Fairneß, Randomisierung und Konspiration in verteilten Algorithmen

Hagen Völzer

1. Februar 2001

Fairneß, Randomisierung und Konspiration in verteilten Algorithmen

Dissertation

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (doctor rerum naturalium)
im Fach Informatik

eingereicht an der

Mathematisch-Naturwissenschaftlichen Fakultät II
der Humboldt-Universität zu Berlin

von

Diplom-Informatiker

Hagen Völzer

geboren am 20. April 1971 in Schwerin

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Jürgen Mlynek

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II:
Prof. Dr. sc. nat. Bodo Krause

Gutachter:

1. Prof. Dr. Wolfgang Reisig
2. Prof. Dr. K. Rüdiger Reischuk
3. Prof. Dr. Miroslaw Malek

eingereicht am: 2. November 2000
Tag der mündlichen Prüfung: 8. Dezember 2000

Abstract

Fairness, Randomization, and Conspiracy in Distributed Algorithms

Concepts such as *fairness* (i.e., fair conflict resolution), *randomization* (i.e., coin flips), and *partial synchrony* are frequently used to solve fundamental synchronization- and coordination-problems in distributed systems such as the *mutual exclusion problem* (*mutex problem* for short) and the *consensus problem*. For some problems it is proven that, without such concepts, no solution to the particular problem exists. Impossibilty results of that kind improve our understanding of the way distributed algorithms work. They also improve our understanding of the trade-off between a tractable model and a powerful model of distributed computation.

In this thesis, we prove two new impossibility results and we investigate their reasons. We are in particular concerned with models for randomized distributed algorithms since little is yet known about the limitations of randomization with respect to the solvability of problems in distributed systems. By a solution through randomization we mean that the problem under consideration is solved with probability 1.

In the first part of the thesis, we investigate the relationship between fairness and randomization. On the one hand, it is known that to some problems (e.g. to the consensus problem), randomization admits a solution where fairness does not admit a solution. On the other hand, we show that there are problems (viz. the mutex problem) to which randomization does not admit a solution where fairness does admit a solution. These results imply that fairness cannot be implemented by coin flips.

In the second part of the thesis, we consider a model which combines fairness and randomization. Such a model is quite powerful, allowing solutions to the mutex problem, the consensus problem, and a solution to the *generalized mutex problem*. In the *generalized mutex problem* (a.k.a. the *dining philosophers problem*), a neighborhood relation is given and mutual exclusion must be achieved for each pair of neighbors. We finally consider the *crash-tolerant generalized mutex problem* where every hungry agent eventually becomes critical provided that neither itself nor one of its neighbors crashes. We prove that even the combination of fairness and randomization does not admit a solution to the crash-tolerant generalized mutex problem.

We argue that the reason for this impossibility is the inherent occurrence of an undesirable phenomenon known as *conspiracy*. Conspiracy was not yet properly characterized. We characterize conspiracy on the basis of non-sequential runs, and we show that conspiracy can be prevented by help of the additional assumption of *partial synchrony*, i.e., we show that every conspiracy-prone system can be refined to a randomized system which is, with probability 1, conspiracy-free under the assumptions of partial synchrony and fairness. *Partial synchrony* means that each event consumes a bounded amount of time where, however, the bound is not known.

We use a non-sequential semantics for distributed algorithms which is essential to some parts of the thesis. In particular, we develop a non-sequential semantics for randomized distributed algorithms since there is no such semantics in the literature. In this non-sequential semantics, causal independence is reflected by stochastic independence.

Keywords:
distributed algorithms, fairness, randomization, conspiracy

Zusammenfassung

Fairneß (d.h. faire Konfliktlösung), *Randomisierung* (d.h. Münzwürfe) und *partielle Synchronie* sind verschiedene Konzepte, die häufig zur Lösung zentraler Synchronisations- und Koordinationsprobleme in verteilten Systemen verwendet werden. Beispiele für solche Probleme sind das *Problem des wechselseitigen Ausschlusses* (kurz: *Mutex-Problem*) sowie das *Konsens-Problem*. Für einige solcher Probleme wurde bewiesen, daß ohne die oben genannten Konzepte keine Lösung für das betrachtete Problem existiert. Unmöglichkeitsresultate dieser Art verbessern unser Verständnis der Wirkungsweise verteilter Algorithmen sowie das Verständnis des Trade-offs zwischen einem leicht analysierbaren und einem ausdrucksstarken Modell für verteiltes Rechnen.

In dieser Arbeit stellen wir zwei neue Unmöglichkeitsresultate vor. Darüberhinaus beleuchten wir ihre Hintergründe. Wir betrachten dabei Modelle, die Randomisierung einbeziehen, da bisher wenig über die Grenzen der Ausdrucksstärke von Randomisierung bekannt ist. Mit einer Lösung eines Problems durch Randomisierung meinen wir, daß das betrachtete Problem mit Wahrscheinlichkeit 1 gelöst wird.

Im ersten Teil der Arbeit untersuchen wir die Beziehung von Fairneß und Randomisierung. Einerseits ist bekannt, daß einige Probleme (z.B. das Konsens-Problem) durch Randomisierung nicht aber durch Fairneß lösbar sind. Wir zeigen nun, daß es andererseits auch Probleme gibt (nämlich das Mutex-Problem), die durch Fairneß, nicht aber durch Randomisierung lösbar sind. Daraus folgt, daß Fairneß nicht durch Randomisierung implementiert werden kann.

Im zweiten Teil der Arbeit verwenden wir ein Modell, das Fairneß und Randomisierung vereint. Ein solches Modell ist relativ ausdrucksstark: Es erlaubt Lösungen für das Mutex-Problem, das Konsens-Problem, sowie eine Lösung für das *allgemeine Mutex-Problem*. Beim *allgemeinen Mutex-Problem* (auch bekannt als *Problem der speisenden Philosophen*) ist eine Nachbarschaftsrelation auf den Agenten gegeben und ein Algorithmus gesucht, der das Mutex-Problem für jedes Paar von Nachbarn simultan löst. Schließlich betrachten wir das *ausfalltolerante allgemeine Mutex-Problem* – eine Variante des allgemeinen Mutex-Problems, bei der Agenten ausfallen können. Wir zeigen, daß sogar die Verbindung von Fairneß und Randomisierung nicht genügt, um eine Lösung für das ausfalltolerante allgemeine Mutex-Problem zu konstruieren.

Ein Hintergrund für dieses Unmöglichkeitsresultat ist ein unerwünschtes Phänomen, für das in der Literatur der Begriff *Konspiration* geprägt wurde. Konspiration wurde bisher nicht adäquat charakterisiert. Wir charakterisieren Konspiration auf der Grundlage *nicht-sequentieller Abläufe*. Desweiteren zeigen wir, daß Konspiration für eine große Klasse von Systemen durch die zusätzliche Annahme von partieller Synchronie verhindert werden kann, d.h. ein konspirationsbehaftetes System kann zu einem randomisierten System verfeinert werden, das unter Fairneß und partieller Synchronie mit Wahrscheinlichkeit 1 konspirationsfrei ist. *Partielle Synchronie* fordert, daß alle relativen Geschwindigkeiten im System durch eine Konstante beschränkt sind, die jedoch den Agenten nicht bekannt ist.

Die Darstellung der Unmöglichkeitsresultate und die Charakterisierung von Konspiration wird erst durch die Verwendung *nicht-sequentieller Abläufe* möglich. Ein nicht-sequentieller Ablauf repräsentiert im Gegensatz zu einem sequentiellen Ablauf kausale Ordnung und nicht zeitliche Ordnung von Ereignissen. Wir entwickeln in dieser Arbeit eine nicht-sequentielle Semantik für randomisierte verteilte Algorithmen, da es bisher keine in der Literatur gibt. In dieser Semantik wird kausale Unabhängigkeit durch stochastische Unabhängigkeit widergespiegelt.

Schlagwörter:
Verteilte Algorithmen, Fairneß, Randomisierung, Konspiration

Vorwort

Den Hintergrund der vorliegenden Dissertation bildet meine Forschungstätigkeit im Projekt „Konsensalgorithmen“ am Lehrstuhl für Theorie der Programmierung des Instituts für Informatik der Humboldt-Universität zu Berlin. Das Projekt wurde von der Deutschen Forschungsgemeinschaft gefördert und von Prof. Dr. Wolfgang Reisig und Prof. Dr. Miroslaw Malek geleitet.

Ich danke Prof. Dr. Wolfgang Reisig für die Betreuung und Begutachtung der Arbeit, sowie besonders dafür, daß ich an seinem Lehrstuhl unter hervorragenden Bedingungen arbeiten durfte. Für die Begutachtung der Arbeit möchte ich auch herzlich Prof. Dr. Rüdiger Reischuk sowie Prof. Dr. Miroslaw Malek danken.

Für das kritische Lesen von Vorversionen sowie für die damit verbundenen Diskussionen danke ich Ekkart Kindler, Sibylle Peuker, Felix Gärtner, Stefan Haar, Karsten Schmidt und Stephan Roch. Besonderer Dank gilt Ekkart Kindler und Stefan Haar mit denen ich besonders oft und lange diskutiert habe, mit Ekkart Kindler vor allem zu nicht-sequentiellen Abläufen und zu Fairneß, mit Stefan Haar vor allem zu Randomisierung und zu probabilistischen Abläufen. Michael Weber danke ich für wertvolle LaTeX-Hinweise sowie für Abbildung 6.1. Desweiteren danke ich allen Teilnehmern der Kaffeerunde des o.g. Lehrstuhls für ihren Beitrag zur guten Arbeitsatmosphäre, in der die vorliegende Arbeit entstanden ist.

Nicht zuletzt möchte ich meinen Eltern danken, die immer meinen Weg, insbesondere auch mein Interesse an Mathematik gefördert haben. Von Sibylle habe ich immer zum richtigen Zeitpunkt Motivation und Rückhalt erhalten. Danke!

Berlin, im Februar 2001 Hagen Völzer

Inhaltsverzeichnis

Einleitung **1**

1 Grundlagen **7**

1.1 Mathematische Grundlagen 7

1.2 Petrinetze und deren Abläufe 9

1.2.1 Petrinetze 9

1.2.2 Abläufe und Abwicklungen 12

1.2.3 Sequentialisierung von Abläufen 23

1.3 Ablaufeigenschaften 24

1.3.1 Sicherheits- und Lebendigkeitseigenschaften 24

1.3.2 Temporallogische Eigenschaften 25

1.4 Petrinetzmodellierung verteilter Algorithmen 27

1.5 Algebraische Netze 30

1.5.1 Signaturen, Variablen und Terme 30

1.5.2 Algebraische Netze 31

1.5.3 Entfaltung eines algebraischen Netzes 34

1.6 Wahrscheinlichkeitsräume 36

I Fairneß und Randomisierung **37**

2 Netzsysteme **39**

2.1 Netzsysteme 39

2.1.1 Progreß 39

2.1.2 Netzsysteme 40

2.1.3 Progreß und schwache Fairneß 42
2.2 Lebendigkeit 44
2.3 Mutex in Netzsystemen 46
2.3.1 Das Problem des wechselseitigen Ausschlusses 46
2.3.2 Formalisierung von Mutex 47
2.3.3 Unmöglichkeit von Mutex in Netzsystemen 48
2.4 Konsens in Netzsystemen 50
2.4.1 Das ausfalltolerante Konsens-Problem 51
2.4.2 Formalisierung des Konsens-Problems 52
2.4.3 Ein kleiner Konsensalgorithmus 55
2.4.4 Unmöglichkeit von Konsens in Netzsystemen 58

3 Faire Netzsysteme 63
3.1 Faire Netzsysteme 63
3.1.1 Faire Schaltsequenzen 63
3.1.2 Faire Abläufe und faire Netzsysteme 65
3.1.3 Ein Problem von starker Fairneß 67
3.2 Mutex in fairen Netzsystemen 70
3.3 Konsens in fairen Netzsystemen 71
3.3.1 Das Modell von Fischer, Lynch und Paterson 71
3.3.2 Unmöglichkeit von Konsens in fairen Netzsystemen 72

4 Randomisierte Netzsysteme 77
4.1 Ein Petrinetzmodell für randomisierte Algorithmen 77
4.1.1 Randomisierte Algorithmen 77
4.1.2 Randomisierte Netzsysteme 79
4.1.3 Probabilistische Schaltbäume 81
4.1.4 Probabilistische Abläufe 83
4.1.5 Probabilistische Gültigkeit von Ablaufeigenschaften 85
4.1.6 Beispiele 87
4.1.7 Vergleich von sequentieller und nicht-sequentieller Semantik . 89
4.1.8 Extreme Fairneß 92

4.2 Konsens in randomisierten Netzsystemen 95
4.3 Mutex in randomisierten Netzsystemen 98
4.3.1 Unmöglichkeit von Mutex in randomisierten Netzsystemen . 98
4.3.2 Zwei Aspekte von Fairneß 99

II Konspiration **103**

5 Faire randomisierte Netzsysteme **105**
5.1 Faire randomisierte Netzsysteme 106
5.2 Allgemeiner Mutex in fairen randomisierten Netzsystemen 107
5.2.1 Das allgemeine Mutex-Problem 107
5.2.2 Das ausfalltolerante allgemeine Mutex-Problem 108
5.2.3 Unmöglichkeit von ausfalltolerantem allgemeinen Mutex . . . 109

6 Konspiration **113**
6.1 Charakterisierung von Konspiration 113
6.1.1 Die konspirierenden Philosophen 114
6.1.2 Konspiration in Multiparty-Interaktionen 115
6.1.3 Charakterisierung von Konspiration 116
6.1.4 Weitere Beispiele für Konspiration 119
6.2 Konspiration in der Literatur 121
6.2.1 ∞-Fairneß 121
6.2.2 Hyperfairneß 124
6.3 Konspiration und Ausfalltoleranz 126
6.3.1 Ausfalltoleranter allgemeiner Mutex 126
6.3.2 Ausfalltoleranter Konsens 127
6.3.3 Ein weiteres Problem 128

7 Konspirationsfreiheit **129**
7.1 Konspiration bezüglich einer Transition 129
7.1.1 Beschränkte und unbeschränkte Konspiration 129
7.1.2 Quasisynchronie 131

7.1.3 Der Nutzen von Quasisynchronie 134
7.2 Konspiration bezüglich mehrerer Transitionen 136
7.2.1 Adaptive Timeouts an mehreren Transitionen 136
7.2.2 Randomisierte Timeouts 137

Abschließende Bemerkungen **143**

Anhang **145**

A Beweise **147**
A.1 Konstruktion des Wahrscheinlichkeitsraumes für probabilistische Abläufe 147
A.1.1 Grundbegriffe der Maßtheorie 147
A.1.2 Konstruktion der Mengenalgebra 148
A.1.3 Konstruktion des Maßes 151

Definitionsverzeichnis **155**

Abbildungsverzeichnis **159**

Literaturverzeichnis **163**

Index **171**

Einleitung

Verteilte Systeme zu entwerfen und zu verstehen ist schwer, weil uns Intuition dafür fehlt. Eine Methode, um eine Intuition für ein Objekt zunächst zu entwickeln und dann zu verfeinern, besteht in der mathematischen Modellierung und Analyse des Objekts. In dieser Arbeit geht es um die Analyse mathematischer Modelle verteilter Systeme.

Verteilte Systeme sind vielgestaltig. Das Internet, ein lokales Netz, ein Rechnercluster, ein Parallelrechner, ein asynchroner Schaltkreis sowie ein Geschäftsprozeß oder ein Produktionsprozeß – all dies sind verteilte Systeme. Jedes verteilte System besteht aus handelnden Einheiten, die miteinander kommunizieren. Eine handelnde Einheit nennen wir in dieser Arbeit *Agent*. Jedes der genannten verteilten Systeme hat spezielle Charakteristika, in denen es sich von den anderen verteilten Systemen unterscheidet, zum Beispiel unterscheiden sich die genannten Systeme in der Geschwindigkeit der Kommunikation. Ein Modell, das von den Charakteristika verschiedener verteilter Systeme abstrahiert, ist *allgemein* – analysieren wir ein allgemeines Modell, so erhalten wir Aussagen für viele verschiedene verteilte Systeme. Ein Modell, das von der relativen Geschwindigkeit seiner Agenten und ihrer Kommunikation abstrahiert, ist *asynchron*.

Ein verteiltes System erfüllt meist erst dann seine Aufgabe, wenn die Aktivitäten verschiedener Agenten koordiniert und synchronisiert werden. Die typischen Probleme, die wir in einem verteilten System lösen wollen, sind daher Koordinations- und Synchronisationsprobleme. Beispiele für solche Probleme sind das *Problem des wechselseitigen Ausschlusses* (kurz: *Mutex-Problem*) und das *ausfalltolerante Konsens-Problem* (kurz: *Konsens-Problem*). Beim Mutex-Problem geht es für zwei Agenten, von denen jeder immer wieder *kritisch* sein kann, darum, daß beide Agenten nie gleichzeitig kritisch sind. Beim Konsens-Problem geht es für eine Menge von Agenten darum, eine gemeinsame Entscheidung zu treffen, und zwar auch dann, wenn einige Agenten ausfallen. Eine Lösung für ein Koordinations- oder Synchronisationsproblem heißt *verteilter Algorithmus*. Ein *verteilter Algorithmus* weist jedem Agenten des Systems eine Handlungsvorschrift zu. Diese Handlungsvorschrift nennen wir das *Programm* des Agenten.

Eine typische Frage, die wir für ein Modell verteilter Systeme beantworten wollen, ist die Frage nach den Problemen, die wir in diesem Modell lösen können. Eine Erkennt-

nis, daß ein bestimmtes Problem in einem bestimmten Modell nicht gelöst werden kann, heißt *Unmöglichkeitsresultat*. Ein berühmt gewordenes Unmöglichkeitsresultat für verteilte Systeme stammt von Fischer, Lynch und Paterson, die zeigten, daß das Konsens-Problem in einem allgemeinen asynchronen Modell nicht lösbar ist, bei dem das Programm jedes Agenten deterministisch ist [36]. Dieses Resultat ragt aus anderen heraus, weil es sich beim Konsens-Problem um ein *paradigmatisches* Problem handelt – ein Problem, das für eine ganze Klasse von Problemen steht, weil es wesentliche Merkmale anderer Probleme enthält.

Einige Unmöglichkeitsresultate sparen uns Aufwand in Entwurf, Implementation und Test eines verteilten Systems. Oft impliziert ein Unmöglichkeitsresultat jedoch nicht, daß wir das Problem in der Praxis nicht lösen können – Fischer, Lynch und Paterson schreiben beispielsweise zu ihrem Resultat: „...; rather, they [these results] point up the need for more refined models of distributed computing that better reflect realistic assumptions ..., and for less stringent requirements on the solution to such problems." [36]

Tatsächlich erhalten wir durch Verfeinerung des Modells von Fischer, Lynch und Paterson eine Lösung des Konsens-Problems. Ben-Or zeigt, daß das Konsens-Problem mit Wahrscheinlichkeit 1 in einem asynchronen Modell lösbar ist, falls Agenten in ihren Programmen Münzen werfen können [14]. Ein verteilter Algorithmus, bei dem Agenten Münzen werfen, heißt *randomisierter verteilter Algorithmus*. Sprechen wir im folgenden davon, daß ein randomisierter Algorithmus ein Problem löst, so meinen wir, daß der Algorithmus das Problem mit Wahrscheinlichkeit 1 löst.

Mit randomisierten Algorithmen kann man also mehr Probleme als mit herkömmlichen, deterministischen Algorithmen lösen. Auf der anderen Seite ist die Analyse randomisierter Algorithmen leider schwerer als die Analyse herkömmlicher Algorithmen. Lehmann und Rabin schreiben: „The realm of proofs of correctness for concurrent processes is not well known. As the reader will realize, proofs of correctness of probabilistic distributed algorithms are extremely slippery." [59]; Pnueli und Zuck schreiben: „this [verification] becomes specially crucial when dealing with probabilistic concurrent algorithms, where intuition often fails to grasp the full intricacy of the algorithm." [70].

Verfeinern wir ein Modell, so wird es komplexer, weniger allgemein und oft schwerer analysierbar. Dies ist ein typischer Trade-Off: Je allgemeiner und leichter analysierbar ein Modell ist, desto weniger Probleme sind darin lösbar. Ein Modell, in dem wir viele Probleme lösen können, nennen wir *ausdrucksstark*. Indem wir den Trade-Off zwischen einem allgemeinen und einem ausdrucksstarken Modell mit Hilfe von Unmöglichkeitsresultaten verstehen, verbessern wir unsere Intuition für verteilte Systeme. Die Bedeutung von Unmöglichkeitsresultaten wird durch ihren signifikanten Anteil in Lehrbüchern über verteilte Algorithmen widergespiegelt [10, 62, 88].

Der Trade-Off zwischen einem allgemeinen und einem ausdrucksstarken Modell für

verteilte Systeme ist schwer zu verstehen. Dies liegt zum einen an der Vielzahl verschiedener Modelle, die durch die Kombination einer Vielzahl von Modellparametern entsteht. Wichtige Modellparameter sind neben der Verfügbarkeit von Randomisierung oder der Annahme von Synchronie, die Verfügbarkeit von Kommunikationsprimitiven wie z.B. Broadcast sowie die Annahme von *Fairneß* (siehe unten). Um den Trade-Off für das Konsens-Problem auszuloten, definieren Dolev, Dwork und Stockmeyer in [31] beispielsweise nicht weniger als 32 Modelle, in denen sie die Lösbarkeit des Konsens-Problems untersuchen. Ihr Papier und andere Papiere zeigen, daß Unmöglichkeitsresultate oft sensibel gegenüber kleinen Änderungen an verschiedenen Modellparametern sind. Dadurch ist es schwer, wenn nicht unmöglich, eine einzelne Ursache für das Unmöglichkeitsresultat zu isolieren. Oft liest man in Papieren, die das Ergebnis von Fischer, Lynch und Paterson zitieren, daß die Ursache dieses Unmöglichkeitsresultates darin besteht, daß man in einem asynchronen System einen ausgefallenen Agenten nicht von einem sehr langsamen Agenten unterscheiden kann. Dies widerspricht Ben-Ors Resultat, der zeigt, daß weder irgendeine Form von Synchronie, noch die Erkennung von Ausfällen zwingend nötig ist, um das Konsens-Problem zu lösen. Insgesamt ergibt sich das Bild eines schwach strukturierten Raums von Modellen, in dem viele Modelle bezüglich ihrer Ausdrucksstärke unvergleichbar sind.

Wenig ist bisher über die Grenzen von Randomisierung bekannt. Wie im Beispiel des Konsens-Problems hilft es bei einigen Problemen, Randomisierung in das Modell einzubeziehen. Dies gilt jedoch nicht für alle Probleme. Hart, Sharir und Pnueli stellen in [40] erstaunt fest, daß sich sogar ähnliche Probleme bei Hinzunahme von Randomisierung unterschiedlich verhalten können: Während das eine Problem durch Hinzunahme von Randomisierung lösbar wird, bleibt ein ähnliches Problem bei Hinzunahme von Randomisierung unlösbar. Ein Grund für dieses Phänomen ist bisher nicht bekannt. Hart, Sharir und Pnueli schreiben in [40]: „These phenomena call for further study to understand better the distinction between those concurrent problems that admit probabilistic solutions that are better than deterministic solutions, and those problems that do not benefit from introduction of randomization.“ Insgesamt gibt es nur wenige Unmöglichkeitsresultate für Randomisierung, die darüberhinaus selten an formalen Modellen dargestellt sind.

In dieser Arbeit stellen wir zwei neue Unmöglichkeitsresultate für paradigmatische Probleme vor. Wir betrachten dabei Modelle, die Randomisierung einbeziehen, und zeigen so Grenzen der Ausdrucksstärke von Randomisierung auf.

Im ersten Teil der Arbeit geht es um die Beziehung von Randomisierung und *Fairneß* in verteilten Systemen. *Fairneß* fordert die *faire* Auflösung von *Konflikten* im System. Ein *Konflikt* besteht in einem Zustand zwischen zwei in diesem Zustand ausführbaren Programmaktionen (desselben oder verschiedener Agenten), falls beide Programmaktionen auf eine gemeinsame Ressource zugreifen wollen, so daß bei-

de Programmaktionen in diesem Zustand nicht nebenläufig zueinander ausgeführt werden können. In einem Ablauf kann ein und derselbe Konflikt zwischen zwei Programmaktionen immer wieder auftreten. Ein Konflikt wird in einem Ablauf *fair* gelöst, falls gilt: Tritt der Konflikt unendlich oft auf, so wird er unendlich oft zugunsten jeder Programmaktion aufgelöst.

Eine naheliegende Idee ist es, ein System so zu konstruieren, daß es von selbst Konflikte fair löst, in dem es Randomisierung verwendet: Das System soll dabei durch Münzwurf entscheiden, zugunsten welcher Programmaktion ein Konflikt gelöst wird. Gelingt es, solch ein System zu konstruieren, so sprechen wir von der Implementation von Fairneß durch Randomisierung. Wir weisen in dieser Arbeit nach, daß es für das Mutex-Problem, für das es bekanntlich eine Lösung unter Fairneß gibt, keine Lösung durch Randomisierung existiert. Daraus folgt, daß die Implementation von Fairneß durch Randomisierung im allgemeinen nicht möglich ist.

Unser Resultat zeigt, daß Fairneß und Randomisierung bezüglich ihrer Ausdrucksstärke unvergleichbar sind: Einerseits gibt es ein Problem (nämlich das Mutex-Problem), das durch Fairneß, nicht aber durch Randomisierung gelöst werden kann. Andererseits gibt es ein Problem (nämlich das Konsens-Problem), das durch Randomisierung, nicht aber durch Fairneß gelöst werden kann.

Im zweiten Teil der Arbeit verwenden wir ein Modell, das Fairneß und Randomisierung vereint. Ein solches Modell ist relativ ausdrucksstark: Es erlaubt Lösungen für das Mutex-Problem, das Konsens-Problem, sowie eine Lösung für das *allgemeine Mutex-Problem*. Beim *allgemeinen Mutex-Problem* (auch bekannt als *Problem der speisenden Philosophen* [24]) ist eine Nachbarschaftsrelation auf den Agenten gegeben und ein Algorithmus gesucht, der das Mutex-Problem für jedes Paar von Nachbarn simultan löst. Schließlich betrachten wir das *ausfalltolerante allgemeine Mutex-Problem* – eine Variante des allgemeinen Mutex-Problems, bei der Agenten ausfallen können[1]. Wir beweisen, daß sogar die Verbindung von Fairneß und Randomisierung nicht genügt, um eine Lösung für das ausfalltolerante allgemeine Mutex-Problem zu konstruieren. Dies zeigt, daß – im Gegensatz zum Konsens-Problem – Randomisierung nicht immer geeignet ist, Ausfalltoleranz zu erzielen.

Der Beweis dieses Unmöglichkeitsresultates offenbart, daß das ausfalltolerante allgemeine Mutex-Problem inhärent ein unerwünschtes Phänomen enthält, für das in der Literatur der Begriff *Konspiration* geprägt wurde. Konspiration wurde bisher in Systemen untersucht, in denen Agenten entweder gemeinsame Variablen oder gemeinsame Aktionen haben. Wir zeigen, daß Konspiration auch in völlig verteilten Systemen vorkommt, d.h. in Systemen, in denen Agenten weder gemeinsame Variablen noch gemeinsame Aktionen haben. Darüberhinaus ist die Verwendung von Konspiration zum Verständnis fehlertoleranter Algorithmen neu.

[1] Hier wird gefordert, daß jeder hungrige Agent kritisch wird, es sei denn er oder einer seiner Nachbarn fällt aus.

Konspiration wurde bisher nicht adäquat charakterisiert. Uns gelingt es, Konspiration auf der Grundlage *nicht-sequentieller Abläufe* (siehe unten) adäquat zu charakterisieren. Desweiteren zeigen wir, daß wir Konspiration für eine große Klasse von Systemen durch die zusätzliche Annahme von *Quasisynchronie* verhindern können, d.h. wir verfeinern ein konspirationsbehaftetes System zu einem randomisierten System, das unter Fairneß und Quasisynchronie mit Wahrscheinlichkeit 1 konspirationsfrei ist. *Quasisynchronie* fordert, daß alle relativen Geschwindigkeiten im System durch eine Konstante beschränkt sind, die jedoch den Agenten nicht bekannt ist.

Die Darstellung der Unmöglichkeitsresultate und die Charakterisierung von Konspiration wird erst durch die Verwendung *nicht-sequentieller Abläufe* möglich. Ein nicht-sequentieller Ablauf repräsentiert im Gegensatz zu einem sequentiellen Ablauf kausale Ordnung und nicht zeitliche Ordnung von Ereignissen.

Nicht-sequentielle Abläufe werden im Bereich verteilter Algorithmen selten verwendet, da man gegenüber sequentiellen Abläufen mit einer komplexeren Struktur umgehen muß. Auf der anderen Seite gewinnt man durch die zusätzliche Struktur nichtsequentieller Abläufe tiefere Einsichten in die Zusammenhänge verschiedener Phänomene verteilter Systeme. Da es für randomisierte verteilte Algorithmen bisher noch keine nicht-sequentielle Semantik gibt, entwickeln wir in dieser Arbeit eine. Diese Semantik hat im Unterschied zur klassischen sequentiellen Semantik randomisierter Algorithmen die Eigenschaft, daß zwei kausal unabhängige Entscheidungen auch immer stochastisch unabhängig sind.

Die Grundlage unserer Modelle sind Petrinetze. Petrinetze haben im Gegensatz zu vielen anderen Formalismen eine kanonische nicht-sequentielle Semantik.

Abschließend wollen wir bemerken, daß auch unsere Unmöglichkeitsresultate nicht bedeuten, daß die betreffenden Probleme in der Praxis unlösbar sind. Sie weisen aber auf Unzulänglichkeiten hin, die jede praktische Lösung hat. Lamport beschreibt dies in [56] für die Unlösbarkeit des *Arbiter-Problems* wie folgt: „The significance of Buridan's Principle [the impossibility result] lies in its warning that decisions may, in rare circumstances, take much longer than expected." Schneider schätzt in [85] die Bedeutung solcher Unmöglichkeitsresultate wie folgt ein: „Lastly, the various models can and should be regarded as limiting cases. The behavior of a real system is bounded by our models. Thus, understanding the feasibility and costs associated with solving problems in these models, can give us insight into the feasibility and cost of solving a problem in some given real system whose behavior lies between the models."

Die Arbeit ist wie folgt strukturiert. Nachdem wir in Kapitel 1 die Grundlagen für die weiteren Erörterungen gelegt haben, definieren wir im Kapitel 2 mit *Netzsystemen* unser Ausgangsmodell und untersuchen die Lösbarkeit von Mutex- und Konsens-Problem in Netzsystemen. In Kapitel 3 erweitern wir ein Netzsystem um

eine Fairneßannahme zu einem *fairen Netzsystem*. Wir untersuchen dann die Lösbarkeit von Mutex- und Konsens-Problem in fairen Netzsystemen. In Kapitel 4 definieren wir *randomisierte Netzsysteme*. Ein *randomisiertes Netzsystem* ist ein um ein Münzwurfkonstrukt erweitertes Netzsystem. Randomisierte Netzsysteme stellen, insbesondere durch ihre nicht-sequentielle Semantik, die wir in Kapitel 4 entwickeln, ein neues Modell für randomisierte verteilte Algorithmen dar. Desweiteren bestimmen wir in Kapitel 4 die Lösbarkeit von Mutex- und Konsens-Problem in randomisierten Netzsystemen.

Mit Kapitel 5 gehen wir zum zweiten Teil der Arbeit über, der den Titel „Konspiration“ trägt. In Kapitel 5 zeigen wir die Unmöglichkeit einer Lösung des ausfalltoleranten allgemeinen Mutex-Problems in *fairen randomisierten Netzsystemen*. Ein *faires randomisiertes Netzsystem* ist ein um eine Fairneßannahme erweitertes randomisiertes Netzsystem. In Kapitel 6 charakterisieren wir Konspiration und diskutieren die Beziehung unserer Definition zur Literatur. Kapitel 7 zeigt schließlich wann und wie wir Konspiration verhindern können.

1 Grundlagen

In diesem Kapitel legen wir die Grundlagen für die Erörterungen der folgenden Kapitel. Den Kern dieses Kapitels stellt Abschnitt 1.2 über Petrinetze und ihre nicht-sequentielle Semantik dar. In Abschnitt 1.3 beschäftigen wir uns mit Ablaufeigenschaften und ihrer Darstellung durch temporale Logik. Abschnitt 1.4 führt in die Modellierung verteilter Algorithmen durch spezielle Petrinetze – *algebraische Netze* ein. In Abschnitt 1.5 formalisieren wir dann algebraische Netze. Schließlich halten wir in Abschnitt 1.6 noch einige Grundbegriffe der Wahrscheinlichkeitsrechnung fest. Wir beginnen nun mit der Festlegung mathematischer Notationen und Begriffe.

1.1 Mathematische Grundlagen

1.1.1 Mengen und Relationen

Mit $\varnothing$ bezeichnen wir die leere Menge, mit $\mathbb{B} = \{\text{true}, \text{false}\}$ die Menge der Wahrheitswerte und mit $\mathbb{N} = \{0, 1, \ldots\}$ die Menge der natürlichen Zahlen sowie mit $\mathbb{R}$ die Menge der reellen Zahlen. Für eine positive reelle Zahl r bezeichne $\lfloor r \rfloor$ die größte natürliche Zahl, die kleiner oder gleich r ist.

Sei A eine Menge. Die Kardinalität von A bezeichnen wir durch $|A|$, die Menge aller Teilmengen von A mit 2^A sowie die Menge aller endlichen Teilmengen von A durch $\mathfrak{S}(A)$. Für eine Menge A bezeichnet $\mathfrak{W}(A)$ die Menge aller nicht-leeren endlichen Sequenzen über A. Für eine Relation $R \subseteq A \times A$ über A bezeichnet R^+ den transitiven Abschluß und R^* den reflexiv-transitiven Abschluß von R. Wir schreiben auch $x\ R\ y$ anstelle von $(x, y) \in R$. Eine *Familie* von Mengen über einer *Indexmenge* I wird durch $(A_i)_{i \in I}$ bezeichnet. Eine Familie $(A_i)_{i \in I}$ ist *paarweise disjunkt*, falls für alle $i, j \in I$ mit $i \neq j$ gilt $A_i \cap A_j = \varnothing$.

Sei A eine Menge und $A' \subseteq A$. Die *charakteristische Funktion* von A' ist die Abbildung $\chi_{A'} : A \to \{0, 1\}$, die definiert ist durch $\chi_{A'}(x) = 1$ gdw. $x \in A'$.

1.1.2 Multimengen

Sei A eine Menge. Eine *Multimenge* über A ist eine Abbildung $M : A \to \mathbb{N}$. Für ein Element $x \in A$ heißt $M(x)$ *Vielfachheit* von x in M. Statt $M(x)$ schreiben wir im folgenden $M[x]$. Für eine Multimenge M heißt die Menge $\{x \in A \mid M[x] \neq 0\}$ *Träger* von M. Eine Multimenge M ist *endlich* falls der Träger von M endlich ist. Die Menge aller endlichen Multimengen über A bezeichnen wir durch $\mathfrak{M}(A)$. Manchmal betrachten wir Teilmengen von A als spezielle Multimengen über A, indem wir die Teilmenge mit ihrer charakteristischen Funktion identifizieren.

Wir definieren Inklusion, Addition und Subtraktion auf Multimengen punktweise wie folgt: Es sei $M_1 \leq M_2$ gdw. für alle $x \in A : M_1[x] \leq M_2[x]$. Es sei $(M_1 + M_2)[x] = M_1[x] + M_2[x]$ und für $M_1 \leq M_2$ sei $(M_2 - M_1)[x] = M_2[x] - M_1[x]$. Desweiteren sei für $k \in \mathbb{N}$ und eine Multimenge M die Multimenge $k \cdot M$ definiert durch $(k \cdot M)[x] = k \cdot M[x]$.

1.2 Petrinetze und deren Abläufe

In diesem Abschnitt definieren wir Petrinetze und ihre Semantik. Dabei definieren wir in Unterabschnitt 1.2.1 *Schaltsequenzen* als sequentielle Semantik und in Unterabschnitt 1.2.2 *Abläufe* als nicht-sequentielle Semantik von Petrinetzen. In Unterabschnitt 1.2.3 stellen wir die Beziehung von Abläufen und Schaltsequenzen her.

1.2.1 Petrinetze

Eine Einführung in die Theorie der Petrinetze findet man in [77, 83]. Wir beginnen mit der Definition eines Petrinetzes, das wir im weiteren einfach als *Netz* bezeichnen.

Definition 1.1 (Netz)

Ein *Netz* $N = (P, T; F)$ besteht aus zwei disjunkten, nicht-leeren, abzählbaren Mengen P und T sowie einer Relation $F \subseteq (P \times T) \cup (T \times P)$. Die Elemente von P heißen *Stellen* (oder *Plätze*), die Elemente von T *Transitionen* und die Elemente von F *Kanten* des Netzes. ∘

Graphisch stellen wir eine Stelle durch einen Kreis oder eine Ellipse, eine Transition durch ein Quadrat und eine Kante durch einen Pfeil zwischen den betreffenden Elementen dar. Für Netze haben sich eine Reihe von Standardnotationen durchgesetzt, die wir im folgenden definieren.

Definition 1.2 (Standardnotationen für Netze)

Sei $N = (P, T; F)$ ein Netz. Wir schreiben $x \in N$ an Stelle von $x \in P \cup T$. Ein $x \in N$ heißt *Element* von N. Wir definieren für $x \in N$ den *Vorbereich* von x durch ${}^\bullet x = \{y \in N \mid (y, x) \in F\}$ und den *Nachbereich* von x durch $x^\bullet = \{y \in N \mid (x, y) \in F\}$. Die Mengen der *minimalen* bzw. *maximalen* Elemente von N sind definiert durch ${}^\circ N = \{x \in N \mid {}^\bullet x = \varnothing\}$ und $N^\circ = \{x \in N \mid x^\bullet = \varnothing\}$. Für $x \in N$ bezeichnet $\downarrow x = \{y \in N \mid y F^+ x\}$ die Menge der *Vorgänger* von x. ∘

Zur Modellierung von Systemen wollen wir nur solche Netze verwenden, bei denen Vor- und Nachbereich jeder Transition nicht-leer und endlich sind[1]. Ein Netz mit dieser Eigenschaft nennen wir *Systemnetz*.

Definition 1.3 (Struktureigenschaften von Netzen)

Sei $N = (P, T; F)$ ein Netz. N ist

[1] Dies vermeidet Anomalien bei der Definition nicht-sequentieller Abläufe, vgl. [19].

(a) *endlich T-verzweigt*[2], falls für jede Transition t von N der Vorbereich ${}^\bullet t$ und der Nachbereich $t^\bullet$ endlich ist.

(b) *stellenberandet*, falls ${}^\circ N \subseteq P$ und $N^\circ \subseteq P$.

(c) *schlicht*[3], falls für alle $t_1, t_2 \in T$ gilt: ${}^\bullet t_1 = {}^\bullet t_2$ und $t_1^\bullet = t_2^\bullet$ impliziert $t_1 = t_2$.

(d) ein *Systemnetz*, falls N endlich T-verzweigt, stellenberandet und schlicht ist.

(e) *vorgängerendlich*, falls für alle $x \in N$ die Menge $\downarrow x$ endlich ist.

(f) *azyklisch*, falls für alle Elemente x gilt $x \notin \downarrow x$.

Wir kommen nun zur Dynamik eines Netzes. Ein Zustand eines Netzes wird auch als *Markierung* bezeichnet. Eine Markierung ist eine endliche Multimenge von Stellen des Netzes. Wir definieren weiterhin, wann eine Transition in einer Markierung *schalten* kann und zu welcher *Nachfolgermarkierung* dieses Schalten führt.

Definition 1.4 (Markierung, Schalten)

Sei $N = (P, T; F)$ ein Systemnetz.

(a) Eine *Markierung* von N ist eine endliche Multimenge $M \in \mathfrak{M}(P)$ über der Stellenmenge von N. Eine Stelle $p \in P$ heißt *markiert* in M, falls $M[p] \neq 0$. Eine Markierung M heißt *sicher*, falls $\forall p \in P : M[p] \leq 1$.

(b) Zu jeder Transition $t \in T$ definieren wir die *Vormarkierung* t^- und die *Nachmarkierung* t^+ durch

$$t^-[p] = \begin{cases} 1 & \text{falls } (p,t) \in F \\ 0 & \text{sonst} \end{cases} \quad \text{und} \quad t^+[p] = \begin{cases} 1 & \text{falls } (t,p) \in F \\ 0 & \text{sonst} \end{cases}.$$

(c) Für eine Transition t sei die *Schaltrelation* $\xrightarrow{t} \subseteq \mathfrak{M}(P) \times \mathfrak{M}(P)$ definiert durch $M_1 \xrightarrow{t} M_2$ gdw. $M_1 \geq t^-$ und $M_2 = (M_1 - t^-) + t^+$. Ist $M_1 \xrightarrow{t} M_2$, so sagen wir t *kann in* M_1 *schalten* (oder t *ist in* M_1 *aktiviert*). Gilt $M_1 \xrightarrow{t} M_2$ für irgendein t, so schreiben wir $M_1 \rightarrow M_2$ und sagen M_2 ist eine *Nachfolgermarkierung* von M_1. Wir schreiben $\xrightarrow{*}$ anstelle von $\rightarrow^*$. Gilt $M_1 \xrightarrow{*} M_2$, so nennen wir M_2 *von* M_1 *erreichbar*.

Eine Markierung stellen wir graphisch durch schwarze Marken in den Stellen des Netzes dar. Da in jeder Markierung nur endlich viele Stellen markiert sind, ist die

[2] auch: *von endlicher Synchronisation*

[3] in der Literatur manchmal: *transitionsschlicht*; in der Literatur wird für Schlichtheit meist zusätzlich dieselbe Bedingung für die Stellen des Netzes gefordert.

Menge aller Markierungen abzählbar. Zwei Transitionen t_1, t_2 eines Netzes N stehen *in Konflikt* in M, falls beide Transitionen in M aktiviert sind und ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \varnothing$.

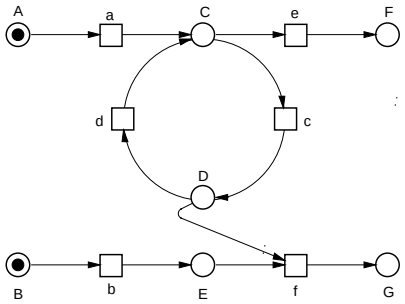


Abb. 1.1: Ein Netz Σ_1.

Abb. 1.1 zeigt ein Netz Σ_1 mit sicherer Markierung $\{A, B\}$, für die wir im folgenden AB schreiben. In AB können die Transitionen a und b schalten. Das Schalten von a führt zu CB, das Schalten von b führt zu AE. Sei M eine Markierung eines Netzes N. In der Markierung CB von Σ_1 stehen die Transitionen e und c in Konflikt, in der Markierung DE die Transitionen d und f.

Wir definieren nun sequentielle Abläufe, die klassische sequentielle Semantik von Netzen. Einen sequentiellen Ablauf bezeichnen wir als *Schaltsequenz*. Dazu betrachten wir ein Netz zusammen mit einer *Anfangsmarkierung*. Als Anfangsmarkierungen wollen wir aus technischen Gründen nur sichere Markierungen zulassen.

Definition 1.5 (Initialisiertes Netz, Schaltsequenz)

(a) Ein Paar $\Sigma = (N, M^0)$ heißt *initialisiertes Netz*, falls N ein Systemnetz und M^0 eine sichere Markierung von N ist; M^0 heißt *Anfangsmarkierung* von Σ. Eine Markierung M von Σ heißt *erreichbare Markierung von* Σ, falls M von der Anfangsmarkierung erreichbar ist. Ein initialisiertes Netz Σ heißt *sicher*, wenn jede erreichbare Markierung von M^0 sicher ist.

(b) Sei $\Sigma = (N, M^0)$ ein initialisiertes Netz. Eine *Schaltsequenz* von Σ ist eine (endliche oder unendliche) alternierende Sequenz $\sigma = M_0, t_1, M_1, t_2, M_2, \ldots$ von Markierungen und Transitionen von N, so daß $M_0 = M^0$ und für alle i ist $M_i \overset{t_{i+1}}{\rightarrow} M_{i+1}$. Wir schreiben auch $M_0 \overset{t_1}{\rightarrow} M_1 \overset{t_2}{\rightarrow} M_2, \ldots$ für σ. Ist σ endlich, so endet σ in einer Markierung M_n, die wir als *Endmarkierung von* σ bezeichnen. Die Schaltsequenz $M_0, t_1, M_1, \ldots, M_k$ heißt das k-*te Präfix* von σ und die Sequenz $M_k, t_{k+1}, M_{k+1}, \ldots$ das k-*te Suffix* von σ. ○

Die unendliche Sequenz $\sigma_1 = AB, a, CB, (c, DB, d, CB)^\infty$ ist eine Schaltsequenz von Σ_1. Dabei bezeichnen wir für eine endliche Sequenz σ mit $\sigma^\infty = \sigma\sigma\sigma\ldots$ die unendliche Wiederholung von σ. Ist Σ fest, so ist eine Schaltsequenz σ von Σ eindeutig sowohl durch ihre Markierungssequenz $M_0, M_1, M_2, \ldots$ als auch durch ihre Transitionssequenz $t_1, t_2, \ldots$ bestimmt. Dabei ist die angenommene Schlichtheit des Netzes notwendig.

Die Menge aller Schaltsequenzen eines initialisierten Netzes kann man durch einen *maximalen Schaltbaum* zusammenfassend repräsentieren. Schaltbäume spielen in

dieser Arbeit eine untergeordnete Rolle. Wir werden sie daher nur skizzieren. Abbildung 1.2 zeigt den maximalen Schaltbaum von Σ_1. Ein *maximaler Schaltbaum* eines initialisierten Netzes Σ ist ein ggf. unendlicher Baum ϑ, dessen Knoten mit Markierungen von Σ und dessen Kanten mit Transitionen von Σ beschriftet sind. Dabei ist die Wurzel von ϑ mit der Anfangsmarkierung von Σ beschriftet. Weiterhin gilt für jeden Knoten ν des maximalen Schaltbaumes, der mit einer Markierung M von Σ beschriftet ist: Gibt es eine Markierung M' von Σ mit $M \xrightarrow{t} M'$ für irgendeine Transition t, so gibt es genau einen Nachfolgerknoten ν' von ν, der mit M' beschriftet ist. Die Kante von ν nach ν' ist dabei mit t beschriftet.

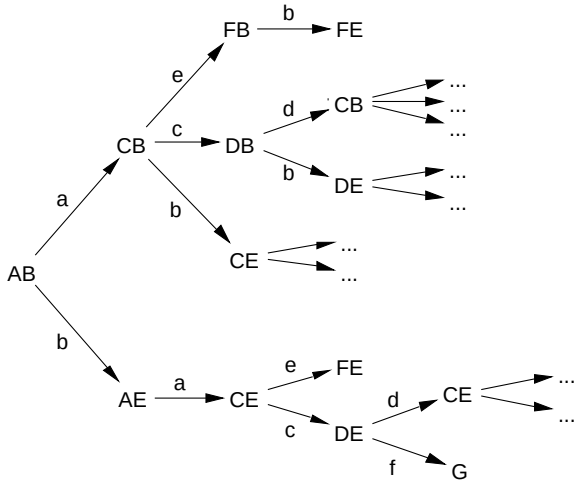


Abb. 1.2: Der maximale Schaltbaum von Σ_1.

Der maximale Schaltbaum eines initialisierten Netzes ist bis auf Isomorphie eindeutig bestimmt. Ein Teilbaum des maximalen Schaltbaums eines initialisierten Netzes Σ, dessen Wurzel mit der Wurzel des maximalen Schaltbaums übereinstimmt, heißt *Schaltbaum* von Σ. Jeder Weg in einem Schaltbaum von Σ, der von der Wurzel ausgeht, repräsentiert eine Schaltsequenz von Σ.

Im nächsten Abschnitt definieren wir die nicht-sequentielle Semantik von Netzen.

1.2.2 Abläufe und Abwicklungen

In diesem Abschnitt definieren wir nicht-sequentielle Abläufe eines initialisierten Netzes sowie deren Einbettung in eine verzweigte Struktur – die *maximale Abwicklung* eines initialisierten Netzes. Einen nicht-sequentiellen Ablauf nennen wir im folgenden auch kurz *Ablauf*. Bevor wir uns formalen Definitionen zuwenden, beschreiben wir die Konzepte informell. Eine Theorie nicht-sequentieller Abläufe findet man in [19].

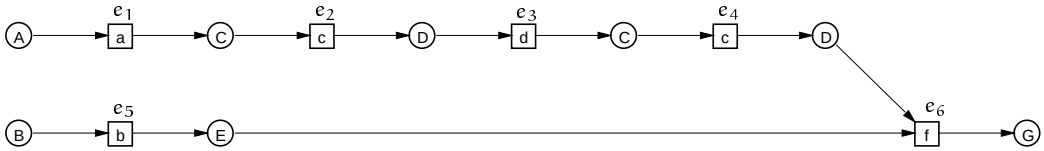


Abb. 1.3: Ein nicht-sequentieller Ablauf ρ_1 von Σ_1.

Sowohl Abläufe als auch Abwicklungen repräsentieren wir mit Hilfe spezieller azyklischer Netze, die wir als *Abwicklungsnetze* bezeichnen. Zur deutlichen Unterscheidung eines Abwicklungsnetzes von einem gewöhnlichen Netz, nennen wir eine Stelle eines Abwicklungsnetzes *Bedingung* und eine Transition eines Abwicklungsnetzes *Ereignis*. Abb. 1.3 zeigt einen endlichen nicht-sequentiellen Ablauf ρ_1 von Σ_1. Die Ereignisse $e_1, \ldots, e_6$ von ρ_1 sind mit Transitionen von Σ_1 beschriftet – ein Ereignis repräsentiert das Schalten einer Transition. Die Bedingungen von ρ_1 sind mit den Stellen von Σ_1 beschriftet – eine Bedingung repräsentiert eine Marke auf einer Stelle. Das Ereignis e_1 von ρ_1 repräsentiert das Schalten von Transition a in der Anfangsmarkierung von Σ_1. Dabei wird die Marke auf A verbraucht und eine Marke auf C erzeugt.

Die Kanten des dem Ablauf zugrundeliegenden Abwicklungsnetzes erzeugen eine Ordnung auf den Ereignissen und Bedingungen. Diese Ordnung bezeichnen wir als *Kausalordnung*. Zentrale Eigenschaft nicht-sequentieller Abläufe ist es, daß die Ereignisse eines Ablaufs – nicht wie in Schaltsequenzen totalgeordnet – sondern halbgeordnet sind. In ρ_1 sind e_1 und e_2 kausal geordnet, e_1 und e_5 jedoch ungeordnet. Wir sagen: e_1 und e_5 finden *nebenläufig* (oder *unabhängig voneinander*) statt. Die Menge der Bedingungen, die bezüglich der Kausalordnung minimal sind (in Abb. 1.3: die am weitesten links sind), repräsentiert die Anfangsmarkierung von Σ_1.

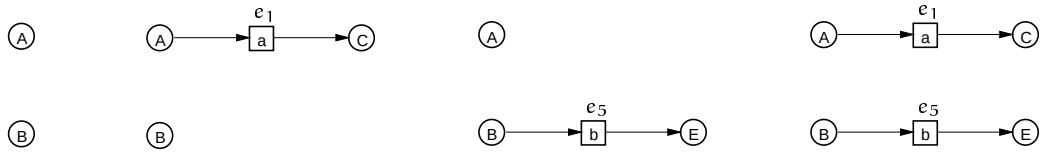
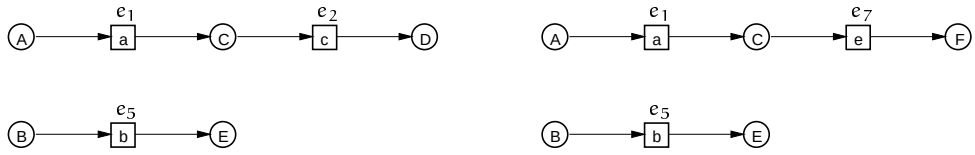


Abb. 1.4: Vier Präfixe von ρ_1.

Abb. 1.4 zeigt vier weitere Abläufe $\rho_2, \ldots, \rho_5$ von Σ_1, die *Präfixe* von ρ_1 sind. Dabei ist ρ_2 der *ereignislose Ablauf* von Σ_1, d.h. der Ablauf von Σ_1, in dem keine Transition schaltet. Der Ablauf ρ_3 entsteht aus ρ_2 durch Anhängen des Ereignisses

e_1, d.h. durch Schalten von a. Der Ablauf ρ_4 entsteht aus ρ_2 durch Anhängen des Ereignisses e_5, d.h. durch Schalten von b. Bei einem nicht-sequentiellen Ablauf ist es also möglich, an verschiedenen Stellen des „Endes“ des Ablaufs Ereignisse anzuhängen. Der Ablauf ρ_5 kann sowohl aus ρ_3 als auch aus ρ_4 durch Anhängen eines einzelnen Ereignisses gewonnen werden, d.h. sowohl ρ_3 als auch ρ_4 ist Präfix von ρ_5. Ist für zwei Abläufe ρ, ρ' eines initialisierten Netzes ρ ein Präfix von ρ', so sagen wir umgekehrt ρ' ist eine *Fortsetzung* von ρ. Die Abläufe ρ_3 und ρ_4 haben die gemeinsame Fortsetzung ρ_5. Haben zwei Abläufe eine gemeinsame Fortsetzung, so nennen wir sie *kompatibel*.



(a) ρ_6 (b) ρ_7

Abb. 1.5: Zwei zueinander inkompatible Fortsetzungen von ρ_5.

Abb. 1.5 zeigt zwei Fortsetzungen ρ_6 und ρ_7 von ρ_5, die ihrerseits keine gemeinsame Fortsetzung haben – ρ_6 und ρ_7 sind *inkompatibel*. Diese Inkompatibilität reflektiert den Konflikt um die von e_1 erzeugte Marke auf C. Diese kann höchstens von einem der beiden Ereignisse e_2 und e_7 verbraucht werden, d.h. entweder durch Schalten von c oder durch Schalten von e.

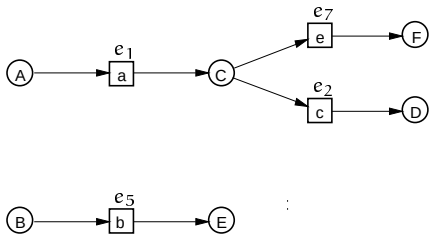


Abb. 1.6: Abwicklung π_1 von Σ_1 – Zusammenfassung von ρ_6 und ρ_7.

So wie verschiedene Schaltsequenzen zu einem Schaltbaum zusammengefaßt werden können, so können verschiedene Abläufe zu einer Struktur zusammengefaßt werden, die wir *Abwicklung* (in der Literatur: *branching process*) nennen. Beim Zusammenfassen von zwei Abläufen werden ihre gemeinsamen Präfixe miteinander verschmolzen. Fassen wir zwei kompatible Abläufe zusammen, so entsteht wieder ein Ablauf – die Zusammenfassung von ρ_3 und ρ_4 ist ρ_5. Jeder Ablauf ist auch eine Abwicklung. Fassen wir zwei inkompatible Abläufe zusammen, so ist die entstehende

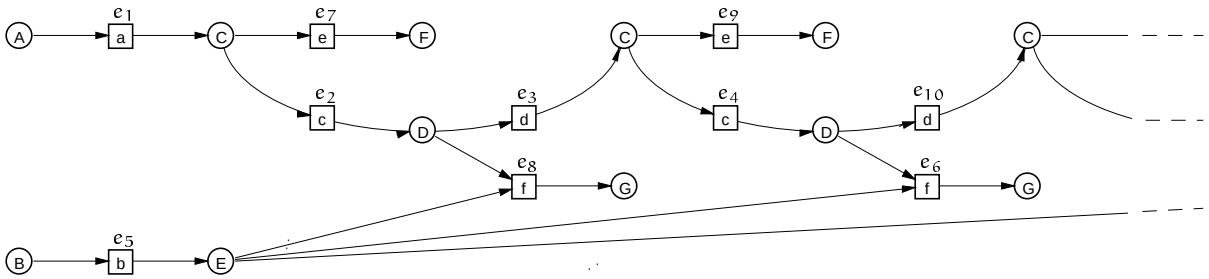


Abb. 1.7: Die maximale Abwicklung π von Σ_1.

Abwicklung kein Ablauf. Abb. 1.6 zeigt die Abwicklung π_1 von Σ_1, die die Abläufe ρ_6 und ρ_7 zusammenfasst.

Während in einem Ablauf jede Bedingung höchstens ein Ereignis im Nachbereich hat, kann eine Bedingung einer Abwicklung auch mehrere Ereignisse im Nachbereich haben. Hat eine Bedingung mehrere Ereignisse im Nachbereich, so sprechen wir von einem *Konflikt* der Ereignisse. Genauer: Zwei verschiedene Ereignisse e_1 und e_2 einer Abwicklung *stehen in unmittelbarem Konflikt zueinander*, falls sie eine gemeinsame Bedingung im Vorbereich haben. In π_1 stehen e_2 und e_7 im unmittelbaren Konflikt zueinander.

Abb. 1.7 zeigt die (bis auf Isomorphie eindeutig bestimmte) maximale Abwicklung π von Σ_1. Die maximale Abwicklung eines initialisierten Netzes Σ repräsentiert alle Abläufe von Σ. So finden wir beispielsweise ρ_1 als Teilstruktur in π wieder. Wir betrachten nun Ereignisse von π die im unmittelbaren Konflikt zueinander stehen. In π sind zum Beispiel e_2 und e_7 sowie e_3 und e_8 in unmittelbarem Konflikt zueinander. Stehen zwei Ereignisse in unmittelbarem Konflikt, so kommen sie in keinem Ablauf gemeinsam vor. Auch e_7 und e_3, die nicht in unmittelbarem Konflikt stehen, können nicht gemeinsam in einem Ablauf vorkommen, da e_7 und e_2 in unmittelbarem Konflikt stehen und e_3 von e_2 kausal abhängt. Können zwei Ereignisse einer Abwicklung nicht gemeinsam in einem Ablauf vorkommen, so sind sie *im Konflikt zueinander*. Konflikt, Kausalität und Nebenläufigkeit formalisieren wir nun in der folgenden Definition.

Definition 1.6 (Kausalität, Konflikt, Nebenläufigkeit)

Sei $K = (B, E; <)$ ein vorgängerendliches, azyklisches Systemnetz.

(a) Da K azyklisch ist, ist $<^+$ eine Ordnung, die wir als *Kausalordnung* bezeichnen. Dabei schreiben wir im folgenden $<$ anstelle von $<^+$. Gilt für zwei Elemente x_1, x_2 die Beziehung $x_1 < x_2$, so sagen wir x_1 ist *kausaler Vorgänger* von x_2. Gilt $x_1 < x_2$ oder $x_1 = x_2$, so schreiben wir $x_1 \leq x_2$. Zwei Elemente x_1 und x_2 sind *kausal abhängig*, falls $x_1 < x_2$ oder $x_2 < x_1$ gilt. Eine Menge paarweise

kausal abhängiger Elemente heißt *li-Menge*[4].

(b) Zwei verschiedene Ereignisse e_1, e_2 sind *in unmittelbarem Konflikt*, falls ${}^\bullet e_1 \cap {}^\bullet e_2 \neq \emptyset$. Zwei Elemente x_1, x_2 sind *in Konflikt* (Notation: $x_1 \# x_2$), falls es zwei verschiedene Ereignisse e_1, e_2 gibt, die in unmittelbarem Konflikt sind, so daß $e_i \leq x_i$ ist, für $i = 1, 2$.

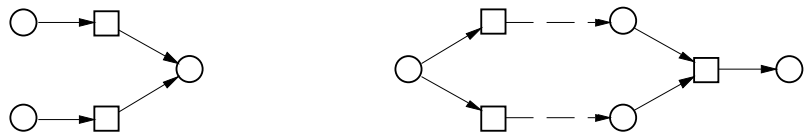
(c) Zwei verschiedene Elemente x_1, x_2 sind *nebenläufig* (oder *unabhängig*) (Notation: x_1 co x_2), falls sie weder kausal abhängig noch im Konflikt sind. Eine Menge paarweise nebenläufiger Elemente heißt co-*Menge*[5].

Wir definieren nun Abwicklungsnetze, die die Grundlage für Abwicklungen bilden.

