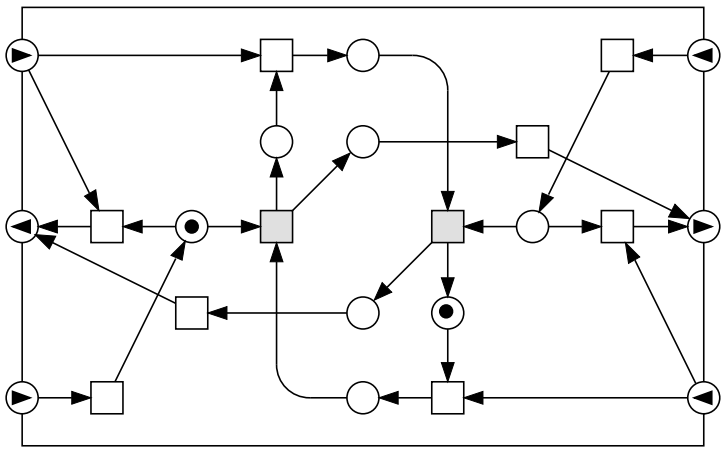
Fakultät für Informatik
der Technischen Universität München

# Modularer Entwurf verteilter Systeme mit Petrinetzen

*Ekkart Kindler*



Ekkart Kindler

# Modularer Entwurf verteilter Systeme mit Petrinetzen

EDITION VERSAL 1

EDITION VERSAL
Herausgeber: Wolfgang Reisig

| | | |
|---|---|---|
| **Band 1:** | E. Kindler: | Modularer Entwurf verteilter Systeme mit Petrinetzen |
| **Band 2:** | R. Walter: | Petrinetzmodelle verteilter Algorithmen. Beweistechnik und Intuition |

*In Vorbereitung:*

| | | |
|---|---|---|
| **Band 3:** | D. Gomm: | Modellierung und Analyse verzögerungs-unabhängiger Schaltungen mit Petrinetzen |

Ekkart Kindler

# Modularer Entwurf verteilter Systeme mit Petrinetzen

Dieter Bertz Verlag

Die Deutsche Bibliothek – CIP Einheitsaufnahme

**Kindler, Ekkart:**
Modularer Entwurf verteilter Systeme mit Petrinetzen /
Ekkart Kindler. – Berlin : Bertz, 1995
(Edition Versal ; Bd. 1)
Zugl.: München, Techn. Univ., Diss., 1995
ISBN 3-929470-51-9
NE: GT


Görlitzer Str. 37, 10997 Berlin
Druck: Offset-Druckerei Weinert, Berlin
Printed in Germany
ISBN 3-929470-51-9

# Abstract

This thesis introduces a formalism for the design of distributed systems. The formalism is based on Petri nets an their partial order runs (processes), which we call distributed runs. The emphasis of this thesis is on the following two aspects:

1. In contrast to classical approaches we use a non-sequential model for runs. We investigate the common features of sequential and distributed runs and the difference. To this end, we show that the distributed runs together with a prefix relation form a Scott domain.

   It turns out that many classical concepts, such as safety and liveness properties carry over to distributed runs. But, we will show that there is one essential difference which lies in the very nature of distributedness.

2. We introduce a module concept for Petri nets. This concept allows systems to be built from components; the behaviour of a system should be completely determined by the behaviour of its components. The use of distributed runs provides a simple model for such compositional behaviour.

   Based on the compositional behaviour we introduce RG-Graphs for specifying components in a rely-guarantee-style; i.e. we specify the behaviour of a component, which depends on the correct behaviour of its environment. When the environment does not behave correctly, the component needn't behave correct either. Then, a specification of a system can be split into specifications of its components, which can be implemented separately.

## Zu diesem Band

Ein verteiltes System wird nicht „in einem Stück“ entwickelt. Vielmehr liegen oft einige Komponenten vor, weitere werden neu entwickelt und eingefügt. Das Verhalten eines gesamten Systems ergibt sich so aus dem Verhalten seiner Komponenten.
Hier setzt die vorliegende Arbeit ein mit der Frage, wie man die wichtigen Eigenschaften einzelner Komponenten beschreibt und wie sie in die Beschreibung des Gesamtverhaltens eines Systems eingehen. *Verteiltheit* und *Modularität* sind dabei die zentralen Begriffe.
Petrinetze und Temporale Logik bilden das Handwerkszeug der Arbeit. Ausgehend von einem kompositionalen Verhaltensbegriff wird ein Modulkonzept für verteilte Abläufe eingeführt. Anhand eines Beispiels wir demonstriert, wie sich die Spezifikation eines Gesamtsystems in Spezifikationen für einzelne Komponenten zerlegen läßt. Diese Komponenten können dann separat implementiert werden.

Berlin, im November 1995

W. Reisig,
Herausgeber

# Vorwort

In diesem Text stellen wir einen Formalismus zum Entwurf verteilter Systeme vor. Der zugrundeliegende Systembegriff baut auf Petrinetzen auf. Das Verhalten von Systemen beschreiben wir durch halbgeordnete Abläufe (Prozesse) von Petrinetzen. Wir nennen sie zur Unterscheidung von sequentiellen Abläufen verteilte Abläufe. Dieser Text hat zwei Schwerpunkte:

1. Im Gegensatz zu den klassischen Entwurfsmethoden benutzen wir ein nicht-sequentielles Ablaufmodell; wir arbeiten die Gemeinsamkeiten und Unterschiede unseres Ablaufmodells mit dem sequentiellen Ablaufmodell heraus.

2. Wir führen ein Modulkonzept für Petrinetze ein, das es erlaubt Systeme aus Systemkomponenten zusammenzusetzen. Dabei wollen wir das Verhalten des Gesamtsystems aus dem Verhalten seiner Komponenten gewinnen. Die verteilten Abläufe erweisen sich als eine gute Basis für die Definition eines derartigen kompositionalen Verhaltens.

In der Arbeit untersuchen wir zunächst die mathematische Struktur, die den verteilten Abläufen zugrundeliegt. Zusammen mit der Präfixrelation bildet die Menge der verteilten Abläufe einen Scott-Bereich. Mit diesem Ergebnis lassen sich die Konzepte der Sicherheits- und Lebendigkeitseigenschaften kanonisch von sequentiellen Abläufen auf verteilte Abläufe übertragen; viele Zusammenhänge gelten für verteilte Abläufe analog zum sequentiellen Fall.
Es gibt jedoch einen wesentlichen Unterschied zum sequentiellen Fall, der inhärent mit der Verteiltheit einhergeht. Dieser Unterschied und seine Bedeutung für den Systementwurf wird näher untersucht.

Im Hinblick auf den modularen Entwurf erweitern wir Petrinetze zu Systemkomponenten: Eine Systemkomponente besitzt eine Menge von Ein- und Ausgabestellen, über die sie mit ihrer Umgebung kommuniziert. Über diese Schnittstellen können Systemkomponenten zusammengesetzt werden. Das Verhalten eines zusammengesetzten Systems ergibt sich aus dem Verhalten der einzelnen Systemkomponenten.
Aufbauend auf diesem kompositionalen Verhalten können wir Systemkomponenten modular spezifizieren. Wir beschreiben dabei das Verhalten einer Systemkomponente in Abhängigkeit von dem Verhalten ihrer Umgebung (im sog. Rely-Guarantee-Stil). Im Gegensatz zu vie-

len anderen Methoden kann das Verhalten in Abhängigkeit von Lebendigkeitseigenschaften der Umgebung beschrieben werden. Wir nennen diese Beschreibung des Verhaltens von Systemkomponenten Modul. Ein Modul besitzt im allgemeinen verschiedene Implementierungen. Ein Modul kann in Teilmodule zerlegt werden, die dann unabhängig voneinander implementiert werden können. Zum „Rechnen“ mit Modulen und zum Beweis von temporallogisch formulierten Eigenschaften geben wir geeignete Regeln an.

Die Anwendung des Formalismus demonstrieren wir an einem verteilten Algorithmus zur Gewährleistung des wechselseitigen Ausschlusses. Wir zeigen anhand dieses Beispiels, daß man ein Modul in Teilmodule zerlegen kann, die dann separat implementierbar sind.

Der vorliegende Text ist eine überarbeitete Version meiner Dissertation, die ich im April 1995 an der Technischen Universität München eingereicht habe. Für die Betreuung dieser Arbeit bedanke ich mich bei Prof. W. Reisig, der viele Vorschläge zur Präsentation dieser Arbeit machte. Prof. M. Broy danke ich für die Kommentare zu Vorversionen dieser Arbeit. Beiden danke ich für die Übernahme der Gutachten.
Die Ideen zu dieser Arbeit sind im Rahmen meiner Mitarbeit im SFB-342 an der TU München entstanden. Sie sind unter anderem das Ergebnis von vielen intensiven Gesprächen im Teilprojekt A3 „SEMAFOR“. Insbesondere haben die Mitglieder der „uNETy“-Gruppe zur Weiterentwicklung dieser Ideen beigetragen. Ihnen allen gilt mein Dank.
Die Arbeit selbst ist hauptsächlich an der Humboldt-Universität zu Berlin entstanden. Hier bedanke ich mich bei Dr. Jörg Desel, dessen kritische Anmerkungen zur Verbesserung dieser Arbeit beigetragen haben und bei Rolf Walter, der stets bereitwillig meine Entwürfe gelesen und hilfreich kommentiert hat. Dank gebührt auch meinem Münchner Kollegen Dominik Gomm, der neben inhaltlichen Diskussionen auch durch organisatorische Tätigkeiten in München zum erfolgreichen Abschluß meiner Dissertation beigetragen hat.

Berlin, im November 1995 Ekkart Kindler

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

Verteilte Systeme bestimmen heute unser tägliches Leben. Angefangen beim Telefonsystem, über Geldautomaten und andere Möglichkeiten des bargeldlosen Zahlungsverkehrs, bis hin zum vernetzten Arbeitsplatz, verbirgt sich hinter jedem dieser Systeme eine Menge von *Systemkomponenten*, die lose miteinander gekoppelt sind, um uns die gewünschten Dienste zu erbringen. Diese Systemkomponenten werden heute fast ausschließlich von digitalen Rechnern realisiert[1]. Die verschiedenen Systemkomponenten sind meist physikalisch *verteilt*.
Eine Aufgabe der Informatik war und ist die Entwicklung von Methoden, um Anforderungen an verteilte Systeme zu *beschreiben*, aus dieser Beschreibung systematisch ein System zu *entwerfen* und dessen Korrektheit zu *beweisen*. Häufig wird versucht, solche Methoden auf die bereits „klassischen“ Methoden für sequentielle Algorithmen zurückzuführen. Bisher konnte sich jedoch keine dieser Methoden entscheidend durchsetzen. Bestimmte Phänomene verteilter Systemen werden von sequentiellen Ablaufmodellen nicht erfaßt. Ein solches Phänomen werden wir in Abschnitt 1.1 an einem Beispiel demonstrieren. In dieser Arbeit benutzen wir deshalb ein Modell für verteilte Systeme und ihre Abläufe, das Verteiltheit explizit berücksichtigt.
Verteilte Systeme kann man meist nicht „in einem Stück“ entwickeln. Häufig baut man ein System aus bereits vorhandenen Systemkomponenten auf; nur einige Systemkomponenten werden neu entwickelt. Über die Zeit kommen dann neue Systemkomponenten hinzu und bestimmte Systemkomponenten werden durch andere ersetzt. Ein ein-

[1] Besonders deutlich ist dies beim digitalen Telefonnetz, das einen eigentlich „analogen“ Dienst mit Hilfe digitaler Rechner erbringt.

drucksvolles Beispiel dafür ist das Telefonsystem. Es ist deshalb notwendig, Systemkomponenten auch einzeln zu beschreiben und zu entwickeln. Das Verhalten des Gesamtsystems sollte sich deshalb aus dem Verhalten der einzelnen Systemkomponenten ergeben. Wenn eine Verhaltensbeschreibung dies ermöglicht, nennt man sie *kompositional*.
Eine einzelne Systemkomponenten ist meist nur in Interaktion mit ihrer Umgebung vorstellbar. Derartige Systeme interessieren uns in dieser Arbeit besonders. Sie werden *reaktive Systeme* [36] genannt. Ein reaktives System kann einen bestimmten Dienst häufig nur dann erbringen, wenn seine Umgebung bestimmte Anforderungen erfüllt.
Wir sind an einem Formalismus zur Beschreibung von Systemen interessiert, der das Zusammensetzen von Systemen aus Systemkomponenten unterstützt. Darüber hinaus wollen wir das Verhalten einer Komponente in Abhängigkeit vom Verhalten der Umgebung beschreiben. Eine solche Beschreibung von Systemkomponenten, die unabhängig von einer konkreten Implementierung ist, nennen wir *Modul*. Ein Modul kann im allgemeinen durch verschiedene Systemkomponenten implementiert werden. In Abschnitt 1.2 werden wir unser Modulkonzept mit Hilfe eines Beispiels informell vorstellen.

In den Abschnitten 1.1 und 1.2 werden wir nun die wesentliche Grundbegriffe dieser Arbeit anhand einfacher Beispiele vorstellen. Im Anschluß daran geben wir in Abschnitt 1.3 einen Überblick über diese Arbeit.

## 1.1 Verteilte Systeme

Wir stellen unseren Systembegriff anhand eines kleinen Beispiels vor. An diesem Beispiel werden wir ein Phänomen verdeutlichen, welches nur mit verteilten Abläufen adäquat modelliert wird. Als Systemmodell benutzen wir Stellen/Transitions-Systeme (S/T-Systeme) [80], die wir in dieser Arbeit *verteilte Transitionssysteme* nennen (vgl. [76]).

### Verteilte Transitionssysteme

Abbildung 1.1 zeigt ein verteiltes Transitionssystem. Es modelliert zwei Philosophen („1“ und „2“ genannt), die nach der Initialisierung unabhängig voneinander die Zustände „denkend“, „hungrig“ und „essend“ (abgekürzt durch $d$, $h$ und $e$) durchlaufen. Ein verteiltes Transitionssystem besteht aus aktiven und passiven Elementen, die durch

Pfeile miteinander in Beziehung gesetzt werden. Die passiven Elemente repräsentieren die lokalen Zustände des verteilten Systems. Sie werden *Stellen* genannt und graphisch durch Kreise dargestellt. Die aktiven Elemente beschreiben die Zustandsübergänge des Systems. Sie werden *Transitionen* genannt und graphisch durch Quadrate dargestellt. Die Transition $t_0$ des Systems aus Abb. 1.1 beschreibt beispielsweise den Übergang von dem Zustand, in dem $a$ erfüllt ist, in den Zustand, in dem $d_1$ und $d_2$ erfüllt sind. Lokale Zustände, die beim Start eines verteilten Transitionssystems vorliegen, werden durch einen schwarzen Punkt (der sog. *Marke*) in den entsprechenden Stellen *markiert*. Im Beispiel ist das nur die Stelle $a$.

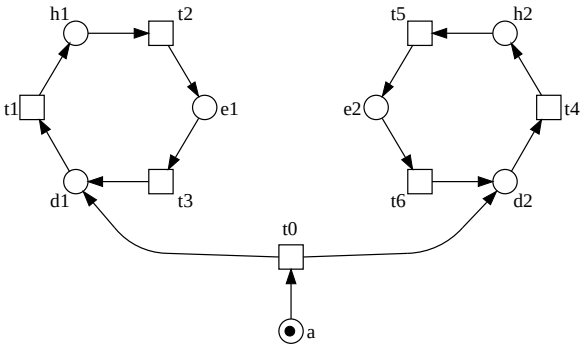



Abbildung 1.1: Zwei Philosophen

Das *Verhalten* des verteilten Transitionssystems läßt sich durch das wiederholte Verändern der lokalen Zustände durch die Transitionen des Systems beschreiben. Graphisch stellen wir das als „Abwicklung“ des Systems dar, wie Abb. 1.2 zeigt. Diese Abwicklung nennen wir *verteilten Ablauf* des Transitionssystems aus Abb. 1.1. Links in diesem Ablauf sehen wir die einzige Stelle, die zu Beginn markiert ist. Ausgehend davon werden mögliche Zustandsänderungen protokolliert, dabei wird der Zusammenhang zu den Stellen und Transitionen des verteilten Transitionssystems durch eine entsprechende *Beschriftung* hergestellt. Zur graphischen Unterscheidung von Systemen und Abläufen setzen wir die Beschriftung von Abläufen in die entsprechenden Stellen bzw. Transitionen. Dabei können in einem Ablauf durchaus mehrere Stellen gleich beschriftet sein. Zum Beispiel tritt $d_1$ („Philosoph 1 denkt“) nach einem vollen Durchlauf des Zyklus ($h_1$ und $e_1$) erneut ein.

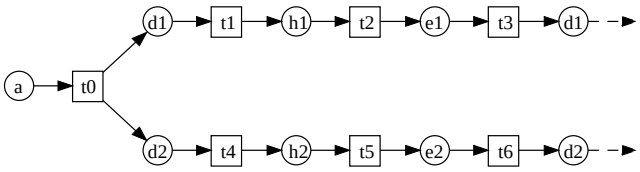



Abbildung 1.2: Ablauf des Philosophensystems

In diesem Ablauf sind die Zustandsübergänge der beiden Philosophen unabhängig voneinander. Zwischen dem oberen und unteren „Strang“ des Ablaufs besteht keine Beziehung. Wir können also keine Aussage über die Reihenfolge des Auftretens der Transitionen $t1$ und $t4$ machen. Deshalb nennen wir zwei Stellen oder zwei Transitionen eines Ablaufs *nebenläufig*, wenn sie nicht durch die Pfeile des Ablaufs geordnet sind. Beispielsweise sind in obigem Ablauf jeweils zwei mit $e_1$ bzw. $e_2$ beschriftete Stellen nebenläufig. Dagegen sind die beiden (verschiedenen) Stellen, die mit $d_1$ beschriftet sind, nicht nebenläufig, da es eine Folge von Pfeilen zwischen diesen Stellen gibt, die eine Reihenfolge zwischen diesen Stellen festlegt.

Zwei Komponenten, die völlig unabhängig voneinander sind — wie im Beispiel die beiden Philosophen — sind im allgemeinen nicht von Interesse. Vielmehr interessieren uns Komponenten, die miteinander interagieren. Dies ist beispielsweise notwendig, um zu gewährleisten, daß nie beide Philosophen gleichzeitig essen; d.h. $e_1$ und $e_2$ kommen in keinem Ablauf nebenläufig vor. Dies entspricht dem Paradigma des „wechselseitigen Ausschlußes“, das in abstrakter Form das Verwalten knapper Ressourcen oder das Verhindern unerwünschter Systemzustände beschreibt.

### Eigenschaften

Der wechselseitige Ausschluß ist ein Beispiel für eine *Eigenschaft*. Allgemein nennen wir jede Menge von Abläufen Eigenschaft. Ein Ablauf *erfüllt* eine Eigenschaft, wenn er in dieser Menge liegt; anderenfalls *verletzt* er die Eigenschaft. Wenn alle Abläufe eines verteilten Transitionssystems eine Eigenschaft erfüllen, dann *erfüllt das System* diese Eigenschaft.
Als Eigenschaft formuliert ist der wechselseitige Ausschluß die Menge

aller Abläufe, in denen keine zwei nebenläufige Stellen vorkommen, die mit $e_1$ bzw. $e_2$ beschriftet sind. Wir nennen diese Eigenschaft im folgenden kurz die *Mutex-Eigenschaft*[2].
Wir werden nun ein System angeben, welches die Mutex-Eigenschaft erfüllt. Dazu ergänzen wir unser erstes Beispiel wie in Abb. 1.3 dargestellt. Die beiden Philosophen synchronisieren sich über eine gemeinsame Stelle $g$. Diese Stelle modelliert eine Gabel, die die Philosophen zum Essen benötigen. Da es nur eine Gabel gibt und ein Philosoph nur mit einer Gabel ißt, essen niemals beide Philosophen zugleich. Abbildung 1.4 zeigt einen Ablauf dieses Systems. Das System besitzt

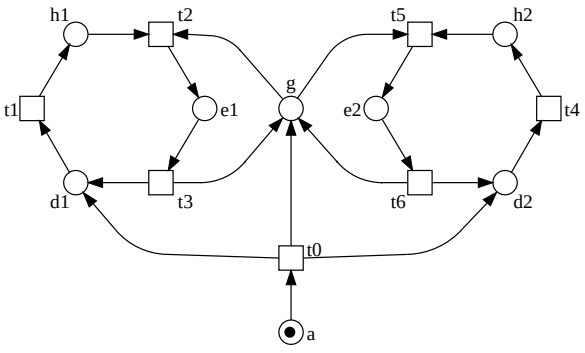



Abbildung 1.3: Wechselseitiger Ausschluß für $e_1$ und $e_2$

jedoch noch viele andere Abläufe. In dem dargestellten Ablauf ißt Philosoph 1 vor Philosoph 2. Ein anderer Ablauf ergibt sich, wenn zunächst Philosoph 2 ißt und der andere Philosoph solange wartet. Alle Abläufe des Systems aus Abb. 1.3 erfüllen die Mutex-Eigenschaft und damit erfüllt auch das System die Mutex-Eigenschaft.

In der Literatur hat sich die Unterscheidung von *Sicherheits-* und *Lebendigkeitseigenschaften* durchgesetzt[3]. Informell fordert eine Sicherheitseigenschaft, daß in einem Ablauf nie etwas „Unerwünschtes“ eintritt. Eine Lebendigkeitseigenschaft fordert, daß in einem Ablauf irgendwann etwas „Erwünschtes“ eintritt [49]. Die Mutex-Eigenschaft

[2]Mutex steht für die englische Bezeichnung des wechselseitigen Ausschluß: mutual exclusion.

[3]Wir weisen hier — wie Lamport [49] vor fast 20 Jahren — darauf hin, daß sich diese Begriffe von den entsprechenden Begriffen der Petrinetztheorie unterscheiden.

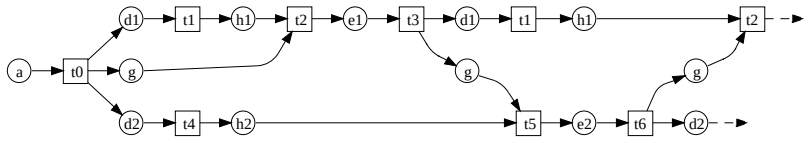



Abbildung 1.4: Ein Ablauf des Systems mit wechselseitigem Ausschluß

ist demgemäß eine Sicherheitseigenschaft. Das „Unerwünschte“ ist in diesem Fall das nebenläufige Auftreten zweier Stellen, wobei eine mit $e_1$ und die andere mit $e_2$ beschriftet ist.

In der Literatur werden Sicherheits- und Lebendigkeitseigenschaften meist nur für sequentielle Abläufe formalisiert. Wenn nur Sequenzen[4] als Abläufe betrachtet werden, besitzt jede Sicherheitseigenschaft eine wichtige Charakteristik: Wenn ein Ablauf eine Sicherheitseigenschaft verletzt, dann gibt es in diesem Ablauf ein eindeutiges erstes Ereignis, das die Sicherheitseigenschaft verletzt; nämlich das erste Ereignis mit dem das Unerwünschte eintritt. Dieses Ereignis ist für das Verletzen der Sicherheitseigenschaft verantwortlich.
Wir werden nun am Beispiel der Mutex-Eigenschaft zeigen, daß in verteilten Abläufen nicht unbedingt eine einzelne Transition existiert, die für das Verletzen der Sicherheitseigenschaft „verantwortlich“ ist. Sicherheitseigenschaften für verteilte Abläufe unterscheiden sich also in diesem Punkt von Sicherheitseigenschaften für sequentielle Abläufe. Die beiden Abläufe aus Abb. 1.5 erfüllen die Mutex-Eigenschaft. Also


Abbildung 1.5: Zwei Abläufe, die die Mutex-Eigenschaft erfüllen

verletzt keine der Transitionen, die in diesen Abläufen vorkommen, die Mutex-Eigenschaft. In Abb. 1.6 ist eine gemeinsame Fortsetzung

[4]Der Einfachheit halber betrachten wir hier nur Sequenzen von Ereignissen.

der beiden Abläufe dargestellt. In diesem Ablauf kommen nur Transitionen vor, die auch in den beiden vorangegangenen Abläufen aufgetreten sind. Trotzdem verletzt der Ablauf aus Abb. 1.6 die Mutex-

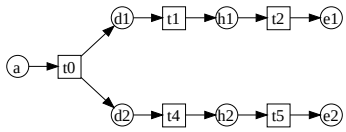



Abbildung 1.6: Ein Ablauf, der die Mutex-Eigenschaft verletzt

Eigenschaft. Wir sehen also, daß im verteilten Fall durch das unabhängige (d.h. nebenläufige) Eintreten von Transitionen eine einzelne für das Verletzen der Sicherheitseigenschaft verantwortliche Transition nicht mehr existiert. Die nebenläufigen Transitionen $t_2$ und $t_5$ im Ablauf sind gemeinsam für das Verletzen der Mutex-Eigenschaft verantwortlich.

Dieses Phänomen rührt nicht von unserem Ablaufmodell her. Vielmehr erfaßt unser Ablaufmodell dieses Phänomen der „Wirklichkeit“, welches von sequentiellen Abläufen nicht erfaßt wird.

Im sequentiellen Fall hat sich die Unterscheidung von Sicherheits- und Lebendigkeitseigenschaften weitgehend durchgesetzt. Dies liegt einerseits daran, daß die mathematische Formalisierung [6] die intuitive Vorstellung sehr gut trifft, und andererseits daran, daß zum Beweis von Sicherheits- und Lebendigkeitseigenschaften zwei völlig verschiedene Beweistechniken benutzt werden. Darüber hinaus bilden Sicherheits- und Lebendigkeitseigenschaften eine elegante mathematische Struktur auf den Abläufen: Die Sicherheitseigenschaften entsprechen den geschlossenen Mengen einer Topologie; die Lebendigkeitseigenschaften sind die dichten Mengen dieser Topologie.

Die intuitive Vorstellung von Sicherheits- und Lebendigkeitseigenschaften ist nicht auf sequentielle Abläufe beschränkt. Deshalb ist ein Modell für verteilte Abläufe nur tragfähig, wenn sich in diesem Modell Sicherheits- und Lebendigkeitseigenschaften charakterisieren lassen. Wir werden deshalb in Kapitel 3 Sicherheits- und Lebendigkeitseigenschaften für verteilte Abläufe charakterisieren und den oben vorgestellten Unterschied zum sequentiellen Fall genauer untersuchen.

## 1.2 Module

Häufig sind reaktive Systeme in eine Umgebung eingebettet und sollen auf Ereignisse der Umgebung reagieren. Beispielsweise sind aus der Sicht eines Telefonsystems die Benutzer die Umgebung. Die Benutzer interagieren mit dem Telefonsystem, indem sie beispielsweise den Hörer eines Telefons abnehmen und eine Nummer wählen. Das Telefonsystem muß daraufhin eine Verbindung mit dem entsprechenden Teilnehmer aufbauen. Ein einzelnes Telefon ist ein System, dessen Umgebung sich aus dem Telefonbenutzer und der Telefondose, in die es eingesteckt ist, zusammensetzt.

An diesem Beispiel werden bereits einige wichtige Aspekte deutlich. Einerseits besteht ein komplexes System — wie im Beispiel das Telefonsystem — aus vielen einfacheren Systemen. Es ist für die Entwicklung solcher Systeme wichtig, daß sich die Beschreibung des Verhaltens des Gesamtsystems aus der Beschreibung des Verhaltens der einzelnen Komponenten in einfacher Weise gewinnen läßt. Andererseits kann ein System häufig nur dann einen gewünschten Dienst erbringen, wenn sich die Umgebung „richtig“ verhält. So wird beim Telefon beispielsweise nur dann eine korrekte Verbindung aufgebaut, wenn der Benutzer nach dem Abheben des Hörers wartet, bis er das Freizeichen hört, und das Telefon in die dafür vorgesehene Dose eingesteckt ist. Die Umgebung eines Systems sind teilweise Menschen (z.B. der Telefonbenutzer), die das System benutzen, teilweise andere technische Systeme. Insbesondere beim Menschen ist offensichtlich, daß das System sein Verhalten nicht beeinflußen kann. Wenn die Umgebung (z.B. der Mensch) sich nicht wie gewünscht verhält, wird aber das System seinerseits bestimmte gewünschte Dienste nicht erbringen können. Wir werden in dieser Arbeit einen Formalismus entwickeln, um Systemkomponenten in dieser Weise zu beschreiben.

Wir werden nun den in dieser Arbeit verfolgten Ansatz zur Beschreibung von Systemkomponenten in Abhängigkeit von ihrer Umgebung anhand eines Beispiels vorstellen. Wir geben ein System an, das als Umgebung die beiden in Abschnitt 1.1 vorgestellten Philosophen besitzt. Die beiden Philosophen interagieren mit diesem System. Wenn sie sich an bestimmte Regeln halten, dann gewährleistet das System den wechselseitigen Ausschluß (beim Essen). Wir nennen dieses System Mutex-System. Wir legen zunächst fest, wie die Philosophen mit dem System kommunizieren. Dazu führen wir den Begriff der *Schnittstelle* als besonders ausgezeichnete Stellen des Systems ein. Auf diese Stellen dürfen die Philosophen entweder nur eine Marke legen oder sie

wegnehmen. Auf diese Weise modellieren wir *Kommunikationskanäle* zwischen System und Umgebung. Durch Pfeilspitzen in diesen Schnittstellen kennzeichnen wir die Richtung der Kommunikationskanäle. Die Festlegung der Schnittstelle entspricht in gewisser Weise der Vereinbarung einer Steckernorm.

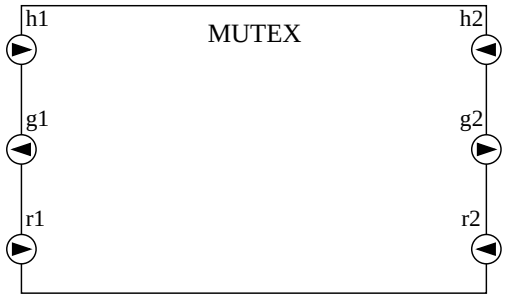



Abbildung 1.7: Die Schnittstelle des Mutex-Systems

In Abb. 1.7 ist die Schnittstelle des Mutex-Systems dargestellt. Wir beschreiben kurz die Bedeutung der verschiedenen Schnittstellen: Über $h_i$ teilt der jeweilige Philosoph $i$ dem Mutex-System mit, daß er hungrig ist und gerne essen würde. Durch $g_i$ wird dem Philosophen $i$ vom System eine Gabel zugeteilt, was bedeutet, daß er mit dem Essen beginnen darf. Wenn der Philosoph $i$ satt ist, gibt er die Gabel über $r_i$ zurück. Das Mutex-System selbst ist in Abb. 1.8 dargestellt. In diesem System gibt es genau eine Gabel $g$, die je nach Anforderung einem der beiden Philosophen $i$ über $g_i$ zugeteilt wird.

In Abb. 1.9 ist das Mutex-System in eine Umgebung von zwei Philosophen eingebettet. Wir lassen nur solche Umgebungen zu, die zumindest die Richtung der Schnittstellen respektieren. Zusammen erhalten wir im wesentlichen wieder das bereits bekannte System aus Abb. 1.3, welches die Mutex-Eigenschaft offensichtlich erfüllt. Um den Zusammenhang zum System aus Abb. 1.3 zu verdeutlichen, haben wir die Transitionen entsprechend beschriftet.

Wir können jedoch leicht eine Umgebung angeben, die zwar die Richtung der Schnittstellen respektiert, für die der wechselseitige Ausschluß von $e_1$ und $e_2$ nicht gewährleistet ist. In Abb. 1.10 sind zwei Philosophen als Umgebung des Mutex-Systems dargestellt, für die die Mutex-Eigenschaft nicht gilt: Der erste Philosoph fordert zwar eine Gabel an,

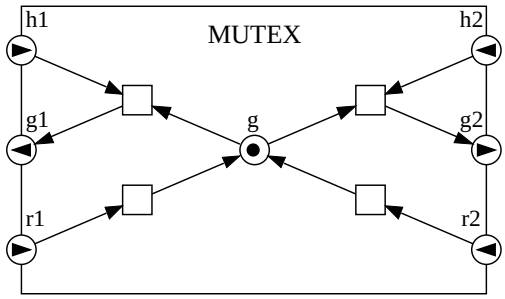



Abbildung 1.8: Das Mutex-System

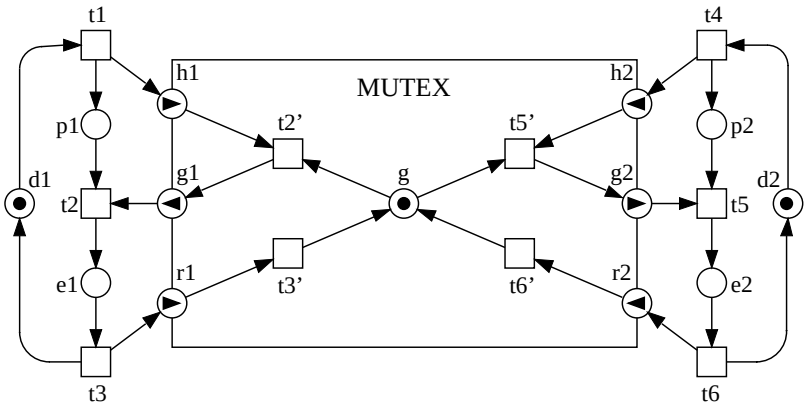



Abbildung 1.9: Das Mutex-System in einer Umgebung

beginnt aber schon ohne Gabel zu essen, der zweite Philosoph gibt sie zurück bevor er überhaupt mit dem Essen begonnen hat und ißt dann ohne Gabel. Wir sehen sofort, daß das Mutex-System in dieser Umgebung Abläufe besitzt, in denen beide Philosophen zugleich essen.

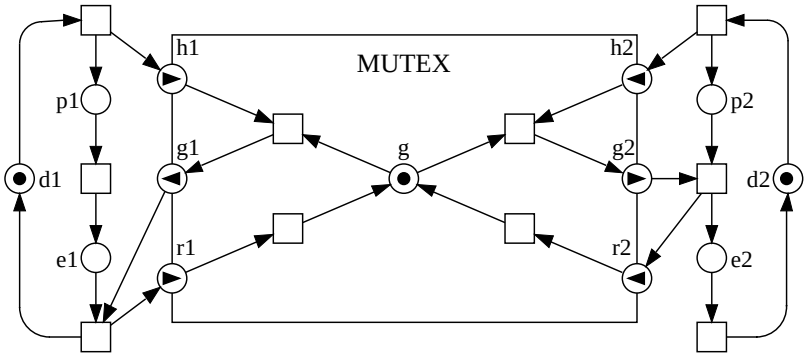



Abbildung 1.10: Das Mutex-System in einer „falschen“ Umgebung

Das Mutex-System kann also nur unter gewissen Annahmen an die Umgebung den wechselseitigen Ausschluß gewährleisten. Informell fordern wir von den beiden Philosophen, daß sie nur mit einer Gabel essen und die Gabel nicht vervielfältigen. Dazu benutzen wir die Notation **p-i**$(g_i, e_i, r_i)$, die zusammen mit der Aussage $\neg e_i$ besagt, daß die Umgebung (Philosoph $i$) nur mit einer Gabel mit dem Essen beginnt und die Gabel mit dem Beenden des Essens an das Mutex-System zurück gibt. Die Mutex-Eigenschaft notieren wir durch $\Box\neg(e_1 \wedge e_2)$. Das gewünschte Verhalten des Mutex-Systems beschreiben wir durch den *RG-Graphen*[5], der in Abb. 1.11 dargestellt ist. Dieser RG-Graph *spezifiziert* das gewünschte Verhalten der Systemkomponente (im Rechteck angegeben) in Abhängigkeit von dem Verhalten der Umgebung (in den Rauten angegeben). Allgemein kann ein RG-Graph mehrere Rechtecke und Rauten mit einer komplexeren Abhängigkeitsstruktur enthalten. Ein System *implementiert* einen RG-Graphen, wenn jede Eigenschaft in einem Rechteck des RG-Graphen unter der Annahme erfüllt ist, daß die Umgebung alle Eigenschaften in den Rauten erfüllt, die davor liegen. Ein *Modul* besteht aus einer Schnittstelle und einem RG-Graphen,

[5]RG steht für Rely-Guarantee.

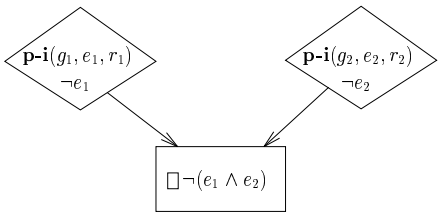



Abbildung 1.11: RG-Graph für das Mutex-System

der das Verhalten spezifiziert.
Wir werden in Kapitel 5 sehen, wie man mit Hilfe eines RG-Graphen sehr differenzierte Annahmen über das Zusammenspiel von Eigenschaften eines Systems und seiner Umgebung treffen kann. Darüber hinaus werden wir zeigen, wie man RG-Graphen kombinieren und modifizieren kann. Auf diese Weise können wir Module kombinieren ohne ihre Implementierung zu kennen.

## 1.3 Überblick über diese Arbeit

In dieser Arbeit spielen die Begriffe der *Verteiltheit* und der *Modularität* eine zentrale Rolle. Dies spiegelt sich auch im Aufbau dieser Arbeit wider. Wir behandeln diese beiden Aspekte unabhängig voneinander. Der benutze Formalismus für verteilte Systeme wurde so gewählt, daß er zur formalen Behandlung beider Aspekte gleichermaßen geeignet ist.
Bestimmte Entscheidungen bei der Festlegung unseres Formalismuses mögen zunächst etwas willkürlich erscheinen. Sie erklären sich aber damit, daß sie entweder im Hinblick auf die Verteiltheit oder im Hinblick auf die Modularität notwendig oder zumindest hilfreich sind. Ein Beispiel für eine solche Entscheidung ist die Wahl von S/T-Systemen als zugrundeliegende Petrinetz-Klasse. Eigentlich sind wir — wie an unseren Beispielen deutlich wird — nur an Systemen interessiert, die 1-sicher sind, d.h. bei denen in keinem Ablauf zwei nebenläufige Stellen gleich beschriftet sind. Für eine Systemkomponente mit einer beliebigen Umgebung ist es aber nur schwer möglich diese Eigenschaft a priori zu gewährleisten. Deshalb betrachten wir zunächst beliebige S/T-Systeme. Die 1-Sicherheit können wir ggf. durch eine Spezifika-

tion explizit fordern.

In Kapitel 2 führen wir die mathematischen Grundlagen ein. Es handelt sich im wesentlichen um Standardbegriffe aus der Mathematik und der Petrinetztheorie [80, 14]. In Abschnitt 2.5 wird der zentrale Begriff des verteilten Transitionssystems (S/T-Systems) und seiner verteilten Abläufe eingeführt. Darüber hinaus werden in Kapitel 2 einige Beweismethoden eingeführt die erst in den folgenden Kapiteln benutzt werden. Am Ende jedes Kapitels stellen wir die Verbindung der von uns eingeführten Begriffe mit ähnlichen Begriffen, die in der Literatur bereits bekannt sind, her.

In Kapitel 3 werden wir Abläufe unabhängig von Systemen betrachten. Wir definieren Sicherheits- und Lebendigkeitseigenschaften für verteilte Abläufe analog zu [6]. Dazu benötigen wir nur eine Präfixordnung auf verteilten Abläufen. Der *Zerlegungssatz*[6] von Alpern und Schneider gilt auch für unsere Charakterisierung von Sicherheits- und Lebendigkeitseigenschaften.
Der Schwerpunkt dieses Kapitels liegt auf dem Nachweis, daß die Präfixordnung auf der Menge der verteilten Abläufe ein *Scott-Bereich* ist. Die Charakterisierung von Sicherheits- und Lebendigkeitseigenschaften läßt sich damit kanonisch auf verteilte Abläufe übertragen und der Zerlegungssatz folgt aus dieser Eigenschaft. Ein zweiter Schwerpunkt ist die Herausarbeitung des zuvor diskutierten Unterschiedes zum sequentiellen Fall. Bestimmte Entwurfsmethoden nutzen spezielle Eigenschaften von Sicherheitseigenschaften aus, die im verteilten Fall im allgemeinen nicht gelten. Deshalb charakterisieren wir hier solche Sicherheitseigenschaften, die diese speziellen Eigenschaften erfüllen und deshalb eine besondere Bedeutung haben.

Kapitel 4 und 5 sind die zentralen Kapitel dieser Arbeit. In Kapitel 4 werden wir den Begriff der Systemkomponente formalisieren. Dazu erweitern wir ein verteiltes Transitionssystem um Lebendigkeitsannahmen, die das Schalten gewisser Transitionen erzwingen, und um Schnittstellen, über die die Systemkomponente mit ihrer Umgebung kommuniziert. Systemkomponenten können an diesen Schnittstellen zusammengesetzt werden. Ein Beispiel dafür haben wir im vorangegangenen Abschnitt gesehen. Das wesentliche Ergebnis dieses Kapitels ist, daß das Verhalten der Systemkomponenten kompositional ist:

[6] Jede Eigenschaft ist der Durchschnitt einer geeigneten Sicherheits- und Lebendigkeitseigenschaft.

D.h. das Verhalten des zusammengesetzen Systems ergibt sich allein aus dem Verhalten der Teilkomponenten.
In Kapitel 5 geben wir einen Formalismus zum Spezifizieren von Systemkomponenten an. Dazu führen wir den Begriff des Moduls ein. Das Verhalten der Systemkomponente wird in Abhängigkeit vom Verhalten der Umgebung beschrieben[7]. Aufbauend auf der Kompositionalität des Verhaltens der Systemkomponenten, können Module unabhängig von einer Implementierung kombiniert werden. Das Modulkonzept ermöglicht also das Zerlegen eines Moduls in Teilmodule. Die Teilmodule können dann unabhängig voneinander implementiert werden. In Kapitel 8 werden wir dies an einem Beispiel demonstrieren.

In Kapitel 6 stellen wir eine temporale Logik zur syntaktischen Repräsentation von Eigenschaften vor. Die Eigenschaften, die wir zum Spezifizieren in Modulen benutzen, sind Abkürzungen für spezielle temporallogische Aussagen. Neben *Invarianten* und *Leadsto-Eigenschaften* integrieren wir *S-Invarianten* in diese Logik. Sie erhalten ihre Bedeutung durch die *partiellen S-Invarianten*, die für die Spezifikation von Modulen besonders nützlich sind. In Kapitel 7 führen wir Beweisregeln für diese Eigenschaften ein.

In Kapitel 8 zeigen wir am Beispiel eines speziellen Mutex-Protokolls, wie man ein Modul in Teilmodule zerlegen und die Teilmodule einzeln implementieren und verifizieren kann.

[7] Dafür hat sich der Begriff Rely-Guarantee-Style [42] durchgesetzt

# Kapitel 2

# Grundlagen

In diesem Kapitel führen wir die grundlegenden Notationen und Begriffe ein, die wir in dieser Arbeit benutzen. Zunächst stellen wir die allgemein üblichen Notationen für Relationen und Funktionen zusammen. Dann führen wir spezielle Begriffe der „Concurrency-“ bzw. Ordnungstheorie ein. Darauf aufbauend werden die Begriffe der Netztheorie gebildet. Wir halten uns dabei soweit wie möglich an die Definitionen von [80] und [14].

Die zentralen Begriffe sind das *Netz*, auf dem später der Begriff des *verteilten Transitionssystems* aufbaut, und das *Prozeßnetz*, das zur Formalisierung verteilter Abläufe benutzt wird. Kurzgefaßt ist ein Prozeßnetz ein abzählbares, stellenberandetes und vorgänger-endliches Kausalnetz.

Außerdem führen wir *Netzhomomorphismen* ein, die den Zusammenhang zwischen Systemen und ihren Abläufen formalisieren. Ein *Ablauf* eines verteilten Transitionssystem ist ein Prozeßnetz zusammen mit einem speziellen Netzhomomorphismus vom Prozeßnetz in das verteilte Transitionssystem. Abläufe verteilter Transitionssysteme entsprechen also im wesentlichen Prozessen von S/T-Systemen [32]. Darüber hinaus werden wir in dieser Arbeit Netzhomomorphismen zur Definition der Präfixrelation auf Abläufen benutzen. Wir werden für den Umgang mit Netzhomomorphismen einige Beweistechniken einführen, die wir später bei der Untersuchung der Präfixrelation benötigen.

Ein wichtiger Teil dieser Arbeit ist der Nachweis, daß die Präfixrelation ein Scott-Bereich ist. Dazu definieren wir hier die speziellen Anforderungen eines Scott-Bereiches.

Leser, die mit diesen Grundbegriffen vertraut sind, können die ersten Abschnitte zunächst überspringen und mit Abschnitt 2.5 beginnen, in dem *verteilte Transitionssysteme* und ihre *Abläufe* definiert werden. Wenn nötig, kann die Definitionen der einzelnen Begriffe später mit Hilfe des Index nachgeschlagen werden.

## 2.1 Mengen, Relationen und Abbildungen

Hier vereinbaren wir Schreibweisen für die üblichen mathematischen Begriffe. Dabei benutzen wir das Symbol $\widehat{=}$ zur Definition von Begriffen, um Verwechslungen mit dem normalen Gleichheitszeichen $=$ zu vermeiden.

### 2.1.1 Mengen

Mit $\mathbb{N}$ bezeichnen wir die Menge der *natürlichen Zahlen* inklusive 0. Für die *leere Menge* schreiben wir $\emptyset$. Für eine Menge $X$ notieren wir mit $x \in X$ und $x \notin X$, wenn $x$ ein Element von $X$ ist bzw. $x$ kein Element von $X$ ist. Mengen charakterisieren wir durch die Notation $\{x \mid \ldots\}$.
Für zwei Mengen $X$ und $Y$ bezeichnet $X \subseteq Y$ die *Inklusion* von $X$ in $Y$ und $X \subset Y$ die *echte Inklusion*. Die *Vereinigung*, den *Durchschnitt* und die *Differenz* der Mengen notieren wir durch $X \cup Y$, $X \cap Y$ bzw. $X \setminus Y$. Die unendliche Vereinigung bzw. der unendliche Durchschnitt einer Familie von Mengen $(X_i)_{i \in I}$ notieren wir durch $\bigcup_{i \in I} X_i$ bzw. $\bigcap_{i \in I} X_i$.
Für das *Komplement* einer Menge $X$ bzgl. einer anderen Menge $Y$ schreiben wir $\neg X \mathrel{\widehat{=}} X \setminus Y$, wenn sich $Y$ aus dem Kontext ergibt. Zwei Mengen heißen *disjunkt*, wenn gilt $X \cap Y = \emptyset$. Eine Familie von Mengen $(X_i)_{i \in I}$ heißt *paarweise disjunkt*, wenn für jedes $i$ und $j$ aus $X_i \cap X_j \neq \emptyset$ folgt $i = j$.
Mit $|X|$ bezeichnen wir die *Kardinalität* einer Menge, die für endliche Mengen der Anzahl der Elemente von $X$ entspricht.
Die Menge aller Teilmengen einer Menge $X$ heißt *Potenzmengen* von $X$ und wir schreiben dafür $2^X$.
Für zwei Mengen $X$ und $Y$ bezeichnet $X \times Y \mathrel{\widehat{=}} \{(x, y) \mid x \in X, y \in Y\}$ das *Produkt* von $X$ und $Y$.

### 2.1.2 Abbildungen

Eine *Abbildung* $f$ von $X$ nach $Y$ notieren wir durch $f : X \to Y$. Meist verstehen wir unter einer Abbildung eine totale Abbildung. Wenn eine

Abbildung partiell ist, d.h. wenn für gewisse Werte $f$ nicht definiert ist, reden wir explizit von einer *partiellen Abbildung*.
Für $x \in X$ bezeichnet $f(x)$ das Bild von $x$. Für $f : X \to Y$ und $g : Y \to Z$ ist $g \circ f : X \to Z$ eine Abbildung und es gilt für alle $x \in X$: $g \circ f(x) = f(g(x))$.
Mit $f^{-1}$ bezeichnen wir die *Umkehrung* von $f$, die im allg. keine Abbildung ist. Für $y \in Y$ bezeichnet $f^{-1}(y) \mathrel{\widehat{=}} \{x \mid f(x) = y\}$. Wenn $f^{-1}$ eine Abbildung ist, nennen wir sie die *inverse Abbildung* von $f$.
Für eine Menge $X' \subseteq X$ und $Y' \subseteq Y$ definieren wir $f(X') = \bigcup_{x \in X'} f(x)$ und $f^{-1}(Y') = \bigcup_{y \in Y'} f(y)$. Für eine Abbildung $f : X \to Y$ definieren wir $f$ auf $X \times X$ durch $f : X \times X \to Y \times Y$ mit $f(x_1, x_2) = (f(x_1), f(x_2))$.
Für $X' \subseteq X$ bezeichnet $f|_{X'} : X' \to Y$ die *Restriktion* von $f$ auf $X'$.
Für zwei disjunkte Mengen $X$ und $X'$ und zwei Abbildungen $f : X \to Y$ und $f' : X' \to Y'$ definieren wir die *Vereinigung* $f \cup f'$ von $f$ und $f'$ durch $f \cup f' : X \cup X' \to Y \cup Y'$, wobei für $x \in X$ gilt $(f \cup f')(x) = f(x)$ und für $x \in X'$ gilt $(f \cup f')(x) = f'(x)$.
Eine Abbildung $f : X \to Y$ heißt *injektiv*, wenn für $x, y \in X$ aus $f(x) = f(y)$ folgt $x = y$. $f$ heißt *surjektiv*, wenn gilt $f(X) = Y$. $f$ heißt *bijektiv*, wenn $f$ injektiv und surjektiv ist.

### 2.1.3 Abzählbare Mengen

Eine Menge $X$ heißt *abzählbar*, wenn es eine injektive Abbildung $f : X \to \mathbb{N}$ gibt. Gemäß dieser Definition sind alle endlichen Mengen abzählbar. Wenn wir die endlichen Mengen ausschließen wollen, sprechen wir von *abzählbar unendlichen* Mengen.

### 2.1.4 Multimengen

Eine *Multimenge* über $X$ ist eine Abbildung $M : X \to \mathbb{N}$. Eine Multimenge $M$ über $X$ heißt endlich, wenn die Menge $\{n \in \mathbb{N} \mid M(n) \neq 0\}$ endlich ist. Sind $M$ und $M'$ zwei Multimengen über $X$, so gilt $M \leq M'$ genau dann, wenn für jedes $x \in X$ gilt $M(x) \leq M'(x)$. Ebenso definieren wir die Operation $+$ und $-$ elementweise. Dabei ist $M - M'$ nur für $M' \leq M$ definiert.
Jede Menge $Y \subseteq X$ können wir als spezielle Multimenge $Y : X \to \{0, 1\}$ auffassen. Umgekehrt können wir eine Multimenge $M$, für die für jedes $x \in X$ $M(x) \leq 1$ gilt, als Menge interpretieren. Wir sagen dann $M$ ist eine Menge. Auf „Mengen-Multimengen“ sind dann die

Mengenoperatoren $\cup$, $\cap$ und $\setminus$ anwendbar. Insbesondere gilt für diese Multimengen $M \leq M'$ genau dann, wenn $M \subseteq M'$.

### 2.1.5 Relationen

Für $R \subseteq X \times X$ nennen wir das Paar $(X, R)$ eine (zweistellige) *Relation* über $X$. Wenn sich $X$ aus dem Kontext ergibt, schreiben wir meist nur $R$ an Stelle von $(X, R)$. Wir schreiben auch $x\ R\ y$ an Stelle von $(x, y) \in R$. Die *identische Relation id* über einer Menge $X$ ist definiert durch $id \mathrel{\widehat{=}} \{(x, x) \mid x \in X\}$.
Für eine Relation $R$ über $X$ und eine Menge $X' \subseteq X$ ist $R|_{X'} \mathrel{\widehat{=}} \{(x, y) \mid x, y \in X' \text{ und } x\ R\ y\}$ die Restriktion von $R$ auf $X'$.
Eine Relation $(X, R)$ heißt:

1. *reflexiv*, wenn für jedes $x \in X$ gilt $x\ R\ x$.
2. *irreflexiv*, wenn für kein $x \in X$ gilt $x\ R\ x$.
3. *transitiv*, wenn für alle $x, y, z \in X$ mit $x\ R\ y$ und $y\ R\ z$ auch $x\ R\ z$ gilt.
4. *symmetrisch*, wenn für alle $x, y \in X$ mit $x\ R\ y$ auch $y\ R\ x$ gilt.
5. *antisymmetrisch*, wenn für alle $x, y \in X$ aus $x\ R\ y$ und $y\ R\ x$ folgt $x = y$.
6. *konnex*, wenn für alle $x, y \in X$ $x = y$, $x\ R\ y$ oder $y\ R\ x$ gilt.

Die *transitive Hülle* einer Relation $R$ ist die kleinste Relation, die $R$ umfaßt, und transitiv ist. Wir bezeichnen die transitive Hülle von $R$ mit $R^+$. Die *transitiv-reflexive Hülle* von $R$ bezeichnen wir mit $R^*$. Eine reflexive, transitive und symmetrische Relation heißt *Äquivalenzrelation*.

## 2.2 Geordnete Mengen

Geordnete Mengen (kurz: Ordnungen) spielen in dieser Arbeit ein wichtige Rolle. Einerseits können wir Abläufe eines verteilten Systems als spezielle Ordnung auffassen. Andererseits werden wir auf der Menge der Abläufe die Präfixrelation definieren, die ebenfalls eine Ordnung ist. Wir werden in dieser Arbeit zeigen, daß die Präfixordnung auf

Abläufen ein Scott-Bereich ist. Deshalb führen wir hier auch die Begriffe ein, die wir zur Definition des Scott-Bereiches benötigen. Insbesondere definieren wir den Begriff der gerichteten Menge und der vollständigen Ordnung (cpo).
Neben diesen beiden Anwendungen der Ordnung, treten Ordnungen in dieser Arbeit an vielen anderen Stellen auf.

**Definition 2.1 (Ordnung, lineare Ordnung)**

Sei $X$ eine Menge. Eine Relation $R$ über $X$ heißt *Ordnung* auf $X$, wenn $R$ irreflexiv und transitiv ist.

Die Ordnung $R$ heißt *linear*, wenn $R$ konnex ist.

Manchmal bezeichnet man auch eine reflexive, transitive und antisymmetrische Relation als Ordnung. Eine solche Ordnung nennen wir zur Unterscheidung *reflexive Ordnung*. Eine Ordnung gemäß Definition 2.1 nennen wir, wenn eine Unterscheidung notwendig ist, *strikte Ordnung*. Für Ordnungen benutzen wir spezielle Symbole, die schon syntaktisch darauf hinweisen, daß die betrachtete Relation eine Ordnung ist: Für ein strikte Ordnungen benutzen wir meist das Symbol $<$, für die entsprechende reflexive Ordnung benutzen wir das Symbol $\leq$. Auch die Symbole $\sqsubset$, $\sqsubseteq$ und $\prec$, $\preceq$ verwenden wir ausschließlich für eine strikte bzw. die entsprechende reflexive Ordnung.
Graphisch können wir die Elemente $X$ einer Ordnung $(X, <)$ durch beschriftete Knoten darstellen. Die Beziehung $x < y$ wird durch einen Pfeil von $x$ nach $y$ dargestellt. In Abb. 2.1 ist die übliche „Kleiner-Beziehung“ $(\mathbb{N}, <)$ zwischen den natürlichen Zahlen auf diese Weise dargestellt.

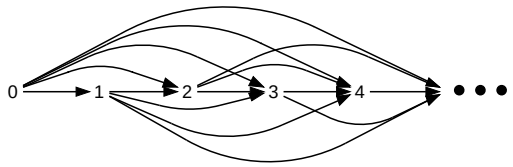



Abbildung 2.1: Graphische Darstellung der Ordnung $(\mathbb{N}, <)$

Diese Darstellung hat jedoch den Nachteil, daß sie sehr viele Pfeile enthält, die sich bereits aus der Transitivität der Ordnung ergeben. Beispielsweise können in Abb. 2.1 die Pfeile zwischen 0 und 2, 0 und

3 usw. entfallen. Die Ordnung auf den natürlichen Zahlen kann deshalb übersichtlicher als *Hassediagramm* dargestellt werden, in dem alle transitiven Abhängigkeiten weggelassen werden. Dies ist in Abb. 2.2 dargestellt.

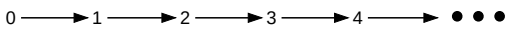



Abbildung 2.2: Hassediagramm für $(\mathbb{N}, <)$

Für eine Ordnung $<$ definieren wir das zugehörige Hassediagramm als selbständige Relation, die wir mit $\lessdot$ bezeichnen. Wenn das Hassediagramm $\lessdot$ einer Ordnung $<$ die Ordnung vollständig repräsentiert, nennen wir die Ordnung *kombinatorisch*.

**Definition 2.2 (Hassediagramm, kombinatorische Ordnung)**

Für eine Ordnung $(X, <)$ definieren wir das zugehörige *Hassediagramm* $\lessdot$ als Relation über $X$ wie folgt: $x \lessdot y$, wenn $x < y$ und für kein $z \in X$ gilt $x < z < y$.

Wenn $\lessdot^+ = <$ gilt, dann heißt die Ordnung $<$ *kombinatorisch*.

Die Ordnung $(\mathbb{N}, <)$ ist — wie wir an den obigen Abbildungen sehen — kombinatorisch. Ein Beispiel für eine nicht-kombinatorische Ordnung ist in Abb. 2.3 dargestellt. Die Menge $\mathbb{N}^\omega$ enthält ein zusätzliches Element $\omega$, das größer als alle anderen Elemente von $\mathbb{N}$ ist. Im Hassediagramm $\lessdot$ dieser Ordnung fallen alle Pfeile zwischen $n \in \mathbb{N}$ und $\omega$ weg. Die meisten Ordnungen, die wir in dieser Arbeit betrachten, sind kombinatorisch.

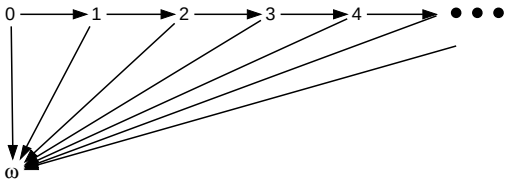



Abbildung 2.3: Darstellung der Ordnung $(\mathbb{N}^\omega, <)$

Für ein Element einer Ordnung wollen wir manchmal über die Vorgänger, die unmittelbaren Vorgänger und die unmittelbaren Nachfolger

bzgl. der Ordnung reden. Wir führen für diese Mengen Abkürzungen ein. Außerdem führen wir eine Abkürzung für die minimalen und maximalen Elemente einer Ordnung ein.

**Notation 2.3 (Punktnotation für Ordnungen)**

Sei $(X, <)$ eine Ordnung und $x \in X$. Wir definieren:

1. $\downarrow x \mathrel{\widehat{=}} \{y \in X \mid y < x\}$.
2. ${}^\bullet x \mathrel{\widehat{=}} \{y \in X \mid y \lessdot x\}$.
3. $x^\bullet \mathrel{\widehat{=}} \{y \in X \mid x \lessdot y\}$.
4. ${}^\circ X \mathrel{\widehat{=}} \{x \in X \mid \text{es gibt kein } y \in X \text{ mit } y < x\}$.
5. $X^\circ \mathrel{\widehat{=}} \{x \in X \mid \text{es gibt kein } y \in X \text{ mit } x < y\}$.

Die Elemente von $\downarrow x$, ${}^\bullet x$ und $x^\bullet$ nennen wir die *Vorgänger*, die *unmittelbaren Vorgänger*, bzw. die *unmittelbaren Nachfolger* von $x$. Die Elemente aus ${}^\circ X$ und $X^\circ$ nennen wir *minimale* bzw. *maximale* Elemente der Ordnung.

Für Teilmengen $Y \subseteq X$ definieren wir $\downarrow Y \mathrel{\widehat{=}} \bigcup_{x \in Y} \downarrow x$, ${}^\bullet Y \mathrel{\widehat{=}} \bigcup_{x \in Y} {}^\bullet x$ und $Y^\bullet \mathrel{\widehat{=}} \bigcup_{x \in Y} x^\bullet$.

Wenn wir Abläufe mit Hilfe von Ordnungen formalisieren sind besonders zwei Relationen interessant: Die **li**-Relation drückt aus, daß zwei Elemente durch $<$ geordnet werden, die **co**-Relation drückt aus, daß zwei Elemente durch $<$ nicht geordnet werden (also in der Sprechweise der Einleitung nebenläufig sind).

**Definition 2.4 (Linien und Schnitte)**

Sei $(X, <)$ eine Ordnung. Wir definieren die Relationen **li** und **co** auf $X$:

$$\begin{array}{rll} x \mathbin{\mathbf{li}} y & \text{gdw.} & x \leq y \text{ oder } y \leq x \\ x \mathbin{\mathbf{co}} y & \text{gdw.} & x \not< y \text{ und } y \not< x \end{array}$$

Sei $Y \subseteq X$. $Y$ heißt **li**-*Menge* bezüglich $<$, wenn für je zwei $x, y \in Y$ gilt $x \mathbin{\mathbf{li}} y$. $Y$ heißt **co**-*Menge* bezüglich $<$, wenn für je zwei $x, y \in Y$ gilt $x \mathbin{\mathbf{co}} y$.

$Y$ heißt *Linie* von $<$ wenn $Y$ eine **li**-Menge ist und keine echte Obermenge von $Y$ **li**-Menge ist. $Y$ heißt *Schnitt* von $<$ wenn $Y$ eine **co**-Menge ist und keine echte Obermenge von $Y$ **co**-Menge ist.

Eine wichtige Rolle spielen vor allem die **co**-Mengen. In Abläufen entsprechen sie den Zuständen. Eine wichtige Beobachtung ist, daß die unmittelbaren Vorgänger und die unmittelbaren Nachfolger eines Elements $x$, sowie die minimalen und die maximalen Elemente einer Ordnung $<$ jeweils eine **co**-Menge dieser Ordnung sind.

**Lemma 2.5**

Sei $(X, <)$ eine Ordnung und $x, y, z \in X$. Dann gilt:

1. Aus $x, y \in {}^\bullet z$ oder $x, y \in z^\bullet$ folgt $x$ **co** $y$.
2. ${}^\circ X$ und $X^\circ$ sind **co**-Mengen.

Um später die Ordnungen, die als Grundlage für Abläufe dienen, zu charakterisieren, stellen wir hier einige weitere Eigenschaften von Ordnungen zusammen.

**Definition 2.6 (Eigenschaften von Ordnungen)**

Eine Ordnung $(X, <)$ heißt

1. *endlich-verzweigt*, wenn ${}^\bullet x$ und $x^\bullet$ für jedes $x \in X$ endliche Mengen sind.
2. *initialisiert*, wenn ${}^\circ X$ ein Schnitt der Ordnung ist.
3. *vorgänger-endlich*, wenn für jedes $x \in X$ die Menge der Vorgänger $\downarrow x$ endlich ist.
4. *fundiert*, wenn für jedes $x \in X$ und jede Linie $L$ der Ordnung die Menge $\downarrow x \cap L$ endlich ist.

Zwischen vorgänger-endlichen, fundierten und kombinatorischen Ordnungen besteht ein enger Zusammenhang:

**Bemerkung 2.7**

Jede vorgänger-endliche Ordnung ist fundiert. Jede fundierte Ordnung ist initialisiert. Jede vorgänger-endliche Ordnung ist kombinatorisch.

Jede fundierte und endlich-verzweigte Ordnung ist vorgänger-endlich.

Wenn eine Ordnung $(X, <)$ kombinatorisch und initialisiert ist, können wir die Menge $X$ induktiv aus den minimalen Elementen von $X$ generieren. Auf diese Weise können wir induktive Beweise über die Menge $X$ führen.

**Lemma 2.8 (Induktive Generierung der Elemente)**

Sei $(X, <)$ eine initialisierte kombinatorische Ordnung. Für jedes $n \in \mathbb{N}$ definieren wir die Menge $X_n$ induktiv:

1. $X_0 \mathrel{\widehat{=}} {}^\circ X$
2. $X_{n+1} \mathrel{\widehat{=}} X_n^\bullet$

Dann gilt $X = \bigcup_{n \in \mathbb{N}} X_n$.

**Beweis:** Wenn $(X, <)$ initialisiert ist, dann gilt für jedes $x \in X$ entweder $x \in {}^\circ X$ oder es existiert ein $y \in {}^\circ X$ mit $y < x$, denn sonst wäre ${}^\circ X$ kein Schnitt.

Im ersten Fall gilt $x \in X_0$ und somit $x \in \bigcup_{n \in \mathbb{N}} X_n$.

Im zweiten Fall gibt es (weil $(X, <)$ kombinatorisch ist) eine endliche Folge $y = x_0 \lessdot x_1 \lessdot x_2 \lessdot \ldots \lessdot x_n = x$ von Elementen aus $X$. Dann gilt mit der Definition von $x_i^\bullet$ und von $X_{i+1}$: $x \in X_n$. □

Das Lemma können wir unmittelbar anwenden, um die Menge der abzählbaren initialisierten, kombinatorischen und endlich-verzweigten Ordnungen zu charakterisieren.

**Satz 2.9 (Abzählbare Ordnungen)**

Sei $(X, <)$ eine initialisierte, endlich-verzweigte und kombinatorische Ordnung. Die Menge $X$ ist genau dann abzählbar, wenn ${}^\circ X$ abzählbar ist.

**Beweis:** Wir benutzen Lemma 2.8. Da $(X, <)$ endlich verzweigt ist, ist mit $X_n$ auch $X_{n+1}$ abzählbar. Da nach Voraussetzung $X_0$ abzählbar ist, ist jedes $X_n$ abzählbar.

Da $X$ die abzählbare Vereinigung von abzählbaren Mengen ist, ist $X$ abzählbar.

Die umgekehrte Richtung der Aussage ist offensichtlich: Wenn ${}^\circ X$ nicht abzählbar ist, dann ist $X \supseteq {}^\circ X$ erst recht nicht abzählbar. □

## 2.3 Scott-Bereich

Wir betrachten nun spezielle Eigenschaften von Ordnungen, die im Zusammenhang mit der Präfixbeziehung zwischen Abläufen eine Rolle spielen. Der zentrale Begriff dafür ist die vollständige (Halb-)Ordnung

(*cpo* = complete partial order) und noch spezieller der Scott-Bereich. In diesem Kontext betrachten wir immer eine reflexive Ordnung und benutzen dafür das Symbol $\sqsubseteq$.
Der Scott-Bereich ist eine Verfeinerung der vollständigen Ordnung. Er spielt immer dann eine Rolle, wenn unendliche Objekte durch endliche *approximiert* werden sollen. Die Approximation wird durch Supremumsbildung über speziellen Teilmengen formalisiert. Die Endlichkeit wird im Scott-Bereich durch die *kompakten* Elemente formalisiert. Die Kompaktheit eines Elements ist nur mit Hilfe der Ordnungsrelation definiert; die Struktur des Elements spielt dabei keine Rolle.
Wir definieren zunächst die kleinsten und größten Elemente einer Ordnung, um darauf aufbauend das Supremum (und Infimum) einer Menge zu definieren.

**Definition 2.10 (Kleinstes und größtes Element)**

Sei $(X, \sqsubseteq)$ eine reflexive Ordnung und $Y \subseteq X$ und $x \in Y$.

$x$ heißt *kleinstes Element* von $Y$ (bzgl. $\sqsubseteq$), wenn für jedes $y \in Y$ gilt $x \sqsubseteq y$. $x$ heißt *größtes Element* von $Y$ (bzgl. $\sqsubseteq$), wenn für jedes $y \in Y$ gilt $y \sqsubseteq x$.

Im Gegensatz zu den minimalen und maximalen Elementen einer Menge $Y$ ist das kleinste und größte Element immer eindeutig bestimmt (wenn es existiert).

**Definition 2.11 (Obere und untere Schranke)**

Sei $(X, \sqsubseteq)$ eine reflexive Ordnung, $Y \subseteq X$ eine nicht-leere Menge und $x \in X$.

$x$ heißt *obere Schranke* von $Y$, wenn für jedes $y \in Y$ gilt $y \sqsubseteq x$. $x$ heißt *untere Schranke* von $Y$, wenn für jedes $y \in Y$ gilt $x \sqsubseteq y$. Die Menge $Y$ heißt *kompatibel*, wenn $Y$ eine obere Schranke besitzt.

Die kleinste obere Schranke von $Y$ heißt das *Supremum* von $Y$, welches wir mit $\bigsqcup Y$ bezeichnen. Die größte untere Schranke von $Y$ heißt das *Infimum* von $Y$, welches wir mit $\sqcap Y$ bezeichnen.

Für eine beliebige Menge existiert möglicherweise das Supremum und das Infimum nicht. Beispielsweise existiert $\bigsqcup \mathbb{N}$ (bzgl. der Ordnung $\leq$) nicht; dagegen gilt $\sqcap \mathbb{N} = 0$.
Scott-Bereiche dienen dazu Objekte zu approximieren. Deshalb sind wir daran interessiert, daß für bestimmte Mengen das Supremum existiert. Wie üblich wählen wir dafür die gerichteten Mengen.

**Definition 2.12 (Gerichtete Mengen)**

Sei $(X, \sqsubseteq)$ eine reflexive Ordnung. Eine nicht-leere Menge $Y \subseteq X$ heißt *gerichtet*, wenn es für jedes Paar $x, y \in Y$ ein $z \in Y$ mit $x \sqsubseteq z$ und $y \sqsubseteq z$ gibt.

Eine gerichtete Menge enthält für jede endliche und nicht-leere Teilmenge eine obere Schranke.
In einer vollständigen Ordnung existiert für jede gerichtete Teilmenge das Supremum.

**Definition 2.13 (Vollständige Ordnung)**

Eine reflexive Ordnung $(X, \sqsubseteq)$ heißt *vollständige Ordnung*[1], wenn sie ein kleinstes Element besitzt und für jede gerichtete Teilmenge von $X$ das Supremum existiert.

Ein Scott-Bereich ist eine vollständige Ordnung, in der unendliche Elemente durch „endliche“ approximiert werden können. Wir definieren nun die „endlichen “ Elemente der Ordnung. Da diese Charakterisierung nur von der Ordnungsrelation abhängt, nennen wir diese Elemente *kompakt*. Wir werden aber später sehen, daß die bzgl. der Präfixrelation kompakten Abläufe genau den endlichen Abläufen entsprechen.

**Definition 2.14 (Kompakte Elemente)**

Sei $(X, \sqsubseteq)$ eine vollständige Ordnung. Ein Element $x \in X$ heißt *kompakt*, wenn jede gerichtete Teilmenge $G \subseteq X$ mit $x \sqsubseteq \bigsqcup G$ ein Element $g \in G$ mit $x \sqsubseteq g$ besitzt.

Wenn sich jedes Element durch seine kompakten Vorgänger approximieren läßt, dann heißt die Ordnung *algebraisch*.

**Definition 2.15 (Algebraische Ordnung)**

Eine vollständige Ordnung $(X, \sqsubseteq)$ heißt *algebraisch*, wenn für jedes Element $x \in X$ die Menge $Y_x \mathrel{\widehat{=}} \{y \sqsubseteq x \mid y \text{ ist kompakt}\}$ gerichtet ist und gilt $x = \bigsqcup Y_x$.

Neben den zuvor beschriebenen Eigenschaften verlangt die Definition des Scott-Bereiches, daß die Menge der kompakten Elemente abzählbar ist und daß für jede kompatible Menge ein Supremum existiert.

[1] Da wir unter einer Ordnung meist eine Halbordnung verstehen, benutzen wir nicht den Begriff *vollständige Halbordnung*, der der englischen Bezeichnung *complete partial order* (kurz: cpo) am besten entspricht.

**Definition 2.16 (Scott-Bereich)**

Eine vollständige Ordnung $(X, \sqsubseteq)$ heißt *Scott-Bereich*, wenn

1. die Menge ihrer kompakten Elemente abzählbar ist,
2. $(X, \sqsubseteq)$ algebraisch ist und
3. für jede kompatible Menge das Supremum existiert.

## 2.4 Netze und Kausalnetze

Wir werden nun *Petrinetze* einführen, die uns als Grundlage zur Definition des Systembegriffs dienen. Eine spezielle Teilklasse, die *Prozeßnetze*, benutzen wir zur Formalisierung des Ablaufbegriffs.

### 2.4.1 Netze

Beispiele für Netze haben wir bereits in der Einleitung kennengelernt. Wir definieren sie nun formal.

**Definition 2.17 (Netz)**

Ein geordnetes Tripel $N = (S, T; F)$ ist ein *Netz*, wenn $S$ und $T$ disjunkte Mengen sind und $F$ eine Relation $F \subseteq (S \times T) \cup (T \times S)$ ist.

Die Elemente von $S$ nennen wir *Stellen*, die Elemente von $T$ *Transitionen* und $F$ nennen wir die *Flußrelation* von $N$. Graphisch stellen wir Netze in der üblichen Weise dar (vgl. Abb. 2.4): Stellen werden durch Kreise, Transitionen durch Quadrate und die Flußrelation durch Pfeile zwischen Stellen und Transitionen notiert.
Wie üblich definieren wir den Vor- und Nachbereich von Stellen bzw. Transitionen des Netzes:

**Notation 2.18 (Punktnotation für Netze)**

Sei $N = (S, T; F)$ ein Netz und $x \in S \cup T$.

Wir definieren ${}^\bullet x \mathrel{\widehat{=}} \{y \mid (y, x) \in F\}$ und $x^\bullet \mathrel{\widehat{=}} \{y \mid (x, y) \in F\}$. ${}^\bullet x$ nennen wir den *Vorbereich* von $x$, $x^\bullet$ nennen wir den *Nachbereich* von $x$.

Für eine Teilmenge $Y \subseteq S \cup T$ definieren wir ${}^\bullet Y \mathrel{\widehat{=}} \bigcup_{x \in Y} {}^\bullet x$ und $Y^\bullet \mathrel{\widehat{=}} \bigcup_{x \in Y} x^\bullet$.

Den *Anfang* von $N$ definieren wir durch ${}^\circ N \mathrel{\widehat{=}} \{x \in S \cup T \mid {}^\bullet x = \emptyset\}$.
Das *Ende* von $N$ definieren wir durch $N^\circ \mathrel{\widehat{=}} \{x \in S \cup T \mid x^\bullet = \emptyset\}$.

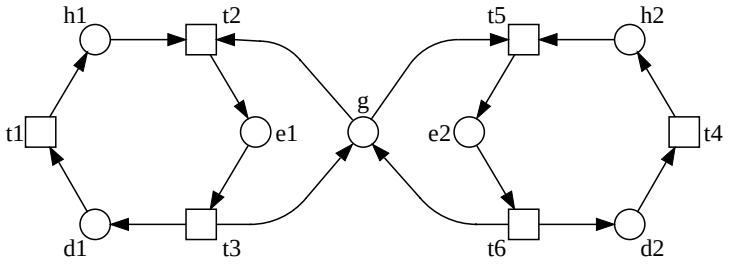



Abbildung 2.4: Ein Netz

Bestimmte Netze werden wir von unseren weiteren Betrachtungen ausschließen. Dazu geben wir nun einige strukturelle Eigenschaften von Netzen an.

**Definition 2.19 (Strukturelle Netzeigenschaften)**

Ein Netz $N = (S, T; F)$ heißt

1. *endlich*, wenn $S$ und $T$ endlich sind.
2. *abzählbar*, wenn $S \cup T$ eine (höchstens) abzählbare Menge ist.
3. *endlich-transitions-verzweigt*, wenn für jede Transition $t \in T$ die Mengen ${}^\bullet t$ und $t^\bullet$ endlich sind.
4. *stellen-unverzweigt*, wenn für jede Stelle $s \in S$ gilt $|{}^\bullet s| \leq 1$ und $|s^\bullet| \leq 1$.
5. *stellenberandet*, wenn gilt ${}^\circ N \cup N^\circ \subseteq S$.
6. *transitions-schlicht*, wenn für alle $t, t' \in T$ aus ${}^\bullet t = {}^\bullet t'$ und $t^\bullet = t'^\bullet$ folgt $t = t'$.

Da wir die Schlichtheit nur für Transitionen benötigen, reden wir im weiteren kurz von *schlichten* Netzen, wenn wir transitions-schlichte Netze meinen. Für schlichte Netze ist jede Transition $t$ eindeutig durch ihren Vor- und Nachbereich bestimmt. Wir können eine Transition eines schlichten Netzes also als Paar $(S_1, S_2)$ darstellen, wobei $S_1$ und $S_2$ Teilmengen der Stellen des Netzes sind. Insbesondere ist ein schlichtes Netz genau dann endlich, wenn $S$ endlich ist.
Für schlichte Netze führen wir eine besondere Schreibweise ein, da sich so einige der folgenden Definitionen einfacher notieren lassen.

**Notation 2.20 (Schreibweise für schlichte Netze)**

Sei $S$ eine beliebige Menge und $T \subseteq 2^S \times 2^S$. Dann steht $(S;T)$ für das Netz $N = (S,T;F)$ wobei $F \mathrel{\widehat{=}} \{(s,(S_1,S_2)) \in S \times T \mid s \in S_1\} \cup \{((S_1,S_2),s) \in T \times S \mid s \in S_2\}$. Das Netz $N$ ist per Definition transitions-schlicht. Für $t = (S_1,S_2) \in T$ gilt insbesondere ${}^\bullet t = S_1$ und $t^\bullet = S_2$.

Im weiteren betrachten wir nur noch endlich-transitions-verzweigte, transitions-schlichte und stellenberandete Netze[2]. Wir können also — je nach Bedarf — zwischen der üblichen Schreibweise $N = (S,T;F)$ und der Schreibweise für schlichte Netze wechseln $(S;T)$.

### 2.4.2 Markierung und Erreichbarkeit

Der Zustand eines Netzes ist eine Multimenge über den Stellen des Netzes. Wir nennen diese Multimenge *Markierung* des Netzes. Ein Netz beschreibt, welche Übergänge zwischen Markierungen möglich sind. Bei der Definition dieser Übergänge machen wir Gebrauch davon, daß die Mengen ${}^\bullet t$ und $t^\bullet$ insbesondere Multimengen über $S$ sind.

**Definition 2.21 (Markierung, Erreichbarkeit)**

Sei $N = (S,T;F)$ ein Netz. Eine Multimenge $M : S \to \mathbb{N}$ nennen wir *Markierung* von $N$.

Eine Transition $t \in T$ heißt *von $M$ aktiviert*, wenn gilt ${}^\bullet t \leq M$. Eine von $M$ aktivierte Transition $t$ kann schalten. Das *Schalten von $t$ in $M$* führt zu der Markierung $M' \mathrel{\widehat{=}} (M - {}^\bullet t) + t^\bullet$. Wir schreiben dafür $M \xrightarrow{t} M'$. Eine Markierung $M'$ heißt *Nachfolgemarkierung* von $M$, wenn es eine Transition $t \in T$ gibt, für die gilt $M \xrightarrow{t} M'$. Wir schreiben dafür $M \longrightarrow M'$.

Die transitive und reflexive Hülle der Relation $\longrightarrow$ notieren wir durch $\xrightarrow{*}$. Eine Markierung $M'$ heißt *von $M$ erreichbar*, wenn gilt $M \xrightarrow{*} M'$.

Die Markierung eines Netzes stellen wir graphisch durch eine entsprechende Anzahl von schwarzen Punkten (Marken) in den Stellen dar. Da wir später als Anfangsmarkierung nur Mengen zulassen, wird eine Anfangsmarkierung immer durch einen oder keinen Punkt in einer Stelle dargestellt (vgl. Kapitel 1).

[2] Später werden wir darüber hinaus nur noch abzählbare Netze benutzen.

### 2.4.3 Kausalnetze und Prozeßnetze

Wir führen nun Begriffe ein, die wir benötigen, um die verteilten Abläufe eines Systems zu formalisieren. Ein Netz, dessen Flußrelation eine Ordnung repräsentiert und das *stellen-unverzweigt* ist, nennen wir Kausalnetz.

**Definition 2.22 (Kausalnetz)**

Ein Netz $N = (S, T; F)$ heißt *Kausalnetz*, wenn

1. $N$ stellen-unverzweigt ist und
2. $F^+$ eine strikte Ordnung auf $S \cup T$ ist.

Da wir später ein Netz mit einem Kausalnetz in Beziehung setzen, führen wir geeignete Sprechweisen zur Unterscheidung der Elemente dieser beiden Netze ein.

**Notation 2.23 (Schreibweisen für Kausalnetze)**

Stellen von Kausalnetzen nennen wir *Bedingungen* und Transitionen von Kausalnetzen nennen wir *Ereignisse*.

Da die Flußrelation $F$ eines Kausalnetzes genau der unmittelbaren Nachfolgerrelation bezüglich $F^+$ entspricht, schreiben wir meist $\lessdot$ für $F$ und $<$ für $F^+$.

