^a \cup \{\top_2\}}{(q_1, q_2) \overset{P}{\dashrightarrow} (q_1, q_2')} \quad q_1 \sim q_2' \quad [\text{C4R}]$$

Figure 4.20: Rules for conjunction of contracts.

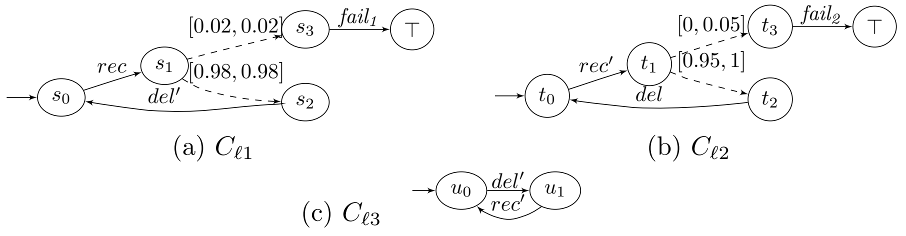


Figure 4.21: Example: Conjunction of Contracts

the Server and deliver it to the Client (del); $C_{\ell 3}$ requires the response (rec′) received from the Server to occur after the request (del′) delivered to the Server. We can verify that $M_\ell \models (C_{\ell 1} \wedge C_{\ell 3}) \wedge (C_{\ell 2} \wedge C_{\ell 3})$ (where M_ℓ is in Figure 4.12(b)).

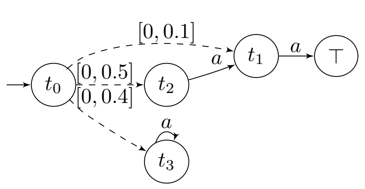
Theorem 4.9 (Associativity of conjunction over the same alphabet) *For all unambiguous contracts C_1, C_2, and C_3 over the same alphabet, $(C_1 \wedge C_2) \wedge C_3 = C_1 \wedge (C_2 \wedge C_3)$.*

Theorem 4.10 (Soundness of conjunction) *For all unambiguous contracts C_1 and C_2, if $\pi_{\mathcal{A}_i}(C_1 \wedge C_2)$ is defined then $\pi_{\mathcal{A}_i}(C_1 \wedge C_2) \leq C_i$ for $i = 1, 2$.*

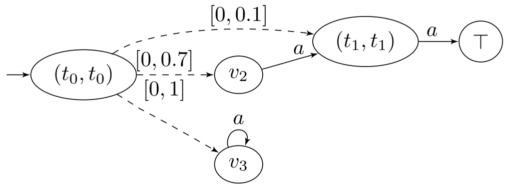
Example 4.16 *Figure 4.22 motivates the requirement of conjunction (Definition 4.34) for* unambiguous *contracts. The resulting contract $C_b \wedge C_b$ is reduced such that the model relation can be seen easily. In Figure 4.22(b), v_2 denotes the equivalent class $\{(t_1, t_2), (t_2, t_1), (t_2, t_2)\}$ while v_3 denotes the equivalent class $\{(t_1, t_3), (t_2, t_3), (t_3, t_1), (t_3, t_2), (t_3, t_3)\}$. Since $t_1 \sim t_2 \sim t_3$, duplicated intervals lead to an unsound result.*

Theorem 4.11 (Completeness of conjunction over the same alphabet) *For all delimited unambiguous contracts C_1, C_2, and C, if $C \leq C_1$ and $C \leq C_2$, then $C \leq C_1 \wedge C_2$.*

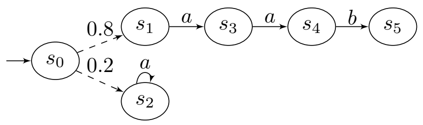
Theorem 4.12 (Congruence of refinement for $\wedge$ over the same alphabet) *For all delimited unambiguous contracts C_1, C_2, C_3, and C_4 over the same alphabet, if $C_1 \leq C_2$ and $C_3 \leq C_4$, then $C_1 \wedge C_3 \ \leq\ C_2 \wedge\ C_4$.*



(a) Ambiguous contract C_b



(b) $\overline{C_b \wedge C_b}$



(c) A model M_b

Figure 4.22: Example where $M_b \models C_b \wedge C_b$ but $M_b \not\models C_b$.

4.2.5 Case Study

We study a dependable computing system with time redundancy. The system specification is expressed by the contract C_S of Figure 4.23 (top left), which specifies that the computation *comp* should have a success probability of at least 0.999. If the computation fails, then nothing is specified (state ⊤).

The processor P the system is running on is specified by the contract C_P of Figure 4.23 (top right). Following an execution request *exe*, either the processor succeeds and replies with *ok* (with a probability at least p), or fails and replies with *nok* (with a probability at most $1-p$). The failure rates for successive executions are independent. The probability p is a *parameter* of the contract.

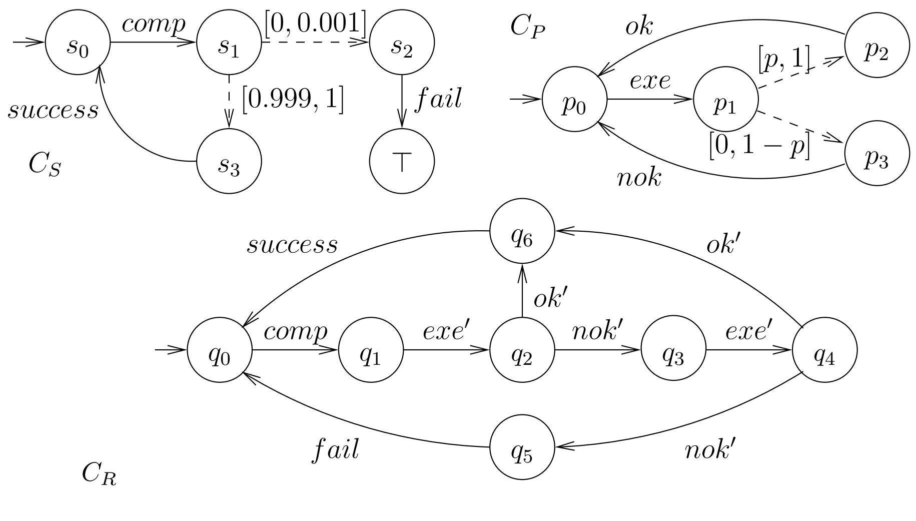


Figure 4.23: (top left) Specification C_S; (top right) Processor contract C_P; (bottom) Time redundancy contract C_R.

We place ourselves in a setting where the reliability level guaranteed by C_P alone (as expressed by p) cannot fulfill the requirement of C_S (that is, 0.999), and hence some form of redundancy must be used. We propose to use time redundancy, as expressed by the contract C_R of Figure 4.23 (bottom). Each computation *comp* is first launched on the processor P (*exe'*), either followed by a positive (*ok'*) or negative (*nok'*) answer from P. In the latter case, the execution is launched a second time, therefore implementing time redundancy. The contract C_R finally answers with *success* if *either* execution is followed by ok', or with $fail$ is *both* executions are followed by nok'.

In terms of component-based design for reliability, we wonder what is the minimum value of p that guarantees the reliability level of C_S. To compute this minimum value, we first compute the parallel composition $C_R\|_{IM}\,C_P$, with the interaction set $IM = \{comp, exe|exe', ok|ok', nok|nok', success, fail\}$. The reduction modulo bisimulation of this parallel composition is shown in Figure 4.24 (top), where the interactions $exe|exe'$, $ok|ok'$, and $nok|nok'$ have been replaced for conciseness by **exe**, **ok**, and **nok**, respectively. We call this new contract $C_{R||P}$. We then compute the projection of $C_{R||P}$ onto the set $\mathcal{B} = \{comp, success, fail\}$. The result $C_\pi = \pi_\mathcal{B}(C_{R||P})$ is shown in Figure 4.24 (bottom left).

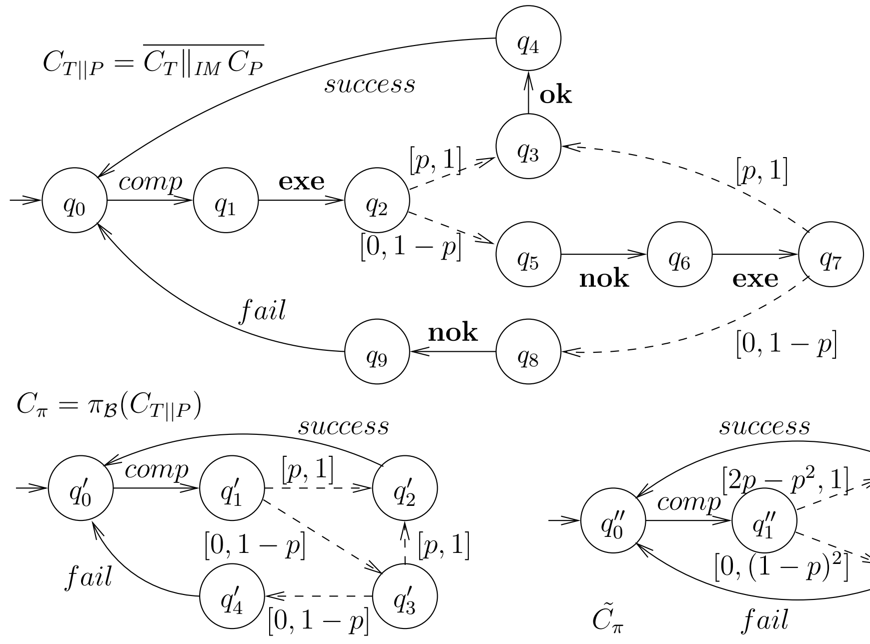


Figure 4.24: Parallel composition $C_{T||P}$; Projection C_π; Transitive closure $\tilde{C_\pi}$.

We are thus faced with a contract C_π having *sequences* of probabilistic transitions; more precisely, since some probabilistic states have several outgoing transitions, we have DAGs of probabilistic transitions. We therefore compute the transitive closure for each such DAG: that is, for each sequence of probabilistic transitions from the initial state of the DAG (e.g., q'_1 in C_π) to one of its final states (e.g., q'_2 and q'_4 in C_π), we compute the equivalent probabilistic transition. Starting from q'_1, the probability interval of reaching q'_2 (resp. q'_4) is given by $\{p' + (1-p')p' \mid p' \in [p,1]\}$ (resp. $\{(1-p')^2 \mid p' \in [p,1]\}$), that is, $[2p-p^2, 1]$ (resp. $[0,(1-p)^2]$). The resulting contract $\tilde{C_\pi}$ is shown in Figure 4.24 (bottom right).

The last step involves checking under which condition on p the contract $\tilde{C_\pi}$ refines the specification C_S. We have $\tilde{C_\pi} \leq C_S \Leftrightarrow (1-p)^2 \leq 0.001 \Leftrightarrow p \geq 0.968$. This means that, with time redundancy and a processor with a reliability level of at least 0.969, we are able to ensure an overall reliability level of 0.999.

To demonstrate the versatility of our contract framework, we show in Figure 4.25 the alternative contract C'_R for *spatial* redundancy. This time, the execution is launched both on processor 1 (exe_1) and on processor 2 (exe_2). We call C_{P1} the contract of processor 1, which is identical to C_P in Figure 4.23 (top right). We call C_{P2} the contract of processor 2, which is identical to C_{P1} upto a renaming of the index. The contract C'_R answers with *success* if *either* ok_1 or ok_2 is received, or with $fail$ is *both* nok_1 and nok_2 are received, in any order.

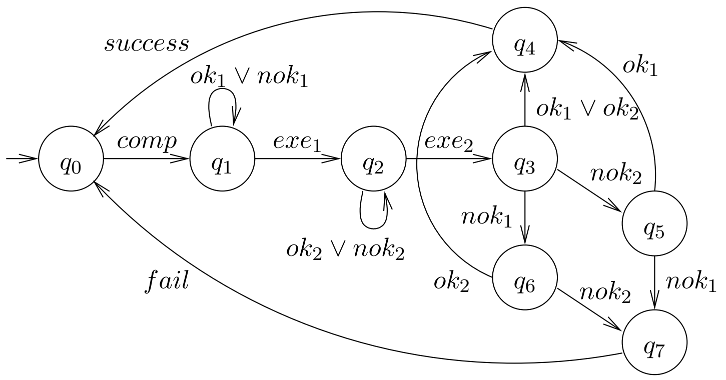


Figure 4.25: Spatial redundancy: the contract C'_R.

We leave the intermediate computations as exercises for the reader. These are:

- $C_A = C_{P1} \|_{IM} C_{P2}$ with $IM = \{exe'_1, ok'_1, nok'_1, exe'_2, ok'_2, nok'_2\}$.
- $C_B = C_A \|_{IM'} C'_R$ with $IM' = \{comp, success, fail, exe_1|exe'_1, ok_1|ok'_1, nok_1|nok'_1, exe_2|exe'_2, ok_2|ok'_2, nok_2|nok'_2\}$.

We then compute the projection $\pi_{\mathcal{B}}(\overline{C_B})$ onto the set $\mathcal{B} = \{comp, success, fail\}$. The reduction modulo bisimulation of the result, called C'_π, is shown in Figure 4.26 (left). Like with the time redundancy contract, we compute the transitive closure for each DAG of probabilistic transitions. The result $\tilde{C'_\pi}$ is shown in Figure 4.26 (right).

The last step involves checking under which condition on p_1 and p_2 the contract $\tilde{C'_\pi}$ refines the specification C_S. We have $\tilde{C'_\pi} \leq C_S \Leftrightarrow (1-p_1)(1-p_2) \leq 0.001$.

4.2.6 Discussion

We have introduced a design framework based on probabilistic contracts and proved essential properties for its use in component-based design. Our definition of contracts adapts ideas from [173, 270, 111], although the frameworks in [173, 111] do not support interactions between contracts.

Design choices

A fundamental syntactic choice in defining a symbolic contract framework is to define a contract either as a pair (assumption, guarantee) as in §4.1 — call them *assume/guarantee contracts* — or as a single *implicit* transition system where the distinction of assumption and guarantee is made by means of a specific $\top$ state, as in the present work. Whereas assume/guarantee contracts have the benefit of making explicit the assumptions of how a component is used and the guarantees provided by the component in this case, they come at the price of introducing some redundancy whenever the assumptions and the guarantees refer to the same sub-alphabet of the component. From a more technical point of view, another downside of assume/guarantee contracts is that parallel composition and conjunction of symbolic representations usually require computing an equivalent implicit form of the contract, whose definition is not obvious for probabilistic contracts.

A further choice is where to represent the probabilistic behavior: in the model of a component (i.e., the implementation), in the contract (i.e., the specification), or both. We

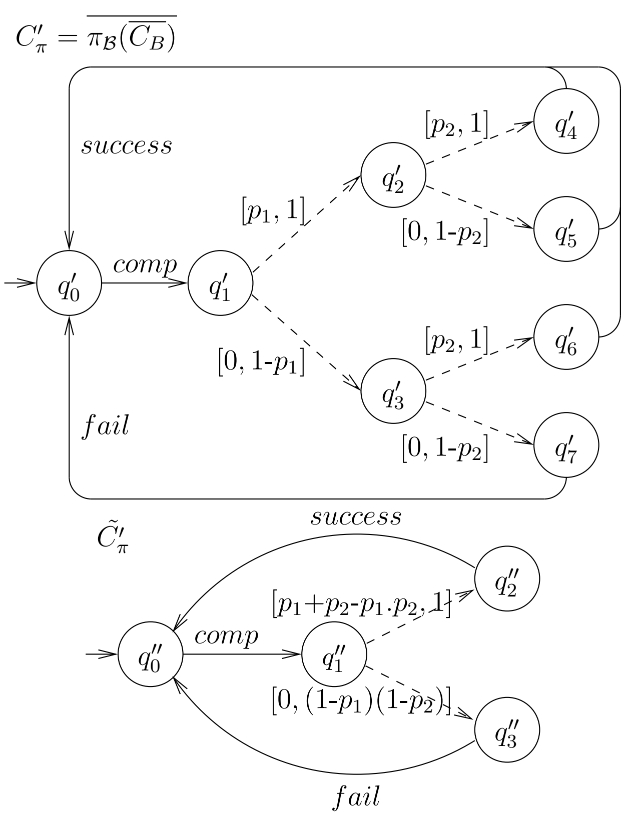


Figure 4.26: Projection $C'_\pi = \overline{\pi_{\mathcal{B}}(\overline{C_B})}$ onto the set $\mathcal{B} = \{comp, success, fail\}$; Transitive closure $\tilde{C}'_\pi$.

have chosen the last option, as it allows us to model both the *expected* probabilistic behavior and the behavior *offered* by existing components, and reason about how the specification can be realized.

Moreover, probability distributions can be *local* to contract states or *global*. In this work we have adopted the first option, as state-dependent distributions occur naturally in models of physical behavior: e.g., the failure rate of a microprocessor increases as the processor ages. The price of distinguishing local distributions are more involved definitions of refinement and conjunction.

A final parameter of the contract framework is the definition of parallel composition. We have chosen to support the BIP interaction model [154] for its expressiveness. In this framework, the direction of communications is not represented; it would be quite straightforward, however, to add this information by typing ports as input or output ports.

Related work

Several authors have proposed probabilistic extensions of Hoare triples and Dijkstra's *wp*-calculus, see e.g. [216]. A trace-based theory of probabilistic system with compositional semantics and refinement is introduced in [98]. Later on, shared refinement of interfaces and conjunction of modal specifications over possibly different alphabets have been defined in [107, 231]. A framework of modal assume/ guarantee contracts is introduced in [148], for which both parallel composition and conjunction are defined. [174] introduces a compositional framework based on continuous time IMCs, adopting a similar interaction model as done in this work. [174] supports projection, parallel and symmetric composition, but not conjunction.

A trace-based theory of probabilistic contracts has been introduced in [103], where a contract consists of an assumption A and a guarantee G, both being sets of traces. A trace is a sequence of valuations of global variables, a subset of which is probabilistic. The probabilistic variables are supposed to obey a distribution that is independent of the state. Two types of satisfaction of a contract C by a (non-probabilistic) model S are defined: R-satisfaction (for reliability) is the probability that S satisfies C; A-satisfaction (for availability) measures the expected time ratio during which S satisfies C. Conjunction and refinement are defined for both types of satisfaction. In contrast to our framework, probability distributions are defined globally.

Assume/guarantee verification of probabilistic models is studied in [182]. Probabilistic automata are used to model probabilistic and non-deterministic behavior. Several assume/guarantee rules are introduced using pairs (A, G) of probabilistic safety properties, where a probabilistic safety property is itself a pair of a (non-probabilistic) regular safety property and a probability.

The recently introduced Constraint Markov Chains (CMC) [63] generalize Markov chains by introducing constraints on state valuations and transition probability distributions, aiming at a similar goal of providing a probabilistic component-based design framework. Whereas CMCs do not support explicit interactions among components, they allow the designer to expressively specify constraints on probability distributions. In this framework, conjunction is shown to be sound and complete.

Much work remains to be done in contract-based design. In particular, we need results for behavioral contracts that help us move from pure theory to approaches that are usable in practice. We are currently investigating, in a PhD thesis co-advised by Thao Dang from VERIMAG, a contract framework for hybrid systems. In order to overcome the problems of complexity and undecidability, we intend to explore the use of set-based numerical approximations, with the support of tools such as SpaceEx [120].

Chapter 5

Trace-based Blaming

5.1 Introduction

In a concurrent, possibly embedded and distributed system, it is often crucial to determine which component(s) caused an observed failure. Understanding causality relationships between component failures and the violation of system-level properties can be especially useful to understand the occurrence of errors in execution traces, to allocate responsibilities, or to try to prevent errors (by limiting error propagation or the potential damages caused by an error).

The notion of causality inherently relies on a form of counterfactual reasoning: basically the goal is to try to answer questions such as "would event e_2 have occurred if e_1 had not occurred?" to decide if e_1 can be seen as a cause of e_2 (assuming that e_1 and e_2 have both occurred, or could both occur in a given context). But this question is not as simple as it may look:

1. First, we have to define what could have happened if e_1 had not occurred, in other words what are the *alternative worlds*.
2. In general, the set of alternative worlds is not a singleton and it is possible that in some of these worlds e_2 would occur while in others e_2 would not occur.
3. We also have to make clear what we call an event and when two events in two different traces can be considered as similar. For example, if e_1 had not occurred, even if an event potentially corresponding to e_2 might have occurred, it would probably not have occurred at the same time as e_2 in the original sequence of events; it could also possibly have occurred in a slightly different way (for example with different parameters, because of the potential effect of the occurrence of e_1 on the value of some variables).

Causality has been studied in many disciplines (philosophy, mathematical logic, physics, law, etc.) and from different points of view. In this work, we are interested in causality for the analysis of execution traces in order to establish the origin of a system-level failure. The main trend in the use of causality in computer science consists in mapping the abstract notion of event in the general definition of causality proposed by Halpern and Pearl in their seminal contribution [162] to properties of execution traces. Halpern and Pearl's model of causality relies on a counterfactual condition mitigated by subtle contingency properties to improve the accurateness of the definition and alleviate the limitations of the counterfactual reasoning in the occurrence of multiple causes. While Halpern and Pearl's model is a very precious contribution to the analysis of the notion of causality, we believe that a fundamentally different approach considering traces as first-class citizens is required in the computer science context considered here: The model proposed by Halpern and Pearl is based on an abstract notion of event defined in terms of propositional variables and causal models expressed as sets of equations between these variables. The equations define the basic causality dependencies between variables (such as $F = L_1$ or L_2 if F is a variable denoting the occurrence of a fire and L_1 and L_2 two lightning events that can cause the fire). In order to apply this model to execution traces, it is necessary to map the abstract notion of event onto properties of execution traces. But these properties and their causality dependencies are not given *a priori*, they should be derived from the system under study. In addition, a key feature of trace properties is the temporal ordering of events which is also intimately related to the idea of causality but is not an explicit notion in Halpern and Pearl's framework (even if notions of time can be encoded within events). Even though this application is not impossible, as shown by [32], we believe that definitions in terms of execution traces are preferable because (a) in order to determine the responsibility of components for an observed outcome, component traces provide the relevant granularity, and (b) they can lead to more direct and clearer definitions of causality.

As suggested above, many variants of causality have been proposed in the literature and used in different disciplines. It is questionable that one single definition of causality could fit all purposes. For example, when using causality relationships to establish liabilities, it may be useful to ask different questions, such as: "could event e_2 have occurred in some cases if e_1 had not occurred?" or "would event e_2 have occurred if e_1 had occurred but not e'_1?". These questions correspond to different variants of causality which can be perfectly legitimate and useful in different situations. To address this need, we propose two definition of causality relationships that can express these kinds of variants, called *necessary* and *sufficient* causality.

The framework presented here was introduced in [145]. It distinguishes a set of black-box components, each equipped with a specification. On a given execution trace, the causality of the components is analyzed with respect to the violation of a system-level property. In order to keep the definitions as simple as possible without losing generality — that is, applicability to various models of computation and communication —, we provide a language-based formalization of the framework. We believe that our general, trace-based definitions are unique features of our framework.

Traces can be obtained from an execution of the actual system, but also as counter-examples from testing or model-checking. For instance, we can model-check whether a behavioral model satisfies a property; causality on the counter-example can then be established against the component specifications.

5.2 Modeling Framework

In order to focus on the fundamental issues in defining causality on execution traces we introduce a simple, language-based modeling framework.

Definition 5.1 (Component signature) *A* component signature *C_i is a tuple $(\Sigma_i, \mathcal{S}_i)$ where Σ_i is an alphabet and $\mathcal{S}_i \subseteq \Sigma_i^*$ is a prefix-closed specification (set of allowed behaviors) over Σ_i.*

A component signature is the abstraction of an actual component that is needed to apply the causality analysis introduced here. Similarly, a system signature is the abstraction of a system composed of a set of interacting components.

Definition 5.2 (System signature) *A* system signature *is a tuple (C, Σ, B, ρ) where*

- $C = \{C_1, ..., C_n\}$ *is a finite set of component signatures* $C_i = (\Sigma_i, \mathcal{S}_i)$ *with pairwise disjoint alphabets;*
- $\Sigma \subseteq \Sigma'_1 \times ... \times \Sigma'_n$ *is a system alphabet with* $\Sigma'_i = \Sigma_i \cup \{\epsilon\}$*, where* ϵ *is a distinct element denoting that* C_i *does not participate in an interaction* $\alpha \in \Sigma$*;*
- $B \subseteq \Sigma^* \cup \Sigma^\omega$ *is a prefix-closed behavioral model;*
- $\rho \subseteq \left(\bigcup_i \Sigma_i\right) \times \left(\bigcup_i \Sigma_i\right)$ *is a relation modeling* information flow *among components.*

The behavioral model B is used to express assumptions and constraints on the possible (correct and incorrect) behaviors. The relation ρ models possible information flow among components. Intuitively, $(a, b) \in \rho$ means that any occurrence of a may influence the next occurrence of b (possibly in the same interaction), e.g., by triggering or constraining the occurrence of b, or by transmitting information.

Notation 5.1 *Given a word* $w = \alpha_1 {\cdot} \alpha_2 \cdots \in \Sigma^*$ *and an index* $i \in \mathbb{N}$ *let* $w[1..i] = \alpha_1 \cdots \alpha_i$*, let* $w[i] = \alpha_i$*, and* $w[i...] = \alpha_i \alpha_{i+1} \cdots$. *Let* $|w|$ *denote the length of* w. *For* $\alpha = (a_1, ..., a_n) \in \Sigma$ *let* $\alpha[k] = a_k$ *denote the action of component* k *in* α *(*$a_k = \epsilon$ *if* k *does not participate in* α*); for* $w = \alpha_1 \cdots \alpha_k \in \Sigma^*$ *and* $i \in 1...n$ *let* $\pi_i(w) = \alpha_1[i] \cdots \alpha_k[i]$ *(where* ϵ *letters are removed from the resulting word).*

For the sake of compactness of notations we define composition $\| : \Sigma_1^* \times ... \times \Sigma_n^* \to \Sigma^*$ such that $w_1 \| ... \| w_n = \{w \in \Sigma^* \mid \forall i \in 1...n : \pi_i(w) = w_i\}$, and extend $\|$ to languages.

5.2.1 Logs

A (possibly faulty) execution of a system may not be fully observable; therefore we base our analysis on *logs*. A log of a system $S = (C, \Sigma, B, \rho)$ with components $C = \{C_1, ..., C_n\}$ of alphabets Σ_i is a vector $\vec{tr} = (tr_1, ..., tr_n) \in \Sigma_1^* \times ... \times \Sigma_n^*$ of component traces such that there exists a trace $tr \in \Sigma^*$ with $\forall i \in 1...n : tr_i = \pi_i(tr)$. A log $\vec{tr} \in \mathcal{L}$ is thus the projection of an actual system-level trace $tr \in B$. This relation between the actual execution and the log on which causality analysis will be performed allows us to model the fact that only a partial order between the events in tr may be observable rather than their exact precedence.[1]

Let $\mathcal{L}(S)$ denote the set of logs of S. Given a log $\vec{tr} = (tr_1, ..., tr_n) \in \mathcal{L}(S)$ let $\vec{tr}^{\uparrow} = \{tr \in B \mid \forall i \in 1...n : \pi_i(tr) = tr_i\}$ be the set of behaviors resulting in $\vec{tr}$.

Definition 5.3 (Consistent specification) *A consistently specified system is a tuple* $(S, \mathcal{P})$ *where* $S = (C, \Sigma, B, \rho)$ *is a system signature with* $C = \{C_1, ..., C_n\}$ *and* $C_i = (\Sigma_i, \mathcal{S}_i)$*, and* $\mathcal{P} \subseteq B$ *is a prefix-closed property such that for all traces* $tr \in B$*,*

$$(\forall i \in 1...n : \pi_i(tr) \in \mathcal{S}_i) \implies tr \in \mathcal{P}$$

Under a consistent specification, property $\mathcal{P}$ may be violated only if at least one of the components violates its specification. Throughout this chapter we focus on consistent specifications.