Definition 1.7 (Abwicklungsnetz)

Ein vorgängerendliches, azyklisches Systemnetz K heißt *Abwicklungsnetz*, wenn gilt

1. ${}^\circ K$ ist endlich.
2. Für jede Bedingung b von K gilt $|{}^\bullet b| \leq 1$.
3. Für kein Ereignis e von K gilt $e \# e$. ○



(a) Forderung 2 (b) Forderung 3

Abb. 1.8: Durch Definition 1.7 verbotene Strukturen.

Die beiden wesentlichen Forderungen 2. und 3. in Definition 1.7 verbieten die Strukturen, die in Abb. 1.8 dargestellt sind. Forderung 2 verbietet dabei die Struktur in Abb. 1.8(a). Diese Struktur heißt *Rückwärtskonflikt*. Forderung 2 besagt, daß jede Bedingung eines Abwicklungsnetzes höchstens durch ein Ereignis erzeugt wird. Forderung 3 verbietet Maschen von einer Bedingung zu einem Ereignis wie in Abb. 1.8(b). Sie besagt, daß die Vorbedingungen eines Ereignisses nie in Konflikt zueinander stehen, d.h. für jedes Ereignis e eines Abwicklungsnetzes gilt: ${}^\bullet e$ ist eine co-Menge.

[4] *li* steht für *line*; mit *li* wird traditionell kausale Abhängigkeit in der Petrinetztheorie bezeichnet.
[5] *co* steht für *concurrent*

Aus Definition 1.7 ergibt sich, daß auch für keine Bedingung b eines Abwicklungsnetzes $b \# b$ gilt. Desweiteren gilt für jedes Ereignis e auch: $e^\bullet$ ist eine co-Menge. Zentrale Eigenschaft eines Abwicklungsnetzes ist: Für je zwei Elemente x_1, x_2 gilt genau eine der folgenden fünf Bedingungen: $x_1 = x_2$ oder $x_1 < x_2$ oder $x_2 < x_1$ oder $x_1 \# x_2$ oder $x_1 \text{ co } x_2$.

Wir definieren nun die *Abwicklungen* eines initialisierten Netzes nach Engelfriet [33]. Später definieren wir Abläufe als spezielle Abwicklungen. Abwicklungen wurden bereits von Nielsen, Plotkin und Winskel in [68] eingeführt. Wir definieren zunächst den Begriff der Abwicklung und diskutieren die Definition im Anschluß.

Definition 1.8 (Abwicklung)

Sei $\Sigma = (N, M^0)$ ein initialisiertes Netz mit $N = (P, T; F)$ und sei $K = (B, E, <)$ ein Abwicklungsnetz.

(a) Eine N-*Beschriftung* von K ist eine Abbildung $l : B \cup E \to P \cup T$, so daß $l(B) \subseteq P$ und $l(E) \subseteq T$. Wir erweitern l zu $\overline{l} : \mathfrak{S}(B) \to \mathfrak{M}(P)$ durch $\overline{l}(B')(p) = |\{b \in B' \mid l(b) = p\}|$ für alle endlichen $B' \subseteq B$ und wir schreiben l anstelle von $\overline{l}$ wenn dies eindeutig ist.

(b) $\pi = (K, l)$ heißt *Abwicklung* von Σ, falls l eine N-Beschriftung von K ist, so daß:

1. $l(^\circ K) = M^0$.
2. Für jedes Ereignis e von K gilt: $l(^\bullet e) = l(e)^-$ und $l(e^\bullet) = l(e)^+$.
3. Für alle Ereignisse e_1, e_2 von K gilt: $^\bullet e_1 = {}^\bullet e_2 \wedge l(e_1) = l(e_2) \Rightarrow e_1 = e_2$. $\circ$

Wir erläutern nun die Forderungen 1 bis 3 aus Definition 1.8(b). Forderung 1 besagt, daß der Anfang einer Abwicklung genau die Anfangsmarkierung des initialisierten Netzes repräsentiert. Forderung 2 formalisiert, daß jedes Ereignis einer Abwicklung genau das Schalten einer Transition repräsentiert, d.h. Forderung 2 formalisiert, daß Verbrauch und Produktion von Bedingungen der Schaltregel gehorcht. Forderung 3 besagt, daß der Verbrauch einer Menge von Bedingungen durch das Schalten einer Transition t höchstens durch ein Ereignis der Abwicklung repräsentiert wird, d.h. kann eine endliche co-Menge B' von Bedingungen durch das Schalten einer Transition t verbraucht werden (d.h. $l(B') = t^-$), so gibt es höchstens ein Ereignis e mit $^\bullet e = B'$ und $l(e) = t$.

Zum besseren Verständnis von Abwicklungen wollen wir hier noch den Abwicklungsprozeß beschreiben, d.h. wir erklären endliche Abwicklungen induktiv. Eine unendliche Abwicklung kann als Grenzwert einer Folge von endlichen Abwicklungen aufgefaßt werden. Sei $\Sigma = (N, M^0)$ ein initialisiertes Netz. Der Abwicklungsprozeß beginnt im ereignislosen Ablauf von Σ, d.h. der ereignislose Ablauf von Σ stellt die minimale Abwicklung von Σ dar. Der ereignislose Ablauf besteht lediglich aus einer

endlichen Menge B von Bedingungen, so daß $l(B) = M^0$. Sei nun für den Induktionsschritt π eine endliche Abwicklung von Σ sowie t eine Transition von Σ. Aus π erhalten wir eine neue Abwicklung π' durch Anfügen eines neuen Ereignisses e' mit $l(e') = t$ und neuer Bedingungen B' mit $l(B') = t^+$ an π, falls die folgenden zwei Bedingungen erfüllt sind: (1) Es gibt eine endliche co-Menge B'' von Bedingungen von π mit $l(B'') = t^-$ und (2) es gibt noch kein Ereignis e in π mit ${}^\bullet e = B''$ und $l(e) = t$. Dann ensteht π' aus π durch Hinzunahme von e' und B' und Fortsetzung der Kausalordnung und der Beschriftung, so daß ${}^\bullet e' = B''$ und $l(e') = t$ und $e'^\bullet = B'$ sowie $l(B') = t^+$. Dann ist π' eine endliche Abwicklung von Σ.

Die *maximale Abwicklung* eines initialisierten Netzes entsteht, falls der Abwicklungsprozeß „maximal" durchgeführt wird, d.h.: Eine Abwicklung π von Σ heißt *maximal*, falls für jede endliche co-Menge von Bedingungen B' von π und jede Transition t von Σ gilt: Ist $l(B') = t^-$, dann gibt es ein Ereignis e von π mit ${}^\bullet e = B'$ und $l(e) = t$.

Zwei Abwicklungen π_1 und π_2 können sich in der Menge von Bedingungen und Ereignissen unterscheiden, ansonsten „strukturell" aber identisch sein – π_1 und π_2 sind dann *isomorph* zueinander. Die Isomorphie zweier Abwicklungen formalisieren wir weiter unten. Die maximale Abwicklung eines initialisierten Netzes ist bis auf Isomorphie eindeutig bestimmt. Jede Abwicklung von Σ ist ein *Präfix* der maximalen Abwicklung von Σ. Ein Präfix einer gegebenen Abwicklung π eines initialisierten Netzes Σ ist intuitiv eine Abwicklung π' von Σ, die weniger weit als π abgewickelt ist. Die Präfixbildung bei Abwicklungen können wir uns über *vorgängerabgeschlossene Ereignismengen* vorstellen.

Eine Menge E' von Ereignissen einer Abwicklung mit Gesamtereignismenge E heißt *vorgängerabgeschlossen*, falls für alle $e \in E'$ gilt: $\downarrow e \cap E \subseteq E'$. Die Vereinigung und der Durchschnitt von zwei vorgängerabgeschlossenen Ereignismengen ist wieder vorgängerabgeschlossen. Jede vorgängerabgeschlossene Ereignismenge E' einer Abwicklung $\pi = (K, l)$ definiert einen Präfix $\pi_{E'}$ von π. Die Ereignismenge von $\pi_{E'}$ ist E', die Menge der Bedingungen von $\pi_{E'}$ ist $B' = {}^\circ K \cup \bigcup_{e \in E'} ({}^\bullet e \cup e^\bullet)$. Die Kausalordnung von $\pi_{E'}$ ist die Einschränkung der Kausalordnung von π auf $E' \cup B'$ und die Beschriftung von $\pi_{E'}$ ist die Einschränkung der Beschriftung von π auf $E' \cup B'$. Jede zu $\pi_{E'}$ isomorphe Abwicklung betrachten wir auch als Präfix von π, d.h. wir wollen bei der Definition von Präfixen von Isomorphie abstrahieren. Daher definieren wir nun die Präfixrelation auf der Menge aller Abwicklungen eines initialisierten Netzes mit Hilfe von Homomorphismen.

Definition 1.9 (Präfix)

Seien Σ ein initialisiertes Netz und π_1, π_2 Abwicklungen von Σ mit $\pi_i = ((B_i, E_i, \leqslant_i), l_i)$ für $i = 1, 2$. π_1 ist *Präfix* von π_2 (Notation: $\pi_1 \sqsubseteq \pi_2$), falls zwei Injektionen $g : E_1 \to E_2$ und $h : B_1 \to B_2$ existieren, so daß für alle $b \in B_1$ und alle $e \in E_1$ gilt:

1. $h({}^\bullet e) = {}^\bullet g(e)$ und $h(e^\bullet) = g(e)^\bullet$,
2. $l_1(e) = l_2(g(e))$ und $l_1(b) = l_2(h(b))$ sowie
3. $h({}^\circ K_1) = {}^\circ K_2$.

Für $\pi_1 \sqsubseteq \pi_2$ sagen wir auch: π_2 *setzt* π_1 *fort.* ○

Forderung 1 in Definition 1.9 besagt, daß π_1 und π_2 die gleiche Netzstruktur haben. Forderung 2 besagt, daß strukturell einander entsprechende Elemente auch gleich beschriftet sind. Forderung 3 sagt, daß der Anfang des Präfixes auf den Anfang der Fortsetzung abgebildet wird.

Die Abbildungen g und h in Definition 1.9 heißen *Präfixinjektionen.* Ist π_1 ein Präfix von π_2, so sind die Präfixinjektionen eindeutig bestimmt (Beweis wie in [47]; wichtig hierbei ist, daß die Anfangsmarkierung sowie für jede Transition t die Vormarkierung t^- sicher ist). Gilt sowohl $\pi_1 \sqsubseteq \pi_2$ als auch $\pi_2 \sqsubseteq \pi_1$, so sind π_1 und π_2 *isomorph* (Notation: $\pi_1 \equiv \pi_2$). Gilt $\pi_1 \sqsubseteq \pi_2$ und nicht $\pi_1 \equiv \pi_2$, so schreiben wir auch $\pi_1 \sqsubset \pi_2$. Oft unterscheiden wir isomorphe Abwicklungen nicht, d.h. jede Abwicklung π steht dann für ihre Isomorphieklasse $[\pi] = \{\pi' \mid \pi' \equiv \pi\}$. Dann schreiben wir auch π anstelle von $[\pi]$. Im weiteren werden wir – je nach Zweckmäßigkeit – eine Abwicklung manchmal als einzelnes Objekt und manchmal als Repräsentant ihrer Isomorphieklasse betrachten.

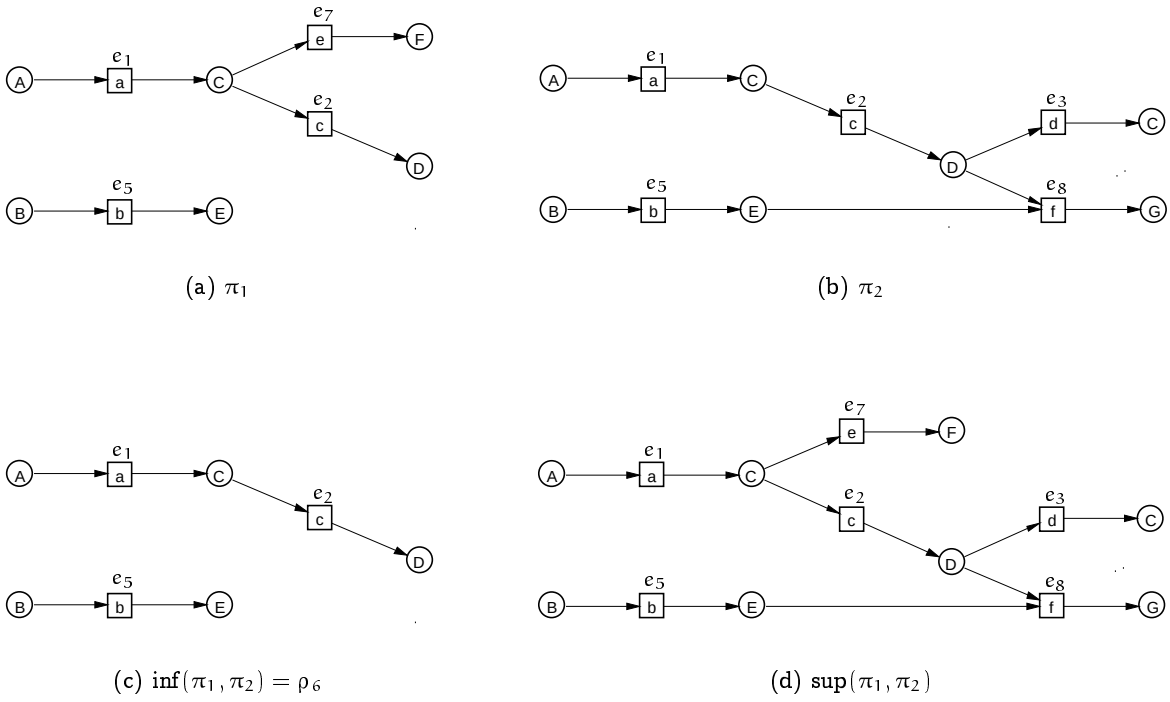
Engelfriet zeigt in [33], daß die Präfixrelation auf den Isomorphieklassen einen vollständigen Verband bildet. Daher sind das Infimum und das Supremum zweier Abwicklungen (sogar jeder Menge von Abwicklungen) bis auf Isomorphie eindeutig bestimmt:

Definition 1.10 (Infimum, Supremum)

Seien Σ ein initialisiertes Netz und π_1, π_2 Abwicklungen von Σ. Es sei $\inf(\pi_1, \pi_2) = \kappa$, falls κ die bzgl. $\sqsubseteq$ größte Abwicklung mit der Eigenschaft $\kappa \sqsubseteq \pi_i$ für $i = 1, 2$ ist. Analog sei $\sup(\pi_1, \pi_2) = \kappa$, falls κ die bzgl. $\sqsubseteq$ kleinste Abwicklung mit der Eigenschaft $\pi_i \sqsubseteq \kappa$ für $i = 1, 2$ ist. ○

Die Infimums- und Supremumsbildung können wir uns über vorgängerabgeschlossene Ereignismengen vorstellen: Sind E_1, E_2 vorgängerabgeschlossene Mengen einer Abwicklung und ist π_i das von E_i erzeugte Präfix für $i = 1, 2$, dann erzeugt $E_1 \cap E_2$ das Infimum $\inf(\pi_1, \pi_2)$ und $E_1 \cup E_2$ das Supremum $\sup(\pi_1, \pi_2)$. Abb. 1.9 zeigt ein Beispiel für die Infimums- und Supremumsbildung von zwei Abwicklungen.

Manchmal wollen wir auch Ereignisse von Abwicklungen nur modulo Isomorphie betrachten, d.h. wir wollen Ereignisse verschiedener Abwicklungen in Beziehung setzen. Dies können wir mittels der eindeutigen Präfixinjektion tun: Seien π_1, π_2



(a) π_1 (b) π_2

(c) $\inf(\pi_1, \pi_2) = \rho_6$ (d) $\sup(\pi_1, \pi_2)$

Abb. 1.9: Infimums- und Supremumsbildung auf Abwicklungen.

zwei Abwicklungen mit Ereignismengen E_i und sei e_i ein Ereignis von π_i sowie g_i die Präfixinjektion von π_i nach $\sup(\pi_1, \pi_2)$ jeweils für $i = 1, 2$. Dann schreiben wir $e_1 \equiv e_2$, falls $g_1(e_1) = g_2(e_2)$ gilt sowie $e_1 \; R \; e_2$ für $R \in \{<, co, \#\}$, falls $g_1(e_1) \; R \; g_2(e_2)$ gilt. Gilt $\pi_1 \sqsubseteq \pi_2$ so ist $\sup(\pi_1, \pi_2) = \pi_2$. Dann schreiben wir $\pi_1 \overset{e}{\sqsubset} \pi_2$ für $e \in E_2$, falls $E_2 = g_1(E_1) \cup \{e\}$, d.h. falls π_2 aus π_1 durch Anhängen von e hervorgeht.

Wir definieren nun den Begriff des (nicht-sequentiellen) Ablaufs. Ein *Ablauf* eines initialisierten Netzes Σ ist eine konfliktfreie Abwicklung[6] von Σ.

Definition 1.11 (Ablauf)

Sei Σ ein initialisiertes Netz. Ein *Ablauf* von Σ ist eine Abwicklung ρ von Σ, so daß für alle Bedingungen b von ρ gilt: $|b^\bullet| \leq 1$.

Sei π eine Abwicklung von Σ. Ein Ablauf ρ von Σ ist ein *Ablauf von* π, falls $\rho \sqsubseteq \pi$. Die Menge aller Abläufe von π bezeichnen wir durch $\mathfrak{R}(\pi)$, die Menge aller bzgl. $\sqsubseteq$ maximalen Abläufe von π durch $\mathfrak{R}_{max}(\pi)$ und die Menge aller endlichen Abläufe von π durch $\mathfrak{R}_{fin}(\pi)$. Mit $\mathfrak{R}(\Sigma)$ bezeichnen wir die Menge aller Abläufe von Σ. Der bzgl. $\sqsubseteq$ minimale Ablauf von Σ heißt auch *ereignisloser Ablauf* von Σ. ○

[6] In der Literatur wird dieser Begriff als *Prozeß* bezeichnet. Dieses Wort wollen wir wegen seiner vielfältigen Bedeutung in verteilten Systemen vermeiden.

Wie bereits beschrieben, repräsentiert der ereignislose Ablauf die Anfangsmarkierung des initialisierten Netzes. Der Ablauf ρ_6 von Σ_1 in Abb. 1.9(c) ist ein maximaler Ablauf von π_1 in Abb. 1.9(a) und ein nicht-maximaler Ablauf von π_2 in Abb. 1.9(b). In einem Ablauf gilt für je zwei Elemente x_1 und x_2 genau eine der folgenden vier Beziehungen: $x_1 = x_2$ oder $x_1 < x_2$ oder $x_2 < x_1$ oder x_1 co x_2.

Wir führen nun eine neue Notation für die Beschriftung von Bedingungen und Ereignissen einer Abwicklung ein.

Notation 1.12

Sei $\pi = (K, l)$ eine Abwicklung. Für eine Bedingung b von π sowie für ein Ereignis e von π schreiben wir in Zukunft $\widetilde{b}$ und $\widetilde{e}$ anstelle von $l(b)$ bzw. $l(e)$, falls Mehrdeutigkeiten ausgeschlossen sind. Auch für endliche Mengen B' von Bedingungen schreiben wir $\widetilde{B'}$ anstelle von $l(B')$. Desweiteren schreiben wir π° anstelle von K°; π° heißt auch das *Ende* von π.

Ist das Ende einer Abwicklung leer, so ist die Abwicklung unendlich. Umgekehrt ist das Ende einer unendlichen Abwicklung nicht notwendig leer. Dies gilt insbesondere auch für Abläufe. Abb. 1.10 zeigt den unendlichen Ablauf ρ_8 von Σ_1 . Das Ende von ρ_8 ist nicht leer – es enthält eine mit B beschriftete Bedingung – die Marke auf B aus der Anfangsmarkierung von Σ_1 wurde in ρ_8 nicht verbraucht. Der Ablauf ρ_8 zeigt, daß manche unendliche nicht-sequentielle Abläufe – im Gegensatz zu unendlichen sequentiellen Abläufen – echt fortgesetzt werden können: Transition b kann im Ende von ρ_8 schalten. Die Fortsetzung von ρ_8 ist in Abb. 1.10 gestrichelt dargestellt.

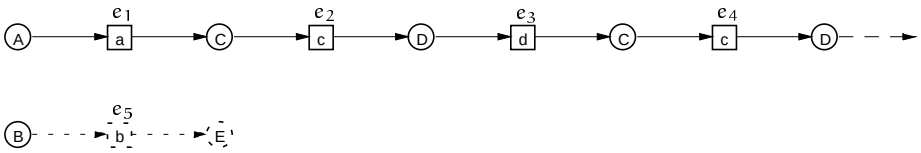


Abb. 1.10: Ein unendlicher, fortsetzbarer Ablauf ρ_8 von Σ_1 .

Wir definieren nun noch den Begriff des *Markierungsschnitts*. Ein *Markierungsschnitt* einer Abwicklung π repräsentiert das Ende eines endlichen Ablaufs von π.

Definition 1.13 (Markierungsschnitt)

Sei $K = (B, E, <)$ ein Abwicklungsnetz. Eine maximale co-Menge von K heißt *Schnitt* von K. Ein endlicher Schnitt C heißt *Markierungsschnitt*, falls $C \subseteq B$. Sind C_1, C_2 zwei Markierungsschnitte, so heißt C_2 *von* C_1 *erreichbar* (Notation: $C_1 \xrightarrow{*} C_2$), falls für alle $b_1 \in C_1$ und alle $b_2 \in C_2$ gilt: $b_1 = b_2$ oder $b_1 < b_2$ oder b_1 co b_2. Für $e \in E$ schreiben wir $C_1 \xrightarrow{e} C_2$, falls $C_2 = (C_1 \setminus {}^\bullet e) \cup e^\bullet$. Der Markierungsschnitt $^\circ K$ heißt *Anfangsschnitt* von ρ. ∘

Ein Markierungsschnitt C einer Abwicklung π repräsentiert den endlichen Ablauf $\alpha \in \mathfrak{R}_{\text{fin}}(\pi)$ der durch die endliche, vorgängerabgeschlossene Ereignismenge $\downarrow C \cap E$ gegeben ist, wobei E die Menge der Ereignisse von π und $\downarrow C = \bigcup_{b \in C} \downarrow b$ ist. Ist C ein Markierungsschnitt einer Abwicklung eines initialisierten Netzes Σ, so ist $\widetilde{C}$ eine erreichbare Markierung von Σ.

Wir wollen uns nun mit Infima und Suprema von Abläufen beschäftigen. Sind ρ_1, ρ_2 Abläufe eines initialisierten Netzes Σ, so ist $\inf(\rho_1, \rho_2)$ ein Ablauf von Σ. Das Supremum zweier Abläufe ist nicht notwendig ein Ablauf. Dies führt zur folgenden Definition:

Definition 1.14 (Kompatible Abläufe)

Sei Σ ein initialisiertes Netz. Zwei Abläufe ρ_1, ρ_2 von Σ sind *kompatibel* (Notation: $\rho_1 \parallel \rho_2$), falls $\sup(\rho_1, \rho_2)$ ein Ablauf von Σ ist. Sind ρ_1 und ρ_2 nicht kompatibel, so sind sie *inkompatibel* (Notation: $\rho_1 \nparallel \rho_2$). ○

Zwei Abläufe ρ_1, ρ_2 sind genau dann inkompatibel, falls es ein Ereignis e_1 von ρ_1 und ein Ereignis e_2 von ρ_2 gibt, so daß $e_1 \# e_2$.

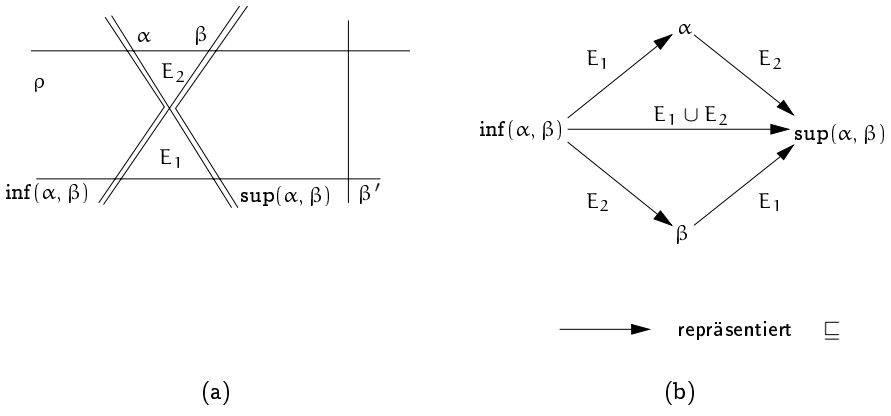


Abb. 1.11: Infimum und Supremum von kompatiblen endlichen Abläufen.

Abb. 1.11(a) zeigt einen Ablauf ρ, in dem vertikale Linien Markierungsschnitte und damit endliche Präfixe von ρ darstellen. Zwei Präfixe desselben Ablaufs sind immer kompatibel. Die Präfixe α und β sind in allgemeiner Lage: Es gilt weder $\alpha \sqsubseteq \beta$ noch $\beta \sqsubseteq \alpha$ – die zugehörigen Markierungsschnitte sind nicht voneinander erreichbar, wohingegen der Markierungsschnitt von β' sowohl vom α- als auch vom β-Markierungsschnitt erreichbar ist. Die Präfixe α und β unterscheiden sich durch die Ereignismengen E_1 bzw. E_2. Für alle $e_1 \in E_1$ und alle $e_2 \in E_2$ gilt: e_1 co e_2. Abb. 1.11(b) stellt denselben Sachverhalt als Diamant dar.

1.2.3 Sequentialisierung von Abläufen

In diesem Unterabschnitt stellen wir die Beziehung von Schaltsequenzen und Abläufen eines initialisierten Netzes dar. Mit diesem Unterabschnitt schließen wir den Abschnitt über Petrinetze und ihre Semantik ab.

Jeder Ablauf repräsentiert eine Menge von Schaltsequenzen. Eine Schaltsequenz eines Ablauf ρ erhält man, indem man die Kausalordnung auf den Ereignissen von ρ zu einer Totalordnung vervollständigt.

Definition 1.15 (Schaltsequenz eines Ablaufs)

Sei Σ ein initialisiertes Netz und ρ ein Ablauf von Σ mit Ereignismenge E. Eine alternierende Sequenz $\tau = C_0, e_1, C_1, e_2, \ldots$ von Markierungsschnitten und Ereignissen von ρ heißt *Sequentialisierung* von ρ, falls die folgenden drei Bedingungen gelten: (1) C_0 ist der Anfangsschnitt von ρ, (2) für alle i gilt: $C_i \stackrel{e_{i+1}}{\rightarrow} C_{i+1}$ sowie (3) $E = \{e_0, e_1, \ldots\}$. Die Schaltsequenz $\sigma_\tau = \widetilde{C_0}, \widetilde{e_1}, \ldots$ von Σ heißt auch *Schaltsequenz von* ρ. ○

In einer Schaltsequenz eines Ablaufs werden nach Bedingung (3) in Definition 1.15 alle Ereignisse von ρ repräsentiert. Ist Σ sicher, so definiert jede Schaltsequenz σ von ρ eindeutig eine Sequentialisierung von ρ und jede Position i von σ bestimmt genau ein Ereignis e_i und genau einen Markierungsschnitt C_i von ρ. Sind zwei Ereignisse e_i, e_j einer Sequentialisierung τ nebenläufig, so sagen wir, daß e_i und e_j in τ *zeitlich geordnet* sind.

Die nun folgenden Abschnitte 1.3 bis 1.6 haben gegenüber diesem Abschnitt nur untergeordnete Bedeutung für den Rest der Arbeit. Sie müssen zunächst nicht genau gelesen werden.

1.3 Ablaufeigenschaften

Um später Probleme in verteilten Systemen formal spezifizieren zu können, beschäftigen wir uns in diesem Abschnitt mit *Ablaufeigenschaften*. In 1.3.1 unterscheiden wir Sicherheits- und Lebendigkeitseigenschaften und in 1.3.2 führen wir temporale Logik zur Spezifikation von Ablaufeigenschaften ein.

1.3.1 Sicherheits- und Lebendigkeitseigenschaften

Wir beginnen mit der Definition einer Ablaufeigenschaft. Hierzu wollen wir Abläufe unabhängig davon betrachten, zu welchem initialisierten Netz sie gehören.

Definition 1.16 (Ablaufeigenschaft)

Ist Σ ein initialisiertes Netz mit Stellenmenge P, M eine Markierung, ρ ein Ablauf und σ eine Schaltsequenz von Σ, so heißt M auch *Markierung über* P, ρ auch *Ablauf über* P und σ auch *Schaltsequenz über* P. Eine *Ablaufeigenschaft über* P ist eine Menge von Abläufen über P, eine *Schaltsequenzeigenschaft über* P ist eine Menge von Schaltsequenzen über P. ○

Ist E eine Ablaufeigenschaft über P und ρ ein Ablauf über P, so sagen wir ρ *erfüllt* E, falls $\rho \in E$ und ρ *verletzt* E, falls $\rho \notin E$.

Sowohl bei der Spezifikation als auch bei der Verifikation verteilter Systeme ist die Unterscheidung von *Sicherheits-* und *Lebendigkeitseigenschaften* nützlich. Eine *Sicherheitseigenschaft* drückt intuitiv aus, daß nie etwas Schlimmes passiert, z.B. beim Mutex-Problem, daß beide Agenten nie zugleich in ihrem kritischen Abschnitt sind. Eine *Lebendigkeitseigenschaft* drückt intuitiv aus, daß irgendwann etwas Gutes passiert, z.B. beim Mutex-Problem, daß ein Agent, der in seinen kritischen Abschnitt möchte, irgendwann in seinen kritischen Abschnitt kommt.

Sicherheits- und Lebendigkeitseigenschaften wurden von Alpern und Schneider in [5] für sequentielle Abläufe formalisiert. Alpern und Schneider zeigen, daß jede Schaltsequenzeigenschaft als Durchschnitt einer Sicherheits- und einer Lebendigkeitseigenschaft darstellbar ist. Kindler überträgt in seiner Dissertation [47] die Begriffe und Ergebnisse von Alpern und Schneider auf nicht-sequentielle Abläufe.

Eine Sicherheitseigenschaft ist eine Ablaufeigenschaft, deren Verletzung immer im Endlichen stattfindet:

Definition 1.17 (Sicherheitseigenschaft)

Sei P eine abzählbare Menge. Eine Ablaufeigenschaft S über P heißt *Sicherheitseigenschaft*, falls für jeden Ablauf ρ über P, der S verletzt, ein endlicher Präfix $\alpha \sqsubseteq \rho$ existiert, so daß alle Abläufe über P, die α fortsetzen, S verletzen. ○

Die Menge aller Abläufe eines initialisierten Netzes ist eine Sicherheitseigenschaft. Eine Lebendigkeitseigenschaft ist eine Ablaufeigenschaft, deren Verletzung nie im Endlichen stattfindet:

Definition 1.18 (Lebendigkeitseigenschaft)

Sei P eine abzählbare Menge. Eine Ablaufeigenschaft L über P heißt *Lebendigkeitseigenschaft*, falls es für alle endlichen Abläufe α über P eine Fortsetzung $\rho \sqsupseteq \alpha$ gibt, die L erfüllt. ∘

1.3.2 Temporallogische Eigenschaften

Einige Ablaufeigenschaften beschreiben wir durch *Temporalformeln*. Eine Sprache, die Ablaufeigenschaften beschreibt, heißt auch *Linear-Time-Temporallogik*. Wir orientieren uns dabei an Manna und Pnueli [63], verwenden jedoch nur die temporalen Operatoren $\Diamond$ „irgendwann“, $\Box$ „immer“ und $\rhd$ „führt zu“. Der Operator $\rhd$ „führt zu“ ist dabei dem Formalismus UNITY [24] entlehnt. Wir interpretieren Temporalformeln – im Gegensatz zur Standardliteratur – auf nicht-sequentiellen Abläufen und nicht auf sequentiellen Abläufen. Dieser Unterschied spielt aber in dieser Arbeit keine wesentliche Rolle.

Wir beginnen mit *Zustandsformeln*, die Markierungseigenschaften beschreiben. Eine Zustandsformel ist eine Formel der Form $p(n)$, wobei p eine Stelle und n eine natürliche Zahl ist; $p(n)$ ist in einer Markierung *gültig*, falls in dieser Markierung auf p mindestens n Marken liegen.

Definition 1.19 (Zustandsformel)

Sei P eine abzählbare Menge. Ist $p \in P$ und $n \in \mathbb{N}$, so ist $p(n)$ eine *Zustandsformel* über P. Die Menge aller Zustandsformeln über P bezeichnen wir durch $\mathrm{ZF}(P)$. Anstelle von $p(1)$ schreiben wir auch einfach p. Eine Zustandsformel $\varphi = p(n)$ über P ist in einer Markierung M über P *gültig* (Notation: $M \models \varphi$), falls $M[p] \geq n$. ∘

Aus Zustandsformeln und den Operatoren $\Diamond$, $\Box$ und $\rhd$ sowie booleschen Kombinatoren bauen wir nun *Temporalformeln* auf. Wir lassen dabei beliebige Verschachtelungen zu.

Definition 1.20 (Temporalformel)

Sei P eine abzählbare Menge. Wir definieren die Menge der *Temporalformeln* über P, $\mathrm{TF}(P)$ induktiv:

1. Ist $\varphi \in \mathrm{ZF}(P)$, so ist $\varphi \in \mathrm{TF}(P)$.
2. Sind $\Phi, \Psi \in \mathrm{TF}(P)$, so sind $\Diamond\,\Phi, \Box\,\Phi, \Phi \rhd \Psi \in \mathrm{TF}(P)$.

3. Sind $\Phi, \Psi \in \mathrm{TF}(P)$, so sind $\neg\Phi, \Phi \vee \Psi, \Phi \wedge \Psi, \Phi \Rightarrow \Psi \in \mathrm{TF}(P)$. ∘

Eine Temporalformel interpretieren wir zunächst in einem Markierungsschnitt (also am Ende eines endlichen Präfixes) eines Ablaufs. Eine Zustandsformel wird in einem Markierungsschnitt C durch die durch C gegebene Markierung ausgewertet. Eine Temporalformel $\Box\,\Phi$ ist in einem Markierungsschnitt C gültig, falls Φ in jedem von C aus erreichbaren Markierungsschnitt gültig ist; $\Diamond\,\Phi$ gilt in C, falls es einen von C aus erreichbaren Markierungsschnitt gibt, in dem Φ gültig ist; $\Phi \rhd \Psi$ steht abkürzend für $\Box(\Phi \Rightarrow \Diamond\,\Psi)$.

Definition 1.21 (Gültigkeit von Temporalformeln)

Sei P eine abzählbare Menge, ρ ein Ablauf über P und C ein Markierungsschnitt von ρ. Desweiteren sei $\varphi \in \mathrm{ZF}(P)$ sowie $\Phi, \Psi \in \mathrm{TF}(P)$. Wir definieren:

(a) $\rho, C \models \varphi$, falls $\widetilde{C} \models \varphi$

(b) $\rho, C \models \Diamond\,\Phi$, falls ein von C aus erreichbarer Markierungsschnitt C' von ρ existiert, so daß $\rho, C' \models \Phi$.

(c) Die Gültigkeit boolescher Kombinationen wird wie üblich definiert:

  (i) $\rho, C \models \neg\Phi$, falls nicht $\rho, C \models \Phi$,

  (ii) $\rho, C \models (\Phi \vee \Psi)$, falls $\rho, C \models \Phi$ oder $\rho, C \models \Psi$,

  (iii) $\rho, C \models (\Phi \wedge \Psi)$, falls $\rho, C \models \Phi$ und $\rho, C \models \Psi$ sowie

  (iv) $\rho, C \models (\Phi \Rightarrow \Psi)$, falls $\rho, C \models \neg\Phi$ oder $\rho, C \models \Psi$.

(d) $\rho, C \models \Box\,\Phi$, falls $\rho, C \models \neg\,\Diamond\,\neg\Phi$.

(e) $\rho, C \models \Phi \rhd \Psi$, falls $\rho, C \models \Box(\Phi \Rightarrow \Diamond\,\Psi)$.

Wir sagen Φ *gilt* in ρ (Notation: $\rho \models \Phi$), falls $\rho, C_0 \models \Phi$, wobei C_0 der Anfangsschnitt von ρ ist. ∘

In jedem Ablauf von Σ_1 (siehe Abb. 1.1 auf Seite 11) gilt $\Box(B \vee E \vee G)$; in jedem Ablauf von Σ_1, in dem a schaltet, gilt $A \rhd C$. Als abkürzende Schreibweise verwenden wir manchmal Quantifikation über endlichen Mengen, z.B. steht $\forall x \in A : \Phi(x)$ für $\bigwedge_{x \in A} \Phi(x)$. Gilt $\rho, C \models \Phi$ und ist α das von C bestimmte endliche Präfix von ρ, so schreiben wir auch $\rho, \alpha \models \Phi$.

Eine *temporallogische Eigenschaft* ist eine Ablaufeigenschaft, die durch eine Temporalformel beschrieben wird.

Definition 1.22 (Temporallogische Eigenschaft)

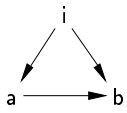
Eine Ablaufeigenschaft E über P heißt *temporallogische Eigenschaft*, falls eine Temporalformel $\Phi \in \mathrm{TF}(P)$ existiert, so daß $E = \{\rho \mid \rho \models \Phi\}$. ∘

1.4 Petrinetzmodellierung verteilter Algorithmen

In diesem Abschnitt führen wir in die Petrinetzmodellierung verteilter Algorithmen ein. Das Studium dieses Abschnittes ist nötig, um die in dieser Arbeit durch Petrinetzmodelle angegebenen verteilten Algorithmen im Detail verstehen zu können. Ein intuitives Verständnis dieser Algorithmen ist jedoch auch ohne weitere Grundlagen möglich.

Oft spielen Daten eine wichtige Rolle in einem verteilten Algorithmus. Die bisher betrachteten Netze sind dann zur Darstellung des verteilten Algorithmus ungeeignet. Wir verwenden dann ein *algebraisches Netz* zur Darstellung des verteilten Algorithmus. In diesem Abschnitt erläutern wir anhand eines einfachen Beispiels, wie wir durch ein algebraisches Netz einen verteilten Algorithmus modellieren. Weitere Beispiele findet man in [81, 82, 49, 91]. Textbücher zu verteilten Algorithmen sind [10, 62, 88]. Das Beispiel, das wir hier betrachten wollen, ist ein einfacher *Netzwerkalgorithmus*. Ein *Netzwerkalgorithmus* ist ein verteilter Algorithmus, bei dem Agenten asynchron und ausschließlich über Nachrichten kommunizieren. In dieser Arbeit kommen einige Netzwerkalgorithmen vor, die als algebraisches Netz modelliert sind. Die formale Definition algebraischer Netze folgt erst im nächsten Abschnitt 1.5.

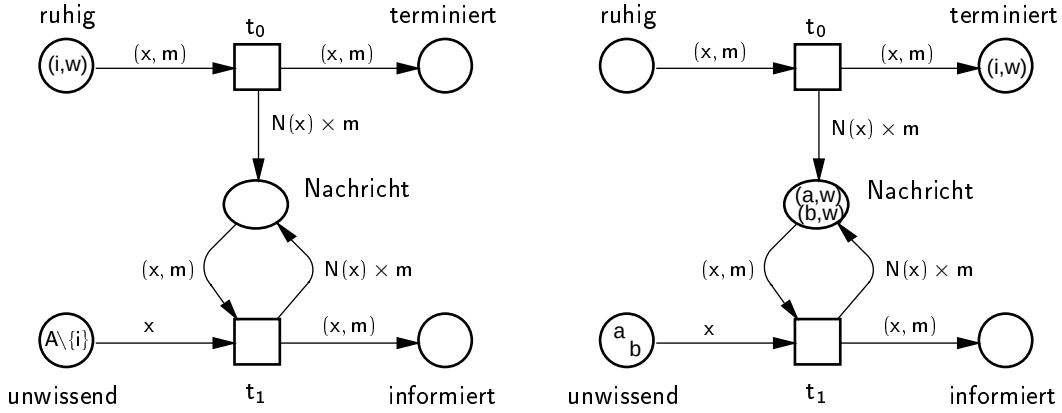
Sei A eine endliche Menge von Agenten und $i \in A$ ein ausgezeichneter Agent, den wir den *Initiator* nennen. Weiterhin sei $N \subseteq A \times A$ eine Relation auf den Agenten, so daß im Graphen (A, N) jeder Agent vom Initiator erreichbar ist. Ein Beispielgraph ist im nebenstehenden Bild zu sehen. Ist $(x, y) \in N$, so heißt y *Nachfolger* von x. Für $x \in A$ bezeichne $N(x)$ die Menge aller Nachfolger von x.



Der Initiator möchte alle Agenten des Netzwerks von einer Information w in Kenntnis setzen. Dazu wird ein einfacher Diffusionsalgorithmus angewendet: Der Initiator versendet die Information durch eine Nachricht an jeden seiner Nachfolger und jeder Nicht-Initiator sendet bei Empfang der ersten Nachricht die Information an jeden seiner Nachfolger weiter.

Diesen Algorithmus modellieren wir durch das *algebraische Petrinetz* in Abb. 1.12. Ein algebraisches Netz ist ein mit Termen einer Algebra beschriftetes endliches Systemnetz. Marken sind jetzt nicht mehr ununterscheidbar schwarz, sondern strukturiert, d.h. Elemente der Algebra.

Die obere Zeile des Netzes beschreibt das Verhalten des Initiators. Der Initiator ist am Anfang *ruhig* und besitzt die Information w, was durch eine Marke (i, w) auf der Stelle *ruhig* modelliert wird. Die einzige Aktion des Initiators wird durch Transition t_0 modelliert. Führt der Initiator diese Aktion aus, so sagen wir t_0 schaltet im *Modus* $[x = i, m = w]$, das heißt: Eine Aktion ist eine Transition zusammen mit einer Belegung der Variablen, die um die Transition herum auftreten. Führt der Initiator



(a) Anfangszustand für beliebige Nachfolgerrelation

(b) Zustand nach Schalten von t_0 für Beispielgraph

Abb. 1.12: Σ_2 – Ein einfacher Diffusionsalgorithmus.

t_0 aus, so sendet er eine Nachricht an jeden seiner Nachfolger. Eine Nachricht wird als Marke (y, m) auf der Stelle *Nachricht* modelliert, wobei y den Empfänger der Nachricht und m den Inhalt der Nachricht darstellt. Daher wird der Versand einer Nachricht mit dem Inhalt m an jeden Nachfolger $y \in N(x)$ modelliert, indem die Menge $\{(y, m) \mid y \in N(x)\} = N(x) \times \{m\}$ von Marken auf die Stelle *Nachricht* gelegt wird[7]. Dies beschreiben wir durch die Anschrift $N(x) \times m$ an der Kante von t_0 zur Stelle *Nachricht*, wobei $N(x) \times m$ eine abkürzende Schreibweise für $N(x) \times \{m\}$ ist.

In graphischen Darstellungen algebraischer Netze verwenden wir Großbuchstaben wie A und $N(x)$ immer zur Bezeichnung von Mengen von Marken, während Kleinbuchstaben einzelne Marken bezeichnen, die abkürzend für eine Menge mit genau einer Marke stehen. Desweiteren verwenden wir Ellipsen, um asynchrone Nachrichtenkanäle zu kennzeichnen, während lokale Zustände und gemeinsame Variablen durch Kreise gekennzeichnet werden. Dies soll graphische Darstellungen lesbarer machen, semantisch hat diese Konvention keine Bedeutung.

Führt der Initiator t_0 aus, so ändert er seinen Zustand von *ruhig* zu *terminiert*, was dadurch modelliert wird, daß das Token (i, w) von der Stelle *ruhig* entfernt und auf die Stelle *terminiert* gelegt wird.

Die untere Zeile des Netzes beschreibt das Verhalten eines Nicht-Initiators. Jeder Nicht-Initiator ist anfangs *unwissend* (modelliert durch die Menge $A \setminus \{i\}$ von Marken auf der Stelle *unwissend*). Empfängt ein Nicht-Initiator x eine Nachricht (x, m),

[7] Damit modellieren wir sichere asynchrone Nachrichtenübertragung: Alle Nachrichten, die gesendet wurden, kommen nach unbestimmter endlicher Zeit an, nicht notwendigerweise in der Reihenfolge, in der sie gesendet wurden.

dann sendet er eine Nachricht (y, m) an jeden seiner Nachfolger $y \in N(x)$ und wechselt seinen Zustand von *unwissend* nach *informiert*. Daher entfernt t_1 eine Marke x von *unwissend* und eine Marke (x, m) von *Nachricht* und legt die Menge $N(x) \times \{m\}$ von Marken auf *Nachricht* sowie eine Marke (x, m) auf *unwissend*. Eine Aktion ist in einer Markierung *aktiviert*, falls alle Marken, die von der Aktion abgezogen werden, in der Markierung vorhanden sind.

Ist eine Algebra für Σ_2 fixiert, d.h. sind A, N, i und w konkret festgelegt, so beschreibt Σ_2 genau ein initialisiertes Netz. Jedes algebraische Netz kann als kompakte Darstellung eines großen, ggf. unendlichen initialisierten Netzes betrachtet werden. Das initialisierte Netz, das durch ein algebraisches Netz $\dot{N}$ beschrieben wird, heißt *Entfaltung von* $\dot{N}$. Die Entfaltung eines algebraischen Netzes definieren wir in Abschnitt 1.5.3.

Jeder Netzwerkalgorithmus kann auf die beschriebene Weise durch ein algebraisches Netz modelliert werden. Desel untersucht in [26] die umgekehrte Fragestellung, wann ein Petrinetz als verteilter Algorithmus aufgefaßt werden kann. Desel zeigt dort desweiteren, daß auch ein verteilter Algorithmus, bei dem Agenten über gemeinsame Variablen kommunizieren, durch ein algebraisches Netz modelliert werden kann.

1.5 Algebraische Netze

Um große, insbesondere unendliche, Systemnetze zu repräsentieren, verwenden wir *algebraische Netze*. Ein Beispiel für ein algebraisches Netz haben wir bereits im Abschnitt 1.4 kennengelernt. Hier definieren wir nun algebraische Netze formal. Dieser Abschnitt kann beim ersten Lesen übersprungen werden. Er ermöglicht es, später aufgeführte algebraische Netze vollständig und detailliert zu erschließen.

Algebraische Netze [80] können auch als Darstellungsform *gefärbter Netze* [44] angesehen werden. Wir definieren algebraische Netze ähnlich wie in [50] – eine Erweiterung der ursprünglichen Definition von [80]. Zunächst definieren wir Signaturen, Variablen und Terme, die eine Grundlage algebraischer Netze bilden. In Unterabschnitt 1.5.3 definieren wir, welches initialisierte Netz durch ein algebraisches Netz repräsentiert wird.

1.5.1 Signaturen, Variablen und Terme

Eine *Signatur* $SIG = (S, OP)$ besteht aus einer endlichen Menge S von *Sortensymbolen* und einer paarweise disjunkten Familie $OP = (OP_a)_{a \in \mathfrak{W}(S)}$ von *Operationssymbolen*. Eine *SIG-Algebra* $\mathcal{A} = ((A_s)_{s \in S}, (f_{op})_{op \in OP})$ besteht aus einer Familie $A = (A_s)_{s \in S}$ von Mengen und einer Familie $(f_{op})_{op \in OP}$ totaler Abbildungen, so daß für jede Operation $op \in OP_{s_1 \ldots s_n s_{n+1}}$ gilt $f_{op} : A_{s_1} \times \ldots \times A_{s_n} \to A_{s_{n+1}}$. Eine Menge A_s der Algebra nennen wir *Trägermenge* und eine Abbildung f_{op} der Algebra nennen wir *Operation*.

Zu einer Signatur $SIG = (S, OP)$ heißt eine paarweise disjunkte Familie $X = (X_s)_{s \in S}$ mit $X \cap OP = \varnothing$ eine *(sortierte) SIG-Variablenmenge*. Aus SIG und X konstruieren wir *Terme* über X. Dabei ist jedem Term genau eine Sorte zugeordnet. Die *Menge der SIG-Terme über* X *der Sorte* s wird durch $\mathfrak{T}_s^{SIG}(X)$ bezeichnet und wie folgt induktiv definiert:

1. Ist $x \in X_s$, so ist $x \in \mathfrak{T}_s^{SIG}(X)$.
2. Sind $u_i \in \mathfrak{T}_{s_i}^{SIG}(X)$ für $i = 1, \ldots, n$ und $op \in OP_{s_1 \ldots s_n s_{n+1}}$, so ist $op(u_1, \ldots, u_n) \in \mathfrak{T}_{s_{n+1}}^{SIG}(X)$.

Die Menge aller Terme (beliebiger Sorte) wird durch $\mathfrak{T}^{SIG}(X)$ bezeichnet. Ein Term ohne Variablen heißt *Grundterm*. Die Menge aller Grundterme bezeichnen wir durch $\mathfrak{T}^{SIG} = \mathfrak{T}^{SIG}(\varnothing)$, die Menge aller Grundterme der Sorte s bezeichnen wir durch $\mathfrak{T}_s^{SIG} = \mathfrak{T}_s^{SIG}(\varnothing)$.

Sei $SIG = (S, OP)$ eine Signatur, $X = (X_s)_{s \in S}$ eine sortierte Variablenmenge über SIG und $\mathcal{A} = ((A_s)_{s \in S}, (f_{op})_{op \in OP})$ eine SIG-Algebra. Eine Abbildung $\beta : X \to A$

heißt *Belegung* von X, falls für jedes $s \in S$ und $x \in X_s$ gilt: $\beta(x) \in A_s$. Wir setzen β induktiv zu einer Abbildung $\overline{\beta} : \mathfrak{T}^{SIG}(X) \to A$ wie folgt fort:

$$\overline{\beta}(op(u_1, \ldots, u_n)) = f_{op}(\overline{\beta}(u_1), \ldots, \overline{\beta}(u_n)) \text{ für } op(u_1, \ldots, u_n) \in \mathfrak{T}^{SIG}(X).$$

Die Abbildung $\overline{\beta}$ heißt β-*Auswertung in* $\mathcal{A}$. Mit $\beta_\varnothing : \varnothing \to A$ bezeichnen wir die eindeutig bestimmte Belegung für die leere Variablenmenge; $\overline{\beta}_\varnothing : \mathfrak{T}^{SIG} \to A$ wertet Grundterme aus.

1.5.2 Algebraische Netze

In diesem Abschnitt definieren wir algebraische Netze. Für algebraische Netze haben Terme, die zu endlichen Mengen ausgewertet werden, spezielle Bedeutung. Um solche Terme auszuzeichnen, definieren wir *Mengensignaturen* und dazu passende Algebren[8]. Eine *Mengensignatur* ist eine Signatur, in der *Grundsorten* und *Mengensorten* unterschieden werden, die mittels einer Bijektion einander zugeordnet werden, d.h. jeder Grundsorte ist genau eine Mengensorte zugeordnet und jeder Mengensorte ist genau eine Grundsorte zugeordnet. In der zu einer Mengensignatur passenden Algebra unterscheiden wir *Grundträgermenge* und *Mengenträgermenge*. Ist zum Beispiel $\mathbb{N}$ eine Grundträgermenge, so ist $\mathfrak{S}(\mathbb{N})$, d.h. die Menge aller endlichen Teilmengen von $\mathbb{N}$, die zugehörige Mengenträgermenge.

Definition 1.23 (Mengensignatur, *MSIG*-Algebra)

Sei $SIG = (S, OP)$ eine Signatur und $GS, MS \subseteq S$; $MSIG = (S, OP, ms)$ ist eine *Mengensignatur* falls $ms : GS \to MS$ eine bijektive Abbildung ist. Ein Element von GS heißt *Grundsorte*, ein Element von MS heißt *Mengensorte* von $MSIG$. Eine SIG-Algebra $\mathcal{A}$ ist eine *$MSIG$-Algebra* falls für jedes $s \in GS$ gilt: $A_{ms(s)} = \mathfrak{S}(A_s)$, d.h., falls für jede *Grundträgermenge* die entsprechende *Mengenträgermenge* tatsächlich die Menge aller endlichen Mengen über der Grundträgermenge ist. ∘

Wir nehmen an, daß zu jeder Mengensignatur $MSIG$ das Sortensymbol $bool \in S$ enthalten ist und in jeder $MSIG$-Algebra die zugehörige Trägermenge $A_{bool} = \mathbb{B}$ ist. Jede Mengensignatur $MSIG = (S, OP, ms)$ ist eine Signatur $SIG = (S, OP)$ und jede $MSIG$-Algebra ist eine SIG-Algebra. Daher sind Variablen, Terme, Belegungen und Auswertung auch für Mengensignaturen definiert.

Wir definieren nun algebraische Netze. Wir betrachten ein algebraisches Netz gleich zusammen mit einer Anfangsmarkierung und sprechen daher von einem *initialisier-*

[8] In diesem Punkt weichen wir von [50] ab, wo Multimengensignaturen statt Mengensignaturen verwendet werden. Wir tun dies, um bei der Entfaltung immer ein Systemnetz (ohne Kantengewichtung) zu erhalten.

ten algebraischen Netz. Ein initialisiertes algebraisches Netz über einer Mengensignatur $MSIG$ besteht aus einem endlichen Systemnetz und einer $MSIG$-Algebra sowie einer Beschriftung der Kanten, Transitionen und Stellen des Systemnetzes mit Termen der $MSIG$-Algebra. Die Beschriftung der Stellen dient dabei zur Spezifikation einer Anfangsmarkierung. Die Beschriftung einer Transition stellt eine zusätzliche Aktivierungsbedingung für diese Transition dar (*transition guard*).

Definition 1.24 (Initialisiertes algebraisches Netz)

Sei $MSIG = (S, OP, ms)$ eine Mengensignatur mit Grundsorten GS. Ein *initialisiertes algebraisches Netz* $\dot{N} = (N, \mathcal{A}, X, i)$ über $MSIG$ besteht aus

1. einem endlichen Systemnetz $N = (P, T; F)$ wobei P über GS sortiert ist, d.h. $P = (P_s)_{s \in GS}$ – jeder Stelle ist eine *Sorte* zugeordnet,
2. einer $MSIG$-Algebra $\mathcal{A}$,
3. einer sortierten $MSIG$-Variablenmenge X,
4. einer *Netzbeschriftung* $i : P \cup T \cup F \to \mathfrak{T}^{MSIG}(X)$, so daß
   a) für alle $p \in P_s : i(p) \in \mathfrak{T}^{MSIG}_{ms(s)}$,
   b) für alle $t \in T : i(t) \in \mathfrak{T}^{MSIG}_{bool}(X)$, und
   c) für alle $t \in T$, und für alle $p \in P_s$ mit $f = (t,p) \in F$ oder $f = (p,t) \in F$ gilt $i(f) \in \mathfrak{T}^{MSIG}_{ms(s)}(X)$.

Für ein Stelle $p \in P$ heißt die Beschriftung $i(p)$ *symbolische Anfangsmarkierung* von p; für eine Transition $t \in T$ heißt der Term $i(t)$ *Aktivierungbedingung* von t. ∘

Jedes initialisierte algebraische Netz beschreibt genau ein (ggf. unendliches) initialisiertes (herkömmliches) Netz, das wir die *Entfaltung* des algebraischen Netzes nennen. Über die Entfaltung kann man für algebraische Netze das Schalten, Schaltsequenzen, Abwicklungen und Abläufe erklären. All diese Begriffe können auch direkt für algebraische Netze definiert werden, was bis auf Isomorphie identische Begriffe ergibt (siehe [50]). Wir stellen die Schaltregel für algebraische Netze direkt dar und verzichten auf die direkte Darstellung der anderen Begriffe. Im Abschnitt 1.5.3 definieren wir dann die Entfaltung eines algebraischen Netzes.

Wir definieren nun den Begriff der Markierung eines algebraischen Netzes. Eine Markierung eines algebraischen Netzes weist jeder Stelle eine endliche Multimenge über der Sorte dieser Stelle zu.

Definition 1.25 (Markierung eines algebraischen Netzes)

Sei $\dot{N}$ ein initialisiertes algebraisches Netz wie in Definition 1.24. Eine *Markierung* von $\dot{N}$ ist eine Abbildung $M : P \to \mathfrak{M}(A)$ mit $p \in P_s \Rightarrow M(p) \in \mathfrak{M}(A_s)$. Die

Markierung M^0 mit $M^0(p) = \overline{\beta}_{\varnothing}(i(p))$ für jedes $p \in P$ heißt *Anfangsmarkierung von* $\dot{N}$. Auf den Markierungen eines algebraischen Netzes definieren wir wie auf Multimengen Addition, Inklusion und Differenz elementweise. Eine Markierung M ist *sicher*, falls für alle $p \in P$ uns alle $a \in A$ gilt: $M(p)[a] \leq 1$. ∘

Nach Definition 1.24 ist die Anfangsmarkierung eines algebraischen Netzes sicher, da Stellen mit mengenwertigen Termen beschriftet werden.

Transitionen eines algebraischen Netzes schalten in *Modi*. Ein *Modus* ist eine Belegung der Variablen X des algebraischen Netzes. Für das Schalten einer Transition t ist nur die Belegung der Variablen von Bedeutung, die in den Beschriftungen der Kanten vorkommen, die entweder zur dieser Transition hin oder von dieser Transition weg führen. Eine Transition t ist in einem Modus β in einer Markierung *aktiviert*, falls alle Marken, die durch die Kanten zu t spezifiziert sind, in der Markierung vorliegen und falls die Auswertung der Aktivierungsbedingung von t true ergibt. Die Schaltregel algebraischer Netze formalisieren wir mit Hilfe der Markierungen t^-_β und t^+_β. Die Markierung t^-_β enthält die Marken, die beim Schalten von t im Modus β entfernt werden, die Markierung t^+_β enthält die Marken, die beim Schalten von t im Modus β hinzugefügt werden.

Definition 1.26 (Schalten eines algebraischen Netzes)

Sei $\dot{N}$ ein algebraisches Netz wie in Definition 1.24.

(a) Eine Belegung β von X heißt *Modus* von $\dot{N}$. Ein Paar (t, β) aus einer Transition und einem Modus von $\dot{N}$ heißt *Aktion* von $\dot{N}$. Wir schreiben im folgenden $t.\beta$ anstelle von (t, β).

(b) Zu jeder Aktion $t.\beta$ definieren wir die *Vormarkierung* t^-_β und die *Nachmarkierung* t^+_β durch

$$t^-_\beta(p) = \begin{cases} \overline{\beta}(i(p,t)) & \text{falls } (p,t) \in F \\ \varnothing & \text{sonst} \end{cases} \quad \text{und } t^+_\beta(p) = \begin{cases} \overline{\beta}(i(t,p)) & \text{falls } (t,p) \in F \\ \varnothing & \text{sonst.} \end{cases}$$

(c) Für eine Aktion $t.\beta$ ist die *Schaltrelation* $\xrightarrow{t.\beta}$ auf den Markierungen von $\dot{N}$ definiert durch

$$M_1 \xrightarrow{t.\beta} M_2 \text{ gdw. } \overline{\beta}(i(t)) = \text{true sowie } M_1 \geq t^-_\beta \text{ und } M_2 = (M_1 - t^-_\beta) + t^+_\beta$$

Gilt $\overline{\beta}(i(t)) = \text{true}$ und $M_1 \geq t^-_\beta$, so sagen wir $t.\beta$ *kann in* M_1 *schalten* (oder $t.\beta$ *ist in* M_1 *aktiviert*).

Bemerkung 1.27

Aus Gründen, die im nächsten Abschnitt ersichtlich werden, wollen wir im weiteren nur solche algebraischen Netze $\dot{N}$ betrachten, bei denen für jede Aktion $t.\beta$ von $\dot{N}$ die Markierungen t^-_β und t^+_β nicht-leer sind.

1.5.3 Entfaltung eines algebraischen Netzes

Wir definieren jetzt die Entfaltung eines algebraischen Netzes. Eine Stelle der Entfaltung ist ein Paar (p, a) (im folgenden notieren wir solche Paare durch $p.a$), wobei p eine Stelle des algebraischen Netzes ist und a ein Element der Algebra ist, so daß die Sorte von a mit der Sorte von p übereinstimmt. Eine Transition der Entfaltung ist eine Aktion $t.\beta$ des algebraischen Netzes, so daß $\overline{\beta}(i(t)) = \text{true}$ ist. Wir entfalten auch eine Markierung eines algebraischen Netzes zu einer Markierung eines Systemnetzes.

