Beschreibung und Analyse von Markovmodellen mit großem Zustandsraum

Der Technischen Fakultät der Universität Erlangen–Nürnberg

zur Erlangung des Grades

DOKTOR – INGENIEUR

vorgelegt von

Markus Siegle

Erlangen — 1995

Als Dissertation genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung:	8. Mai 1995
Tag der Promotion:	20. Juli 1995
Dekan:	Prof. Dr. Dr. h.c. F. Durst
Berichterstatter:	Prof. Dr.-Ing. U. Herzog
	Prof. Dr. F. Hofmann

Kurzfassung

In dieser Arbeit geht es um die Leistungsbewertung von parallelen und verteilten Rechensystemen mit Hilfe von Markovmodellen. Es werden Methoden zur Beschreibung und Analyse von Markovmodellen mit großem Zustandsraum untersucht. Stochastische Automaten, stochastische Prozeßalgebren, stochastische Petrinetze und Warteschlangenmodelle werden als gebräuchliche Methoden für die Modellbeschreibung anhand eines gemeinsamen Beispielproblems vorgestellt und anschließend bezüglich einschlägiger Kriterien verglichen. Für ein effizientes Vorgehen bei der Modellierung großer Systeme gibt es unterschiedliche Ansätze, die in der Arbeit diskutiert und zum Teil erweitert werden. Ein zentraler Inhalt der Arbeit ist die Entwicklung einer neuen Modellwelt, die sich durch eine strukturierte Beschreibung des Gesamtmodells, aufgebaut aus interagierenden Submodellen, auszeichnet. Diese Modellwelt ist besonders geeignet zur Beschreibung von Systemen mit replizierten Komponenten. Sie beinhaltet einen effizienten Reduktionsalgorithmus, mit dessen Hilfe der Zustandsraum eines Modells unter Umständen dramatisch reduziert werden kann. Diese Zustandsraumreduktion hat den zusätzlichen Vorteil, daß sie in Verbindung mit einer strukturierten numerischen Analyse eingesetzt werden kann. Dem Zustandsraumproblem wird also in der neuen Modellwelt durch zwei sich gegenseitig ergänzende Methoden Rechnung getragen.

Summary

This thesis addresses problems which arise during performance evaluation of parallel and distributed computer systems using Markov models. Methods for the description and analysis of Markov models with large state space are studied. Stochastic automata, stochastic process algebras, stochastic Petri nets and queueing models are commonly used model description methods. They are explained by means of a common example. A comparison of these methods is then given, using a set of relevant criteria. Different approaches to the efficient modelling of large systems are discussed and partially extended. One major part of this work is the development of a new modelling environment which is characterized by a structured description of the overall model, consisting of communicating submodels. This environment is especially suited for describing systems with replicated components. It contains a reduction algorithm which, under certain circumstances, allows to reduce a model's state space dramatically. This kind of state space reduction can also be used in combination with structured numerical analysis. Therefore, our modelling environment provides two methods for dealing with the state space explosion problem.

Danksagung

Ich möchte an dieser Stelle Herrn Prof. Herzog für die Betreuung dieser Arbeit sehr herzlich danken. Er hat es während meiner Zeit an seinem Lehrstuhl stets verstanden, durch anregende und kritische Diskussionen mein Interesse zu wecken und die Arbeit zu fördern. Herrn Prof. Hofmann danke ich für sein Interesse an der Arbeit und für die Übernahme des Zweitgutachtens.

Schließlich liegt es mir am Herzen, den Kollegen in der MMB-Gruppe für die stets freundschaftliche Unterstützung zu danken. Allen voran sei hier Herr Dr. Klar genannt, der mir mit seinem fachlichen Rat eine unschätzbare Hilfe gewesen ist. Er hat mir außerdem durch seine persönliche Verbundenheit in vielen Situationen sehr geholfen und dadurch entscheidend zum Gelingen dieser Arbeit beigetragen.

Inhalt

1 Einleitung

1.1 Leistungsbewertung mit Markovmodellen

In der Praxis der Systementwicklung stellt sich oft die Frage nach den Eigenschaften von Systemen, die real noch gar nicht existieren. Da Beobachtung und Messung solcher Systeme offensichtlich nicht möglich sind, wird oft von der Möglichkeit Gebrauch gemacht, Modellstudien durchzuführen. Modelle bieten eine unverzichtbare Unterstützung der Systementwicklung, indem sie in einer frühen Phase des Systementwurfs Leistungsvorhersagen ermöglichen. Eine zweite Motivation für den Einsatz von Modellen findet man in Situationen, in denen man durch Modifikation eines existierenden Systems dessen Verhalten verbessern möchte. Experimente am System selbst sind sehr aufwendig und wegen Störung des laufenden Betriebs oft nicht akzeptabel. Umgekehrt kann der laufende Betrieb ein Experiment so stark beeinflussen, daß es unmöglich ist, gesicherte Rückschlüsse aus dem Experiment zu ziehen. Deshalb greift man auch in dieser Situation gern auf die Modellierung zurück. Sie erlaubt Studien zum Zweck der Verbesserung, bis hin zur Optimierung einer Systemkonfiguration oder von variierbaren Systemparametern. Modelle werden also immer dort eingesetzt, wo Experimente am realen System nicht oder nur mit großen Schwierigkeiten durchgeführt werden können.

In der Informatik, insbesondere bei der Leistungsbewertung von Rechensystemen, spielen Modelle seit Jahrzehnten eine wichtige Rolle. Einen naheliegenden und auch viel verwendeten Ansatz bilden Modellierungsmethoden, bei denen versucht wird, die Zusammenhänge zwischen interessierenden Größen mit Hilfe von geschlossenen Formeln auszudrücken. Sobald es jedoch um die Beschreibung und Analyse des Ablaufgeschehens in einem Rechen- oder Kommunikationssystem geht, ist eine derartige summarische Betrachtungsweise zur Beantwortung der anstehenden Fragen nicht mehr ausreichend. Ereignisorientierte Modelle haben sich dagegen für diese Aufgabenstellung als sehr geeignet erwiesen und werden daher auch vorwiegend benutzt. Solche Modelle beschreiben das dynamische Verhalten des realen Systems durch Zustände und Zustandsübergänge (Ereignisse) im Modell. Sie geben dem Modellierer also eine ablauforientierte Sicht auf das System und erlauben gleichzeitig durch die Beschränkung auf die Betrachtung wichtiger Ereignisse eine Abstraktion der Realität auf interessierende Aspekte.

Bei Modellstudien in der industriellen Praxis herrscht die Verwendung von Simulationsmodellen vor, mit denen ein realer Ablauf auf beliebig hohem Detaillierungsgrad unter der Kontrolle einer Simulationsuhr nachgespielt wird. Solche diskrete-Ereignis-Simulationen haben den Vorteil, sehr allgemein zu sein. Um mit ihnen relevante Ergebnisse zu erzielen, d.h. gute Konfidenzintervalle, müssen aber zum Teil äußerst lange Simulationsläufe durchgeführt werden. Die zweite Möglichkeit zur Auswertung ereignisorientierter Modelle besteht darin, analytische Lösungsverfahren einzusetzen, bei denen die interessierenden Leistungsmaße mit Hilfe eines Lösungsalgorithmus bestimmt werden. Derartige Lösungsverfahren sind allerdings nur für Unterklassen von Modellen, in denen gewisse Randbedingungen eingehalten werden, bekannt. Markovmodelle stellen

eine spezielle Klasse analytisch auswertbarer Modelle dar. Die ihnen eigene Einschränkung bei den Verteilungsfunktionen der modellierten Laufzeiten resultiert zwar durchaus in Nachteilen in bezug auf die Freiheitgrade des Modells, bringt aber aus mathematischer Sicht eine gute Beherrschbarkeit mit sich. Um die stationären Zustandswahrscheinlichkeiten eines Markovmodells zu bestimmen, ist “nur” die Lösung eines linearen Gleichungssystems erforderlich. Dafür stehen zwar erprobte und leistungsfähige numerische Lösungsverfahren zur Verfügung, was aber nicht ausschließt, daß der Rechenaufwand bei sehr großen Modellen gewaltig sein kann. Insgesamt ist festzuhalten, daß Markov’sche Modelle in vielen Fällen eine zu bevorzugende Alternative zur Simulation darstellen, weil ihre Analyse über einen viel eleganteren Lösungsweg möglich ist.

Bereits im Jahr 1907 erschien die Originalpublikation von A.A. Markov, in welcher er die Eigenschaften einer Klasse von stochastischen Prozessen definierte und untersuchte, die uns heute als Markovprozesse bekannt sind [Markov, 1907]. Mit Markovmodellen sind im kontinuierlichen Bereich zunächst nur exponentiell verteilte Zeiten beschreibbar. Durch die Verwendung von Phasenverteilungen kann die Klasse der darstellbaren Verteilungsfunktionen fast beliebig erweitert werden, jedoch ist damit ein zum Teil enormer Aufwand verbunden. Einige reale Phänomene lassen sich mit exponentiell verteilten Zeiten adäquat beschreiben, z.B. die Zwischenankunftszeit von Aufträgen an einer Warteschlange, die Dauer eines Telefongesprächs, oder die Zeit zwischen zwei aufeinanderfolgenden Ausfällen einer Hardware-Komponente. Für andere in der Realität auftretende Zeitverteilungen ist die exponentielle Verteilung allerdings weniger geeignet, man denke z.B. an Prozedurausführungszeiten oder Timeouts. Wir wollen die Theorie der Markovprozesse in dieser Arbeit nicht wiederholen, haben jedoch die wichtigsten Grundlagen in Anhang A zusammengefaßt.

Gemäß dem Alter der Disziplin existiert schon ein riesiger Berg von Literatur zum Thema Markovmodellierung. Warum bedarf es also weiterer Arbeiten wie der hier vorliegenden? Durch die Weiterentwicklung in der Computertechnik muß sich, ebenso wie andere Teilbereiche der Informatik, auch das Gebiet der Leistungsmodellierung neuen Herausforderungen stellen. Die heutigen Rechenanlagen und Kommunikationsnetze unterscheiden sich in vielen Punkten von ihren Vorgängern. Wir finden in zunehmendem Maße große, stark strukturierte Systeme vor, deren wachsende Komplexität es schwer macht, Einsicht in ihre dynamischen internen Abläufe zu gewinnen. Dem steht gegenüber, daß gerade solche Systeme oft strengste Kriterien bezüglich Leistungsfähigkeit und Zuverlässigkeit erfüllen müssen und daher unbedingt einer Leistungsbewertung zugänglich sein sollten. Angesichts dieses Zwiespalts besteht die Aufgabe der Leistungsbewerter zunächst darin, zu untersuchen, inwiefern schon bekannte Modellierungstechniken zur Analyse dieser modernen Systeme erfolgreich eingesetzt werden können. Darüberhinaus müssen für die Modellierung solcher Systeme neue, adäquate Techniken entwickelt werden, die die speziellen Eigenschaften innovativer Systeme berücksichtigen bzw. ausnutzen. Zum Beispiel ist es erforderlich, für die Modellierung modularer oder hierarchisch strukturierter Systeme Modelle mit entsprechender Struktur bereitzustellen und dafür eine strukturierte Form der Analyse zu entwickeln.