[1]It is straight-forward to allow for additional information in traces $tr \in B$ that is not observable in the log, by adding to the cartesian product of Σ another alphabet that does not appear in the projections. For instance, events may be recorded with some timing uncertainty rather than precise time stamps [264].

5.3 Motivating Example

Consider a database system consisting of three components communicating by message passing over point-to-point FIFO buffers. Component C_1 is a client, C_2 the database server, and C_3 is a journaling system. The specifications of the three components are as follows:

$\mathcal{S}_1$: sends a lock request lock to C_2, followed by a request m to modify the locked data.

$\mathcal{S}_2$: receives a write request m, possibly preceded by a lock request lock. Access control is optimistic in the sense that the server accepts write requests without checking whether a lock request has been received before; however, in case of a missing lock request, a policy violation may be detected later on and signaled by an event x. After the write, a message journ is sent to C_3.

$\mathcal{S}_3$: keeps receiving journ events from C_2 for journaling, and acknowledges them with ok.

The system is modeled by the system signature (C, Σ, B, ρ) where $C = \{C_1, C_2, C_3\}$ with component signatures $C_i = (\Sigma_i, \mathcal{S}_i)$, and

- $\Sigma_1 = \{\mathsf{a}, \mathsf{m!}, \mathsf{lock!}\}$, $\Sigma_2 = \{\mathsf{m?}, \mathsf{journ!}, \mathsf{x}, \mathsf{lock?}\}$, and $\Sigma_3 = \{\mathsf{b}, \mathsf{journ?}\}$, where $\mathsf{m!}$ and $\mathsf{m?}$ stand for the emission and reception of a message m, respectively, and a, b, and x are internal events;
- $\mathcal{S}_1 = \{\mathsf{lock!.m!}\}$[2], $\mathcal{S}_2 = \{\mathsf{lock?.m?.journ!.ok?},\ \mathsf{m?.journ!.ok?.x}\}$, and $\mathcal{S}_3 = \{(\mathsf{journ?.ok!})^{\mathsf{i}} \mid \mathsf{i} \in \mathbb{N}\}$;
- $\Sigma = (\Sigma_1 \times \{\epsilon\} \times \{\epsilon\}) \cup (\{\epsilon\} \times \Sigma_2 \times \{\epsilon\}) \cup (\{\epsilon\} \times \{\epsilon\} \times \Sigma_3)$: component actions interleave;
- $B = \big\{w \in \Sigma^* \cup \Sigma^\omega \mid \forall u, v : w = u.v \implies (|u|_{\mathsf{m?}} \leq |u|_{\mathsf{m!}} \wedge |u|_{\mathsf{journ?}} \leq |u|_{\mathsf{journ!}} \wedge |u|_{\mathsf{lock?}} \leq |u|_{\mathsf{lock!}} \wedge w \text{ respects lossless FIFO semantics})\big\}$ (where $|u|_a$ stands for the number of occurrences of a in w): communication buffers are point-to-point FIFO queues;
- $\rho = \{(\mathsf{m!}, \mathsf{m?}), (\mathsf{journ!}, \mathsf{journ?}), (\mathsf{lock!}, \mathsf{lock?}), (\mathsf{ok!}, \mathsf{ok?})\}$: any component may influence another component's state only by sending a message that is received by the latter.

We are interested in the global safety property $\mathcal{P} = \Sigma_{ok}^* \cup \Sigma_{ok}^\omega$ with $\Sigma_{ok} = \Sigma \setminus \{(\epsilon, \mathsf{x}, \epsilon)\}$ modeling the absence of a conflict event x. It can be seen that if all three components satisfy their specifications, x will not occur.

Figure 5.1 shows the log $\vec{tr} = (tr_1, tr_2, tr_3)$. In the log, tr_1 violates $\mathcal{S}_1$ at event a and tr_3 violates $\mathcal{S}_3$ at b. The dashed lines between $\mathsf{m!}$ and $\mathsf{m?}$, and between $\mathsf{journ!}$ and $\mathsf{journ?}$ stand for communications.

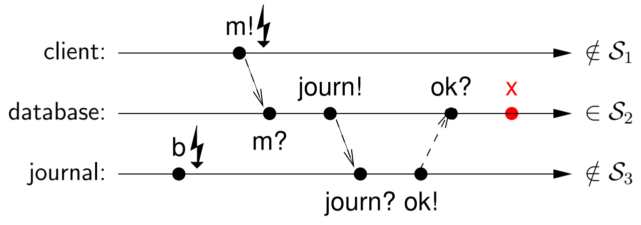


Figure 5.1: A scenario with three component logs.

In order to analyze which component(s) caused the violation of $\mathcal{P}$ we can use an approach based on *counterfactual reasoning.* Informally speaking,

[2]For the sake of readability we omit the prefix closure of the specifications in the examples.

- C_i is a *necessary cause* for the violation of $\mathcal{P}$ if in all executions where C_i behaves correctly and all other components behave as observed, $\mathcal{P}$ is satisfied.
- Conversely, C_i is a *sufficient cause* for the violation of $\mathcal{P}$ if in all executions where all incorrect traces of components other than C_i are replaced with correct traces, and the remaining traces (i.e., correct traces and the trace of C_i) are as observed, $\mathcal{P}$ is still violated.

Applying these criteria to our example we obtain the following results:

If C_1 had worked correctly, it would have produced the trace $tr'_1 = \mathsf{lock!}\ .\ \mathsf{m!}$. This gives us the counterfactual scenario consisting of the traces $\vec{tr}' = (tr'_1, tr_2, tr_3)$. However, this scenario is not consistent as C_1 now emits lock, which is not received by C_2 in tr_2. According to B, the FIFO buffers are not lossy, such that lock would have been received before m if it had been sent before m. By vacuity (as no execution yielding the traces $\vec{tr}'$ exists), C_1 is a necessary cause and C_3 is a sufficient cause according to our definitions above. While the first result matches our intuition, the second result is not what we would expect. As far as C_2 is concerned, it is not a cause since its trace satisfies $\mathcal{S}_2$.

Why do the above definitions fail to capture causality? It turns out that our definition of counterfactual scenarios is too narrow, as we substitute the behavior of one component (e.g., tr_1 to analyze sufficient causality of C_3) without taking into account the impact of the new trace on the remainder of the system. When analyzing causality "by hand", one would try to evaluate the effect of the altered behavior of the first component on the other components. This is what we will formalize in the next section.

5.4 Causality Analysis

In this section we improve our definition of causality of component traces for the violation of a system-level property. We suppose the following inputs to be available:

- A system signature (C, Σ) with components $C_i = (C_i, \Sigma_i)$.
- A log $\vec{tr} = (tr_1, ..., tr_n)$. In the case where the behavior of two or more components is logged into a common trace, the trace of each component can be obtained by projection.
- A set $\mathcal{I} \subseteq 1...n$ of component indices, indicating the set of components to be jointly analyzed for causality. Being able to reason about *group causality* is useful, for instance, to determine liability of software vendors that have provided several components.

5.4.1 Temporal Causality

As stated in the introduction, the temporal order of the events has an obvious impact on causality relations. We use Lamport's temporal causality [185] to over-approximate the parts of a log that are impacted by component failures. This technique will allow us, in the next section, to give counterfactual definitions of causality addressing the question of "what would have been the outcome if the failure of component C had not occurred?".

Given a trace $tr \in B$ let $tr_i = \pi_i(tr)$. The trace tr is analyzed as follows, for a fixed set $\mathcal{I}$ of components to be checked.

Definition 5.4 (Cone of influence, $\mathcal{C}(\vec{tr}, \mathcal{I})$) *Given a consistently specified system $(S, \mathcal{P})$ with $S = (C, \Sigma, B, \rho)$, $C = \{C_1, ..., C_n\}$, and $C_i = (\Sigma_i, \mathcal{S}_i)$, a log $\vec{tr} \in \mathcal{L}(S)$, and a set of* *component indices $\mathcal{I} \subseteq 1...n$, let $g_i : \mathbb{N} \to \{\bot, \top\}$ be a function associating with the length of each prefix of tr_i a value in $\{\bot, \top\}$ (with $\bot < \top$). Let $(g_1^*, ..., g_n^*)$ be the least fixpoint of*

$$g_i(\ell) = \begin{cases} \top & \textit{if } \big(\ell = \min\{k \mid tr_i[1..k] \notin \mathcal{S}_i\} \wedge i \in \mathcal{I}\big) \vee \\ & \quad \big(\exists k < \ell : g_i(k) = \top\big) \vee \\ & \quad \big(\exists tr' \in \vec{tr}^{\uparrow}\ \exists j, k, m, n : m \leq n \wedge k = |\pi_j(tr'[1..m])| \wedge \\ & \qquad \ell = |\pi_i(tr'[1..n])| \wedge g_j(k) = \top \wedge (tr'[m][j], tr'[n][i]) \in \rho \wedge \\ & \qquad tr_i[1..\ell - 1] \in \mathcal{S}_i\big) \\ \bot & \textit{otherwise} \end{cases}$$

for $i \in 1...n$ and $1 \leq \ell \leq |tr_i|$. Let $\mathcal{C}(\vec{tr}, \mathcal{I}) = (c_1, ..., c_n)$ such that

$$\forall i \in 1...n : c_i = \min\big(\{|tr_i| + 1\} \cup \{\ell \mid g_i^*(\ell) = \top\}\big)$$

The cone of influence *spanned by the components $\mathcal{I}$ is the vector of suffixes $tr_i[c_i...]$ of the component traces.*

That is, as soon as a component $i \in \mathcal{I}$ violates $\mathcal{S}_i$ on a prefix $tr_i[1..\ell]$, g_i is set to $\top$ (first line). Once $g_i(k) = \top$, it remains $\top$ for all larger indices (second line). Each time a component i participates in an interaction $\beta = tr'[n]$ for some possible trace tr' on which another component j has previously participated in an interaction $\alpha = tr'[m]$ after a prefix of length k such that $g_j(k) = \top$ and $(\alpha[j], \beta[i]) \in \rho$, then g_i is set to $\top$, provided that the prefix of tr_i satisfied $\mathcal{S}_i$ before (third line). The last condition $tr_i[1..\ell - 1] \in \mathcal{S}_i$ means that a possibly incorrect behavior of C_i following an endogenous violation of $\mathcal{S}_i$ is blamed on C_i rather than on the components in $\mathcal{I}$.

The cone of influence spanned by the components $\mathcal{I}$ is the vector of suffixes of the component traces starting with the first component action that may have been impacted by the behavior of the components $\mathcal{I}$ starting in one of their failures. For the sake of simplicity we will refer to $\mathcal{C}(\vec{tr}, \mathcal{I})$ as the cone.

Example 5.1 *Figure 5.2 shows the cones $\mathcal{C}(\vec{tr}, \{1\}) = (1, 1, 3)$ and $\mathcal{C}(\vec{tr}, \{2, 3\}) = (3, 4, 1)$ for the example of Section 5.3 and Figure 5.1.*

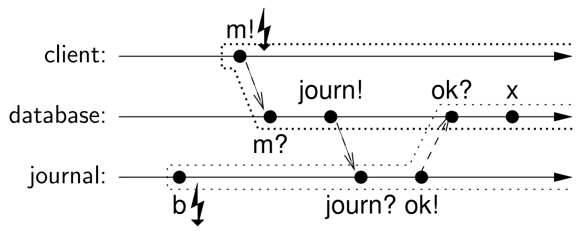


Figure 5.2: The scenario with the cones $\mathcal{C}(\vec{tr}, \{1\})$ and $\mathcal{C}(\vec{tr}, \{2, 3\})$, respectively.

5.4.2 Logical Causality

Using the cone of influence defined above we are able to define, for a given log $\vec{tr}$ and set of component indices $\mathcal{I}$, the set of *counterfactual traces* modeling *alternative worlds* in which the failures $F_{\mathcal{I}}$ of components in $\mathcal{I}$ do not happen, and the behavior of the remaining components is as observed in $\vec{tr}$ up to the part lying inside the cone spanned by $F_{\mathcal{I}}$.

Definition 5.5 (Counterfactuals) *Let $\vec{tr} = (tr_1, ..., tr_n) \in \mathcal{L}$, $\vec{\mathcal{C}} = (c_1, ..., c_n)$ be a cone of influence, and $\vec{\mathcal{S}} = (\mathcal{S}_1, ..., \mathcal{S}_n)$.*

$$\begin{aligned}
\sigma\big(\vec{tr}, \vec{\mathcal{C}}, \vec{\mathcal{S}}\big) = \big\{tr' \in B \mid \forall i : pr_i \textit{ is a prefix of } \pi_i(tr') \wedge \quad & (5.1)\\
(pr_i \in \mathcal{S}_i \implies \pi_i(tr') \in \mathcal{S}_i) \wedge \quad & (5.2)\\
(pr_i \notin \mathcal{S}_i \implies \pi_i(tr') = pr_i) \wedge \quad & (5.3)\\
(c_i = |tr_i| + 1 \implies \pi_i(tr') = pr_i)\big\} \quad & (5.4)
\end{aligned}$$

where $pr_i = tr_i[1..c_i - 1]$.

Intuitively, σ returns the set of alternative behaviors $tr' \in B$ where for each component i, the prefix pr_i before entering c_i matches its logged behavior in tr_i (line 5.1), and if the prefix is correct and a strict prefix of tr_i then the suffix is substituted such that the whole behavior of i in trace tr' is correct (line 5.2); otherwise pr_i is not extended in the alternative behavior (lines 5.3 and 5.4). The rationale behind Definition 5.5 is to compute the set of *alternative worlds* where the failures spanning $\mathcal{C}$ do not occur. To this end we have to prune out their possible impact on the logged behavior, and substitute with correct behaviors. Prefixes violating their specifications (line 5.3) and component traces that never enter the cone (line 5.4) are not extended since we want to determine causes for system-level failures observed in the log, rather than exhibiting causality chains that are not complete yet and whose consequence would have shown only in the future.

Definition 5.6 (Necessary cause) *Given*

- *a consistently specified system $(S, \mathcal{P})$ with $S = (C, \Sigma, B, \rho)$, $C = \{C_1, ..., C_n\}$, and $C_i = (\Sigma_i, \mathcal{S}_i)$,*
- *a log $\vec{tr} \in \mathcal{L}$ such that $\vec{tr}^{\uparrow} \cap \mathcal{P} = \emptyset$, and*
- *an index set $\mathcal{I}$,*

let $\vec{\mathcal{C}} = \mathcal{C}(\vec{tr}, \mathcal{I})$. The set of traces indexed by $\mathcal{I}$ is a necessary cause for the violation of $\mathcal{P}$ by $\vec{tr}$ if $\sigma(\vec{tr}, \vec{\mathcal{C}}, \vec{\mathcal{S}}) \subseteq \mathcal{P}$.

That is, the set of logs indexed by $\mathcal{I}$ is a necessary cause for the violation of $\mathcal{P}$ if in the observed behavior where the cone spanned by the incorrect behaviors of $\mathcal{I}$ is replaced by a correct behavior, $\mathcal{P}$ is satisfied. In other words, if the components in $\mathcal{I}$ had satisfied their specifications, and all components had behaved as in the logs before entering the cone, then $\mathcal{P}$ would have been satisfied.

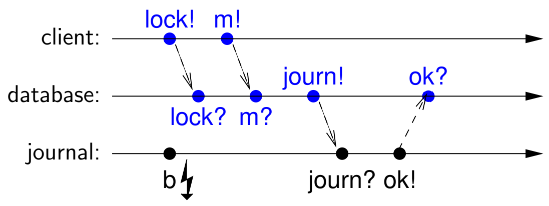
According to the construction of the cone of influence, this definition of necessary causality makes the assumption that the violation of a component specification $\mathcal{S}_j$ within the cone of other components $\mathcal{I}$, $j \notin \mathcal{I}$, cannot be blamed for certain on component j.

Example 5.2 *Coming back to Example 5.1, let $\vec{\mathcal{C}} = \mathcal{C}\big(\vec{tr}, \{1\}\big)$. We have $\sigma(\vec{tr}, \vec{\mathcal{C}}, \vec{\mathcal{S}}) = \mathcal{S}_1 \| \mathcal{S}_2 \| \{tr_3\}$, as shown in Figure 5.3(a). According to Definition 5.6, tr_1 is a necessary cause for the violation of $\mathcal{P}$ since $\mathcal{P}$ is satisfied in $\sigma(\vec{tr}, \vec{\mathcal{C}}, \vec{\mathcal{S}})$. It can be shown that tr_3 is not a necessary cause.*

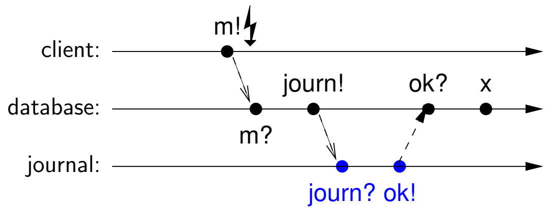
The definition of *sufficient causality* is dual to necessary causality, where in the alternative worlds we remove the failures of components *not in $\mathcal{I}$* and verify whether $\mathcal{P}$ is *still violated.*

For a set of traces S, let $\max S = \{s \in S \mid \forall t \in S : s \text{ is not a strict prefix of } t\}$.

Definition 5.7 (Sufficient cause) *Given*



(a) $\sigma(\vec{tr},\vec{\mathcal{C}},\vec{\mathcal{S}})$ with $\vec{\mathcal{C}} = \mathcal{C}(\vec{tr},\{1\})$



(b) $\sigma(\vec{tr},\vec{\mathcal{C}},\vec{\mathcal{S}})$ with $\vec{\mathcal{C}} = \mathcal{C}(\vec{tr},\{2,3\})$

Figure 5.3: The scenario where the cone (a) $\mathcal{C}(\vec{tr},\{1\})$ and (b) $\mathcal{C}(\vec{tr},\{2,3\})$ is substituted with suffixes satisfying the component specifications.

- *a consistently specified system* $(S,\mathcal{P})$ *with* $S = (C,\Sigma,B,\rho)$, $C = \{C_1,...,C_n\}$, *and* $C_i = (\Sigma_i,\mathcal{S}_i)$,
- *a log* $\vec{tr} \in \mathcal{L}$ *with* $\vec{tr}^{\uparrow} \cap \mathcal{P} = \emptyset$, *and*
- *an index set* $\mathcal{I}$,

let $\overline{\mathcal{I}} = 1...n \setminus \mathcal{I}$ *and* $\vec{\mathcal{C}} = \mathcal{C}(\vec{tr},\overline{\mathcal{I}})$. *The set of traces indexed by* $\mathcal{I}$ *is a sufficient cause for the violation of* $\mathcal{P}$ *by* $\vec{tr}$ *if*

$$\left(\max\sigma(\vec{tr},\vec{\mathcal{C}},\vec{\mathcal{S}})\right) \cap \mathcal{P} = \emptyset$$

That is, the set of logs indexed by $\mathcal{I}$ is a sufficient cause for the violation of $\mathcal{P}$ if in the observed behavior where the cone spanned by the violations of specifications by the complement of $\mathcal{I}$ is replaced by a correct behavior, the violation of $\mathcal{P}$ is inevitable (even though $\mathcal{P}$ may still be satisfied for non-maximal counterfactual traces). In other words, even if the components in the complement $\overline{\mathcal{I}}$ of $\mathcal{I}$ had satisfied their specifications and no component had failed in the cone spanned by the failures of $\overline{\mathcal{I}}$, then $\mathcal{P}$ would still have been violated. The inclusion of infinite traces in the behavioral model B (Definition 5.2) ensures the least upper bound of the set of counterfactual traces to be included in B.

In Definitions 5.6 and 5.7 the use of temporal causality helps in constructing alternative scenarios in B where the components indexed by $\mathcal{I}$ (resp. $\overline{\mathcal{I}}$) behave correctly while keeping the behaviors of all other components close to their observed behaviors.

Example 5.3 *In Example 5.2 let* $\vec{\mathcal{C}} = \mathcal{C}(\vec{tr},\{2,3\})$. *We obtain* $\sigma(\vec{tr},\vec{\mathcal{C}},\vec{\mathcal{S}}) = \{tr_1\}\|\{tr_2\}\|\mathcal{S}_3$, *as shown in Figure 5.3(b). By Definition 5.7,* tr_1 *is a sufficient cause for the violation of* $\mathcal{P}$ *since* $\mathcal{P}$ *is still violated in* $\sigma(\vec{tr},\vec{\mathcal{C}},\vec{\mathcal{S}})$. *It can be shown that* tr_3 *is not a sufficient cause.*

Properties.

The following results show that our analysis does not blame any set of innocent components, and that it finds a necessary and a sufficient cause for every system-level failure.

Theorem 5.1 (Soundness) *Each cause contains an incorrect trace.*

Theorem 5.2 (Completeness) *Each violation of $\mathcal{P}$ has a necessary and a sufficient cause.*

5.5 Application to BIP

In [147] we have presented a preliminary approach to causality analysis on a BIP component system $S = \|_{IM}\{K_i \mid K_i \in K\}$ where $K_i = (B_i, IM[K_i], true)$. A conservative approximation was discussed for applications where only incomplete traces with bounded past are available. We proposed a symbolic implementation, and applied the analysis to the model of an adaptive cruise control system.

5.6 Application to Synchronous Data Flow

In this section we use the general framework to model a synchronous data flow example, and illustrate a set of well-known phenomena studied in the literature.

Consider a simple filter that propagates, at each clock tick, the input when it is stable in the sense that it has not changed since the last tick, and holds the output when the input is unstable. Using LUSTRE [161]-like syntax the filter can be written as follows:

$$\begin{aligned} change &= false \rightarrow in \neq pre(in) \\ h &= pre(out) \\ out &= \begin{cases} in & \text{if } \neg change \\ h & \text{otherwise} \end{cases} \end{aligned}$$

That is, component *change* is initially *false*, and subsequently *true* if and only if the input *in* has changed between the last and the current tick. *h* latches the previous value of *out*; its value is $\perp$ ("undefined") at the first instant. *out* is equal to the input if *change* is false, and equal to *h* otherwise. Thus, each signal consists of an infinite sequence of values, e.g., $change = \langle change_1, change_2, ...\rangle$.

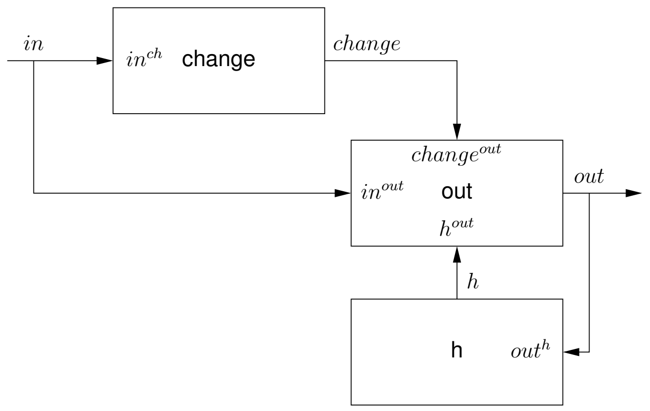


Figure 5.4: Architecture of the filter.

Figure 5.4 visualizes the architecture and signal names. We formalize the system as follows.

- $\Sigma_{ch} = \mathbb{R} \times \mathbb{B} \times \mathbb{N} \times \{\mathsf{ch}\}$ where the first two components stand for the value of the input to and output from *change*, the third component is the index of the clock tick, and ch is a tag we will use to distinguish the alphabets of different components. Similarly, let $\Sigma_h = \mathbb{R} \times (\mathbb{R} \cup \{\perp\}) \times \mathbb{N} \times \{\mathsf{h}\}$ and $\Sigma_{out} = \mathbb{R} \times \mathbb{R} \times \mathbb{B} \times \mathbb{R} \times \mathbb{N} \times \{\mathsf{out}\}$.

- $\mathcal{S}_{ch} = \{(r_1, r_2, ...) \in \Sigma^*_{ch} \mid r_i = (in_i, change_i, i, \mathsf{ch}) \wedge change_1 = \mathit{false} \wedge (i \geq 2 \implies change_i = in_{i-1} \neq in_i)\}$ is the specification of *change*. Similarly, $\mathcal{S}_h = \{(r_1, r_2, ...) \in \Sigma^*_h \mid r_i = (out_i, h_i, i, \mathsf{h}) \wedge (i \geq 2 \implies h_i = out_{i-1})\}$ and
$$\mathcal{S}_{out} = \Big\{(r_1, r_2, ...) \in \Sigma^*_{out} \mid r_i = (in_i, h_i, change_i, out_i, i, \mathsf{out}) \wedge \\ out_i = \begin{cases} in_i & \text{if } \neg change_i \\ h_i & \text{otherwise} \end{cases} \Big\}$$
- $\Sigma = \{(r_{ch}, r_h, r_{out}) \in \Sigma_{ch} \times \Sigma_h \times \Sigma_{out} \mid r_{ch} = (in^{ch}, change, i_1, \mathsf{ch}) \wedge r_h = (out^h, h, i_2, \mathsf{h}) \wedge r_{out} = (in^{out}, h^{out}, ch^{out}, out, i_3, \mathsf{out}) \mid i_1 = i_2 = i_3\}$ is the system alphabet (where all components react synchronously).
- $B = \{(r_1, r_2, ...) \in \Sigma^* \cup \Sigma^\omega \mid \forall i : r_i = \big((in^{ch}_i, change_i, i_1, \mathsf{ch}), (out^h_i, h_i, i_2, \mathsf{h}), (in^{out}_i, h^{out}_i, ch^{out}_i, out_i, i_3, \mathsf{out})\big) \wedge in^{ch}_i = in^{out}_i \wedge change_i = ch^{out}_i \wedge out^h_i = out_i \wedge h_i = h^{out}_i\}$ is the set of possible behaviors, meaning that connected flows are equal.
- $\rho = \big\{\big((\cdot, \cdot, i, \mathsf{in}), (\cdot, \cdot, i, \mathsf{ch})\big),\ \big((\cdot, \cdot, i, \mathsf{in}), (\cdot, \cdot, i, \mathsf{out})\big),\ \big((\cdot, \cdot, i, \mathsf{ch}), (\cdot, \cdot, i, \mathsf{out})\big),\ \big((\cdot, \cdot, i, \mathsf{h}), (\cdot, \cdot, i, \mathsf{out})\big),\ \big((\cdot, \cdot, i, \mathsf{out}), (\cdot, \cdot, i+1, \mathsf{h})\big) \mid i \geq 1\big\}$ models the data dependencies.
- $\mathcal{P} = \{(r_1, r_2, ...) \in B \mid \forall i : r_i = \big(..., (..., out_i, ...)\big) \wedge out_i = out_{i+1} \vee out_{i+1} = out_{i+2}\}$ is the stability property, meaning that there are no two consecutive changes in output.

A log of a valid execution is for instance (where connected signals only appear once and the tick number is omitted):

in	0	0	3	2	2
change	*false*	*false*	*true*	*true*	*false*
h	$\perp$	0	0	0	0
out	0	0	0	0	2

Figure 5.5 shows four logs of faulty executions.

in	0	0	1	2
change	*false*	*false*	$\underline{\textit{false}}$	$\underline{\textit{false}}$
h	$\perp$	0	$\underline{-1}$	$\underline{-3}$
out	0	0	1	2

(a) $\vec{tr}^1$: early preemption.

in	0	0	0	0
change	*false*	*false*	$\underline{\textit{true}}$	$\underline{\textit{true}}$
h	$\perp$	0	$\underline{-1}$	$\underline{1}$
out	0	0	-1	1

(b) $\vec{tr}^2$: joint causation.

Figure 5.5: Two logs of faulty executions.

Consider Figure 5.5a. Two components violate their specifications (incorrect values are underlined): *change* and *h*, both at the third instant. The stability property $\mathcal{P}$ is violated at the fourth output. Let us apply our definitions to analyze causality of each of the two faulty components.