Ein Kausalnetz $K$ notieren wir dann durch $K = (B, E; \lessdot)$. Mit $(K, <)$ bezeichnen wir die Ordnung $(B \cup E, <)$.

Stellenberandete Kausalnetze sind immer schlicht. Deshalb benutzen wir für sie auch die Darstellung $K = (B; E)$ aus Notation 2.20.

Da ein Kausalnetz eine spezielle Ordnung ist, überträgt sich auch der Begriff des Schnitts und der Linie auf das Kausalnetz. Die Notationen ${}^\bullet x$, $x^\bullet$, ${}^\circ K$ und $K^\circ$ für das Kausalnetz $K$ sind mit den entsprechenden Notationen für die Ordnung $(K, <)$ identisch, so daß hier keine Zweideutigkeiten auftreten.
Abbildung 2.5 zeigt ein Beispiel für ein unendliches Kausalnetz. Dieses Beispiel zeigt, daß es Kausalnetze gibt, die intuitiv keinem Ablauf entsprechen: Bevor ein Ereignis aus der unteren Reihe auftreten kann müssen alle Ereignisse der oberen Reihe auftreten (also unendlich viele). Ein Kausalnetz, das wir als Ablauf interpretieren können, nennen wir *Prozeßnetz*. Wir schließen aus, daß ein Ereignis unendlich viele Vorgänger besitzt (also insbesondere den Ablauf aus Abb. 2.5). Außerdem lassen wir nur stellenberandete und endlich-verzweigte Netze zu, deren Anfang ${}^\circ K$ abzählbar ist.

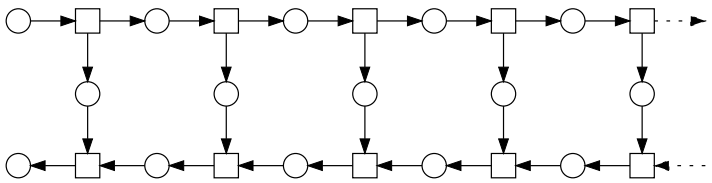


Abbildung 2.5: Ein Kausalnetz

**Definition 2.24 (Prozeßnetz)**

Ein Kausalnetz $K = (B, E; <)$ heißt *Prozeßnetz*, wenn gilt:

1. $K$ ist ein stellenberandet.
2. ${}^\circ K$ ist abzählbar.
3. $(K, <)$ ist endlich-verzweigt.
4. $(K, <)$ ist vorgänger-endlich.

Aus dieser Definition folgt, daß jedes Prozeßnetz abzählbar ist.

**Satz 2.25 (Abzählbarkeit von Prozeßnetzen)**

Jedes Prozeßnetz ist abzählbar.

**Beweis:** Sei $K = (B, E; <)$ ein beliebiges Kausalnetz. Gemäß Definition ist $(K, <)$ vorgänger-endlich und endlich-verzweigt. Mit Bem.2.7 ist $(K, <)$ also initialisiert und kombinatorisch. Weil ${}^\circ K$ abzählbar ist folgt mit Satz 2.9 die Abzählbarkeit von $K$. □

Bei der Definition der Gültigkeit von temporallogischen Aussagen in Abläufen benutzen wir die Erreichbarkeit einer **co**-Menge von einer anderen. Wir werden dort nur **co**-Mengen betrachten, die ausschließlich Bedingungen enthalten, weil diese Mengen den Zuständen entsprechen. Insbesondere entsprechen Schnitte, die nur Bedingungen enthalten, den globalen Zuständen. Wir nennen diese Mengen *Scheiben*.

**Definition 2.26 (Scheibe)**

Sei $K = (B, E; <)$ ein Prozeßnetz. Eine Schnitt $C$ von $(K, <)$ mit $C \subseteq B$ heißt *Scheibe* von $K$.

Eine Teilmenge der Bedingungen eines Prozeßnetzes ist insbesondere eine Markierung des Prozeßnetzes. Deshalb sind zwischen Bedingungsmengen eines Prozeßnetzes insbesondere die Relationen $\longrightarrow$ und $\stackrel{*}{\longrightarrow}$ definiert. Von einer **co**-Bedingungsmenge sind in einem Prozeßnetz nur **co**-Bedingungsmengen erreichbar. Darüber hinaus werden von Scheiben nur Scheiben erreicht.

**Satz 2.27 (Erreichbarkeit von co-Mengen und Scheiben)**

Sei $K = (B, E; <)$ ein Prozeßnetz und $Q \subseteq B$ eine **co**-Menge von $(K, <)$ und $Q'$ eine von $Q$ erreichbare Markierung. Dann ist $Q'$ eine **co**-Menge von $(K, <)$. Darüber hinaus ist $Q'$ genau dann eine Scheibe von $K$, wenn $Q$ eine Scheibe von $K$ ist.

**Beweis:** Da $\stackrel{*}{\longrightarrow}$ die transitiv-reflexive Hülle von $\longrightarrow$ ist, reicht es die Behauptungen für $Q \longrightarrow Q'$ zu zeigen.

1. Sei $Q \subseteq B$ eine **co**-Menge und $e \subseteq E$ ein von $Q$ aktiviertes Ereignis. Weil $Q$ und ${}^\bullet e$ Mengen (keine echten Multimengen) sind, gilt ${}^\bullet e \subseteq Q$. Offensichtlich ist $Q \setminus {}^\bullet e$ als Teilmenge von $Q$ eine **co**-Menge. Außerdem gilt $e^\bullet \cap Q = \emptyset$, denn sonst gäbe es mit $b \in {}^\bullet e$ und $b' \in Q \cap e^\bullet$ zwei Elemente von $Q$, die durch $<$ geordnet sind. Also ist $(Q - {}^\bullet e) + e^\bullet$ eine Menge und es gilt $(Q - {}^\bullet e) + e^\bullet = (Q \setminus {}^\bullet e) \cup e^\bullet$.

   Es bleibt zu zeigen, daß $(Q \setminus {}^\bullet e) \cup e^\bullet$ eine **co**-Menge ist. Angenommen sie ist keine **co**-Menge. Dann gibt es ein $b \in Q \setminus {}^\bullet e$ und ein $b' \in e^\bullet$ mit $b$ **li** $b'$. Wenn $b' < b$ gilt, dann ist $Q$ keine **co**-Menge, da eine Bedingung $b'' \in {}^\bullet e \subseteq Q$ existiert, die vor $b'$ und somit vor $b \in Q$ liegt. Sei also $b < b'$. Gemäß Definition des Kausalnetzes ist $e$ einziger unmittelbarer Vorgänger von $b'$ und ${}^\bullet e$ sind die einzigen unmittelbaren Vorgänger von $e$. Deshalb gilt mit $b < b'$ entweder $b \in {}^\bullet e$ oder es gibt ein $b'' \in {}^\bullet e$ mit $b < b''$. Der Fall $b \in {}^\bullet e$ ist durch die Wahl von $b \in Q \setminus {}^\bullet e$ ausgeschlossen. Für $b'' \in {}^\bullet e$ mit $b < b''$ ist $Q$ keine **co**-Menge, da gilt $b, b'' \in Q$.

2. Seien $Q \stackrel{e}{\longrightarrow} Q'$ zwei **co**-Bedingungsmengen.

   Wir zeigen zunächst: Wenn $Q'$ keine Scheibe ist, dann ist auch $Q$ keine Scheibe. Weil $Q'$ keine Scheibe ist, gibt es ein $x \notin Q'$, so daß $Q' \cup \{x\}$ eine **co**-Menge ist. Insbesondere gilt $x \notin {}^\bullet e$. Denn sonst gibt es wegen $e^\bullet \neq \emptyset$ ein $b \in e^\bullet \subseteq Q'$ mit $x < b$. Wir zeigen nun, daß $x \notin Q$ gilt und $Q \cup \{x\}$ eine **co**-Menge ist. Damit ist $Q$ keine Scheibe. Wegen $x \notin {}^\bullet e$ und $Q' = Q \setminus {}^\bullet e \cup e^\bullet$ und

$x \notin Q'$ gilt $x \notin Q$. Es bleibt zu zeigen, daß für jedes $b \in Q$ gilt $x$ **co** $b$. Für $b \in Q \setminus {}^\bullet e$ gilt $b \in Q'$ und somit nach Voraussetzung $x$ **co** $b$. Für $x \in {}^\bullet e$ gilt ebenfalls $x$ **co** $b$: Angenommen $b < x$. Dann gilt $x = e$ oder für ein $b' \in e^\bullet$ gilt $b' < x$ und damit wäre $Q' \cup \{x\}$ keine **co**-Menge. Angenommen $x < b$. Dann gilt für $b' \in e^\bullet$ auch $x < b'$ und damit ist $Q' \cup \{x\}$ keine **co**-Menge.
Also ist mit $Q'$ auch $Q$ keine Scheibe.

Die umgekehrte Richtung (d.h. wenn $Q$ keine Scheibe ist, dann ist auch $Q'$ keine Scheibe) geht analog.

□

Eine besondere Rolle spielen später die Scheiben, die von der initialen Scheibe aus erreichbar sind, und deren Teilmengen. Wir nennen diese Mengen *erreichbare Scheiben* bzw. *erreichbare* **co***-Mengen*. Satz 2.27 gewährleistet, daß diese Mengen tatsächlich Scheiben bzw. **co**-Mengen sind.

**Definition 2.28 (Erreichbare Scheiben und co-Mengen)**

Sei $K = (B, E; <)$ ein Prozeßnetz.

Eine Menge $Q \subseteq B$ heißt *erreichbare Scheibe* von $K$, wenn gilt ${}^\circ K \xrightarrow{*} Q$.

Eine Menge $Q \subseteq B$ heißt *erreichbar*, wenn es eine erreichbare Scheibe $Q'$ von $K$ gibt, so daß $Q \subseteq Q'$.

### 2.4.4 Netzhomomorphismen

Für eine präzise Definition des Ablaufs eines Netzes müssen wir eine Beziehung zwischen Netzen und Kausalnetzen herstellen. Dazu definieren wir *Netzhomomorphismen* als „strukturerhaltende“ Abbildungen zwischen Netzen. Da wir nur noch transitions-schlichte Netze betrachten, benutzen wir für diese Definition die Darstellung für schlichte Netze.
Ein Netzhomomorphismus von einem Netz $N$ in ein Netz $N'$ ist eine Abbildung $f$ von den Stellen von $N$ in die Stellen von $N'$, so daß das Bild $f(t)$ einer Transition von $N$ eine Transition von $N'$ ist. Dabei überträgt sich gemäß unserer Konvention die Abbildung $f$ wie folgt auf die Transitionen von $N$: Das Bild einer Transition $t = (S_1, S_2)$ ist definiert durch $f(t) = (f(S_1), f(S_2))$.

**Definition 2.29 (Netzhomomorphismus)**

Seien $N = (S;T)$ und $N' = (S';T')$ zwei Netze. Eine Abbildung $f : S \to S'$ heißt *Netzhomomorphismus von $N$ nach $N'$*, wenn für jede Transition $t \in T$ gilt $f(t) \in T'$. Ein Netzhomomorphismus $f$ heißt

1. *Injektion*, wenn $f$ injektiv ist.
2. *Präfixinjektion*, wenn $f$ injektiv ist und zusätzlich gilt $f({}^\circ N) \subseteq {}^\circ N'$.
3. *Netzisomorphismus*, wenn $f$ bijektiv ist und außerdem gilt $f(T) = T'$.

Der Begriff des Netzisomorphismus stimmt mit dem üblichen Isomorphismusbegriff auf Netzen überein. Die Bedingung $f(T) = T'$ müssen wir nur deshalb explizit fordern, weil $f$ zunächst nur auf den Stellen definiert ist.

Die Präfixinjektion benutzen wir später zur Definition der Präfixbeziehung zwischen Abläufen. Zum Beweis von Eigenschaften der Präfixrelation benötigen wir Beweistechniken für Präfixinjektionen. Diese Techniken stellen wir hier zusammen. Sie spielen ausschließlich beim Beweis von Eigenschaften der Präfixrelation eine Rolle und können deshalb beim schnellen Lesen übersprungen werden.

**Lemma 2.30 (Eigenschaften der Präfixinjektion)**

Seien $N = (S;T)$ und $N' = (S';T')$ zwei Netze und $f : S \to S'$ eine Präfixinjektion von $N$ nach $N'$.

1. Wenn für eine Stelle $s \in S$ gilt $f(s) \in {}^\circ N'$, dann gilt auch $s \in {}^\circ N$.
2. Sei $N'$ stellen-unverzweigt und seien $t \in T$, $s \in S$ und $t' \in T'$. Aus $s \in {}^\bullet t$ und $f(s) \in {}^\bullet t'$ folgt $f(t) = t'$. Analog folgt aus $s \in t^\bullet$ und $f(s) \in t'^\bullet$ auch $f(t) = t'$.
3. Sei $N'$ stellen-unverzweigt und $s \in S$. Wenn in $N'$ von $f(s)$ eine unendlich absteigende Kette von Stellen ausgeht, dann geht in $N$ von $s$ eine unendlich absteigende Kette von Stellen aus.
4. Sei $N'$ stellen-unverzweigt und $s \in S$. Wenn $f(s)$ auf einem Zyklus von $N'$ liegt, dann liegt $s$ auf einem Zyklus von $N$.

**Beweis:**

1. Wir zeigen die Kontraposition: Sei $s \notin {}^\circ N$. Dann gibt es eine Transition $t \in T$ mit $s \in t^\bullet$. Mit der Homomorphiebedingung folgt $f(s) \in f(t)^\bullet$. Also gilt $f(s) \notin {}^\circ N'$.
2. Sei $s \in {}^\bullet t$ und $f(s) \in {}^\bullet t'$. Wegen der Homomorphiebedingung ist $f(t)$ Transition von $N'$ und es gilt $f(s) \in {}^\bullet f(t)$. Also gilt ${}^\bullet f(t) \cap {}^\bullet t' \neq \emptyset$. Wegen der Unverzweigtheit der Stellen von $N'$ folgt $f(t) = t'$.
3. Sei $s'_0 \mathrel{\widehat{=}} f(s), s'_1, s'_2, \ldots$ eine unendlich absteigende Kette in $N'$ (d.h. $s'_{i+1} \in {}^\bullet({}^\bullet s_i)$). Wir konstruieren induktiv eine unendlich absteigende Kette $s_0, s_1, s_2, \ldots$ von $N$ mit $f(s_i) = s'_i$:
   (a) $s_0 = s$.
   (b) Da $f(s_i) = s'_i \notin {}^\circ N'$ existiert nach 1. ein $t \in T$ mit $s_i \in t^\bullet$. Da $N'$ stellen-unverzweigt ist, ist $f(t)$ die einzige Transition im Vorbereich von $s'_i$. Also gilt $s'_{i+1} \in {}^\bullet f(t)$. Wegen der Homomorphiebedingung existiert ein $s_{i+1} \in {}^\bullet t$ mit $f(s_{i+1}) = s'_{i+1}$.

   Nach Konstruktion ist $s_0, s_1, \ldots$ eine unendlich absteigende Kette von $N$, die von $s = s_0$ ausgeht.
4. Sei $s'_0 \mathrel{\widehat{=}} f(s), s'_1, s'_2, \ldots, s'_n = s'_0$ ein Zyklus in $N'$. Dann existiert in $N'$ insbesondere eine von $s'_0$ ausgehende unendlich absteigende Kette. Nach 3. gibt es in $N$ eine unendlich absteigende Kette $s_0 = s, s_1, s_2, \ldots, s_n, \ldots$ mit $f(s_i) = s'_i$. Da $f$ injektiv ist gilt $s_n = s_0$ und somit enthält auch $N$ einen Zyklus.

□

Mit Hilfe dieser Eigenschaften können wir zeigen, daß ein stellenunverzweigtes Netz $N = (S; T)$, in dem jede Stelle und jede Transition das Bild einer geeigneten Präfixinjektion von einem Prozeßnetz $N'$ nach $N$ ist, alle Eigenschaften eines Prozeßnetzes besitzt. Nur die Abzählbarkeit der initialen Scheibe ${}^\circ N$ müssen wir explizit fordern.

**Satz 2.31 (Prozeßnetzeigenschaft)**

Sei $N = (S; T)$ ein stellen-unverzweigtes Netz mit abzählbarem Anfang ${}^\circ N$ und $\mathcal{K}$ eine Menge von Prozeßnetzen, und für jedes Netz $K = (B; E) \in \mathcal{K}$ sei $f_K : B \to S$ eine Präfixinjektion.

Wenn gilt $S = \bigcup_{K \in \mathcal{K}} f_K(B)$ und $T = \bigcup_{K \in \mathcal{K}} f_K(E)$. Dann ist $N$ ein Prozeßnetz.

**Beweis:** Zunächst zeigen wir, daß $N$ ein Kausalnetz ist. Nach Voraussetzung ist $N$ stellen-unverzweigt. Wir zeigen nun durch Widerspruch, daß $N$ keine Zyklen besitzt. Angenommen $N$ enthält einen Zyklus und $s \in S$ liegt auf diesem Zyklus. Nach Voraussetzung gibt es ein Prozeßnetz $K = (B; E) \in \mathcal{K}$ und ein $b \in B$ mit $f_K(b) = s$. Dann enthält nach Lemma 2.30 (4) auch $K$ einen Zyklus. Dies ist ein Widerspruch zur Annahme, daß $K$ ein Prozeßnetz ist.

Wir zeigen nun, daß $N$ ein Prozeßnetz ist. Nach Voraussetzung ist jede Transition $t \in T$ ein Bild eines Ereignisses eines Prozeßnetzes aus $\mathcal{K}$. Wegen der Homomorphiebedingung ist $N$ stellenberandet und endlich-verzweigt.

Die Abzählbarkeit von ${}^\circ N$ haben wir explizit in der Voraussetzung gefordert.

Da jede Stelle $s \in S$ Bild einer Bedingung eines Prozeßnetzes $K \in \mathcal{K}$ ist, geht mit Lemma 2.30 (3) von $s$ keine unendlich absteigende Kette aus (d.h. $N$ ist fundiert). Aus Bemerkung 2.7 folgt zusammen mit der endlichen Verzweigtheit von $N$, daß $N$ vorgänger-endlich ist. □

Für ein Prozeßnetz $K$ können wir die Netze, die eine Präfixinjektion nach $K$ besitzen, explizit konstruieren. Wir lassen dazu bestimmte Bedingungen und Ereignisse von $K$ weg. Allerdings müssen wir mit einem Ereignis alle seine Nachfolger weglassen. Außerdem müssen wir mit einer Bedingung auch ihre Ereignisse im Vorbereich und Nachbereich weglassen. Wir definieren diese *Ausschnitte* mit Hilfe der *vollständigen (Stellen-)Mengen* eines Netzes.

**Definition 2.32 (Vollständige Mengen, Ausschnitt)**

Sei $N = (S; T)$ ein Netz. Eine Menge $Q \subseteq S$ heißt *vollständig* (in $N$), wenn $({}^\bullet Q)^\bullet \subseteq Q$ und ${}^\bullet({}^\bullet Q) \subseteq Q$ gilt.

Für eine vollständige Menge $Q$ definieren wir das Netz $N|_Q \mathrel{\widehat{=}} (Q; T|_Q)$, wobei gilt $T|_Q \mathrel{\widehat{=}} T \cap (2^Q \times 2^Q)$. $N|_Q$ heißt *Ausschnitt* von $N$.

Für ein Netz $N = (S; T)$ und eine vollständige Menge $Q$ von $N$ ist die Abbildung $f : Q \to S$ (mit $f(b) = b$ für $b \in Q$) eine Präfixinjektion von $N|_Q$ nach $N$. Die vollständigen Mengen von $N$ sind unter Durchschnitt und Vereinigung abgeschlossen. Sie bilden also einen vollständigen Verband mit kleinstem Element $\emptyset$ und größtem Element $S$. Insbesondere gilt $N|_S = N$.

Mit Hilfe der vollständigen Mengen können wir später Infima von Ablaufmengen konstruieren. Die Techniken dazu sind in folgendem Satz zusammengefaßt.

**Satz 2.33 (Umkehrung der Präfixinjektion)**

Seien $K = (B; E)$ und $K' = (B'; E')$ zwei stellenberandete Kausalnetze und $f : B \to B'$ eine Präfixinjektion von $K$ nach $K'$.

1. $f(B)$ ist eine vollständige Menge von $K'$.
2. Für eine vollständige Menge $Q \subseteq f(B)$ von $K'$ ist $f^{-1}|_Q : Q \to B$ eine Präfixinjektion von $K'|_Q$ nach $K$.
3. Insbesondere ist $f^{-1}|_{f(B)} : f(B) \to B$ ein Isomorphismus von $K'|_{f(B)}$ nach $K$.
4. Für eine vollständige Menge $Q \supseteq f(B)$ ist $f : B \to Q$ eine Präfixinjektion von $K$ nach $K'|_Q$.

**Beweis:**

1. Sei $b \in B$. Es ist zu zeigen, daß für jedes $e' \in {}^\bullet f(b)$ gilt ${}^\bullet e' \subseteq f(B)$ und $e^\bullet \subseteq f(B)$. Mit $f(b) \notin {}^\circ K'$ gilt (wg. Lemma 2.30 1) auch $b \notin {}^\circ K$. Also ex. ein $e \in E$ mit $e \in {}^\bullet b$. Dann gilt mit der Homomorphiebedingung $f(b) \in f(e)^\bullet$. Da Bedingungen unverzweigt sind folgt damit $f(e) = e'$. Damit gilt ${}^\bullet e' = f({}^\bullet e) \subseteq f(B)$ und $e'^\bullet = f(e^\bullet) \subseteq f(B)$.
2. Wegen $Q \subseteq f(B)$ und der Injektivität von $f$ ist $f^{-1}|_Q : Q \to B$ eine injektive Abbildung.

   Wir zeigen zunächst, daß $f^{-1}|_Q$ ein Homomorphismus ist. Sei $e' \in E'|_Q$. Dann gilt definitionsgemäß ${}^\bullet e' \subseteq Q$ und $e'^\bullet \subseteq Q$. Sei $b' \in e'^\bullet$ und $b \in B$ mit $f(b) = b'$. Dann gilt mit $b' \notin {}^\circ K'$ auch $b \notin {}^\circ K$. Also gibt es ein $e \in {}^\bullet b$ und $b' \in f(e)^\bullet$. Da $K'$ stellenunverzweigt ist, gilt $f(e) = e'$ und umgekehrt $f^{-1}(e') = e$.

   Wir zeigen nun durch Widerspruch, daß $f^{-1}|_Q$ eine Präfixinjektion ist. Sei $b' \in {}^\circ K'|_Q$ und es gelte $b \mathrel{\widehat{=}} f^{-1}|_Q(b') \notin {}^\circ K$. Dann gibt es ein $e \in {}^\bullet b$. Da $f(e) \notin E|_Q$ gibt es ein $b'' \in {}^\bullet f(e) \cup f(e)^\bullet$ mit $b'' \notin Q$. Dann ist $Q$ aber nicht vollständig bezüglich $K'$.
3. Aus 1. folgt, daß $f(B)$ vollständig bzgl. $K'$ ist. Somit ist $K'|_{f(B)}$ wohldefiniert. $f^{-1}|_{f(B)} : f(B) \to B$ ist eine bijektive Abbildung.

Es bleibt zu zeigen, daß $E'|_{f(B)} = f(E)$. Da $K'$ stellenberandet ist, besitzt jedes Ereignis $e' \in E'|_{f(B)}$ eine Stelle $b' \in f(B)$ in ihrem Nachbereich $b' \in e'^{\bullet}$. Es gibt also ein $b \in B$ und $e \in E$ mit $f(b) = b'$ und $b \in e^{\bullet}$. Da die Bedingungen von $K'$ unverzweigt sind, folgt $f(e) = e'$.

4. Offensichtlich, da $f(B) \subseteq Q$.

□

Aus diesem Satz ergeben sich unmittelbar zwei weitere Ergebnisse.

**Korollar 2.34**

Sei $K = (B; E)$ ein stellenberandetes Kausalnetz, und $Q \subseteq Q' \subseteq B$ zwei vollständige Mengen bzgl. $K$.

Die Abbildung $f : Q \to Q'$ mit $f(b) = b$ ist eine Präfixinjektion von $K|_Q$ nach $K|_{Q'}$. Insbesondere ist für $Q' = B$ die Abbildung $f$ eine Präfixinjektion von $K|_Q$ nach $K$.

## 2.5 Verteilte Transitionssysteme und ihre Abläufe

Als Systembegriff betrachten wir hier S/T-Systeme. Ein Ablauf eines S/T-Systems ist ein Prozeßnetz, welches mittels eines speziellen Netzhomomorphismus mit dem Netz des S/T-Systems in Beziehung gesetzt wird. Dies entspricht dem Prozeßbegriff aus [32].
Wir benutzen S/T-Systeme als Basisbegriff, auf dem später ein verfeinerter Systembegriff aufbaut. Dies ist analog zum sequentiellen Fall, in dem Transitionssysteme als Basisbegriff definiert werden. Wegen dieser Analogie nennen wir S/T-Systeme in dieser Arbeit *verteilte Transitionssysteme*.
Neben den bereits erwähnten Einschränkungen benutzen wir — im Unterschied zur üblichen Definition von S/T-Systemen — bei der Definition von verteilten Transitionssystemen die Schreibweise für schlichte Netze.

**Definition 2.35 (Verteiltes Transitionssysteme)**

Sei $N = (S; T)$ ein endlich-transitions-verzweigtes, stellenberandetes und abzählbares Netz und sei $M \subseteq S$. Wir nennen $\mathcal{T} = (N, M)$ ein *verteiltes Transitionssystem* mit *Anfangsmarkierung* $M$.

Sei $K = (B; E)$ ein Prozeßnetz und $\rho : B \to S$ ein Netzhomomorphismus von $K$ nach $N$. Das Paar $(K, \rho)$ nennen wir *Ablauf von* $\mathcal{T}$, wenn

1. die Abbildung $\rho|_{\circ K}$ injektiv ist,
2. $\rho(^\circ K) \subseteq M$ gilt und
3. für jedes $e \in E$ die Abbildungen $\rho|_{\bullet e}$ und $\rho|_{e^\bullet}$ injektiv sind.

Die Menge aller Abläufe von $\mathcal{T}$ bezeichnen wir mit $\mathbf{R}(\mathcal{T})$. Wir nennen einen Ablauf $(K, \rho)$ *endlich*, wenn das zugrundeliegende Prozeßnetz endlich ist. Die Menge aller endlichen Abläufe von $\mathcal{T}$ bezeichnen wir mit $\mathbf{R_f}(\mathcal{T})$.

In der Einführung haben wir bereits Beispiele für verteilte Transitionssysteme und ihre Abläufe gesehen. Wir wiederholen hier das Mutex-Beispiel mit einer kleinen Modifikation. Abbildung 2.6 zeigt ein verteiltes Transitionssystem, wobei die Stellen $g$, $d_1$ und $d_2$ initial markiert sind. In den Abb. 2.7 bis 2.9 haben wir drei zugehörige Abläufe

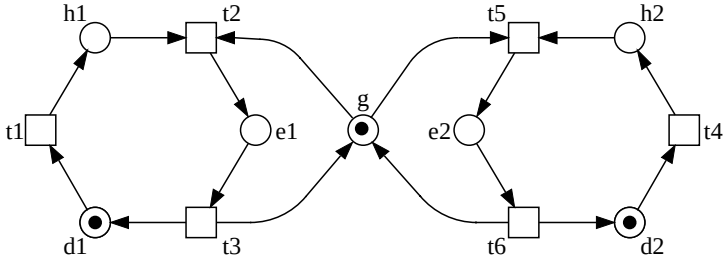



Abbildung 2.6: Ein verteiltes Transitionssystem $\mathcal{T}$

dargestellt. Die Abbildung $\rho$ ist jeweils durch die Beschriftung der Bedingungen angedeutet. Wie der Ablauf aus Abb. 2.7 zeigt, müssen am Anfang eines Ablaufs nicht alle Stellen der Anfangsmarkierung vorkommen. Dieser Ablauf kann aber so fortgesetzt werden (Abb. 2.8 und 2.9), daß alle Stellen der Anfangsmarkierung am Anfang vorkommen. Diese Abläufe können nur noch an ihrem Ende fortgesetzt werden. Formal werden wir den Begriff der Fortsetzung eines Ablaufs erst im nächsten Kapitel mit Hilfe der Präfixinjektion definieren.

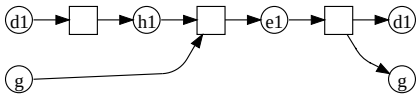



Abbildung 2.7: Ein Ablauf von $\mathcal{T}$

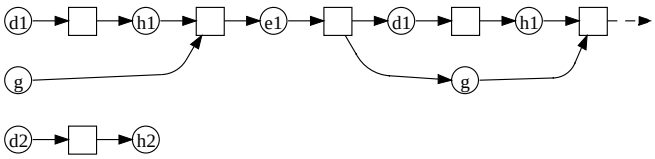



Abbildung 2.8: Eine Fortsetzung des obigen Ablaufs

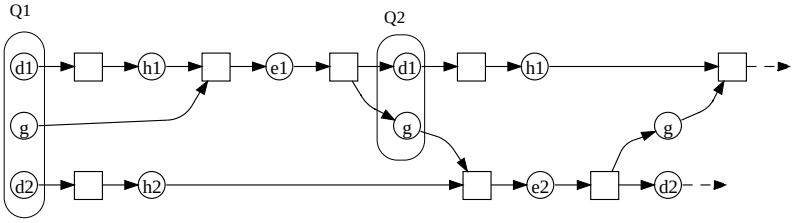



Abbildung 2.9: Ein weiterer Ablauf von $\mathcal{T}$

Zwischen den Scheiben eines Ablaufs $(K, \rho)$ eines Transitionssystems und seinen Markierungen besteht ein enger Zusammenhang. Dazu ordnen wir einer erreichbaren **co**-Menge $Q$ des Ablaufs $\rho$ eine Multimenge $M_{\rho,Q}$ zu, die für jede Stelle $s$ die Anzahl der in $Q$ mit $s$ beschrifteten Bedingungen enthält. Beispielsweise gilt für die im Ablauf $\rho$ aus Abb. 2.9 gekennzeichneten **co**-Mengen $Q_1$ und $Q_2$: $M_{\rho,Q_1} = \{d_1, g, d_2\}$ und $M_{\rho,Q_2} = \{d_1, g\}$. Dabei ist in diesem Ablauf $M_{\rho,Q}$ für jede **co**-Menge $Q$ ein Menge und keine (echte) Multimenge.
Wir zeigen im Anschluß an diese Definition, daß für erreichbare **co**-Mengen die Abbildung $M_{\rho,Q}$ total und damit eine Multimenge ist.

**Definition 2.36 (co-Mengen und Markierungen)**

Sei $\mathcal{T} = (N, M)$ ein Transitionssystem, $(K, \rho)$ ein Ablauf von $\mathcal{T}$ und $Q$ eine **co**-Menge von $K$. Wir definieren die partielle Abbildung $M_{\rho,Q} : S \to \mathbb{N}$ elementweise durch

$$M_{\rho,Q}(s) \mathrel{\widehat{=}} |\, \rho^{-1}(s) \cap Q|$$

falls $|\, \rho^{-1}(s) \cap Q|$ endlich ist.

Für nicht erreichbare **co**-Mengen eines Ablaufs ist $M_{\rho,Q}$ möglicherweise für bestimmte $s$ nicht definiert und ist deshalb keine Multimenge. Für jede erreichbare **co**-Menge $Q$ eines Ablaufes ist $M_{\rho,Q}$ eine Multimenge. Darüber hinaus hängen die Markierungen zweier aufeinanderfolgender **co**-Mengen eng mit dem entsprechenden Zustandsübergang im System zusammen.

**Satz 2.37**

Sei $\mathcal{T} = (N, M)$ ein Transitionssystem, $(K, \rho)$ ein Ablauf von $\mathcal{T}$. Für jede erreichbar **co**-Menge $Q$ von $K$ ist $M_{Q,\rho}$ eine Multimenge (d.h. eine totale Abbildung).

Sei $Q$ eine erreichbare **co**-Menge. Wenn $Q \xrightarrow{e} Q'$ in $K$ gilt, dann gilt $M_{\rho,Q} \xrightarrow{\rho(e)} M_{\rho,Q'}$ in $N$.

**Beweis:**

1. Wir zeigen: Wenn für $Q$ die Abbildung $M_{\rho,Q}$ eine Multimenge ist, dann ist für jede **co**-Menge $Q'$ mit $Q \longrightarrow Q'$ $M_{\rho,Q'}$ eine Multimenge. Dies folgt unmittelbar aus der endlichen Verzweigtheit der Ereignisse. Deshalb ändern sich die Einträge der Multimengen nur um einen endlichen Betrag.

Da $M_{\rho,}{}_{\circ K} \subseteq M$ eine Menge (also insbesondere eine Multimenge) ist, folgt durch Induktion über die Erreichbarkeit, daß für jede erreichbare Scheibe $Q$ die Abbildung $M_{\rho,Q}$ eine Multimenge ist. Da die erreichbaren **co**-Mengen Teilmengen der erreichbaren Scheiben sind, folgt die Aussage.

2. Diese Aussage folgt daraus, daß $\rho$ auf ${}^\bullet e$ und $e^\bullet$ injektiv ist. Damit sind $M_{\rho,{}^\bullet e}$ und $M_{\rho,e^\bullet}$ Mengen und es gilt $M_{\rho,{}^\bullet e} = \rho({}^\bullet e)$ und $M_{\rho,e^\bullet} = \rho(e^\bullet)$. Deshalb gilt $M_{\rho,Q} = M_{\rho,Q\setminus {}^\bullet e} + M_{\rho,{}^\bullet e} = M_{\rho,Q\setminus {}^\bullet e} + \rho({}^\bullet e)$ und $M_{\rho,Q'} = M_{\rho,(Q\setminus {}^\bullet e)\cup e^\bullet} = M_{\rho,(Q\setminus {}^\bullet e)} + M_{\rho,e^\bullet} = M_{\rho,Q\setminus {}^\bullet e} + \rho(e^\bullet)$. Damit ist $\rho(e)$ in $M_{\rho,Q}$ aktiviert und es gilt $M_{\rho,Q} \xrightarrow{\rho(e)} M'$ mit $M' = (M_{\rho,Q} - \rho({}^\bullet e)) + \rho(e^\bullet) = M_{\rho,Q\setminus {}^\bullet e} + \rho(e^\bullet) = M_{\rho,Q'}$.

□

Die Übergänge zwischen **co**-Mengen eines Ablaufs spiegeln also die Markierungsübergänge des verteilten Transitionssystems wider.

Die meisten Spezifikationsmethoden benutzen als zugrundeliegendes Systemmodell ein sequentielle Transitionssystem. Wir wollen später unsere Ergebnisse mit den Ergebnissen des sequentiellen Falls vergleichen und einige Unterschiede diskutieren. Deshalb führen wir den Begriff des sequentiellen Transitionssystems als Spezialfall eines verteilten Transitionssystems ein. Dabei ist es nicht unsere Absicht, diesen Formalismus als Alternative zum üblichen Begriff eines sequentiellen Transitionssystems einzuführen. Wir erhalten aber viele Ergebnisse für den sequentiellen Fall „gratis“ aus Ergebnissen des verteilten Falls, wenn wir sequentielle Systeme als Spezialfall der verteilten auffassen.
Wir nehmen bei dieser Definition zur Vereinfachung an, daß jedes sequentielle Transitionssystem denselben ausgezeichneten Anfangszustand $i$ besitzt.

**Definition 2.38 (Sequentielles Transitionssystem)**

Ein verteiltes Transitionssystem $\mathcal{T} = (N, M)$ heißt *sequentiell*, wenn gilt

1. $M = \{i\}$ und
2. für jede Transition $t \in T$ von $N$ gilt $|{}^\bullet t| = 1 = |t^\bullet|$.

Der Begriff des sequentiellen Transitionssystems kommt in der Literatur in verschiedenen Varianten vor. Eine häufige Variante ist, daß

mehrere alternative Anfangszustände möglich sind. Wir legen für unsere Betrachtungen einen eindeutigen Anfangszustand fest, weil dies einerseits unsere Intuition von einem technischen System besser trifft, welches in einem definierten Zustand anfängt, und wir andererseits durch eine einfache Konstruktion die nichtdeterministische Auswahl des Anfangszustands explizit machen können. Dazu müssen wir für jeden möglichen Anfangszustand $s$ eine Transition $(\{i\}, \{s\})$ zum Netz hinzufügen.
Die Abläufe eines sequentiellen Transitionssystems sind immer sequentiell (d.h. die Ordnung $(K, <)$ des zugrundeliegenden Kausalnetzes ist linear). Für jeden nicht-leeren Ablauf $(K, \rho)$ eines sequentiellen Systems gilt $\rho(^\circ K) = \{i\}$.

Jedes verteilte Transitionssystem besitzt mindestens den leeren Ablauf.

**Notation 2.39 (Leerer Ablauf)**

Wir definieren den *leeren Ablauf* $(K, \varepsilon)$ als leeres Kausalnetz $K = (\emptyset, \emptyset, \emptyset)$ mit Beschriftung $\varepsilon : \emptyset \to S$.

Im leeren Ablauf $\varepsilon$ eines Systems tritt kein Ereignis auf. Intuitiv gesprochen passiert in diesem Ablauf nichts. Generell wird bei dem hier verwendeten Ablaufbegriff nie erzwungen, daß Ereignisse eintreten. Dies entspricht im sequentiellen Fall den Transitionssystemen ohne Fairnessannahme.
Wir werden den Systembegriff später um *Fairnessannahmen* erweitern, so daß wir in ihren Abläufen das Eintreten von bestimmten Ereignissen erzwingen können. Die Kriterien für ein sinnvolles Fairness-Konzepts werden wir aber erst im nachfolgenden Kapitel diskutieren. Dazu betrachten wir Abläufe zunächst losgelöst von Transitionssystemen und formalisieren das Konzept der *Sicherheits-* und *Lebendigkeitseigenschaften*.

## 2.6 Literatur

Die in diesem Kapitel eingeführten Begriffe und Notationen sind im wesentlichen Standard. Wir halten uns bei der Definition der Standardbegriffe aus der Netztheorie im wesentlichen an die Definitionen von Reisig [80]. Bei den Definition der verteilten Abläufe und der „Concurrency-Theorie“ benutzen wir überwiegend die Bezeichnungen und Notationen von Best und Fernández [14, 13].

Wir definieren Abläufe, in ähnlicher Weise wie sie erstmals von Golz und Reisig [32] für S/T-Systeme vorgeschlagen wurden. Allerdings beschränken wir uns auf Systeme, deren Anfangsmarkierungen Mengen sind. Die gleiche Einschränkung wird aus ähnlichen technischen Gründen von Engelfriet [26] getroffen. Für verteilte Abläufe von S/T-Systemen wurde von Best und Devillers [11] eine Äquivalenz vorgeschlagen, in der nebenläufige Bedingungen mit gleicher Beschriftung ausgetauscht werden dürfen[3]. Die eigentliche Semantik ist dann der Quotient der Abläufe bzgl. dieser Äquivalenz (siehe auch [24, 60]). Da wir — wie in der Einleitung angekündigt — nicht wirklich an S/T-Systemen interessiert sind, begnügen wir uns mit dem hier eingeführten einfacheren Ablaufbegriff.
Netzhomomorphismen werden seit langem verwendet (vgl. [80]) um Abläufe (Prozesse) eines Systems zu definieren. Wir benötigen darüber hinaus den Begriff der Präfixinjektion, um im folgenden Kapitel die Präfixrelation zwischen Abläufen zu definieren.

Der Begriff des Scott-Bereiches wird in der Literatur hauptsächlich in der Theorie der Semantischen Bereiche benutzt. Gunter und Scott [35] motivieren die verschiedenen Bedingungen des Scott-Bereiches[4] ausführlicher als wir. Im Zusammenhang mit der Präfixrelation für Abläufe kommen Scott-Bereiche in der Literatur aber auch vor [89, 48].

[3] *Swapping* genannt.
[4] In [35] werden sie *bounded complete domain* genannt.

# Kapitel 3

# Eigenschaften verteilter Systeme

In diesem Kapitel werden wir den Begriff der Eigenschaft eines verteilten Systems näher untersuchen. Dabei verstehen wir unter einer *Eigenschaft* eine Menge von Abläufen. Wenn ein System nur Abläufe einer Eigenschaft besitzt, dann *erfüllt das System diese Eigenschaft.* Diesem Eigenschaftsbegriff liegt die Auffassung zugrunde, daß nur die Abläufe eines Systems von Interesse sind[1].
Bei der Untersuchung des Begriffs der Eigenschaft betrachten wir Abläufe losgelöst von einem Transitionssystem. Deshalb charakterisieren wir zunächst solche *Beschriftungen* eines Prozeßnetzes, die einem Ablauf eines Transitionssystems entsprechen. Dann definieren wir, wann ein Ablauf ein *Anfang* (oder *Präfix*) eines anderen Ablaufs ist. Wir werden zeigen, daß die Menge der Abläufe mit der Präfixrelation einen Scott-Bereich bilden.
Damit lassen sich die — für den sequentiellen Fall klassischen — Begriffe der *Sicherheits-* und *Lebendigkeitseigenschaft* auf verteilte Abläufe übertragen. Dabei besagt eine Sicherheitseigenschaft, daß in einem Ablauf nie etwas „Unerwünschtes“ passiert. Eine Lebendigkeitseigenschaft besagt, daß irgendwann etwas „Erwünschtes“ passiert.
Da die Abläufe mit der Präfixrelation einen Scott-Bereich bilden, gilt auch für verteilte Abläufe das Zerlegungstheorem von Alpern und Schneider [6]: Jede Eigenschaft läßt sich als Durchschnitt einer geeig-

[1] Wir sind hier beispielsweise nicht an typischen Petrinetzeigenschaften, wie z.B. „eine Transition $t$ des Netzes ist lebendig“, interessiert.

neten Sicherheits- und Lebendigkeitseigenschaft darstellen. Die Charakterisierung von Sicherheits- und Lebendigkeitseigenschaften für verteilte Abläufe ist nicht grundsätzlich neu. Wenn die Präfixordnung ein Scott-Bereich ist, können Sicherheits- und Lebendigkeitseigenschaften kanonisch definiert werden, und der Zerlegungssatz gilt. Ein wesentlicher Beitrag dieses Kapitels ist der Nachweis, daß unsere verteilten Abläufe zusammen mit der Präfixrelation tatsächlich einen Scott-Bereich bilden.
Ein weiterer Beitrag dieses Kapitels ist die Beleuchtung des Unterschieds zwischen dem klassischen sequentiellen und dem verteilten Fall. Dieser Unterschied ist deshalb so wichtig, weil die klassischen Entwurfsmethoden (mehr oder weniger unausgesprochen) von einem Ergebnis Gebrauch machen, welches nur für den sequentiellen Fall gilt: Im Gegensatz zum sequentiellen Fall ist im verteilten Fall nicht jede Sicherheitseigenschaft *sackgassenfrei generierbar*. Dies bedeutet, daß eine Sicherheitseigenschaft im verteilten Fall nicht immer unabhängig von einer Lebendigkeitseigenschaft implementiert werden kann.

## 3.1 Verteilte Abläufe mit Präfixrelation

Wir betrachten in diesem Kapitel Abläufe losgelöst von Systemen. Deshalb charakterisieren wir, wann ein Prozeßnetz $K = (B; E)$ zusammen mit einer Abbildung $\rho : B \to S$ ein Ablauf eines geeigneten Systems ist.

**Satz 3.1 (Charakterisierung von Abläufen)**

Sei $K = (B; E)$ ein Prozeßnetz, $S$ eine abzählbare Menge und $\rho : B \to S$ eine Abbildung.

Das Paar $(K, \rho)$ ist genau dann ein Ablauf eines verteilten Transitionssystems, wenn

1. $\rho|_{^\circ K}$ injektiv ist und
2. für jedes $e \in E$ die Abbildungen $\rho|_{^\bullet e}$ und $\rho|_{e^\bullet}$ injektiv sind.

Das Paar $(K, \rho)$ ist genau dann Ablauf eines sequentiellen Transitionssystems, wenn $|^\circ K| \leq 1$ und $\rho(^\circ K) \subseteq \{i\}$ gilt und für jedes Ereignis $e \in E$ gilt $|^\bullet e| = 1 = |e^\bullet|$.

**Beweis:** Wähle $\mathcal{T} = (N, M)$ mit $M = \rho(^\circ K)$ und $N = (S; T)$, so daß $T = \{(\rho(^\bullet e), \rho(e^\bullet)) \mid e \in E\}$. Offensichtlich ist $(K, \rho)$ Ablauf von $\mathcal{T}$.

Darüber hinaus ist $\mathcal{T}$ sequentiell, wenn $|{}^\circ K| \leq 1$, $\rho({}^\circ K) \subseteq \{i\}$ und $|{}^\bullet e| = 1 = |e^\bullet|$ für jedes $e \in E$ gilt. Die Abbildungen $\rho|_{\circ K}$, $\rho|_{\bullet e}$ und $\rho|_{e^\bullet}$ sind offensichtlich injektiv, da die entsprechenden Mengen nur aus einem Element bestehen. □

Für die Menge aller Abläufe sowie für die Spezialfälle der endlichen Abläufe und der Abläufe von sequentiellen Transitionssystemen führen wir die folgenden Bezeichnungen ein.

**Definition 3.2 (Bezeichnungen für Ablaufmengen)**

Einen Ablauf eines verteilten Transitionssystems mit Stellenmenge $S$ nennen wir *Ablauf über Stellenmenge S*. Die Menge aller Abläufe über $S$ bezeichnen wir mit $\mathbf{R}(S)$, die Menge aller endlichen Abläufe über $S$ bezeichnen wir mit $\mathbf{R_f}(S)$. Die Menge der sequentiellen bzw. endlichen sequentiellen Abläufe über $S$ bezeichnen wir mit $\mathbf{R^s}(S)$ bzw. $\mathbf{R^s_f}(S)$.

Wir werden uns im folgenden immer wieder auf Abläufe beziehen. Um eine übersichtlichere Schreibweise zu erreichen, geben wir häufig nur die Beschriftung $\rho$ an, wenn wir einen Ablauf $(K, \rho)$ meinen.

**Notation 3.3 (Konventionen für Abläufe)**

Wenn $K$ und $S$ aus dem Kontext hervorgehen, schreiben wir $\rho$ an Stelle von $(K, \rho)$.

Bei Verwendung eines indizierten oder gestrichenen Symbols zur Bezeichnung des Ablaufs, wie z.B. $\rho_1$ oder $\rho'$, überträgt sich dies auf das entsprechende Prozeßnetz und seine Bedingungs- und Ereignismengen (z.B. steht $\rho_1$ für $(K_1, \rho_1)$, wobei $K_1 = (B_1; E_1)$ das zugehörige Prozeßnetz und die Abbildung $\rho_1 : B_1 \to S$ ist).

Da ein Prozeßnetz meist mit einer abzählbar unendlichen Menge $S$ beschriftet ist, betrachten wir im Rest dieses Kapitels nur eine beliebige aber feste abzählbar unendliche Menge $S$. Wenn wir die Stellenmenge $S$ nicht explizit erwähnen, verstehen wir unter einem Ablauf immer einen Ablauf über $S$. Im Hinblick auf die sequentiellen Abläufe nehmen wir an, daß $i \in S$ gilt.

Wir formalisieren nun, wann ein Ablauf ein *Anfang* oder *Präfix* eines anderen Ablaufs ist. Wir haben bereits in Kapitel 2 ein Beispiel für den Präfix eines Ablaufs gesehen. Der Ablauf aus Abb 2.7 auf Seite 39 ist ein Präfix des Ablaufes aus Abb. 2.8. Diese Beziehung werden wir nun mit Hilfe der Präfixinjektion definieren. Anschließend werden wir zeigen, daß die Präfixrelation eine reflexive Ordnung auf der Menge der Abläufe ist.

**Definition 3.4 (Ablaufhomomorphismus, Präfixrelation)**

Seien $(K, \rho)$ und $(K', \rho')$ zwei Abläufe über $S$. Ein Netzhomomorphismus $f : K \to K'$ heißt Ablaufhomomorphismus, wenn für jedes $b \in B$ gilt $\rho(b) = \rho'(f(b))$.

Ein Ablaufhomomorphismus $f$ von $\rho$ nach $\rho'$ heißt *Ablaufisomorphismus*, wenn $f$ ein Isomorphismus auf den zugrundeliegenden Prozeßnetzen ist. Wenn ein Ablaufisomorphismus zwischen $\rho$ und $\rho'$ existiert, dann heißen $\rho$ und $\rho'$ *isomorph* und wir schreiben $\rho \equiv \rho'$.

Ein Ablaufhomomorphismus $f$ von $\rho$ nach $\rho'$ heißt *Präfixhomomorphismus*, wenn $f$ eine Präfixinjektion zwischen den zugrundeliegenden Kausalnetzen $K$ und $K'$ ist. Wenn ein Präfixhomomorphismus von $\rho$ nach $\rho'$ existiert, dann nennen wir $\rho$ einen *Präfix* von $\rho'$ und schreiben dafür $\rho \sqsubseteq \rho'$. Wir sagen auch $\rho$ ist *Anfang* von $\rho'$ oder $\rho'$ ist *Fortsetzung* von $\rho$.

Ein Ablauf $\rho$ heißt *echter Präfix* von $\rho'$, wenn $\rho \sqsubseteq \rho'$ gilt aber nicht $\rho' \sqsubseteq \rho$.

Offensichtlich ist die Isomorphie von Abläufen eine Äquivalenzrelation. Wir werden im folgenden isomorphe Abläufe nicht mehr unterscheiden. Formal betrachten wir also nicht $\mathbf{R}(S)$ als die Abläufe, sondern den Quotienten $\mathbf{R}(S)/_{\equiv}$. Wir verzichten hier jedoch auf eine formale Ausführung, sondern beweisen im folgenden alle Ergebnisse „modulo“ Isomorphie. Insbesondere ist $\sqsubseteq$ nur „modulo“ Isomorphie eine Ordnungsrelation.

Bevor wir beweisen, daß die Präfixrelation eine Ordnung ist, betrachten wir zwei Beispiele. Diese Beispiele demonstrieren den Unterschied zur üblichen Präfixrelation auf Sequenzen. Im Gegensatz zur Präfixordnung auf Sequenzen, besitzt unsere Präfixordnung Elemente, die bzgl. $\sqsubseteq$ unvergleichbar sind, aber trotzdem eine gemeinsame Fortsetzung besitzen (also kompatibel bzgl. $\sqsubseteq$ sind). In Abb. 3.1 sind zwei solche Abläufe mit einer gemeinsamen Fortsetzung dargestellt.

Ein anderes Merkmal der Präfixordnung auf Sequenzen ist, daß jeder echte Präfix eines möglicherweise unendlichen Ablaufs endlich ist. Auch dies gilt für unsere Präfixordnung nicht mehr. Abbildung 3.2 zeigt einen unendlichen Ablauf mit einem echten Präfix, das ebenfalls unendlich ist.

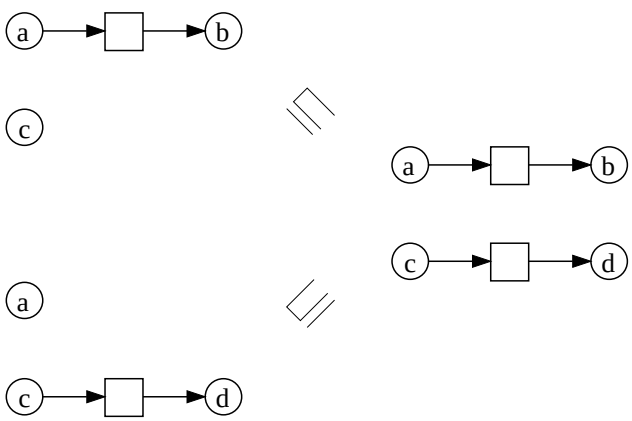



Abbildung 3.1: Unvergleichbare Abläufe mit gemeinsamer Fortsetzung

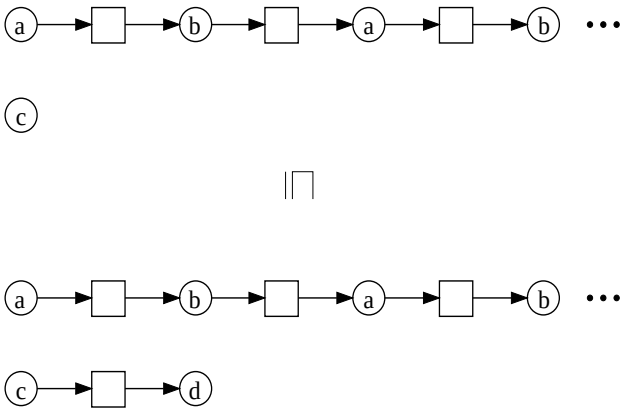



Abbildung 3.2: Ablauf mit einem unendlichen echten Präfix

## 3.2 Die Präfixrelation als Scott-Bereich

Wir zeigen in diesem Abschnitt, daß die Präfixrelation ein Scott-Bereich ist. Wir beweisen zunächst, daß $\sqsubseteq$ eine Ordnungsrelation ist. Das Argument in diesem Beweis ist die Eindeutigkeit des Präfixhomomorphismus; d.h. zwischen zwei Abläufen $\rho \sqsubseteq \rho'$ gibt es genau einen Präfixhomomorphismus. Die Eindeutigkeit des Präfixhomomorphismus spielt auch beim Nachweis der weiteren Eigenschaften der Präfixrelation eine wichtige Rolle.
Der Nachweis, daß die Präfixrelation ein Scott-Bereich ist, ist relativ aufwendig und teilweise sehr technisch. Da diese Aussage ein zentraler Bestandteil dieser Arbeit ist, verzichten wir jedoch nicht auf einen formalen Nachweis. Die technischen Einzelheiten sind aber für die weitere Arbeit nicht von Bedeutung.
Der Grund für den aufwendigen Beweis liegt in der Definition der Präfixrelation mit Hilfe der Präfixhomomorphismen. Eine scheinbare Alternative, die ohne Präfixhomomorphismus auskommt, ist die folgende Definition: $(K', \rho')$ ist Präfix von $(K, \rho)$, wenn eine vollständige Stellenmenge $Q$ (vgl. Def. 2.32) von $K$ existiert, so daß $K' = K|_Q$ und $\rho' = \rho|_Q$ gilt. Diese Definition hat allerdings den Nachteil, daß die Namensgebung der Bedingungen der zugrundeliegenden Prozeßnetze eine Rolle spielt. Ein zu $K'$ isomorphes Prozeßnetz $K''$ mit disjunkter Bedingungsmenge und entsprechender Beschriftung $\rho''$ ist mit dieser Definition kein Präfix von $\rho$. Von zwei isomorphen Abläufen kann deshalb der eine Präfix eines dritten Ablaufs sein, obwohl der andere nicht Präfix dieses dritten Ablaufes ist.
Bei der Definition der Präfixrelation ist uns wichtig, daß die Namensgebung der zugrundeliegenden Prozeßnetze keine Rolle spielt (und wir deshalb isomorphe Abläufe identifizieren können). Aus diesem Grund haben wir die Präfixrelation mit Hilfe der Präfixhomomorphismen definiert.

### 3.2.1 Präfixrelation ist eine Ordnung

Wir beweisen nun, daß die Präfixrelation eine Ordnung ist. Die Reflexivität und Transitivität folgt unmittelbar aus dem folgenden Lemma.

**Lemma 3.5 (Reflexivität und Transitivität)**

Seien $\rho$, $\rho'$ und $\rho''$ Abläufe, $f$ ein Präfixhomomorphismus von $\rho$ nach $\rho'$ und $f'$ ein Präfixhomomorphismus von $\rho'$ nach $\rho''$. Die Abbildung $f' \circ f$ ist ein Präfixhomomorphismus von $\rho$ nach $\rho''$.

Die identische Abbildung $id$ eines Ablaufs $\rho$ auf sich selbst ist ein Präfixhomomorphismus.

Zum Nachweis der Antisymmetrie von $\sqsubseteq$ zeigen wir zunächst, daß zwischen zwei Abläufen höchstens ein Präfixhomomorphismus existiert. Die Beweisargumente für diesen Satz sind die Injektivität der Beschriftung der initialen Scheibe des Prozeßnetzes[2] und die Injektivität der Beschriftung auf dem Nachbereich der Ereignisse. So können wir ausgehend von der initialen Scheibe des Prozeßnetzes induktiv beweisen, daß zwei Präfixhomomorphismen auf allen Elementen des Prozeßnetzes übereinstimmen.

**Satz 3.6 (Eindeutigkeit des Präfixhomomorphismus)**

Seien $\rho$ und $\rho'$ Abläufe. Wenn gilt $\rho \sqsubseteq \rho'$, dann existiert genau ein Präfixhomomorphismus von $\rho$ nach $\rho'$.

**Beweis:** Seien $f, f'$ zwei Präfixhomomorphismen von $K = (B; E)$ nach $K' = (B'; E')$.

Wir beweisen ausgehend von der initialen Scheibe ${}^\circ K$ von $K$ induktiv, daß $f$ und $f'$ übereinstimmen. Dazu definieren wir zwei Folgen von Mengen $B_i$ und $E_i$:

$$\begin{array}{lcl} B_0 & = & {}^\circ K \\ E_i & = & B_i^\bullet \\ B_{i+1} & = & E_i^\bullet \end{array}$$

Da $K$ ein Prozeßnetz ist, ist $(K, <)$ initialisiert und kombinatorisch. Damit gilt $B = \bigcup_{i \in \mathbb{N}} B_i$ und $E = \bigcup_{i \in \mathbb{N}} E_i$ (vgl. Lemma 2.8). Wir zeigen nun durch Induktion über $i$, daß $f$ und $f'$ für jedes $b \in B_i$ übereinstimmen (d.h. $f(b) = f'(b)$).

$i = 0$**:** Sei $b \in {}^\circ K$. Da $f$ und $f'$ Präfixhomomorphismen sind, gilt $\rho'(f(b)) = \rho(b) = \rho'(f'(b))$.
Da $f$ und $f'$ Präfixinjektionen sind, gilt $f(b), f'(b) \in {}^\circ K'$. Da $\rho'$ auf ${}^\circ K'$ injektiv ist (Satz 3.1 (1)) folgt $f(b) = f'(b)$.

$i \to i+1$**:** Wir nehmen an, daß für jedes $b \in B_i$ gilt $f(b) = f'(b)$, und zeigen für $b' \in B_{i+1}$ $f(b') = f'(b')$.
Für ein beliebiges $b' \in B_{i+1}$ existiert ein Ereignis $e \in {}^\bullet b'$ und eine Bedingung $b \in {}^\bullet e \cap B_i$. Für $b$ gilt gemäß Induktionsvoraussetzung $f(b) = f'(b)$ und es folgt $f(b) \in {}^\bullet f(e)$

[2] Deshalb haben wir für verteilte Transitionssysteme nur Mengen als Anfangsmarkierung zugelassen.

und $f(b) = f'(b) \in {}^\bullet f'(e)$. Da $K$ stellen-unverzweigt ist, folgt $f(e) = f'(e)$.
Damit gilt $f(b') \in f(e)^\bullet$ und $f'(b') \in f(e)^\bullet$. Weil $f$ und $f'$ Ablaufhomomorphismen sind, gilt $\rho'(f(b')) = \rho(b') = \rho'(f'(b'))$. Da $\rho'|_{f(e)^\bullet}$ injektiv ist (vgl. Satz 3.1 (2)), folgt $f(b') = f'(b')$.

□

Dieser Satz besagt informell, daß jeder Präfix eindeutig in seine Fortsetzung eingebettet ist. Die Bedeutung dieser Eigenschaft wird schnell an einem Gegenbeispiel klar. Abbildung 3.3 zeigt zwei beschriftete Prozeßnetze, wobei das rechte kein Ablauf ist, weil zwei verschiedene Ele-

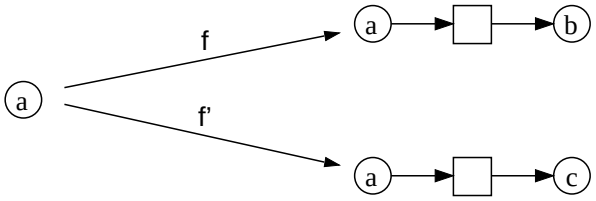



Abbildung 3.3: Nichteindeutige Einbettung eines Präfixes

mente der initialen Scheibe mit $a$ beschriftet sind. Intuitiv ist der linke Ablauf ein Präfix des rechten. Der linke Ablauf kann jedoch auf zwei verschiedene Weisen in den rechten eingebettet werden: Entweder wird die Bedingung des linken Ablaufs auf die obere Bedingung des rechten Prozeßnetzes abgebildet, oder auf die untere (entspr. den Abbildungen $f$ bzw. $f'$). Derartige Mehrdeutigkeiten sind — wie obiger Satz zeigt — bei Abläufen ausgeschlossen.
Aus der Eindeutigkeit des Präfixhomomorphismus zwischen Abläufen folgt die Antisymmetrie der Präfixordnung („modulo“ Isomorphie) fast unmittelbar.

**Korollar 3.7**

Seien $\rho$ und $\rho'$ zwei Abläufe. Aus $\rho \sqsubseteq \rho'$ und $\rho' \sqsubseteq \rho$ folgt $\rho \equiv \rho'$.

**Beweis:** Gelte $\rho \sqsubseteq \rho'$ und $\rho' \sqsubseteq \rho$ und seien $f$, $f'$ die Präfixhomomorphismen von $\rho$ nach $\rho'$ bzw. $\rho'$ nach $\rho$.

Dann ist nach Lemma 3.5 $f' \circ f$ ein Präfixhomomorphismus von $\rho$ auf sich selbst. Die identische Abbildung $id : B \to B$ mit $id(b) = b$ ist eine Präfixhomomorphismus von $\rho$ auf sich selbst. Da nach Satz 3.6

die Präfixinjektionen eindeutig sind, gilt: $f' \circ f = id$. Also ist $f'$ surjektiv und wg. $(f' \circ f)(E) = E$ gilt $f'(E') = E$. Damit ist $f'$ ein Ablaufisomorphismus. □

Insgesamt haben wir gezeigt, daß die Präfixrelation eine Ordnung auf der Menge aller Abläufe bildet.

**Satz 3.8 (Präfixordnung)**

Die Präfixrelation $\sqsubseteq$ auf Abläufen $\mathbf{R}(S)$ ist eine reflexive Ordnung.

### 3.2.2 Beweistechniken

Zur Vereinfachung der Beweise von weiteren Eigenschaften der Präfixordnung führen wir zunächst einige Beweistechniken ein, die insbesondere die Eindeutigkeit der Präfixhomomorphismen ausnutzen.
Die Eindeutigkeit des Präfixhomomorphismus können wir in Form von *kommutierenden Diagrammen* darstellen. Abbildung 3.4 zeigt zwei Beispiele für kommutierende Diagramme, wobei die Knoten Abläufe (bzw. deren zugrundeliegenden Prozeßnetze) sind und die Kanten dazwischen Präfixinjektionen. Das linke Diagramm drückt aus $f_2 \circ f_1 = f_3$ und das rechte besagt $f_3 \circ f_1 = f_4 \circ f_2$. Dies gilt jeweils für zwei Funktionskompositionen, die am selben Knoten $K$ beginnen und beim selben Knoten $K'$ enden.

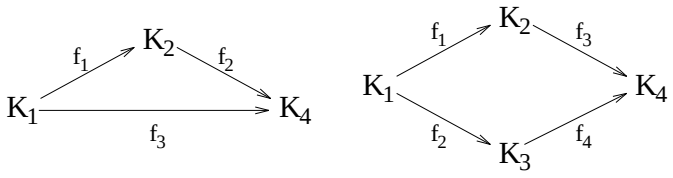



Abbildung 3.4: Zwei kommutierende Diagramme

In den obigen Diagrammen kommen nur die Prozeßnetze und die zugehörigen Präfixhomomorphismen vor. Wir können auch einen Ablauf und die zugehörigen Präfixhomomorphismen als kommutierendes Diagramm darstellen. Abbildung 3.5 stellt die Präfixbeziehung zwischen zwei Abläufen dar. In diesem Diagramm kommen zwei Arten von Abbildungen vor. Einerseits ist $f$ die Präfixinjektion auf den zugrundeliegenden Prozeßnetzen. Andererseits sind $\rho$ und $\rho'$ Beschriftungen von

$K$ bzw. $K'$. Die Kommutativität des Diagramms drückt die Forderung $\rho(b) = \rho'(f(b))$ für jedes $b \in B$ aus.

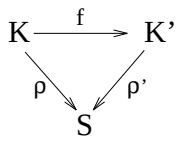



Abbildung 3.5: Diagramm für die Präfixrelation: $\rho \sqsubseteq \rho'$

In obigen Beispielen wäre es auch möglich gewesen, die Gleichheiten explizit aufzuzählen. In etwas komplizierteren Situationen sind die Diagramme aber übersichtlicher als entsprechende Gleichungen. Wir werden kommutierende Diagramme im Beweis der folgenden Lemmata ausnutzen.
Das erste Lemma charakterisiert das Infimum zweier kompatibler Abläufe. Dazu wählen wir eine obere Schranke der beiden Abläufe aus, die nach Voraussetzung existiert. Aus dem zugrundeliegenden Prozeßnetz bilden wir den Ausschnitt (vgl. Def. 2.32), der alle Bedingungen enthält, die im Bild der beiden Präfixhomomorphismen vorkommen.

**Lemma 3.9 (Infimum zweier kompatibler Abläufe)**

Seien $\rho_1$ und $\rho_2$ zwei Abläufe mit einer gemeinsamen Fortsetzung $\rho$.

Sind $f_1$ und $f_2$ die zugehörigen Präfixhomomorphismen von $\rho_1$ bzw. $\rho_2$ nach $\rho$, dann ist $(\overline{K}, \overline{\rho})$ mit

$$\overline{K} \mathrel{\widehat{=}} K|_{f_1(B_1) \cap f_2(B_2)} \qquad \text{und} \qquad \overline{\rho} \mathrel{\widehat{=}} \rho|_{f_1(B_1) \cap f_2(B_2)}$$

das Infimum von $\rho_1$ und $\rho_2$.

**Beweis:** Das linke Diagramm aus Abb. 3.6 zeigt die Ausgangssituation. Nach Satz 2.33 existiert ein Prozeßnetz $\overline{K} \mathrel{\widehat{=}} K|_{f_1(B_1) \cap f_2(B_2)}$ und Präfixinjektionen $g_1$ und $g_2$ von $\overline{K}$ nach $K_1$ bzw. $K_2$. Mit $\overline{\rho}(b) = \rho(b)$ ist $(\overline{K}, \overline{\rho})$ ein Ablauf und $g_1$ und $g_2$ sind Präfixhomomorphismen. $\overline{\rho}$ ist also eine untere Schranke von $\rho_1$ und $\rho_2$. Dies ist im rechten Diagramm von Abb. 3.6 dargestellt.

Es bleibt zu zeigen, daß $(\overline{K}, \overline{\rho})$ die größte untere Schranke von $\rho_1$ und $\rho_2$ ist. Sei $(K', \rho')$ eine untere Schranke von $\rho_1$ und $\rho_2$. Dann existieren Präfixhomomorphismen $g_1'$ und $g_2'$ von $\rho'$ nach $\rho_1$ und

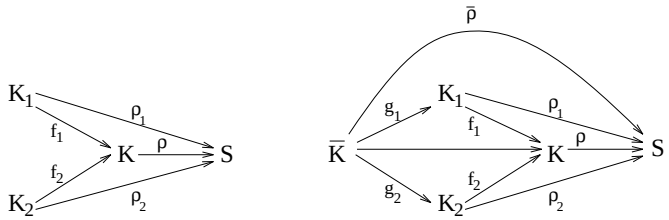



Abbildung 3.6: Diagramme zur Konstruktion des Infimums

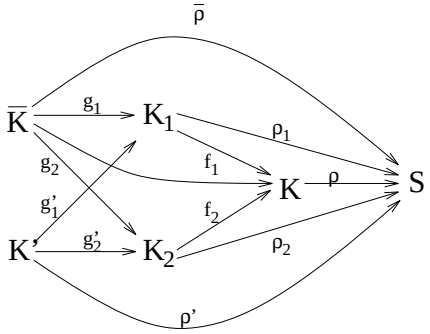



Abbildung 3.7: Beweis der Infimumeigenschaft

$\rho_2$, wie dies in Abb. 3.7 dargestellt ist. Insbesondere sind $f_1 \circ g_1' = f_2 \circ g_2'$ Präfixinjektionen von $K'$ nach $K$. Außerdem gilt $f_1(g_1'(B')) \subseteq f_1(B_1)$ und $f_2(g_2'(B')) \subseteq f_2(B_2)$. Zusammen gilt also $f_1(g_1'(B')) = f_2(g_2'(B')) \subseteq f_1(B_1) \cap f_2(B_2)$. Daher existiert eine Präfixinjektion von $K'$ nach $\overline{K}$ (Korollar 2.34). Weil das Diagramm kommutiert ist diese Präfixinjektion auch ein Ablaufhomomorphismus. Jede untere Schranke von $\rho_1$ und $\rho_2$ ist damit Präfix von $\overline{\rho}$.

Zusammengenommen ist $\overline{\rho}$ das Infimum von $\rho_1$ und $\rho_2$. □

Wir werden nun in ähnlicher Weise für eine kompatible Ablaufmenge das Supremum konstruieren. Zum Beweis benötigen wir jedoch noch eine weitere Hilfsaussage: Wenn zwei Abläufe eine gemeinsame Fortsetzung besitzen und die zugehörigen Präfixinjektionen zwei Bedingungen auf dieselbe Bedingung der Fortsetzung abbilden, dann werden diese beiden Bedingungen in allen gemeinsamen Fortsetzungen jeweils auf dieselbe Bedingung abgebildet.

**Lemma 3.10 (Fortsetzungen kompatibler Abläufe)**

Seien $\rho_1$ und $\rho_2$ zwei kompatible Abläufe, die Abläufe $\rho$ und $\rho'$ zwei gemeinsame Fortsetzungen davon (d.h. $\rho_1, \rho_2 \sqsubseteq \rho$ und $\rho_1, \rho_2 \sqsubseteq \rho'$) und $f_1$, $f_2$ bzw. $f_1'$ und $f_2'$ die entsprechenden Präfixinjektionen in $\rho$ bzw $\rho'$.

Wenn für $b_1 \in B_1$ und $b_2 \in B_2$ gilt $f_1(b_1) = f_2(b_2)$, dann gilt auch $f_1'(b_1) = f_2'(b_2)$.

**Beweis:** Seien $K_1 = (B_1; E_1)$, $K_2 = (B_2; E_2)$, $K = (B; E)$ und $K' = (B'; E')$ die den Abläufen $\rho_1$, $\rho_2$, $\rho$ bzw. $\rho'$ zugrundeliegenden Prozeßnetze. Nach Lemma 3.9 ist $(\overline{K}, \overline{\rho})$ mit $\overline{K} \mathrel{\widehat{=}} K|_{f_1(B_1) \cap f_2(B_2)}$

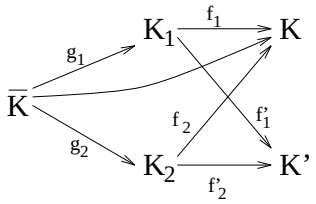



Abbildung 3.8: Diagramm zum Beweis von Lemma 3.10

und $\overline{\rho} \mathrel{\widehat{=}} \rho|_{f_1(B_1) \cap f_2(B_2)}$ das Infimum von $\rho_1$ und $\rho_2$. Dies ist in

Abb. 3.8 mit den zugehörigen Präfixhomomorphismen als kommutierendes Diagramm dargestellt.

Sei $b = f_1(b_1)$. Mit der Voraussetzung $f_1(b_1) = f_2(b_2)$ gilt auch $f_2(b_2) = b$. Aus der Definition von $\overline{\rho}$ folgt $f_1(g_1(b)) = b = f_1(b_1)$ und $f_2(g_2(b)) = b = f_2(b_2)$. Mit der Injektivität der Präfixinjektionen folgt $g_1(b) = b_1$ und $g_2(b) = b_2$. Da das Diagramm kommutiert, gilt $f'_1(g_1(b)) = f'_2(g_2(b))$. Zusammengenommen gilt dann $f'_1(b_1) = f'_1(g_1(b)) = f'_2(g_2(b)) = f'_2(b_2)$. □

Wir charakterisieren nun, wann eine obere Schranke einer Ablaufmenge das Supremum dieser Menge ist: Wenn jede Bedingung einer oberen Schranke $\widehat{\rho}$ einer Ablaufmenge $\mathcal{R}$ in einem Bild eines Präfixhomomorphismus vorkommt, dann ist $\widehat{\rho}$ das Supremum von $\mathcal{R}$.

**Lemma 3.11 (Charakterisierung des Supremums)**

Sei $\mathcal{R} \subseteq \mathbf{R}(S)$ eine Menge, $\widehat{\rho}$ eine obere Schranke von $\mathcal{R}$ und für jeden Ablauf $\rho \in \mathcal{R}$ sei $f_\rho : B \to \widehat{B}$ der Präfixhomomorphismus von $\rho$ nach $\widehat{\rho}$. Wenn gilt $\widehat{B} = \bigcup_{\rho \in \mathcal{R}} f_\rho(B)$, dann ist $\widehat{\rho}$ das Supremum von $\mathcal{R}$.

**Beweis:** Wir zeigen, daß für jede obere Schranke $(K', \rho')$ ein Präfixhomomorphismus von $\widehat{\rho}$ nach $\rho'$ existiert. Damit ist $\widehat{\rho}$ die kleinste obere Schranke von $\mathcal{R}$.

Sei $(K', \rho')$ eine obere Schranke von $\mathcal{R}$. Für jeden Ablauf $\rho \in \mathcal{R}$ existiert ein Präfixhomomorphismus $f'_\rho : B \to B'$ von $\rho$ nach $\rho'$. Wir definieren eine Abbildung $f : \widehat{B} \to B'$ wie folgt: Sei $b \in B$ eine Bedingung eines Ablaufs $\rho \in \mathcal{R}$. Wir definieren $f(f_\rho(b)) = f'_\rho(b)$. Die Abbildung $f$ ist wohldefiniert, denn wenn für $b_1 \in B_1$ und $b_2 \in B_2$ für Abläufe $\rho_1, \rho_2 \in \mathcal{R}$ gilt $f_{\rho_1}(b_1) = f_{\rho_2}(b_2)$, dann gilt nach Lemma 3.10 $f'_{\rho_1}(b_1) = f'_{\rho_2}(b_2)$. Außerdem ist $f$ total, da nach Voraussetzung für jedes $\widehat{b} \in \widehat{B}$ ein Ablauf $\rho \in \mathcal{R}$ und ein $b \in B$ existiert mit $f_\rho(b) = \widehat{b}$.

Wir zeigen nun, daß $f$ ein Ablaufhomomorphismus ist: Sei $\widehat{e} \in \widehat{E}$, $\widehat{b} \in \widehat{e}^\bullet$ und sei $\rho \in \mathcal{R}$ mit einer Bedingung $b \in B$, so daß $f_\rho(b) = \widehat{b}$ und $e \in {}^\bullet b$. Mit Lemma 2.30 (2) gilt dann $f_\rho(e) = \widehat{e}$. Da $f'_\rho$ Präfixhomomorphismus ist, gilt auch $f(\widehat{e}) = (f({}^\bullet\widehat{e}), f(\widehat{e}^\bullet)) = (f'_\rho({}^\bullet e), f'_\rho(e^\bullet)) = f'_\rho(e)$. Damit ist $f$ ein Homomorphismus auf den zugrundeliegenden Prozeßnetzen. Aus der Definition von $f$ folgt unmittelbar $\widehat{\rho}(\widehat{b}) = \rho'(f(\widehat{b}))$. Damit ist $f$ ein Ablaufhomomorphismus.

Es bleibt zu zeigen, daß $f$ eine Präfixinjektion ist. Für $\widehat{b} \in {}^\circ\widehat{K}$ und $f_\rho(b) = \widehat{b}$ gilt mit Lemma 2.30 (1) $b \in {}^\circ B$ und somit $f(\widehat{b}) = f'_\rho(b) \in {}^\circ K'$. Zuletzt zeigen wir, daß $f$ injektiv ist. Seien $\widehat{b}_1, \widehat{b}_2 \in \widehat{B}$ und $f(\widehat{b}_1) = f(\widehat{b}_2)$. Dann existieren Abläufe $\rho_1, \rho_2 \in \mathcal{R}$ und Bedingungen $b_1 \in B_1$ und $b_2 \in B_2$ mit $f_{\rho_1}(b_1) = \widehat{b}_1$, $f_{\rho_2}(b_2) = \widehat{b}_2$ und $f'_{\rho_1}(b_1) = f'_{\rho_2}(b_2)$. Mit Lemma 3.10 gilt dann $\widehat{b}_1 = f_{\rho_1}(b_1) = f_{\rho_2}(b_2) = \widehat{b}_2$.

Damit gilt für jede obere Schranke $\rho'$ von $\mathcal{R}$: $\widehat{\rho} \sqsubseteq \rho'$. Da nach Voraussetzung $\widehat{\rho}$ eine obere Schranke von $\mathcal{R}$ ist, ist $\widehat{\rho}$ das Supremum von $\mathcal{R}$. □

Als unmittelbare Folge ergibt sich, daß die Präfixordnung die dritte Bedingung des Scott-Bereiches erfüllt:

**Korollar 3.12 (Supremum einer kompatiblen Menge)**

Jede kompatible Menge $\mathcal{R} \subseteq \mathbf{R}(S)$ besitzt ein Supremum

**Beweis:** Nach Voraussetzung existiert eine obere Schranke von $\mathcal{R}$. Wir konstruieren aus dieser oberen Schranke das Supremum, indem wir alle Bedingungen weglassen, die nicht Bild eines Präfixhomomorphismus von einem Ablauf der Menge in die obere Schranke sind. Wir zeigen dann, daß dieser Ablauf das Supremum ist.

Sei $\rho'$ eine obere Schranke von $\mathcal{R}$ und für jeden Ablauf $\rho \in \mathcal{R}$ sei $f_\rho : B \to B'$ der eindeutige Präfixhomomorphismus. Wir definieren $\widehat{B} = \bigcup_{\rho \in \mathcal{R}} f_\rho(B)$.

Dann ist $\widehat{B}$ als Vereinigung von vollständigen Mengen von $K'$ ebenfalls vollständig und für jeden Ablauf gilt $f_\rho(B) \subseteq \widehat{B}$. Mit Satz 2.33 ist $(K|_{\widehat{B}}, \widehat{\rho}|_{\widehat{B}})$ obere Schranke von $\mathcal{R}$.

Nach Definition gilt $\widehat{B} = \bigcup_{\rho \in \mathcal{R}} f_\rho(B)$. Mit Lemma 3.11 ist $\widehat{\rho}$ Supremum von $\mathcal{R}$. □

Außerdem folgt aus Lemma 3.11, daß das Supremum einer endlichen Menge von endlichen Abläufen endlich ist.

**Korollar 3.13 (Endliches Supremum endlicher Abläufe)**

Sei $\mathcal{R} \subseteq \mathbf{R_f}(S)$ eine endliche Menge von endlichen Abläufen.

Wenn das Supremum $\bigsqcup \mathcal{R}$ existiert, dann ist es endlich.

**Beweis:** Sei $\widehat{\rho} \mathrel{\widehat{=}} \bigsqcup \mathcal{R}$ und für jeden Ablauf $\rho \in \mathcal{R}$ sei $f_\rho : B \to \widehat{B}$ die eindeutige Präfixinjektion von $\rho$ nach $\widehat{\rho}$.

Nach Lemma 3.11 gilt $\widehat{B} = \bigcup_{\rho \in \mathcal{R}} f_\rho(B)$. Da die Bedingungsmengen $B$ für jeden Ablauf $\rho \in \mathcal{R}$ endlich sind, ist $\widehat{B}$ als endliche Vereinigung von endlichen Mengen ebenfalls endlich. □

### 3.2.3 Abläufe als Scott-Bereich

Mit den zuvor bereitgestellten Hilfsmitteln und Eigenschaften können wir nun beweisen, daß die Präfixordnung einen Scott-Bereich bildet. Zunächst zeigen wir, daß die Präfixordnung vollständig ist.

**Theorem 3.14 (Vollständigkeit der Präfixordnung)**

Die Präfixordnung $\sqsubseteq$ ist eine vollständige Ordnung auf der Menge aller Abläufe $\mathbf{R}(S)$.

**Beweis:** Wir müssen zeigen, daß $\mathbf{R}(S)$ ein kleinstes Element besitzt und für jede gerichtete Teilmenge ein Supremum besitzt.

Offensichtlich ist $\varepsilon$ das kleinste Element von $\mathbf{R}(S)$.

Wir betrachten nun eine beliebige gerichtete Menge $\mathcal{R} \subseteq \mathbf{R}(S)$ und konstruieren einen Ablauf $(\widehat{K}, \widehat{\rho})$. Wir zeigen anschließend, daß $\widehat{\rho}$ das Supremum von $\mathcal{R}$ ist.