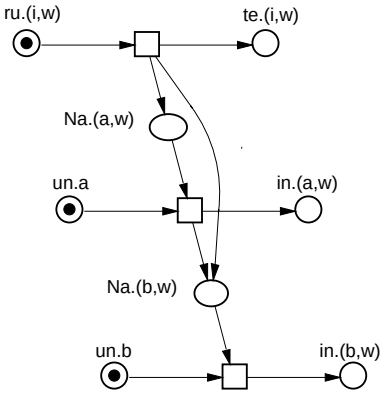
Definition 1.28 (Entfaltung)

Sei $\dot{N} = (N, \mathcal{A}, X, i)$ ein algebraisches Netz über $MSIG = (S, OP, ms)$ mit $N = (P, T; F)$ und Grundsorten GS. Wir definieren

1. $\dot{P} = \{p.a \mid s \in GS, p \in P_s, a \in A_s\}$
2. $\dot{T} = \{t.\beta \mid t.\beta \text{ ist eine Aktion von } \dot{N} \text{ mit } \beta(i(t)) = \text{true}\}$
3. $(p.a, t.\beta) \in \dot{F} \Leftrightarrow \overline{\beta}(i(p,t))[a] \neq 0$ und $(t.\beta, p.a) \in F \Leftrightarrow \overline{\beta}(i(t,p))[a] \neq 0$
4. Ist M eine Markierung von $\dot{N}$, so ist die *entfaltete Markierung* $\dot{M} \in \mathfrak{M}(\dot{P})$ definiert durch $\dot{M}[p.a] = M(p)[a]$.

Dann heißt das initialisierte Netz $((\dot{P}, \dot{T}; \dot{F}), \dot{M}^0)$ *Entfaltung* von $\dot{N}$. ∘

Wegen Bemerkung 1.27 ist die Entfaltung stellenberandet. Da Vor- und Nachmarkierung immer endlich sind, ist die Entfaltung endlich T-verzweigt. Also ist jede Entfaltung tatsächlich ein Systemnetz. Für die Entfaltung gilt: $\dot{M}_1 \xrightarrow{t.\beta} \dot{M}_2 \Leftrightarrow M_1 \xrightarrow{t.\beta} M_2$ gilt, d.h. das Schalten im algebraischen Netz und in seiner Entfaltung ist äquivalent.



ruhig, terminiert	$\{(i, w)\}$
unwissend	$A \setminus \{i\}$
Nachricht, informiert	$(A \setminus \{i\}) \times \{w\}$

Abb. 1.13: Σ_3 – Entfaltung von Σ_2.

Abb. 1.13 zeigt die Entfaltung von Σ_2 aus Abb. 1.12 auf Seite 28. Die Stellennamen von Σ_2 haben wir auf die ersten beiden Buchstaben abgekürzt. Neben dem Netz sind die Sorten der Stellen von Σ_2 angegeben.

Zum Schluß dieses Abschnittes wollen wir noch bemerken, daß jedes endliche initialisierte Netz als algebraisches Netz aufgefaßt werden kann, dessen Algebra nur die Trägermenge $\{\bullet\}$ enthält.

1.6 Wahrscheinlichkeitsräume

In diesem Abschnitt definieren wir einige Grundbegriffe aus der Wahrscheinlichkeitsrechnung. Für eine Motivation der Begriffe verweisen wir z.B. auf [13].

Sei Ω eine nicht-leere Menge. Eine σ-*Algebra*[9] über Ω ist ein Mengensystem $\mathcal{A} \subseteq 2^\Omega$, das gegenüber Komplement- und abzählbarer Vereinigungsbildung abgeschlossen ist und für das $\Omega \in \mathcal{A}$ gilt. Die Mengensysteme $\{\varnothing, \Omega\}$ und 2^Ω sind σ-Algebren über Ω.

Sei $\mathcal{A}$ eine σ-Algebra über Ω. Eine Abbildung $P : \mathcal{A} \to [0,1]$ heißt *Wahrscheinlichkeitsmaß* auf $\mathcal{A}$, falls $P(\Omega) = 1$ und falls für jede Familie paarweise disjunkter Mengen $(A_i)_{i\in\mathbb{N}}$ gilt:

$$P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i).$$

Das Tripel $(\Omega, \mathcal{A}, P)$ heißt *Wahrscheinlichkeitsraum*, falls $\mathcal{A}$ eine σ-Algebra über Ω und P ein Wahrscheinlichkeitsmaß über $\mathcal{A}$ ist. Sei $(\Omega, \mathcal{A}, P)$ ein Wahrscheinlichkeitsraum. Eine Menge $A \subseteq \Omega$ heißt in $(\Omega, \mathcal{A}, P)$ *meßbar*[10], falls $A \in \mathcal{A}$.

Sind $\mathcal{A}_1$ und $\mathcal{A}_2$ σ-Algebren, so ist auch $\mathcal{A}_1 \cap \mathcal{A}_2$ eine σ-Algebra. Auch der Durchschnitt beliebig vieler σ-Algebren ist wieder eine σ-Algebra und so ist für eine Menge $\mathcal{E} \subseteq 2^\Omega$ von Teilmengen von Ω die von $\mathcal{E}$ *erzeugte* σ-*Algebra* $\sigma(\mathcal{E})$ durch

$$\sigma(\mathcal{E}) = \bigcap \{\mathcal{A} \mid \mathcal{A} \text{ ist } \sigma\text{-Algebra über } \Omega \text{ und } \mathcal{E} \subseteq \mathcal{A}\}$$

definiert. $\mathcal{E}$ heißt dabei *Erzeuger* von $\sigma(\mathcal{E})$. Zwei meßbare Mengen A_1, A_2 heißen *stochastisch unabhängig* in $(\Omega, \mathcal{A}, P)$, falls $P(A_1 \cap A_2) = P(A_1) \cdot P(A_2)$.

[9] nicht zu verwechseln mit *SIG*-Algebra

[10] Wir vezichten hier auf den Gebrauch des Wortes *Ereignis* (im stochastischen Sinne), um Verwechslungen mit Ereignissen von Abwicklungen zu vermeiden.

Teil I

Fairneß und Randomisierung

2 Netzsysteme

In diesem Kapitel definieren wir unser erstes Systemmodell – *Netzsysteme*. Ein Netzsystem ist ein um eine *Progreßannahme* erweitertes initialisiertes Netz. Eine Progreßannahme ist eine spezielle *Lebendigkeitsannahme*. Progreß definieren wir in Abschnitt 2.1.1, den Begriff der Lebendigkeitsannahme definieren wir in Abschnitt 2.2. Desweiteren führen wir in diesem Kapitel das Problem des wechselseitigen Ausschlusses (Mutex-Problem) sowie das Konsens-Problem ein und untersuchen ihre Lösbarkeit in Netzsystemen.

2.1 Netzsysteme

In diesem Abschnitt definieren wir Progreß, Netzsysteme und zeigen die Beziehung von Progreß zum Begriff der *schwachen Fairneß* auf, der in der Literatur häufig verwendet wird.

2.1.1 Progreß

Geschieht es jetzt, so geschieht es nicht in Zukunft; geschieht es nicht in Zukunft, so geschieht es jetzt; geschieht es jetzt nicht, so geschieht es doch einmal in Zukunft.
– Hamlet

Ein Netz beschreibt zunächst nur *mögliches* Schalten von Transitionen und damit *mögliches* Verhalten eines Systems. Um auch *notwendiges* Verhalten zu beschreiben, muß man zusätzlich zu einem Netz eine *Lebendigkeitsannahme* angeben. Eine Lebendigkeitsannahme sondert einige Abläufe eines Netzes als *nicht lebendig* aus. Abb. 2.1 zeigt ein initialisiertes Netz Σ_4, das zwei unabhängige, nichtkommunizierende Agenten modelliert, von denen jeder Agent jeweils zwei Zustände zyklisch durchläuft. Nach Definition 1.11 hat Σ_4 einen Ablauf in dem keine Transition schaltet; beispielsweise diesen Ablauf wollen wir durch eine Lebendigkeitsannahme aussondern.

Die schwächste Lebendigkeitsannahme, die in der Literatur verwendet wird, ist die Annahme von der *Maximalität der Schaltsequenzen*. Eine Schaltsequenz ist *maximal*, falls sie entweder unendlich ist oder falls sie endlich ist und im Endzustand keine Transition aktiviert ist. Die Schaltsequenz $AC \stackrel{a}{\rightarrow} BC$ von Σ_4 ist demnach nicht maximal. Bei sequentiellen Algorithmen ist die Forderung der Maximalität

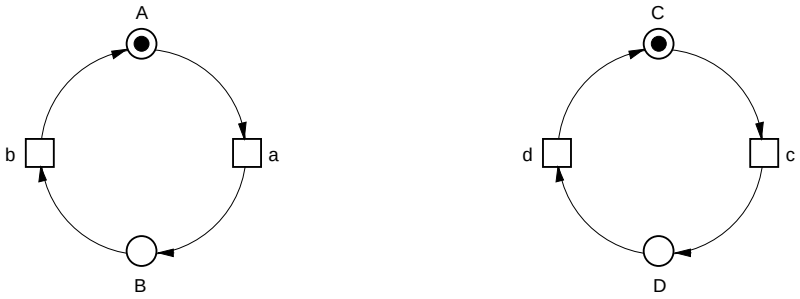


Abb. 2.1: Σ_4 : Zwei unabhängige zyklische Agenten.

von Schaltsequenzen selbstverständlich. Dort bedeutet diese Annahme, daß der einzige Prozessor (Agent) nicht unerwartet stehenbleibt und die nächste Anweisung nicht mehr abarbeitet.

Wollen wir die Annahme, daß kein Agent unerwartet stehenbleibt für mehrere Agenten, also für einen verteilten Algorithmus, ausdrücken, so genügt die Annahme der Maximalität von Schaltsequenzen nicht. Dies wird bei Betrachtung von Σ_4 deutlich: Die unendliche Schaltsequenz $AC(\stackrel{a}{\rightarrow} BC \stackrel{b}{\rightarrow} AC)^\infty$ ist maximal – der rechte Agent schaltet jedoch überhaupt nicht in dieser Schaltsequenz.

Die Annahme, daß kein Agent unerwartet stehenbleibt, wird durch *Maximalität nicht-sequentieller Abläufe* ausgedrückt. Ein Ablauf ρ eines intialisierten Netzes Σ ist *maximal*, falls kein Ablauf ρ' von Σ existiert mit $\rho \sqsubset \rho'$. Σ_4 hat genau einen maximalen Ablauf. Maximalität des Ablaufes bedeutet für jede Transition t des Netzes: Ist t aktiviert, so schaltet t irgendwann oder irgendwann schaltet eine zu t in Konflikt stehende Transition. Maximalität eines Ablaufes läßt sich also leicht bezüglich einzelner Transitionen formulieren: Ein Ablauf ρ eines intialisierten Netzes Σ ist *maximal bzgl.* t, falls kein Ablauf ρ' von Σ existiert mit $\rho \stackrel{e}{\sqsubset} \rho'$ und $\widetilde{e} = \mathrm{t}$. Die Maximalität eines Ablaufs bezüglich einer Transition t bezeichnen wir auch als *Progreß von* t.

2.1.2 Netzsysteme

Manchmal möchten wir Progreß nicht von allen Transitionen eine Netzes fordern. Dies ist insbesondere dann der Fall, wenn wir Umgebungsverhalten modellieren. Modelliert ein Netz zum Beispiel einen Süßigkeitenautomaten und modelliert eine Transition t des Netzes den Geldeinwurf in den Automaten, so ist es plausibel, von t keinen Progreß zu fordern, da wir nicht wissen, ob jemals ein Geldstück in den Automaten eingeworfen wird. (Insbesondere hängt die Korrektheit des Automaten nicht davon ab, ob jemals ein Geldstück eingeworfen wird.) Eine Transition von der wir keinen Progreß verlangen, bezeichnen wir als *externe Transition.*

Definition 2.1 (Netzsystem)

Ein *Netzsystem* $\Sigma = (\Sigma, \mathsf{T}^{\mathrm{ext}})$ besteht aus einem initialisierten Netz Σ mit Transitionsmenge T und einer Menge ausgezeichneter Transitionen $\mathsf{T}^{\mathrm{ext}} \subseteq \mathsf{T}$. Ein Element von $\mathsf{T}^{\mathrm{ext}}$ heißt *externe Transition* von Σ, ein Element von $\mathsf{T} \setminus \mathsf{T}^{\mathrm{ext}}$ heißt *interne Transition* von Σ. ○

Eine externe Transition stellen wir graphisch grau schraffiert dar (vgl. Abb. 2.2). Im weiteren wollen wir nur solche Abläufe des dem Netzsystem zugrundeliegenden initialisierten Netzes betrachten, die den Progreß aller internen Transitionen des Netzsystems erfüllen. Einen solchen Ablauf nennen wir *progressiv*. Allgemeiner definieren wir nun Progreß als Eigenschaft von Abwicklungen.

Definition 2.2 (Progressive Abwicklung)

Sei Σ ein Netzsystem und t eine Transition von Σ. Eine Abwicklung π von Σ ist *progressiv bzgl.* t, falls keine Abwicklung π' von Σ existiert mit $\pi \overset{e}{\sqsubset} \pi'$ und $\widetilde{e} = \mathrm{t}$; π ist *progressiv*, falls π progressiv bzgl. jeder internen Transition von Σ ist. ○

Abb. 2.2 zeigt das Netzsystem Σ_5. Es stellt zwei zyklische Agenten dar, die durch einen asynchronen Nachrichtenkanal miteinander verbunden sind. Der linke Agent erzeugt Nachrichten (auf E), der rechte Agent verbraucht diese und sendet eine Antwort (auf F), sobald er bereit ist, eine neue Nachricht zu verbrauchen. Der linke Agent wartet auf diese Antwort um in seinen Anfangszustand A zurückzugehen.

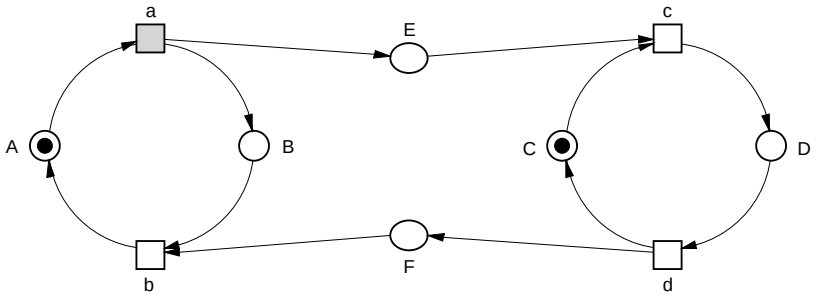


Abb. 2.2: Σ_5 : Ein Erzeuger/Verbraucher-System.

Transition a in Σ_5 ist extern. Demnach kann der linke Agent im Zustand A eine Nachricht erzeugen, muß dies aber nicht tun. Der Ablauf von Σ_5, in dem keine Transition schaltet, ist progressiv. Der Ablauf von Σ_5, in dem nur Transition a einmal schaltet, ist nicht progressiv. Σ_5 hat unendlich viele progressive Abläufe aber nur einen maximalen Ablauf. Dieses Phänomen nennen wir *externen Nichtdeterminismus*.

2.1.3 Progreß und schwache Fairneß

Wir haben Progreß auf nicht-sequentiellen Abläufe definiert. In sicheren Netzen kann man Progreß äquivalent auf Schaltsequenzen definieren (vgl. [81]). Für Schaltsequenzen gibt es den Begriff der *schwachen Fairneß* (*weak fairness*, in [58]: *justice*), der Progreß ähnlich ist. Eine Schaltsequenz σ ist *schwach fair* bezüglich einer Transition t, falls gilt: Ist t ab irgendeinem Zeitpunkt in σ in jeder Markierung aktiviert, dann schaltet t irgendwann nach diesem Zeitpunkt.

Definition 2.3 (Schwache Fairneß)

Sei Σ ein initialisiertes Netz und t eine Transition von Σ. Eine Schaltsequenz σ von Σ ist nicht *schwach fair bzgl.* t, falls es ein Suffix σ' von σ gibt, so daß t in jeder Markierung von σ' aktiviert ist und t nicht in σ' vorkommt. ○

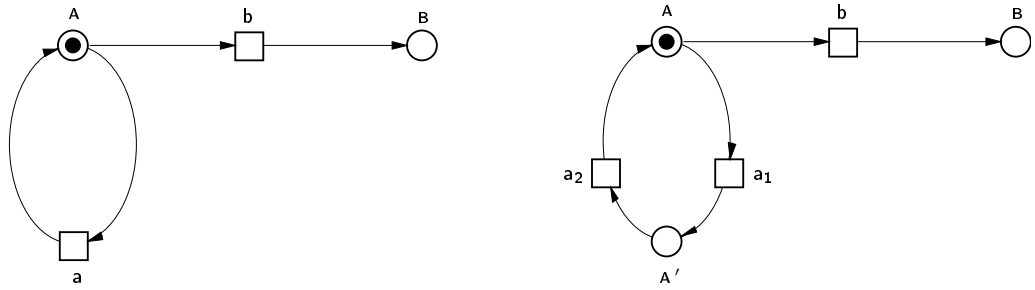
Schwache Fairneß sondert in sicheren Netzen mindestens genauso viele Abläufe aus wie Progreß:

Proposition 2.4

Sei Σ ein sicheres initialisiertes Netz und t eine Transition von Σ. Ist ein Ablauf ρ von Σ nicht progressiv bzgl. t, dann ist jede Schaltsequenz von ρ nicht schwach fair bzgl. t.

Die Umkehrung von Proposition 2.4 gilt im allgemeinen nicht. Das heißt: Es gibt Systeme, in denen schwache Fairneß mehr Abläufe aussondert als Progreß. Dazu betrachten wir Σ_6 in Abb. 2.3(a). Die Schaltsequenz $\sigma = A(\xrightarrow{a} A)^\infty$ ist Schaltsequenz eines bzgl. b progressiven Ablaufs, σ ist jedoch nicht schwach fair bzgl. b. Dieses Netz illustriert das folgende intuitive Problem bei schwacher Fairneß. Schwache Fairneß (bzgl. t) wird oft analog zu Progreß beschrieben: Ist t *kontinuierlich* aktiviert, so schaltet t irgendwann. Doch ist b in σ kontinuierlich aktiviert? Intuitiv nein, da wir uns ja vorstellen, daß Transition a die Marke auf A beim Schalten zuerst verbraucht und dann wieder erzeugt. In einem virtuellen Zwischenzustand, während des Schaltens von a, ist keine Marke auf A und Transition b damit nicht aktiviert.

Dieses intuitive Problem formalisieren wir mittels Verfeinerung. In Σ_7 in Abb. 2.3(b) wurde Transition a durch die sequentielle Ausführung zweier Transitionen a_1 und a_2 ersetzt. Die σ entsprechende Schaltsequenz in Σ_7 ist $\sigma' = A(\xrightarrow{a_1} A' \xrightarrow{a_2} A)^\infty$; σ' ist schwach fair bzgl. b. Ein nicht schwach fairer Ablauf kann also durch Verfeinerung zu einem schwach fairen Ablauf werden. Wir sagen auch: Schwache Fairneß ist nicht *robust* gegenüber Verfeinerung. Die Nicht-Robustheit gegenüber Verfeinerung tritt auch in anderen Zusammenhängen bei Schaltsequenzen auf (siehe [20] und weiterführend [38]). Wir kommen später auf dieses Phänomen zurück.



(a) Σ_6 (b) Σ_7: Verfeinerung von Σ_6

Abb. 2.3: Problem bei schwacher Fairneß.

Der beschriebene Unterschied zwischen Progreß und schwacher Fairneß wird in Σ_6 durch eine *Schleife* hervorgerufen. Eine *Schleife* ist ein Paar (p,t) aus einer Stelle p und einer Transition t, so daß $(p,t) \in F$ und $(t,p) \in F$. Die Schleife (A,a) in Σ_6 sorgt dafür, daß b nach dem Schalten von a sofort wieder aktiviert ist. In sicheren Netzen ohne Schleifen fallen Progreß und schwache Fairneß zusammen. Die natürliche Annahme vom Progreß einer Transition kann also durch schwache Fairneß ersetzt werden, falls man sich auf sichere Netze ohne Schleifen einschränkt[1].

[1] In nicht notwendig sicheren Netzen sind Progreß und schwache Fairneß unvergleichbar.

2.2 Lebendigkeit

Wir haben bisher verschiedene Beispiele von Lebendigkeitsannahmen diskutiert – Maximalität von Schaltsequenzen, Progreß und schwache Fairneß. In diesem Abschnitt wollen wir nun definieren, was wir allgemein unter einer Lebendigkeitsannahme verstehen. Eine Lebendigkeitsannahme beziehen wir auf ein gegebenes initialisiertes Netz Σ, d.h. wir sprechen von einer *Lebendigkeitsannahme für* Σ. Eine Lebendigkeitsannahme für Σ ist eine Lebendigkeitseigenschaft L über der Stellenmenge von Σ, so daß Σ jederzeit die Möglichkeit hat, so fortzusetzen, daß L erfüllt wird[2].

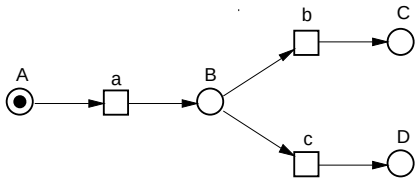


Abb. 2.4: Σ_8

Betrachten wir als Beispiel Σ_8 in Abb. 2.4. Lebendigkeitseigenschaften über der Stellenmenge von Σ_8 sind z.B. $\Diamond B, \Box \Diamond B$ und $\Diamond C$. Von diesen Lebendigkeitseigenschaften wollen wir nur $\Diamond B$ als Lebendigkeitsannahme für Σ_8 akzeptieren. Schalten nämlich a und c, so kann das System nicht mehr so fortsetzen, daß $\Box \Diamond B$ oder $\Diamond C$ erfüllt werden. Progreß entspricht in Σ_8 der Lebendigkeitseigenschaft $\Diamond(C \vee D)$.

Wir formalisieren dies nun und definieren zuerst, wann ein Ablauf eines Netzes bzgl. einer Ablaufeigenschaft *lebendig* ist. Ein Ablauf eines initialisierten Netzes Σ ist bzgl. einer Ablaufeigenschaft E *lebendig*, falls es in dem Ablauf zu jedem Zeitpunkt möglich ist, durch Σ so fortzusetzen, daß E erfüllt wird.

Definition 2.5 (Lebendigkeit)

Sei Σ ein initialisiertes Netz mit Stellenmenge P und sei E eine Ablaufeigenschaft über P. Ein endlicher Ablauf α von Σ ist *lebendig* bzgl. E, falls Σ einen Ablauf $\rho \sqsupseteq \alpha$ besitzt, der α fortsetzt und E erfüllt. Ein Ablauf ρ von Σ ist *lebendig* bzgl. E, falls jeder endliche Präfix von ρ bzgl. E lebendig ist. Eine Ablaufeigenschaft E' über P ist *lebendig* bzgl. E, falls jeder Ablauf aus E' bzgl. E lebendig ist. ○

Proposition 2.6

Sei P eine abzählbare Menge, E eine Ablaufeigenschaft über P und ρ ein Ablauf über P. Dann gilt:

(a) Erfüllt ρ die Eigenschaft E, so ist ρ lebendig bzgl. E.

[2] In der Literatur wird dies oft durch die Aussage „Das System kann sich nicht in die Ecke streichen." illustriert, mit der Vorstellung, daß der Fußboden eines Zimmers gestrichen wird.

(b) Ist E eine Sicherheitseigenschaft, so gilt: ρ erfüllt E gdw. ρ lebendig bzgl. E ist.

Beweis: (a) ist trivial, (b) folgt direkt aus Definition 1.17. □

Die Lebendigkeit einer Eigenschaft gegenüber einer anderen Eigenschaft wird in der Literatur auch als *Maschinenabgeschlossenheit* (auch: *feasibility*) bezeichnet. Wir definieren nun den Begriff der Lebendigkeitsannahme.

Definition 2.7 (Lebendigkeitsannahme)

Sei Σ ein initialisiertes Netz mit Stellenmenge P. Eine *Lebendigkeitsannahme für* Σ ist ein Lebendigkeitseigenschaft L über P, so daß jeder Ablauf von Σ bzgl. L lebendig ist. Seien L_1, L_2 zwei Lebendigkeitsannahmen für Σ. Wir sagen L_2 ist *mindestens so stark* wie L_1, falls $\mathfrak{R}(\Sigma) \cap L_2 \subseteq \mathfrak{R}(\Sigma) \cap L_1$; gilt $\mathfrak{R}(\Sigma) \cap L_2 \neq \mathfrak{R}(\Sigma) \cap L_1$, so sagen wir: L_2 ist *stärker* als L_1. ○

Ist L eine Lebendigkeitsannahme für Σ, so heißt ein Ablauf aus $\mathfrak{R}(\Sigma) \cap L$ auch *unter* L *zulässig*. Wir sagen, daß eine Ablaufeigenschaft E in Σ *unter* L *gültig* ist, falls jeder unter L zulässige Ablauf von Σ die Eigenschaft E erfüllt. Die Menge der progressiven Abläufe eines Netzsystems Σ ist eine Lebendigkeitsannahme für Σ. Im vorigen Abschnitt 2.1.3 haben wir gesehen, daß schwache Fairneß in manchen Netzen stärker als Progreß ist.

Apt, Francez und Katz postulieren in [8], daß jede Lebendigkeitsannahme[3] neben der Maschinenabgeschlossenheit noch *äquivalenzrobust* sein soll. Dabei betrachten sie Schaltsequenzen. Zwei Schaltsequenzen werden in [8] als äquivalent betrachtet, falls sie zum selben nicht-sequentiellen Ablauf gehören, d.h. zwei äquivalente Schaltsequenzen gehen durch Umordnen unabhängiger Ereignisse auseinander hervor. Eine Schaltsequenzeigenschaft S ist *äquivalenzrobust*, falls gilt: Verletzt eine Schaltsequenz σ die Eigenschaft S, dann verletzt jede zu σ äquivalente Schaltsequenz auch S. Schwache Fairneß ist nur in sicheren Netzen äquivalenzrobust, starke Fairneß (siehe Abschnitt 3.1.1) ist nicht äquivalenzrobust.

Da wir Eigenschaften nicht-sequentieller Abläufe betrachten, ist eine Lebendigkeitsannahme in unserem Sinne trivialerweise äquivalenzrobust. Lamport vertritt in [57] die Meinung, daß von den Kriterien für die Vernünftigkeit einer Lebendigkeitsannahme von Apt, Francez und Katz nur die Maschinenabgeschlossenheit relevant ist.

[3] in [8]: Fairneßannahme

2.3 Mutex in Netzsystemen

Wir beschäftigen uns nun mit der Lösbarkeit verschiedener Probleme in Netzsystemen. In Netzsystemen kann man viele Probleme lösen, u.a. Verbreitung einer Information mit Feedback, Feststellung von verteilter Terminierung, Berechnung von kürzesten Wegen im Netzwerk und Snapshot verteilter Zustände (vgl. [81, 91]). Wir zeigen in diesem Abschnitt, daß das Mutex-Problem in Netzsystemen nicht lösbar ist. Damit bereiten wir spätere Resultate vor. Die Unmöglichkeit von Mutex in Netzsystemen wurde bereits von Kindler und Walter bewiesen [52].

Wir stellen das Mutex-Problem zunächst in 2.3.1 informell vor, formalisieren es dann in 2.3.2 und zeigen schließlich in 2.3.3 die Unmöglichkeit.

2.3.1 Das Problem des wechselseitigen Ausschlusses

Das *Problem des wechselseitigen Ausschlusses* (Mutex-Problem) ist wohl das bekannteste Synchronisationsproblem in verteilten Systemen. Es besitzt zentrale Bedeutung für die Konstruktion verteilter Systeme und ist seit seiner Einführung durch Dijkstra 1965 [29] in zahlreichen Publikationen untersucht worden.

Beim Mutex-Problem sind in seiner einfachsten Form zwei Agenten gegeben, die sich jeweils in einem der drei Zustände *ruhig*, *hungrig* oder *kritisch* befinden. Ein ruhiger Agent kann hungrig werden, ein hungriger Agent kritisch und ein kritischer Agent wird irgendwann wieder ruhig. Unter diesen Voraussetzungen hat ein Algorithmus *Mutex-Verhalten*, falls

1. beide Agenten nie zugleich kritisch sind (wechselseitiger Ausschluß), und
2. jeder hungrige Agent irgendwann kritisch wird.

Zudem wollen wir einen Algorithmus nur dann als Mutex-Algorithmus akzeptieren, falls

3. ein ruhiger Agent vom Algorithmus weder gezwungen noch gehindert werden kann, hungrig zu werden.

Wir können uns damit vorstellen, daß die Entscheidung, ob und wann ein ruhiger Agent hungrig wird, extern durch die Umgebung des Algorithmus festgelegt wird. Insbesondere ist es möglich, daß ein ruhiger Agent nie wieder hungrig wird.

Im Netzsystem Σ_9 in Abb. 2.5 sind fast alle Anforderungen an einen Mutex-Algorithmus erfüllt. Die Transitionen a_l und a_r sind extern – ein ruhiger Agent kann hungrig werden, muß dies aber nicht. Der wechselseitige Ausschluß wird in Σ_9 durch einen

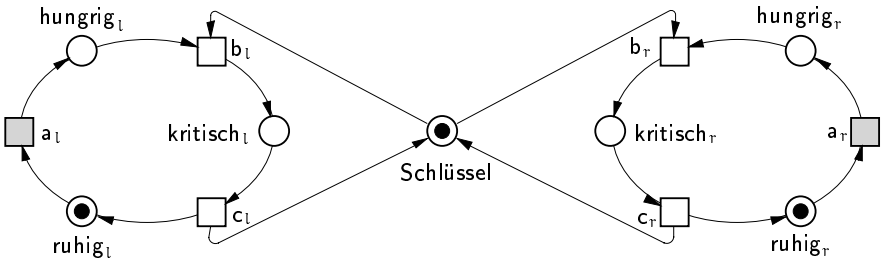


Abb. 2.5: Σ_9 – ein Netzsystem mit wechselseitigem Ausschluß.

zentralen Schlüssel organisiert: Um kritisch zu werden, benötigt ein hungriger Agent den Schlüssel, wird ein kritischer Agent wieder ruhig, so gibt er den Schlüssel zurück. Eigenschaft 2 ist in Σ_9 nicht erfüllt: Σ_9 hat einen progressiven Ablauf, in dem der linke Agent immer wieder kritisch wird, während der rechte Agent für immer hungrig bleibt.

2.3.2 Formalisierung von Mutex

Dieser Abschnitt dient zur Formalisierung der Anforderungen an einen Mutex-Algorithmus. Wir folgen dabei Kindler und Walter in [52] indem wir in zwei Schritten vorgehen. Ein Netzsystem stellt genau dann einen Mutex-Algorithmus dar, falls es sowohl *Mutex-Struktur* als auch *Mutex-Verhalten* besitzt. Der Begriff Mutex-Verhalten formalisiert dabei die temporallogischen Eigenschaften 1. und 2. aus dem vorigen Abschnitt. Mutex-Struktur formalisiert alle übrigen Nebenbedingungen. Wir beginnen mit Mutex-Verhalten:

Definition 2.8 (Mutex-Verhalten)

Ein Netzsystem Σ besitzt *Mutex-Verhalten*, falls jeder progressive Ablauf von Σ die folgenden beiden temporallogischen Eigenschaften (2.1) und (2.2) erfüllt.

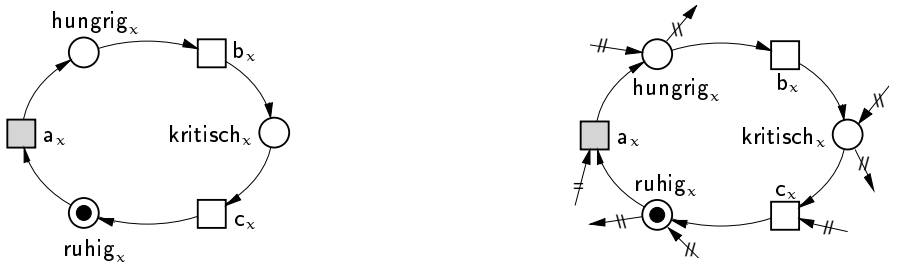
$$\Box\neg(\mathit{kritisch}_l \wedge \mathit{kritisch}_r) \tag{2.1}$$

$$\forall x \in \{l, r\} : \mathit{hungrig}_x \rhd \mathit{kritisch}_x \tag{2.2}$$

○

Wir definieren nun Mutex-Struktur, um z.B. solche Netzsysteme als Mutex-Lösung nicht zuzulassen, bei denen ein Agent gezwungen wird, hungrig zu werden. Ein Netzsystem Σ hat Mutex-Struktur, falls für jeden Agenten x ein Teilnetz Σ_x von Σ wie in Abb. 2.6(a) existiert, so daß die Verbindung von Σ_x zum Rest von Σ bestimmten Bedingungen genügt, die in Abb. 2.6(b) illustriert werden. Dabei darf der Rest von Σ mit Σ_x nur über Kanten von und zu Transitionen von Σ_x und nicht von und zu Stellen von Σ_x verbunden sein – mit der zusätzlichen Einschränkung, daß keine Kante zu a_x oder zu c_x führt. Damit kann ein ruhiger Agent nicht daran gehindert

werden, hungrig zu werden und jeder kritische Agent wird irgendwann ruhig. Da a_x extern ist, kann kein ruhiger Agent gezwungen werden, kritisch zu werden. Wir definieren Mutex-Struktur für eine beliebige endliche Menge von Agenten, da wir dies später für Varianten des Mutex-Problems benötigen.



(a) Σ_x

(b) Illustration von 2. und 3. in Definition 2.9

Abb. 2.6: Mutex-Struktur

Definition 2.9 (Mutex-Struktur)

Sei A eine endliche Menge von Agenten. Ein Netzsystem $\Sigma = (N, M_0, T^{ext})$ mit $N = (P, T; F)$ besitzt *Mutex-Struktur für* A, falls für jeden Agenten $x \in A$ genau ein Netzsystem $\Sigma_x = (N_x, M_x, T_x^{ext})$ mit $N_x = (P_x, T_x; F_x)$ wie in Abb. 2.6(a) existiert, so daß alle Σ_x paarweise disjunkt sind und für alle $x \in A$ gilt:

1. $P_x \subseteq P$, $T_x \subseteq T$, $F_x \subseteq F$, $T_x^{ext} \subseteq T^{ext}$
2. Für alle $(p,t) \in F$ mit $t \in \{a_x, c_x\}$ ist $(p,t) \in F_x$.
3. Für alle $(u,v) \in F$ mit $u \in P_x$ oder $v \in P_x$ ist $(u,v) \in F_x$.
4. $p \in P_x$ impliziert $M_0(p) = M_x(p)$. ○

2.3.3 Unmöglichkeit von Mutex in Netzsystemen

Netzsystem Σ_9 auf Seite 47 hat Mutex-Struktur jedoch kein Mutex-Verhalten. Verändern wir Σ_9, so daß der Schlüssel nicht Vorbedingung von b_x, sondern von a_x ist, für $x \in \{l, r\}$ (ein Agent benötigt dann den Schlüssel, um hungrig zu werden), so hat das entstehende Netzsystem Mutex-Verhalten, jedoch keine Mutex-Struktur. Wir zeigen nun die Unmöglichkeit von Mutex in Netzsystemen, welche bereits von Kindler und Walter 1997 in [52] bewiesen wurde. Wir nehmen ihren Beweis hier (leicht verändert) auf, um das Verständnis späterer Resultate zu erleichtern.

Satz 2.10 (Kindler und Walter [52])

Es gibt kein Netzsystem, das sowohl Mutex-Struktur für $\{l, r\}$ als auch Mutex-Verhalten hat.

Beweis: Wir führen einen indirekten Beweis. Sei Σ ein Netzsystem mit Mutex-Struktur und Mutex-Verhalten. Dann hat Σ einen progressiven Ablauf ρ_1, in dem r nie hungrig wird, l aber immer wieder – ρ_1 wird wie folgt konstruiert: Man beginnt mit dem ereignislosen Ablauf von Σ und läßt a_l schalten. Sei α der entstandene endliche Ablauf. Jetzt setzen wir mit Progreß fort, d.h. wir wählen eine minimale progressive Fortsetzung von α. Dann können wir wieder a_l schalten lassen usw. Durch unendliche Iteration erhalten wir den progressiven Ablauf ρ_1 (siehe Abb. 2.7).

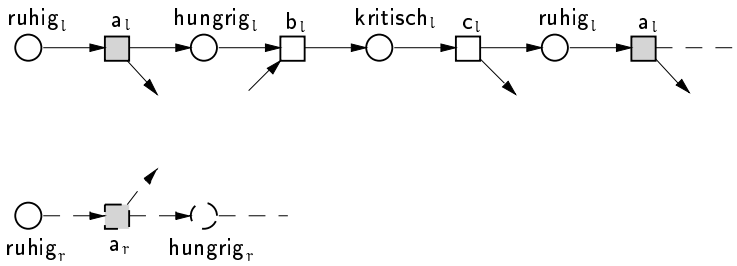


Abb. 2.7: ρ_1 und Fortsetzung ρ_2 (gestrichelt)

Es gibt einen progressiven Ablauf $\rho_2 \sqsupset \rho_1$, der ρ_1 fortsetzt, in dem r hungrig wird. (Wir lassen a_r im Ende von ρ_1 schalten und setzen mit Progreß fort.) In Abb. 2.7 ist ρ_2 gestrichelt dargestellt.

Da Σ Mutex-Verhalten besitzt, gilt $\rho_2 \models \Diamond\, \mathit{kritisch}_r$. Sei B_i die Menge der Bedingungen von ρ_i für $i = 1, 2$ und sei b eine Bedingung von $B_2 \setminus B_1$ mit $\widetilde{b} = \mathit{kritisch}_r$ und sei C ein Markierungsschnitt mit $b \in C$. In C gilt entweder ruhig_l oder $\mathit{hungrig}_l$ oder $\mathit{kritisch}_l$. In jedem Fall gilt wegen (2.2) dann $\rho_2, C \models \Diamond\, \mathit{kritisch}_l$.

Dann gibt es eine $\mathit{kritisch}_l$-Bedingung b', die von C erreichbar ist. Es gilt:

1. Es ist nicht $b = b'$, da $\widetilde{b} \neq \widetilde{b'}$.
2. Es ist nicht $b' < b$, da b' von C erreichbar ist.
3. Es ist nicht $b < b'$, da $\rho_1 \sqsubseteq \rho_2$ sowie $b \in B_2 \setminus B_1$ und $b' \in B_1$.
4. Es ist nicht b co b', da (2.1) in jedem Ablauf gültig ist.

Dies ist aber ein Widerspruch dazu, daß ρ_2 ein Ablauf ist. □

2.4 Konsens in Netzsystemen

Das Konsens-Problem gehört wie das Mutex-Problem zu den am häufigsten untersuchten Problemen im Bereich verteilter Algorithmen. Seit Anfang der Achtziger-Jahre ist eine nunmehr kaum noch zu überschauende Menge an Publikationen zum Konsens-Problem entstanden. Die Tatsache, daß heute immer noch zu diesem Problem publiziert wird, zeigt, daß immer noch neue Varianten und Facetten des Konsens-Problems zu entdecken sind.

Beim Konsens-Problem geht es für eine Menge verteilter Agenten darum, Einigkeit über eine vorliegende Fragestellung, im einfachsten Fall über einen binären Wert zu erlangen. Konsens zu erzielen bedeutet für die verteilten Agenten, eine Information miteinander zu teilen. Das Konsens-Problem ist in einer fehlerfreien Umgebung einfach zu lösen. Können Agenten jedoch ausfallen oder noch schwererwiegendes Fehlverhalten zeigen, so ist das Konsens-Problem nur noch schwer zu lösen.

Die Popularität des Konsens-Problems erklärt sich einerseits aus seiner Allgegenwärtigkeit beim Entwurf verteilter Systeme und andererseits aus seiner faszinierenden Komplexität. Die Lösung des Konsens-Problems ist der Kern vieler Algorithmen zur verteilten Verarbeitung von Daten, verteiltem Dateimanagement sowie fehlertoleranter Anwendungen. Eine Form des Konsens-Problems ist das *Transaction Commit-Problem* in verteilten Datenbanken. Überblicke über das Konsens-Problem und einige Konsensalgorithmen bieten Barborak, Malek und Dahbura [12], Turek und Shasha [89] sowie Fischer [35]. Barborak, Malek und Dahbura gehen in [12] besonders auf die Anwendung von Konsensalgorithmen in fehlertoleranten Anwendungen ein.

Die Interessantheit des Konsens-Problems ergibt sich aus einer Reihe von Unmöglichkeitsresultaten, beginnend mit dem gefeierten Resultat von Fischer, Lynch und Paterson, die 1983 zeigten, daß das Konsens-Problem bereits unter Möglichkeit von nur einem ausfallendem Agenten nicht lösbar ist, falls die Agenten deterministisch sind und falls Agenten und Kanäle asynchron sind [36]. Viele Unmöglichkeitsresultate stehen zu diesem Resultat in enger Beziehung, z.B. die Unmöglichkeit von ausfalltolerantem *Group Membership* [22]. Einen einführenden Überblick über die zentralen Unmöglichkeitsresultate Konsens-ähnlicher Probleme gibt Reischuk [76].

In diesem Abschnitt machen wir uns mit dem Konsens-Problem vertraut und zeigen, daß eine ausfalltolerante Lösung in Netzsystemen nicht möglich ist. Netzsysteme sind einerseits nicht so stark wie das Modell von Fischer, Lynch und Paterson (im folgenden: das *FLP-Modell*), da im FLP-Modell Mutex lösbar ist, in Netzsystemen hingegen nicht. Andererseits müssen wir für die Unmöglichkeit in Netzsystemen weniger technische Annahmen als im FLP-Modell treffen (vgl. Abschnitt 3.3.1), was zu einem einfacheren und transparenteren Beweis führt, der die wesentliche Schwierigkeit bei der Lösung des Konsens-Problems zeigt.

Wir verfahren wie folgt. Zunächst beschreiben wir das Konsens-Problem informell und geben dann in Abschnitt 2.4.2 eine Formalisierung des Konsens-Problems an. In Abschnitt 2.4.3 illustrieren wir die Schwierigkeit, das Konsens-Problem ausfalltolerant zu lösen anhand eines kleinen, nicht publizierten Konsensalgorithmus. In Abschnitt 2.4.4 zeigen wir schließlich die Unmöglichkeit von Konsens in Netzsystemen.

2.4.1 Das ausfalltolerante Konsens-Problem

In diesem Abschnitt beschreiben wir das ausfalltolerante Konsens-Problem informell.

Das ausfalltolerante Konsens-Problem (im folgenden kurz: Konsens-Problem) ist im Gegensatz zum Mutex-Problem ein Terminationsproblem, d.h. abhängig von einer Eingabe soll beim Konsens-Problem ein bestimmter Endzustand erreicht werden. Gegeben sei eine endliche Menge A von Agenten. Jedem Agent sei ein *Anfangswert* aus der Menge $\{0, 1\}$ zugewiesen. Gesucht ist ein Algorithmus mit den folgenden drei Eigenschaften:

1. Jeder Agent entscheidet sich irgendwann unwiderruflich für einen Wert.
2. Die Entscheidungswerte aller Agenten sind gleich.
3. Sind alle Anfangswerte gleich, so ist der (gemeinsame) Entscheidungswert gleich dem gemeinsamen Anfangswert.

Die Eigenschaft 3 schließt einen trivialen Algorithmus aus, bei dem sich jeder Agent unbhängig von den Anfangswerten ohne jede Kommunikation sofort für einen festen Wert, sagen wir 0, entscheidet. Die Eigenschaften 1., 2. und 3. werden oft als Terminierung, Konsens und Nichttrivialität bezeichnet.

Wir nehmen an, daß jeder Agent mit jedem anderen Agenten direkt über Nachrichtenaustausch kommunizieren kann. Dann ist ein Algorithmus, der alle drei Eigenschaften erfüllt, schnell angegeben: Jeder Agent sende seinen Anfangswert an alle anderen Agenten und warte auf eine Nachricht von allen anderen Agenten. Fügt jeder Agent nun seinen eigenen Wert zur Multimenge der erhaltenen Werte hinzu, so hat jeder Agent dieselbe Multimenge von Werten, nämlich die Multimenge aller Anfangswerte. Mittels einer vorher festgelegten Funktion bestimme nun jeder Agent den Entscheidungswert aus dieser Multimenge – man nehme beispielweise den Wert, der am häufigsten in der Multimenge vorkommt.

Dieser einfache Algorithmus erreicht sein Ziel nicht mehr, falls Agenten *ausfallen* können. Dabei reden wir von dem Ausfall eines Agenten, falls dieser keine seiner aktivierten Aktionen mehr ausführt. Ein Ausfall besteht also dauerhaft. Wir nehmen nun im weiteren an, daß Agenten ausfallen können und fordern nur noch von nichtausfallenden Agenten, daß sie sich irgendwann entscheiden.

2.4.2 Formalisierung des Konsens-Problems

In diesem Abschnitt formalisieren wir das Konsens-Problem. Zunächst wollen wir den Transitionen eines Netzsystems Agenten zuordnen. Dies tun wir wie folgt:

Definition 2.11 (Netzsystem für A, Nachrichtensystem)

Sei A eine Menge von Agenten. Ein Netzsystem $\Sigma = (N, M^0, T^{ext})$ mit $N = (P, T; F)$ heißt *Netzsystem für* A, falls T über A sortiert ist[4], d.h. $T = (T^x)_{x \in A}$. Eine Transition $t \in T^x$ heißt *Transition von* x. Σ heißt *Nachrichtensystem für* A, falls für alle Agenten $x, y \in A$ mit $x \neq y$ gilt:

$$t \in T^x \wedge t' \in T^y \Rightarrow {}^\bullet t \cap {}^\bullet t' = \varnothing \tag{2.3}$$

$\circ$

Bedingung (2.3) formalisiert, daß Agenten nur durch Nachrichtenaustausch miteinander kommunizieren: Transitionen verschiedener Agenten stehen nicht im Konflikt zueinander. Damit kann kein Agent eine Transition eines anderen Agenten deaktivieren. Wir nehmen für den Rest dieses Abschnitts an, daß Agenten nur durch Nachrichtenaustausch kommunizieren.

Wir gehen im weiteren wie beim Mutex-Problem in mehreren Schritten vor. Zunächst definieren wir die Struktur, die ein Netzsystem haben soll, das das Konsens-Problem löst. Abb. 2.8 zeigt diese Struktur. Ein Netzsystem besitzt *Konsens-Struktur* für eine Menge von Agenten, falls für jeden Agenten vier Stellen wie in Abb. 2.8 existieren, so daß die *initial*-Stellen keine Vortransitionen und die *entschieden*-Stellen keine Nachtransitionen haben. Eine *initial*-Stelle repräsentiert einen Anfangswert eines Agenten; genauer: Eine Marke auf $initial_{x,v}$ modelliert, daß v der Anfangswert von x ist. Daher werden wir noch fordern, daß in einem gültigen Anfangszustand für jeden Agent x genau eine Marke auf einer der beiden $initial_{x,\cdot}$-Stellen liegt. Eine Marke auf einer $entschieden_{x,v}$-Stelle modelliert, daß Agent x sich für den Wert v entschieden hat.

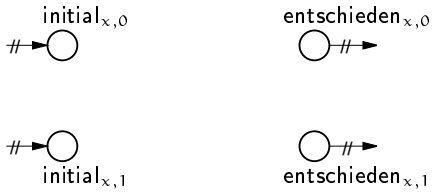


Abb. 2.8: Konsens-Struktur P_x

[4] Man stelle sich vor, daß $T = T' \times A$ für irgendein T'.

Die Einschränkung der Verbindung von *initial*-Stellen und *entschieden*-Stellen formalisiert den Ein-/Ausgabe-Charakter des Konsensproblems: Kein Anfangswert kann während eines Ablaufes des Algorithmus erzeugt werden und eine Entscheidung eines Agenten kann nicht rückgängig gemacht werden. Wir formalisieren:

Definition 2.12 (Konsens-Struktur)

Sei A ein endliche Menge von Agenten. Ein Netzsystem $\Sigma = (N, M^0, T^{ext})$ für A mit $N = (P, T; F)$ besitzt *Konsens-Struktur für* A, falls für jeden Agenten $x \in A$ genau eine Stellenmenge P_x wie in Abb. 2.8 existiert, so daß für verschiedene Agenten x und y die Stellenmengen P_x und P_y disjunkt sind und daß für alle $x \in A$ gilt:

1. $P_x \subseteq P$,
2. $(p,t) \in F \Rightarrow p \neq \mathit{entschieden}_{x,v}$ für $v = 0, 1$,
3. $(t,p) \in F \Rightarrow p \neq \mathit{initial}_{x,v}$ für $v = 0, 1$,
4. $p \in P_x \Rightarrow M^0(p) = \varnothing$,
5. $t \in \mathit{initial}_{x,v}^{\bullet} \cup {}^{\bullet}\mathit{entschieden}_{x,v} \Rightarrow t \in T^x$ ○

In Definition 2.12 Punkt 4 haben wir gefordert, daß für jeden Agenten x die Stellen aus P_x anfangs nicht markiert sind. Der Grund dafür ist, daß wir beim Konsens-Problem mehrere Anfangsmarkierungen betrachten wollen – für jede Zuordnung von Anfangswerten zu Agenten genau eine. Erst jetzt werden wir mit Hilfe des Begriffs der *Initialisierung* formalisieren, wie eine Anfangsmarkierung beschaffen sein soll.

Definition 2.13 (Initialisierung)

Sei A eine endliche Menge von Agenten und $\Sigma = (N, M^0, T^{ext})$ ein Netzsystem mit Konsens-Struktur für A. Sei $P_{init} = \{\mathit{initial}_{x,v} \mid x \in A, v \in \{0,1\}\}$ die Menge aller *initial*-Stellen von Σ. Eine Markierung I von P_{init} heißt *Initialisierung* von Σ, falls für alle Agenten $x \in A$ gilt: $I(\mathit{initial}_{x,0}) + I(\mathit{initial}_{x,1}) = 1$, d.h. falls für jeden Agenten genau eine seiner *initial*-Stellen mit genau einer Marke markiert ist. Die I-*Initialisierung von* Σ ist das Netzsystem $\Sigma^I = (N, M^0 + I, T^{ext})$. Mit I^0 (bzw. I^1) bezeichnen wir die Initialisierung, bei der der Anfangswert aller Agenten 0 (bzw. 1) ist. ○

Die Definitionen 2.12 und 2.13 schränken nur die Anfangsmarkierung der *initial*- und der *entschieden*-Stellen ein. Alle anderen Stellen eines initialisierten Netzsystems mit Konsens-Struktur können beliebig markiert sein.

Als nächstes erklären wir, wie wir Ausfälle modellieren. Wir wollen Ausfälle von Agenten durch nicht-progressive Abläufe modellieren[5]: Ein Agent x fällt in einem

[5] Alternativ modelliert man einen Aufall durch zusätzliche Transitionen, die Marken aus dem Netz entfernen; vgl. [90].

Ablauf ρ aus, falls es eine Transition von x gibt, bezüglich der ρ nicht progressiv ist. Damit wird ein Agent auch dann als ausgefallen angesehen, falls eine seiner Transitionen nicht progressiv ist, während andere Transitionen dieses Agenten unendlich oft schalten. Dieser Fall ist denkbar, falls man nicht-sequentielle Agenten betrachtet, d.h. Agenten, die aus mehreren unabhängigen Komponenten bestehen. Solche Agenten werden im weiteren allerdings keine besondere Rolle spielen, so daß wir uns der Einfachheit halber auch vorstellen können, daß jeder Agent sequentiell ist.

Definition 2.14 (Ausfall)

Sei A eine Menge von Agenten, Σ ein Netzsystem für A und ρ ein Ablauf von Σ. Ein Agent $x \in A$ *ist in* ρ *ausgefallen*, falls es eine interne Transition $t \in T^x$ von x gibt, so daß ρ bzgl. t nicht progressiv ist. Ist C ein Markierungsschnitt von ρ, so sagen wir x *ist in* C *ausgefallen* (Notation: $\rho, C \models \mathit{ausgefallen}(x)$), falls t in $C \cap \rho^\circ$ aktiviert ist. ○

Ein Ausfall kann in einem asynchronen System nicht festgestellt werden, da es ohne Zeit nicht möglich ist, zwischen einem ausgefallenen und einem sehr langsam fortschreitenden Agenten zu unterscheiden. Diese Tatsache wird durch unsere Modellierung wiedergespiegelt: Ein Ablauf, in dem ein Agent ausgefallen ist, kann zu einem Ablauf fortgesetzt werden, in dem dieser Agent nicht ausgefallen ist. Wir definieren nun, wann ein Netzsystem ausfalltolerantes Konsens-Verhalten hat. Dabei werden wir im folgenden Stellen mit Index auch in Prädikatschreibweise notieren, d.h. wir schreiben z.B. $\mathit{entschieden}(x,0)$ anstelle von $\mathit{entschieden}_{x,0}$.

Definition 2.15 (Ausfalltolerantes Konsens-Verhalten)

Sei A eine endliche Menge von Agenten und Σ ein Netzsystem für A mit Konsens-Struktur für A. Es sei weiterhin $k \in \mathbb{N}$ mit $k \leq |A|$. Σ besitzt k-*ausfalltolerantes Konsens-Verhalten*, falls für jede Initialisierung I von Σ jeder Ablauf ρ von Σ^I, in dem höchstens k Agenten ausfallen, die folgenden drei temporallogischen Eigenschaften (2.4)–(2.6) erfüllt:

$$\forall x, y \in A : \Box\, \neg(\mathit{entschieden}(x,0) \wedge \mathit{entschieden}(y,1)) \tag{2.4}$$

$$(\exists v : \forall x \in A : \mathit{initial}(x,v)) \Rightarrow (\Box\, \mathit{entschieden}(y,w) \Rightarrow v = w) \tag{2.5}$$

$$\forall x \in A : \Diamond\, \mathit{entschieden}(x,0) \vee \mathit{entschieden}(x,1) \vee \mathit{ausgefallen}(x) \tag{2.6}$$

○

Wir zeigen jetzt, daß k-ausfalltolerantes Konsensverhalten nicht möglich ist, falls $k \geq \frac{|A|}{2}$. Dies zeigen wir mit einem Standardargument: Kann die Hälfte aller Agenten ausfallen, so ist es immer möglich, daß das Agentennetzwerk in zwei Hälften zerfällt, die sich wechselseitig nicht wahrnehmen, wodurch beide Hälften widersprüchliche Werte entscheiden können.

Satz 2.16 (Notwendigkeit einer Mehrheit nicht-ausfallender Agenten)

Sei A eine endliche Menge von Agenten. Dann gibt es kein Netzsystem Σ für A mit Konsens-Struktur und k-ausfalltolerantem Konsensverhalten, falls $k \geq \frac{|A|}{2}$.

Beweis: Wir führen einen indirekten Beweis. Der Einfachheit halber betrachten wir geradzahlig viele Agenten. Sei also $A = \{x_1, \ldots, x_{2k}\}$. Sei I die Initialisierung von Σ, bei der der Anfangswert von $x_1, \ldots, x_k$ gleich 0 und der Anfangswert von $x_{k+1}, \ldots, x_{2k}$ gleich 1 ist. Dann gibt es einen Ablauf ρ_0 von Σ^I, bei dem alle Agenten x_i mit $i > k$ nichts tun, während alle x_i mit $i \leq k$ sich entscheiden. Da ρ_0 auch ein Ablauf von Σ^{I^0} ist, ist wegen (2.5) 0 der Entscheidungswert von ρ_0. Analog gibt es einen Ablauf ρ_1 von Σ^I, bei dem sich alle Agenten x_i mit $i > k$ sich für 1 entscheiden, während alle x_i mit $i \leq k$ nichts tun. Die Abläufe ρ_0 und ρ_1 sind kompatibel und $\sup(\rho_0, \rho_1)$ ist sowohl für 0 als auch für 1 entschieden – ein Widerspruch zu (2.4). □

Wir nehmen im folgenden $k < \frac{|A|}{2}$ an. Der kleinste interessante Fall ist also $k = 1$ und $|A| = 3$. Wir betrachten im nächsten Abschnitt einen Konsensalgorithmus für 3 Agenten unter Annahme eines Ausfalls.

2.4.3 Ein kleiner Konsensalgorithmus

In diesem Abschnitt gewinnen wir anhand eines einfachen Konsensalgorithmus ein Gefühl für die Schwierigkeit, Konsens ausfalltolerant zu lösen. Wir betrachten dabei drei Agenten, die paarweise miteinander Nachrichten austauschen können und nehmen an, daß ein Agent ausfallen kann. Den Algorithmus, den wir vorstellen, findet man nicht in der Literatur; vielleicht deshalb, weil er nicht offensichtlich auf beliebig viele Agenten verallgemeinerbar ist.

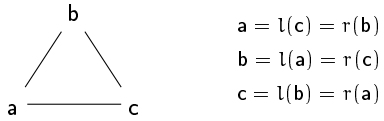


Abb. 2.9: Ein Ring von drei Agenten.

Es sei $A = \{a, b, c\}$ eine Menge von drei Agenten und N vollständig über A, so wie in Abb. 2.9 dargestellt. Dieses Netzwerk stellt einen Ring dar, so daß für jeden Agenten x genau ein *linker Nachbar* $l(x)$ und ein *rechter Nachbar* $r(x)$ existiert. Der Algorithmus wird durch das Netzsystem Σ_{10} in Abb. 2.10 dargestellt. Wir erläutern den Algorithmus entlang der einzelnen Transitionen von Σ_{10}. Am Anfang befindet sich ein Agent im Zustand *initial* und kennt seinen Anfangswert. Ein Agent im Zustand *initial* führt irgendwann Transition t_0 aus:

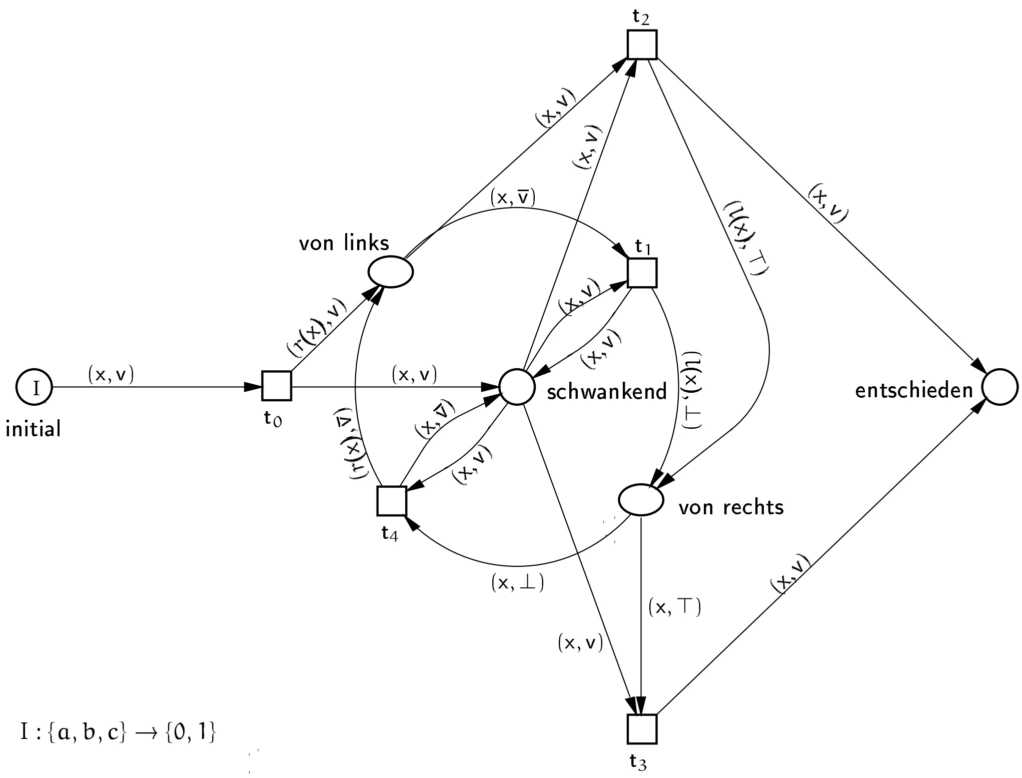


Abb. 2.10: Σ_{10} – Ein kleiner Konsensalgorithmus.

t_0 Ein initialer Agent sendet seinen Anfangswert an seinen rechten Nachbarn und wird *schwankend*.

Ein schwankender Agent kann von seinem linken Nachbarn einen Wert bekommen, der entweder mit seinem Wert *übereinstimmt* oder von seinem Wert *abweicht*.

t_1 Ein schwankender Agent bekommt von seinem linken Nachbarn einen abweichenden Wert und schickt eine *Änderungsantwort* $\perp$ zurück.

t_2 Ein schwankender Agent bekommt von seinem linken Nachbarn einen übereinstimmenden Wert, schickt eine *Erfolgsantwort* $\top$ zurück und entscheidet sich für seinen Wert.

t_3 Ein schwankender Agent bekommt von seinem rechten Nachbarn eine Erfolgsantwort und entscheidet sich für seinen Wert.

t_4 Ein schwankender Agent bekommt von seinem rechten Nachbarn eine Änderungsantwort, kehrt daraufhin seinen Wert um und sendet den neuen Wert erneut an seinen rechten Nachbarn.