Es existiert eine ganze Reihe von Modellbeschreibungsmethoden, die dem Menschen die Spezifikation von Markovmodellen erleichtern. Sie erlauben eine problemorientierte

Beschreibung des Modells auf einer höheren Ebene als der von Zuständen und Zustandsübergängen. Warteschlangenmodelle und stochastische Petrinetze sind wohl die am weitesten verbreiteten Vertreter unter den klassischen Modellbeschreibungsmethoden. Trotz der gemeinsamen Basis gibt es durchaus Unterschiede zwischen den einzelnen Modellbeschreibungsmethoden, sowohl was ihre Mächtigkeit betrifft als auch in bezug auf die Modellauswertung. Insbesondere sind auf die jeweilige Methode hin zugeschnittene und dadurch besonders effiziente Analyseverfahren bekannt. In Anbetracht der sich bei der Modellierung moderner Systeme ergebenden neuartigen Problemstellungen sind in jüngster Zeit auch neue — bisher wenig erforschte — Beschreibungstechniken für Markovmodelle entwickelt worden, man denke z.B an die erst seit wenigen Jahren untersuchten stochastischen Prozeßalgebren. Eine weitere wichtige Aufgabe besteht nun darin, Analysemethoden zu entwickeln, die auf solche Beschreibungstechniken abgestimmt sind.

Heute haben wir es oft mit Systemen zu tun, die eine Vielzahl gleichartiger Komponenten beinhalten. Die Leistungsbewertung von Systemen mit dieser Eigenschaft spielt in der Tat eine große Rolle, man denke nur an die meisten Parallelrechner, die aus vielen identischen Prozessor- und Speichermodulen bestehen, oder an Rechnernetze mit einer großen Zahl angeschlossener Stationen. Ist die Anzahl einer gewissen Komponente dabei variabel, so handelt es sich zudem um ein skalierbares System. Die Replikation identischer oder zumindest sehr ähnlicher Komponenten führt einerseits zu äußerst komplexen Systemen, so daß eine Modellierung mit klassischen Methoden meist zum Scheitern verurteilt ist. Andererseits weist die Struktur solcher Systeme aber eine gewisse Regularität auf, die als Vorteil gewertet werden kann. In dieser Arbeit wird daher die Frage gestellt, wie solche regulären Systeme “effizient” modelliert werden können, wie man also die sich ergebenden Symmetrien im Modell ausnutzen kann.

Wenn man die heutige Situation der Leistungsbewertung mit Markovmodellen betrachtet, stellt man fest, daß sich das Gebiet mit zwei grundlegenden Problemen auseinanderzusetzen hat. Erstens gilt es, das Problem der Modellbeschreibung neu zu überdenken, da die traditionellen Modellbeschreibungsmethoden für die Spezifikation moderner Systeme zum Teil nicht mehr geeignet sind, und weil sich die klassischen Methoden nicht gut in den Rahmen der modernen Systementwicklung einfügen. Das zweite Problem ist das der Modellauswertung in Gegenwart komplexer Modelle mit großen Zustandsräumen. Diese beiden Probleme werden im nächsten Abschnitt näher beleuchtet.

1.2 Probleme in der Praxis

1.2.1 Problem der Modellbeschreibung: Akzeptanz bei den Systementwicklern

Ein großes Problem ergibt sich daraus, daß Leistungsbewertung mit Modellen bisher meist außerhalb des System-Entwicklungszyklus steht. Oft spielt Leistungsbewertung erst in einer sehr späten Phase der Systementwicklung eine Rolle. Dann kann es passieren, daß Entwickler feststellen, daß die Realisierung eines Entwurfs zwar funktional korrekt ist, jedoch die gestellten Leistungsanforderungen nicht erfüllt. Hardware- und

Software-Entwickler sind bei Design-Entscheidungen in zunehmendem Maße auf Ergebnisse der Leistungsbewertung angewiesen und brauchen daher verstärkt Zugang zu Modellierungsmethoden. Um eine Integration der Leistungsbewertung in den System-Entwicklungszyklus zu erreichen und dadurch auch der Modellierung zu einer höheren Akzeptanz zu verhelfen, müssen Modellbeschreibungsmethoden stärker den bei der System-Entwicklung gebräuchlichen Beschreibungsmethoden angepaßt werden. Dies sind bei der Software-Entwicklung programmiersprachliche Beschreibungen und formale Spezifikationen. Software-Entwickler sind gewohnt, mit diesen Beschreibungsmethoden umzugehen, und deshalb sollten diese Mittel auch für die Beschreibung von Leistungsmodellen verwendet werden.

Mit dem eben genannten Problem hängt ein weiteres, nicht weniger wichtiges zusammen: Große unstrukturierte Modelle sind für den Menschen nur schwer zu überschauen. Die Modellierung leidet ab einer gewissen Modellgröße und der dadurch verursachten Unübersichtlichkeit unter starker Fehleranfälligkeit. Außerdem sind riesige monolithische Modelle schwer zu modifizieren und daher sehr unhandlich. Deshalb sollten Modellierungstechniken entwickelt werden, die ein strukturiertes, also modulares oder hierarchisches Vorgehen bei der Modellierung ermöglichen. Diese Forderung wird von vielen etablierten Modellierungsmethoden gar nicht oder nur zu einem geringen Grad erfüllt.

Ein möglicher Lösungsweg für diese Probleme besteht darin, die bei der Systementwicklung eingesetzten formalen Spezifikationen so zu erweitern, daß mit ihnen auch die Beschreibung von Markovmodellen möglich ist. Da formale Spezifikationstechniken starke Strukturierungsmöglichkeiten anbieten, kann auf diese Weise neben der mangelnden Akzeptanz auch das Problem der monolithischen Modelle überwunden werden. Um die neuen Beschreibungstechniken und die strukturierten Modellierungstechniken der Praxis zugänglich zu machen, müssen sie natürlich durch Werkzeuge unterstützt werden.

1.2.2 Auswerteproblem: Kombinatorisches Anwachsen des Zustandsraums

Größe und die Komplexität eines realen Systems spiegeln sich in entsprechenden Eigenschaften des Modells wider. So resultiert die Beschreibung komplexer Zusammenhänge mit einem Markovmodell in der Regel in einem großen Zustandsraum. Aus Sicht der theoretischen Behandlung wirft dies zwar keine Probleme auf, für die praktische Durchführung der Analyse kann sich die Größe des Zustandsraums jedoch fatal auswirken. Da die bei der Analyse eingesetzten Rechner, ganz gleich wie leistungsfähig sie auch sein mögen, immer nur über einen begrenzten Speicher verfügen, können mit ihnen nur Modelle bis zu einer bestimmten Größe gespeichert und analysiert werden. Ab einer gewissen Modellgröße führen zudem Swapping-Aktivitäten zu unakzeptablen Geschwindigkeitsverlusten bei der Modellauswertung. Abgesehen vom reinen Speicheraufwand ist bei sehr großen Modellen natürlich auch die Rechenzeit enorm. Besonders skalierbare parallele Systeme haben oft die unangenehme Eigenschaft, daß der Zustandsraum mit zunehmendem Parallelitätsgrad exponentiell anwächst. Man spricht dann vom Phänomen der sogenannten Zustandsraumexplosion.

Schon anhand eines sehr einfachen Beispiels kann man das schnelle Anwachsen des Zustandsraums eindrucksvoll demonstrieren: Im Vorgriff auf Kapitel 2 verwenden wir dazu die Beschreibungsmethode der verallgemeinerten stochastischen Petrinetze (GSPN). Das Petrinetz in Abb. 1.1 stellt N Prozesse dar, die unter gegenseitigem Ausschluß auf ein gemeinsames Betriebsmittel zugreifen. Das Modell ist skalierbar in dem Sinn, daß — wie in der Abbildung durch die Punkte angedeutet — die Anzahl der Prozeß-Subnetze variiert werden kann.

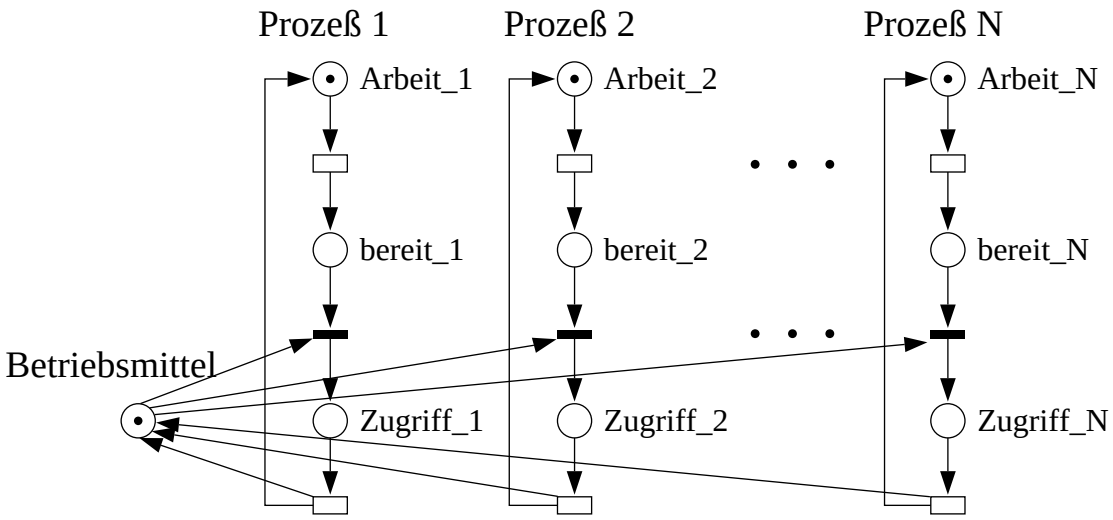


Abbildung 1.1: Modellierung von gegenseitigem Ausschluß mit Hilfe eines Petrinetzes

Dieses Modell wurde mit Hilfe des Werkzeugs DSPNexpress [Lindemann, 1992] analysiert. In der nachfolgenden Tabelle 1.1 sind die Zustandszahlen und Berechnungszeiten für verschiedene Werte von N angegeben.

# Prozessoren N	**Zeit [min:s:ms]**	**# Zustände** $S(N)$
1	0:15:960	2
2	0:16:730	5
5	0:16:830	81
8	0:20:790	1025
10	0:41:780	5121
12	2:35:220	24577
13	> 3 h	53249

Tabelle 1.1 Zustandszahlen für das Beispiel “Gegenseitiger Ausschluß”

Für eine Prozessorenzahl von $N = 13$ mußte die Analyse nach ca. 3 h wegen Speicherplatzmangels abgebrochen werden!

Hier haben wir es mit einem sehr einfachen aber typischen Beispiel für ein exponentielles Wachstum des Zustandsraums bei skalierbaren Systemen zu tun.

Es sei an dieser Stelle darauf hingewiesen, daß sich für die in Abb. 1.1 dargestellte Familie von Netzen aufgrund einfacher kombinatorischer Überlegungen leicht auch eine geschlossene Formel für die Zustandszahl $S(N)$ in Abhängigkeit von N angeben läßt. Für die Anzahl der stabilen Netzmarkierungen, die der Zustandszahl der zugehörigen Markovkette entspricht, gilt nämlich die Beziehung

$$S(N) = 1 + N2^{N-1}$$

Der Term 1 steht für den Fall, daß kein Prozeß auf das Betriebsmittel zugreift, was nur in einer einzigen stabilen Netzmarkierung zutrifft. Der Term $N2^{N-1}$ entspricht dem Fall, daß ein Zugriff durch einen der N Prozesse vorliegt, wobei alle übrigen $N-1$ Prozesse entweder arbeiten oder auf den Zugriff warten, also zwei Alternativen für ihr Verhalten haben.

Für den Speicherbedarf eines Markovmodells ist die Anzahl der Zustände nicht alleine maßgebend. Auch die Anzahl möglicher Übergänge zwischen den Zuständen stellt einen ganz entscheidenden Einflußfaktor dar. Die Speicherung der möglichen Zustandsübergänge geschieht in Matrixform, wobei Modelle mit vielen möglichen Übergängen in stärker besetzten Matrizen, d.h. Matrizen mit mehr von Null verschiedenen Elementen, resultieren. Da wegen der im allgemeinen dünnen Belegung zur Speicherung sinnvollerweise Techniken für dünnbesetzte Matrizen eingesetzt werden, bei denen nur für von Null verschiedene Matrixelemente Speicher benötigt wird, hat die Anzahl der Zustandsübergänge direkten Einfluß auf den Speicherbedarf.