**Konstruktion des Supremums:** $\widehat{\rho}$ bilden wir, indem wir alle Abläufe aus $\mathcal{R}$ „nebeneinander“ legen und dann die Bedingungen und Ereignisse miteinander verschmelzen, die durch zwei Präfixhomomorphismen zwischen den Abläufen der Menge $\mathcal{R}$ auf dasselbe Element abgebildet werden. Wir zeigen anschließend, daß das so konstruierte Netz ein Prozeßnetz ist und zusammen mit der Abbildung $\widehat{\rho}$ einen Ablauf bildet. Zum Schluß beweisen wir, daß dieser Ablauf das Supremum der Menge $\mathcal{R}$ ist.

Formal ist $\widehat{K}$ der Quotient bzgl. einer Äquivalenzrelation $\sim$. O.B.d.A. nehmen wir an, daß die Bedingungsmengen aller Abläufe aus $\mathcal{R}$ disjunkt sind. Wir definieren $B' \mathrel{\widehat{=}} \bigcup_{\rho \in \mathcal{R}} B$ und $E' \mathrel{\widehat{=}} \bigcup_{\rho \in \mathcal{R}} E$. Für zwei Abläufe $\rho_1, \rho_2 \in \mathcal{R}$ mit $\rho_1 \sqsubseteq \rho_2$ sei jeweils $f_{\rho_1,\rho_2} : B_1 \to B_2$ die eindeutige Präfixinjektion von $\rho_1$ nach $\rho_2$.

Die Äquivalenzrelation auf $B'$ definieren wir wie folgt: Für $b_1 \in B_1$ und $b_2 \in B_2$ gilt $b_1 \sim b_2$ genau dann, wenn ein Ablauf $\rho_3 \in \mathcal{R}$ existiert mit $f_{\rho_1,\rho_3}(b_1) = f_{\rho_2,\rho_3}(b_2)$. Offensichtlich ist $\sim$ reflexiv und symmetrisch. Wir zeigen nun die Transitivität von $\sim$: Wenn $b_1 \sim b_2$

und $b_2 \sim b_3$ gilt, dann gibt es Präfixinjektionen, wie sie auf der linken Seite von Abb. 3.9 dargestellt sind. Für die Bedingungen $b_1$, $b_2$ und $b_3$ gilt: $f_1(b_1) = f_2(b_2)$ und $f_3(b_2) = f_4(b_3)$. Da $\mathcal{R}$ gerichtet ist,

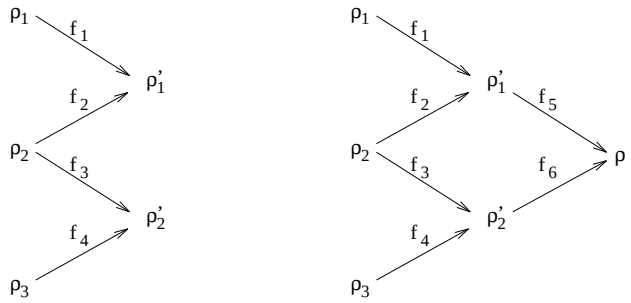



Abbildung 3.9: Zum Beweis der Transitivität von $\sim$

existiert für $\rho'_1$ und $\rho'_2$ eine gemeinsame obere Schranke $\rho \in \mathcal{R}$, wie sie auf der rechten Seite von Abb. 3.9 dargestellt ist. Wegen der Eindeutigkeit der Präfixhomomorphismen, gilt $f_5(f_2(b_2)) = f_6(f_3(b_2))$. Zusammen mit den obigen Gleicheiten folgt $f_5(f_1(b_1)) = f_6(f_4(b_3))$. Also gilt $b_1 \sim b_3$.

Aus der Definition der Äquivalenz mit Hilfe der Präfixhomomorphismen folgt unmittelbar die folgende Eigenschaft: Für zwei Abläufe $\rho_1$ und $\rho_2$ mit Bedingungen $b_1 \sim b_2$ folgt $\rho_1(b_1) = \rho_2(b_2)$ (d.h. es werden nur Elemente mit gleicher Beschriftung durch $\sim$ identifiziert).

Das Kausalnetz $\widehat{K} = (\widehat{B}; \widehat{E})$ definieren wir nun als Quotienten der Vereinigung aller Netze. Dabei definieren wir für eine Menge $Q \subseteq B'$ den Quotienten von $Q$ bzgl $\sim$ durch

$$Q/\sim \widehat{=} \{[b]_\sim \mid b \in Q\}$$

wobei $[b]_\sim \widehat{=} \{b' \in B' \mid b' \sim b \text{ und } b \in Q\}$. Wir definieren $\widehat{B} \widehat{=} B'/_\sim$ und $\widehat{E} \widehat{=} \{({}^\bullet e/_\sim, e^\bullet/_\sim) \mid e \in E\}$. Die Abbildung $\widehat{\rho}$ definieren wir $\widehat{\rho}([b]_\sim) = \rho(b)$ für eine Bedingung $b$ aus einem Ablauf $\rho$. Da nur Elemente mit gleicher Beschriftung durch $\sim$ identifiziert werden, ist diese Definition eindeutig. Da in jeder Äquivalenzklasse von $B'/_\sim$ mindestens ein Element liegt ist die Abbildung total.

**$(\widehat{K}, \widehat{\rho})$ ist ein Ablauf:** Wir zeigen zunächst, daß die Bedingungen $\widehat{B}$ in $\widehat{K}$ unverzweigt sind. Seien $e, e' \in \widehat{E}$ mit $e = ({}^\bullet e_1/_\sim, e_1^\bullet/_\sim)$ und

$e' = ({}^\bullet e_2/_\sim, e_2^\bullet/_\sim)$ für geeignete Ereignisse $e_1$ und $e_2$ von Abläufen $\rho_1$ und $\rho_2$. Wenn ${}^\bullet e \cap {}^\bullet e' \neq \emptyset$, dann gibt es ein $b_1 \in {}^\bullet e_1$ und $b_2 \in {}^\bullet e_2$ mit $b_1 \sim b_2$. Gemäß Definition von $\sim$ gibt es Präfixhomomorphismen $f$ und $f'$ mit $f(b_1) = f'(b_2)$. Da Bedingungen in den zugrundeliegenden Prozeßnetzen unverzweigt sind gilt damit $f(e_1) = f'(e_2)$ und somit gilt ${}^\bullet e_1/_\sim = {}^\bullet e_2/_\sim$ und $e_1^\bullet/_\sim = e_2^\bullet/_\sim$. Also gilt $e = e'$. Für $e^\bullet \cap e'^\bullet \neq \emptyset$ geht der Beweis analog.

Wir zeigen nun, daß für jeden Ablauf $\rho \in \mathcal{R}$ die Abbildung $f_\rho : B \to \widehat{B}$ mit $f(b) = [b]_\sim$ ein Präfixhomomorphismus ist. Offensichtlich ist $f_\rho$ ein Ablaufhomomorphismus. Außerdem ist $f$ injektiv: Sei $f_\rho(b) = f_\rho(b')$ für $b, b' \in B$ (d.h. $b \sim b'$). Gemäß Definition von $\sim$ gibt es eine Präfixinjektion $f_{\rho,\rho'}$ mit $f_{\rho,\rho'}(b) = f_{\rho,\rho'}(b')$. Damit folgt $b = b'$.

Offensichtlich gilt mit $b_1 \sim b_2$ und $b_1 \in {}^\circ K_1$ auch $b_2 \in {}^\circ K_2$ (mit Lemma 2.30 (1)). Damit folgt aus $b \in {}^\circ K$ auch $[b]_\sim \in {}^\circ \widehat{K}$. Damit ist $f_\rho$ eine Präfixinjektion.

Da für alle Bedingungen $b_1 \in {}^\circ K_1$ und $b_2 \in {}^\circ K_2$ mit $\rho_1(b_1) = \rho_2(b_2)$ gilt $b_1 \sim b_2$, gibt es für jedes $s \in S$ höchstens eine Äquivalenzklasse $[b]_\sim$ mit $[b]_\sim \in {}^\circ \widehat{K}$ und $\widehat{\rho}([b]_\sim) = s$. Weil $S$ abzählbar ist folgt daraus die Abzählbarkeit von ${}^\circ \widehat{K}$.

Nun gelten die Voraussetzungen von Satz 2.31: Aus der Definition von $\widehat{K}$ folgt unmittelbar, daß $\widehat{B} = \bigcup_{\rho \in \mathcal{R}} f_\rho(B)$ und $\widehat{E} = \bigcup_{\rho \in \mathcal{R}} f_\rho(E)$. Nach dem zuvor Gezeigten ist $\widehat{K}$ stellen-unverzweigt und ${}^\circ \widehat{K}$ ist abzählbar. Mit Satz 2.31 ist $\widehat{K}$ ein Prozeßnetz.

**$(\widehat{K}, \widehat{\rho})$ ist Supremum:** Offensichtlich ist $\widehat{\rho}$ mit den zuvor angegebenen Präfixhomomorphismen $f_\rho$ für $\rho \in \mathcal{R}$ eine obere Schranke von $\mathcal{R}$. Damit sind auch die Voraussetzungen von Lemma 3.11 erfüllt und damit ist $\widehat{\rho}$ Supremum von $\mathcal{R}$. □

Zum Nachweis der restlichen Anforderungen des Scott-Bereich betrachten wir jetzt die endlichen Abläufe, von denen wir am Ende zeigen werden, daß sie den kompakten Elementen bzgl. der Präfixordnung entsprechen.
Aus Korollar 3.13 wissen wir bereits, daß das Supremum einer endlichen Menge von endlichen Abläufen endlich ist (wenn es existiert). Wir zeigen nun, daß jeder Ablauf das Supremum seiner endlichen Präfixe ist. Außerdem ist die Menge der endlichen Präfixe eines Ablaufs gerichtet. Wir zeigen diese Eigenschaft in etwas allgemeinerer Form für beliebige Ablaufmengen.

**Satz 3.15 (Approximation durch endliche Abläufe)**

Sei $\mathcal{R} \subseteq \mathbf{R}(S)$ eine kompatible Menge von Abläufen. Die Menge der endlichen Präfixe von $\mathcal{R}$ ist definiert durch $\mathcal{R}_\mathbf{f} \mathrel{\widehat{=}} \{\rho' \in \mathbf{R_f}(S) \mid \rho' \sqsubseteq \rho \text{ und } \rho \in \mathcal{R}\}$

Die Menge $\mathcal{R}_\mathbf{f}$ ist gerichtet, wenn $\mathcal{R}$ gerichtet ist.

Es gilt $\bigsqcup \mathcal{R}_\mathbf{f} = \bigsqcup \mathcal{R}$.

**Beweis:** Wir zeigen zunächst, daß $\mathcal{R}_\mathbf{f}$ gerichtet ist, wenn $\mathcal{R}$ gerichtet ist. Wir nehmen an, daß $\mathcal{R}$ gerichtet ist. Sind $\rho_1, \rho_2 \in \mathcal{R}_\mathbf{f}$, dann gibt es $\rho_1', \rho_2' \in \mathcal{R}$ mit $\rho_1 \sqsubseteq \rho_1'$ und $\rho_2 \sqsubseteq \rho_2'$. Da $\mathcal{R}$ gerichtet ist, gibt es ein $\rho' \in \mathcal{R}$ mit $\rho_1' \sqsubseteq \rho'$ und $\rho_2' \sqsubseteq \rho'$. Also ist $\rho'$ obere Schranke von $\rho_1$ und $\rho_2$. Damit gilt insbesondere $\rho_1 \sqcup \rho_2 \sqsubseteq \rho'$. Wegen Korollar 3.13 ist $\rho_1 \sqcup \rho_2$ endlich. Also gilt $\rho_1 \sqcup \rho_2 \in \mathcal{R}_\mathbf{f}$.

Wir zeigen nun, daß gilt $\bigsqcup \mathcal{R}_\mathbf{f} = \bigsqcup \mathcal{R}$. Offensichtlich ist mit $\mathcal{R}$ auch $\mathcal{R}_\mathbf{f}$ kompatibel, da jede obere Schranke von $\mathcal{R}$ auch obere Schranke von $\mathcal{R}_\mathbf{f}$ ist. Insbesondere ist die kleinste obere Schranke von $\mathcal{R}$ obere Schranke von $\mathcal{R}_\mathbf{f}$. Also gilt $\bigsqcup \mathcal{R}_\mathbf{f} \sqsubseteq \bigsqcup \mathcal{R}$.

Wir zeigen, daß jede Bedingung von $\widehat{\rho} \mathrel{\widehat{=}} \bigsqcup \mathcal{R}$ im Bild eines Präfixhomomorphismus eines Ablaufes aus $\mathcal{R}_\mathbf{f}$ vorkommt. Dann ist mit Lemma 3.11 die Behauptung gezeigt.

Für jede Bedingung $\widehat{b} \in \widehat{B}$ ist $Q = {\downarrow}\widehat{b} \cap \widehat{B}$ endlich. Wegen der endlichen Verzweigtheit ist $({}^\bullet Q)^\bullet$ ebenfalls endlich und $Q \cup ({}^\bullet Q)^\bullet$ ist die kleinste vollständige Menge von $\widehat{K}$, die $\widehat{b}$ enthält. Wegen Lemma 3.11 gibt es einen Ablauf $\rho \in \mathcal{R}$ und eine Präfixinjektion $f_\rho : B \to \widehat{B}$ von $\rho$ nach $\widehat{\rho}$ mit $\widehat{b} \in f_\rho(B)$. Da $f_\rho(B)$ eine vollständige Menge von $\widehat{K}$ ist, gilt $Q \cup ({}^\bullet Q)^\bullet \subseteq f(B)$. Wegen Satz 2.33 (2) gibt es von $(\widehat{K}|_{Q \cup ({}^\bullet Q)^\bullet}, \widehat{\rho}|_{Q \cup ({}^\bullet Q)^\bullet})$ einen Präfixhomomorphismus nach $\rho$. Also ist $(\widehat{K}|_{Q \cup ({}^\bullet Q)^\bullet}, \widehat{\rho}|_{Q \cup ({}^\bullet Q)^\bullet})$ ein endlicher Präfix von $\rho \in \mathcal{R}$.

Für jede Bedingung $\widehat{b} \in \widehat{B}$ gibt es einen endlichen Präfix eines Ablaufes $\rho \in \mathcal{R}$, so daß $\widehat{b}$ Bild einer Bedingung dieses Ablaufes ist. Mit Lemma 3.11 ist $\bigsqcup \mathcal{R}$ Supremum von $\mathcal{R}_\mathbf{f}$.

□

Aus diesem Satz folgt insbesondere, daß die Menge der endlichen Präfixe eines einzelnen Ablaufs $\rho$ gerichtet ist und $\rho$ als Supremum besitzt. Damit läßt sich jeder Ablauf durch endliche Abläufe „approximieren“.

**Korollar 3.16 (Approximation durch endliche Abläufe)**

Für jeden Ablauf $\rho \in \mathbf{R}(S)$ ist die Menge seiner endlichen Präfixe $\mathcal{R}_\mathbf{f}$ gerichtet und es gilt $\rho = \bigsqcup \mathcal{R}_\mathbf{f}$.

**Beweis:** Sei $\mathcal{R}_{\mathbf{f}} \mathrel{\widehat{=}} \{\rho' \in \mathbf{R_f}(S) \mid \rho' \sqsubseteq \rho\}$ die Menge aller endlichen Präfixe von $\rho$. Da $\{\rho\}$ (als einelementige Menge) gerichtet ist, folgt aus Satz 3.15, daß $\mathcal{R}_{\mathbf{f}}$ gerichtet ist und gilt $\bigsqcup \mathcal{R}_{\mathbf{f}} = \bigsqcup \{\rho\} = \rho$. □

Die endlichen Abläufe sind Abläufe mit einem endlichen zugrundeliegenden Prozeßnetz. Die Endlichkeit ist also über die Struktur des Ablaufs definiert. In einem Scott-Bereich sind die endlichen Elemente über die Struktur der Ordnung (und nicht der Elemente) charakterisiert. Zur Unterscheidung haben wir diese Elemente kompakt genannt. Wir zeigen nun, daß die endlichen Abläufe genau den kompakten Elementen der Präfixordnung entsprechen.

**Satz 3.17 (Kompakte und endliche Elemente)**

Ein Ablauf $\rho \in \mathbf{R}(S)$ ist genau dann kompakt (bzgl. $\sqsubseteq$), wenn $\rho$ endlich ist.

**Beweis:** Wir zeigen zunächst, daß jeder endliche Ablauf $\rho \in \mathbf{R_f}(S)$ kompakt ist; d.h. für jede gerichtete Menge $\mathcal{R}$ mit $\rho \sqsubseteq \bigsqcup \mathcal{R}$ gibt es ein $\rho' \in \mathcal{R}$ mit $\rho \sqsubseteq \rho'$. Sei $\widehat{\rho} \mathrel{\widehat{=}} \bigsqcup \mathcal{R}$ und für jeden Ablauf $\rho' \in \mathcal{R}$ sei $f_{\rho'} : B \to \widehat{B}$ der eindeutige Präfixhomomorphismus von $\rho'$ nach $\widehat{\rho}$.

Nach Lemma 3.11 gibt es für jede Bedingung $\widehat{b} \in \widehat{B}$ einen Ablauf $\rho' \in \mathcal{R}$ mit $\widehat{b} = f_{\rho'}(B')$. Wir wählen nun für jede Bedingung $\widehat{b} \in f_{\rho}(B)$ einen Ablauf $\rho' \in \mathcal{R}$ aus, für den gilt $\widehat{b} \in f_{\rho'}(B')$. Diese Menge bezeichnen wir mit $\mathcal{R}'$. Da $B$ endlich ist, ist $\mathcal{R}'$ endlich. Gemäß Definition von $\mathcal{R}'$ und wegen Lemma 3.11 gilt $\rho \sqsubseteq \bigsqcup \mathcal{R}'$. Da $\mathcal{R}$ gerichtet ist und $\mathcal{R}' \subseteq \mathcal{R}$ endlich ist besitzt $\mathcal{R}'$ eine obere Schranke $\rho'' \in \mathcal{R}$; d.h. $\bigsqcup \mathcal{R}' \sqsubseteq \rho''$. Insbesondere gilt $\rho \sqsubseteq \rho''$.

Wir zeigen nun, daß jeder kompakte Ablauf endlich ist: Sei $\rho$ ein kompakter Ablauf. Für jede gerichtete Menge $\mathcal{R}$ mit $\rho \sqsubseteq \bigsqcup \mathcal{R}$ gibt es ein $\rho' \in \mathcal{R}$ mit $\rho \sqsubseteq \rho'$. Insbesondere ist nach Korollar 3.16 die Menge $\mathcal{R}_{\mathbf{f}}$ der endlichen Präfixe von $\rho$ gerichtet und es gilt $\rho \sqsubseteq \bigsqcup \mathcal{R}_{\mathbf{f}}$. Also gibt es ein $\rho' \in \mathcal{R}_{\mathbf{f}}$ mit $\rho \sqsubseteq \rho'$. Damit ist $\rho$ als Präfix eines endlichen Ablaufs ebenfalls endlich. □

Als letzte Eigenschaft zum Nachweis, daß die Präfixordnung ein Scott-Bereich ist, müssen wir zeigen daß es nur abzählbar viele kompakte (d.h. endliche) Abläufe gibt. Dies gilt natürlich wieder nur „modulo“ Isomorphie.

**Lemma 3.18**

Die Menge $\mathbf{R_f}(S)$ ist (bis auf Isomorphie) abzählbar.

**Beweis:** Für jedes $n \in \mathbb{N}$ gibt es bis auf Isomorphie nur endlich viele Prozeßnetze mit $n$ Bedingungen. Weil $S$ abzählbar ist, gibt es für jedes endliche Prozeßnetz $K = (B; E)$ höchstens abzählbar viele Abbildungen $\rho : B \to S$, so daß $(K, \rho)$ ein Ablauf ist.

Die Menge der endlichen Abläufe ist als abzählbare Vereinigung von abzählbaren Mengen abzählbar. □

Wir fassen nun die Aussagen dieses Abschnitts als Theorem zusammen:

**Theorem 3.19 (Die Präfixordnung ist ein Scott-Bereich)**

Die Präfixordnung $(\mathbf{R}(S), \sqsubseteq)$ ist ein Scott-Bereich.

**Beweis:** Nach Theorem 3.14 ist die Präfixordnung eine vollständige Ordnung.

Die Menge der endlichen und damit (gemäß Satz 3.17) kompakten Abläufe ist nach Lemma 3.18 abzählbar.

Nach Korollar 3.16 ist die Menge der endlichen und damit (gemäß Satz 3.17) kompakten Präfixe eines Ablaufs gerichtet und das Supremum dieser Menge ist der Ablauf.

Nach Korollar 3.12 besitzt jede kompatible Menge ein Supremum. □

### 3.2.4 Der Spezialfall der sequentiellen Abläufe

Wir haben zuvor an Beispielen gesehen, daß sich die Präfixordnung in manchen Eigenschaften von der Präfixordnung auf Sequenzen unterscheidet. Dies spiegelt sich darin wieder, daß eine Reihe von Konzepten, die sich im allgemeinen unterscheiden, für den Spezialfall der sequentiellen Abläufe zusammenfallen.

**Satz 3.20 (Sequentiell kompatible Ablaufmengen)**

Für eine Menge von Abläufen $\mathcal{R} \subseteq \mathbf{R}(S)$ sind die folgenden Aussagen äquivalent:

1. $\bigsqcup \mathcal{R} \in \mathbf{R^s}(S)$.
2. $\mathcal{R} \subseteq \mathbf{R^s}(S)$ und $\mathcal{R}$ ist total geordnet.
3. $\mathcal{R} \subseteq \mathbf{R^s}(S)$ und $\mathcal{R}$ ist gerichtet.

4. $\mathcal{R} \subseteq \mathbf{R}^{\mathbf{s}}(S)$ und $\mathcal{R}$ ist kompatibel.

**Beweis:** Wir beweisen die Implikationen $1 \Rightarrow 2$, $2 \Rightarrow 3$, $3 \Rightarrow 4$ und $4 \Rightarrow 1$.

$1 \Rightarrow 2$**:** Sei $\widehat{\rho} \mathrel{\widehat{=}} \bigsqcup \mathcal{R}$ sequentiell. Dann sind alle Präfixe von $\widehat{\rho}$ sequentiell. Insbesondere ist jeder Ablauf aus $\mathcal{R}$ sequentiell.

Wenn $\rho_1, \rho_2 \in \mathcal{R}$ zwei Abläufe mit den Präfixhomomorphismen $f_{\rho_1} : B_1 \to \widehat{B}$ und $f_{\rho_1} : B_2 \to \widehat{B}$ nach $\widehat{\rho}$ sind, dann sind die Mengen $f_{\rho_1}(B_1)$ und $f_{\rho_2}(B_2)$ vollständig in $\widehat{K}$. Da die Bedingungen von $\widehat{K}$ eine Sequenz bilden, gilt $f_{\rho_1}(B_1) \subseteq f_{\rho_2}(B_2)$ oder $f_{\rho_1}(B_1) \supseteq f_{\rho_2}(B_2)$.

Sei o.B.d.A. $f_{\rho_1}(B_1) \subseteq f_{\rho_2}(B_2)$. Wegen Korollar 2.34 gibt es eine Präfixinjektion von $K_1$ nach $K_2$. Diese Präfixinjektion ist offensichtlich ein Ablaufhomomorphismus. Also gilt $\rho_1 \sqsubseteq \rho_2$.

$2 \Rightarrow 3$**:** Jede total geordnete Menge ist gerichtet.

$3 \Rightarrow 4$**:** Jede gerichtete Menge besitzt ein Supremum. Also ist ist jede gerichtete Menge insbesondere kompatibel.

$4 \Rightarrow 1$**:** Wenn $\mathcal{R}$ kompatibel ist, dann besitzt $\mathcal{R}$ ein Supremum (Korollar 3.12). Da jede Bedingung und jedes Ereignis von $\widehat{\rho} \mathrel{\widehat{=}} \bigsqcup \mathcal{R}$ Bild einer Bedingung bzw. eines Ereignisses von $\rho \in \mathcal{R}$ ist, und die Abläufe in $\mathcal{R}$ sequentiell sind, ist jedes Ereignis in $\widehat{\rho}$ unverzweigt und es gilt $\widehat{\rho}(^\circ\widehat{K}) = \{i\}$. $\widehat{\rho}$ ist also sequentiell.

□

Wenn wir nur sequentielle Abläufe betrachten, ist es also egal, ob wir von einer kompatiblen, einer gerichteten oder einer total geordneten Menge sprechen. Da das Konzept der totalen Ordnung das einfachste zu sein scheint, bauen die meisten Definitionen im sequentiellen Fall auf diesem Konzept auf.

## 3.3 Eigenschaften verteilter Abläufe

Wir wenden uns nun dem Eigenschaftsbegriff zu. Wie bereits erwähnt, ist jede Menge von Abläufen eine Eigenschaft. Wir unterscheiden wieder den Spezialfall der *sequentiellen Eigenschaft*.

**Definition 3.21 (Eigenschaft, Sequentielle Eigenschaft)**

Sei $S$ eine abzählbare Menge. Eine Teilmenge $\mathcal{P} \subseteq \mathbf{R}(S)$ nennen wir *Eigenschaft* über $S$. Eine Teilmenge $\mathcal{P} \subseteq \mathbf{R}^{\mathbf{s}}(S)$ nennen wir *sequentielle Eigenschaft*.

Ein Ablauf $\rho \in \mathbf{R}(S)$ *erfüllt* eine Eigenschaft $\mathcal{P}$, wenn gilt $\rho \in \mathcal{P}$; anderenfalls *verletzt* er die Eigenschaft $\mathcal{P}$.

In der Einleitung haben wir bereits die Mutex-Eigenschaft als ein Beispiel für eine Eigenschaft kennengelernt: Sie enthält genau die Abläufe, in denen von zwei nebenläufigen Bedingungen nie die eine mit $e_1$ und die andere mit $e_2$ beschriftet ist. Wir beschäftigen uns hier jedoch nur mit dem Begriff der Eigenschaft an sich; deshalb führen wir hier noch keine bestimmte syntaktische Repräsentation für Eigenschaften ein. Dies werden wir erst in Kapitel 6 tun.
In diesem Abschnitt charakterisieren wir zwei unterschiedliche Arten von Eigenschaften: *Sicherheitseigenschaften* und *Lebendigkeitseigenschaften*. Diese Unterscheidung hat sich als grundlegendes Prinzip beim Spezifizieren und Verifizieren von reaktiven Systemen herausgestellt [53]. Ein mathematisches Modell für verteilte Abläufe ist deshalb nur tragfähig, wenn in diesem Modell der Begriff der Sicherheits- und der Lebendigkeitseigenschaft formulierbar ist. Wir zeigen, daß dies für unser Ablaufmodell möglich ist; zum Beweis der Adäquatheit zeigen wir, daß der Zerlegungssatz [6] auch für unsere Charakterisierung von Sicherheits- und Lebendigkeitseigenschaften gilt: Jede Eigenschaft ist der Durchschnitt einer geeigneten Sicherheits- und Lebendigkeitseigenschaft.
Die Formalisierung der Sicherheits- und Lebendigkeitseigenschaften benutzt nur den Begriff des endlichen Ablaufs und die Präfixbeziehung zwischen Abläufen. Diese Begriffe werden von jedem Scott-Bereich in natürlicher Weise zur Verfügung gestellt. Alle Ergebnisse[3], die wir hier speziell für unser Ablaufmodell zeigen, gelten für jedes Ablaufmodell, welches ein Scott-Bereich ist. Deshalb ist es wichtig, daß die Abläufe mit der Präfixrelation einen Scott-Bereich bilden.

### 3.3.1 Sicherheitseigenschaften

Wir charakterisieren zunächst die Sicherheitseigenschaften. Informell wurden Sicherheitseigenschaften von Lamport [49] folgendermaßen beschrieben: Ein Sicherheitseigenschaft gewährleistet, daß nie etwas „Unerwünschtes“ passiert. In dieser Form ist diese Beschreibung noch

[3] außer Theorem 3.32

sehr unpräzise. Wir betrachten daher genauer von welcher Art das „Unerwünschte“ sein soll. Dazu betrachten wir nochmals das Mutex-Beispiel. Das „Unerwünschte“ in diesem Beispiel ist, daß die beiden Philosophen zugleich essen. Falls dieses „Unerwünschte“ in einem Ablauf eintritt, können wir das bereits an einem (geeigneten) endlichen Anfang dieses Ablaufs feststellen. Außerdem läßt sich das Verletzen der Mutex-Bedingung nicht mehr durch Fortsetzung des Ablaufes rückgängig machen; die Mutex-Bedingung ist und bleibt verletzt. Formulieren wir diese beiden Forderungen an das „Unerwünschte“ mit dem Begriff des endlichen Ablaufs und der Präfixbeziehung, so erhalten wir die Definition für Sicherheitseigenschaften.

**Definition 3.22 (Sicherheitseigenschaft)**

Eine Eigenschaft $\mathcal{S} \subseteq \mathbf{R}(S)$ heißt *Sicherheitseigenschaft*, wenn für jeden Ablauf $\rho$, der $\mathcal{S}$ verletzt, gilt:

1. Es existiert ein endlicher Anfang $\rho'$ von $\rho$, der $\mathcal{S}$ ebenfalls verletzt.
2. Jede Fortsetzung $\rho'$ von $\rho$ verletzt $\mathcal{S}$.

Diese Definition ist im wesentlichen äquivalent zu der von Alpern und Schneider [6]. Jedoch wird in [6] der Begriff des endlichen Anfangs eines Ablaufs nur implizit gebraucht. Dederichs und Weber [23, 88] benutzen endliche Abläufe explizit. Ihre Definition ist daher bis auf eine kleine Umformung identisch[4]. Die folgende Umformulierung entspricht genau der aus [23]. Dabei entsprechen die beiden Bedingungen aus Definition 3.22 im wesentlichen den beiden Richtungen der „genau-dann-wenn-Beziehung“ von 2. bzw. der Supremums- und Präfixbildung von 3:

**Satz 3.23 (Äquivalente Charakterisierungen)**

Für eine Eigenschaft $\mathcal{S} \subseteq \mathbf{R}(S)$ sind die folgenden Aussagen äquivalent:

1. $\mathcal{S}$ ist eine Sicherheitseigenschaft.
2. Ein Ablauf $\rho \in \mathbf{R}(S)$ erfüllt $\mathcal{S}$ genau dann, wenn jeder endliche Anfang von $\rho$ die Eigenschaft $\mathcal{S}$ erfüllt.
3. $\mathcal{S}$ ist abgeschlossen unter Präfixbildung und Supremumsbildung bzgl. gerichteter Teilmengen von $\mathcal{S}$.

[4] Dort werden allerdings nur Sequenzen als Abläufe betrachtet.

**Beweis:** Wir zeigen die Implikationen $1 \Rightarrow 2$, $2 \Rightarrow 3$ und $3 \Rightarrow 1$:

$1 \Rightarrow 2$**:** Sei $\mathcal{S}$ eine Sicherheitseigenschaft.

Für jeden Ablauf $\rho \in \mathcal{S}$ ist jeder Präfix von $\rho'$ von $\rho$ in $\mathcal{S}$ (Kontraposition der 2. Bed. von Def. 3.22). Insbesondere ist jeder endliche Präfix von $\rho$ in $\mathcal{S}$.

Wenn jeder endliche Präfix $\rho'$ von $\rho$ die Sicherheitseigenschaft $\mathcal{S}$ erfüllt, dann erfüllt auch $\rho$ die Eigenschaft $\mathcal{S}$ (Kontraposition der 1. Bed. von Def. 3.22).

$2 \Rightarrow 3$**:** Sei $\mathcal{S}$ eine Eigenschaft, die die 2. Bed. des Satzes erfüllt.

Da jeder endliche Präfix $\rho'$ eines Ablaufs $\rho \in \mathcal{S}$ die Eigenschaft $\mathcal{S}$ erfüllt, ist $\mathcal{S}$ abgeschlossen unter Bildung endlicher Präfixe.

Bevor wir zeigen, daß $\mathcal{S}$ bzgl. beliebiger Präfixbildung abgeschlossen ist, zeigen wir die Abgeschlossenheit gegenüber Supremumsbildung von gerichteten Teilmengen: Sei $\mathcal{R} \subseteq \mathcal{S}$ eine gerichtete Menge. Wir müssen zeigen, daß $\bigsqcup \mathcal{R} \in \mathcal{S}$ gilt. Sei $\mathcal{R}_\mathbf{f}$ die Menge der endlichen Präfixe von $\mathcal{R}$. Da $\mathcal{S}$ unter Bildung endlicher Präfixe abgeschlossen ist, gilt $\mathcal{R}_\mathbf{f} \subseteq \mathcal{S}$. Wegen Satz 3.15 gilt $\bigsqcup \mathcal{R}_\mathbf{f} = \bigsqcup \mathcal{R}$ und $\mathcal{R}_\mathbf{f}$ ist gerichtet. Wir zeigen nun, daß $\mathcal{R}_\mathbf{f}$ jeden endlichen Präfix von $\bigsqcup \mathcal{R}_\mathbf{f}$ enthält; weil die 2. Bed. des Satzes erfüllt ist, gilt damit $\bigsqcup \mathcal{R}_\mathbf{f} \in \mathcal{S}$. Sei $\rho_1$ ein endlicher Präfix von $\bigsqcup \mathcal{R}_\mathbf{f}$. Weil $\rho_1$ nach Satz 3.17 kompakt ist und $\rho_1 \sqsubseteq \bigsqcup \mathcal{R}_\mathbf{f}$ gilt, gibt es einen Ablauf $\rho_2 \in \mathcal{R}_\mathbf{f}$ mit $\rho_1 \sqsubseteq \rho_2$. Da $\mathcal{R}_\mathbf{f}$ nach Def. gegen endliche Präfixbildung abgeschlossen ist, gilt $\rho_1 \in \mathcal{R}_\mathbf{f}$.

Damit ist gezeigt, daß $\mathcal{S}$ gegen Supremumsbildung von gerichteten Teilmengen abgeschlossen ist.

Es bleibt zu zeigen, daß $\mathcal{S}$ bzgl. beliebiger Präfixbildung abgeschlossen ist. Sei $\rho \in \mathcal{S}$ und $\rho' \sqsubseteq \rho$. Da $\mathcal{S}$ unter endlicher Präfixbildung abgeschlossen ist sind alle endlichen Präfixe von $\rho$ und damit auch alle endlichen Präfixe $\mathcal{R}'_\mathbf{f}$ von $\rho'$ in $\mathcal{S}$. $\mathcal{R}'_\mathbf{f}$ ist gerichtet und es gilt $\bigsqcup \mathcal{R}'_\mathbf{f} = \rho'$. Da $\mathcal{S}$ bzgl. Supremumsbildung von gerichteten Teilmengen abgeschlossen ist, gilt $\rho' \in \mathcal{S}$.

$3 \Rightarrow 1$**:** Sei $\mathcal{S}$ abgeschlossen gegen Präfixbildung und Supremumsbildung von gerichteten Teilmengen.

Wenn ein Ablauf $\rho$ die Eigenschaft $\mathcal{S}$ nicht erfüllt, dann gibt es einen endlichen Präfix von $\rho$, der $\mathcal{S}$ verletzt. Denn sonst gilt für die Menge der endlichen Präfixe $\mathcal{R}_\mathbf{f}$ von $\rho$: $\mathcal{R}_\mathbf{f} \subseteq \mathcal{S}$. Da nach Satz 3.15 die Menge $\mathcal{R}_\mathbf{f}$ gerichtet ist und $\rho = \bigsqcup \mathcal{R}_\mathbf{f}$ gilt, würde nach Voraussetzung (3. Bed.) gelten $\rho \in \mathcal{S}$.

Wenn ein Ablauf $\rho$ die Eigenschaft $\mathcal{S}$ nicht erfüllt, erfüllt keine Fortsetzung $\rho'$ von $\rho$ die Eigenschaft. Denn sonst wäre wegen der Präfixabgeschlossenheit von $\mathcal{S}$ mit $\rho' \in \mathcal{S}$ auch $\rho \in \mathcal{S}$.

□

Damit haben wir einige verschiedene Sichtweisen von Sicherheitseigenschaften angegeben. Jede davon ist in einer anderen Situation nützlich, um Eigenschaften von Sicherheitseigenschaften zu beweisen. Wir geben nun einige Beispiele für Sicherheitseigenschaften an.

**Beispiel 3.1**

1. Die Mutex-Eigenschaft ist eine Sicherheitseigenschaft.
2. Sei $c \in S$ beliebig aber fest. Die Menge aller Abläufe in denen keine Bedingung mit $c$ beschriftet ist, ist eine Sicherheitseigenschaft.
3. Die Menge aller Abläufe, in denen nie zwei nebenläufige Bedingungen gleich beschriftet sind, ist eine Sicherheitseigenschaft. Diese Eigenschaft nennen wir *1-Sicherheit*[5] .
4. Die Menge $\mathbf{R^s}(S)$ ist eine Sicherheitseigenschaft.
5. Sei $\mathcal{T}$ ein verteiltes Transitionssystem. Dann ist $\mathbf{R}(\mathcal{T})$ eine Sicherheitseigenschaft.
6. Sei $n \in \mathbb{N}$ und $Elem_n$ die Menge aller Abläufe mit höchstens $n$ Bedingungen im zugrundeliegenden Prozeßnetz. Dann ist $Elem_n$ eine Sicherheitseigenschaft.

Im Gegensatz zu $Elem_n$ ist die Menge aller endlichen Abläufe $\mathbf{R_f}(S)$ keine Sicherheitseigenschaft, da das Supremum von einer gerichteten Menge von endlichen Abläufen durchaus unendlich sein kann. Auch die Menge der unendlichen Abläufe ($\mathbf{R}(S) \setminus \mathbf{R_f}(S)$ oder kurz $\neg\, \mathbf{R_f}(S)$) ist keine Sicherheitseigenschaft, da $\neg\, \mathbf{R_f}(S)$ nicht präfixabgeschlossen ist.
Da gilt $\mathbf{R_f}(S) = \bigcup_{n\in\mathbb{N}} Elem_n$ und für jedes $n$ die Eigenschaft $Elem_n$ eine Sicherheitseigenschaften ist, sind Sicherheitseigenschaften nicht unter (unendlicher) Vereinigung abgeschlossen. Sicherheitseigenschaften sind aber unter beliebigem Durchschnitt und endlicher Vereinigung abgeschlossen sind.

[5]Dieser Begriff wird in der Petrinetztheorie für Systeme und nicht für Abläufe definiert. Wir benutzen ihn hier für die entsprechende (Ablauf-)Eigenschaft.

**Satz 3.24**

1. Sicherheitseigenschaften sind unter Durchschnitt abgeschlossen, d.h. für eine beliebige Familie $(\mathcal{S}_i)_{i\in I}$ von Sicherheitseigenschaften $\mathcal{S}_i$ ist $\bigcap_{i\in I}\mathcal{S}_i$ eine Sicherheitseigenschaft.
2. Sicherheitseigenschaften sind unter endlicher Vereinigung abgeschlossen, d.h. für Sicherheitseigenschaften $\mathcal{S}_1,\ldots,\mathcal{S}_n$ ist $\bigcup_{i=1}^{n}\mathcal{S}_i$ eine Sicherheitseigenschaft.

**Beweis:**

1. Wir zeigen $\bigcap_{i\in I}\mathcal{S}_i$ ist abgeschlossen unter Präfix- und Supremumsbildung.

   Sei $\rho\in\bigcap_{i\in I}$, dann gilt für jedes Präfix $\rho'\sqsubseteq\rho$ und jedes $i\in I$ $\rho'\in\mathcal{S}_i$, da nach Voraussetzung $\mathcal{S}_i$ Sicherheitseigenschaft ist; somit gilt $\rho'\in\bigcap_{i\in I}\mathcal{S}_i$.

   Sei $\mathcal{R}\subseteq\bigcap_{i\in I}\mathcal{S}_i$ eine gerichtete Menge, dann gilt für jedes $i\in I$ $\mathcal{R}\subseteq\mathcal{S}_i$ und somit $\bigsqcup\mathcal{R}\in\mathcal{S}_i$;
2. Es reicht zu zeigen, daß die leere Vereinigung $\emptyset$ und die Vereinigung von jeweils zwei Sicherheitseigenschaften $\mathcal{S}_1\cup\mathcal{S}_2$ Sicherheitseigenschaften sind.

   Offensichtlich ist die leere Menge $\emptyset$ abgeschlossen unter Präfix- und Supremumsbildung von gerichteten Teilmengen. Also ist $\emptyset$ eine Sicherheitseigenschaft.

   Seien nun $\mathcal{S}_1$ und $\mathcal{S}_2$ zwei beliebige Sicherheitseigenschaften. Wir zeigen, daß $\mathcal{S}_1\cup\mathcal{S}_2$ auch Sicherheitseigenschaft gemäß Def. 3.22 ist.

   1. Sei $\rho\notin\mathcal{S}_1\cup\mathcal{S}_2$. Damit gilt $\rho\notin\mathcal{S}_1$ und $\rho\notin\mathcal{S}_2$. Also ex. $\rho',\rho''\in\mathbf{R_f}(S)$ mit $\rho'\sqsubseteq\rho$ und $\rho'\notin\mathcal{S}_1$, $\rho''\sqsubseteq\rho$ und $\rho''\notin\mathcal{S}_2$. Dann gilt $\rho'\sqcup\rho''\sqsubseteq\rho$ und $\rho'\sqcup\rho''\in\mathbf{R_f}(S)$. Wegen $\rho'\sqsubseteq\rho'\sqcup\rho''$ und $\rho''\sqsubseteq\rho'\sqcup\rho''$ und weil $\mathcal{S}_1$ und $\mathcal{S}_2$ Sicherheitseigenschaften sind folgt auch $\rho'\sqcup\rho''\notin\mathcal{S}_1\cup\mathcal{S}_2$.

   2. Sei $\rho\notin\mathcal{S}_1\cup\mathcal{S}_2$. Da $\mathcal{S}_1$ und $\mathcal{S}_2$ Sicherheitseigenschaften sind, gilt für jedes $\rho'$ mit $\rho\sqsubseteq\rho'$ auch $\rho'\notin\mathcal{S}_1$ und $\rho'\notin\mathcal{S}_2$. Also auch $\rho'\notin\mathcal{S}_1\cup\mathcal{S}_2$.

□

Da Satz 3.24 die Spezialfälle der leeren Vereinigung und des leeren Durchschnitts enthält, ist sowohl die leere Menge $\emptyset$ als auch die Menge aller Abläufe $\mathbf{R}(S)$ eine Sicherheitseigenschaft. Damit bilden die

Sicherheitseigenschaften gerade die abgeschlossenen Mengen einer Topologie (vgl. [6]).
Außerdem folgt aus Satz 3.24 (1) unmittelbar, daß es für jede Eigenschaft $\mathcal{P}$ eine kleinste Sicherheitseigenschaft gibt, die $\mathcal{P}$ enthält. Diese bezeichnen wir als den Sicherheitsabschluß von $\mathcal{P}$.

**Definition 3.25 (Sicherheitsabschluß einer Eigenschaft)**
Sei $\mathcal{P} \subseteq \mathbf{R}(S)$ eine beliebige Eigenschaft, dann nennen wir

$$\overline{\mathcal{P}} \mathrel{\widehat{=}} \bigcap \{\mathcal{S} | \mathcal{S} \text{ ist Sicherheitseigenschaft und } \mathcal{P} \subseteq \mathcal{S}\}$$

den *Sicherheitsabschluß* von $\mathcal{P}$.

Offensichtlich ist $\overline{\mathcal{P}}$ als Durchschnitt von Sicherheitseigenschaften eine Sicherheitseigenschaft und es gilt $\mathcal{P} \subseteq \overline{\mathcal{P}}$. Außerdem ist $\overline{\mathcal{P}}$ kleiner als jede Sicherheitseigenschaft $\mathcal{S} \supseteq \mathcal{P}$. Damit gilt $\overline{\mathcal{P}} = \mathcal{P}$ genau dann, wenn $\mathcal{P}$ eine Sicherheitseigenschaft ist. Beispielsweise gilt $\mathbf{R_f}(S) \neq \overline{\mathbf{R_f}(S)} = \mathbf{R}(S)$, weil $\mathbf{R_f}(S)$ keine Sicherheitseigenschaft ist. Der Sicherheitsabschluß $\overline{\mathcal{P}}$ ist zwar als Durchschnitt von Sicherheitseigenschaften definiert, man erhält $\overline{\mathcal{P}}$ auch dadurch, daß man zu $\mathcal{P}$ alle Präfixe von Abläufen aus $\mathcal{P}$ hinzufügt und anschließend die Suprema aller gerichteten Teilmengen hinzufügt.
Den Sicherheitsabschluß einer Eigenschaft werden wir später benutzen, um den Zusammenhang zwischen Sicherheits- und Lebendigkeitseigenschaften herzustellen. Insbesondere erlauben die „Rechenregeln“ für den Sicherheitsabschluß einen einfachen Beweis des Zerlegungssatzes. Als „Rechenregeln“ geben wir die *Axiome von Kuratowski* an, die eine Topologie mit Hilfe des Hüllenoperators axiomatisieren.

**Lemma 3.26 (Eigenschaften des Sicherheitsabschlusses)**
Für beliebige Eigenschaften $\mathcal{P}, \mathcal{P}' \subseteq \mathbf{R}(S)$ gilt:

1. $\overline{\mathcal{P} \cup \mathcal{P}'} = \overline{\mathcal{P}} \cup \overline{\mathcal{P}'}$
2. $\mathcal{P} \subseteq \overline{\mathcal{P}}$
3. $\overline{\overline{\mathcal{P}}} \subseteq \overline{\mathcal{P}}$
4. $\overline{\emptyset} = \emptyset$

**Beweis:**

2. Folgt unmittelbar aus der Definition, da jede Mengen, über die der Durchschnitt gebildet wird, Obermenge von $\mathcal{P}$ ist. Damit ist der Durchschnitt $\overline{\mathcal{P}}$ selbst auch eine Obermenge von $\mathcal{P}$.

3. $\overline{\mathcal{P}}$ ist als Durchschnitt von Sicherheitseigenschaften wieder eine Sicherheitseigenschaft, für die trivialerweise $\overline{\mathcal{P}} \subseteq \overline{\mathcal{P}}$ gilt. Gemäß Def. von $\overline{\overline{\mathcal{P}}}$ kommt $\overline{\mathcal{P}}$ in der Menge vor, über die der Durchschnitt gebildet wird. Also gilt $\overline{\overline{\mathcal{P}}} \subseteq \overline{\mathcal{P}}$.
1. $\overline{\mathcal{P}} \cup \overline{\mathcal{P}'}$ ist wegen Satz 3.24 (2) eine Sicherheitseigenschaft, für die wg. 2 gilt $\mathcal{P} \cup \mathcal{P}' \subseteq \overline{\mathcal{P}} \cup \overline{\mathcal{P}'}$. Gemäß Def. von $\overline{\mathcal{P} \cup \mathcal{P}'}$ kommt deshalb $\overline{\mathcal{P}} \cup \overline{\mathcal{P}'}$ in der Menge vor, über die der Durchschnitt gebildet wird. Also gilt $\overline{\mathcal{P} \cup \mathcal{P}'} \subseteq \overline{\mathcal{P}} \cup \overline{\mathcal{P}'}$.
   Andererseits ist jede Sicherheitseigenschaft, die Obermenge von $\mathcal{P} \cup \mathcal{P}'$ ist, auch Obermenge von $\mathcal{P}$ und $\mathcal{P}'$ und damit auch Obermenge von $\overline{\mathcal{P}}$ und $\overline{\mathcal{P}'}$ und somit auch von $\overline{\mathcal{P}} \cup \overline{\mathcal{P}'}$. Damit gilt $\overline{\mathcal{P}' \cup \mathcal{P}'} \supseteq \overline{\mathcal{P}} \cup \overline{\mathcal{P}'}$.
4. $\emptyset$ ist Sicherheitseigenschaft.

□

Aus Lemma 3.26 folgt unmittelbar eine weitere „Rechenregel“ für den Sicherheitsabschluß.

**Korollar 3.27**

Für zwei Eigenschaften $\mathcal{P}$ und $\mathcal{P}'$ mit $\mathcal{P} \subseteq \mathcal{P}'$ gilt $\overline{\mathcal{P}} \subseteq \overline{\mathcal{P}'}$.

**Beweis:** Sei $\mathcal{P} \subseteq \mathcal{P}'$ und damit $\mathcal{P} \cup \mathcal{P}' = \mathcal{P}'$. Also gilt $\overline{\mathcal{P} \cup \mathcal{P}'} = \overline{\mathcal{P}'}$ und mit Lemma 3.26 (1) folgt $\overline{\mathcal{P}} \cup \overline{\mathcal{P}'} = \overline{\mathcal{P}}$, was gleichbedeutend mit $\overline{\mathcal{P}} \subseteq \overline{\mathcal{P}'}$ ist. □

### 3.3.2 Lebendigkeitseigenschaften

Eine Lebendigkeitseigenschaft beschreibt, daß irgendwann etwas „Erwünschtes“ passiert. Dieses Erwünschte kann z.B. das Eintreten einer bestimmten Transition oder das Erreichen eines bestimmten Zustands sein. Ein wesentliches Merkmal des Erwünschten ist, daß es nach jedem endlichen Anfang eines Ablaufes — egal wie schlecht er auch begonnen hat — in Zukunft noch eintreten kann. Dieses Merkmal bildet die Grundlage zur Formalisierung der Lebendigkeitseigenschaften.

**Definition 3.28 (Lebendigkeitseigenschaft)**

Eine Eigenschaft $\mathcal{L} \subseteq \mathbf{R}(S)$ heißt *Lebendigkeitseigenschaft*, wenn für jeden endlichen Ablauf $\rho \in \mathbf{R_f}(S)$ eine Fortsetzung $\rho'$ von $\rho$ existiert, die $\mathcal{L}$ erfüllt.

Wir betrachten zunächst einige Beispiele für Lebendigkeitseigenschaften.

**Beispiel 3.2**

1. Sei $c \in S$ beliebig aber fest. Die Menge $\mathcal{L}$ der Abläufe, in denen mindestens eine Bedingung mit $c$ beschriftet ist, ist eine Lebendigkeitseigenschaft. Jeder endliche Ablauf kann so fortgesetzt werden, daß mindestens ein $c$ auftritt; man muß an den endlichen Ablauf ja nur eine Bedingung, die mit $c$ beschriftet ist, anhängen.

   Bei den Beispielen für Sicherheitseigenschaften 3.1 (1) haben wir gesehen, daß $\neg\mathcal{L}$ eine Sicherheitseigenschaft ist. Jedoch ist das Komplement einer Sicherheitseigenschaft nicht immer eine Lebendigkeitseigenschaft oder umgekehrt das Komplement einer Lebendigkeitseigenschaft eine Sicherheitseigenschaft. Dies wird an den folgenden beiden Beispielen deutlich.
2. Die Menge der endlichen Abläufe $\mathbf{R_f}(S)$ ist trivialerweise eine Lebendigkeitseigenschaft, da bereits jeder endliche Ablauf (ohne Fortsetzung) die Eigenschaft erfüllt.

   Diese Eigenschaft ist gleichbedeutend mit der Forderung nach Terminierung eines Systems. Damit ist also die Forderung nach Terminierung eines Systems eine Lebendigkeitseigenschaft.
3. Überraschenderweise ist auch das Komplement $\neg\,\mathbf{R_f}(S)$ bzgl. $\mathbf{R}(S)$ — also die Menge aller unendlichen Abläufe — eine Lebendigkeitseigenschaft, da jeder endliche Ablauf eine unendliche Fortsetzung besitzt.

   Damit ist sowohl die Forderung nach Terminierung als auch die Forderung nach Nichtterminierung eine Lebendigkeitseigenschaft.

Wir stellen nun den Zusammenhang zwischen Lebendigkeits- und Sicherheitseigenschaften her. Ansatzpunkt dafür ist die Beobachtung, daß die Menge aller Präfixe einer Lebendigkeitseigenschaft die Menge der endlichen Abläufe umfaßt, denn so wurden Lebendigkeitseigenschaften definiert. Bildet man also den Sicherheitsabschluß einer Lebendigkeitseigenschaft $\mathcal{L}$, so erhält man durch Präfixbildung zunächst alle endlichen Abläufe $\mathbf{R_f}(S)$ und durch anschließende Supremumsbildung die Menge aller Abläufe $\mathbf{R}(S)$. Der Sicherheitsabschluß einer Lebendigkeitseigenschaft ist also immer die triviale Sicherheitseigenschaft $\mathbf{R}(S)$.

**Satz 3.29 (Lebendigkeitseigenschaft)**

Eine Eigenschaft $\mathcal{L}$ ist genau dann eine Lebendigkeitseigenschaft, wenn gilt $\overline{\mathcal{L}} = \mathbf{R}(S)$.

**Beweis:**

„⇒“ Sei $\mathcal{L}$ eine Lebendigkeitseigenschaft. Präfix- und Supremumsbildung von gerichteten Teilmengen liefern sofort $\overline{\mathcal{L}} = \mathbf{R}(S)$.

„⇐“ Gelte nun $\overline{\mathcal{L}} = \mathbf{R}(S)$. Wir zeigen durch Widerspruch, daß $\mathcal{L}$ eine Lebendigkeitseigenschaft ist. Angenommen $\mathcal{L}$ ist keine Lebendigkeitseigenschaft, dann ex. ein $\rho \in \mathbf{R_f}(S)$, so daß für alle $\rho'$ mit $\rho \sqsubseteq \rho'$ gilt $\rho' \notin \mathcal{L}$. Für diesen Ablauf $\rho$ definieren wir die Menge $\mathcal{P}_\rho = \{\rho' \in \mathbf{R}(S) | \rho \not\sqsubseteq \rho'\}$. Offensichtlich gilt $\mathcal{L} \subseteq \mathcal{P}_\rho$ und wegen $\rho \notin \mathcal{P}_\rho$ gilt $\mathcal{P}_\rho \subset \mathbf{R}(S)$. Darüberhinaus ist $\mathcal{P}_\rho$ eine Sicherheitseigenschaft, denn jeder Ablauf, der die Eigenschaft verletzt, besitzt $\rho$ als endlichen Präfix, der $\mathcal{P}_\rho$ verletzt. Also gilt mit Korollar 3.27 $\overline{\mathcal{L}} \subseteq \overline{\mathcal{P}_\rho} = \mathcal{P}_\rho \subset \mathbf{R}(S)$ und damit $\overline{\mathcal{L}} \neq \mathbf{R}(S)$.

□

Lebendigkeitseigenschaften liegen also so dicht[6] in $\mathbf{R}(S)$, daß Präfix- und Supremumsbildung ausreichen um alle Abläufe zu erhalten. Anders formuliert bedeutet dies, daß es nur eine einzige Eigenschaft gibt, die zugleich Sicherheits- und Lebendigkeitseigenschaft ist.

**Korollar 3.30**

Sei $\mathcal{L}$ eine Lebendigkeitseigenschaft und $\mathcal{P}$ eine beliebige Eigenschaft, dann gilt:

1. $\mathcal{L}$ ist genau dann Sicherheitseigenschaft, wenn gilt $\mathcal{L} = \mathbf{R}(S)$.
2. $\mathcal{L} \cup \mathcal{P}$ ist Lebendigkeitseigenschaft.
3. $\mathcal{P} \cup \neg\overline{\mathcal{P}}$ ist Lebendigkeitseigenschaft.

**Beweis:**

1. Wenn $\mathcal{L}$ Sicherheitseigenschaft ist, dann gilt $\mathcal{L} = \overline{\mathcal{L}}$. Da $\mathcal{L}$ auch Lebendigkeitseigenschaft ist gilt außerdem $\overline{\mathcal{L}} = \mathbf{R}(S)$.
2. Es gilt $\mathbf{R}(S) = \overline{\mathcal{L}} \subseteq \overline{\mathcal{L} \cup \mathcal{P}}$. Daraus folgt $\overline{\mathcal{L} \cup \mathcal{P}} = \mathbf{R}(S)$ und $\mathcal{L} \cup \mathcal{P}$ ist damit Lebendigkeitseigenschaft.
3. Mit Lemma 3.26 folgt $\overline{\mathcal{P} \cup \neg\overline{\mathcal{P}}} = \overline{P} \cup \overline{\neg\overline{\mathcal{P}}} = \overline{\overline{\mathcal{P}}} \cup \overline{\neg\overline{\mathcal{P}}} = \overline{\overline{\mathcal{P}} \cup \neg\overline{\mathcal{P}}} = \overline{\mathbf{R}(S)} = \mathbf{R}(S)$.

□

[6] Dicht ist hier informell gemeint; die Bezeichnung trifft aber auch im topologischen Sinne zu.

Im Gegensatz zu Sicherheitseigenschaften sind Lebendigkeitseigenschaften nicht abgeschlossen unter Durchschnitt. In Beispiel 3.2 haben wir gesehen, daß sowohl $\mathbf{R_f}(S)$ als auch $\neg\,\mathbf{R_f}(S)$ Lebendigkeitseigenschaften sind. Offensichtlich ist der Durchschnitt dieser beiden Eigenschaften die leere Menge $\emptyset$, die keine Lebendigkeitseigenschaft ist. In Spezifikationsformalismen sind jedoch häufig nur Lebendigkeitseigenschaften formulierbar, deren Durchschnitt wieder eine Lebendigkeitseigenschaft ist[7]. Diese Eigenschaften sind für einen systematischen Entwurf von Systemen besonders interessant, da nachträgliches Hinzufügen (durch Konjunktion) solcher Eigenschaften zur Lebendigkeitsforderung möglich ist.

### 3.3.3 Zerlegungssatz

Der Zerlegungssatz von Alpern und Schneider [6] besagt, daß sich jede beliebige Eigenschaft als Durchschnitt einer geeigneten Sicherheits- und Lebendigkeitseigenschaft darstellen läßt. Mit Hilfe des Sicherheitsabschlusses können wir für jede Eigenschaft explizit eine solche Zerlegung angeben.

**Theorem 3.31 (Zerlegungssatz)**

Für jede Eigenschaft $\mathcal{P}$ gibt es eine Sicherheitseigenschaft $\mathcal{S}$ und eine Lebendigkeitseigenschaft $\mathcal{L}$, so daß $\mathcal{P} = \mathcal{S} \cap \mathcal{L}$.

**Beweis:** Sei $\mathcal{P}$ eine beliebige Eigenschaft. Wir wählen $\mathcal{S} \mathrel{\widehat{=}} \overline{\mathcal{P}}$ und $\mathcal{L} \mathrel{\widehat{=}} \mathcal{P} \cup \neg\overline{\mathcal{P}}$. Nach Definition ist $\mathcal{S}$ eine Sicherheitseigenschaft. Aus Korollar 3.30 (3) folgt, daß $\mathcal{L}$ eine Lebendigkeitseigenschaft ist. Außerdem gilt:

$$\mathcal{S} \cap \mathcal{L} = \overline{\mathcal{P}} \cap (\mathcal{P} \cup \neg\overline{\mathcal{P}}) = (\overline{\mathcal{P}} \cap \mathcal{P}) \;\cup\; (\overline{\mathcal{P}} \cap \neg\overline{\mathcal{P}}) = \overline{\mathcal{P}} \cap \mathcal{P} = \mathcal{P}$$

□

Dieses Theorem untermauert die gängigen Spezifikationsmethoden, die fast ausnahmslos eine getrennte Spezifikation von Sicherheits- und Lebendigkeitseigenschaften eines Systems verlangen. Das Theorem besagt, daß durch dieses eingeschränkte Vorgehen prinzipiell immer noch alle Eigenschaften spezifizierbar sind. Da sich herausgestellt hat, daß zum Beweis von Sicherheitseigenschaften vollkommen andere Methoden benötigt werden als zum Beweis von Lebendigkeitseigenschaften, hat sich diese Zerlegung durchgesetzt.

[7] Typische dafür ist beispielsweise die Leadsto-Eigenschaft, die wir später betrachten.

Wir haben hier gezeigt, daß sich diese Begriffe kanonisch (samt des Zerlegungssatzes) auf unser verteiltes Ablaufmodell übertragen lassen. Im nachfolgenden Abschnitt werden wir jedoch sehen, daß im verteilten Fall ein für Entwurfsmethoden wesentlicher Zusammenhang für unser Ablaufmodell nicht analog zum sequentiellen Fall gilt.

Zunächst wollen wir jedoch eine weitere — bereits von Alpern und Schneider [6] erwähnte — Zerlegungsmöglichkeit für Eigenschaften zeigen, deren praktische Relevanz jedoch sehr fraglich ist. Wir stellen diese Zerlegung als eine weitere Analogie zwischen verteilten und sequentiellen Eigenschaften nur der Vollständigkeit halber vor.

**Theorem 3.32**

Jede Eigenschaft läßt sich als Durchschnitt zweier Lebendigkeitseigenschaften darstellen.

**Beweis:** Wähle $\mathcal{L}_1 = P \cup \mathbf{R_f}(S)$ und $\mathcal{L}_2 = P \cup \neg\,\mathbf{R_f}(S)$. Da sowohl $\mathbf{R_f}(S)$ als auch $\neg\,\mathbf{R_f}(S)$ Lebendigkeitseigenschaften sind (vgl. Bsp. 3.2) sind wegen Korollar 3.30 auch $\mathcal{L}_1$ und $\mathcal{L}_2$ Lebendigkeitseigenschaften. Außerdem gilt: $\mathcal{L}_1 \cap \mathcal{L}_2 = (\mathcal{P} \cup \mathbf{R_f}(S)) \cap (\mathcal{P} \cup \neg\,\mathbf{R_f}(S)) = \mathcal{P} \;\cup\; (\mathcal{P} \cap \mathbf{R_f}(S)) \;\cup\; (P \cap \neg\,\mathbf{R_f}(S)) \;\cup\; (\mathbf{R_f}(S) \cap \neg\,\mathbf{R_f}(S)) = \mathcal{P}$ □

Dieses Theorem macht Gebrauch davon, daß es eine Lebendigkeitseigenschaft gibt, deren Komplement ebenfalls eine Lebendigkeitseigenschaft ist. Die Existenz einer solchen Menge läßt sich — im Gegensatz zu allen anderen Beweisen in diesem Abschnitt — nicht für jede Ordnung $\sqsubseteq$ beweisen, die ein Scott-Bereich ist. Dies untermauert die Sonderstellung dieses Theorems.

## 3.4 Generierbarkeit

Bisher haben wir gesehen, daß sich viele Aussagen über Abläufe und Eigenschaften vom sequentiellen Fall auf den verteilten Fall übertragen. Wir werden nun zeigen, daß es dennoch einen wichtigen Unterschied zwischen dem sequentiellen und dem verteilten Fall gibt. Dieser Unterschied ist deshalb so bemerkenswert, weil manche sequentielle Entwurfsmethoden auf einer Aussage beruhen, die im verteilten Fall nur noch eingeschränkt gilt.

Wir beschreiben zunächst die Idee dieser Entwurfsmethoden: Jede Eigenschaft $\mathcal{P}$ läßt sich als Paar von Eigenschaften $(\mathcal{S}, \mathcal{L})$ darstellen, so daß gilt $\mathcal{P} = \mathcal{S} \cap \mathcal{L}$ und $\overline{\mathcal{P}} = \mathcal{S}$ (vgl. Beweis von Theorem 3.31). Wir nennen ein solches Paar $(\mathcal{S}, \mathcal{L})$ eine *maschinen-abgeschlossene Zerlegung* von $\mathcal{P}$. Wir wollen nun die Sicherheitseigenschaft $\mathcal{S}$ als Ablaufmenge eines Transitionssystems darstellen. Offensichtlich lassen sich bestimmte Sicherheitseigenschaften nur mit Hilfe von *internen Zuständen* darstellen. Beispielsweise ist die (sequentielle) Eigenschaft, bei der in einem Ablauf ein $b$ nur auftreten darf, wenn zuvor ein $a$ aufgetreten ist, nur mit Hilfe eines internen Zustands modellierbar. Der interne Zustand wird benötigt um sich zu merken, daß bereits ein $a$ aufgetreten ist. Nachdem die Sicherheitseigenschaft $\mathcal{S}$ als Transitionssystem mit internen Zuständen dargestellt ist, will man die Lebendigkeitseigenschaft $\mathcal{L}$ als *Fairnessanforderung* an das Transitionssystem darstellen. Wir werden sehen, daß dies für ein beliebiges Transitionssystem nicht immer möglich ist. Für ein *sackgassenfreies Transitionssystem* läßt sich eine zusätzliche Lebendigkeitseigenschaft $\mathcal{L}$ jedoch immer als Fairnessanforderung an das Transitionssystem darstellen. Darüber hinaus werden wir zeigen, daß sich jede sequentielle Sicherheitseigenschaft durch ein sackgassenfreies Transitionssystem darstellen läßt. Damit ist es im sequentiellen Fall immer möglich, eine Eigenschaft durch ein sackgassenfreies Transitionssystem mit einer Fairnessanforderung darzustellen.
Im verteilten Fall gibt es leider einige in der Praxis relevante Sicherheitseigenschaften, die sich nicht durch ein sackgassenfreies Transitionssystem darstellen lassen. Ein Beispiel dafür ist die Mutex-Eigenschaft. Für diese Eigenschaften ist deshalb das obige Vorgehen nicht möglich.

In Abschnitt 3.4.1 werden wir Transitionssysteme mit internen Zuständen definieren und die sackgassenfreien Transitionssysteme auszeichnen. Dann werden wir diejenigen Eigenschaften charakterisieren, die Ablaufmenge eines sackgassenfreien Transitionssystems sind; wir nennen solche Eigenschaften kurz *sackgassenfrei generierbar*. Insbesondere werden wir sehen, daß jede sequentielle Sicherheitseigenschaft sackgassenfrei generierbar ist; es gibt jedoch nicht-sequentielle Sicherheitseigenschaften, die nicht sackgassenfrei generierbar sind. Dies ist ein wichtiger Unterschied zwischen dem sequentiellen und dem verteilten Fall.
In Abschnitt 3.4.2 geben wir ein nicht-sackgassenfreies Transitionssystem und eine Lebendigkeitseigenschaft an, die nicht durch eine Fair-

nessannahme an das zugrundeliegende Transitionssystem darstellbar ist.
In Abschnitt 3.4.3 werden wir dann formulieren und beweisen, daß sich für ein sackgassenfreies Transitionssystem jede Lebendigkeitseigenschaft durch eine Fairnessannahme an das zugrundeliegende Transitionssystem mit internen Zuständen darstellen läßt. Dazu wird der Begriff der Maschinen-Abgeschlossenheit benutzt.

### 3.4.1 Transitionssystem mit internen Zuständen

Ein Transitionssystem mit internen Zuständen ist ein speziell beschriftetes Transitionssystem.

**Definition 3.33 (Transitionssystem mit internen Zuständen)**

Sei $\mathcal{T} = (N, M)$ ein Transitionssystem mit $N = (Z; T)$, $S$ eine abzählbare Menge und $l : Z \to S$ eine Abbildung, die die folgenden Bedingungen erfüllt:

1. $l|_M$ ist injektiv und
2. für jedes $t \in T$ sind $l|_{\bullet t}$ und $l|_{t\bullet}$ injektiv.

Wir nennen das Paar $(\mathcal{T}, l)$ *Transitionssystem mit internen Zuständen* und $l$ eine *Beschriftung* von $\mathcal{T}$.

Wenn $(\rho, K)$ ein Ablauf von $\mathcal{T}$ über Stellenmenge $Z$ ist, dann nennen wir $(l \circ \rho, K)$ *Ablauf von* $(\mathcal{T}, l)$ über Stellenmenge $S$. Die Menge aller Abläufe des Transitionssystems mit internen Zuständen $(\mathcal{T}, l)$ bezeichen wir mit $\mathbf{R}(\mathcal{T}, l)$ oder $l(\mathbf{R}(\mathcal{T}))$.

Zur Unterscheidung nennen wir die Abläufe $\mathbf{R}(\mathcal{T})$ *interne Abläufe* und die Abläufe $\mathbf{R}(\mathcal{T}, l)$ *externe Abläufe* von $(\mathcal{T}, l)$. Graphisch stellen wir ein Transitionssystem mit internen Zuständen wie ein Transitionssystem dar. Der einzige Unterschied besteht darin, daß verschiedene Stellen dieselbe Beschriftung besitzen können. Abbildung 3.10 zeigt ein sequentielles Transitionssystem mit internen Zuständen, in dem jede Stelle (außer der initialen) mit $a$ beschriftet ist.
Es ist bekannt, daß die Menge der externen Abläufe eines Transitionssystems mit internen Zuständen in bestimmten Fällen keine Sicherheitseigenschaft ist. Das System aus Abb. 3.10 ist ein Beispiel für ein solches System. Dieses Transitionssystem mit unendlich vielen internen Zuständen, die jeweils mit $a$ beschriftet sind, besitzt gerade alle

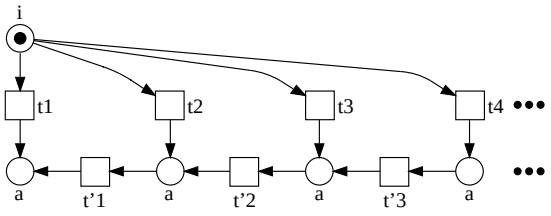



Abbildung 3.10: Transitionssystem mit internen Zuständen

endlichen sequentiellen Abläufe, deren initiale Bedingung mit $i$ und alle anderen Bedingungen mit $a$ beschriftet sind. Diese Menge ist keine Sicherheitseigenschaft, da sie nicht bezüglich Supremumsbildung für gerichtete Mengen abgeschlossen ist. Im Vergleich dazu ist die Ablaufmenge des Systems aus Abb. 3.11 eine Sicherheitseigenschaft. Es besitzt dieselben endlichen Abläufe wie das erste System; zusätzlich besitzt es einen unendlichen Ablauf, der das Supremum der endlichen Abläufe ist.

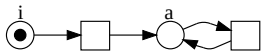



Abbildung 3.11: Transitionssystem mit denselben endlichen Abläufen

Wenn die Ablaufmenge eines Transitionssystems keine Sicherheitseigenschaft ist, dann sind unendlich viele Stellen des Transitionssystems gleich beschriftet. Hinter demselben externen Ablauf verbergen sich dann unendlich viele interne Abläufe. Abadi und Lamport [3] nennen dies *unendlichen internen Nichtdeterminismus*. Sie zeigen, daß die Forderung des *endlichen internen Nichtdeterminismus* eine hinreichende Bedingung dafür ist, daß die Ablaufmenge eines Transitionssystems mit internen Zuständen eine Sicherheitseigenschaft ist.
Wir geben hier die *Sackgassenfreiheit* als eine andere hinreichende Bedingung dafür an, daß die Ablaufmenge eines Transitionssystems mit internen Zuständen eine Sicherheitseigenschaften ist. Die Sackgassenfreiheit hat gegenüber der Forderung des endlichen internen Nichtdeterminismus den eingangs beschriebenen Vorteil, auf den wir in Abschnitts 3.4.3 näher eingehen. Ein Transitionssystem mit internen

Zuständen $(\mathcal{T}, l)$ ist *sackgassenfrei*, wenn jeder Ablauf $\rho \in \mathbf{R}(\mathcal{T}, l)$, für den ein Ablauf $\rho' \in \mathbf{R}(\mathcal{T}, l)$ mit $\rho \sqsubseteq \rho'$ existiert, vom System tatsächlich zu $\rho'$ fortgesetzt werden kann.

**Definition 3.34 (Sackgassenfreies Transitionssystem)**

Ein Transitionssystem mit internen Zuständen $(\mathcal{T}, l)$ heißt *sackgassenfrei*, wenn für je zwei interne Abläufe $\rho$ und $\rho'$ von $(\mathcal{T}, l)$ mit $l \circ \rho \sqsubseteq l \circ \rho'$ ein interner Ablauf $\rho''$ von $\mathcal{T}$ existiert, so daß $\rho \sqsubseteq \rho''$ und $l \circ \rho'' = l \circ \rho'$.

Eine Eigenschaft $\mathcal{P}$ nennen wir *sackgassenfrei generierbar*, wenn es ein sackgassenfreies Transitionssystem mit internen Zuständen $(\mathcal{T}, l)$ mit $\mathcal{P} = \mathbf{R}(\mathcal{T}, l)$ gibt.

Beispielsweise ist das Transitionssystem mit internen Zuständen aus Abb. 3.10 nicht sackgassenfrei: Wenn im internen Ablauf die Transition $t_1$ eintritt, dann kann dieser Ablauf nicht mehr fortgesetzt werden. Der entsprechende externe Ablauf besitzt aber eine Fortsetzung, in der beispielsweise ein weiteres „$a$“ angehängt wird. Dieser „scheinbaren“ Fortsetzung liegt aber ein anderer interner Ablauf zugrunde (z.B. der Ablauf, in dem nacheinander $t_2$ und $t_1'$ eintreten).
Daß dieses System nicht sackgassenfrei ist, sehen wir auch daran, daß die Ablaufmenge dieses Systems keine Sicherheitseigenschaft ist. Wie wir nun zeigen werden, ist die Ablaufmenge jedes sackgassenfreien Transitionssystems eine (spezielle) Sicherheitseigenschaft.

**Satz 3.35**

Sei $(\mathcal{T}, l)$ ein sackgassenfreies Transitionssystem mit internen Zuständen. Dann ist $\mathbf{R}(\mathcal{T}, l)$

1. bzgl. Präfixbildung abgeschlossen und
2. für jede kompatible Teilmenge $\mathcal{P} \subseteq \mathbf{R}(\mathcal{T}, l)$ gilt $\bigsqcup \mathcal{P} \in \mathbf{R}(\mathcal{T}, l)$.

**Beweis:**

1. folgt unmittelbar aus der Präfixabgeschlossenheit von $\mathbf{R}(\mathcal{T})$.
2. Wir zeigen zunächst, daß für zwei kompatible Abläufe $l \circ \rho_1$ und $l \circ \rho_2$ von $(\mathcal{T}, l)$ auch das Supremum $\rho \mathrel{\widehat{=}} l \circ \rho_1 \sqcup l \circ \rho_2$ ein Ablauf von $(\mathcal{T}, l)$ ist. Das Diagramm in Abb. 3.12 zeigt die Ausgangssituation (wobei wir die Bezeichnungen für die Präfixinjektionen weggelassen haben). Insbesondere ist jede Bedingung von $K$ das Bild einer Bedingung von $K_1$ oder $K_2$.

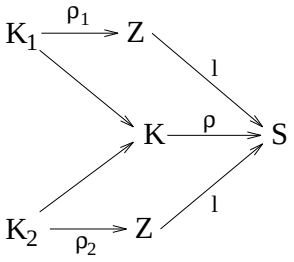



Abbildung 3.12: Ausgangssituation $\rho \mathrel{\widehat{=}} l \circ \rho_1 \sqcup l \circ \rho_2$

Nach Lemma 3.9 gibt es das Infimum von $l \circ \rho_1 \sqcap l \circ \rho_2$. Weil $\mathbf{R}(\mathcal{T}, l)$ präfixabgeschlossen ist, gibt es insbesondere einen Ablauf $\overline{\rho}$ von $\mathcal{T}$ mit $l \circ \overline{\rho} = l \circ \rho_1 \sqcap l \circ \rho_2$. Damit ergibt sich das kommutierende Diagramm aus Abb. 3.13.

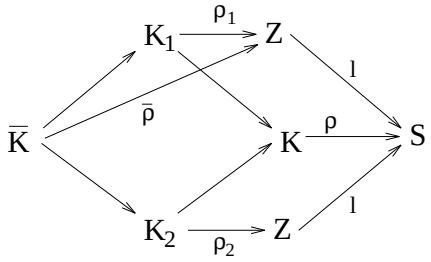



Abbildung 3.13: Infimum der Abläufe $l \circ \rho_1$ und $l \circ \rho_1$

Weil $(\mathcal{T}, l)$ sackgassenfrei ist, gibt es einen Ablauf $\rho_3$ von $\mathcal{T}$ mit $\overline{\rho} \sqsubseteq \rho_3$ und $l \circ \rho_3 = l \circ \rho_2$. Insbesondere ist das zugrundeliegende Prozeßnetz $K_3$ isomorph zu $K_2$. Damit erhalten wir das Diagramm aus Abb. 3.14. Da das Diagramm kommutiert und jede Bedingung von $K$ ein Bild einer Bedingung von $K_1$ oder $K_2$ ist, können wir $\rho' : B \to Z$ eindeutig ergänzen. Weil $\rho_1$ und $\rho_3$ Abläufe von $\mathcal{T}$ sind und jedes Element von $K$ entweder in $K_1$ oder in $K_2$ vorkommt, ist $\rho'$ ein Ablauf von $\mathcal{T}$. Da das Diagramm kommutiert gilt $l \circ \rho' = \rho$. Damit ist für zwei kompatible Abläufe aus $\mathbf{R}(\mathcal{T}, l)$ auch das Supremum aus $\mathbf{R}(\mathcal{T}, l)$.

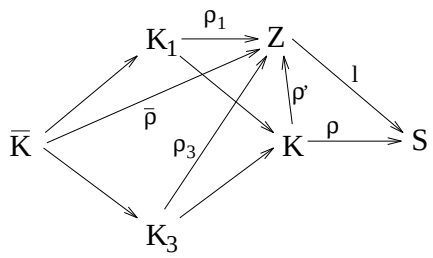



Abbildung 3.14: $l \circ \rho'$ ist Fortsetzung von $l \circ \rho_1$ und $l \circ \rho_1$.

Wir zeigen nun, daß für eine abzählbare kompatible Menge $\mathcal{P} \subseteq \mathbf{R}(\mathcal{T}, l)$ das Supremum in $\mathbf{R}(\mathcal{T}, l)$ liegt. Sei $\rho_1, \rho_2, \ldots$ eine Aufzählung von $\mathcal{P}$. Mit obiger Aussage können wir eine Folge $\rho'_1, \rho'_2, \ldots$ von Abläufen von $\mathcal{T}$ konstruieren, so daß für jedes $i \in \mathbb{N}$ $\rho'_i \sqsubseteq \rho'_{i+1}$ und $l \circ \rho_i \sqsubseteq l \circ \rho'_i$ gilt. Da die Abläufe von $\mathcal{T}$ eine Sicherheitseigenschaft bilden, gilt somit $\widehat{\rho} \mathrel{\widehat{=}} \bigsqcup_{i \in \mathbb{N}} \rho'_i \in \mathbf{R}(\mathcal{T})$ und damit auch $l \circ \widehat{\rho} \in \mathbf{R}(\mathcal{T}, l)$. Außerdem gilt $\bigsqcup \mathcal{P} \sqsubseteq l \circ \bigsqcup_{i \in \mathbb{N}} \rho_i$. Wegen der Präfixabgeschlossenheit existiert ein Ablauf $l \circ \rho \in \mathbf{R}(\mathcal{T}, l)$ mit $\bigsqcup \mathcal{P} = l \circ \rho$.

Da für jede Menge $\mathcal{P}$ die Menge ihrer endlichen Präfixe $\mathcal{P}_{\mathbf{f}}$ abzählbar ist und $\bigsqcup \mathcal{P} = \bigsqcup \mathcal{P}_{\mathbf{f}}$ gilt, folgt die Behauptung.

□

Dieser Satz hat zwei Interpretationen: Einerseits besagt er, daß die Menge der Abläufe eines sackgassenfreien Transitionssystems eine Sicherheitseigenschaft ist. Andererseits besagt er, daß es Sicherheitseigenschaften gibt, die keine Ablaufmenge eines sackgassenfreien Transitionssystems mit internen Zuständen sind: Nämlich solche Sicherheitseigenschaften, die nicht für jede kompatible Menge das Supremum enthalten. Ein einfaches Beispiel dafür ist die Mutex-Eigenschaft: Die beiden Abläufe aus Abb. 3.15 erfüllen die Mutex-Eigenschaft und sind kompatibel; das Supremum dieser beiden Abläufe, das in Abb. 3.16 dargestellt ist, erfüllt die Mutex-Eigenschaft nicht. Die Mutex-Eigenschaft ist also nicht gegen beliebige Supremumsbildung abgeschlossen und deshalb auch nicht sackgassenfrei generierbar.

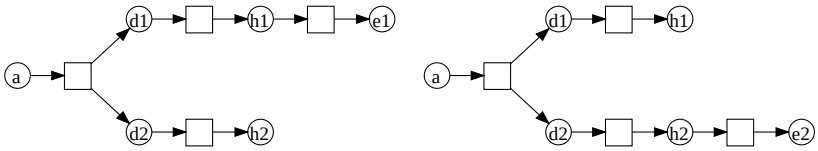



Abbildung 3.15: Zwei Abläufe, die die Mutex-Eigenschaft erfüllen

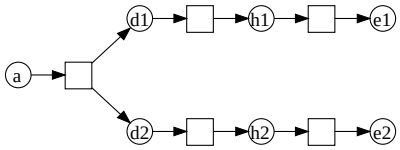



Abbildung 3.16: Supremum der beiden Abläufe aus Abb. 3.15

Wir sehen also, daß nicht jede Sicherheitseigenschaft gegen beliebige Supremumsbildung abgeschlossen ist. Wir kommen hier noch einmal auf unsere Beispiele für Sicherheitseigenschaften (Beispiel 3.1) zurück und geben für sie an, ob sie gegen beliebige Supremumsbildung abgeschlossen sind.