Ein entschiedener Agent ist terminiert – für ihn gibt es keine aktivierte Transition.

Σ_{10} erfüllt die Sicherheitseigenschaften (2.4) und (2.5), nicht jedoch die Lebendigkeitseigenschaft (2.6). Wir werden dies hier nicht formal beweisen, führen im folgenden aber einige Korrektheitsargumente informell an.

Sind alle Anfangswerte gleich, so läuft der Algorithmus deterministisch ab, und man sieht leicht, daß (2.5) in Σ_{10} erfüllt ist. Für den Beweis von (2.4) ist die folgende Invariante wichtig: Entscheidet sich ein Agent, so entscheidet er sich immer für den aktuellen Wert seines linken Nachbarn, d.h. es gilt:

$$\Sigma_{10} \models \Box\, \mathit{entschieden}(x, v) \Rightarrow \mathit{schwankend}(l(x), v) \vee \mathit{entschieden}(l(x), v) \tag{2.7}$$

Invariante (2.7) bedeutet auch, daß ein schwankender Agent seinen Wert nicht mehr ändert, wenn sein rechter Nachbar bereits entschieden ist. Aus (2.7) kann man die Gültigkeit von (2.4) ableiten.

Die Ablaufeigenschaft (2.6) ist lebendig in Σ_{10}, wird jedoch nicht erfüllt: Σ_{10} hat einen unendlichen progressiven Ablauf ρ, in dem sich kein Agent entscheidet. Eine Schaltsequenz von ρ wird in Abb. 2.11 illustriert. Wir beginnen in dem Anfangszustand, in dem der Anfangswert von a und c jeweils 0 und der Anfangswert von b 1 ist.

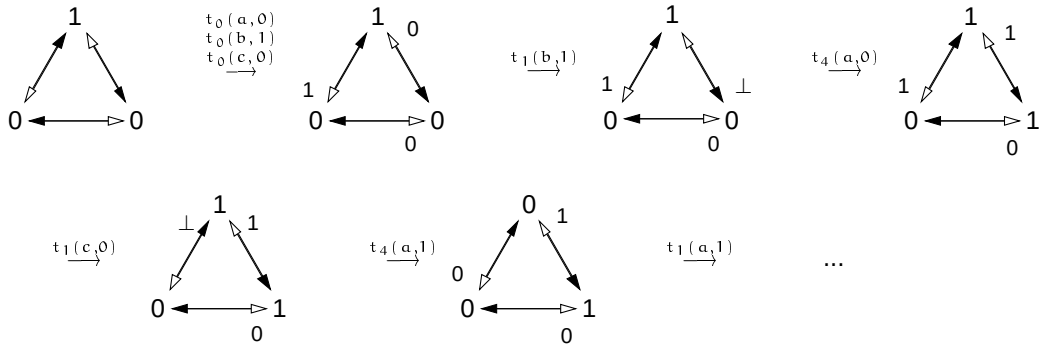


Abb. 2.11: Eine unendliche Schaltsequenz von Σ_{10}.

Wir schalten nun für jeden Agenten jeweils t_0 und erhalten so den zweiten Zustand in Abb. 2.11. Alle Agenten sind nun schwankend (mit ihren Anfangswerten) und drei Nachrichten sind unterwegs. Eine Nachricht haben wir dem Pfeil zugeordnet, der auf den Agenten zeigt, zu dem die Nachricht unterwegs ist. Als nächstes empfängt b die Nachricht von c und schickt eine Änderungsantwort zurück – wir erhalten den dritten Zustand. Jetzt erhält c diese Änderungsantwort, woraufhin er seinen Wert umkehrt und wieder an b verschickt. Wir erhalten den vierten Zustand in Abb. 2.11. Invertieren wir den vierten Zustand, d.h. kehren wir alle Werte um, so erhalten wir einen Zustand der symmetrisch zum zweiten Zustand ist. Im vierten

Zustand schickt nun a seinem linken Nachbarn b eine Änderungsantwort, woraufhin dieser im fünften Zustand seinen Wert ändert. Mit dem sechsten Zustand erhalten wir einen zum zweiten Zustand symmetrischen Zustand und können so unendlich fortsetzen.

Bei der Konstruktion der Schaltsequenz in Abb. 2.11 haben wir die Asynchronie des Systems ausgenutzt. Die Ankunft mancher Nachrichten wurde zur Konstruktion der Schaltsequenz solange hinausgezögert, bis diese Nachricht nicht mehr von Nutzen war, d.h. zu einer Entscheidung geführt hätte.

2.4.4 Unmöglichkeit von Konsens in Netzsystemen

In diesem Abschnitt zeigen wir, daß k-aufalltoleranter Konsens durch Nachrichtenaustausch für $k > 0$ in Netzsystemen unmöglich ist. Die Unmöglichkeit von ausfalltolerantem Konsens wurde in verschiedenen Systemmodellen bereits vielfach gezeigt, erstmals von Fischer, Lynch und Paterson 1983 [36] (vgl. auch [62] S. 377ff). Wir beweisen also ein im wesentlichen bekanntes Resultat. Dies soll wiederum der Vorbereitung von Resultaten folgender Kapitel dienen. Unser Beweis ist jedoch neu. Er nutzt die Struktur nicht-sequentieller Abläufe aus und erklärt die Unmöglichkeit mit Hilfe der Begriffe Konflikt und Nebenläufigkeit.

Wir führen zunächst einige nützliche Begriffe ein. Die Begriffsbildung der folgenden Definition geht auf [36] zurück.

Definition 2.17 (Bivalenter Ablauf)

Sei A eine endliche Menge von Agenten, Σ ein Netzsystem für A mit Konsensstruktur, I eine Initialisierung von Σ und sei $\nu \in \{0,1\}$. Ein (endlicher oder unendlicher) Ablauf α von Σ^I heißt *für ν entschieden* falls α eine Bedingung besitzt, die mit $entschieden_{x,\nu}$ für irgendeinen Agenten x beschriftet ist; ν heißt *möglicher Entscheidungswert* von α falls eine Fortsetzung von α in Σ^I existiert, die für ν entschieden ist; α heißt

(a) *univalent*, falls α genau einen möglichen Entscheidungswert hat.

(b) *ν-valent*, falls ν der einzig mögliche Entscheidungswert von α ist.

(c) *bivalent* falls sowohl 0 als auch 1 ein möglicher Entscheidungswert von α ist.

Σ^I ist *univalent* (bzw. *ν-valent* bzw. *bivalent*), falls der ereignislose Ablauf von Σ^I univalent (bzw. ν-valent bzw. bivalent) ist. ○

Proposition 2.18

Seien A, Σ und I wie in Definition 2.17. Dann gilt für alle Abläufe $\alpha, \beta, \alpha_0, \alpha_1$ von Σ^I:

(a) Ist α univalent und ist $\alpha \sqsubseteq \beta$, so ist β nicht bivalent.

(b) Ist α ν-valent, $\alpha \sqsubseteq \beta$ sowie β w-valent, so ist $\nu = w$.

(c) Sind α_ν ν-valent für $\nu = 0, 1$, so ist $\alpha_0 \not\parallel \alpha_1$.

Beweis: Die Eigenschaft (a) und (b) folgen sofort aus Definition 2.17; (c) folgt aus (b), da bei Annahme vom Gegenteil $\sup(\alpha_0, \alpha_1)$ sowohl 0-valent als auch 1-valent wäre. □

Wir zeigen nun, daß 1-ausfalltolerantes Konsens-Verhalten eines Netzsystems Σ die Existenz eines bivalenten Σ^I impliziert.

Lemma 2.19 (Existenz eines bivalenten Anfangs nach [36])

Seien A und Σ wie in Definition 2.17. Besitzt Σ 1-ausfalltolerantes Konsens-Verhalten, dann gibt es eine Initialisierung I von Σ, so daß Σ^I bivalent ist.

Beweis: Σ^{I^0} ist wegen (2.5) 0-valent. Analog ist Σ^{I^1} 1-valent. Wir können nun alle Initialisierungen beginnend mit I^0 und abschließend mit I^1 aufreihen, so daß sich zwei benachbarte Initialisierungen nur durch den Anfangswert eines Agenten unterscheiden. Nehmen wir an, daß keine Initialisierung bivalent ist, so gibt es zwei benachbarte Initialisierungen I_k und I_{k+1}, so daß Σ^{I_k} 0-valent ist und $\Sigma^{I_{k+1}}$ 1-valent ist. Sei x derjenige Agent, bei dem sich I_k und I_{k+1} unterscheiden. Es gibt einen Ablauf ρ^k von Σ^{I_k} und einen Ablauf ρ^{k+1} von $\Sigma^{I_{k+1}}$, bei denen x jeweils gleich zu Beginn ausfällt, und die sich lediglich in der $init_x$ Bedingung am Anfang unterscheiden. Dies ist ein Widerspruch dazu, daß ρ^k 0-valent und ρ^{k+1} 1-valent ist. □ *(vgl. Def. 2.13)*

Das folgende Lemma ist zentral für die Unmöglichkeit von ausfalltolerantem Konsens. Es besagt, daß es keinen Agenten gibt, der jemals durch eine Konfliktlösung zwischen 0-Valenz und 1-Valenz des gesamten Ablaufs entscheiden kann.

Lemma 2.20 (Verteiltheit der Entscheidung)

Sei A eine endliche Menge von Agenten, Σ ein Nachrichtensystem für A mit Konsensstruktur, I eine Initialisierung von Σ und α ein endlicher Ablauf von Σ. Weiterhin seien e_0, e_1 zwei Ereignisse mit $\alpha \stackrel{e_i}{\sqsubset} \alpha_i$ für $i = 0, 1$ mit $e_0 \# e_1$. Dann gilt: Ist α_0 0-valent, so ist α_1 nicht 1-valent.

Beweis: Da e_0 und e_1 in unmittelbarem Konflikt stehen, gehören wegen (2.3) beide Ereignisse zum selben Agenten. Sei x dieser Agent. Wir betrachten einen Ablauf ρ von Σ mit $\alpha \sqsubseteq \rho$, in dem nach dem von α bestimmten Markierungsschnitt C_α keine Transition von x mehr schaltet. Da x in C_α ausgefallen ist, ist ρ fortsetzbar: $\rho \stackrel{e_i}{\sqsubset} \rho_i$

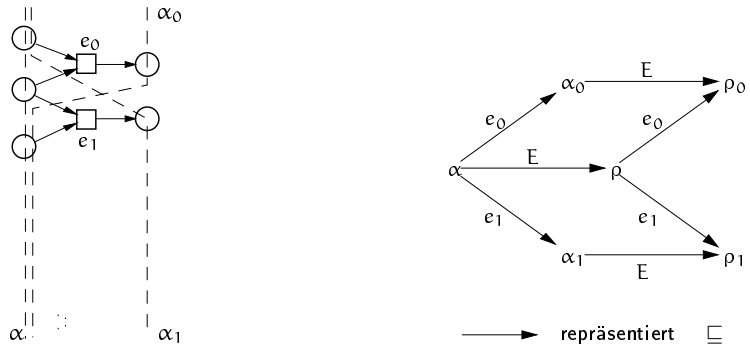


Abb. 2.12: Beweisillustration zu Lemma 2.20.

für $i = 0, 1$. Wegen (2.6) ist ρ univalent. Nehmen wir an, daß ρ 0-valent ist, so ist auch ρ_1 0-valent, was ein Widerspruch zur 1-Valenz von α_1 ist (Es gilt $\alpha_i \sqsubseteq \rho_i$ für $i = 0, 1$, vgl. Abb. 2.12). Analog erhält man einen Widerspruch unter der Annahme, daß ρ 1-valent ist. □

Mit Hilfe von Lemma 2.20 können wir jetzt die Unmöglichkeit von ausfalltolerantem Konsens in Netzsystemen beweisen.

Satz 2.21 (Unmöglichkeit von ausfalltolerantem Konsens in Netzsystemen)
Sei A eine endliche Menge von Agenten. Es gibt kein Nachrichtensystem für A, das sowohl Konsens-Struktur für A als auch k-ausfalltolerantes Konsens-Verhalten für $k > 0$ besitzt.

Beweis: Wir nehmen an, es gibt ein Nachrichtensystem Σ mit Konsens-Struktur für A und k-ausfalltolerantem Konsens-Verhalten und führen diese Annahme zum Widerspruch. Nach Lemma 2.19 gibt es eine Initialisierung I, so daß Σ^I bivalent ist. Sei π die maximale Abwicklung von Σ^I. Da Σ Konsensverhalten hat, ist jeder maximale Ablauf von π univalent. Daher gibt es ein endliches Präfix $\pi' \sqsubseteq \pi$ von π, so daß jeder maximale Ablauf von π' univalent ist. Da der ereignislose Ablauf von π' bivalent ist, gibt es ein endlichen bivalenten Ablauf α, so daß keine Fortsetzung von α bivalent ist.

Da α, aber keine Fortsetzung von α bivalent ist, gibt es zwei Ereignisse e_i mit $\alpha \overset{e_i}{\sqsubset} \alpha_i$, so daß α_i i-valent ist, für $i = 0, 1$. Die Annahme $e_0 \# e_1$ ist ein Widerspruch zu Lemma 2.20. Die Annahme e_0 co e_1 ist ein Widerspruch zu Proposition 2.18. □

Der Beweis von Satz 2.21 läßt sich auch teilweise konstruktiv führen. Dazu konstruieren wir einen progressiven Ablauf, in dem kein Agent ausfällt und in dem sich

kein Agent entscheidet. Da wir wissen, daß es einen bivalenten Anfang gibt, zeigen wir, wie man einen endlichen bivalenten Ablauf bivalent fortsetzen kann, so daß bei unendlicher Fortsetzung Progreß erfüllt ist. Wir betrachten dazu Abb. 2.13. Sei α ein endlicher bivalenter Ablauf und $\alpha \overset{e_0}{\sqsubset} \alpha_0$ mit $\widetilde{e_0} = \mathrm{t}$, so daß α_0 o.B.d.A. 0-valent ist. Da α bivalent ist, gibt es eine Fortsetzung β, die 1-valent ist. Das Ereignis e_0 ist in β nicht aktiviert, da sonst α_0 und β kompatibel wären. Da e_0 in β nicht aktiviert ist, schaltet in β ein Ereignis e, das im direkten Konflikt zu e_0 steht. Dann gibt es einen Präfix α' von β, der sowohl e_0 als auch e aktiviert; α' ist bivalent, da sowohl α_0 als auch β zu α' kompatibel sind. Sei $\alpha' \overset{e}{\sqsubset} \alpha''$. Ablauf α'' ist nicht 0-valent, da $\alpha'' \sqsubseteq \beta$; wegen Lemma 2.20 ist α'' auch nicht 1-valent. Also ist α'' bivalent. Wir können also den bivalenten Ablauf α so zu einem bivalenten Ablauf α'' fortsetzen, daß die unendliche Fortsetzung progressiv bzgl. jeder Transition t ist.

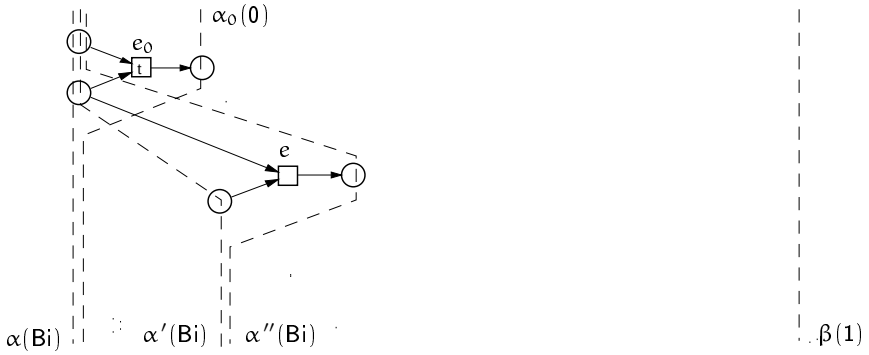


Abb. 2.13: Konstruktion eines nicht-entscheidenden progressiven Ablaufs.

3 Faire Netzsysteme

In diesem Kapitel definieren wir unser zweites Systemmodell – *faire Netzsysteme*. Ein faires Netzsystem ist ein um eine *Fairneßannahme* erweitertes Netzsystem. Desweiteren bestimmen wir in diesem Kapitel die Lösbarkeit von Mutex- und Konsens-Problem in fairen Netzsystemen.

In Abschnitt 3.1 definieren wir faire Netzsysteme, in Abschnitt 3.2 behandeln wir das Mutex-Problem und in Abschnitt 3.3 das Konsens-Problem in fairen Netzsystemen.

3.1 Faire Netzsysteme

In diesem Abschnitt erklären wir Fairneß und definieren faire Netzsysteme. In Unterabschnitt 3.1.1 definieren wir zunächst Fairneß in klassischer Weise auf Schaltsequenzen. In Unterabschnitt 3.1.2 übertragen wir Fairneß auf nicht-sequentielle Abläufe. In Unterabschnitt 3.1.3 behandeln wir dann noch ein konzeptionelles Problem unserer Fairneßdefinition.

3.1.1 Faire Schaltsequenzen

In diesem Unterabschnitt wollen wir definieren, wann eine Schaltsequenz fair bzgl. einer Transition ist. Im Abschnitt 2.3 haben wir gezeigt, daß eine Mutex-Lösung allein durch Progreß nicht möglich ist. Bekannte Mutex-Lösungen verwenden zusätzlich zu Progreß irgendeine Form von *Fairneß*. In der Literatur wird der Begriff Fairneß für eine Vielzahl verschiedener Konzepte verwendet, z.B. für Progreß, für Umgebungsfreiheit eines Systems, für probabilistische Wahl sowie für die Gleichbehandlung von Teilprozessen eines Systems durch Arbiter und Scheduler. Überblicke zu Fairneß findet man in [58, 54, 37]. Wir verwenden den Begriff Fairneß für *faire Konfliktlösung*, d.h. eine Transition t wird in einer Schaltsequenz σ fair behandelt, falls in σ gilt:

Wird unendlich oft ein Konflikt zu t *gelöst, dann unendlich oft zugunsten von* t. (3.1)

Uns wird Fairneß bezüglich einer Transition t nur dann interessieren, wenn Progreß bzgl. t erfüllt ist. Ist σ eine Schaltsequenz eines bzgl. t progressiven Ablaufs, so ist (3.1) dasselbe wie

Ist t *unendlich oft aktiviert, dann schaltet* t *unendlich oft.* (3.2)

(3.2) ist das klassische Konzept der *starken Fairneß* (*strong fairness*, in [58]: *compassion*). Eine Schaltsequenz σ ist nicht stark fair bzgl t, falls t in σ unendlich oft aktiviert ist, aber ab irgendeinem Zeitpunkt nicht mehr vorkommt. Wir formalisieren:

Definition 3.1 (Faire Schaltsequenz)

Sei Σ ein initialisiertes Netz und t eine Transition von Σ. Eine Schaltsequenz σ von Σ ist nicht *fair bzgl.* t, falls es unendlich viele Positionen i von σ gibt, so daß t in M_i aktiviert ist und t höchstens endlich oft in σ schaltet. ○

Als Beispiel betrachten wir Σ_{11} in Abbildung 3.1. Σ_{11} hat genau zwei unendliche, progressive Abläufe ρ_1 und ρ_2 – in ρ_1 schaltet b nicht, in ρ_2 schaltet b (beide Abläufe findet man in Abb. 1.10 auf Seite 21); ρ_1 hat genau eine Schaltsequenz – diese ist nicht fair bzgl. e; ρ_2 hat unendlich viele Schaltsequenzen – jede Schaltsequenz von ρ_2 ist nicht fair bzgl. e und f.

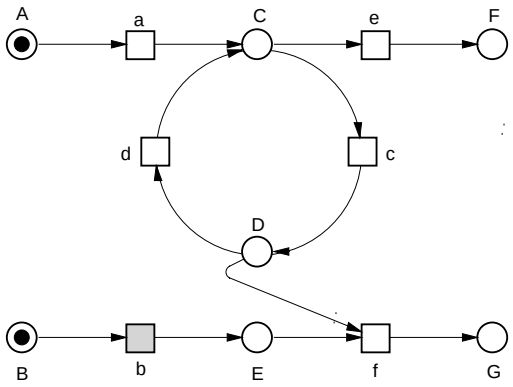


Abb. 3.1: Σ_{11}

In einem sicheren Netzsystem ist starke Fairneß bzgl. einer Transition t dasselbe wie Progreß bzgl. t plus faire Konfliktauflösung bzgl. t. Auch schwache Fairneß (siehe Abschnitt 2.1.3) hat Einfluß auf die Konfliktauflösung, jedoch nur in Konflikten, in denen Schleifen vorkommen.

3.1.2 Faire Abläufe und faire Netzsysteme

Es ist vielfach darauf hingewiesen worden [79, 53, 87], daß eine direkte Definition von starker Fairneß auf nicht-sequentiellen Abläufen problematisch ist. Dies liegt daran, daß die zeitliche Ordnung von Ereignissen in Schaltsequenzen Einfluß auf die Aktiviertheit von Transitionen hat. Betrachten wir dazu als Beispiel die in Abb. 3.2 abgebildete Abwicklung von Σ_{11}. Ist in dem maximalen Ablauf der Abwicklung, in dem e_3 vorkommt, Transition f aktiviert? Die Antwort hängt von der zeitlichen Reihenfolge der unabhängiger Ereignisse e_3 und e_4 ab: Tritt e_3 vor e_4 auf, dann ist f nicht aktiviert, tritt jedoch e_4 vor e_3 auf, so ist f aktiviert. Hängt das Auftreten eines Konfliktes von der Reihenfolge unabhängiger Ereignisse ab, so spricht man in der Petrinetztheorie von *Konfusion* (siehe z.B. [84, 87]). In einer konfusen Situation überlappen sich Konflikt und Synchronisation. Reisig bezeichnet einen derartigen Konflikt zwischen e_3 und e_5 in [79] als *nicht objektiv* mit der Intuition, daß manche Beobachter des Ablaufs den Konflikt zwischen e_3 und e_5 sehen, andere aber nicht. Dies liegt daran, daß die nebenläufigen Bedingungen b_2 und b_5 nicht notwendig gleichzeitig zu sehen sind. Reisig charakterisiert in [79] mit dem Begriff *strong concurrency*, wann zwei Bedingungen eines Ablaufs in jeder Schaltsequenz dieses Ablaufs gleichzeitig zu sehen sind. Mit Konfusion werden wir uns später in dieser Arbeit noch intensiver beschäftigen.

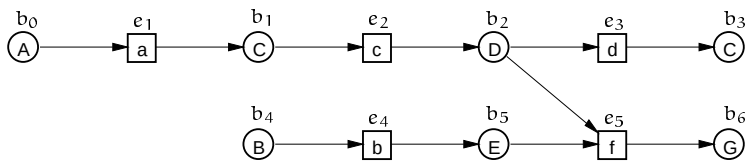


Abb. 3.2: Eine Abwicklung von Σ_{11} mit Konfusion.

Wir erklären nun Fairneß auf einem nicht-sequentiellen Ablauf mit Hilfe seiner Schaltsequenzen. Einen Ablauf betrachten wir dann als nicht fair bzgl. einer Transition t, falls jede zeitliche Ordnung dieses Ablaufs nicht fair bzgl. t ist.

Definition 3.2 (Fairer Ablauf)

Sei Σ ein initialisiertes Netz und t eine Transition von Σ. Ein bzgl. t progressiver Ablauf ρ ist nicht *fair bzgl.* t, falls jede Schaltsequenz von ρ nicht fair bzgl. t ist. Ein bzgl. t nicht progressiver Ablauf ist fair bzgl t. ○

Der unendliche Ablauf ρ von Σ_{11}, in dem b schaltet, ist unfair gegenüber e und f. Während jeder Konflikt in ρ zwischen e und c objektiv ist, ist kein Konflikt in ρ zwischen d und f objektiv. In ρ stimmen keine zwei Schaltsequenzen darin überein, ab wann der Konflikt zwischen d und f eintritt, d.h. nach dem wievielten Schalten

von c ein Konflikt zwischen d und f vorliegt, alle Schaltsequenzen stimmen jedoch darin überein, daß ein Konflikt zwischen d und f unendlich oft in ρ vorliegt.

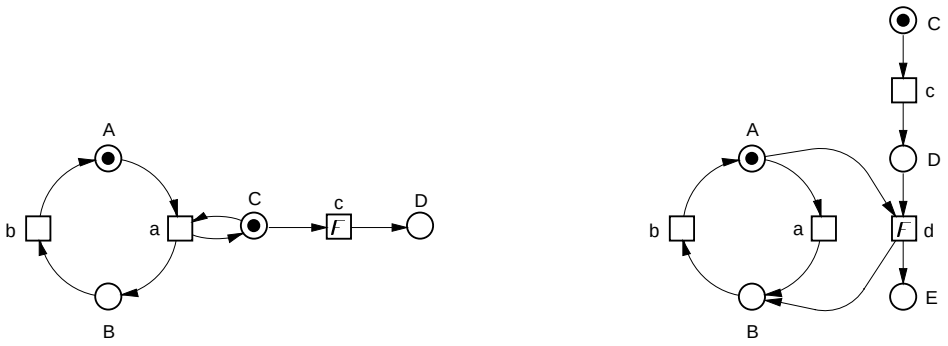
Bevor wir noch einige Beispiele für Fairneß kennenlernen, kommen wir nun zur Definition des *fairen Netzsystems*. Ein *faires Netzsystem* ist ein Netzsystem, in dem einige interne Transitionen als *faire Transitionen* ausgezeichnet sind.

Definition 3.3 (Faires Netzsystem)

Ein *faires Netzsystem* $\acute{\Sigma} = (\Sigma, T^{fair})$ besteht aus einem Netzsystem Σ mit Transitionsmenge T und Menge externer Transitionen T^{ext}, und einer Menge $T^{fair} \subseteq T \setminus T^{ext}$ ausgezeichneter interner Transitionen von Σ. Ein Element von T^{fair} heißt *Fairneßtransition*. Ein Ablauf ρ von $\acute{\Sigma}$ ist *fair*, falls ρ fair bzgl. jedem $t \in T^{fair}$ ist. ○

Eine Fairneßtransition stellen wir graphisch durch das Symbol F dar (vgl. Abb. 3.3).

Abb. 3.3 zeigt zwei faire Netzsysteme. In jedem progressiven und fairen Ablauf von Σ_{12} gilt $\Diamond D$, in jedem progressiven und fairen Ablauf von Σ_{13} gilt $\Diamond E$. Wir wollen nun durch Interpretation der beiden fairen Netzsysteme illustrieren, was eine Fairneßannahme in realen Systemen bedeutet.



(a) Σ_{12} (b) Σ_{13}

Abb. 3.3: Zwei faire Netzsysteme.

Für Σ_{12} stellen wir uns vor, daß die Stellen C und D zwei Zustände einer Tür modellieren: Ist C markiert, so ist die Tür offen, ist D markiert, so ist die Tür geschlossen. Ist die Tür offen, so kann eine Person durch die Tür gehen (Transition a). Fairneß bzgl. c bedeutet dann, daß die Tür irgendwann geschlossen wird – trotz eines nicht abreißenden Stroms von Personen, die durch die Tür gehen wollen.

Bei Σ_{13} in Abb. 3.3(b) stellen wir uns einen Wartenummernspender vor, von dem immer wieder eine Wartenummer gezogen werden kann (Transition a). Transition

c modelliert die asynchrone Ankunft einer ausgezeichneten Person, die eine Wartenummer ziehen möchte (Transition d). Fairneß bzgl. d fordert, daß die ausgezeichnete Person irgendwann eine Wartenummer zieht – trotz beliebig großen Andrangs.

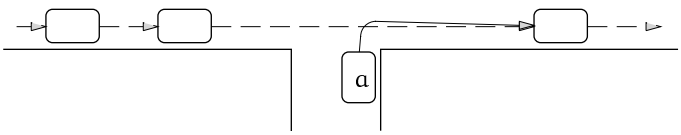


Abb. 3.4: Fairneß beim Einbiegen in eine Hauptstraße.

Abb. 3.4 zeigt ein weiteres Fairneßproblem. Abgebildet ist eine Nebenstraße, die in eine Hauptstraße einmündet. Fairneß verlangt, daß Auto a irgendwann nach rechts in die Hauptstraße einbiegt – trotz eines nichtabreißenden Stroms von Autos, die von links kommen. Ein ähnliches Problem tritt auf, wenn ein Rechnerknoten eine Nachricht über Ethernet verschicken möchte.

In Digitalrechnern werden ähnliche Situationen (die Wahrnehmung asynchron eintreffender Signale) durch spezielle Hardware, sog. *Arbiter*, behandelt. Ein *Arbiter* entscheidet zwischen zwei unabhängig eintreffenden Signalen, welches von beiden Signalen „zuerst“ eintrifft. Ein Arbiter kann anomales Verhalten zeigen, indem er in seltenen Fällen ungewöhnlich viel Zeit braucht, um zu entscheiden. Lamport beleuchtet in [56] den mathematischen Hintergrund des Arbiter-Problems (auch: *glitch phenomenon*) und argumentiert, daß jeder Arbiter, unabhängig von seiner physikalischen Realisierung, immer in einigen Fällen beliebig viel Zeit für eine Entscheidung benötigt. Smith zeigt in [87], daß jeder Arbiter Konfusion enthält.

3.1.3 Ein Problem von starker Fairneß

In diesem Abschnitt zeigen wir ein Problem unserer Fairneßdefinition auf und zeigen, wie wir dieses Problem vermeiden können.

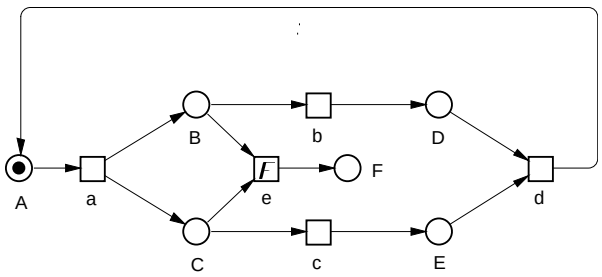


Abb. 3.5: Σ_{14} – Ein zeitbehaftete Fairneßannahme.

Abb. 3.5 zeigt das faire Netzsystem Σ_{14}. Wir stellen uns für dieses System vor, daß ein Agent durch Transition a zwei unabhängige Nachrichten B und C an einen zweiten Agenten schickt. Empfängt der zweite Agent beide Nachrichten gleichzeitig, so kann er beide Nachrichten vernichten (Transition e). Eine einzelne Nachricht kann der Agent an einen dritten Agenten weiterschicken (Transition b bzw. c).

Jeder faire Ablauf von Σ_{14} ist endlich, da Fairneß das Schalten von e erzwingt. Die Annahme, daß e irgendwann schaltet, entspricht jedoch nicht unserer Intuition von Fairneß, da sie eine Zeitannahme beeinhaltet, nämlich die Annahme, daß die Nachrichten B und C irgendwann einmal gleichzeitig ankommen.

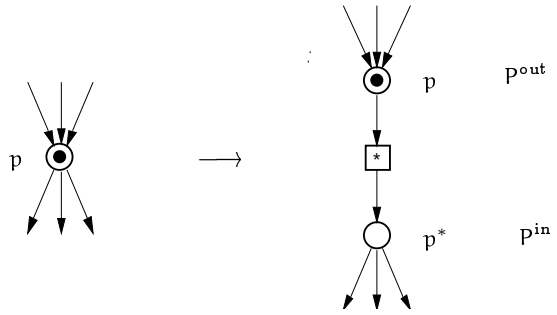


Abb. 3.6: Verfeinerung zu einem zeitlosen Netzsystem.

Den unerwünschten Effekt können wir einerseits auf eine nicht-adäquate Modellierung unseres vorgestellten Systems durch Σ_{14}, andererseits auf die Grobheit des durch Schaltsequenzen implizierten Zeitbegriffes schieben. Verhindern können wir den unerwünschten Effekt, indem wir die Stellen B und C wie in Abb. 3.6 verfeinern. Auf diese Weise werden Kanäle explizit modelliert. Im verfeinerten System ist der unendliche Ablauf nicht mehr unfair. Hier handelt es sich also um dasselbe Phänomen wie bei schwacher Fairneß in Abschnitt 2.1.3: auch starke Fairneß ist nicht robust gegenüber einfacher Verfeinerung.[1]

Um das beschriebene Problem von vornherein auszuschließen, können wir alle Stellen eines fairen Netzsystems wie in Abb. 3.6 verfeinern. Das dabei entstehende faire Netzsystem nennen wir *zeitloses faires Netzsystem.*

Definition 3.4 (Zeitloses faires Netzsystem)

Sei $\acute{\Sigma}$ ein faires Netzsystem. Verfeinern wir alle Stellen von $\acute{\Sigma}$ wie in Abb. 3.6, so heißt das verfeinerte faire Netzsystem $\acute{\Sigma}'$ *zeitloses faires Netzsystem.* Dabei gibt es zu jeder Stelle p von $\acute{\Sigma}$ in $\acute{\Sigma}'$ die Stellen p und p*. Die Stelle p von $\acute{\Sigma}'$ heißt *Eingabestelle*, die Stelle p* heißt *Ausgabestelle* von $\acute{\Sigma}'$. Die Menge der Eingabestellen von $\acute{\Sigma}'$

[1] Der für nicht-sequentielle Abläufe entwickelte Begriff der *Kantenfairneß* [51, 91] vermeidet den beschriebenen unerwünschten Effekt. Um aber einen stärkeren Bezug zur Literatur herzustellen, haben wir starke Fairneß als Fairneßbegriff für diese Arbeit ausgewählt.

bezeichnen wir mit P^{in}, die Menge der Ausgabestellen durch P^{out}. Eine hinzugefügte Transition von $\acute{\Sigma}'$ heißt *Transporttransition* und wird graphisch durch das Symbol * gekennzeichnet. Jede Transporttransition von $\acute{\Sigma}'$ ist eine interne Transition. Eine Transition von $\acute{\Sigma}'$, die keine Transporttransition ist, heißt *Originaltransition* von $\acute{\Sigma}'$. ○

3.2 Mutex in fairen Netzsystemen

In diesem Abschnitt halten wir der Vollständigkeit halber fest, daß das Mutex-Problem mit Hilfe von Fairneß lösbar ist.

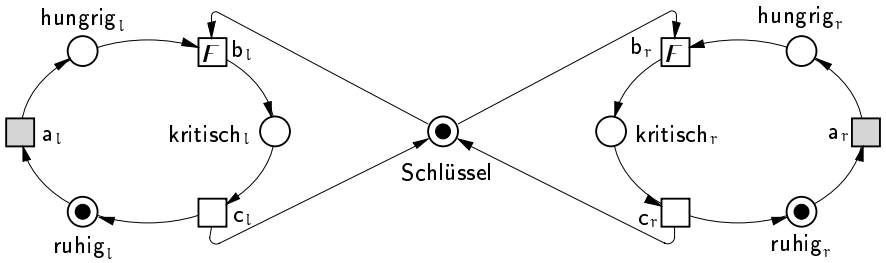


Abb. 3.7: Σ_{15} – Ein faires Netzsystem mit Mutex-Struktur und Mutex-Verhalten.

Ein faires Netzsystem, das Mutex löst, zeigt Abb. 3.7. Das zugrundeliegende Netzsystem, das wir bereits in Abschnitt 2.3.1 betrachtet haben, wurde hier um die Fairneßforderung für die Transitionen b_l und b_r erweitert. Ein Ablauf, in dem ein Agent hungrig ist und nicht kritisch wird, weil der andere Agent immer wieder den Schlüssel bekommt, ist unfair und damit ausgeschlossen. Jeder faire Ablauf von Σ_{15} in Abb. 3.7 hat Mutex-Verhalten. Wir halten fest:

Satz 3.5 (Mutex in fairen Netzsystemen)

Es gibt ein faires Netzsystem, das sowohl Mutex-Struktur als auch Mutex-Verhalten hat.

3.3 Konsens in fairen Netzsystemen

In diesem Abschnitt zeigen wir, daß das Konsens-Problem in fairen Netzsystemen nicht gelöst werden kann. Damit erweitern wir Satz 2.21 auf faire Netzsysteme. Auch Fischer, Lynch und Paterson benutzen in [36] ein Modell (im folgenden: das *FLP-Modell*, ein konkretes System dieses Modells nennen wir *FLP-System*), das Fairneß annimmt. Fairneß wird im FLP-Modell jedoch nur an bestimmten Stellen angenommen. (Wir präzisieren dies im nächsten Abschnitt.) Faire Netzsysteme erlauben Fairneß an beliebiger Stelle und sind daher technisch gesehen allgemeiner als FLP-Systeme. Wir zeigen jedoch, daß dieser Unterschied in den meisten Fällen keine Rolle spielt. Genauer: Wir zeigen, daß jedes faire Netzsystem $\acute{\Sigma}$ auf ein FLP-System Σ' abgebildet werden kann, so daß Σ' dieselben Schaltsequenzeigenschaften wie $\acute{\Sigma}$ besitzt. Damit gilt: Ist $\acute{\Sigma}$ eine Konsenslösung, dann auch Σ'. Somit folgt aus dem Ergebnis von Fischer, Lynch und Paterson die Nichtexistenz eines fairen Netzsystems, das Konsens löst.

Der Abschnitt ist wie folgt gegliedert. Zunächst stellen wir das FLP-Modell in Unterabschnitt 3.3.1 vor. In Unterabschnitt 3.3.2 zeigen wir dann, wie wir ein faires Netzsystem auf ein FLP-System abbilden.

3.3.1 Das Modell von Fischer, Lynch und Paterson

Das FLP-Modell [36] ist I/O-Automaten [61] ähnlich. Jeder Agent ist im FLP-Modell durch einen deterministischen sequentiellen Automaten modelliert. Alle Agenten sind durch ein asynchrones nichtdeterministisches Kommunikationssystem verbunden, über das jeder Agent Nachrichten versenden kann. Ein Agent x kann in einem atomaren Schritt eine Nachricht empfangen, seinen Zustand ändern und eine Menge von Nachrichten versenden. Dabei fragt der Agent x zunächst das Kommunikationssystem, ob eine Nachricht für ihn vorliegt. Dann gibt das Kommunikationssystem nichtdeterministisch entweder ein spezielles Symbol $\perp$ für „keine Nachricht“ oder eine an x gesendete Nachricht m zurück. Im letzteren Fall sagen wir x *empfängt* m. Jeder Agent ist in jedem Zustand bereit, jede Nachricht zu empfangen, d.h. die Übergangsfunktion eines Agenten ist total.

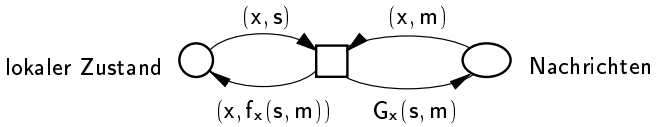


Abb. 3.8: Σ_{16} – Netzdarstellung eines FLP-Systems.

Abb. 3.8 zeigt ein FLP-System als Netz. Dabei bezeichnet x einen Agenten, s den Zustand eines Agenten und m einen Nachrichteninhalt; f_x bezeichnet die Zustands-

übergangsfunktion von x und $G_x(s, m)$ bezeichnet die Menge von Nachrichten, die x sendet, wenn er im Zustand s die Nachricht m empfangen hat[2].

Im FLP-Modell werden zwei Lebendigkeitsannahmen getroffen. Erstens wird angenommen, daß jeder nicht-ausfallende Agent maximal viele Schritte macht. Zweitens wird im FLP-Modell angenommen, daß jede Nachricht, die an einen nichtausfallenden Agenten gesendet wurde, irgendwann von diesem Agenten empfangen wird. Diese letztere Annahme enthält Fairneß. Um dies zu sehen, betrachten wir drei Agenten wie in Abb. 3.9. Wir nehmen an, daß Agent a nacheinander unendlich viele Nachrichten $n_1, n_2, \ldots$ an c schickt. Völlig unabhängig von Agent a schickt Agent b eine Nachricht m an c. Da c sequentiell empfängt, müssen Konflikte gelöst werden – ein Konflikt besteht darin, ob c als nächstes m oder das nächste n_i empfängt. Die Annahme, daß m irgendwann von c empfangen wird, ist also eine Fairneßannahme.

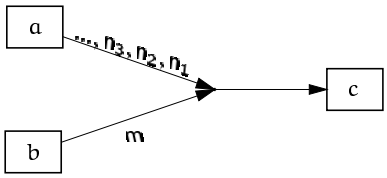


Abb. 3.9: Fairneß im FLP-Modell.

Halten wir nun die Lebendigkeitsannahmen des FLP-Modells formal fest. Ein Ablauf ρ von Σ_{16} ist *zulässig*, falls für jeden Agenten x gilt: Gibt es in ρ° eine an x gesendete Nachricht, so ist x in ρ ausgefallen. Eine Schaltsequenz σ von Σ_{16} ist *zulässig*, falls σ die Schaltsequenz eines zulässigen Ablaufs ist. Im nun folgenden Abschnitt bilden wir jedes faire Netzsystem auf ein hinreichend äquivalentes FLP-System ab.

3.3.2 Unmöglichkeit von Konsens in fairen Netzsystemen

Wir konstruieren nun zu jedem zeitlosen fairen Nachrichtensystem $\acute{\Sigma}$ ein FLP-System, das im wesentlichen[3] dieselben Schaltsequenzeigenschaften besitzt. Dabei betrachten wir *Konfliktcluster* von Transitionen. Sei t eine Originaltransition von $\acute{\Sigma}$. Das *Konfliktcluster* $\Gamma(t)$ von t ist die kleinste Menge von Transitionen, für die $t \in \Gamma(t)$ und $t_1 \in \Gamma(t) \wedge {}^\bullet t_1 \cap {}^\bullet t_2 \neq \varnothing \Rightarrow t_2 \in \Gamma(t)$ gilt. Sei Γ die Menge aller Konfliktcluster von Originaltransitionen von $\acute{\Sigma}$. Verschiedene Konfliktcluster sind disjunkt. Alle Transitionen eines Konfliktclusters gehören wegen der Verteiltheitsbedingung (2.3) zum selben Agenten. Andererseits kann ein Agent aus mehreren Konfliktclustern bestehen. Aufgrund der Zeitlosigkeit hat jedes Konfliktcluster eine Form wie in Abb. 3.10. Dabei sei für ein Konfliktcluster $x \in \Gamma$ die Menge der Eingangsstellen

[2] Die Möglichkeit, daß ein Agent auch ohne Nachricht seinen Zustand ändern kann, haben wir in Σ_{16} nicht modelliert. Dies kann aber leicht ergänzt werden.

[3] Wir präzisieren dies weiter unten.

von x durch P_x^{in} sowie die Menge der Ausgangsstellen von x durch P_x^{out} bezeichnet. Die Menge der Originaltransitionen von x bezeichnen wir durch T_x.

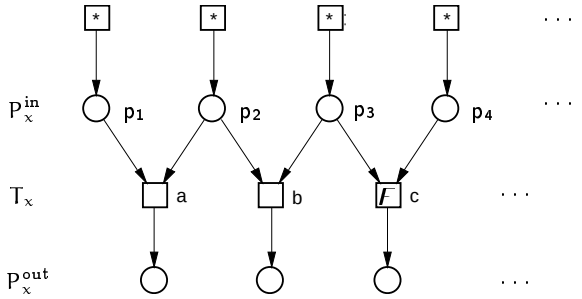


Abb. 3.10: Ein Konfliktcluster $x \in \Gamma$.

Abb. 3.11 zeigt Σ_{17} – das FLP-System, auf das wir $\acute{\Sigma}$ abbilden. Wir erläutern nun Σ_{17}. Ein Konfliktcluster $x \in \Gamma$ ist eine handelnde Einheit in Σ'. Auf der Stelle A liegt für jedes Konfliktcluster $x \in \Gamma$ immer genau eine Marke (x, M, s), wobei $M \in \mathfrak{M}(P_x^{in})$ eine Multimenge von Eingangsstellen von x und s ein zusätzlicher Zustand ist. Die Multimenge M repräsentiert die Markierung der Eingangsstellen von x in $\acute{\Sigma}$, die anfangs leer ist. Die Bedeutung des zusätzlichen Zustands s erklären wir weiter unten (dabei bezeichnet s_x in Abb. 3.11 den Anfangswert dieses Zustands für Agent x). Die Stelle B repräsentiert die Ausgangsstellen von $\acute{\Sigma}$. Eine Marke p auf B ist eine Ausgangsstelle von $\acute{\Sigma}$. Anfangs liegt auf B die Multimenge M_0 der in $\acute{\Sigma}$ markierten Ausgangsstellen. Wir erklären nun die beiden Transitionen von Σ_{17}.

- t_1 Konfliktcluster x empfängt eine Marke p und nimmt diese in seine Multimenge M auf. Dies simuliert das Schalten der Transporttransition $t \in {}^\bullet p$ in $\acute{\Sigma}$.

- t_2 Transition t_2 simuliert das Schalten von Originaltransitionen von $\acute{\Sigma}$. Dabei wird das Schalten einer Originaltransitionen von $\acute{\Sigma}$ immer sobald wie möglich simuliert, d.h.: Bevor t_1 schaltet, ist durch die Multimenge M keine Originaltransition von x aktiviert. Durch das Schalten von t_1 ist zu M eine Marke p hinzugekommen und durch $M \cup \{p\}$ sind ggf. mehrere Originaltransitionen von x aktiviert. Diese stehen aber paarweise im Konflikt zueinander (nämlich über p); t_2 simuliert nun das Schalten einer dieser durch $M \cup \{p\}$ aktivierten Originaltransitionen. Liegt ein Konflikt vor, so wird dieser fair gelöst. Zur fairen Lösung von Konflikten (z.B. durch das Standardverfahren der dynamischen Priorität, siehe [24], Kap. 12) wird der zusätzliche Zustand s verwendet. Ist keine Transition durch $M \cup \{p\}$ aktiviert, so bleibt die Multimenge M sowie die Markierung von B durch Schalten von t_2 unverändert. In diesem Fall bezeichnen wir das Schalten von t_2 als *Stotterschritt*.

Der formale Beweis, daß Σ_{17} im wesentlichen alle Schaltsequenzeigenschaften des fairen Netzsystemn $\acute{\Sigma}$ besitzt, wird mit Hilfe einer klassischen *Verfeinerungabbildung*

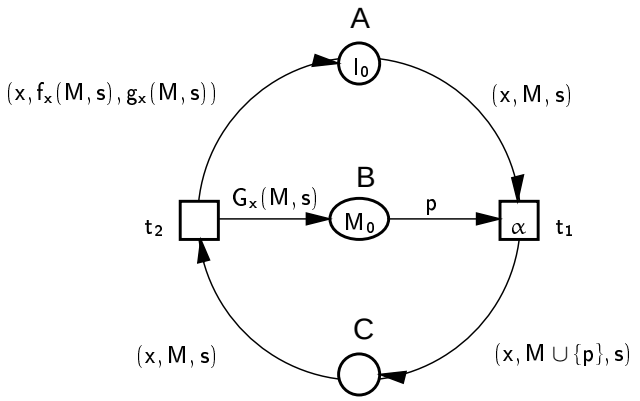


Abb. 3.11: Σ_{17} – das FLP-System zu $\acute{\Sigma}$.

[2] durchgeführt. Dies ist eine Abbildung φ, die jede Markierung M von Σ_{17} auf eine Markierung $\varphi(M)$ von $\acute{\Sigma}$ abbildet; φ ist wie folgt definiert:

$$\text{für } p \in P_x^{in} : \varphi(M)[p] = k, \text{ falls } M \models A(x,N,s) \vee C(x,N,s) \text{ mit } N[p] = k$$
$$\text{für } p \in P_x^{out} : \varphi(M)[p] = M(B)[p]$$

Wir präzisieren nun, was es heißt, daß Σ_{17} im wesentlichen die gleichen Schaltsequenzeigenschaften wie $\acute{\Sigma}$ hat.

Lemma 3.6 (Abbildung eines fairen Netzsystems auf ein FLP-System)

Sei $\sigma = M_0, M_1, \ldots$ die Markierungssequenz einer zulässigen Schaltsequenz von Σ_{17}, sei $\varphi(\sigma) = \varphi(M_0), \varphi(M_1), \ldots$ und sei $\psi(\sigma)$ die Sequenz von Markierungen von $\acute{\Sigma}$, die aus $\varphi(\sigma)$ durch Streichung endlich vieler aufeinanderfolgender Stotterschritte entsteht. Dann ist $\psi(\sigma)$ eine faire und progressive Schaltsequenz von $\acute{\Sigma}$.

Beweis: Wir gehen in zwei Schritten vor.

1. Wir zeigen zunächst für eine Markierungssequenz einer beliebigen Schaltsequenz von Σ_{17}, daß $\psi(\sigma)$ eine Schaltsequenz von $\acute{\Sigma}$ ist. Sei M^0 die Anfangsmarkierung von Σ_{17}. Dann gilt: $\varphi(M^0)$ ist Anfangsmarkierung von $\acute{\Sigma}$. Außerdem gilt: Aus $M \rightarrow M'$ folgt $\varphi(M) \rightarrow \varphi(M')$ oder $\varphi(M) = \varphi(M')$, wobei der letzte Fall höchstens beim Schalten von t_2 vorkommt, d.h. es gibt nie zwei aufeinanderfolgende Stotterschritte.

2. Ist σ zulässig, dann ist $\varphi(\sigma)$ und damit auch $\psi(\sigma)$ progressiv und fair. Der Fairneßannahme gegenüber Nachrichten von Σ_{17} entspricht der Progreß bzgl. einer Transporttransition in $\acute{\Sigma}$, Progreß gegenüber t_2 entspricht Fairneß und Progreß von Originaltransitionen von $\acute{\Sigma}$. □

Σ_{17} kann auch so konstruiert werden, daß die Agenten von $\acute{\Sigma}$ die handelnden Einheiten von Σ_{17} sind. Dazu müssen nur alle Konfliktcluster eines Agenten zusammengefaßt werden. Wir können nun aus Lemma 3.6 die Unmöglichkeit von Konsens in fairen Netzsystemen ableiten.

Satz 3.7 (Unmöglichkeit von Konsens in fairen Netzsystemen)

Sei A eine endliche Menge von Agenten. Dann gibt es kein faires Netzsystem für A, das sowohl Konsens-Struktur für A als auch k-ausfalltolerantes Konsens-Verhalten für $k > 0$ besitzt.

Beweis: Gäbe es ein faires Netzsystem für A, das sowohl Konsens-Struktur für A als auch k-ausfalltolerantes Konsens-Verhalten für $k > 0$ besitzt, so gäbe es nach Lemma 3.6 ein FLP-System, das Konsens 1-ausfalltolerant löst. Das ist aber nach [36] nicht der Fall. □

Der wesentliche Beitrag dieses Abschnitts ist die Erkenntnis, daß starke Fairneß im Kontext sequentieller Agenten, die ausschließlich über Nachrichten kommunizieren, gerade bedeutet, daß jede Nachricht irgendwann verbraucht wird.

4 Randomisierte Netzsysteme

In diesem Kapitel erweitern wir Netzsysteme um Münzwürfe zu *randomisierten Netzsystemen*. Mit randomisierten Netzsystemen kann man randomisierte verteilte Algorithmen modellieren. Wir entwickeln eine nicht-sequentielle Semantik für randomisierte Netzsysteme und erhalten so die erste nicht-sequentielle Semantik für randomisierte verteilte Algorithmen. Diese nicht-sequentielle Semantik weist einige Unterschiede zur klassischen sequentiellen Semantik randomisierter Algorithmen auf, die wir in Abschnitt 4.1.7 diskutieren.

Desweiteren bestimmen wir die Lösbarkeit von Mutex- und Konsens-Problem in randomisierten Netzsystem. Wir zeigen, daß überraschenderweise – trotz der beträchtlichen Ausdrucksstärke randomisierter Netzsysteme – das Mutex-Problem in randomisierten Netzsystemen nicht lösbar ist. Die Hintergründe dieses Resultats diskutieren wir in Abschnitt 4.3.2. Wir beginnen mit der Modellierung randomisierter Algorithmen durch Petrinetze.

4.1 Ein Petrinetzmodell für randomisierte Algorithmen

In diesem Abschnitt definieren wir randomisierte Netzsysteme und ihre Semantik. Wir beginnen mit einem Überblick über die Modellierung randomisierter verteilter Algorithmen.

4.1.1 Randomisierte Algorithmen

Randomisierte Algorithmen (sequentielle und verteilte) gewinnen immer mehr an Bedeutung, da sie oft Probleme einfacher und effizienter lösen als herkömmliche Algorithmen. Einige randomisierte Algorithmen lösen Probleme mit Wahrscheinlichkeit 1, die herkömmliche Algorithmen nachweislich nicht lösen können. Da der Unterschied zwischen den Aussagen „der Algorithmus erreicht seine Ziele mit Wahrscheinlichkeit 1“ und „der Algorithmus erreicht seine Ziele immer“ praktisch keine Rolle spielt, sagen wir in Zukunft, daß ein randomisierter Algorithmus ein Problem löst, wenn der Algorithmus das Problem mit Wahrscheinlichkeit 1 löst. Die genaue Bedeutung dessen werden wir uns in diesem Kapitel noch erarbeiten. Beispiele für

Probleme, die durch randomisierte Algorithmen, nicht aber durch herkömmliche Algorithmen gelöst werden können, sind das Symmetriebruch-Problem [42, 45], das Choice Coordination-Problem [73] und das Konsens-Problem [36]. Einen Überblick über randomisierte Algorithmen gibt [39].

Das wohl bekannteste Beispiel für die Anwendung randomisierter verteilter Algorithmen ist *Symmetriebruch*. Dabei betrachtet man *symmetrische Zustände* und *symmetrische Algorithmen*. Ein *symmetrischer Zustand* eines verteilten Systems ist ein globaler Zustand, bei dem die lokalen Zustände aller Agenten identisch sind. Symmetrische Zustände betrachtet man in *anonymen Netzwerken* – das sind Netzwerke in denen Agenten keine Identifikatoren besitzen. Ein *symmetrischer Algorithmus* ist ein verteilter Algorithmus, bei dem die Programme aller Agenten identisch sind. Bei einem symmetrischen verteilten Algorithmus kann in einem symmetrischen Zustand durch simultane Ausführung identischer Aktionen aller Agenten wieder ein symmetrischer Folgezustand entstehen. Beim Symmetriebruch soll nun aus einem symmetrischen Anfangszustand irgendwann ein asymmetrischer Zustand erreicht werden. Dies ist durch einen symmetrischen Algorithmus, bei dem alle Agenten deterministisch sind, nicht möglich. Mit Münzwurf ist dies jedoch leicht zu erreichen.

Auf diese Weise kann man mit Randomisierung symmetrische Lösungen (d.h. einen symmetrischen Algorithmus mit symmetrischem Anfangszustand) für solche Probleme erhalten, die das Erreichen eines asymmetrischen Zustands fordern. Asymmetrische Zustände sind zum Beispiel: genau ein Leader ist gewählt (bei Leader Election), genau ein Token existiert (Token Ring) und genau ein Agent ist kritisch (Mutex). Hart, Sharir und Pnueli stellen in [40] mit Erstaunen fest, daß es allerdings vom konkreten Problem abhängt, ob es tatsächlich einen randomisierten symmetrischen Algorithmus gibt, der das Problem löst. Während Leader Election unter Randomisierung eine einfache symmetrische Lösung hat[1], gibt es für das Mutex-Problem auch unter Randomisierung keine symmetrische Lösung, falls Agenten nur über Nachrichten kommunizieren. Dies zeigen Hart, Sharir und Pnueli in [40]. Sie schließen ihr Papier mit der Bemerkung:

> These phenomena call for further study to understand better the distinction between those concurrent problems that admit probabilistic solutions that are better than deterministic solutions, and those problems that do not benefit from introduction of randomization.

[1] Voraussetzung hierfür ist, daß jedem Agenten die Anzahl aller Agenten des Netzwerks bekannt ist, vgl. [42].

4.1.2 Randomisierte Netzsysteme

Ein randomisierter verteilter Algorithmus ist ein verteilter Algorithmus, bei dem das Programm eines Agenten Befehle der Form

$$x := \text{Resultat des Wurfs einer (ggf. n-seitigen) idealen Münze}$$

enthält. Daher erhält man einen Modellierungsformalismus für randomisierte verteilte Algorithmen, indem man einen Modellierungsformalismus für verteilte Algorithmen um ein Münzwurf-Konstrukt erweitert.

Ein Modell für randomisierte verteilte Algorithmen, das auf Petrinetzen beruht, gibt es bisher nicht. Auf anderen Formalismen beruhende Modelle für randomisierte Algorithmen sind *probabilistische I/O-Automaten* [86], *probabilistische Programme* [71] und *probabilistische nebenläufige Prozesse* [11] sowie ein von Rao vorgeschlagenes Modell [75], das auf UNITY [24] beruht. Jedes dieser Modelle benutzt eine sequentielle Semantik.

Einen Münzwurf modellieren wir im Petrinetz durch einen *Free-Choice-Konflikt* wie in Abb. 4.1(a). In einem *Free-Choice-Konflikt* hat jede Transition nur eine Vorbedingung. Der in Abb. 4.1(a) modellierte einfache Münzwurf hat zwei Ausgänge: *Kopf* und *Zahl*. Jedem dieser Ausgänge ordnen wir die *Wahrscheinlichkeit* $\frac{1}{2}$ zu. Eine Transition, die einen Ausgang eines Münzwurfs modelliert, nennen wir *probabilistisch*. Eine probabilistische Transition ist graphisch durch das Symbol % gekennzeichnet.

Wir wollen auch *allgemeine Münzwürfe* modellieren. Ein *allgemeiner Münzwurf* hat n mögliche Ausgänge, wobei jeder Ausgang durch eine Transition t_i, $i = 1, \ldots, n$ repräsentiert wird (vgl. Abb. 4.1(b)). Alle Transitionen t_i bilden zusammen einen Free-Choice-Konflikt. Jeder Transition t_i ist eine Wahrscheinlichkeit p_i zugeordnet, so daß $\sum_{i=1}^{n} p_i = 1$. Die Wahrscheinlichkeit p_i notieren wir graphisch an der Kante, die zu t_i führt.

(a) Einfacher Münzwurf

(b) Allgemeiner Münzwurf

Abb. 4.1: Modellierung eines Münzwurfs.

Wir ordnen nur Free-Choice-Konflikten Wahrscheinlichkeiten zu. Dies ist durch unsere Intuition eines Münzwurfes vorgegeben. In einem *Extended-Free-Choice-Konflikt* hat jede Transition dieselbe Menge von Vorbedingungen. Extended-Free-Choice-Konflikte können, wie in Abb. 4.2 dargestellt, zu Free-Choice-Konflikten verfeinert werden. Auf diese Weise können wir uns vorstellen, daß auch ein Extended-Free-Choice-Konflikt einen Münzwurf modelliert.

(a) (b)

Abb. 4.2: Verfeinerung eines Extended-Free-Choice-Konfliktes

Wir definieren jetzt den Begriff des randomisierten Netzsystems. Ein randomisiertes Netzsystem besteht aus einem Netzsystem, in dem einige Transitionen als *probabilistisch* ausgezeichnet sind, und aus einer Abbildung, die jeder probabilistischen Transition eine Wahrscheinlichkeit zuordnet. Eine probabilistische Transition hat genau eine Stelle in ihrem Vorbereich.