1. In order to check whether *change* is a necessary cause, we first compute the cone spanned by the violation by *change* as $\mathcal{C}(\vec{tr}^1, \{change\}) = (3, 5, 3)$. Thus, the prefixes of the component traces before entering the cone are as shown in Figure 5.6a. Next we compute the set of counterfactuals, according to Definition 5.5, as $(\vec{tr}')^\uparrow$, where $\vec{tr}'$ is shown in Figure 5.6b. $\mathcal{P}$ is still violated by the (unique) counterfactual trace, hence *change* is not a necessary cause.

in	0	0	1	2
change	*false*	*false*		
h	$\perp$	0	$\underline{-1}$	$\underline{-3}$
out	0	0		

(a) $\vec{tr}^1$ after removing $\mathcal{C}(\vec{tr}^1, \{change\})$.

in	0	0	1	2
change	*false*	*false*	*true*	*true*
h	$\perp$	0	$\underline{-1}$	$\underline{-3}$
out	0	0	-1	-3

(b) $\vec{tr}'$ such that $(\vec{tr}')^\uparrow = \sigma(\vec{tr}^1, \vec{\mathcal{C}}, \vec{\mathcal{S}})$

Figure 5.6: Computing necessary causality of *change* for the violation of $\mathcal{P}$ in $\vec{tr}^1$.

We can show, using the same construction, that h is a sufficient cause for the violation of $\mathcal{P}$.

2. In order to check whether *change* is a sufficient cause, we first compute the cone spanned by the violation by h as $\mathcal{C}(\vec{tr}^1, \{h\}) = (5, 3, 3)$. That is, the cone encompasses the last two values of h and *out*. Due to *change* being (incorrectly) *false*, the only possible counterfactual trace according to Definition 5.5 is $\sigma(\vec{tr}^1, \vec{\mathcal{C}}, \vec{\mathcal{S}}) = (tr_{change}, tr'_h, tr_{out})^\uparrow$ where tr_{change} is as observed in $\vec{tr}^2$, $tr'_h = (\perp, 0, 0, 1)$, and $tr'_{out} = (0, 0, 1, 2)$. $\mathcal{P}$ is still violated by the unique counterfactual trace, hence *change* is a sufficient cause.

   We can show, using the same construction, that h is not a necessary cause for the violation of $\mathcal{P}$.

The example of log $\vec{tr}^1$ shows two phenomena called *over-determination* (there are two sufficient causes, one of which would have sufficed to violate $\mathcal{P}$) and *early preemption*: the causal chain from the violation of $\mathcal{S}_h$ to the violation of $\mathcal{P}$ is interrupted by the causal chain from the violation of $\mathcal{S}_{change}$ to the violation of $\mathcal{P}$, since due to *change* being *false*, the incorrect value of h is discarded in the computation of *out* in log $\vec{tr}^1$.

Figure 5.5b shows a case of *joint causation*: both *change* and h are necessary causes for the violation of $\mathcal{P}$ in $\vec{tr}^2$, but none of them alone is a sufficient cause.

5.7 Related Work

Causality has been studied for a long time in different disciplines (philosophy, mathematical logic, physics, law, etc.) before receiving an increasing attention in computer science during the last decade. Hume discusses definitions of causality in [169]:

> Suitably to this experience, therefore, we may define a cause to be an object, followed by another, and where all the objects similar to the first are followed by objects similar to the second. Or in other words where, if the first object had not been, the second never had existed.

In computer science, various approaches to causality analysis have been developed recently. They differ in their assumptions on what pieces of information are available for causality analysis: a model of causal dependencies, a program as a black-box that can be used to replay different scenarios, the observed actual behavior (e.g. execution traces, or inputs and outputs), and/or the expected behavior (that is, component specifications). Existing frameworks consider different subsets of these entities. We cite the most significant settings and approaches for these settings.

A specification and an observation. In the preliminary work of [147], causality of components for the violation of a system-level property under the BIP interaction model [154, 22] has been defined using a rudimentary definition of counterfactuals where only faulty traces are substituted but not the parts of other component traces impacted by the former. An application in the context of legal contracts has been discussed in [146]. However, the definition of [147] suffered from the conditions for causality being true by vacuity when no consistent counterfactuals exist. A slightly improved approach is used in [263] for causality analysis in real-time systems.

With a similar aim of independence from a specific model of computation as in our work, [234] formalizes a theory of diagnosis in first-order logic. A *diagnosis* for an observed incorrect behavior is essentially defined as a minimal set of components whose failure explains the observation.

A causal model. [162] proposes what has become the most influential definition of causality for computer science so far, based on a model over a set of propositional variables partitioned into *exogenous* variables $\mathcal{U}$ and *endogenous* variables $\mathcal{V}$. A function $\mathcal{F}_X$ associated with each variable $X \in \mathcal{V}$ uniquely determines the value of X depending on the value of all variables in $(\mathcal{U} \cup \mathcal{V}) \setminus \{X\}$. These functions define a set of *structural equations* relating the values of the variables. The equations are required to be *recursive*, that is, the dependencies form an acyclic graph whose nodes are the variables. The observed values of a set X of variables is an *actual cause* for an observed property φ if with different values of X, φ would not hold, and there exists a context (a *contingency*) in which the observed values of X entail φ. With the objective of better representing causality in processes evolving over time, *CP-logic* defines actual causation based on probability trees [31].

In [171], fault localization and repair in a circuit with respect to an LTL property are formulated as a game between the environment choosing inputs and the system choosing a fix for a faulty component.

A model and a trace. In several applications of Halpern and Pearl's SEM, the model is used to encode and analyze one or more execution traces, rather than a behavioral model.

The definition of actual cause from [162] is used in [32] to determine potential causes for the first violation of an LTL formula by a trace. As [162] only considers a propositional setting without any temporal connectors, the trace is modeled as a matrix of propositional variables. In order to make the approach feasible in practice, an over-approximation is proposed. In this approach, the structure of the LTL formula is used as a model to determine which events may have caused the violation of the property.

Given a counter-example in model-checking, [159] uses a distance metric to determine a cause of the property violation as the difference between the error trace and a closest correct trace.

An approach to fault localization in a sequential circuit with respect to a safety specification in LTL is presented in [115]: given a counter-example trace, a propositional formula is generated that holds if a different behavior of a subset of gates entails the satisfaction of the specification.

A set of traces. [180] extends the definition of actual causality of [162] to totally ordered sequences of events, and uses this definition to construct from a set of traces a fault tree. Using a probabilistic model, the fault tree is annotated with probabilities. The accuracy of the diagnostic depends on the number of traces used to construct the model. An approach for on-the-fly causality checking is presented in [198].

An input and a black box. Delta debugging [271] is an efficient technique for automatically isolating a cause of some error. Starting from a failing input and a passing input, delta debugging finds a pair of a failing and a passing input with minimal distance. The approach is syntactical and has been applied to program code, configuration files, and context switching in schedules. By applying delta debugging to *program states* represented as *memory graphs*, analysis has been further refined to program semantics. Delta debugging isolates failure-inducing causes in the *input* of a program, and thus requires the program to be available.

5.8 Conclusion

We have presented a general approach for causality analysis of system failures based on component specifications and observed component traces. Applications include identification of faulty components in black-box testing, recovery of critical systems at runtime, and determination of the liability of component providers in the aftermath of a system failure.

Future work. This article opens a number of directions for future work. First of all, we will instantiate and implement the framework for specific models of computation and communication, such as Timed Automata [8] and functional programs. The tagged signal model [196] provides a formal basis for representing such models in our framework. In order to make the definitions of causality effectively verifiable, we will reformulate them as operations on symbolic models, and use efficient data structures — such as the event structures used in [109] for distributed diagnosis — to represent and manipulate logs.

At design time, the code of the components can be instrumented so as to log relevant information for analyzing causality with respect to a set of properties to be monitored. For instance, precise information on the actual (partial) order of execution can be preserved by tagging the logged events with vector clocks [117, 207]. More generally, we intend to investigate the aspect of ensuring *accountability* [181] by design in future work. Similar to the approach of [48] to derive logging requirements from a privacy policy that produce minimal but sufficient logs for auditing the policy, an interesting work direction will be to study how to automatically determine from the system signature, a minimal logging requirement for blaming.

A second extension we will investigate is the distinction between observable and non-observable events, only the former of which are logged. In particular, whenever component failures are not observable, we have to combine fault diagnosis [241] and causality analysis.

In this chapter we assume only the logs to be available. However, in some situations such as post-mortem analysis the (black-box) components may be available, in which case counterfactual scenarios could be replayed on the system to evaluate their outcome more precisely. In the same vein, an alternative behavior of the control part of a closed-loop systems is likely to impact the physical process, as in our cruise control example: a counterfactual trace with different brake or throttle control will impact the speed of the car. This change should be propagated through a model of the physical process to make the counterfactual scenario as realistic as possible.

Chapter 6

Symbolic Abstractions of Hybrid Systems

6.1 Modeling and Analysis of Genetic Networks

In this section we show how to leverage the design principles of BIP for the component-based modeling of genetic networks so as to construct and analyze complex behaviors of interacting sub-systems. In particular, we propose a goal-directed simulation algorithm for high-dimensional biological models. The approach is applied to a well-known signaling network. Finally, we use formal verification for parameter estimation of a genetic regulatory network so as to fit an experimentally observed or desired behavior.

6.1.1 Motivation

In bioinformatics, the study of the cellular functions on the molecular level has been strongly stimulated by the development of high-throughput experimental methods. In addition to such experimental methods, both mathematical and computer science tools are necessary for the analysis of genetic and metabolic networks. More and more, models and experiments are used jointly to better understand the different cellular processes. This trend has led to the emergence of a new domain called *systems biology*, aiming at a system-level understanding of cells [178]. This focus shift from the study of narrow aspects of cellular behavior towards an integrated view is creating a strong need to jointly model and analyze different, interacting fragments of behavior. A methodology allowing to break down the complexity of model construction, composition, and validation in a formal framework, is of strategic interest for systems biology.

This case study shows and explores how to leverage the design principles of BIP for the component-based construction of biological models, so as to construct and analyze complex behaviors of interacting sub-systems. Furthermore, we study goal-directed simulation of high-dimensional biological models. The technique of [139, 141] allows us to guide simulation through huge state spaces.

Finally, we use formal verification for parameter estimation of a genetic regulatory network so as to fit an experimentally observed or desired behavior.

6.1.2 Modeling and Compositional Reachability Analysis

Introduction

Genetic regulatory networks usually encompass a multitude of complex, interacting feedback loops. Being able to model and analyze their behavior is crucial for understanding the interactions between the genes, and their functions. Genetic regulatory networks have been modeled as discrete transition systems by many approaches — see e.g. [256, 99] —, benefiting from a large number of formal verification algorithms available for the analysis of discrete transition systems. However, most of these approaches face the problem of state space explosion, as even models of modest size (from a biological point of view) usually lead to large transition systems, due to a combinatorial blow-up of the number of states. Even if the modeling formalism allows for a structured representation of the state space such as Petri nets [233], subsequent analysis algorithms have to be able to exploit this information. In practice, monolithic approaches for the analysis of genetic regulatory networks do not scale.

In order to deal with the problem of state space explosion, different techniques have been developed in the formal verification community, such as partial order reduction, abstraction, and compositional approaches. In this work, we explore the use of compositionality for the analysis of genetic regulatory networks. Compositional analysis means that the behavior of a system consisting of interacting components is analyzed by separately examining the behavior of each component within an abstraction of its environment, rather than by monolithically analyzing the behavior of the overall system. It can therefore be more efficient than non-compositional analysis, and scale better. As cellular functions are often composed of functional modules [164, 266, 236], compositional analysis may take advantage of this modular structure of genetic regulatory networks.

A precondition for compositional algorithms to be applicable, is that the model be structured. Therefore, this work makes two contributions: first, we present a modeling framework for genetic regulatory networks in which the different components of the system (in our case, protein concentrations) and the way they constrain each other, are modeled separately and modularly. Second, we propose a compositional algorithm allowing us to efficiently analyze reachability properties of the model.

The approach presented here [139, 141] is based on qualitative simulation [101]. Given a system of piecewise linear differential equations, qualitative simulation abstracts the continuous dynamics into a set of qualitative states and discrete transitions between them. Each qualitative state corresponds either to a *regulatory domain*, in all points of which the system obeys the same linear differential equation, or to a *switching domain* on the frontier between two or more regulatory domains, where the values of one or more variables take some threshold.

This work introduces a new, modular discrete abstraction of piecewise linear differential equations. It extends and generalizes the preliminary, short version of [139]. New contributions include a generalization to switching domains of arbitrary order — where the order of a domain is defined as the number of variables taking threshold values —, and new correctness and completeness results. Representing the full state space including higher-order switching domains is important for two reasons: first, reachability may depend on the existence of paths crossing higher-order switching domains, and second, biologically relevant states like equilibria are often located on higher-order switching domains.

Related work. By now there is a large number of approaches to model and analyze genetic networks. An overview can be found in the survey of [99]. The modeling approaches adopt different mathematical frameworks, which vary in expressiveness and the availability and efficiency of verification algorithms.

Reachability properties can be efficiently approximated even for high-dimensional linear and hybrid systems [129]. However, in order to faithfully model the dynamics of the system, the exact numerical values of its kinetic parameters have to be known. This is only the case for few well-studied genetic networks. Therefore, many approaches adopt frameworks based on hybrid systems or piecewise linear differential equations and use discrete abstractions representing the qualitative behavior of the model. [126] uses predicate abstraction to automatically compute backward reachable sets of piecewise affine hybrid automata. [128] presents an algorithm for reachability analysis of piecewise affine hybrid automata based on iterated refinement of discrete abstractions. [222] addresses the bounded reachability problem of hybrid automata through algebraic decomposition of the state space, using an approximation based on Taylor polynomials. [126, 128, 222] tackle models of the *Delta-Notch* signaling network encompassing up to four cells. [93] introduces dynamic hybridization to compute abstractions over-approximating the reachable states of nonlinear models. Compositional reachability analysis of discrete models using automatic generation of context constraints — that is, abstractions of sub-systems — has been studied in [77, 78].

In contrast to these frameworks working on general piecewise linear or hybrid systems, some approaches take advantage of the specific form of piecewise linear differential equations modeling genetic regulatory networks, see [134] as an early example. The approach of [101] defines a conservative qualitative abstraction of a piecewise linear system by using differential inclusions [118, 156].

In order to deal with complex networks, some approaches choose to directly model genetic networks in a discrete framework, such as the influential early work of [255] and more recently, modeling approaches based on logical equations [256], Petri nets [73, 89], Reo connectors [83], or multi-valued decision diagrams [218]. Formal verification can then be carried out enumeratively (for instance, using the tools [100, 26, 136]) or symbolically, see for example [72, 43]. Except for [83], and [73] using results from Petri net theory to establish properties of the model from the structure of the Petri net, these approaches "flatten" the model and work on the global state space, without computationally taking advantage of the modularity of the problem.

Only recently there has been an increasing focus on abstraction techniques helping to tame the complexity of biological models by introducing some degree of compositional reasoning. [110] examines the use of abstract interpretation for several modeling frameworks in systems biology. [44] defines an exact condition for the behavior of a genetic network to be preserved when it is embedded in a larger network, aimed at modular analysis of networks.

Stochastic approaches are known to be an efficient and, in presence of only few reacting molecules, more faithful alternative to continuous frameworks. Several techniques such as model reduction by numerical aggregation [62] and structured representation by stochastic automata networks [267] have been proposed to circumvent state space explosion of numerical analysis.

Component-based Modeling of Genetic Networks

This section briefly introduces the notions of piecewise linear system, and its qualitative simulation as defined in [101]. We then define a modularized approximation of qualitative simulation, and compare both frameworks.

Piecewise linear systems. The production of a protein in a cell is regulated by the current protein concentrations, which can activate or inhibit the production, for instance by binding to the gene and disabling transcription. At the same time, proteins are degraded. This behavior of a genetic network can be modeled by a system of differential equations of the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) - \mathbf{g}(\mathbf{x}, \mathbf{u})\mathbf{x} \tag{6.1}$$

where $\mathbf{x}$ is a vector of protein concentrations representing the current state, $\mathbf{u}$ is a vector of input concentrations, and the vector-valued function $\mathbf{f}$ and matrix-valued function $\mathbf{g}$ model the production rates, and degradation rates, respectively.

The approach of [101] considers an abstraction where the state space of each variable x_i is partitioned into a set of intervals $\mathcal{D}_i^r$ and a set of p_i *threshold values* $\mathcal{D}_i^s$. This induces a partition of the continuous state space into a discrete set of *domains*, in each of which Equation (6.1) is approximated with a system of linear differential equations.

Definition 6.1 (Domain) *Consider a Cartesian product* $\theta = \theta_1 \times ... \times \theta_n$ *with* $\theta_i = \{\theta_i^1, ..., \theta_i^{p_i}\}$ *an ordered set of* thresholds *such that* $0 < \theta_i^1 < ... < \theta_i^{p_i} < max_i$*. Let*

$$\mathcal{D}_i^r(\theta) = \{[0, \theta_i^1)\} \cup \{(\theta_i^j, \theta_i^{j+1}) \mid 1 \leq j < p_i\} \cup \{(\theta_i^{p_i}, max_i]\}$$

and $\mathcal{D}_i^s(\theta) = \left\{\{\theta_i^j\} \mid 1 \leq j \leq p_i\right\}$*. We omit the argument* θ *when it is clear from the context. Let* $\mathcal{D}_i = \mathcal{D}_i^r \cup \mathcal{D}_i^s$*, and* $\mathcal{D} = \mathcal{D}_1 \times \mathcal{D}_2 \times ... \times \mathcal{D}_n$ *be the set of* domains*. The domains in* $\mathcal{D}^r = \mathcal{D}_1^r \times \mathcal{D}_2^r \times ... \times \mathcal{D}_n^r$ *are called* regulatory *domains, the domains* $\mathcal{D}^s = \mathcal{D} \setminus \mathcal{D}^r$ *are called* switching domains*.*

The state space $[0, max_1] \times \cdots \times [0, max_n]$ is thus partitioned into the set of domains $\mathcal{D}$.

Definition 6.2 (Piecewise linear system) *A* piecewise linear system *is a tuple* $M = (X, \theta, \boldsymbol{\mu}, \boldsymbol{\nu})$ *where*

- $X = \{x_1, ..., x_n\}$ *a set of real-valued* state variables*;*
- $\theta = \theta_1 \times ... \times \theta_n$*, with* $\theta_i = \{\theta_i^1, ..., \theta_i^{p_i}\}$ *such that* $0 < \theta_i^1 < ... < \theta_i^{p_i} < max_i$*, associates with each dimension* i *an ordered set of* p_i thresholds*;*
- $\boldsymbol{\mu} : \mathcal{D}^r(\theta) \to \mathbb{R}^n_{\geq 0}$ *associates with each regulatory domain a vector of* production rates*;*
- $\boldsymbol{\nu} : \mathcal{D}^r(\theta) \to diag(\mathbb{R}^n_{>0})$ *associates with each regulatory domain a diagonal matrix of* degradation rates*.*

Within a regulatory domain $D \in \mathcal{D}^r$, the protein concentrations $\mathbf{x}$ evolve according to the ratio of production rate and degradation rate:

$$\dot{\mathbf{x}} = \boldsymbol{\mu}(D) - \boldsymbol{\nu}(D)\mathbf{x} \tag{6.2}$$

and thus converge monotonically towards the *focal point* $\boldsymbol{\phi}$, solution of $\mathbf{0} = \boldsymbol{\mu}(D) - \boldsymbol{\nu}(D)\mathbf{x}$.

Definition 6.3 ($\boldsymbol{\phi}$) *For any* $D \in \mathcal{D}^r$*, let* $\boldsymbol{\phi}(D)$ *denote the* focal point *of* D *such that*

$$\phi_i(D) = \mu_i(D)/\nu_i(D)$$

for any variable $x_i \in X$*.*

Hypothesis 6.1 *Throughout this section we make the generic assumption that for any regulatory domain* D*,* $\exists D' \in \mathcal{D}^r : \boldsymbol{\phi}(D) \in D'$*, that is, all focal points lie within regulatory domains, as in [101].*

If $\boldsymbol{\phi}(D) \in D$ then the systems stays in D, otherwise it eventually leaves D and enters an adjacent switching domain. In switching domains, where $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ are not defined, the behavior of M is defined using differential inclusions as proposed by [118, 156].

Notations: For any $i \in 1...n$, let reg_i and $switch_i$ be predicates on $\mathcal{D}$ characterizing the regulatory intervals and thresholds of $\mathcal{D}_i$, respectively: $reg_i(D) \iff D_i \in \mathcal{D}_i^r$, and $switch_i(D) \iff D_i \in \mathcal{D}_i^s$ for any $D = (D_1, ..., D_n) \in \mathcal{D}$. For a domain D, let $switching(D) = \{x_i \mid switch_i(D)\}$ be the set of switching variables in D. The *order* of a domain D is the number of variables taking a threshold value in D, that is, $order(D) = |switching(D)|$. Let $succ_i : \mathcal{D}_i \to \mathcal{D}_i$ and $prec_i : \mathcal{D}_i \to \mathcal{D}_i$ be the successor and predecessor function on the ordered set of intervals $\mathcal{D}_i$ (in the sense that for any $D_1, D_2 \in \mathcal{D}_i$, $D_1 < D_2$ if $\forall x_1 \in D_1\ \forall x_2 \in D_2 : x_1 < x_2$). We put $succ_i\big((\theta_i^{p_i}, max_i]\big) = prec_i\big([0, \theta_i^1)\big) = \bot$ ("undefined").

Definition 6.4 ($R(D)$) *For any domain $D = (D_1, ..., D_n) \in \mathcal{D}$, let $R(D)$ be the set of regulatory domains that have D in their boundary:*

$$\begin{aligned} R(D) = \big\{(D'_1, ..., D'_n) \mid \forall i \in 1...n : (reg_i(D) \wedge D'_i = D_i) \\ \vee\ \big(switch_i(D) \wedge \big(D'_i = prec(D_i) \vee D'_i = succ(D_i)\big)\big)\big\} \end{aligned}$$

As a special case we have $R(D) = \{D\}$ for $D \in \mathcal{D}^r$.

Example 6.1 *Consider the example of two types of proteins a and b inhibiting each other's production [101]. The respective production rates of proteins a and b are defined by*

$$\mu_a = \begin{cases} 20 & \text{if } 0 \le x_a < \theta_a^2 \wedge 0 \le x_b < \theta_b^1 \\ 0 & \text{otherwise} \end{cases}$$

$$\mu_b = \begin{cases} 20 & \text{if } 0 \le x_a < \theta_a^1 \wedge 0 \le x_b < \theta_b^2 \\ 0 & \text{otherwise} \end{cases}$$

with $\theta_a^1 = \theta_b^1 = 4$ and $\theta_a^2 = \theta_b^2 = 8$. The degradation rate ν of both proteins is always 2. The example is thus modeled by the piecewise linear system $M = \big(\{x_a, x_b\}, \{\theta_a^1, \theta_a^2\} \times \{\theta_b^1, \theta_b^2\}, (\mu_a, \mu_b)^t, diag(\nu, \nu)\big)$. For the switching domain $D = \big([0, \theta_a^1), \{\theta_b^2\}\big)$, we have $R(D) = \big\{\big([0, \theta_a^1), (\theta_b^1, \theta_b^2)\big), \big([0, \theta_a^1), (\theta_b^2, max_b]\big)\big\}$.

Qualitative model. In this section we shortly present the qualitative model of a given piecewise linear system, as defined in [101]. The continuous behaviors can be approximated by a discrete transition graph on the set of domains $\mathcal{D}$. This graph simulates the behavior of the underlying genetic network.

In contrast to previous work where the computation of transitions to and from a domain D required the enumeration of an exponential number of domains surrounding D [101], the approach presented here represents symbolically the relative positions of the focal points, and the successors in the qualitative abstraction.

Definition 6.5 (eq) *Given $\theta = \theta_1 \times ... \times \theta_n$, we define predicates $eq_i^{\#}$ on $\mathcal{D}$, $i \in 1...n$, $\# \in \{<, \le, \ge, >\}$ such that for any domain $D = (D_1, ..., D_n) \in \mathcal{D}$,*

$$\begin{aligned} eq_i^{\#}(D) &\iff \exists D' \in R(D)\ \forall x \in D'_i : \phi_i(D') \# x \\ &\qquad \text{for } \# \in \{<, >\} \\ eq_i^{=}(D) &\iff \exists D' \in R(D)\ \exists x \in D'_i : \phi_i(D') = x \end{aligned}$$

and $eq_i^{\le} = (eq_i^{<} \vee eq_i^{=})$, $eq_i^{\ge} = (eq_i^{=} \vee eq_i^{>})$.

Intuitively, the predicates $eq_i^{\#}$ reflect the relative position of focal points of the adjacent regulatory domains. The predicates $eq_i^{<}(D)$ and $eq_i^{>}(D)$ specify if some adjacent regulatory domain has its focal point "left" of D_i and "right" of D_i, respectively.

Example 6.2 *For the network of Example 6.1, the predicate $eq_a^{<}$ holds in the domains covering the state space $[\theta_a^1, max_a] \times [\theta_b^1, max_b]) \cup ([\theta_a^2, max_a] \times [0, max_b])$. The predicate $eq_a^{=}$ does not hold for any domain.*

Definition 6.6 *Given a piecewise linear system $M = (X, \theta, \boldsymbol{\mu}, \boldsymbol{\nu})$ of dimension n, the qualitative model $Q(M)$ is defined as the transition graph $Q(M) = (\mathcal{D}, \rightarrow)$ with transitions $\rightarrow \subseteq \mathcal{D} \times \mathcal{D}$ such that $\forall D, D' \in \mathcal{D}$, there is a transition $D \rightarrow D'$ if $D \neq D'$ and one of the following conditions holds*

1. *$switching(D) \subset switching(D') \wedge not\_tr(D) \wedge order\_inc\_enabled(D, D')$*
2. *$switching(D') \subset switching(D) \wedge not\_tr(D') \wedge order\_dec\_enabled(D, D')$*

where

$$not\_tr(D) = \bigwedge_{i \in 1 \ldots n} \big(reg_i(D) \quad \vee \\ \min_{D' \in R(D)} \phi_i(D') < D_i < \max_{D' \in R(D)} \phi_i(D')\big)$$

$$\begin{aligned}
order\_inc\_enabled(D, D') = \bigwedge_{i \in 1 \ldots n} \Big(& D'_i = D_i \vee \\
& \big(D'_i = prec_i(D_i) \wedge eq_i^{<}(D)\big) \quad \vee \\
& \big(D'_i = succ_i(D_i) \wedge eq_i^{>}(D)\big)\Big) \\
order\_dec\_enabled(D, D') = \bigwedge_{i \in 1 \ldots n} \Big(& D'_i = D_i \vee \\
& \big(D'_i = prec_i(D_i) \wedge eq_i^{\leq}(D')\big) \quad \vee \\
& \big(D'_i = succ_i(D_i) \wedge eq_i^{\geq}(D')\big)\Big)
\end{aligned}$$

Intuitively, condition $not\_tr(D)$ is satisfied if domain D is not *transient*, in the sense that it is instantaneously crossed by any trajectory of M reaching D [101]. The predicates $order\_inc\_enabled(D, D')$ (resp. $order\_dec\_enabled(D, D')$) specify that a transition from D to a higher (resp. lower) order domain D' is possible only if the direction of D' relative to D is in every dimension consistent with the relative position of the focal points of the source (resp. target) domain. $Q(M)$ corresponds exactly to the qualitative transition graph defined in [101].

Example 6.3 *Continuing Example 6.1, Figure 6.1 shows the qualitative model $Q(M)$ of the piecewise linear system M, according to Definition 6.6.*

Component model of genetic networks. The qualitative model $Q(M)$ (Definition 6.6) defines one global transition graph whose complexity quickly becomes prohibitive with growing M. Moreover, $Q(M)$ encompasses "diagonal" transitions involving qualitative state changes in more than one dimension at the same time, which makes its decomposition into modules even more difficult. Therefore, we now define a component-based model $C(M)$ from a piecewise linear system M. In $C(M)$, the discrete abstraction of each protein concentration is modeled separately. This leads to a structured model, whose modularity can be exploited by compositional verification algorithms.