**Beispiel 3.3**

1. Wie gerade gesehen ist die Mutex-Eigenschaft nicht gegen beliebige Supremumsbildung abgeschlossen.
2. Die Menge der Abläufe, in der keine Bedingung mit einem $c$ beschriftet ist, ist gegen beliebige Supremumsbildung abgeschlossen.
3. Die 1-Sicherheit ist nicht gegen beliebige Supremumsbildung abgeschlossen.
4. Die Menge der sequentiellen Abläufe $\mathbf{R}^{\mathbf{s}}(S)$ ist gegen beliebige Supremumsbildung abgeschlossen.
5. Für jedes Transitionssystem $\mathcal{T}$ ist die Menge $\mathbf{R}(\mathcal{T})$ gegen beliebige Supremumsbildung abgeschlossen. Dies ergibt sich als Spezialfall von Satz 3.35.

6. Für jedes $n > 0$ ist die Menge $Elem_n$ (die Menge alle Abläufe mit höchstens $n$ Bedingungen) nicht gegen beliebige Supremumsbildung abgeschlossen.

Informell ist eine Sicherheitseigenschaft gegen beliebige Supremumsbildung abgeschlossen, wenn ihre Einhaltung lokal gewährleistet werden kann; d.h. allein aufgrund der „Vergangenheit“ der Vorbedingungen des betrachteten Ereignisses. Die Information über nebenläufiges Geschehen im Ablauf darf dagegen nicht ausgenutzt werden. Wenn ein Ablauf eine solche Sicherheitseigenschaft verletzt, dann verletzt entweder eine Bedingung der initialen Scheibe diese Eigenschaft, oder es gibt ein Ereignis, das für das Verletzen der Eigenschaft verantwortlich gemacht werden kann.
Diese Idee benutzen wir, um für eine Sicherheitseigenschaft, die gegen beliebige Supremumsbildung abgeschlossen ist, ein sackgassenfreies Transitionssystem mit internen Zuständen (mit deren Hilfe die Vergangenheit protokolliert wird) zu konstruieren, das genau diese Menge als Abläufe besitzt. Deshalb gilt auch die Umkehrung von Satz 3.35.

**Satz 3.36**

Sei $\mathcal{P}$ eine Eigenschaft, die

1. bzgl. Präfixbildung abgeschlossen ist und
2. für jede kompatible Teilmenge $\mathcal{P}' \subseteq \mathcal{P}$ auch das Supremum $\bigsqcup \mathcal{P}'$ enthält.

Es existiert ein sackgassenfreies Transitionssystem mit internen Zuständen $(\mathcal{T}, l)$ mit $\mathbf{R}(\mathcal{T}, l) = \mathcal{P}$.

**Beweis:** Idee: Wir konstruieren ein Transitionssystem mit den Stellen $S \times \mathbf{R_f}(S)$ und der Beschriftung $l : S \times \mathbf{R_f}(S) \to S$ mit $l(s, \rho) = s$. Die Abbildung $l$ abstrahiert also von der zweiten Komponente, die wir dazu benutzen die endliche Vergangenheit, der entsprechend der ersten Komponente beschrifteten Bedingung des Ablaufs, aufzuzeichnen. Da $S$ und $\mathbf{R_f}(S)$ abzählbar sind, ist $S \times \mathbf{R_f}(S)$ ebenfalls abzählbar.

Die Anfangsmarkierung von $\mathcal{T}$ definieren wir wie folgt: Sei $S'$ die Menge aller Beschriftungen, die in einem Anfang eines Ablaufes aus $\mathcal{P}$ auftritt. Wir wählen $M = \{(s, \overline{s}) \mid s \in S'\}$ als Anfangsmarkierung von $\mathcal{T}$, wobei wir unter $\overline{s}$ den endlichen Ablauf verstehen, der nur aus einer einzigen mit $s$ beschrifteten Bedingung besteht.

Die Transitionen von $\mathcal{T}$ sind genau die Transitionen, die an einen Ablauf der sich aus den Vergangenheitskomponente der Stellen im Vorbereich ergibt noch angehängt werden dürfen, ohne $\mathcal{P}$ zu verletzen. Für die Stellen im Nachbereich, muß die Transition die Vergangenheit entsprechend fortschreiben.

Das so konstruierte Transitionssystem $\mathcal{T}$ mit der Beschriftung $l$ besitzt keine Sackgassen. Außerdem besitzt es genau die externen Abläufe $\mathcal{P}$. Der Grund dafür ist, daß aufgrund der intern gespeicherten Vergangenheit jeweils lokal sichtbar ist, ob eine Transition noch an einen Ablauf angehängt werden darf, ohne $\mathcal{P}$ zu verletzen. Weil ein Ablauf, der $\mathcal{P}$ verletzt, eine einzelne Transition besitzt, die $\mathcal{P}$ verletzt, brauchen wir keine Information über nebenläufige Ereignisse im Ablauf. □

Damit ist jede Sicherheitseigenschaft, die für jede kompatible Teilmengen das Supremum enthält, sackgassenfrei generierbar. Zusammengefaßt ergibt sich die folgende Charakterisierung der sackgassenfrei generierbaren Eigenschaften.

**Theorem 3.37 (Sackgassenfrei generierbare Eigenschaften)**

Eine Eigenschaft ist genau dann sackgassenfrei generierbar, wenn sie präfixabgeschlossen und bezüglich beliebiger Supremumsbildung abgeschlossen ist.

Wir haben damit wieder ein zum sequentiellen Fall analoges Ergebnis. Allerdings ist im verteilten Fall ganz wesentlich, daß $\mathcal{P}$ bezüglich beliebiger Supremumsbildung abgeschlossen ist. Diese Bedingung ist nur für sequentielle Eigenschaften äquivalent zu der Forderung, daß $\mathcal{P}$ bezüglich Supremumsbildung von gerichteten Teilmengen abgeschlossen ist. Deshalb gilt für sequentielle Sicherheitseigenschaften:

**Korollar 3.38 (Generierbarkeit sequentieller Eigenschaften)**

Jede sequentielle Sicherheitseigenschaft ist sackgassenfrei generierbar.

**Beweis:** Nach Satz 3.20 ist eine sequentielle Eigenschaft $\mathcal{S}$ genau dann kompatibel, wenn sie gerichtet ist. Jede sequentielle Sicherheitseigenschaft ist deshalb bzgl. beliebiger Supremumsbildung abgeschlossen. Satz 3.36 ist also anwendbar. □

Dieses Ergebnis wird implizit von Abadi und Lamport [3] gezeigt. Sie formulieren in einem Satz, daß jede Sicherheitseigenschaft (im sequentiellen Fall) als Ablaufmenge eines Transitionssystems mit endlichem internen Nichtdeterminismus dargestellt werden kann. Wenn man den Beweis genauer betrachtet, wird dort gezeigt, daß jede Sicherheitseigenschaft auch in deren Formalismus als Ablaufmenge eines sackgassenfreien Transitionssystems dargestellt werden kann. In ihrem Beweis benutzen sie eine interne Variable, in der die Vergangenheit aufgezeichnet wird. Dies ist ganz analog zu unserem Beweis von Satz 3.36; wir müssen die Vergangenheit allerdings „verteilt“ aufzeichnen.

Nicht-sequentielle Sicherheitseigenschaften sind — wie wir am Beispiel der Mutex-Eigenschaft gesehen haben — im allgemeinen nicht sackgassenfrei generierbar.

### 3.4.2 Die Bedeutung der Sackgassenfreiheit

Wir werden nun anhand eines Beispiels zeigen, daß die Sackgassenfreiheit für den Entwurf von Systemen eine große Bedeutung hat. Dazu kommen wir nochmals auf das Beispiel aus der Einleitung zurück, das wir hier etwas verfeinern: Ein denkender Philosoph kann sich entscheiden, nie wieder zu essen (Stellen $u_1$ und $u_2$). Dies wird durch die Transitionen von $t_1$–$t_4$ modelliert. Außerdem nehmen wir zur Vereinfachung der Beispielabläufe an, daß ein essender Philosoph für immer essend bleibt. Wir implementieren die Mutex-Eigenschaft dadurch, daß initial entschieden wird, welcher der beiden Philosophen essen darf. Diese Information wird durch interne Zustände „gespeichert“. Das zugehörige Transitionssystem mit internen Zuständen ist in Abb. 3.17 dargestellt. Zur Verdeutlichung haben wir die Stellen entsprechend der initialen Entscheidung verschieden schattiert. Die Menge der externen Abläufe dieses Transitionssystems ist nicht gegen beliebige Supremumsbildung abgeschlossen. Das heißt, es gibt kein sackgassenfreies Transitionssystem, das genau diese Ablaufmenge besitzt. Insbesondere ist das dargestellte System nicht sackgassenfrei (vgl. Ablauf aus Abb. 3.18).
Wir betrachten nun eine Lebendigkeitsanforderung $\mathcal{L}$: Wenn in einem Ablauf zwei Bedingungen mit der Beschriftungen $h_1$ und $u_2$ vorkommen, dann kommt auch eine Bedingung mit der Beschriftung $e_1$ vor. Für jeden externen Ablauf $\rho_1 \in \mathbf{R}(\mathcal{T}, l)$ gibt es einen Ablauf $\rho_2 \in \mathbf{R}(\mathcal{T}, l)$ mit $\rho_1 \sqsubseteq \rho_2$, der die Lebendigkeitsanforderung $\mathcal{L}$ erfüllt. Es gibt jedoch einen internen Ablauf $\rho_1' \in \mathbf{R}(\mathcal{T})$ von $\mathcal{T}$, der keine interne Fortsetzung $\rho_2' \in \mathbf{R}(\mathcal{T})$ in $\mathcal{T}$ besitzt, für die $l \circ \rho_2'$ die Leben-

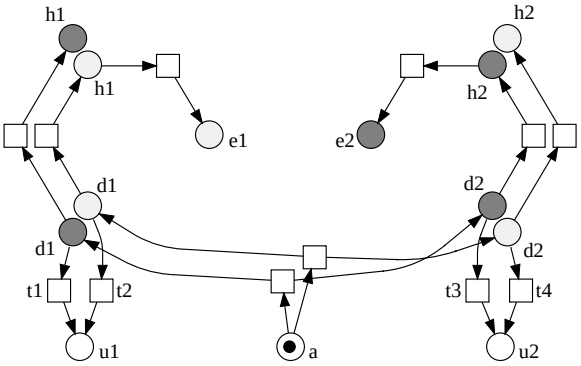



Abbildung 3.17: Ein System $(\mathcal{T}, l)$ mit einer Sackgasse

digkeitsanforderung $\mathcal{L}$ erfüllt. Ein Beispiel für einen solchen Ablauf $\rho'_1$ ist in Abb. 3.18 dargestellt. Der zugehörige externe Ablauf ergibt sich, wenn wir die Schattierung der verschiedenen Stellen ignorieren. Dieser interne Ablauf ist in $\mathcal{T}$ nicht mehr fortsetzbar und erfüllt die Lebendigkeitsanforderung $\mathcal{L}$ nicht.

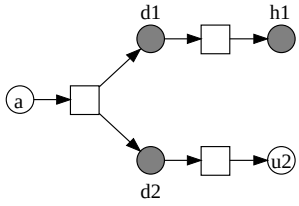



Abbildung 3.18: Ablauf von $\mathcal{T}$ bzw. $(\mathcal{T}, l)$

Wir können also die Lebendigkeitsanforderung, die auf den externen Abläufen von $(\mathcal{T}, l)$ möglich erscheint, auf der Grundlage von $\mathcal{T}$ nicht realisieren, weil das System in eine Sackgasse bezüglich der Lebendigkeitsforderung geraten kann.
Wenn wir nur die Ablaufmenge $\mathcal{S}$ eines Transitionssystems $(\mathcal{T}, l)$ mit internen Zuständen kennen, können wir damit keine Aussage darüber machen, ob eine Lebendigkeitseigenschaft auf der Grundlage von $\mathcal{T}$ realisierbar ist. Wenn hingegen $(\mathcal{T}, l)$ sackgassenfrei ist, dann können

wir dem Paar $(\mathcal{S}, \mathcal{L})$ ansehen, ob $\mathcal{L}$ auf der Grundlage von $\mathcal{T}$ realisierbar ist.
Diese informelle Aussage, werden wir nun mit Hilfe des Begriffs der maschinen-abgeschlossenen Zerlegung $(\mathcal{S}, \mathcal{L})$ formalisieren.

### 3.4.3 Maschinen-abgeschlossene Zerlegung

Wenn für eine Sicherheitseigenschaft $\mathcal{S}$ und eine Eigenschaft $\mathcal{L}$ jeder endliche Ablauf aus $\mathcal{S}$ eine Fortsetzung besitzt, die $\mathcal{S}$ und $\mathcal{L}$ erfüllt, dann heißt das Paar $(\mathcal{S}, \mathcal{L})$ eine *maschinen-abgeschlossene Zerlegung* der Eigenschaft $\mathcal{S} \cap \mathcal{L}$. Wir sagen auch kurz, daß das Paar *maschinen-abgeschlossen* ist. Ein Grund für die zentrale Bedeutung der maschinen-abgeschlossenen Zerlegung ist, daß man eine Lebendigkeitseigenschaft $\mathcal{L}$ durch einen geeigneten „Scheduler“ im nachhinein für ein System mit den Abläufen $\mathcal{S}$ implementieren kann, wenn $(\mathcal{S}, \mathcal{L})$ maschinen-abgeschlossen ist. Das System kann wegen der Maschinen-Abgeschlossenheit nie in einen Zustand geraten von dem aus $\mathcal{S} \cap \mathcal{L}$ nicht mehr möglich ist: „Das System kann sich nicht in die Ecke malen“ [8]. Deshalb ist für Apt, Francez und Katz [8] die Maschinen-Abgeschlossenheit eine wichtige Bedingung für eine sinnvolle Fairnessforderung $\mathcal{L}$ für ein System mit Ablaufmenge $\mathcal{S}$.
Wir geben nun eine formale Charakterisierung der maschinen-abgeschlossenen Paare an. Anschließend zeigen wir, daß diese Charakterisierung mit der obigen informellen Beschreibung übereinstimmt.

**Definition 3.39 (Maschinen-Abgeschlossenheit)**

Sei $\mathcal{S}$ eine Sicherheitseigenschaft und $\mathcal{L}$ eine Eigenschaft. Das Paar $(\mathcal{S}, \mathcal{L})$ heißt *maschinen-abgeschlossen*, wenn gilt $\overline{\mathcal{S} \cap \mathcal{L}} = \mathcal{S}$.

Für ein maschinen-abgeschlossenes Paar $(\mathcal{S}, \mathcal{L})$ ist der Sicherheitsabschluß von $\mathcal{S} \cap \mathcal{L}$ mit $\mathcal{S}$ identisch. Das heißt, solange ein Ablauf $\mathcal{S}$ erfüllt, kann er auch zu einem Ablauf in $\mathcal{S} \cap \mathcal{L}$ fortgesetzt werden.

**Lemma 3.40**

Sei $\mathcal{S}$ eine Sicherheitseigenschaft und $\mathcal{L}$ eine Eigenschaft. Das Paar $(\mathcal{S}, \mathcal{L})$ ist genau dann maschinen-abgeschlossen, wenn für jeden endlichen Ablauf, der $\mathcal{S}$ erfüllt, eine Fortsetzung existiert, die $\mathcal{S} \cap \mathcal{L}$ erfüllt.

**Beweis:**

„⇒“ Sei $\rho \in \mathcal{S}$ ein endlicher Ablauf, der keine Fortsetzung in $\mathcal{S} \cap \mathcal{L}$ besitzt. Wir zeigen, daß dann nicht gilt $\overline{\mathcal{S} \cap \mathcal{L}} = \mathcal{S}$.

Wir definieren $\mathcal{S}_\rho \mathrel{\widehat{=}} \{\rho' \mid \rho \not\sqsubseteq \rho'\}$. Weil $\rho$ keine Fortsetzung in $\mathcal{S} \cap \mathcal{L}$ besitzt, gilt $(\mathcal{S} \cap \mathcal{L}) \subseteq \mathcal{S}_\rho$. Außerdem ist $\mathcal{S}_\rho$ eine Sicherheitseigenschaft. Damit gilt $\overline{\mathcal{S} \cap \mathcal{L}} \subseteq \mathcal{S}_\rho$. Insbesondere gilt dann $\rho \notin \overline{\mathcal{S} \cap \mathcal{L}}$. Nach Voraussetzung gilt aber $\rho \in \mathcal{S}$. Also gilt $\overline{\mathcal{S} \cap \mathcal{L}} \neq \mathcal{S}$.

„⇐“ Wir zeigen, daß die Menge der endlichen Präfixe von $\mathcal{S}$ und $\mathcal{S} \cap \mathcal{L}$ gleich sind. Daraus folgt dann $\mathcal{S} = \overline{\mathcal{S} \cap \mathcal{L}}$.

Offensichtlich ist jeder endliche Präfix von $\mathcal{S} \cap \mathcal{L}$ auch endlicher Präfix von $\mathcal{S}$. Sei $\rho$ ein endlicher Präfix von $\mathcal{S}$. Es existiert nach Voraussetzung eine Fortsetzung von $\rho$, die in $\mathcal{S} \cap \mathcal{L}$ liegt. $\rho$ ist also auch ein endlicher Präfix von $\mathcal{S} \cap \mathcal{L}$.

□

Wir formulieren nun die informelle Aussage aus dem vorangegangenen Abschnitt: Wenn $\mathcal{S}$ die Ablaufmenge eines sackgassenfreien Transitionssystems $(\mathcal{T}, l)$ mit internen Zuständen ist und $(\mathcal{S}, \mathcal{L})$ maschinenabgeschlossen ist, dann ist $\mathcal{L}$ auf Grundlage des Transitionssystems $\mathcal{T}$ realisierbar.

**Satz 3.41**

Sei $(\mathcal{T}, l)$ ein sackgassenfreies Transitionssystem mit internen Zuständen $Z$, $\mathcal{S} \mathrel{\widehat{=}} \mathbf{R}(\mathcal{T}, l)$ und $\mathcal{L}$ eine Lebendigkeitseigenschaft.

Wenn das Paar $(\mathcal{S}, \mathcal{L})$ maschinen-abgeschlossen ist, dann gibt es eine Eigenschaft $\mathcal{L}'$ über $Z$, so daß $(\mathbf{R}(\mathcal{T}), \mathcal{L}')$ maschinen-abgeschlossen ist und $\mathcal{S} \cap \mathcal{L} = l(\mathbf{R}(\mathcal{T}) \cap \mathcal{L}')$.

**Beweis:** Wir definieren $\mathcal{L}' \mathrel{\widehat{=}} \{\rho \mid l \circ \rho \in \mathcal{L}\}$.

1. Wir zeigen zunächst, daß $\mathcal{S} \cap \mathcal{L} = l(\mathbf{R}(\mathcal{T}) \cap \mathcal{L}')$ gilt.

   Sei $\rho' \in \mathcal{S} \cap \mathcal{L}$. Es existiert ein $\rho \in \mathbf{R}(\mathcal{T})$ mit $l \circ \rho = \rho'$. Offensichtlich gilt $\rho \in \mathcal{L}'$ und damit $\rho' \in l(\mathbf{R}(\mathcal{T}) \cap \mathcal{L}')$.

   Sei umgekehrt $\rho' \in l(\mathbf{R}(\mathcal{T}) \cap \mathcal{L}')$. Es existiert ein Ablauf $\rho \in \mathbf{R}(\mathcal{T}) \cap \mathcal{L}'$ mit $l \circ \rho = \rho'$. Damit gilt $\rho' \in \mathcal{S}$ und mit der Definition von $\mathcal{L}'$ (in Abhängigkeit von $\mathcal{L}$) auch $\rho' \in \mathcal{L}$.

2. Wir zeigen nun, daß $(\mathbf{R}(\mathcal{T}), \mathcal{L}')$ maschinen-abgeschlossen ist, wenn $(\mathcal{S}, \mathcal{L})$ maschinen-abgeschlossen ist. Dazu benutzen wir die Charakterisierung aus Lemma 3.40.

   Sei $\rho \in \mathbf{R}(\mathcal{T})$ ein endlicher Ablauf. Wir zeigen, daß ein Ablauf $\rho' \in \mathbf{R}(\mathcal{T}) \cap \mathcal{L}'$ mit $\rho \sqsubseteq \rho'$ existiert. Weil $(\mathcal{S}, \mathcal{L})$ maschinen-abgeschlossen ist, gibt es ein $\rho'' \in \mathbf{R}(\mathcal{T})$ mit $l \circ \rho'' \in \mathcal{S} \cap \mathcal{L}$ und $l \circ \rho \sqsubseteq l \circ \rho''$. Da $(\mathcal{T}, l)$ sackgassenfrei ist, gibt es ein $\rho' \in \mathbf{R}(\mathcal{T})$ mit $\rho \sqsubseteq \rho'$ und $l \circ \rho' = l \circ \rho''$. Nach Definition von $\mathcal{L}'$ gilt auch $\rho' \in \mathcal{L}$.

□

Damit kann eine Sicherheitseigenschaft $\mathcal{S}$, die durch ein sackgassenfreies Transitionssystem dargestellt ist, im nachhinein mit einer Lebendigkeitsforderung $\mathcal{L}$ überlagert werden (wenn $(\mathcal{S}, \mathcal{L})$ maschinen-abgeschlossen ist).
Für beliebige Transitionssysteme mit internen Zuständen gilt dieser nicht. Ein Gegenbeispiel haben wir in Abschnitt 3.4.2 gesehen. Deshalb sind sackgassenfreie Transitionssysteme für den Systementwurf besonders interessant. Dies wiederum begründet die zentrale Rolle der Sicherheitseigenschaften, die gegen beliebige Supremumsbildung abgeschlossen sind.
Leider sind viele typische Sicherheitseigenschaften — z.B. die Mutex-Eigenschaft — nicht sackgassenfrei generierbar. Diese Eigenschaften werden deshalb meist durch stärkere Sicherheitseigenschaften implementiert, die sackgassenfrei generierbar sind.

## 3.5 Literatur

Die Unterscheidung von Sicherheits- und Lebendigkeitseigenschaften wurde von Lamport [49] vorgeschlagen. Diese Unterscheidung war zunächst durch die unterschiedlichen Beweistechniken begründet. In [5] gibt Lamport eine präzise Formalisierung für Sicherheitseigenschaften an.
Alpern und Schneider [6] geben sowohl für Sicherheits- als auch für Lebendigkeitseigenschaften eine formale Charakterisierung an. Sie zeigen, daß die Sicherheitseigenschaften gerade die geschlossenen Mengen einer Topologie sind. Die Lebendigkeitseigenschaften sind dann die dichten Mengen dieser Topologie. Damit ist der Zerlegungssatz auf eine klassische Eigenschaft der Topologie zurückgeführt. Sowohl

Lamport [5] als auch Alpern und Schneider [6] betrachten nur unendliche Abläufe. Deshalb unterscheidet sich diese Topologie bzgl. ihrer Trennungseigenschaften[8] von der Topologie, die bei uns entsteht. In der Charakterisierung von Dederichs und Weber [23, 88] werden endliche Abläufe explizit berücksichtigt. Deshalb erfüllt die Topologie, die ihrem Ansatz zugrunde liegt, die gleichen Trennungseigenschaften[9]wie unsere. Unsere Charakterisierung der Sicherheits- und Lebendigkeitseigenschaften ist eine kanonische Erweiterung der Definition von Dederichs und Weber auf unseren Ablaufbegriff.
Neben dieser Klassifizierung von Eigenschaften hat sich inzwischen eine weitere Einteilung etabliert. Chang, Manna und Pnueli [55, 19] schlagen eine Hierarchie von Eigenschaften vor, deren unterste Klasse die Sicherheitseigenschaften sind. Diese Hierarchie ist ebenfalls durch die verschiedenen Beweistechniken motiviert. Die Charakterisierung basiert auf der Struktur von temporallogischen Formeln, mit der eine Eigenschaft ausgedrückt werden kann. Ein etwas detailierterer Überblick über Arbeiten zu Sicherheits- und Lebendigkeitseigenschaften findet sich in [44].

Das Konzept der Maschinen-Abgeschlossenheit ist von verschiedenen Autoren mit verschiedenen Zielsetzungen vorgeschlagen worden. Deshalb gibt es dafür in der Literatur neben der englischen Bezeichnung *machine closed* [3] die weiteren Bezeichnungen *feasible* [8], *congrous* [37] und *live with respect to a safety property* [23]. Zunächst wurde die maschinen-abgeschlossene Zerlegung als ein Kriterium für sinnvolle Fairnessannahmen für ein System aufgestellt. Außerdem ist die Maschinen-Abgeschlossenheit im Ansatz von Abadi und Lamport eine wichtige Voraussetzung für das Komponieren von Systemen bzw. ihrer Spezifikationen [1, 2] und für das Verfeinern von Systemen [3]. Dederichs und Weber [23] schlagen sogar vor, Eigenschaften von Systemen nur noch maschinen-abgeschlossen zu spezifizieren.
Als Alternative zu unserem Konzept der Sackgassenfreiheit kann man den *endlichen internen Nichtdeterminismus* (dort kurz *fin* genannt) ansehen. Abadi und Lamport [3] zeigen, daß die Ablaufmenge eines Systems mit internen Zuständen eine Sicherheitseigenschaft ist, wenn es nur endlichen internen Nichtdeterminismus besitzt. Allerdings reicht die Forderung des endlichen internen Nichtdeterminismus nicht, um eine Lebendigkeitseigenschaft auf der Menge der externen Abläufe in

[8]Die Topologie von [6] erfüllt die Trennungseigenschaft $T_2$, unsere Topologie erfüllt nur die Trennungseigenschaft $T_0$.

[9]Die Topologie aus [23, 88] erfüllt ebenfalls nur die Trennungseigenschaft $T_0$.

eine Lebendigkeitseigenschaft auf der Menge der internen Abläufe zu transformieren. Aus diesem Grund haben wir den Begriff der Sackgassenfreiheit eingeführt. Darüber hinaus besitzen die Ablaufmengen der sackgassenfreien Transitionssysteme eine elegante ordnungs-theoretische Charakterisierung: Sie sind gegen beliebige Präfix- und Supremumsbildung abgeschlossen.
In [7] werden Sicherheitseigenschaften durch spezielle Büchi-Automaten charakterisiert, die im wesentlichen einem sequentiellen sackgassenfreien Transitionssystem mit internen Zuständen entsprechen. Insofern ist schon lange bekannt, daß im sequentiellen Fall jede Sicherheitseigenschaft Ablaufmenge eines geeigneten sackgassenfreien Transitionssystems ist.

Wir haben hier die Konzepte der Sicherheits- und Lebendigkeitseigenschaft und der Maschinen-Abgeschlossenheit kanonisch auf verteilte Abläufe erweitert. Um den Zerlegungssatz von Alpern und Schneider [6] zu erhalten, reicht es, daß die zugehörige Präfixordnung ein Scott-Bereich ist. Deshalb liegt der Schwerpunkt dieses Kapitels auf dem Nachweis dieser Eigenschaft. Kwiatkowska [48] führt diesen Nachweis für ein anderes Modell verteilter Abläufe, das auf *traces* basiert. Allerdings stellt sie den Zusammenhang zu Sicherheits- und Lebendigkeitseigenschaften nicht her.
Engelfriet [26] benutzt das gleiche Systemmodell wie wir: Er betrachtet auch S/T-Systeme, die eine Menge als Anfangsmarkierung besitzen und deren Kanten ausschließlich Gewicht 1 haben. Als Systemverhalten betrachtet er allerdings die sogenannten *branching processes*, die im Gegensatz zu unserem Ansatz eine branching-time Semantik sind. Er kann damit zeigen, daß die Menge der branching processes einen vollständigen Verband bilden. Insbesondere beschreibt der größte Ablauf in diesem Verband das Systemverhalten vollständig. Da wir an linear-time Eigenschaften interessiert sind, sind seine Ergebnisse für uns nicht nutzbar: Die Menge der Abläufe bildet bei uns keinen vollständigen Verband, sondern „nur“ einen Scott-Bereich.

# Kapitel 4

# Modellierung von Systemkomponenten

In Kapitel 3 haben wir Abläufe und ihre mathematische Struktur im wesentlichen unabhängig von einem Systembegriff untersucht. Wir werden nun ein Modell zur Darstellung von verteilten Systemen vorstellen. Dazu erweitern wir das Konzept des verteilten Transitionssystems:

1. Durch die *Progressannahme* für bestimmte Transitionen schließen wir aus, daß das System in bestimmten Zuständen einfach stehen bleibt, obwohl noch Ereignisse eintreten könnten. Durch die *Fairnessannahme* für eine Transition wird gewährleistet, daß für eine Transition, die in einem Ablauf immer wieder aktiviert ist, irgendwann ein entsprechendes Ereignis im Ablauf vorkommt.

   Wir werden bestimmte Transitionen auszeichnen, für die wir Progress bzw. Fairness annehmen und die Ablaufmenge entsprechend einschränken. So können wir Systeme modellieren, die eine nicht-triviale Lebendigkeitseigenschaft erfüllen.

2. Durch die Definition einer *Schnittstelle* kann eine *Systemkomponente* als Teil eines Systems aufgefaßt werden. Wir zeichnen dazu bestimmte Stellen als *Ein-* bzw. *Ausgabestellen* aus, über die die Systemkomponente mit der Umgebung kommuniziert.

   Verschiedene Systemkomponenten mit geeigneter Schnittstelle können zusammengesetzt werden. Das Verhalten einer Systemkomponente wird so definiert, daß wir beim Zusammensetzen

von Systemkomponenten das Gesamtverhalten aus dem Verhalten der einzelnen Komponenten gewinnen können (d.h. das Verhalten ist *kompositional*).

In Abschnitt 4.1 beschäftigen wir uns zunächst mit den *Progress-* und *Fairnessannahmen*. In Abschnitt 4.2 werden wir Systemkomponenten und ihr Verhalten formalisieren. In Abschnitt 4.3 beschäftigen wir uns mit der Komposition von Systemkomponenten und ihrem Verhalten. Zum Schluß werden wir zeigen, daß wir durch geeignete Umbenennung auch Systemkomponenten miteinander kombinieren können, die aufgrund von Namenskonflikten nicht (unmittelbar) komponierbar sind.

## 4.1 Progress und Fairness

Die Progress- und die Fairnessannahme für eine Menge von Transitionen formulieren wir nun als Eigenschaften. Diese Eigenschaften dienen im nächsten Abschnitt der Formalisierung der Abläufe von Systemen mit Lebendigkeitsannahmen.
Zunächst betrachten wir ein Beispiel zur Motivation der Progressannahme. In Abb. 4.1 ist nochmals das Mutex-System dargestellt. Abbil-

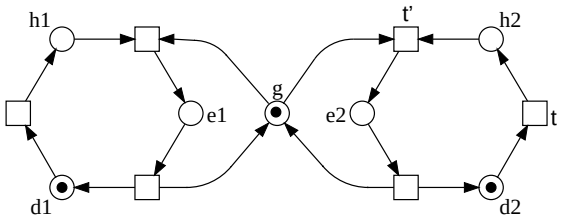



Abbildung 4.1: Das Mutex-System

dung 4.2 zeigt einen unendlichen Ablauf $\rho$ des Mutex-Systems, in dem die mit $d_2$ beschriftete Bedingung nie von einem Ereignis aufgebraucht wird. Offensichtlich kann dieser Ablauf durch Anhängen der Transition $t$ des Mutex-Systems zum Ablauf $\rho'$ aus Abb. 4.3 fortgesetzt werden (obwohl der Ablauf bereits unendlich ist). Daß der Ablauf $\rho$ durch Transition $t$ fortgesetzt werden kann, sieht man unmittelbar daran, daß $d_2$ im Ablauf nicht benutzt wird und $d_2$ die einzige Voraussetzung zum Schalten von $t$ ist; formal drücken wir dies durch ${}^\bullet t \subseteq \rho(K^\circ)$ aus. Der Ablauf $\rho$ hört auf, obwohl er noch durch $t$ fortgesetzt werden kann.

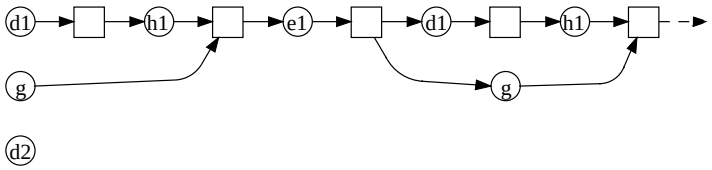



Abbildung 4.2: Ein fortsetzbarer Ablauf $\rho$

Durch die Progressannahme für $t$ schließen wir solche Abläufe aus.

**Definition 4.1 (Progressannahme)**

Sei $\mathcal{T} = ((S';T),M)$ ein verteiltes Transitionssystem mit $S' \subseteq S$ und $t \in T$ eine Transition des Transitionssystems. Ein Ablauf $\rho \in \mathbf{R}(S)$ mit zugrundeliegendem Prozeßnetz $K$ erfüllt die *Progressannahme* bezüglich $t$ genau dann, wenn gilt ${}^\bullet t \not\subseteq \rho(K^\circ)$. Die Menge aller Abläufe (über Stellenmenge $S$), die die Progressannahme bezüglich $t$ erfüllen, bezeichnen wir mit $Prog_t$.

Für eine Teilmenge von Transitionen $T' \subseteq T$ definieren wir $Prog_{T'} \mathrel{\widehat{=}} \bigcap_{t \in T'} Prog_t$.

Der Ablauf aus Abb. 4.3 erfüllt bezüglich jeder Transition des Mutex-Systems die Progressannahme. Trotzdem möchten wir diesen Ablauf

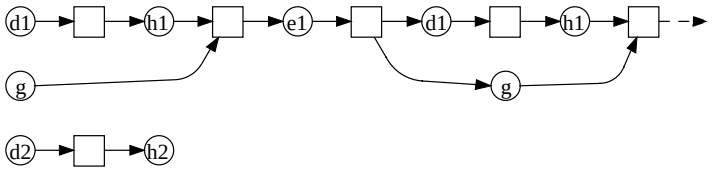



Abbildung 4.3: Ein nicht fairer Ablauf $\rho'$

ausschließen. Denn in diesem Ablauf ist $h_2$ am Ende des Ablaufs verfügbar und $g$ kommt im Ablauf immer wieder vor. Das heißt, daß die Transition $t'$ in diesem Ablauf immer wieder aktiviert ist. Sie kommt aber in dem Ablauf nie vor. Im Bild der Philosophen gesprochen bedeutet das, daß Philosoph „1“ dem Philosophen „2“ die Gabel immer wieder wegschnappt, obwohl Philosoph „2“ sie auch haben möchte.

Der *Konflikt* um die Gabel wir also immer zugunsten des ersten Philosophen entschieden. Transition $t'$ wird in Ablauf $\rho'$ *unfair* behandelt. Wir formalisieren nun, wann ein Ablauf bzgl. einer Transition fair ist. Die Formalisierung ist der Definition der Progressannahme sehr ähnlich: Sei $s \in {}^\bullet t$ und am Ende eines Ablaufes $\rho$ kommen alle Elemente ${}^\bullet t \setminus \{s\}$ vor. Wenn $s$ in dem Ablauf immer wieder auftritt, dann wird $t$ in Ablauf $\rho$ unfair behandelt.
Um technischen Aufwand zu vermeiden, definieren wir die Fairnessannahme nur für solche Transitionen $t$, die jeweils mindestens zwei Stellen im Vorbereich besitzen. Für den Fall, daß eine Transition nur eine Stelle im Vorbereich besitzt, definieren wir die Fairnessannahme wie die Progressannahme.

**Definition 4.2 (Fairnessannahme)**

Sei $\mathcal{T} = ((S'; T), M)$ ein Transitionssystem mit $S' \subseteq S$ und $t \in T$ eine Transition mit $|{}^\bullet t| \geq 2$. Ein Ablauf $\rho \in \mathbf{R}(S)$ mit zugrundeliegendem Prozeßnetz $K$ verletzt die *Fairnessannahme* bezüglich $t$ genau dann, wenn es eine Stelle $s \in {}^\bullet t$ gibt, so daß

1. $\rho(K^\circ) \supseteq {}^\bullet t \setminus \{s\}$ und
2. für jede erreichbare Scheibe $C$ des Ablaufs existiert eine von $C$ erreichbare Scheibe $C'$, mit $s \in \rho(C')$.

Die Menge aller Abläufe (über Stellenmenge $S$), die die Fairnessannahme bzgl. der Transition $t$ erfüllen (d.h. nicht verletzen), bezeichnen wir mit $\mathit{Fair}_t$. Für eine Transition $t$ mit $|{}^\bullet t| = 1$ definieren wir $\mathit{Fair}_t \mathrel{\widehat{=}} \mathit{Prog}_t$.

Für eine Teilmenge $T' \subseteq T$ definieren wir $\mathit{Fair}_{T'} := \bigcap_{t \in T'} \mathit{Fair}_t$.

Zur Definition der Fairnessannahme gibt es viele Alternativen. Wir haben eine möglichst schwache Definition gewählt. So ist einerseits die Verletzung der Fairnessannahme von jedem sequentiellen Beobachter[1] des Ablaufs feststellbar. Andererseits vermeiden wir durch eine schwache Fairnessannahme, daß schwer zu implementierende Lebendigkeitseigenschaften durch eine unrealistische Fairnessannahme garantiert werden. Unsere Fairnessannahme kann bei der Umsetzung des Systemmodells in Hard- oder Software sehr einfach realisiert werden. Da in den meisten Systemen nur für sehr wenige Transitionen Fairness angenommen wird, kann die Fairness an diesen Stellen gezielt berücksichtigt

[1] Wir verzichten hier auf die formale Definition eines sequentiellen Beobachters und einen formalen Nachweis dieser Behauptung.

werden. An vielen Fallstudien hat sich gezeigt, daß unsere Fairnessannahme zur Modellierung verteilter Systemen ausreichend ist.[2]

Die Progressannahme entspricht im wesentlichen der *schwachen Fairness* und die Fairnessannahme der *starken Fairness* [29]. Die Progressannahme macht aber im Gegensatz zur schwachen Fairness keine Aussagen darüber, wie Konflikte entschieden werden. Deshalb ist die Progressannahme keine *echte Fairnessforderung*. Im Gegensatz dazu hat unsere Fairnessannahme Einfluß auf die Entscheidung von Konflikten. Sie ist also eine echte Fairnessforderung.

Weder die Progress- noch die Fairnessannahme schließen den leeren Ablauf $\varepsilon$ aus. Wir stellen deshalb eine zusätzliche Forderung auf: Die Beschriftung der initialen Scheibe eines Ablaufs muß die Anfangsmarkierung des System umfassen. Wir formulieren diese Anforderung hier als Lebendigkeitseigenschaft.

**Definition 4.3 (Anfangsbedingung)**

Für eine gegebene Anfangsmarkierung $M \subseteq S$ bezeichnet $\mathit{Init}_M$ die Menge aller Abläufe $\rho$, für die $\rho(^\circ K) \supseteq M$ gilt.

Für ein Transitionssystem $\mathcal{T} = (N, M)$ gilt für jeden Ablauf $\rho \in \mathbf{R}(\mathcal{T}) \cap \mathit{Init}_M$: $\rho(^\circ K) = M$. In dieser Form werden wir die Eigenschaft $\mathit{Init}_M$ später auch benutzen. Wir haben diese Eigenschaft hier als Lebendigkeitseigenschaft formuliert. Damit können wir formulieren, daß die zusätzlichen Forderungen an Abläufe eines verteilten Transitionssystemen vernünftig sind [8]: Sie schränken die endlichen Anfänge eines Ablaufs des Systems nicht ein. Formal wird dies mit Hilfe der maschinen-abgeschlossenen Zerlegung ausgedrückt.

**Bemerkung 4.4 (Maschinen-Abgeschlossenheit)**

Für ein verteiltes Transitionssystem $\mathcal{T} = ((S'; T), M)$ mit $S' \subseteq S$ und $T^P, T^F \subseteq T$ ist die Eigenschaft $\mathcal{L} \mathrel{\widehat{=}} \mathit{Prog}_{T^P} \cap \mathit{Fair}_{T^F} \cap \mathit{Init}_M$ eine Lebendigkeitseigenschaft. Außerdem ist das Paar $(\mathbf{R}(\mathcal{T}), \mathcal{L})$ maschinen-abgeschlossen.

[2] Für manche Fallstudien ist es zweckmäßig, auch für Transitionen mit nur einer Stelle im Vorbereich eine echte Fairnessforderung aufzustellen; weil wir diesen Fall hier nicht benötigen und dieser Fall in der Definition einen Sonderfall darstellt, haben wir für diese Transitionen keine Fairnessannahmen definiert.

Für ein verteiltes Transitionssystem ergibt sich die Forderung $Init_M$ unmittelbar aus der Anfangsmarkierung des Systems. Für welche Transitionen die Progressannahme und für welche die Fairnessannahme gelten soll, müssen wir dagegen explizit angeben.

**Definition 4.5 (Lebendigkeitsannahme)** Sei $\mathcal{T} = ((S'; T), M)$ ein verteiltes Transitionssystem und $T^P, T^F \subseteq T$. Dann nennen wir $\mathcal{L} = (T^P, T^F)$ Lebendigkeitsannahme für $\mathcal{T}$. Die Menge $T^P$ nennen wir *Progressmenge*, die Menge $T^F$ die *Fairnessmenge*.

Für eine Transition $t$ ist die Fairnessannahme stärker als die Progressannahme (d.h. $Fair_t \subseteq Prog_t$). Deshalb können wir im folgenden immer annehmen, daß für eine Lebendigkeitsannahme $\mathcal{L} = (T^P, T^F)$ gilt $T^F \subseteq T^P$. Graphisch stellen wir die Transitionen, die weder in $T^P$ noch in $T^F$ vorkommen, durch gestrichelte Quadrate, die Transitionen, die in $T^P$ und nicht in $T^F$ vorkommen, durch die bisher verwendeten Quadrate für Transitionen[3] und die Transitionen in $T^F$ durch grau unterlegte Quadrate dar. Abbildung 4.4 zeigt jeweils ein Beispiel für diese Darstellung.

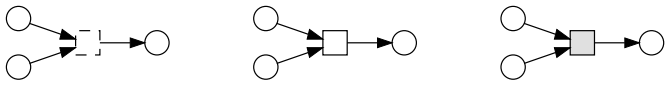


Abbildung 4.4: Trans. ohne Progress, mit Progress und mit Fairness

In Abb. 4.5 haben wir das Mutex-System um eine intuitiv plausible Lebendigkeitsannahme erweitert. So wird beispielsweise nicht gefordert, daß ein Philosoph der denkt tatsächlich einmal hungrig wird (gestrichelt dargestellte Transitionen $t_1$ und $t_4$). Dagegen verlangen wir durch die Progressannahme für $t_3$ bzw. $t_6$, daß jeder Philosoph irgendwann aufhört zu essen und somit die Gabel $g$ zurück gibt. Für die Transition $t_2$ und $t_5$ fordern wir Fairness. Mit dieser Lebendigkeitsannahme gelangt jeder hungrige Philosoph irgendwann zum Essen.

## 4.2 Systemkomponenten

Eine *Systemkomponente* ist ein Transitionssystem mit einer *Schnittstelle*, über die die Systemkomponente mit der Umgebung interagiert.

[3]Die Progressannahme ist gewissermaßen der Normalfall.

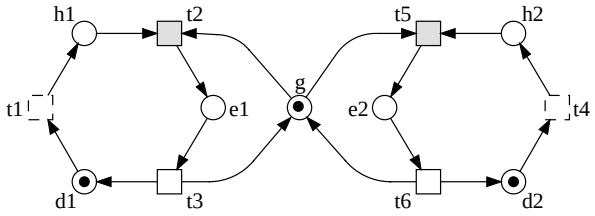



Abbildung 4.5: Mutex-System mit Lebendigkeitsannahmen

In diesem Abschnitt werden wir Systemkomponenten und ihr Verhalten formalisieren. Im nächsten Abschnitt werden wir dann zeigen, wie Systemkomponenten zusammengesetzt werden können.
Die Schnittstelle einer Systemkomponente zeichnet bestimmte Stellen des zugehörigen Transitionssystems als *Ein-* bzw. *Ausgabestellen* aus. Die Menge der Eingabestellen bezeichnen wir mit $I$ und die Menge der Ausgabestellen mit $O$. Die Systemkomponente kommuniziert entsprechend der „Richtung“ dieser Stellen mit der Umgebung. In Abb. 4.6 ist

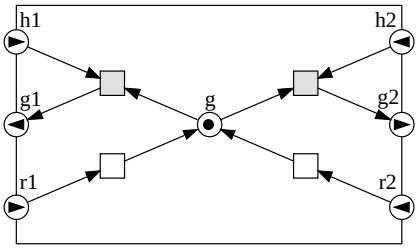



Abbildung 4.6: Die Mutex-Komponente

eine Systemkomponente mit einer Schnittstelle zur Umgebung dargestellt. Wie schon in der Einleitung erläutert handelt es sich bei dieser Systemkomponente um einen Verwalter, der die Gabel auf Anforderung einem der beiden Philosophen zuteilt. Die Pfeilspitzen in den Schnittstellen geben die Richtung der entsprechenden Kanäle an: Eine Pfeilspitze, die in die Systemkomponente hinein zeigt, kennzeichnet eine Eingabestelle; eine Pfeilspitze, die aus der Systemkomponente heraus zeigt, kennzeichnet eine Ausgabestelle. In obigem Beispiel sind $h_1$,

$r_1$, $h_2$ und $r_2$ die Eingabestellen und $g_1$ und $g_2$ die Ausgabestellen. Damit die Bezeichnung Ein- bzw. Ausgabestellen gerechtfertigt ist, darf keine Kante des Transitionssystems von einer Ausgabestelle wegführen und keine Kante zu einer Eingabestelle hinführen. Außerdem nehmen wir an, daß Schnittstellen initial unmarkiert sind[4].

**Definition 4.6 (Schnittstelle)**

Sei $\mathcal{T} = ((S'; T), M)$ ein verteiltes Transitionssystem mit Stellenmenge $S' \subseteq S$ und $Z, I, O \subseteq S'$ paarweise disjunkte Stellenmengen mit $Z \cup I \cup O = S'$.

Wir nennen $\mathcal{I} = (Z, I, O)$ eine *Schnittstelle für* $\mathcal{T}$, wenn $M \cap O = \emptyset = M \cap I$ und ${}^\bullet I = \emptyset = O^\bullet$ gilt.

Die Stellen aus $Z$, $I$, $O$ und $S \setminus S'$ nennen wir *interne Stellen*, *Eingabestellen*, *Ausgabestellen* bzw. *externe Stellen* der Schnittstelle $\mathcal{I}$.

Eine Systemkomponente ist ein verteiltes Transitionssystem zusammen mit einer Lebendigkeitsannahme und einer Schnittstelle.

**Definition 4.7 (Systemkomponente)**

Sei $\mathcal{T} = ((S'; T), M)$ ein verteiltes Transitionssystem mit Stellenmenge $S' \subseteq S$, $\mathcal{L} = (T^P, T^F)$ eine Lebendigkeitsannahme für $\mathcal{T}$ und $\mathcal{I} = (Z, I, O)$ eine Schnittstelle für $\mathcal{T}$.

Dann heißt $\Sigma = (\mathcal{I}, \mathcal{T}, \mathcal{L})$ eine *Systemkomponente* über $S$.

$\mathcal{I}$ nennen wir *Schnittstelle von* $\Sigma$, $\mathcal{T}$ nennen wir das *zugrundeliegende Transitionssystem von* $\Sigma$ und $\mathcal{L}$ die *Lebendigkeitsannahme von* $\Sigma$.

Da wir eine Systemkomponente häufig nur mit $\Sigma$ bezeichnen, ohne die Schnittstelle, das Transitionssystem und die Lebendigkeitsannahme explizit anzugeben, legen wir kanonische Bezeichnungen für die verschiedenen Elemente der Systemkomponente fest:

**Notation 4.8**

Für eine Systemkomponente $\Sigma = (\mathcal{I}, \mathcal{T}, \mathcal{L})$ über $S$ mit $\mathcal{I} = (Z, I, O)$, $\mathcal{T} = ((S'; T), M)$ und $\mathcal{L} = (T^P, T^F)$ bezeichnen wir

1. die Menge der internen Stellen mit $Z_\Sigma \mathrel{\widehat{=}} Z$, die Menge der Eingabestellen mit $I_\Sigma \mathrel{\widehat{=}} I$, die Menge der Ausgabestellen mit $O_\Sigma \mathrel{\widehat{=}} O$, die Menge der externen Stellen mit $S^U_\Sigma \mathrel{\widehat{=}} S \setminus (Z \cup I \cup O)$,

[4] Daher können wir die Richtung der Schnittstellen durch Pfeilspitzen in den Stellen darstellen.

2. die Anfangsmarkierung mit $M_\Sigma \mathrel{\widehat{=}} M$.
3. die Menge der *internen Transitionen* mit $T_\Sigma \mathrel{\widehat{=}} T$, die Progressmenge mit $T_\Sigma^P \mathrel{\widehat{=}} T^P$, die Fairnessmenge mit $T_\Sigma^F \mathrel{\widehat{=}} T^F$ und
4. die Menge der (potentiellen) *externen Transitionen* mit

$$T_\Sigma^U \mathrel{\widehat{=}} \{(S_1, S_2) \mid \quad S_1 \subseteq S_\Sigma^U \cup O_\Sigma, S_2 \subseteq S_\Sigma^U \cup I_\Sigma \text{ und } S_1, S_2 \text{ sind nicht-leere endliche Mengen}\}$$

Die Menge $T_\Sigma^U$ beschreibt die Menge der Transitionen, die in der Umgebung der Systemkomponente $\Sigma$ vorkommen können: Interne Stellen der Systemkomponente werden nicht berührt, Ausgabestellen von $\Sigma$ liegen nur im Vorbereich, Eingabestellen von $\Sigma$ nur im Nachbereich dieser Transitionen. Damit gilt $T_\Sigma^U \cap T_\Sigma = \emptyset$.
In einem Ablauf der Systemkomponente dürfen nur Ereignisse auftreten, die entweder einer Transition der Systemkomponente oder ihrer Umgebung entsprechen. Zusätzlich muß die Anfangsmarkierung bzgl. der internen Stellen mit denen der initialen Scheibe des Ablaufes übereinstimmen und die Lebendigkeitsannahmen müssen erfüllt sein.
Dabei fassen wir die Sicherheitseigenschaft $\rho(^\circ K) \cap Z_\Sigma \subseteq M_\Sigma$ und die Lebendigkeitseigenschaft $Init_M$ zur Vereinfachung zu $\rho(^\circ K) \cap Z_\Sigma = M_\Sigma$ zusammen.

**Definition 4.9 (Verhalten einer Systemkomponente)**

Sei $\Sigma$ eine Systemkomponente. Ein Ablauf $(K, \rho) \in \mathbf{R}(S)$ heißt Ablauf von $\Sigma$ genau dann, wenn die folgenden Bedingungen erfüllt sind:

1. $\rho(^\circ K) \cap Z_\Sigma = M_\Sigma$ und $\rho(^\circ K) \cap I_\Sigma = \emptyset = \rho(^\circ K) \cap O_\Sigma$
2. Für jedes Ereignis $e \in E$ des Ablaufes gilt $\rho(e) \in T_\Sigma$ oder $\rho(e) \in T_\Sigma^U$
3. $\rho \in Prog_{T_\Sigma^P}$
4. $\rho \in Fair_{T_\Sigma^F}$

Die Menge aller Abläufe von $\Sigma$ bezeichnen wir mit $\mathbf{R}(\Sigma)$.

Die Menge der *Abläufe der Systemkomponente* $\Sigma$ *in Isolation* definieren wir durch $\widehat{\mathbf{R}}(\Sigma) \mathrel{\widehat{=}} \mathbf{R}(\mathcal{T}_\Sigma) \cap Init_M \cap Prog_{T_\Sigma^P} \cap Fair_{T_\Sigma^F}$, wobei $\mathcal{T}_\Sigma$ das zugrundeliegende Transitionssystem von $\Sigma$ ist.

In Abb. 4.7 haben wir einen Ablauf der Mutex-Komponente aus Abb. 4.6 dargestellt. Zur Verdeutlichung haben wir die Schnittstelle

zwischen dem Verhalten der Systemkomponente und der Umgebung gestrichelt eingezeichnet. In der Mitte sind die Ereignisse dargestellt, die Transitionen des Systems entsprechen. Rechts und links sind Ereignisse der Umgebung dargestellt.
Dieser Ablauf zeigt, daß sich die Umgebung an keinerlei Konventionen halten muß (außer an die syntaktischen Einschränkungen durch die Schnittstelle): so gibt z.B. die „rechte“ Umgebung der Systemkomponente über $r_2$ eine Gabel zurück, bevor sie eine bekommen hat. Dies führt dazu, daß im System zwei Gabeln existieren. Im weiteren Ablauf führt dies dann dazu, daß sich sowohl die linke als auch die rechte Umgebung nebenläufig beim Essen $e_1$ bzw. $e_2$ aufhalten.

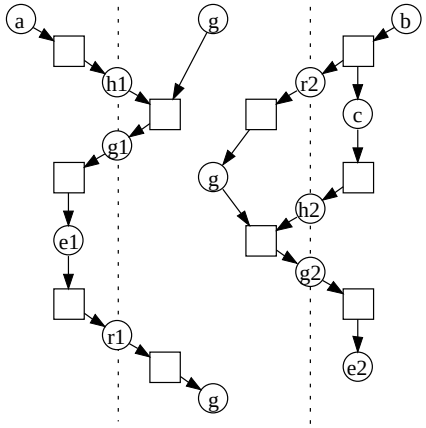



Abbildung 4.7: Ein Ablauf der Systemkomponente aus Abb. 4.6

Neben dem Verhalten $\mathbf{R}(\Sigma)$ der Systemkomponente haben wir in Def. 4.9 das Verhalten der Systemkomponente in Isolation $\widehat{\mathbf{R}}(\Sigma)$ definiert. Im Verhalten $\widehat{\mathbf{R}}(\Sigma)$ interagiert die Systemkomponente nicht mehr mit einer Umgebung. Es entspricht den Abläufen des zugrundeliegenden Transitionssystem mit den zusätzlichen Lebendigkeitsannahmen. In diesen Abläufen kommen keine Transitionen der Umgebung mehr vor und die initiale Scheibe entspricht genau der Anfangsmarkierung der Systemkomponente. Die Mutex-Komponente aus Abb. 4.6 besitzt in Isolation nur den leeren Ablauf. Dies ist typisch für solche Komponenten, die nur auf Anforderung der Umgebung bestimmte Dienste erbringen. Als isoliertes System betrachtet ist die Mutex-

Komponente nicht sinnvoll.

## 4.3 Komposition

Wir definieren nun einen Kompositionsoperator, mit dem wir zwei Systemkomponenten zu einer zusammensetzen können. Zwei Systemkomponenten werden an ihren gemeinsamen (entgegengesetzt gerichteten) Schnittstellen „zusammengeklebt“. Diese gemeinsamen Stellen werden im komponierten System zu internen Stellen. Dabei setzen wir voraus, daß die anderen Stellen der beiden Systemkomponenten disjunkt sind. Wir betrachten zunächst ein Beispiel. In Abb. 4.8 ist eine Systemkomponente als Umgebung für die Mutex-Komponente aus Abb. 4.6 dargestellt. Die Schnittstellen $h_1$, $g_1$ und $r_1$ sind jeweils entgegengesetzt zu den gleichnamigen Stellen der Mutex-Komponente; sonst haben die beiden Systemkomponenten keine gemeinsamen Stellen. Die beiden Systemkomponenten sind also komponierbar. Das Ergebnis der

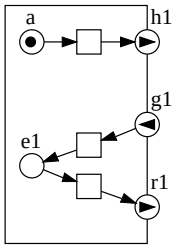



Abbildung 4.8: Eine Umgebung für die Mutex-Komponente

Komposition ist in Abb. 4.9 dargestellt. Bei der Komposition werden die gemeinsamen Schnittstellen $h_1$, $g_1$ und $r_1$ zu unmarkierten internen Stellen. Die Schnittstellen, die nur in einer der Systemkomponenten vorkommen, sind Schnittstellen des komponierten Systems. Das komponierte System kann über diese Schnittstellen mit weiteren Systemkomponenten komponiert werden. Die Stellen, die Anfangsmarkierung, die Transitionen, die Progress- und Fairnessmenge des komponierten Systems ergeben sich durch die Vereinigung der entsprechenden Mengen der beiden Komponenten.

**Definition 4.10 (Komponierbarkeit)**

Zwei Schnittstellen $\mathcal{I}_1 = (Z_1, I_1, O_1)$ und $\mathcal{I}_2 = (Z_2, I_2, O_2)$ heißen

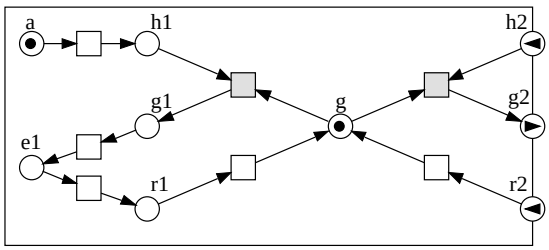



Abbildung 4.9: Ergebnis der Komposition

*komponierbar*, wenn $(Z_1 \cup I_1 \cup O_1) \cap (Z_2 \cup I_2 \cup O_2) = (I_1 \cap O_2) \cup (I_2 \cap O_2)$ gilt.

Wenn $\mathcal{I}_1$ und $\mathcal{I}_2$ komponierbar sind, definieren wir die Komposition $\mathcal{I}_1 \square \mathcal{I}_2$ der beiden Schnittstellen durch:

$$\mathcal{I}_1 \square \mathcal{I}_2 \mathrel{\widehat{=}} (Z_1 \cup Z_2 \cup (I_1 \cap O_2) \cup (I_2 \cap O_2), I_1 \setminus O_2 \cup I_2 \setminus O_1, O_1 \setminus I_2 \cup O_2 \setminus I_1)$$

Für zwei verteilte Transitionssysteme $\mathcal{T}_1 = ((S_1'; T_1), M_1)$ und $\mathcal{T}_2 = ((S_2'; T_2), M_2)$ definieren wir

$$\mathcal{T}_1 \square \mathcal{T}_2 \mathrel{\widehat{=}} ((S_1' \cup S_2'; T_1 \cup T_2), M_1 \cup M_2)$$

Für zwei Lebendigkeitsannahmen $\mathcal{L}_1 = (T_1^P, T_1^F)$ und $\mathcal{L}_1 = (T_2^P, T_2^F)$ definieren wir

$$\mathcal{L}_1 \square \mathcal{L}_2 \mathrel{\widehat{=}} (T_1^P \cup T_2^P, T_1^F \cup T_2^F)$$

Zwei Systemkomponenten sind komponierbar, wenn ihre Schnittstellen komponierbar sind. Die Komposition zweier komponierbarer Systemkomponenten ergibt sich durch die komponierte Schnittstelle und die Komposition der zugrundeliegenden verteilten Transitionssysteme und der Lebendigkeitsannahmen.

**Definition 4.11 (Komposition von Systemkomponenten)**

Zwei Systemkomponenten $\Sigma_1 = (\mathcal{I}_1, \mathcal{T}_1, \mathcal{L}_1)$ und $\Sigma_2 = (\mathcal{I}_2, \mathcal{T}_2, \mathcal{L}_2)$ heißen *komponierbar*, wenn die Schnittstellen $\mathcal{I}_1$ und $\mathcal{I}_2$ komponierbar sind.

Wenn $\Sigma_1$ und $\Sigma_2$ komponierbar sind, definieren wir die *Komposition* $\Sigma_1 \square \Sigma_2$ der beiden Systemkomponenten durch

$$\Sigma_1 \square \Sigma_2 \mathrel{\widehat{=}} (\mathcal{I}_1 \square \mathcal{I}_2, \mathcal{T}_1 \square \mathcal{T}_2, \mathcal{L}_1 \square \mathcal{L}_2)$$

Gemäß dieser Definition sind zwei Systemkomponenten nur komponierbar, wenn ihre internen Stellen disjunkt sind und die gemeinsamen Schnittstellen entgegengesetzt gerichtet sind. Sonst müssen wir zuvor eine geeignete Umbenennung durchführen (siehe Abschnitt 4.4).
Zunächst geben wir einige syntaktische Eigenschaften der Komposition an.

**Satz 4.12 (Komposition ist kommutativ und assoziativ)**

Seien $\Sigma_1, \Sigma_2$ und $\Sigma_3$ drei Systemkomponenten.

$\Sigma_1$ ist genau dann mit $\Sigma_2$ komponierbar, wenn $\Sigma_2$ mit $\Sigma_1$ komponierbar ist. Sind $\Sigma_1$ und $\Sigma_2$ komponierbar, dann gilt $\Sigma_1 \square \Sigma_2 = \Sigma_2 \square \Sigma_1$.

Ist $\Sigma_1$ mit $\Sigma_2$ und $\Sigma_1 \square \Sigma_2$ mit $\Sigma_3$ komponierbar, so auch $\Sigma_2$ mit $\Sigma_3$ und $\Sigma_1$ mit $\Sigma_2 \square \Sigma_3$ und es gilt $(\Sigma_1 \square \Sigma_2) \square \Sigma_3 = \Sigma_1 \square (\Sigma_2 \square \Sigma_3)$.

Wir kommen nun zu dem zentralen Satz, der den Zusammenhang zwischen dem Verhalten zweier komponierbarer Systemkomponenten und dem Verhalten ihrer Komposition herstellt. Dieser Satz ist die Grundlage für die modulare Spezifikations- und Verifikationsmethode, die wir später vorstellen.

**Satz 4.13 (Kompositionalität des Verhaltens)**

Seien $\Sigma_1$ und $\Sigma_2$ zwei miteinander komponierbare Systemkomponenten, dann gilt $\mathbf{R}(\Sigma_1 \square \Sigma_2) = \mathbf{R}(\Sigma_1) \cap \mathbf{R}(\Sigma_2)$.

**Beweis:** Folgt unmittelbar aus der Definition des Verhaltens einer Systemkomponente und des Kompositionsbegriffs. Sei $\Sigma = \Sigma_1 \square \Sigma_2$.

Wir zeigen die wesentliche Aussage $T_\Sigma \cup T_\Sigma^U = (T_{\Sigma_1} \cup T_{\Sigma_1}^U) \cap (T_{\Sigma_2} \cup T_{\Sigma_2}^U)$: Man überprüft leicht die Aussagen $T_\Sigma = T_{\Sigma_1} \cup T_{\Sigma_2}$, $T_{\Sigma_1} \cap T_{\Sigma_2} = \emptyset$, $T_{\Sigma_1} \cap T_{\Sigma_2}^U = T_{\Sigma_1}$, $T_{\Sigma_2} \cap T_{\Sigma_1}^U = T_{\Sigma_2}$ und $T_\Sigma^U = T_{\Sigma_1}^U \cap T_{\Sigma_2}^U$. Zusammen gilt $T_\Sigma \cup T_\Sigma^U = (T_{\Sigma_1} \cup T_{\Sigma_2}) \cup (T_{\Sigma_1}^U \cap T_{\Sigma_2}^U) = (T_{\Sigma_1} \cup T_{\Sigma_1}^U) \cap (T_{\Sigma_2} \cup T_{\Sigma_2}^U)$. Mit dieser Aussage erfüllt ein Ablauf $\rho$ die Bed. 2 der Ablaufdefinition (Def. 4.9) für $\Sigma_1$ und $\Sigma_2$ genau dann, wenn $\rho$ die Bed. 2 der Ablaufdefinition für $\Sigma$ erfüllt.

Ein Ablauf $\rho$ erfüllt die Bed. 1 der Ablaufdefinition für $\Sigma_1$ und $\Sigma_2$ genau dann, wenn $\rho$ diese Bedingung für $\Sigma$ erfüllt; dies folgt daraus, daß die Anfangsmarkierung die Vereinigung Anfangsmarkierungen der beiden Systemkomponenten ist und die Schnittstellen initial unmarkiert sind. Schnittstellen, die in dem komponierten System zu internen Stellen werden, sind ebenfalls unmarkiert.

Für Bed. 3. und 4. gilt diese Aussage analog, weil die Progress- bzw Fairnessmengen der beiden Systemkomponenten vereinigt werden. □

Wir werden nun noch zeigen, daß das Verhalten $\mathbf{R}(\Sigma)$ von Systemkomponenten nicht unnötig viele Unterscheidungen trifft. Dabei soll das Verhalten zumindest die Systemkomponenten unterscheiden, die sich als abgeschlossenes System unterschiedlich verhalten (Satz 4.14). Dafür haben wir $\widehat{\mathbf{R}}(\Sigma)$ als „Referenzverhalten“ definiert. Darüber hinaus soll das Verhalten $\mathbf{R}(\Sigma)$ zwei Systemkomponente nur dann unterscheiden, wenn dies für die Kompositionalität des Verhaltens notwendig ist. Formal bedeutet dies, daß das Verhalten *vollständig abstrahierend* bzgl. des Verhaltens in Isolation und der Komposition $\Box$ ist (Theorem 4.15).

**Satz 4.14 (Zusammenhang zum Verhalten in Isolation)**

Seien $\Sigma_1$ und $\Sigma_2$ zwei Systemkomponenten mit derselben Schnittstelle $\mathcal{I}_1 = \mathcal{I}_2 = (Z, I, O)$. Wenn gilt $\mathbf{R}(\Sigma_1) \neq \mathbf{R}(\Sigma_2)$, dann existiert eine mit $\Sigma_1$ und $\Sigma_2$ komponierbare Komponente $\Sigma$ mit $\widehat{\mathbf{R}}(\Sigma_1 \Box \Sigma) \neq \widehat{\mathbf{R}}(\Sigma_2 \Box \Sigma)$.

**Beweis:** Sei $\mathbf{R}(\Sigma_1) \neq \mathbf{R}(\Sigma_2)$. Dann existiert o.B.d.A. ein Ablauf $\rho \in \mathbf{R}(\Sigma_1) \setminus \mathbf{R}(\Sigma_2)$. Die externen Transitionen dieses Ablaufs falten wir zu einer Systemkomponente zusammen, die mit $\Sigma_1$ und $\Sigma_2$ komponierbar ist:

Wir definieren die Systemkomponente $\Sigma = (\mathcal{I}, \mathcal{T}, \mathcal{L})$ wobei $\mathcal{I} = (S^U_\Sigma, O, I)$, $\mathcal{T} = ((S \setminus Z; \{\rho(e) \mid e \in E \setminus T_{\Sigma_1}\}), \rho(^\circ K) \setminus M_{\Sigma_1})$ und $\mathcal{L} = (\emptyset, \emptyset)$. Dann gilt $\rho \in \mathbf{R}(\Sigma_1 \Box \Sigma)$. Wegen $\rho \notin \mathbf{R}(\Sigma_2)$ gilt mit Satz 4.13 $\rho \notin \mathbf{R}(\Sigma_2 \Box \Sigma)$. Also gilt $\mathbf{R}(\Sigma_1 \Box \Sigma) \neq \mathbf{R}(\Sigma_2 \Box \Sigma)$.

Gemäß der Definition von $\Sigma$ gilt $\widehat{\mathbf{R}}(\Sigma_1 \Box \Sigma) = \mathbf{R}(\Sigma_1 \Box \Sigma)$ und $\widehat{\mathbf{R}}(\Sigma_2 \Box \Sigma) = \mathbf{R}(\Sigma_2 \Box \Sigma)$. Zusammen gilt also $\widehat{\mathbf{R}}(\Sigma_1 \Box \Sigma) \neq \widehat{\mathbf{R}}(\Sigma_2 \Box \Sigma)$. □

Damit ist schon fast gezeigt, daß das Verhalten der Systemkomponenten vollständig abstrahierend bzgl. des Verhaltens der Systemkomponenten in Isolation ist. Im üblichen Sprachgebrauch sagt man dabei meist, daß eine *Semantik* vollständig abstrahierend bzgl. einer anderen Semantik ist. In diesem Sinne ordnen $\mathbf{R}(\Sigma)$ bzw. $\widehat{\mathbf{R}}(\Sigma)$ der Systemkomponente $\Sigma$ eine Semantik zu und die Abbildung $\mathbf{R}$ und $\widehat{\mathbf{R}}()$ können wir als Semantik auffassen.

**Theorem 4.15 (Vollständige Abstraktheit)**

$\mathbf{R}$ ist *vollständig abstrahierend* bezüglich $\widehat{\mathbf{R}}()$ und $\Box$. Das heißt:

1. $\mathbf{R}$ ist kompositional, d.h. für je zwei komponierbare Systemkomponenten $\Sigma_1$ und $\Sigma_2$ ergibt sich $\mathbf{R}(\Sigma_1 \Box \Sigma_2)$ aus $\mathbf{R}(\Sigma_1)$ und $\mathbf{R}(\Sigma_2)$.
2. Für je zwei Systemkomponenten $\Sigma_1$ und $\Sigma_2$ mit derselben Schnittstelle folgt aus $\mathbf{R}(\Sigma_1) = \mathbf{R}(\Sigma_2)$ auch $\widehat{\mathbf{R}}(\Sigma_1) = \widehat{\mathbf{R}}(\Sigma_2)$.
3. Für je zwei Systemkomponenten mit derselben Schnittstelle und $\mathbf{R}(\Sigma_1) \neq \mathbf{R}(\Sigma_2)$ existiert eine mit $\Sigma_1$ und $\Sigma_2$ komponierbare Systemkomponente $\Sigma$, so daß gilt $\widehat{\mathbf{R}}(\Sigma_1 \Box \Sigma) \neq \widehat{\mathbf{R}}(\Sigma_2 \Box \Sigma)$.

**Beweis:**

1. Satz 4.13
2. klar
3. Satz 4.14

□

Häufig werden im Zusammenhang der vollen Abstraktheit sehr einfache Semantiken als „Referenzsemantik“ betrachtet. Beispielsweise werden die Systeme nur in die Klasse der terminierenden und der nicht-terminierenden eingeteilt. Dann wird gezeigt, daß die zugehörige vollständig abstrahierende Semantik (bzgl. eines gegebenen Kompositionsoperators) sehr viel mehr Struktur besitzt.
Unserer „Referenzsemantik“ ist das Verhalten von Systemkomponenten in Isolation. Dieses Verhalten ist bereits sehr komplex. Die entsprechende vollständig abstrahierenden Semantik ist das Verhalten der Systemkomponenten (in einer beliebigen Umgebung). Diese Semantik unterscheidet fast alle Systemkomponenten[5]. Dies ist auch nicht besonders überraschend, da wir nicht a priori festlegen, welche Eigenschaft eines Systems relevant sind. Die relevante Eigenschaft wird erst durch eine Spezifikation festgelegt, die die Systemkomponenten wieder in zwei Klassen einteilt; nämlich solche, die die Spezifikation erfüllen, und solcher die sie nicht erfüllen. Da das Verhalten von Systemkomponenten bzgl. jeder beliebigen Spezifikation kompositional sein soll,

[5] Zwei verschiedene Systemkomponenten haben nur dann das gleiche Verhalten, wenn es in einer Systemkomponente eine tote Transition gibt, die in keinem Ablauf vorkommt.

muß das Verhalten fast alle Systemkomponenten unterscheiden.

Die Kompositionalität des Verhaltens werden wir später zur modularen Spezifikation und Verifikation von Systemkomponenten benutzen. Die volle Abstraktheit des Verhaltens der Systemkomponenten bzgl. des Verhaltens in Isolation zeigt darüber hinaus, daß sich das Verhalten von Systemkomponenten nur dann unterscheidet, wenn dies wegen der Kompositionalität erforderlich ist.

## 4.4 Konsistente Umbenennung

Gemäß Definition sind zwei Systemkomponenten nur komponierbar, wenn die Schnittstellen geeignet benannt sind. Beispielsweise sind die beiden Systemkomponenten $\Sigma_1$ und $\Sigma_2$ aus Abbildung 4.10 nicht komponierbar. Die Namen der Ein- und Ausgabestellen passen nicht zueinander und darüber hinaus kommen Namen von internen Stellen der einen Systemkomponente in der anderen vor.

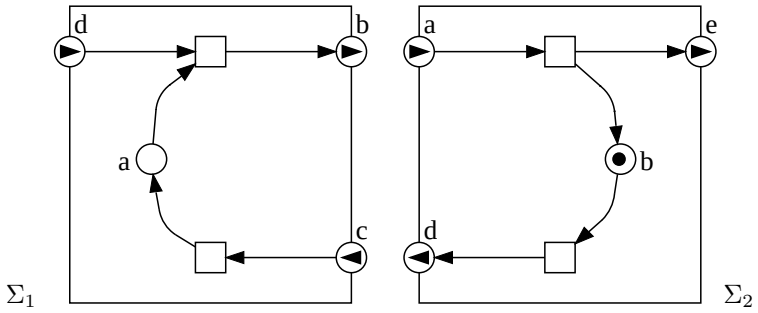



Abbildung 4.10: Zwei nicht komponierbare Systemkomponenten

Dennoch möchte wir die Systemkomponenten $\Sigma_1$ und $\Sigma_2$ wie angedeutet miteinander koppeln. Dazu müssen wir zunächst eine geeignete Umbenennung der Systemkomponenten durchführen.
In diesem Abschnitt formalisieren wir die Umbenennung von Systemkomponenten und zeigen, daß sich das Verhalten einer Systemkomponente nach der Umbenennung durch eine entsprechende Umbenennung des ursprünglichen Verhaltens ergibt (Abschnitt 4.4.1). Darüber hinaus zeigen wir, daß wir eine gewünschte Kopplung durch eine geeignete Umbenennung immer erreichen können (Abschnitt 4.4.2). Die Voraussetzung dafür ist, daß die Menge $S_\Sigma^U$ hinreichend groß (d.h. abzählbar

unendlich) ist.

**Definition 4.16 (Umbenennung)**

Sei $\Sigma$ eine Systemkomponente über $S$. Dann nennen wir eine bijektive Abbildung $u : S \to S'$ einen *Umbenennungsfunktion* für $\Sigma$.

Für eine Umbenennungsfunktion $u$ definieren wir die *Umbenennung* $u(\Sigma)$ über $S'$ wie folgt:

$$u(\Sigma) \mathrel{\widehat{=}} (u(\mathcal{I}_\Sigma), u(\mathcal{T}_\Sigma), u(\mathcal{L}_\Sigma))$$

wobei $u(\mathcal{I}_\Sigma)$, $u(\mathcal{T}_\Sigma)$ und $u(\mathcal{L}_\Sigma)$ wie folgt definiert sind:

- $u(\mathcal{I}_\Sigma) \mathrel{\widehat{=}} (u(Z_\Sigma), u(I_\Sigma), u(O_\Sigma))$,
- $u(\mathcal{T}_\Sigma) \mathrel{\widehat{=}} ((u(Z_\Sigma) \cup u(I_\Sigma) \cup u(O_\Sigma); u(T_\Sigma)), u(M_\Sigma))$ und
- $u(\mathcal{L}_\Sigma) \mathrel{\widehat{=}} (u(T_\Sigma^P), u(T_\Sigma^F))$.

Dabei gilt für $t \in T_\Sigma$: $u(t) \mathrel{\widehat{=}} (u({}^\bullet t), u(t^\bullet))$.

### 4.4.1 Umbenennung und Verhalten

Die Umbenennung $u(\Sigma)$ ist wieder eine Systemkomponente. Das Verhalten von $u(\Sigma)$ können wir einfach gewinnen, indem wir in jedem Ablauf die Beschriftungen entsprechend umbenennen. Gemäß unserer Konvention gilt $u(\mathbf{R}(\Sigma)) = \{u \circ \rho \mid \rho \in \mathbf{R}(\Sigma)\}$.

**Satz 4.17 (Umbenennung und Verhalten)**

Sei $\Sigma$ eine Systemkomponente über $S$ und $u : S \to S'$ eine Umbenennungsfunktion für $\Sigma$. Dann gilt $\mathbf{R}(u(\Sigma)) = u(\mathbf{R}(\Sigma))$.

**Beweis:** Man kann anhand der Definition der Abläufe einer Systemkomponente überprüfen, daß für jeden Ablauf $\rho \in \mathbf{R}(\Sigma)$ auch gilt $u \circ \rho \in \mathbf{R}(u(\Sigma))$. Daher gilt $\mathbf{R}(u(\Sigma)) \supseteq u(\mathbf{R}(\Sigma))$.

Die Umkehrabbildung $u^{-1} : S' \to S$ ist eine Umbenennungsfunktion für $u(\Sigma)$ und es gilt $u^{-1}(u(\Sigma)) = \Sigma$. Wenden wir obige Aussage für $u^{-1}$ an, so erhalten wir $\mathbf{R}(\Sigma) = \mathbf{R}(u^{-1}(u(\Sigma))) \supseteq u^{-1}(\mathbf{R}(u(\Sigma)))$. Also gilt $u(\mathbf{R}(\Sigma)) \supseteq u(u^{-1}(\mathbf{R}(u(\Sigma)))) = \mathbf{R}(u(\Sigma))$.

Zusammen gilt $\mathbf{R}(u(\Sigma)) = u(\mathbf{R}(\Sigma))$. □

Wir können somit das Verhalten der Systemkomponente nach der Umbenennung einfach aus dem Verhalten der ursprünglichen Systemkomponente gewinnen. Wenn wir später Eigenschaften durch temporallogische Aussagen repräsentieren, müssen wir die entsprechende

Umbenennung nur auf die temporallogische Aussage anwenden. Damit lassen sich alle für eine Systemkomponente bewiesenen temporalen Aussagen durch entsprechende Umbenennung auf die umbenannte Systemkomponente übertragen.

### 4.4.2 Existenz von Umbenennungsfunktionen

Wir zeigen nun, daß wir gewünschte Kopplungen von Systemkomponenten durch eine geeignete Umbenennung immer erreichen können. Dazu ist es notwendig, daß für die Umgebung immer genügend (d.h. abzählbar unendlich viele) Bezeichnungen „übrig“ bleiben. Dies hängt damit zusammen, daß die Umbenennungsfunktion bijektiv sein muß. Wenn wir die Surjektivität der Umbenennungsfunktion nicht fordern, stimmt Satz 4.17 nicht mehr. Dazu betrachten wir nochmals die Systemkomponente $\Sigma_1$ aus Abb. 4.10 und wählen $S = \{a, b, c, d\}$. Dann gilt $\mathcal{I}_{\Sigma_1} = (\{a\}, \{c, d\}, \{b\})$. Die Systemkomponente $\Sigma_1$ besitzt über dieser Menge $S$ nur den leeren Ablauf $\varepsilon$. Der Grund dafür ist, daß alle internen Stellen von $\Sigma$ initial unmarkiert sind und für die Umgebung von $\Sigma_1$ keine Bezeichnungen mehr übrig bleiben (d.h. $S^U_{\Sigma_1} = \emptyset$), die in der initialen Scheibe eines Ablaufes vorkommen können.

Wenn wir nun $S' = \{a, b, c, d, e\}$ und eine Umbenennungsfunktion $u : S \to S'$ mit $u(x) = x$ wählen, dann besitzt $u(\Sigma_1)$ einen Ablauf über $S'$, in dessen initialer Scheibe $e$ vorkommt. Dieser Ablauf liegt nicht in $u(\mathbf{R}(\Sigma)) = \{\varepsilon\}$. Es gilt also $u(\mathbf{R}(\Sigma)) \neq \mathbf{R}(u(\Sigma))$.

Der Grund für diesen Effekt ist also, daß die Schnittstelle der Systemkomponente $\Sigma_1$ zuwenig „Bewegungsfreiheit“ für die Umgebung läßt, indem sie die Menge der externen Stellen einschränkt (im Beispiel auf die leere Menge). Deshalb werden wir im folgenden immer annehmen, daß die Menge der externen Stellen $S^U_\Sigma$ einer Systemkomponente $\Sigma$ abzählbar unendlich ist. Unter dieser Annahme ist es immer möglich „gewünschte“ Kopplungen durch eine Umbenennung zu ermöglichen.

Um den Begriff der gewünschten Kopplung zu präzisieren, führen wir die *Kopplungsfunktion* ein, die bestimmte Schnittstellen zweier Systemkomponenten in Beziehung setzt. In unserem Beispiel aus Abb. 4.10 soll die Ausgabestelle $b$ von $\Sigma_1$ mit der Eingabestelle $a$ von $\Sigma_2$ und die Eingabestelle $c$ von $\Sigma_1$ mit der Ausgabestelle $d$ von $\Sigma_2$ gekoppelt werden. Dies wird formal durch die Kopplungsfunktion $f$ mit $f(b) = a$ und $f(c) = d$ ausgedrückt.