Definition 4.1 (Randomisiertes Netzsystem)

Ein *randomisiertes Netzsystem* $\dot{\Sigma} = (\Sigma, T^{flip}, \mu)$ besteht aus

1. einem Netzsystem Σ mit Transitionsmenge T und Menge der externen Transitionen T^{ext},
2. einer Menge $T^{flip} \subseteq (T \setminus T^{ext})$ ausgezeichneter interner Transitionen von Σ, so daß $t \in T^{flip} \Rightarrow |{}^\bullet t| = 1$ und so daß $({}^\bullet t)^\bullet \cap T^{flip}$ endlich ist, sowie
3. einer Abbildung $\mu : T^{flip} \to [0,1]$, die jedem $t \in T^{flip}$ eine reelle Zahl im Intervall $[0,1]$ zuordnet, so daß für alle $t \in T^{flip}$ gilt: $\mu(t) > 0$ und

$$\sum_{t' \in ({}^\bullet t)^\bullet \cap T^{flip}} \mu(t') = 1 \qquad (4.1)$$

Eine Transition aus T^{flip} heißt *probabilistisch*, die Abbildung μ heißt *Verteilung* von $\dot{\Sigma}$; $\dot{\Sigma}$ ist *beschränkt randomisiert*, falls eine Konstante $c > 0$ existiert, so daß $\forall t \in T^{flip} : \mu(t) > c$. ∘

Ein Konflikt eines randomisierten Netzsystems heißt *probabilistisch*, falls jede Transition des Konflikts probabilistisch ist. Nicht jeder Konflikt eines randomisierten

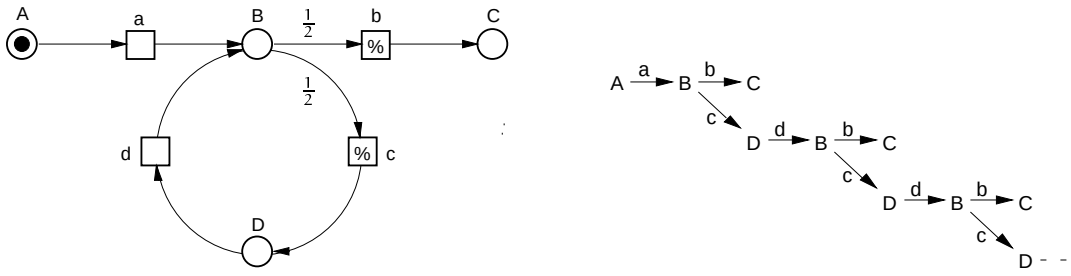
Netzsystems ist probabilistisch. Konflikte, die nicht probabilistisch sind, werden nichtdeterministisch gelöst. Nichtdeterminismus in randomisierten Netzsystem ist nötig, um den in randomisierten verteilten Algorithmen vorkommenden Nichtdeterminismus zu modellieren: Auch in randomisierten Algorithmen ist die zeitliche Reihenfolge unabhängiger Ereignisse sowie das Umgebungsverhalten unvorhersagbar. Nichtdeterminismus kann darüberhinaus Wahlfreiheit der Implementation modellieren.

Wir wenden uns nun der Semantik randomisierter Netzsysteme zu. Wir stellen zunächst die traditionelle sequentielle Semantik – *probabilistische Schaltbäume* – dar. Danach übertragen wir die in der sequentiellen Semantik verwendeten Konzepte auf nicht-sequentielle Abläufe. Schließlich werden wir beide Semantiken miteinander vergleichen.

4.1.3 Probabilistische Schaltbäume

In diesem Abschnitt erläutern wir die traditionelle sequentielle Semantik randomisierter verteilter Algorithmen, die wir von probabilistischen Programmen [71] auf randomisierte Netzsysteme übertragen.

Zunächst betrachten wir sequentielle randomisierte Netzsysteme ohne Nichtdeterminismus. Ein randomisiertes Netzsystem $\dot{\Sigma}$ heißt *probabilistisch* , falls in der maximalen Abwicklung von $\dot{\Sigma}$ kein externes Ereignis vorkommt und alle Konflikte probabilistisch sind. Hinreichend dafür ist $T^{ext} = \varnothing$ und für alle Transitionen t_1, t_2 von $\dot{\Sigma}$ gilt: ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \varnothing \Rightarrow t_1, t_2 \in T^{flip}$. Ein Netzsystem heißt *sequentiell*, falls es keine nebenläufigen Ereignisse in seiner maximalen Abwicklung gibt. Ist ein Netzsystem sequentiell, so repräsentiert jede Verzweigung seines maximalen Schaltbaums einen Konflikt.



(a) Σ_{18} (b) ϑ

Abb. 4.3: Ein randomisiertes Netzsystem und sein probabilistischer Schaltbaum.

Abb. 4.3(a) zeigt Σ_{18} – ein randomisiertes Netzsystem, das sowohl sequentiell als

auch probabilistisch ist. Von Σ_{18} erwarten wir, daß es die Eigenschaft $\Diamond\, C$ mit Wahrscheinlichkeit 1 erfüllt. Die genaue Bedeutung dessen wird traditionell (z.B. für probabilistische Transitionssysteme) wie folgt erklärt.

Dazu betrachten wir den maximalen Schaltbaum ϑ von Σ_{18} in Abb. 4.3(b). Da Σ_{18} sowohl sequentiell als auch probabilistisch ist, repräsentiert jede Verzweigung in ϑ einen Münzwurf, d.h. jedem Zweig ist durch μ eine bedingte Wahrscheinlichkeit zugeordnet, so daß die Summe aller bedingten Wahrscheinlichkeiten an jeder Verzweigung 1 ergibt. Einen Schaltbaum, bei dem jede Verzweigung einen Münzwurf repräsentiert, heißt *probabilistisch*. Einem probabilistischen Schaltbaum ϑ wird wie folgt ein Wahrscheinlichkeitsraum $(\Omega, \mathcal{A}, P)$ zugeordnet. Als Vorbereitung weisen wir jeder endlichen Schaltsequenz $\tau = M_0, t_1, \ldots, t_n, M_n$ eine Wahrscheinlichkeit $p(\tau)$ durch $p(\tau) = \prod_{i=1,\ldots,n} \mu(t_i)$ zu, wobei für $t_i \notin T^{\text{flip}}$ $\mu(t_i) = 1$ gesetzt wird. Für Σ_{18} ist $p(A, a, B, c, D, d, B, c, D) = \frac{1}{4}$.

Es sei $\Omega = \{\tau \mid \tau \text{ ist maximale Schaltsequenz von } \vartheta\}$. In dem zu konstruierenden Wahrscheinlichkeitsraum über Ω wird die Wahrscheinlichkeit $p(\tau)$ der Menge $K(\tau) = \{\tau' \in \Omega \mid \tau \text{ ist Präfix von } \tau'\}$ zugeordnet. Eine solche Menge $K(\tau)$ bezeichnen wir als *Kegel*. Die Menge aller Kegel $\mathcal{E} = \{K(\tau) \mid \tau \text{ ist endliche Schaltsequenz von } \vartheta\}$ bildet den Erzeuger der σ-Algebra $\mathcal{A}$ unseres Wahrscheinlichkeitsraumes, d.h. $\mathcal{A} = \sigma(\mathcal{E})$. Man kann nun zeigen, daß es einen eindeutigen Wahrscheinlichkeitsraum $(\Omega, \mathcal{A}, P)$ gibt, so daß

$$P(K(\tau)) = p(\tau) \tag{4.2}$$

für alle endliche Schaltsequenzen τ gilt. Man kann weiterhin zeigen, daß jede temporallogische Sequenzeigenschaft E in diesem Wahrscheinlichkeitsraum meßbar ist und man definiert dann: E *gilt mit Wahrscheinlichkeit* p, falls $P(E) = p$. Dann gilt $\Diamond\, C$ mit Wahrscheinlichkeit 1 tatsächlich in Σ_{18}.

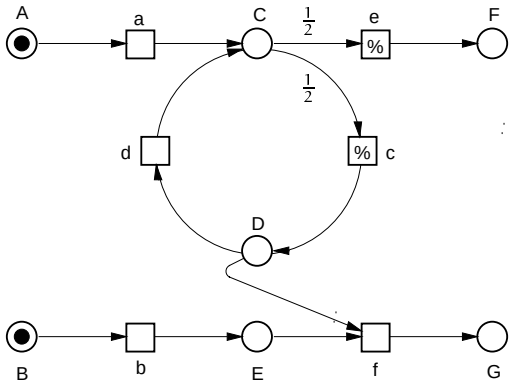


Abb. 4.4: Σ_{19}

Das randomisierte Netzsystem Σ_{19} in Abb. 4.4 ist weder probabilistisch noch sequentiell: Es besitzt sowohl einen nichtdeterministischen Konflikt als auch neben-

läufige Ereignisse. Daher ist der maximale Schaltbaum von Σ_{19} kein probabilistischer Schaltbaum. Wir können jedoch im maximalen Schaltbaum von Σ_{19} maximale probabilistische Schaltbäume auswählen. (Σ_{19} besitzt unendlich viele maximale Schaltbäume.) Die Auswahl eines maximalen probabilistischen Schaltbaums ϑ und damit eines Wahrscheinlichkeitsraumes $(\Omega_\vartheta, \mathcal{A}_\vartheta, P_\vartheta)$ ist nichtdeterministisch. Demzufolge definiert man für allgemeine randomisierte Netzsysteme $\dot{\Sigma}$: Eine temporallogische Schaltsequenzeigenschaft E *gilt in* $\dot{\Sigma}$ *mindestens mit Wahrscheinlichkeit* p, falls für jeden maximalen probabilistischen Schaltbaum ϑ von $\dot{\Sigma}$ gilt: $P_\vartheta(E) \geq p$. In Σ_{19} gilt $\Diamond F$ mindestens mit Wahrscheinlichkeit $\frac{1}{2}$ und $\Diamond F \vee G$ mindestens mit Wahrscheinlichkeit 1, d.h. Σ_{19} terminiert mit Wahrscheinlichkeit 1.

In der gerade definierten sequentiellen Semantik haben wir Maximalität von Schaltsequenzen als Lebendigkeitsannahme gefordert. Der nun folgenden nicht-sequentiellen Semantik legen wir Progreß zugrunde.

4.1.4 Probabilistische Abläufe

In diesem Abschnitt definieren wir in Analogie zu probabilistischen Schaltbäumen *probabilistische (nicht-sequentielle) Abläufe*. Ein probabilistischer Ablauf von $\dot{\Sigma}$ ist eine Abwicklung von $\dot{\Sigma}$, bei der alle Konflikte probabilistisch sind, d.h. in einem probabilistischen Ablauf kommt kein Nichtdeterminismus vor. Wir werden zeigen, daß es genau einen kanonischen Wahrscheinlichkeitsraum für jeden probabilistischen Ablauf gibt.

Definition 4.2 (Probabilistischer Ablauf)

Sei $\dot{\Sigma} = (\Sigma, T^{flip}, \mu)$ ein randomisiertes Netzsystem und π eine Abwicklung von Σ mit Ereignismenge E. Ein Ereignis $e \in E$ heißt *probabilistisch*, falls $\widetilde{e} \in T^{flip}$. Sei E^{flip} die Menge der probabilistischen Ereignisse von π. Dann heißt π *probabilistischer Ablauf* von $\dot{\Sigma}$, falls für jede Bedingung b von π gilt:

1. $b^\bullet = \varnothing$ oder
2. $b^\bullet = \{e\}$ und $e \in E \setminus E^{flip}$ oder
3. $b^\bullet \subseteq E^{flip}$ und $\sum_{e \in b^\bullet} \mu(\widetilde{e}) = 1$. ∘

Da jeder Münzwurf nur endlich viele Ausgänge hat (Def. 4.1 Punkt 2), ist jeder probabilistische Ablauf endlich verzweigt. Abb. 4.5 zeigt einen endlichen, maximalen probabilistischen Ablauf von Σ_{19}.

Wir werden jedem probabilistischen Ablauf eines randomisierten Netzsystems einen Wahrscheinlichkeitsraum zuordnen, der sich aus der Verteilung des randomisierten Netzsystems ergibt. Um diesen Bezug herzustellen, definieren wir nun für einen

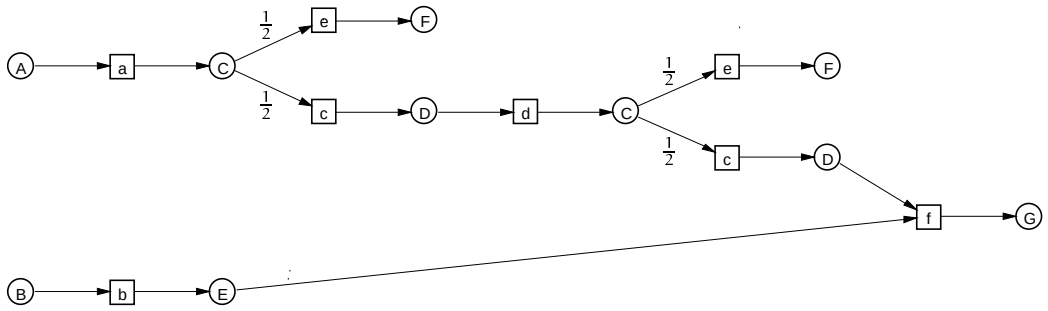


Abb. 4.5: Ein endlicher, maximaler probabilistischer Ablauf von Σ_{19}.

probabilistischen Ablauf π die Wahrscheinlichkeit $p(\alpha)$, mit der ein endlicher Ablauf α von π realisiert wird; $p(\alpha)$ definieren wir wie in der sequentiellen Semantik als Produkt der Wahrscheinlichkeiten der in α schaltenden Münzwurftransitionen.

Definition 4.3 (Wahrscheinlichkeit von endlichen Abläufen)

Sei $\dot{\Sigma}$ ein randomisiertes Netzsystem und π ein probabilistischer Ablauf von $\dot{\Sigma}$. Zu jedem endlichen Ablauf α von π, dessen Menge probabilistischer Ereignisse wir mit E_α^{flip} bezeichnen, definieren wir seine Wahrscheinlichkeit $p(\alpha)$ durch $p(\alpha) = 1$, falls $E_\alpha^{flip} = \varnothing$ und

$$p(\alpha) = \prod_{e \in E_\alpha^{flip}} \mu(\widetilde{e}) \tag{4.3}$$

sonst. ○

Mit Definition 4.3 nehmen wir die stochastische Unabhängigkeit aller Münzwürfe an. Wir zeigen nun die eindeutige Existenz eines kanonischen Wahrscheinlichkeitsraumes zu jedem probabilistischem Ablauf.

Satz 4.4 (Wahrscheinlichkeitsraum eines probabilistischen Ablaufs)

Sei $\dot{\Sigma}$ ein randomisiertes Netzsystem und π ein probabilistischer Ablauf von $\dot{\Sigma}$. Sei $\Omega = \mathfrak{R}_{max}(\pi)$ und zu jedem endlichen Ablauf α von π sei $K(\alpha) = \{\rho \in \mathfrak{R}_{max}(\pi) \mid \alpha \sqsubseteq \rho\}$ die Menge aller maximalen Abläufe von π, die α fortsetzen. Desweiteren sei $\mathcal{E} = \{K(\alpha) \mid \alpha \in \mathfrak{R}_{fin}(\pi)\}$. Dann gibt es genau einen Wahrscheinlichkeitsraum $(\Omega, \mathcal{A}, P)$, so daß $\mathcal{A} = \sigma(\mathcal{E})$ und daß für alle endlichen Abläufe α von π gilt:

$$P(K(\alpha)) = p(\alpha) \tag{4.4}$$

Beweis: Den Beweis von Satz 4.4 führen wir im Anhang A. Wir geben hier eine Zusammenfassung des Beweises. Eine Menge $K(\alpha)$ für einen endlichen Ablauf α von π bezeichnen wir dabei als *Kegel*. Zu konstruieren ist ein Wahrscheinlichkeitsmaß

P auf $\sigma(\mathcal{E})$, das bereits durch (4.4) auf $\mathcal{E}$, d.h. auf allen Kegeln definiert ist. Dafür muß gezeigt werden, daß P auf Kegelkomplemente sowie auf Vereinigungen von Kegeln und Kegelkomplementen fortsetzbar ist. Zentraler Schritt des Beweises ist die Angabe einer *Mengenalgebra* über Ω, die $\mathcal{E}$ enthält und $\sigma(\mathcal{E})$ erzeugt. Eine *Mengenalgebra* über Ω ist ein Mengensystem $\mathcal{M} \subseteq 2^{\Omega}$, das unter Komplement und endlicher Vereinigung abgeschlossen ist und für das $\Omega \in \mathcal{M}$ gilt[2]. Die gesuchte Mengenalgebra enthält gerade alle endlichen Vereinigungen paarweise disjunkter Kegel. Auf dieser Mengenalgebra kann man durch

$$P(\bigcup_{i=1}^{n} A_i) = \sum_{i=1}^{n} P(A_i) \qquad A_i \in \mathcal{E} \text{ paarweise disjunkt}$$

P auf $\mathcal{M}$ fortsetzen. Dies zu zeigen, verlangt den größten Aufwand innerhalb des Beweises. Die weitere Fortsetzung von P auf die σ-Algebra $\sigma(\mathcal{M}) = \sigma(\mathcal{E})$ ist dann eine Anwendung des *Fortsetzungssatzes*, eines Standardsatzes der Maßtheorie. Die Eindeutigkeit des Wahrscheinlichkeitsraumes ergibt sich aus seiner Konstruktion. □

Definition 4.5 (Wahrscheinlichkeitsraum eines probabilistischen Ablaufs)

Sei $\dot{\Sigma}$ ein randomisiertes Netzsystem und π ein probabilistischer Ablauf von $\dot{\Sigma}$. Den in Theorem 4.4 eindeutig bestimmten *Wahrscheinlichkeitsraum von* π bezeichnen wir durch $(\Omega_\pi, \mathcal{A}_\pi, P_\pi)$. ○

Mit Hilfe des Wahrscheinlichkeitsraumes eines probabilistischen Ablaufs erklären wir nun im folgenden Abschnitt die probabilistische Gültigkeit von Ablaufeigenschaften.

4.1.5 Probabilistische Gültigkeit von Ablaufeigenschaften

In diesem Abschnitt definieren wir, wann eine Ablaufeigenschaft mindestens mit Wahrscheinlichkeit p in einem randomisierten Netzsystem gilt. Dabei schränken wir uns auf *meßbare Ablaufeigenschaften* ein – das sind Ablaufeigenschaften, die in dem Wahrscheinlichkeitsraum jedes probabilistischen Ablaufs meßbar sind. Wir zeigen, daß jede temporallogische Eigenschaft eine meßbare Ablaufeigenschaft ist.

Definition 4.6 (Meßbare Ablaufeigenschaft)

Sei $\dot{\Sigma}$ ein randomisiertes Netzsystem und π ein probabilistischer Ablauf von $\dot{\Sigma}$. Eine Ablaufeigenschaft E heißt *meßbar* in π, falls $(E \cap \mathfrak{R}(\pi)) \in \mathcal{A}_\pi$. Eine Ablaufeigenschaft E heißt *meßbar* in $\dot{\Sigma}$, falls E in jedem probabilistischen Ablauf von $\dot{\Sigma}$ meßbar ist. Für eine in π meßbare Eigenschaft E schreiben wir $P_\pi(E)$ anstelle von $P_\pi(\mathfrak{R}(\pi) \cap E)$. ○

[2] Wir erinnern uns, daß eine σ-Algebra darüberhinaus auch unter abzählbarem Durchschnitt abgeschlossen ist.

Proposition 4.7 (Meßbarkeit temporallogischer Eigenschaften)

Jede temporallogische Eigenschaft ist in jedem randomisierten Netzsystem meßbar.

Beweis: Sei $\dot{\Sigma}$ ein randomisiertes Netzsystem und π ein probabilistischer Ablauf von $\dot{\Sigma}$ mit Wahrscheinlichkeitsraum $(\Omega, \mathcal{A}, P)$. Sei Φ eine Temporalformel. Dann ist für alle endlichen Abläufe α von π die Menge $\{\rho \in \Omega \mid \alpha \sqsubseteq \rho \text{ und } \rho, \alpha \models \Phi\}$ meßbar. Dies beweisen wir induktiv über den Aufbau von Temporalformeln:

1. Ist $\Phi = \varphi$ eine Zustandsformel, so ist $\{\rho \in \Omega \mid \alpha \sqsubseteq \rho \wedge \rho, \alpha \models \varphi\}$ gleich $K(\alpha)$ falls $\widetilde{\alpha^\circ} \models \varphi$ und $\varnothing$ sonst, und damit meßbar in π.
2. Boolesche Kombinationen sind wegen der Definition der σ-Algebra meßbar.
3. Sei $\{\rho \in \Omega \mid \alpha \sqsubseteq \rho \wedge \rho, \alpha \models \Phi\}$ für alle α meßbar in π. Dann ist auch $\{\rho \in \Omega \mid \alpha \sqsubseteq \rho \wedge \rho, \alpha \models \Diamond \Phi\} = \bigcup_{\alpha \sqsubseteq \beta} \{\rho \in \Omega \mid \beta \sqsubseteq \rho \wedge \rho, \beta \models \Phi\}$ meßbar, da es nur abzählbar viele endliche Abläufe von π gibt.
4. Die übrigen temporalen Operatoren ergeben sich aus 3. □

Wir definieren nun die probabilistische Gültigkeit von Ablaufeigenschaften.

Definition 4.8 (Probabilistische Gültigkeit)

Sei $\dot{\Sigma}$ ein randomisiertes Netzsystem und E eine in $\dot{\Sigma}$ meßbare Ablaufeigenschaft; E *gilt mindestens mit Wahrscheinlichkeit* p, falls für jeden progressiven probabilistischen Ablauf π von $\dot{\Sigma}$ gilt:

$$P_\pi(E) \geq p \tag{4.5}$$

E ist *probabilistisch gültig* in $\dot{\Sigma}$ (Notation: $\dot{\Sigma} \Vdash E$), falls E in $\dot{\Sigma}$ mindestens mit Wahrscheinlichkeit 1 gilt. ○

Wir halten nun eine notwendige Bedingung für die probabilistische Gültigkeit einer Ablaufeigenschaft fest.

Lemma 4.9

Sei $\dot{\Sigma}$ ein randomisiertes Netzsystem und E eine in $\dot{\Sigma}$ meßbare Ablaufeigenschaft, die in $\dot{\Sigma}$ probabilistisch gültig ist. Dann ist E in jedem probabilistischen Ablauf π von $\dot{\Sigma}$ lebendig.

Beweis: Sei E nicht lebendig in π. Dann gibt es einen endlichen Ablauf α von π mit $K(\alpha) \cap E = \varnothing$. Dann gilt $P_\pi(E) \leq 1 - P_\pi(K(\alpha))$. Da $P_\pi(K(\alpha)) > 0$ gilt (weil $\mu(t) > 0$ für alle t), ist E nicht probabilistisch gültig. □

Folgerung 4.10

Sei $\dot{\Sigma}$ ein randomisiertes Netzsystem und E eine in $\dot{\Sigma}$ meßbare Sicherheitseigenschaft. Dann gilt: E ist in $\dot{\Sigma}$ probabilistisch gültig, genau dann wenn jeder Ablauf von $\dot{\Sigma}$ die Eigenschaft E erfüllt.

Beweis: Die Richtung $\Rightarrow$ ist trivial, die Richtung $\Leftarrow$ folgt aus Proposition 2.6 2. □

Die Umkehrung von Lemma 4.9 gilt im allgemeinen nicht. Dazu betrachten wir das sequentielle randomisierte Netzsystem Σ_{20} in Abb. 4.6. Σ_{20} hat genau einen probabilistischen Ablauf π. Aus der Random-Walk-Theorie wissen wir (siehe z.B. [34]), daß der Zustand s' genau dann mit Wahrscheinlichkeit 1 erreicht wird, falls $p \leq \frac{1}{2}$. Die Eigenschaft $\Diamond s'$ ist aber lebendig in π – unabhängig von p, insbesondere für $p > \frac{1}{2}$.

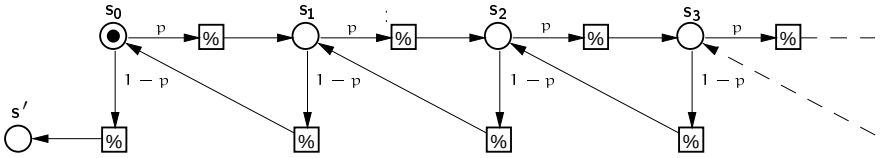


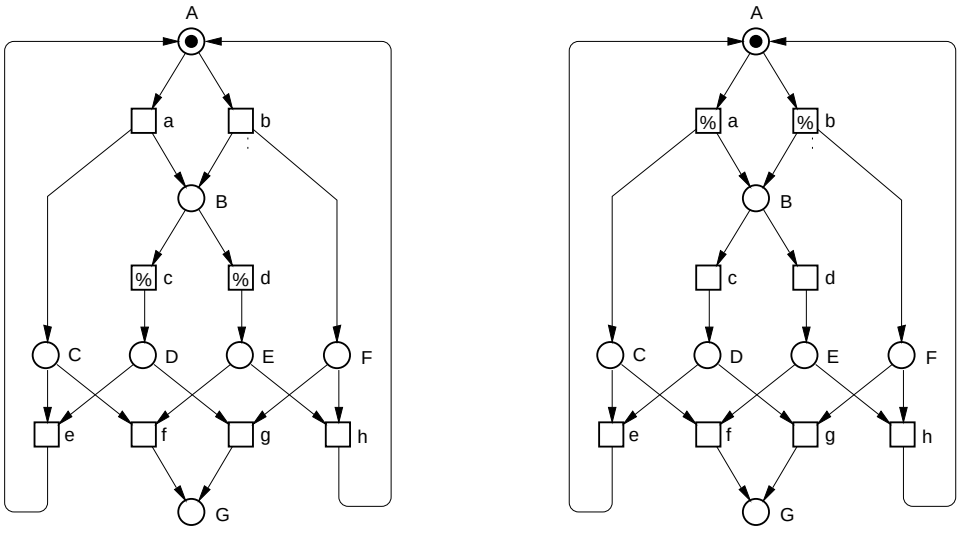
Abb. 4.6: Σ_{20}

4.1.6 Beispiele

In diesem Abschnitt betrachten wir einige Beispiele für randomisierte Netzsysteme, um zu illustrieren, was Randomisierung leistet. Wir lassen dabei im folgenden in der graphischen Darstellung randomisierter Netzsysteme die Anschrift konkreter Wahrscheinlichkeiten weg, da diese keine Rolle mehr spielen. Wichtig ist nur noch, ob ein Konflikt durch Münzwurf oder nichtdeterministisch entschieden wird.

In jedem der randomisierten Netzsysteme Σ_{21} und Σ_{22}, dargestellt in Abb. 4.7, werden nacheinander zwei Free-Choice-Konflikte gelöst. In Abhängigkeit von den Konfliktentscheidungen ist dann genau eine der vier Transitionen e bis h aktiviert. Haben nacheinander a und d bzw. b und c stattgefunden, so schaltet f bzw. g und der Ablauf ist beendet. Haben nacheinander a und c bzw. b und d stattgefunden, so schaltet e bzw. h und der Anfangszustand ist wieder hergestellt.

In Σ_{21} wird der Konflikt zwischen c und d durch Münzwurf entschieden. Ob Transition c oder d schaltet, hängt damit nicht davon ab, ob vorher a oder b geschaltet hat; Σ_{21} terminiert mit Wahrscheinlichkeit 1 ($\Diamond G$ ist in Σ_{21} probabilistisch gültig). Dies zeigt die Stärke von Münzwurf gegenüber Fairneß: Fordern wir in Σ_{21} anstelle der Konfliktauflösung durch Münzwurf die faire Auflösung aller Konflikte, so ist keine Termination garantiert: Die Schaltsequenz $(a, c, e, b, d, h)^\infty$ ist fair bzgl. aller Transitionen. Randomisierung leistet in Σ_{21} also mehr als Fairneß.



(a) Σ_{21} (b) Σ_{22}

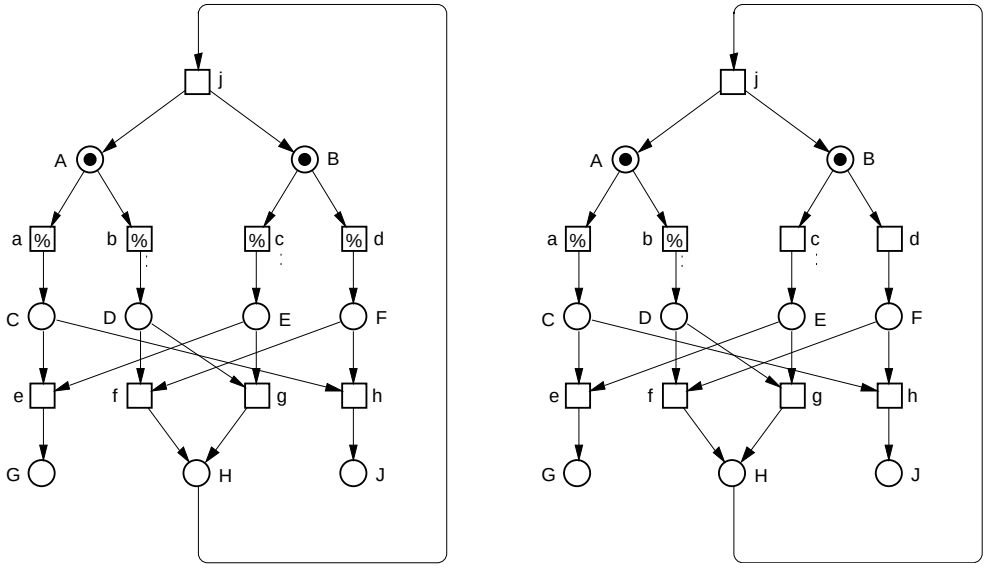
Abb. 4.7: Zwei randomisierte Netzsysteme.

Betrachten wir nun Σ_{22} . Hier wird nicht der Konflikt zwischen c und d, sondern der Konflikt zwischen a und b durch Münzwurf entschieden. Dabei kann die nichtdeterministische Entscheidung zwischen c und d davon abhängen, ob a oder b vorher stattgefunden hat: Es gibt einen progressiven probabilistischen Ablauf π von Σ_{22} , in dem auf jedes Schalten von a das Schalten von c und auf jedes Schalten von b das Schalten von d folgt. In π ist kein Ablauf endlich, und damit terminiert Σ_{22} nicht mit Wahrscheinlichkeit 1 ($\Diamond\, G$ ist in Σ_{22} nicht probabilistisch gültig).

Σ_{23} und Σ_{24} in Abb. 4.8 sind ähnlich zu den randomisierten Netzsystemen in Abb. 4.7. Wieder werden jeweils zwei Free-Choice-Konflikte gelöst, diesmal jedoch nicht nacheinander sondern nebenläufig zueinander.

In Σ_{23} werden beide Konflikte durch Münzwurf entschieden. Σ_{23} terminiert mit Wahrscheinlichkeit 1. Dies ist ein wichtiges Beispiel für Randomisierung. Beide Entscheidungen werden mit Wahrscheinlichkeit 1 irgendwann *koordiniert* zueinander getroffen, ohne daß beide Entscheidungen synchronisiert werden. Solche wiederholte, nebenläufige Münzwurfe treten bei allen am Anfang des Kapitels zitierten randomisierten Algorithmen auf, die Probleme lösen, die durch herkömmliche, nichtdeterministische Algorithmen nicht gelöst werden können.

Betrachten wir nun Σ_{24} . Hier wird nur einer der beiden Konflikte durch Münzwurf gelöst. Auch Σ_{24} terminiert mit Wahrscheinlichkeit 1, jedoch nur in der nicht-sequentiellen Semantik. In der sequentiellen Semantik terminiert Σ_{24} (im Gegensatz zu Σ_{23})



(a) Σ_{23} (b) Σ_{24}

Abb. 4.8: Koordinierung nebenläufiger Entscheidungen.

nicht mit Wahrscheinlichkeit 1: Es gibt einen probabilistischen Schaltbaum von Σ_{24}, in dem keine Schaltsequenz endlich ist. Warum, sehen wir im nächsten Abschnitt.

4.1.7 Vergleich von sequentieller und nicht-sequentieller Semantik

In diesem Abschnitt untersuchen wir den Unterschied zwischen sequentieller und nicht-sequentieller Semantik randomisierter Netzsysteme.

Von der vorgestellten sequentiellen Semantik randomisierter Algorithmen gibt es in der Literatur verschiedene Abstufungen. Die Unterschiede verschiedener Abstufungen werden dabei mit Hilfe des Begriffs des *Gegenspielers* (*adversary*, manchmal: *scheduler*) beschrieben. Dabei stellen wir uns vor, daß alle probabilistischen Konflikte von einem virtuellen *Münzwerfer* (durch Münzwurf) und alle nichtdeterministischen Konflikte von einem virtuellen Gegenspieler entschieden werden. Jede Schaltsequenz kann dann als Resultat der Interaktion von Münzwerfer und Gegenspieler aufgefaßt werden: In jedem Zustand entscheidet der Gegenspieler, welcher Konflikt als nächstes aufgelöst werden soll. Ist der Konflikt probabilistisch, so entscheidet der Münzwerfer darüber, wie der Konflikt aufgelöst wird; ist der Konflikt hingegen nichtdeterministisch, so entscheidet der Gegenspieler, wie der Konflikt aufgelöst wird. Das Verhalten eines Gegenspielers kann als Funktion beschrieben werden, die jeder endlichen Schaltsequenz $\sigma = M_0, \ldots, M_n$ entweder einen in M_n aktivierten

probabilistischen Konflikt oder aber eine in M_n aktivierte nicht-probabilistische Transition zuordnet. Jedes Verhalten des Gegenspielers erzeugt genau einen probabilistischen Schaltbaum.

Verschiedene Abstufungen der sequentiellen Semantik schränken das Verhalten des Gegenspielers ein. Dabei unterscheiden wir zwei Arten von Einschränkungen (vgl. [86]): *ablaufbasierte Gegenspieler* und *Gegenspieler mit partieller on-line-Information*. Bei einem *ablaufbasierten Gegenspieler* werden die Schaltsequenzen eingeschränkt, die ein Gegenspieler erzeugen kann; z.B. kann man verlangen, daß der Gegenspieler alle nichtdeterministischen Konflikte fair auflöst. Bei einem *Gegenspieler mit partieller on-line-Information* wird das Wissen des Gegenspielers, anhand dessen er Entscheidungen trifft, eingeschränkt; z.B. kann man fordern, daß der Gegenspieler den Ausgang früherer Münzwürfe nicht kennt – ein solcher Gegenspieler heißt *partiell vergeßlicher Gegenspieler*. Ein partiell vergeßlicher Gegenspieler trifft für zwei endliche Schaltsequenzen, die sich nur im Ausgang von Münzwürfen unterscheiden, dieselbe Entscheidung. Unter einem partiell vergeßlichen Gegenspieler terminiert das randomisierte Netzsystem in Abb. 4.7(b) mit Wahrscheinlichkeit 1.

Ist das Verhalten des Gegenspielers nicht eingeschränkt, so sprechen wir von einem *allgemeinen (sequentiellen) Gegenspieler*. Abgeschwächte Gegenspieler haben gegenüber dem allgemeinen Gegenspieler den Vorteil, daß sie einfachere und effizientere Algorithmen erlauben. Algorithmen, die unter einem abgeschwächten Gegenspieler korrekt sind, sind dies nicht notwendig unter dem allgemeinen Gegenspieler. Zum Beispiel verwendet Rabin in [74] einen abgeschwächten Gegenspieler für einen randomisierten Synchronisationsalgorithmus. Hart, Sharir und Pnueli zeigen in [40], daß der Algorithmus von Rabin unter dem allgemeinen Gegenspieler seine Ziele nicht mehr erreicht. Andererseits gilt ein Unmöglichkeitsresultat unter einem abgeschwächten Gegenspieler auch unter dem allgemeinen Gegenspieler.

Die in den vorigen Abschnitten entwickelte nicht-sequentielle Semantik randomisierter Netzsysteme können wir uns im Vergeich zur sequentiellen Semantik durch einen Gegenspieler mit partieller on-line-Information vorstellen. Diesem Gegenspieler, wir nennen ihn den *nicht-sequentiellen Gegenspieler*, fehlt gegenüber dem allgemeinen sequentiellen Gegenspieler Wissen über die zeitliche Reihenfolge unabhängiger Ereignisse. Insbesondere kann der nicht-sequentielle Gegenspieler im Gegensatz zum allgemeinen sequentiellen Gegenspieler die Auflösung nichtdeterministischer Konflikte nicht vom Ausgang nebenläufiger Münzwürfe abhängig machen. Um dies zu sehen, betrachten wir Abb. 4.9. Die Abbildung zeigt das randomisierte Netzsystem Σ_{25}, seine beiden probabilistischen Abläufe π_1 und π_2 sowie seine sechs probabilistischen Schaltbäume ϑ_1 bis ϑ_6. Aus jedem probabilistischen Ablauf kann durch Sequentialisierung eine Menge probabilistischer Schaltbäume gewonnen werden. Aus π_1 gewinnt man ϑ_1 und ϑ_2 und aus π_2 gewinnt man ϑ_3 und ϑ_4. Σ_{25} zeigt nun, daß nicht jeder probabilistische Schaltbaum aus einem probabilistischen Ablauf gewon-

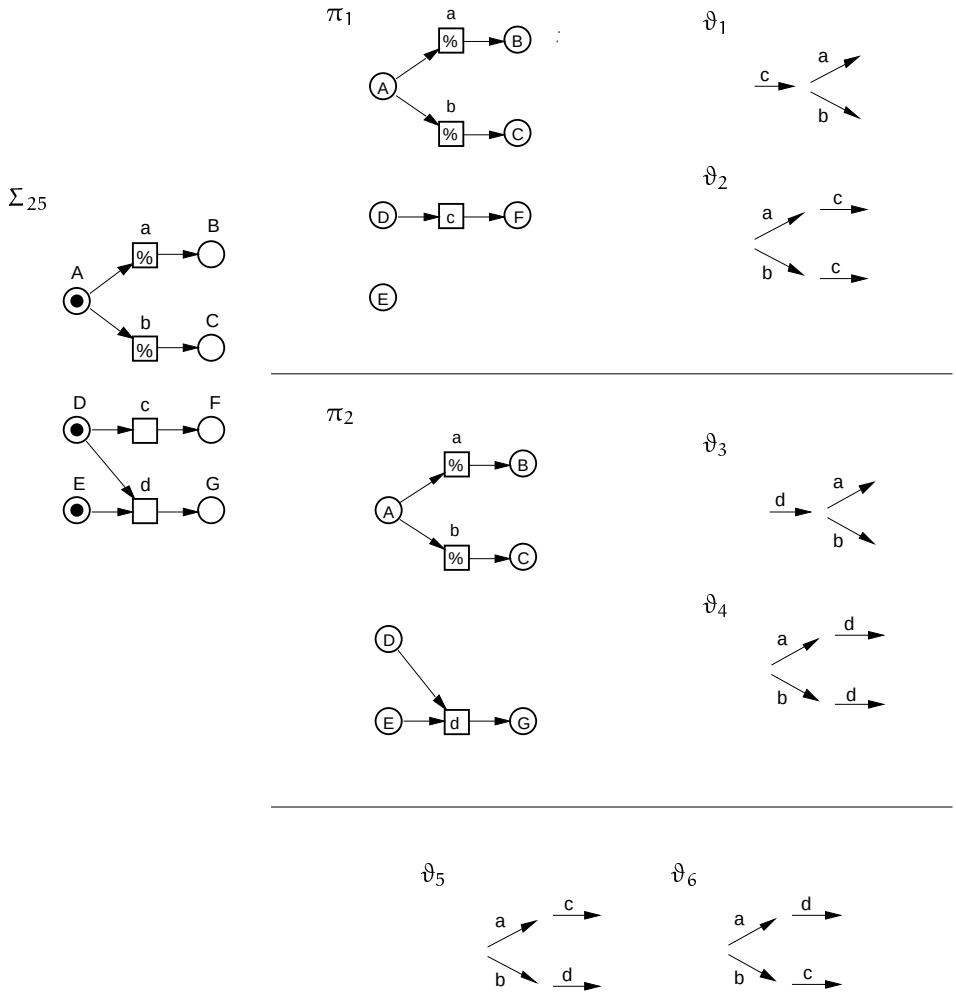


Abb. 4.9: Vergleich von sequentieller und nicht-sequentieller Semantik.

nen werden kann: ϑ_5 und ϑ_6 können keinem probabilistischen Ablauf zugeordnet werden; ϑ_5 und ϑ_6 repräsentieren ein Verhalten des Gegenspielers, bei dem Entscheidungen des Gegenspielers vom Ausgang nebenläufiger Münzwürfe abhängen. Auf diese Weise kann auch der Gegenspieler beim randomisierten Netzsystem in Abb. 4.8(b) handeln und immer dann c schalten lassen, wenn nebenläufig b schaltet und immer dann d schalten lassen, wenn nebenläufig a schaltet. So entsteht ein probabilistischer Schaltbaum, in dem keine Schaltsequenz endlich ist. Σ_{24} terminiert daher unter dem allgemeinen sequentiellen Gegenspieler nicht mit Wahrscheinlichkeit 1.

Jeder der probabilistischen Schaltbäume ϑ_5 und ϑ_6 repräsentiert eine probabilistische Entscheidung, wo keine ist, nämlich zwischen den Transitionen c und d. Dies drückt sich dadurch aus, daß in den zugehörigen Wahrscheinlichkeitsräumen

von ϑ_5 und ϑ_6 kausal unabhängige Entscheidungen nicht stochastisch unabhängig sind: Bei Gleichverteilung in Σ_{25} gilt für das Wahrscheinlichkeitsmaß P von ϑ_5: $P(a \text{ schaltet}) \cdot P(d \text{ schaltet}) = \frac{1}{4} \neq P(a \text{ und } d \text{ schalten}) = 0$. Diese Betrachtung wirft die Frage auf, wann welche Semantik adäquat ist. Diese Frage wollen wir hier nicht versuchen zu beantworten, wir wollen jedoch festhalten, daß es durch die neue nicht-sequentielle Semantik möglich wird, den nicht-sequentiellen Gegenspieler zu modellieren und dadurch die Unabhängigkeit verschiedener Systemkomponenten in randomisierten Algorithmen auszunutzen[3]. Desweiteren halten wir fest, daß der allgemeine Gegenspieler leicht in der nicht-sequentiellen Semantik durch Sequentialisierung des Systems simuliert werden kann. Eine umgekehrte Simulation des nicht-sequentiellen Gegenspielers ist schwieriger, da dafür in Schaltbäumen Unabhängigkeit kodiert werden muß. Für diese Arbeit ist lediglich relevant, daß sich alle Unmöglichkeitsresultate in dieser Arbeit von nicht-sequentieller Semantik auf sequentielle Semantik übertragen.

Wir wollen an dieser Stelle noch anmerken, daß *stochastische Petrinetze* [64] zur Modellierung randomisierter Algorithmen ungeeignet sind. Sie enthalten keinen echten Nichtdeterminismus, insbesondere wird die Reihenfolge kausal unabhängiger Ereignisse probabilistisch bestimmt (stochastische Petrinetze verwenden einen expliziten Zeitbegriff).

4.1.8 Extreme Fairneß

In diesem Abschnitt wollen wir mit Hilfe des Begriffs der *extremen Fairneß* (Pnueli [69]) die Ausdrucksstärke von Randomisierung weiter verdeutlichen.

Starke Fairneß verlangt, daß ein Konflikt bzgl. einer Transition fair aufgelöst wird. *Zustandsfairneß*[4] [72] verlangt, daß ein Konflikt bzgl. einer Markierung und einer Transition aufgelöst wird: Wird ein Konflikt immer wieder von einer Markierung aus aufgelöst, dann wird der Konflikt von dieser Markierung aus zugunsten jeder Transition aufgelöst. Extreme Fairneß verallgemeinert Zustandsfairneß durch Betrachtung von Zustandsformeln. Sei φ eine Zustandsformel und t eine Transition. Eine Markierung M heißt φ-Markierung, falls $M \models \varphi$. Eine Schaltsequenz σ ist *extrem fair* bzgl. φ und t, falls gilt: Wird in σ immer wieder ein Konflikt zu t von einer φ-Markierung aus aufgelöst, dann wird in σ immer wieder dieser Konflikt von einer φ-Markierung aus zugunsten von t aufgelöst. Wir formalisieren:

[3] In [3] wird zum Beispiel nach einem Modell gesucht, bei dem kausal unabhängige Ereignisse auch stochastisch unabhängig sind – dort geht es allerdings nicht um randomisierte Algorithmen in unserem Sinne.

[4] in [72]: fair choice from states

Definition 4.11 (Extreme Fairneß)

Sei $\dot{\Sigma}$ ein randomisiertes Netzsystem mit Stellenmenge P, φ eine Zustandsformel über P und t eine probabilistische Transition von $\dot{\Sigma}$. Eine Schaltsequenz σ von $\dot{\Sigma}$ ist nicht *extrem fair bzgl.* φ *und* t, falls t höchstens endlich oft in σ schaltet und für unendlich viele Positionen i von σ gilt $M_i \models \varphi$ und $t_{i+1} \in ({}^\bullet t)^\bullet \cap T^{flip}$. Eine Schaltsequenz σ ist *extrem fair*, falls sie bzgl. jeder Zustandsformel $\varphi \in ZF(P)$ und jeder probabilistischen Transition von $\dot{\Sigma}$ extrem fair ist. ∘

Pnueli zeigt in [69], daß die Menge aller extrem fairen Schaltsequenzen im Wahrscheinlichkeitsraum jedes probabilistischen Schaltbaums Wahrscheinlichkeit 1 hat. Daraus folgt:

> Gilt eine Schaltsequenzeigenschaft E in jeder extrem fairen Schaltsequenz von $\dot{\Sigma}$, dann gilt E mindestens mit Wahrscheinlichkeit 1 in $\dot{\Sigma}$. (4.6)

Auf diese Weise kann man Eigenschaften, die mindestens mit Wahrscheinlichkeit 1 gelten, auf klassische Weise verifizieren – ohne Techniken der Wahrscheinlichkeitsrechnung zu verwenden. Möchte man z.B. im randomisierten Netzsystem in Abb. 4.7(a) auf Seite 88 die probabilistische Gültigkeit von $\Diamond\, G$ zeigen, so geht man wie folgt vor. Zunächst zeigt man mit Hilfe von Invarianten, daß in jedem Ablauf, in dem nicht $\Diamond\, G$ gilt, stattdessen $(\Box \Diamond\, C \wedge B) \vee (\Box \Diamond\, F \wedge B)$ gilt. Ist ρ extrem fair, so ist ρ sowohl bzgl. $\varphi = C \wedge B$ und d als auch bzgl. $\varphi = F \wedge B$ und c extrem fair. Dann schaltet entweder d in $C \wedge B$ oder c in $F \wedge B$, wodurch entweder g oder f schaltet, womit wir $\Diamond\, G$ erhalten. Jeder extrem faire Ablauf erfüllt also $\Diamond\, G$, womit $\Diamond\, G$ probabilistisch gültig ist.

Die Umkehrung von (4.6) gilt im allgemeinen nicht, d.h. extreme Fairneß ist nicht *vollständig* bzgl. der Verifikation von Schaltsequenzeigenschaften, die mit Wahrscheinlichkeit 1 gelten. Daher stellt sich die Frage nach stärkeren Fairneßbegriffen, die (4.6) erfüllen. Lichtenstein, Pnueli und Zuck stellen in [60] den Begriff der α-*Fairneß* vor. Sie zeigen in [60, 71], daß in endlichen randomisierten Systemen für α-Fairneß sowohl (4.6) als auch die Umkehrung von (4.6) gilt; α-Fairneß fordert die faire Auflösung von probabilistischen Konflikten bzgl. *Vergangenheitsformeln* und Transitionen. Eine *Vergangenheitsformel* beschreibt eine Menge H endlicher Schaltsequenzen. Eine Vergangenheitsformel ist in einer Position i einer Schaltsequenz gültig, falls das i-te Präfix zu H gehört. Die Definition von α-Fairneß erhält man, indem man in Definition 4.11 „Zustandsformel“ durch „Vergangenheitsformel“ ersetzt.

Baier und Kwiatkowska verallgemeinern α-Fairneß in [11]. Sie beschriften sowohl Präfixe als auch probabilistische Transitionen mit *Labeln* einer abzählbaren Menge und fordern die faire Behandlung jedes Labels. Der Fairneßbegriff aus [11] erfüllt (4.6) in jedem beschränkt randomisierten Netzsystem (vgl. Definition 4.1).

Jaeger definiert in [43] den Begriff *computable fairness*. Eine Schaltsequenz σ ist nicht *computable fair*, falls es eine berechenbare Funktion gibt, die den Ausgang der Münzwürfe in σ vorhersagt. Dieser Fairneßbegriff ist stärker als α-Fairneß und erfüllt (4.6).

Die in diesem Abschnitt vorgestellten Fairneßbegriffe haben die folgende allgemeine Form:

> Wird immer wieder eine Münze in einem bestimmten Kontext geworfen, dann gibt es immer wieder jeden Münzwurfausgang in diesem Kontext. (4.7)

Randomisierung garantiert also, daß ein Free-Choice-Konflikt irgendwann koordiniert zu einem Kontext gelöst wird.

Extreme Fairneß genügt für die Verifikation vieler Algorithmen. Rao gibt in [75] einen Beweiskalkül für extreme Fairneß unter UNITY [24] an. Extreme Fairneß läßt sich leicht auf probabilistische Abläufe übertragen.

4.2 Konsens in randomisierten Netzsystemen

In diesem Abschnitt stellen wir den randomisierten Algorithmus von Ben-Or [14] vor, der asynchron Konsens mit Wahrscheinlichkeit 1 löst. Wir modellieren den Algorithmus von Ben-Or als algebraisches Netz und zeigen so, daß es ein randomisiertes Netzsystem gibt, das das Konsensproblem ausfalltolerant löst.

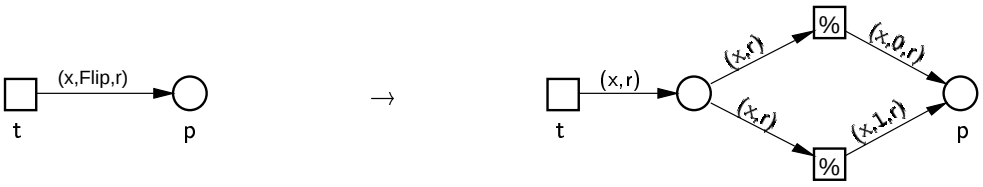


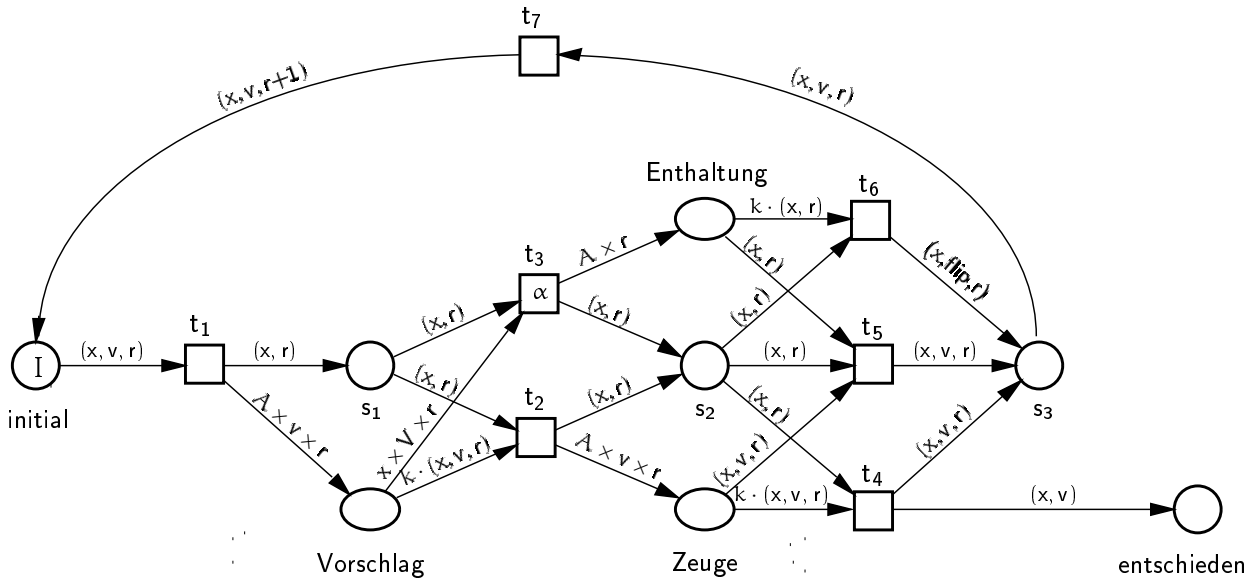
Abb. 4.10: Entfaltung einer flip-Kante.

Das algebraische Netzsystem Σ_{26} in Abb. 4.11 stellt den Algorithmus dar. Die Kante von t_6 zu s_3 modelliert einen Münzwurf mit zwei Ausgängen, was durch die Inschrift *flip* angezeigt wird. Abb. 4.10 zeigt, wie wir eine mit *flip* beschriftete Kante verstehen wollen. Mit Hilfe der Konstruktion in Abb. 4.10 können wir Σ_{26} zu einem randomisierten Netzsystem entfalten[5]. Wir erklären nun den Algorithmus anhand von Σ_{26}.

Sei $k \in \mathbb{N}$, $k > 0$ und $A = \{x_1, \ldots, x_{2k-1}\}$ die Menge der Agenten (der Einfachheit halber ungeradzahlig viele). Wir nehmen an, daß höchstens $k-1$ Agenten ausfallen. Der Algorithmus verläuft in Runden: Jeder Agent durchläuft zyklisch die Zustände *initial*, *s1*, *s2* und *s3*. Dabei hält jeder Agent einen Rundenzähler, der mit 0 initialisiert ist und beim Wechsel vom Zustand *s3* zum Zustand *initial* erhöht wird. Jeder Nachricht, die ein Agent im Algorithmus versendet, ist die aktuelle Rundennummer r dieses Agenten beigefügt. Am Anfang des Algorithmus ist jeder Agent x mit seinem Anfangswert $i(x)$ und seinem Rundenzähler r im Zustand *initial* – dies wird durch eine Marke $(x, i(x), 0)$ auf der Stelle *initial* modelliert. Wir erklären nun die einzelnen Aktionen eines Agenten x (vgl. auch die in Abschnitt 1.4 eingeführten Konventionen).

- t_1 Agent x sendet jedem Agenten (inklusive sich selbst) einen *Vorschlag*; am Anfang ist dies sein Initialwert.
- t_2 Agent x empfängt k mal den Wert v als Vorschlag und ist damit *Zeuge* einer Mehrheit für v in Runde r. Er sendet an jeden Agenten eine Zeugen-Nachricht für v.

[5] Σ_{26} ist formal kein algebraisches Netz nach Definition 1.24, da in Σ_{26} mache Kanten mit einer Multimenge und nicht mit einer Menge beschriftet sind. Dies können wir z.B. dadurch reparieren, indem wir für jede Nachricht neben dem Empfänger auch den Absender mit in die Nachricht aufnehmen.



$i : A \to \{0,1\}, \quad I = \{(x, i(x), 0) \mid x \in A\}$

$\alpha : |V| \geq k, V \geq \{0,1\}$

Abb. 4.11: Σ_{26} – Der Algorithmus von Ben-Or.

t_3 Agent x empfängt k Vorschläge, dabei kommt sowohl 0 als auch 1 als vorgeschlagener Wert vor; x wird dabei nicht Zeuge einer Mehrheit, x sendet an jeden Agenten eine Enthaltungs-Nachricht. Eine Enthaltungs-Nachricht enthält keinen Wert.

t_4 Agent x entscheidet ν falls er von k Zeugen für ν erfährt. Er terminiert nicht, sondern geht in die nächste Runde mit ν als Vorschlagswert.

t_5 Agent x erfährt von einem Zeugen für ν und empfängt außerdem noch eine Enthaltungs-Nachricht. Dann entscheidet er sich nicht und geht mit ν als Vorschlagswert in die nächste Runde[6].

t_6 Agent x empfängt k Enthaltungs-Nachrichten. Dann wirft x eine Münze, um seinen Vorschlagswert für die nächste Runde festzulegen.

t_7 Agent x geht in den Zustand *initial* zurück und erhöht dabei seinen Rundenzähler.

Wir begründen nun die Korrektheit von Ben-Or's Algorithmus. Genauere Beweise findet man in [4] und [86]. Sind alle Anfangswerte der Agenten gleich, so verläuft

[6] In Ben-Ors Originalalgorithmus wartet der Agent hier bis er insgesamt k Nachrichten – Enthaltungs- oder Zeugen-Nachrichten – empfangen hat.

der Algorithmus bis auf Ausfälle deterministisch ab: Alle nicht-ausfallenden Agenten senden dieselben Vorschläge, werden Zeugen und entscheiden sich bereits in der ersten Runde. Daher gilt (2.5) in Σ_{26}. Wir wollen nun begründen, warum zwei Agenten sich nicht für verschiedene Werte entscheiden können. Es kann nur dann einen Zeugen für ν in Runde r geben, falls eine Mehrheit von Agenten (mindestens k) in Runde r den Wert ν als Vorschlagswert haben. Daher kann es in einer Runde nie Zeugen für verschiedene Werte geben. Daraus folgt, daß sich zwei Agenten nicht in derselben Runde für unterschiedliche Werte entscheiden können.

Kann sich ein Agent in Runde r für den Wert ν entscheiden, so gibt es k Zeugen für ν in Runde r. Dann kann es keine k Enthaltungen in Runde r geben und alle Agenten schalten entweder t_4 oder t_5 in Runde r. Damit übernimmt jeder den Wert eines Zeugen für Runde $r+1$, woraus folgt, daß alle Agenten in Runde $r+1$ denselben Vorschlagswert haben. Kann sich also ein Agent in Runde r für den Wert ν entscheiden, so entscheiden sich alle nicht-ausfallenden Agenten in Runde $r+1$ auch für ν. Es ist also auch in verschiedenen Runden nicht möglich, daß Agenten sich für verschiedene Werte entscheiden.

Für die Herbeiführung einer Entscheidung mit Wahrscheinlichkeit 1 sorgt der Münzwurf von Transition t_6. Werfen alle Agenten immer wieder in einer Runde gemeinsam eine Münze, so geht der Münzwurf irgendwann in derselben Runde gleich für alle Agenten aus. Dann haben alle Agenten denselben Vorschlagswert in der folgenden Runde, in der sich jeder Agent dann entscheidet.

Nach der Entscheidung läuft der Algorithmus unendlich lange weiter. Der Algorithmus kann aber leicht modifiziert werden, so daß jeder Agent terminiert: Da jeder Agent spätestens eine Runde nach der ersten Entscheidung entschieden ist, genügt es, daß jeder Agent nach seiner Entscheidung noch eine weitere Runde am Algorithmus teilnimmt.

Satz 4.12 (Konsens in randomisierten Netzsystemen)

Sei A eine endliche Menge von Agenten. Dann gibt es ein randomisiertes Netzsystem für A, das sowohl Konsens-Struktur für A als auch probabilistisches k-ausfalltolerantes Konsens-Verhalten für $k < \frac{|A|}{2}$ besitzt.

4.3 Mutex in randomisierten Netzsystemen

In diesem Abschnitt zeigen wir, daß das Mutex-Problem in randomisierten Netzsystemen nicht gelöst werden kann. Dieses Resultat ist neu. Es bedeutet, daß die Fairneßannahme, die zur Lösung des Mutex-Problems benötigt wird, nicht durch Münzwurf implementiert werden kann. Damit kann auch Fairneß im allgemeinen nicht durch Münzwurf implementiert werden. Im Unterabschnitt 4.3.2 diskutieren wir die Hintergründe dieses Resultats.

4.3.1 Unmöglichkeit von Mutex in randomisierten Netzsystemen

Ein randomisiertes Netzsystem stellt eine Mutex-Lösung dar, falls es *Mutex-Struktur* und *probabilistisches Mutex-Verhalten* besitzt. Ein randomisiertes Netzsystem hat Mutex-Struktur für eine endliche Menge A von Agenten, falls das zugrundeliegende Netzsystem Mutex-Struktur für A hat (vgl. Definition 2.9 auf Seite 48). Damit nehmen wir an, daß keine der durch Definition 2.9 vorgegebenen Transitionen probabilistisch ist. Probabilistisches Mutex-Verhalten definieren wir wie folgt:

Definition 4.13 (Probabilistisches Mutex-Verhalten)

Ein randomisiertes Netzsystem $\dot{\Sigma}$ besitzt *probabilistisches Mutex-Verhalten*, falls in $\dot{\Sigma}$ die beiden Eigenschaften (2.1) und (2.2) (vgl. Seite 47) probabilistisch gültig sind. ○

Wir zeigen nun die Unmöglichkeit einer Mutex-Lösung in randomisierten Netzsystemen.

Satz 4.14 (Unmöglichkeit von Mutex in randomisierten Netzsystemen)

Es gibt kein randomisiertes Netzsystem, das sowohl Mutex-Struktur für $\{\mathrm{l},\mathrm{r}\}$ als auch probabilistisches Mutex-Verhalten hat.

Beweis: Wir führen einen indirekten Beweis. Sei $\dot{\Sigma}$ ein randomisiertes Netzsystem mit Mutex-Struktur und probabilistischem Mutex-Verhalten. $\dot{\Sigma}$ hat einen progressiven probabilistischen Ablauf π_1, in dem r nie hungrig wird, l aber so oft wie möglich; π_1 wird wie folgt konstruiert: Man beginnt mit dem ereignislosen probabilistischen Ablauf und läßt a_l schalten. Sei κ der entstandene endliche probabilistische Ablauf. Jetzt setzen wir mit Progreß fort, d.h. wir wählen eine minimale progressive Fortsetzung κ' von κ. An jede $ruhig_{\mathrm{l}}$-Bedingung im Ende von κ' hängen wir nun durch Schalten von a_l eine $hungrig_{\mathrm{l}}$-Bedingung an und verfahren so wie eben. Durch unendliche Iteration erhalten wir den progressiven probabilistischen Ablauf π_1.