Die Rechenzeit für die Analyse einer Markovkette hängt außer von der Zustandszahl und der Zahl möglicher Zustandsübergänge sehr stark von den numerischen Werten der Parameter des Modells, im zeitkontinuierlichen Fall also von den verwendeten Raten ab. Für die Analyse komplexer Markovmodelle werden meist iterative Lösungsverfahren herangezogen, deren Konvergenzverhalten bei unterschiedlichen Parametern völlig anders ausfallen kann. Sind in einem Modell z.B. Raten enthalten, die sich um mehrere Dimensionen unterscheiden (stiffness), so wirkt sich das negativ auf die Konvergenz des Lösungsverfahrens aus.

1.3 Ziele und Inhalte dieser Arbeit

Trotz der prinzipiellen mathematischen Beherrschbarkeit von Markovmodellen ergeben sich also Schwierigkeiten, wenn es um die praktische Durchführung von Modellstudien geht. Diese Probleme haben ihre Ursache in der Größe und Komplexität der mit Hilfe von Modellen zu untersuchenden realen Systeme. Ziel der Forschung muß es also sein, in Gegenwart komplexer Systeme Modellierungstechniken zu entwickeln und bereitzustellen, die die Durchführung von Modellstudien mit möglichst geringem Aufwand an Speicherplatz und Rechenzeit gestatten. Dafür wollen wir den Begriff *effiziente Modellierung* benutzen. Um ein effizientes Vorgehen bei der Modellierung zu erreichen, wird gefordert, daß einerseits Struktur und Verhalten des realen Systems vom Modell in adäquater Weise repräsentiert werden, daß jedoch auf der anderen Seite eine möglichst starke Abstraktion von der Realität stattfindet. Um unnötigen Aufwand zu vermeiden, sollte das Modell nur diejenigen Aspekte des zu untersuchenden Systems

beschreiben, die zur Beantwortung der anstehenden Fragen benötigt werden. Es ist also wichtig, daß im Modell eine Beschränkung auf das Wesentliche vorgenommen wird. Die im Rahmen der anstehenden Untersuchung unwesentlichen Details des realen Systems sollten nicht Bestandteil des Modells sein. Unter Beachtung dieser Leitlinien kann eine Verschwendung von Ressourcen vermieden werden. Zwei wichtige Voraussetzungen für ein effizientes Vorgehen bei der Modellierung sind die Wahl einer geeigneten Methode für die Beschreibung des Modells und die Verfügbarkeit eines geeigneten Analyseverfahrens zur anschließenden Modellauswertung. Beide Aufgaben, sowohl die Modellerstellung als auch die Analyse des Modells, sind selbstverständlich nur mit Unterstützung durch Software-Werkzeuge praktisch durchführbar.

Mit dieser Arbeit werden die folgenden Ziele verfolgt:

- Wir möchten durch eine Übersicht und Gegenüberstellung existierender Modellbeschreibungsmethoden für die Leistungsbewertung paralleler und verteilter Systeme den heutigen Stand der Forschung darstellen.
- Im Rahmen einer Weiterentwicklung sollen bereits bekannte Techniken kombiniert und in eine neue Methode integriert werden.
- Es geht uns vor allem darum, neue Wege aufzuzeigen, die sich aus der Berücksichtigung der speziellen Eigenschaften moderner paralleler und verteilter Systeme ergeben. Dies sind Strukturiertheit, Regularität und die daraus resultierenden Symmetrien auf der Ebene des Zustandsraums.
- Das Gesamtziel der Arbeit ist es, einen Beitrag dazu zu leisten, die traditionellen Schwierigkeiten bei Modellen mit großem Zustandsraum zumindest teilweise zu überwinden.

Diesen Zielen entsprechen auch die Inhalte der einzelnen Kapitel der Arbeit:

- Es erfolgt zunächst eine Sichtung und Klassifizierung existierender Methoden für die Leistungsbewertung mit Markovmodellen. Dabei wird insbesondere eine Übersicht über gängige aber auch neuartige Modelbeschreibungsmethoden auf höherer Ebene gegeben, und anschließend ein wertender Vergleich zwischen diesen vorgenommen. (Kapitel 2)
- Die Arbeit enthält eine Übersicht und Bewertung von vorgefundenen Ansätzen, die sich speziell mit der Überwindung der Schwierigkeiten bei Modellen mit großem Zustandsraum beschäftigen. In diesem Zusammenhang werden auch Weiterentwicklungen dargestellt und neue eigene Ansätze vorgeschlagen. (Kapitel 3)
- Wir stellen eine neue Modellwelt vor, die auf den Resultaten der Untersuchung von Kapitel 3 basiert. Eine wesentliche Erweiterung gegenüber anderen Methoden besteht in der Berücksichtigung von Modellen, die bestimmte Teile mehrfach, d.h. repliziert beinhalten. Die Modellwelt sieht Konstrukte für die Beschreibung von Systemen mit Replikationen vor und beinhaltet effiziente Techniken für die Analyse der sich ergebenden Modelle. (Kapitel 4)
- Schließlich befassen wir uns mit möglichen Verallgemeinerungen und Weiterentwicklungen der Modellwelt. (Kapitel 5)

Diese Arbeit umfaßt also sowohl Aspekte der Analyse als auch der Synthese von Methoden für die effiziente Modellierung paralleler und verteilter Systeme. Die Analyse

nimmt sich der Aufgabe an, die auf dem Gebiet der Markovmodellierung in jüngster Zeit publizierten Beschreibungs- und Auswertemethoden zu sichten und zu klassifizieren. Aus dieser analysierenden Arbeit erwächst dann eine synthetisierende: Sie besteht in der Entwicklung einer neuen Modellwelt und neuer, dafür geeigneter Auswertemethoden.

2 Modellbeschreibung auf höherer Ebene

Die Bedeutung von Markovmodellen als bewährte Methode für die Leistungsbewertung ist unumstritten. Bevor ein eigentliches Markovmodell existiert, das ausgewertet werden kann, stellt sich aber zunächst die Frage nach seiner Beschreibung. Der Leistungsbewerter geht aus von einer Problemstellung, die das zu analysierende System und eine Reihe durch das Modell zu beantwortende Fragen beinhaltet. Auf den Entschluß, mit Markovmodellen zu arbeiten, hat als nächster Schritt die Auswahl einer geeigneten Beschreibungsmethode zu folgen. An sie werden wichtige Anforderungen gestellt, deren Erfülltsein ganz entscheidend zum Erfolg einer Leistungsbewertungsstudie beiträgt. So sollte die Modellbeschreibung eine möglichst realitätsnahe und problemorientierte Darstellung der interessierenden Strukturen und Abläufe ermöglichen und dabei für den menschlichen Betrachter gut verständlich sein. Die Verwendung von Methoden der Modellbeschreibung auf höherer Ebene erspart es dem Modellierer, sich bei der Erstellung eines Modells um sämtliche Details wie Zustände und Zustandsübergänge zu kümmern. Das heißt, daß die Modellierung auf einem gewissen Abstraktionsgrad stattfinden kann, der dem menschlichen Benutzer die Spezifizierung von Markovmodellen und den Umgang damit erleichtert und ihm dadurch eine Konzentration auf das eigentliche Bewertungsproblem gestattet.

In diesem Kapitel werden Methoden diskutiert, mit deren Hilfe sich Markovmodelle auf einer höheren Ebene als der der eigentlichen Markovkette spezifizieren lassen. Durch die Darstellung der verschiedenen Modellbeschreibungsmethoden anhand eines gemeinsamen Beispiels, das auch in Kapitel 4 wieder aufgegriffen wird, und die daran anknüpfende Diskussion sollen einerseits Gemeinsamkeiten, andererseits Unterschiede zwischen den einzelnen Methoden der Modellbeschreibung sichtbar werden. Dafür wird im Anschluß an die Vorstellung der einzelnen Methoden ein Vergleich gezogen.

Das wichtigste Anliegen dieses Kapitels ist es, die Eignung der verschiedenen Modellbeschreibungsmethoden für eine effiziente Beschreibung großer Systeme zu untersuchen.

2.1 Methodenunabhängige Beschreibung eines Beispielproblems

Parallelrechner bilden ein wichtiges Anwendungsgebiet für die Leistungsbewertung mit Modellen [Ajmone Marsan *et al.*, 1986], wobei in jüngster Zeit eine kombinierte Bewertung der Leistungsfähigkeit und der Zuverlässigkeit von Multiprozessorsystemen stark an Interesse gewonnen hat (Schlagwort “Performability” [Meyer, 1980; Trivedi und Malhotra, 1993]). In den folgenden Abschnitten wird deshalb als typisches Beispiel das Modell eines Multiprozessorsystems mit möglichen Prozessorausfällen betrachtet und mit Hilfe der verschiedenen Methoden beschrieben. Das Beispiel soll jedoch nun zunächst allgemein und informell erläutert werden.

Es handelt sich um ein Multiprozessorsystem mit N Prozessoren, wobei die Anzahl N zunächst unspezifiziert bleibt, siehe Abb. 2.1. Alle Prozessoren greifen auf den (verteilten oder gemeinsamen) Speicher über einen gemeinsamen Bus zu. Der Bus darf zu jedem Zeitpunkt von höchstens einem Prozessor benutzt werden und es wird

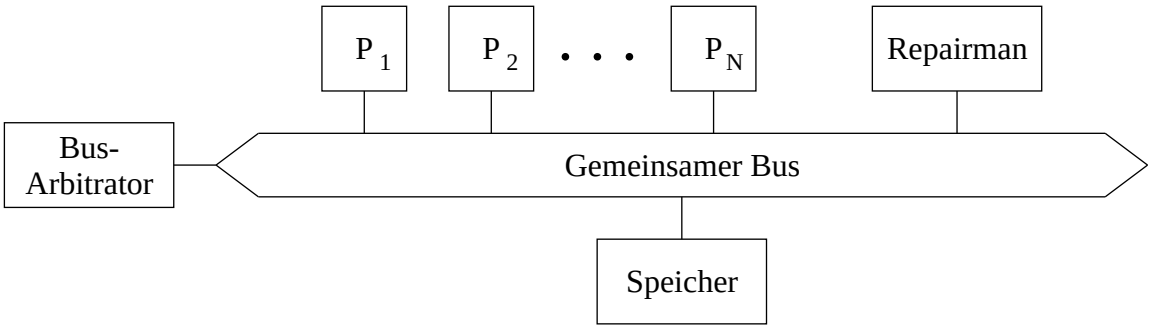


Abbildung 2.1: Busgekoppeltes Multiprozessorsystem
mit Bus-Arbitrator und Repairman

angenommen, daß der Bus während des gesamten Zugriffsvorgangs für die anderen Prozessoren gesperrt bleibt. Der exklusive Zugriff auf den Bus wird durch eine Arbitrierungskomponente, den Bus-Arbitrator, sichergestellt.

Prozessoren können ausfallen. Der Einfachheit halber wird angenommen, daß ein Ausfall eines Prozessors nur während der normalen Arbeitsphase, nicht aber während eines Speicherzugriffs auftreten kann. Der Ausfall eines Prozessors aktiviert eine Reparaturkomponente, den sogenannten Repairman. Dieser nimmt so schnell wie möglich eine Rekonfigurierung des Multiprozessorsystems vor (downgrade), um den ausgefallenen Prozessor auszukoppeln. Da für die Rekonfigurierung der Bus benötigt wird, belegt der Repairman mit hoher Priorität den Bus, bzw. ist sogar berechtigt, den Bus einem gerade zugreifenden Prozessor zu entziehen (preemption). Nach erfolgter Rekonfigurierung wird der Bus wieder freigegeben und das System kann mit verminderter Prozessorzahl weiterarbeiten, während der ausgefallene Prozessor vom Repairman repariert wird. Zum anschließenden Wiederhinzuschalten (upgrade) des reparierten Prozessors benötigt der Repairman wieder den Bus. Diesmal ist er wegen der geringeren Dringlichkeit des Vorgangs aber nicht berechtigt, den Bus einem gerade auf den Speicher zugreifenden Prozessor zu entziehen.