**Definition 4.18 (Kopplungsfunktion)**

Sei $\Sigma_1$ eine Systemkomponente über $S$ und $\Sigma_2$ eine Systemkomponente über $S'$ und $K \subseteq S$. Eine Abbildung $f : K \to S'$ heißt *Kopplungsfunktion*, wenn

1. $f$ injektiv ist,
2. $K \subseteq I_{\Sigma_1} \cup O_{\Sigma_1}$,
3. $f(K \cap I_{\Sigma_1}) \subseteq O_{\Sigma_2}$ und $f(K \cap O_{\Sigma_1}) \subseteq I_{\Sigma_2}$ gilt.

Wir werden nun zeigen, daß es für jede Kopplungsfunktion $f$ eine Umbenennungsfunktion gibt, die die gewünschten Schnittstellen zweier Systemkomponenten identifiziert und alle anderen Stellen verschieden macht. Nach der Umbenennung sind die beiden Systemkomponenten also in der gewünschten Weise komponierbar. Dazu nehmen wir an, daß die Menge der externen Stellen abzählbar unendlich ist. Außerdem zeigen wir, daß die Menge der externen Stellen der umbenannten und dann komponierten Systemkomponenten weiterhin abzählbar unendlich ist, so daß das komponierte System die Voraussetzung für weiteres Umbenennen und Komponieren erfüllt.

**Theorem 4.19**

Seien $\Sigma_1$ und $\Sigma_2$ zwei Systemkomponenten über $S$ bzw. $S'$ mit abzählbar unendlichen externen Stellenmengen $S^U_{\Sigma_1}$ und $S^U_{\Sigma_2}$. Für jede Kopplungsfunktion $f : K \to S'$ existiert eine Umbenennungsfunktion $u : S \to S'$, so daß gilt:

1. $u(\Sigma_1)$ und $\Sigma_2$ sind komponierbar,
2. $u|_K = f$,
3. $u(I_{\Sigma_1} \setminus K) \cap O_{\Sigma_2} = \emptyset$, $u(O_{\Sigma_1} \setminus K) \cap I_{\Sigma_2} = \emptyset$ und
4. $S^U_{u(\Sigma_1)\square\Sigma_2}$ ist abzählbar unendlich.

**Beweis:** Da wir angenommen haben, daß $S$ und $S'$ höchstens abzählbar sind, gilt mit der Voraussetzung des Satzes, daß $S$ und $S'$ abzählbar unendlich sind.

$u : S \to S'$ ist also eine Abbildung zwischen zwei abzählbar unendlichen Mengen. Wir stellen die Umbenennungsfunktion $u$ für eine gegebene Kopplungsfunktion $f$ in Abb. 4.11 graphisch dar. Die Umbenennungsfunktion $u$ bildet die Stellen aus $K$ entsprechend $f$ ab, die anderen Ein- bzw. Ausgabestellen von $\Sigma_1$ (d.h. $(I_{\Sigma_1} \cup O_{\Sigma_1}) \setminus K$) und die internen Stellen von $\Sigma_1$ werden auf externe Stellen von $\Sigma_2$

injektiv abgebildet. Da $(I_{\Sigma_1} \cup O_{\Sigma_1}) \setminus K$ und $Z_{\Sigma_1}$ höchstens abzählbar sind und $S^U_{\Sigma_2}$ abzählbar unendlich ist, können wir $u$ so wählen, daß dabei noch abzählbar unendlich viele Elemente von $S^U_{\Sigma_2}$ übrig bleiben.

Außerdem bildet $u$ eine Teilmenge der externen Stellen von $\Sigma_1$ injektiv auf die internen Stellen und die übrigen Schnittstellen von $\Sigma_2$ ab. Dabei kann $u$ wieder so gewählt werden, daß abzählbar unendlich viele externe Stellen von $\Sigma_1$ übrig bleiben.

Zwischen diesen übrigen abzählbar unendlich vielen externen Stellen der beiden Systemkomponenten wählen wir $u$ geeignet, so daß insgesamt eine bijektive Umbenennungsfunktion für $\Sigma_1$ entsteht.

Abbildung 4.11 verdeutlicht, daß $u$ alle Anforderungen aus dem Satz erfüllt. Dazu haben wir auf der rechten Seite auch die Schnittstellen, die internen und die externen Stellen des komponierten Systems $u(\Sigma_1)\square\Sigma_2$ dargestellt. □

Damit haben wir gezeigt, daß eine gewünschte Kopplung von zwei Systemkomponenten immer durch geeignete Umbenennung einer Systemkomponente erreicht werden kann. Da dafür $S^U_\Sigma$ abzählbar unendlich sein muß, setzen wir dies im folgenden immer voraus. Diese Einschränkung ist jedoch rein theoretischer Natur, da in der Praxis immer genügend Bezeichner für Stellen zur Verfügung stehen.
Weil sich das Verhalten der umbenannten Systemkomponente einfach aus dem ursprünglichen Verhalten ergibt, müssen wir beim Entwurf von Systemen nicht a priori auf eine geeignete Namensgebung der Schnittstellen achten. Bei Bedarf können wir die Stellen einer Systemkomponente geeignet umbenennen.

## 4.5 Literatur

Petrinetze sind „bekannt“ als ein nicht-kompositionales Systemmodell. Der Grund dafür sind die unzureichenden Strukturierungsmechanismen für Petrinetze. In der Literatur wurden einige Strukturierungsmechanismen für Petrinetze vorgeschlagen. Wir geben hier einige Beispiele für Ansätze an, die Teilnetze miteinander koppeln. Dabei kann man verschiedene Arten der Kopplung unterscheiden: Zum einen können bestimmte Transitionen von zwei Teilnetzen miteinander verschmolzen werden (z.B. [57, 20]); auf diese Weise wird synchrone Kommunikation modelliert. Zum anderen können bestimmte Stellen von Teilnetzen

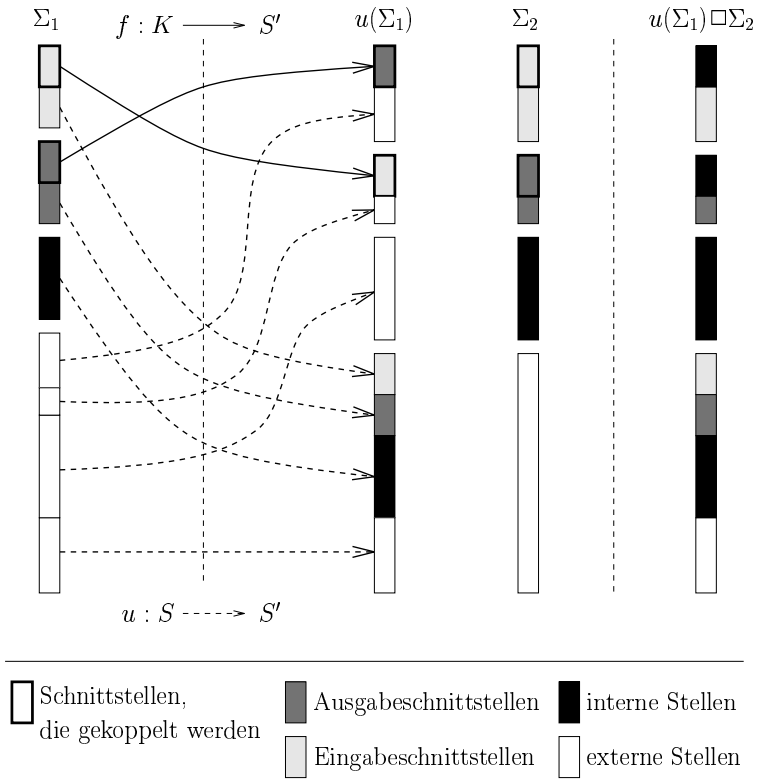



Abbildung 4.11: Umbenennung $u$ für eine Kopplungsfunktion $f$

verschmolzen werden (z.B. [84, 86]), was im wesentlichen der Kommunikation über gemeinsame Variablen entspricht. Nicht zuletzt können zwischen bestimmten Transitionen des einen Teilnetzes und Stellen des anderen Teilnetzes neue Kanten eingefügt werden (z.B. [10, 82, 83]). Dieser Ansatz entspricht im wesentlichen der asynchronen Kommunikation über Nachrichtenaustausch. Außerdem wird im Bereich der Petrinetze versucht die eleganten Strukturierungsmöglichkeiten von Prozeßalgebren wie z.B. CCS [61] auf Netzmodelle zu übertragen. Ein Beispiel für einen solchen Ansatz ist BPN$^2$ [12].
Darüber hinaus gibt es eine Reihe von Verfeinerungstechniken, die es erlauben, Teilnetze durch verfeinerte Netze zu ersetzen. [16] gibt einen Überblick über die verschiedenen Möglichkeiten der Verfeinerungstechniken. Die Verfeinerungstechniken zielen mehr auf die Abstraktion von Systemen als auf das Zusammensetzen von Teilsystemen.
Unser Ansatz zur Komposition von Systemen basiert auch auf der Verschmelzung von Stellen. Da wir aber nur Eingabe- mit Ausgabestellen verschmelzen, modelliert unser Ansatz — im Gegensatz zu [84, 86] — die asynchrone Kommunikation über unidirektionale Kanäle (vgl. [83]).

Viele Ansätze zur Strukturierung von Petrinetzen beschränken sich jedoch auf die syntaktische Definition eines Kompositionsoperators. Ihr Hauptziel ist es, große Petrinetze überschaubar darzustellen. Dagegen wird in [57] eine Halbordnungssemantik[6] für S/T-Systeme vorgeschlagen, die bezüglich Komposition durch Verschmelzung gleichbeschrifteter Transitionen kompositional ist. Dazu wird auf der semantischen Seite ein Synchronisationsoperator auf Halbordnungen eingeführt. Unsere Semantik ist stärker zustandsorientiert und hat den Vorteil, daß wir das Verhalten (die Semantik) des komponierten Systems als Durchschnitt des Verhaltens der Teilkomponenten erhalten. Das heißt insbesondere, daß eine Eigenschaft, die für eine Systemkomponente bewiesen wurde, in jeder Komposition mit einer (komponierbaren) Umgebung gilt. Die Komposition entspricht damit im wesentlichen der logischen Konjunktion von Eigenschaften.

Obwohl wir hier Petrinetze als formales Systemmodell verwenden, ist die hier vorgeschlagene Semantik eher von anderen Ansätzen beeinflußt. Im Verhalten einer Systemkomponente modellieren wir das beliebige Verhalten der Umgebung explizit. Es gibt eine Reihe von sequentiellen Verhaltensmodellen, die auf dieser Idee beruhen [9, 74, 1, 68]:

[6] Ein Ablauf ist eine Halbordnung von beschrifteten Ereignissen (*pomsets*).

Jeder Zustandsübergang in einem Ablauf entspricht entweder einem Übergang der Systemkomponente oder der Umgebung. Ob es sich um einen Übergang der Umgebung oder der Systemkomponente handelt, geht aus einer entsprechenden Beschriftung der Übergänge hervor.
Im sequentiellen Fall treten aber zwei Probleme auf: In einer Sequenz kann die Umgebung die Systemkomponente „ausspielen“ [1], indem nur sie Übergänge macht und die Systemkomponente nie zum Zug kommt. Dies muß im sequentiellen Fall explizit ausgeschlossen werden. Dieses Problem tritt bei uns nicht auf, da in einem verteilten Ablauf die Umgebung das Eintreten eines Ereignisses der Systemkomponente nicht verhindern kann (und umgekehrt). Das zweite Problem tritt beim iterierten Komponieren von Systemen auf. Eine einfache Beschriftung der Übergänge im Ablauf mit $u$ für „Umgebung“ und $s$ für „System“ reicht dann nicht aus, weil bzgl. des komponierten Systems diese Beschriftung nicht stimmt. Aus dem Ablauf geht nicht mehr hervor, welche Übergänge bzgl. der Komposition intern oder extern sind. Deshalb muß die Menge der Beschriftungen viel reichhaltiger sein (siehe z.B. [1]). Auch dieses Problem wird in unserem Ansatz vermieden. Bei uns ist anhand der Schnittstelle erkenntlich, ob ein Ereignis eines Ablaufs einer Transition der Umgebung oder der Systemkomponente entspricht. Die Ereignisse eines Ablaufs müssen deshalb nicht besonders gekennzeichnet werden.

Neben der Möglichkeit Systeme zu kombinieren haben wir Petrinetze um Lebendigkeitsannahmen erweitert. Dabei unterscheiden wir Progress- und Fairnessannahmen. Die Progressannahme entspricht fast der schwachen Fairness und die Fairnessannahme der starken Fairness [29].
Eine Lebendigkeitsannahme nennen wir nur dann eine echte Fairnessannahme, wenn sie die Entscheidung von Konflikten beeinflußt. Die Progressannahme ist also keine echte Fairnessannahme. Die Progressannahme gewährleistet lediglich, daß ein System nicht „unmotiviert“ stehen bleibt (vgl. Diskussion von Reisig [76]). Diese Unterscheidung wird von schwacher und starker Fairness nicht sauber getroffen. Obwohl die schwache Fairness eigentlich das gleiche Ziel verfolgt wie unsere Progressannahme, ist sie unter gewissen Umständen doch eine echte Fairnessannahme.
Die Lebendigkeitsannahme für eine Systemkomponente ist bzgl. der Ablaufmenge des zugrundeliegenden Transitionssystems maschinenabgeschlossen und erfüllt damit eine wesentliche Charakteristik von Fairnessannahmen [8, 52]. Unser Begriff von Fairness ist deutlich schwä-

cher als die Fairnessbegriffe aus [47, 65], die ebenfalls für Halbordnungen definiert sind. Diese Begriffe fordern Fairness auch für konfuse Situationen, so daß es sequentielle Beobachter des Ablaufs gibt, die die unfaire Behandlung einer Transition nicht erkennen. Unsere Fairnessannahme ist sogar schwächer als die Strong-Fairness aus [76]. Der Grund dafür ist, daß wir einen möglichst schwachen (echten) Fairnessbegriff zur Verfügung stellen wollen. So ist die Fairnessannahme bei der Umsetzung des Systemmodells in Hard- oder Software einfach realisierbar[7].

[7] z.B. durch geeignete Annahmen an Signallaufzeiten.

# Kapitel 5

# Spezifikation von Systemkomponenten

Im vorangegangenen Kapitel haben wir einen Formalismus zur Modellierung von Systemkomponenten kennengelernt. Wir werden nun Systemkomponenten spezifizieren.

Da das Verhalten einer Systemkomponente eine Menge von Abläufen ist, können wir jede Eigenschaft als *Spezifikation* ansehen. Eine Systemkomponente erfüllt diese Spezifikation, wenn die Menge ihrer Abläufe eine Teilmenge der Eigenschaft ist. Damit ist im Prinzip jeder Formalismus, in dem Eigenschaften syntaktisch repräsentiert werden können, zur Spezifikation von Systemkomponenten geeignet.

Wir wollen hier jedoch einen Formalismus einführen, der die Eigenschaften einer Systemkomponente in Abhängigkeit vom Verhalten der Umgebung beschreibt. Diese Art der Beschreibung des gewünschten Verhaltens wird häufig *Rely-Guarantee-Stil* [42] genannt. Wir repräsentieren die Abhängigkeit des Systemverhaltens vom Verhalten der Umgebung explizit in Form von *Rely-Guarantee-Graphen* (kurz: *RG-Graphen*). Diesem Ansatz liegt eine Idee von Pnueli [74] zugrunde, die im Gegensatz zu anderen Methoden [62, 1, 4, 2] auch Lebendigkeitsannahmen über die Umgebung zuläßt.

RG-Graphen werden zunächst unabhängig von einer speziellen syntaktischen Repräsentation für Eigenschaften eingeführt und erst in Kapitel 6 mit einer temporalen Logik für (halbgeordnete) Abläufe kombiniert. An Stelle dieser Logik kann jeder andere Formalismus zur Repräsentation von Eigenschaften benutzt werden. Unabhängig davon

welcher Formalismus zur Repräsentation der Eigenschaften verwendet wird, geben wir für RG-Graphen eine Kompositionsregel und weitere Regeln zur Vereinfachung an.
Einen RG-Graphen zusammen mit einer Schnittstelle nennen wir *Modul*. Module können wir mit Hilfe der Kompositionsregel — ähnlich wie Systemkomponenten — kombinieren und den resultierenden RG-Graph anschließend vereinfachen. Wenn wir jedes Modul einzeln durch eine Systemkomponente implementieren, so implementiert die Komposition der Einzelkomponenten das zusammengesetzte Modul. Wir können also ein Modul in Teilmodule zerlegen, die dann unabhängig voneinander implementiert werden können.

## 5.1 Module

Zunächst zeigen wir an einem Beispiel, wie wir das Zusammenspiel des Verhaltens einer Systemkomponente und ihrer Umgebung beschreiben. Dazu geben wir ein Modul zur Spezifikation der Mutex-Komponente an. Dieses Modul nennen wir im folgenden *Mutex-Modul*. Der Ablauf aus Abb. 4.7 hat gezeigt, daß die Mutex-Komponente aus Abb. 4.6 den wechselseitigen Ausschluß von $e_1$ und $e_2$ nur dann gewährleistet, wenn die Umgebung bestimmte Anforderungen erfüllt. Diese Anforderungen an die Umgebung müssen in der Verhaltensbeschreibung der Systemkomponenten festgeschrieben sein. Erfüllt die Umgebung die Anforderungen, so gewährleistet die Mutex-Komponente den wechselseitigen Ausschluß für die zwei Stellen $e_1$ und $e_2$, obwohl sie gar nicht unmittelbar auf diese Stellen zugreifen kann.
Das Mutex-Modul besteht aus einer Schnittstelle und einem RG-Graphen. Die Schnittstelle ist in Abb. 5.1 dargestellt. In dieser Darstellung sind nur die Ein- bzw. Ausgabestellen explizit erwähnt. Die internen Stellen sind nicht angegeben. Auch die externen Stellen sind nicht alle angegeben; wir geben nur an, daß $e_1$ und $e_2$ externe Stellen sind, so daß wir im RG-Graphen auf diese Stellen Bezug nehmen können. Die Namen der restlichen internen und externen Stellen sind unerheblich, weil wir sie durch geeignete Umbenennung später anpassen können. Abbildung 5.2 zeigt den RG-Graphen des Mutex-Moduls. Die Knoten des RG-Graphen sind mit Eigenschaften beschriftet. Zur Repräsentation dieser Eigenschaften bedienen wir uns in diesem Beispiel bereits der temporalen Logik, die wir erst später einführen. Wir geben die Bedeutung der temporalen Aussagen hier nur informell an. Eigenschaften, die von der Umgebung erwartet werden, sind in den auf der Spitze

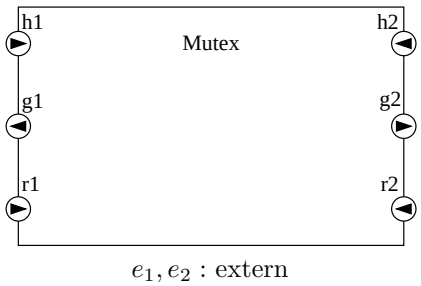



Abbildung 5.1: Die Schnittstelle des Mutex-Moduls

stehenden Rauten dargestellt. Eigenschaften, die vom Modul gewährleistet werden, sind in den Rechtecken dargestellt. Die Pfeile zwischen den Rauten und Rechtecken drücken aus, auf welche Eigenschaften der Umgebung sich das Modul abstützen darf, um die zugehörige Eigenschaft zu gewährleisten. Die temporalen Aussagen der Knoten 1 und 2 des RG-Graphen drücken aus, daß eine Anforderung auf $h_i$ jeweils zu (höchstens) einer Zuteilung einer Gabel auf $g_i$ führt. Wenn die Umgebung nur Gabeln an die Systemkomponente zurückgibt, die sie über $g_1$ und $g_2$ erhalten hat (ausgedrückt durch die temporalen Aussagen von Knoten 3 und 4), dann gewährleistet das Modul den wechselseitigen Ausschluß von $e_1$ und $e_2$ (Knoten 5). Wenn die Umgebung außerdem gewährleistet, daß mit den zugeteilten Gabeln irgendwann gegessen und der kritische Bereich wieder verlassen wird (Knoten 6 und 7), dann gewährleistet das Modul, daß jeder hungrige Philosoph irgendwann essen kann (Knoten 8).
Aufgrund der englischen Bezeichnungen *rely* und *guarantee* nennen wir die Rauten des RG-Graphen *R-Knoten* und die Rechtecke *G-Knoten*. Wir formalisieren nun RG-Graphen und ihre Bedeutung.

**Definition 5.1 (RG-Graph)**

Seien $G$ und $R$ zwei disjunkte Mengen, $\prec$ eine fundierte (strikte) Ordnung über $G \cup R$ und $l : R \cup G \to 2^{\mathbf{R}(S)}$ eine Abbildung in die Menge der Eigenschaften über $S$.

Dann nennen wir $\mathcal{RG} = (R, G, \prec, l)$ einen *RG-Graph* über $S$.

Graphisch repräsentieren wir — wie im vorangegangenen Beispiel — die R-Knoten eines RG-Graphen durch auf der Spitze stehende Rauten, die G-Knoten durch Rechtecke und die Ordnung $\prec$ durch Pfeile

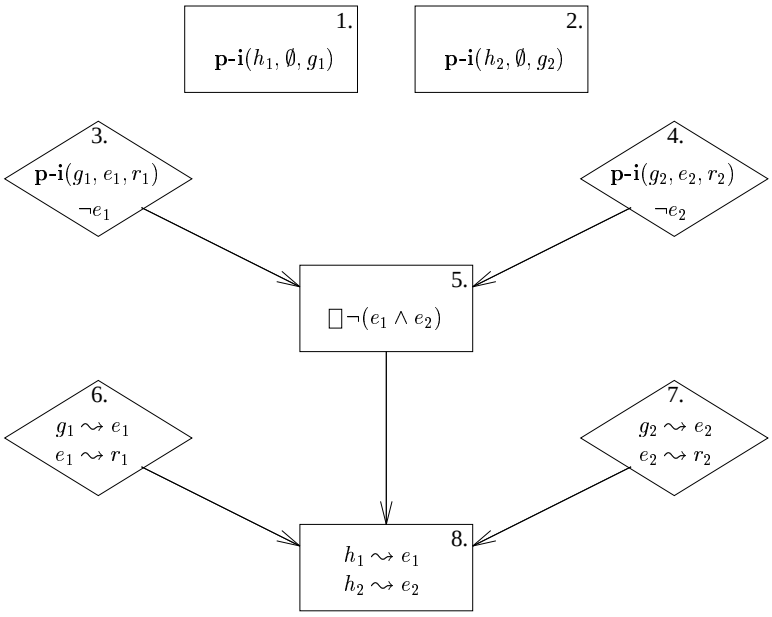



Abbildung 5.2: Der RG-Graph des Mutex-Moduls

zwischen den entsprechenden Knoten. Transitive Abhängigkeiten lassen wir der Übersichtlichkeit halber weg. Als Inschrift $l(x)$ eines Knotens des RG-Graphen lassen wir beliebige Eigenschaften (Mengen von Abläufen) zu. In unseren Beispielen repräsentieren wir Eigenschaften immer durch temporale Aussagen.
Eine Schnittstelle zusammen mit einem RG-Graphen nennen wir Modul.

**Definition 5.2 (Modul)**

Ein Paar $\mathcal{M} = (\mathcal{I}, \mathcal{RG})$, nennen wir *Modul über* $S$, wenn $\mathcal{I}$ eine Schnittstelle und $\mathcal{RG}$ ein RG-Graph über $S$ ist.

Eine Systemkomponente implementiert ein Modul, wenn sie für jeden G-Knoten $g$ die zugehörige Eigenschaft unter der Annahme erfüllt, daß alle Eigenschaften der Vorgängerknoten von $g$ gelten. Um dies formal zu definieren, führen wir zunächst einige Abkürzungen ein.

**Notation 5.3 (RG-Graph)**

Sei $\mathcal{RG} = (R, G, \prec, l)$ ein RG-Graph. Für einen Knoten $x \in R \cup G$ schreiben wir kurz $\mathcal{P}_x$ an Stelle von $l(x)$ und $\mathcal{P}_x^{\prec}$ an Stelle von $\bigcap_{y \prec x} \mathcal{P}_y$.

Für einen G-Knoten $g$ steht also $\mathcal{P}_g^{\prec}$ für die Eigenschaft, die aufgrund der Vorgängerknoten angenommen werden kann. $\mathcal{P}_g$ steht für die zu gewährleistende Eigenschaft. Die Aussage $\mathbf{R}(\Sigma) \cap \mathcal{P}_g^{\prec} \subseteq \mathcal{P}_g$ bedeutet dann „in $\Sigma$ gilt $\mathcal{P}_g$ unter Annahme $\mathcal{P}_g^{\prec}$“. Damit definieren wir nun die Implementierung eines RG-Graphen.

**Definition 5.4 (Implementierung eines Moduls)**

Sei $\mathcal{M} = (\mathcal{I}, \mathcal{RG})$ ein Modul mit RG-Graph $\mathcal{RG} = (R, G, \prec, l)$. Eine Systemkomponente $\Sigma$ mit Schnittstelle $\mathcal{I}$ *implementiert* Modul $\mathcal{M}$, wenn für jedes $g \in G$ gilt $\mathbf{R}(\Sigma) \cap \mathcal{P}_g^{\prec} \subseteq \mathcal{P}_g$.

Wir schreiben dafür auch $\Sigma \models \mathcal{M}$ und nennen $\Sigma$ eine *Implementierung* von $\mathcal{M}$.

Beispielsweise implementiert die Mutex-Komponente aus Abb. 4.1 das Mutex-Modul mit der Schnittstelle aus Abb. 5.1 und dem RG-Graphen aus Abb. 5.2. Dies werden wir später beweisen.

## 5.2 Rechnen mit Modulen

Wir werden nun „Rechenregeln“ angeben, die es erlauben Module zu kombinieren und anschließend zu vereinfachen. So können wir Module zu größeren Modulen zusammensetzen, ohne ihre Implementierung zu kennen. Die einzelnen Module können wir unabhängig voneinander implementieren und die Kombination dieser Einzelimplementierungen ist dann eine Implementierung des zusammengesetzten Moduls. Insbesondere können wir eine Implementierung eines Teilmoduls gegen eine andere Implementierung austauschen.
Zunächst definieren wir die Komposition zweier Module. Die Komposition zweier Module entspricht der Komposition der beiden Schnittstellen und dem „unabhängigen Nebeneinanderlegen“ der beiden RG-Graphen.

**Definition 5.5 (Komposition von Modulen)**

Seien $\mathcal{M}_1 = (\mathcal{I}_1, \mathcal{RG}_1)$ und $\mathcal{M}_2 = (\mathcal{I}_2, \mathcal{RG}_2)$ zwei Module über $S$ und seien o.B.d.A. die Knotenmengen $(R_1 \cup G_1)$ und $(R_2 \cup G_2)$ von $\mathcal{RG}_1$ bzw. $\mathcal{RG}_2$ disjunkt.

Wenn die Schnittstellen $\mathcal{I}_1$ und $\mathcal{I}_2$ komponierbar sind, dann heißen $\mathcal{M}_1$ und $\mathcal{M}_2$ *komponierbar*. Die *Komposition* von $\mathcal{M}_1$ und $\mathcal{M}_2$ definieren wir durch

$$\mathcal{M}_1 \square \mathcal{M}_2 \mathrel{\widehat{=}} (\mathcal{I}_1 \square \mathcal{I}_2, (R_1 \cup R_2, G_1 \cup G_2, \prec_1 \cup \prec_2, l_1 \cup l_2))$$

Die folgende *Kompositionsregel* formalisiert die obige Aussage: Die Komposition einer Implementierung $\Sigma_1$ von $\mathcal{M}_1$ und einer Implementierung $\Sigma_2$ von $\mathcal{M}_2$ ist eine Implementierung für das komponierte Modul $\mathcal{M}_1 \square \mathcal{M}_2$. Diese Regel ist eine unmittelbare Konsequenz der Kompositionalität des Verhaltens der Systemkomponenten.

**Satz 5.6 (Kompositionsregel)**

Seien $\mathcal{M}_1$ bzw. $\mathcal{M}_2$ zwei komponierbare Module und $\Sigma_1$ und $\Sigma_2$ zwei Systemkomponenten. Wenn $\Sigma_1$ eine Implementierung von $\mathcal{M}_1$ und $\Sigma_2$ eine Implementierung von $\mathcal{M}_2$ ist, dann sind $\Sigma_1$ und $\Sigma_2$ komponierbar und $\Sigma_1 \square \Sigma_2$ ist eine Implementierung von $\mathcal{M}_1 \square \mathcal{M}_2$.

**Beweis:** Die Komponierbarkeit von $\Sigma_1$ und $\Sigma_2$ folgt unmittelbar aus der Komponierbarkeit der Schnittstellen.

$\Sigma_1 \square \Sigma_2 \models \mathcal{M}_1 \square \mathcal{M}_2$ folgt unmittelbar aus Satz 4.13. □

Wenn wir ein Modul mit einer „intendierten“ Umgebung komponieren können wir die Annahmen des Moduls über die Umgebung eliminieren. Die Umgebung gewährleistet dann ja diese Annahmen. Im komponierten Modul bedeutet dies formal, daß es G-Knoten und R-Knoten gibt, die gleich beschriftet sind. Der R-Knoten kann dann eliminiert werden. Dabei müssen wir darauf achten, daß keine zyklischen Abhängigkeiten entstehen. Deshalb führen wir die Elimination in zwei Schritten durch: Im ersten Schritt wird eine zusätzliche Abhängigkeit zwischen den betreffenden G-Knoten und R-Knoten eingeführt, wobei die neue Relation $\prec$ weiterhin eine fundierte Ordnung sein muß. Anschließend eliminieren wir einen R-Knoten $r$, wenn seine Eigenschaft bereits aus den Eigenschaften der Vorgängerknoten folgt (d.h. $\mathcal{P}_r^{\prec} \subseteq \mathcal{P}_r$).
Wir formalisieren zunächst die Regeln und zeigen in Abschnitt 5.3, wie diese Regeln benutzt werden können, um R-Knoten in einem komponierten Modul zu eliminieren. Die erste Regel erlaubt das Hinzufügen von weiteren Abhängigkeiten in den RG-Graph.

**Satz 5.7 (Hinzufügen von Kanten)**

Seien $\mathcal{M}_1 = (\mathcal{I}, (R, G, \prec_1, l))$ und $\mathcal{M}_2 = (\mathcal{I}, (R, G, \prec_2, l))$ zwei Module. Wenn $\prec_1 \subseteq \prec_2$ gilt, dann ist jede Implementierung von $\mathcal{M}_1$ auch eine Implementierung von $\mathcal{M}_2$.

**Beweis:** Folgt unmittelbar aus der Def. 5.4, denn wegen $\prec_1 \subseteq \prec_2$ gilt für jeden Knoten $x \in R \cup G$: $\mathcal{P}_x^{\prec_2} \subseteq \mathcal{P}_x^{\prec_1}$. □

Wir geben nun zwei Regeln zum Eliminieren von Knoten aus dem RG-Graphen an. Zunächst definieren wir die Elimination eines Knotens $x$ aus dem RG-Graphen eines Moduls $\mathcal{M}$, wie man dies intuitiv erwartet. Wir schreiben dafür $\mathcal{M} \setminus x$. Anschließend formulieren wir damit die beiden Regeln.

**Definition 5.8 (Elimination eines Knotens eines Modul)**

Sei $\mathcal{M} = (\mathcal{I}, (R, G, \prec, l))$ ein Modul und $x \in R \cup G$. Wir definieren das Modul $\mathcal{M} \setminus x$ durch

$$\mathcal{M} \setminus x \mathrel{\widehat{=}} (\mathcal{I}, (R \setminus \{x\}, G \setminus \{x\}, \prec |_{(R \cup G) \setminus \{x\}}, l|_{(R \cup G) \setminus \{x\}}))$$

Ein G-Knoten $g$ eines RG-Graphen kann jederzeit weggelassen werden, weil damit nur Anforderungen wegfallen.

**Satz 5.9 (Elimination eines G-Knotens)**

Sei $\mathcal{M}$ ein Modul und $g \in G$ ein G-Knoten des zugehörigen RG-Graphen. Jede Implementierung von $\mathcal{M}$ ist auch eine Implementierung von $\mathcal{M} \setminus g$.

**Beweis:** Sei $\Sigma$ eine Implementierung von $\mathcal{M}$. Wir zeigen, daß $\Sigma$ auch $\mathcal{M} \setminus g$ erfüllt, wobei wir die Abhängigkeitsrelation von $\mathcal{M} \setminus g$ mit $\prec'$ bezeichnen. Für den Knoten $g$ ist in $\mathcal{M} \setminus g$ nichts mehr zu zeigen. Für die G-Knoten $g'$ mit $g \not\prec g'$ gilt $\mathcal{P}_{g'}^{\prec} = \mathcal{P}_{g'}^{\prec'}$ und damit folgt die Behauptung. Für die anderen G-Knoten $g'$ mit $g \prec g'$ gilt $\mathcal{P}_{g'}^{\prec'} = \mathcal{P}_{g'}^{\prec'} \cap \mathcal{P}_g^{\prec}$ und $\mathcal{P}_{g'}^{\prec} = \mathcal{P}_{g'}^{\prec'} \cap \mathcal{P}_g$. Weil $\Sigma$ Implementierung von $\mathcal{M}$ ist gilt $\mathbf{R}(\Sigma) \cap \mathcal{P}_g^{\prec} \subseteq \mathcal{P}_g$ und $\mathbf{R}(\Sigma) \cap \mathcal{P}_{g'}^{\prec} \subseteq \mathcal{P}_{g'}$. Zusammen gilt $\mathbf{R}(\Sigma) \cap \mathcal{P}_{g'}^{\prec'} = \mathbf{R}(\Sigma) \cap (\mathcal{P}_{g'}^{\prec'} \cap \mathcal{P}_g^{\prec}) = \mathbf{R}(\Sigma) \cap \mathcal{P}_{g'}^{\prec'} \cap (\mathbf{R}(\Sigma) \cap \mathcal{P}_g^{\prec}) \subseteq \mathbf{R}(\Sigma) \cap \mathcal{P}_{g'}^{\prec'} \cap \mathcal{P}_g = \mathbf{R}(\Sigma) \cap \mathcal{P}_{g'}^{\prec} \subseteq \mathcal{P}_{g'}$. □

Ein R-Knoten kann dagegen nur weggelassen werden, wenn die Eigenschaften der Knoten, die davor liegen, die Eigenschaft des R-Knotens implizieren.

**Satz 5.10 (Elimination eines R-Knotens)**

Sei $\mathcal{M}$ ein Modul und $r$ ein R-Knoten des RG-Graphen des Moduls mit $\mathcal{P}_r^{\prec} \subseteq \mathcal{P}_r$.

Jede Implementierung des Moduls $\mathcal{M}$ ist auch Implementierung von $\mathcal{M} \setminus r$.

**Beweis:** Folgt unmittelbar aus der Def. 5.4. □

Darüber hinaus können wir R-Knoten weglassen, auf die sich kein G-Knoten abstützt. Diese Knoten sind im RG-Graphen überflüssig.

**Satz 5.11 (Elimination überflüssiger R-Knotens)**

Sei $\mathcal{M}$ ein Modul und $r$ ein R-Knoten des RG-Graphen des Moduls. Wenn in dem RG-Graphen von $\mathcal{M}$ kein G-Knoten $g$ mit $r \prec g$ existiert, dann ist jede Implementierung von $\mathcal{M}$ auch eine Implementierung von $\mathcal{M} \setminus r$.

Zuletzt geben wir Regel zum Abschwächen von RG-Graphen durch Modifikation der Inschriften der Knoten an. Die Abschwächung bedeutet für G-Knoten eine Abschwächung der entsprechenden Inschrift

(d.h. $\mathcal{P}_g$ wird durch $\mathcal{P}'_g \supseteq \mathcal{P}_g$ ersetzt). Für R-Knoten bedeutet die Abschwächung dagegen eine Verschärfung der entsprechenden Inschrift: Das Modul nimmt von der Umgebung mehr an und ist damit schwächer. Wir formulieren die Regeln etwas allgemeiner, indem wir die Eigenschaften der Vorgängerknoten bei der Abschwächung berücksichtigen.

**Satz 5.12 (Abschwächung eines G-Knotens)**

Seien $\mathcal{M}_1 = (\mathcal{I}, (R, G, \prec, l_1))$ und $\mathcal{M}_2 = (\mathcal{I}, (R, G, \prec, l_2))$ zwei Module und $g \in G$.

Wenn für jeden Knoten $x \in R \cup G$ mit $x \neq g$ gilt $l_1(x) = l_2(x)$ und $l_1(g) \cap \mathcal{P}_g^{\prec} \subseteq l_2(g)$, dann ist jede Implementierung von $\mathcal{M}_1$ auch eine Implementierung von $\mathcal{M}_1$.

**Beweis:** Der Beweis ist sehr ähnlich wie der Beweis von Satz 5.8. □

Satz 5.12 können wir als Verallgemeinerung von Satz 5.9 auffassen. Das Weglassen eines Knotens entspricht im wesentlichen dem Ersetzen einer Inschrift eines $G$-Knotens durch $\mathbf{R}(S)$. Dies ist ein Spezialfall von Satz 5.12
Wie oben angedeutet bedeutet die Abschwächung für R-Knoten eine Verschärfung der entsprechenden Inschrift. Auch hier berücksichtigen wir die Eigenschaften der Vorgängerknoten.

**Satz 5.13 (Verschärfung eines R-Knotens)**

Seien $\mathcal{M}_1 = (\mathcal{I}, (R, G, \prec, l_1))$ und $\mathcal{M}_2 = (\mathcal{I}, (R, G, \prec, l_2))$ zwei Module und $r \in R$.

Wenn für jeden Knoten $x \in R \cup G$ mit $x \neq r$ gilt $l_1(x) = l_2(x)$ und $\mathcal{P}_r^{\prec} \cap l_2(r) \subseteq l_1(r)$, dann ist jede Implementierung von $\mathcal{M}_1$ auch eine Implementierung von $\mathcal{M}_1$.

**Beweis:** Folgt unmittelbar aus der Def. 5.4. □

Satz 5.13 können wir als eine Verallgemeinerung von Satz 5.10 interpretieren. Knoten (sowohl R- als auch G-Knoten) mit der Inschrift $\mathbf{R}(S)$ können wir immer aus einem RG-Graphen entfernen. Wenn die Voraussetzung für das Eliminieren eines R-Knotens nach Satz 5.10 gegeben sind, dann können wir mit Satz 5.13 die Inschrift dieses Knotens in $\mathbf{R}(S)$ ändern. Es hätte also gereicht, die Sätze 5.12 und 5.13 als Regeln zusammen mit der Regel zur Elimination von Knoten mit

der Inschrift $\mathbf{R}(S)$ anzugeben. Die anderen beiden Regeln sind dann ableitbar.
Manchmal ist es zweckmäßig zusätzliche Knoten in einen RG-Graphen einzufügen. Wir geben dafür wieder eine Regel für G-Knoten und eine für R-Knoten an. Eingefügte G-Knoten müssen mit $\mathbf{R}(S)$ beschriftet sein, so daß sich die Menge der Implementierungen durch das Einfügen des Knotens nicht verändert.

**Satz 5.14 (Einfügen eines G-Knotens)**

Sei $\mathcal{M}$ ein Modul und $g$ ein G-Knoten des zugehörigen RG-Graphen mit Inschrift $l(x) = \mathbf{R}(S)$. Jede Implementierung von $\mathcal{M} \setminus g$ ist eine Implementierung von $\mathcal{M}$.

Die Beschriftung eines eingefügten R-Knoten ist im Gegensatz dazu beliebig.

**Satz 5.15 (Einfügen eines R-Knotens)** Sei $\mathcal{M}$ ein Modul und $r$ ein R-Knoten des zugehörigen RG-Graphen. Jede Implementierung von $\mathcal{M} \setminus r$ ist eine Implementierung von $\mathcal{M}$.

## 5.3 Ein Beispiel

Wir werden nun die Anwendung der obigen Regeln anhand eines kleinen Beispiels demonstrieren. Dazu betrachten wir ein Modul $\mathcal{M}_l$ als „vernünftige“ Umgebung an der linken Schnittstelle des Mutex-Moduls. Der RG-Graph von $\mathcal{M}_l$ ist „symmetrisch“ zu den Anforderungen des Mutex-Moduls. Die Schnittstelle und der RG-Graph von $\mathcal{M}_l$ sind in Abb. 5.3 angegeben. Der RG-Graph der Komposition der beiden Module ist in Abb. 5.4 dargestellt.
Den RG-Graph des komponierten Moduls werden wir nun vereinfachen. Zunächst fügen wir zwischen den korrespondierenden R- und G-Knoten der beiden Teilgraphen neue Kanten ein, was nach Satz 5.7 zulässig ist, und erhalten damit den RG-Graphen aus Abb. 5.5. In diesem Graphen sind nunmehr die R-Knoten 3, 6 und 10 überflüssig, da vor ihnen bereits ein G-Knoten liegt, der mit derselben Eigenschaft beschriftet ist. Mit der Eliminationsregel (Satz 5.10) können wir diese R-Knoten weglassen und wir erhalten den RG-Graphen aus Abb. 5.6.
Dieser RG-Graph enthält einige G-Knoten mit Eigenschaften, die uns im weiteren nicht interessieren. Diese Knoten lassen wir ebenfalls weg und erhalten den RG-Graphen aus Abb. 5.7. Für diesen Schritt ist natürlich wesentlich, welche Eigenschaften später noch von Interesse

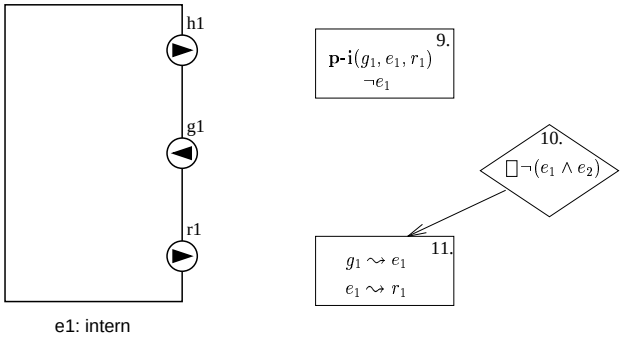



Abbildung 5.3: Modul als linke Umgebung des Mutex-Moduls

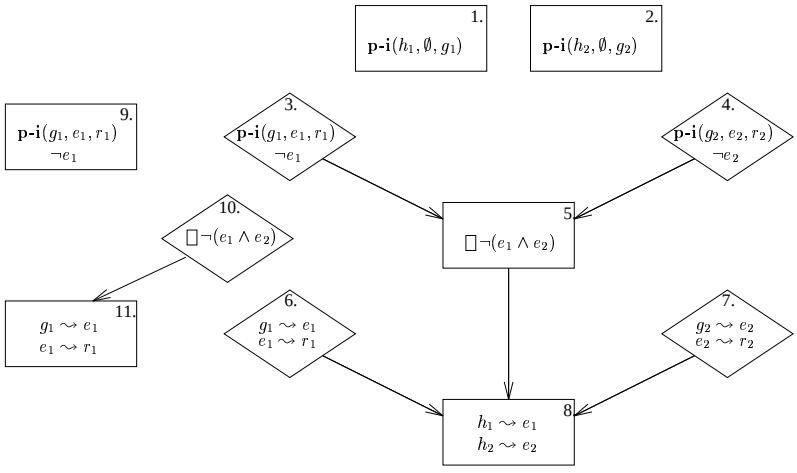



Abbildung 5.4: RG-Graph des komponierten Moduls

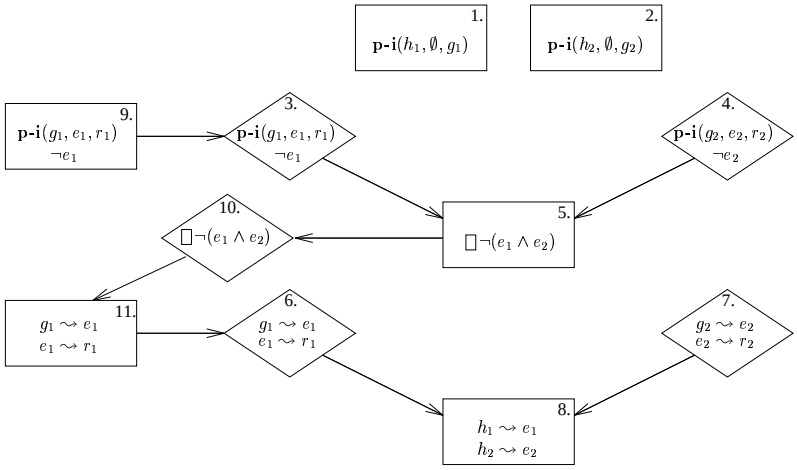



Abbildung 5.5: Hinzufügen neuer Kanten

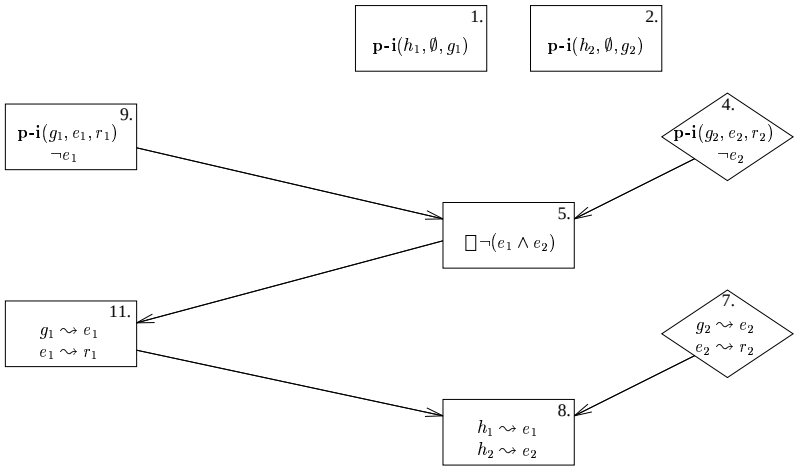



Abbildung 5.6: Entfernen der überflüssigen R-Knoten

sind und welche nicht. Im Beispiel haben wir nur die beibehalten, die im Zusammenhang mit dem wechselseitigen Ausschluß stehen. In anderen Fällen können durchaus andere Eigenschaften wegfallen. Insgesamt

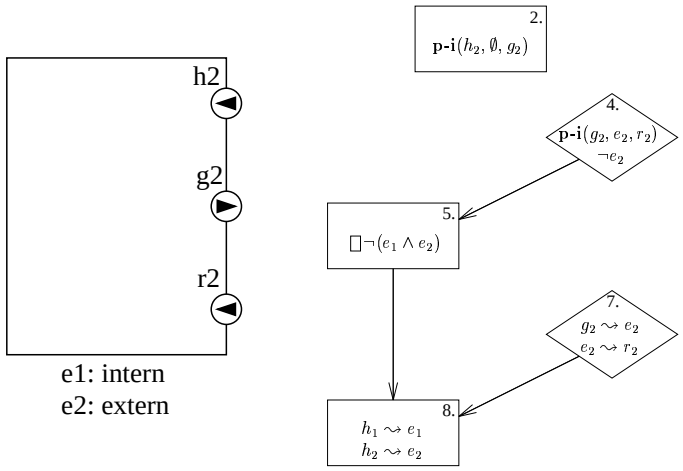



Abbildung 5.7: Resultierendes Modul

haben wir (weil wir die Regeln zur Modifikation des RG-Graphen angewendet haben) gezeigt, daß jede Implementierung des Mutex-Moduls und der linken Mutex-Umgebung komponiert das Modul aus Abb. 5.7 implementieren.

Wir können nun auch noch für die rechte Umgebung ein analoges Modul angeben und den entsprechenden RG-Graphen wieder analog vereinfachen. Ohne dies hier im einzelnen durchzuführen, erhalten wir dann den RG-Graphen aus Abb. 5.8, in dem kein R-Knoten mehr enthalten ist. Die Mutex-Eigenschaften gelten jetzt ohne Annahmen über Eigenschaften der Umgebung.

Allgemein erfüllt eine Implementierung eines RG-Graphen, der nur G-Knoten enthält, alle Eigenschaften der G-Knoten. Damit dies gilt, haben wir explizit die Fundiertheit der Ordnung $\prec$ des RG-Graphen gefordert.

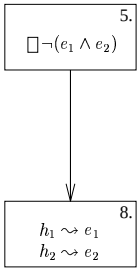



Abbildung 5.8: RG-Graph für Gesamtsystem mit leerer Schnittstelle

## 5.4 Die Frage der Vollständigkeit

Wir haben eine Reihe von Regeln zur Modifikation von Modulen angegeben. Die Regeln sind *korrekt*. Das heißt, jede Implementierung eines Moduls ist auch eine Implementierung des modifizierten Moduls. Bei manchen Regeln kommen durch die Modifikation neue Implementierungen hinzu.
Eine naheliegende Frage ist nun, ob die angegebenen Regeln *vollständig* sind. Das heißt, ob sich zwei Module, die die gleiche Menge von Implementierungen besitzen unter Anwendung der angegebenen Regeln in endlich vielen Schritten ineinander überführen lassen. Diese Frage müssen wir selbst für endliche[1] RG-Graphen negativ beantworten. Beispielsweise besitzen für zwei nicht-triviale Eigenschaften $\mathcal{R}$ und $\mathcal{G}$ die RG-Graphen aus Abb. 5.9 dieselbe Implementierungsmenge (wobei wir für $\neg\mathcal{R} \cup \mathcal{G}$ suggestiv $\mathcal{R} \Rightarrow \mathcal{G}$ schreiben). Der linke RG-Graph läßt sich aber nicht in den rechten überführen, da es nicht möglich ist, den R-Knoten zu eliminieren, ohne zuvor den G-Knoten wegzulassen. Allerdings können wir den rechten RG-Graphen in den linken umformen.

Es ist aber keine Schwierigkeit, einen Satz von Regeln anzugeben, der vollständig ist. Dazu müssen wir nur die Regel aus Satz 5.11 etwas verallgemeinern. Ein R-Knoten $r$, dessen Eigenschaft in jedem nachfolgenden G-Knoten $g$ nicht benötigt wird, kann ebenfalls eliminiert

[1] Für unendliche RG-Graphen ist die Unvollständigkeit nicht überraschend, da bei jeder Regelanwendung nur ein Knoten eliminiert werden kann. Es gibt ein Modul mit unendlichem RG-Graphen, das die gleichen Implementierungen besitzt wie ein Modul mit einem endlichen RG-Graphen. Diese Module lassen sich nicht in endlich vielen Schritten ineinander überführen.


Abbildung 5.9: Unvollständigkeit der Regeln

werden. Formal können wir das durch die Forderung $\neg\mathcal{P}_r \subseteq \mathcal{P}_g$ ausdrücken. Mit dieser Regel können wir den G-Knoten des linken RG-Graphen aus Abb. 5.9 zunächst Abschwächen zu $\mathcal{R} \Rightarrow \mathcal{G}$ und dann den R-Knoten eliminieren. Damit ist es möglich einen endlichen RG-Graphen in einen mit nur einem G-Knoten umzuformen, der genau die gleichen Implementierungen besitzt.
Wir nehmen diese Regel jedoch nicht zu unseren Regeln hinzu. Denn es ist nicht sinnvoll, einen RG-Graphen in einen mit nur einem Knoten umzuwandeln. Dabei verlieren wir ja gerade die Abhängigkeiten der Eigenschaften des Moduls von denen der Umgebung, was die Motivation für die Einführung der RG-Graphen war. Ein weiteres Problem mit der obigen Regel entsteht, wenn wir Eigenschaften syntaktisch durch logische Ausdrücke repräsentieren. Wir werden in den Knoten später nur (ungeschachtelte) Aussagen der Form $\Box\varphi$, $\varphi \rightsquigarrow \psi$, etc. zulassen. Insbesondere lassen wir keine Disjunktionen oder Negationen dieser Aussagen zu. Damit sind die Eigenschaften, die aus der obigen Regel resultieren, syntaktisch nicht repräsentierbar.

## 5.5 Platzhalter für Zustände

In einem RG-Graphen wird teilweise auf Stellen der Umgebung Bezug genommen. Beispielsweise kommen im RG-Graphen des Mutex-Moduls die Stellen $e_1$ und $e_2$ vor. Die Namen der Stellen für die der wechselseitige Ausschluß gewährleistet werden soll, sind dem Modul aber a priori nicht bekannt, da sie ja Stellen der Umgebung sind. Andererseits ist es notwendig über diese Stellen zu reden, da für sie der wechselseitige Ausschluß gewährleistet werden soll. Außerdem wollen wir im RG-Graphen nicht über die Namen der internen Stellen reden,

da so die Menge der Implementierung durch die spezielle Namensgebung eingeschränkt wird. Dieses Problem könnten wir mit Hilfe der Umbenennung lösen. Wenn wir aber erlauben wollen, daß es in der Umgebung mehrere verschiedene Stellen für den kritischen Bereich geben kann, deren Anzahl uns nicht bekannt ist, so benötigen wir einen *Platzhalter* für diese Menge.
Im RG-Graph ersetzen wir also den konkreten Namen $e_1$ durch einen *Platzhalter* $E_1$, der in einer Implementierung durch eine konkrete Menge von Stellen ersetzt wird. Da Platzhalter im allgemeinen für Stellenmengen stehen, benutzen wir für sie Großbuchstaben. Für einen Platzhalter geben wir an, ob er für eine Menge von interne oder externe Stellen steht. So führen wir z.B. im Mutex-Modul Platzhalter für externe Stellenmengen $E_1$ und $E_2$ ein, für die der wechselseitige Ausschluß garantiert wird. Wir notieren dies durch $E_1, E_2 : \mathit{extern}$[2]. Konkret wird dann z.B. $E_1$ durch $\{e_1\}$ bzw. $E_2$ durch $\{e_2\}$ *instantiiert*. Im Mutex-Beispiel ist jedoch nur eine Instantiierung von $E_1$ und $E_2$ mit disjunkten Mengen sinnvoll. Solche Einschränkungen spielen auch beim Rechnen mit Modulen, eine wesentliche Rolle. Deshalb notieren wir zusätzlich zur Art der Platzhalter auch diese Einschränkungen. Für unser Beispiel schreiben wir $E_1 \cap E_2 = \emptyset$. Die konkrete Instantiierung muß dann diese Einschränkung erfüllen.
Eine Implementierung eines Moduls mit Platzhaltern ist dann eine Systemkomponenten mit passender Schnittstelle zusammen mit einer Instantiierung für die internen Platzhalter, wobei die Instantiierung die Einschränkungen erfüllen muß. Eine Systemkomponente erfüllt den entsprechenden RG-Graphen, wenn die internen Platzhalter durch die gewählte Instantiierung ersetzt werden und für jede Instantiierung der externen Platzhalter (unter Beachtung der zusätzlichen Einschränkungen) die Systemkomponente die entsprechenden RG-Graphen erfüllt. Zum Nachweis ist es jedoch nicht erforderlich, für die externen Platzhalter eine Substitution durchzuführen. Wir können Beweise auch mit den Platzhaltern führen. Wir müssen dabei nur berücksichtigen, daß zwei verschiedene Platzhalter möglicherweise durch Instantiierung dieselbe Bedeutung erhalten können. Die einzige Information über die Bedeutung der Platzhalter darf den Einschränkungen und den Eigenschaften im RG-Graph entnommen werden. Dies werden wir in Kapitel 8 an einem Beispiel demonstrieren

Noch allgemeiner können wir sogar Platzhalter für beliebige Zustands-

[2]Für einen Platzhalter $X$ für eine interne Stellenmengen schreiben wir entsprechend *X: intern*

aussagen einführen, deren Bedeutung erst bei der Implementierung konkretisiert werden. Diese Zustandsaussage kann dann in den verschiedenen Inschriften des RG-Graphen wieder auftauchen und wird so näher charakterisiert. Auch für diese Aussagen muß festgelegt werden, ob sie bei der Implementierung der Systemkomponente festgelegt wird, oder von der Umgebung beliebig gewählt werden darf (unter Respektierung der im RG-Graphen geforderten Eigenschaften). Platzhalter für Zustandsaussagen notieren wir ähnlich wie bei den Platzhaltern für Stellenmengen durch $\varphi$*: intern* bzw. $\varphi$*: extern*.
Wir werden später anhand einiger Beispiele sehen, daß die Platzhalter manche Spezifikationen vereinfachen oder sogar erst ermöglichen. Außerdem zeigen wir, daß man trotz vorhandener Platzhalter Beweise führen kann.

## 5.6 Implementierbarkeit

Zuletzt wollen wir kurz auf den Aspekt der Implementierbarkeit von Modulen eingehen. Offensichtlich gibt es Module, die nicht implementierbar sind. Es gibt verschiedene Gründe, die eine Implementierung eines Moduls unmöglich machen. Die naheliegendste Möglichkeit eines nicht-implementierbaren Moduls ist ein G-Knoten mit der Inschrift $\emptyset$. Da jede Systemkomponente mindestens einen Ablauf besitzt, gibt es keine Implementierung für ein solches Modul. Dies Art der Unimplementierbarkeit wollen wir hier aber nicht näher betrachten.
Eine andere Art der Unimplementierbarkeit tritt auf, wenn der RG-Graph über externe Stellen Aussagen macht, ohne hinreichend viele Annahmen über die Umgebung zu treffen. So ist ein Modul mit zwei externen Stellen $e_1$ und $e_2$ nicht implementierbar, wenn ein G-Knoten ohne R-Vorgänger mit der Mutex-Eigenschaft $\Box\neg(e_1 \wedge e_2)$ beschriftet ist. Denn die Umgebung kann $e_1$ und $e_2$ gleichzeitig markieren. Diese Art der Unimplementierbarkeit sieht man einem Modul manchmal nicht unmittelbar an. Mit etwas Erfahrung sieht man in realistischen Beispielen relativ schnell, ob die Annahmen an die Umgebung ausreichen, um bestimmte Eigenschaften zu gewährleisten. Diese Art der Unimplementierbarkeit könnten wir vermeiden, indem wir in G-Knoten keine Aussagen über externe Stellen machen. Diese Einschränkung schließt dann aber auch unser Beispiel des Mutex-Moduls aus. Das Mutex-Modul zeigt, daß es durchaus sinnvoll ist, über externe Stellen Aussagen zu machen. Deshalb erscheint es uns sinnvoller, die Möglichkeit der Unimplementierbarkeit eines Moduls in Kauf zu nehmen.

Ein weitere Frage betrifft die Implementierbarkeit der Umgebung. Wenn z.B. ein R-Knoten eines RG-Graphen mit $\emptyset$ beschriftet ist, ist das Modul selbst einfach implementierbar. Allerdings gibt es keine Umgebung, die diese Anforderung erfüllt und in die das Modul sinnvoll eingebettet werden kann. Auch hier gibt es wieder subtilere Fälle, bei denen dies nicht sofort erkennbar ist. Diese Fälle schließen wir ebensowenig aus wie die Unimplementierbarkeit des Moduls selbst. Um dieses Problem in der Praxis zu vermeiden, kann man beispielsweise ein einfaches Modell für die Umgebung angeben, selbst wenn die Umgebung nicht implementiert werden muß. So ist gewährleistet, daß man keine unerfüllbaren Anforderungen an die Umgebung stellt.

Die Forderung der Implementierbarkeit der Umgebungsannahmen hat eine weitere wichtige Konsequenz, auf die wir im folgenden näher eingehen. Dazu betrachten wir nochmals den RG-Graphen auf der linken Seite von Abb. 5.9. Für bestimmte Sicherheitseigenschaften $\mathcal{R}$ und $\mathcal{G}$ gibt es nun Systemkomponenten, die einen Ablauf mit der folgenden Eigenschaft besitzen: In dem Ablauf verletzt zunächst die Systemkomponente die Eigenschaft $\mathcal{G}$; erst später verletzt die Umgebung die Eigenschaft $\mathcal{R}$. Eine derartige Systemkomponente sollte als „Implementierung“ des obigen RG-Graphen nicht zulässig sein, weil die Systemkomponente die zu gewährleistende Eigenschaft $\mathcal{G}$ bereits verletzt, bevor die Anforderung an die Umgebung $\mathcal{R}$ verletzt wird [1]. In der Literatur wird einiger formaler Aufwand betrieben um solche „Implementierungen“ von der Betrachtung auszuschließen [62, 1, 2, 59]. Zunächst sieht es so aus, als ob unser Ansatz solche „Implementierungen“ nicht ausschließt, weil der betrachtete Ablauf zwar $\mathcal{G}$ aber auch $\mathcal{R}$ verletzt. Wenn die Umgebungsannahme $\mathcal{R}$ jedoch implementierbar ist, dann tritt in keiner der Implementierungen des RG-Graphen ein Ablauf auf, wie er oben beschrieben wurde.
Die Forderung der Implementierbarkeit der Umgebungsannahmen ersetzt damit einen aufwendigen Formalismus, der die beschriebenen unerwünschten „Implementierungen“ ausschließt.

## 5.7 Literatur

Wie wir gezeigt haben, ist es sehr einfach auf dem kompositionalen Verhalten des Systemmodells einen Spezifikationsformalismus im Rely-Guarantee-Stil [42] aufzubauen. Das prinzipielle Problem bei der

Kombination von zwei Spezifikationen sind *zyklische Abhängigkeiten* [74, 1, 68, 2]. Wir geben hierzu ein einfaches Beispiel an: Angenommen wir haben zwei Spezifikationen $\mathcal{R}_1 \Rightarrow \mathcal{G}_1$ und $\mathcal{R}_2 \Rightarrow \mathcal{G}_2$; außerdem möge $\mathcal{G}_1$ die Eigenschaft $\mathcal{R}_2$ implizieren und $\mathcal{G}_2$ die Eigenschaft $\mathcal{R}_1$ implizieren. Dann liegt der Schluß nahe, daß $\mathcal{G}_1$ und $\mathcal{G}_2$ im zusammengesetzen System ohne weitere Voraussetzung gelten. Daß dies im allgemeinen nicht gilt, sieht man leicht am Speziallfall $\mathcal{G} \mathrel{\widehat{=}} \mathcal{R}_1 = \mathcal{R}_2 = \mathcal{G}_1 = \mathcal{G}_2$. Die beiden Systeme verlassen sich zyklisch auf die gleiche Eigenschaft und können sie deshalb trivialerweise garantieren. Trotzdem gilt die Aussage $\mathcal{G}$ im zusammengesetzten System nicht unbedingt.
Bei der Kombination von Spezifikationen müssen wir solche Zyklen aufbrechen. Dazu gibt es im wesentlichen zwei Möglichkeiten [74] (siehe auch [68]):

1. Wir beschreiben explizit die Abhängigkeiten zwischen den Eigenschaften der Umgebung und des Moduls mit Hilfe einer Relation. Bei der Kombination von Eigenschaften der beiden Spezifikationen darf kein Zyklus entstehen. Dieser Ansatz wird von Francez und Pnueli [30, 74] verfolgt.

2. Wenn wir auf die explizite Repräsentation der Abhängigkeiten verzichten wollen, können wir die Struktur der Abläufe selbst nutzen. Ein System[3] erfüllt die Spezifikation $\mathcal{R} \Rightarrow \mathcal{G}$, wenn für jeden Ablauf des Systems und jedes $n \in \mathbb{N}$ gilt: Wenn der Präfix der Länge $n$ die Eigenschaft $\mathcal{R}$ erfüllt, dann erfüllt der Präfix der Länge $n+1$ die Eigenschaft $\mathcal{G}$.

   Dieser Ansatz wurde erstmals von Misra und Chandy [62] vorgeschlagen und von Abadi und Lamport [1, 2] und anderen [4, 59] weiter ausgearbeitet und verallgemeinert.

Der zweite Ansatz hat den Nachteil, daß er im wesentlichen nur für Sicherheitseigenschaften geeignet ist [2]. Deshalb haben wir uns für den ersten Ansatz entschieden. Unser Ansatz unterscheidet sich jedoch von Pnuelis [74] Vorschlag. Dort wird für beide Spezifikationen dieselbe Ordnung zugrundegelegt. Durch die gemeinsame Ordnung ist bei der Komposition klar welche Eigenschaften der beiden Teile aufeinander aufbauen. Außerdem ist dort wiederholtes Kombinieren von Spezifikationen nicht vorgesehen. In unserem Ansatz muß die Ordnungen der beiden RG-Graphen nicht gleich sein. Bei der Kombination der Spezifikation legen wir die beiden RG-Graphen unabhängig nebeneinander.

[3] Wir beschreiben den Ansatz hier für sequentielle Abläufe; dieser Ansatz läßt sich jedoch auch an unseren Ablaufbegriff anpassen.

Anschließend können wir beliebige weitere Abhängigkeiten hinzunehmen (solange die Relation eine Ordnung bleibt) und ggf. gewisse Knoten eliminieren. Anschließend kann das resultierende Modul mit einem weiteren kombiniert werden. Damit ist unsere Ansatz allgemeiner und flexibler als [74]. Trotzdem erhalten wir eine einfache Kompositionsregel.

Neben den oben genannten Ansätzen gibt es andere Ansätze zum kompositionalen Beweisen, die nicht so sehr an der Rely-Guarantee-Struktur interessiert sind. In [9] ist es zwar noch möglich Spezifikationen im Rely-Guarantee-Stil zu schreiben, indem man geeignete temporalen Aussagen wählt. Dies ist aber dem Anwender überlassen. Ähnlich verhält es sich mit [22]. Hier wird — ebenfalls am Beispiel des wechselseitigen Ausschluß — gezeigt, daß man Spezifikationen in FOCUS [17] als Rely-Guarantee-Spezifikationen formulieren kann. Die Abhängigkeiten der Eigenschaften der Komponente von Eigenschaften der Umgebung wir dort als logische Implikation formuliert. Eine ausführlichere Darstellung dieses Ansatzes gibt Weber [88].

Darüber hinaus gibt es einige eingeschränktere Ansätze für Rely-Guarantee-Spezifikationen [50, 85]. Lamport [50] beschreibt zulässige Zustandsübergänge der Umgebung (allowed changes). In [85] ist es mit der „wait-condition“ sogar möglich, bestimmte einfache Lebendigkeitseigenschaften von der Umgebung vorauszusetzen. Allerdings ist dieser Ansatz zugeschnitten auf terminierende Programme und weniger geeignet für reaktive Systeme. Stølen selbst sieht diese Arbeit als kompositionale Erweiterung der Owicki-Gries-Methode [66] an.

Gemäß de Roever [21] ist eine Beweismethode nur dann kompositional, wenn ein Korrektheitsbeweis für eine Systemkomponente keine Informationen über die Implementierung der Umgebung erfordert. In diesem Sinne sind unsere und die oben diskutierten Methoden kompositional. Einige klassische Methoden zur Verifikation paralleler Programme (z.B. [66]) sind nicht kompositional, da zusätzlich für das Gesamtsystem die „Interferenzfreiheit“ bewiesen werden muß; d.h. daß die Implementierung der Umgebung den Beweis nicht zerstört. Alle oben diskutierten Methoden versuchen die „Interferenzfreiheit“ durch explizite Annahmen über die Umgebung zu gewährleisten.

# Kapitel 6

# Logik für verteilte Abläufe

In den vorangegangenen Kapiteln haben wir beliebige Teilmengen von Abläufen als Eigenschaften betrachtet. Das in Kapitel 5 vorgestellte Modulkonzept ist unabhängig von einer speziellen syntaktischen Repräsentation für Eigenschaften. Wir werden nun eine *temporale Logik* für verteilte Abläufe zur Repräsentation von Eigenschaften einführen. Die *temporalen Aussagen* benutzen wir in Knoteninschriften von RG-Graphen als Repräsentant für die entsprechende Eigenschaft. Damit haben wir einen Formalismus mit einer präzisen Bedeutung, mit dem wir Systemkomponenten spezifizieren und die Korrektheit von Implementierungen nachweisen können.

Die temporale Logik für verteilte Abläufe ist eine Erweiterung der temporalen Logik für sequentielle Abläufe [54]. Neben den klassischen temporalen Operatoren $\Box$, $\Diamond$ und $\bigcirc$ besitzt unsere Logik einen weiteren Operator $\triangledown$, mit dem bestimmte Aspekte der Nebenläufigkeit in einem Ablauf ausgedrückt werden können. Mit Hilfe dieser Operatoren werden wir dann die Eigenschaften formulieren, die wir zur Spezifikation von Systemen benutzen.

Wie üblich baut die temporale Logik auf einer Logik für Zustände auf. Da Zustände von verteilten Transitionssystemen Multimengen sind, wird unsere *Zustandslogik* über Multimengen interpretiert.

## 6.1 Zustandslogik

Wir führen nun die *Zustandslogik* ein. Mit dieser Logik können wir Aussagen über Multimengen formulieren. Wir geben zunächst einige Beispiele für Zustandsaussagen mit ihrer Interpretation an:

1. $\#\{a\} \geq 1$ (später abgekürzt durch $a$) gilt in jeder Multimenge, die mindestens ein Element $a$ besitzt.

2. $\#\{a\} \geq \#\{b\} \wedge \#\{b\} \geq \#\{a\}$ (später abgekürzt durch $\#a = \#b$) gilt in jeder Multimenge, die genauso viele Elemente $a$ wie Elemente $b$ enthält.

3. Für eine endliche Menge $Z \subseteq S$ gilt $\#Z \geq 7$ in jeder Multimenge, die mindestens 7 Elemente aus $Z$ enthält.

Die Grundbausteine der Zustandsaussagen sind also *Terme* der Form $\#Z$, wobei $Z$ eine endliche Menge ist. Über einer Multimenge interpretiert, ordnen wir $\#Z$ ein natürliche Zahl zu. Daneben erlauben wir in Termen natürliche $n \in \mathbb{N}$ und Variablen über den natürlichen Zahlen $v \in V$. Zusammen mit dem Summenoperator $+$ definieren wir aus diesen Grundbausteinen induktiv die Terme der Zustandslogik. Für die Variablen benutzen wir im folgenden die Symbole $u$, $v$, $w$, ... und $v_1$, $v_2$, ....
Dabei legen wir uns nicht auf eine bestimmte syntaktische Repräsentation für natürliche Zahlen und endliche Mengen fest. So lassen wir beispielsweise für eine Transition $t$ eines Systems $|{}^\bullet t|$ als Repräsentation einer natürlichen Zahl zu. Im Gegensatz dazu ist $\#Z$ keine natürliche Zahl; diesem Term wird erst bei der Interpretation eine natürliche Zahl zugeordnet.

**Definition 6.1 (Terme)**

Sei $S$ eine Menge und $V$ eine Menge von Variablen. Dann definieren wir die Menge der *Terme* $\mathbf{ZT}(S, V)$ über $S$ und $V$ induktiv durch:

1. Für $n \in \mathbb{N}$ gilt $n \in \mathbf{ZT}(S, V)$.
2. Für $v \in V$ gilt $v \in \mathbf{ZT}(S, V)$.
3. Für jede endliche Menge $Z \subseteq S$ gilt $\#Z \in \mathbf{ZT}(S, V)$.
4. Für $t, t' \in \mathbf{ZT}(S, V)$ gilt auch $t + t' \in \mathbf{ZT}(S, V)$.

Für zwei Terme $t, t' \in \mathbf{ZT}(S, V)$ ist $t \geq t'$ eine Zustandsaussage. Aus diesen Zustandsaussagen bilden wir mit den üblichen logischen Operatoren und der Quantifikation die Menge der Zustandsaussagen. Zusätzlich lassen wir in Zustandsaussagen *atomare Aussagen* zu, die zunächst nicht näher bestimmte Aussagen sind. Für die atomaren Aussagen reservieren wir die Symbole $A, B, C, \ldots$ und $A_1, A_2, \ldots$. Die Menge der *atomaren Aussagensymbole* bezeichnen wir mit $P$.

**Definition 6.2 (Zustandsaussagen)**

Seien $S, V, P$ drei Mengen, dann definieren wir die Menge der *Zustandsaussagen* $\mathbf{ZA}(S, V, P)$ über $S, V$ und $P$ induktiv durch:

1. $P \subseteq \mathbf{ZA}(S, V, P)$.
2. Für $t, t' \in \mathbf{ZT}(S, V)$ ist $t \geq t' \in \mathbf{ZA}(S, V, P)$.
3. Für $\varphi, \psi \in \mathbf{ZA}(S, V, P)$ und $v \in V$ sind
   (a) $(\varphi \wedge \psi) \in \mathbf{ZA}(S, V, P)$,
   (b) $\neg\varphi \in \mathbf{ZA}(S, V, P)$ und
   (c) $\exists v : \varphi \in \mathbf{ZA}(S, V, P)$.

Für $\mathbf{ZA}(S, V, \emptyset)$ schreiben wir kurz $\mathbf{ZA}(S, V)$.

Die logischen Operatoren $\Rightarrow$, $\vee$ und $\Leftrightarrow$ sowie die Relation $>$, $\leq$, $<$ und $=$ können wir als abkürzende Schreibweisen in Zustandsaussagen verwenden, da sie sich mit Hilfe der anderen Operationssymbole und der $\geq$-Relation ausdrücken lassen. Beispielsweise können wir $t = t'$ durch $(t \geq t' \ \wedge \ t' \geq t)$ ausdrücken. Darüber hinaus führen wir die folgenden Abkürzungen ein.

**Notation 6.3**

Wir nehmen im folgenden an, daß die Mengen $S$, $V$ und $P$ paarweise disjunkt sind. Wir schreiben

1. $\top$ an Stelle der immer wahren Aussage $0 \leq 0$,
2. $\bot$ an Stelle der immer falschen Aussage $\neg\top$,
3. für $s \in S$ auch $s$ an Stelle von $\#\{s\} \geq 1$ und
4. für eine endliche Menge $Z \subseteq S$ auch $Z$ an Stelle von $\#Z \geq 1$. Dies ist äquivalent zu der Disjunktion $\bigvee_{s \in Z} s$ (vgl. 3.).
5. Für eine endliche Menge $Z \subseteq S$ schreiben wir $\bigwedge Z$ abkürzend für $\bigwedge_{s \in Z} s$.

Um die Gültigkeit einer Zustandsaussage für eine Multimenge zu definieren, ordnen wir den Variablen $V$ und den atomaren Aussagen eine Bedeutung zu. Dazu definieren wir den Begriff der *Belegung* $\beta$, die jeder Variablen eine natürliche Zahl zuordnet, und den Begriff der *Wahrheitstafel* $\pi$, die die wahren atomaren Aussagen von $P$ auszeichnet. Eine Belegung $\beta'$, die sich nur bezüglich einer Variablen $v$ von $\beta$ unterscheidet, nennen wir *$v$-Variante* von $\beta$.

**Definition 6.4 (Belegung und Wahrheitstafel)**

Sei $V$ eine Variablenmenge. Eine Abbildung $\beta : V \to \mathbb{N}$ nennen wir *Belegung* für $V$.

Sei $\beta$ eine Belegung für $V$ und $v \in V$ eine Variable. Eine Belegung $\beta'$ für $V$ heißt *Belegungsvariante von $\beta$ bzgl. $v$* oder kurz *$v$-Variante* von $\beta$, wenn für alle Variablen $w \in V$ mit $w \neq v$ gilt $\beta(w) = \beta'(w)$.

Sei $P$ eine Menge von atomaren Aussagensymbolen. Eine Teilmenge $\pi \subseteq P$ nennen wir *Wahrheitstafel* für $P$.

Bei der *Interpretation* einer Zustandsaussage werden wir alle atomaren Aussagen $\pi \in P$ als wahr ansehen, alle anderen als falsch. Damit können wir nun die Gültigkeit einer Zustandsaussage $\varphi$ für eine Multimenge $M$, eine Belegung $\beta$ und eine Wahrheitstafel $\pi$ induktiv über den Aufbau der Terme bzw. der Aussage definieren. Wir schreiben dafür $M, \beta, \pi \models \varphi$.

**Definition 6.5 (Gültigkeit von Zustandsaussagen)**

Sei $M$ eine Multimenge und $\beta$ ein Belegung $\beta : V \to \mathbb{N}$ für die Variablenmenge $V$. Wir interpretieren die Terme in $M$ unter der Belegung $\beta$ durch $\beta_M : \mathbf{ZT}(S, V) \to \mathbb{N}$, wobei $\beta_M$ induktiv definiert ist durch:

1. $\beta_M(n) = n$ für $n \in \mathbb{N}$.
2. $\beta_M(v) = \beta(v)$ für $v \in V$.
3. $\beta_M(\#Z) = \Sigma_{s \in Z} M(s)$ für $Z \subseteq S$.
4. $\beta_M(t + t') = \beta_M(t) + \beta_M(t')$ für $t, t' \in \mathbf{ZT}(S, V)$.

Die Gültigkeit einer Zustandsaussage $\varphi \in \mathbf{ZA}(S, V, P)$ in einer Multimenge $M$ unter Belegung $\beta$ und Wahrheitstafel $\pi$ wird notiert durch $M, \beta, \pi \models \varphi$ und induktiv definiert durch:

1. Für $p \in P$ gilt $M, \beta, \pi \models p$ genau dann, wenn $p \in \pi$.

2. Für $t, t' \in \mathbf{ZT}(S, V)$ gilt $M, \beta, \pi \models t \geq t'$ genau dann, wenn $\beta_M(t) \geq \beta_M(t')$
3. Für $\varphi, \psi \in \mathbf{ZA}(S, V, P)$ und $v \in V$ gilt
   (a) $M, \beta, \pi \models (\varphi \wedge \psi)$ genau dann, wenn $M, \beta, \pi \models \varphi$ und $M, \beta, \pi \models \psi$ gilt,
   (b) $M, \beta, \pi \models \neg\varphi$ genau dann, wenn nicht $M, \beta, \pi \models \varphi$ gilt und
   (c) $M, \beta, \pi \models \exists v : \varphi$ genau dann, wenn es eine $v$-Variante $\beta'$ von $\beta$ gibt, so daß $M, \beta', \pi \models \varphi$ gilt.

Diese Definition zeigt, daß die Endlichkeit der Menge $Z$ in einem Term $\#Z$ wesentlich ist; sonst wäre $\beta_M(\#Z) = \Sigma_{s \in Z} M(s)$ für bestimmte Markierungen $M$ nicht definiert (bzw. keine natürliche Zahl).
Wir sagen für $M, \beta, \pi \models \varphi$ auch, daß $\varphi$ in $M$ unter $\beta$ und $\pi$ gilt. Wenn eine Zustandsaussage $\varphi$ für eine Multimenge $M$ unter $\beta$ und $\pi$ nicht gilt, dann schreiben wir dafür auch $M, \beta, \pi \not\models \varphi$.

**Definition 6.6 (Allgemeingültigkeit)**

Sei $M$ eine Multimenge und $\beta$ eine Belegung für $V$ und $\varphi \in \mathbf{ZA}(S, V, P)$ eine Zustandsaussage. Wir schreiben

1. $M, \beta \models \varphi$, wenn für jede Wahrheitstafel $\pi$ für $P$ gilt $M, \beta, \pi \models \varphi$.
2. $M \models \varphi$, wenn für jede Belegung $\beta'$ von $V$ gilt $M, \beta' \models \varphi$.
3. $\models \varphi$, wenn für jede Multimenge $M'$ gilt $M' \models \varphi$.

Wenn $\models \varphi$ gilt, dann heißt $\varphi$ *allgemeingültig*.

## 6.2 Temporale Logik

Wir können die Gültigkeit einer Zustandsaussage unmittelbar auf erreichbare **co**-Mengen eines Ablaufs $\rho$ anwenden. Dazu haben wir jeder **co**-Menge $Q$ eines Ablaufs $\rho$ die Abbildung $M_{\rho,Q}$ zugeordnet, die für erreichbare **co**-Mengen eine Multimenge ist (siehe Def. 2.36 und Satz 2.37). Wir nennen eine erreichbare **co**-Menge $Q$ eines Ablaufs $\rho$ $\varphi$-Menge, wenn gilt $M_{\rho,Q} \models \varphi$.
Damit ist es möglich, Aussagen über einzelne **co**-Mengen eines Ablaufs zu machen. Die temporale Logik stellt darüber hinaus Operatoren zur Verfügung, um zwischen verschiedenen **co**-Mengen eine Beziehung herzustellen. Beispielsweise gilt in einer **co**-Menge $Q$ die Aussage

$\Diamond\varphi$, wenn von $Q$ eine $\varphi$-Menge erreichbar ist. In $Q$ gilt $\diamondsuit\varphi$, wenn ein (unmittelbarer) Nachfolger $Q'$ von $Q$ eine $\varphi$-Menge ist. Die beiden Operatoren entsprechen also den klassischen temporalen Operatoren $\Diamond$ und $\bigcirc$. Allerdings gibt es im Gegensatz zu Sequenzen für eine **co**-Menge im allgemeinen mehrere unmittelbare Nachfolger. Deshalb gibt es nicht nur für $\Diamond$ den *dualen* Operator $\Box$, sondern auch für $\diamondsuit$ den dualen Operator $\boxdot$. Für Sequenzen ist $\bigcirc$ dual zu sich selbst[1].
Neben den klassischen Operatoren führen wir einen neuen Operator $\triangledown$ ein, mit dem wir einige Aspekte der Nebenläufigkeit in einem Ablauf ausdrücken können. Die Aussage $\triangledown\varphi$ gilt in einer **co**-Menge $Q$, wenn $\varphi$ in einer echten Teilmenge von $Q$ gilt. Zusammen mit dem Operator $\diamondsuit$ können wir beispielsweise die **co**-Mengen eines Ablaufs charakterisieren, die Vorbereich eines Ereignisses sind: $(\diamondsuit\top) \wedge \neg\triangledown\diamondsuit\top$.
Wir definieren nun aufbauend auf den Zustandsaussagen induktiv die Menge der temporalen Aussagen.