Dann gibt es einen progressiven probabilistischen Ablauf π_2, der π_1 fortsetzt, in dem r hungrig wird. (Wir lassen a_r im Ende von ρ_1 schalten und setzen mit Progreß fort.)

Da in $\dot{\Sigma}$ Eigenschaft (2.2) probabilistisch gültig ist, gilt $\pi_2 \Vdash \Diamond \mathit{kritisch}_r$. Sei B_i die Menge der Bedingungen von π_i für $i = 1, 2$ und sei b eine Bedingung von $B_2 \setminus B_1$ mit $\widetilde{b} = \mathit{kritisch}_r$ und sei C ein Markierungsschnitt mit $b \in C$. In C gilt entweder ruhig_l oder $\mathit{hungrig}_l$ oder $\mathit{kritisch}_l$. Gilt ruhig_l so folgt nach Konstruktion eine $\mathit{hungrig}_l$-Bedingung. In jedem Fall gibt es wegen der probabilistischen Gültigkeit von (2.2) eine $\mathit{kritisch}_l$-Bedingung b', die von C erreichbar ist. Es gilt:

1. Es ist nicht $b = b'$, da $\widetilde{b} \neq \widetilde{b'}$.
2. Es ist weder $b \# b'$ noch $b' < b$, da b' von C erreichbar ist.
3. Es ist nicht $b < b'$, da $\pi_1 \sqsubseteq \pi_2$ sowie $b \in B_2 \setminus B_1$ und $b' \in B_1$.
4. Es ist nicht b co b', da die Sicherheitseigenschaft (2.1) in jedem Ablauf gültig ist. (Folgerung 4.10)

Dies ist aber ein Widerspruch dazu, daß π_2 eine Abwicklung ist. □

Das Mutex- und das Konsens-Problem sind wegen Satz 4.14 bezüglich ihrer Lösbarkeit unvergleichbar: Es gibt ein Modell (nämlich faire Netzsysteme), in dem das Mutex-Problem lösbar, das Konsens-Problem aber unlösbar ist und es gibt ein Modell (nämlich randomisierte Netzsysteme), in dem das Konsens-Problem lösbar, das Mutex-Problem aber unlösbar ist. Daraus ergibt sich auch die Unvergleichbarkeit der Ausdrucksstärke von fairen und randomisierten Netzsystemen.

4.3.2 Zwei Aspekte von Fairneß

In diesem Abschnitt wollen wir die Unmöglichkeit von Mutex in randomisierten Netzsystemen besser verstehen, indem wir verschiedene Konfliktarten unterscheiden.

Wir betrachten zunächst einen Free-Choice-Konflikt wie in Abb. 4.12(a). Fairneß bzgl. a und b kann einfach durch Randomisierung wie in Abb. 4.12(b) implementiert werden. Abb. 4.12(c) zeigt, daß Fairneß bzgl. a und b sogar durch Progreß implementiert werden kann. Dabei können D und E auch mehr Marken enthalten als in Abb. 4.12(c) dargestellt.

Fairneß in einem Extended-Free-Choice-Konflikt kann durch die Konstruktion in Abb. 4.2 auf Seite 80 auf Fairneß im Free-Choice-Konflikt zurückgeführt werden. In unserer Mutex-Lösung in Abb. 3.7 auf Seite 70 finden wir Fairneß in einem Konflikt, der kein Extended-Free-Choice-Konflikt ist. Wir haben diesen Konflikt in Abb. 4.13 noch einmal dargestellt. Dieser Konflikt erlaubt im Gegensatz zu einem Extended-Free-Choice-Konflikt Konfusion (vgl. Abschnitt 3.1.2). Ein Charakteristikum des Konflikts in Abb. 4.13 ist, daß jede Transition nur über eine Stelle ihres Vorbereich

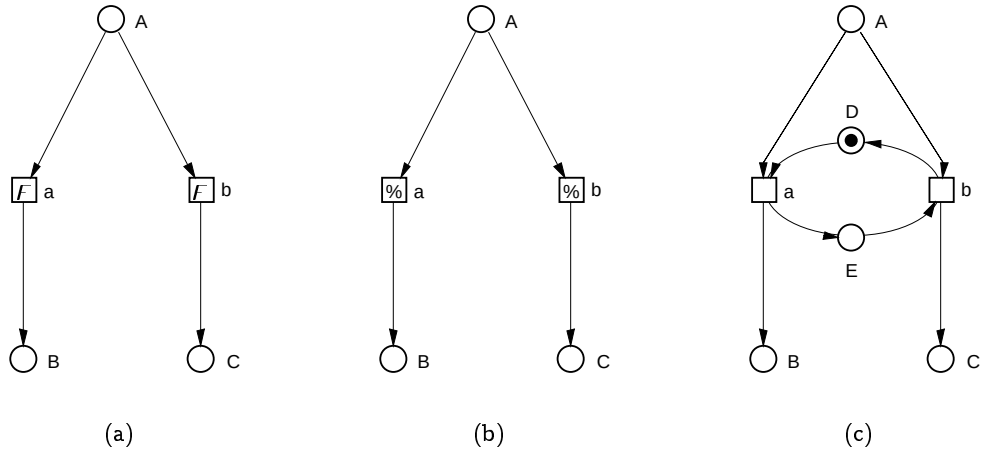


Abb. 4.12: Implementierung freier Fairneß.

mit einer anderen Transition in Konflikt steht. Eine solche Transition (bzw. einen solchen Konflikt) nennen wir *einfach*. Wir halten nun verschiedene Konfliktarten in der folgenden Definition fest.

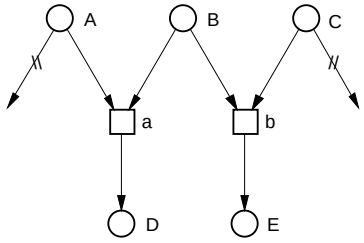


Abb. 4.13: Ein einfacher Konflikt.

Definition 4.15 (Konfliktarten für Transitionen)

Sei N ein Netz. Eine Transition t von N ist

a) *frei*, falls für alle $p, q \in {}^\bullet t$ gilt: $p \neq q \Rightarrow |p^\bullet| = |q^\bullet| = 1$,

b) *quasi-frei*, falls für alle $p, q \in {}^\bullet t$ gilt: $p^\bullet = q^\bullet$,

c) *schwach konfus*, falls sie nicht quasi-frei ist,

d) *einfach*, falls für alle $p, q \in {}^\bullet t$ gilt: $p \neq q \Rightarrow |p^\bullet| = 1 \vee |q^\bullet| = 1$,

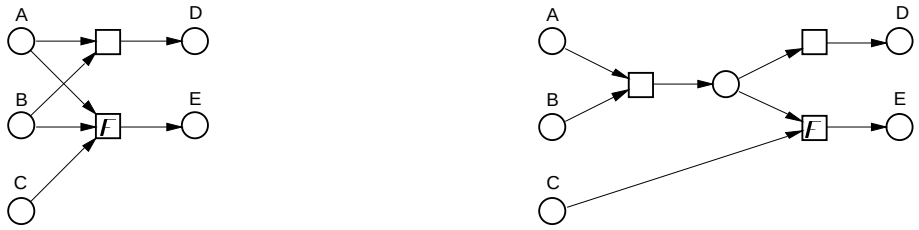
e) *quasi-einfach*, falls für alle $p, q \in {}^\bullet t$ gilt: $p^\bullet \subseteq q^\bullet \vee q^\bullet \subseteq p^\bullet$,

f) *stark konfus*, falls sie nicht quasi-einfach ist. ○

Ein Netz heißt *Extended-Free-Choice-Netz*[7], falls all seine Transitionen quasi-frei sind. Ein Netz heißt *Extended-Simple-Netz*, falls all seine Transitionen quasi-einfach sind. Für Free-Choice-Netze gibt es eine reichhaltige Theorie und viele gute Analyseverfahren [27, 18]. Viele Verfahren, die für Extended-Free-Choice-Netze entwickelt wurden, konnten auf Extended-Simple-Netze übertragen werden. Eine zu Definition 4.15 äquivalente Charakterisierung der Konfliktarten findet man in [1].

Bei Fairneß bezüglich einer freien Transition sprechen wir von *freier Fairneß*, bei Fairneß bezüglich einer einfachen Transition von *einfacher Fairneß*. Freie Fairneß bzgl. einer Transition t können wir wie in Abb. 4.12(c) durch Progreß implementieren. Gibt es dabei eine Transition t' mit ${}^\bullet t \subseteq {}^\bullet t'$, so kann t' in der Implementierung weggelassen werden.

Aus der Unmöglichkeit von Mutex in randomisierten Netzsystemen folgt, daß einfache Fairneß im allgemeinen weder durch Progreß noch durch Randomisierung implementiert werden kann.



(a) (b)

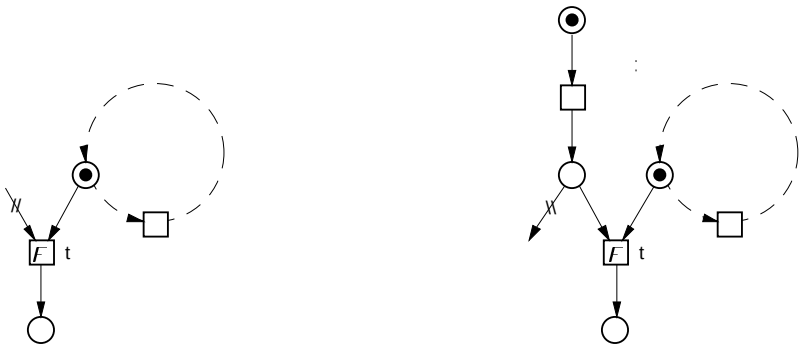
Abb. 4.14: Verfeinerung eines Extended-Simple-Konfliktes.

Fairneß bezüglich einer quasi-einfachen Transition kann wie in Abb. 4.14 auf einfache Fairneß zurückgeführt werden. Best gibt in [15] ein ausführliche Beschreibung der Überführung eines Extended-Simple-Netz in ein *Simple-Netz*, wobei ein *Simple-Netz* ein Netz mit ausschließlich einfachen Transitionen ist. Auf Fairneß bzgl. stark konfuser Transitionen gehen wir später ein.

Dieser Abschnitt hat gezeigt, daß Fairneß mindestens zwei verschiedene Aspekte vereint – freie und einfache Fairneß. Abb. 4.15 verdeutlicht noch einmal den Unterschied zwischen freier und einfacher Fairneß. Wir führen zur Erklärung von Abb. 4.15 noch zwei Begriffe ein. Sei Σ ein initialisiertes Netz. Eine Stelle p von Σ ist in einem Ablauf ρ von Σ *persistent*, falls eine Bedingung $b \in \rho^\circ$ im Ende von ρ existiert mit $\widetilde{b} = p$; p ist *rekurrent* in ρ, falls es in ρ unendlich viele Bedingungen b mit $\widetilde{b} = p$ gibt. Progreß fordert das Schalten einer Transition t in einem Ablauf ρ, falls alle Ressourcen

[7] Extended-Free-Choice und Free-Choice werden in der Literatur nicht immer unterschieden. Ein Extended-Free-Choice-Netz heißt deshalb oft auch *Free-Choice-Netz*.

von t in ρ persistent sind. Freie Fairneß fordert darüberhinaus das Schalten einer Transition t, falls die einzige Ressource von t in ρ rekurrent ist. Einfache Fairneß fordert das Schalten von t, falls eine Ressource rekurrent ist, während alle anderen Ressourcen von t persistent sind, d.h. einfache Fairneß verlangt die Synchronisation endlich vieler persistenter Ressourcen mit einer rekurrenten Ressource[8].



(a) Freie Fairneß (b) Einfache Fairneß

Abb. 4.15: Zwei Aspekte von Fairneß.

Wir gehen nun zum Teil der Arbeit über. Dort werden wir Fairneß und Randomisierung in einem Modell vereinen, und untersuchen, wo die Grenzen der Ausdrucksstärke des neuen Modells liegen.

[8] Einen bzgl. freier Fairneß unfairen Ablauf sollten wir *ungerecht* nennen, einen bzgl. einfacher Fairneß unfairen Ablauf *ignorant*.

Teil II

Konspiration

5 Faire randomisierte Netzsysteme

In diesem Kapitel definieren wir *faire randomisierte Netzsysteme*. Faire randomisierte Netzsysteme vereinen die Ausdrucksstärke fairer Netzsysteme mit der Ausdrucksstärke randomisierter Netzsysteme, so daß sowohl das Mutex-Problem als auch das Konsens-Problem in fairen randomisierten Netzsystemen lösbar ist (vgl. Abb. 5.1). Auch für faire randomisierte Netzsysteme geben wir ein nicht-lösbares

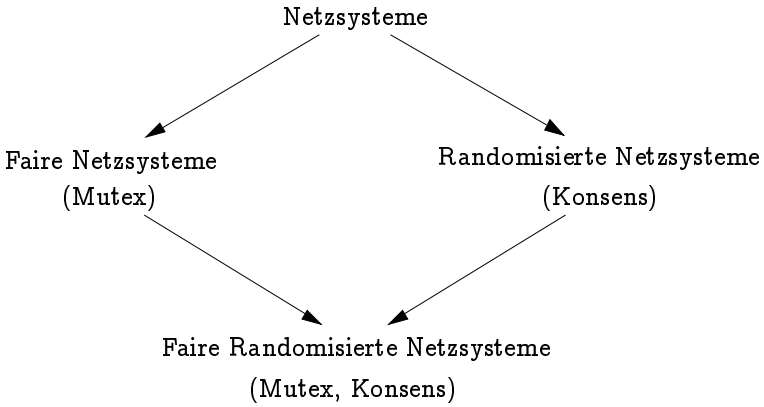


Abb. 5.1: Beziehungen der Modelle.

populäres Problem an – das *ausfalltolerante allgemeine Mutex-Problem*. Dieses Problem ist eine ausfalltolerante Version des *allgemeinen Mutex-Problems*, welches wiederum auch als *Problem der speisenden Philosophen* [24] bekannt ist. Beim allgemeinen Mutex-Problem ist eine Nachbarschaftsrelation auf endlich vielen Agenten gegeben und ein Algorithmus gesucht, der das Mutex-Problem simultan für jedes Paar von Nachbarn löst.

Die Unmöglichkeit von ausfalltolerantem allgemeinen Mutex in fairen randomisierten Systemen ist ein neues Ergebnis. Es zeigt eine Grenze der Anwendbarkeit asynchroner randomisierter Algorithmen auf. Wir werden später sehen, daß das ausfalltolerante allgemeine Mutex-Problem unter einer zusätzlichen schwachen Synchronieannahme gelöst werden kann. Damit unterstreicht unser Ergebnis die Bedeutung von Synchronieannahmen für Ausfalltoleranz. Nachdem fehlende Synchronie oft für die Unmöglichkeit von ausfalltolerantem Konsens in asynchronen Systemen verantwortlich gemacht wurde, wurde diese Bedeutung von Synchronie durch Ben-Or [14]

relativiert, der zeigte, daß ausfalltoleranter Konsens durch Randomisierung völlig asynchron möglich ist. Unser Resultat zeigt nun eine Grenze dafür auf, Ausfalltoleranz mit Hilfe von Randomisierung zu erzielen. Das Konsens-Problem besitzt vermutlich mit seiner asynchronen Lösbarkeit durch Randomisierung eine Ausnahmestellung innerhalb fehlertoleranter Probleme.

Wir gehen wie folgt vor. Zunächst definieren wir in Abschnitt 5.1 faire randomisierte Netzsysteme und ihre Semantik. In Abschnitt 5.2 betrachten wir dann das allgemeine Mutex-Problem und das aufalltolerante allgemeine Mutex-Problem und beweisen die Unlösbarkeit des aufalltoleranten allgemeinen Mutex-Problems in fairen randomisierten Netzsystemen. Den Hintergrund für diese Unlösbarkeit diskutieren wir dann ausführlich in Kapitel 6.

5.1 Faire randomisierte Netzsysteme

In diesem Abschnitt definieren wir faire randomisierte Netzsysteme und ihre Semantik – *faire probabilistische Abläufe*. Ein *faires randomisiertes Netzsystem* ist ein randomisiertes Netzsystem, in dem einige interne, nicht-probabilistische Transitionen als fair ausgezeichnet sind.

Definition 5.1 (Faires randomisiertes Netzsystem)

Ein *faires randomisiertes Netzsystem* $\ddot{\Sigma} = (\dot{\Sigma}, T^{fair})$ besteht aus einem randomisierten Netzsystem $\dot{\Sigma} = (\Sigma, T^{flip}, \mu)$ mit Transitionsmenge T und Menge externer Transitionen T^{ext}, und einer Menge $T^{fair} \subseteq (T \setminus T^{ext}) \setminus T^{flip}$ ausgezeichneter interner nicht-probabilistischer Transitionen von Σ. Ein Element von T^{fair} heißt *Fairneß-transition* von $\ddot{\Sigma}$. ○

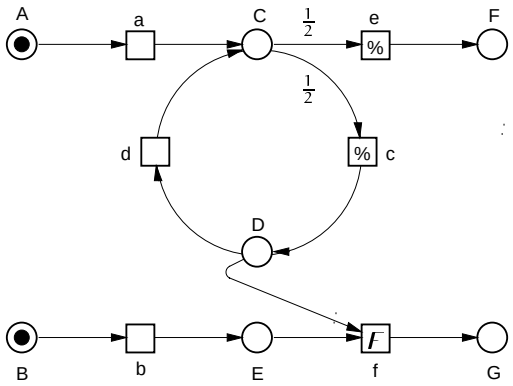


Abb. 5.2: Σ_{27}

Abb. 5.2 zeigt das faires randomisiertes Netzsystem Σ_{27}. Der Free-Choice-Konflikt zwischen e und c wird durch Münzwurf gelöst, der Konflikt zwischen f und d

wird fair gelöst. Die Auszeichnung fairer Transitionen dient wie üblich dazu, unfaire Abläufe des zugrundeliegenden Systems auszusondern. Wir wollen nun Fairneß auf probabilistischen Abläufen definieren.

Es gibt zwei kanonische Möglichkeiten, Fairneß auf probabilistischen Abläufen zu definieren. Wir können einen probabilistischen Ablauf π als fair bezeichnen, falls

a) jeder maximale Ablauf von π fair ist, oder falls

b) ein maximaler Ablauf von π mit Wahrscheinlichkeit 1 fair ist, d.h. falls $P_\pi\{\rho \in \mathfrak{R}_{max}(\pi) \mid \rho \text{ ist fair}\} = 1$.

Ist ein probabilistischer Ablauf nach a) fair, so ist er auch nach b) fair. Die Umkehrung gilt nicht: Σ_{27} hat genau einen unendlichen probabilistischen Ablauf π; dieser ist nach b) fair, nach a) jedoch nicht fair. Der Unterschied beider Definitionen ist aber nicht wesentlich, d.h. unter Fairneß nach a) und nach b) gelten diegleichen temporallogischen Eigenschaften. Dies zeigen Hart, Sharir und Pnueli in [40] für sequentielle Semantik. Sie zeigen, daß jeder nach b) unfaire probabilistische Ablauf der Grenzwert einer Folge von nach a) unfairen probabilistischen Abläufen ist. Wir werden diesen Zusammenhang für unsere nicht-sequentielle Semantik nicht beweisen, da er im weiteren keine Rolle spielt. Im Beispiel von Σ_{27} kann man leicht überprüfen, daß durch den Ausschluß des unendlichen probabilistischen Ablaufs π nicht mehr temporallogische Eigenschaften in Σ_{27} probabilistisch gültig werden. Wir definieren nun Fairneß auf probabilistischen Abläufen nach a):

Definition 5.2 (Fairer probabilistischer Ablauf)

Sei $\ddot{\Sigma} = (\dot{\Sigma}, T^{fair})$ ein faires randomisiertes Netzsystem und $t \in T^{fair}$. Ein probabilistischer Ablauf π von $\dot{\Sigma}$ ist *fair* bzgl. t, falls jeder maximale Ablauf von π fair bzgl. t ist; π ist *fair*, falls π fair bzgl. jedem $t \in T^{fair}$ ist. ∘

Wir wenden uns nun dem allgemeinen Mutex-Problem zu.

5.2 Allgemeiner Mutex in fairen randomisierten Netzsystemen

In diesem Abschnitt stellen wir das allgemeine Mutex-Problem und das ausfalltolerante allgemeine Mutex-Problem vor und beweisen, daß das ausfalltolerante allgemeine Mutex-Problem in fairen randomisierten Netzsystemen nicht lösbar ist.

5.2.1 Das allgemeine Mutex-Problem

Beim allgemeinen Mutex-Problem für eine endliche Menge A von Agenten ist eine *Nachbarschaftsrelation* auf A gegeben, d.h. eine irreflexive und symmetrische Relation $N \subseteq A \times A$. Ist $(x, y) \in N$, so heißen x und y *Nachbarn*. Für einen Agenten x

bezeichnet $N(x) = \{y \mid (x,y) \in N\}$ die Menge aller Nachbarn von x. Gesucht ist ein Algorithmus, der für jedes Paar von Nachbarn das Mutex-Problem simultan löst, d.h. ein Algorithmus der gewährleistet, das

1. zwei Nachbarn nie zugleich kritisch sind, und
2. jeder hungrige Agent irgendwann kritisch wird.

Gesucht ist also ein System mit Mutex-Struktur für A, das die folgenden beiden temporallogischen Eigenschaften erfüllt:

$$\forall (x,y) \in N : \Box \neg (\mathit{kritisch}(x) \wedge \mathit{kritisch}(y)) \tag{5.1}$$

$$\forall x \in A : \mathit{hungrig}(x) \rhd \mathit{kritisch}(x) \tag{5.2}$$

Das allgemeine Mutex-Problem ist auch als *Problem der speisenden Philosophen* [24] bekannt. (Den speisenden Philosophen werden wir noch in Abschnitt 6.1.1 begegnen.) In [24], Kapitel 12 zeigen Chandy und Misra, daß das allgemeine Mutex-Problem unter Fairneß für jede Nachbarschaftsrelation eine Lösung hat. Bei dieser Lösung verwalten die Agenten einen dynamischen Wartegraphen, d.h. einen azyklischen, zusammenhängenden, gerichteten Graphen (A, W) auf den Agenten, so daß $W \subseteq N$. Ist $(x,y) \in W$, so hat x *Priorität* über y, so daß im Falle, daß sowohl x als auch y hungrig ist, y auf x warten muß, d.h. x wird vor y kritisch. Nachdem x kritisch war, verliert x all seine Prioritäten über seine Nachbarn. Dabei bleibt der Wartegraph azyklisch – eine wichtige Eigenschaft, damit der Algorithmus nicht verklemmt. Ein Petrinetzmodell einschließlich Korrektheitsbeweis dieser Lösung findet man in [81] und [91].

Wir gehen jetzt zum ausfalltoleranten allgemeinen Mutex-Problem über.

5.2.2 Das ausfalltolerante allgemeine Mutex-Problem

Wir betrachten weiterhin das allgemeine Mutex-Problem und nehmen nun an, daß Agenten ausfallen können[1]. Da ein Agent jederzeit ausfallen kann, kann auch ein hungriger Agent ausfallen. Wir können daher höchstens von einem nicht-ausfallenden Agenten verlangen, daß er irgendwann kritisch wird. Dies führt zur Forderung der folgenden temporallogischen Eigenschaft (5.3), eine Abschwächung von (5.2):

$$\forall x \in A : \mathit{hungrig}(x) \rhd (\mathit{kritisch}(x) \vee \mathit{ausgefallen}(x)) \tag{5.3}$$

Aber auch (5.3) läßt sich zusammen mit (5.1) in asynchronen Systemen nicht erfüllen. Fällt nämlich ein kritischer Agent x aus, so kann kein Nachbar y von x mehr

[1] In Anlehnung an die speisenden Philosophen können wir hier von den *sterbenden Philosophen* sprechen.

kritisch werden, da sich y des Ausfalls von x nie sicher sein kann. Damit die Lebendigkeitseigenschaft für das allgemeine Mutex-Problem unter Annahme von Ausfällen erfüllbar ist, schwächen wir (5.3) nun weiter zur folgenden Eigenschaft (5.4) ab. Wir fordern, daß jeder hungrige Agent kritisch wird, es sei denn, er selbst oder einer seiner Nachbarn fällt aus.

$$\forall x \in A : \mathit{hungrig}(x) \rhd (\mathit{kritisch}(x) \vee \mathit{ausgefallen}(x) \vee \exists y \in N(x) : \mathit{ausgefallen}(y)) \tag{5.4}$$

Wir definieren nun, wann ein faires randomisiertes Netzsystem *probabilistisches ausfalltolerantes allgemeines Mutex-Verhalten für* A *und* N besitzt.

Definition 5.3 (Ausfalltolerantes allgemeines Mutex-Verhalten)

Sei A eine endliche Menge von Agenten und N eine Nachbarschaftsrelation auf A. Ein faires (randomisiertes) Netzsystem $\ddot{\Sigma}$ besitzt *(probabilistisches) ausfalltolerantes allgemeines Mutex-Verhalten für* A *und* N, falls in jedem fairen (probabilistischen) Ablauf π von $\ddot{\Sigma}$ die beiden Eigenschaften (5.1) und (5.4) (probabilistisch) gültig sind. ∘

Definition 5.3 definiert zusammen mit Mutex-Struktur (siehe Definition 2.9 auf Seite 48) das ausfalltolerante allgemeine Mutex-Problem. Im nächsten Abschnitt zeigen wir nun, daß es kein faires randomisiertes Netzsystem gibt, das das ausfalltolerante allgemeine Mutex-Problem löst.

5.2.3 Unmöglichkeit von ausfalltolerantem allgemeinen Mutex

Wir zeigen nun, daß für jede Menge A von mindestens drei Agenten eine Nachbarschaftsrelation existiert, so daß es kein faires randomisiertes Netzsystem gibt, das sowohl Mutex-Struktur für A als auch probabilistisches ausfalltolerantes allgemeines Mutex-Verhalten für A und N hat. Ein faires randomisiertes Netzsystem hat dabei Mutex-Struktur für A, falls das zugrundeliegende randomisierte Netzsystem Mutex-Struktur für A hat.

Satz 5.4 (Unmöglichkeit von ausfalltolerantem allgemeinen Mutex)

Sei A eine endliche Menge von Agenten mit $|A| \geq 3$. Dann gibt es eine Nachbarschaftsrelation N auf A, so daß kein faires randomisiertes Netzsystem $\ddot{\Sigma}$ existiert, das sowohl Mutex-Struktur für A als auch probabilistisches ausfalltolerantes allgemeines Mutex-Verhalten für A und N besitzt.

Zum besseren Verständnis des Beweises von Satz 5.4 beweisen wir zunächst den folgenden schwächeren Satz 5.5.

Satz 5.5

Sei A eine endliche Menge von Agenten mit $|A| \geq 3$. Dann gibt es eine Nachbarschaftsrelation N auf A, so daß kein faires Netzsystem $\acute{\Sigma}$ existiert, das sowohl Mutex-Struktur für A als auch ausfalltolerantes allgemeines Mutex-Verhalten für A und N besitzt.

Beweis: (von Satz 5.5) Wir betrachten drei Agenten $a, b, c \in A$ mit der Nachbarschaftsrelation N wie in Abb. 5.3. Wir nehmen an, es gibt ein faires Netzsystem

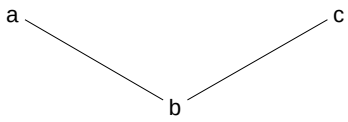


Abb. 5.3: Eine Nachbarschaftsrelation auf drei Agenten.

$\acute{\Sigma}$, das sowohl Mutex-Struktur für A als auch ausfalltolerantes allgemeines Mutex-Verhalten für A und N besitzt und führen diese Annahme zum Widerspruch. Wir konstruieren dazu einen fairen Ablauf ρ von $\acute{\Sigma}$, der (5.4) verletzt. Wir beginnen mit dem ereignislosen Ablauf ρ_0 von $\acute{\Sigma}$.

1. Agent a wird hungrig, d.h. wir fügen ein Ereignis an ρ_0 an, das $ruhig_a$ in $hungrig_a$ überführt. Wir setzen dann fair fort bis $kritisch_a$ gilt (wegen (5.4) ist das irgendwann der Fall); den entstandenen endlichen Ablauf nennen wir ρ_1. In ρ_1 gibt es keine $kritisch_b$-Bedingung, da b in ρ_1 nie hungrig ist.

2. Agent b wird hungrig.

3. Agent c wird hungrig; wir setzen fair fort ohne Ereignisse von a zu verwenden (d.h. wir tun so, als ob a ausfällt) bis $kritisch_c$ gilt (wegen (5.4) ist das irgendwann der Fall); den entstandenen endlichen Ablauf nennen wir ρ_2. In ρ_2 gilt $\Diamond\, kritisch_b$ nicht, da $kritisch_b$ nicht nebenläufig zu $kritisch_a$ sein kann und wir in diesem Schritt nebenläufig zu $kritisch_a$ fortgesetzt haben.

4. Agent a wird ruhig und dann hungrig; wir setzen dann fair fort ohne Ereignisse von c zu verwenden bis $kritisch_a$ gilt; den entstandenen endlichen Ablauf nennen wir ρ_3. In ρ_3 gilt $\Diamond\, kritisch_b$ nicht, da $kritisch_b$ nicht nebenläufig zu $kritisch_c$ sein kann und wir in diesem Schritt nebenläufig zu $kritisch_c$ fortgesetzt haben.

5. Wir führen Schritte 3 und 4 unendlich oft hintereinander aus.

Durch die unendliche Fortsetzung erhalten wir einen fairen Ablauf ρ, in welchem $\Diamond\, kritisch_b$ nicht gilt und der damit (5.4) verletzt. □

Im konstruierten Ablauf ρ im Beweis von Satz 5.5 werden die Agenten a und c immer unabhängig voneinander kritisch. Dies erlaubt eine zeitliche Reihenfolge des kritisch-werdens, in der zu jeder Zeit mindestens einer der beiden Agenten a und c kritisch ist, weshalb Agent b nie kritisch werden kann. Nach dieser zeitlichen Reihenfolge haben wir ρ gerade konstruiert. Auf ähnliche Art und Weise beweisen wir jetzt Satz 5.4.

Beweis: (von Satz 5.4) Wir nehmen an, es gibt ein faires randomisiertes Netzsystem $\ddot{\Sigma}$, das sowohl Mutex-Struktur für A als auch probabilistisches ausfalltolerantes allgemeines Mutex-Verhalten für A und N besitzt und führen diese Annahme zum Widerspruch. Wir konstruieren dazu einen fairen probabilistischen Ablauf π von $\ddot{\Sigma}$ mit $P_\pi((5.4)) < 1$. Wir beginnen mit dem ereignislosen probabilistischen Ablauf π_0 von $\ddot{\Sigma}$.

1. Agent a wird hungrig, d.h. wir fügen ein Ereignis an π_0, das $ruhig_a$ in $hungrig_a$ überführt. Nun gibt es einen fairen progressiven probabilistischen Ablauf π_0', der den bisherigen fortsetzt und in dem $\Diamond\, kritisch_a$ mit Wahrscheinlichkeit 1 gilt. Dann gibt es für jedes $p_1 < 1$ einen endlichen Präfix π_1 von π_0', in dem $\Diamond\, kritisch_a$ mindestens mit Wahrscheinlichkeit p_1 gilt, wobei wir hinter allen $kritisch_a$-Bedingungen abschneiden. In π_1 gibt es keine $kritisch_b$-Bedingungen, da b in π_1 nie hungrig ist.

2. Agent b wird hungrig.

3. Agent c wird hungrig; wir setzen fair fort zu einem endlichen probabilistischen Ablauf $\pi_2 \sqsupseteq \pi_1$ ohne Ereignisse von a zu verwenden (d.h. wir tun so als ob a ausfällt), so daß $\Diamond\, kritisch_c$ in π_2 mindestens mit Wahrscheinlichkeit p_2 gilt und schneiden überall hinter $kritisch_c$ ab. In π_2 gilt $\Diamond\, kritisch(b)$ höchstens mit Wahrscheinlichkeit $1-p_1 =: \varepsilon_1$, da $kritisch_b$ nicht nebenläufig zu $kritisch_a$ sein kann und wir in diesem Schritt mit Wahrscheinlichkeit p_1 nebenläufig zu $kritisch_a$ fortgesetzt haben.

4. Agent a wird in jedem Ablauf von π_2 hungrig, in dem er noch nicht hungrig ist; wir setzen fair fort zu einem endlichen probabilistischen Ablauf $\pi_3 \sqsupseteq \pi_2$ ohne Ereignisse von c zu verwenden, so daß $hungrig_a \rhd kritisch_a$ in π_3 mindestens mit Wahrscheinlichkeit p_3 gilt und schneiden wiederum hinter $kritisch_a$ ab. In π_3 gilt $\Diamond\, kritisch_b$ höchstens mit Wahrscheinlichkeit $\varepsilon_1+\varepsilon_2$, wobei $\varepsilon_2 := 1-p_2$, da $kritisch_b$ nicht nebenläufig zu $kritisch_c$ sein kann und wir in diesem Schritt mit Wahrscheinlichkeit p_2 nebenläufig zu $kritisch_c$ fortgesetzt haben.

5. Wir führen Schritte 3 und 4 unendlich oft hintereinander aus.

Durch unendliche Fortsetzung erhalten wir einen fairen probabilistischen Ablauf π, in dem $\Diamond\, \mathit{kritisch}_b$ höchstens mit Wahrscheinlichkeit

$$\varepsilon = \sum_{i=1}^{\infty} \varepsilon_i$$

gilt. Nun können wir aber die p_i so wählen, daß ε beliebig klein wird, z.B. durch $\varepsilon_i = (\frac{1}{4})^i$. Dann ist

$$\varepsilon = \sum_{i=1}^{\infty} \left(\frac{1}{4}\right)^i = \frac{\frac{1}{4}}{1-\frac{1}{4}} = \frac{1}{3}.$$

(geometrische Reihe) □

Benutzt man sequentielle Semantik für faire randomisierte Netzsysteme, also probabilistische Schaltbäume, so kann man einen einfacheren Beweis führen (das Resultat ist ja auch schwächer). Man kann dann sogar einen probabilistischen Schaltbaum konstruieren, so daß die Lebendigkeitseigenschaft (5.4) in keiner maximalen Schaltsequenz des probabilistischen Schaltbaums gilt. Der allgemeine sequentielle Gegenspieler kann nämlich beispielsweise in Schritt 4 abwarten bis $\mathit{kritisch}_a$ gilt. Ist das nie der Fall, d.h. wird in Schritt 4 unendlich fortgesetzt, so ist $\mathit{hungrig}_a \rhd \mathit{kritisch}_a \vee \mathit{ausgefallen}(a) \vee \mathit{ausgefallen}(b)$ verletzt.

Im Beweis von Satz 5.4 haben wir wieder die Unabhängigkeit des kritisch-werdens der Agenten a und c ausgenutzt, wodurch eine zeitliche Reihenfolge möglich ist, bei der ständig mindestens einer der beiden Agenten a und c kritisch ist, was es b nicht erlaubt, kritisch zu werden. Ein derartiges Szenario wird in der Literatur manchmal als *Konspiration* (von a und c gegen b) bezeichnet. Im Rest der Arbeit beschäftigen wir uns nun ausführlich mit Konspiration.

6 Konspiration

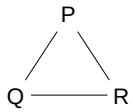
In diesem Kapitel schlagen wir eine Charakterisierung für ein Phänomen in verteilten Systemen vor, für das in der Literatur der Begriff *Konspiration* geprägt wurde. Wir stellen einen Zusammenhang von Konspiration und Ausfalltoleranz dar und erklären mit Konspiration die Unlösbarkeit des ausfalltoleranten allgemeinen Mutex-Problems.

6.1 Charakterisierung von Konspiration

In diesem Abschnitt führen wir Konspiration zunächst informell, dann anhand von Beispielen aus der Literatur ein. Im Unterabschnitt 6.1.3 definieren wir schließlich Konspiration und setzen in Unterabschnitt 6.2 unsere Definition zur Literatur in Beziehung.

Nehmen wir an, Annemarie und Bert gehen zusammen Einkaufen. Sie besuchen dazu ein Kaufhaus mit drei Abteilungen P, Q und R. Nach einer Weile merken Annemarie und Bert, daß sie sich verloren haben, und beschließen, den Einkauf zu unterbrechen, um sich zu suchen.

Annemarie geht zunächst in die Abteilung, von der sie vermutet, daß Bert sich dort aufhält. Sie stellt fest, daß er dort nicht ist, wartet aber noch eine Weile, um Bert die Chance zu geben, zu erscheinen. Da er auch nach einer Weile nicht kommt, versucht sie es mit einer anderen Abteilung. Bert verhält sich nach demselben Prinzip wie Annemarie.



Im Kaufhaus sind je zwei Abteilungen durch genau einen Weg miteinander verbunden. Weiterhin nehmen wir an, daß Annemarie und Bert sich treffen, falls sie auf demselben Weg in unterschiedlichen Richtungen unterwegs sind. Selbst in einem derart idealisierten Kaufhaus kann es passieren, daß sich Annemarie und Bert, obwohl sie sich schon lange kennen, nie treffen. Das ist selbst dann möglich, falls sie immer wieder jede Abteilung besuchen und immer wieder auf jedem Weg in jede Richtung gehen.

Irgendwann gewinnen Annemarie und Bert daß Gefühl, daß die Welt gegen sie *konspiriert*. Warum treffen sich die beiden nicht? Der Kaufhausdetektiv, der das verdächtige Verhalten von Annemarie und Bert auf dem Monitor verfolgt, denkt immer

wieder: „Hätte doch Annemarie jetzt ein bißchen länger gewartet“ und: „Hätte sich Bert doch jetzt für P und nicht für Q entschieden.“

Konspiration, wie wir sie gerade beschrieben haben, ist ein typisches Phänomen von Systemen, die keine zentrale Kontrolle besitzen. Dabei schaffen es zwei oder mehrere Prozesse (im Beispiel: Annemarie und Bert) es nicht, sich zu synchronisieren, weil sie unabhängig voneinander mit unbekannter Geschwindigkeit fortschreiten. Konspiration wurde bisher nur in Systemen untersucht, in denen Agenten entweder gemeinsame Variablen (siehe Abschnitt 6.1.1) oder aber gemeinsame Aktionen (siehe Abschnitt 6.1.2) haben. Wir werden später sehen, daß Konspiration auch in Systemen vorkommt, in denen Agenten ausschließlich über Nachrichten kommunizieren.

In den folgenden beiden Abschnitten stellen wir die beiden Kontexte vor, in denen Konspiration in der Literatur beschrieben wurde.

6.1.1 Die konspirierenden Philosophen

Der Begriff der Konspiration (in verteilten Systemen) wurde von Dijkstra in [30] für das System der *fünf speisenden Philosophen* geprägt. Dort sitzen fünf Philosophen um einen runden Tisch auf dem sich ein großer Topf Spaghetti sowie fünf Gabeln befinden, so daß jeder Philosoph genau zwei Gabeln greifen kann und daß jede Gabel genau zwischen zwei benachbarten Philosophen geteilt wird (vgl. Abb. 6.1). Ein Philosoph denkt entweder oder er ißt Spaghetti. Ein denkender Philosoph kann nur dann beginnen zu essen, falls seine beiden Gabeln verfügbar sind. Zum Essen ergreift ein Philosoph gleichzeitig beide Gabeln, nach dem Essen legt er beide Gabeln gleichzeitig zurück. Auf diese Weise hindert ein essender Philosoph beide seiner Nachbarn daran, mit ihm gleichzeitig zu essen.

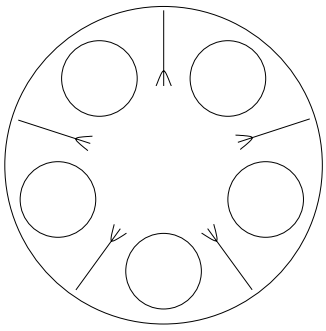


Abb. 6.1: Die fünf Philosophen.

Wir können dieses System der fünf Philosophen damit als Lösungsansatz für das allgemeine Mutex-Problem für eine Menge von fünf Agenten und einen Ring als Nachbarschaftsrelation verstehen. Dijkstra schreibt in [30] zu dieser Lösung:

> Firstly the solution presented is free from the danger of deadlock, as it should be. Yet it is highly improbable that a solution like this can be accepted because it contains possibility of a particular philosopher being starved to death by a conspiration of his two neighbours.

In diesem System der fünf Philosophen kann es also zum Verhungern einzelner Philosophen kommen: Ein Philosoph x verhungert, falls seine beiden Nachbarn derartig abwechselnd essen, daß seine Gabeln nie gleichzeitig auf dem Tisch liegen. Dijkstra sagt, daß die Nachbarn von x gegen x *konspirieren*. Aufgrund der Konspiration ist die Lebendigkeitseigenschaft vom Mutex-Problem verletzt, nämlich daß jeder hungrige Philosoph irgendwann ißt.

Wir wollen hier festhalten, daß die Konspiration gegen den Philosophen x darin besteht, daß eine Transition von x nie aktiviert ist. Wir halten weiterhin fest, daß die Konspiration gegen x aufgrund der Unabhängigkeit der Nachbarn von x zustande kommt. In einem Ring von drei Philosophen gibt es deshalb keine Konspiration. Stellen wir uns vor, daß wir von außen in das System eingreifen können, so können wir die Konspiration verhindern, indem wir einen Nachbarn von x zu einem geeigneten Zeitpunkt für einige Zeit anhalten.

6.1.2 Konspiration in Multiparty-Interaktionen

Das intuitive Verständnis von Konspiration bei den fünf Philosophen wurde im Zusammenhang mit *Multiparty-Interaktionen* wiederverwendet (zum Beispiel von Francez in [37]). Eine *Multiparty-Interaktion* (auch: *Rendevouz*) ist eine gemeinsame, synchrone Aktion mehrerer Agenten. Sie kann nur dann ausgeführt werden, wenn alle beteiligten Agenten gleichzeitig dazu *bereit* sind. Multiparty-Interaktionen sind Bestandteil verschiedener Programmier- und Spezifikationssprachen wie Ada, CSP und LOTOS. Der Kern von Multi-Party-Interaktionen wurde von Chandy und Misra in [24] als *Committee-Coordination-Problem* beschrieben.

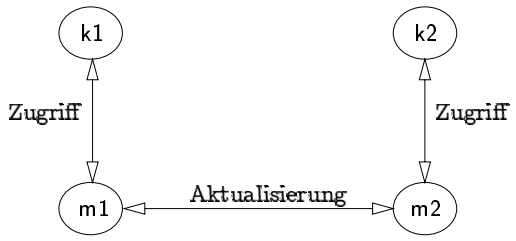


Abb. 6.2: Eine replizierte Datenbank.

Betrachten wir als Beispiel[1] eine replizierte Datenbank mit zwei Managern *m1* und

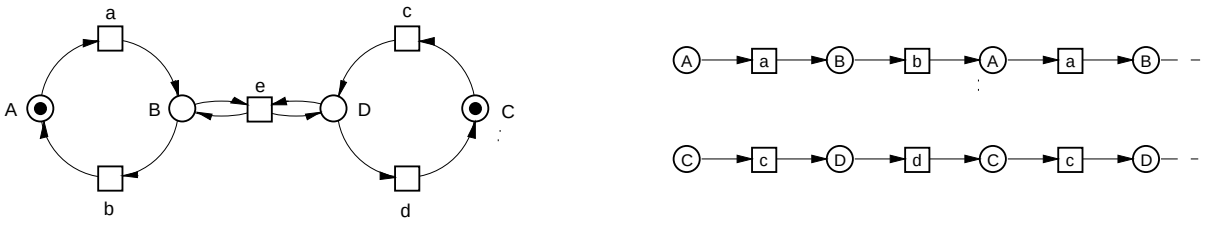
[1] Dieses Beispiel stammt aus [46].

m2 und zwei Klienten *k1* und *k2*, für die wie folgt Interaktionen zugeordnet sind (vgl. Abb. 6.2): Klient *k1* interagiert mit Manager *m1* sowie *k2* mit *m2*, um auf die Datenbank zuzugreifen. Die Manager *m1* und *m2* interagieren miteinander, um die Datenbank konsistent zu halten (Aktualisierung). Hier kann es nun vorkommen, daß eine Aktualisierung nie möglich ist, weil beide Manager derart abwechselnd in Zugriffsoperationen mit ihren Klienten involviert sind, daß beide Manager nie gleichzeitig zu einer Aktualisierung bereit sind. Bei einem solchen Ablauf sprechen beispielsweise Attie, Francez und Grumberg in [9] von einer *Konspiration gegen die Aktualisierung*. Hier wird also nicht gegen einen Agenten, sondern gegen eine Interaktion konspiriert. Auch die Verursacher der Konspiration sind in diesem Fall eher Aktionen als Agenten: Wir können sagen, daß die Zugriffe gegen die Aktualisierung konspirieren.

Nach [9] findet in einem Ablauf eine Konspiration bzgl. einer Interaktion α statt, falls alle Teilnehmer von α, die im Ablauf nicht kontinuierlich zu α bereit sind, immer wieder unabhängig voneinander zu α bereit werden, aber nicht gleichzeitig.

6.1.3 Charakterisierung von Konspiration

In diesem Abschnitt charakterisieren wir Konspiration. Konspiration ist eine Ablaufeigenschaft und bezieht sich auf eine Transition, d.h. wir werden definieren, wann ein Ablauf *konspirativ* bezüglich einer Transition ist. Damit drücken wir sowohl die Konspiration gegenüber Multiparty-Interaktionen als auch die Konspiration gegenüber einem Philosophen aus: Interaktionen sind spezielle Transitionen und eine Konspiration gegenüber einem Philosophen kann als Konspiration bezüglich des Essen-gehens (d.h. des kritisch-werdens) angesehen werden.



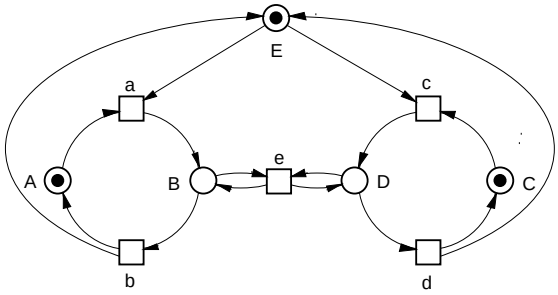
(a) Σ_{28} (b) ρ_{28}

Abb. 6.3: Ein Netzsystem mit konspirativem Ablauf.

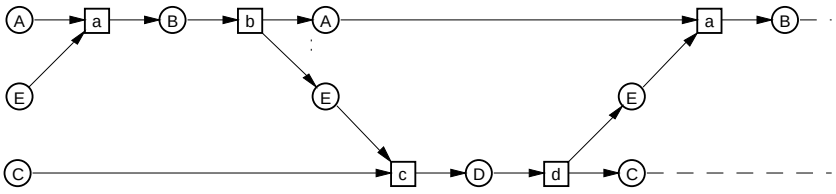
Um Konspiration zu formalisieren, betrachten wir nun ein Netzsystem mit einem Ablauf, in dem Konspiration vorkommt. Σ_{28} in Abb. 6.3 ist ein solches Netzsystem. Es stellt zwei zyklische Agenten dar. Der linke Agent durchläuft die Zustände A und B der rechte Agent durchläuft die Zustände C und D. Ist der linke Agent im Zustand

B und der rechte Agent im Zustand D, so können beide Agenten zusammen die gemeinsame Transition e schalten. Transition e modelliert eine Interaktion zwischen dem linken und dem rechten Agenten[2].

Σ_{28} kann auch als allgemeiner wechselseitiger Ausschluß von drei Agenten a, b und c in einer Nachbarschaftsrelation wie in Abb. 5.3 auf Seite 110 angesehen werden (a und b sind Nachbarn und b und c sind Nachbarn): Eine Marke auf B repräsentiert den Schlüssel zwischen a und b, eine Marke auf D repräsentiert den Schlüssel zwischen b und c. Agent b kann nur dann kritisch werden (Transition e), falls beide Schlüssel verfügbar sind. Jeder Schlüssel kann auch durch das kritisch-werden eines Nachbaragenten von b weggenommen werden (Transition b bzw. d).



(a) Σ_{29}



(b) ρ_{29}

Abb. 6.4: Ein Netzsystem mit nicht-konspirativem Ablauf.

Abb. 6.3(b) zeigt ρ_{28} – den einzigen Ablauf von Σ_{28}, in dem Transition e nie schaltet; ρ_{28} ist fair bzgl. e, da es eine Schaltsequenz σ von ρ_{28} gibt, in der e nie aktiviert ist, nämlich $\sigma = AC(,a,BC,b,AC,c,AD,d,AC)^{\infty}$. Die Schaltsequenz σ repräsentiert eine für die Aktivierung von e ungünstige zeitliche Reihenfolge der Ereignisse von ρ. Es gibt aber auch eine günstige zeitliche Reihenfolge der Ereignisse von ρ:

[2]Transition e können wir uns als Telefongespräch zwischen Annemarie und Bert vorstellen. Eine Marke auf B bedeutet dann „Annemarie ist zu Hause“, eine Marke auf D bedeutet dann „Bert ist zu Hause“. (Annemarie und Bert haben noch kein Mobiltelefon.)

In der Schaltsequenz $\sigma' = AC(, a, BC, c, BD, b, AD, d, AC)^\infty$ von ρ_{28} ist e unendlich oft aktiviert. Bei einem bzgl. t konspirativen Ablauf hängt es also von der zeitlichen Reihenfolge ab, ob t unendlich oft aktiviert ist.

Wir betrachten nun Σ_{29} und einen Ablauf ρ_{29} von Σ_{29} in Abb. 6.4. Gegenüber Σ_{28} hat Σ_{29} eine zusätzliche Stelle E, die dafür sorgt, daß das Verhalten von Σ_{29} sequentiell ist. Die Transitionen a und c schalten in Σ_{29} nicht wie in Σ_{28} unabhängig voneinander, sondern kausal geordnet. In ρ_{29} sind zwar beide Agenten immer wieder zur Interaktion e bereit, jedoch nicht unabhängig voneinander. Dies hat zur Folge, daß es keine Schaltsequenz von ρ gibt, in der e unendlich oft aktiviert ist. Der Ablauf ρ_{29} wird deshalb weder von Attie, Francez und Grumberg noch von uns als bzgl. e konspirativ angesehen[3].

Wir kommen nun zur Formalisierung von Konspiration. Wir basieren unsere Definition von Konspiration bzgl. einer Transition t auf der Beobachtung, daß es von der zeitlichen Reihenfolge abhängt, ob t unendlich oft aktiviert ist. Nach der Definition geben wir eine äquivalente Charakterisierung an.

Definition 6.1 (Konspiration)

Sei Σ ein Netzsystem und t eine Transition von Σ. Ein Ablauf ρ von Σ ist *konspirativ* bzgl. t, falls es sowohl eine Schaltsequenz σ von ρ gibt, in der t unendlich oft aktiviert ist als auch eine Schaltsequenz σ' von ρ gibt, in der t höchstens endlich oft aktiviert ist. Ein Ablauf ist *konspirativ*, falls er bzgl. irgendeiner Transition konspirativ ist. Ein Netzsystem heißt *konspirationsbehaftet*, falls es einen konspirativen Ablauf besitzt und *konspirationsfrei* sonst. ○

Aus Definition 6.1 folgt: Ist ein Ablauf ρ konspirativ bzgl. t, so ist ρ fair bzgl. t und t schaltet in ρ höchstens endlich oft. Eine äquivalente Charakterisierung eines bzgl. t konspirativen Ablauf ρ ist, daß in ρ alle Vorbedingungen von t immer wieder unabhängig voneinander eintreten:

Proposition 6.2 (Äquivalente Charakterisierung von Konspiration)

Sei Σ ein Netzsystem und t eine Transition von Σ. Ein bzgl. t fairer Ablauf ρ von Σ ist konspirativ bzgl. t genau dann wenn für jeden Markierungsschnitt C von ρ ein erreichbarer Markierungsschnitt C′ existiert, der t aktiviert, t aber höchstens endlich oft in ρ vorkommt.

Beweis: Die Richtung $\Leftarrow$ ist trivial. Für $\Rightarrow$ ordne man unabhängige Ereignisse von ρ so, daß die Markierungsschnitte von ρ, die t aktivieren in der Schaltsequenz σ realisiert werden. □

[3] Attie, Francez und Grumberg sprechen von *conspiracy due to race-conditions*.

Die Netzsysteme in den Abbildungen 4.7 und 4.8 auf Seite 88f sind konspirationsfrei. Die Abwesenheit von Konspiration in einem Ablauf ist eine Lebendigkeitsannahme. Wir können daher die Abwesenheit von Konspiration auch als Fairneßbegriff verstehen. Ein Ablauf, der fair bzgl. einer Transition t und nicht konspirativ bzgl. t ist, heißt auch *synchronisationsfair* bzgl. t.

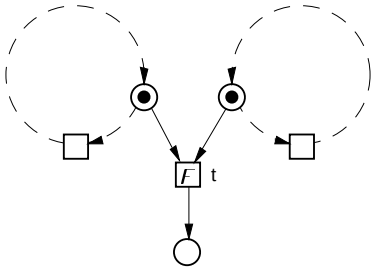


Abb. 6.5: Synchronisationsfairneß

Abb. 6.5 illustriert Synchronisationsfairneß bzgl. t. Synchronisationsfairneß fordert im Gegensatz zu einfacher Fairneß, daß unabhängige rekurrente Ressourcen von t irgendwann durch das Schalten von t synchronisiert werden.

6.1.4 Weitere Beispiele für Konspiration

In diesem Abschnitt lernen wir zwei weitere Beispiele für Konspiration kennen.

Ein typisches Beispiel für Konspiration ist Σ_{30} in Abb. 6.6. Σ_{30} modelliert einen Wettlauf zwischen zwei Prozessen a und b. Prozeß a besteht aus den Transitionen a_1 und a_2, Prozeß b aus b_1, b_2 und b_3. Transition c startet den Wettlauf beider Prozesse um Ressource R. Gewinnt Prozeß a, d.h. schaltet a_2, so wird auch b_3 schalten und ein neuer Wettlauf wird gestartet. Gewinnt Prozeß b den Wettlauf, d.h. schaltet b_2, so sind keine weiteren Transitionen mehr möglich. Ein unendlicher Ablauf von Σ_{30} ist konspirativ bzgl. b_2.

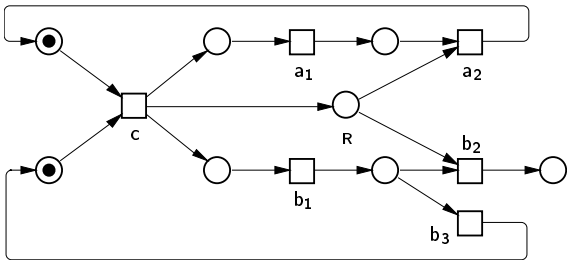


Abb. 6.6: Σ_{30} – Wettlauf zwischen zwei Prozessen.

Konspiration begegnet uns auch im täglichen Leben. So zum Beispiel, wenn man als Autofahrer von einer Nebenstraße nach links in eine Hauptstraße einbiegen möchte

(vgl. Abb. 6.7). Dies ist nur unter zwei Bedingungen möglich: (1) von links kommt kein Auto und (2) von rechts kommt kein Auto. Nun kann es passieren, daß zwar immer wieder eine Seite frei ist, aber nie beide zugleich – dies ist ein konspirativer Ablauf, der verhindert, daß man in die Hauptstraße einbiegen kann. Die nebenläufigen Übergänge „links wird frei“ und „rechts wird belegt“ (und umgekehrt) treten dabei immer wieder in einer ungünstigen Reihenfolge auf.

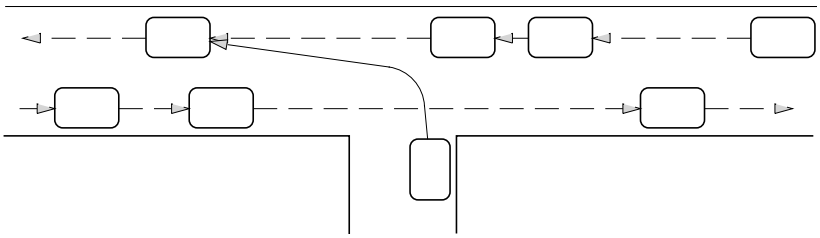


Abb. 6.7: Eine Haupt- und eine Nebenstraße.

6.2 Konspiration in der Literatur

In diesem Abschnitt stellen wir den Stand der Forschung zum Thema Konspiration vor. Eine zufriedenstellende Charakterisierung von Konspiration fehlt bisher in der Literatur. Dijkstra verwendet den Begriff in [30] genau wie Francez in [37] informell. Darüberhinaus beschäftigen sich Best in [17] sowie Attie, Francez und Grumberg in [9] mit Konspiration. Best definiert den Begriff der ∞-*Fairneß*, Attie, Francez und Grumberg definieren den Begriff *Hyperfairneß*, um Konspiration auszuschließen. Beide Arbeiten werden wir im folgenden genauer diskutieren.

6.2.1 ∞-Fairneß

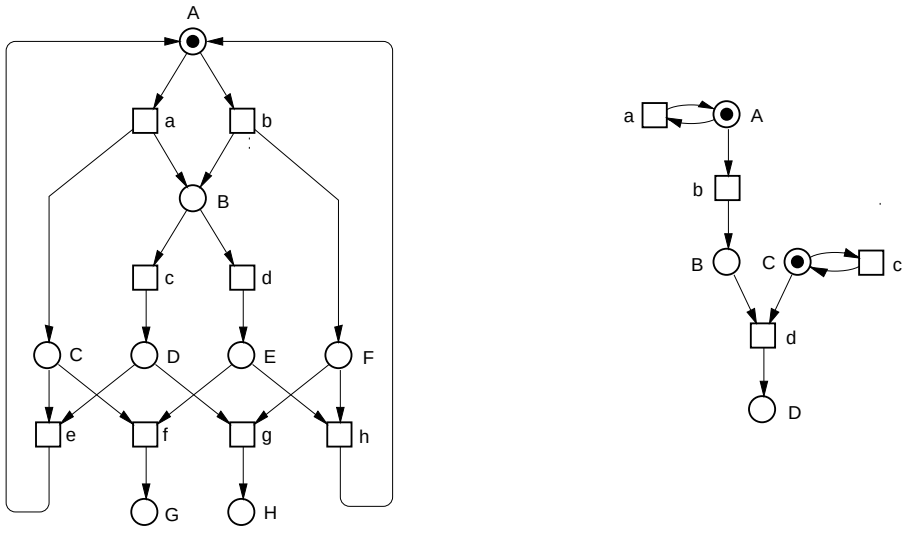
Best behandelt in [17] als erster Konspiration formal. Er definiert die folgende unendliche Hierarchie von Fairneßbegriffen zum Ausschluß von Konspiration.

Definition 6.3 (k-Fairneß, ∞-Fairneß (nach Best))

Seien Σ ein initialisiertes Netz und t eine Transition von Σ. Sei $k \in \mathbb{N}$. Eine Markierung M' von Σ ist von einer Markierung M von Σ k-*erreichbar*, falls es eine Sequenz von Markierungen $M_0, \ldots, M_n$ gibt mit $n \leq k$, so daß $M_0 = M$, $M_n = M'$ und $M_i \rightarrow M_{i+1}$ für alle $i \leq n$. Eine maximale Schaltsequenz σ von Σ ist nicht k-*fair* (bzw. nicht ∞-*fair*), falls σ unendlich viele Positionen i hat, so daß von M_i eine Markierung k-erreichbar (bzw. erreichbar) ist, die t aktiviert, t jedoch höchstens endlich oft in σ schaltet. ○

0-Fairneß ist dasselbe wie Fairneß. In jeder ∞-fairen Schaltsequenz eines initialisierten Netzes Σ schaltet jede lebendige Transition von Σ unendlich oft. Best schreibt in [17]: „∞-fairness indicates the absence of any kind of conspiracy“. Diese Aussage stimmt mit unserer Definition von Konspiration überein: ∞-Fairness ist stärker als Synchronisationsfairneß: Ist ein Ablauf ρ nicht synchronisationsfair bzgl. t, so ist jede Schaltsequenz von ρ nicht ∞-fair bzgl. t. Durch ∞-Fairness werden aber nicht nur konspirative Abläufe ausgeschlossen. Dazu betrachten wir Σ_{31} in Abb. 6.8(a) (Σ_{31} ist das den randomisierten Netzsystemen in Abb. 4.7 auf Seite 88 zugrundeliegende Netzsystem). Ist ρ ein unendlicher Ablauf von Σ_{31}, so gilt: ρ ist nicht konspirativ, jede Schaltsequenz von ρ ist jedoch nicht ∞-fair bzgl. f und g. Ist in einer nicht ∞-fairen, maximalen Schaltsequenz von Σ_{31} Transition f nicht unendlich oft aktiviert, so liegt dies nicht – wie bei Konspiration – an der ungünstigen Reihenfolge unabhängiger Ereignisse, sondern an der nichtdeterministischen Auflösung von Konflikten. Die Annahme von ∞-Fairneß vermischt also den Ausschluß verschiedener Phänomene miteinander.