Ein zweiter, hinzukommender Prozessorausfall während des Ausfalls eines ersten Prozessors kann in diesem System erst nach erfolgter Reparatur und Wiederhinzuschalten des ersten ausgefallenen Prozessors bearbeitet werden.

2.2 Methoden der Modellbeschreibung auf höherer Ebene

Die Reihenfolge der folgenden Darstellungen verschiedener Modellbeschreibungsmethoden ist so gewählt, daß mit zustandsnahen Methoden begonnen und dann nach und nach zu abstrakteren Methoden übergegangen wird.

2.2.1 Stochastische Automaten

Betrachtet man die historische Entwicklung stochastischer Automaten, so stellt man fest, daß sie als Erweiterung aus den endlichen Automaten (finite state automata, FSA, oder finite state machines, FSM) hervorgegangen sind [Liu, 1989]. Wie wir auch bei anderen Beschreibungsmethoden sehen werden, ist es eine typische Entwicklung, daß

man ausgehend von einem rein funktionalen Formalismus zu einer Beschreibungsmethode für Markovmodelle gelangt. Bei endlichen Automaten spielt die Zeit nur insofern eine Rolle, als Reihenfolgebeziehungen betrachtet werden, nicht aber quantifizierbare Zeitdauern. Endliche Automaten werden vor allem beim Entwurf und bei der Verifikation von Kommunikationsprotokollen eingesetzt [Brand und Zafiropulo, 1983]. Dabei werden FSA-Modelle hinsichtlich qualitativer Aspekte wie Deadlockfreiheit und Konformität mit den funktionalen Anforderungen eines Protokolls analysiert. Ein endlicher Automat besteht aus Zuständen und Übergängen zwischen Zuständen (Transitionen). Zustandsübergänge können Eingaben erfordern (input signal, gekennzeichnet durch das Symbol "$+$") und Ausgaben erzeugen (output signal, Symbol "$-$"). Daneben sind auch interne Zustandsübergänge ohne Ein- oder Ausgabe möglich.

Abb. 2.2 zeigt, wie sich das Beispielmodell mit endlichen Automaten darstellen läßt. Es sind nur einer der insgesamt N gleichartigen Prozessoren, der Bus-Arbitrator und der Repairman abgebildet. Zunächst sollen sich sämtliche Prozessoren im Ausgangszustand *work* befinden. Im Normalbetrieb signalisiert ein Prozessor dem Bus-Arbitrator einen Buszugriffswunsch durch Ausgabe eines *request bus l* Signals (das *l* steht für "low" und deutet an, daß es sich um einen Zugriff mit niedriger Priorität handelt). Dann begibt sich der Prozessor in einen Wartezustand. Sobald der Bus zur Verfügung steht, wird er dem anfragenden Prozessor durch Übermittlung des Signals *grant bus l* zugeteilt. Der Prozessor darf dann auf den Bus zugreifen und kehrt danach unter Aussendung eines *release bus* Signals in den Ausgangszustand zurück.

Wenn einer der Prozessoren ausfällt, wird die Fehlerbehandlung eingeleitet, bei welcher der Repairman zum Einsatz kommt. Dieser Vorgang wird durch den Übergang eines Prozessors vom Zustand *work* in den Zustand *down* angestoßen. Der zentrale Repairman wird dabei durch das Signal *breakdown* aktiviert. Er fordert daraufhin durch das Signal *request bus h* (hohe Priorität) den Bus vom Bus-Arbitrator an. Unter Umständen erfordert die Zuteilung des Busses an den Repairman, daß der Bus einem gerade zugreifenden Prozessor mittels des Signals *preempt* explizit entzogen wird. Sobald der Repairman im Besitz des Busses ist, wird das Multiprozessorsystem rekonfiguriert (*downgrade*). Danach wird der Bus vom Repairman wieder freigegeben. Das System kann dann mit verminderter Leistung weiterarbeiten, während die Repairman-Komponente die Reparatur des ausgefallenen Prozessors durchführt. Ist diese abgeschlossen, wird das Gesamtsystem wieder zurückkonfiguriert (*upgrade*), wozu der Repairman wiederum in den Besitz des Busses gebracht werden muß. Dies geschieht mit normaler, d.h. niedriger Priorität.

Sämtliche Signale in diesem Modell müssen aus Gründen der Eindeutigkeit mit einer Angabe über die Identität des sendenden Automaten versehen sein, was jedoch in der Abbildung nicht berücksichtigt ist. Nur so ist es möglich, daß beispielsweise das *grant bus* Signal als Antwort auf ein empfangenes *request bus* Signal vom Bus-Arbitrator an den richtigen Prozessor gesendet wird. Die gestrichelten Pfeile in Abb. 2.2 stellen die Kommunikationskanäle zwischen den einzelnen Automaten für den Austausch von Signalen dar.

Um mit den Modellen auch eine Überprüfung der zeitlichen Eigenschaften eines Protokolls durchführen zu können, insbesondere um dessen Konformität mit den zeitlichen Anforderungen zu untersuchen, wurden aufbauend auf den endlichen Automaten zeit-

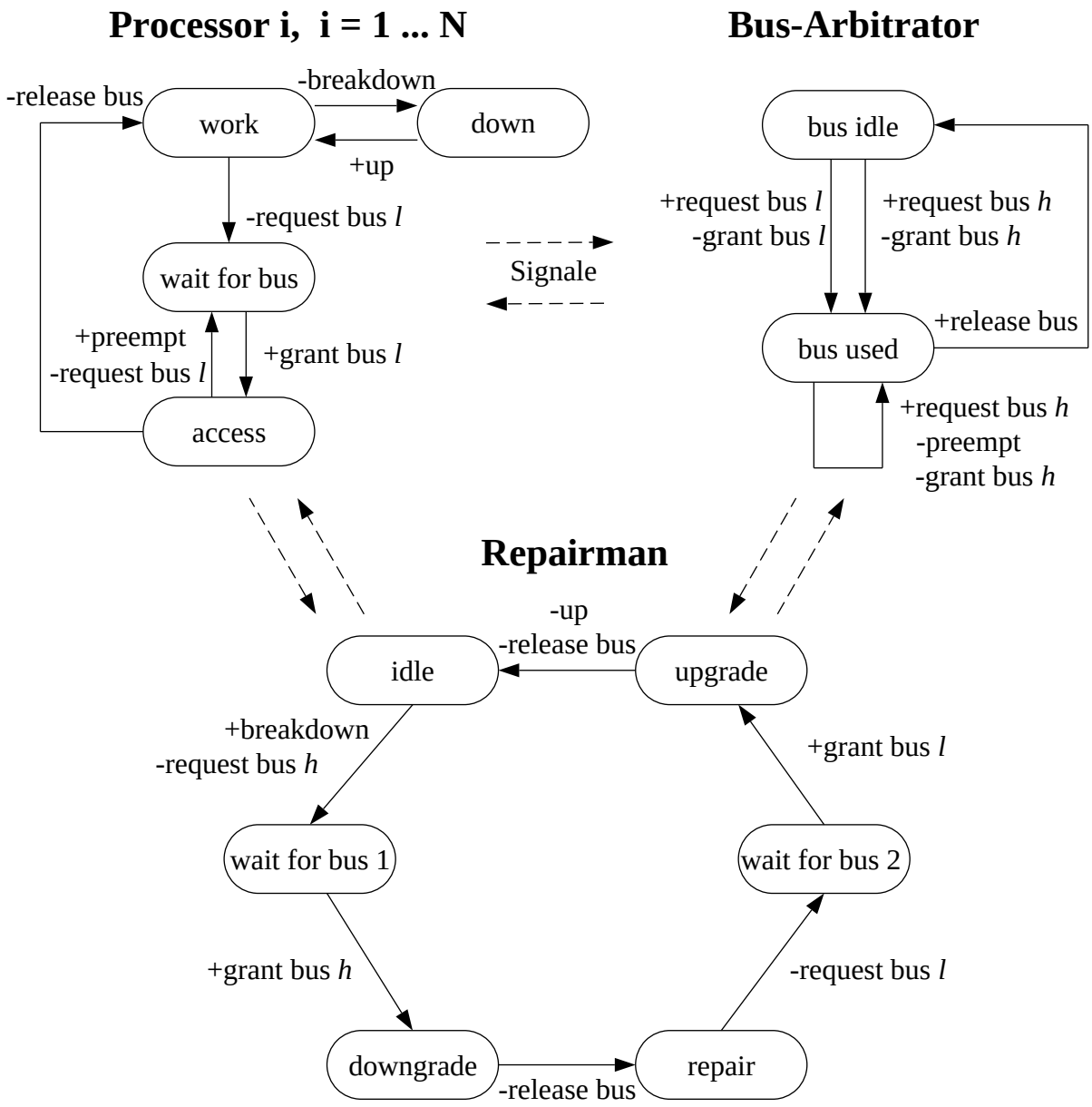


Abbildung 2.2: Asynchrones FSA-Modell des Multiprozessors mit Breakdown und Repair, dargestellt sind nur ein Prozessor, der Bus-Arbitrator und der Repairman

behaftete endliche Automaten entwickelt. Nachdem zunächst deterministische Zeiten betrachtet wurden [Rudin, 1983], kam es bald auch zur Verwendung stochastisch verteilter Zeiten [Kritzinger, 1985; Rudin, 1985]. Stochastisch verteilte Zeiten werden benötigt, um Leistungsbewertung durchzuführen. Bei interagierenden stochastischen Automaten kommt es aber sehr schnell zum Problem des zu großen Zustandsraums, so daß eine Markovanalyse praktisch nicht durchführbar ist. Einige Autoren haben dieses Problem durch die Konzentration auf einen Automaten in Isolation umgangen, wobei der Einfluß der anderen Automaten durch Parameter in die Berechnung eingeht. Auf diese Weise gelangt man zu iterativen Analyseverfahren, die zu einer Approximation der Lösung führen (siehe Zitate in [Plateau, 1985]).

Beim Übergang von endlichen Automaten zu stochastischen Automaten trifft man auf folgende Problematik: Obwohl bei kombinierten endlichen Automaten urspünglich von einem synchronen Fortschreiten ausgegangen wurde (vgl. z.B. [Bode und Händler, 1980]), gehen die meisten im Rahmen des Protokollentwurfs eingesetzten Automatenmodelle von einer *asynchronen Kommunikation* aus. So liegt auch dem FSA-Modell in Abb. 2.2 die Vorstellung einer asynchronen Kommunikation zwischen den Automaten zugrunde, d.h. Signale werden als Nachrichten zwischen den Automaten versandt. Ankommende Signale werden in einem Eingangspuffer abgelegt und (zumeist) in der Reihenfolge ihres Eintreffens bearbeitet. Im Gegensatz dazu wird bei dem in dieser Arbeit verwendeten Typus von stochastischen Automaten zur Darstellung der Abhängigkeiten und Koordinierung zwischen den einzelnen Automaten ein *Rendezvouzkonzept* verwendet. Die Synchronisation von zwei oder mehr Automaten geschieht mit Hilfe von speziellen, synchronisierenden Ereignissen, deren Semantik so definiert ist, daß bei allen an der Synchronisation beteiligten Automaten gleichzeitig ein Zustandsübergang stattfindet. Solche stochastische Automaten werden wir als Stochastic Automata Networks (SAN) bezeichnen. Das SAN-Konzept unterscheidet sich von dem asynchronen, nachrichtenorientierten Konzept bei den endlichen Automaten. Die Auswirkungen dieses Unterschieds werden weiter unten im Beispiel aus Abb. 2.5 verdeutlicht.