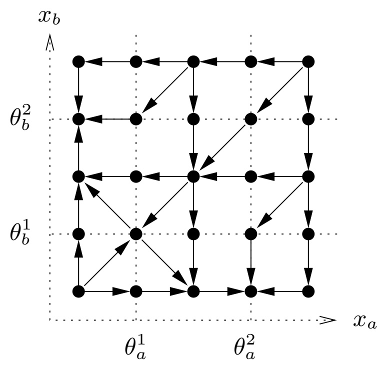


Figure 6.1: Transition graph of the qualitative model $Q(M)$ (solid graph). θ_a^i and θ_b^i are thresholds on the concentrations x_a and x_b, respectively.

Definition 6.7 **($C(M)$)** *Given a piecewise linear system $M = (X, \theta, \boldsymbol{\mu}, \boldsymbol{\nu})$ with $|X| = n$, we define the LTS $C(M) = (\mathcal{D}, A, \rightsquigarrow) := (B_1 \| B_2 \| ... \| B_n)/U_M$, where the LTS B_i and the restriction U_M are defined as follows.*

- *$\forall i \in 1...n : B_i = \mathsf{counter}(\mathcal{D}_i)$, where $\mathsf{counter}(\mathcal{D}_i)$ is a counter defined on $\mathcal{D}_i(\theta)$ by the LTS $\mathsf{counter}(\mathcal{D}_i) = \big(\mathcal{D}_i, \{inc_i, dec_i\}, \{(d, succ_i, d') \mid d, d' \in \mathcal{D}_i \wedge succ_i(d) = d'\} \cup \{(d, prec_i, d') \mid d, d' \in \mathcal{D}_i \wedge prec_i(d) = d'\}\big)$. The set of states is the set of intervals $\mathcal{D}_i$, and inc_i and dec_i are the actions of $\mathsf{counter}(\mathcal{D}_i)$.*
- *U_M is an action constraint such that $U_M(inc_i) = V_i^>$ and $U_M(dec_i) = V_i^<$ with*

$$V_i^< = \big(reg_i \wedge not\_tr \wedge eq_i^<\big) \quad \vee \tag{6.3}$$

$$\big(switch_i \wedge decr_i(eq_i^{\leq})\big) \tag{6.4}$$

$$V_i^> = \big(reg_i \wedge not\_tr \wedge eq_i^>\big) \quad \vee \tag{6.5}$$

$$\big(switch_i \wedge incr_i(eq_i^{\geq})\big) \tag{6.6}$$

Actions inc_i (dec_i) correspond to an increase (decrease) by one of the discretized concentration $level_i$ of protein i. All LTSs $\mathsf{counter}(\mathcal{D}_i)$ are deterministic, therefore $C(M)$ is deterministic.

The predicates $V_i^<$ and $V_i^>$ specify when a transition decrementing $level_i$ and incrementing $level_i$, respectively, is enabled. More precisely, lines (6.3) and (6.5) specify that there is a transition from a lower-order to a higher-order switching domain in the direction of the focal point of the source domain. Condition not_tr makes sure that there is a transition from a switching domain $D \in \mathcal{D}^s$ to a higher-order switching domain only if D is not transient for any dimension. Lines (6.4) and (6.6) give the conditions for transitions decreasing the order: they must be compatible with the relative position of the focal point of the destination domain. Intuitively, lines (6.3) and (6.5) are obtained from condition 1) in Definition 6.6 by considering only transitions increasing the order by one. Lines (6.4) and (6.6) are obtained from condition 2) by considering only transitions decreasing the order by one and relaxing the condition $not\_tr(D')$.

The above modeling framework enforces *separation of concerns* by making a clear distinction between the behaviors of the individual components, and constraints between the components.

Remark 6.1 *Since $\|$ is associative, Definition 6.7 leaves open how the system is actually partitioned into components (in the sense of sets of LTSs). The two extreme cases are that* *each B_i is considered as one component, or that $B_1 \| B_2 \| ... \| B_n$ is considered as one single component. This choice will usually depend on the degree of interaction between the modeled genes. Putting all protein concentrations in one component amounts to a non-modular model leading to non-compositional analysis. Representing each concentration with a separate component may lead to a large number of components, and loss of efficiency. Usually, a good choice is to have the components reflect the functional and spatial structure of the network by gathering closely interacting proteins in the same component, while modeling more loosely interacting proteins with separate components.*

Notations: by construction of $C(M) = (\mathcal{D}, A, \rightsquigarrow)$ we have $\forall D, D' \in \mathcal{D}\ \forall a, b \in A : D \overset{a}{\rightsquigarrow} D' \wedge D \overset{b}{\rightsquigarrow} D' \implies a = b$. Therefore, we will omit the action labels on the transitions, as they are uniquely defined by the source and target domain. We write $\rightsquigarrow^*$ for the transitive and reflexive closure of $\rightsquigarrow$. Given states q and q', q' is *reachable* from q if $q \rightsquigarrow^* q'$.

Example 6.4 *Figure 6.2 shows the transition graphs of* counter($\mathcal{D}_a$)*,* counter($\mathcal{D}_b$)*, and $C(M)$ for the piecewise linear system M of Example 6.1.*

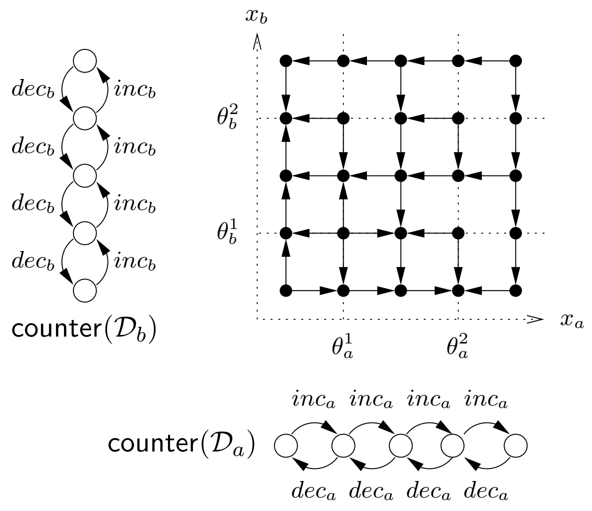


Figure 6.2: The transition graphs of counter($\mathcal{D}_a$), counter($\mathcal{D}_b$), and $C(M) = (\text{counter}(\mathcal{D}_a) \| \text{counter}(\mathcal{D}_b))/U_M$. Transition labels are omitted on $C(M)$.

Model comparison. We show that the component model $C(M) = (\mathcal{D}, A, \rightsquigarrow)$ faithfully models $Q(M) = (\mathcal{D}, \rightarrow)$, up to the representation of "diagonal" transitions of $Q(M)$ synchronously changing more than one state variable. Order-decreasing transitions synchronously changing more than one state variable are serialized, whereas order-increasing transitions synchronously changing more than one state variable are not represented in $C(M)$. From a biological point of view, this approximation is justified by the neglectably small probability of two protein concentrations reaching threshold values exactly at the same instant.

Lemma 6.1 (Correctness of order-increasing transitions) *For any D, D' with $switching(D) \subset switching(D')$, $D \rightsquigarrow D' \implies D \rightarrow D'$.*

Lemma 6.2 (Correctness of order-decreasing transitions) *For any D, D' with $switching(D) \supset switching(D')$ and $not\_tr(D')$, if $\exists D^1, ..., D^{k-1}$ such that $D = D^0 \rightsquigarrow D^1 \rightsquigarrow ... \rightsquigarrow D^{k-1} \rightsquigarrow D^k = D'$ and $order(D^i) = order(D) - i$, then $D \rightarrow D'$.*

In particular, if D' is not transient then $D \rightsquigarrow D' \implies D \to D'$.

Theorem 6.1 (Correctness) *Consider a piecewise linear system $M = (X, \theta, \boldsymbol{\mu}, \boldsymbol{\nu})$. $C(M)$ under-approximates $Q(M)$ in the sense that for any $D, D' \in \mathcal{D}$ with $not\_tr(D')$, $D \rightsquigarrow D' \implies D \to D'$, and $D \rightsquigarrow^* D' \implies D \to^* D'$ (where $\to^*$ is the transitive and reflexive closure of $\to$).*

Proposition 6.1 (Completeness of order-increasing trans.) *If $D \to D'$ for some $D, D' \in \mathcal{D}$ with $order(D) + 1 = order(D')$, then $D \rightsquigarrow D'$.*

Intuitively, if $Q(M)$ can make some transition changing only the value of one variable, then $C(M)$ can make the same transition.

When some diagonal order-decreasing transition is possible in $Q(M)$, then any interleaving is possible in $C(M)$. Formally:

Proposition 6.2 (Completeness of order-decreasing trans.) *If $D \to D'$ is an order-decreasing transition in $Q(M)$ with $switching(D) \supset switching(D')$, then for any sequence $D = D^0, D^1, ..., D^{k-1}, D^k = D'$ of (possibly transient) domains with $\forall i \in 0...k-1 : adj(D^i, D^{i+1}) \wedge switching(D^{i+1}) \subset switching(D^i)$ we have $D \rightsquigarrow D^1 \rightsquigarrow ... \rightsquigarrow D^{k-1} \rightsquigarrow D'$, where $adj(D, D') \iff \exists k : D'_k \in \{prec_k(D_k), succ_k(D_k)\} \wedge \bigwedge_{i \neq k} D'_i = D_i$.*

The example of Figures 6.1 and 6.2 illustrates the relation between $Q(M)$ and $C(M)$ discussed above.

Under-approximation. The component model $C(M)$ allows for compositional analysis of its behavior, by taking advantage of its structure. However, with growing size of the piecewise linear system M, even $C(M)$ may become too complex to analyze. Restricting the component model $C(M)$ to regulatory and first-order switching domains drastically reduces the complexity of computing the constraints $V_i^<$ and $V_i^>$ of Definition 6.7 when constructing the model. The restricted model is often still precise enough to prove reachability properties in practice.

Definition 6.8 ($\mathcal{D}^{01}$) *Let $\mathcal{D}^{01}$ characterize the set of regular domains and first-order switching domains:*

$$\mathcal{D}^{01} = \bigvee_{i \in 1...n} \bigwedge_{j \in 1...n \setminus \{i\}} reg_j$$

Consider the model $C(M)/\mathcal{D}^{01}$, the restriction of $C(M) = (\mathcal{D}, A, \rightsquigarrow)$ to the regular and first-order switching domains, and let $\rightsquigarrow_{\mathcal{D}^{01}}$ denote its transition relation. This is clearly an under-approximation of both $Q(M) = (\mathcal{D}, \to)$ and $C(M)$:

Proposition 6.3 *For any two qualitative states $D, D' \in \mathcal{D}$, $D \rightsquigarrow_{\mathcal{D}^{01}} D' \implies \big(D \to D' \wedge D \rightsquigarrow D'\big)$.*

Remark 6.2 *The transition relation of $C(M)/\mathcal{D}^{01}$ is identical with the restriction of $Q(M)$ to regulatory and first-order switching domains.*

Example 6.5 *Consider the example of two types of proteins a and b modeled by the piecewise linear system $M = \big(\{x_a, x_b\}, \{\theta_a\} \times \{\theta_b\}, (\mu_a, \mu_b)^t, diag(\nu, \nu)\big)$ where*

$$\mu_a = \begin{cases} 2 & if \ \big(0 \le x_a < \theta_a \wedge 0 \le x_b < \theta_b\big) \vee \\ & \qquad \theta_a < x_a \le max_a \\ 0 & otherwise \end{cases}$$

$\mu_b = 2$, $\nu = 1$, and $\theta_a = \theta_b = 1$. Figure 6.3 shows the transition graph of $Q(M)$. Obviously, all paths from domain D_0 to domain D_8 traverse the second-order switching domain D_4. By Propositions 6.1 and 6.2, D_8 is reachable from D_0 in $C(M)$, which is not the case in the under-approximation $C(M)/\mathcal{D}^{01}$.

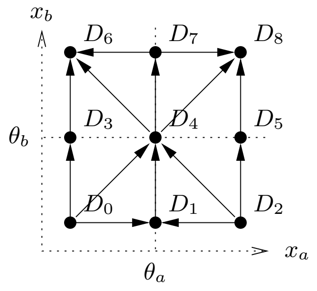


Figure 6.3: Reachability depends on higher-order switching domains being taken into account.

Compositional Reachability Analysis

The component model $C(M)$ faithfully represents the qualitative model $Q(M)$, as discussed above, and can therefore serve as an input for various algorithms analyzing different properties. In the sequel, we will focus on the particular property of reachability. Based on the LTS $C(M)$, the compositional goal-directed simulation algorithm shown below allows us to verify reachability of a goal domain, or set of domains, from an initial domain. The algorithm exhibits a path, if one exists, that solves the reachability problem.

In the sequel we consider an LTS $B = (Q, A, \rightarrow) = (B_1 \| \dots \| B_N)/U$ with $B_i = (Q_i, A_i, \rightarrow_i)$, $i \in K = \{1, \dots, N\}$, and U an action constraint. That is, we suppose the n protein concentrations to be modeled with N $(1 \leq N \leq n)$ components, according to Remark 6.1. Since B is deterministic, we put $post_a(q) = q'$ if $q \xrightarrow{a} q'$.

Let $path_k : Q_k \times \mathcal{P}(Q_k) \rightarrow 2^{A_k}$ be a function on the LTS B_k telling which actions move the state of component k along some path starting in the current component state towards a target predicate. This function can be computed locally: for any predicate $P \in \mathcal{P}(Q_k)$ and state $q \in Q_k$, let

$$\begin{aligned} path_k(q, P) = \big\{ & a \in A_k \mid \exists q', q'' \in Q_k \setminus \{q\} : \\ & \neg P(q) \ \wedge \ q \xrightarrow{a}_k q' \rightarrow_k^* q'' \ \wedge \ P(q'') \big\} \end{aligned} \tag{6.7}$$

That is, $path_k(q, P)$ contains an action a if and only if there exists some path from q to P in the LTS B_k whose first transition is labeled with a.

Given a conjunction $c = c_1 \wedge \dots \wedge c_N$ of predicates $c_i \in \mathcal{P}(Q_i)$, $i \in 1 \dots N$, let $c[i] = c_i$ denote the *projection* of c on Q_i. For a set of actions act, let $\mathsf{enabled}(act, q) = \{a \in act \mid \exists q' \in Q : q \xrightarrow{a} q'\}$. Let $enabling(act)$ be a predicate characterizing the set of states $\{q \mid \exists a \in act \ \ \exists q' \in Q : q \xrightarrow{a} q'\}$ from which some action in act is enabled. Let $\oplus$ denote list concatenation. Given a non-empty list l, we write $l = e : l'$ where e is the first element and l' the rest of the list, and $l.last$ for the last element of l.

Algorithm 1 is constructive, that is, it establishes reachability from some initial domain D_{init} to a set of domains characterized by a predicate P by constructing a path from D_{init} to P. Function **move** takes as arguments the current domain D, a list P of predicates to be reached (that is, the target predicate and a list of sub-goals), and the path constructed so far. It returns a Boolean indicating whether a path was found, and in case of success, the constructed path.

Function **move** works as follows. If the current state satisfies the initial predicate to be reached, the successfully constructed path π is returned (line 3). Otherwise, the tail of list P is removed up to the first predicate that holds in the current state D (line 4). In other words, previous sub-goals that have become obsolete are removed. The remainder of the algorithm consists in a for-loop iterating through all prime implicants (conjunctions) c of P

Algorithm 1 Compositional goal-directed simulation. Initial call to construct a path σ from domain D_{init} to predicate P_{goal}: $(\sigma, success) = \mathbf{move}(D_{init}, \langle P_{goal}\rangle, \langle D_{init}\rangle)$.

```
 1: move $(D, P, \pi)$ {                        — (current domain, list of goals, path prefix)
 2:     if $P.first(D)$                          — $P_{goal}$ reached?
 3:         return $(\pi, true)$;                — then return path and “success”
 4:     $P := \mathbf{pop}(P, D)$;               — remove tail of $P$ from first satisfied sub-goal
 5:     $\forall c \in \bigvee_{p\in P} p$ do    — iterate over conjunctions $c$ of $P$
 6:         $goal := \bigcup_k path_k(D[k], c[k])$;  — compute first actions on local paths to $c$
 7:         $enabled := \mathsf{enabled}(goal, D)$;  — subset of actions enabled in $D$
 8:         $\forall a \in enabled$ do
 9:             $D' := post_a(D)$;               — compute domain $D'$ s.t. $D \xrightarrow{a} D'$
10:             if $D' \notin \pi$ do            — if not yet visited, search path from $D'$
11:                 $(\pi', success) := \mathbf{move}(D', P, \pi \oplus \langle D'\rangle)$;
12:                 if $success$
13:                     return $(\pi', true)$;
14:             od;
15:         od;
16:         $sub\_goal := enabling(goal \setminus enabled)$;        — compute new sub-goal
17:         if $\neg(sub\_goal \implies \bigvee_{p\in P} p)$ do     — if not subsumed by $P$
18:             $(\pi', success) :=$ move $(D, P \oplus \langle sub\_goal\rangle, \pi)$;  — use new sub-goal
19:             if $success$
20:                 return $(\pi', true)$;
21:         od;
22:     od;
23:     return $(\langle\rangle, false)$;        — return “fail” from this call of move
24: }
```

where

$$\mathbf{pop}(x : xs, D) = \begin{cases} \langle\rangle & \text{if } x(D) \\ \langle x\rangle & \text{if } \neg\big(x(D)\big) \wedge xs = \langle\rangle \\ x : \mathbf{pop}(xs, D) & \text{otherwise} \end{cases}$$

(line 5). For each c, a new set of actions *goal* is computed which are prefixes on some local path from the local projections of D to c (line 6), and the sub-set $enabled \subseteq goal$ of actions enabled in the current state is computed (line 7). For each enabled action a, the successor state D' is computed (line 9). If it has not already been visited in the path constructed so far, a recursive call to **move** is made to complete the prefix $\pi \oplus \langle D'\rangle$ with a path leading from D' to the first predicate in P (line 11). If none of these recursive calls succeeds, or $enabled = \emptyset$, a new sub-goal is computed (line 16), consisting of the predicate from which the actions in $goal \setminus enabled$ are enabled. If this sub-goal is not already subsumed by the previous goals (line 17), a recursive call to **move** is made to solve the reachability problem with the new sub-goal added to P (line 18). If all above fails for all $c \in P.last$, then **move** returns $(\langle\rangle, false)$ to indicate failure of the current call.

It can be shown that Algorithm 1 is guaranteed to terminate.

Theorem 6.2 (Correctness) *Algorithm 1 is correct, in the sense that if* $\mathbf{move}(D, \langle P\rangle, \langle\rangle) = (\pi, true)$, *$\pi$ is a path from D to $D' = \pi.last$ in $(B_1\| \ldots \|B_N)/U$ with $P(D')$.*

If for a predicate P characterizing a set of non-transient states, a path $D \rightsquigarrow^* D'$ is found on $C(M)$ (thus, $P(D')$ holds), then Theorem 6.1 ensures that a path $D \to^* D'$ exists in the qualitative model $Q(M)$.

Theorem 6.3 (Completeness) *Algorithm 1 is complete, in the sense that if a path from D to P exists in $(B_1\| \dots \|B_N)/U$, then* $\mathbf{move}(D, \langle P\rangle, \langle\rangle) = (\pi, \mathit{true})$*, and π is a path from D to P.*

Algorithm 1 is compositional in the sense that it independently computes local paths through the state spaces of the components (line 6). A global path is then constructed from the local paths and the constraints between the components: when an action a to be executed is blocked by a constraint involving other components, the function **move** is called recursively to move the blocking components into a domain where a is enabled.

According to Proposition 6.3, reachability in the under-approximation implies reachability in the qualitative model. Therefore, reachability can be checked efficiently on the restricted model $C(M)/\mathcal{D}^{01}$. If no path exists, the search can be refined by verifying reachability directly on $C(M)$. If some reachability problem has no solution, then in the worst case the full state space must be explored to verify that no path exists.

Example 6.6 (Example 6.4 continued.) *The functioning of Algorithm 1 is illustrated by the path construction from domain $D_{init} = \big(\{\theta_a^2\}, (\theta_b^1, \theta_b^2)\big)$ to domain $D_{goal} = \big(\{\theta_a^2\}, [0, \theta_b^1)\big)$ representing a stable equilibrium (Figure 6.2). The subsequent calls of* **move** *are (with the local variables being indexed by the number of the recursive call):*

```
$\mathbf{move}_0(D_{init}, \langle D_{goal}\rangle, \langle D_{init}\rangle)$
4-7:  $P_0 = \langle D_{goal}\rangle$, $c_0 = D_{goal}$, $goal_0 = \{dec_b\}$,
          $enabled_0 = \emptyset$
16:   $sub\_goal_0 = \big\{\big([0, \theta_a^1), (\theta_b^2, max_b]\big), \big((\theta_a^1, \theta_a^2),$
          $(\theta_b^1, max_b]\big), \big((\theta_a^2, max_a], (\theta_b^1, max_b]\big)\big\}$
18:   $\mathbf{move}_1(D_{init}, \langle D_{goal}, sub\_goal_0\rangle, \langle D_{init}\rangle)$
4-9:      $P_1 = \langle D_{goal}, sub\_goal_0\rangle$, $c_1 = \big((\theta_a^1, \theta_a^2),$
              $(\theta_b^1, max_b]\big)$, $goal_1 = \{dec_a\}$,
              $enabled_1 = \{dec_a\}$, $D'_1 = \big((\theta_a^1, \theta_a^2), (\theta_b^1, \theta_b^2\}\big)$
11:       $\mathbf{move}_2(D'_1, \langle D_{goal}, sub\_goal_0\rangle, \langle D_{init}, D'_1\rangle)$
4-9:          $P_2 = \langle D_{goal}\rangle$, $c_2 = D_{goal}$,
                  $goal_2 = \{inc_a, dec_b\}$, $enabled_2 = \{dec_b\}$,
                  $D'_2 = \big((\theta_a^1, \theta_a^2), \{\theta_b^1\}\big)$
11:           $\mathbf{move}_3(D'_2, \langle D_{goal}\rangle, \langle D_{init}, D'_1, D'_2\rangle)$
4-9:              $P_3 = \langle D_{goal}\rangle$, $c_3 = D_{goal}$, $goal_3 = \{dec_b\}$,
                      $enabled_3 = \{dec_b\}$,
                      $D'_3 = \big((\theta_a^1, \theta_a^2), [0, \theta_b^1)\big)$
11:               $\mathbf{move}_4(D'_3, \langle D_{goal}\rangle, \langle D_{init}, D'_1, D'_2, D'_3\rangle)$
2-7:                  $P_4 = \langle D_{goal}\rangle$, $c_4 = D_{goal}$,
                          $goal_4 = \{inc\_a\}$,
                          $enabled_4 = \{inc\_a\}$, $D'_4 = D_{goal}$
11:                   $\mathbf{move}_5(D'_4, \langle D_{goal}\rangle,$
                          $\langle D_{init}, D'_1, D'_2, D'_3, D'_4\rangle)$
3:                        $= (\langle D_{init}, D'_1, D'_2, D'_3, D'_4\rangle, true)$
13:                   $= (\langle D_{init}, D'_1, D'_2, D'_3, D'_4\rangle, true)$
13:               $= (\langle D_{init}, D'_1, D'_2, D'_3, D'_4\rangle, true)$
13:           $= (\langle D_{init}, D'_1, D'_2, D'_3, D'_4\rangle, true)$
20:   $= (\langle D_{init}, D'_1, D'_2, D'_3, D'_4\rangle, true)$
$= (\langle D_{init}, D'_1, D'_2, D'_3, D'_4\rangle, true)$
```

Thus, D_{goal} is reached from D_{init} by decrementing $level_a$, then $level_b$ twice, and then incrementing $level_a$ again.

We have implemented Algorithm 1 in the compositional analysis tool Prometheus [138]. A query language allows for interactive queries to the algorithm, and navigation through the state space. Obviously, the order in which the elements are iterated over in lines 5 and 8 is not fixed in Algorithm 1. Heuristics may be used to accelerate the algorithm in case a path exists, by first examining promising candidates of conjunctions and actions, respectively. Our implementation processes the conjunctions in line 5 by increasing number of conjuncts (that is, decreasing number of states characterized by the predicate). In line 8, the actions are processed by increasing length of their shortest path in (6.7).

Case Study: Delta-Notch Cell Differentiation

Cell differentiation by *Delta-Notch lateral inhibition* is an important step in the embryonic development of many species, as it causes initially uniform cells to assume different functions. Examples are the emergence of ciliated cells in *Xenopus* embryonic skin [206], or neurogenesis in *Drosophila*. *Delta-Notch* lateral inhibition is a well-studied phenomenon, see e.g. [206, 127, 128]. *Delta* is a transmembrane protein that binds and activates its receptor, the transmembrane protein *Notch*, in neighboring cells. High *Notch* levels in a cell inhibit its *Delta* production. Thus, a cell with a high *Delta* concentration prevents its immediate neighbors from adopting the same fate; those neighbors then take a different fate or remain undetermined. Figure 6.4 illustrates these interactions.

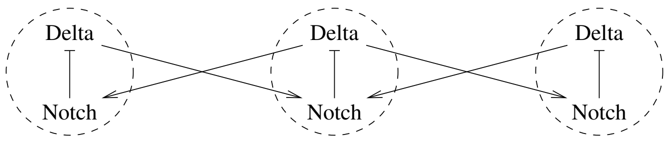


Figure 6.4: Interactions within and between neighbor cells.

For our case study, we consider networks ranging from 19 to 343 cells, with the networks of 49 and 245 cells shown in Figure 6.5. For each protein we partition the continuous state space into two intervals and one threshold value: $\mathcal{D}_D = \{[0, \theta_D), \{\theta_D\}, (\theta_D, max_D]\}$ and $\mathcal{D}_N = \{[0, \theta_N), \{\theta_N\}, (\theta_N, max_N]\}$. Cells with low *Delta* and high *Notch* levels ($0 \le Delta < \theta_D$, $\theta_N < Notch \le max_N$) are undifferentiated, whereas cells with high *Delta* and low *Notch* concentrations ($\theta_D < Delta \le max_D$, $0 \le Notch < \theta_N$) are differentiated, that is, set to adopt a different fate than undifferentiated cells. The actual values of the thresholds θ_D and θ_N are unknown; possible ranges have been derived in [127]. We suppose the focal point ϕ to satisfy

$$\begin{aligned}
0 \le \phi_{Delta_i} < \theta_D &\text{ if } Notch_i > \theta_N \\
\theta_D < \phi_{Delta_i} \le max_D &\text{ if } Notch_i < \theta_N \\
0 \le \phi_{Notch_i} < \theta_N &\text{ if } \\
&\quad \max\{Delta_j \mid j \in neighbors(i)\} < \theta_D \\
\theta_N < \phi_{Notch_i} \le max_N &\text{ if } \\
&\quad \max\{Delta_j \mid j \in neighbors(i)\} > \theta_D
\end{aligned}$$

In the following, we will use our implementation of Algorithm 1 to benchmark the impact of three parameters on the cost of reachability analysis: the size of the model M, the use of $C(M)$ versus the under-approximation $C(M)/\mathcal{D}^{01}$, and the degree of modularity of the model.

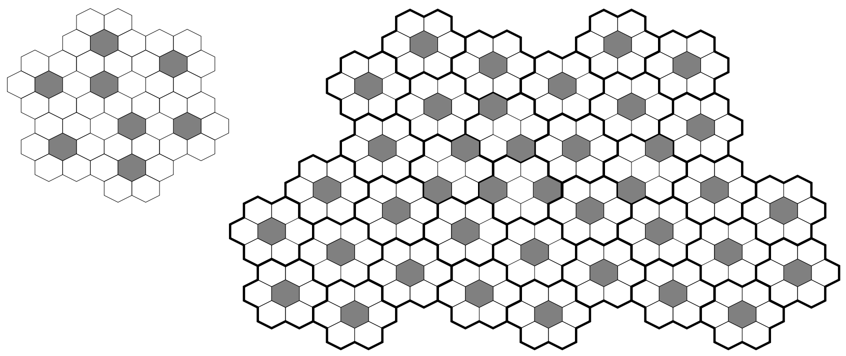


Figure 6.5: Examples of stable equilibrium states involving 49 cells (left) and 245 cells (right). Dark cells express *Delta*.