**Definition 6.7 (Temporale Aussagen)**

Sei $S$ eine Menge, $V$ eine Variablenmenge und $P$ eine Menge von atomaren Aussagen. Wir definieren die Menge der temporalen Aussagen $\mathbf{TA}(S, V, P)$ über $S$, $V$ und $P$ induktiv wie folgt:

1. $\mathbf{ZA}(S, V, P) \subseteq \mathbf{TA}(S, V, P)$.
2. Seien $\varphi, \psi \in \mathbf{TA}(S, V, P)$ , so gilt auch
   (a) $(\varphi \wedge \psi) \in \mathbf{TA}(S, V, P)$,
   (b) $\neg\varphi \in \mathbf{TA}(S, V, P)$ und
   (c) $\exists v : \varphi \in \mathbf{TA}(S, V, P)$ für $v \in V$.
3. Sei $\varphi \in \mathbf{TA}(S, V)$, so gilt auch
   (a) $\diamondsuit\varphi \in \mathbf{TA}(S, V, P)$,
   (b) $\Diamond\varphi \in \mathbf{TA}(S, V, P)$ und
   (c) $\triangledown\varphi \in \mathbf{TA}(S, V, P)$.

Für die Menge der temporalen Aussagen $\mathbf{TA}(S, V, \emptyset)$, in denen also keine atomaren Aussagen vorkommen, schreiben wir kurz $\mathbf{TA}(S, V)$.

Wir lassen in temporalen Aussagen atomare Aussagen syntaktisch zu. Wir benötigen die atomaren Aussagen in temporalen Aussagen jedoch nur zur Formulierung von Beweisregeln. Die atomaren Aussagen dienen dort als Platzhalter zum Einsetzen von anderen temporalen Aussagen.

[1] wenn die Logik nur über unendlichen Sequenzen interpretiert wird.

Zur Formulierung von Eigenschaften benötigen wir dagegen keine atomaren Aussagen. Deshalb definieren wir die Gültigkeit von temporalen Aussagen nur für temporale Aussagen, die keine atomaren Aussagen enthalten.

**Definition 6.8 (Gültigkeit von temporalen Aussagen)**

Sei $S$ eine Menge, $V$ eine Variablenmenge, $\beta$ eine Belegung und $(K, \rho)$ ein Ablauf über $S$. Die *Gültigkeit* einer temporalen Aussage $\varphi \in \mathbf{TA}(S, V)$ in einer erreichbaren **co**-Menge $Q$ von $K$ notieren wir durch $\rho, Q, \beta \models \varphi$ und definieren die Gültigkeit induktiv:

1. Für eine Zustandsaussage $\varphi \in \mathbf{ZA}(S, V)$ gilt $\rho, Q, \beta \models \varphi$ genau dann, wenn für die entsprechende Multimenge $M_{\rho,Q}$ gilt $M_{\rho,Q}, \beta \models \varphi$.
2. Für $\varphi, \psi \in \mathbf{TA}(S, V)$ gilt
   (a) $\rho, Q, \beta \models (\varphi \wedge \psi)$ genau dann, wenn $\rho, Q, \beta \models \varphi$ und $\rho, Q, \beta \models \psi$ gilt
   (b) $\rho, Q, \beta \models \neg\varphi$ genau dann, wenn nicht $\rho, Q, \beta \models \varphi$ gilt
   (c) $\rho, Q, \beta \models \exists v : \varphi$ (mit $v \in V$) genau dann, wenn eine $v$-Variante $\beta'$ von $\beta$ existiert, so daß $Q, \rho, \beta' \models \varphi$ gilt
3. Für $\varphi \in \mathbf{TA}(S, V)$ gilt
   (a) $\rho, Q, \beta \models \Diamond\!\!\!\!-\varphi$ genau dann, wenn ein $Q'$ mit $Q \longrightarrow Q'$ existiert, so daß gilt $\rho, Q', \beta \models \varphi$
   (b) $\rho, Q, \beta \models \Diamond\varphi$ genau dann, wenn ein $Q'$ mit $Q \stackrel{*}{\longrightarrow} Q'$ existiert, so daß gilt $\rho, Q', \beta \models \varphi$
   (c) $\rho, Q, \beta \models \triangledown\varphi$ genau dann, wenn ein $Q' \subset Q$ existiert, so daß gilt $\rho, Q', \beta \models \varphi$.

Wie bei Zustandsaussagen schreiben wir $\rho, Q \models \varphi$, wenn für jede Belegung $\beta$ gilt $\rho, Q, \beta \models \varphi$. Wir sagen dann, daß $\varphi$ in $Q$ gilt, und nennen $Q$ eine *$\varphi$-Menge*.

Der Operator $\triangledown$ ist mit Hilfe der echten Teilmengenbeziehung $\subset$ definiert. Die Gültigkeit von $\varphi$ in einer (möglicherweise nicht echten) Teilmenge von $Q$ wird dann durch die temporale Aussage $\varphi \vee \triangledown\varphi$ ausgedrückt. Hätten wir dagegen den Operator $\triangledown$ mit Hilfe der Teilmengenbeziehung $\subseteq$ definiert, wäre die Gültigkeit von $\varphi$ in einer echte Teilmengen von $Q$ nicht durch eine temporale Aussage formulierbar.
Ein Ablauf erfüllt eine temporale Aussage $\varphi$, wenn seine initiale Scheibe die Aussage $\varphi$ erfüllt. Durch diese Definition werden alle Variablen in $\varphi$ implizit universell quantifiziert.

**Definition 6.9 (Gültigkeit in einem Ablauf)**

Eine temporale Aussage $\varphi \in \mathbf{TA}(S, V)$ gilt in einem Ablauf $\rho$ über $S$ mit zugrundeliegendem Prozeßnetz $K$, wenn $\varphi$ in $\rho(^\circ K)$ gilt. Wir schreiben dann auch $\rho \models \varphi$ und sagen, daß $\rho$ die Aussage $\varphi$ erfüllt.

Die Menge aller Abläufe über $S$, die die temporale Aussage $\varphi$ erfüllen, bezeichnen wir mit $\mathcal{P}_\varphi$.

Damit können wir nun eine Eigenschaft mit Hilfe einer temporalen Aussage repräsentieren. Eine temporale Aussage $\varphi$ als Inschrift eines Knotens eines RG-Graphen steht dann für die Eigenschaft $\mathcal{P}_\varphi$.
Wir führen nun einige Abkürzungen ein. Insbesondere definieren wir die dualen Operatoren zu $\Diamond$, $\diamondsuit$ und $\triangledown$. Neben diesen Abkürzungen benutzen wir die üblichen logischen Operatoren $\Rightarrow$, $\vee$, $\forall v$, etc. als abkürzende Schreibweisen.

**Notation 6.10 (Abkürzungen)**

Sei $\varphi \in \mathbf{TA}(S, V, P)$. Dann schreiben wir

1. $\boxdot\varphi$ an Stelle von $\neg\diamondsuit\neg\varphi$,
2. $\Box\varphi$ an Stelle von $\neg\Diamond\neg\varphi$,
3. $\blacktriangledown\varphi$ an Stelle von $\neg\triangledown\neg\varphi$ und
4. $\blacktriangle\varphi$ an Stelle von $(\varphi \wedge \blacktriangledown\varphi)$.
5. **preset** an Stelle von $(\diamondsuit\top \wedge \neg\triangledown\diamondsuit\top)$.

Um temporale Aussagen übersichtlicher zu schreiben, führen wir Präzedenzregeln ein, so daß wir Klammern aus temporalen Aussagen weglassen können: Temporale Operatoren binden am stärksten; dann folgen mit abnehmender Bindungsstärke $\neg$, $\wedge$, $\vee$, $\Rightarrow$ und $\Leftrightarrow$. In Kombination mit dem Operator $\Box$ benutzen wir zur besseren Lesbarkeit häufig eckige Klammersymbole (z.B. $\Box[\varphi \Rightarrow \psi]$)
Wir geben nun einige Beispiele für temporale Aussagen und **co**-Mengen, in denen sie gültig sind.

**Beispiel 6.1**

In Abb. 6.1 ist ein Ablauf mit einigen ausgezeichneten **co**-Mengen $Q_0$ bis $Q_4$ dargestellt. Es gelten die folgenden Aussagen:

1. $\rho, Q_0 \models a$, weil die Zustandsaussage $a$ (abkürzend für $\#\{a\} \geq 1$) in $Q_0$ gilt.

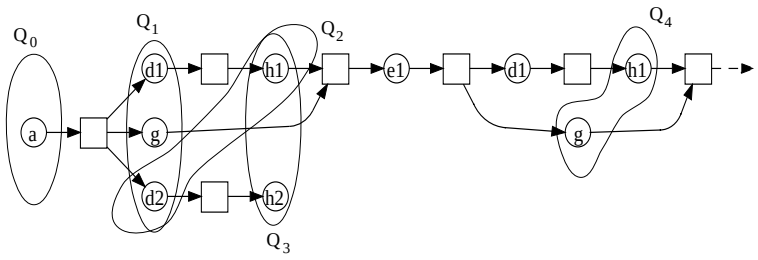



Abbildung 6.1: Beispielablauf $\rho$

2. $\rho, Q_0 \models \diamondsuit(d_1 \wedge g)$, weil für einen[2] unmittelbaren Nachfolger $Q_1$ von $Q_0$ die Zustandsaussage $d_1 \wedge g$ gilt. Es gilt sogar $\rho, Q_0 \models \boxdot(d_1 \wedge g)$.
3. $\rho, Q_1 \models \Diamond(h_1 \wedge h_2)$, weil eine von $Q_1$ erreichbare **co**-Menge existiert, in der $h_1 \wedge h_2$ gilt.
4. $\rho, Q_1 \models \neg\diamondsuit(h_1 \wedge h_2)$, weil kein unmittelbarer Nachfolger von $Q_1$ die Zustandsaussage $h_1 \wedge h_2$ erfüllt.
5. $\rho, \emptyset \models \triangledown\bot$, weil die leere Menge keine echte Teilmenge besitzt.
6. $\rho, Q_2 \models \boxdot\boxdot\bot$, weil für den einzigen unmittelbare Nachfolger $Q_3$ von $Q_2$ gilt $\boxdot\bot$. D.h. $Q_3$ besitzt keinen Nachfolger mehr.
7. $\rho, Q_0 \models$ **preset** und $\rho, Q_4 \models$ **preset**. Damit gilt auch $\rho \models$ **preset**; aber es gilt nicht $\rho \models \Box$**preset**, weil es eine von $Q_0$ erreichbare **co**-Menge gibt (z.B. $Q_1$) für die nicht **preset** gilt.
8. Für jede erreichbare **co**-Menge $Q$ von $\rho$ gilt $\rho, Q \models \neg(e_1 \wedge e_2)$. Damit gilt auch $\rho \models \Box\neg(e_1 \wedge e_2)$.
9. $\rho \models \Box\Diamond h_1$, weil im Ablauf immer wieder **co**-Mengen vorkommen, in denen die Zustandsaussage $h_1$ gilt.

## 6.3 Formulierung wichtiger Eigenschaften

In diesem Abschnitt formulieren wir die Eigenschaften, die wir als Knoteninschriften von RG-Graphen zulassen, und führen dafür geeignete Abkürzungen ein. Für diese Eigenschaften werden wir in Kapitel 7 Beweisregeln angeben.

[2] im Beispiel auch den einzigen

Die zentralen Eigenschaften sind *Invarianten* zur Spezifikation von Sicherheitseigenschaften und *Leadsto-Eigenschaften* zur Spezifikation von Lebendigkeitseigenschaften. Darüber hinaus spielen *partielle S-Invarianten* beim modularen Spezifizieren und Beweisen eine große Rolle. Mit partiellen S-Invarianten können Invarianten eines zusammengesetzten Systems aus partiellen Invarianten der Teile zusammengesetzt werden.

### 6.3.1 Invarianten

Eine Zustandsaussage $\varphi$ heißt *Invariante eines Ablaufs* $\rho$, wenn jede erreichbare Scheibe des Ablaufs eine $\varphi$-Menge ist. Dies ist gleichbedeutend mit $\rho \models \Box\varphi$. Die Zustandsaussage $\varphi$ heißt *Invariante einer Systemkomponente* $\Sigma$, wenn $\varphi$ eine Invariante alle Abläufe von $\Sigma$ ist.
Beispielsweise ist die Aussage $\neg(e_1 \wedge e_2)$ eine Invariante des Ablaufs aus Abb. 6.1 (siehe Bsp. 6.1). Sie ist sogar Invariante des Mutex-Systems, da jeder Ablauf diese Invariante erfüllt.

Wir werden nun weitere Eigenschaften formulieren, die in engem Zusammenhang zu Invarianten stehen. Insbesondere die partiellen S-Invarianten dienen dazu Invarianten aus „Invarianten-Teilen“ verschiedener Systemkomponenten zusammenzusetzen. Sie spielen deshalb beim modularen Beweis von Invarianten eine große Rolle.

#### S-Invarianten

In der Petrinetztheorie sind *S-Invarianten* ein wichtiges Hilfsmittel zum Beweis von Invarianten eines Systems. Grob gesprochen ist eine S-Invariante eines S/T-Systems eine Menge von Stellen, für die sich die Summe der Marken, die auf diesen Stelle liegen, beim Schalten nicht verändert[3]. Wir werden einige typische Beweismethoden, die auf S-Invarianten basieren, in unserer Logik übernehmen.
Ein Beispiel für eine S-Invariante im Mutex-Systems (nochmals in Abb. 6.2 dargestellt) ist die Menge $\{e_1, g, e_2\}$. Weil jede Transition beim Schalten genauso viele Marken in diese Stellenmenge hineinlegt, wie sie entfernt, ist in jeder erreichbaren Markierung des Systems die Summer der Marken auf diesen Stellen gleich der Summe am Anfang.

[3] Wir betrachten hier nur S-Invarianten mit Gewicht 1. Prinzipiell ist es möglich S-Invarianten mit beliebiger (insbes. auch negativer) Gewichtung mit Hilfe unserer Logik auszudrücken. Darauf verzichten wir in dieser Arbeit.

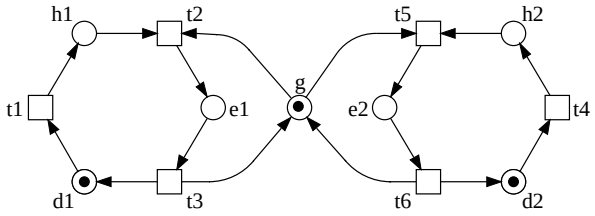



Abbildung 6.2: Das Mutex-System

Wir formulieren diese Eigenschaft für eine (endliche) Menge von Stellen $Z$ als eine temporale Aussage für einen Ablauf $\rho$:

$$\rho \models \#Z = v \;\Rightarrow\; \Box\#Z = v$$

Die Variable $v$ dient dazu, sich die Summe der Marken auf den Stellen $Z$ am Anfang des Ablaufs zu merken. $\Box\#Z = v$ besagt dann, daß in allen erreichbaren Scheiben genauso viele Marken auf Stellen von $Z$ liegen. Wir führen für diese Aussage eine Abkürzung ein.

**Notation 6.11 (S-Invarianten)**

Sei $Z \subseteq S$ eine endliche Menge. Wir definieren die abkürzende Schreibweise

$$\mathbf{inv}(Z) \mathrel{\widehat{=}} \#Z = v \;\Rightarrow\; \Box\#Z = v$$

Wenn $\mathbf{inv}(Z)$ in jedem Ablauf eines Systems $\Sigma$ gilt, nennen wir $Z$ eine *S-Invariante* von $\Sigma$.

S-Invarianten spielen in der Petrinetztheorie eine große Rolle, weil sie einfach mit Techniken der linearen Algebra aus der Netz-Struktur berechnet werden können. Für uns ist es ausreichend, daß wir sie für ein gegebenes System einfach verifizieren können. Dazu muß für jede Transition des Systems überprüft werden, ob sie beim Schalten genauso viele Marken aus der Stellenmenge entfernt, wie sie hinzufügt. S-Invarianten können also „lokal“ für jede einzelne Transition überprüft werden. Wir formulieren diese Aussage für alle Ereignisse eines Ablaufs $\rho$ als eine temporale Aussage.

**Lemma 6.12 (Lokalität der S-Invarianten)**

Sei $Z \subseteq S$ eine endliche Menge. Ein Ablauf $\rho$ erfüllt die Aussage $\rho \models \mathbf{inv}(Z)$ genau dann, wenn gilt

$$\rho \models \Box\triangledown[\mathbf{preset} \wedge \#Z = v \;\Rightarrow\; \boxdot\#Z = v]$$

Die Lokalität der Aussage wird dabei durch die Prämisse **preset** in der temporalen Aussage ausgedrückt. Deshalb ist die Aussage trivial für alle **co**-Mengen erfüllt, die nicht Vorbereich eines Ereignisses sind. Die temporale Aussage aus obigem Lemma stellt daher nur den Zusammenhang zwischen den Vor- und Nachbereichen der Ereignissen des Ablaufs her.
S-Invarianten hängen eng mit Invarianten zusammen. Beispielsweise folgt aus der obigen S-Invariante $\{e_1, g, e_2\}$ für das Mutex-System unmittelbar $\Box\neg(e_1 \wedge e_2)$, wenn wir wissen, daß initial nur $g$ markiert ist. In Kapitel 7 werden wir Beweisregeln angeben, die aus der Anfangsmarkierung und einer S-Invarianten eine Invariante beweisen.

### Partielle S-Invarianten

In zusammengesetzten Systemen kommt es häufig vor, daß sich eine S-Invariante aus den Stellenmengen mehrerer Teilkomponenten zusammensetzt. Eingeschränkt auf eine Teilkomponente ist diese Menge meist keine S-Invariante, weil eine Teilkomponente in eine beliebige Umgebung eingebettet sein kann.

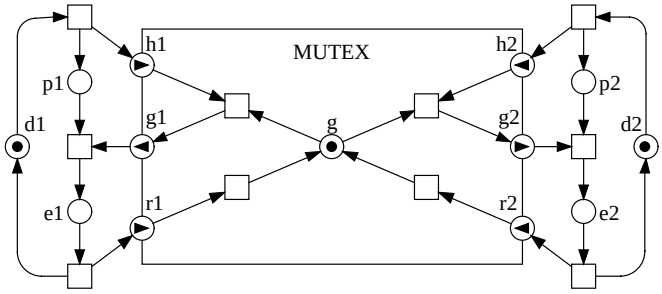



Abbildung 6.3: Die Mutex-Komponente in einer Umgebung

Dazu betrachten wir nochmals die Mutex-Komponente aus der Einleitung, die in eine Umgebung eingebettet ist. Dies ist nochmals in Abbildung 6.3 dargestellt. Offensichtlich gilt für die Mutex-Komponente in dieser Umgebung die S-Invariante $\{g_1, e_1, r_1, g, g_2, e_2, r_2\}$. Wenn wir allerdings eine andere Umgebung wählen, gilt diese Invariante nicht. Eine solche Umgebung ist in Abb. 6.4 dargestellt. Trotzdem erfüllt die Mutex-Komponente „ihren Teil“ der Invariante; die S-Invariante wird von der Umgebung verletzt. Um dies auszudrücken, führen wir

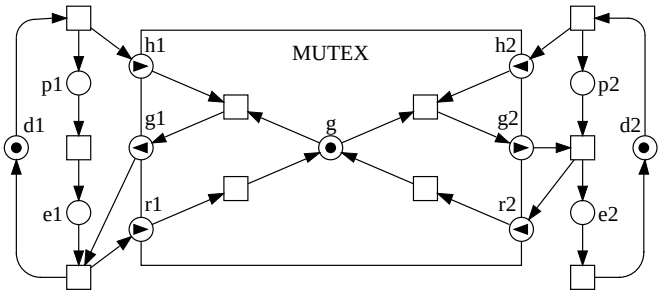



Abbildung 6.4: Die Mutex-Komponente in einer „falschen“ Umgebung

den Begriff der *partiellen S-Invariante* ein. Im Beispiel schreiben wir **p-i**$(\{r_1, r_2\}, \{g\}, \{g_1, g_2\})$. Dabei sind $r_1$ und $r_2$ Stellen, über die neue Marken in die „S-Invariante“ hinein kommen können, und $g_1$ und $g_2$ Stellen, über die Marken die S-Invariante verlassen können. Wenn die Umgebung ihren Teil der S-Invariante erfüllt, dann können wir die beiden Teile zu der oben angegebene S-Invariante zusammensetzen. Im Beispiel notieren wir den Teil, den die Umgebung erfüllen muß, durch **p-i**$(\{g_1, g_2\}, \{e_1, e_2\}, \{r_1, r_2\})$. Ähnlich wie beim Kombinieren von Systemkomponenten können wir partielle Invarianten zusammensetzen und wir erhalten aus den beiden obigen partiellen S-Invarianten die partielle S-Invariante **p-i**$(\emptyset, \{g_1, e_1, r_1, g, g_2, e_2, r_2\}, \emptyset)$. Da nun die Mengen, über die Marken in die „Invariante“ hinein kommen und über die Marken die „Invariante“ verlassen können, leer sind, ist diese partielle S-Invariante eine S-Invariante.
Die Eigenschaft einer partiellen S-Invariante für drei Stellenmengen $I$, $Z$ und $O$ formalisieren wir als temporale Aussage. Die Struktur der temporalen Aussage ist der lokalen Charakterisierung der S-Invariante sehr ähnlich (vgl. Lemma 6.12). Die Menge $I$ nennen wir *Anfang*, die Menge $O$ nennen wir *Ende* und die Menge $Z$ nennen wir *Inhalt* der partiellen S-Invariante.

**Notation 6.13 (partielle S-Invarianten)**

Seien $I, Z, O \subseteq S$ endliche Mengen. Wir definieren

$$\textbf{p-i}(I, Z, O) \mathrel{\widehat{=}} \Box\triangledown[\textbf{preset} \wedge \#I \cup Z = v \;\Rightarrow\; \boxdot\#Z \cup O = v]$$

Offensichtlich ist die Abkürzung **p-i**$(\emptyset, Z, \emptyset)$ identisch mit der Charakterisierung von S-Invarianten in Lemma 6.12. Damit gilt in einem

Ablauf die Aussage $\mathbf{inv}(Z)$ genau dann, wenn $\mathbf{p\text{-}i}(\emptyset, Z, \emptyset)$ gilt. Das Ziel der Kombination von partiellen S-Invarianten ist deshalb immer eine partielle S-Invariante mit leerem Anfang und leerem Ende.
Wir geben nun eine Regel zur Kombination von partiellen S-Invarianten an. Wie wir am Beispiel gesehen haben, können wir partielle S-Invarianten an ihren Anfangs- und Endstellen kombinieren, die dadurch zu Inhaltsstellen werden.

**Satz 6.14 (Kombination von partiellen S-Invarianten)**

Seien $I_1, I_2, Z_1, Z_2, O_1, O_2 \subseteq S$ endliche Mengen mit $(Z_1 \cup I_1) \cap (Z_2 \cup I_2) = (Z_1 \cup O_1) \cap (Z_2 \cup O_2) = \emptyset$ und $I \mathrel{\widehat{=}} (I_1 \setminus O_2) \cup (I_2 \setminus O_1)$, $Z \mathrel{\widehat{=}} Z_1 \cup Z_2 \cup (I_1 \cap O_2) \cup (I_2 \cap O_1)$ und $O \mathrel{\widehat{=}} (O_1 \setminus I_2) \cup (O_2 \setminus I_1)$.

Wenn für einen Ablauf $\rho$ die Aussagen $\rho \models \mathbf{p\text{-}i}(I_1, Z_1, O_1)$ und $\rho \models \mathbf{p\text{-}i}(I_2, Z_2, O_2)$ gelten, dann gilt auch $\rho \models \mathbf{p\text{-}i}(I, Z, O)$.

**Beweis:** Sei $\rho$ ein Ablauf für den die Voraussetzung der Aussage erfüllt sind. Für jede **co**-Menge dieses Ablaufes, die Vorbereich eines Ereignisses ist, gelten die beiden Aussagen $\#Z_1 \cup I_1 = u \Rightarrow \boxdot \#Z_1 \cup O_1 = u$ und $\#Z_1 \cup I_1 = v \Rightarrow \boxdot \#Z_1 \cup O_1 = v$. Wir müssen zeigen, daß dann für diese **co**-Mengen auch gilt $\#Z \cup I = w \Rightarrow \boxdot \#Z \cup O = w$.

Sei $Q$ eine beliebige **co**-Menge und $Q'$ ein unmittelbare Nachfolger von $Q'$. In $Q$ gilt:

$$w = \#(Z \cup I) =$$
$$\#(Z_1 \cup Z_2 \cup (I_1 \cap O_2) \cup (I_2 \cap O_1) \cup (I_1 \setminus O_2) \cup (I_2 \setminus O_1)) =$$
$$\#((Z_1 \cup I_1) \cup (Z_2 \cup I_2))$$

Wegen der Disjunktheitsannahmen gilt außerdem

$$\#((Z_1 \cup I_1) \cup (Z_2 \cup I_2)) = \#(Z_1 \cup I_1) + \#(Z_2 \cup I_2) = u + v$$

In $Q'$ gilt dann nach Voraussetzung: $u + v = \#(Z_1 \cup O_1) + \#(Z_2 \cup O_2)$. Zusammen mit den Disjunktheitsannahmen folgt dann $u + v = \#Z \cup O$ (mit derselben Argumentation wie zuvor). Mit $u + v = w$ gilt in $Q'$ die Aussage $\#Z \cup O = w$.

Damit gilt in $Q$ die Aussage $\#Z \cup I = w \Rightarrow \boxdot \#Z \cup O = w$. □

### 6.3.2 Erlaubte Übergänge und Synchronisation

Die Definition der partiellen S-Invarianten stellt den Zusammenhang zwischen Vor- und Nachbereichen von Ereignissen des Ablaufs her.

Partielle S-Invarianten machen also Aussagen über lokale Zustandsübergänge. Manchmal wollen wir auch über globale Übergänge eines Ablaufes reden, indem wir den Zusammenhang zwischen zwei unmittelbar aufeinanderfolgenden Scheiben eines Ablaufes herstellen. Dazu führen wir eine weitere Abkürzung ein.

**Notation 6.15 (Der alw-Operator)**

Seien $\varphi$ und $\psi$ zwei Zustandsaussagen (ohne atomare Aussagen), dann definieren wir

$$\varphi \ \mathbf{alw} \ \psi \mathrel{\widehat{=}} \Box[\varphi \Rightarrow \boxdot(\varphi \vee \psi)]$$

In einem Ablauf $\rho$ gilt die temporale Aussage $\varphi$ **alw** $\psi$ genau dann, wenn für jede erreichbare Scheibe $C$ von $\rho$, in der $\varphi$ gilt, für jede unmittelbare Nachfolger-Scheibe $C'$ entweder weiterhin $\varphi$ gilt oder $\psi$ gilt. Eine **alw**-Aussage beschreibt damit die „erlaubten globalen Übergänge“ eines Ablaufs.
Eine spezielle **alw**-Eigenschaft hat die Form $\varphi$ **alw** $\varphi$ (was äquivalent zu $\varphi$ **alw** $\bot$ ist). Diese Eigenschaft nennen wir[4] *stabile Eigenschaft*. Wenn $\varphi$ einmal gilt, dann gilt $\varphi$ für immer. Damit ist auch der Zusammenhang zu den Invarianten hergestellt: Wenn für einen Ablauf $\rho$ sowohl $\rho \models \varphi$ (d.h. initial gilt in dem Ablauf $\varphi$) als auch $\rho \models \varphi$ **alw** $\varphi$ gilt, dann gilt auch $\rho \models \Box\varphi$. Zwischen stabilen Eigenschaften und Invarianten besteht also ein ähnlicher Zusammenhang, wie zwischen S-Invarianten und Invarianten.
Wenn wir beliebige **alw**-Aussagen als Gegenstück zu den partiellen S-Invarianten sehen, geht die Analogie noch weiter. Wenn in einem Ablauf die Eigenschaften $\varphi$ **alw** $\psi$ und $\psi$ **alw** $\varphi$ gelten, dann gilt auch $\varphi \vee \psi$ **alw** $\varphi \vee \psi$. Wir können also **alw**-Aussagen zu stabilen Eigenschaften kombinieren. Mit Hilfe der stabilen Eigenschaften können wir dann ggf. (d.h. wenn die Aussage initial gilt) eine Invariante beweisen. Das Zusammensetzen von **alw**-Eigenschaften ist dann sinnvoll, wenn die kombinierten **alw**-Eigenschaften von verschiedenen Komponenten des Systems herrühren und so eine Invariante des Gesamtsystems gewonnen werden kann. Beispielsweise kann ein Modul von einer Umgebung **alw**-Eigenschaft annehmen. Diese **alw**-Eigenschaft kann mit einer eigenen **alw**-Eigenschaft zu einer Invariante kombiniert werden. Wir haben also neben den Petrinetz-spezifischen partiellen S-Invarianten eine weitere Möglichkeit, Invarianten modular zu beweisen.

[4] In Anlehnung an [18].

Neben dem modularen Nachweis von Invarianten spielen die **alw**-Eigenschaften beim Nachweis von Lebendigkeitseigenschaften eine Rolle. In diesem Zusammenhang werden meist Eigenschaften der Form

$$\varphi \wedge \neg\psi \ \mathbf{alw} \ \varphi$$

benutzt. Diese Aussage drückt aus, daß $\varphi$ solange gültig bleibt bis $\psi$ gültig wird. Sie drückt gewissermaßen die *Synchronisation* von $\varphi$ mit $\psi$ aus.

### 6.3.3 Leadsto- und Causes-Eigenschaften

Wir wenden uns nun den Lebendigkeitseigenschaften zu. Auch hier führen wir zunächst wieder eine abkürzende Schreibweise für eine Eigenschaft ein, die wir Leadsto-Eigenschaft nennen. Die Aussage $\varphi \leadsto \psi$ bedeutet, daß für jede erreichbare $\varphi$-Scheibe eine davon erreichbare $\psi$-Scheibe existiert.

**Notation 6.16 (Leadsto-Eigenschaft)**

Für zwei Zustandsaussagen (ohne atomare Aussagen) $\varphi$ und $\psi$ definieren wir

$$\varphi \leadsto \psi \mathrel{\widehat{=}} \Box[\varphi \Rightarrow \Diamond\psi]$$

Für unser Mutex-System können wir beispielsweise mit der Aussage $h_1 \leadsto e_1$ ausdrücken, daß der Philosoph „1“ immer, wenn er hungrig ist, auch irgendwann essen kann. Für beliebige Zustandsaussagen $\varphi$ und $\psi$ bedeutet $\varphi \leadsto \psi$ — im Gegensatz zur klassischen Definition der Leadsto-Eigenschaft auf Sequenzen von Zuständen (vgl. [67, 18]) — jedoch nicht, daß jeder sequentielle Beobachter, der einen $\varphi$-Zustand sieht, später einen $\psi$-Zustand sieht. Unsere Definition besagt nur, daß in dem verteilten Ablauf von jeder $\varphi$-Scheibe eine $\psi$-Scheibe erreichbar ist. Für den Spezialfall $\psi \mathrel{\widehat{=}} \#Z \geq 1$ für eine endliche Menge $Z \subseteq S$ stimmt unsere Definition jedoch mit der „sequentiellen“ überein (also insbesondere für obiges Beispiel $h_1 \leadsto e_1$).

Leadsto-Eigenschaften machen nur Aussagen über globale Zustände (d.h. Scheiben) des verteilten Ablaufs. Das Modell der verteilten Abläufe ermöglicht darüber hinaus auch Aussagen über die Erreichbarkeit von beliebigen **co**-Mengen. Eine Möglichkeit bietet der Causes-Operator [78]. Wir können diese Eigenschaft durch folgende temporale Aussage ausdrücken:

$$\Box\triangledown[\varphi \Rightarrow \Diamond\psi]$$

Diese Aussage gilt in einem Ablauf, wenn für jede erreichbare $\varphi$-Menge eine davon erreichbare $\psi$-Menge existiert.
Wir sehen an der Struktur der beiden Aussagen, daß die Leadsto-Eigenschaft und die Causes-Eigenschaft sehr ähnlich sind. Der einzige Unterschied besteht in dem Vorsatz $\Box$ bzw. $\Box\triangledown$. Damit ist die Causes-Eigenschaft eine stärkere Forderung als die Leadsto-Eigenschaft. Deshalb gelten für die Causes-Eigenschaft einige Beweisregeln, die für Leadsto-Eigenschaften nicht gelten.
Eine genauere Diskussion der Causes-Eigenschaft findet sich in [78] und [87]. Wir werden den Causes-Operator im weiteren nicht verwenden, weil unserer Verifikationsregeln (siehe Kapitel 7) auch ohne Verwendung des Causes-Operators zum Beweis der von uns betrachteten Leadsto-Eigenschaften ausreichen. Da der Causes-Operator unseres Erachtens nur zum Beweisen von Leadsto-Eigenschaften aber nicht zum Spezifizieren von Systemen benutzt werden sollte, wird er in dieser Arbeit nirgends benutzt. Deshalb führen wir auch keine Abkürzung für Causes-Eigenschaften ein.

## 6.4 Diskussion der Logik

Wir werden nun einige wichtige Aussagen über die Logik selbst formalisieren. Ein solche Aussage ist, daß wir in einer temporalen Aussage $\varphi$ zustandslogische Teilaussagen durch äquivalente ersetzen können, ohne daß sich dabei die Gültigkeit ändert. So können wir beispielsweise die Aussage $\Box\neg(e_1 \wedge e_2)$ durch $\Box(\neg e_1 \vee \neg e_2)$ ersetzen, weil $\neg(e_1 \wedge e_2)$ und $(\neg e_1 \vee \neg e_2)$ zustandslogisch äquivalent sind.
Außerdem können wir in einer zustandslogischen Tautologie $\varphi$ alle atomaren Aussagen $A_1, \ldots, A_n$ konsistent durch temporale Aussagen ersetzen. Diese Aussage gilt dann (unter gewissen syntaktischen Nebenbedingungen) für jeden Ablauf in allen erreichbaren **co**-Mengen. Beispielsweise ist $A \vee \neg A$ eine zustandslogische Tautologie. Damit ist für jede temporale Aussage $\varphi$ die Aussage $\Box[\varphi \vee \neg\varphi]$ in jedem Ablauf gültig.
Diese Aussagen sind nicht besonders überraschen. Aber insbesondere die zweite Aussage ist von syntaktischen Randbedingungen abhängig, was eine präzise Formulierung der Aussage erforderlich macht. Deshalb führen wir hier zunächst die nötigen Notationen ein, so daß wir die obigen Aussagen formalisieren können. Darüber hinaus führen wir Notationen ein, die wir erst später zur Formulierung von Beweisregeln benötigen.

### 6.4.1 Syntaktische Begriffe

Wir definieren nun die Notationen, die wir zur Formalisierung der obigen Aussagen benötigen. Bevor wir diese Notationen formal einführen, geben wir ihre Bedeutung informell an, so daß die anschließenden induktiven Definitionen beim schnellen Lesen übersprungen werden können.

**Substitution** (Siehe Def. 6.17) Für eine temporale Aussage $\varphi$ bezeichnet $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n]$ die temporale Aussage, in der jedes Vorkommen der atomaren Aussagen $A_1, \ldots, A_n$ in $\varphi$ durch die entsprechenden temporalen Aussagen $\psi_1, \ldots, \psi_n$ ersetzt werden.

Wir nehmen dabei an, daß $A_1, \ldots, A_n$ verschiedene atomare Aussagen sind.

**Umbenennung** (Siehe Def. 6.18) Für eine bijektive Abbildung $u : S \to S'$ und eine temporale Aussage $\varphi \in \mathbf{TA}(S, V, P)$ bezeichnet $u(\varphi)$ die temporale Aussage aus $\mathbf{TA}(S', V, P)$, die wir erhalten, wenn wir in $\varphi$ jede Menge $Z \subseteq S$ durch $u(Z)$ ersetzen.

**Neuer Zustandsraum** (Siehe Def. 6.19) Einer Menge $S$ ordnen wir kanonisch eine disjunkte Menge $\overline{S}$ und einer bijektive Abbildung $\overline{u} : S \to \overline{S}$ mit $\overline{u}(s) = \overline{s}$ zu. Die Aussage $\overline{\varphi}$ bezeichnet dann die Aussage $\overline{u}(\varphi)$.

Damit ist $\overline{\varphi}$ ein Spezialfall der Umbenennung.

**Variable** (Siehe Def. 6.20 und Def. 6.21) Die freien Variablen einer temporalen Aussage $\varphi$ bezeichnen wir mit $\mathbf{frei}(\varphi)$. Die Menge der Variablen, die in $\varphi$ quantifiziert werden bezeichen wir mit $\mathbf{quant}(\varphi)$.

Dabei gilt $v \in \mathbf{quant}(\exists v : \varphi)$ auch dann, wenn $v$ in $\varphi$ nicht vorkommt.

**Zustandsmenge** (Siehe Def. 6.22) Die Menge der Zustände, über die in $\varphi$ eine Aussage gemacht wird, d.h. die in einer Menge $Z$ in $\varphi$ vorkommen, bezeichnen wir mit $\mathbf{symb}(\varphi)$.

Wir formalisieren nun diese Begriffe induktiv über den Aufbau der Terme, Zustandsaussagen und temporalen Aussagen. Dabei fassen wir die Fälle der Operationen für $\wedge$, $\neg$ und $\exists v$ : für Zustands- und temporale Aussagen zusammen.

**Definition 6.17 (Substitution)**

Seien $\psi_1, \ldots, \psi_n$ temporale Aussagen und $A_1, \ldots, A_n \subseteq P$ paarweise verschiedene atomare Aussagen. Wir definieren für temporale Aussagen $\varphi \in \mathbf{TA}(S, V, P)$ die Bedeutung von $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n]$ induktiv:

1. Sei $\varphi \subseteq P$ eine atomare Aussage. Dazu unterscheiden wir zwei Fälle:
   Für $\varphi \notin \{A_1, \ldots, A_n\}$ gilt $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n] \mathrel{\widehat{=}} \varphi$.
   Für $\varphi = A_i$ für ein geeignetes $i$ gilt $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n] \mathrel{\widehat{=}} \psi_i$.
2. Für $\varphi = t \geq t'$ mit $t, t' \in \mathbf{ZT}(S, V)$ gilt $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n] \mathrel{\widehat{=}} t \geq t'$.
3. (a) Für $\varphi = (\psi \wedge \chi)$ gilt $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n] \mathrel{\widehat{=}} (\psi[A_1 \to \psi_1, \ldots, A_n \to \psi_n] \wedge \chi[A_1 \to \psi_1, \ldots, A_n \to \psi_n])$.
   (b) Für $\varphi = \neg\psi$ gilt $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n] \mathrel{\widehat{=}} \neg\psi[A_1 \to \psi_1, \ldots, A_n \to \psi_n]$
   (c) Für $\varphi = \exists v : \psi$ gilt $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n] \mathrel{\widehat{=}} \exists v : \psi[A_1 \to \psi_1, \ldots, A_n \to \psi_n]$
4. (a) Für $\varphi = \diamondsuit\psi$ gilt $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n] \mathrel{\widehat{=}} \diamondsuit\psi[A_1 \to \psi_1, \ldots, A_n \to \psi_n]$.
   (b) Für $\varphi = \Diamond\psi$ gilt $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n] \mathrel{\widehat{=}} \Diamond\psi[A_1 \to \psi_1, \ldots, A_n \to \psi_n]$.
   (c) Für $\varphi = \triangledown\psi$ gilt $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n] \mathrel{\widehat{=}} \triangledown\psi[A_1 \to \psi_1, \ldots, A_n \to \psi_n]$.

**Definition 6.18 (Umbenennung)**

Sei $u : S \to S'$ eine bijektive Abbildung. Wir definieren für $t \in \mathbf{ZT}(S, V)$ und $\varphi \in \mathbf{TA}(S, V, P)$ die Bedeutung von $u(t)$ bzw. $u(\varphi)$ induktiv:

**Terme**
1. Für $t = n \in \mathbb{N}$ gilt $u(t) \mathrel{\widehat{=}} n$.
2. Für $t = v \in V$ gilt $u(v) \mathrel{\widehat{=}} v$.
3. Für $t = \#Z$ gilt $u(t) \mathrel{\widehat{=}} \#u(Z)$, wobei gemäß Konvention $u(Z) = \{u(s) \mid s \in Z\}$ gilt.
4. Für $t = t' + t''$ gilt $u(t) \mathrel{\widehat{=}} u(t') + u(t'')$.

**Aussagen**
1. Für $\varphi \subseteq P$ gilt $u(\varphi) \mathrel{\widehat{=}} \varphi$.
2. Für $\varphi = t \geq t'$ gilt $u(\varphi) \mathrel{\widehat{=}} u(t) \geq u(t')$.

3. (a) Für $\varphi = (\psi \wedge \chi)$ gilt $u(\varphi) \mathrel{\widehat{=}} (u(\psi) \wedge u(\chi))$.
   (b) Für $\varphi = \neg\psi$ gilt $u(\varphi) \mathrel{\widehat{=}} \neg u(\psi)$.
   (c) Für $\varphi = \exists v : \psi$ gilt $u(\varphi) \mathrel{\widehat{=}} \exists v : u(\psi)$.
4. (a) Für $\varphi = \diamondsuit\psi$ gilt $u(\varphi) \mathrel{\widehat{=}} \diamondsuit u(\psi)$.
   (b) Für $\varphi = \Diamond\psi$ gilt $u(\varphi) \mathrel{\widehat{=}} \Diamond u(\psi)$.
   (c) Für $\varphi = \triangledown\psi$ gilt $u(\varphi) \mathrel{\widehat{=}} \triangledown u(\psi)$.

**Definition 6.19 (Neuer Zustandsraum)**

Jeder Menge $S$ ordnen wir eine disjunkte Menge $\overline{S}$ und eine bijektive Abbildung $\overline{u} : S \to \overline{S}$ mit $\overline{u}(s) = \overline{s}$ zu. Wir definieren $\overline{\varphi} \mathrel{\widehat{=}} \overline{u}(\varphi)$.

**Definition 6.20 (Freie Variablen)**

Für die Terme $t \in \mathbf{ZT}(S, V)$ und temporalen Aussagen $\varphi \in \mathbf{TA}(S, V, P)$ definieren wir die Menge der freien Variablen $\mathbf{frei}(t)$ bzw. $\mathbf{frei}(\varphi)$ induktiv:

**Terme**
1. Für $t = n \in \mathbb{N}$ und $t = \#Z$ gilt $\mathbf{frei}(t) \mathrel{\widehat{=}} \emptyset$.
2. Für $t = v \in V$ gilt $\mathbf{frei}(t) \mathrel{\widehat{=}} \{v\}$.
3. Für $t = t' + t''$ gilt $\mathbf{frei}(t) \mathrel{\widehat{=}} \mathbf{frei}(t') \cup \mathbf{frei}(t'')$.

**Aussagen**
1. Für $\varphi \subseteq P$ gilt $\mathbf{frei}(t) \mathrel{\widehat{=}} \emptyset$.
2. Für $\varphi = t \geq t'$ gilt $\mathbf{frei}(\varphi) \mathrel{\widehat{=}} \mathbf{frei}(t) \cup \mathbf{frei}(t')$.
3. (a) Für $\varphi = (\psi \wedge \chi)$ gilt $\mathbf{frei}(\varphi) \mathrel{\widehat{=}} \mathbf{frei}(\psi) \cup \mathbf{frei}(\chi)$.
   (b) Für $\varphi = \neg\psi$ gilt $\mathbf{frei}(\varphi) \mathrel{\widehat{=}} \mathbf{frei}(\psi)$.
   (c) Für $\varphi = \exists v : \psi$ gilt $\mathbf{frei}(\varphi) \mathrel{\widehat{=}} \mathbf{frei}(\psi) \setminus \{v\}$.
4. Für $\varphi = \diamondsuit\psi$, $\varphi = \Diamond\psi$ und $\varphi = \triangledown\psi$ gilt $\mathbf{frei}(\varphi) \mathrel{\widehat{=}} \mathbf{frei}(\psi)$.

**Definition 6.21 (Quantifizierte Variablen)**

Für temporale Aussagen $\varphi \in \mathbf{TA}(S, V, P)$ definieren wir die Menge der quantifizierten Variablen $\mathbf{quant}(\varphi)$ induktiv:

1. Für $\varphi \subseteq P$ und $\varphi = t \geq t'$ gilt $\mathbf{quant}(\varphi) \mathrel{\widehat{=}} \emptyset$.
2. Für $\varphi = t \geq t'$ gilt $\mathbf{quant}(\varphi) \mathrel{\widehat{=}} \emptyset$.
   (a) Für $\varphi = (\psi \wedge \chi)$ gilt $\mathbf{quant}(\varphi) \mathrel{\widehat{=}} \mathbf{quant}(\psi) \cup \mathbf{quant}(\chi)$.
   (b) Für $\varphi = \neg\psi$ gilt $\mathbf{quant}(\varphi) \mathrel{\widehat{=}} \mathbf{quant}(\psi)$.
   (c) Für $\varphi = \exists v : \psi$ gilt $\mathbf{quant}(\varphi) \mathrel{\widehat{=}} \{v\} \cup \mathbf{quant}(\psi)$.
3. (a) Für $\varphi = \diamondsuit\psi$, $\varphi = \Diamond\psi$ und $\varphi = \triangledown\psi$ gilt $\mathbf{quant}(\varphi) \mathrel{\widehat{=}} \mathbf{quant}(\psi)$.

**Definition 6.22 (Zustandsmenge einer Aussage)**

Für Terme $t \in \mathbf{ZT}(S, V)$ und temporalen Aussagen $\varphi \in \mathbf{TA}(S, V, P)$ definieren wir die Menge der vorkommenden Zustände $\mathbf{symb}(t)$ bzw. $\mathbf{symb}(\varphi)$ induktiv:

**Terme**
1. Für $t = n \in \mathbb{N}$ und $t = v \in V$ gilt $\mathbf{symb}(t) \mathrel{\widehat{=}} \emptyset$.
2. Für $t = \#Z$ gilt $\mathbf{symb}(t) \mathrel{\widehat{=}} Z$.
3. Für $t = t' + t''$ gilt $\mathbf{symb}(t) \mathrel{\widehat{=}} \mathbf{symb}(t') \cup \mathbf{symb}(t'')$.

**Aussagen**
1. Für $\varphi \subseteq P$ gilt $\mathbf{symb}(t) \mathrel{\widehat{=}} \emptyset$.
2. Für $\varphi = t \geq t'$ gilt $\mathbf{symb}(\varphi) \mathrel{\widehat{=}} \mathbf{symb}(t) \cup \mathbf{symb}(t')$.
3. (a) Für $\varphi = (\psi \wedge \chi)$ gilt $\mathbf{symb}(\varphi) \mathrel{\widehat{=}} \mathbf{symb}(\psi) \cup \mathbf{symb}(\chi)$.
   (b) Für $\varphi = \neg\psi$ und $\varphi = \exists v : \psi$ gilt $\mathbf{symb}(\varphi) \mathrel{\widehat{=}} \mathbf{symb}(\psi)$.
4. Für $\varphi = \diamondsuit\psi$, $\varphi = \Diamond\psi$ und $\varphi = \fatsemi\psi$ gilt $\mathbf{symb}(\varphi) \mathrel{\widehat{=}} \mathbf{symb}(\psi)$.

Wie man leicht sieht, ist für jede temporale Aussage $\varphi$ die Menge $\mathbf{symb}(\varphi)$ eine endliche Menge.

### 6.4.2 Einige Aussagen über die Logik

Wir können nun die zu Beginn dieses Abschnitts formulierten Eigenschaften formalisieren. Zunächts formulieren wir, daß zustandslogische Teilaussagen durch äquivalente Aussagen ersetzt werden können, ohne dabei die Gültigkeit zu verändern. Das Ersetzen einer Teilaussage $\psi$ durch $\psi'$ formulieren wir dadurch, daß wir in $\varphi$ eine atomare Aussage $A$ zunächst durch $\psi$ und dann durch $\psi'$ substituieren.

**Satz 6.23 (Ersetzen von äquivalenten Teilaussagen)**

Sei $\varphi \in \mathbf{TA}(S, V, \{A\})$ eine temporale Aussage und $\psi, \psi' \in \mathbf{ZA}(S, V, \emptyset)$ Zustandsaussagen ohne atomare Aussagen. Es gilt $\varphi[A \to \psi], \varphi[A \to \psi'] \in \mathbf{TA}(S, V, \emptyset)$.

Sei $\rho$ ein Ablauf, $Q$ eine erreichbare **co**-Menge und $\beta$ eine Belegung für $V$. Wenn $\psi \Leftrightarrow \psi'$ allgemeingültig ist, dann gilt $\rho, Q, \beta \models \varphi[A \to \psi]$ genau dann, wenn $\rho, Q, \beta \models \varphi[A \to \psi']$ gilt; insbesondere gilt $\rho \models \varphi[A \to \psi]$ genau dann, wenn $\rho \models \varphi[A \to \psi']$.

In zustandslogischen Tautologien können alle atomaren Aussagen durch beliebige temporale Aussagen ersetzen. Wenn dabei keine „unerwünschten“ Bindungen entstehen, ist diese Aussage in jedem Ablauf in jeder erreichbaren **co**-Menge gültig. Eine „unerwünschte“ Bindung entsteht

beispielsweise, wenn wir in der zustandslogischen Tautologie $A \vee \neg\exists v : A$ die atomare Aussage $A$ durch $v = 0$ ersetzen. Wir erhalten so die Aussage $v = 0 \vee \neg\exists v : v = 0$. Diese Aussage ist bereits zustandslogisch nicht allgemeingültig. Der Grund dafür ist, daß $A$ unter verschiedenen Bindungen von $v$ vorkommt. Um dies auszuschließen verlangen wir, daß alle freien Variablen der Aussagen, die wir für atomare Aussagen substituieren, in $\varphi$ nicht quantifiziert werden.

**Satz 6.24 (Einsetzen in zustandslogische Tautologien)**

Sei $\varphi \in \mathbf{ZA}(S, V, \{A_1, \ldots, A_n\})$ eine allgemeingültige Zustandsaussage und $\psi_1, \ldots, \psi_n \in \mathbf{TA}(S, V)$. Es gilt $\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n] \in \mathbf{TA}(S, V)$.

Wenn gilt $\mathbf{quant}(\varphi) \cap (\mathbf{frei}(\psi_1) \cup \ldots \cup \mathbf{frei}(\psi_n)) = \emptyset$, dann gilt für jeden Ablauf $\rho \models \Box\varphi[A_1 \to \psi_1, \ldots, A_n \to \psi_n]$.

Wir formulieren nun noch eine Aussage, die ausnutzt, daß die Variablenbelegung $\beta$ bei der Interpretation in jeder **co**-Menge eines Ablaufs dieselbe ist. Wenn initial $v = 1$ gilt, dann gilt diese Aussage in allen erreichbaren Scheiben des Ablaufes. Beispielsweise gilt in jedem Ablauf $\rho \models v = 1 \Rightarrow \Box v = 1$. Allgemein können wir diese Aussage für jede Zustandsaussage $\chi$ formulieren, für die $\mathbf{symb}(\chi) = \emptyset$ gilt. Dabei müssen wir wie zuvor darauf achten, daß die freien Variablen von $\chi$ beim Einsetzen nicht gebunden werden.

**Satz 6.25 (Unveränderliche Variablen)**

Sei $\chi \in \mathbf{ZA}(\emptyset, V, \emptyset)$, $\psi, \psi' \in \mathbf{TA}(S, V)$ und $\varphi \in \mathbf{TA}(S, V, \{A\})$ mit $\mathbf{quant}(\varphi) \cap \mathbf{frei}(\chi) = \emptyset$.

Wenn für einen Ablauf $\rho$ gilt $\rho \models \varphi[A \to \psi]$, dann gilt auch $\rho \models \chi \Rightarrow \varphi[A \to (\chi \wedge \psi) \vee (\neg\chi \wedge \psi')]$

**Beweis:** Aus der Voraussetzung $\chi$ folgt, daß $\chi$ unter der betrachten Belegung gültig ist. Da die freien Variablen von $\chi$ von $\varphi$ nicht quantifiziert werden, wird $\chi$ an der Stelle $A$ so ausgewertet wie am Anfang; nach der Prämisse also zu wahr. Deshalb wird $(\chi \wedge \psi) \vee (\neg\chi \wedge \psi')$ bei der induktiven Auswertung immer wie $\psi$ ausgewertet. □

Zuletzt zeigen wir, daß die Umbenennung von Beschriftungen eines Ablaufes durch eine entsprechende Umbenennung in den temporalen Aussagen nachgebildet werden kann.

**Satz 6.26 (Umbenennung)**

Sei $\varphi \in \mathbf{TA}(S, V)$ eine temporale Aussage, $u : S \to S'$ eine bijektive Abbildung und $\rho$ ein Ablauf.

Dann gilt $\rho \models \varphi$ genau dann, wenn $u \circ \rho \models u(\varphi)$.

**Beweis:** Den beiden Abläufen $\rho$ und $u \circ \rho$ liegt dasselbe Kausalnetz zugrunde. Es genügt zu zeigen, daß für jede **co**-Menge $Q$ von $K$, jede Belegung $\beta$ und jede Zustandsaussage $\varphi$ die Aussage $\rho, Q, \beta \models \varphi$ genau dann gilt, wenn $u \circ \rho, Q, \beta \models u(\varphi)$.

Dazu zeigen wir $\beta_{M_{\rho,Q}}(\#Z) = \beta_{M_{u\circ\rho,Q}}(u(\#Z))$:

$$
\begin{array}{lcl}
\beta_{M_{\rho,Q}}(\#Z) & = & \text{(nach Def.)} \\
|\{b \in Q \mid \rho(b) \in Z\}| & = & \text{(wg. Injektivität von } u\text{)} \\
|\{b \in Q \mid u \circ \rho(b) \in u(Z)\}| & = & \text{(wg. Injektivität } u\text{)} \\
|\{u(b) \mid b \in Q, u \circ \rho(b) \in u(Z)\}| & = & \text{(nach Def.)} \\
\beta_{M_{u\circ\rho,Q}}(u(\#Z)) & &
\end{array}
$$

Damit folgt die Behauptung induktiv über den Aufbau der Aussage $\varphi$. □

Zusammen mit den Aussagen über die Umbenennungen eines Systems folgt dann, daß eine Systemkomponente $\Sigma$ eine temporale Aussage $\varphi$ genau dann erfüllt, wenn die umbenannte Systemkomponente $u(\Sigma)$ die temporale Aussage $u(\varphi)$ erfüllt. Die Bijektivität haben wir hier nur gefordert, weil eine Umbenennungen eines Systems bijektiv sein muß; in Satz 6.26 hätte es genügt, die Injektivität von $u$ zu fordern.

## 6.5 Literatur

Neben der klassischen temporalen Logik [46, 54], die auf sequentiellen Abläufen interpretiert wird[5], gibt es inzwischen eine Reihe von temporalen Logiken, die auf nicht-sequentiellen Abläufen interpretiert werden. Ein kurzer Überblick über aktuelle Ansätze findet sich in [33]. Diese Logiken werden über verschiedenen Modellen verteilter Abläufe interpretiert. So benutzen beispielsweise Mukund und Thiagarajan [64] *event structures* [89], Peled und Pnueli [70] *traces* [58] und Esparza [27] *branching processes* [26].

[5] In [54] wird insbesondere gezeigt, wie die temporale Logik auf Ablaufsequenzen von S/T-Systemen interpretiert wird.

Unabhängig von dem Ablaufmodell, das den Logiken zugrundeliegt, können wir zwei Arten von Logiken unterscheiden: Die einen machen nur Aussagen über globale Zustände in einem Ablauf [71, 81, 43]. Die anderen machen nur Aussagen über lokale Zustände [73, 72, 64]; mit diesen Logiken können insbesondere Invarianten nicht formuliert werden.

Wir wollen hier die *Interleaving Set Temporal Logic* (ISTL) [43, 70] besonders hervorheben: ISTL benutzt die branching-time Logik CTL, die über den Scheiben *eines* verteilten Ablaufs interpretiert wird. Die Übergangsrelation zwischen den Scheiben entspricht dabei der unmittelbaren Nachfolgerrelation. Damit kann in ISTL ein Operator formuliert werden, der genau unserem Leadsto-Operator entspricht. Aussagen über **co**-Mengen sind in ISTL aber nicht möglich. Deshalb kann beispielsweise der Causes-Operator in ISTL nicht ausgedrückt werden.

In unserer Logik orientieren wir uns im wesentlichen an der klassischen temporalen Logik [54]. Neben den klassischen Operatoren $\Diamond$ und $\Diamond\!\!\!\!-$ definieren wir den zusätzlichen Operator $\triangledown$, mit dem wir Aussagen über Ausschnitte eines Ablaufs machen können. Dieser Operator ist dem SUB-Operator aus [77] sehr ähnlich. Damit ist es möglich sowohl Aussagen über globale Zustände als auch über lokale Zustände zu machen. Mit diesen Operatoren kann insbesondere der Causes-Operator aus [78] ausgedrückt werden. Ebenso können die Operatoren $\rightsquigarrow$, **alw**, **inv**(.) **p-i**(.,.,.) mit den drei Grund-Operatoren unserer Logik ausgedrückt werden.

Die Verwendung von Invarianten und Leadsto-Eigenschaften zur Spezifikation von Systemen wurde im sequentiellen Fall von Owicki und Lamport [67] vorgeschlagen. Wir haben den Leadsto-Operator auf verteilte Abläufe übertragen. Der **alw**-Operator wurde in etwas anderer Form in [50] vorgeschlagen. Wie bei uns, war auch dort die Beschreibung des erwarteten Umgebungsverhaltens ein Grund für die „*allowed-changes*“. Die Beschreibung der erlaubten Zustandsübergänge taucht auch in Lamports *Temporal Logic of Actions* (TLA) [51] in Form der Aktion wieder auf. Darüber hinaus wird die Beschreibung von erlaubten Zustandsübergängen in vielen anderen Spezifikationsformalismen (in verschiedenen Varianten) benutzt. Insbesondere dient der *unless*-Operator in UNITY [18] als Grundlage für eine kompositionale Beweismethode.

S-Invarianten sind das wesentliche Werkzeug zum Beweisen von Invarianten von S/T-Systemen [80]. Auch partielle S-Invarianten tauchen in der Literatur auf: Im Modulkonzept von [82, 83] wird gefordert, daß Paare von Ein-/Ausgabe-Schnittstellen eines Moduls von

der Umgebung zu S-Invarianten ergänzt werden. Diese Forderung wird dort jedoch nicht logisch, sondern strukturell aufgestellt. Weil wir S-Invarianten und partielle S-Invarianten mit Hilfe der Logik ausdrücken können, sind diese Eigenschaft viel flexibler einsetzbar. Insbesondere können wir sie in der Spezifikation von Systemkomponenten benutzen, um die Gültigkeit bestimmter partieller S-Invarianten in der Umgebung einer Systemkomponente anzunehmen.

Neben Logiken für verteilte Abläufe von Systemen, gibt es auch Logiken für Petrinetze, die auf dem Zustandsgraphen definiert sind. Eine sehr allgemeine Logik wird beispielsweise in [15] vorgestellt. Dort werden Eigenschaften mit Hilfe des modalen $\mu$-Kalküls formuliert. Weil diese Logiken nicht auf der Menge der Abläufe, sondern auf dem Zustandsgraph eines Systems interpretiert werden, sind sie *branching-time* Logiken, die für uns in dieser Arbeit nicht interessant sind.

# Kapitel 7

# Verifikation

In diesem Kapitel beschäftigen wir uns mit der *Verifikation* von Systemkomponenten. Dabei verstehen wir unter Verifikation, den formalen Nachweis, daß eine Systemkomponente ein Modul implementiert. Dazu geben wir Beweisregeln an, mit denen wir ausgehend von der Struktur der Systemkomponente temporale Aussagen herleiten können, die in der Systemkomponente gelten. Da uns hier nur der „temporale Aspekt“ von Beweisen interessiert, geben wir keine Regeln für die Zustandslogik an. Insbesondere geben wir keine Regeln zum Nachweis der Allgemeingültigkeit von Zustandsaussagen an.

Die temporallogischen Regeln können wir grob in drei Gruppen einteilen: Die erste Gruppe enthält Regeln, die für beliebige temporale Aussagen gelten. Die zweite Gruppe enthält die Regeln zum Nachweis von Invarianten; dazu gehören auch die Regeln für S-Invarianten, partielle S-Invarianten und **alw**-Eigenschaften. Die dritte Gruppe enthält die Regeln zum Nachweis von Leadsto-Eigenschaften. Wir geben hier jedoch nur Regeln an, die entweder später in den Beispielen benötigt werden oder eine besondere Bedeutung in unserer Logik besitzen.

Um den Zusammenhang zwischen der Struktur einer Systemkomponente und ihren temporallogisch formulierten Eigenschaften herzustellen, führen wir den Begriff der *Zusicherung* ein. Eine Zusicherung stellt den Zusammenhang zwischen zwei aufeinanderfolgenden Zuständen in einem Ablauf der Systemkomponente her. Die Gültigkeit einer Zusicherung läßt sich auf die Allgemeingültigkeit von bestimmten Zustandsaussagen zurückführen. Deshalb ist es nicht notwendig, einen eigenen Kalkül (wie z.B. in [28, 38]) für Zusicherungen einzuführen.

Am Ende dieses Kapitels werden wir die Anwendung einiger Regeln anhand eines kleinen Beispiels demonstrieren. Im nächsten Kapitel werden wir dann ein etwas größeres Beispiel betrachten.

## 7.1 Zusicherungen

Für zwei Zustandsaussagen $\varphi$ und $\psi$ und eine Transition $t$ gilt die *Zusicherung* $\langle\varphi\rangle\ t\ \langle\psi\rangle$, wenn für jede Markierung, in der $t$ aktiviert ist und in der $\varphi$ gilt, nach dem Schalten von $t$ die Aussage $\psi$ gilt.

**Definition 7.1 (Zusicherung)**

Sei $t$ eine Transition eines verteilten Transitionssystems und $\varphi, \psi \in \mathbf{ZA}(S, V)$ Zustandsaussagen. Die *Zusicherung* $\langle\varphi\rangle\ t\ \langle\psi\rangle$ gilt genau dann, wenn für alle Markierungen $M$ und $M'$ mit $M \xrightarrow{t} M'$ und jede Belegung $\beta$ für $V$ mit $M, \beta \models \varphi$ auch $M', \beta \models \psi$ gilt.

Für eine Menge von Transitionen $T$ über $S$ gilt die Zusicherung $\langle\varphi\rangle\ T\ \langle\psi\rangle$ genau dann, wenn für jede Transition $t \in T$ die Zusicherung $\langle\varphi\rangle\ t\ \langle\psi\rangle$ gilt.

Im Hinblick auf den Beweis von temporalen Aussagen einer Systemkomponente sind Zusicherungen deshalb interessant, weil Zusicherungen mit Satz 2.37 auch Aussagen über aufeinanderfolgende **co**-Mengen $Q \xrightarrow{e} Q'$ des Ablaufs machen. Wenn beispielsweise für eine Systemkomponente $\Sigma$ die Zusicherung $\langle\varphi\rangle\ T_\Sigma \cup T_\Sigma^U\ \langle\varphi \vee \psi\rangle$ gilt, dann gilt für jeden Ablauf $\rho$ von $\Sigma$ die Aussage $\rho \models \varphi\ \mathbf{alw}\ \psi$.
Wir betrachten nun ein Beispiel für eine Zusicherung. Für die Transition $t$ aus Abb. 7.1 gilt die Zusicherung $\langle a\rangle\ t\ \langle a \wedge c \wedge d\rangle$. Das heißt, wenn $t$ in einem Zustand schaltet, in dem $a$ gilt, dann gilt nach dem Schalten von $t$ die Aussage $a \wedge c \wedge d$.
Die Gültigkeit einer Zusicherung führen wir nun auf die Allgemeingültigkeit einer Zustandsaussage zurück. Diese Zustandsaussage beschreibt den Zusammenhang zwischen der Markierung vor und nach dem Schalten der Transition $t$. Da eine Zustandsaussage formal nur über einer Multimenge interpretiert wird, wenden wir zur Repräsentation der zweiten Multimenge einen „Trick“ an (vgl. Variablen $x$ und $x'$ in [51, 54]): Wir erweitern $S$ um eine disjunkte Menge $\overline{S}$ (vgl. Def. 6.19). Die Symbole aus $S$ beschreiben die Markierung vor dem Schalten der Transition, die Symbole aus $\overline{S}$ beschreiben die Markierung nach dem Schalten der Transition. Formal werden die Zustandsaussagen über *einer* Multimenge über $S \cup \overline{S}$ interpretiert.

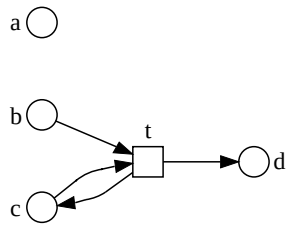



Abbildung 7.1: Es gilt die Zusicherung $\langle a \rangle \ t \ \langle a \wedge c \wedge d \rangle$

Beispielsweise können wir die Aktiviertheit von Transition $t$ aus Abb. 7.1 und die Veränderung der Markierung, die durch das Schalten von $t$ bewirkt wird, durch die Aussage

$$\tau \mathrel{\widehat{=}} (b \wedge c) \ \wedge \ (\#\{\overline{a}\} = \#\{a\} \ \wedge \ \#\{\overline{c}\} = \#\{c\} \ \wedge \ \#\{\overline{d}\} = \#\{d\} + 1)$$

beschreiben. Die Aussage $(b \wedge c)$ beschreibt die Aktiviertheit von $t$; der Rest des Ausdrucks beschreibt die Veränderung der Markierungen für die verschiedenen Stellen $a$, $c$ und $d$. Dabei beschreiben wir nur die Veränderung der Stellen, die in der betrachteten Zusicherung relevant sind. Beispielsweise haben wir die Veränderung von $b$ nicht beschrieben, weil $b$ in der Aussage $a \wedge c \wedge d$ der Zusicherung $\langle a \rangle \ t \ \langle a \wedge c \wedge d \rangle$ nicht vorkommt. Dieser Zusicherung ordnen wir die Zustandsaussage

$$a \wedge \tau \ \Rightarrow \ \overline{a} \wedge \overline{c} \wedge \overline{d}$$

zu. Wenn wir alle Abkürzungen expandieren erhalten wir die Aussage

$$\left( \begin{array}{l} \#\{a\} \geq 1 \wedge \\ (\#\{b\} \geq 1 \wedge \#\{c\} \geq 1) \wedge \\ (\#\{\overline{a}\} = \#\{a\} \ \wedge \\ \quad \#\{\overline{c}\} = \#\{c\} \ \wedge \\ \quad \#\{\overline{d}\} = \#\{d\} + 1) \end{array} \right) \ \Rightarrow \ (\#\{\overline{a}\} \geq 1 \wedge \#\{\overline{c}\} \geq 1 \wedge \#\{\overline{d}\} \geq 1)$$

Diese Aussage ist zustandslogisch allgemeingültig und somit gilt die Zusicherung $\langle a \rangle \ t \ \langle a \wedge c \wedge d \rangle$.
Wir ordnen nun allgemein jeder Zusicherung $\langle \varphi \rangle \ t \ \langle \psi \rangle$ eine zustandslogische Aussage zu, die genau dann allgemeingültig ist, wenn die Zusicherung gilt. Damit ist das Überprüfen der Gültigkeit einer Zusicherung auf das Überprüfen der Allgemeingültigkeit einer Zustandsaussage zurückgeführt.

**Satz 7.2 (Gültigkeit einer Zusicherung)**

Sei $t$ eine Transition eines verteilten Transitionssystems und $\varphi, \psi \in \mathbf{ZA}(S, V)$ Zustandsaussagen. Der Zusicherung $\langle\varphi\rangle\ t\ \langle\psi\rangle$ ordnen wir die Zustandsaussage

$$\mathcal{Z}(\varphi, t, \psi) \mathrel{\widehat{=}} \varphi \wedge (\bigwedge {}^\bullet t) \wedge \left( \begin{array}{l} \bigwedge_{s \in Z_u} \#\{\overline{s}\} = \#\{s\} \ \wedge \\ \bigwedge_{s \in Z_-} \#\{\overline{s}\} + 1 = \#\{s\} \ \wedge \\ \bigwedge_{s \in Z_+} \#\{\overline{s}\} = \#\{s\} + 1 \end{array} \right) \Rightarrow \overline{\psi}$$

mit

$$\begin{array}{l} Z_u \mathrel{\widehat{=}} (\mathbf{symb}(\psi) \cap {}^\bullet t \cap t^\bullet) \cup (\mathbf{symb}(\psi) \setminus ({}^\bullet t \cup t^\bullet)) \\ Z_- \mathrel{\widehat{=}} \mathbf{symb}(\psi) \cap ({}^\bullet t \setminus t^\bullet) \\ Z_+ \mathrel{\widehat{=}} \mathbf{symb}(\psi) \cap (t^\bullet \setminus {}^\bullet t) \end{array}$$

zu.

Die Zusicherung $\langle\varphi\rangle\ t\ \langle\psi\rangle$ gilt genau dann, wenn die Zustandsaussage $\mathcal{Z}(\varphi, t, \psi)$ über $S \cup \overline{S}$ allgemeingültig ist.

**Beweis:**

**„⇐“:** Wir zeigen: Wenn die Zusicherung $\langle\varphi\rangle\ t\ \langle\psi\rangle$ nicht gilt, dann ist die Zustandsaussage $\mathcal{Z}(\varphi, t, \psi)$ nicht allgemeingültig. Wenn $\langle\varphi\rangle\ t\ \langle\psi\rangle$ nicht gilt, dann gibt zwei es Markierungen $M \xrightarrow{t} M'$ und eine Belegung $\beta$ mit $M, \beta \models \varphi$ und $M', \beta \not\models \psi$. Da $t$ in $M$ aktiviert ist gilt auch $M, \beta \models \bigwedge {}^\bullet t$. Wir definieren die Multimenge $\widehat{M}$ über $S \cup \overline{S}$ durch $\widehat{M}(s) \mathrel{\widehat{=}} M(s)$ und $\widehat{M}(\overline{s}) = M'(s)$ für alle $s \in S$. Durch Induktion über den Aufbau der Zustandsaussagen (und der Terme) zeigt man leicht, daß $\widehat{M}, \beta \models \varphi \wedge \bigwedge {}^\bullet t$ und $\widehat{M}, \beta \not\models \overline{\psi}$. Gemäß Definition von $M \xrightarrow{t} M'$ gilt auch $\widehat{M}, \beta \models (\bigwedge_{s \in Z_u} \#\{\overline{s}\} = \#\{s\} \wedge \bigwedge_{s \in Z_-} \#\{\overline{s}\} + 1 = \#\{s\} \wedge \bigwedge_{s \in Z_+} \#\{\overline{s}\} = \#\{s\} + 1)$.

Zusammen gilt also $\widehat{M}, \beta \not\models \mathcal{Z}(\varphi, t, \psi)$. Damit ist $\mathcal{Z}(\varphi, t, \psi)$ nicht allgemeingültig.

**„⇒“:** Wir zeigen: Wenn die Zustandsaussage $\mathcal{Z}(\varphi, t, \psi)$ nicht allgemeingültig ist, dann gilt auch die Zusicherung $\langle\varphi\rangle\ t\ \langle\psi\rangle$ nicht. Wenn $\mathcal{Z}(\varphi, t, \psi)$ nicht allgemeingültig ist, dann gibt es eine Markierung $\widehat{M}$ über $S \cup \overline{S}$ und eine Belegung $\beta$ mit $\widehat{M}, \beta \not\models \mathcal{Z}(\varphi, t, \psi)$. Also gilt $\widehat{M}, \beta \models \varphi \wedge \bigwedge {}^\bullet t \wedge (\bigwedge_{s \in Z_u} \#\{\overline{s}\} = \#\{s\} \wedge \bigwedge_{s \in Z_-} \#\{\overline{s}\} + 1 = \#\{s\} \ \wedge \ \bigwedge_{s \in Z_+} \#\{\overline{s}\} = \#\{s\} + 1)$ und $\widehat{M}, \beta \not\models \overline{\psi}$.

Wir definieren die Multimengen $M$ über $S$ durch $M(s) \mathrel{\widehat{=}} \widehat{M}(s)$ für alle $s \in S$. Man zeigt leicht induktiv über den Aufbau der Zustandsaussagen, daß gilt $M, \beta \models \varphi$ und $M, \beta \models \bigwedge {}^\bullet t$. Also ist $t$ in $M$ aktiviert. Sei $M'$ die Markierung mit $M \xrightarrow{t} M'$. Für jedes $s \in \mathbf{symb}(\psi)$ gilt wegen $\widehat{M}, \beta \models (\bigwedge_{s \in Z_u} \#\{\overline{s}\} = \#\{s\} \wedge \bigwedge_{s \in Z_-} \#\{\overline{s}\} + 1 = \#\{s\} \wedge \bigwedge_{s \in Z_+} \#\{\overline{s}\} = \#\{s\} + 1)$ und der Def. von $M \xrightarrow{t} M'$: $M'(s) = \widehat{M}(\overline{s})$. Induktiv über den Aufbau von $\psi$ kann man damit zeigen, daß mit $\widehat{M}, \beta \not\models \overline{\psi}$ auch $M', \beta \not\models \psi$ gilt. Also gilt die Zusicherung $\langle \varphi \rangle \ t \ \langle \psi \rangle$ nicht.

□

Damit ist die Gültigkeit einer Zusicherung auf die Allgemeingültigkeit einer Zustandsaussage zurückgeführt. Wie man die Allgemeingültigkeit einer Zustandsaussage beweist, ist jedoch nicht Gegenstand dieser Arbeit.

In den Beweisregeln kommen häufig Zusicherungen für alle (potentiellen) Transitionen der Umgebung einer Systemkomponente $\Sigma$ vor. Diese Menge ist im allgemeinen unendlich, so daß wir bei der Anwendung von Satz 7.2 die Allgemeingültigkeit von unendlich vielen Zustandsaussagen überprüfen müssen. Wir werden nun zeigen, daß wir eine Zusicherung für alle Transitionen der Umgebung einer Systemkomponente auf die Allgemeingültigkeit einer einzigen Zustandsaussage zurückführen können. Die Idee, die dieser Zustandsaussage zugrunde liegt, ist dieselbe, wie in Satz 7.2. Sie beschreibt die möglichen Transitionen der Umgebung in Abhängigkeit von der Schnittstelle der Systemkomponente.

**Satz 7.3 (Zusicherung für die Umgebung)**

Sei $\Sigma$ eine Systemkomponente über $S$ (mit unendlich vielen externen Stellen) und $\varphi, \psi \in \mathbf{ZA}(S, V)$ zwei Zustandsaussagen. Der Zusicherung $\langle \varphi \rangle \ T_\Sigma^U \ \langle \psi \rangle$ ordnen wir die Zustandsaussage $\mathcal{Z}(\varphi, T_\Sigma^U, \psi) \mathrel{\widehat{=}}$

$$\varphi \wedge \left( \begin{array}{l} \bigwedge_{s \in Z_u} \#\{\overline{s}\} = \#\{s\} \ \wedge \\ \bigwedge_{s \in Z_x} (\#\{\overline{s}\} \leq \#\{s\} + 1 \ \wedge \ \#\{\overline{s}\} + 1 \leq \#\{s\}) \wedge \\ \bigwedge_{s \in Z_i} (\#\{\overline{s}\} = \#\{s\} \ \vee \ \#\{\overline{s}\} = \#\{s\} + 1) \wedge \\ \bigwedge_{s \in Z_o} (\#\{\overline{s}\} = \#\{s\} \ \vee \ \#\{\overline{s}\} + 1 = \#\{s\}) \end{array} \right) \Rightarrow \overline{\psi}$$

zu, wobei $Z_u \mathrel{\widehat{=}} \mathbf{symb}(\psi) \cap Z_\Sigma$, $Z_x \mathrel{\widehat{=}} \mathbf{symb}(\psi) \cap S_\Sigma^U$, $Z_i \mathrel{\widehat{=}} \mathbf{symb}(\psi) \cap I_\Sigma$ und $Z_o \mathrel{\widehat{=}} \mathbf{symb}(\psi) \cap O_\Sigma$.

Die Zusicherung $\langle \varphi \rangle \ T_\Sigma^U \ \langle \psi \rangle$ gilt genau dann, wenn $\mathcal{Z}(\varphi, T_\Sigma^U, \psi)$ allgemeingültig ist.

**Beweis:** Der Beweis geht im wesentlichen analog zum Beweis von Satz 7.2. □

Damit haben wir alle Zusicherungen, die später in den Beweisregeln vorkommen, auf die Allgemeingültigkeit von Zustandsaussagen zurückgeführt. Bei der Verifikation der Beispiele werden wir Zusicherungen als Argumente ohne weitere formale Begründung angeben. In der Praxis sollte an dieser Stelle die Unterstützung durch ein Computerwerkzeug beginnen, das für eine gegebene Systemkomponente die Gültigkeit einer Zusicherung automatisch überprüft. Die Überprüfung von Zusicherungen ist zu stupide (und deshalb zu fehleranfällig), um sie dem Menschen zu überlassen.

## 7.2 Logik, Module, Systemkomponenten

Bevor wir die Beweisregeln einführen, müssen wir uns überlegen was wir eigentlich beweisen wollen. Im einfachsten Fall müssen wir zeigen, daß jeder Ablauf $\rho$ einer Systemkomponente $\Sigma$ eine temporale Aussage $\varphi$ erfüllt. Wir schreiben dafür $\Sigma \models \varphi$.
Beim Nachweis, daß eine Systemkomponente $\Sigma$ ein Modul $\mathcal{M}$ implementiert, müssen wir zeigen, daß eine temporale Aussage $\varphi$ von $\Sigma$ unter der Annahme erfüllt wird, daß die Umgebung bestimmte Eigenschaften $\mathcal{E}$ gewährleistet. Die Aussage $\varphi$ ist dabei die Inschrift eines G-Knotens $g$ eines RG-Graphen und $\mathcal{E}$ ist die Menge der Inschriften der Vorgänger von $g$ im RG-Graphen. Wir schreiben dafür $\Sigma, \mathcal{E} \models \varphi$.

**Definition 7.4 ($\Sigma, \mathcal{E} \models \varphi$)**

Sei $\Sigma$ eine Systemkomponente, $\varphi$ eine temporale Aussage und $\mathcal{E}$ eine Menge von temporalen Aussagen.

Wir schreiben $\Sigma, \mathcal{E} \models \varphi$, wenn jeder Ablauf $\rho \in \mathbf{R}(\Sigma)$ der Systemkomponente, der jedes $\psi \in \mathcal{E}$ erfüllt, auch $\varphi$ erfüllt; wir sagen dann, daß $\Sigma$ *die Aussage $\varphi$ unter der Annahme $\mathcal{E}$ erfüllt.*

$\Sigma \models \varphi$ ist dann eine Abkürzung für $\Sigma, \emptyset \models \varphi$ und wir sagen für $\Sigma \models \varphi$, daß $\Sigma$ die Aussage $\varphi$ erfüllt.
Zum Nachweis, daß eine Systemkomponente $\Sigma$ ein Modul $\mathcal{M}$ implementiert, müssen wir für jeden G-Knoten $g$ des zugehörigen RG-Graphen $\Sigma, \mathcal{E} \models \varphi$ zeigen[1], wobei $\varphi$ die Inschrift des Knotens $g$ ist und $\mathcal{E}$

[1] $\Sigma, \mathcal{E} \models \varphi$ ist die logische Formulierung der Forderung $\mathbf{R}(\Sigma) \cap \mathcal{P}_g^{\prec} \subseteq \mathcal{P}_g$.

die Menge der Inschriften der Vorgänger von $g$. Zum Beweis von Aussagen dieser Form geben wir in den nachfolgenden Abschnitten Regeln an.