Zur Einführung in die Modellbeschreibungsmethode der SANs werde jedoch zunächst in Abb. 2.3 eine vereinfachte Version des Multiprozessormodells betrachtet. In dieser Version werden Prozessorausfall und Reparatur außer acht gelassen, es geht also nur um den vom Bus-Arbitrator koordinierten Zugriff der Prozessoren auf den Speicher. Wie der Abbildung zu ersehen ist, gibt es in SANs zwei Arten von Zustandsübergängen: Die einen sind durch Pfeile gekennzeichnet, die als Beschriftung lediglich eine Rate tragen, mit der der betreffende Zustandsübergang stattfindet, während die anderen mit Bezeichnern versehen sind, welche den Namen eines synchronisierenden Ereignisses definieren. Erstere betreffen nur einen einzelnen stochastischen Automaten, während die letzteren zur Darstellung der Synchronisation von zwei oder mehr Automaten verwendet werden. Für jedes synchronisierende Ereignis ist an genau einer Stelle seine Rate

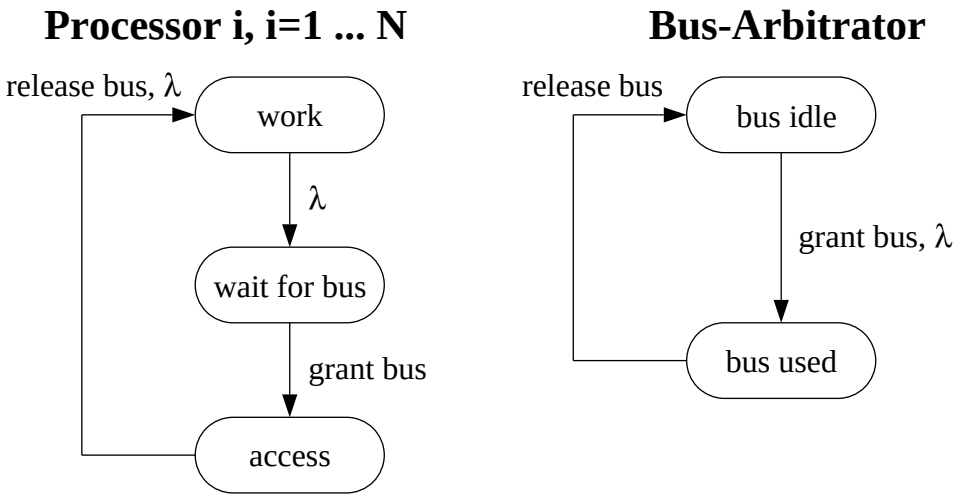


Abbildung 2.3: SAN-Modell des Multiprozessors, bestehend aus $N+1$ interagierenden stochastischen Automaten (vereinfachte Version ohne Breakdown und Repair)

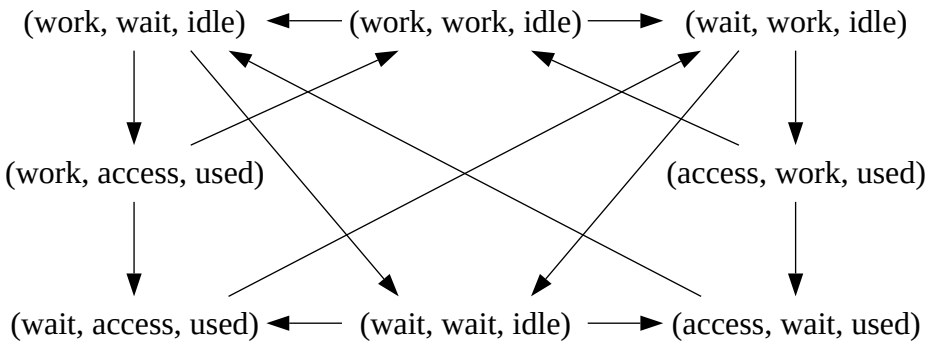


Abbildung 2.4: Zustandsdiagramm für das
SAN-Modell aus Abb. 2.3 mit zwei Prozessoren

angegeben. Im Beispiel haben der Einfachheit halber alle Raten den einheitlichen Wert λ, aber selbstverständlich kann für jedes Ereignis eine andere Rate angegeben werden.

Bei den stochastischen Automaten handelt es sich um eine zustandsorientierte Darstellung, d.h. um eine Modellbschreibung auf relativ niedriger Ebene ("low-level"), welche eine sehr große Ähnlichkeit mit der zugrundeliegenden Markovkette aufweist. Ein einzelner stochastischer Automat kann als graphische Darstellung einer Markovkette angesehen werden. SANs, die aus mehreren interagierenden Automaten bestehen, definieren eine Markovkette, deren Zustandsraum eine Teilmenge des cartesischen Produkts der Einzelzustandsräume der beteiligten Automaten ist. Im allgemeinen sind nicht alle Elemente dieses cartesischen Produkts mögliche Zustände der durch das SAN definierten Markovkette, da bestimmte Zustandskombinationen aufgrund der Synchronisationsbeziehungen nicht erreichbar sind. Als Beispiel für die einem SAN zugeordnete Markovkette ist in Abb. 2.4 das Zustandsdiagramm (der Erreichbarkeitsgraph) des Modells aus Abb. 2.3 für $N = 2$ Prozessoren abgebildet. Es gibt 8 erreichbare Zustände, d.h. von den insgesamt $3 \times 3 \times 2 = 18$ Zuständen der Produktmenge sind 10 nicht erreichbar.

Abb. 2.5 zeigt das dem FSA-Modell aus Abb. 2.2 entsprechende SAN-Modell, also ein SAN-Modell das gegenüber demjenigen aus Abb. 2.3 wieder um die Funktionen Breakdown und Repair erweitert ist. Obwohl zwischen den beiden Abbildungen eine sehr große Ähnlichkeit festgestellt werden kann, zeigt sich, daß es nicht genügt, das FSA-Modell mit Angaben über Raten zu versehen, um ein entsprechendes SAN-Modell zu erhalten. Unterschiede sind in erster Linie dadurch bedingt, daß beim FSA-Konzept die Automaten über Signale miteinander kommunizieren, die als Nachrichten verschickt werden, während bei SAN-Modellen der simultane Zustandsübergang von zwei oder mehr Automaten zur Synchronisation benutzt wird. Bei FSA-Modellen wie dem aus Abb. 2.2 ist eine implizite Speicherkomponente in den in der Abbildung nicht dargestellten Nachrichtenwarteschlangen vorhanden. Diese muß bei SAN-Modellen bei Bedarf explizit durch zusätzliche Zustände hinzugefügt werden. Ein gutes Beispiel hierfür ist das Zustandspaar *down* und *under repair* im SAN-Modell von Abb. 2.5, welches benötigt wird, um Information über den zeitlich überlappten Ausfall von zwei oder mehr Prozessoren nicht zu verlieren. Diese Information wird beim FSA-Modell durch *breakdown*-Nachrichten in der Eingangswarteschlange des Bus-Arbitrators gespeichert. Dabei ist zu beachten, daß die Reparaturwünsche mehrerer ausgefallener Prozessoren im FSA-Modell in der Reihenfole ihres Eintreffens (oder nach einer beliebigen anderen

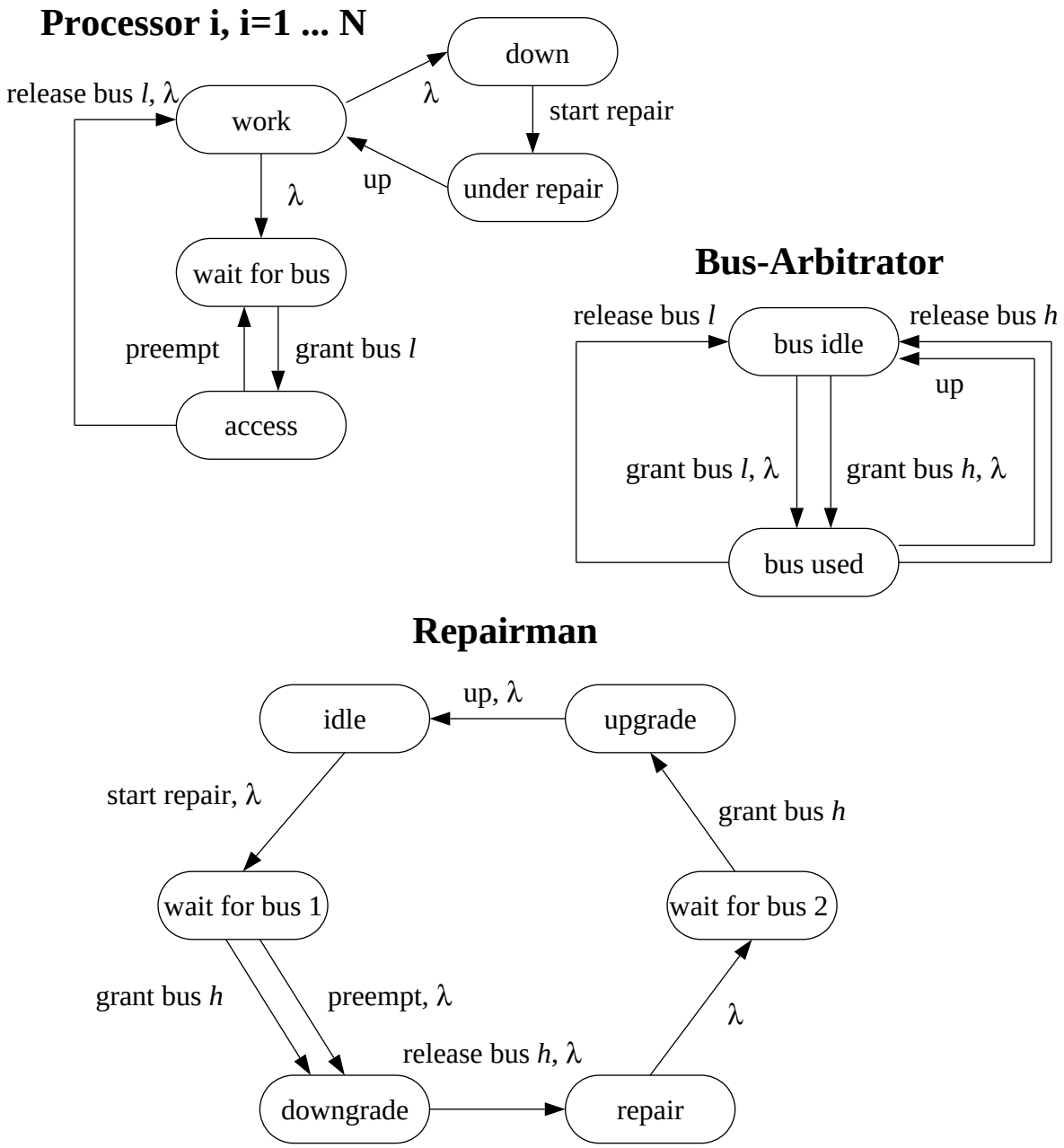


Abbildung 2.5: SAN-Modell des Multiprozessors mit Breakdown und Repair, bestehend aus $N + 2$ interagierenden stochastischen Automaten

Strategie) bearbeitet werden können, während im SAN-Modell bei mehreren ausgefallenen Prozessoren nicht mehr unterschieden werden kann, in welcher Reihenfolge diese in den Zustand *down* eintraten, weshalb hier die Bedienstrategie “Random” gewählt werden muß. Weiterhin fällt auf, daß die *release bus*-Nachricht des FSA-Modells sich in zwei synchronisierenden Ereignissen, nämlich *release bus l* und *release bus h* widerspiegelt. Diese Trennung ist notwendig, um die an der Synchronisation teilnehmenden Automaten eindeutig und korrekt zu bestimmen.