Influence of size and precision. In this section, we fix the granularity of the model and choose to represent each cell by one component.

For a piecewise linear model M of n cells, and thus a $2n$-dimensional global state space, both the qualitative model $Q(M)$ and the component model $C(M)$ encompass 9^n domains, whereas the under-approximation $C(M)/\mathcal{D}^{01}$ has a state space of 4^n regulatory domains and $2n \times 2^{2n-1}$ first-order switching domains.

We check reachability of a given stable equilibrium from the initial state where all cells are non differentiated, for the models of 19, 37, 49, 245, and 343 cells. Table 6.1 shows the state spaces and execution times for each of the models. "Exact" and "under-approximation" mean that the model $C(M)$ and its restriction $C(M)/\mathcal{D}^{01}$ were used, respectively. The subsequent columns show the number of dimensions, the number of domains of the model, the time for constructing the component model, the number of differentiated cells in the final state, and the time to construct a path to the final state using Algorithm 1. All measurements have been made on the same machine, a Pentium4 at 3 GHz with 512 MB of memory.

number of cells	dim.	state space	model	diff.	reachab.
19 (under-app.)	38	5.5×10^{12}	0.01 s	7	0.23 s
19 (exact)	38	1.4×10^{18}	13.6 s	7	1.30 s
37 (under-app.)	74	7.2×10^{23}	0.04 s	7	0.64 s
49 (under-app.)	98	1.6×10^{31}	0.06 s	8	0.80 s
245 (under-app.)	490	7.9×10^{149}	1.8 s	39	2 m 31 s
343 (under-app.)	686	1.1×10^{209}	4.1 s	52	9 m 22 s

Table 6.1: Performance on models of different size and precision.

In all cases, the results reported by Prometheus are consistent with the actual, experimentally observed behavior [206]. For the case of 245 cells and the state shown in Figure 6.5 involving 39 cells where *Delta* is expressed, Prometheus finds, in spite of the considerable size of the model, a shortest path of length 156 reaching the state. The benchmarks suggest the use of the under-approximation as long as the model remains precise enough to establish the desired property. This significantly accelerates computation of the constraints $V_i^{\cdot}$ (Definition 6.7), which is the most costly part in constructing the model.

Influence of modularity. In order to evaluate the performance increase due to modularity, we now fix the size of M to 49 cells and use the under-approximation $C(M)/\mathcal{D}^{01}$. The only parameter that varies is the granularity of the components, where extreme cases are given by the model of 98 components each modeling one protein concentration, and the model consisting of one single component. According to Remark 6.1, the choice of the structure of the component model does not influence its behavior. On each instance of the system we apply Algorithm 1 to find a path from the initial, undifferentiated state to the state of Figure 6.5 (left).

cells per comp.	0.5	1	3	7	10	49
reachability	2.5 s	2.0 s	1.9 s	9.5 s	77 s	> 12 h

Table 6.2: Benchmarks for different levels of modularity of *Delta-Notch* 49.

The measured performances are shown in Table 6.2. For this example, the optimal degree of granularity lies around two cells per component. It should be noted that the optimal partitioning of proteins into components depend on the system, and cannot be easily generalized. For a higher degree of modularity (one component per protein concentration), the algorithm performs somewhat slower due to an overhead in coordination between closely interacting components. As the component size increases, complexity of the (non compositional) path construction within the components exponentially blows up.

To complete the benchmarks, Table 6.3 shows the comparison with two other tools, GNA 5.5 and the model checking tool CADP [113]. Both tools are targeted at different functionalities, and do not compare directly to our approach. However, this comparison helps to confront the performance of compositional analysis with non-compositional techniques. The second column of Table 6.3 shows the time required by GNA to explore the state space reachable from the initial state in the model $Q(M)$. The last two columns measure the performance of reachability analysis using CADP on the models $C(M)/\mathcal{D}^{01}$ and $C(M)$ previously generated by Prometheus. Notice that although CADP allows for compositional verification, we did not use this feature as it currently does not support action constraints.

	Delta-Notch 19	*Delta-Notch* 37
GNA ($Q(M)$)	(*)	(*)
CADP ($C(M)/\mathcal{D}^{01}$)	9.8 s	(*)
CADP ($C(M)$)	(*)	(*)

Table 6.3: Comparison with other tools. (*): computation interrupted after 24 hours.

Discussion

The study of genetic regulatory networks controlling cellular functions on the molecular level has been strongly stimulated by the development of high-throughput experimental methods. As the knowledge of genetic interaction grows, state space explosion is becoming a limiting factor in the formal analysis of models in systems biology. Modularity and compositionality are a way to cope with this complexity.

We have presented a modeling framework for genetic regulatory networks in which a discrete abstraction of the network dynamics, defined by a system of piecewise linear differential equations, is constructed component-wise. We have formally compared our approach with the discrete abstraction of [101] and shown their equivalence for biologically sound models. We have further proposed a compositional goal-directed simulation algorithm taking advantage of the modularity of the constructed model in order to efficiently analyze reachability properties. On high-dimensional, structured networks the benchmarks clearly show its efficiency. When used interactively, goal-directed compositional simulation can help the user to navigate through huge state spaces.

An interesting work direction, underlined by the results of §6.1.2, is to study how to reflect the structure of the modeled network by aggregating closely interacting genes into components and compute abstractions of their context, in order to computationally take advantage of this topological information.

The focus shift from the study of narrow aspects of cellular behavior towards the integrated view of systems biology, aiming at a system-level understanding of cells [178], is creating a strong need to jointly model and analyze different, interacting fragments of behavior. Component-based modeling is crucial to overcome the complexity of such systems. More work is needed to better understand how existing techniques can be adapted to the specific needs and constraints of systems biology.

6.1.3 Parametric Models

A central problem in the analysis of biological regulatory networks concerns the relation between their structure and dynamics. This problem can be narrowed down to the following two questions: (a) Is a hypothesized structure of the network consistent with the observed behavior? (b) Can a proposed structure generate a desired behavior?

Qualitative models of regulatory networks, such as (synchronous or asynchronous) Boolean models and piecewise-affine differential equation (PADE) models, have been proven useful for addressing the above questions. The models are coarse-grained, in the sense that they do not explicitly specify the biochemical mechanisms. However, they include the logic of gene regulation and allow different expression levels of the genes to be distinguished. They are interesting in their own right, as a way to capture in a simple manner the complex dynamics of a large regulatory network. They can also be used as a first step to orient the development of more detailed quantitative ODE models.

Qualitative models bring specific advantages when studying the relation between structure and dynamics. In order to answer questions (a) and (b), one has to search the parameter space to check if for some parameter values the network is consistent with the data or can attain a desired control objective. In qualitative models the number of different parametrizations is finite and the number of possible values for each parameter is usually rather low. This makes parameter search easier to handle than in quantitative models, where exhaustive search of the continuous parameter space is in general not feasible. Moreover, qualitative models are concerned with trends rather than with precise quantitative values, which corresponds to the nature of much of the available biological data [68].

Nevertheless, the parametrization of qualitative models remains a complex problem. For most models of networks of biological interest the state and parameter spaces are too large to exhaustively test all combinations of parameter values. The aim of our work in [27] is to address this search problem for PADE models by treating it in the context of formal verification and symbolic model checking [85].

Our contributions in [27] are twofold. On the methodological side, we develop a method that in comparison with our previous work [28] makes it possible to efficiently analyze large and possibly incompletely parametrized PADE models. This is achieved by a *symbolic encoding* of the model structure, constraints on parameter values, and transition rules describing the qualitative dynamics of the system. We can thus take full advantage of symbolic model checkers for testing the consistency of the network structure with dynamic properties expressed in temporal logics. The computer tool GNA has been extended to export the symbolic encoding of PADE models in the NuSMV language [82]. In comparison with related work [43, 90, 122], our method applies to incompletely instead of fully parametrized models, provides more precise results, and the encoding is efficient without (strongly) simplifying the PADE dynamics.

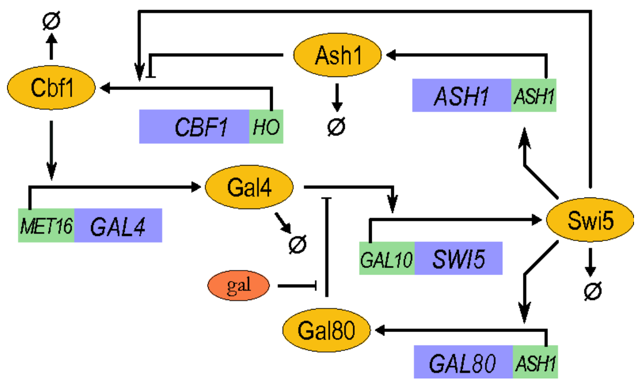


Figure 6.6: Synthetic IRMA network in yeast. Schematic representation of the synthetic IRMA network in yeast constructed in [68]. The green and blue boxes are promoter and genes, and the yellow and red ovals are proteins and metabolites.

On the application side, we show that the *method performs well on real problems*, by means of the IRMA synthetic network and benchmark experimental data sets [68]. More precisely, we are able to find parameter values for which the network satisfies temporal logic properties describing observed expression profiles, both on the level of individual and averaged time-series. The method is selective in the sense that only a small part of the parameter space is found to be compatible with the observations. Analysis of these parameter values reveals that biologically-relevant constraints have been identified. Moreover, we make suggestions to improve the robustness of the external control of the IRMA behavior by proposing a rewiring of the network.

In comparison with traditional quantitative approaches, the results we obtain are quite general, since they do not depend on specific molecular mechanisms or parameter values. Moreover, the analysis is exhaustive in the sense that the entire parameter space is scanned. These two features are particularly interesting for "negative results", such as showing that a given design is not likely to show a desired behavior. In contrast, quantitative ODE models like those developed in [68] do not predict a range of possible behaviors but rather single out one likely behavior with quantitative precision. Qualitative and quantitative approaches provide complementary information on system dynamics.

In comparison with other analysis and verification methods developed for similar modeling formalisms [43, 90, 122], our approach is original in two respects. First, it applies to incompletely parametrized models and can handle any dynamical property expressible in temporal logics supported by the model checker. Second, we reason at a finer abstraction level, in that we take into account dynamics on the thresholds and work with a partition of the state space preserving derivative sign patterns. The latter feature is particularly well-suited for the comparison of model predictions with time-series data in IRMA.

An interesting direction for further research is to consider more general problems in which not only parameters but also regulation functions are incompletely specified.

6.1.4 Discussion

The needs in embedded systems design discussed in the introduction — techniques for constructing complex systems from simpler components, support for heterogeneity, compositional reasoning — also hold for modeling biological systems, and are exacerbated by the plethora and complexity of interactions at all levels of living organisms. Not surprisingly, a great deal of effort in bioinformatics, and more particularly the young domain of *systems biology* [178], has been devoted to formal modeling of such systems. Advanced component-based design techniques are needed to bridge the gap between models of specific aspects, and integrated models that are crucial for gaining a system-level understanding of cells.

On the other hand, it is fascinating to compare the solutions invented by life — including mutual exclusion [206] as well as clock synchronization and distributed consensus [242], to name just a few examples — with approaches from virtually all domains of computer science. For instance, understanding the principles excluding unwanted interferences in spite of tight interaction within and between many levels of abstraction, could provide an inspiring insight for component-based design.

6.2 Multi-scale Controller Synthesis for Switched Systems

When available, discrete abstractions provide an appealing approach to controller synthesis. Recently, an approach for computing discrete abstractions of incrementally stable switched systems has been proposed, using the notion of approximate bisimulation. This approach is based on sampling of time and space where the sampling parameters must satisfy some relation in order to achieve a certain precision. Particularly, the smaller the sampling period, the finer the lattice approximating the state-space and the larger the number of states in the abstraction. This renders the use of these abstractions for synthesis of fast switching controllers computationally prohibitive. In [66] we have presented a novel class of multiscale discrete abstractions for switched systems that allows us to deal with fast switching while keeping the number of states in the abstraction at a reasonable level. The transitions of our abstractions have various durations: for transitions of longer duration, it is sufficient to consider abstract states on a coarse lattice; for transitions of shorter duration, it becomes necessary to use finer lattices. These finer lattices are effectively used only on a restricted area of the state-space where the fast switching occurs. The abstractions are computed on-the-fly during controller synthesis. The finest scales of the abstraction are effectively explored only when fast switching is needed. We present two synthesis algorithms that exploit the specificities of multi-scale abstractions for reachability specifications under time optimization, and for safety synthesis. We illustrate the benefits of our approach by applying it to a boost DC-DC converter. The experimental results show drastic improvements of the complexity of controller synthesis using multi-scale abstractions instead of uniform abstractions.

6.2.1 Introduction

The use of discrete abstractions for continuous dynamics has become standard in hybrid systems design (see e.g. [252] and the references therein). The main advantage of this approach is that it offers the possibility to leverage controller synthesis techniques developed in the areas of supervisory control of discrete-event systems [232] or algorithmic game theory [16]. Historically, the first attempts to compute discrete abstractions for hybrid systems were based on traditional systems behavioral relationships such as simulation or bisimulation [213], initially proposed for discrete systems most notably in the area of formal methods. These notions require inclusion or equivalence of observed behaviors which is often too restrictive when dealing with systems observed over metric spaces. For such systems, a more natural abstraction requirement is to ask for closeness of observed behaviors. This leads to the notions of approximate simulation and bisimulation introduced in [131].

These notions enabled the computation of approximately equivalent discrete abstractions for several classes of dynamical systems, including nonlinear control systems with or without disturbances (see [225] and [224], respectively) and switched systems [132]. These approaches are based on sampling of time and space where the sampling parameters must satisfy some relation in order to obtain abstractions of a prescribed precision. Particularly, it should be noticed that the smaller the time sampling parameter, the finer the lattice used for approximating the state-space; this may result in abstractions with a very large number of states when the sampling period is small. However, there are a number of applications where sampling has to be fast; though this is generally necessary only on a small part of the state-space. For instance, in sliding mode control of switched systems (see e.g. [237]), the sampling has to be fast only in the neighborhood of a sliding surface.

In [66] we have presented a novel class of multiscale discrete abstractions for incrementally stable switched systems that allows us to deal with fast switching while keeping the number of states in the abstraction at a reasonable level. Following the self-triggered control paradigm [261, 265, 12], we assume that the controller of the switched system has to decide the control input and the time period during which it will be applied before the controller executes again. In this context, it is natural to consider abstractions where transitions have various durations. For transitions of longer duration, it is sufficient to consider abstract states on a coarse lattice. For transitions of shorter duration, it becomes necessary to use finer lattices. These finer lattices are effectively used only on a restricted area of the state-space where the fast switching occurs.

These abstractions allow us to use multiscale iterative approaches for controller synthesis as follows. An initial controller is synthesized based on the dynamics of the abstraction at the coarsest scale where only transitions of longer duration are enabled. An analysis of this initial controller allows us to identify regions of the state-space where transitions of shorter duration may be useful (e.g. to improve the performance of the controller). Then, the controller is refined by enabling transitions of shorter duration in the identified regions. The last two steps can be repeated until we are satisfied with the obtained controller.

The concept of approximately bisimilar multi-scale discrete abstractions has also been explored in [253] where the multi-scale feature was used for accommodating locally the precision of the abstraction while the time sampling period remained constant. On the contrary, the approach presented in [66] seeks for a uniform precision but varying time sampling periods. In both works, the multi-scale abstractions were used to synthesize suboptimal reachability controllers.

We also propose to use these multi-scale abstractions for the synthesis of safety controllers for switched systems. The abstractions are computed on-the-fly during controller synthesis and the dynamics at the finest scales are explored only when necessary. We provide experimental results that show drastic improvements of the complexity of controller synthesis using multi-scale abstractions instead of the uniform abstractions defined in [132].

6.2.2 Preliminaries

Incrementally stable switched systems

We propose an approach for controller synthesis for a class of switched systems formalized in the following definition.

Definition 6.9 *A switched system is a quadruple* $\Sigma = (\mathbb{R}^n, P, \mathcal{P}, F)$, *where:*

- $\mathbb{R}^n$ *is the state space;*
- $P = \{1, \ldots, m\}$ *is the finite set of modes;*
- $\mathcal{P}$ *is the set of piecewise constant functions from* $\mathbb{R}^+$ *to* P, *continuous from the right and with a finite number of discontinuities on every bounded interval of* $\mathbb{R}^+$*;*

- *$F = \{f_1, \dots, f_m\}$ is a collection of smooth vector fields indexed by P.*

A *switching signal* of Σ is a function $\mathbf{p} \in \mathcal{P}$, the discontinuities of $\mathbf{p}$ are called *switching times*. A piecewise $\mathcal{C}^1$ function $\mathbf{x} : \mathbb{R}^+ \to \mathbb{R}^n$ is said to be a *trajectory* of Σ if it is continuous and there exists a switching signal $\mathbf{p} \in \mathcal{P}$ such that, at each $t \in \mathbb{R}^+$ where the function $\mathbf{p}$ is continuous, $\mathbf{x}$ is continuously differentiable and satisfies:

$$\dot{\mathbf{x}}(t) = f_{\mathbf{p}(t)}(\mathbf{x}(t)).$$

We will denote $\mathbf{x}(t, x, \mathbf{p})$ the point reached at time $t \in \mathbb{R}^+$ from the initial condition x under the switching signal $\mathbf{p}$. If $\mathbf{p}$ is constantly equal to $p \in P$, the associated trajectory is denoted $\mathbf{x}(t, x, p)$.

The results presented in this chapter apply to switched systems satisfying the incremental stability property (i.e. δ-GUAS [11, 132]). Essentially, a switched system is incrementally stable if all trajectories associated with the same switching signal converge asymptotically to the same reference trajectory independently of their initial condition. Unless all vector fields share a common equilibrium, this does not imply stability.

In the following, $\|.\|$ denotes the Euclidean norm over $\mathbb{R}^n$. Incremental stability of a switched system can be characterized using Lyapunov functions:

Definition 6.10 *A smooth function $V : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+$ is a common δ-GUAS Lyapunov function for Σ if there exist $\mathcal{K}_\infty$ functions*[1] *$\underline{\alpha}$, $\overline{\alpha}$ and $\kappa > 0$ such that for all $x, y \in \mathbb{R}^n$, for all $p \in P$:*

$$\underline{\alpha}(\|x - y\|) \leq V(x, y) \leq \overline{\alpha}(\|x - y\|); \tag{6.8}$$

$$\frac{\partial V}{\partial x}(x, y) f_p(x) + \frac{\partial V}{\partial y}(x, y) f_p(y) \leq -\kappa V(x, y). \tag{6.9}$$

The computation of a δ-GUAS Lyapunov function is generally hard and out of the scope of this report. However, if all vector fields are affine one can compute a quadratic δ-GUAS Lyapunov function by solving a set of linear matrix inequalities. As in [132], we will make the supplementary assumption on the δ-GUAS Lyapunov function that there exists a $\mathcal{K}_\infty$ function γ such that

$$\forall x, y, z \in \mathbb{R}^n,\ |V(x, y) - V(x, z)| \leq \gamma(\|y - z\|). \tag{6.10}$$

This assumption was shown to be not restrictive provided V is smooth and we are interested in the dynamics of Σ on a compact subset of $\mathbb{R}^n$, which is often the case in practice.

In [132], it was proved that under the existence of common δ-GUAS Lyapunov function V satisfying equation (6.10), it is possible to compute discrete abstractions that approximate the dynamics of Σ at any desired level of accuracy.

Approximate bisimulation

In this section, we present the notion of approximate equivalence which will relate a switched system to the discrete systems that we construct. We start by introducing transition systems which allow us to model switched and discrete systems in a common mathematical framework.

Definition 6.11 *A transition system is a tuple $T = (Q, L, \longrightarrow, O, H, I)$ consisting of:*

- *a set of states Q;*

[1]A continuous function $\gamma : \mathbb{R}^+ \to \mathbb{R}^+$ is said to belong to class $\mathcal{K}_\infty$ if it is strictly increasing, $\gamma(0) = 0$ and $\gamma(r) \to \infty$ when $r \to \infty$.

- *a set of labels or actions L;*
- *a transition relation $\longrightarrow \subseteq Q \times L \times Q$;*
- *an output set O;*
- *an output function $H : Q \to O$;*
- *a set of initial states $I \subseteq Q$.*

T is said to be metric if the output set O is equipped with a metric d, discrete if Q and L are finite or countable sets.

The transition $(q, l, q') \in \longrightarrow$ will be denoted $q \xrightarrow{l} q'$, or alternatively $q' \in \mathrm{succ}(q, l)$; this means that the system can evolve from state q to state q' under the action l. An action $l \in L$ belongs to the set of *enabled actions* at state q, denoted $\mathrm{enab}(q)$, if $\mathrm{succ}(q, l) \neq \emptyset$. If $\mathrm{enab}(q) = \emptyset$, then q is said to be a *blocking* state; otherwise it is said to be *non-blocking.* If all states are non-blocking, we say that the transition system T is non-blocking. A *trajectory* of the transition system is a finite sequence of transitions

$$\sigma = q_0 \xrightarrow{l_0} q_1 \xrightarrow{l_1} q_2 \xrightarrow{l_2} \dots \xrightarrow{l_{N-1}} q_N.$$

It is *initialized* if $q_0 \in I$. $N \in \mathbb{N}$ is referred to as the *length* of the trajectory. A state $q \in Q$ is *reachable* if there exists an initialized trajectory reaching q. The *observed behavior* associated to a trajectory is the sequence of outputs $o_0 o_1 o_2 \dots o_N$ where $o_i = H(q_i)$, for all $i \in \{0, \dots, N\}$.

Transition systems can model the dynamics of switched systems. Given a switched system $\Sigma = (\mathbb{R}^n, P, \mathcal{P}, F)$, we define the transition system $T(\Sigma) = (Q, L, \longrightarrow, O, H, I)$, where the set of states is $Q = \mathbb{R}^n$; the set of labels is $L = P \times \mathbb{R}^+$; the transition relation is given by

$$x \xrightarrow{p,\tau} x' \text{ iff } \mathbf{x}(\tau, x, p) = x',$$

i.e. the switched system Σ goes from state x to state x' by applying the constant mode p for a duration τ; the set of outputs is $O = \mathbb{R}^n$; the observation map H is the identity map over $\mathbb{R}^n$; the set of initial states is $I = \mathbb{R}^n$. The transition system $T(\Sigma)$ is non-blocking and deterministic, it is metric when the set of outputs $O = \mathbb{R}^n$ is equipped with the metric $d(x, x') = \|x - x'\|$. Note that the state space of $T(\Sigma)$ is uncountable.

Traditional equivalence relationships for transition systems rely on equality of observed behaviors. One of the most common notions is that of bisimulation equivalence [213, 252]. For metric transition systems, requiring strict equivalence of observed behaviors is often too restrictive. A natural relaxation is to ask for closeness of observed behaviors where closeness is measured with respect to the metric on the output set. This leads to the notion of approximate bisimulation introduced in [131].

Definition 6.12 *Let $T_i = (Q_i, L, \xrightarrow[i]{}, O, H_i, I_i)$, with $i = 1, 2$ be metric transition systems with the same sets of labels L and outputs O equipped with the metric d. Let $\varepsilon \geq 0$ be a given precision. A relation $R \subseteq Q_1 \times Q_2$ is said to be an ε-approximate bisimulation relation between T_1 and T_2 if for all $(q_1, q_2) \in R$:*

- *$d(H_1(q_1), H_2(q_2)) \leq \varepsilon$;*
- *$\forall q_1 \xrightarrow[1]{l} q_1'$, $\exists q_2 \xrightarrow[2]{l} q_2'$, such that $(q_1', q_2') \in R$;*
- *$\forall q_2 \xrightarrow[2]{l} q_2'$, $\exists q_1 \xrightarrow[1]{l} q_1'$, such that $(q_1', q_2') \in R$.*

The transition systems T_1 and T_2 are said to be approximately bisimilar with precision ε, denoted $T_1 \sim_\varepsilon T_2$, if:

- *$\forall q_1 \in I_1$, $\exists q_2 \in I_2$, such that $(q_1, q_2) \in R$;*
- *$\forall q_2 \in I_2$, $\exists q_1 \in I_1$, such that $(q_1, q_2) \in R$.*

If T_1 is a system we want to control and T_2 is a simpler system that we want to use for controller synthesis, then T_2 is called an *approximately bisimilar abstraction* of T_1.

6.2.3 Multi-scale abstractions for switched systems

Let Σ be a switched system and let us assume that the switching in Σ is determined by a time-triggered controller of period $\tau > 0$. Then, the dynamics of Σ can be described by the transition system $T_\tau(\Sigma)$ obtained from $T(\Sigma)$ by selecting the transitions that describe trajectories of duration τ. In [132], an approach to compute approximately bisimilar abstractions of $T_\tau(\Sigma)$ was presented, based on a quantization of the state-space $\mathbb{R}^n$ which is approximated by the lattice:

$$[\mathbb{R}^n]_\eta = \left\{ q \in \mathbb{R}^n \;\middle|\; q[i] = k_i \frac{2\eta}{\sqrt{n}},\ k_i \in \mathbb{Z},\ i \in 1...n \right\}$$

where $q[i]$ is the i-th coordinate of q and $\eta > 0$ is a state space discretization parameter. The resulting abstraction $T_{\tau,\eta}(\Sigma)$ is discrete, its set of states and its set of actions are respectively countable and finite. It is shown in [132] that under the existence of a common δ-GUAS Lyapunov function and equation (6.10), $T_\tau(\Sigma)$ and $T_{\tau,\eta}(\Sigma)$ are approximately bisimilar:

Theorem 6.4 *[132] Consider a switched system Σ, time and state space sampling parameters $\tau, \eta > 0$ and a desired precision $\varepsilon > 0$. If there exists a common δ-GUAS Lyapunov function V for Σ such that equation (6.10) holds and*

$$\eta \leq \min \left\{ \gamma^{-1}\left((1 - e^{-\kappa\tau})\underline{\alpha}(\varepsilon)\right), \overline{\alpha}^{-1}\left(\underline{\alpha}(\varepsilon)\right) \right\} \tag{6.11}$$

then, $T_\tau(\Sigma) \sim_\varepsilon T_{\tau,\eta}(\Sigma)$.

Particularly, it should be noted that given a time sampling parameter $\tau > 0$ and a desired precision $\varepsilon > 0$, it is always possible to choose $\eta > 0$ such that equation (6.11) holds. This essentially means that approximately bisimilar discrete abstractions of arbitrary precision can be computed for $T_\tau(\Sigma)$. However, the smaller τ or ε, the smaller η must be to satisfy equation (6.11). In practice, for a small time sampling parameter τ, the ratio ε/η can be very large and discrete abstractions with an acceptable precision may have a very large number of states (see e.g. [132]).

Unfortunately, there are number of applications where the switching has to be fast though this fast switching is generally necessary only on a restricted part of the state space. For instance, for safety controllers, fast switching is needed only when approaching the unsafe set. In order to enable fast switching while dealing with abstractions with a reasonable number of states, one may consider discrete abstractions enabling transitions of different durations. For transitions of long duration, it is sufficient to consider abstract states on a coarse lattice to meet the desired precision ε. As we consider transitions of shorter durations, it becomes necessary to use finer lattices for the abstract state-space. These finer lattices are effectively used only on a restricted area of the state space, where the fast switching is necessary. This allows us to keep the number of states in the abstraction at a reasonable level. This results naturally in a notion of multi-scale discrete abstraction presented next.

For that purpose, we use self-triggered controllers [261, 12], where the controller not only determines the mode of the switched system but also the duration during which the mode remains active. We assume that the controller can choose from a finite set of durations $\Theta_\tau^N = \{2^{-s}\tau \mid s = 0, \dots, N\}$ that consists of dyadic fractions of a time sampling parameter $\tau > 0$ up to some scale parameter $N \in \mathbb{N}$. The dynamics of the switched system is then naturally described by the transition system $T_\tau^N(\Sigma) = (Q_1, L, \xrightarrow[1]{}, O, H_1, I_1)$ where the set of states is $Q_1 = \mathbb{R}^n$; the set of labels is $L = P \times \Theta_\tau^N$; the transition relation is given by

$$x \xrightarrow[1]{p,2^{-s}\tau} x' \text{ iff } \mathbf{x}(2^{-s}\tau, x, p) = x';$$

the set of outputs is $O = \mathbb{R}^n$; the observation map H_1 is the identity map over $\mathbb{R}^n$; the set of initial states is $I_1 = \mathbb{R}^n$.