Beim Rechnen mit Modulen (siehe Kapitel 5) benötigen wir eine etwas andere Art von Aussagen, da wir beim Rechnen mit Modulen die Implementierung $\Sigma$ des Moduls nicht kennen. Beispielsweise müssen wir beim Eliminieren eines R-Knotens $r$ des RG-Graphen mit der Inschrift $\varphi$ zeigen[2] $\mathcal{E} \models \varphi$, wobei $\mathcal{E}$ die Menge der Inschriften der Vorgängerknoten von $r$ ist. Zum Nachweis von Aussagen dieser Form führen wir keine gesonderten Beweisregeln ein, weil sie im wesentlichen mit den Regeln für Aussagen der Form $\Sigma, \mathcal{E} \models \varphi$ übereinstimmen. Lediglich die Regeln, die in den Voraussetzungen Aussagen über die Struktur von $\Sigma$ machen, sind für den Nachweis von Aussagen der Form $\mathcal{E} \models \varphi$ nicht benutzbar. Wir benutzen deshalb zum Beweis von Aussagen der Form $\mathcal{E} \models \varphi$ die Regeln für $\Sigma, \mathcal{E} \models \varphi$, in denen keine Annahme über die Struktur von $\Sigma$ gemacht wird.

## 7.3 Allgemeine Verifikationsregeln

Die *Beweisregeln*, die wir im folgenden angeben, haben die Form

| (Regelname) | Voraussetzungen / Schlußfolgerung | Nebenbedingungen |
|---|---|---|

Die *Voraussetzungen* einer Regel sind meist Aussagen der Form $\Sigma, \mathcal{E} \models \varphi$, Zusicherungen $\langle\varphi\rangle\ t\ \langle\psi\rangle$ oder die zustandslogische Allgemeingültigkeit einer Aussage $\models \varphi$. Die *Nebenbedingungen* der Regel sind meist syntaktische Einschränkungen der Aussagen, die in den Voraussetzungen vorkommen. Die *Schlußfolgerung* einer Regel ist *immer* eine Aussage der Form $\Sigma, \mathcal{E} \models \varphi$. Jede Regel besitzt einen Namen, so daß wir uns in einem Beweis auf die angewendete Regel beziehen können.
Eine Regel heißt *korrekt*, wenn aus der Gültigkeit aller Voraussetzungen und der Nebenbedingungen die Gültigkeit der Schlußfolgerung folgt. Wir sind natürlich nur an korrekten Beweisregeln interessiert. Damit können wir aus gültigen Aussagen weitere gültige Aussagen *herleiten*.

[2] $\mathcal{E} \models \varphi$ ist die temporallogische Formulierung der Forderung $\mathcal{P}_r^{\prec} \subseteq \mathcal{P}_r$.

In Tabelle 7.1 sind Regeln zum Beweis von allgemeinen temporalen Aussagen zusammengestellt. Diese Regeln werden häufig implizit oder in Kombination mit der Anwendung anderer Regeln angewendet. Die Regel (EQUIV) drückt aus, daß in einer temporalen Aussage eine zustandslogische Teilaussagen durch eine zustandslogisch äquivalente ersetzt werden kann (vgl. Satz 6.23). Regel (TAUT) besagt, daß jede allgemeingültige Zustandsaussage insbesondere am Anfang jedes Ablaufs gilt. Die Regeln (MP) (abgekürzt für *Modus Ponens*) und (CONJ) verknüpfen bereits bewiesene temporallogische Aussagen miteinander — wie in der klassischen mathematische Logik (siehe z.B. [25])

$$\text{(EQUIV)}\ \frac{\models \psi \Leftrightarrow \psi' \quad \Sigma, \mathcal{E} \models \varphi[A \to \psi]}{\Sigma, \mathcal{E} \models \varphi[A \to \psi']} \quad \begin{array}{l} \psi, \psi' \in \mathbf{ZA}(S, V) \\ \varphi \in \mathbf{TA}(S, V, \{A\}) \end{array}$$

$$\text{(TAUT)}\ \frac{\models \varphi}{\Sigma, \mathcal{E} \models \varphi} \quad \varphi \in \mathbf{ZA}(S, V)$$

$$\text{(MP)}\ \frac{\Sigma, \mathcal{E} \models \varphi \Rightarrow \psi \quad \Sigma, \mathcal{E} \models \varphi}{\Sigma, \mathcal{E} \models \psi} \quad \varphi, \psi \in \mathbf{TA}(S, V)$$

$$\text{(CONJ)}\ \frac{\Sigma, \mathcal{E} \models \varphi \quad \Sigma, \mathcal{E} \models \psi}{\Sigma, \mathcal{E} \models \varphi \wedge \psi} \quad \varphi, \psi \in \mathbf{TA}(S, V)$$

$$\text{(ASS)}\ \frac{}{\Sigma, \mathcal{E} \models \varphi} \quad \begin{array}{l} \mathcal{E} \subseteq \mathbf{TA}(S, V) \\ \varphi \in \mathcal{E} \end{array}$$

$$\text{(S-ASS)}\ \frac{\Sigma, \mathcal{E}' \models \varphi}{\Sigma, \mathcal{E} \models \varphi} \quad \begin{array}{l} \mathcal{E} \subseteq \mathbf{TA}(S, V) \\ \mathcal{E}' \subseteq \mathcal{E} \\ \varphi \in \mathbf{TA}(S, V) \end{array}$$

Tabelle 7.1: Allgemeine Beweisregeln

Die beiden Regeln (ASS) und (S-ASS) sind ebenfalls klassische Regeln der mathematische Logik. Die Regel (ASS) drückt aus, daß jede (von der Umgebung) angenommene Aussage aus $\mathcal{E}$ gilt; sie benötigt keine Voraussetzung. Die Regel (S-ASS) erlaubt weitere Annahmen hinzuzufügen.

## 7.4 Regeln für Invarianten

In diesem Abschnitt werden Regeln zum Nachweis von Invarianten eingeführt. Tabelle 7.2 stellt die Regeln zum Beweisen von Invarianten zusammen. Wir haben bereits gesehen, daß sich Invarianten mit Hilfe von S-Invarianten oder **alw**-Eigenschaften und Informationen über den Anfangszustand beweisen lassen. Mit der Regel (INIT) können Aussagen über die Anfangsmarkierung hergeleitet werden. Dabei ist die Nebenbedingung dieser Regel, die in $\varphi$ keine Symbole der Umgebung zuläßt, wesentlich. Aussagen über die Anfangsmarkierung der Umgebung sind nur mit Hilfe von Umgebungsannahmen beweisbar.
Bei allen Regeln, die wir hier und in den nächsten Abschnitten vorstellen sind $\varphi, \psi, \chi, \ldots$ Zustandsaussagen aus $\mathbf{ZA}(S, V)$, wenn in den Nebenbedingungen der Regel nichts anderes gesagt ist. Mit dieser Vereinbarung können wir bei vielen Regeln auf die Angabe einer Nebenbedingung verzichten.
Die Regel (TAUT-INV) besagt, daß jede allgemeingültige Zustandsaussage $\varphi$ eine (triviale) Invariante einer Systemkomponente ist. Diese Regel ist ein Spezialfall von Satz 6.24.
Die Regeln ($\geq$ 1-INV) und ($\leq$ 1-INV) ermöglichen den Beweis von Invarianten mit Hilfe von S-Invarianten. Die Regel ($\geq$ 1-INV) besagt, daß immer mindestens eine Stelle der S-Invariante $Z$ markiert ist, wenn dies am Anfang der Fall ist. In der Schlußfolgerung der Regel haben wir die Menge $Z$ in zwei Mengen $Z_1$ und $Z_2$ aufgeteilt und die Aussage $\Box[Z_1 \vee Z_2]$ äquivalent zu $\Box[\neg Z_1 \Rightarrow Z_2]$ umgeformt, weil wir die Beweisregel meist in dieser Form anwenden. Den folgenden Spezialfall der Regel ($\geq$ 1-INV) bezeichnen mit demselben Namen, da sich aus der Struktur der Schlußfolgerung ergibt, welche der beiden Regeln gemeint ist:

$$(\geq \text{1-INV}) \quad \frac{\Sigma, \mathcal{E} \models \mathbf{inv}(Z) \quad \Sigma, \mathcal{E} \models Z}{\Sigma, \mathcal{E} \models \Box Z}$$

Die Regel ($\leq$ 1-INV) macht eine Aussage für eine S-Invariante $Z$, von deren Stellen am Anfang höchstens eine markiert ist. Wenn in einem Zustand eine Stelle der Menge $Z_1 \subseteq S$ markiert ist, dann sind die restlichen Stellen $Z_2$ der Invariante $Z$ unmarkiert. Wir werden die beiden Regeln ($\geq$ 1-INV) und ($\leq$ 1-INV) insbesondere für den Fall $\Sigma, \mathcal{E} \models \#Z = 1$ anwenden (zusammen mit der impliziten Abschwächung zu $\#Z \geq 1$ oder $\#Z \leq 1$).
Die Regel (ALW-INV) leitet aus einer stabilen Eigenschaft ($\varphi$ **alw** $\varphi$) die entsprechende Invariante her. Für den Beweis von **alw**-Aussagen

$$\text{(INIT)}\ \frac{M_\Sigma \models \varphi}{\Sigma, \mathcal{E} \models \varphi} \quad \varphi \in \mathbf{ZA}(Z_\Sigma \cup I_\Sigma \cup O_\Sigma, V)$$

$$\text{(TAUT-INV)}\ \frac{\models \varphi}{\Sigma, \mathcal{E} \models \Box\varphi}$$

$$\text{(}\geq\text{ 1-INV)}\ \frac{\Sigma, \mathcal{E} \models \mathbf{inv}(Z) \quad \Sigma, \mathcal{E} \models Z}{\Sigma, \mathcal{E} \models \Box[\neg Z_1 \Rightarrow Z_2]} \quad Z_1 \cup Z_2 \supseteq Z$$

$$\text{(}\leq\text{ 1-INV)}\ \frac{\Sigma, \mathcal{E} \models \mathbf{inv}(Z) \quad \Sigma, \mathcal{E} \models \#Z \leq 1}{\Sigma, \mathcal{E} \models \Box[Z_1 \Rightarrow \neg Z_2]} \quad Z_1, Z_2 \subseteq Z,\ Z_1 \cap Z_1 = \emptyset$$

$$\text{(ALW-INV)}\ \frac{\Sigma, \mathcal{E} \models \varphi\ \mathbf{alw}\ \varphi \quad \Sigma, \mathcal{E} \models \varphi}{\Sigma, \mathcal{E} \models \Box\varphi}$$

$$\text{(W-INV)}\ \frac{\Sigma, \mathcal{E} \models \Box[\varphi \Rightarrow \psi] \quad \Sigma, \mathcal{E} \models \Box\varphi}{\Sigma, \mathcal{E} \models \Box\psi}$$

Tabelle 7.2: Regeln für Invarianten

und S-Invarianten werden wir in den nachfolgenden Abschnitten Regeln angeben.
Zunächst betrachten wir noch die Regel (W-INV) etwas genauer. Diese Regel entspricht der Regel (MP) für Invarianten. Zusammen mit (TAUT-INV) kann die Regel (W-INV) zum „Abschwächen“ von Invarianten benutzt werden: Wenn $\varphi \Rightarrow \psi$ allgemeingültig ist, dann ist mit (TAUT-INV) die Aussage $\Sigma, \mathcal{E}, \models \Box[\varphi \Rightarrow \psi]$ herleitbar. Zusammen mit der Aussage $\Sigma, \mathcal{E}, \models \Box\varphi$ ist mit der Regel (W-INV) die schwächere Invariante $\Sigma, \mathcal{E}, \models \Box\psi$ herleitbar. Auf ähnliche Weise können wir unter Anwendung der Regel (TAUT-INV)[3] und zweimaliger Anwendung der Regel (W-INV) die folgende Regel herleiten:

$$\text{(W-INV)}\quad \frac{\Sigma, \mathcal{E} \models \Box[\varphi \Rightarrow \psi] \quad \Sigma, \mathcal{E} \models \Box[\psi \Rightarrow \chi]}{\Sigma, \mathcal{E} \models \Box[\varphi \Rightarrow \chi]}$$

Diese Regel nennen wir ebenfalls (W-INV), weil sich aus der Struktur der Schlußfolgerung ergibt, welche der beiden gleichnamigen Regeln gemeint ist.

## 7.5 Regeln für S-Invarianten

Die Regeln zum Beweis von S-Invarianten, sind hauptsächlich Regeln zum Beweis von partiellen S-Invarianten. Eine partielle S-Invariante kann in eine S-Invariante umgewandelt werden, wenn ihr Anfang und Ende leer sind (Regel (S-INV)). Die nötigen Regeln für partielle S-Invarianten sind in Tabelle 7.3 zusammengestellt.
Die Regel (P-INV) dient dem Beweis einer partiellen S-Invariante. Dazu wird im wesentlichen für jede Transition der Systemkomponente die strukturelle Bedingung $|{}^\bullet t \cap (I \cup Z)| = |t^\bullet \cap (Z \cup O)|$ vorausgesetzt. Allerdings muß diese Bedingung nur dann überprüft werden, wenn die Transition $t$ in einem Ablauf der Systemkomponente vorkommt. Wenn durch die Invariante $\Box\neg\bigwedge {}^\bullet t$ gezeigt ist, daß $t$ nie aktiviert ist, muß für diese Transition keine weitere Bedingung erfüllt sein. In der Regel (P-INV) wird diese Unterscheidung durch die Aufteilung der Menge $T_\Sigma$ in die Mengen $T_1$ und $T_2$ getroffen. Weil der Anfang $I$ und das Ende $O$ der partiellen S-Invariante wegen der Nebenbedingungen der Regeln Ein- bzw. Ausgabestellen der Systemkomponente und $Z$ interne Stellen sind, erfüllt jede potentielle Transition $t$ der Umgebung der Systemkomponente die Bedingung $|{}^\bullet t \cap (I \cup Z)| = 0 = |t^\bullet \cap (Z \cup O)|$.

[3] auf die allgemeingültige Aussage $(\varphi \Rightarrow \psi) \Rightarrow ((\psi \Rightarrow \chi) \Rightarrow (\varphi \Rightarrow \chi))$

$$\text{(P-INV)}\quad \frac{\begin{array}{l}\text{Für jede Transition } t \in T_1 \text{ gilt:}\\ \quad |{}^\bullet t \cap (I \cup Z)| = |t^\bullet \cap (Z \cup O)|\\ \\ \text{Für jede Transition } t \in T_2 \text{ gilt:}\\ \quad \Sigma, \mathcal{E} \models \Box\neg \bigwedge {}^\bullet t\end{array}}{\Sigma, \mathcal{E} \models \textbf{p-i}(I, Z, O)} \quad \begin{array}{l} T_1 \cup T_2 = T_\Sigma\\ Z \subseteq Z_\Sigma\\ I \subseteq I_\Sigma\\ O \subseteq O_\Sigma\end{array}$$

$$\text{(}\cup\text{-P-INV)}\quad \frac{\begin{array}{l}\Sigma, \mathcal{E} \models \textbf{p-i}(I_1, Z_1, O_1)\\ \Sigma, \mathcal{E} \models \textbf{p-i}(I_2, Z_2, O_2)\end{array}}{\Sigma, \mathcal{E} \models \textbf{p-i}(I, Z, O)} \quad \begin{array}{l}(Z_1 \cup O_1) \cap (Z_2 \cup O_2) = \emptyset\\ (Z_1 \cup I_1) \cap (Z_2 \cup I_2) = \emptyset\\ \\ I \mathrel{\widehat{=}} I_1 \setminus O_2 \ \cup\ I_2 \setminus O_1\\ Z \mathrel{\widehat{=}} Z_1 \cup Z_1 \cup I_1 \cap O_2 \cup I_2 \cap O_1\\ O \mathrel{\widehat{=}} O_1 \setminus I_2 \ \cup\ O_2 \setminus I_1\end{array}$$

$$\text{(S-INV)}\quad \frac{\Sigma, \mathcal{E} \models \textbf{p-i}(\emptyset, Z, \emptyset)}{\Sigma, \mathcal{E} \models \textbf{inv}(Z)}$$

Tabelle 7.3: Regeln für S-Invarianten und partielle S-Invarianten

Deshalb kommen in den Voraussetzungen der Regel nur Anforderungen für Transitionen der Systemkomponente vor.
Die Regel ($\cup$-P-INV) kombiniert zwei partielle S-Invarianten, wie wir dies in Satz 6.14 gezeigt haben.

## 7.6 Regeln für alw-Eigenschaften

Die Regeln zum Beweis von **alw**-Eigenschaften sind in Tabelle 7.4 zusammengestellt. Ähnlich wie bei partiellen S-Invarianten gibt es eine Regel, die **alw**-Eigenschaften direkt aus der Systemstruktur gewinnt: Die Regel (ALW) leitet die **alw**-Eigenschaft unmittelbar aus entsprechenden Zusicherungen für die Transitionen der Systemkomponente und der Umgebung her.

$$\text{(ALW)}\ \frac{\langle\varphi\rangle\ T_\Sigma\ \langle\varphi \vee \psi\rangle \quad \langle\varphi\rangle\ T_\Sigma^U\ \langle\varphi \vee \psi\rangle}{\Sigma, \mathcal{E} \models \varphi\ \mathbf{alw}\ \psi}$$

$$\text{(W-ALW)}\ \frac{\Sigma, \mathcal{E} \models (\varphi \wedge \chi)\ \mathbf{alw}\ (\neg\chi \vee \psi) \quad \Sigma, \mathcal{E} \models \Box\chi}{\Sigma, \mathcal{E} \models \varphi\ \mathbf{alw}\ \psi}$$

$$\text{($\vee$-ALW)}\ \frac{\Sigma, \mathcal{E} \models \varphi\ \mathbf{alw}\ \psi \quad \Sigma, \mathcal{E} \models \varphi'\ \mathbf{alw}\ \psi'}{\Sigma, \mathcal{E} \models (\varphi \vee \varphi')\ \mathbf{alw}\ (\psi \vee \psi')}$$

$$\text{($\wedge$-ALW)}\ \frac{\Sigma, \mathcal{E} \models \varphi\ \mathbf{alw}\ \psi \quad \Sigma, \mathcal{E} \models \varphi'\ \mathbf{alw}\ \psi'}{\Sigma, \mathcal{E} \models (\varphi \wedge \varphi')\ \mathbf{alw}\ (\varphi \wedge \psi' \ \vee\ \psi \wedge \varphi' \ \vee\ \psi \wedge \psi')}$$

Tabelle 7.4: Regeln für **alw**-Eigenschaften

Mit Hilfe von Invarianten können **alw**-Eigenschaften vereinfacht bzw. abgeschwächt werden. Dafür geben wir die Regel (W-INV) an. Mit dieser Regel kann eine Invariante $\chi$ aus der **alw**-Eigenschaft $\varphi \wedge \chi$ **alw** $\neg\chi \vee \psi$ eliminiert werden. Mit den Regeln ($\vee$-ALW) und ($\wedge$-ALW) können **alw**-Eigenschaften durch Disjunktion oder Konjunktion miteinander verknüpft werden. Die Korrektheit dieser Regeln folgt unmittelbar aus der Definition des **alw**-Operators.

## 7.7 Regeln für Leadsto-Eigenschaften

Wir werden in diesem Abschnitt Regeln zum Beweisen von Leadsto-Eigenschaften einführen. Zunächst geben wir zwei Regeln an, die aus den strukturellen Eigenschaften einer Systemkomponente eine Leadsto-Eigenschaft herleiten. Die erste Regel (PROGR) benutzt eine Transition der Progressmenge der Systemkomponente. Die zweite Regel (FAIR) benutzt eine Transition der Fairnessmenge der Systemkomponente.

Im Anschluß an diese beiden Regeln, führen wir Regeln zur Kombination von Leadsto-Eigenschaften ein. Hierbei spielt der *Leadsto-Graph*[4] eine zentrale Rolle. Die Regeln zur Kombination von Leadsto-Eigenschaften entsprechen den üblichen Regeln für den sequentiellen Leadsto-Operator (z.B. [67, 18, 54]). Dagegen sind die oben genannten Regeln zum „Ablesen“ von Leadsto-Eigenschaften aus der Systemstruktur speziell auf unseren Leadsto-Operator und auf unsere Begriffe von Progress und Fairness zugeschnitten. Deshalb widmen wir der Herleitung der beiden Regeln (PROGR) und (FAIR) besondere Aufmerksamkeit, auch wenn wir die Regel (PROGR) in dieser Arbeit nicht in der vollen Allgemeinheit benötigen.

Am Ende dieses Abschnitts fassen wir alle Regeln für Leadsto nochmals in Tabelle 7.6 zusammen.

### 7.7.1 Die Regel (PROGR)

Wir entwickeln nun schrittweise die Regel (PROGR), die mit einer Transition $t_0$ der Progressmenge einer Systemkomponente eine Leadsto-Eigenschaft herleitet. Dazu betrachten wir zunächst ein einfaches Beispiel. In Abb. 7.2 ist ein Ausschnitt einer Systemkomponente $\Sigma$ dargestellt. Wir gehen davon aus, daß in $\Sigma$ nur die Transitionen $t_0$ und $t_1$ im Nachbereich von $a$, $b$ und $c$ vorkommen. Wenn in einer Scheibe eines Ablaufes dieser Systemkomponente die Beschriftungen $a$ und $b$ vorkommen, dann wissen wir aufgrund der Progressannahme für $t_0$, daß entweder $t_0$ oder $t_1$ irgendwann eintritt; denn sonst wäre mit $a$ und $b$ die Transition $t_0$ am Ende des Ablaufs aktiviert. Wenn $t_0$ oder $t_1$ eintritt, gilt anschließend $e$ oder $f$. Deshalb gilt für die Systemkomponente $\Sigma \models a \wedge b \rightsquigarrow e \vee f$.

Für diesen Fall können wir eine einfache Regel formulieren, die die

[4] In [67] werden Leadsto-Graphen *proof lattice* genannt.

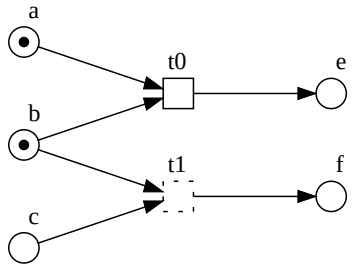



Abbildung 7.2: Systemkomponente $\Sigma$ mit $\Sigma \models a \wedge b \leadsto e \vee f$

obige Leadsto-Eigenschaft beweist:

$$\text{(PROGR1)}\ \frac{\langle \bigwedge {}^\bullet t_0 \rangle\ ({}^\bullet t_0)^\bullet\ \langle \psi \rangle}{\Sigma, \mathcal{E} \models \bigwedge {}^\bullet t_0 \leadsto \psi} \Big|\ t_0 \in T_\Sigma^P$$

Dabei charakterisiert $({}^\bullet t_0)^\bullet$ genau die Transitionen, die mit $t_0$ *in Konflikt* stehen. Diese Regel ist — zusammen mit den später eingeführten Regeln zur Kombination von Leadsto-Eigenschaften — für viele Beispiele ausreichend, um die gewünschten Eigenschaften zu beweisen. Es gibt jedoch Beispiele, in denen die Regel (PROGR1) nicht ausreicht.
Wir werden nun Beispiele betrachten, die die verschiedenen Verallgemeinerungen und Ergänzungen der Regel (PROGR1) motivieren. Am Ende werden wir dann die Regel (PROGR) angeben und ihre Korrektheit zeigen. Zunächst zeigen wir, daß die Regel (PROGR1) nicht für beliebige Zustandsaussagen $\varphi$ gilt: Wenn für eine Transition $t$ der Progressmenge einer Systemkomponente $\Sigma$ gilt $\langle \varphi \rangle\ ({}^\bullet t)^\bullet\ \langle \psi \rangle$, gilt im allgemeinen nicht $\Sigma \models \varphi \leadsto \psi$. Ein Grund dafür ist, daß $t$ in einer $\varphi$-Scheibe möglicherweise nicht aktiviert ist. Beispielsweise gilt in der Systemkomponente $\Sigma$ aus Abb 7.2 $({}^\bullet t_0)^\bullet = \{t_0, t_1\}$ und die Zusicherung $\langle a \rangle\ \{t_0, t_1\}\ \langle e \vee f \rangle$. In einer Scheibe, in der nur $a$ markiert ist, kann aber weder $t_0$ noch $t_1$ schalten. Deshalb gilt nicht $\Sigma \models a \leadsto e \vee f$. Die Aktiviertheit der Transition $t_0$ können wir durch eine zusätzlichen Voraussetzung $\Sigma \models \Box[\varphi \Rightarrow \bigwedge {}^\bullet t_0]$ gewährleisten. Diese zusätzliche Voraussetzung ist aber immer noch nicht ausreichend. Den Grund dafür sehen wir, wenn wir einen etwas größeren Ausschnitt von $\Sigma$ betrachten, wie er in Abb 7.3 dargestellt ist. Offensichtlich gilt für $\Sigma$ die Zusicherung $\langle a \wedge b \wedge g \rangle\ \{t_0, t_1\}\ \langle g \wedge e\ \vee\ g \wedge f \rangle$ und auch die Voraussetzung $\Sigma \models \Box[a \wedge b \wedge g \Rightarrow \bigwedge {}^\bullet t_0]$ ist erfüllt. Trotzdem gilt nicht

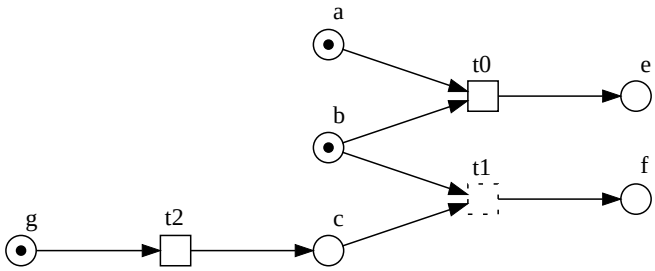



Abbildung 7.3: Erweiterter Ausschnitt der Systemkomponente $\Sigma$

$\Sigma \models a \wedge b \wedge g \rightsquigarrow g \wedge e \vee g \wedge f$. In Abb. 7.4 ist ein Ausschnitt aus einem Ablauf angegeben der dies verdeutlicht: Das Problem ist, daß nicht $t_0$

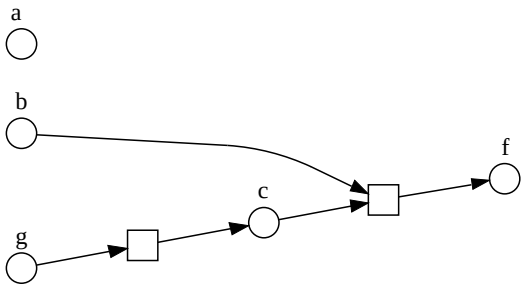



Abbildung 7.4: Ablaufs, für den $a \wedge b \wedge g \rightsquigarrow g \wedge e \vee g \wedge f$ nicht gilt

sondern $t_1$ in dem Ablauf eintritt und $t_1$ erst aktiviert wird, wenn $g$ bereits ungültig ist. Damit gilt unmittelbar vor dem Eintreten von $t_1$ nicht $a \wedge b \wedge g$ und die Zusicherung $\langle a \wedge b \wedge g \rangle \ \{t_0, t_1\} \ \langle g \wedge e \ \vee \ g \wedge f \rangle$ macht deshalb keine Aussage über den Zustand nach dem Eintreten von $t_1$.
Es gibt aber Systeme, die dem obigen System sehr ähnlich sind, für die die Aussage $\Sigma \models a \wedge b \wedge g \rightsquigarrow g \wedge e \vee g \wedge f$ gilt. Ein Ausschnitt einer solchen Systemkomponente ist in Abb. 7.5 dargestellt. In diesem System wird $c$ erst gültig, nachdem $t_0$ oder $t_1$ eingetreten ist, und damit der gewünschte Zustand $g \wedge e \ \vee \ g \wedge f$.
Um den Unterschied zwischen diesen beiden Systemen auszudrücken, formalisieren wir für eine Menge von Stellen $Z$ durch eine temporale

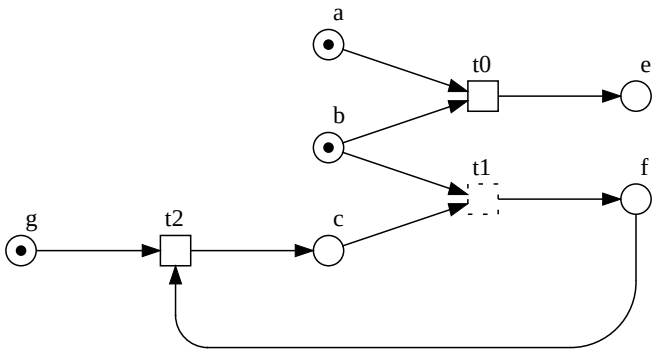



Abbildung 7.5: Systems, in dem $a \wedge b \wedge g \rightsquigarrow g \wedge e \vee g \wedge f$ gilt

Aussage, daß für jede Stelle $s \in Z$ die Anzahl der Marken kleiner wird, bis eine Transition aus der Menge $T_1 \mathrel{\widehat{=}} ({}^\bullet t_0)^\bullet$ eintritt. Dazu definieren wir die Zustandsaussagen $\theta^{=}_{Z \cup {}^\bullet t_0} \mathrel{\widehat{=}} \bigwedge_{s \in Z} \#\{s\} = v_s$ und $\theta^{\leq}_{Z} \mathrel{\widehat{=}} \bigwedge_{s \in Z} \#\{s\} \leq v_s$ und $\theta_{t_0} \mathrel{\widehat{=}} \bigvee_{s \in {}^\bullet t_0} \#\{s\} < v_s$, wobei $v_s$ für jedes $s \in Z$ eine neue Variable ist. Die temporale Aussage $\theta^{=}_{Z \cup {}^\bullet t_0}$ **alw** $\theta^{\leq}_{Z} \vee \theta_{t_0} \vee \psi$ garantiert, daß die Anzahl der Marken auf jeder einzelnen Stelle aus $Z$ abnimmt, bis entweder eine Transition aus $T_1$ geschaltet hat oder $\psi$ gilt. In der Regel (PROGR) wird diese Forderung mit Hilfe einer geeigneten Zustandsaussage $\chi$ noch etwas differenzierter formuliert:

$$(\varphi \vee \chi) \wedge \theta^{=}_{Z \cup {}^\bullet t_0} \textbf{ alw } (\varphi \ \vee \ \theta^{\leq}_{Z} \wedge \chi \ \vee \ \theta_{t_0} \ \vee \ \psi)$$

Allerdings können **alw**-Eigenschaften keine Schlingen auf Stellen erkennen. Deshalb reicht die obige **alw**-Eigenschaft für Systemkomponente, die Transitionen mit Schlingen enthalten, nicht aus. Für das System aus Abb. 7.6 gilt nicht $\Sigma \models a \wedge b \wedge g \rightsquigarrow g \wedge e \vee g \wedge f$, obwohl sogar die stärkere Aussage $\theta^{=}_{\{a,b,c\}}$ **alw** $\theta^{\leq}_{\{a,b,c\}}$ gilt. Der Ablauf aus Abb. zeigt 7.7, daß dies an der Schlinge von $t_2$ zur Stelle $c$ liegt. Deshalb müssen wir in der Regel (PROGR) für Transitionen $t$, die Schlingen auf eine Stelle im Vorbereich von $T_1$ haben und nicht in $T_1$ liegen, die Zusicherung $\langle \varphi \vee \chi \rangle \ t \ \langle \varphi \vee \psi \rangle$ voraussetzen. Die Menge dieser Transitionen wird in der Regel (PROGR) mit $T_2$ bezeichnet.
Manchmal kommt es vor, daß für bestimmte Transitionen aus $t \in ({}^\bullet t_0)^\bullet$ nicht nur die Zusicherung $\langle \varphi \rangle \ t \ \langle \psi \rangle$ gilt, sondern auch die stärke-

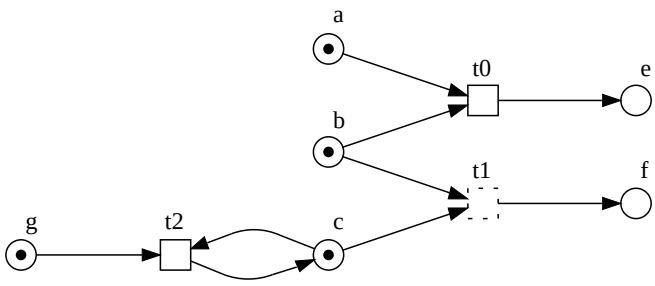



Abbildung 7.6: Systems, in dem $a \wedge b \wedge g \rightsquigarrow g \wedge e \vee g \wedge f$ nicht gilt

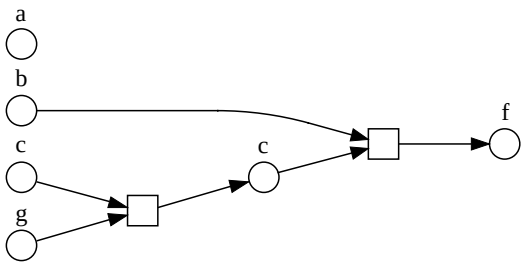



Abbildung 7.7: Ablauf, der $a \wedge b \wedge g \rightsquigarrow g \wedge e \vee g \wedge f$ nicht erfüllt

re Zusicherung $\langle \varphi \vee \chi \rangle \ t \ \langle \psi \rangle$ gilt. Dabei ist $\chi$ die Zustandsaussage, die auch in obiger **alw**-Eigenschaft vorkommt. Für Stellen, die nur im Vorbereich solcher Transitionen liegen, müssen wir die oben beschriebene **alw**-Eigenschaft nicht voraussetzen; ebenso müssen Transitionen mit Schlingen zu diesen Stellen keine zusätzlichen Anforderungen erfüllen. In der Regel (PROGR) teilen wir deshalb die Menge $({}^\bullet t_0)^\bullet$ in zwei Mengen $T_1$ und $T_1'$ auf.
Insgesamt erhalten wir dann die folgende Regel:

(PROGR)

(1) $\Sigma, \mathcal{E} \models \Box[\varphi \Rightarrow \bigwedge {}^\bullet t_0]$
(2) $\langle \varphi \rangle \ T_1 \ \langle \psi \rangle$
(3) $\langle \varphi \vee \chi \rangle \ T_1' \ \langle \psi \rangle$
(4) $\langle \varphi \vee \chi \rangle \ T_2 \ \langle \varphi \vee \psi \rangle$
(5) $\Sigma, \mathcal{E} \models (\varphi \vee \chi) \wedge \theta^{=}_{Z \cup {}^\bullet t_0}$ **alw** $(\varphi \ \vee \ \theta^{\leq}_{Z} \wedge \chi \ \vee \ \theta_{t_0} \ \vee \ \psi)$

---

$\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \psi$

$t_0 \in T_\Sigma^P$
$T_1 \cup T_1' = ({}^\bullet t_0)^\bullet$
$Z = {}^\bullet T_1$
$T_2 = \{t \mid Z \cap {}^\bullet t \cap t^\bullet\} \setminus T_1$
$\theta^{=}_{Z \cup {}^\bullet t_0} \mathrel{\widehat{=}} \bigwedge_{s \in Z} \#\{s\} = v_s$
$\theta^{\leq}_{Z} \mathrel{\widehat{=}} \bigwedge_{s \in Z} \#\{s\} \leq v_s$
$\theta_{t_0} \mathrel{\widehat{=}} \bigvee_{s \in {}^\bullet t_0} \#\{s\} < v_s$
wobei $v_s$ paarweise verschiedene neue Variablen sind.

Damit wir uns im Beweis der Korrektheit der Regel auf die einzelnen Voraussetzungen beziehen können, haben wir die Voraussetzung mit (1)–(5) numeriert. Der Beweis verdeutlicht nochmals genauer, warum mit den Voraussetzungen der Regel die Gültigkeit der Schlußfolgerung gilt.

**Beweis:** Sei $\Sigma$ eine Systemkomponente, $\varphi$, $\psi$ und $\chi$ Zustandsaussagen, $t_0$ eine Transition von $\Sigma$ und $T_1$ und $T_1'$ Mengen von Transitionen, so daß die Voraussetzungen (1)–(5) und Nebenbedingungen der Regel (PROGR) erfüllt sind.

Sei nun $\rho$ ein Ablauf von $\Sigma$, $\beta$ eine Belegung und $C_0$ eine erreichbare Scheibe von $\rho$ mit $\rho, C_0, \beta \models \varphi$. Es ist zu zeigen, daß eine von $C_0$ aus erreichbare Scheibe $C'$ mit $\rho, C', \beta \models \psi$ existiert.

Wegen Voraussetzung (1) der Regel gilt $\rho(C_0) \supseteq {}^\bullet t_0$. Da $t_0$ eine Transition aus der Progressmenge $T_\Sigma^P$ ist, existiert eine von $C_0$ aus erreichbare Scheibe $C_n$ und ein Ereignis $e$ mit ${}^\bullet e \subseteq C_n$, ${}^\bullet e \cap C_0 \neq \emptyset$ und $t \mathrel{\widehat{=}} \rho(e) \in T_1 \cup T_1'$. Wir definieren $\widehat{C} \mathrel{\widehat{=}} (C_n \setminus {}^\bullet e) \cup e^\bullet$. Diese Situation ist in Abb. 7.8 dargestellt.

In diesem Ablauf existiert eine Folge von unmittelbar aufeinanderfolgenden Scheiben $C_0 \longrightarrow C_1 \longrightarrow \ldots \longrightarrow C_n$. Falls $\rho, C_n, \beta \models \varphi$

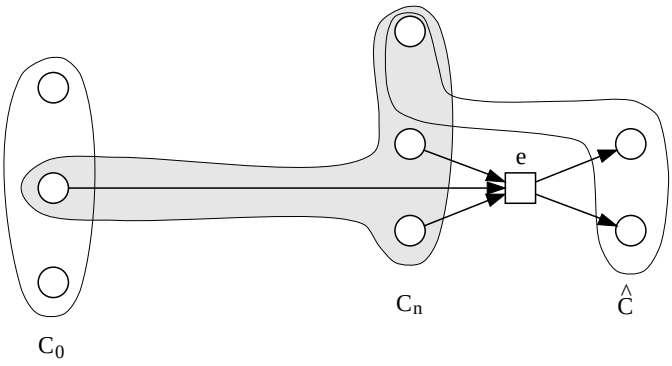



Abbildung 7.8: Graphische Darstellung der Beweis-Situation

gilt, dann gilt wegen $t \in T_1 \cup T_1'$ und Voraussetzung (2) bzw. (3) der Regel $\rho, \widehat{C}, \beta \models \psi$. Mit $C' = \widehat{C}$ ist die Aussage gezeigt.

Nehmen wir also an, daß $\rho, C_n, \beta \models \varphi$ nicht gilt. Dann existiert ein größtes $k$ mit $0 \leq k < n : \rho, C_k, \beta \models \varphi$. Wir zeigen nun für jedes $i$ mit $k \leq i \leq n$ durch Induktion, daß

1. eine von $C_0$ aus erreichbare Scheibe $C'$ mit $\rho, C', \beta \models \psi$ existiert oder
2. $\rho, C_i, \beta \models \chi \vee \varphi$ und $C_i \cap \rho^{-1}(Z) \subseteq C_k \cap \rho^{-1}(Z)$ gilt, d.h. alle mit $Z$ beschrifteten Bedingungen in Scheibe $C_i$ sind bereits in Scheibe $C_k$ enthalten.

**Induktionsanfang:** Für $i = k$ ist die Aussage 2 offensichtlich erfüllt.

**Induktionsschritt** $i \to i+1$**:** Es gilt $i+1 > k$. Da nach Voraussetzung $k$ maximal gewählt wurde, gilt nicht $\rho, C_{i+1}, \beta \models \varphi$. Nach Induktionsvoraussetzung gilt für $i$ Aussage 1 oder 2. Wenn Aussage 1 für $i$ gilt, gilt sie auch für $i+1$ (da sie unabhängig von $i$ ist).

Nehmen wir also an, daß für $i$ die Aussage 2 gilt. Wir definieren $e' \mathrel{\widehat{=}} (C_i \setminus C_{i+1}, C_{i+1} \setminus C_i)$ und $t' \mathrel{\widehat{=}} \rho(e')$. Wir unterscheiden nun die Fälle $t' \in T_1$, $t' \in T_1'$, $t' \in T_2$ und $t' \notin T_1 \cup T_1' \cup T_2$.

1. $t' \in T_1$: Gemäß Definition von $Z$ gilt ${}^\bullet t' \subseteq Z$. Also gilt $\rho({}^\bullet e') \subseteq Z$. Mit der Induktionsvoraussetzung folgt ${}^\bullet e' \subseteq$

$C_k$. Also ist $C' = (C_k \setminus {}^\bullet e') \cup e'^\bullet$ eine von $C_0$ erreichbare Scheibe, für die wegen Voraussetzung (2) der Regel und $\rho, C_k, \beta \models \varphi$ gilt: $\rho, C', \beta \models \psi$. Damit ist die Aussage 1 gezeigt.

2. $t' \in T_1'$: Wegen der Induktionsvoraussetzung gilt $\rho, C_i, \beta \models \chi \vee \varphi$. Mit der Voraussetzung (3) der Regel folgt $\rho, C_{i+1}, \beta \models \psi$. Mit $C' = C_{i+1}$ gilt die Aussage 1.
3. $t' \in T_2$: Wegen Voraussetzung (4) der Regel und der Induktionsvoraussetzung $\rho, C_i, \beta \models \chi \vee \varphi$ gilt $\rho, C_{i+1}, \beta \models \varphi \vee \psi$. Da nicht gilt $\rho, C_{i+1}, \beta \models \varphi$, gilt $\rho, C_{i+1}, \beta \models \psi$. Mit $C' = C_{i+1}$ gilt die Aussage 1.
4. $t' \notin T_1 \cup T_1' \cup T_2$: Nach Induktionsvoraussetzung gilt $\rho, C_i, \beta \models \varphi \vee \chi$. Sei $\beta'$ eine Erweiterung der Belegung von $\beta$ für die neuen Variablen $v_s$ mit $\beta'(v_s) = |\{b \in C_i \mid \rho(b) = s\}|$. Da $v_s$ neue und paarweise verschiedene Variablen sind gilt auch $\rho, C_i, \beta' \models (\varphi \vee \chi) \wedge \theta_Z^=$. Wegen Voraussetzung (5) der Regel gilt $\rho, C_{i+1}, \beta' \models \varphi \vee \theta_Z^\leqslant \wedge \chi \vee \theta_{t_0} \vee \psi$. Da $t' \notin T_1 \cup T_1$, gilt nicht $\rho, C_{i+1}, \beta' \models \theta_{t_0}$. Auch $\rho, C_{i+1}, \beta' \models \varphi$ gilt nicht. Also gilt $\rho, C_{i+1}, \beta' \models \theta_Z^\leqslant \wedge \chi$ oder $\rho, C_{i+1}, \beta' \models \psi$. Im letzten Fall haben wir mit $C' = C_{i+1}$ Aussage 1 gezeigt. Für den Fall $\rho, C_{i+1}, \beta' \models \theta_Z^\leqslant \wedge \chi$, zeigen wir durch Widerspruch, daß für kein $b \in e'^\bullet$ gilt $\rho(b) \in Z$: Angenommen es gäbe ein $b \in e'^\bullet$ mit $\rho(b) \in Z$ dann gibt es wegen $\rho, C_{i+1}, \beta' \models \theta_Z^\leqslant$ ein $b' \in {}^\bullet e'$ mit $\rho(b') = \rho(b)$. Also gilt $t' \in T_1 \cup T_1' \cup T_2$, was im betrachteten Fall ausgeschlossen wurde.
   Damit gilt $\rho(e'^\bullet) \cap Z = \emptyset$. Dies bedeutet $C_{i+1} \cap \rho^{-1}(Z) \subseteq C_i \cap \rho^{-1}(Z)$ und zusammen mit der Induktionsvoraussetzung folgt $C_{i+1} \cap \rho^{-1}(Z) \subseteq C_k \cap \rho^{-1}(Z)$. Zusammen mit $\rho, C_{i+1}, \beta \models \chi$ gilt für $i+1$ die Aussage 2.

Wenn für $i = n$ die Aussage 1 gilt, so ist die Gesamtaussage gezeigt.

Betrachten wir den Fall, daß für $i = n$ die Aussage 2 gilt: Für $t \in T_1'$ folgt mit Voraussetzung (3) der Regel unmittelbar $\rho, \widehat{C}, \beta \models \psi$. Wir wählen also $C' = \widehat{C}$. Für $t \in T_1$ gilt wegen Aussage 2: ${}^\bullet e \subseteq C_k$. Die Scheibe $C' = (C_k \setminus {}^\bullet e) \cup e^\bullet$ ist dann von $C_0$ aus erreichbar. Wegen Voraussetzung (2) der Regel gilt $\rho, C', \beta \models \psi$. □

Die Regel (PROGR) besitzt viele sehr einfache Spezialfälle. Ein Spezialfall ist die Regel (PROGR1), die wir bereits kennengelernt haben.

Neben diesem Spezialfall gibt es noch einige interessante Spezialfälle für 1-sichere Systeme bzw. für 1-sichere Stellenbereiche von Systemkomponenten $\Sigma$ (d.h. für Stellenmengen $Z$, für die für jedes $s \in Z$ gilt $\Sigma \models \Box\#\{s\} \leq 1$). Da wir diese Regeln in dieser Arbeit nicht benötigen geben wir hier keine weiteren Spezialfälle an.

### 7.7.2 Die Regel (FAIR)

Wir führen nun die Regel (FAIR) ein, die mit einer Transition aus der Fairnessmenge einer Systemkomponente eine Lebendigkeitseigenschaft beweist. Zum besseren Verständnis dieser Regel betrachten wir zunächst das System aus Abb. 7.9. Wegen der Fairnessannahme für $t_0$ gilt in diesem System $\Sigma \models a \rightsquigarrow c$.

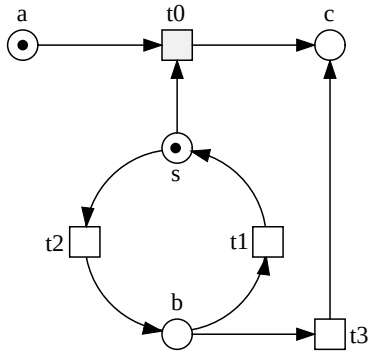



Abbildung 7.9: System $\Sigma$ mit fairer Transition $t_0$

Im einzelnen sind die Argumente für die Gültigkeit von $\Sigma \models a \rightsquigarrow c$ wie folgt: $a$ wird erst ungültig, wenn $c$ eintritt (d.h. $a$ **alw** $c$). Wenn $a$ gilt, wird entweder $s$ immer wieder gültig oder mit dem Eintreten von $t_0$ oder $t_3$ wird $c$ irgendwann gültig (d.h. $a \rightsquigarrow s \vee c$). Das heißt entweder wird $c$ ohnehin irgendwann gültig oder $a$ und $s$ sind immer wieder zusammen gültig. Im zweiten Fall ist $t_0$ mit $a$ und $s$ immer wieder aktiviert (d.h. $\Box[a \wedge s \Rightarrow \bigwedge {}^\bullet t_0]$). Deshalb tritt $t_0$ aufgrund der Fairnessannahme irgendwann ein. Nach dem Schalten von $t_0$ gilt dann wegen $\langle a \rangle \ t_0 \ \langle c \rangle$ die Zustandsaussage $c$.
Im Beispiel steht $t_0$ über die Stelle $a$ mit keiner Transition in Konflikt. In der Regel (FAIR) lassen wir zu, daß weitere Transitionen über ${}^\bullet t_0 \setminus \{s\}$ mit $t_0$ in Konflikt stehen. Für diese Transitionen fordern wir —

wie für $t_0$ — $\langle\varphi\rangle\ (^\bullet t_0 \setminus \{s\})^\bullet\ \langle\psi\rangle$. Dabei sind in der Regel (FAIR) die Forderungen für beliebige Zustandsaussagen $\varphi$ und $\psi$ formuliert.

$$\text{(FAIR)}\quad \frac{\begin{array}{l}\Sigma, \mathcal{E} \models \Box[(\varphi \wedge s) \Rightarrow \bigwedge {}^\bullet t_0] \\ \Sigma, \mathcal{E} \models \varphi \ \textbf{alw}\ \psi \\ \langle\varphi\rangle\ (^\bullet t_0 \setminus \{s\})^\bullet\ \langle\psi\rangle \\ \Sigma, \mathcal{E} \models \varphi \rightsquigarrow s \vee \psi\end{array}}{\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \psi} \quad \begin{array}{l} \\ \\ t_0 \in T_\Sigma^F \\ |{}^\bullet t_0| \geq 2 \\ s \in {}^\bullet t_0\end{array}$$

Für obiges Beispiel wählen wir $\varphi \mathrel{\widehat{=}} a$ und $\psi \mathrel{\widehat{=}} c$. Damit sind die Voraussetzungen der Regel (FAIR) offensichtlich erfüllt. Zum Beweis der Voraussetzung $\Sigma \models a \rightsquigarrow s \vee c$ benötigen wir neben der Regel (PROGR1) noch weitere Regeln für Leadsto-Eigenschaften. Die anderen (temporallogischen) Voraussetzungen sind mit den zuvor vorgestellten Regeln beweisbar.

### 7.7.3 Kombinationsregeln für Leadsto

Neben den Regeln zum „Ablesen“ von Leadsto-Eigenschaften für eine Systemkomponente, benötigen wir Regeln zum Kombinieren von Leadsto-Eigenschaften. Insbesondere können wir dazu Invarianten und **alw**-Eigenschaften ausnutzen. Die folgenden beiden Regeln nutzen Invarianten aus:

$$\text{(INV-}\rightsquigarrow\text{)}\quad \frac{\Sigma, \mathcal{E} \models \Box[\varphi \Rightarrow \psi]}{\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \psi} \qquad \text{(W-}\rightsquigarrow\text{)}\quad \frac{\begin{array}{l}\Sigma, \mathcal{E} \models \Box[\varphi \Rightarrow \varphi'] \\ \Sigma, \mathcal{E} \models \Box[\psi' \Rightarrow \psi] \\ \Sigma, \mathcal{E} \models \varphi' \rightsquigarrow \psi'\end{array}}{\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \psi}$$

Mit Regel (INV-$\rightsquigarrow$) können wir eine Invariante $\varphi \Rightarrow \psi$ in eine Leadsto-Eigenschaft $\varphi \rightsquigarrow \psi$ „umwandeln“. Da jede $\varphi$-Scheibe wegen der Invariante auch eine $\psi$-Scheibe ist, gilt die entsprechende Leadsto-Eigenschaft trivial. Die Regel (W-$\rightsquigarrow$) schwächt eine Leadsto-Eigenschaft mit Hilfe von Invarianten ab. Diese Regel können wir zusammen mit der später vorgestellten Regel für Leadsto-Graphen aus (INV-$\rightsquigarrow$) ableiten. Weil wir diese Regel in Beweisen häufig benutzen, geben wir sie hier als gesonderte Regel an.
Die Regel (SYNC) kombiniert eine Leadsto-Eigenschaft mit einer **alw**-Aussage:

$$\text{(SYNC)}\quad \frac{\begin{array}{l}\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \psi \\ \Sigma, \mathcal{E} \models \varphi \wedge \neg\psi\ \textbf{alw}\ \varphi\end{array}}{\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \varphi \wedge \psi}$$

Wenn $\varphi \rightsquigarrow \psi$ gilt und $\varphi$ solange gilt, bis $\psi$ gültig wird, dann gilt damit auch $\varphi \rightsquigarrow \varphi \wedge \psi$. Die Idee dieser Regel ist der PSP-Regel[5] aus UNITY [18] sehr ähnlich, in der eine Leadsto-Eigenschaft mit einer unless-Eigenschaft zu einer neuen Leadsto-Eigenschaft kombiniert wird.

### 7.7.4 Leadsto-Graphen

Die Regeln für die transitive und disjunktive Verknüpfung von Leadsto-Eigenschaften fassen wir zu einer allgemeineren Regel zusammen, die sehr anschaulich ist. Insbesondere besitzt diese Regel eine suggestive graphische Darstellung. Wir betrachten zunächst wieder ein kleines Beispiel: Die drei Leadsto-Eigenschaften $\varphi \rightsquigarrow \psi \vee \psi'$, $\psi \rightsquigarrow \chi$ und $\psi' \rightsquigarrow \chi$ stellen wir graphisch wie in Abb. 7.10 dar. Diese Darstellung

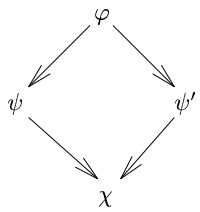



Abbildung 7.10: Ein einfacher Leadsto-Graph

nennen wir *Leadsto-Graph*. Der Leadsto-Graph suggeriert bereits, daß mit den drei genannten Leadsto-Eigenschaften auch die Eigenschaft $\varphi \rightsquigarrow \chi$ gilt.
Allgemein ist ein Leadsto-Graph eine kombinatorische[6] fundierte Ordnung mit genau einem minimalen und einem maximalen Element. Die Elemente der Ordnung sind mit Zustandsaussagen beschriftet. In Ermangelung eines weiteren Symbols für strikte Ordnungen benutzen wir für diese Ordnung wieder das Symbol $<$ und die zugehörigen Notationen $\lessdot$ und ${}^\bullet x$. Da wir in der Regel (LTO-GRAPH) keine strukturellen Aussagen über die betrachtete Systemkomponente machen, bezieht sich $<$, $\lessdot$ und ${}^\bullet x$ in dieser Regel immer auf den betrachteten Leadsto-Graphen.

[5] PSP steht für progress-safety-progress.

[6] Es ist eigentlich nicht notwendig zu fordern, daß die Ordnung kombinatorisch ist. Wir können aber die Regel (LTO-GRAPH) einfacher formulieren, wenn die Ordnung kombinatorisch ist.

**Definition 7.5 (Leadsto-Graph)**

Sei $(X, <)$ eine kombinatorische fundierte Ordnung mit einem kleinsten und einem größten Element und $f : X \to \mathbf{ZA}(S, V)$ eine Abbildung.

Wir nennen $(X, <, f)$ einen *Leadsto-Graphen*.

Graphisch repräsentieren wir einen Leadsto-Graphen wie in Abb 7.10, wobei Pfeile von größeren zu kleineren Elementen zeigen. Wir wählen diese Repräsentation **im Gegensatz zur bisherigen Repräsentation von Ordnungen**, weil so die Richtung der Pfeile der Richtung der Leadsto-Eigenschaft entspricht, die wir im folgenden Satz betrachten.

**Satz 7.6 (Leadsto-Graphen und Leadsto-Eigenschaften)**

Sei $(X, <, f)$ ein Leadsto-Graph, $a$ das größte und $b$ das kleinste Element von $(X, <)$.

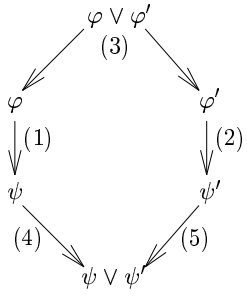
Wenn für jedes $x \in X \setminus \{b\}$ des Leadsto-Graphen $\Sigma, \mathcal{E} \models f(x) \rightsquigarrow \bigvee_{y \in \bullet x} f(y)$ gilt, dann gilt auch $\Sigma, \mathcal{E} \models f(a) \rightsquigarrow f(b)$.

**Beweis:** durch noethersche Induktion über die Relation $\lessdot$ und die Transitivität der Erreichbarkeit. □

Dieser Satz ist bereits die Beweisregel (LTO-GRAPH): Wenn wir für jeden (bis auf den kleinsten) Knoten eines Leadsto-Graphen die entsprechende Leadsto-Eigenschaft $f(x) \rightsquigarrow \bigvee_{y \in \bullet x} f(y)$ bewiesen haben, dann ist mit der Regel (LTO-GRAPH) auch $f(a) \rightsquigarrow f(b)$ bewiesen. Wenn diese Eigenschaften gelten, dann nennen wir den Leadsto-Graphen *gültig* (in $\Sigma$ unter Annahme $\mathcal{E}$).

| | | |
|---|---|---|
| (LTO-GRAPH) | Für jedes $x \in X \setminus \{b\}$ gilt: $\Sigma, \mathcal{E} \models f(x) \rightsquigarrow \bigvee_{y \in \bullet x} f(y)$ / $\Sigma, \mathcal{E} \models f(a) \rightsquigarrow f(b)$ | $(X, <, f)$ ist ein Leadsto-Graph mit kleinstem Element $b$ und größtem Element $a$ |

Als Beispiel für die Anwendung der Regel (LTO-GRAPH) wird in Tabelle 7.5 die Disjunktionsregel für Leadsto-Eigenschaften bewiesen: Wenn $\varphi \rightsquigarrow \psi$ und $\varphi' \rightsquigarrow \psi'$ gilt, dann auch $\varphi \vee \varphi' \rightsquigarrow \psi \vee \psi'$.
Bei der Anwendung der Regel (LTO-GRAPH) geben wir die Referenzen (im Beispiel (1)–(5)) auf die bereits bewiesenen Leadsto-Eigenschaften an. Damit diese Referenzen den einzelnen Knoten des Leadsto-Graphen zugeordnet werden können, geben wir die Referenzen




| | | |
|---|---|---|
| (1) | $\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \psi$ | Annahme |
| (2) | $\Sigma, \mathcal{E} \models \varphi' \rightsquigarrow \psi'$ | Annahme |
| (3) | $\Sigma, \mathcal{E} \models \varphi \vee \varphi' \rightsquigarrow \varphi \vee \varphi'$ | (INV-$\rightsquigarrow$), (TAUT-INV) |
| (4) | $\Sigma, \mathcal{E} \models \psi \rightsquigarrow \psi \vee \psi'$ | (INV-$\rightsquigarrow$), (TAUT-INV) |
| (5) | $\Sigma, \mathcal{E} \models \psi' \rightsquigarrow \psi \vee \psi'$ | (INV-$\rightsquigarrow$), (TAUT-INV) |
| (6) | $\Sigma, \mathcal{E} \models \varphi \vee \varphi' \rightsquigarrow \psi \vee \psi'$ | (LTO-GRAPH) mit (1)–(5) |

Tabelle 7.5: Beweis der Disjunktionsregel

auch in den entsprechenden „Verzweigungen“ des Leadsto-Graphen an. Außerdem fassen wir in diesem Beispiel die Anwendung der Regel (TAUT-INV) zum Beweis von $\Sigma, \mathcal{E} \models \Box[\varphi \vee \varphi' \Rightarrow \varphi \vee \varphi']$ in Zeile (3) mit der Anwendung der Regel (INV-$\rightsquigarrow$) zusammen. So vermeiden wir, daß ein Beweis durch triviale Beweisschritte unnötig lang wird.

In Tabelle 7.6 haben wir nochmals alle Regeln zum Beweis von Leadsto-Eigenschaften zusammengestellt.

## 7.8 Ein kleines Beispiel

Wir werden nun anhand eines Beispiels ein Gefühl für die Anwendung der Verifikationsregeln vermitteln. Wir geben dazu eine einfache Implementierung für das Mutex-Modul aus dem vorangegangenen Kapitel an. Zur Erinnerung wiederholen wir die Beschreibung des Mutex-Moduls. Wir verallgemeinern dabei das Mutex-Modul etwas, indem wir Platzhalter $X_1$ und $X_2$ für interne Stellenmengen einführen. Die Schnittstelle des Mutex-Moduls ist in Abb. 7.11 dargestellt. Der RG-Graph des Mutex-Moduls ist in Abb. 7.12 dargestellt.
Die Systemkomponente $\Sigma$ aus Abb. 7.13 ist eine Implementierung des Mutex-Moduls, wenn $X_1$ und $X_2$ mit der leeren Menge instantiiert werden. Dies werden wir sowohl informell als auch formal beweisen. Der Beweis im laufenden Text ist eher informell gehalten. In den Tabellen 7.7 und 7.8 ist dieser informelle Beweis in einzelne Schritte unter Anwendung der Beweisregeln aufgelöst. Im laufenden Text stellen wir die Bezüge zum formalen Beweis durch Angabe der Nummer der

(PROGR)
$$\frac{\begin{array}{l}\Sigma, \mathcal{E} \models \Box[\varphi \Rightarrow \bigwedge {}^\bullet t_0] \\ \langle \varphi \rangle \ T_1 \ \langle \psi \rangle \\ \langle \varphi \vee \chi \rangle \ T_1' \ \langle \psi \rangle \\ \langle \varphi \vee \chi \rangle \ T_2 \ \langle \varphi \vee \psi \rangle \\ \Sigma, \mathcal{E} \models (\varphi \vee \chi) \wedge \theta^{=}_{Z \cup {}^\bullet t_0} \textbf{ alw} \\ \qquad (\varphi \ \vee \ \theta^{\leq}_{Z} \wedge \chi \ \vee \ \theta_{t_0} \ \vee \ \psi)\end{array}}{\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \psi}$$

$t_0 \in T^P_\Sigma$
$T_1 \cup T_1' = ({}^\bullet t_0)^\bullet$
$Z = {}^\bullet T_1$
$T_2 = \{t \mid Z \cap {}^\bullet t \cap t^\bullet\} \setminus T_1$
$\theta^{=}_{Z \cup {}^\bullet t_0} \mathrel{\widehat{=}} \bigwedge_{s \in Z} \#\{s\} = v_s$
$\theta^{\leq}_{Z} \mathrel{\widehat{=}} \bigwedge_{s \in Z} \#\{s\} \leq v_s$
$\theta_{t_0} \mathrel{\widehat{=}} \bigvee_{s \in {}^\bullet t_0} \#\{s\} < v_s$
wobei $v_s$ paarweise verschiedene neue Variablen sind.

(PROGR1)
$$\frac{\left\langle \bigwedge {}^\bullet t_0 \right\rangle \ ({}^\bullet t_0)^\bullet \ \langle \psi \rangle}{\Sigma, \mathcal{E} \models \bigwedge {}^\bullet t_0 \rightsquigarrow \psi}$$
$t_0 \in T^P_\Sigma$

(FAIR)
$$\frac{\begin{array}{l}\Sigma, \mathcal{E} \models \Box[(\varphi \wedge s) \Rightarrow \bigwedge {}^\bullet t_0] \\ \Sigma, \mathcal{E} \models \varphi \textbf{ alw } \psi \\ \langle \varphi \rangle \ ({}^\bullet t_0 \setminus \{s\})^\bullet \ \langle \psi \rangle \\ \Sigma, \mathcal{E} \models \varphi \rightsquigarrow s \vee \psi\end{array}}{\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \psi}$$
$t_0 \in T^F_\Sigma$
$|{}^\bullet t_0| \geq 2$
$s \in {}^\bullet t_0$

(INV-$\rightsquigarrow$)
$$\frac{\Sigma, \mathcal{E} \models \Box[\varphi \Rightarrow \psi]}{\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \psi}$$

(W-$\rightsquigarrow$)
$$\frac{\begin{array}{l}\Sigma, \mathcal{E} \models \Box[\varphi \Rightarrow \varphi'] \\ \Sigma, \mathcal{E} \models \Box[\psi' \Rightarrow \psi] \\ \Sigma, \mathcal{E} \models \varphi' \rightsquigarrow \psi'\end{array}}{\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \psi}$$

(SYNC)
$$\frac{\begin{array}{l}\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \psi \\ \Sigma, \mathcal{E} \models \varphi \wedge \neg\psi \textbf{ alw } \varphi\end{array}}{\Sigma, \mathcal{E} \models \varphi \rightsquigarrow \varphi \wedge \psi}$$

(LTO-GRAPH)
$$\frac{\begin{array}{l}\text{Für jedes } x \in X \setminus \{b\} \text{ gilt:} \\ \Sigma, \mathcal{E} \models f(x) \rightsquigarrow \bigvee_{y \in {}^\bullet x} f(y)\end{array}}{\Sigma, \mathcal{E} \models f(a) \rightsquigarrow f(b)}$$
$(X, <, f)$ ist ein Leadsto-Graph mit kleinstem Element $b$ und größtem Element $a$

Tabelle 7.6: Regeln zum Beweis von Leadsto-Eigenschaften

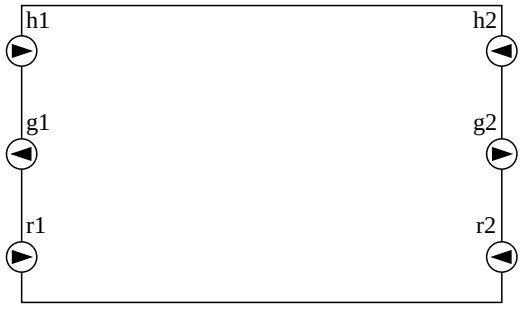



Abbildung 7.11: Schnittstelle des Mutex-Moduls

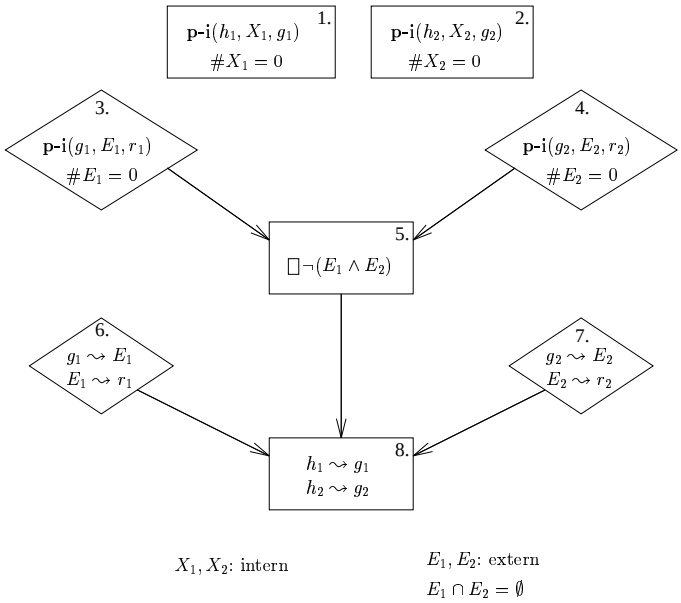
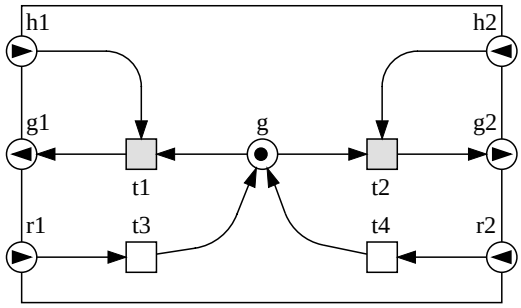



Abbildung 7.12: RG-Graph des Mutex-Moduls




$$X_1 \mathrel{\widehat{=}} \emptyset, X_2 \mathrel{\widehat{=}} \emptyset$$

Abbildung 7.13: Implementierung $\Sigma$ des Mutex-Moduls

entsprechenden Beweiszeile her. Die Beweise sind jedoch unabhängig voneinander lesbar.
Im formalen Beweis in Tabelle 7.7 und 7.8 werden nur die temporalen Aussagen aufgeführt und bewiesen. Aussagenlogische und strukturelle Anforderungen an das Systems ergeben sich in den Beweisschritten aus der jeweils angewendeten Regel. Wie im vorangegangenen Beispiel, werden wir an manchen Stellen im formalen Beweis mehrere einfache Beweisschritte zusammenfassen.
Wir müssen zeigen, daß die Implementierung $\Sigma$ die Eigenschaften der Knoten $1, 2, 5$ und 8 des RG-Graphen unter den jeweiligen Annahmen an die Umgebung erfüllt.

1./2. Aus der Struktur von $\Sigma$ folgt unmittelbar die partielle S-Invariante **p-i**$(h_1, \emptyset, g_1)$ (1).

Die Aussage $\#\emptyset = 0$ ist eine zustandslogische Tautologie. Deshalb gilt diese Aussage auch im initialen Zustand (2).

Mit (1) und (2) und $X_1 \mathrel{\widehat{=}} \emptyset$ sind die Eigenschaften in Knoten 1 des RG-Graphen gezeigt.

Die Eigenschaften des 2. Knotens lassen sich symmetrisch dazu beweisen. Dies führen wir hier jedoch nicht aus.

5. Wir zeigen nun, daß die Mutex-Bedingung gilt, wenn die Forderungen 3 und 4 von der Umgebung erfüllt sind ($\mathcal{E}_5$).

Dazu benutzen wir die partielle S-Invariante **p-i**$(\{r_1, r_2\}, \{g\}, \{g_1, g_2\})$, die unmittelbar aus der Struktur von

$\Sigma$ folgt (3). Diese partielle S-Invariante können wir mit den partiellen S-Invarianten, die die Umgebung gewährleistet (4) und (5) zur S-Invariante $\mathbf{inv}(E_1 \cup E_2 \cup \{g_1, g_2, g, r_1, r_2\})$ zusammensetzen (7).

Wegen der Anfangsmarkierung von $\Sigma$ gilt initial die Aussage $\#\{g_1, g_2, g, r_1, r_2\} = 1$. Zusammen mit den Annahmen über die Umgebung wissen wir, daß die S-Invariante (7) initial genau eine Marke besitzt. Da $E_1$ und $E_2$ beide von dieser S-Invariante überdeckt werden und disjunkt sind, ist die Invariante $\Box\neg(E_1 \wedge E_2)$ gezeigt (12).

| | | |
|---|---|---|
| (1) | $\Sigma \models \mathbf{p\text{-}i}(h_1, \emptyset, g_1)$ | (P-INV) mit $T_1 \mathrel{\widehat{=}} T_\Sigma$ und $T_2 \mathrel{\widehat{=}} \emptyset$ |
| (1') | $\Sigma \models \mathbf{p\text{-}i}(h_2, \emptyset, g_2)$ | symmetrisch zu (1) |
| (2) | $\Sigma \models \#\emptyset = 0$ | (TAUT) |
| | $\mathcal{E}_5 \mathrel{\widehat{=}} \{\mathbf{p\text{-}i}(g_1, E_1, r_1), \#E_1 = 0, \mathbf{p\text{-}i}(g_2, E_2, r_2), \#E_2 = 0\}$ | |
| (3) | $\Sigma, \mathcal{E}_5 \models \mathbf{p\text{-}i}(\{r_1, r_2\}, \{g\}, \{g_1, g_2\})$ | (P-INV) mit $T_1 \mathrel{\widehat{=}} T_\Sigma$ und $T_2 \mathrel{\widehat{=}} \emptyset$ |
| (4) | $\Sigma, \mathcal{E}_5 \models \mathbf{p\text{-}i}(g_1, E_1, r_1)$ | (ASS) |
| (5) | $\Sigma, \mathcal{E}_5 \models \mathbf{p\text{-}i}(g_2, E_2, r_2)$ | (ASS) |
| (6) | $\Sigma, \mathcal{E}_5 \models \mathbf{p\text{-}i}(r_2, E_1 \cup \{g_1, r_1, g\}, g_2)$ | ($\cup$-P-INV) mit (3),(4) |
| (7) | $\Sigma, \mathcal{E}_5 \models \mathbf{inv}(\{g_1, g_2, r_1, r_2, g\} \cup E_1 \cup E_2)$ | ($\cup$-P-INV) mit (5),(6) und (S-INV) |
| (8) | $\Sigma, \mathcal{E}_5 \models \#\{g_1, g_2, g, r_1, r_2\} = 1$ | (INIT) |
| (9) | $\Sigma, \mathcal{E}_5 \models (\#E_1 = 0) \wedge (\#E_2 = 0)$ | (ASS), (ASS) und (CONJ) |
| (10) | $\Sigma, \mathcal{E}_5 \models \#(E_1 \cup E_2 \cup \{g_1, g_2, g, r_1, r_2\}) = 1$ | (CONJ) mit (8),(9) und Abschwächung mit (MP) |
| (11) | $\Sigma, \mathcal{E}_5 \models \Box[E_1 \Rightarrow \neg E_2]$ | ($\leq$1-INV) mit (7),(10) |
| (12) | $\Sigma, \mathcal{E}_5 \models \Box\neg(E_1 \wedge E_2)$ | (EQUIV) auf (11) |

Tabelle 7.7: Formale Verifikation der Mutex-Implementierung I

Damit haben wir die Eigenschaft von Knotens 5. des RG-Graphen gezeigt.

8. Wir erweitern nun die Annahmen um die Eigenschaften der Knoten 6 und 7 des RG-Graphen ($\mathcal{E}_8$).

   Zunächst zeigen wir die Eigenschaft $h_1 \rightsquigarrow g$ (18). Dazu zeigen wir, daß der Leadsto-Graph aus Abb. 7.14. gültig ist. Aus obiger

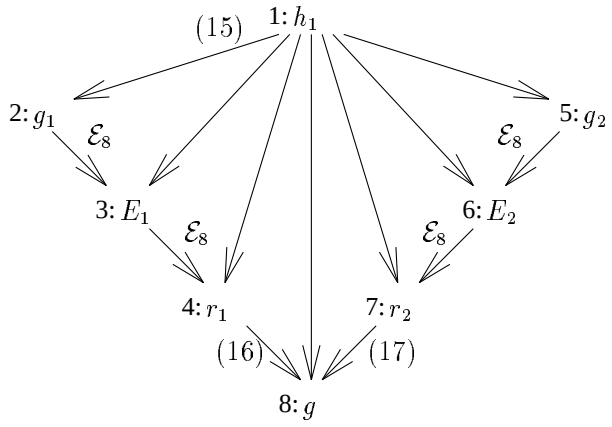



Abbildung 7.14: Leadsto-Graph für die Eigenschaft $h_1 \rightsquigarrow g$

S-Invariante (7) und der initialen Bedingung (10) folgt die Invariante $g_1 \vee E_1 \vee r_1 \vee g \vee r_2 \vee E_2 \vee g_2$. Insbesondere bedeutet dies $h_1 \rightsquigarrow g_1 \vee E_1 \vee r_1 \vee g \vee r_2 \vee E_2 \vee g_2$ (15). Die Forderung für die Knoten 2, 3, 5 und 6 des Leadsto-Graphen folgt unmittelbar aus den Annahmen für die Umgebung $\mathcal{E}_8$. Für die Knoten 4 bzw. 7 ist zu zeigen $r_1 \rightsquigarrow g$ bzw. $r_2 \rightsquigarrow g$ (16) und (17). Diese Eigenschaften sind wegen der Transitionen $t_3$ bzw. $t_4$ gewährleistet, die mit keiner anderen Transition in Konflikt stehen und den gewünschten Übergang gewährleisten.

Da $h_1$ nur ungültig wird, wenn dabei $g_1$ gültig wird (20), und wegen $h_1 \rightsquigarrow g$ muß $t_1$ aufgrund der Fairnessannahme irgendwann im Ablauf auftreten. Somit erhalten wir unter den Annahmen über die Umgebung die Eigenschaft $h_1 \rightsquigarrow g_1$ (22).

Die Eigenschaft $h_2 \rightsquigarrow g_2$ kann wieder symmetrisch dazu bewiesen werde (22'). Damit sind auch die Eigenschaften von Knoten 8. des RG-Graphen erfüllt.

Insgesamt ist damit gezeigt, daß $\Sigma$ eine Implementierung des Mutex-Moduls ist (23).

| | | |
|---|---|---|
| | $\mathcal{E}_8 \mathrel{\widehat{=}} \mathcal{E}_5 \cup \{g_1 \rightsquigarrow E_1, E_1 \rightsquigarrow r_1, g_2 \rightsquigarrow E_2, E_2 \rightsquigarrow r_2\}$ | |
| (13) | $\Sigma, \mathcal{E}_8 \models \Box[g_1 \vee E_1 \vee r_1 \vee g \vee g_2 \vee E_2 \vee r_2]$ | ($\geq$1-INV) auf (7), (10) mit $Z_1 \mathrel{\widehat{=}} \emptyset$ |
| (14) | $\Sigma, \mathcal{E}_8 \models \Box[h_1 \Rightarrow (g_1 \vee E_1 \vee r_1 \vee g \vee g_2 \vee E_2 \vee r_2)]$ | (TAUT-INV), (W-INV) auf (13) |
| (15) | $\Sigma, \mathcal{E}_8 \models h_1 \rightsquigarrow (g_1 \vee E_1 \vee r_1 \vee g \vee g_2 \vee E_2 \vee r_2)$ | (INV-$\rightsquigarrow$) auf (14) |
| (16) | $\Sigma, \mathcal{E}_8 \models r_1 \rightsquigarrow g$ | (PROGR1) mit $t_0 \mathrel{\widehat{=}} t_3$ und $\psi \mathrel{\widehat{=}} g$ |
| (17) | $\Sigma, \mathcal{E}_8 \models r_2 \rightsquigarrow g$ | (PROGR1) mit $t_0 \mathrel{\widehat{=}} t_4$ und $\psi \mathrel{\widehat{=}} g$ |
| (18) | $\Sigma, \mathcal{E}_8 \models h_1 \rightsquigarrow g$ | (LTO-GRAPH) mit Abb. 7.14 und (15),(16), (17) und $\mathcal{E}_8$ |
| (19) | $\Sigma, \mathcal{E}_8 \models h_1 \rightsquigarrow g \vee g_1$ | (W-$\rightsquigarrow$) auf (18) und (TAUT-INV) |
| (20) | $\Sigma, \mathcal{E}_8 \models h_1$ **alw** $g_1$ | (ALW) |
| (21) | $\Sigma, \mathcal{E}_8 \models \Box[h_1 \wedge g \Rightarrow h_1 \wedge g]$ | (TAUT-INV) |
| (22) | $\Sigma, \mathcal{E}_8 \models h_1 \rightsquigarrow g_1$ | (FAIR) mit (21),(20) (19) und $t_0 \mathrel{\widehat{=}} t_1$ und $s \mathrel{\widehat{=}} g$ |
| (22') | $\Sigma, \mathcal{E}_8 \models h_2 \rightsquigarrow g_2$ | symmetrisch zu (22) |
| (23) | $\Sigma \models$ MUTEX | (1),(2), (1'), (12), (22), und (22') |

Tabelle 7.8: Formale Verifikation der Mutex-Implementierung II

## 7.9 Literatur

Die hier eingeführten Beweisregeln für Invarianten und Leadsto-Eigenschaften orientieren sich an den Beweisregeln von Manna und Pnueli [56]. Lediglich die Regeln (PROGR) und (FAIR) sind speziell auf unser Systemmodell zugeschnitten. In der Regel (PROGR) schlägt sich auch nieder, daß unser Leadsto-Operator auf Halbordnungen definiert ist. Mit dieser Regel sind auch Leadsto-Eigenschaften beweisbar, die für den sequentiellen Leadsto-Operator nicht gelten.
Die logische Formulierung von partiellen S-Invarianten und S-Invarianten ist neu. Damit sind auch die hier eingeführten Regel zur Kombination von partiellen S-Invarianten und zum Beweis von Invarianten mit Hilfe von S-Invarianten (als Regeln einer Logik formuliert) neu. Allerdings werden S-Invarianten schon lange zum Beweis von Invarianten von Petrinetzen benutzt. Der Regel (SYNC) liegt eine ähnliche Idee wie der PSP-Regel in UNITY [18] zugrunde.

Die Regel (LTO-GRAPH) und der Begriff des Leadsto-Graphen sind von Owicki und Lamport [67] übernommen (von ihnen *proof lattices* genannt). In [56] werden etwas speziellere Graphen zum Beweis von Leadsto-Eigenschaften benutzt, die *proof diagrams* genannt werden. Wir benutzen Leadsto-Graphen neben den „Ableseregeln“ (PROGR) und (FAIR) als zentrale Regel zum Beweis von Leadsto-Eigenschaften. Insbesondere sind die Transitivität, die Reflexivität und — wie wir als Beispiel gezeigt haben — die Disjunktion von Leadsto-Eigenschaften im wesentlichen Spezialfälle dieser Regel.
Die Verwendung von Zusicherungen zur Formulierung und zum Beweis von Eigenschaften von Programmen ist sehr alt [28, 38]. Sie werden in den meisten gängigen Beweiskalkülen für den Übergang von strukturellen Eigenschaften des Systems zu temporalen Aussagen benutzt [18, 56]. In manchen Beweiskalkülen (z.B. [77, 34, 87]) können Eigenschaften auch direkt aus der Systemstruktur abgeleitet werden; diese sogenannten Ableseregeln (pickup rules) können wir als Spezialfälle der Zusicherungen ansehen. Insbesondere können S-Invarianten als ein Spezialfall von Zusicherungen angesehen werden.

Die Beweisregeln, die wir hier vorgestellt haben, orientieren sich im wesentlichen an Beispielen. Die Logik und der hier eingeführte Beweiskalkül soll hauptsächlich zeigen, daß unser Modulkonzept in Kombination mit einer Logik zum modularen Entwurf geeignet ist. Ein vollständiger Beweiskalkül für diese Logik ist nicht Gegenstand dieser

Arbeit.
Generell scheint es sehr schwer zu sein, vollständige Beweiskalküle für (hinreichend mächtige) Logiken für verteilte Abläufe zu erhalten. Standardtechniken zum Beweis der Vollständigkeit für die klassische temporale Logik (siehe z.B. [46]) sind meist nicht übertragbar. Peled und Pnueli [70] geben für einen wichtigen Ausschnitt der Logik ISTL$^*$ einen vollständigen Beweiskalkül an.