In [Plateau, 1985] und mehreren darauf aufbauenden Veröffentlichungen [Plateau und Fourneau, 1991; Plateau und Atif, 1991] findet man eine intensive Auseinandersetzung mit stochastischen Automaten. Auch dort werden exponentiell verteilte Verweilzeiten

in den einzelnen Zuständen angenommen, so daß Markov'sche Eigenschaften vorliegen. Diese Modellwelt wird durch ein Werkzeug unterstützt, welches den Namen PEPS trägt [Plateau *et al.*, 1988]. Das Prinzip der von Plateau beschriebenen speicherplatzsparenden Methode für die exakte Gleichgewichtsanalyse von gegenseitig voneinander abhängigen, interagierenden stochastischen Automaten wird in Abschnitt 3.5.2 erläutert.

Es besteht ein enger Zusammenhang zwischen den beim Softwareentwurf eingesetzten formalen Spezifikationstechniken und endlichen Automaten. Estelle und SDL [Hogrefe, 1989; Linn, 1985; CCITT, 1992] basieren auf dem FSM Konzept (im Gegensatz dazu liegt Lotos eine Prozeßalgebra zugrunde). Diese Tatsache läßt erwarten, daß in Zukunft auch formale Spezifikationen als Modellbeschreibung für Markovmodelle eingesetzt werden. Ein erster Ansatz in dieser Richtung ist eine SDL-basierte, nichtexhaustive Markovanalyse [Bause und Buchholz, 1993].

2.2.2 Stochastische Prozeßalgebren

Prozeßalgebren stellen eine wichtige Klasse von formalsprachlichen Spezifikationstechniken zur Beschreibung nebenläufiger Systeme dar. Sie erlauben eine strukturierte Beschreibung und Analyse solcher Systeme und heben sich gegenüber den meisten konventionellen Modellierungstechniken dadurch ab, daß sie die erstrebenswerte Eigenschaft der Konstruktivität besitzen. Konstruktivität umfaßt in diesem Zusammenhang drei Aspekte: Der erste Aspekt ist die Tatsache, daß komplexe Beschreibungen aus einfacheren zusammengesetzt werden können (Komposition), wobei aus den Eigenschaften der zum Aufbau verwendeten Module auf die Eigenschaften des zusammengesetzten Systems geschlossen werden kann. Die Erstellung übersichtlicher zusammengesetzter Modelle wird ferner durch den zweiten Aspekt der Konstruktivität unterstützt, nämlich die Möglichkeit, mittels Ausblendung von solchen Aktionen, die für die Betrachtung des Gesamtsystems eine untergeordnete Rolle spielen (interne Aktionen), eine Abstraktion auf das Wesentliche zu erzielen. Zu den Wesensmerkmalen einer Algebra gehört schließlich auch der dritte Aspekt der Konstruktivität, nämlich das Vorhandensein algebraischer Gesetze (eines Kalküls), die das Umformen von Termen erlauben. In Prozeßalgebren kann die Äquivalenz von Prozeßausdrücken (Prozeßtermen) algebraisch charakterisiert, d.h. durch algebraische Regeln festlegt werden. Der Äquivalenzbegriff kann z.B. auf der Basis der möglichen Folgen von Aktionen zweier Prozeßterme definiert werden (Spurenäquivalenz).

Die Syntax einer Prozeßalgebra, d.h. die Menge der zulässigen Prozeßterme wird durch eine Grammatik festgelegt. Die Semantik von Prozeßausdrücken wird im allgemeinen definiert, indem deren sprachliche Beschreibung über semantische Regeln auf ein beschriftetes Transitionssystem (labelled transition system, LTS) abgebildet wird.

Die wichtigsten und am weitesten verbreiteten klassischen Prozeßalgebren sind wohl CSP [Hoare, 1985] und CCS [Milner, 1989]. Sie wurden entwickelt, um Prozeßsysteme zu beschreiben und deren qualitative (funktionale) Eigenschaften zu analysieren. Dies geschah insbesondere zum Zweck der Verifikation, d.h. man untersuchte, ob eine Implementierung bestimmte Eigenschaften wie z.B. Deadlockfreiheit besitzt und ob das Verhalten einer bestimmten Implementierung mit dem gewünschten Verhalten übereinstimmt.

Erst seit kurzer Zeit werden solche Prozeßalgebren entwickelt, die neben der Untersuchung qualitativer Eigenschaften auch für die Untersuchung von quantitativen Fragestellungen geeignet sind, also auch für die Leistungsbewertung. Dazu war vor allem die Einführung eines quantifizierbaren Zeitbegriffs (im Gegensatz zur rein logischen Zeit, die lediglich Reihenfolgebeziehungen kennt) notwendig. Ähnlich wie bei den endlichen Automaten wurde auch im Bereich der Prozeßalgebren zunächst versucht, mit deterministischen Zeiten zu operieren. Die nächste Stufe war dann die Betrachtung stochastisch verteilter Zeiten, was zum Begriff der stochastischen Prozeßalgebra (SPA) führte.

Eine solche stochastische Prozeßalgebra wurde an der Universität Erlangen-Nürnberg entwickelt. Sie trägt den Namen TIPP (Timed Processes and Performability Evaluation) und ist unter anderem in [Herzog, 1990; Rettelbach, 1991; Götz *et al.*, 1993] ausführlich dokumentiert. Wir wollen hier kurz den Sprachumfang einer Markov'schen, also einer auf exponentielle Zeitdauern eingeschränkten Version von TIPP wiedergeben. Die Menge aller zulässigen Terme der Sprache wird durch die folgende Grammatik definiert:

$$P ::= 0 \mid X \mid \alpha.P \mid P + P \mid P \parallel_S P \mid P\backslash L \mid recX : P$$

Die Symbole 0 und X bezeichnen den gestoppten Prozeß bzw. eine Prozeßvariable. Der Präfixoperator "." beschreibt mit $\alpha.P$ das Vorschalten einer Aktion vor einen Prozeß, wobei α von der Form (a, r) ist, also aus einem Aktionsnamen und zugehöriger Rate besteht. Mit dem Auswahloperator "$+$" wird ein Prozeß spezifiziert, der sich wahlweise wie der eine oder der andere seiner beiden Operandenprozesse verhalten kann. Mit dem Parallelоperator "$\|_S$" wird die parallele Ausführung zweier Prozesse beschrieben, wobei die Menge S alle diejenigen Aktionen enthält, die beide Partner synchron durchführen müssen. Durch die Anwendung des Ausblendeoperators "$\backslash L$" kann man erreichen, daß nicht alle Aktionen, die innerhalb eines Prozesses stattfinden, nach außen sichtbar werden. Die in der Menge L enthaltenen Aktionen werden für den externen Beobachter ausgeblendet. Schließlich kann rekursives Verhalten mit Hilfe des Rekursionsoperators rec spezifiziert werden.

Bisher existieren erst wenige Werkzeuge zur Unterstützung des Modellierungsvorgangs mit stochastischen Prozeßalgebren. Neben dem TIPP-Tool [Studt, 1995], das als Prototyp realisiert wurde, ist die ebenfalls prototypische PEPA-Workbench [Gilmore und Hillston, 1994] für die stochastische Prozeßalgebra PEPA [Hillston, 1994] zu erwähnen.

Im folgenden wird dargestellt, wie das in diesem Kapitel verwendete Multiprozessorbeispiel mit Hilfe von TIPP beschrieben werden kann. Zunächst wird die vereinfachte Version, d.h. eine Version ohne Breakdown und Repair, betrachtet, die durch die Prozeßdefinition in Abb. 2.6 beschrieben wird.

$$
\begin{aligned}
Processor &:= (work, \lambda).(grant\_bus, -).(release\_bus, \lambda).Processor \\
Multiprocessor &:= \underbrace{Processor \parallel_{\{\}} Processor \parallel_{\{\}} \ldots \parallel_{\{\}} Processor}_{N\ mal} \\
Bus\_Arbitrator &:= (grant\_bus, \lambda).(release\_bus, -).Bus\_Arbitrator \\
System &:= Multiprocessor \parallel_{\{grant\_bus, release\_bus\}} Bus\_Arbitrator
\end{aligned}
$$

Abbildung 2.6: TIPP-Beschreibung des Multiprozessorsystems
(vereinfachte Version ohne Breakdown und Repair)

Man erkennt in Abb. 2.6 den modularen Aufbau des Gesamtmodells *System* aus den parallel geschalteten Komponenten *Multiprocessor* und *Bus_Arbitrator,* wobei erstere wiederum als parallele Komposition mehrerer *Processor*-Komponenten definiert ist. Bei der Anwendung des Paralleloperators "||" für den Aufbau des Prozesses *Multiprocessor* sind die Synchronisationsmengen leer, d.h. es besteht keine direkte Synchronisation zwischen den einzelnen Prozessoren. Bei der Parallelschaltung von *Multiprocessor* und *Bus_Arbitrator* wird dagegen über die beiden Aktionen *grant_bus* und *release_bus* synchronisiert. Jede Aktion ist als Paar (a, r) von Aktionsname und Rate angegeben. Im Falle von $r = -$ handelt es sich um eine passive Aktion, d.h. der Zeitpunkt, zu dem diese Aktion eintritt, wird von der Partneraktion gleichen Namens bestimmt.

Das semantische Modell zu dem in Abb. 2.6 definierten Prozeßsystem ist in Abb. 2.8 dargestellt. Es handelt sich hierbei um ein beschriftetes Transitionssystem, das man erhält, wenn man auf das Prozeßsystem die operationalen semantischen Regeln von TIPP anwendet. Die Knoten des Transitionssystems enthalten Prozeßterme, die den Zustand des Prozeßsystems beschreiben. Die Kanten sind mit den Aktionen beschriftet, deren Eintreten das Fortschreiten des Prozeßsystems von einem Zustand in einen anderen bewirkt. Man beachte die (natürlich zu erwartende) strukturelle Übereinstimmung dieses Transitionssystems mit dem in Abb. 2.4 dargestellten Zustandsdiagramm.

Schließlich ist in Abb. 2.7 eine vollständige TIPP-Beschreibung des Multiprozessors mit Breakdown und Repair angegeben. Beim Studium dieser Beschreibung stellt man auch hier — ebenso wie bei der vereinfachten Version — eine genaue Übereinstimmung mit dem entsprechenden SAN-Modell fest: Würde man die zu den drei Prozessen *Processor*, *Bus_Arbitrator* und *Repairman* zugehörigen beschrifteten Transitionssysteme zeichnen, so erhielte man die drei stochastischen Automaten der Abb. 2.5.