Best untersucht in [17], wann verschiedene Fairneßbegriffe der Hierarchie zusammenfallen. Aus Definition 6.3 ergibt sich sofort folgendes:



(a) Σ_{31} (b) Σ_{32}

Abb. 6.8: Zwei konspirationsfreie Netzsysteme.

Proposition 6.4

Sei Σ ein initialisiertes Netz und σ eine maximale Schaltsequenz von Σ. Dann gilt:

(a) σ ist ∞-fair $\Rightarrow \forall k \in \mathbb{N} : \sigma$ ist k-fair

(b) σ ist $(k+1)$-fair $\Rightarrow \sigma$ ist k-fair

Die Rückrichtungen der Implikationen in Proposition 6.4 gelten im allgemeinen nicht. Ein Gegenbeispiel für (a) geben wir in Kürze an. Ein Gegenbeispiel für (b) ist Σ_{31}: Eine 0-faire (d.h. faire) Schaltsequenz von Σ_{31}, die unendlich ist, ist nicht 1-fair bzgl. der Transitionen f und g. Best gibt in [17] strukturelle Bedingungen an das Netz an, unter denen auch die Rückrichtungen von Proposition 6.4 gelten. Best beweist das folgende:

Proposition 6.5 (Zusammenfall der k-Fairneß-Hierarchie)

Sei Σ ein initialisiertes Netz und σ eine Schaltsequenz von Σ.

(a) Ist Σ endlich, so gilt: σ ist ∞-fair $\Leftrightarrow \forall k \in \mathbb{N} : \sigma$ ist k-fair.

(b) Ist Σ Extended-Simple, so gilt: σ ist $(k+1)$-fair $\Leftrightarrow \sigma$ ist k-fair.

In unendlichen initialisierten Netzen gilt Proposition 6.5(a) nicht. Ein Gegenbeispiel ist das Netzsystem, das dem randomisierten Netzsystem aus Abb. 4.6 auf Seite 87

zugrunde liegt. Die unendliche Schaltsequenz $\sigma = s_0, s_1, \ldots$ ist für jedes k k-fair, aber nicht ∞-fair. Das Netzsystem Σ_{32} in Abb. 6.8(b) ist ein Extended-Simple-Netz, das Proposition 6.5(b) illustriert: Transition d wird genau dann 1-fair behandelt, falls b und d 0-fair behandelt werden.

In endlichen Extended-Simple-Netzen fällt nach Proposition 6.5 die gesamte Hierarchie zusammen, womit jede faire Schaltsequenz ∞-fair ist. Aus dem Zusammenfall der Hierarchie ergibt sich: Jeder Ablauf eines endlichen Extended-Simple-Netzes, der bzgl. aller Transitionen des Netzes fair ist, ist nicht konspirativ. Kindler und van der Aalst verschärfen in [48] Proposition 6.5(b), indem sie die Aussage auf der größere Netzklasse von *extended asymmetric choice*-Netzen beweisen (siehe auch [1]). Best beweist Proposition 6.5(b) mit einem Standardargument für Extended-Simple-Netze, das wir zum Beweis der folgenden Aussage verwenden, die die Konspirationsfreiheit bei Extended-Simple-Netzen präzisiert und auf unendliche Netze ausdehnt:

Proposition 6.6 (Konspirationsfreiheit für quasi-einfache Transitionen)

Sei Σ ein Netzsystem und t eine Transition von Σ. Ist t quasi-einfach, dann hat Σ keinen bzgl. t konspirativen Ablauf.

Beweis: Wir führen einen Widerspruchsbeweis. Sei ρ ein bzgl. t konspirativer Ablauf von Σ. Dann gibt es eine Schaltsequenz σ von ρ, in der t höchstens endlich oft aktiviert ist. Sei σ' ein Suffix von σ, in dem t nie aktiviert ist. Wegen Proposition 6.2 ist jede Stelle $p \in {}^\bullet t$ immer wieder in σ' markiert. Wegen der Quasi-Einfachheit von t kann ${}^\bullet t$ dargestellt werden als ${}^\bullet t = \{p_1, \ldots, p_n\}$, so daß $i < j \Rightarrow p_i^\bullet \subseteq p_j^\bullet$. Wir zeigen nun durch Induktion über k, daß für jedes k die Menge $\{p_1, \ldots, p_k\}$ immer wieder *simultan* in σ' markiert ist – d.h. alle Stellen der Menge sind in einer Markierung gleichzeitig markiert:

- $k = 1$: Die Menge $\{p_1\}$ ist immer wieder simultan markiert, da jede Stelle von ${}^\bullet t$ immer wieder in σ' markiert ist.
- $k \rightarrow k + 1$: Sei $\{p_1, \ldots, p_k\}$ in σ' immer wieder simultan in σ' markiert. Auch p_{k+1} ist immer wieder in σ' markiert. Da eine Marke auf $\{p_1, \ldots, p_k\}$ erst dann entfernt werden kann, wenn p_{k+1} bereits markiert ist, ist auch $\{p_1, \ldots, p_{k+1}\}$ in σ' immer wieder simultan markiert.

Daraus folgt, daß ${}^\bullet t$ immer wieder simultan in σ' markiert ist, wonach t immer wieder in σ' aktiviert ist – ein Widerspruch zur Ausgangsannahme. □

Die Quasi-Einfachheit einer Transition t ist hinreichend aber nicht notwendig für Konspirationsfreiheit bzgl. t, wie das Netzsystem Σ_{31} zeigt. Da wir in Satz 5.5 gezeigt

haben, daß Konspiration inhärent in manchen Problemen enthalten ist, können wir nun folgern, daß Extended-Simple-Netze nicht immer zur Modellierung verteilter Algorithmen genügen.

6.2.2 Hyperfairneß

Attie, Francez und Grumberg definieren in [9] den Begriff *Hyperfairneß* [4], um Konspiration gegenüber Multi-Party-Interaktionen auszuschließen. Sie betrachten dazu eine CSP-artige Sprache mit Multi-Party-Interaktionen. Hyperfairneß schließt Konspirationen gegenüber Interaktionen aus, die nicht von anderen Interaktionen abhängen (sog. *top-level-Interaktionen*). Da Hyperfairneß ein komplizierter Begriff ist, wollen wir ihn hier nur skizzieren. Attie, Francez und Grumberg definieren zunächst für ein Programm P ihrer Sprache, wann es *konspirationsresistent* ist. Ein Programm P ist *konspirationsresistent*, falls für jede top-level Interaktion t von P und jeden erreichbaren Zustand z von P gilt: Ist A die nicht-leere Menge von Teilnehmern von t, die in z zu t bereit sind, dann verhindert das Einfrieren der Teilnehmer aus A in z nicht, daß alle anderen Teilnehmer irgendwann für t bereit werden. Das bedeutet: Ein Programm P ist genau dann konspirationsresistent, falls in jedem Ablauf von P alle Teilnehmer von t immer wieder zu t bereit sind und falls sie immer nur unabhängig voneinander zu t bereit werden. Ob ein Programm konspirationsresistent ist, hängt also vom Gesamtverhalten des Programms ab. Die Formalisierung von Konspirationsresistenz in [9] ist stark sprachabhängig, da dort versucht wird, Unabhängigkeit auf sequentiellen Abläufen zu definieren.

Hyperfairneß von sequentiellen Abläufen wird in [9] in Abhängigkeit von der Konspirationsresistenz des Programms wie folgt definiert: Ist P nicht konspirationsresistent, so ist jeder sequentielle Ablauf von P hyperfair. Ist P konspirationsresistent, so ist ein unendlicher sequentieller Ablauf σ von P genau dann hyperfair, wenn jede top-level-Interaktion in σ unendlich oft aktiviert ist und stark fair behandelt wird. Jeder endliche Ablauf von P ist hyperfair. Wir halten die folgenden Schwächen von Hyperfairneß aus [9] fest:

1. Der Begriff ist stark sprachabhängig, insbesondere wird Konspiration nicht allgemein, sondern nur in Bezug auf Multi-Party-Interaktionen betrachtet.
2. Es wird nur Konspiration bzgl. top-level-Interaktionen ausgeschlossen und nur dann falls das Programm konspirationsresistent ist.
3. Ob ein Ablauf hyperfair ist, hängt vom Gesamtverhalten des Programms ab.

Zusammen mit Hyperfairneß wird in [9] ein Scheduler vorgeschlagen, der Hyperfairneß implementiert. Damit kann man Programme unter Hyperfairneß entwerfen

[4] verschieden von Lamports Hyperfairneß [57]

und verifizieren und bei der Ausführung des Programms durch den Scheduler wird zugesichert, daß jeder Ablauf hyperfair ist. Der Vorteil dieser Herangehensweise ist die komfortable Nutzbarkeit des ausdrucksstarken Programmierkonstrukts Multi-Party-Interaktion. Das Problem des in [9] vorgeschlagenen Schedulers ist allerdings, daß dieser bei Ausführung eines Programms, das nicht konspirationsresistent ist, Deadlocks erzeugen kann. Für einen solchen Fall schlagen Attie, Francez und Grumberg Deadlock-Erkennung vor. Eine alternativer Ausweg ist die syntaktische Erkennung von Konspirationsresistenz. Die Durchführbarkeit beider Auswegmöglichkeiten bleibt in [9] offen.

Wir beschäftigen uns in Kapitel 7 mit der allgemeinen Implementation von Konspirationsfreiheit. Eine Implementation von ∞-Fairneß ist nicht bekannt und vermutlich unmöglich.

Lamport kritisiert Hyperfairneß von Attie, Francez und Grumberg als „completely language-dependent“ [55]. Lamport definiert daraufhin in [57] einen eigenen, sprachunabhängigen Begriff „Hyperfairneß“, der allerdings nichts anderes ist als ∞-Fairneß von Best.

Das allgemeine Problem der in diesem und im vorangegangenen Abschnitt vorgestellten Arbeiten ist es, daß jeweils versucht wird, Konspiration auf Grundlage von sequentiellen Abläufen auszuschließen: Im Abschnitt 6.1 haben wir gesehen, daß die Unabhängigkeit verschiedener Ressourcen zentrale Eigenschaft von Konspiration ist. In einem internen Papier [16] vermutet Best, daß es sinnvoll ist, nicht-sequentielle Abläufe zu betrachten, um Konspirationen auszuschließen. Dies Idee wurde jedoch nie weiterverfolgt. Synchronisationsfairneß wurde von Reisig in [78] als *Quasifairneß* und von Merceron in [65] als *0-Transitionsfairneß* als theoretische Möglichkeit, Fairneß auf nicht-sequentiellen Abläufen zu definieren, untersucht. Ein Zusammenhang zu Konspiration wurde jeweils nicht hergestellt.

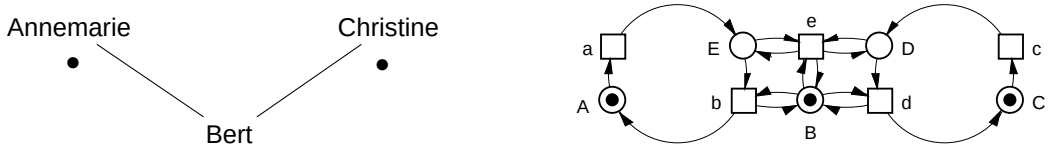
6.3 Konspiration und Ausfalltoleranz

In diesem Abschnitt betrachten wir einige Beispiele für den Zusammenhang von Konspiration und Ausfalltoleranz, den wir in Abschnitt 5.2.3 kennengelernt haben. In der Literatur ist Konspiration bisher nur in den beiden Kontexten behandelt worden, die wir in Abschnitten 6.1.1 und 6.1.2 vorgestellt haben. Die Verwendung von Konspiration zum Verständnis fehlertoleranter Algorithmen ist neu.

In Unterabschnitt 6.3.1 betrachten wir noch einmal das ausfalltolerante allgemeine Mutex-Problem, in Unterabschnitt 6.3.2 das Konsens-Problem und in Unterabschnitt 6.3.3 ein weiteres Problem.

6.3.1 Ausfalltoleranter allgemeiner Mutex

Wir beschäftigen uns hier noch einmal mit dem ausfalltoleranten allgemeinen Mutex-Problem für drei Agenten in einer Nachbarschaftsrelation wie in Abb. 5.3 auf Seite 110. Um Verwechslungen mit Transitionsnamen vorzubeugen, nennen wir hier die Agenten Annemarie, Bert und Christine[5] . Wir nehmen (wie schon früher einmal) an, daß es für jedes Paar von Nachbarn genau einen Schlüssel gibt, der jetzt jedoch nicht eine gemeinsame Variable darstellt, sondern mittels Nachrichtenaustausch zwischen den Nachbarn versendet wird. Abb. 6.9 zeigt unsere drei Agenten in der angenommenen Nachbarschaftsrelation mit den Schlüsseln, die sich anfangs bei Annemarie und Christine befinden. Nehmen wir an, Bert ist hungrig und möchte kritisch wer-



(a) (b) Σ_{33}

Abb. 6.9: Drei Agenten mit zwei Schlüsseln.

den und fordert dazu beide Schlüssel an. Erhält Bert den Schlüssel von Annemarie, so benötigt er nur noch den Schlüssel von Christine. Bert weiß leider nicht, ob er den Schlüssel von Christine bekommen wird, da diese möglicherweise ausgefallen ist. Fordert nun Annemarie ihren Schlüssel zurück, so befindet sich Bert in einer Konfusionssituation: Soll er den Schlüssel Annemarie zurückgeben oder noch eine Weile auf den Schlüssel von Christine warten? Wartet Bert mit der Rückgabe

[5] Für Annemarie, Bert und Christine bedeutet kritisch sein vielleicht, auf einer der im gemeinsamen Freundeskreis immer wieder stattfindenden Parties zu sein. Im Beispiel möchten Annemarie und Bert sowie Bert und Christine nicht gemeinsam auf Parties sein.

des Schlüssels an Annemarie bis er den Schlüssel von Christine bekommt, so wartet er beim Ausfall von Christine unendlich lange und Annemarie bekommt den Schlüssel überhaupt nicht zurück – eine Verletzung der Lebendigkeitseigenschaft vom ausfalltoleranten allgemeinen Mutex-Problem. Gibt Bert jedoch den Schlüssel an Annemarie zurück bevor er den Schlüssel von Christine bekommt, so kann er nicht kritisch werden und möglicherweise kommt nun der Schlüssel von Christine, woraufhin Bert in eine Konfusionssituation kommen kann, die symmetrisch zu der gerade erlebten Konfusionssituation ist. Bei unendlicher Iteration dieses Verhaltens entsteht ein konspirativer Ablauf.

Das Netzsystem Σ_{33} in Abb. 6.9(b) stellt das gerade beschriebene Verhalten vereinfacht dar. Im Zentrum von Σ_{33} ist Bert, der auf Stelle E den Schlüssel von Annemarie und auf Stelle D den Schlüssel von Christine empfangen kann. Nur wenn Bert beide Schlüssel hat, kann er kritisch werden (Transition e). Mit Transition b gibt Bert den Schlüssel an Annemarie, mit Transition d gibt Bert den Schlüssel an Christine zurück. Abb. 6.10 zeigt einen bzgl. e konspirativen Ablauf ρ_{33} von Σ_{33} mit den Markierungsschnitten, die e aktivieren.

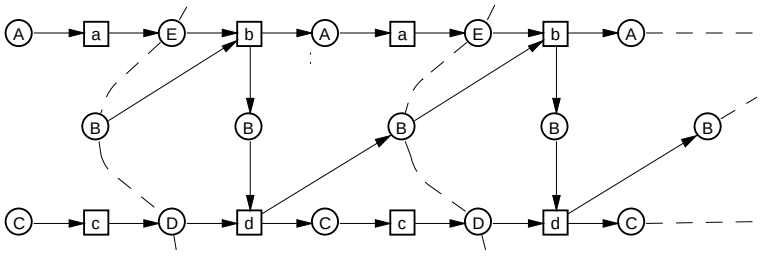


Abb. 6.10: ρ_{33}

Der Beweis von Satz 5.5 zeigt, daß Konspiration inhärent im ausfalltoleranten allgemeinen Mutex-Problem enthalten ist. Satz 5.4 zeigt darüberhinaus, daß auch durch Randomisierung Konspiration in diesem Problem nicht ausgeschlossen werden kann.

6.3.2 Ausfalltoleranter Konsens

Auch in asynchronen Konsensalgorithmen kommt Konspiration vor. Betrachten wir beispielsweise den kleinen Konsensalgorithmus aus Abschnitt 2.4.3. Abb. 6.11 zeigt Agent a mit Wert 0 im Zustand *schwankend*. Bekommt a nun eine Änderungsnachricht von rechts, so befindet er sich in einer ähnlichen Konfusionssituation wie Bert im vorigen Abschnitt: Soll Agent a die Änderungsnachricht akzeptieren oder auf eine Nachricht von links warten mit der er entscheiden kann? Jeder unendliche Ablauf des kleinen Konsensalgorithmus ist konspirativ bzgl. $t_2(x, v)$ für jeden Agenten x und jeden Wert v.

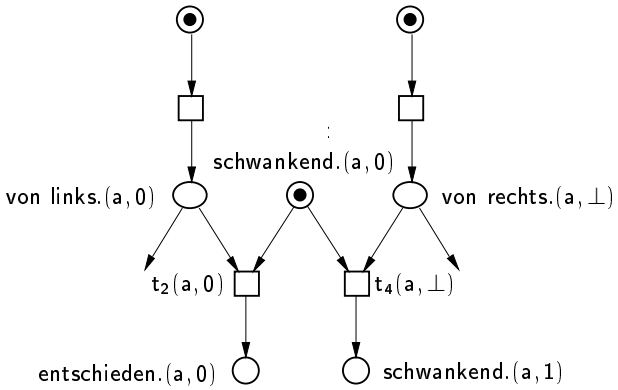


Abb. 6.11: Konspiration im kleinen Konsensalgorithmus.

6.3.3 Ein weiteres Problem

Gegeben sei eine Datenbank mit einem Manager *m* und zwei verteilten Klienten *k1* und *k2*, die jeweils mit einer Transaktion t1 bzw. t2 auf dasselbe Datum der Datenbank schreibend zugreifen wollen (Abb. 6.12). Nehmen wir an, daß Klient *k1* eine Sperre (*lock*) zum Schreiben bekommt. Schreitet *k1* nun nicht schnell genug voran, so vermutet der Datenbankmanager, daß *k1* ausgefallen ist und bricht die Transaktion t1 ab, um *k2* das Schreiben zu ermöglichen. Ist *k1* nicht ausgefallen, so fordert *k1* erneut eine Sperre beim Datenbankmanager an, was möglicherweise dazu führt, daß t2 abbgebrochen wird, falls *k2* zu langsam fortschreitet. Eine unendliche Iteration stellt eine Konspiration gegen den Abschluß beider Transaktionen dar.

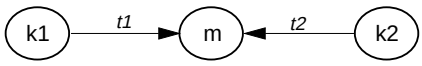


Abb. 6.12: Eine Datenbank mit zwei langsamen Klienten.

Dieses Beispiel zeigt, daß Time-Outs in asynchronen Systemen Konspiration verursachen können.

7 Konspirationsfreiheit

In diesem Kapitel beschreiben wir, wie Konspirationsfreiheit in vielen Fällen durch Fairneß, Randomisierung und *Quasisynchronie* implementiert werden kann. *Quasisynchronie* ist eine schwache Synchronieannahme.

In Abschnitt 7.1 implementieren wir Konspirationsfreiheit zunächst nur bezüglich einer einzelnen Transition. In Abschnitt 7.2 implementieren wir dann Konspirationsfreiheit bezüglich beliebig vieler Transitionen.

7.1 Konspiration bezüglich einer Transition

Wir werden uns bei der Implementierung von Konspirationsfreiheit auf eine bestimmte Klasse von Konspiration zurückziehen, nämlich auf *beschränkte Konspiration.* In 7.1.1 definieren wir beschränkte Konspiration, in 7.1.2 definieren wir Quasisynchronie, in 7.1.3 zeigen wir dann, wie durch Quasisynchronie und Fairneß beschränkte Konspiration bzgl. einer Transition implementiert werden kann.

7.1.1 Beschränkte und unbeschränkte Konspiration

In einem bzgl. t konspirativen Ablauf ρ gibt es immer wieder Markierungsschnitte, die t aktivieren. Genauer: Ist C ein Markierungsschnitt von ρ, dann kommen wir

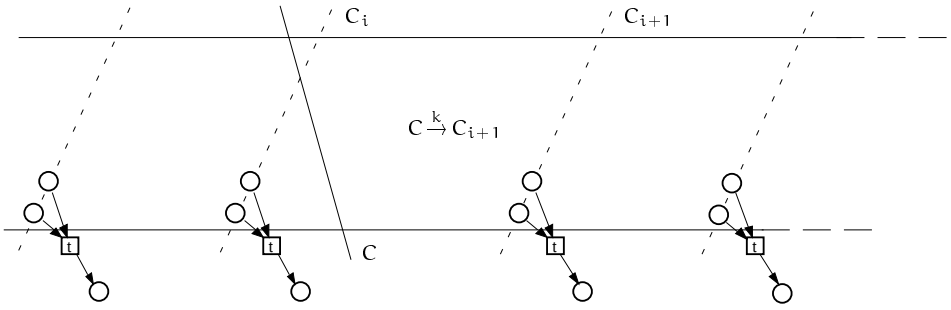


Abb. 7.1: Beschränkte Konspiration.

durch endlich viele Ereignisse von C zu einem Markierungsschnitt, der t aktiviert.

Seien dies k Ereignisse für C (vgl. Abb. 7.1). Betrachten wir k als Aufwand, um t zu aktivieren, so wollen wir nun Abläufe, bei denen dieser Aufwand beschränkt ist, von Abläufen unterscheiden, bei denen dieser Aufwand unbeschränkt wächst. Ist der Aufwand, t zu aktivieren, beschränkt, so nennen wir den Ablauf *beschränkt konspirativ* bzgl. t und *unbeschränkt konspirativ* bzgl. t sonst. Dies formalisieren wir wie folgt.

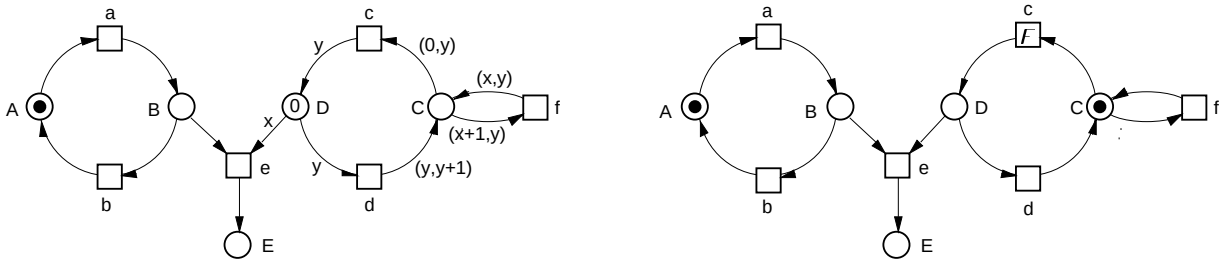
Definition 7.1 (Beschränkte Konspiration)

Sei Σ ein Netzsystem und t eine Transition von Σ. Sei ρ ein Ablauf von Σ mit Ereignismenge E und C ein Markierungsschnitt von ρ. Sei $k \in \mathbb{N}$. Ein von C aus erreichbarer Markierungsschnitt C' von ρ ist *von* C *aus* k-*erreichbar* (Notation: $C \xrightarrow{k} C'$), falls $|\{e \in E \mid \exists b \in C, b' \in C' : b < e < b'\}| \leq k$. Ein bzgl. t konspirativer Ablauf ρ von Σ ist k-*konspirativ* bzgl. t, falls von jedem Markierungsschnitt C von ρ ein von C aus k-erreichbarer Markierungsschnitt C' existiert, der t aktiviert; ρ heißt *beschränkt konspirativ* bzgl. t, falls ein k existiert, so daß ρ k-konspirativ bzgl. t ist und *unbeschränkt konspirativ* bzgl. t sonst. ○

Aus Definition 7.1 ergibt sich sofort folgendes:

Proposition 7.2

Sei Σ ein Netzsystem, t eine Transition von Σ und ρ ein bzgl. t k-konspirativer Ablauf. Dann ist jede Schaltsequenz σ von ρ nicht k-fair.



(a) Σ_{34} (b) Σ_{35}

Abb. 7.2: Unbeschränkte Konspiration.

Alle bisher in dieser Arbeit gezeigten konspirativen Abläufe sind beschränkt konspirativ. Abb. 7.2(a) zeigt Netzsystem Σ_{34}, dessen einziger unendlicher Ablauf unbeschränkt konspirativ bzgl. *e* ist. Bei einem bzgl. t unbeschränkt konspirativen Ablauf werden die Markierungsschnitte, die t aktivieren, immer seltener. Jeder unendliche faire Ablauf von Σ_{35} in Abb. 7.2(b) ist konspirativ bzgl. *e*. Einige Abläufe davon sind beschränkt, andere unbeschränkt konspirativ.

7.1.2 Quasisynchronie

Ein Postulat, das eine Aussage über die relative Geschwindigkeit von Systemkomponenten (Agenten, Kanäle) trifft, heißt *Synchronieannahme*. Bisher haben wir keine Synchronieannahmen getroffen: Alle bisher in dieser Arbeit verwendeten Modelle sind asynchron. Ein *synchrones* System ist ein System, in dem die relativen Geschwindigkeiten aller Komponenten durch eine bekannte Konstante K beschränkt sind. Das Wissen um diese Schranke kann zur Implementation von sicheren Timeouts verwendet werden. In einem synchronen System können deshalb Ausfälle erkannt werden: Dabei sendet der beobachtete Agent in regelmäßigem Abstand Nachrichten an den Beobachter. Empfängt der Beobachter irgendwann keine Nachricht mehr, so kann er sich irgendwann des Ausfalls des beobachteten Agenten sicher sein.

Auch bei *Quasisynchronie* sind die relativen Geschwindigkeiten aller Komponenten durch eine Konstante K beschränkt, die jedoch – im Gegensatz zu Synchronie – nicht bekannt ist. Da diese Schranke nicht bekannt ist, erlaubt Quasisynchronie keine sicheren Timeouts und damit auch keine Erkennung von Ausfällen.

Quasisynchronie ist in der Literatur unter den Namen *partial synchrony* und *unknown bounded delay* bekannt. Quasisynchronie wurde vielfältig verwendet, um „bessere“ Algorithmen zu erhalten: Unter Annahme von Quasisynchronie können sogar Probleme gelöst werden, die sonst unlösbar sind: In [32] zeigen Dwork, Lynch und Stockmeyer, daß Quasisynchronie eine Lösung des Konsens-Problems mit deterministischen Agenten ermöglicht. Alur, Attiya und Taubenfeld zeigen in [6], daß Quasisynchronie eine effizientere Lösung von Konsens und Mutex in Architekturen mit gemeinsamen Speicher erlaubt. Joung and Liao verwenden Quasisynchronie in [46], um mit Randomisierung starke Fairneß für Multi-Party-Interaktionen zu implementieren.

Wir wollen nun Quasisynchronie formalisieren. Uns wird es weiterhin genügen, Schaltsequenzen als Approximation von Zeit zu verwenden, d.h. wir wollen definieren, wann eine Schaltsequenz eines Ablaufs quasisynchron ist. Stellen wir uns zunächst zwei zyklische Agenten vor, die nicht miteinander kommunizieren, etwa wie in Abb. 2.1 auf Seite 40. Wir nehmen nun an, daß die relative Geschwindigkeit beider Agenten durch eine Konstante K beschränkt ist – diese Annahme nennen wir K-*Synchronie*. Dies bedeutet: Schaltet ein Agent in einem Zeitraum $K+1$ mal dann schaltet der andere Agent mindestens einmal. Die Schaltsequenz $c, (a, b)^3, d$ vom Netzsystem in Abb. 2.1 auf Seite 40 ist damit 6-synchron, aber nicht 5-synchron.

Dies läßt sich wie folgt auf beliebige Systeme verallgemeinern: Sei dazu ρ ein Ablauf und σ eine Schaltsequenz von ρ. Sei weiterhin M_i eine Markierung von σ und C_i der zugehörige Markierungsschnitt in ρ. Dann bedeutet K-Synchronie: Schreitet eine in C_i beginnende Kausalkette $K+1$ Ereignisse in σ voran, dann schreitet jede andere in C_i beginnende, von der ersten unabhängige Kausalkette mindestens ein Ereignis

voran. Um dies zu formalisieren, definieren wir zunächst den *Abstand* eines Ereignisses e von einem Markierungsschnitt C. Dies ist die Länge der längsten Kausalkette von C zu e.

Definition 7.3 (Abstand)

Sei ρ ein Ablauf, C ein Markierungsschnitt und e ein Ereignis von ρ. Der *Abstand* $\Delta(C, e)$ von e zu C ist definiert durch

$$\Delta(C, e) = \max \{|E'| \mid E' \subseteq E \text{ ist li-Menge mit } \exists b \in C : \forall e' \in E' : b < e' \leq e\}$$

○

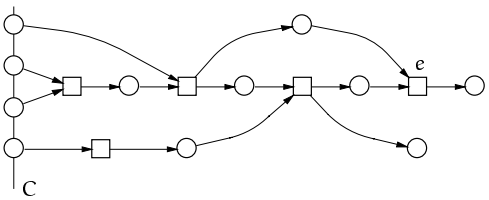


Abb. 7.3: $\Delta(C, e) = 4$

Ist e von C nicht erreichbar, so ist $\Delta(C, e) = 0$. Ist e in C aktiviert, so ist $\Delta(C, e) = 1$. Weiterhin gilt für alle Markierungsschnitte C und alle Ereignisse e: $\Delta(C, e) = 1 + \max \{\Delta(C, e') \mid e' < e\}$. Abb. 7.3 gibt ein Beispiel für den Abstand. Wir definieren nun K-Synchronie und Quasisynchronie.

Definition 7.4 (Quasisynchronie)

Sei ρ ein Ablauf und sei $K \in \mathbb{N}$. Eine Sequentialisierung $\tau = C_0, e_1, C_1, \ldots$ von ρ heißt K-*synchron*, falls für alle Positionen n, i, j von τ mit $n < i < j$ gilt:

$$\frac{\Delta(C_n, e_i)}{\Delta(C_n, e_j)} \leq K \tag{7.1}$$

Eine Schaltsequenz $\sigma = \sigma_\tau$ von ρ ist K-*synchron*, falls sie aus einer K-synchronen Sequentialisierung τ hervorgeht; σ heißt *quasisynchron*, falls ein K existiert, so daß σ K-synchron ist. ○

Abb. 7.4 illustriert die Situation in Definition 7.4. Nach Definition 7.4 gilt insbesondere: Ist e im Markierungsschnitt C_n aktiviert, so finden höchstens K Ereignisse einer zu e nebenläufigen Kette vor e statt.

Wir haben Quasisynchronie mit Definition 7.4 schwächer als in der Literatur definiert, da wir nur die Existenz einer Schranke für jede Schaltsequenz, nicht aber die Existenz einer globalen Schranke für alle Schaltsequenzen des Systems gefordert haben. Diese schwächere Definition genügt uns für die weiteren Ausführungen.

Aus Definition 7.4 ergibt sich die folgende Proposition.

Proposition 7.5

Sei ρ ein Ablauf und σ eine Schaltsequenz von ρ. Dann gilt:

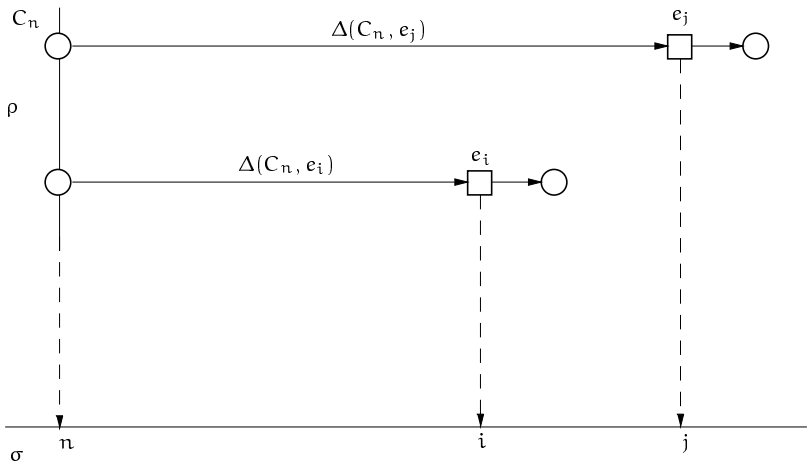


Abb. 7.4: Quasisynchronie.

(a) σ ist K-synchron $\Rightarrow$ σ ist $(K+1)$-synchron.

(b) Zu jedem $K > 0$ gibt es eine K-synchrone Schaltsequenz von ρ.

Beweis: Aussage (a) ist trivial. Für (b) genügt nach (a) zu zeigen, daß es eine 1-synchrone Schaltsequenz gibt. Um eine 1-synchrone Schaltsequenz von ρ zu erhalten, schaltet man in jedem Markierungsschnitt ein Ereignis mit minimalem Abstand zum Anfangsschnitt. □

Da jeder Ablauf eine quasisynchrone Schaltsequenz besitzt, wird durch Quasisynchronie allein noch kein Verhalten, d.h. kein Ablauf ausgeschlossen. Der Ausschluß von Abläufen erfolgt erst durch die Verbindung von Quasisynchronie und Fairneß. Wir definieren nun, wann ein Ablauf *unter Quasisynchronie fair* ist.

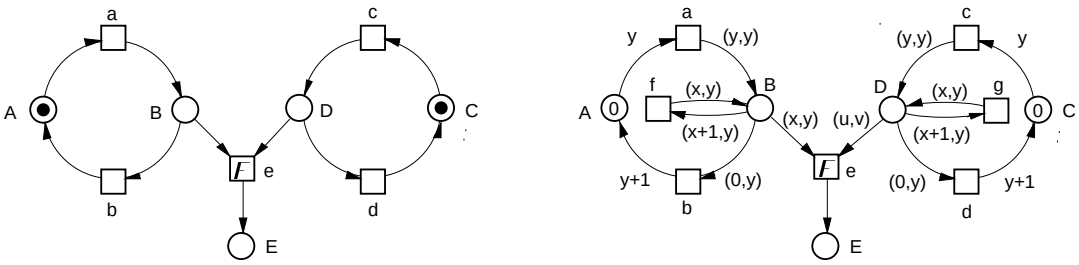
Definition 7.6 (Fairneß unter Quasisynchronie)

Sei Σ ein Netzsystem, ρ ein Ablauf und t eine Transition von Σ. Ein Ablauf ρ von Σ ist nicht *unter Quasisynchronie fair bzgl.* t falls alle quasisynchronen Schaltsequenzen von ρ nicht fair bzgl. t sind. ○

Beispiele für Fairneß unter Quasisynchronie lernen wir im nächsten Abschnitt kennen. Alur und Henzinger definieren in [7] Begriffe, die den in diesem Unterabschnitt definierten Begriffe ähnlich sind. Sie benötigen dazu die Annahme, daß ein System nur endlich viele Transitionen besitzt. Quasisynchronie kann in unserer Formalisierung als Verallgemeinerung des Begriffs *finitary weak fairness* aus [7] auf Systeme mit unendlich vielen Transitionen angesehen werden.

7.1.3 Der Nutzen von Quasisynchronie

Wir wollen in diesem Unterabschnitt ein faires Netzsystem betrachten, in dem Fairneß unter Quasisynchronie stärker als Fairneß ist. Σ_{36} in Abb. 7.5(a) kennen wir so ähnlich bereits. Σ_{37} in Abb. 7.5(b) ist eine Verfeinerung von Σ_{36}. Hinzugekommen ist für jeden Kreis ein Zähler, der mitzählt, wie oft Stelle B bzw. D schon markiert war. Wird Stelle B markiert, so wird die Marke auf B durch Transition f festgehalten, beim ersten Mal schaltet f 0-mal, beim zweiten Mal schaltet f 1-mal usw. Transitionen f und g dienen allein dem Zeitverbrauch – sie können als Uhren angesehen werden. Transitionen b und d können als Timeouts angesehen werden. Dieses Verfahren – auf eine Bedingung immer länger zu warten – heißt auch *adaptiver Timeout*.



(a) Σ_{36} (b) Σ_{37}

Abb. 7.5: Adaptiver Timeout.

Σ_{37} hat genau wie Σ_{36} einen unendlichen fairen Ablauf. Aber: Jeder unter Quasisynchronie faire Ablauf von Σ_{37} ist endlich. Um dies zu sehen, sei ρ der einzige unendliche Ablauf von Σ_{37} und σ eine K-synchrone, faire Schaltsequenz von ρ. Da σ fair ist, gibt es einen Suffix σ' von σ, in dem e nie aktiviert ist. Sei M_i ein Zustand von σ' in dem eine Marke (y,y) mit $y > K$ auf B liegt. Da e nicht in M_i aktiviert ist, ist nicht D, sondern C markiert. Da e auch nicht mehr in σ' aktiviert wird, schaltet b vor c – ein Widerspruch zur K-synchronie von σ.

Wie unsere Argumentation zeigt, genügt zur Termination von Σ_{37} ein adaptiver Timeout an einer von beiden Ressourcen von e. An welcher Ressource der adaptive Timeout angebracht wird, ist dabei egal. Für Σ_{38} in Abb. 7.6 kann man das Schalten von e durch Fairneß unter Quasisynchronie erzwingen, falls man adaptive Timeouts an B und E anbringt oder aber ausschließlich an B. Ein adaptiver Timeout ausschließlich an E garantiert das Schalten von e nicht.

Betrachten wir noch einmal Σ_{37} in Abb. 7.5(b). Falls Transition e in einem Ablauf nicht schaltet, sorgt der adaptive Timeout für immer größere Bereiche im Ablauf, so daß e in jedem Markierungsschnitt des Bereiches aktiviert ist (vgl. Abb. 7.7, ein

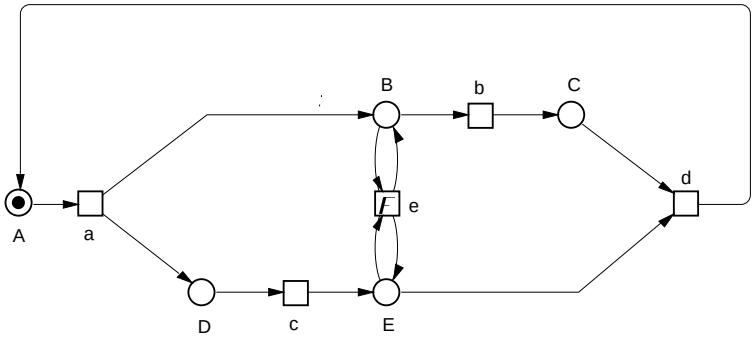


Abb. 7.6: Σ_{38}

Bereich ist grau dargestellt). Quasisynchronie sorgt dafür, daß jede Schaltsequenz irgendwann immer wieder Markierungsschnitte in diesen Bereichen realisiert.

Möchte man beschränkte Konspiration bzgl. einer Transition eines beliebigen Systems ausschließen, so kann man einen adaptiven Timeout verwenden. Wir können also bereits jetzt das ausfalltolerante allgemeine Mutex-Problem für drei Agenten in einer Reihe lösen. Möchte man Konspiration bzgl. mehrerer Transitionen eines Systems ausschließen, so funktioniert dieses Verfahren im allgemeinen nicht mehr. Warum, sehen wir im nächsten Abschnitt.

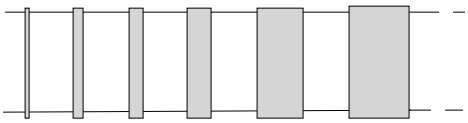


Abb. 7.7: Ablaufstruktur bei adaptivem Timeout.

7.2 Konspiration bezüglich mehrerer Transitionen

In diesem Abschnitt zeigen wir, wie Konspirationsfreiheit bezüglich mehrerer Transitionen erzielt werden kann.

7.2.1 Adaptive Timeouts an mehreren Transitionen

Im vorigen Abschnitt 7.1.3 haben wir gesehen, wie das ausfalltolerante allgemeine Mutex-Problem mittels Quasisynchronie und Fairneß für drei Agenten in einer Reihe gelöst werden kann. Leider kann man auf diese Weise dieses Problem nicht für beliebige Nachbarschaftsrelationen lösen. Betrachten wir dazu einen Ring von vier Agenten (Abb. 7.8) mit vier Schlüsseln, die durch Nachrichten versendet werden. Wir stellen uns nun vor, daß jeder Agent für jeden Schlüssel einen adaptiven Timeout verwendet, um irgendwann beide Schlüssel gleichzeitig zu besitzen. Dann gibt es einen unter Quasisynchronie fairen Ablauf, in dem kein hungriger Agent kritisch wird. In diesem Ablauf hat jeder Agent immer einen Schlüssel und alle Agenten halten ihren Schlüssel dieselbe Zeit fest. Die Schlüssel bewegen sich gemeinsam wie die Unruhe eines Uhrwerks.

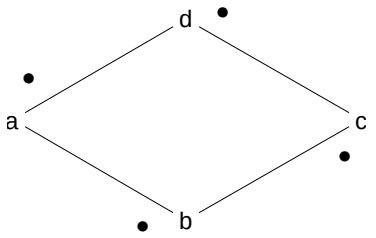


Abb. 7.8: Ein Ring von vier Agenten.

Das Problem an diesem Lösungsansatz scheint die Symmetrie des Systems zu sein. Genauer sehen wir das Problem in Σ_{39} in Abb. 7.9: Σ_{39} hat zwei konspirationsgefährdete Transitionen, nämlich a und b, die miteinander um zwei verschiedene Marken konkurrieren. Das Festhalten einer Marke auf F führt zur Verzögerung der Ankunft der Marke auf D, woraufhin das Festhalten der Marke auf C nichts nützt. Diese Symmetrie läßt sich beispielsweise dadurch brechen, daß ein adaptiver Timeout nur für C und E, hingegen nicht für D und F verwendet wird. Im obigen Beispiel der vier Agenten verwende man einen adaptiven Timeout für den Schlüssel zwischen a und b sowie für den Schlüssel zwischen c und d.

Wir lassen es offen, ob auf diese Weise für jede Nachbarschaftsrelation unter Quasisynchronie eine Lösung für das ausfalltolerante allgemeine Mutex-Problem gefunden werden kann. Für die Implementierung von Konspirationsfreiheit für ein beliebiges System vermuten wir, daß Quasisynchronie im allgemeinen nicht ausreicht. Ein Indiz

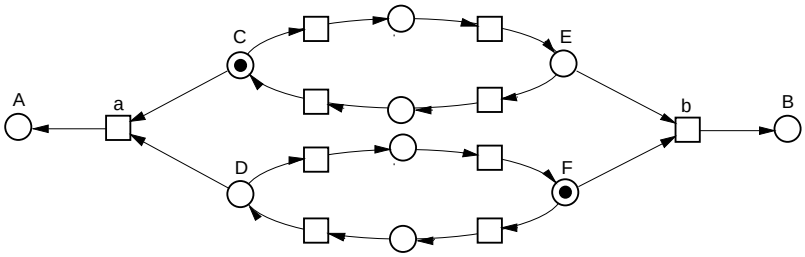


Abb. 7.9: Σ_{39} – Zwei konkurrierende konspirationsgefährdete Transitionen.

dafür ist Σ_{40} in Abb. 7.10. In Σ_{40} sind die Transitionen d und f konspirationsgefährdet. Zur Implementierung von Konspirationfreiheit bzgl. d und f können wir an den Stellen B und C adaptive Timeouts anbringen. Um die Symmetrie zu brechen, dürfen wir nicht an beiden Stellen adaptive Timeouts verwenden. Entscheiden wir uns aber beispielsweise für B, so wird es weiterhin Konspiration bzgl. f geben. Durch eine feste Verteilung von adaptiven Timeouts auf einige Stellen eines Netzes erhält man also im allgemeinen keine Konspirationsfreiheit[1].

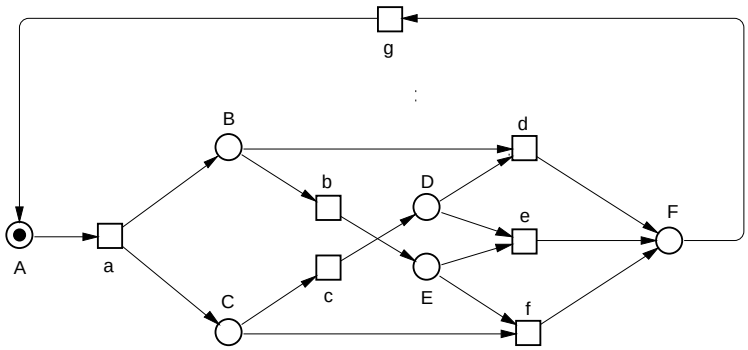


Abb. 7.10: Σ_{40} – Noch zwei konkurrierende konspirationsgefährdete Transitionen.

7.2.2 Randomisierte Timeouts

Wir wissen bereits, daß durch Randomisierung Symmetrie gebrochen werden kann. In diesem Abschnitt zeigen wir nun, daß durch Randomisierung und Quasisynchronie Konspirationsfreiheit gewährleistet werden kann, falls unbeschränkte Konspiration ausgeschlossen ist. Dazu verbinden wir Randomisierung mit adaptiven Timeouts. Um Randomisierung mit adaptiven Timeouts zu verbinden, gibt es zwei Möglichkeiten:

[1] Denkbar ist hier nun eine dynamische Verteilung adaptiver Timeouts, d.h. manchmal wird B und manchmal wird C festgehalten. Um dabei eine korrekte Lösung zu erhalten, muß man dann allerdings das Gesamtverhalten des Systems kennen – selbst dann gibt es vermutlich nicht immer eine Lösung.

a) Ob eine Marke festgehalten wird, wird durch Münzwurf entschieden.

b) Wie lange (eigentlich: wie oft) eine Marke festgehalten wird, wird durch Münzwurf entschieden.

Beide Möglichkeiten führen zum Ziel. Wir führen im folgenden b) näher aus.

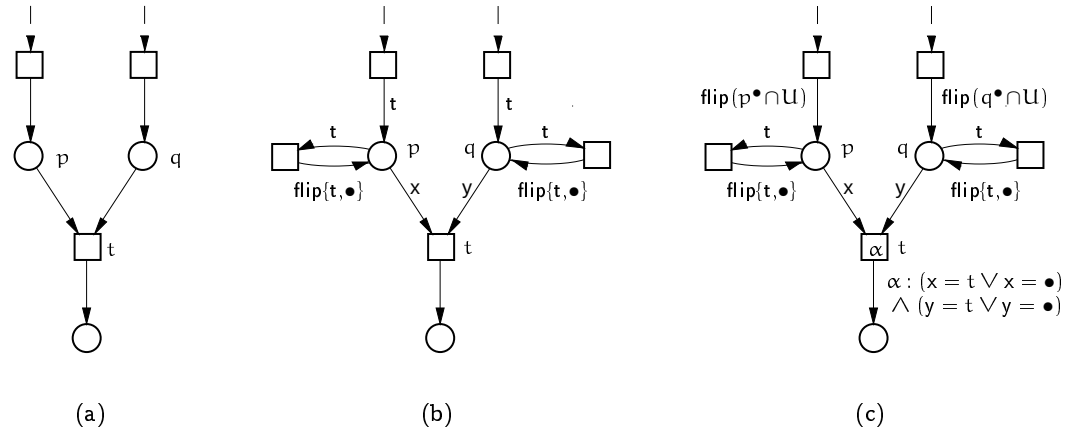


Abb. 7.11: Verfeinerung von $t \in U$.

Gegeben sei ein konspirationsbehaftetes Netzsystem Σ und eine endliche Menge U von (konspirationsgefährdeten) Transitionen von Σ, so daß es für alle $t \in U$ keinen bzgl. t unbeschränkt konspirativen Ablauf gibt. Wir konstruieren nun ein randomisiertes Netzsystem $\dot{\Sigma}$ wie folgt. Dabei nehmen wir zunächst an, daß die Vorbereiche aller Transitionen aus U disjunkt sind. Die Umgebung jeder Transition $t \in U$ wie in Abb. 7.11(a) verfeinern wir wie in Abb. 7.11(b). Betrachten wir einen Vorplatz p von t. Wird in Σ eine schwarze Marke auf p gelegt, dann wird in $\dot{\Sigma}$ eine Marke t auf p gelegt, wodurch diese Marke für Transition t reserviert wird, d.h. nur t kann diese Marke verbrauchen. Ein ggf. wiederholter Münzwurf entscheidet dann, ob die Marke auf p weiterhin für t reserviert bleibt, oder ob sie für alle Transitionen aus $p^\bullet$ freigegeben wird. Im ersten Fall sagen wir, daß die Marke *gehalten* wird. Im letzteren Fall wird eine schwarze Marke auf p gelegt. Die Inschrift flip{t,•} bedeutet, daß der Münzwurf die beiden Ausgänge t und $\bullet$ hat (Zur genauen Bedeutung von flip siehe Abschnitt 4.2). Durch diese Konstruktion gibt es für jedes $k > 0$ eine positive Wahrscheinlichkeit dafür, daß die Marke auf p k Mal hintereinander gehalten wird.

Wir nehmen nun an, daß die Vorbereiche der Transitionen aus U nicht notwendig disjunkt sind. Dann verfeinern wir $t \in U$ wie in Abb. 7.11(c). Dabei wird eine Marke auf p am Anfang nicht notwendig für t reserviert, sondern es wird durch Münzwurf entschieden, für welche der Transitionen aus $p^\bullet \cap U$ die Marke reserviert

wird. Die Inschrift flip($p^\bullet \cap U$) bedeutet, daß jedes Element aus $p^\bullet \cap U$ ein Ausgang des Münzwurfes ist.

Wir zeigen nun, daß dieses Verfahren zum Ziel führt. Dabei sei ein probabilistischer Ablauf π *unter Quasisynchronie fair bzgl.* t, falls jeder Ablauf von π unter Quasisynchronie fair bzgl. t ist.

Satz 7.7 (Probabilistische Konspirationfreiheit unter Quasisynchronie)

Sei Σ ein Netzsystem und U eine Menge von Transitionen von Σ, so daß für alle Stellen p von Σ die Menge $p^\bullet \cap U$ endlich ist und so, daß Σ für kein $t \in U$ einen bzgl. t unbeschränkt konspirativen Ablauf hat. Sei $\dot{\Sigma}$ das randomisierte Netzsystem, das aus Σ entsteht, indem die Umgebung jeder Transition $t \in U$ wie in Abb. 7.11(c) verfeinert wird. Dann ist jeder unter Quasisynchronie faire probabilistische Ablauf von $\dot{\Sigma}$ mit Wahrscheinlichkeit 1 bzgl. aller $t \in U$ konspirationsfrei.

Beweis: Sei φ, die Abbildung, die jeden Ablauf ρ von $\dot{\Sigma}$ auf einen Ablauf $\varphi(\rho)$ von Σ durch Entfernung der probabilistischen Ereignisse abbildet.

Sei $t \in U$ und sei π ein unter Quasisynchronie fairer probabilistischer Ablauf von $\dot{\Sigma}$. Sei ρ ein maximaler Ablauf von π, der bzgl. t konspirativ ist. Dann ist auch $\varphi(\rho)$ konspirativ bzgl. t. Da $\varphi(\rho)$ nach Voraussetzung nicht unbeschränkt konspirativ bzgl. t ist, gibt es ein k, so daß $\varphi(\rho)$ k-konspirativ bzgl. t ist. Sei σ eine quasisynchrone Schaltsequenz von ρ. Dann gibt es ein K, so daß σ K-synchron ist. Da ρ konspirativ bzgl. t ist, gibt es einen Suffix von σ, in dem t nicht vorkommt. Da σ fair bzgl. t ist, gibt es einen Suffix σ' von σ in dem t nie aktiviert ist.

Für zwei Markierungschnitte C, D von ρ bezeichne $\delta_1(C, D)$ die Anzahl der nicht-probabilistischen Ereignisse von ρ zwischen C und D (also die Anzahl der Ereignisse von $\varphi(\rho)$ zwischen $\varphi(C)$ und $\varphi(D)$) und $\delta_2(C, D)$ bezeichne die Anzahl der Ereignisse von ρ zwischen C und D.

Sei n eine Position von σ' und C_n der zugehörige Markierungsschnitt von ρ. Da $\varphi(\rho)$ k-konspirativ bzgl. t ist, gibt es einen von C_n erreichbaren Markierungsschnitt D_n von ρ, der t aktiviert, so daß $\delta_1(C_n, D_n) \leq k$. Wir charakterisieren nun, wann die Münzwürfe hinter C_n günstig für die Aktivierung von t ausgegangen sind: Seien für C_n und D_n die folgenden drei Bedingungen (vgl. Abb. 7.12) erfüllt:

1. alle Ressourcen von t sind in D_n für t reserviert,
2. keine Ressource einer Transition $t' \in U$ wird zwischen C_n und D_n gehalten, d.h. jeder Halte-Münzwurf zwischen C_n und D_n hat den Ausgang $\bullet$. Damit gibt es zu jedem nicht-probabilistischen Ereignis e zwischen C_n und D_n im direkten Anschluß von e höchstens zwei probabilistische Ereignisse, womit $\delta_2(C_n, D_n) \leq 3k$ gilt.

3. im Anschluß von D_n wird jede Ressource von t mindestens $(3k+1)\cdot K$ für t gehalten.

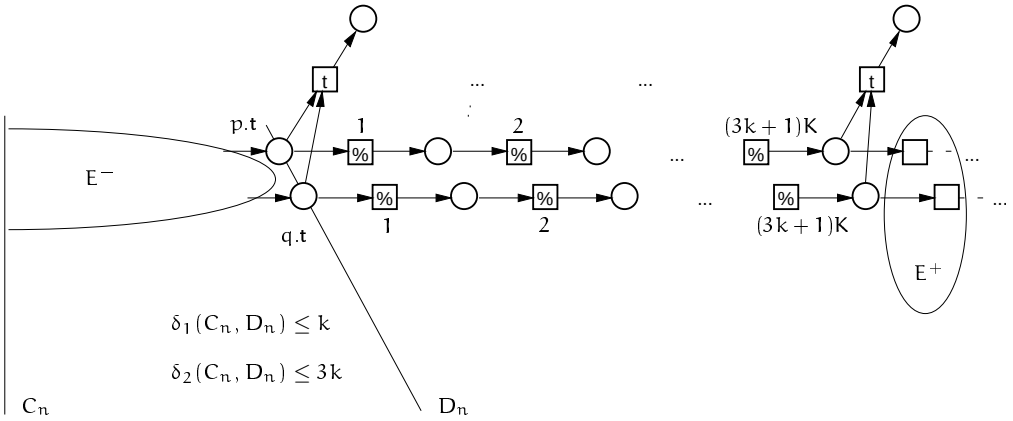


Abb. 7.12: Ablauf ρ

Sei E^- die Menge der Ereignisse von ρ, die kausale Vorgänger der Ressourcen von t in D_n sind und sei E^+ die Menge der Ereignisse von ρ, die direkte kausale Nachfolger der $(3k+1)\cdot K$-ten Halteereignisse für t hinter D_n sind (vgl. Abb. 7.12). Sei e^- das Ereignis aus E^-, welches als letztes aller Ereignisse von E^- in σ' eintritt und e^+ das Ereignis aus E^+, welches als erstes aller Ereignisse von E^+ in σ' eintritt. Da t in σ' nicht aktiviert ist, tritt e^+ vor e^- in σ' ein.

Wegen $\Delta(C_n, e^-) \leq \delta_2(C_n, D_n) \leq 3k$ und $\Delta(C_n, e^+) \geq \Delta(D_n, e^+) \geq (3k+1)\cdot K$ gilt:

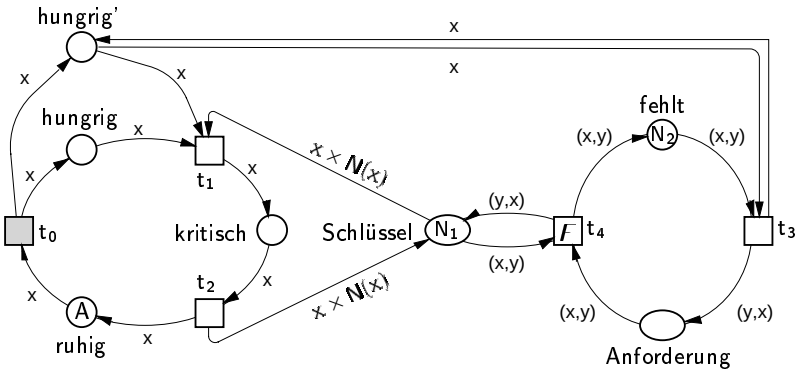
$$\frac{\Delta(C_n, e^+)}{\Delta(C_n, e^-)} \geq K + \frac{K}{3k} > K$$

– ein Widerspruch zur K-Synchronie von σ. Demzufolge gibt es keine Markierungsschnitte C_n, D_n in ρ die 1.–3. erfüllen.

Nun gibt es aber für jeden Markierungsschnitt C von π und jede natürliche Zahl i eine feste positive Wahrscheinlichkeit, daß von C aus die nächsten i Münzwürfe günstig für t ausgehen. Dann gibt es mit Wahrscheinlichkeit 1 in ρ immer wieder Markierungsschnitte C_n für die die nächsten $2k+(3k+1)K$ Münzwürfe günstig für t ausgehen, womit dann die Bedingungen 1. bis 3. erfüllt sind. Es folgt, daß π mit Wahrscheinlichkeit 1 konspirationsfrei bzgl. t ist. Da U abzählbar ist, ist π dann auch mit Wahrscheinlichkeit 1 konspirationsfrei bzgl. aller $t \in U$. □

Mit Satz 7.7 erhalten wir eine Lösung für das ausfalltolerante allgemeine Mutex-Problem. Sei A eine Menge von Agenten und N eine Nachbarschaftsrelation auf A. Unser Ausgangspunkt ist eine Standardlösung mit Schlüsseln, die über Nachrichten versendet werden; Σ_{41} zeigt ein Petrinetzmodell: Für jedes Paar von Nachbarn

$(x,y) \in N$ gibt es genau einen Schlüssel, der sich immer entweder bei x befindet (modelliert durch eine Marke (x,y) auf der Stelle *Schlüssel*) oder der sich bei y befindet (modelliert durch eine Marke (y,x) auf der Stelle *Schlüssel*). Um kritisch zu werden, benötigt ein Agent x alle Schlüssel, die er mit seinen Nachbarn teilt (Transition t_1). Ist x hungrig, so fordert er seine fehlenden Schlüssel bei den entsprechenden Nachbarn an (Transition t_3). Erhält x eine Anforderung von y, so kann er den Schlüssel an y abgeben (Transition t_4).



$N_1 + N_2 = N, \quad (x,y) \in N_1 \Leftrightarrow (y,x) \in N_2$

Abb. 7.13: Σ_{41} – Konspirationsbehafteter ausfalltoleranter allgemeiner Mutex.

Σ_{41} erfüllt die Sicherheitseigenschaft des allgemeinen Mutex-Problems: Zwei Nachbarn sind nie gleichzeitig kritisch. Desweiteren erfüllt jeder konpirationsfreie Ablauf von Σ_{41} die Lebendigkeitseigenschaft des allgemeinen Mutex-Problems. Da es keine unbeschränkten Konspirationen in Σ_{41} gibt, ist die Lebendigkeitseigenschaft in dem durch randomisierte Timeouts verfeinerten System unter Quasisynchronie und Fairneß probabilistisch gültig. Wir halten fest:

Folgerung 7.8 (Lösung von ausfalltolerantem allgemeinen Mutex)

Für jede endliche Menge A von Agenten und jede Nachbarschaftsrelation N auf A gibt es ein faires randomisiertes Netzsystem für A, das sowohl Mutex-Struktur für A als auch unter Quasisynchronie ausfalltolerantes allgemeines Mutex-Verhalten besitzt.