# Kapitel 8

# Ein Beispiel

In diesem Kapitel werden wir an einem Beispiel demonstrieren, wie man ein Modul in Teilmodule zerlegen kann. Wir werden zeigen, daß die Komposition dieser Teilmodule die gewünschten Eigenschaften des Gesamtmoduls erfüllt. Anschließend werden die Teilmodule unabhängig voneinander implementiert. Die Korrektheit der Implementierung der Teilmodule werden wir mit Hilfe der Beweisregeln aus dem vorangegangenen Kapitel beweisen.
Zum besseren Verständnis der Zerlegung des Moduls in zwei Teilmodule geben wir zuerst eine Implementierung des Gesamtmoduls an. Diese Implementierung zerlegen wir in zwei Teilkomponenten. Für diese Teilkomponenten überlegen wir uns dann eine geeignete Spezifikation und geben einen RG-Graphen dafür an. Prinzipiell ist es möglich, ein Modul unabhängig von einer speziellen Implementierung in Teilmodule zu zerlegen. Häufig ist es jedoch zweckmäßig, bereits bei der Zerlegung eines Moduls eine Idee für die Implementierungen der Teilmodule zu haben.

Das Beispiel, das wir hier betrachten, ist eine verfeinerte Version der Mutex-Komponente aus dem vorangegangenen Kapitel. Diese Mutex-Komponente hatte den Nachteil, daß von zwei Seiten auf die Gabel $g$ zugegriffen wird. Deshalb läßt sich diese Lösung nicht in zwei symmetrische Komponenten aufteilen, die jeweils nur über Nachrichtenaustausch miteinander kommunizieren.
Wir betrachten nun eine Implementierung des Mutex-Moduls, die sich in zwei Teilkomponenten zerlegen läßt. Diese Lösung wurde von der Münchner uNETy-Gruppe entwickelt und in verschiedenen Fallstudien

(als Gesamtsystem) untersucht [63, 45].
Wir beweisen die Korrektheit dieser Implementierung in zwei Schritten: Zunächst zerlegen wir das Mutex-Modul in zwei komponierbare Module und zeigen dann, daß ihre Komposition das Mutex-Modul impliziert. Entsprechend zerlegen wir die Implementierung in zwei Komponenten und zeigen, daß diese das jeweilige Teilmodul implementieren.

## 8.1 Eine verteilte Implementierung

In Abb. 8.1 ist die „verteilte“ Implementierung des Mutex-Moduls dargestellt. Die Stelle $g$ der bisherigen Implementierung haben wir im wesentlichen auf die Stellen $s_1$ und $s_2$ aufgeteilt. Die „Gabel“ kann über die Stellen $ack_1$ und $ack_2$ ausgetauscht werden. Wird an der linken Schnittstelle eine Anforderung $h_1$ abgesetzt, so ist entweder $s_1$ markiert und die Gabel kann unmittelbar über $g_1$ zugeteilt werden oder $s_1$ ist nicht vorhanden. Dann wird die Gabel von der rechten Komponente mit $rq_1$ angefordert. Sobald die Gabel in $ack_1$ übergeben wird, kann sie an die linke Umgebung über $g_1$ weitergegeben werden. Um die Lebendigkeitsanforderung zu gewährleisten, müssen wir für Transition $t_5$ und $t_5'$ Fairness fordern. Dadurch wird eine Aufforderung die Gabel an die andere Seite zu übergeben nicht unendlich oft übergangen.

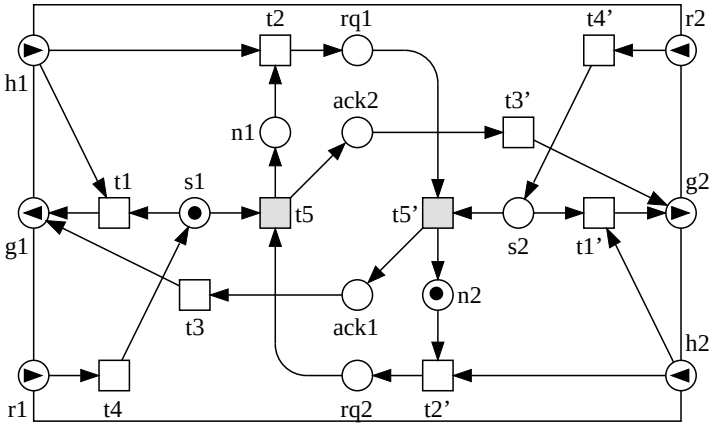



Abbildung 8.1: Verteilte Implementierung $\Sigma$ des Mutex-Moduls

Im Gegensatz zur bisherigen Darstellung der Schnittstelle und der Implementierung ist diese Lösung nicht achsensymmetrisch sondern punktsymmetrisch dargestellt. Dadurch ist die Schnittstelle auf der rechten Seite von unten nach oben zu lesen. Diese punktsymmetrische Darstellung ist für die graphische Darstellung der Zerlegung in Teilkomponenten besser geeignet. Die beabsichtigte Zerlegung ist in Abb. 8.2 dargestellt. Entsprechend dieser Zerlegung werden wir nun das Mutex-Modul in Teilmodule aufteilen. Natürlich ist die Zerlegung des Mutex-Moduls möglich, ohne eine spezielle Implementierung und deren Zerlegung festzulegen. Es ist jedoch bei der Erstellung der RG-Graphen der Teilmodule hilfreich, wenn wir eine Implementierung vor Augen haben. Darum haben wir mit der Vorstellung einer Implementierung des Mutex-Moduls begonnen.

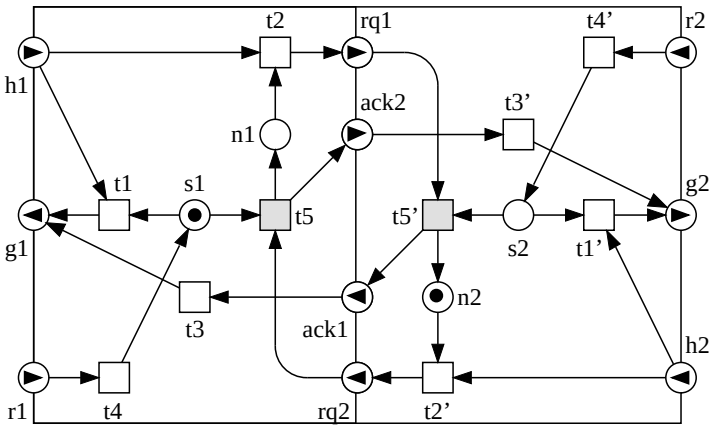



Abbildung 8.2: Zerlegung in Teilkomponenten

## 8.2 Zerlegung in Teilmodule

Den RG-Graphen und die Schnittstelle des Mutex-Moduls kennen wir bereits aus den vorangegangenen Kapiteln. Wir werden nun das Mutex-Modul, wie oben angedeutet, in Teilmodule zerlegen. Dabei konzentrieren wir uns auf das linke Teilmodul, da das rechte Teilmodul im wesentlichen symmetrisch dazu ist. Wir nennen das linke Teilmodul im folgenden L-Mutex. In Abb. 8.3 ist die Schnittstelle von L-Mutex

dargestellt. Abbildung 8.4 zeigt den RG-Graphen von L-Mutex. Die

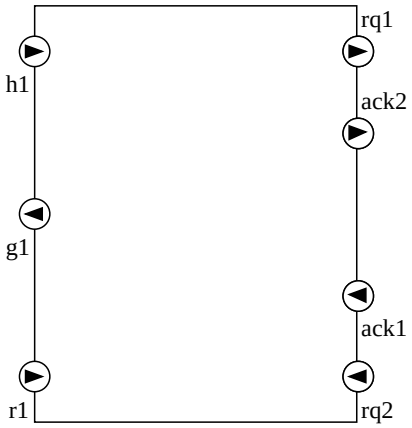



Abbildung 8.3: Schnittstelle von L-Mutex

Bedeutung der einzelnen Knoten werden wir nun informell beschreiben. Im RG-Graphen kommen jeweils drei interne ($X_1$, $Y_1$ und $Z_1$) und externe Platzhalter ($A_1$, $B_1$ und $E_1$) vor. Die Menge $E_1$ entspricht wie bisher dem kritischen Bereich der linken Umgebung. Die Stellen aus $A_1$ charakterisieren den Zustand der rechten Umgebung, wenn sie die Anforderung $rq_1$ zwar registriert, aber noch nicht akzeptiert ($ack_1$) hat. Die Menge $B_1$ entspricht dem Zustand, in dem die rechte Umgebung die Gabel besitzt. Die internen Platzhalter $X_1$ und $Y_1$ werden in der konkreten Implementierung durch die leere Menge ersetzt. Im allgemeinen steht $X_1$ für den internen Zustand, in dem die Komponente eine Anforderung $rq_2$ registriert, aber noch nicht akzeptiert $ack_2$ hat (symmetrisch zu $A_1$). Die Menge $Y_1 \cup A_1 \cup \{rq_1, ack_1\}$ steht für den Zustand, in dem die Komponente die Anforderung $h_1$ bereits zur Kenntnis genommen hat, die Gabel aber noch nicht über $g_1$ an die linke Umgebung zugeteilt hat. In unserer Implementierung gilt $Z_1 = \{s1\}$. Allgemein entspricht $Z_1$ dem Zustand, in dem die linke Teilkomponente die Gabel verwaltet.

Die Knoten 1–6 des RG-Graphen sind Formalisierungen diese informellen Anforderungen an die Platzhalter. Dabei beschreiben Knoten 1–3 die Eigenschaften der Komponente und 4–6 die nötigen Annahmen über die Umgebung. Insbesondere entspricht die Forderung von

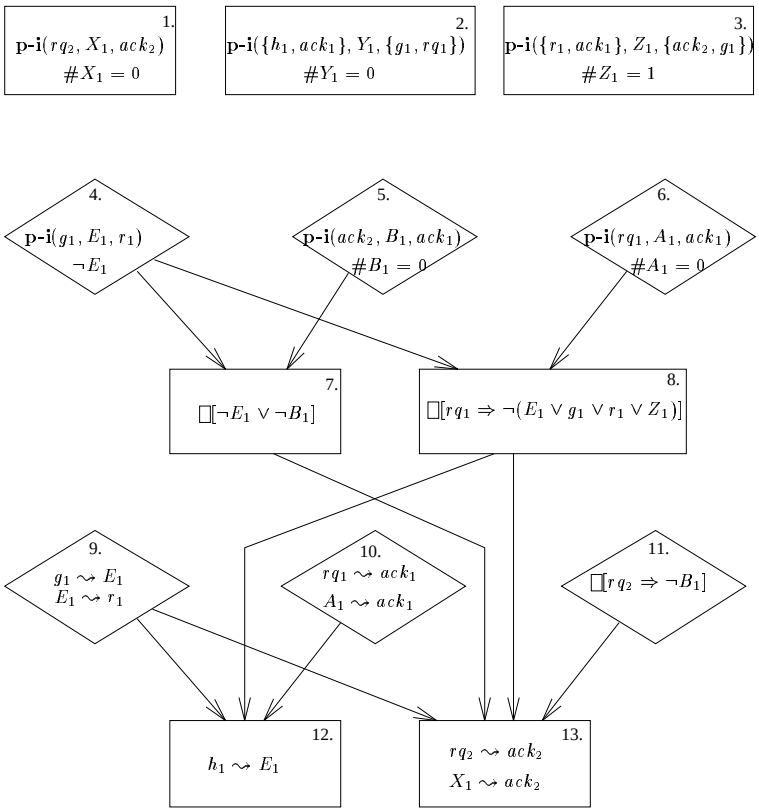



Abbildung 8.4: RG-Graph von L-Mutex

Knoten 4 der Annahme, daß die linke Umgebung eine zugeteilte Gabel nicht verliert oder vervielfältigt.
Die Annahmen 4 und 5 über die Umgebung reichen bereits aus, um den wechselseitigen Ausschluß zu gewährleisten. Dabei wird hier die Eigenschaft $\Box[\neg E_1 \vee \neg B_1]$ gewährleistet, da $E_2$ in dieser Spezifikation nicht vorkommt. $E_2$ können wir uns aber als Teilmenge von $B_1$ vorstellen. Die Annahmen 4 und 6 über die Umgebung genügen, um zu gewährleisten, daß die linke Komponente nur dann eine Gabel von der rechten Komponente anfordert ($rq_1$), wenn sie selbst (und ihre linke Umgebung) keine Gabel besitzt (G-Knoten 8).
Gewährleistet die Umgebung zusätzlich die entsprechende Eigenschaft (R-Knoten 11), dann garantiert die Komponente, daß jede Anforderung $rq_2$ mit einem $ack_2$ beantwortet wird (G-Knoten 13).
Gewährleistet entsprechend die rechte Umgebung, daß jede Anforderung $rq_1$ irgendwann mit $ack_1$ beantwortet wird (R-Knoten 10), und gibt die linke Umgebung eine zugeteilte Gabel immer wieder zurück (R-Knoten 9), dann führt jede Anforderung der linken Umgebung $h_1$ irgendwann zum Betreten des kritischen Bereiches $E_1$ (G-Knoten 12).
Der RG-Graph des rechten Teilmoduls R-Mutex ist im wesentlichen symmetrisch[1]. Er ist in Abb. 8.5 dargestellt. Lediglich die Knoten 3 bzw. 3' und 5 bzw. 5' unterscheiden sich: $Z_2$ ist initial unmarkiert (d.h. die rechte Komponente besitzt initial keine Gabel), dafür wird von der Umgebung erwartet, daß sich die Gabel dort befindet $\#B_2 = 1$. Da die Schnittstelle ebenfalls symmetrisch zu der von L-Mutex ist und sich auch aus Abb. 8.2 ergibt, stellen wir sie hier nicht noch einmal graphisch dar.

## 8.3 Korrektheit der Zerlegung

Wir zeigen nun, daß die Zerlegung des Mutex-Moduls in die Teilmodule L-Mutex und R-Mutex korrekt ist. Das heißt die Komposition einer beliebigen Implementierung von L-Mutex mit einer beliebigen Implementierung von R-Mutex ist eine Implementierung des Mutex-Moduls. Dazu ist einerseits zu zeigen, daß die Komposition der Schnittstellen der beiden Teilmodule die Schnittstelle des Mutex-Moduls ergibt. Dies ist aufgrund von Abb. 8.2 unmittelbar einsichtig.
Andererseits müssen wir zeigen, daß sich aus den kombinierten RG-Graphen von L-Mutex und R-Mutex der RG-Graph des Mutex-Moduls

[1] Wir vertauschen den Index 1 mit 2.

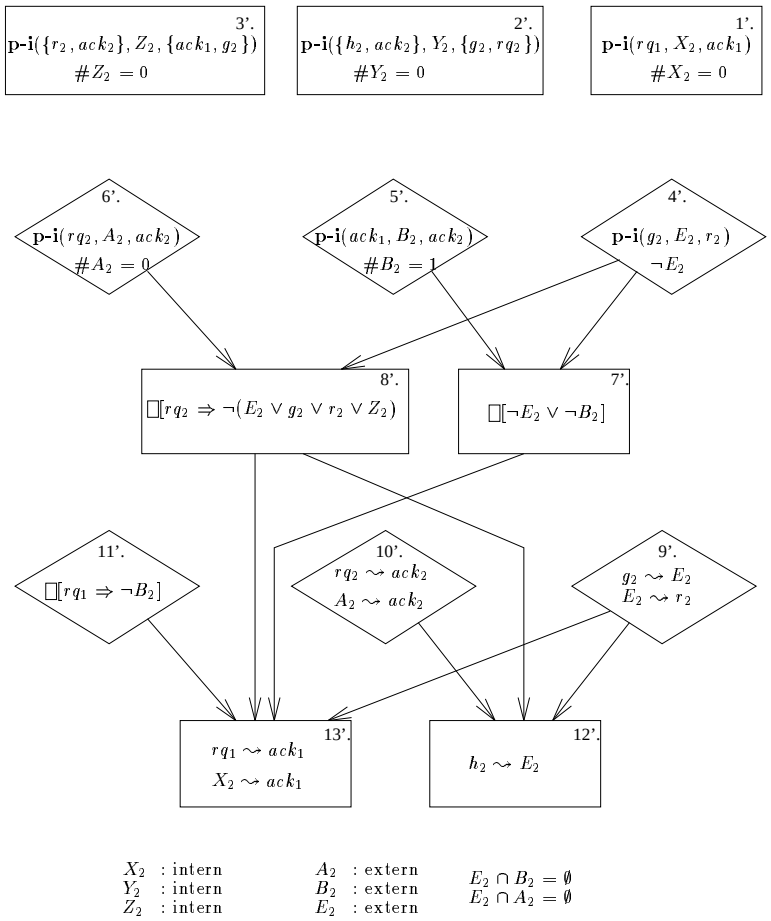



Abbildung 8.5: RG-Graph von R-Mutex

ableiten läßt. Dazu benutzen wir die Regeln aus Abschnitt 5.1: Zunächst legen wir die beiden RG-Graphen unabhängig nebeneinander und fügen dann weitere Kanten in diesen Graphen ein. Anschließend lassen sich einige R-Knoten aus diesem RG-Graphen eliminieren und der verbleibende Graph kann leicht in den RG-Graphen des Mutex-Moduls überführt werden. Dabei nutzen wir die Annahmen über die Schnittstellen der Teilmodule aus.

Abbildung 8.6 zeigt die nebeneinander gelegten RG-Graphen der beiden Komponenten, wobei wir bereits einige zusätzliche Kanten in diesen RG-Graphen eingezeichnet haben. Beim Zusammenfügen dürfen wir die externen Stellenmengen der beiden Module genauer festlegen. Wir setzen $A_1 \mathrel{\widehat{=}} X_2$ und $A_2 \mathrel{\widehat{=}} X_1$. Dies ist zulässig, weil die internen Stellen der einen Teilkomponente externe der anderen sind und umgekehrt. Damit werden $A_1$ und $A_2$ zu internen Stellenmengen des zusammengesetzten Moduls. $E_1$ und $E_2$ bleiben weiterhin unbestimmt, da sie Stellen der Umgebung des zusammengesetzen Systems sein sollen. $B_1$ setzt sich dagegen aus internen und externen Stellenmengen (im komponierten System) zusammen: $B_1 \mathrel{\widehat{=}} Z_2 \cup \{g_2, r_2\} \cup E_2$ und $B_2 \mathrel{\widehat{=}} Z_1 \cup \{g_1, r_1\} \cup E_1$. Bezüglich der Teilkomponenten sind jedoch $B_1$ und $B_2$ jeweils externe Stellenmengen; deshalb ist diese Wahl zulässig.

Im zusammengesetzten Modul fordern wir $E_1 \cap E_2 = \emptyset$: Damit gelten die Disjunktheitsannahmen, die in den Teilmodulen gefordert sind: $E_1 \cap A_1 = E_1 \cap X_2 = \emptyset$, weil $E_1$ externe Stellen des zusammengesetzen Moduls sind und $X_2$ interne. $E_1 \cap B_1 = E_1 \cap (Z_2 \cup \{g_2, r_2\} \cup E_2) = \emptyset$ gilt, weil $Z_2$ und $\{g_2, r_2\}$ intern sind und $E_1$ extern ist und wir $E_1 \cap E_2 = \emptyset$ explizit fordern.

Unter Ausnutzung der Festlegung von $A_1$, $A_2$, $B_1$ und $B_2$ können wir nun die R-Knoten 5, 6, 10 und 11 (und symmetrisch dazu 5', 6', 10' und 11') eliminieren, da sie aus den Eigenschaften der Vorgängerknoten ableitbar sind. Für die R-Knoten 6 und 10 ist dies bereits syntaktisch sichtbar: Mit $A_1 = X_2$ ist die Inschrift von R-Knoten 6 mit der Inschrift des Vorgängers 1' identisch. Ebenso ist die Inschrift von R-Knoten 10 mit dem G-Knoten 13' identisch.

Die Eigenschaft von R-Knoten 11 ist mit der Definition von $B_1$ eine aussagenlogisch äquivalente Umformung der Eigenschaft von G-Knoten 8'. Etwas komplizierter ist die Argumentation für die Elimination von Knoten 5: Aus den partiellen S-Invarianten $\mathbf{p\text{-}i}(\{r_2, ack_2\}, Z_2, \{ack_1, g_2\})$ und $\mathbf{p\text{-}i}(g_2, E_2, r_2)$ folgt mit Regel (∪-

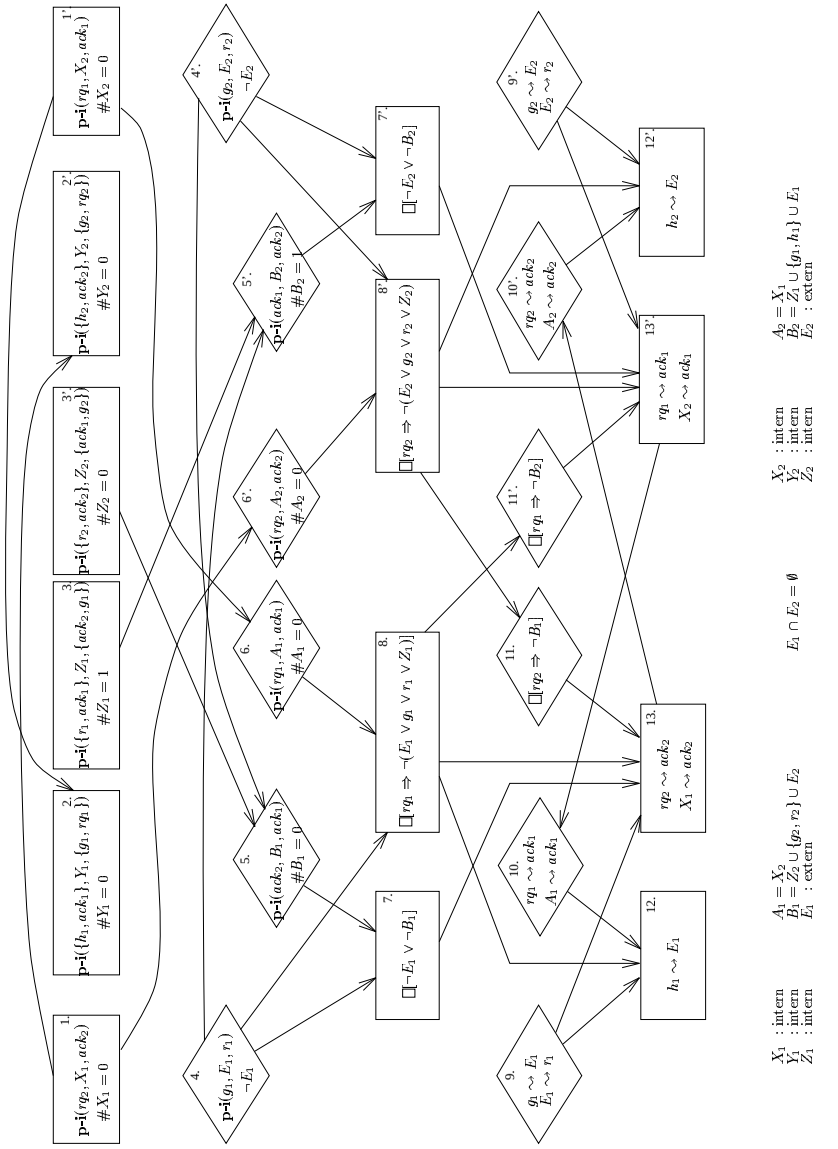



Abbildung 8.6: Kombinierter RG-Graph mit zusätzlichen Kanten

P-INV) unmittelbar[2] $\mathbf{p\text{-}i}(ack_2, Z_2 \cup \{g_2, r_2\} \cup E_2, ack_1)$. Mit $B_1 = Z_2 \cup \{g_2, r_2\} \cup E_2$ ist die Eigenschaft $\mathbf{p\text{-}i}(ack_2, B_1, ack_1)$ gezeigt. Ebenso folgt aus $\#Z_2 = 0$ (Knoten 3'), $\neg E_2$ (Knoten 4') und der Kenntnis, daß $g_2$ und $r_2$ Schnittstellen sind, $\#B_1 = 0$.

Die Elimination der Knoten 5', 6', 10' und 11' folgt mit symmetrischen Argumenten.

Die partiellen S-Invarianten der G-Knoten 2 und 1' können zu der partiellen S-Invariante $\mathbf{p\text{-}i}(h_1, Y_1 \cup X_2 \cup \{ack_1, rq_1\}, g_1)$ kombiniert werden. Die nötigen Disjunktheitsaussagen für $X_2$ und $Y_1$ gelten, da die internen Stellen von L-Mutex $Y_1$ mit den internen Stellen von R-Mutex $X_2$ disjunkt sind. Ebenso folgt $\#Y_1 \cup X_2 \cup \{ack_1, rq_1\} = 0$: $\#Y_1 = 0$ und $\#X_2 = 0$ folgt aus den Knoteninschriften und $ack_1$ bzw. $rq_1$ sind als Schnittstellen der Teilmodule initial unmarkiert. Damit dürfen wir die Inschrift von Knoten 2 zu $\mathbf{p\text{-}i}(h_1, Y_1 \cup X_2 \cup \{ack_1, rq_1\}, g_1)$ und $\#Y_1 \cup X_2 \cup \{ack_1, rq_1\} = 0$ abschwächen. Den G-Knoten 1' eliminieren wir anschließend.

Knoten 2' modifizieren wir wieder symmetrisch zu Knoten 2; anschließend können wir auch Knoten 1 eliminieren.

Zuletzt können wir die Aussage von G-Knoten 7 abschwächen zu $\Box[\neg E_1 \vee \neg E_2]$, da $B_1$ nach Definition $E_2$ umfaßt. Den G-Knoten 7' lassen wir ganz weg.

Insgesamt erhalten wir so den RG-Graphen aus Abb. 8.7. Da in den Knoteninschriften die Platzhalter $A_1$, $A_2$, $B_1$, $B_2$, $Z_1$ und $Z_2$ nicht mehr auftauchen, können wir die entsprechenden "Deklarationen" ebenfalls weglassen. Dieser RG-Graph ist dem RG-Graph des Mutex-Moduls schon sehr ähnlich. Wenn wir für die Menge $Y_1 \cup X_2 \cup \{ack_1, rq_1\}$ von internen Stellen und symmetrisch dazu für die Menge $Y_2 \cup X_1 \cup \{ack_2, rq_2\}$ zwei neue Platzhalter $W_1$ bzw. $W_2$ einführen und die Knoten 12. und 12' zu einem Knoten zusammenfassen, erhalten wir den RG-Graphen aus Abb. 8.8. Dieser ist bis auf Umbenennung der Platzhalter mit dem RG-Graphen des Mutex-Moduls identisch.

Insgesamt ist damit gezeigt, daß die Komposition einer Implementierung von L-Mutex und einer Implementierung von R-Mutex das Mutex-Modul implementiert. Wir können jetzt also den linken und rechten Teil unabhängig voneinander weiterentwickeln bzw. implementieren.

[2] Die notwendigen Disjunktheitsannahmen folgen aus der Schnittstellen.

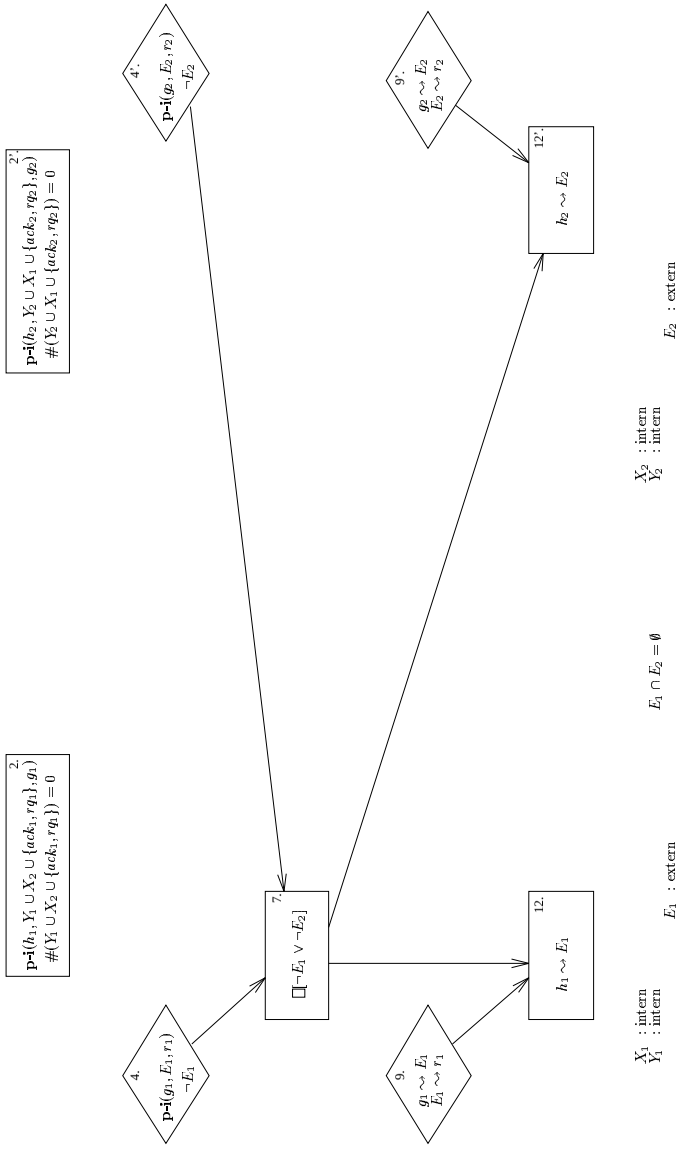



Abbildung 8.7: Vereinfachung des kombinierten RG-Graphen

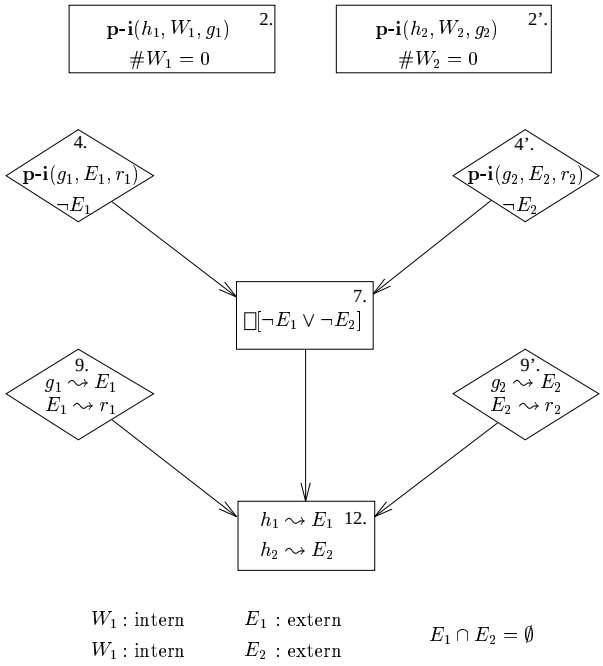



Abbildung 8.8: Vereinfachter kombinierter RG-Graph

## 8.4 Implementierung der Teilmodule

Die Zerlegung der verteilten Implementierung von Mutex aus Abb. 8.2 zeigt bereits die Implementierungen der Teilmodule L-Mutex und R-Mutex. Wir weisen nun formal nach, daß die linke Komponente $\Sigma_l$ das Modul L-Mutex implementiert. Die Komponente $\Sigma_l$ ist in Abb. 8.9 nochmals dargestellt. Der Nachweis für die rechte Komponente geht analog und wird deshalb hier nicht ausgeführt.

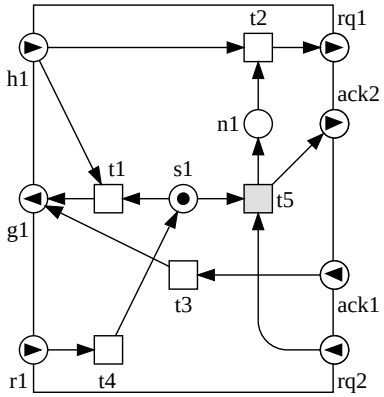



Abbildung 8.9: Linke Teilkomponente $\Sigma_l$

Den RG-Graphen von R-Mutex haben wir nochmals in Abb. 8.10 dargestellt. Zunächst geben wir für die Platzhalter für interne Stellenmengen die konkreten Belegungen in der Implementierung $\Sigma_l$ an: $X_1 \mathrel{\widehat{=}} \emptyset$, $Y_1 \mathrel{\widehat{=}} \emptyset$ und $Z_1 \mathrel{\widehat{=}} \{s_1\}$. Wir zeigen nun, daß $\Sigma_l$ den RG-Graphen von L-Mutex erfüllt, wenn wir die Platzhalter entsprechend substituieren. Der formale Beweis ist in den Tabellen 8.1 bis 8.5 ausgeführt. Im Text führen wir wieder nur den informellen Beweis mit Referenzen auf die formalen Argumente.

Die Systemkomponente $\Sigma_l$ erfüllt offensichtlich die Eigenschaften der G-Knoten 1, 2 und 3 des RG-Graphen. Sie folgen unmittelbar aus der Struktur des Netzes und der Anfangsmarkierung (vgl. Tabelle 8.1).
Zum Beweis der Eigenschaft $\Box[\neg E_1 \vee \neg B_1]$ des G-Knotens 7 (vgl. Tabelle 8.2) kombinieren wir die partiellen S-Invarianten **p-i**$(g_1, E_1, r_1)$ und **p-i**$(ack_2, B_1, ack_1)$, die von der Umgebung

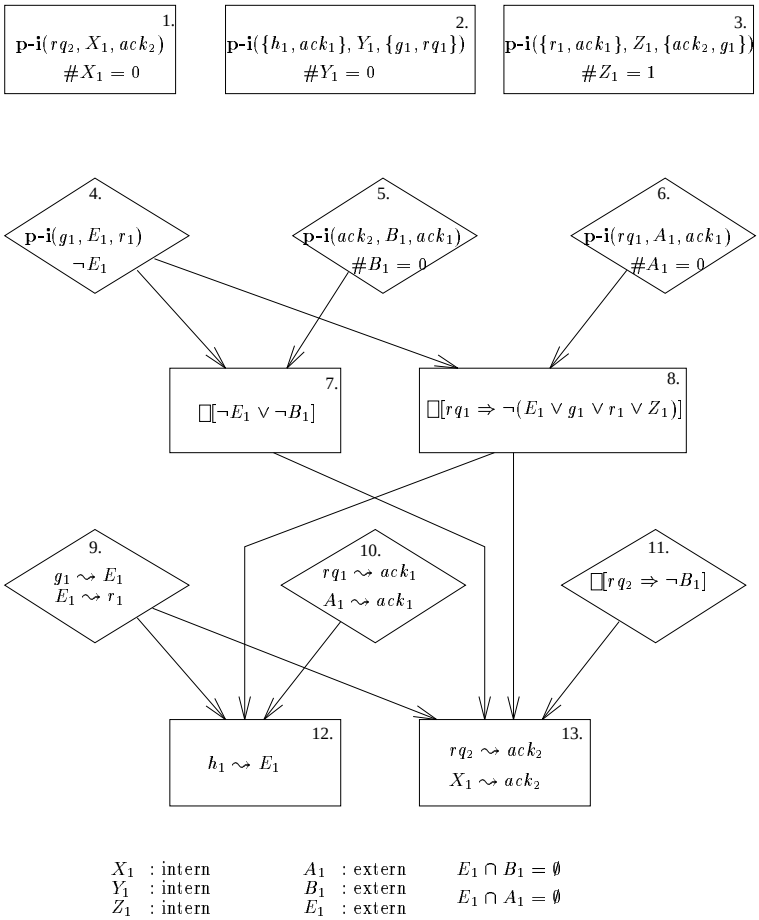



Abbildung 8.10: RG-Graph von L-Mutex

| | | |
|---|---|---|
| (1) | $\Sigma \models$ **p-i**$(rq_2, \emptyset, ack_2)$ | (P-INV) mit $T_1 = T_\Sigma$, $T_2 = \emptyset$ |
| (2) | $\Sigma \models \#\emptyset = 0$ | (TAUT) |
| (3) | $\Sigma \models$ **p-i**$(\{h_1, ack_1\}, \emptyset, \{g_1, rq_1\})$ | (P-INV) mit $T_1 = T_\Sigma$, $T_2 = \emptyset$ |
| (4) | $\Sigma \models$ **p-i**$(\{r_1, ack_1\}, s_1, \{ack_2, g_1\})$ | (P-INV) mit $T_1 = T_\Sigma$, $T_2 = \emptyset$ |
| (5) | $\Sigma \models \#s_1 = 1$ | (INIT) mit $M_\Sigma(s_1) = 1$ |

Tabelle 8.1: Formaler Beweis für die G-Knoten 1–3

gewährleistet werden (R-Knoten 4 und 5), mit der partiellen S-Invariante $\mathbf{p\text{-}i}(\{r_1, ack_1\}, s_1, \{ack_2, g_1\})$ (4) zur S-Invariante $\mathbf{inv}(\{ack_1, ack_2, s_1, r_1, g_1\} \cup E_1 \cup B_1)$ (7). Initial ist genau eine dieser Stellen markiert (10)–(12). Damit erhalten wir $\Box[\neg E_1 \vee \neg B_1]$ (14).

$\mathcal{E}_7 = \{\mathbf{p\text{-}i}(g_1, E_1, r_1), \neg E_1, \mathbf{p\text{-}i}(ack_2, B_1, ack_1), \#B_1 = 0\}$

| | | |
|---|---|---|
| (6) | $\Sigma, \mathcal{E}_7 \models \mathbf{p\text{-}i}(g_1, E_1, r_1)$ | (ASS) |
| (7) | $\Sigma, \mathcal{E}_7 \models \mathbf{p\text{-}i}(ack_1, E_1 \cup \{s_1, r_1, g_1\}, ack_2)$ | ($\cup$-P-INV) mit (4) und (6) |
| (8) | $\Sigma, \mathcal{E}_7 \models \mathbf{p\text{-}i}(ack_2, B_1, ack_1)$ | (ASS) |
| (9) | $\Sigma, \mathcal{E}_7 \models \mathbf{inv}(\{ack_1, ack_2, s_1, r_1, g_1\} \cup E_1 \cup B_1)$ | ($\cup$-P-INV) mit (7) und (8); (S-INV) |
| (10) | $\Sigma, \mathcal{E}_7 \models \#E_1 = 0$ | (ASS) und (EQUIV) |
| (11) | $\Sigma, \mathcal{E}_7 \models \#B_1 = 0$ | (ASS) |
| (12) | $\Sigma, \mathcal{E}_7 \models \#\{ack_1, ack_2, s_1, r_1, g_1\} = 1$ | (INIT) |
| (13) | $\Sigma, \mathcal{E}_7 \models (\#E_1 \cup B_1 \cup \{ack_1, ack_2, s_1, r_1, g_1\}) = 1$ | (CONJ) (10)–(12) und Abschwächung mit (MP) |
| (14) | $\Sigma, \mathcal{E}_7 \models \Box[\neg E_1 \vee \neg B_1]$ | ($\leq$ 1-INV) mit (9) und (13); (EQUIV) |

Tabelle 8.2: Formaler Beweis für den G-Knoten 7

Zum Beweis der Eigenschaft $\Box[rq_1 \Rightarrow \neg(E_1 \vee g_1 \vee r_1 \vee s_1)]$ des G-Knotens 8 (vgl. Tabelle 8.3) kombinieren wir die beiden partiellen S-Invarianten $\mathbf{p\text{-}i}(g_1, E_1, r_1)$ und $\mathbf{p\text{-}i}(rq_1, A_1, ack_1)$, die von der Umgebung gewährleistet werden (R-Knoten 4 und 6), mit der partiellen S-Invariante $\mathbf{p\text{-}i}(\{r_1, ack_1\}, \{s_1, n_1\}, \{rq_1, g_1\})$ (15) zur S-Invariante $\mathbf{inv}(E_1 \cup A_1 \cup \{g_1, r_1, s_1, n_1, rq_1, ack_1\})$ (19). Initial ist genau eine der Stellen dieser Invariante markiert. Damit folgt $\Box[rq_1 \Rightarrow \neg(E_1 \vee g_1 \vee r_1 \vee s_1)]$ (24).
Der formale Beweis für G-Knoten 12 ist in Tabelle 8.4 ausgeführt. Dazu ist die Eigenschaft $h_1 \rightsquigarrow E_1$ zu zeigen. Aufgrund der bereits bewiesenen S-Invariante wissen wir (26)

$$\Box[h_1 \Rightarrow (E_1 \vee r_1 \vee (s_1 \vee n_1 \vee g_1) \vee ack_1 \vee A_1 \vee rq_1)]$$

Diese Situation ist die Grundlage für den Leadsto-Graphen aus Abb. 8.11. Zusammen mit den Eigenschaften der Umgebung und einigen einfachen Leadsto-Eigenschaften der Systemkomponente (27)-(31) werden alle Zustände schrittweise in den Zustand $(s_1 \vee n_1 \vee g_1)$

$\mathcal{E}_8 = \{\mathbf{p\text{-}i}(g_1, E_1, r_1), \neg E_1, \mathbf{p\text{-}i}(rq_1, A_1, ack_1), \#A_1 = 0\}$

| | | |
|---|---|---|
| (15) | $\Sigma, \mathcal{E}_8 \models \mathbf{p\text{-}i}(\{r_1, ack_1\}, \{s_1, n_1\}, \{rq_1, g_1\})$ | (P-INV) mit $T_1 = T_\Sigma$ und $T_2 = \emptyset$ |
| (16) | $\Sigma, \mathcal{E}_8 \models \mathbf{p\text{-}i}(rq_1, A_1, ack_1)$ | (ASS) |
| (17) | $\Sigma, \mathcal{E}_8 \models \mathbf{p\text{-}i}(r_1, \{s_1, n_1\} \cup A_1 \cup \{rq_1, ack_1\}, g_1)$ | ($\cup$-P-INV) |
| (18) | $\Sigma, \mathcal{E}_8 \models \mathbf{p\text{-}i}(g_1, E_1, r_1)$ | (ASS) |
| (19) | $\Sigma, \mathcal{E}_8 \models \mathbf{inv}(E_1 \cup A_1 \cup \{g_1, r_1, s_1, n_1, rq_1, ack_1\})$ | ($\cup$-P-INV) mit (17) und (18); (S-INV) |
| (20) | $\Sigma, \mathcal{E}_8 \models \#A_1 = 0$ | (ASS) |
| (21) | $\Sigma, \mathcal{E}_8 \models \#E_1 = 0$ | (ASS) |
| (22) | $\Sigma, \mathcal{E}_8 \models \#\{g_1, r_1, s_1, n_1, rq_1, ack_1\} = 1$ | (INIT) |
| (23) | $\Sigma, \mathcal{E}_8 \models \#A_1 \cup E_1 \cup \{g_1, r_1, s_1, n_1, rq_1, ack_1\} = 1$ | (CONJ) (20)–(22) und Abschwächung mit (MP) |
| (24) | $\Sigma, \mathcal{E}_8 \models \Box[rq_1 \Rightarrow \neg(E_1 \vee g_1 \vee r_1 \vee s_1)]$ | ($\leq$ 1-INV) mit (19) und (23) |

Tabelle 8.3: Formaler Beweis für den G-Knoten 8

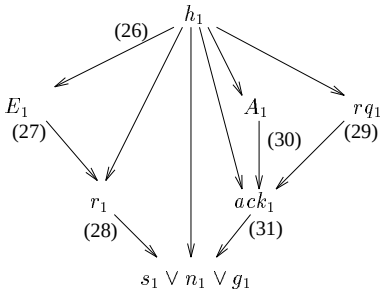



Abbildung 8.11: Leadsto-Graph für $h_1 \rightsquigarrow (s_1 \vee n_1 \vee g_1)$

überführt. Damit gilt $h_1 \rightsquigarrow (s_1 \vee n_1 \vee g_1)$ (32). Da $h_1$ gültig bleibt, solange $s_1 \vee n_1 \vee g_1$ nicht gilt (33), folgt $h_1 \rightsquigarrow (h_1 \wedge s_1 \vee h_1 \wedge n_1 \vee g_1)$ (35). Wir zeigen nun mit Hilfe des Leadsto-Graphen aus Abb 8.12, daß $h_1 \wedge s_1$, $h_1 \wedge n_1$ und $g_1$ immer zum Zustand $E_1$ führen: Der Zustand

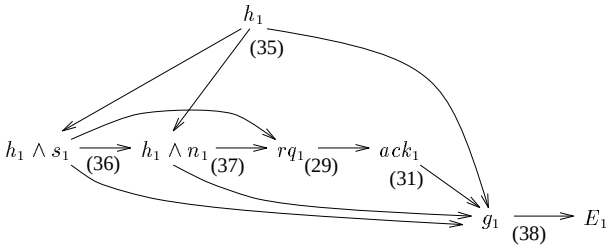



Abbildung 8.12: Leadsto-Graph für $h_1 \rightsquigarrow E_1$

$h_1 \wedge s_1$ wird entweder in $g_1$ oder $h_1 \wedge n_1$ oder $rq_1$ überführt (36); $h_1 \wedge n_1$ wird entweder auch in $g_1$ oder $rq_1$ überführt (37); $rq_1$ führt aufgrund der Umgebungsannahme zu $ack_1$ (29) und $ack_1$ führt unmittelbar zu $g_1$ (31). $g_1$ wird von der Umgebung in $E_1$ überführt (38). Zusammen haben wir damit $h_1 \rightsquigarrow E_1$ gezeigt (39).
Der formale Nachweis für den G-Knoten 13 ist in Tabelle 8.5 durchgeführt. Aus einer S-Invariante beweisen wir zunächst $\Box[\neg B_1 \Rightarrow (E_1 \vee r_1 \vee ack_1 \vee s_1 \vee g_1 \vee ack_2)]$ (41). Zusammen mit der Annahme $\Box[rq_2 \Rightarrow \neg B_1]$ folgt damit $\Box[rq_2 \Rightarrow (ack_1 \vee g_1 \vee E_1 \vee r_1 \vee s_1 \vee g_1 \vee ack_2)]$ (42). Diese Implikation ist der Ausgangspunkt für den Leadsto-Graphen in Abb. 8.13. Alle Zustände werden schrittweise in $s_1$ oder $ack_2$ überführt

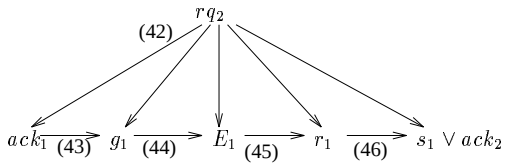



Abbildung 8.13: Leadsto-Graph für $rq_2 \rightsquigarrow s_1 \vee ack_2$

(43)–(46). Damit ist die Eigenschaft $rq_2 \rightsquigarrow s_1 \vee ack_2$ gezeigt (47). Wenn $rq_2$ gültig ist, wird irgendwann $ack_2$ gültig (48) oder $rq_2$ bleibt für

$\mathcal{E}_{12} = \mathcal{E}_8 \cup \{g_1 \rightsquigarrow E_1, E_1 \rightsquigarrow r_1, rq_1 \rightsquigarrow ack_1, A_1 \rightsquigarrow ack_1\}$

| | | |
|---|---|---|
| (25) | $\Sigma, \mathcal{E}_{12} \models \Box[E_1 \vee r_1 \vee s_1 \vee n_1 \vee g_1 \vee ack_1 \vee A_1 \vee rq_1]$ | ($\geq$ 1-INV) mit (19), (23) und $\mathcal{E}_{12} \supseteq \mathcal{E}_8$ |
| (26) | $\Sigma, \mathcal{E}_{12} \models \Box[h_1 \Rightarrow (E_1 \vee r_1 \vee (s_1 \vee n_1 \vee g_1) \vee ack_1 \vee A_1 \vee rq_1)]$ | (W-INV) mit (25); (TAUT-INV) |
| (27) | $\Sigma, \mathcal{E}_{12} \models E_1 \rightsquigarrow r_1$ | (ASS) |
| (28) | $\Sigma, \mathcal{E}_{12} \models r_1 \rightsquigarrow s_1$ | (PROGR1) mit $t_0 = t_4$ |
| (29) | $\Sigma, \mathcal{E}_{12} \models rq_1 \rightsquigarrow ack_1$ | (ASS) |
| (30) | $\Sigma, \mathcal{E}_{12} \models A_1 \rightsquigarrow ack_1$ | (ASS) |
| (31) | $\Sigma, \mathcal{E}_{12} \models ack_1 \rightsquigarrow g_1$ | (PROGR1) mit $t_0 = t_3$ |
| (32) | $\Sigma, \mathcal{E}_{12} \models h_1 \rightsquigarrow (s_1 \vee n_1 \vee g_1)$ | (LTO-GRAPH) mit Abb. 8.11 und (26)–(31) |
| (33) | $\Sigma, \mathcal{E}_{12} \models h_1 \wedge \neg(s_1 \vee n_1 \vee g_1)$ **alw** $h_1$ | (ALW) |
| (34) | $\Sigma, \mathcal{E}_{12} \models h_1 \rightsquigarrow h_1 \wedge (s_1 \vee n_1 \vee g_1)$ | (SYNC) mit (32), (33) |
| (35) | $\Sigma, \mathcal{E}_{12} \models h_1 \rightsquigarrow h_1 \wedge s_1 \ \vee \ h_1 \wedge n_1 \ \vee \ g_1$ | (W-$\rightsquigarrow$) mit (34) |
| (36) | $\Sigma, \mathcal{E}_{12} \models h_1 \wedge s_1 \rightsquigarrow g_1 \ \vee \ h_1 \wedge n_1 \ \vee \ rq_1$ | (PROGR1) mit $t_0 = t_1$ |
| (37) | $\Sigma, \mathcal{E}_{12} \models h_1 \wedge n_1 \rightsquigarrow rq_1 \vee g_1$ | (PROGR1) mit $t_0 = t_2$ |
| (38) | $\Sigma, \mathcal{E}_{12} \models g_1 \rightsquigarrow E_1$ | (ASS) |
| (39) | $\Sigma, \mathcal{E}_{12} \models h_1 \rightsquigarrow E_1$ | (LTO-GRAPH) mit Abb. 8.12 und (29), (31), (35)–(38) |

Tabelle 8.4: Formaler Beweis für den G-Knoten 9

immer gültig. Dann wird aber $s_1$ immer wieder gültig und $t_5$ muß aufgrund der Fairnessannahme schalten; damit wird $ack_2$ ebenfalls gültig (49). Zuletzt bleibt $X_1 \leadsto ack_2$ zu zeigen. Da mit $X_1 = \emptyset$ die Aussage $X_1$ gleichbedeutend mit $\bot$ ist, ist dies trivial (51).

| | | |
|---|---|---|
| $\mathcal{E}_{13} = \mathcal{E}_8 \cup \{\Box[rq_2 \Rightarrow \neg B_1]\}$ | | |
| (40) | $\Sigma, \mathcal{E}_{13} \models \Box[rq_2 \Rightarrow \neg B_1]$ | (ASS) |
| (41) | $\Sigma, \mathcal{E}_{13} \models \Box[\neg B_1 \Rightarrow (E_1 \vee r_1 \vee ack_1 \vee s_1 \vee g_1 \vee ack_2)]$ | ($\leq$ 1-INV) mit (9), (13) |
| (42) | $\Sigma, \mathcal{E}_{13} \models \Box[rq_2 \Rightarrow (E_1 \vee r_1 \vee ack_1 \vee s_1 \vee g_1 \vee ack_2)]$ | (W-INV) auf (41) |
| (43) | $\Sigma, \mathcal{E}_{13} \models ack_1 \leadsto g_1$ | (PROGR1) mit $t_0 = t_3$ |
| (44) | $\Sigma, \mathcal{E}_{13} \models g_1 \leadsto E_1$ | (ASS) |
| (45) | $\Sigma, \mathcal{E}_{13} \models E_1 \leadsto r_1$ | (ASS) |
| (46) | $\Sigma, \mathcal{E}_{13} \models r_1 \leadsto s_1 \vee ack_2$ | (PROGR1) mit $t_0 = t_4$ |
| (47) | $\Sigma, \mathcal{E}_{13} \models rq_2 \leadsto s_1 \vee ack_2$ | (LTO-GRAPH) mit Abb. 8.13 und (42)–(46) |
| (48) | $\Sigma, \mathcal{E}_{13} \models rq_2$ **alw** $ack_2$ | (ALW) |
| (49) | $\Sigma, \mathcal{E}_{13} \models rq_2 \leadsto ack_2$ | (FAIR) mit $t_0 = t_5$, $s = s_1$ und (47), (48) |
| (50) | $\Sigma, \mathcal{E}_{13} \models \Box[\emptyset \Rightarrow ack_2]$ | (TAUT-INV) |
| (51) | $\Sigma, \mathcal{E}_{13} \models \Box\emptyset \leadsto ack_2$ | (INV-$\leadsto$) mit (50) |
| (52) | $\Sigma \models$ L-Mutex | (1), (2), (3), (4), (5), (14), (24), (39), (49) und (51) |

Tabelle 8.5: Formaler Beweis für den G-Knoten 13

Insgesamt haben wir damit gezeigt, daß die linke Systemkomponente $\Sigma_l$ das Modul L-Mutex implementiert. Der Beweis für die rechte Systemkomponente und R-Mutex ist im wesentlichen symmetrisch. Zusammen haben wir damit gezeigt, daß die Komposition der beiden Systemkomponenten, wie wir sie am Anfang dieses Kapitels gesehen haben, das Mutex-Modul implementiert.

## 8.5 Zusammenfassung

Wir haben nun an einem Beispiel gesehen, wie man ein Modul in Teilmodule zerlegen kann. Die Teilmodule können dann separat implementiert werden. Im Beispiel waren die Teilmodule im wesentlichen symmetrisch und wir konnten uns daher auf die Implementierung eines Teilmoduls beschränken.

Meist liegt einer Zerlegung eines Moduls eine bestimmte Implementierungsidee zugrunde. Im Beispiel haben wir bei der Aufteilung sogar schon eine ganz spezielle Implementierung im Kopf gehabt. Die Aufteilung in die Teilmodule beschreibt also auf abstrakter Ebene eine algorithmische Idee, die auf viele verschiedene Weisen konkretisiert werden kann. Diese Beschreibung ist allgemeiner als die konkrete Implementierung, da neben der vorgestellten Implementierung der Teilmodule noch andere existieren. Diese Beschreibung ist aber spezieller als das ursprüngliche Mutex-Modul, da bestimmte Implementierungen (z.B. die Semaphorlösung aus dem vorangegangenen Kapitel) ausgeschlossen werden.
Durch die Wahl unseres Schnittstellenbegriffes (Nachrichtenaustausch) werden bei der Aufteilung in Teilmodule Anforderungen an die Verteiltheit einer Lösung aufgestellt. Ohne den Begriff der Verteiltheit formal zu fassen, ist die hier vorgestellte Lösung „verteilter“ als die Semaphorlösung. Die unverteilte Semaphorlösung wird durch die Aufteilung in Teilmodule ausgeschlossen.
Wie das Beispiel zeigt, kann die Beschreibung der Teilmodule komplizierter werden als die des Ausgangsmoduls. Dies hängt damit zusammen, daß die Teilmodule eine kompliziertere Funktionalität besitzen als das Ausgangsmodul. In den RG-Graphen der Teilmodule muß die algorithmische Idee der hier vorgestellten Mutex-Implementierung präzise beschrieben werden. Sonst ist es nicht möglich, die beiden Teilmodule unabhängig voneinander zu implementieren.

# Kapitel 9

# Zusammenfassung

In dieser Arbeit führen wir einen Formalismus zum modularen Entwurf verteilter Systeme ein. Um die Verteiltheit eines Systems auch in einem Ablauf zu repräsentieren, benutzen wir Petrinetze (S/T-Systeme) und ihre halbgeordneten Abläufe (Prozesse) als mathematisches Modell für Systeme und ihre *verteilten Abläufe*. Wir untersuchen die Struktur der verteilten Abläufe und vergleichen sie mit der Struktur der sequentiellen Abläufe. Im Hinblick auf den modularen Entwurf betrachten wir *Systemkomponenten*, die zu einem Gesamtsystem zusammengesetzt werden können. Das Verhalten definieren wir so, daß sich das Verhalten des Gesamtsystems aus dem Verhalten der einzelnen Komponenten ergibt. Die verteilten Abläufe eignen sich dabei gut für eine derartige *kompositionale* Verhaltensbeschreibung.

In Kapitel 3 betrachten wir verteilte Abläufe im wesentlichen losgelöst von Systemen. Wir betrachten die klassischen sequentiellen Abläufe und unsere verteilten Abläufe in einem einheitlichen mathematischen Rahmen. So ist es möglich Konzepte und Ergebnisse für sequentielle Abläufe kanonisch auf den verteilten Fall zu übertragen. Wir zeigen, daß die Präfixrelation auf der Menge der verteilten Abläufe — wie im sequentiellen Fall — ein Scott-Bereich ist. Die verteilten Abläufe (zusammen mit der Präfixordnung) haben somit eine wohlverstandene und elegante mathematische Struktur, die auch im sequentiellen Fall eine große Rolle spielt. Insbesondere lassen sich damit die Konzepte der *Sicherheits-* und *Lebendigkeitseigenschaft* kanonisch auf unser Ablaufmodell übertragen.
In vielen Entwurfsmethoden der Literatur, die für sequentielle Abläufe

entwickelt wurden, spielen Sicherheitseigenschaften eine zentrale Rolle. Einige dieser Entwurfsmethoden nutzen eine Eigenschaft von Sicherheitseigenschaften aus, die im verteilten Fall für Sicherheitseigenschaften nicht mehr gilt. Wir haben für verteilte Abläufe die Teilklasse der Sicherheitseigenschaften charakterisiert, in denen die Voraussetzung dieser Entwurfsmethoden gilt. Dies begründet die besondere Bedeutung dieser Teilklasse.

Kapitel 4 und 5 beschäftigen sich mit dem Aspekt des modularen Entwurfs. In Kapitel 4 wird ein Modell für *Systemkomponenten* eingeführt. Wir erweitern Petrinetze um Lebendigkeitsannahmen für bestimmte Transitionen und um Schnittstellen, über die eine Systemkomponente mit ihrer Umgebung kommunizieren kann. So kann ein großes System aus einzelnen Systemkomponenten zusammengesetzt werden.
Einer Systemkomponente ordnen wir eine Menge von verteilten Abläufen als *Verhalten* zu. Im Verhalten beschreiben wir sowohl das Verhalten der Systemkomponente selbst, als auch das mögliche Verhalten der Umgebung. Damit erreichen wir, daß das Verhalten bezüglich des Zusammensetzens von Systemkomponenten *kompositional* ist: Das Verhalten eines aus zwei Teilkomponenten zusammengesetzten Systems ist der Durchschnitt des Verhaltens der Teilkomponenten. Damit entspricht die Komposition von zwei Systemkomponenten der logischen Konjunktion ihrer Eigenschaften.
In Kapitel 5 werden Module eingeführt. Ein Modul spezifiziert das Verhalten einer Systemkomponenten in Abhängigkeit vom Verhalten der Umgebung (Rely-Guarantee-Spezifikation). Dabei wird die Abhängigkeit zwischen dem Verhalten der Umgebung und der Systemkomponente durch eine Abhängigkeitsrelation im *RG-Graphen* explizit dargestellt. Deshalb ist es im Gegensatz zu einigen anderen Ansätzen für Rely-Guarantee-Spezifikationen möglich, nicht nur Sicherheitseigenschaften, sondern auch Lebendigkeitseigenschaften von der Umgebung vorauszusetzen. Zum „Rechnen“ mit Modulen wurden einfache Regeln angegeben. Insbesondere gibt es eine Regel um Module in Teilmodule aufzuspalten, die dann unabhängig voneinander implementiert werden können (vgl. Beispiel in Kapitel 8).

Das Modulkonzept ist nicht auf einen bestimmten Formalismus zur Repräsentation von Eigenschaften festgelegt. Es kann deshalb mit vielen Formalismen kombiniert werden. Ein möglicher Formalismus zur syntaktische Repräsentation von Eigenschaften ist die *temporale Logik*, die in Kapitel 6 vorgestellt wird und für die in Kapitel 7 Beweisregeln an-

gegeben werden. Als besonderes Merkmal wurden S-Invarianten in die Logik integriert. S-Invarianten sind in Kombination mit den partiellen S-Invarianten von zentraler Bedeutung, weil sich so relativ einfach Invarianten beweisen lassen, die nur im Zusammenspiel von Systemkomponente und ihrer Umgebung gelten. Dies wird in dem Beispiel aus Kapitel 8 deutlich.

Für die praktische Durchführung von größeren Fallstudien ist die Unterstützung der Verifikation durch einen Rechner unerläßlich. Dabei kann zum Überprüfen der Zusicherungen (und von anderen zustandslogischen Tautologien) ein Theorembeweiser eingesetzt werden. Die korrekte Anwendung der Beweisregeln kann im wesentlichen syntaktisch überprüft werden. Außerdem ist ein Werkzeug zum Rechnen mit Modulen wünschenswert, das die korrekte Anwendung der Kombinations- und Eliminationsregeln für RG-Graphen überprüft.

Wir haben in dieser Arbeit nur Netze mit nicht unterscheidbaren schwarzen Marken betrachtet. Damit können Daten nur sehr unübersichtlich modelliert werden. Zur Integration von Datentypen in Petrinetze wurden in der Literatur Netze mit „strukturierten Marken“ vorgeschlagen (u.a. [79, 41]). Es ist einfach, Systemkomponenten (samt ihres kompositionalen Verhaltens) für algebraische Netze [79] zu erweitern. Damit dies möglich ist, haben wir darauf geachtet, daß die Unendlichkeit von Datentypen keine Probleme aufwirft. Deshalb lassen wir unendliche Anfangsmarkierung von Systemen zu und auch die Menge der Ein- und Ausgabestellen einer Systemkomponente darf unendlich sein. Die Abzählbarkeit, die wir für Systeme fordern, ist ebenfalls keine Einschränkung, weil auch die üblichen Datentypen abzählbar sind.
Dagegen ist es nicht so einfach eine geeignete syntaktische Repräsentation von Eigenschaften für diese Netzklassen anzugeben. Dies betrifft nicht die temporalen Operatoren, sondern die Zustandslogik, auf der die temporale Logik aufbaut. Alle bisher durchgeführten Fallstudien führen mehr oder weniger zu ad-hoc-Repräsentationen von Eigenschaften. Der wesentliche Kern der Formulierung von zustandslogischen Aussagen hat sich bisher noch nicht herausgebildet. Wenn sich ein solcher Formalismus herausbildet, ist es einfach diesen Formalismus mit unserem Modulkonzept zu kombinieren. Dies ist ein weiterer Grund, warum wir das Modulkonzept unabhängig von der Logik eingeführt haben.
Die Frage nach einer geeigneten Zustandslogik für algebraische Netze scheint im Hinblick auf die Schwerpunkte dieser Arbeit — Modularität

und Verteiltheit — eher nebensächlich zu sein. Für die die praktische Anwendbarkeit hat sie aber eine große Bedeutung.

# Literaturverzeichnis


[1] Abadi, Martín, und Leslie Lamport: „Composing Specifications“. In Bakker, J.W. de, und W.-P. de Roever (Eds.): *Stepwise Refinement of Distributed Systems. Models, Formalisms, Correctness.* Springer-Verlag: 1990.

[2] Abadi, Martín, und Leslie Lamport: *Conjoining Specifications.* Research Report 118. Digital Equipment Corporation, System Research Center. Dezember 1993.

[3] Abadi, Martín, und Leslie Lamport: „The Existence of Refinement Mappings“. *Theoretical Computer Science* **82** (1991) 253–284.

[4] Abadi, Martín, und Gordon D. Plotkin: „A logical view of composition“. *Theoretical Computer Science* **114** (1993) 3–30.

[5] Alford, M.W., J.P. Ansart, G. Hommel, L. Lamport, B. Liskov, G.P. Mullery und F.B. Schneider: *Distributed Systems: Methods and Tools for Specification.* LNCS **190**. Springer-Verlag: 1985.

[6] Alpern, Bowen, und Fred B. Schneider: „Defining Liveness“. *Information Processing Letters* **21** (Oktober 1985) 181–185.

[7] Alpern, Bowen, und Fred. B. Schneider: „Recognizing Safety and Liveness“. *Distributed Computing* **2** (1987) 117–126.

[8] Apt, Krysztof R., Nissim Francez und Shmuel Katz: „Appraising Fairness in Languages for Distributed Programming“. *Distributed Computing* **2** (1988) 226–241.

[9] Barringer, Howard, Ruurd Kuiper und Amir Pnueli: „Now You May Compose Temporal Logic Specifications“. In: $16^{th}$ *annual ACM Symposium on Theory of Computing.* ACM. April 1984: 51–63.

[10] Baumgarten, Bernd: „On Internal and External Characterizations of PT-Net Building Block Behaviour“. In Rozenberg, G. (Ed.): *Advances in Petri Nets 1988.* LNCS **340**. Springer-Verlag: 1988: 44–61.

[11] Best, E., und R. Devillers: „Sequential and Concurrent Behaviour in Petri Net Theory“. *Theoretical Computer Science* **55** (1987) 87–136.

[12] Best, E., R. Devillers und F.G. Hall: „The box calculus: A new causal algebra with multi-label communication“. In Rozenberg, G. (Ed.): *Advances in Petri nets 1992*. LNCS **609**. Springer-Verlag: 1992: 21–69.

[13] Best, Eike: *Semantik*. Vieweg-Verlag: 1995.

[14] Best, Eike, und César Fernández: *Nonsequential Processes*. EATCS Monographs on Theoretical Computer Science **13**. Springer-Verlag: 1988.

[15] Bradfield, J. C.: „Proving Temporal Properties of Petri Nets“. In Rozenberg, G. (Ed.): *Advances in Petri Nets 1991*. LNCS **524**. Springer-Verlag: 1991: 29–47.

[16] Brauer, Wilfried, Robert Gold und Walter Vogler: „A Survey of Behaviour and Equivalence Preserving Refinements of Petri Nets“. In Rozenberg, G. (Ed.): *Advances in Petri Nets*. LNCS **483**. Springer-Verlag: 1990: 1–46.

[17] Broy, M., F. Dederichs, C. Dendorfer, M. Fuchs, T.F. Gritzner und R. Weber: *The Design of Distributed Systems — An Introduction to* FOCUS. SFB-Bericht 342/2/92 A. Technische Universität München. Januar 1992.

[18] Chandy, K. M., und J. Misra: *Parallel Program Design: A Foundation*. Addison-Wesley: 1988.

[19] Chang, Edward, Zohar Manna und Amir Pnueli: „The Safety-Progress Classification“. In Bauer, F.L., W. Brauer und H. Schwichtenberg (Eds.): *Logic and Algebra of Specifications*. NATO ASI Series F: Computer and Systems Sciences **94**. Springer-Verlag: 1993: 143–202.

[20] Christensen, Søren, und Niels D. Hansen: „Coloured Petri Nets Extended with Channels for Synchronous Communication“. In Valette, R. (Ed.): *Application and Theory of Petri Nets 1994*. Springer-Verlag: 1994: 159–178.

[21] Roever Jr., Willem P. de: „The Quest for Compositionality“. In Neuhold E.J. (Ed.): *IFIP working group on the role of abstract models in Computer Science*. North-Holland: 1985.

[22] Dederichs, Frank: *System and Environment: The Philosophers revisited*. TU-Bericht TUM-I9040. Technische Universität München. Oktober 1990.

[23] Dederichs, Frank, und Rainer Weber: „Safety and Liveness From a Methodological Point of View“. *Information Processing Letters* **36** (Oktober 1990) 25–30.

[24] Degano, P., J. Meseguer und U. Montanari: „Axiomatizing Net Computations and Processes“. In: $4^{th}$ *Annual Symposium on Logic in Computer Science*. 1989.

[25] Ebbinghaus, Heinz-Dieter, Jörg Flum und Wolfgang Thomas: *Einführung in die mathematische Logik*. 2 Aufl. Wissenschaftliche Buchgesellschaft: 1986.

[26] Engelfriet, Joost: „Branching processes of Petri nets". *Acta Informatica* **28** (1991) 575–591.

[27] Esparza, Javier: „Model checking using net unfoldings". *Science of Computer Programming* **23** (1994) 151–195.

[28] Floyd, Robert W.: „Assigning Meanings to Programs". In Schwartz, J.T. (Ed.): *Mathematical Aspects of Computer Science*. American Mathematical Society. April 1966: 19–32.

[29] Francez, Nissim: *Fairness*. Texts and Monographs in Computer Science. Springer: 1986.

[30] Francez, Nissim, und Amir Pnueli: „A Proof Method for Cyclic Programs". *Acta Informatica* **9** (1978) 133–157.

[31] Goldman, Kenneth J.: „A Compositional Model for Layered Distributed Systems". In: *CONCUR '91*. Springer-Verlag: 1991: 220–234.

[32] Goltz, U., und W. Reisig: „The Non-sequential Behaviour of Petri Nets". *Information and Control* **57** (1983) 125–147.

[33] Golz, Ursula, und Wolfgang Reisig (Eds.): *Logics for Distrubuted Systems, Summary of a Workshop*. GMD: März 1992.

[34] Gomm, Dominik, Ekkart Kindler, Barbara Paech und Rolf Walter: „Compositional Liveness Properties of EN-Systems". In Marsan, M. Ajmone (Ed.): *Applications and Theory of Petri Nets 1993,* $14^{th}$ *International Conference*. Chicago, Illinois, USA: Springer-Verlag: Juni 1993: 262–281.

[35] Gunter, C.A., und D.S. Scott: „Semantic Domains". In Leeuwen, Jan van (Ed.): *Handbook of Theoretical Computer Science*. Band B: Formal Models and Semantics. Elsevier Science Publishers B.V.: 1990: Kap. 12, 633–674.

[36] Harel, D., und A. Pnueli: „On the Development of Reactive Systems". In Apt, K.R. (Ed.): *Logics and Models of Concurrent Systems*. Series F: Computer and System Science **13**. Springer-Verlag: 1985: 477–498.

[37] Henzinger, Thomas A.: „Sooner is safer than later". *Information Processing Letters* **43** (September 1992) 135–141.

[38] Hoare, C.A.R.: „An Axiomatic Basis for Computer Programming". *Communications of the ACM* **12**(10) (Oktober 1969) 576–583.

[39] Hooman, J.J.M., und W.P. de Roever: „An introduction to compositional methods for concurrency and their application to real-time". In Hogrefe (Ed.): *Informatik Aktuell: Formale Beschreibungstechniken für verteilte Systeme.* Springer-Verlag: 1992: 66–109.

[40] Hooman, Jozef, und Willem-P. de Roever: „The Quest Goes On: A Survey of Proofsystems for Partial Correctness of CSP“. In Bakker, J.W. de, W.-P. de Roever und G. Rozenberg (Eds.): *Concurrent Trends in Concurrency*. LNCS **224**. Springer-Verlag: 1986: Kap. 6, 343–395.

[41] Jensen, Kurt: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. EATCS Monographs on Theoretical Computer Science **Volume 1: Basic Concepts**. Springer-Verlag: 1992.

[42] Jones, Cliff. B: „Specification and Design of (Parallel) Programs“. In Mason, R.E.A (Ed.): *Information Processing*. IFIP. Elsevier Science Publishers B.V. (North Holland): 1983: 321–332.

[43] Katz, Shmuel, und Doron Peled: „Interleaving Set Temporal Logic“. *Theoretical Computer Science* **75** (1990) 263–287.

[44] Kindler, Ekkart: „Safety and Liveness Properties: A Survey“. *EATCS-Bulletin* **53** (Juni 1994) 268–272.

[45] Kindler, Ekkart, und Rolf Walter: „Message passing mutex“. In Desel, J. (Ed.): *Structures in Concurrency Theory*. Springer-Verlag: 1995.

[46] Kröger, Fred: *Temporal Logic of Programs*. EATCS Monographs on Theoretical Computer Science **8**. Springer-Verlag: 1987.

[47] Kwiatkowska, Marta Z.: „Event Fairness and Non-Interleaving Concurrency“. *Formal Aspects of Computing* **1** (1989) 213–228.

[48] Kwiatkowska, Marta Z.: „On the Domain of Traces and Sequential Composition“. In Abramsky, S., und T.S.E. Maibaum (Eds.): *TAPSOFT '91*. Springer-Verlag: April 1991: 42–56.

[49] Lamport, Leslie: „Proving the Correctness of Multiprocess Programs“. *IEEE Transactions on Software Engineering* **SE-3**(2) (März 1977) 125–143.

[50] Lamport, Leslie: „Specifying Concurrent Program Modules“. *ACM Transactions on Programming Languages and Systems* **5**(2) (April 1983) 190–222.

[51] Lamport, Leslie: *The Temporal Logic of Actions*. SRC Research Report 79. Digital Equipment Corporation, Systems Research Center. Dezember 1991.

[52] Lamport, Leslie: „Verification and Specification of Concurrent Programs“. In Bakker, J.W. de, W.-P. de Roever und G. Rozenberg (Eds.): *A Decade of Concurrency, Reflections and Perspectives*. Springer-Verlag: Juni 1993: 347–374. REX School/Symposium Noordwijkerhout, The Netherlands.

[53] Lamport, Leslie, und Nancy Lynch: „Distributed Computing: Models and Methods“. In Leeuwen, Jan van (Ed.): *Handbook of Theoretical Computer Science*. Band B: Formal Models and Semantics. Elsevier Science Publishers B.V.: 1990: Kap. 18, 1158–1199.

[54] Manna, Zohar, und Amir Pnueli: „Completing the Temporal Picture“. *Theoretical Computer Science* **83**(1) (1991) 97–130.

[55] Manna, Zohar, und Amir Pnueli: *The Temporal Logic of Reactive and Concurrent Systems. Specification.* Springer-Verlag: 1992.

[56] Manna, Zohar, und Amir Pnueli: „A Temporal Proof Methodology for Reactive Systems“. In Broy, M. (Ed.): *Programm Design Calculi.* 1992: 287–323.

[57] Mazurkiewicz, Antoni: „Compositional Semantics of Pure Place/Transition Systems“. In Rozenberg, G. (Ed.): *Advances in Petri Nets 1988.* LNCS **340**. Springer-Verlag: 1988: 307–330.

[58] Mazurkiewicz, Antoni: „Trace Theory“. In Brauer, W., W. Reisig und G. Rozenberg (Eds.): *Petri Nets: Applications and Relationships to Other Models of Concurrency.* LNCS **255**. Springer-Verlag: September 1986: 279–324.

[59] Merz, Stephan: „An Abstact Account of Composition“. MFCS 1995.

[60] Meseguer, José, und Ugo Montanari: „Petri Nets are Monoids“. *Information and Computation* **88** (1990) 105–155.

[61] Milner, Robin: *Communication and Concurrency.* International Series in Computer Science. Prentice Hall: 1989.

[62] Misra, Jayadev, und K. Mani Chandy: „Proofs of Networks of Processes“. *IEEE Transactions on Software Engineering* **SE-7**(4) (Juli 1981) 417–426.

[63] Mühlbauer, Sabine: *Unety — ein Spezifikations- und Beweiskalkül für elementare Netzsysteme.* Diplomarbeit. Technische Universität München. November 1991.

[64] Mukund, Madhavan, und P.S. Thiagarajan: „A Logical Characterization of Well Branching Event Structures“. *Theoretical Computer Science* **96** (1992) 35–72.

[65] Nolte, Doris, und Lutz Priese: „Fairness in Models with True Concurrency“. In: *Proceedings CONCUR '91.* Springer-Verlag: 1991: 455–469.

[66] Owicki, Susan, und David Gries: „An Axiomatic Proof Technique for Parallel Programs I“. *Acta Informatica* **6** (1976) 310–340.

[67] Owicki, Susan, und Leslie Lamport: „Proving Liveness Properties of Concurrent Programs“. *ACM Transactions on Programming Languages and Systems* **4**(3) (Juli 1982) 455–495.

[68] Pandya, Paritosh K.: „Some Comments on the Assumption-Commitment Framework for Compositional Verification of Distributed Programs“. In Bakker de, de Roever und Rozenberg (Eds.): *Stepwise Refinement of Distributed Systems.* Springer-Verlag: 1990: 622–640.

[69] Pandya, Paritosh K., und Mathai Joseph: „P-A logic – a compositional proof system for distributed programs“. *Distributed Computing* **5** (1991) 37–54.

[70] Peled, Doron, und Amir Pnueli: „Proving partial order properties“. *Theoretical Computer Science* **126** (1994) 143–182.

[71] Peleg, David: „Concurrent Dynamic Logic“. *Journal of the ACM* **34**(2) (1987) 450–479.

[72] Penczek, W.: „A Temporal Logic for Event Structures“. *Fundamenta Informaticae* **11** (1988) 297–326.

[73] Pinter, Shlomit S., und Pierre Wolper: „A Temporal Logic for Reasoning about Partially Ordered Computations“. In: *Proceedings Third Symposium on Principles of Distributed Computations*. ACM. August 1984: 28–37.

[74] Pnueli, Amir: „In Transition From Global to Modular Temporal Reasoning about Programs“. In Apt, K.R. (Ed.): *Logics and Models of Concurrent Systems*. Series F: Computer and System Science **13**. Springer-Verlag: 1985: 123–144.

[75] Pratt, Vaughan: „Modelling Concurrency with Partial Orders“. *International Journal of Parallel Programming* **15**(1) (1986) 33–71.

[76] Reisig, Wolfgang: *Das Verhalten verteilter Systeme*. GMD-Bericht. Oldenbourg Verlag: 1987.

[77] Reisig, Wolfgang: *Elements of a Temporal Logic Coping with Concurrency*. SFB-Bericht 342/23/92 A. Technische Universiät München. November 1992.

[78] Reisig, Wolfgang: *Parallel Composition of Liveness*. SFB-Bericht 342/30/91 A. Technische Universität München. 1991.

[79] Reisig, Wolfgang: „Petri Nets and Algebraic Specifications“. *Theoretical Computer Science* **80** (Mai 1991) 1–34.

[80] Reisig, Wolfgang: *Petrinetze: Eine Einführung*. Studienreihe Informatik. Springer-Verlag: 1982.

[81] Reisig, Wolfgang: „Towards a Temporal Logic for Causality and Choice in Distributed Systems“. In Bakker, J.W. de, und W.-P. de Roever (Eds.): *Linear Time, Branching Time and Partial Order in Logics and Models of Concurrency*. LNCS **354**. Spriger-Verlag: Mai 1988: 603–627.

[82] Sibertin-Blanc, C.: „A Client-Server Protocol for the Composition of Petri Nets“. In Marsan, M-Ajmone (Ed.): *Application and Theory of Petri Nets 1993, $14^{th}$ International Conference*. LNCS **691**. Springer-Verlag: Juni 1993: 377–396.

[83] Sibertin-Blanc, C.: „Cooperative Nets“. In Valette, R. (Ed.): *Application and Theory of Petri Nets, $15^{th}$ International Conference*. LNCS **815**. Springer-Verlag: Juni 1994: 471–490.

[84] Souissi, Younes, und Gérard Memmi: „Composition of Nets Via a Communication Medium“. In Rozenberg, G. (Ed.): *Advances in Petri Nets 1990*. LNCS **483**. Springer-Verlag: 1990: 457–470.

[85] Stølen, Ketil: „An Attempt to Reason about Shared-State Concurrency in the Style of VDM“. In Prehn, S., und W.J. Toetenel (Eds.): *VDM '91 Formal Sofware Development Methods Vol. 1*. Springer-Verlag: Oktober 1991: 324–342.

[86] Valmari, Antti: „Compositional Analysis with Place-Bordered Subnets“. In Valette, R. (Ed.): *Application and Theory of Petri Nets*. Springer-Verlag: 1994: 531–547.

[87] Walter, Rolf: *Petrinetzmodelle verteilter Algorithmen – Intuition und Beweistechnik*. Dissertation. Humboldt-Universität zu Berlin, Institut für Informatik. 1995.

[88] Weber, Rainer: *Eine Methodik für die formale Anforderungsspezifikation verteilter Systeme*. Dissertation. Technische Universität München. 1992.

[89] Winskel, Glynn: „Event Structures“. In Brauer, W., W. Reisig und G. Rozenberg (Eds.): *Petri Nets: Applications and Relationships to Other Models of Concurrency*. Springer-Verlag: 1986: 325–392.

[90] Zwiers, Job: *Compositionality, Concurrency and Partial Correctness*. LNCS **321**. Springer-Verlag: 1989.

[91] Zwiers, Job, und Willem-P. de Roever: „Compositionality and Modularity in Process Specification and Design: A Trace-State Based Approach“. In Banieqbal, B., H. Barringer und A. Pnueli (Eds.): *Temporal Logic in Specification*. LNCS **398**. Springer-Verlag: April 1987: 351–374.

[92] Zwiers, Job, Willem-P. de Roever und Peter van Emde Boas: „Compositionality and Concurrent Networks“. In Brauer, W. (Ed.): *Automata, Languages and Programming* $12^{th}$ *Colloquium*. LNCS **321**. Springer-Verlag: Juli 1985: 509–519.

# Index