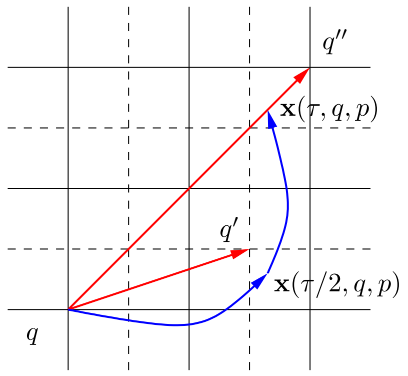


Figure 6.7: Principle for the computation of the discrete abstraction: $q' = \text{succ}_2(q, (p, \tau/2))$ where $q' = \arg\min_{r \in Q_2^1}(\|\mathbf{x}(\tau/2, q, p) - r\|)$ and $q'' = \text{succ}_2(q, (p, \tau))$ where $q'' = \arg\min_{r \in Q_2^0}(\|\mathbf{x}(\tau, q, p) - r\|)$.

The computation of a discrete abstraction of $T_\tau^N(\Sigma)$ can be done using the following approach. We approximate the set of states $Q_1 = \mathbb{R}^n$ by a sequence of embedded lattices: for $s = 0, \dots, N$, let $Q_2^s = [\mathbb{R}^n]_{2^{-s}\eta}$, i.e.

$$Q_2^s = \left\{ q \in \mathbb{R}^n \;\middle|\; q[i] = k_i \frac{2^{-s+1}\eta}{\sqrt{n}},\ k_i \in \mathbb{Z},\ i \in 1...n \right\}$$

where $\eta > 0$ is a state space discretization parameter. Let us remark that we have $Q_2^0 \subseteq Q_2^1 \subseteq \cdots \subseteq Q_2^N$. By simple geometrical considerations, we can check that for all $x \in \mathbb{R}^n$ and $s = 0, \dots, N$, there exists $q \in Q_2^s$ such that $\|x - q\| \le 2^{-s}\eta$.

We now define the abstraction of $T_\tau^N(\Sigma)$ as the transition system $T_{\tau,\eta}^N(\Sigma) = (Q_2, L, \xrightarrow[2]{}, O, H_2, I_2)$ where the set of states is $Q_2 = Q_2^N$; the set of actions remains the same $L = P \times \Theta_\tau^N$; the transition relation is given by

$$q \xrightarrow[2]{p,2^{-s}\tau} q' \text{ iff } q' = \arg\min_{r \in Q_2^s}(\|\mathbf{x}(2^{-s}\tau, q, p) - r\|).$$

If the minimizer $r \in Q_2^s$ is not unique, then one can choose arbitrarily one of them. By definition of the set $Q_2^s = [\mathbb{R}^n]_{2^{-s}\eta}$, if $q \xrightarrow[2]{p,2^{-s}\tau} q'$ then we have $\|\mathbf{x}(2^{-s}\tau, q, p) - q'\| \le 2^{-s}\eta$. The approximation principle is illustrated in Figure 6.7. The set of outputs remains $O = \mathbb{R}^n$; the observation map H_2 is the natural inclusion map from Q_2^N to $\mathbb{R}^n$, i.e. $H_2(q) = q$; the set of initial states is $I_2 = Q_2^0$.

It is important to note that all the transitions of duration $2^{-s}\tau$ end in states belonging to Q_2^s. This means that the states on the finer lattices are only accessible by transitions of shorter duration. Note that the transition system $T^N_{\tau,\eta}(\Sigma)$ is discrete since its sets of states and actions are respectively countable and finite. Also, if we only consider transitions of duration τ, the dynamics of $T^N_{\tau,\eta}(\Sigma)$ coincides with that of the uniform abstraction $T_{\tau,\eta}(\Sigma)$ defined in [132]. Both transition systems $T^N_\tau(\Sigma)$ and $T^N_{\tau,\eta}(\Sigma)$ are non-blocking and deterministic.

Theorem 6.5 *Consider a switched system Σ, time and state space sampling parameters $\tau, \eta > 0$, scale parameter $N \in \mathbb{N}$, and a desired precision $\varepsilon > 0$. Let us assume that there exists a common δ-GUAS Lyapunov function V for Σ such that equation (6.10) holds. If*

$$\eta \leq \min\left\{ \min_{s=0\ldots N} \left[2^s \gamma^{-1}\left((1-e^{-\kappa 2^{-s}\tau})\underline{\alpha}(\varepsilon)\right)\right], \overline{\alpha}^{-1}\left(\underline{\alpha}(\varepsilon)\right)\right\} \tag{6.12}$$

then $R = \{(x,q) \in Q_1 \times Q_2 \mid V(x,q) \leq \underline{\alpha}(\varepsilon)\}$ is an ε-approximate bisimulation relation between $T^N_\tau(\Sigma)$ and $T^N_{\tau,\eta}(\Sigma)$. Moreover, $T^N_\tau(\Sigma) \sim_\varepsilon T^N_{\tau,\eta}(\Sigma)$.

It is interesting to note that given a time sampling parameter $\tau > 0$ and a scale parameter $N \in \mathbb{N}$, for all desired precisions $\varepsilon > 0$, there exists $\eta > 0$ such that equation (6.11) holds. This essentially means that approximately bisimilar multiscale abstractions of arbitrary precision can be computed for $T^N_\tau(\Sigma)$.

6.2.4 Safety Controller Synthesis on Multi-scale Abstractions

We have illustrate the use of multiscale abstractions for synthesizing safety and sub-optimal reachability controllers in [66, 65, 217]. These problems were considered in [208, 130] based on the use of uniform discrete abstractions. We extend the synthesis algorithms to multiscale abstractions that are computed on-the-fly, so as to provide a scalable trade-off between precision and cost, while guaranteeing, for reachability, a lower bound on the performance of the closed-loop system.

Let us consider a system $T = (Q, L, \longrightarrow, O, H, I)$. For simplicity, we assume that T is deterministic. This is satisfied by the transition systems $T^N_\tau(\Sigma)$ and $T^N_{\tau,\eta}(\Sigma)$ defined in the previous section.

In the following, we consider deterministic static state-feedback controllers. However, we just use the term controller for brevity.

Definition 6.13 *A controller for T is a map $\mathcal{S} : Q \to 2^L$ such that for all $q \in Q$, $\mathcal{S}(q) \subseteq \text{enab}(q)$ and $\forall l \in \mathcal{S}(q)$, $\mathcal{S}\big(\text{succ}_l(q)\big) \neq \emptyset$ (deadend freedom).*

The system T controlled by $\mathcal{S}$ is the system $T/\mathcal{S} = (Q, L, \xrightarrow[\mathcal{S}]{}, O, H, I)$ where the transition relation is given by

$$q \xrightarrow[\mathcal{S}]{l} q' \iff \Big[(l \in \mathcal{S}(q)) \wedge (q \xrightarrow{l} q')\Big].$$

$\mathcal{S}(q) = \emptyset$ means that the controller is not defined at q. Since we assumed that T is deterministic, the system $T/\mathcal{S}$ is deterministic as well.

Problem formulation

Definition 6.14 *A state q of T is* controllable *with respect to a safety specifications Q_S if $q \in Q_S$ and there exists an infinite sequence of transitions of T starting in q and remaining in Q_S. We denote the set of controllable states by* $\text{cont}(Q_S)$.

Definition 6.15 *A* safety controller *for $T = (Q, L, \longrightarrow, O, H, I)$ with respect to a safety specification $Q_S \subseteq Q$ is a controller $\mathcal{S}$ for T (see Definition 6.13 such that Q_S is invariant in $T/\mathcal{S}$, and $\mathcal{S}(q) = 0$ for $q \notin \text{cont}(Q_S)$.*

Given the set of controllable states $\mathrm{cont}(Q_S)$, we can define a safety controller $\mathcal{S}^*$ as follows: for all $q \notin \mathrm{cont}(Q_S)$, $\mathcal{S}^*(q) = \emptyset$ and for all $q \in \mathrm{cont}(Q_S)$,

$$\mathcal{S}^*(q) = \{l \in \mathrm{enab}(q) \mid \mathrm{succ}_l(q) \in \mathrm{cont}(Q_S)\}.$$

This safety controller is maximal in the sense that any other safety controller $\mathcal{S}$ satisfies $\mathcal{S}(q) \subseteq \mathcal{S}^*(q)$, for all $q \in Q$. The set $\mathrm{cont}(Q_S)$ and thus $\mathcal{S}^*$ are computable for our discrete abstractions. However, the larger the number of states, the more expensive the computation. For that reason, we want to exploit multi-scale abstractions to propose a more efficient algorithm for the synthesis of safety controllers.

Let us consider that the set of labels of T is $L = P \times \Theta_\tau^N$ as defined in the previous section. The lazy safety synthesis problem consists in controlling a system so as to keep any trajectory starting from some initial state in I within the safe subset of states, while applying in each state a transition of the longest possible duration for which safety can be guaranteed. For that purpose we define priority relations on the set of labels giving priority to transitions of longer duration: for $l, l' \in L$ with $l = (p, \delta)$, $l' = (p', \delta')$, $l \preceq l'$ iff $\delta \leq \delta'$, $l \prec l'$ iff $\delta < \delta'$ and $l \cong l'$ iff $\delta = \delta'$. Given a subset of labels $L' \subseteq L$, we define

$$\max_{\preceq}(L') = \{l' \in L' \mid \forall l \in L', l \preceq l'\}.$$

Definition 6.16 *A maximal lazy safety controller for* $T = (Q, L, \longrightarrow, O, H, I)$ *and* Q_S *is a safety controller* $\mathcal{S}$ *with respect to* Q_S *such that for all controllable initial states in* $q \in I \cap \mathrm{cont}(Q_S)$, $\mathcal{S}(q) \neq \emptyset$, *and for all states* $q \in Q$ *with* $\mathcal{S}(q) \neq \emptyset$, q *is reachable in* $T/\mathcal{S}$ *and the following conditions hold:*

1. *if* $l \in \mathcal{S}(q)$, *then for any* $l \prec l'$, $\mathrm{succ}_{l'}(q) \notin \mathrm{cont}(Q_S)$ *(laziness);*

2. *if* $l \in \mathcal{S}(q)$, *then for any* $l \cong l'$, $l' \in \mathcal{S}(q)$ *iff* $\mathrm{succ}_{l'}(q) \in \mathrm{cont}(Q_S)$ *(maximality).*

The controller $\mathcal{S}$ represents a trade-off between maximal permissiveness and efficiency, in the sense that it contains the same initial states as the maximal safety controller; on the other hand, in each state, the enabled transitions are those of maximal duration for which controllability is preserved.

Theorem 6.6 *There exists a unique maximal lazy safety controller.*

Discrete controller synthesis for multiscale abstractions

Our algorithm for synthesizing the maximal lazy safety controller is based on a depth first search exploration of the trajectories, starting from initial states and exploring transitions of longer duration first. The multi-scale abstraction is computed on-the-fly during the synthesis algorithm.

More precisely, the maximal lazy safety controller is computed by Algorithm 1 which calls the function *explore* (Algorithm 2) for each initial state $q \in I$; the second argument of the function is the set of states already visited by the current trajectory. The global variables are K, U, and C for the sets of controllable and uncontrollable states and the controller, respectively. Function *explore*(q, V) returns whether q is controllable, where V is the set of states already visited. *explore* recursively explores the paths starting from q until either a controllable or an uncontrollable state is reached. If a state already visited by the current trajectory is reached, then a circular path within the set of safe states and containing q has been found, and therefore the state q is controllable. The outer loop explores increasingly short transitions as long as no safe successor of q has been found.

In Algorithm 1, each transition initiating from a state in Q_S is explored at most once. Hence, termination of Algorithm 1 is guaranteed if the sets of labels L, and of safe sets Q_S are finite: this is the case for our multi-scale abstractions $T^N_{\tau,\eta}(\Sigma)$ when the safe set is given by $Q_S = H^{-1}(O_S)$ with O_S a compact subset of $\mathbb{R}^n$. In the worst case (when all states in Q_S are reachable but none is controllable), all the transitions initiating from a state in Q_S need to be explored. This provides us with a worst-case (time and space) complexity given by $|Q_S| \times |L|$. However, in practice this upper-bound is not reached.

Algorithm 1: Maximal lazy safety controller synthesis.

```
Input: Transition system $(Q, L, \rightarrow, O, H, I)$, priority $\preceq \subseteq L \times L$, safe states $Q_S \subseteq Q$
Output: maximal lazy safety controller $C : Q \rightarrow 2^L$
Data: controllable states $K \subseteq Q$, uncontrollable states $U \subseteq Q$
begin
    $(K, U, C) := (\emptyset, Q \setminus Q_S, \emptyset)$ ;
    for $q \in I \cap Q_S$ do
        explore $(q, \emptyset)$ ;
```

Algorithm 2: explore(q, V)

```
Input: state $q \in Q$, visited states $V \subseteq Q$
Output: true iff $q$ is controllable
Data: unexplored labels $L_u \subseteq L$
begin
    if $q \in U$ then
        return false
    if $q \in K$ then
        return true
    if $q \in V$ then
        $K := K \cup \{q\}$ ;
        return true
    $L_u := L$ ;
    while $L_u \neq \emptyset$ do
        foundSucc := false ;
        for $l \in \max_{\preceq}(L_u)$ do
            if explore $(succ(q, l), V \cup \{q\})$ then
                $C(q) := C(q) \cup \{l\}$ ;
                foundSucc := true ;
        if foundSucc then
            $K := K \cup \{q\}$ ;
            return true
        $L_u := L_u \setminus \max_{\preceq}(L_u)$ ;
    $U := U \cup \{q\}$ ;
    return false
```

Controller synthesis for reachability. In [66, 217] we have discussed the special cases of time-optimal and time-bounded reachability.

6.2.5 Experimental Results

Our approach has been implemented in the tool CoSyMA [217] (Controller Synthesis using Multi-scale Abstractions).

DC-DC Converter

As a first case study, we apply our approach to a boost DC-DC converter. It is a switched system with two modes, the two dimensional dynamics associated with both modes are affine of the form $\dot{\mathbf{x}}(t) = a_p\mathbf{x}(t) + b$ for $p \in \{1,2\}$ (see [133] for numerical values). It can be shown that it is incrementally stable and thus approximately bisimilar discrete abstractions can be computed. We consider the problem of keeping the state of the system in a desired region of operation given by the safe set $O_S = [1.15, 1.55] \times [5.45, 5.85]$.

We use approximately bisimilar abstractions to synthesize maximal lazy safety controllers for the DC-DC converter. We compare the cost of controller synthesis for the uniform abstraction $T^0_{\tau_1,\eta_1}$ for parameters $\tau_1 = 0.5s$ and $\eta_1 = 10^{-3}\sqrt{2}/4$ (containing transitions of duration 0.5s) and the multi-scale abstractions $T^2_{\tau_2,\eta_2}$ for parameters $\tau_2 = 4\tau_1$ and $\eta_2 = 4\eta_1$ (containing transitions of durations in $\Theta^2_\tau = \{2s, 1s, 0.5s\}$). These two abstractions have the same precision. Table 6.4 details the experimental results obtained for the synthesis of the controllers for $T^0_{\tau_1,\eta_1}$ and $T^2_{\tau_2,\eta_2}$. We can see that there is a noteworthy reduction of the time used to compute the controller using multi-scale abstractions instead of using uniform ones (up to a 86% improvement between $T^0_{\tau_1,\eta_1}$ and $T^2_{\tau_2,\eta_2}$). This is due to the fact that the size of uniform abstractions grows exponentially with higher resolutions, whereas using multi-scale abstractions are refined only when we get close to unsafe regions (reduction of more than 91% between $T^0_{\tau_1,\eta_1}$ and $T^2_{\tau_2,\eta_2}$). Interestingly, this reduction in computation time and size does not affect the performance of the multi-scale controllers, which yield a ratio of controllable initial states[2] (CR) over the safety specification comparable to that of their its uniform counterparts. It is worth emphasizing that using CoSyMA there is a remarkable reduction of the computation times compared to those reported in [65] obtained by a prototype implementation of the algorithm. Figure 6.8 depicts the maximal lazy safety controller for $T^2_{\tau_2,\eta_2}$ and Q_S and the trace of its simulation starting from the state (1.15,5.6).

	Abstractions $T^N_{\tau,\eta}$ for $\varepsilon = 0.1$	
	$N = 0, \tau = 0.5s,\ \eta = 10^{-3}\sqrt{2}/4$	$N = 2, \tau = 2s,\ \eta = 10^{-3}\sqrt{2}$
Time [s]	8.3	1.1
Size [10^3]	599	53
$\delta(l)$		2 s (63.6%) 1 s (31.7%)
	0.5 s (100%)	0.5 s (4.7%)
CR	93.52%	93.51%

Table 6.4: Experimental results for the maximal lazy safety controller synthesis for the boost DC-DC converter.

Building Temperature Regulation

The second case study deals with temperature regulation in a circular building. Each room is equipped with a heater and at a given instant at most one heater is switched on. The

[2]The ratio of controllable initial states for a controller $\mathcal{S} : Q \to 2^L$ and a system $T = (Q, L, \to, O, H, I)$ is computed as $|\{q \in I | \mathcal{S}(q) \neq \emptyset\}|/|I|$.

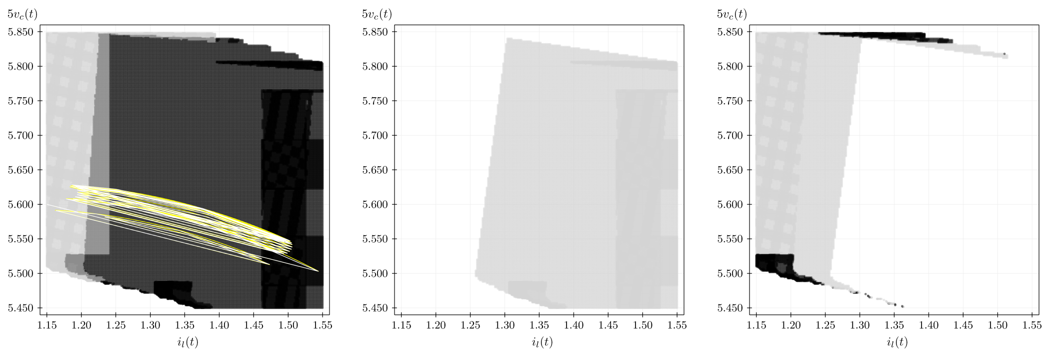


Figure 6.8: The maximal lazy safety controller for $T^2_{\tau_2,\eta_2}$ and Q_S. Left: mode 1 is activated (light gray); mode 2 is activated (black); modes 1 and 2 (gray). Middle: actions of 2s are enabled (light gray). Right: actions of 1s are enabled (light gray), actions of 0.5s are enabled (black).

temperature t_i of the room i is defined by the differential equation $\dot{t}_i = \alpha(t_{i+1} + t_{i-1} - 2t_i) + \beta(t_e - t_i) + \gamma(t_h - t_i)u_i(t)$ where t_{i-1} is the temperature of the room $i-1$; t_{i+1} is the temperature of the room $i+1$; t_e is the temperature of the external environment of the building; t_h is the temperature of the heater; α is the temperature transfer ratio between the rooms $i \pm 1$ and the room i; β is the temperature transfer ratio between the external environment and the room i; γ is the temperature transfer ratio between the heater and the room i; $u_i(t)$ equals to 1 if the room i is heated, or 0 otherwise. Given a number $n \geq 2$ of rooms, we distinguish $n+1$ switching modes. For $1 \leq i \leq n$, the mode p_i represents the mode of activating the heater of room i. The mode p_{n+1} represents that no heater is activated. The values of α, β, γ, t_e, and t_h are respectively 1/20, 1/200, 1/100, 10, and 50. We will increase the system dimension to test the limits of the tool in terms of memory usage and computation time. Given the safety specification $Q_S = [20.0, 22.0]^n$ for $n \in \{3, 4, 5\}$, we synthesize safety controllers for buildings of three, four, and five rooms. The values of τ and η are given in Table 6.5. By looking to the results, we can see the combinatorial explosion of the size of abstractions by increasing the system dimension from 3 to 5. Also, it makes sense that the ratio of controllability of initial states decreases by increasing the number of rooms. On our machine equipped with a Core i5-2430M and 4GB of RAM, synthesis fails for the 6-dimensional instance due to running out of memory.

	Abstractions $T^N_{\tau,\eta}$		
	$n = 3$, $N = 2$, $\eta = 50 \times 10^{-3}$ $\tau = 20s$, $\varepsilon = 0.2$	$n = 4$, $N = 2$, $\eta = 50 \times 10^{-3}$ $\tau = 20s$, $\varepsilon = 0.2$	$n = 5$, $N = 1$ $\eta = 0.1$ $\tau = 10s$, $\varepsilon = 0.4$
Time [s]	2.4	595	571
Size [10^3]	56	3 928	6 135
$\delta(l)$	20 s (20%) 10 s (80%) 5 s (0%)	20 s (9%) 10 s (87%) 5 s (4%)	10 s (86%) 5 s (14%)
CR	99.99%	99.89%	99.79%

Table 6.5: Comparison of experimental results for the safety synthesis for the temperature regulator system of three, four, and five dimensions.

Figure 6.9 shows the maximal lazy safety controller for the transition system $T^2_{20,0.05}$ of three dimensions and the safety specification $Q_S = [20.0, 22.0]^3$. The plots are slices of the

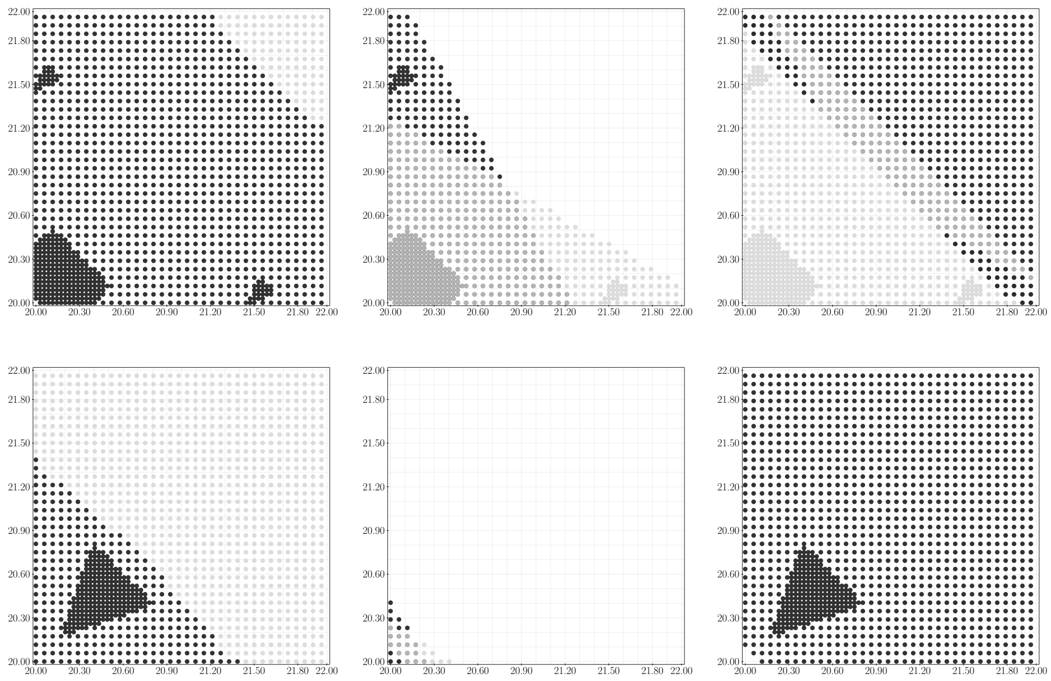


Figure 6.9: Maximal lazy safety controller for $T^2_{20,0.05}$ of three dimensions and Q_S. Horizontal axis: $t_1(t)$; vertical axis: $t_2(t)$. Top: $t_3(t) \approx 20$; bottom: $t_3(t) \approx 22$. Left: actions of 20s are enabled (black); actions of 10s are enabled (gray); middle: mode p_1 (black); mode p_2 (light gray); p_1 and p_2 (gray); bottom: mode p_4 (black); mode p_3 (light gray); p_3 and p_4 (gray).

state space in the dimensions (t_1, t_2) for a fixed $t_3 \approx 20°$ (left) and $t_3(t) \approx 22°$ (right), respectively. The plots on the top depict scales and those in the middle and the bottom depicts modes. We can remark the predominance of the mode p_4 (no heater is activated) by increasing the temperature of third room.

6.2.6 Discussion

We have proposed the use of multiscale, approximately bisimilar discrete abstractions for the computation of controllers, applying them to the specific control problems of time-optimal reachability and safety. This work is a step towards a tight integration of formal design techniques for the discrete and continuous part of embedded systems. Our experimental results have shown that we can achieve a remarkable reduction in the computation time of such controllers in comparison with the use of uniform abstractions, while preserving similar levels of performance. Future work will deal with the application of multi-scale abstractions to other kinds of control problems, and compositional controller synthesis for switched systems.

More recently we have proposed a different approach to the computation of symbolic abstractions of incrementally stable switched systems [194]. The main novelty consists in using mode sequences of given length as symbolic states for our abstractions. We have shown that the resulting symbolic models are approximately bisimilar to the original switched system and that an arbitrary precision can be achieved by considering sufficiently long mode sequences. The advantage of this approach over existing ones is double: first, the transition relation of the symbolic model admits a very compact representation under the form of a shift operator; second, our approach does not use lattices over the state-space and can potentially be used for higher dimensional systems. We have provided a theoretical comparison with the lattice-based approach and presented a simple criterion enabling to choose the most appropriate approach for a given switched system. Finally, we have shown an application to a model of road traffic for which we have synthesized a schedule for the coordination of traffic lights under safety and fairness constraints.

Chapter 7

Conclusion

7.1 Work Achieved

In this document I have presented different approaches to component-based design of embedded systems. Starting with and building on the component framework Bip (§2) I have proposed algorithms for compositional analysis of general properties like deadlock freedom, progress, and liveness (§3.2), and for synthesis (adaptor synthesis (§3.3) and compositional strategy mapping (§3.4)). I have further presented results on conformance checking for choreographies with unbounded communication channels (§3.5), and on fault recovery in Bip (§3.6). As an application of Bip to systems biology, I have studied how to compositionally model and analyze genetic regulatory networks (§6.1.2) and how to use model checking to estimate kinetic parameters of genetic networks (§6.1.3).

I have further proposed formal frameworks for contract-based design, including modal assume/guarantee contracts (§4.1) and probabilistic contracts (§4.2), and developed a novel approach to analyze logical causality in a trace-based framework (§5).

Finally, I have developed an approach making efficient use of discrete controller synthesis for the control of a class of switched continuous systems with respect to safety and reachability properties (§6.2).

The common goal of these results is the construction of complex systems ensuring some crucial correctness properties by construction.

7.2 Looking Ahead

Several of the results presented in this document would deserve being further developed, implemented, and applied to real-world problems. For instance, an implementation of our contract frameworks in the Bip tool platform could implement a type system for components and strengthen the support for a compositional design flow.

The results presented in this document also open numerous new research directions and challenges, a few of which I will present in the form of an interdisciplinary research program.

Causality and Risk Management in Component-based Systems

Previous research on formal methods for component-based design has mostly focused on functional and real-time properties. In my future research I intend to focus on two phenomena that so far received less attention in the formal design of computing systems: logical causality and risk.

Putting Causality Analysis to Use

In §5 I have presented results for defining and effectively analyzing logical causality between component failures. I intend to pursue this research direction along the following lines, with the mid-term goal of transferring the results to actual applications.