So wie es in der Absicht von Hyperfairneß in [9] lag, können wir nun Synchronisationsfairneß als Abstraktion von einer Implementierung verwenden. Dazu verifiziere man ein faires Netzsystem unter Synchronisationsfairneß und implementiere es später durch adaptiven Timeout und, wenn nötig, durch Randomisierung. Um dabei Satz 7.7 anwenden zu können, schwächen wir Synchronisationsfairneß auf k-*Synchronisationsfairneß* ab: Ein Ablauf ρ ist nicht k-*synchronisationsfair* bzgl. einer Transition t, falls t höchstens endlich oft schaltet in ρ und falls für jeden

Markierungsschnitt C von ρ ein von C aus k-erreichbarer Markierungsschnitt C' existiert, der t aktiviert.

Eine Beweisregel für Synchronisationsfairneß ist $\Box\Diamond{}^\bullet t \Rightarrow \Box\Diamond t^\bullet$, eine Beweisregel für k-Synchronisationsfairneß $\Box\Diamond_k{}^\bullet t \Rightarrow \Box\Diamond t^\bullet$, wobei $\Diamond_k \Phi$ bedeutet, daß Φ innerhalb von k Schritten erfüllt werden kann. Dershowitz und Jayasimha schlagen in [25] eine temporale Logik auf Schaltsequenzen mit einem ähnlichen Operator vor.

Literaturbezug

Synchronisationsfairneß ist eine Abstraktion, die man mit Hilfe von Quasisynchronie implementieren kann. Andere Abstraktionen aus der Literatur, die durch Synchronieannahmen implementiert werden sind *Fehlerdetektoren* [23] und *wait-free objects* [41]. Ein *Fehlerdetektor* ist ein Gerät, das jeder Agent in jedem Zustand fragen kann, ob ein anderer Agent ausgefallen ist. Antwortet der Fehlerdetektor immer korrekt, so ist er *perfekt*. Wie wir bereits dargestellt haben, kann in synchronen Systemen ein perfekter Fehlerdetektor implementiert werden. Um das Konsens-Problem zu lösen, muß ein Fehlerdetektor nicht perfekt sein. Chandra, Hadzilacos und Toueg geben in [21] die schwächsten Eigenschaften eines Fehlerdetektors zur Lösung des Konsens-Problem an. Sie zeigen, daß dieses Eigenschaften durch Quasisynchronie implementiert werden können.

Ein *nebenläufiges Objekt* ist ein gemeinsames Objekt mehrerer Agenten. Ein Agent kann über eine Schnittstelle *Operationen* auf dem Objekt durchführen. Ein nebenläufiges Objekt ist *wait-free*, falls jeder Agent jede Operation nach endlicher Zeit abschließt – unabhängig von der Geschwindigkeit anderer, auf das Objekt zugreifender Agenten. Insbesondere schließt ein Agent eine Operation auf einem wait-free-Objekt auch dann nach endlicher Zeit ab, falls ein anderer, auf das Objekt zugreifender Agent ausfällt. Hinter den Eigenschaften eines Objektes können nun Fairneß- und Synchronieannahmen einer Implementierung verborgen werden. Die typische Fragestellung für nebenläufige Objekte ist: Gegeben zwei Objekte X und Y, gibt es eine Implementation von X durch Y, die wait-free ist? Durch Beantwortung dieser Frage können verschiedene Fairneßannahmen zueinander in Beziehung gesetzt werden. Das Konsens-Problem spielt eine zentrale Rolle in der Untersuchung nebenläufiger Objekte. In [66] wurde auch das Mutex-Problem untersucht.

Neiger zeigt in [67] detailliert Beziehungen von Fehlerdetektoren und wait-free objects auf. Randomisierung wurde bisher nicht in Fehlerdetektoren und wait-free objects einbezogen. Dies ist jedoch prinzipiell möglich, wie Attiya und Welch das für den Fall von wait-free objects in [10] anmerken.

Abschließende Bemerkungen

Im ersten Teil der Arbeit haben wir die Beziehung von Fairneß und Randomisierung untersucht. Wir haben gezeigt, daß Fairneß und Randomisierung bezüglich ihrer Ausdrucksstärke unvergleichbar sind. Dabei haben wir zwei Aspekte von Fairneß isoliert – freie Fairneß und einfache Fairneß. Während Randomisierung ausdrucksstärker als freie Fairneß ist, sind einfache Fairneß und Randomisierung bezüglich ihrer Ausdrucksstärke unvergleichbar.

Wir haben sowohl Nutzen als auch Grenzen von Randomisierung in verteilten Systemen demonstriert. Der Nutzen von Randomisierung besteht in der synchronisationsfreien Koordinierung verteilter Entscheidungen. Durch den Verzicht auf Synchronisation kann dabei, wie bei Ben-Or's Konsensalgorithmus, Ausfalltoleranz erzielt werden.

Die Einführung von Randomisierung in ein Modell bleibt immer dann wirkungslos auf die Lösbarkeit eines Problems, falls das Problem Synchronisation verlangt, die das zugrundeliegende Modell nicht leisten kann. Diesen Effekt haben wir sowohl beim Mutex-Problem als auch beim ausfalltoleranten allgemeinen Mutex-Problem gesehen.

Das ausfalltolerante allgemeine Mutex-Problem verlangt dabei eine stärkere Synchronisation als das Mutex-Problem. Diese stärkere Synchronisation kann durch Synchronisationsfairneß erreicht werden. Insgesamt bietet sich das Bild in Abbildung 7.14. Ein Pfeil bedeutet dabei: „ist ausdrucksstärker als".

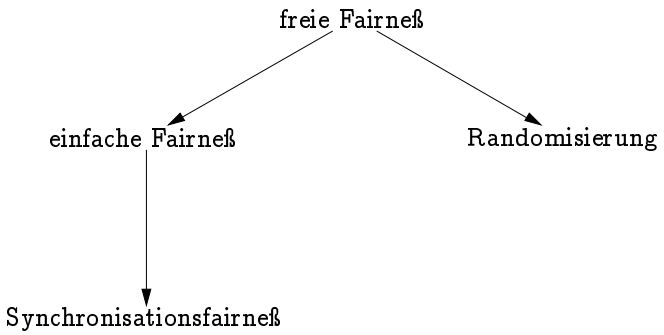


Abb. 7.14: Beziehungen der Modellparameter.

Im zweiten Teil der Arbeit haben wir Konspiration charakterisiert. Wir haben gezeigt, daß Konspiration ein elementares Phänomen ist, das in verschiedenen Problemen und Architekturen auftritt. Dabei haben wir erstmals einen Zusammenhang von Konspiration und Ausfalltoleranz dargestellt. Aus der Inhärenz von Konspiration im ausfalltoleranten allgemeinen Mutex-Problem folgt, daß Extended-Simple Netze nicht ausreichen, um alle verteilten Algorithmen zu modellieren.

Ein Postulat von Synchronisationsfairneß ist in realen Systemen möglicherweise zu stark. Wir haben in der Arbeit gezeigt, daß durch adaptiven und randomisierten Timeout in den meisten Fällen unter Quasisynchronie Konspirationsfreiheit erreicht werden kann.

Desweiteren haben wir in der Arbeit eine nicht-sequentielle Semantik für randomisierte verteilte Algorithmen vorgeschlagen. Eine genauere Untersuchung dieser Semantik ist im Lichte der Diskussion in Abschnitt 4.1.7 lohnenswert. Diese neue Semantik legt die Grundlage für die Integration randomisierter Algorithmen in die Methodik DAWN[2] [81, 91, 49, 90, 28] zur Modellierung und Verifikation verteilter Algorithmen.

In der formalen Verifikation verteilter Systeme ist es wichtig, ein semantisch einfaches Modell zu verwenden. Die klassischen Lebendigkeitsannahmen wie Progreß und Fairneß sind semantisch einfach – man kann mit ihnen beweistechnisch gut umgehen. Wir haben gesehen, daß man beim Übergang von einfachen zu schwierigeren Problemen, insbesondere Fehlertoleranzproblemen, schnell an die Grenzen von Progreß und Fairneß stößt. Die Hinzunahme von Zeit und Wahrscheinlichkeit zum Modell machen ein Modell semantisch komplex und eine Analyse schwer. Neue, stärkere Lebendigkeitsannahmen sind hier gesucht. Zu den existierenden Ansätzen der Literatur finitary fairness [7], wait-free objects [41] und Fehlerdetektoren [23] bietet Synchronisationsfairneß eine interessante Alternative.

Einige Ergebnisse dieser Arbeit, wie die Unlösbarkeit des Mutex-Problems in randomisierten Netzsystemen sowie die Charakterisierung von Konspiration, wurde erst durch die Verwendung nicht-sequentieller Semantik möglich. Diese Arbeit stützt damit die These, daß nicht-sequentielle Semantik wesentlich zu einer besseren Intuition für verteilte Systeme beiträgt.

[2] Distributed Algorithms Working Notation

Anhang

A Beweise

A.1 Konstruktion des Wahrscheinlichkeitsraumes für probabilistische Abläufe

Satz A.1 (Satz 4.4)

Sei $\dot{\Sigma}$ ein randomisiertes Netzsystem und π ein probabilistischer Ablauf von $\dot{\Sigma}$. Sei $\Omega = \mathfrak{R}_{\max}(\pi)$, und zu jedem endlichen Ablauf α von π sei $K(\alpha) = \{\rho \in \mathfrak{R}_{\max}(\pi) \mid \alpha \sqsubseteq \rho\}$ die Menge aller maximalen Abläufe von π, die α fortsetzen. Desweiteren sei $\mathcal{E} = \{K(\alpha) \mid \alpha \in \mathfrak{R}_{\text{fin}}(\pi)\}$. Dann gibt es genau einen Wahrscheinlichkeitsraum $(\Omega, \mathcal{A}, P)$, so daß $\mathcal{A} = \sigma(\mathcal{E})$, d.h. $\mathcal{A}$ ist die von $\mathcal{E}$ erzeugte σ-Algebra und daß für alle endlichen Abläufe α von π gilt:

$$P(K(\alpha)) = p(\alpha) \tag{A.1}$$

Um Satz 4.4 zu beweisen, führen wir zunächst in Abschnitt A.1.1 die notwendigen Grundbegriffe der Maßtheorie ein, auf die wir uns stützen werden. In den darauf folgenden Abschnitten konstruieren wir den gesuchten Wahrscheinlichkeitsraum.

A.1.1 Grundbegriffe der Maßtheorie

Die folgenden Begriffe und Zusammenhänge sind [13] entnommen. Sie können aber auch in jedem anderen Lehrbuch zur Maßtheorie nachgelesen werden. (Vgl. auch Abschnitt 1.6.)

Sei Ω eine Menge. Für eine Menge $A \subseteq \Omega$ bezeichnet im folgenden $\overline{A} = \Omega \setminus A$ ihr Komplement. Eine *Mengenalgebra*[1] über Ω ist ein Mengensystem $\mathcal{M} \subseteq 2^{\Omega}$, das unter Komplement und endlicher Vereinigung abgeschlossen ist und für das $\Omega \in \mathcal{M}$ gilt[2]. Mengenalgebren sind wie σ-Algebren unter Durchschnitt abgeschlossen. Daher existiert für ein Mengensystem $\mathcal{E} \subseteq 2^{\Omega}$ genau eine kleinste Mengenalgebra, die $\mathcal{E}$ enthält.

[1] in [13]: *Algebra*.

[2] Wir erinnern uns, daß eine σ-Algebra darüberhinaus auch unter abzählbarem Durchschnitt abgeschlossen ist.

Sei $\mathcal{E} \subseteq 2^\Omega$ ein Mengensystem über Ω. Eine Abbildung $P : \mathcal{E} \to \mathbb{R} \cup \{\infty\}$ heißt *Mengenfunktion*, falls $P(A) \geq 0$ für alle $A \in \mathcal{E}$. Eine Mengenfunktion P heißt *additiv*, falls für jede endliche, paarweise disjunkte Familie $\mathcal{F} \subseteq \mathcal{E}$ gilt:

$$P(\bigcup_{A \in \mathcal{F}} A) = \sum_{A \in \mathcal{F}} P(A). \tag{A.2}$$

P heißt σ-*additiv*, falls (*A*.2) auch für abzählbare, paarweise disjunkte Familien $\mathcal{F}$ gilt. Eine auf einer Mengenalgebra $\mathcal{M}$ definierte additive Mengenfunktion P heißt *Inhalt* auf $\mathcal{M}$, falls

$$P(\varnothing) = 0. \tag{A.3}$$

Ein σ-additiver Inhalt auf $\mathcal{M}$ heißt *Prämaß* auf $\mathcal{M}$. Ein Prämaß P auf $\mathcal{M}$ ist *endlich*, falls für jedes $A \in \mathcal{M} : P(A) < \infty$.

Satz A.2 (Fortsetzungssatz)

Jedes endliche Prämaß P auf einer Mengenalgebra $\mathcal{M}$ besitzt eine eindeutige Fortsetzung auf $\sigma(\mathcal{M})$, d.h. es existiert genau ein Maß P' auf $\sigma(\mathcal{M})$ mit $P'(A) = P(A)$ für alle $A \in \mathcal{M}$.

Ein auf einer σ-Algebra $\mathcal{A}$ definiertes Prämaß heißt *Maß* auf $\mathcal{A}$. Ein Maß P, für das

$$P(\Omega) = 1 \tag{A.4}$$

gilt, heißt *Wahrscheinlichkeitsmaß*.

Wir konstruieren den gesuchten Wahrscheinlichkeitraum in zwei Schritten. Zunächst konstruieren wir im folgenden Abschnitt A.1.2 eine Mengenalgebra $\mathcal{M}$ über Ω, die $\mathcal{A}$ erzeugt. Danach konstruieren wir in Abschnitt A.1.3 ein endliches Prämaß auf $\mathcal{M}$. Nach dem Fortsetzungssatz existiert dann eine eindeutige Fortsetzung des Prämaßes auf $\sigma(\mathcal{M}) = \mathcal{A}$.

A.1.2 Konstruktion der Mengenalgebra

Sei im folgenden π, Ω und $\mathcal{E}$ wie in Satz 4.4 fixiert. In diesem Abschnitt konstruieren wir die kleinste Mengenalgebra, die $\mathcal{E}$ enthält. Wir stellen zunächst fest, daß $\Omega = K(\alpha_0) \in \mathcal{E}$, wobei α_0 den ereignislosen Ablauf von π bezeichnet.

Definition A.3 (Kegel)

Eine Menge $K(\alpha) \in \mathcal{E}$ sowie die leere Menge bezeichnen wir auch als *Kegel*. Mit $\mathcal{K} = \mathcal{E} \cup \{\varnothing\}$ bezeichnen wir das Mengensystem aller Kegel. ∘

Wir beweisen im folgenden einige Aussagen über Kegel. Aus der Definition der Kompatibilität folgt:

Proposition A.4

Seien $\alpha, \beta \in \mathfrak{R}_{\text{fin}}(\pi)$. Dann gilt

$$\alpha \not\parallel \beta \Leftrightarrow K(\alpha) \cap K(\beta) = \varnothing.$$

Lemma A.5

Der Durchschnitt zweier Kegel ist ein Kegel.

Beweis: Ist einer der beiden zu schneidenden Kegel die leere Menge, so ist die Aussage trivial. Seien nun $K(\alpha), K(\beta) \in \mathcal{E}$ zwei nicht-leere Kegel. Dann gilt entweder $\alpha \not\parallel \beta$ oder $\alpha \parallel \beta$. Im ersten Fall folgt $K(\alpha) \cap K(\beta) = \varnothing$, im zweiten Fall folgt $K(\alpha) \cap K(\beta) = K(\sup(\alpha, \beta))$. □

Lemma A.6

Das Komplement eines Kegels läßt sich als Vereinigung endlich vieler, geeignet gewählter, paarweise disjunkter Kegel darstellen.

Beweis: Sei α ein endlicher Ablauf von π. Es sei $\overline{\alpha} = \{\beta \in \mathfrak{R}_{\text{fin}}(\pi) \mid \beta \not\parallel \alpha \text{ und } \exists e : \inf(\alpha, \beta) \stackrel{e}{\sqsubset} \beta\}$ die Menge der minimalen endlichen Abläufe, die zu α inkompatibel sind.

Illustration: Abb. A.1 zeigt ein Beispiel für einen endlichen Ablauf α zusammen mit seinen Konflikten in π. Die Konflikte sind dabei gestrichelt dargestellt. Für dieses Beispiel ist $\overline{\alpha} = \{\{b, f, g\}, \{a, d, f, g\}, \{a, c, e\}\}$. Dabei entsteht ein $\beta \in \overline{\alpha}$ indem genau

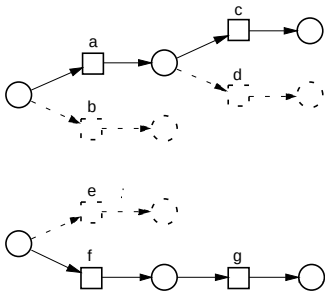


Abb. A.1: Ein endlicher Ablauf α und seine Konflikte.

ein Ereignis von α durch ein Konfliktereignis ersetzt wird. Beispielsweise entsteht $\{a, c, e\}$ aus α durch Ersetzung von f durch e. Dabei fällt auch g weg, da g von f kausal abhängt. Es gilt also für jedes $\beta \in \overline{\alpha}$: Es gibt genau ein Ereignis in β, das nicht zu α gehört; es gibt aber möglicherweise mehrere Ereignisse von α, die nicht zu β gehören. **Ende der Illustration.**

Es gilt

$$\overline{K(\alpha)} = \bigcup_{\beta \in \overline{\alpha}} K(\beta). \tag{A.5}$$

Desweiteren gilt: $\overline{\alpha}$ ist endlich, und die zugehörigen Kegel $K(\beta)$ für $\beta \in \overline{\alpha}$ sind paarweise disjunkt. Wir zeigen zunächst ($A.5$). Dabei bezeichnen wir für einen Ablauf ρ mit E_ρ die Menge der Ereignisse von ρ.

1a) $\subseteq$: Sei $\rho \in \overline{K(\alpha)}$. Dann gilt $\rho \not\parallel \alpha$. Dann existiert ein Ereignis $e \in E_\rho \setminus E_\alpha$ mit $\inf(\alpha, \rho) \overset{e}{\sqsubset} \beta \sqsubseteq \rho$. Dann ist β endlich, und es gilt $\alpha \not\parallel \beta$.

1b) $\supseteq$: Ist $\rho \in K(\beta)$ für $\beta \in \overline{\alpha}$, so gilt wegen $\alpha \not\parallel \beta$ genau $\rho \notin K(\alpha)$ (Prop. A.4).

2) Die Menge $\overline{\alpha}$ ist endlich, da π endlich verzweigt ist.

3) Sei $\beta_i \in \overline{\alpha}$ mit $\inf(\alpha, \beta_i) \overset{e_i}{\sqsubset} \beta_i$ für $i = 1, 2$. Wegen $\alpha \not\parallel \beta_i$ ist e_i zu jedem Ereignis von $\alpha \setminus \inf(\alpha, \beta_i)$ in Konflikt. Sei e das Ereignis von α zu dem e_1 in unmittelbarem Konflikt ist. Ist im Fall (a) $e \in \inf(\alpha, \beta_2)$, so ist $e \in \beta_2$ und damit $\beta_1 \not\parallel \beta_2$. Ist im Fall (b) $e \in \alpha \setminus \inf(\alpha, \beta_2)$, so ist $e \# e_2$, und da e in direktem Konflikt zu e_1 steht, gilt auch $e_1 \# e_2$, womit wiederum $\beta_1 \not\parallel \beta_2$ gilt. Aus $\beta_1 \not\parallel \beta_2$ folgt nun $K(\beta_1) \cap K(\beta_2) = \varnothing$. □

Folgerung A.7

Sowohl die Differenz als auch die Vereinigung zweier Kegel läßt sich als Vereinigung endlich vieler, geeignet gewählter, paarweise disjunkter Kegel darstellen.

Beweis: Die Aussage folgt aus den Lemmata A.5 und A.6 durch Anwendung De Morgan'scher Regeln:

$$K(\alpha) \setminus K(\beta) = K(\alpha) \cap \overline{K(\beta)} = K(\alpha) \cap \bigcup_{\gamma \in \overline{\beta}} K(\gamma) = \bigcup_{\gamma \in \overline{\beta}} (K(\alpha) \cap K(\gamma))$$

$$K(\alpha) \cup K(\beta) = (K(\alpha) \setminus K(\beta)) \cup (K(\beta) \setminus K(\alpha)) \cup (K(\alpha) \cap K(\beta)).$$ □

Wir können nun die gesuchte Mengenalgebra angeben.

Lemma A.8

Das Mengensystem

$$\mathcal{M} = \{ \bigcup_{A \in \mathcal{F}} A \mid \mathcal{F} \subseteq \mathcal{K} \text{ ist endlich und paarweise disjunkt} \}$$

ist eine Mengenalgebra über Ω.

Beweis: Es ist $\Omega \in \mathcal{M}$. Die Abgeschlossenheit unter Vereinigung und Komplement ergibt sich aus Folgerung A.7. □

Wir kommen nun zur Konstruktion des Prämaßes auf $\mathcal{M}$ und des Maßes auf $\sigma(\mathcal{M})$.

A.1.3 Konstruktion des Maßes

Wir beginnen jetzt mit der Konstruktion des Maßes P auf $\sigma(\mathcal{E})$. Dazu sei P zunächst auf $\mathcal{E}$ durch $(A.1)$ definiert, d.h. $P_0 : \mathcal{E} \to [0,1]$ sei definiert durch $P_0(K(\alpha)) = p(\alpha)$. Es gilt $P_0(\Omega) = 1$. Wir setzen P_0 auf $\mathcal{K}$ fort zu $P_1 : \mathcal{K} \to [0,1]$ durch $P_1(A) = P_0(A)$ für $A \in \mathcal{E}$ und $P_1(\varnothing) = 0$. Diese Definition ist gerechtfertigt, da $\varnothing \notin \mathcal{E}$.

Um P_1 auf $\mathcal{M}$ fortsetzen zu können, zeigen wir die Additivität von P_1. Offensichtlich genügt es dafür, die Additivität von P_0 zu zeigen. Wir benötigen dazu ein paar Vorbereitungen.

Definition A.9 (α-vollständiger probabilistischer Ablauf)

Sei α ein endlicher Ablauf von π. Ein endlicher Präfix κ von π heißt α-*vollständig*, falls

$$\bigcup_{\beta \in \mathfrak{R}_{max}(\kappa)} K(\beta) = K(\alpha). \tag{A.6}$$

○

Lemma A.10

Ist κ α-*vollständig*, so gilt:

$$p(\alpha) = \sum_{\beta \in \mathfrak{R}_{max}(\kappa)} p(\beta). \tag{A.7}$$

Beweis: Wir führen den Beweis durch Induktion über Präfixe von κ: (a) Sei zunächst $\kappa = \alpha$. Dann ist $(A.7)$ trivialerweise erfüllt. (b) Sei nun $\kappa \sqsupset \alpha$ und $(A.7)$ gelte für alle α-vollständigen echten Präfixe von κ. Sei κ' ein maximaler α-vollständiger echter Präfix von κ. Dann gibt es eine maximale Menge E' paarweise in direktem Konflikt stehender Ereignisse, so daß

$$\mathfrak{R}_{max}(\kappa) = \{\beta \mid \exists \beta' \in \mathfrak{R}_{max}(\kappa') : \exists e \in E' : \beta' \overset{e}{\sqsubset} \beta\}.$$

Dann gilt:

$$\sum_{\beta \in \mathfrak{R}_{max}(\kappa)} p(\beta) = \sum_{e \in E'} \mu(\tilde{e}) \cdot \sum_{\beta' \in \mathfrak{R}_{max}(\kappa')} p(\beta') = 1 \cdot p(\alpha) = p(\alpha). \qquad \square$$

Lemma A.11

Ist $\kappa \sqsubseteq \pi$ ein endlicher Präfix von π und $\alpha \sqsubseteq \kappa$ ein endlicher Ablauf von κ, so gibt es ein $\beta \in \mathfrak{R}_{max}(\kappa)$ mit $\alpha \sqsubseteq \beta$.

Beweis: Man entferne in κ alle Konflikte zu α, um β zu erhalten. $\square$

Wir zeigen nun die Additivität von P_0.

Lemma A.12

P_0 ist additiv, d.h. Sei A eine endliche Menge von paarweise inkompatiblen Ablaufstücken von π, so daß

$$\bigcup_{\alpha\in A} K(\alpha) = K(\gamma) \tag{A.8}$$

Dann gilt:

$$\sum_{\alpha\in A} p(\alpha) = p(\gamma) \tag{A.9}$$

Beweis: Wir beweisen zunächst einige Hilfsaussagen:

1. Die endliche Abwicklung $\sup A$ ist γ-vollständig. Für die Richtung $\subseteq$ von (A.6) sei $\rho \in K(\beta)$. Es gilt $\gamma \sqsubseteq \alpha \sqsubseteq \beta \sqsubseteq \rho$, also $\rho \in K(\gamma)$. Für die Richtung $\supseteq$ von (A.6) sei nun $\rho \in K(\gamma)$, dann gibt es wegen (A.8) ein $\alpha \in A$ mit $\rho \in K(\alpha)$. Wegen Lemma A.11 gibt es dann ein $\beta \in \mathfrak{R}_{max}(\sup A)$ mit $\rho \in K(\beta)$.

2. Zu jedem $\alpha \in A$ gibt es einen α-vollständigen Präfix κ_α mit $\kappa_\alpha \sqsubseteq \sup A$. (Wegen 1. und $\gamma \sqsubseteq \alpha$; lasse in $\sup A$ alle Konflikte zu α weg).

3. Es gilt: $\alpha_1, \alpha_2 \in A \Rightarrow \mathfrak{R}_{max}(\kappa_{\alpha_1}) \cap \mathfrak{R}_{max}(\kappa_{\alpha_2}) = \varnothing$. Beweis: $\mathfrak{R}_{max}(\kappa_{\alpha_1}) \cap \mathfrak{R}_{max}(\kappa_{\alpha_2}) \neq \varnothing$ impliziert $\bigcup_{\beta\in\mathfrak{R}_{max}(\kappa_{\alpha_1})} K(\beta) \cap \bigcup_{\beta\in\mathfrak{R}_{max}(\kappa_{\alpha_2})} K(\beta) \neq \varnothing$. Daraus folgt $K(\alpha_1) \cap K(\alpha_2) \neq \varnothing$ – ein Widerspruch zu $\alpha_1 \nparallel \alpha_2$.

4. Es gilt: $\bigcup_{\alpha\in A} \mathfrak{R}_{max}(\kappa_\alpha) = \mathfrak{R}_{max}(\sup A)$. Beweis: Es gilt $\kappa_\alpha \sqsubseteq \sup A$ und $\kappa_\alpha^\circ \subseteq (\sup A)^\circ$. Daraus folgt $\mathfrak{R}_{max}(\kappa_\alpha) \subseteq \mathfrak{R}_{max}(\sup A)$. Sei $\beta \in \mathfrak{R}_{max}(\sup A)$. Wegen $\gamma \sqsubseteq \beta$ gibt es ein $\alpha \in A$ mit $\alpha \sqsubseteq \beta$. Dann ist $\beta \sqsubseteq \kappa_\alpha$.

Wir zeigen nun (A.9).

$$\begin{aligned}
&\sum_{\alpha\in A} p(\alpha) \\
&= \quad \{2. \text{ und } (A.7)\} \\
&\sum_{\alpha\in A} \sum_{\beta\in\mathfrak{R}_{max}(\kappa_\alpha)} p(\beta) \\
&= \quad \{3.\} \\
&\sum_{\beta\in \bigcup_{\alpha\in A} \mathfrak{R}_{max}(\kappa_\alpha)} p(\beta) \\
&= \quad \{4.\}
\end{aligned}$$

$$\begin{aligned}
&\sum_{\beta \in \mathfrak{R}_{\max}(\sup A)} p(\beta) \\
&= \quad \{1. \text{ und } (A.7)\} \\
&p(\gamma)
\end{aligned}$$

□

Lemma A.13

Es gibt genau einen Inhalt P_2 auf $\mathcal{M}$ mit $P_2(K(\alpha)) = p(\alpha)$.

Beweis: Setzen wir $P_2(\bigcup_{A \in \mathcal{F}} A) = \sum_{A \in \mathcal{F}} P_1(A)$ für alle endlichen und paarweise disjunkten Mengenfamilien $\mathcal{F} \subseteq \mathcal{K}$, so ist P_2 wegen Lemma A.12 wohldefiniert. Desweiteren ist jeder Inhalt additiv und P_2 damit eindeutig. □

Wir können nun Satz 4.4 beweisen.

Beweis: von Satz 4.4 Aus der Additivität von P_2 folgt die σ-Additivität, da jeder Kegel wegen der endlichen Verzweigung von π nicht in unendlich viele paarweise disjunkte Kegel zerlegt werden kann. Damit ist P_2 ein Prämaß. Offensichtlich ist P_2 endlich und besitzt deshalb nach dem Fortsetzungssatz eine eindeutige Fortsetzung P auf $\sigma(\mathcal{M}) = \sigma(\mathcal{E}) = \mathcal{A}$. P erfüllt $(A.4)$ und ist damit ein Wahrscheinlichkeitsmaß.
□

Definitionsverzeichnis

1.1 Netz 9
1.2 Standardnotationen für Netze 9
1.3 Struktureigenschaften von Netzen 9
1.4 Markierung, Schalten 10
1.5 Initialisiertes Netz, Schaltsequenz 11
1.6 Kausalität, Konflikt, Nebenläufigkeit 15
1.7 Abwicklungsnetz 16
1.8 Abwicklung 17
1.9 Präfix 18
1.10 Infimum, Supremum 19
1.11 Ablauf 20
1.13 Markierungsschnitt 21
1.14 Kompatible Abläufe 22
1.15 Schaltsequenz eines Ablaufs 23
1.16 Ablaufeigenschaft 24
1.17 Sicherheitseigenschaft 24
1.18 Lebendigkeitseigenschaft 25
1.19 Zustandsformel 25
1.20 Temporalformel 25
1.21 Gültigkeit von Temporalformeln 26
1.22 Temporallogische Eigenschaft 26
1.23 Mengensignatur, *MSIG*-Algebra 31
1.24 Initialisiertes algebraisches Netz 32
1.25 Markierung eines algebraischen Netzes 32

1.26 Schalten eines algebraischen Netzes 33
1.28 Entfaltung 34
2.1 Netzsystem 41
2.2 Progressive Abwicklung 41
2.3 Schwache Fairneß 42
2.5 Lebendigkeit 44
2.7 Lebendigkeitsannahme 45
2.8 Mutex-Verhalten 47
2.9 Mutex-Struktur 48
2.11 Netzsystem für A, Nachrichtensystem 52
2.12 Konsens-Struktur 53
2.13 Initialisierung 53
2.14 Ausfall 54
2.15 Ausfalltolerantes Konsens-Verhalten 54
2.17 Bivalenter Ablauf 58
3.1 Faire Schaltsequenz 64
3.2 Fairer Ablauf 65
3.3 Faires Netzsystem 66
3.4 Zeitloses faires Netzsystem 68
4.1 Randomisiertes Netzsystem 80
4.2 Probabilistischer Ablauf 83
4.3 Wahrscheinlichkeit von endlichen Abläufen 84
4.5 Wahrscheinlichkeitsraum eines probabilistischen Ablaufs 85
4.6 Meßbare Ablaufeigenschaft 85
4.8 Probabilistische Gültigkeit 86
4.11 Extreme Fairneß 93
4.13 Probabilistisches Mutex-Verhalten 98
4.15 Konfliktarten für Transitionen 100
5.1 Faires randomisiertes Netzsystem 106
5.2 Fairer probabilistischer Ablauf 107
5.3 Ausfalltolerantes allgemeines Mutex-Verhalten 109

6.1 Konspiration 118
6.3 k-Fairneß, ∞-Fairneß (nach Best) 121
7.1 Beschränkte Konspiration 130
7.3 Abstand 132
7.4 Quasisynchronie 132
7.6 Fairneß unter Quasisynchronie 133
A.3 Kegel 148
A.9 α-vollständiger probabilistischer Ablauf 151

Abbildungsverzeichnis

1.1 Σ_1 – ein Netz. 11
1.2 Der maximale Schaltbaum von Σ_1 12
1.3 Ein nicht-sequentieller Ablauf ρ_1 von Σ_1 13
1.4 Vier Präfixe von ρ_1. 13
1.5 Zwei zueinander inkompatible Fortsetzungen von ρ_5. 14
1.6 Abwicklung π_1 von Σ_1 – Zusammenfassung von ρ_6 und ρ_7. 14
1.7 Die maximale Abwicklung π von Σ_1 15
1.8 Durch Definition 1.7 verbotene Strukturen. 16
1.9 Infimums- und Supremumsbildung auf Abwicklungen. 20
1.10 Ein unendlicher, fortsetzbarer Ablauf ρ_8 von Σ_1 21
1.11 Infimum und Supremum von kompatiblen endlichen Abläufen. . . . 22
1.12 Σ_2 – Ein einfacher Diffusionsalgorithmus. 28
1.13 Σ_3 – Entfaltung von Σ_2 34

2.1 Σ_4 : Zwei unabhängige zyklische Agenten. 40
2.2 Σ_5 : Ein Erzeuger/Verbraucher-System. 41
2.3 Σ_6 , Σ_7 – Problem bei schwacher Fairneß. 43
2.4 Σ_8 44
2.5 Σ_9 – ein Netzsystem mit wechselseitigem Ausschluß. 47
2.6 Mutex-Struktur 48
2.7 ρ_1 und Fortsetzung ρ_2 (gestrichelt) 49
2.8 Konsens-Struktur P_x 52
2.9 Ein Ring von drei Agenten. 55
2.10 Σ_{10} – Ein kleiner Konsensalgorithmus. 56

2.11 Eine unendliche Schaltsequenz von Σ_{10} 57
2.12 Beweisillustration zu Lemma 2.20. 60
2.13 Konstruktion eines nicht-entscheidenden progressiven Ablaufs. . . . 61

3.1 Σ_{11} 64
3.2 Eine Abwicklung von Σ_{11} mit Konfusion. 65
3.3 Σ_{12} , Σ_{13} – Zwei faire Netzsysteme. 66
3.4 Fairneß beim Einbiegen in eine Hauptstraße. 67
3.5 Σ_{14} – Ein zeitbehaftete Fairneßannahme. 67
3.6 Verfeinerung zu einem zeitlosen Netzsystem. 68
3.7 Σ_{15} – Ein faires Netzsystem mit Mutex-Struktur und Mutex-Verhalten. 70
3.8 Σ_{16} – Netzdarstellung eines FLP-Systems. 71
3.9 Fairneß im FLP-Modell. 72
3.10 Ein Konfliktcluster $\chi \in \Gamma$. 73
3.11 Σ_{17} – das FLP-System zu $\acute{\Sigma}$. 74

4.1 Modellierung eines Münzwurfs. 79
4.2 Verfeinerung eines Extended-Free-Choice-Konfliktes 80
4.3 Σ_{18} – Ein randomisiertes Netzsystem und sein probabilistischer Schaltbaum. 81
4.4 Σ_{19} 82
4.5 Ein endlicher, maximaler probabilistischer Ablauf von Σ_{19}. 84
4.6 Σ_{20} 87
4.7 Σ_{21} , Σ_{22} – Zwei randomisierte Netzsysteme. 88
4.8 Σ_{23} , Σ_{24} – Koordinierung nebenläufiger Entscheidungen. 89
4.9 Σ_{25} – Vergleich von sequentieller und nicht-sequentieller Semantik. . 91
4.10 Entfaltung einer flip-Kante. 95
4.11 Σ_{26} – Der Algorithmus von Ben-Or. 96
4.12 Implementierung freier Fairneß. 100
4.13 Ein einfacher Konflikt. 100
4.14 Verfeinerung eines Extended-Simple-Konfliktes. 101
4.15 Zwei Aspekte von Fairneß. 102

5.1 Beziehungen der Modelle. 105
5.2 Σ_{27} 106
5.3 Eine Nachbarschaftsrelation auf drei Agenten. 110

6.1 Die fünf Philosophen. 114
6.2 Eine replizierte Datenbank. 115
6.3 Σ_{28} – Ein Netzsystem mit konspirativem Ablauf. 116
6.4 Σ_{29} – Ein Netzsystem mit nicht-konspirativem Ablauf. 117
6.5 Synchronisationsfairneß 119
6.6 Σ_{30} – Wettlauf zwischen zwei Prozessen. 119
6.7 Eine Haupt- und eine Nebenstraße. 120
6.8 Σ_{31} , Σ_{32} – Zwei konspirationsfreie Netzsysteme. 122
6.9 Σ_{33} – Drei Agenten mit zwei Schlüsseln. 126
6.10 ρ_{33} 127
6.11 Konspiration im kleinen Konsensalgorithmus. 128
6.12 Eine Datenbank mit zwei langsamen Klienten. 128

7.1 Beschränkte Konspiration. 129
7.2 Σ_{34} , Σ_{35} – Unbeschränkte Konspiration. 130
7.3 $\Delta(C, e) = 4$ 132
7.4 Quasisynchronie. 133
7.5 Σ_{36} , Σ_{37} – Adaptiver Timeout. 134
7.6 Σ_{38} 135
7.7 Ablaufstruktur bei adaptivem Timeout. 135
7.8 Ein Ring von vier Agenten. 136
7.9 Σ_{39} – Zwei konkurrierende konspirationsgefährdete Transitionen. . . 137
7.10 Σ_{40} – Noch zwei konkurrierende konspirationsgefährdete Transitionen. 137
7.11 Verfeinerung von $t \in U$. 138
7.12 Ablauf ρ 140
7.13 Σ_{41} – Konspirationsbehafteter ausfalltoleranter allgemeiner Mutex. . 141
7.14 Beziehungen der Modellparameter. 143

A.1 Ein endlicher Ablauf α und seine Konflikte. 149

Literaturverzeichnis

[1] W. van der Aalst, E. Kindler und J. Desel: Beyond Asymmetric Choice: A Note on Some Extensions. *Petri Net Newsletter 55*:3–13. Okt. 1998.

[2] M. Abadi und L. Lamport: The Existence of Refinement Mappings. *Theoretical Computer Science 82*:253–284. Previously: SRC Research Report 27, April 1988. 1991.

[3] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour und C. Jard: Fault Detection and Diagnosis in Distributed Systems: An Approach By Partially Stochastic Petri Nets. *Discrete Event Dynamic Systems: Theory and Applications 8*:203–231. 1998.

[4] M. K. Aguilera und S. Toueg: Correctness Proof of Ben-Or's Randomized Consensus Algorithm. *Technical Report TR98-1682*, Cornell University, Computer Science. Mai 17, 1998.

[5] B. Alpern und F. B. Schneider: Defining Liveness. *Information Processing Letters 21*:181–185. Okt. 1985.

[6] R. Alur, H. Attiya und G. Taubenfeld: Time-adaptive Algorithms for Synchronization. *SIAM Journal of Computing 26*(2):539–556. Apr. 1997.

[7] R. Alur und T. A. Henzinger: Finitary Fairness. In: *Proc. 9th IEEE Symposium on Logic in Computer Science (LICS)*, (S. 52–61), IEEE. 1994.

[8] K. R. Apt, N. Francez und S. Katz: Appraising Fairness in Languages for Distributed Programming. *Distributed Computing 2*:226–241. 1988.

[9] P. C. Attie, N. Francez und O. Grumberg: Fairness and hyperfairness in multiparty interactions. *Distributed Computing 6*:245–254. 1993.

[10] H. Attiya und J. Welch: *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. McGraw-Hill. 1998.

[11] C. Baier und M. Kwiatkowska: On the verification of qualitative properties of probabilistic processes under fairness constraints. *IPL 66*:71–79. 1998.

[12] M. Barborak, M. Malek und A. Dahbura: The Consensus Problem in Fault-Tolerant Computing. *ACM Computing Surveys 25*(2):171–220. Juni 1993.

[13] H. Bauer: *Maß- und Integrationstheorie*. Zweite Aufl., de Gruyter. 1992.

[14] M. Ben-Or: Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. In: *Proc. PODC'83*, (S. 27–30), ACM. 1983.

[15] E. Best: Adequacy Properties of Path Programs. *Theoretical Computer Science 18*:149–171. 1982.

[16] E. Best: Why Three Philosophers Are Different From Five Philosophers. ISF BEGRUND-11. Sept. 1982.

[17] E. Best: Fairness and Conspiracies. *Information Processing Letters 18*:215–220. Erratum in *IPL 19*:162. 1984.

[18] E. Best: Structure Theory of Petri Nets: the Free Choice Hiatus. In: W. Brauer, W. Reisig und G. Rozenberg (Hg.), *Petri Nets: Central Models and Their Properties*, Bd. 254 von *LNCS*, (S. 167–205), Springer. 1987.

[19] E. Best und C. Fernández: *Nonsequential Processes*, Bd. 13 von *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag. 1988.

[20] L. Castellano, G. D. Michelis und L. Pomello: Concurrency vs Interleaving: an instructive example. *EATCS Bulletin 31*:12–15. Febr. 1987.

[21] T. D. Chandra, V. Hadzilacos und S. Toueg: The Weakest Failure Detector for Solving Consensus. *Journal of the ACM 43*(4):685–722. Juli 1996.

[22] T. D. Chandra, V. Hadzilacos, S. Toueg und B. Charron-Bost: On the Impossibility of Group Membership. *Techn. Ber. 2782*, INRIA Rocquencourt. Jan. 1996.

[23] T. D. Chandra und S. Toueg: Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM 43*(2):225–267. März 1996.

[24] K. M. Chandy und J. Misra: *Parallel Program Design: A Foundation*. Addison-Wesley. 1988.

[25] N. Dershowitz und D. Jayasimha: Bounded Fairness. *Techn. Ber. 615*, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL. Dez. 1986.

[26] J. Desel: How distributed algorithms play the token game. In: C. Freksa, M. Jantzen und R. Valk (Hg.), *Foundations of Computer Science — Potential, Theory, Cognition*, Bd. 1337 von *LNCS*, (S. 297–306), Springer-Verlag. 1997.

[27] J. Desel und J. Esparza: *Free Choice Petri Nets*. Cambridge University Press. 1995.

[28] J. Desel und E. Kindler: Proving Correctness of Distributed Algorithms Using High-Level Petri Nets – A Case Study. In: *Proc. CSD'98 Int. Conference on Application of Concurrency to System Design*, (S. 177–186), IEEE Press. 1998.

[29] E. W. Dijkstra: Solution of a problem in concurrent programming control. *Communications of the ACM 8*(9):569. 1965.

[30] E. W. Dijkstra: Hierarchical Ordering of Sequential Processes. *Acta Informatica 1*:115–138. 1971.

[31] D. Dolev, C. Dwork und L. Stockmeyer: On the Minimal Synchronism Needed for Distributed Consensus. *Journal of the ACM 34*(1):77–97. Jan. 1987.

[32] C. Dwork, N. Lynch und L. Stockmeyer: Consensus in the Presence of Partial Synchrony. *Journal of the ACM 35*(2):288–323. Apr. 1988.

[33] J. Engelfriet: Branching processes of Petri nets. *Acta Informatica 28*:575–591. 1991.

[34] W. Feller: *An Introduction to Probability Theory and its Applications*. Wiley. 1968.

[35] M. J. Fischer: The Consensus Problem in Unreliable Distributed Systems (A Brief Survey). In: *4th Conference on Foundations of Computation Theory (FCT)*, (S. 127–140). 1983.

[36] M. J. Fischer, N. A. Lynch und M. S. Paterson: Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM 32*(2):374–382. Apr. 1985.

[37] N. Francez: *Fairness*. Springer. 1986.

[38] R. van Glabbeek: *Comparative Concurrency Semantics and Refinement of Actions*. Dissertation, Vrije Universiteit te Amsterdam. Mai 1990.

[39] R. Gupta, S. A. Smolka und S. Bhaskar: On Randomization in Sequential and Distributed Algorithms. *ACM Computing Surveys 26*(1):7–86. März 1994.

[40] S. Hart, M. Sharir und A. Pnueli: Termination of Probabilistic Concurrent Programs. *ACM ToPLaS 5*(3):356–380. Juli 1983.

[41] M. Herlihy: Wait-Free Synchronization. *ACM ToPLaS 11*(1):124–149. Jan. 1991.

[42] A. Itai und M. Rodeh: Symmetry Breaking in Distributive Networks. In: *Proc. 22nd Annual Symposium on Foundations of Computer Science*, (S. 150–158), IEEE. 1981.

[43] M. Jaeger: Fairness, Computable Fairness, and Randomness. In: *Proc. 2nd PROBMIV Int. Workshop on Probabilistic Methods in Verification*, Technical Report CSR-99-8 School of Computer Science, University of Birmingham. 1999.

[44] K. Jensen: *Coloured Petri Nets*, Bd. 1 von *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag. 1992.

[45] R. E. Johnson und F. B. Schneider: Symmetry and Similarity in Distributed Systems. In: *Proc. 4th PODC*, ACM. 1985.

[46] Y.-J. Joung und J.-Y. Liao: Strong Interaction Fairness in a Fully Distributed System with Unbounded Speed Variability. In: *WDAG97*, Bd. 1320 von *LNCS*, (S. 230–244). 1997.

[47] E. Kindler: *Modularer Entwurf verteilter Systeme mit Petrinetzen*, Bd. 1 von *Edition Versal*. Bertz Verlag. Dissertation, Technische Universität München. Dez. 1995.

[48] E. Kindler und W. van der Aalst: Liveness, Fairness and Recurrence in Petri Nets. *Information Processing Letters 70*(6):269–274. Juni 1999.

[49] E. Kindler, W. Reisig, H. Völzer und R. Walter: Petri Net Based Verification of Distributed Algorithms: An Example. *Formal Aspects of Computing 9*:109–121. 1997.

[50] E. Kindler und H. Völzer: Flexibility in Algebraic Nets. In: *Proc. ICATPN'98: Intl. Conference on Application and Theory of Petri Nets*, Bd. 1420 von *LNCS*, (S. 345–364), Springer. Erscheint in *Theoretical Computer Science*. 1998.

[51] E. Kindler und R. Walter: Message Passing Mutex. In: J. Desel (Hg.), *Structures in Concurrency Theory*, Workshops in Computing, (S. 205–219), Berlin, Springer-Verlag. Mai 1995.

[52] E. Kindler und R. Walter: Mutex Needs Fairness. *Information Processing Letters 62*:31–39. 1997.

[53] M. Z. Kwiatkowska: Event Fairness and Non-Interleaving Concurrency. *Formal Aspects of Computing 1*:213–228. 1989.

[54] M. Z. Kwiatkowska: Survey of fairness notions. *Information and Software Technology 31*(7):371–386. 1989.

[55] L. Lamport: `http://www.research.digital.com/src/personal/lamport/pubs/pubs.html`.

[56] L. Lamport: Buridan's Principle. Unveröffentlicht, verfügbar über [55]. Dez. 1984.

[57] L. Lamport: Fairness and Hyperfairness. *SRC research report 152*, Digital Equipment Corporation. To appear in *Distributed Computing*. Apr. 1998.

[58] D. Lehmann, A. Pnueli und J.Stavi: Impartiality, Justice, and Fairness: The Ethics of Concurrent Termination. In: *Proc. 8th ICALP (Int.. Colloquium on Automata, Languages, and Programming)*, Bd. 115 von *LNCS*, (S. 264–277), Springer. 1981.

[59] D. Lehmann und M. Rabin: On the advantage of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In: *Proc. 8th POPL (Symposium on Principles of Programming Languages)*, (S. 133–138), ACM. Jan. 1981.

[60] O. Lichtenstein, A. Pnueli und L. Zuck: The Glory of the Past. In: *Proc. Workshop on Logics of Programs*, Bd. 193 von *LNCS*. 1985.

[61] N. Lynch und M. Tuttle: An introduction to input/output automata. *CWI-Quarterly 3*(2):219–246. 1989.

[62] N. A. Lynch: *Distributed Algorithms*. Morgan Kaufmann. 1996.

[63] Z. Manna und A. Pnueli: *The Temporal Logic of Reactive and Concurrent Systems – Specification*. Springer. 1992.

[64] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli und G. Franceschinis: *Modelling with Generalized Stochastic Petri Nets*. Series in Parallel Computing, Wiley. 1995.

[65] A. Merceron: Fair Processes. In: *Advances in Petri Nets*, Bd. 266 von *LNCS*, Springer. 1987.

[66] M. Merritt und G. Taubenfeld: Fairness of Shared Objects. In: S. Kutten (Hg.), *DISC'98*, Nr. 1499 in LNCS, (S. 303–317), Springer. Sept. 1998.

[67] G. Neiger: Failure Detectors and the Wait-Free Hierarchy. In: *Proc. 14th PODC*, (S. 100–109), ACM. Aug. 1995.

[68] M. Nielsen, G. Plotkin und G. Winskel: Petri Nets, Event Structures and Domains, Part I. *Theoretical Computer Science 13*:85–108. 1981.

[69] A. Pnueli: On The Extremely Fair Treatment of Probabilistic Algorithms. In: *Proc. 15th Annual Symposium on Theory of Computing (STOC)*, (S. 278–290), ACM. 1983.

[70] A. Pnueli und L. Zuck: Verification of multiprocess probabilistic protocols. *Distributed Computing 1*:53–72. 1986.

[71] A. Pnueli und L. D. Zuck: Probabilistic Verification. *Information and Computation 103*:1–29. 1993.

[72] J. Queille und J. Sifakis: Fairness and Related Properties in Transition Systems – A Temporal Logic to Deal with Fairness. *Acta Informatica 19*:195–220. 1983.

[73] M. O. Rabin: The Choice Coordination Problem. *Acta Informatica 17*:121–134. 1982.

[74] M. O. Rabin: N-Process Mutual Exclusion with Bounded Waiting by log N-shared variables. *Journal of Computer and System Sciences 25*:66–75. 1982.

[75] J. Rao: Reasoning about Probabilistic Algorithms. In: *Proc. PODC 90*, (S. 247–264), ACM. 1990.

[76] R. Reischuk: Zeit und Raum in Rechnernetzen. In: I. Wegener (Hg.), *Highlights aus der Informatik*, (S. 155–176), Springer. 1996.

[77] W. Reisig: *Petrinetze: Eine Einführung*. Studienreihe Informatik, Springer-Verlag. 1982.

[78] W. Reisig: Das Verhalten Verteilter Systeme. *GMD-Bericht 170*, GMD. R. Oldenbourg Verlag. 1987.

[79] W. Reisig: A Strong Part of Concurrency. In: *Advances in Petri Nets*, Bd. 266 von *LNCS*, (S. 238 – 272), Springer Verlag. 1987.

[80] W. Reisig: Petri Nets and Algebraic Specifications. *Theoretical Computer Science 80*:1–34. Mai 1991.

[81] W. Reisig: *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer. 1998.

[82] W. Reisig, E. Kindler, T. Vesper, H. Völzer und R. Walter: Distributed Algorithms for Networks of Agents. In: *Lectures on Petri Nets II: Applications*, Bd. 1492 von *LNCS*, (S. 331–385), Springer. 1998.

[83] W. Reisig und G. Rozenberg (Hg.): *Lectures on Petri Nets I: Basic Models*, Bd. 1491 von *LNCS*. Springer. 1998.

[84] G. Rozenberg und J. Engelfriet: Elementary Net Systems. In: W. Reisig und G. Rozenberg (Hg.), *Lectures on Petri Nets I: Basic Models*, Bd. 1491 von *LNCS*, (S. 12–121), Springer. 1998.

[85] F. B. Schneider: What good are models and what models are good? In: S. Mullender (Hg.), *Distributed Systems*, zweite Aufl., Kap. 2, (S. 17–26), Addison-Wesley. 1993.

[86] R. Segala: *Modeling and Verification of Randomized Distributed Real-Time Systems*. Technical report mit/lcs/tr-676, MIT, Laboratory for Computer Science. Juni 1995.

[87] E. Smith: On the border of causality: contact and confusion. *Theoretical Computer Science 153*:245–270. 1996.

[88] G. Tel: *Introduction to Distributed Algorithms*. Cambridge University Press. 1994.

[89] J. Turek und D. Shasha: The Many Faces of Consensus in Distributed Systems. *Computer 18*(6):8–17. Juni 1992.

[90] H. Völzer: Verifying fault tolerance of distributed algorithms formally: An example. In: *Proc. CSD'98: Intl. Conference on Application of Concurrency to System Design, Fukushima, Japan*, (S. 187–197), IEEE Computer Society Press. März 1998.

[91] M. Weber, R. Walter, H. Völzer, T. Vesper, W. Reisig, S. Peuker, E. Kindler, J. Freiheit und J. Desel: DAWN: Petrinetzmodelle zur Verifikation Verteilter Algorithmen. *Informatik-Bericht 88*, Humboldt-Universität zu Berlin. Dez. 1997.

Index

I^0, I^1, 53
K-synchron, 132
$\mathfrak{M}(A)$, 8
$\mathfrak{R}(\pi)$, $\mathfrak{R}_{max}(\pi)$, $\mathfrak{R}_{fin}(\pi)$, $\mathfrak{R}(\Sigma)$, 20
$\mathfrak{S}(A)$, 7
$\mathfrak{W}(A)$, 7
α-Fairnes, 93
co, 15, 20
#, 15, 20
$\Diamond$,$\Box$,$\triangleright$, 25
$\tilde{\cdot}$, 21
inf, 19
∞-Fairnes, 121
$\lfloor r \rfloor$, 7
π°, 21
$^\circ N$, N°, 9
$\models$, 25, 26
$\nparallel$, $\|$, 22
$\Vdash$, 86
$\sqsubset$, 19
$\sqsubseteq$, 18
$\downarrow x$, 9
σ-Algebra, 36
$\sigma(\mathcal{E})$, 36
sup, 19
Σ^I, 53
k-Fairnes, 121
k-Konspiration, 130
k-Synchronisationsfairnes, 141
k-erreichbarer Markierungsschnitt, 130

Ablauf, 20
Ablauf uber P, 24
Ablaufeigenschaft uber P, 24
Abstand, 132
Abwicklung, 17
Abwicklungsnetz, 16
adaptiver Timeout, 134
Aktivierungbedingung, 32
algebraisches Netz, 27, 32
allgemeiner Gegenspieler, 90
Anfangsschnitt, 21
aquivalenzrobust, 45
Arbiter, 67
Ausfall, 54
ausfalltolerantes allgemeines Mutex-Verhalten, 109
ausfalltolerantes Konsens-Verhalten, 54
Ausgabestelle, 68
Auswertung, 31

Bedingung, 13
beschrankt randomisiert, 80
beschrankte Konspiration, 130
bivalent, 58

charakteristische Funktion, 7
co-Menge, 16
computable fairness, 94

einfache Fairnes, 101
einfache Transition, 100
Eingabestelle, 68
Ende, 21
endlich T-verzweigt, 9
Entfaltung, 34
Ereignis, 13
ereignisloser Ablauf, 20
erfullt, 24
erreichbare Markierung, 10
erreichbare Markierung von Σ, 11

erreichbarer Markierungsschnitt, 21
Erzeuger, 36
erzeugte σ-Algebra, 36
Extended-Free-Choice-Konflikt, 80
Extended-Free-Choice-Netz, 101
Extended-Simple-Netz, 101
externe Transition, 41
extreme Fairnes, 93

faire Schaltsequenz, 64
fairer Ablauf, 65
fairer probabilistischer Ablauf, 107
faires Netzsystem, 66
faires randomisiertes Netzsystem, 106
Fairnes, 63
Fairnes unter Quasisynchronie, 133
Fairnestransition, 106
flip-Kante, 95
FLP-Modell, 50, 71
FLP-System, 71
Fortsetzung, 18
Free-Choice-Konflikt, 79
Free-Choice-Netz, 101
freie Fairnes, 101
freie Transition, 100

Gegenspieler, 90
Gultigkeit, 25

Hyperfairnes, 124

Infimum, 19
initialisiertes algebraisches Netz, 32
initialisiertes Netz, 11
Initialisierung, 53
inkompatibel, 22
interne Transition, 41

Kausalordnung, 15
Kegel, 82
kompatibel, 22
Konflikt, 11, 15
Konfliktarten, 100
Konfliktcluster, 72
Konfusion, 65
Konsens-Struktur, 53
Konspiration, 118
konspirationsbehaftet, 118
konspirationsfrei, 118

lebendig, 44
Lebendigkeitsannahme, 39, 45
Lebendigkeitseigenschaft, 25
li-Menge, 16

Markierung, 10
Markierung uber P, 24
Markierungsschnitt, 21
maximale Schaltsequenz, 39
Mengensignatur, 31
mesbare Ablaufeigenschaft, 85
mesbare Menge, 36
Multiparty-Interaktion, 115
Mutex-Struktur, 48
Mutex-Verhalten, 47

Nachbarn, 107
Nachbarschaftsrelation, 107
Nachbereich, 9
Nachrichtensystem, 52
nebenlaufig, 15
Netz, 9
Netzsystem, 41
Netzsystem fur A, 52
Netzwerkalgorithmus, 27

objektiver Konflikt, 65
Operation, 30
Originaltransition, 68

persistente Ressource, 101
Philosoph, 114
Prafix, 18
Prafixinjektionen, 19
probabilistisch gultig, 86
probabilistische Transition, 80

probabilistischer Ablauf, 83
probabilistischer Schaltbaum, 82
probabilistisches Ereignis, 83
probabilistisches Mutex-Verhalten, 98
probabilistisches Netzsystem, 81
Progres, 40
progressive Abwicklung, 41

quasi-einfache Transition, 100
quasi-freie Transition, 100
quasisynchron, 132
Quasisynchronie, 131

randomisierter Algorithmus, 77
randomisiertes Netzsystem, 80
rekurrente Ressource, 101

Schaltbaum, 11
Schaltsequenz, 11
Schaltsequenz eines Ablaufs, 23
Schaltsequenz uber P, 24
Schaltsequenzeigenschaft uber P, 24
Schleife, 43
schwach konfuse Transition, 100
schwache Fairnes, 42
Sequentialisierung, 23
sequentielles Netzsystem, 81
sichere Markierung, 10
sicheres Netz, 11
Sicherheitseigenschaft, 24
Signatur, 30
Simple-Netz, 101
Sorte, 30
stark konfuse Transition, 100
Stelle, 9
stellenberandet, 9
stochastisch unabhangig, 36
Supremum, 19
synchrones System, 131
Synchronieannahme, 131
Synchronisationsfairnes, 119
Systemnetz, 9

temporallogische Eigenschaft, 26
Term, 30
Tragermenge, 30
Transition, 9
Transporttransition, 68

unabhangig, 15
unbeschrankte Konspiration, 130
univalent, 58
unmittelbarer Konflikt, 15

valent, 58
Vergangenheitsformel, 93
verletzt, 24
Verteilung, 80
Vorbereich, 9
vorgangerabgeschlossen, 18
vorgangerendlich, 9

Wahrscheinlichkeitsmas, 36
Wahrscheinlichkeitsraum, 36
Wahrscheinlichkeitsraum von π, 85
wechselseitiger Ausschlus, 46

zeitlich geordnet, 23
zeitloses faires Netzsystem, 68
Zustandsfairnes, 92
Zustandsformel, 25

Erklärung

Ich erkläre hiermit, daß

- ich die vorliegende Dissertationsschrift „Fairneß, Randomisierung und Konspiration in verteilten Algorithmen“ selbständig und ohne unerlaubte Hilfe angefertigt habe;
- ich mich nicht bereits anderwärts um einen Doktorgrad beworben habe oder einen solchen besitze;
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin bekannt ist.

Lebenslauf

Adresse	Chodowieckistr. 26, 10405 Berlin
Geburt	am 20. April 1971 in Schwerin
Staatsangehörigkeit	Deutsch

Ausbildung

1989	Abitur am Gymnasium „J.W. Goethe“ Schwerin
1992	Vordiplom im Diplomstudiengang Informatik, Humboldt-Universität zu Berlin
1995	Diplom-Informatiker, Humboldt-Universität zu Berlin

Berufserfahrung

April 92 – Sept. 92	Tutor am Institut für Mathematik der Humboldt-Universität zu Berlin
Dez. 92 – April 94	Wissenschaftliche Hilfskraft am Fraunhofer Institut für Software- und Systemtechnik ISST in Berlin
Mai 94 – März 95	Wissenschaftliche Hilfskraft bei Prof. Dr. Lothar Budach, Lehrstuhl für Mathematische Grundlagen der Informatik, Universität Potsdam
Aug. 95 – Nov. 95	Tutor bei Prof. Dr. Wolfgang Reisig, Institut für Informatik der Humboldt-Universität zu Berlin im DFG-Sonderforschungsbereich 342 „Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen“
seit Dez. 95	Wissenschaftlicher Mitarbeiter am Lehrstuhl für Theorie der Programmierung des Instituts für Informatik der Humboldt-Universität zu Berlin im DFG-Forschungsprojekt „Konsensalgorithmen“