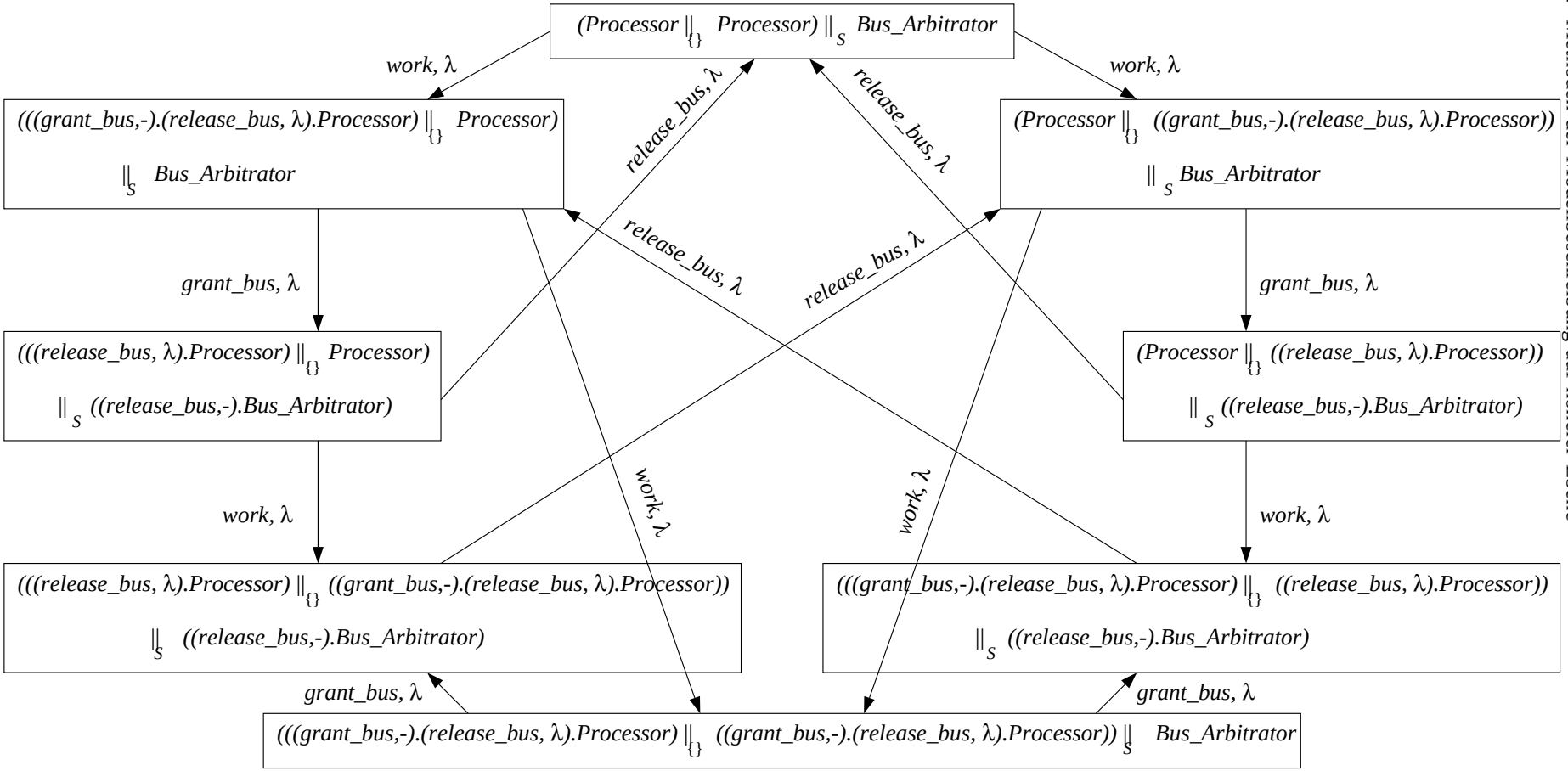


Abbildung 2.8: Semantisches Modell (beschriftetes Transitionssystem) zur prozeßalgebraischen Beschreibung des vereinfachten Modells für den Fall $N = 2$.

$$
\begin{aligned}
Processor :=& (work, \lambda).(grant\_bus\_l, -).Access \\
&+ (fail, \lambda).(start\_repair, -).(up, -).Processor \\
Access :=& (release\_bus\_l, \lambda).Processor \\
&+ (preempt, -).(grant\_bus\_l, -).Access \\
Multiprocessor :=& \underbrace{Processor \,\|_{\{\}} Processor \,\|_{\{\}} \ldots \|_{\{\}} Processor}_{N\ mal} \\
Bus\_Arbitrator :=& (grant\_bus\_l, \lambda).Bus\_Used + (grant\_bus\_h, \lambda).Bus\_Used \\
Bus\_Used :=& (release\_bus\_l, -).Bus\_Arbitrator \\
&+ (release\_bus\_h, -).Bus\_Arbitrator \\
&+ (up, -).Bus\_Arbitrator \\
Repairman :=& (start\_repair, \lambda) \\
&.((grant\_bus\_h, -).Downgrade + (preempt, \lambda).Downgrade) \\
Downgrade :=& (release\_bus\_h, \lambda).(repair, \lambda).(grant\_bus\_h, -).(up, \lambda) \\
&.Repairman \\
System :=& \Big(Multiprocessor \,\|_{\{grant\_bus\_l, release\_bus\_l, up\}} Bus\_Arbitrator\Big) \\
&\|_{\{grant\_bus\_h, release\_bus\_h, preempt, start\_repair, up\}} Repairman
\end{aligned}
$$

Abbildung 2.7: TIPP-Beschreibung des Multiprozessorsystems mit Breakdown und Repair

Die beiden Modellierungsmethoden SAN und SPA sind tatsächlich sehr nah verwandt. Beide erlauben eine modulare Modellbeschreibung und unterscheiden zwischen lokalen und synchronisierenden Ereignissen (Aktionen). Um die Mächtigkeit beider Methoden zu vergleichen, kann man folgende Überlegung anstellen: Jeder stochastische Automat kann auch durch einen Prozeßterm ausgedrückt werden. Umgekehrt kann jeder Operator der stochastischen Prozeßalgebra, mit Ausnahme des Ausblendeoperators, durch ein entsprechendes Konstrukt im SAN-Formalismus nachgebildet werden. Es gibt aber durchaus Unterschiede zwischen den beiden Methoden. Während stochastische Automaten durch ihre graphische Darstellung dem Benutzer eine anschauliche Darstellung des Verhaltens bieten, liegt der Vorteil der stochastischen Prozeßalgebren in deren formalsprachlicher Beschreibung und in dem Vorhandensein eines sehr mächtigen Kalküls, der zur Modellvereinfachung und zum Nachweis der Äquivalenz zwischen Prozessen benutzt werden kann.

2.2.3 Stochastische Petrinetze

In den Abschnitten 2.2.1 und 2.2.2 wurde festgestellt, daß es sich bei den stochastischen Automaten und den stochastischen Prozeßalgebren um Modellierungsmethoden handelt, bei denen die Modelldarstellung auf höherer Ebene sehr stark die Struktur der darunterliegenden Markovkette widerspiegelt. Im Gegensatz dazu erlauben stochasti-

sche Petrinetze eine abstraktere Darstellung. Die möglichen Zustände eines Petrinetzes sind nicht unmittelbar aus der Netzstruktur ersichtlich, sondern durch die erreichbaren Netzmarkierungen verteilt definiert.

Eine gute Übersicht über die historische Entwicklung von Petrinetzen findet man in [Murata, 1989]. In der Dissertation von Petri [Petri, 1962] wurden erstmals zeitlose Netze beschrieben, die bis heute zur Untersuchung des funktionalen Verhaltens von Systemen eine große Rolle spielen. Für qualitative Untersuchungen steht inzwischen eine umfangreiche Theorie zur Verfügung, siehe z.B. [Starke, 1990].

Um Leistungsbewertung mit Petrinetzen zu ermöglichen, wurden zunächst — ähnlich wie bei den endlichen Automaten — deterministische Zeiten eingeführt [Sifakis, 1977] (weitere Referenzen in [Murata, 1989, S. 570]). Der entscheidende Schritt war aber die Entwicklung stochastischer Petrinetze (stochastic Petri nets, SPN) [Natkin, 1980; Molloy, 1981; Molloy, 1982], bei denen den Transitionen des Netzes eine Schaltrate zugeordnet ist. Diese stellt den Parameter der exponentiell verteilten Schaltzeit dar, die zwischen dem Erfülltsein der Schaltbedingung und dem tatsächlichen Schalten einer Transition vergeht. Die Schaltdisziplin (firing policy) bestimmt was geschieht, wenn mehrere Transitionen gleichzeitig schaltbereit sind. Gewöhnlich wird angenommen, daß diejenige Transition schaltet, deren stochastische Schaltzeit zuerst zu Ende ist (race policy).

Die Verteilung von Marken (token) über die Stellen definiert den Zustand eines Petrinetzes. Alle von der Anfangsmarkierung durch das Schalten von Transitionen erreichbaren Markierungen bilden die Erreichbarkeitsmenge des Petrinetzes. Berücksichtigt man auch noch die Übergangsraten zwischen den erreichbaren Markierungen, so gelangt man zum Erreichbarkeitsgraphen des Netzes, welcher isomorph zur zugrundeliegenden Markovkette ist. Durch die Ermittlung der stationären Verteilung der Markovkette erhält man also die den erreichbaren Netzmarkierungen entsprechenden Wahrscheinlichkeiten. Daraus lassen sich dann leicht die verschiedenen typischen Leistungsgrößen des Netzes berechnen, wie die mittlere Anzahl von Marken in einer Stelle oder der Durchsatz einer Transition bzw. eines Subnetzes.

Einen wichtigen Netztyp stellen die sogenannten verallgemeinerten stochastischen Petrinetze (generalized stochastic Petri nets, GSPN) dar, welche von Ajmone Marsan et al. in [Ajmone Marsan *et al.*, 1984] definiert wurden. Das wesentliche Merkmal dieses Netztyps ist die Unterscheidung von zwei Arten von Transitionen: Es gibt zum einen zeitbehaftete Transitionen, welche mit einer exponentiell verteilten Verzögerungszeit versehen sind, und zum andern zeitlose Transitionen, welche sofort schalten, sobald sie aktiviert sind. Die zeitlosen Transitionen spielen eine wichtige Rolle bei der Erstellung kompakter Modelle. Obwohl die Mächtigkeit der modellierbaren Systeme durch die Zuhilfenahme von zeitlosen Transitionen nicht zunimmt, da sowohl mit SPNs als auch mit GSPNs jede beliebige Markovkette beschrieben werden kann, sind sie ein wichtiges Hilfsmittel zur knappen und überschaubaren Darstellung von in der Praxis oft auftretenden Phänomenen wie Synchronisation, Verzweigung, Fork-Join, etc..

Der Erreichbarkeitsgraph (reachability graph) eines GSPN enthält stabile (tangible) und verschwindende (vanishing) Markierungen. Stabile Markierungen sind solche Markierungen, in denen nur zeitbehaftete Transitionen schaltfähig sind. Im Gegensatz dazu

kann in einer verschwindenden Markierung mindestens eine zeitlose Transitionen schalten. Verschwindende Markierungen werden sofort ohne Verzögerung wieder verlassen und stellen daher Zwischenzustände dar, die bei der Markovanalyse keine Rolle spielen. Die durch die Verwendung zeitloser Transitionen entstehenden verschwindenden Markierungen tragen also nicht zur Vergrößerung der Zustandszahl der Markovkette bei. Zur Erzeugung der dem GSPN-Modell zugrundeliegenden Markovkette ist aber außer der Erstellung des Erreichbarkeitsgraphen zusätzlich die Eliminierung der verschwindenden Markierungen aus dem Erreichbarkeitsgraphen notwendig [Ajmone Marsan *et al.*, 1984; Ciardo *et al.*, 1991]. Dazu wird folgende Betrachtung angestellt: Die Matrix U, welche die Übergänge zwischen den Markierungen des Erreichbarkeitsgraphen beschreibt, kann in vier Blöcke aufgeteilt werden.

$$U = \begin{bmatrix} P^{VV} & P^{VT} \\ Q^{TV} & Q^{TT} \end{bmatrix}$$

Dabei sind die Matrizen Q^{TT} und Q^{TV} Ratenmatrizen, die den Übergang zwischen zwei stabilen Zuständen, bzw. von einem stabilen in einen verschwindenden Zustand beschreiben. Die Matrizen P^{VV} und P^{VT} enthalten die Wahrscheinlichkeiten für die Übergänge zwischen zwei verschwindenden, bzw. von einem verschwindenden in einen stabilen Zustand. In Abhängigkeit davon, ob es zeitlose Schleifen gibt, d.h. Pfade, die von einem verschwindenden Zustand zu diesem zurückführen, ohne dabei einen stabilen Zustand zu berühren, definiert man die Matrix G^{∞} wie folgt[1].

$$G^{\infty} = \begin{cases} \left(\sum_{i=0}^{k_0} \left(P^{VV}\right)^i \right) P^{VT} & \text{ohne zeitlose Schleifen} \\ \left(\sum_{i=0}^{\infty} \left(P^{VV}\right)^i \right) P^{VT} = \left(I - P^{VV}\right)^{-1} P^{VT} & \text{mit zeitlosen Schleifen} \end{cases}$$

Im ersten Fall bezeichnet k_0 die maximale Anzahl aufeinanderfolgender verschwindender Zustände. Damit erhält man für die Übergangsmatrix der zugrundeliegenden Markovkette folgende Beziehung

$$Q = Q^{TT} + Q^{TV} G^{\infty}$$

In der Praxis können außer diesem Verfahren auch noch andere Methoden eingesetzt werden, bei denen die verschwindenden Markierungen bereits während der Aufstellung des Erreichbarkeitsgraphen eliminiert werden, siehe Abschnitt 3.4.