- Generalize our definitions of logical causality to different models of computation and communication such as networks of timed automata [8] and the Real-Time Calculus [254].
- Counterfactual reasoning ("what would have been the outcome if component C had behaved correctly?") inherently suffers from inconsistencies between the observed, real behavior and the hypothetical behavior due to side effects of C's behavior [209]. I intend to develop solutions to alleviate these issues in order to improve discriminancy of causality analysis. For instance, a fault model of all possible (correct or incorrect) behaviors of a component will allow a more fine-grained analysis of how a component may have been influenced by failures of its peers.
- Implementation details of components may be hidden but some behaviors may be known to be more likely than others. A probabilistic component model would allow us to determine the probability of counterfactual scenarios so as to achieve a quantitative notion of causality, e.g., a *degree of responsibility* [80].
- An important aspect of this work is to implement and apply the results to real-world problems of causality analysis, in particular from the domain of embedded and medical [197] devices. I have started a collaboration with colleagues from University of Pennsylvania on this topic. I also intend to investigate how to efficiently implement causality analysis, based on the use of appropriate data structures similar to [109]. A possible candidate platform for logging is the OSGI logging framework *LogOS* [121].

Causality, i.e., the logical dependence of an effect on a cause, has long been studied in philosophy [199], natural sciences, law [215], logic [81, 47], and statistics [220], among many other disciplines. The analysis of causality has applications in many areas of computer science. For instance, tracking causality between events in the execution of concurrent system is required to ensure reversibility [187]; to allow the diagnosis of faults in a complex concurrent system [109]; or to enforce accountability [181], that is, designing systems in such a way that it can be determined without ambiguity whether a required safety or security property has been violated, and why. More generally, the goal of fault tolerance can be understood as preventing certain causal chains from occurring by designing systems such that each causal chain either has its premises outside of the fault model (e.g., by introducing redundancy [124]), or is broken (e.g., by limiting fault propagation [239]).

Due to these multiple facets of causality, my research on causality analysis will necessarily be interdisciplinary and in collaboration with colleagues from other domains.

Dynamic Contracts and Risk Management

In a framework of dynamic contracts, which may be proposed at run-time, a component can use the set of contracts it is involved in to evaluate its commitments and analyze the *risk* (or the opportunity) of agreeing upon a set of new contracts. For instance, for behavioral contracts "risk" may be defined as the absence of a strategy of the component to meet all contracts. Scheduling of mixed criticality systems [262] can be seen as a special case of risk management: a task of low criticality is scheduled only if this does not entail the risk for a critical task to miss its deadline. Another special case of interest is the decision whether it is safe to accept a software update.

As an extension I intend to study *quantitative contracts*. Quantitative contracts — e.g., based on automata with costs — may express requirements of components over time, and the guarantees they provide if the requirements are met. For instance, a component can guarantee a lower bound on throughput, provided that the component is granted some minimum amount of resources upon periodic request; another component may give safety guarantees provided that its real-time requirements are met. For quantitative contracts we can define the risk (resp. opportunity) as the increase (resp. decrease) in cost of the best strategy ensuring the combined guarantees of all enacted contracts. Contracts may also encompass *penalties* for the case that a partner is not able or willing to meet its promises [112]. In that case, the risk would amount to the sum of penalties a component incurs in the case of the impossibility to honor previous commitments.

In order to perform risk evaluation efficiently in the case of quantitative contracts, we may investigate the use of optimization techniques such as mixed integer programming [94] to find an optimal strategy.

Besides risk analysis, evaluating the quantities committed through contracts is of interest for the design of embedded systems: the latter can often be reconfigured at run-time to trade off speed and power consumption. However, current scheduling approaches merely react to the workload, available computing power and energy at the current instant, rather than being able to anticipate future demand and available resources[1]. This is mainly because current software is not able to provide this information. In contrast, the set of valid contracts provides information about the future use of resources promised to components, such that additional resources can be activated before contention actually creates a bottleneck.

The research directions on dynamic and quantitative contracts and risk analysis will be investigated as part of the French project C^5 on contract-based design of certified components (submitted).

As for causality, this research direction will have a multidisciplinary character, as similar contract frameworks are used in finance [172] and of course, law. Conversely, formal, computationally tractable approaches to risk management will be of interest for disciplines other than computer science.

Both causality and risk analysis should be applicable to *models* (a priori) — e.g., to evaluate whether a design is sufficiently robust —, to systems *at runtime* — e.g., to decide whether to isolate a faulty component or to act a new contract —, and to *logs* (a posteriori) — e.g., to establish liability. I intend to develop *language support* to enable high-level control over causality and risk management.

[1]Model predictive control uses knowledge from previous executions, and cannot anticipate different behaviors.

Bibliography

[1] M. Abadi and L. Lamport, *The existence of refinement mappings*, Theor. Comput. Sci., 82 (1991), pp. 253–284.

[2] T. Abdellatif, J. Combaz, and J. Sifakis, *Model-based implementation of real-time applications*, in EMSOFT, L. P. Carloni and S. Tripakis, eds., ACM, 2010, pp. 229–238.

[3] F. Achermann and O. Nierstrasz, *A calculus for reasoning about software composition*, Theor. Comput. Sci., 331 (2005), pp. 367–396.

[4] A. Aho, R. Sethi, and J. Ullman, *Compilers – Principles, Techniques, and Tools*, Addison Wesley, 1986.

[5] C. Alleaume, V.-L. Benabou, D. Beras, C. Bidan, N. Craipeau, S. Frénot, G. Gössler, R. Hardouin, J. L. Clainche, D. Le Métayer, M. Maarek, E. Mazza, L. Mé, M.-L. Potet, S. Steer, and V. V. T. Tong, *Liability in software engineering: Overview of the lise approach and illustration on a case study*, Research Report 7148, INRIA, dec 2009.

[6] R. Allen and D. Garlan, *A formal basis for architectural connection*, ACM Trans. Softw. Eng. Methodol., 6 (1997), pp. 213–249.

[7] K. Altisen, G. Gössler, and J. Sifakis, *Scheduler modeling based on the controller synthesis paradigm*, Journal of Real-Time Systems, special issue on "control-theoretical approaches to real-time computing", 23 (2002), pp. 55–84.

[8] R. Alur and D. Dill, *A theory of timed automata*, Theoretical Computer Science, 126 (1994), pp. 183–235.

[9] R. Alur and T. Henzinger, *Local liveness for comopsitional modeling of fair reactive systems*, in Poc. CAV '95, vol. 939 of LNCS, Springer-Verlag, 1995.

[10] ——, *Reactive modules*, in Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS'96), 1996, pp. 207–218.

[11] D. Angeli, *A Lyapunov approach to incremental stability properties*, IEEE Trans. on Automatic Control, 47 (2002), pp. 410–421.

[12] A. Anta and P. Tabuada, *To sample or not to sample: Self-triggered control for nonlinear systems*, IEEE Transactions on Automatic Control, (2010). To appear.

[13] A. Antonik, M. Huth, K. Larsen, U. Nyman, and A. Wasowski, *20 years of modal and mixed specifications*, Bulletin of European Association of Theoretical Computer Science, 1 (2008).

[14] F. Arbab, *Reo: a channel-based coordination model for component composition*, Mathematical Structures in Computer Science, 14 (2004), pp. 329–366.

[15] A. Arnold, G. Point, A. Griffault, and A. Rauzy, *The AltaRica formalism for describing concurrent systems*, Fundam. Inform., 40 (1999), pp. 109–124.

[16] A. Arnold, A. Vincent, and I. Walukiewicz, *Games for synthesis of controllers with partial observation*, Theoretical Computer Science, 28 (2003), pp. 7–34.

[17] A. Arora and S. S. Kulkarni, *Detectors and correctors: A theory of fault-tolerance components*, in International Conference on Distributed Computing Systems (ICDCS), 1998, pp. 436–443.

[18] P. Attie and H. Chockler, *Efficiently verifiable conditions for deadlock-freedom of large concurrent programs*, in Proc. VMCAI'05, R. Cousot, ed., vol. 3385 of LNCS, Springer-Verlag, 2005, pp. 465–481.

[19] P. C. Attie, S. Bensalem, M. Bozga, M. Jaber, J. Sifakis, and F. A. Zaraket, *An abstract framework for deadlock prevention in bip*, in FMOODS/FORTE, D. Beyer and M. Boreale, eds., vol. 7892 of LNCS, Springer, 2013, pp. 161–177.

[20] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, *Metropolis: An integrated electronic system design environment*, IEEE Computer, 36 (2003), pp. 45–52.

[21] H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Rosu, O. Sokolsky, and N. Tillmann, eds., *Runtime Verification - First International Conference, RV 2010. Proceedings*, vol. 6418 of LNCS, Springer, 2010.

[22] A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis, *Rigorous component-based system design using the BIP framework*, IEEE Software, 28 (2011), pp. 41–48.

[23] A. Basu, B. Bonakdarpour, M. Bozga, and J. Sifakis, *Systematic correct construction of self-stabilizing systems: A case study*, in SSS, S. Dolev, J. A. Cobb, M. Fischer, and M. Yung, eds., vol. 6366 of LNCS, Springer, 2010, pp. 4–18.

[24] A. Basu, M. Bozga, and J. Sifakis, *Modeling heterogeneous real-time components in BIP*, in SEFM, IEEE, 2006, pp. 3–12.

[25] A. Basu, M. Gallien, C. Lesire, T.-H. Nguyen, S. Bensalem, F. Ingrand, and J. Sifakis, *Incremental component-based construction and verification of a robotic system*, in ECAI, M. Ghallab, C. Spyropoulos, N.Fakotakis, and N. Avouris, eds., vol. 178 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2008, pp. 631–635.

[26] G. Batt, D. Bergamini, H. de Jong, H. Garavel, and R. Mateescu, *Model checking genetic regulatory networks using GNA and CADP*, in Proc. SPIN'04, vol. 2989 of LNCS, Springer-Verlag, 2004, pp. 158–163.

[27] G. Batt, M. Page, I. Cantone, G. Gössler, P. Monteiro, and H. de Jong, *Efficient parameter search for qualitative models of regulatory networks using symbolic model checking*, Bioinformatics, 26 (2010).

[28] G. Batt, D. Ropers, H. de Jong, J. Geiselmann, R. Mateescu, M. Page, and D. Schneider, *Validation of qualitative models of genetic regulatory networks by model checking: analysis of the nutritional stress response in escherichia coli*, Bioinformatics, 21 (2005), pp. i19–i28.

[29] S. S. Bauer, A. David, R. Hennicker, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski, *Moving from specifications to contracts in component-based design*, in Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, J. de Lara and A. Zisman, eds., vol. 7212 of LNCS, Springer, 2012, pp. 43–58.

[30] H. Baumeister, F. Hacklinger, R. Hennicker, A. Knapp, and M. Wirsing, *A component model for architectural programming*, Electr. Notes Theor. Comput. Sci., 160 (2006), pp. 75–96.

[31] S. Beckers and J. Vennekens, *Counterfactual dependency and actual causation in cp-logic and structural models: a comparison*, in STAIRS, K. Kersting and M. Toussaint, eds., vol. 241 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2012, pp. 35–46.

[32] I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. Trefler, *Explaining counterexamples using causality*, Formal Methods in System Design, 40 (2012), pp. 20–40.

[33] S. Bensalem, M. Bozga, B. Delahaye, C. Jégourel, A. Legay, and A. Nouri, *Statistical model checking qos properties of systems with SBIP*, in ISoLA, T. Margaria and B. Steffen, eds., vol. 7609 of LNCS, Springer, 2012, pp. 327–341.

[34] S. Bensalem, A. Griesmayer, A. Legay, T.-H. Nguyen, J. Sifakis, and R. Yan, *D-finder 2: Towards efficient correctness of incremental design*, in NASA Formal Methods, M. Bobaru, K. Havelund, G. Holzmann, and R. Joshi, eds., vol. 6617 of LNCS, Springer, 2011, pp. 453–458.

[35] A. Benveniste, A. Bouillard, and P. Caspi, *A unifying view of loosely time-triggered architectures*, in EMSOFT, L. Carloni and S. Tripakis, eds., ACM, 2010, pp. 189–198.

[36] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, *Multiple viewpoint contract-based specification and design*, in Proc. FMCO'07, F. de Boer, M. Bonsangue, S. Graf, and W. de Roever, eds., vol. 5382 of LNCS, Springer, 2008, pp. 200–225.

[37] A. Benveniste, B. Caillaud, and R. Passerone, *A generic model of contracts for embedded systems*, Tech. Rep. 6214, INRIA, 2007.

[38] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. LeGuernic, and R. de Simone, *Th synchronous languages twelve years later*, IEEE, (2002).

[39] A. Benveniste, P. Caspi, P. L. Guernic, H. Marchand, J.-P. Talpin, and S. Tripakis, *A protocol for loosely time-triggered architectures*, in Proc. EMSOFT'02, A. Sangiovanni-Vincentelli and J. Sifakis, eds., vol. 2491 of LNCS, Springer-Verlag, 2002.

[40] A. Benveniste, P. Caspi, M. D. Natale, C. Pinello, A. L. Sangiovanni-Vincentelli, and S. Tripakis, *Loosely time-triggered architectures based on communication-by-sampling*, in EMSOFT, C. M. Kirsch and R. Wilhelm, eds., ACM, 2007, pp. 231–239.

[41] A. Benveniste, P. LeGuernic, and C. Jacquemot, *Synchronous programming with events and relations: the SIGNAL language and its semantics*, Science of Computer Programming, 16 (1991), pp. 103–149.

[42] G. Bernat, A. Burns, and A. Llamosí, *Weakly hard real-time systems*, IEEE Transactions on Computers, 50 (2001), pp. 308–321.

[43] G. Bernot, J.-P. Comet, A. Richard, and J. Guespin, *Application of formal methods to biological regulatory networks: Extending Thomas' asynchronous logical approach with temporal logic*, Journal of Theoretical Biology, 229 (2004), pp. 339–348.

[44] G. Bernot and F. Tahi, *Behaviour preservation of a biological regulatory network when embedded into a larger network*, Fundam. Inform., 91 (2009), pp. 463–485.

[45] G. Berry and G. Gonthier, *The ESTEREL synchronous programming language: Design, semantics, implementation*, Science of Computer Programming, 19 (1992), pp. 87–152.

[46] B. Berthomieu, H. Garavel, F. Lang, and F. Vernadat, *Verifying dynamic properties of industrial critical systems using TOPCASED/FIACRE*, ERCIM News, 2008 (2008).

[47] P. Besnard, M.-O. Cordier, and Y. Moinard, *Configurations for inference between causal statements*, in Proc. Knowledge Science, Engineering and Management (KSEM'06), vol. 4092 of LNCS, Springer, 2006, pp. 292–304.

[48] D. Biswas and V. Niemi, *Transforming privacy policies to auditing specifications*, in HASE, T. M. Khoshgoftaar, ed., IEEE Computer Society, 2011, pp. 368–375.

[49] S. Bliudze and J. Sifakis, *The algebra of connectors — structuring interaction in BIP*, in Proc. EMSOFT'07, ACM, 2007, pp. 11–20.

[50] ——, *A notion of glue expressiveness for component-based systems*, in CONCUR, F. van Breugel and M. Chechik, eds., vol. 5201 of LNCS, Springer, 2008, pp. 508–522.

[51] ——, *Causal semantics for the algebra of connectors*, Formal Methods in System Design, 36 (2010), pp. 167–194.

[52] B. Boehm and C. Abts, *Cots integration: Plug and pray?*, IEEE Computer, 32 (1999).

[53] B. Bonakdarpour, M. Bozga, and G. Gössler, *A theory of fault recovery for component-based models*, in SRDS, IEEE, 2011, pp. 265–270.

[54] B. Bonakdarpour, M. Bozga, and G. Gössler, *A theory of fault recovery for component-based models*, in SSS, vol. 7596 of LNCS, Springer, 2012, pp. 314–328.

[55] S. Bornot, G. Gössler, and J. Sifakis, *On the construction of live timed systems*, in Proc. TACAS'00, S. Graf and M. Schwartzbach, eds., vol. 1785 of LNCS, Springer-Verlag, 2000, pp. 109–126.

[56] S. Bouchenak and E. Rutten, *Summary of the 5th international workshop on feedback control implementation and design in computing systems and networks (febid 2010)*, Operating Systems Review, 44 (2010), pp. 38–40.

[57] M. Bozga, M. Jaber, N. Maris, and J. Sifakis, *Modeling dynamic architectures using dy-bip*, in Software Composition, T. Gschwind, F. D. Paoli, V. Gruhn, and M. Book, eds., vol. 7306 of LNCS, Springer, 2012, pp. 1–16.

[58] M. Broy, *Compositional refinement of interactive systems*, J. ACM, 44 (1997), pp. 850–891.

[59] E. Bruneton, T. Coupaye, and J. Stefani, *Recursive and dynamic software composition with sharing*, in Proc. WCOP'02, 2002.

[60] R. Bruni, I. Lanese, and U. Montanari, *A basic algebra of stateless connectors*, Theor. Comput. Sci., 366 (2006), pp. 98–120.

[61] G. Bruns, D. Dantas, and M. Huth, *A simple and expressive semantic framework for policy composition in access control*, in Proc. FMSE'07, ACM, 2007, pp. 12–21.

[62] H. Busch, W. Sandmann, and V. Wolf, *A numerical aggregation algorithm for the enzyme-catalyzed substrate conversion*, in Proc. CMSB'06, 2006, pp. 298–311.

[63] B. Caillaud, B. Delahaye, K. Larsen, A. Legay, M. Pedersen, and A. Wasowski, *Compositional design methodology with Constraint Markov Chains*, in QEST, IEEE Computer Society, 2010, pp. 123–132.

[64] B. Caillaud and J.-B. Raclet, *Ensuring reachability by design*, in ICTAC, A. Roychoudhury and M. D'Souza, eds., vol. 7521 of LNCS, Springer, 2012, pp. 213–227.

[65] J. Cámara, A. Girard, and G. Gössler, *Safety controller synthesis for switched systems using multi-scale symbolic models*, in CDC-ECC, IEEE, 2011, pp. 520–525.

[66] ——, *Synthesis of switching controllers using approximately bisimilar multiscale abstractions*, in HSCC, M. Caccamo, E. Frazzoli, and R. Grosu, eds., ACM, 2011, pp. 191–200.

[67] C. Canal, P. Poizat, and G. Salaün, *Model-based adaptation of behavioral mismatching components*, IEEE Trans. Software Eng., 34 (2008), pp. 546–563.

[68] I. Cantone, L. Marucci, F. Iorio, M. Ricci, V. Belcastro, M. Bansal, S. Santini, M. di Bernardo, D. di Bernardo, and M. Cosma, *A yeast synthetic network for in vivo assessment of reverse-engineering and modeling approaches*, Cell, 137 (2009), pp. 172–181.

[69] J. Carlson, J. Hakansson, and P. Pettersson, *SaveCCM: An analysable component model for real-time systems*, Electr. Notes Theor. Comput. Sci., 160 (2006), pp. 127–140.

[70] J. Carmona, J. Cortadella, M. Kishinevsky, and A. Taubin, *Elastic circuits*, IEEE Trans. on CAD of Integrated Circuits and Systems, 28 (2009), pp. 1437–1455.

[71] P. Caspi and A. Benveniste, *Time-robust discrete control over networked loosely time-triggered architectures*, in CDC, IEEE, 2008, pp. 3595–3600.

[72] N. Chabrier and F. Fages, *Symbolic model checking of biochemical networks*, in Proc. CMSB'03, 2003.

[73] C. Chaouiya, E. Remy, P. Ruet, and D. Thieffry, *Qualitative modelling of genetic networks: From logical regulatory graphs to standard petri nets*, in Proc. ICATPN'04, J. Cortadella and W. Reisig, eds., vol. 3099 of LNCS, Springer-Verlag, 2004, pp. 137–156.

[74] K. Chatterjee and T. Henzinger, *Assume-guarantee synthesis*, in Proc. TACAS'07, O. Grumberg and M. Huth, eds., vol. 4424 of LNCS, Springer-Verlag, 2007, pp. 261–275.

[75] T. Chen, C. Chilton, B. Jonsson, and M. Z. Kwiatkowska, *A compositional specification theory for component behaviours*, in ESOP, H. Seidl, ed., vol. 7211 of LNCS, Springer, 2012, pp. 148–168.

[76] S.-C. Cheung, D. Giannakopoulou, and J. Kramer, *Verification of liveness properties using compositional reachability analysis*, in ESEC / SIGSOFT FSE, M. Jazayeri and H. Schauer, eds., vol. 1301 of LNCS, Springer, 1997, pp. 227–243.

[77] S.-C. Cheung and J. Kramer, *Context constraints for compositional reachability analysis*, ACM Trans. Softw. Eng. Methodol., 5 (1996), pp. 334–377.

[78] ——, *Checking safety properties using compositional reachability analysis*, ACM Trans. Softw. Eng. Methodol., 8 (1999), pp. 49–78.

[79] C. Chilton, B. Jonsson, and M. Z. Kwiatkowska, *Assume-guarantee reasoning for safe component behaviours*, in FACS, C. S. Pasareanu and G. Salaün, eds., vol. 7684 of LNCS, Springer, 2012, pp. 92–109.

[80] H. Chockler and J. Halpern, *Responsibility and blame: A structural-model approach*, J. Artif. Intell. Res. (JAIR), 22 (2004), pp. 93–115.

[81] L. Cholvy, F. Cuppens, and C. Saurel, *Towards a logical formalization of responsibility*, in Proc. ICAIL'97, 1997, pp. 233–242.

[82] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, *Nusmv 2: An opensource tool for symbolic model checking*, in Computer Aided Verification, E. Brinksma and K. Larsen, eds., vol. 2404 of LNCS, Springer, 2002, pp. 241–268.

[83] D. Clarke, D. Costa, and F. Arbab, *Modelling coordination in biological systems*, in ISoLA, T. Margaria and B. Steffen, eds., vol. 4313 of LNCS, Springer, 2004, pp. 9–25.

[84] E. Clarke and E. Emerson, *Design and synthesis of synchronization skeletons using branching-time temporal logic*, Logic of Programs, (1981), pp. 52–71.

[85] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 1999.

[86] E. M. Clarke, D. E. Long, and K. L. McMillan, *Compositional model checking*, in LICS, IEEE Computer Society, 1989, pp. 353–362.

[87] A. Cohen, M. Duranton, C. Eisenbeis, C. Pagetti, F. Plateau, and M. Pouzet, *N-Synchronous Kahn Networks: a Relaxed Model of Synchrony for Real-Time Systems*, in Proc. POPL'06, 2006.

[88] `http://www.combest.eu/`.

[89] J.-P. Comet, H. Klaudel, and S. Liauzu, *Modeling multi-valued genetic regulatory networks using high-level petri nets*, in Proc. ICATPN'05, G. Ciardo and P. Darondeau, eds., vol. 3536 of LNCS, Springer-Verlag, 2005, pp. 208–227.

[90] F. Corblin, S. Tripodi, E. Fanchon, D. Ropers, and L. Trilling, *A declarative constraint-based method for analyzing discrete genetic regulatory networks*, Biosystems, 98 (2009), pp. 91 – 104.

[91] J. Cortadella, A. Kondratyev, L. Lavagno, C. Passerone, and Y. Watanabe, *Quasi-static scheduling of independant tasks for reactive systems*, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 24 (2005), pp. 1492–1514.

[92] D. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo, *A network-centric approach to embedded software for tiny devices*, in Proc. EMSOFT'01, T. Henzinger and C. M. Kirsch, eds., vol. 2211 of LNCS, Springer-Verlag, 2001, pp. 114–130.

[93] T. Dang, C. L. Guernic, and O. Maler, *Computing reachable states for non-linear biological models*, in CMSB, 2009, pp. 126–141.

[94] G. B. Dantzig and M. N. Thapa, *Linear programming*, vol. 1: Introduction, Springer, 1997.

[95] P. Darondeau, B. Genest, P. Thiagarajan, and S. Yang, *Quasi-static scheduling of communicating tasks*, Inf. Comput., 208 (2010), pp. 1154–1168.

[96] L. de Alfaro, L. D. da Silva, M. Faella, A. Legay, P. Roy, and M. Sorea, *Sociable interfaces*, in Proc. Frontiers of Combining Systems, vol. 3717 of LNCS, Springer-Verlag, 2005, pp. 81–105.

[97] L. de Alfaro and T. Henzinger, *Interface automata*, in Proc. 9th Annual Symposium on Foundations of Software Engineering (FSE), ACM Press, 2001, pp. 109–120.

[98] L. de Alfaro, T. Henzinger, and R. Jhala, *Compositional methods for probabilistic systems*, in CONCUR, K. Larsen and M. Nielsen, eds., vol. 2154 of LNCS, Springer, 2001, pp. 351–365.

[99] H. de Jong, *Modeling and simulation of genetic regulatory systems: A literature review*, Journal of Computational Biology, 9 (2002), pp. 69–105.

[100] H. de Jong, J. Geiselmann, C. Hernandez, and M. Page, *Genetic Network Analyzer: Qualitative simulation of genetic regulatory networks*, Bioinformatics, 19 (2003), pp. 336–344.

[101] H. de Jong, J.-L. Gouzé, C. Hernandez, M. Page, T. Sari, and J. Geiselmann, *Qualitative simulation of genetic regulatory networks using piecewise-linear models*, Bulletin of Mathematical Biology, 66 (2004), pp. 301–340.

[102] W.-P. de Roever, H. Langmaack, and A. Pnueli, eds., *Compositionality: The Significant Difference*, vol. 1536 of LNCS, Springer-Verlag, 1997.

[103] B. Delahaye, B. Caillaud, and A. Legay, *Probabilistic contracts: a compositional reasoning methodology for the design of systems with stochastic and/or non-deterministic aspects*, Formal Methods in System Design, 38 (2011), pp. 1–32.

[104] J. Derrick, G. Schellhorn, and H. Wehrheim, *Proving linearizability via non-atomic refinement*, in Proc. IFM'07, J. Davies and J. Gibbons, eds., vol. 4591 of LNCS, Springer, 2007, pp. 195–214.

[105] J. Derrick and H. Wehrheim, *Using coupled simulations in non-atomic refinement*, in Proc. ZB'03, D. B. et al., ed., vol. 2651 of LNCS, Springer, 2003, pp. 127–147.

[106] L. Doyen, T. A. Henzinger, A. Legay, and D. Nickovic, *Robustness of sequential circuits*, in ACSD, L. Gomes, V. Khomenko, and J. M. Fernandes, eds., IEEE Computer Society, 2010, pp. 77–84.

[107] L. Doyen and T. P. T.A. Henzinger, B. Jobstmann, *Interface theories with component reuse*, in Proc. EMSOFT'08, ACM, 2008, pp. 79–88.

[108] V. d'Silva, S. Ramesh, and A. Sowmya, *Synchronous protocol automata: A framework for modelling and verification of SoC communication architectures*, IEE Proc. Computers & Digital Techniques, 152 (2005), pp. 20–27.

[109] E. Fabre, A. Benveniste, S. Haar, and C. Jard, *Distributed monitoring of concurrent and asynchronous systems*, Discrete Event Dynamic Systems, 15 (2005), pp. 33–84.

[110] F. Fages and S. Soliman, *Abstract interpretation and types for systems biology*, Theor. Comput. Sci., 403 (2008), pp. 52–70.

[111] H. Fecher, M. Leucker, and V. Wolf, *Don't know in probabilistic systems*, in International Workshop on Model Checking Software, SPIN'06, vol. 3925 of LNCS, Springer, 2006, pp. 71–88.

[112] S. Fenech, G. Pace, J. Okika, A. Ravn, and G. Schneider, *On the specification of full contracts*, Electr. Notes Theor. Comput. Sci., 253 (2009), pp. 39–55.

[113] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu, *CADP: A protocol validation and verification toolbox*, in Proc. CAV '96, R. Alur and T. Henzinger, eds., vol. 1102 of LNCS, Springer-Verlag, 1996, pp. 437–440.

[114] G. Feuillade and S. Pinchinat, *Modal specifications for the control theory of discrete event systems*, Discrete Event Dynamic Systems, 17 (2007), pp. 211–232.

[115] G. Fey, S. Staber, R. Bloem, and R. Drechsler, *Automatic fault localization for property checking*, IEEE Trans. on CAD of Integrated Circuits and Systems, 27 (2008), pp. 1138–1149.

[116] J. L. Fiadeiro, *Categories for software engineering*, Springer, 2005.

[117] C. Fidge, *Timestamps in message-passing systems that preserve the partial ordering*, in Proc. ACSC'88, K. Raymond, ed., 1988, p. 56 66.

[118] A. Filippov, *Differential equations with discontinuous righthand side*, Mathematics and its Applications, 18 (1988).

[119] A. Finkel, *The minimal coverability graph for petri nets*, in Applications and Theory of Petri Nets, G. Rozenberg, ed., vol. 674 of LNCS, Springer, 1991, pp. 210–243.

[120] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, *Spaceex: Scalable verification of hybrid systems*, in CAV, G. Gopalakrishnan and S. Qadeer, eds., vol. 6806 of LNCS, Springer, 2011, pp. 379–395.

[121] S. Frénot and J. Ponge, *Logos: An automatic logging framework for service-oriented architectures*, in EUROMICRO-SEAA, IEEE Computer Society, 2012, pp. 224–227.

[122] J. Fromentin, J. Comet, P. Le Gall, and O. Roux, *Analysing gene regulatory networks by both constraint programming and model-checking*, in Proc. EMBC'07, 29th IEEE EMBS Annual International Conference, 2007, pp. 4595–4598.

[123] D. Garlan, R. Allen, and J. Ockerbloom, *Architectural mismatch: Why reuse is so hard*, IEEE Software, 12 (1995).

[124] F. Gärtner, *Fundamentals of fault-tolerant distributed computing in asynchronous environments*, ACM Computing Surveys, 31 (1999), pp. 1–26.

[125] B. Gaudin and H. Marchand, *An efficient modular method for the control of concurrent discrete event systems: A language-based approach*, Discrete Event Dynamic System, 17 (2007), pp. 179–209.

[126] R. Ghosh, A. Tiwari, and C. Tomlin, *Automated symbolic reachability analysis; with application to delta-notch signaling automata*, in Proc. HSCC'03, O. Maler and A. Pnueli, eds., vol. 2623 of LNCS, Springer-Verlag, 2003, pp. 233–248.

[127] R. Ghosh and C. Tomlin, *Lateral inhibition through delta-notch signaling: A piecewise affine hybrid model*, in Proc. HSCC'01, M. D. Benedetto and A. Sangiovanni-Vincentelli, eds., vol. 2034 of LNCS, Springer-Verlag, 2001, pp. 232–246.

[128] ——, *Symbolic reachable set computation of piecewise affine hybrid automata and its application to biological modeling: Delta-Notch protein signaling*, IEE Trans. on Systems Biology, 1 (2004), pp. 170–183.

[129] A. Girard, *Reachability of uncertain linear systems using zonotopes*, in Proc. HSCC'05, 2005, pp. 291–305.

[130] ——, *Synthesis using approximately bisimilar abstractions: time-optimal control problems*, in IEEE Conf. on Decision and Control, 2010.

[131] A. Girard and G. Pappas, *Approximation metrics for discrete and continuous systems*, IEEE Trans. on Automatic Control, 52 (2007), pp. 782–798.

[132] A. Girard, G. Pola, and P. Tabuada, *Approximately bisimilar symbolic models for incrementally stable switched systems*, IEEE Transactions on Automatic Control, 55 (2010), pp. 116–126.

[133] A. Girard, G. Pola, and P. Tabuada, *Approximately bisimilar symbolic models for incrementally stable switched systems*, IEEE Transactions on Automatic Control, 55 (2010), pp. 116–126.

[134] L. Glass and S. Kauffman, *The logical analysis of continuous, non-linear biochemical control networks*, Journal of Theoretical Biology, 39 (1973), pp. 103–129.

[135] U. Goltz, *Synchronic distance*, in Advances in Petri Nets, W. Brauer, W. Reisig, and G. Rozenberg, eds., vol. 254 of LNCS, Springer, 1986, pp. 338–358.

[136] A. Gonzalez, A. Naldi, L. SÃ¡nchez, D. Thieffry, and C. Chaouiya, *Ginsim: A software suite for the qualitative modelling, simulation and analysis of regulatory networks*, Biosystems, 84 (2006), pp. 91 – 100. Dynamical Modeling of Biological Regulatory Networks.

[137] G. Gössler, *Compositional Modelling of Real-Time Systems — Theory and Practice*, PhD thesis, Université Joseph Fourier, Grenoble, France, 2001.

[138] ——, *Component-based design of heterogeneous reactive systems in* PROMETHEUS, Research Report 6057, INRIA, 2006.

[139] ——, *Compositional reachability analysis of genetic networks*, in CMSB'06, C. Priami, ed., vol. 4210 of LNBI, Springer, 2006, pp. 212–226.

[140] ——, *Compositional strategy mapping*, in Proc. FSEN'09, F. Arbab and M. Sirjani, eds., vol. 5961 of LNCS, Springer, 2010, pp. 340–354.

[141] ——, *Component-based modeling and reachability analysis of genetic networks*, IEEE/ACM Trans. Comput. Biology Bioinform., 8 (2011), pp. 672–682.

[142] G. Gössler and L. Aştefănoaei, *Blaming in component-based real-time systems*, in EMSOFT, 2014. To appear.

[143] G. Gössler, S. Graf, M. Majster-Cederbaum, M. Martens, and J. Sifakis, *An approach to modelling and verification of component based systems*, in Proc. SOFSEM'07, vol. 4362 of LNCS, Springer-Verlag, 2007, pp. 295–308.

[144] ——, *Ensuring Properties of Interaction Systems by Construction*, vol. 4444 of LNCS, Springer, 2007, pp. 201–224.

[145] G. Gössler and D. Le Métayer, *A General Trace-Based Framework of Logical Causality*, in FACS - 10th International Symposium on Formal Aspects of Component Software - 2013, vol. 8348 of LNCS, Nanchang, China, 2013, Springer.

[146] G. Gössler, D. Le Métayer, E. Mazza, M.-L. Potet, and L. Astefanoaei, *Apport des méthodes formelles pour l'exploitation de logs informatiques dans un contexte contractuel*, TSI, 33 (2014), pp. 63–84.

[147] G. Gössler, D. Le Métayer, and J.-B. Raclet, *Causality analysis in contract violation*, in Barringer et al. [21], pp. 270–284.

[148] G. Gössler and J.-B. Raclet, *Modal contracts for component-based design*, in Proc. SEFM'09, IEEE, 2009, pp. 295–303.

[149] G. Gössler and G. Salaün, *Realizability of choreographies for services interacting asynchronously*, in FACS, LNCS, Springer, 2011. to appear.

[150] G. Gössler and A. Sangiovanni-Vincentelli, *Compositional modeling in metropolis*, in Proc. EMSOFT'02, A. Sangiovanni-Vincentelli and J. Sifakis, eds., vol. 2491 of LNCS, Springer-Verlag, 2002.

[151] G. Gössler and J. Sifakis, *Component-based construction of deadlock-free systems (extended abstract)*, in Proc. FSTTCS'03, vol. 2914 of LNCS, Springer-Verlag, 2003.

[152] ——, *Composition for component-based modeling*, in Proc. FMCO'02, F. de Boer, M. Bonsangue, S. Graf, and W.-P. de Roever, eds., vol. 2852 of LNCS, Springer-Verlag, 2003.

[153] ——, *Priority systems*, in Proc. FMCO'03, F. de Boer, M. Bonsangue, S. Graf, and W.-P. de Roever, eds., vol. 3188 of LNCS, Springer-Verlag, 2004, pp. 314–329.

[154] ——, *Composition for component-based modeling*, Science of Computer Programming, 55 (2005), pp. 161–183.

[155] G. Gössler, D. N. Xu, and A. Girault, *Probabilistic contracts for component-based design*, Formal Methods in System Design, 41 (2012), pp. 211–231.

[156] J.-L. Gouzé and T. Sari, *A class of piecewise linear differential equations arising in biological models*, Dynamical Systems, 17 (2003), pp. 299–316.

[157] S. Graf and B. Steffen, *Compositional minimization of finite state systems*, in CAV, E. Clarke and R. Kurshan, eds., vol. 531 of LNCS, Springer, 1990, pp. 186–196.

[158] P. Green, *Protocol conversion*, IEEE Trans. on Communications, 34 (1986), pp. 257–268.

[159] A. Groce, S. Chaki, D. Kroening, and O. Strichman, *Error explanation with distance metrics*, STTT, 8 (2006), pp. 229–247.

[160] N. Halbwachs, *Synchronous Programming of Reactive Systems*, Kluwer, 1993.

[161] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, *The synchronous dataflow programming language* lustre, Proceedings of the IEEE, 79 (1991), pp. 1305–1320.

[162] J. Halpern and J. Pearl, *Causes and explanations: A structural-model approach. part I: Causes*, British Journal for the Philosophy of Science, 56 (2005), pp. 843–887.

[163] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindberg, J. Lundback, and K.-L. Lundback, *The rubus component model for resource constrained real-time systems*, in Industrial Embedded Systems, 2008. SIES 2008. International Symposium on, June 2008, pp. 177–183.

[164] L. Hartwell, J. Hopfield, S. Leibler, and A. Murray, *From molecular to modular cell biology*, Nature, 402 (1999), pp. C47–C52.

[165] T. A. Henzinger and J. Sifakis, *The embedded systems design challenge*, in Proc. FM'06), vol. 4085 of LNCS, Springer, 2006, pp. 1–15.

[166] M. Herlihy and J. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Trans. on Programming Languages and Systems, 12 (1990), pp. 463–492.

[167] H. Hermanns, *Interactive Markov Chains: The Quest for Quantified Quality*, vol. 2428 of LNCS, Springer, 2002.

[168] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

[169] D. Hume, *An Enquiry Concerning Human Understanding*, 1748.

[170] P. Inverardi, D. Yankelevich, and A. Wolf, *Static checking of system behaviors using derived component assumptions*, ACM TOSEM Volume 9, Number 3, (2000).

[171] B. Jobstmann, S. Staber, A. Griesmayer, and R. Bloem, *Finding and fixing faults*, J. Comput. Syst. Sci., 78 (2012), pp. 441–460.

[172] S. P. Jones, J.-M. Eber, and J. Seward, *Composing contracts: an adventure in financial engineering, functional pearl*, in ICFP, 2000, pp. 280–292.

[173] B. Jonsson and K. Larsen, *Specification and refinement of probabilistic processes*, in Symposium on Logic in Computer Science, LICS'91, 1991, pp. 266–277.

[174] J.-P. Katoen, D. Klink, and M. Neuhäusser, *Compositional abstraction for stochastic systems*, in Proc. FORMATS'09, 2009, pp. 195–211.

[175] N. Kaveh and W. Emmerich, *Object system*, 8th FSE/ESEC, (2001).

[176] J. Kessels, *Arbitration without common modifiable variables*, Acta Informatica, 17 (1982), pp. 135–141.

[177] K. Keutzer, S. Malik, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, *System level design: Orthogonalization of concerns and platform-based design*, IEEE Trans. on Computer-Aided Design, 19 (2000).

[178] H. Kitano, *Looking beyond the details: a rise in system-oriented approaches in genetics an molecular biology*, Current Genetics, 41 (2002), pp. 1–10.

[179] C. Kossentini and P. Caspi, *Approximation, sampling and voting in hybrid computing systems*, in HSCC, J. P. Hespanha and A. Tiwari, eds., vol. 3927 of LNCS, Springer, 2006, pp. 363–376.

[180] M. Kuntz, F. Leitner-Fischer, and S. Leue, *From probabilistic counterexamples via causality to fault trees*, in SAFECOMP, F. Flammini, S. Bologna, and V. Vittorini, eds., vol. 6894 of LNCS, Springer, 2011, pp. 71–84.

[181] R. Küsters, T. Truderung, and A. Vogt, *Accountability: definition and relationship to verifiability*, in ACM Conference on Computer and Communications Security, 2010, pp. 526–535.

[182] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, *Assume-guarantee verification for probabilistic systems*, in TACAS, 2010, pp. 23–37.

[183] S. Lam and A. U. Shankar, *A theory of interfaces and modules i-composition theorem*, IEEE Trans. Software Eng., 20 (1994), pp. 55–71.

[184] C. Lambertz and M. E. Majster-Cederbaum, *Efficient deadlock analysis of component-based software architectures*, Sci. Comput. Program., 78 (2013), pp. 2488–2510.

[185] L. Lamport, *Time, clocks, and the ordering of events in a distributed system*, CACM, 21 (1978), pp. 558–565.

[186] ——, *The temporal logic of actions*, ACM T. on Programming Languages and Systems, 16 (1994), pp. 872–923.

[187] I. Lanese, C. Mezzina, and J.-B. Stefani, *Reversing higher-order pi*, in CONCUR 2010 - Concurrency Theory, P. Gastin and F. Laroussinie, eds., vol. 6269 of LNCS, Springer, 2010, pp. 478–493.

[188] F. Lang, *Refined interfaces for compositional verification*, in FORTE, E. Najm, J.-F. Pradat-Peyre, and V. Donzeau-Gouge, eds., vol. 4229 of LNCS, Springer, 2006, pp. 159–174.

[189] K. Larsen, *Modal specifications*, in Proc. International Workshop on Automatic Verification Methods for Finite State Systems, vol. 407 of LNCS, Springer, 1989, pp. 232–246.

[190] K. Larsen, U. Nyman, and A. Wasowski, *Interface input/output automata*, in Proc. FM'06, J. Misra, T. Nipkow, and E. Sekerinski, eds., vol. 4085 of LNCS, Springer-Verlag, 2006, pp. 92–97.

[191] ——, *On modal refinement and consistency*, in Proc. CONCUR'07, vol. 4703 of LNCS, Springer, 2007, pp. 105–119.

[192] K. Larsen, B. Steffen, and C. Weise, *A constraint oriented proof methodology based on modal transition systems*, in Proc. TACAS'95, LNCS, Springer, 1995, pp. 17–40.

[193] K. G. Larsen, U. Nyman, and A. Wasowski, *Modal I/O automata for interface and product line theories*, in Programming Languages and Systems, 16th European Symposium on Programming, ESOP'07, vol. 4421 of LNCS, Springer, 2007, pp. 64–79.

[194] E. Le Corronc, A. Girard, and G. Gössler, *Mode Sequences as Symbolic States in Abstractions of Incrementally Stable Switched Systems*, in CDC - 52nd Conference on Decision and Control - 2013, IEEE, Dec. 2013, pp. 3225–3230.

[195] E. Lee, *Overview of the Ptolemy project*, Tech. Rep. UCB/ERL M03/25, University of California at Berkeley, 2003.

[196] E. Lee and A. Sangiovanni-Vincentelli, *A framework for comparing models of computation*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17 (1998), pp. 1217–1229.

[197] I. Lee, O. Sokolsky, S. Chen, J. Hatcliff, E. Jee, B. Kim, A. L. King, M. Mullen-Fortino, S. Park, A. Roederer, and K. K. Venkatasubramanian, *Challenges and research directions in medical cyber-physical systems*, Proceedings of the IEEE, 100 (2012), pp. 75–90.

[198] F. Leitner-Fischer and S. Leue, *Causality checking for complex system models*, in VMCAI, R. Giacobazzi, J. Berdine, and I. Mastroeni, eds., vol. 7737 of LNCS, Springer, 2013, pp. 248–267.

[199] D. Lewis, *Counterfactuals*, Blackwell, 2nd ed., 2000.

[200] Z. Liu and M. Joseph, *Specification and verification of fault-tolerance, timing, and scheduling*, ACM T. on Programming Languages and Systems, 21 (1999), pp. 46–89.

[201] D. C. Luckham, J. Vera, D. Bryan, L. Augustin, and F. Belz, *Partial orderings of event sets and their application to prototyping concurrent, timed systems*, Journal of Systems and Software, 21 (1993), pp. 253 – 265.

[202] G. Lüttgen and W. Vogler, *Conjunction on processes: Full abstraction via ready-tree semantics*, TCS, 373 (2007), pp. 19–40.

[203] N. A. Lynch and M. R. Tuttle, *An introduction to input/output automata*, CWI-Quarterly, 2 (1989), pp. 219–246.

[204] M. Majster-Cederbaum, M. Martens, and C. Minnameier, *Liveness in interaction systems*, Electr. Notes Theor. Comput. Sci., 215 (2008), pp. 57–74.

[205] O. Maler, A. Pnueli, and J. Sifakis, *On the synthesis of discrete controllers for timed systems*, in STACS'95, E. Mayr and C. Puech, eds., vol. 900 of LNCS, Springer-Verlag, 1995, pp. 229–242.

[206] G. Marnellos, G. Deblandre, E. Mjolsness, and G. Kintner, *Delta-Notch lateral inhibitory patterning in the emergence of ciliated cells in Xenopus: Experimental observations and a gene network model*, in Proc. PSB'00, vol. 5, World Scientific Publishing, 2000, pp. 326–337.

[207] F. Mattern, *Virtual time and global states of distributed systems*, in Proc. Workshop on Parallel and Distributed Algorithms, M. Cosnard, ed., Elsevier, 1988, p. 215 226.

[208] M. Mazo Jr. and P. Tabuada, *Approximate time-optimal control via approximate alternating simulations*, in ACC, 2010, pp. 10201–10206.

[209] P. Menzies, *Counterfactual theories of causation*, in Stanford Encyclopedia of Philosophy, E. Zalta, ed., http://plato.stanford.edu/entries/causation-counterfactual/, 2009.

[210] B. Meyer, *Advances in Object-Oriented Software Engineering*, Prentice Hall, 1991, ch. Design by Contract, pp. 1–50.

[211] ——, *Applying "design by contract"*, IEEE Computer, 25 (1992), pp. 40–51.

[212] R. Milner, *Calculi for synchrony and asynchrony*, Theoretical Computer Science, 25 (1983), pp. 267–310.

[213] ——, *Communication and Concurrency*, Prentice Hall, 1989.

[214] R. Milner, J. Parrow, and D. Walker, *A calculus of mobile processes*, Inf. Comput., 100 (1992), pp. 1–77.

[215] M. Moore, *Causation and Responsibility*, Oxford, 1999.

[216] C. Morgan, A. McIver, and K. Seidel, *Probabilistic predicate transformers*, ACM Trans. Program. Lang. Syst., 18 (1996), pp. 325–353.

[217] S. Mouelhi, A. Girard, and G. Gössler, *CoSyMA: a tool for controller synthesis using multi-scale abstractions*, in Proceedings of the 16th international conference on Hybrid systems: computation and control, C. Belta and F. Ivancic, eds., ACM, 2013, pp. 83–88.

[218] A. Naldi, D. Thieffry, and C. Chaouiya, *Decision diagrams for the representation and analysis of logical models of genetic networks*, in Proc. CMSB 2006, M. Calder and S. Gilmore, eds., vol. 4695 of LNBI, Springer, 2007, pp. 233–247.

[219] R. Passerone, L. de Alfaro, T. Henzinger, and A. Sangiovanni-Vincentelli, *Convertibility verification and converter synthesis: Two faces of the same coin*, in Proc. International Conference on Computer Aided Design, 2002.

[220] J. Pearl, *Causal inference in statistics: An overview*, Statistics Surveys, 3 (2009), pp. 96–146.

[221] C. Petri, *Kommunikation mit Automaten*, PhD thesis, University of Bonn, 1996.

[222] C. Piazza, M. Antoniotti, V. Mysore, A. Policriti, F. Winkler, and B. Mishra, *Algorithmic algebraic model checking I: Challenges form systems biology*, in Proc. CAV'05, vol. 3576 of LNCS, Springer-Verlag, 2005, pp. 5–19.

[223] G. Pola, A. Girard, and P. Tabuada, *Approximately bisimilar symbolic models for nonlinear control systems*, Automatica, 44 (2008), pp. 2508–2516.

[224] G. Pola, A. Girard, and P. Tabuada, *Approximately bisimilar symbolic models for nonlinear control systems*, Automatica, 44 (2008), pp. 2508–2516.

[225] G. Pola and P. Tabuada, *Symbolic models for nonlinear control systems: Alternating approximate bisimulations*, SIAM J. on Con. and Opt., 48 (2009), pp. 719–733.

[226] J.-P. Queille and J. Sifakis, *Specification and verification of concurrent systems in CESAR*, in Proc. International Symposium on Programming, vol. 137 of LNCS, Springer-Verlag, 1982, pp. 337–351.

[227] S. Quinton and S. Graf, *Contract-based verification of hierarchical systems of components*, in Proc. SEFM'08, IEEE, 2008, pp. 377–281.

[228] J.-B. Raclet, *Quotient de spécifications pour la réutilisation de composants*, PhD thesis, Université de Rennes I, december 2007. (In French).

[229] ———, *Residual for component specifications*, in Proc. FACS'07, ENTCS, 2007.

[230] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone, *A modal interface theory for component-based design*, Fundam. Inform., 108 (2011), pp. 119–149.

[231] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone, *Why modalities are good for interface theories?*, in Proc. ACSD'09, IEEE, 2009.

[232] P. Ramadge and W. Wonham, *Supervisory control of a class of discrete event processes*, SIAM J. Control and Optimization, 25 (1987).

[233] W. Reisig, *Petri Nets: An Introduction*, vol. 4 of Monographs in Theoretical Computer Science. An EATCS Series, Springer, 1985.

[234] R. Reiter, *A theory of diagnosis from first principles*, Artif. Intell., 32 (1987), pp. 57–95.

[235] A. Rensink, *Action contraction*, in Proc. CONCUR'00, C. Palamidessi, ed., vol. 1877 of LNCS, Springer, 2000, pp. 290–305.

[236] O. Resendis-Antonio, J. Freyre-González, R. Menchaca-Méndez, R. Gutiérrez-Ríos, A. Martínez-Antonio, C. Avila-Sánchez, and J. Collado-Vides, *Modular analysis of the transcriptional regulatory network of e. coli*, Trends in Genetics, 21 (2005), pp. 16–20.

[237] P. Richard, H. Cormerais, and J. Buisson, *A generic design methodology for sliding mode control of switched systems*, Nonlinear Analysis: Hybrid Systems and Applications, 65 (2006), pp. 1751–1772.

[238] P. Roop, A. Girault, R. Sinha, and G. Gössler, *Specification enforcing refinement for convertibility verification*, in Proc. ACSD'09, S. Edwards, R. Lorenz, and W. Vogler, eds., IEEE, 2009, pp. 148–157.

[239] J. Rushby, *Partitioning for safety and security: Requirements, mechanisms, and assurance*, Tech. Rep. CR-1999-209347, NASA Langley Research Center, 1999.

[240] N. B. Said, T. Abdellatif, S. Bensalem, and M. Bozga, *Model-driven information flow security for component-based systems*, in From Programs to Systems – The Systems Perspective in Computing, ETAPS Workshop in Honor of Joseph Sifakis, S. Bensalem, Y. Lakhnech, and A. Legay, eds., vol. 8415 of LNCS, Springer, 2014, pp. 1–20.

[241] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, *Diagnosability of discrete-event systems*, IEEE Transactions on Automatic Control, 40 (1995), pp. 1555–1575.

[242] D. Schultza, G. Wolynesa, E. Ben-Jacob, and J. Onuchica, *Deciding fate in adverse times: Sporulation and competence in* bacillus subtilis, PNAS, 106 (2009), pp. 21027–21034.

[243] http://licit.inrialpes.fr/lise/index.html.

[244] http://sites.google.com/site/vedecy/.

[245] M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli, *Quasi-static scheduling of embedded software using equal conflict nets*, in Proc. ICATPN'99, 1999.

[246] J. Sifakis, *Modeling real-time systems-challenges and work directions*, in EMSOFT, T. Henzinger and C. Kirsch, eds., vol. 2211 of LNCS, Springer, 2001, pp. 373–389.

[247] J. Sifakis, *Rigorous system design*, Foundations and Trends in Electronic Design Automation, 6 (2013), pp. 293–362.

[248] G. Smeding, *Verification of Weakly-Hard Requirements on Quasi-Synchronous Systems*, PhD thesis, University of Grenoble, 2013.

[249] G. Smeding and G. Gössler, *A correlation preserving performance analysis for stream processing systems*, in MEMOCODE, IEEE, 2012, pp. 11–20.

[250] ST Microelectronics and CEA, *Platform 2012: A many-core programmable accelerator for ultra- efficient embedded computing in nanometer technology.* White paper, Nov. 2010.

[251] C. Szyperski, D. Gruntz, and S. Murer, *Component Software*, Addison Wesley, 2nd ed., 2002.

[252] P. Tabuada, *Verification and Control of Hybrid Systems - A Symbolic Approach*, Springer, 2009.

[253] Y. Tazaki and J. Imura, *Approximately bisimilar discrete abstractions of nonlinear systems using variable-resolution quantizers*, in America Control Conference, 2010, pp. 1015–1020.

[254] L. Thiele, S. Chakraborty, and M. Naedele, *Real-time calculus for scheduling hard real-time systems*, in Proc. ISCAS'00, vol. 4, 2000, pp. 101–104.

[255] R. Thomas, *Boolean formalisation of genetic control circuits*, J. Theor. Biol., 42 (1973), pp. 565–583.

[256] R. Thomas and R. d'Ari, *Biological Feedback*, CRC Press, 1990.

[257] M. Tivoli, P. Fradet, A. Girault, and G. Gössler, *Adaptor synthesis for real-time components*, in Proc. TACAS'07, O. Grumberg and M. Huth, eds., vol. 4424 of LNCS, Springer, 2007.

[258] S. Tripakis, B. Lickly, T. Henzinger, and E. Lee, *A theory of synchronous relational interfaces*, ACM Trans. Program. Lang. Syst., 33 (2011), pp. 14:1–14:41.

[259] S. Tripakis, C. Pinello, A. Benveniste, A. L. Sangiovanni-Vincentelli, P. Caspi, and M. D. Natale, *Implementing synchronous models on loosely time triggered architectures*, IEEE Trans. Computers, 57 (2008), pp. 1300–1314.

[260] A. Valmari, *Compositionality in state space verification methods*, in Application and Theory of Petri Nets, J. Billington and W. Reisig, eds., vol. 1091 of LNCS, Springer, 1996, pp. 29–56.

[261] M. Velasco, J. Fuertes, and P. Marti, *The self triggered task model for real-time control systems*, in 24th IEEE Real-Time Systems Symposium, 2003, pp. 67–70.

[262] S. Vestal, *Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance*, in RTSS, IEEE Computer Society, 2007, pp. 239–243.

[263] S. Wang, A. Ayoub, B. Kim, G. Gössler, O. Sokolsky, and I. Lee, *A causality analysis framework for component-based real-time systems*, in Proc. Runtime Verification 2013, A. Legay and S. Bensalem, eds., vol. 8174 of LNCS, Springer, 2013, pp. 285–303.

[264] S. Wang, A. Ayoub, O. Sokolsky, and I. Lee, *Runtime verification of traces under recording uncertainty*, in RV, S. Khurshid and K. Sen, eds., vol. 7186 of LNCS, Springer, 2011, pp. 442–456.

[265] X. Wang and M. Lemmon, *State based self-triggered feedback control systems with l2 stability*, in 17th IFAC world congress, 2008.

[266] D. Wolf and A. Arkin, *Motifs, modules and games in bacteria*, Current Opinion in Microbiology, 6 (2003), pp. 125–134.

[267] V. Wolf, *Modelling of biochemical reactions by stochastic automata networks*, ENTCS, 171 (2007), pp. 197–208.

[268] W. Wonham and P. Ramadge, *Modular supervisory control of discrete event systems*, Mathematics of Control Signals and Systems, 1 (1988), pp. 13–30.

[269] D. Xu, G. Gössler, and A. Girault, *Probabilistic contracts for component-based design*, in Proc. ATVA'10, A. Bouajjani and W.-N. Chin, eds., vol. 6252 of LNCS, Springer-Verlag, 2010, pp. 325–340.

[270] W. Yi, *Algebraic reasoning for real-time probabilistic processes with uncertain information*, in Third International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT'94, vol. 863 of LNCS, Springer, 1994, pp. 680–693.

[271] A. Zeller, *Why Programs Fail*, Elsevier, 2009.

Index