---

[1] Geht man von einem irreduziblen stochastischen Prozeß aus, so ist der Spektralradius von P^{VV} kleiner als eins [Varga, 1962, S. 31]. Daraus folgt $\lim_{k \to \infty} \left(P^{VV}\right)^k = 0$, und damit die Existenz von $\lim_{k \to \infty} \sum_{i=0}^{k} \left(P^{VV}\right)^i$.

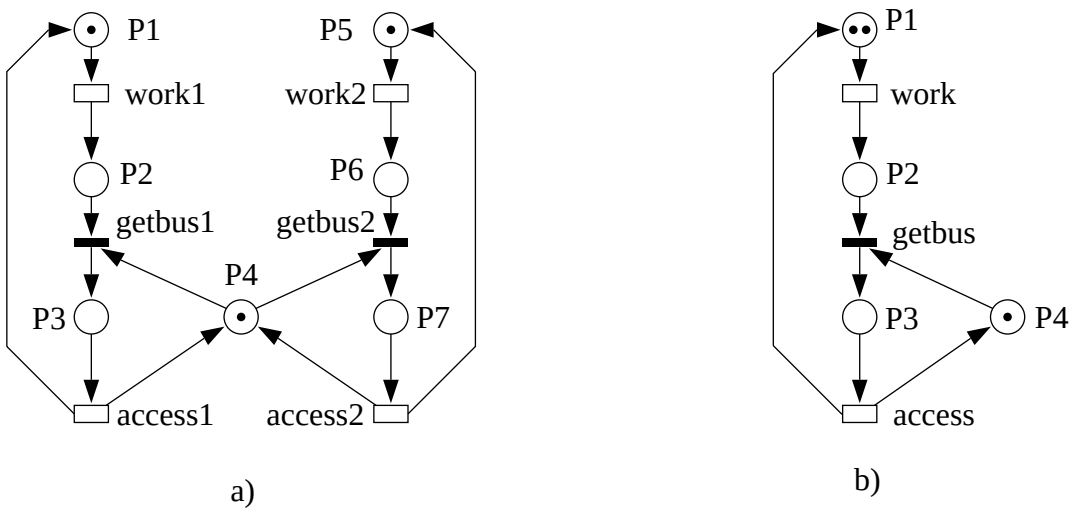


Abbildung 2.9: GSPN-Modell des Multiprozessors
(vereinfachte Version ohne Breakdown und Repair)
a) explizite Darstellung zweier Prozessor-Subnetze b) kompakte Darstellung

In Abbildung 2.9 sind als Beispiele für Netze vom Typ GSPN zwei Modelle des Multiprozessors in der vereinfachten Version ohne Breakdown und Repair dargestellt. Beide Netze a) und b) beschreiben ein Modell mit zwei Prozessoren. In Abbildung 2.9 a) wird jeder Prozessor durch ein eigenes Subnetz repräsentiert. Ein Token in der Stelle *P1* (*P5*) stellt den arbeitenden Prozessor 1 (2) dar. Nach dem Schalten der Transition *work1* (*work2*) ist Prozessor 1 (2) bereit, auf den Bus zuzugreifen. Dieser wird durch das Token in *P4* repräsentiert, welches für das Schalten der zeitlosen Transition *getbus1* (*getbus2*) benötigt wird. Nach Beendigung des exklusiven Zugriffs, d.h. nach dem Schalten der Transition *access1* (*access2*) kehrt Prozessor 1 (2) wieder in die Ausgangsposition *P1* (*P5*) zurück, und das Bus-Token liegt wieder in der Stelle *P4*.

In Abbildung 2.9 b) wurde dagegen eine Darstellung gewählt, bei der die beiden Prozessoren durch zwei ununterscheidbare Tokens repräsentiert werden. Für die Transition *work* gilt in diesem Fall eine sogenannte “infinite-server” Semantik, damit ihre Schaltrate proportional zur der Anzahl der Tokens in Stelle *P1* ist. Neben dem Vorzug der kompakteren graphischen Darstellung bietet die Version in Abbildung 2.9 b) eine leichtere Skalierbarkeit des Modells bezüglich der Anzahl von Prozessoren. Diese kann in einfacher Weise über die Anzahl der Tokens in Stelle *P1* spezifiziert werden, während in Abbildung 2.9 a) für jeden Prozessor ein zusätzliches Subnetz bestehend aus 3 Stellen und 3 Transitionen benötigt wird. Auf einen weiteren wichtigen Vorteil der Version b) wird in Abschnitt 2.4 eingegangen. Im Vorgriff darauf sei aber bereits hier gesagt, daß das Modell b) eine geringere Zustandszahl besitzt als das Modell a). Man erkauft sich diese Vorteile der Version b) mit dem Verlust der Unterscheidbarkeit der Prozessoren.

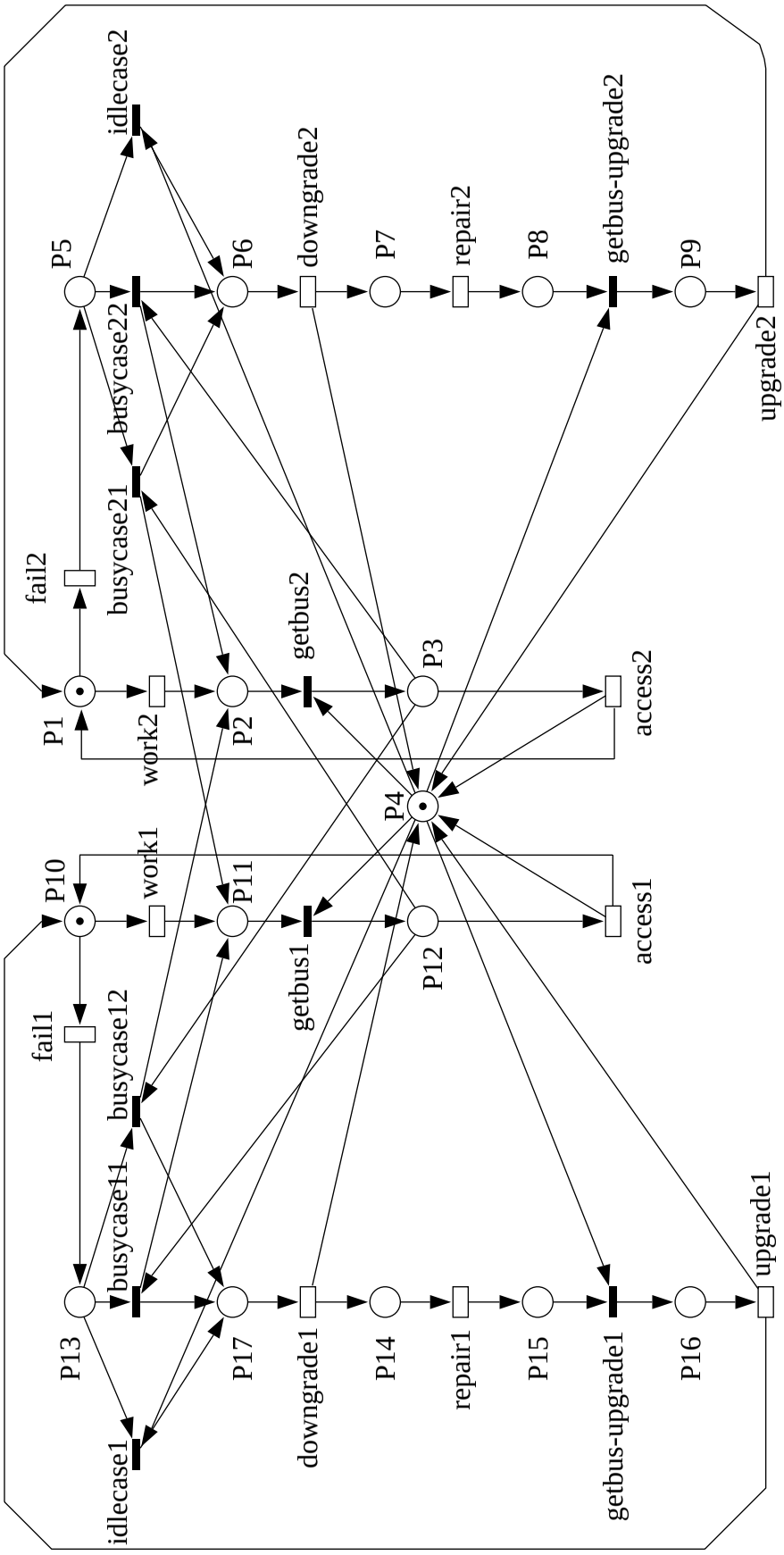


Abbildung 2.11: GSPN-Modell des Multiprozessors mit Breakdown und Repair, explizite Darstellung zweier Prozessor-Subnetze

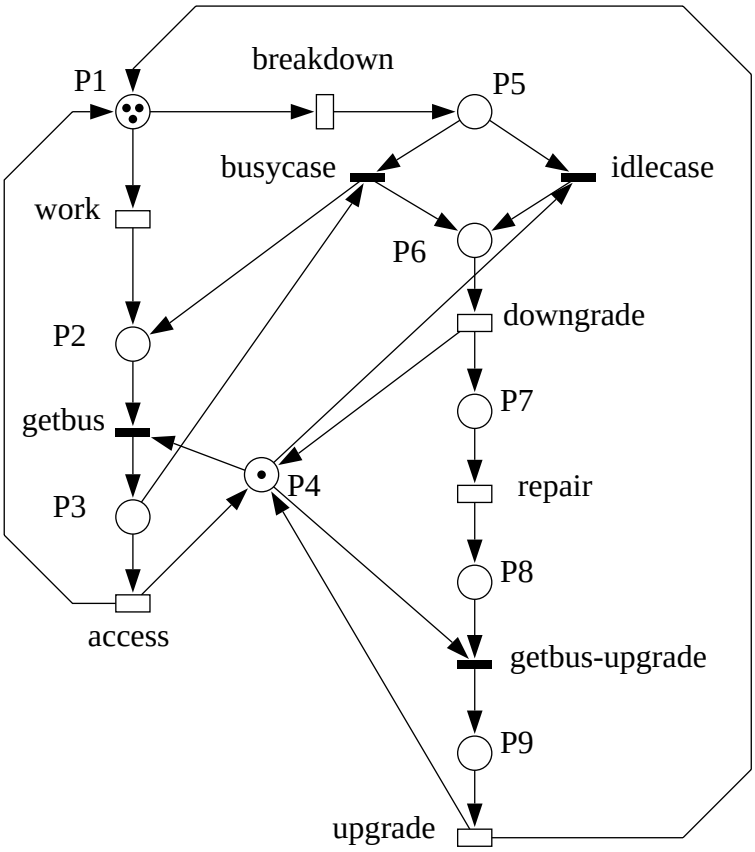


Abbildung 2.10: GSPN-Modell des Multiprozessors mit Breakdown und Repair, kompakte Darstellung

In Abbildung 2.11 und Abbildung 2.10 sind zwei GSPN-Modelle für das rekonfigurierbare Multiprozessorsystem dargestellt, d.h. Modelle die gegenüber denjenigen aus Abb. 2.9 um Breakdown und Repair erweitert sind. Abbildung 2.11 stellt ein System mit zwei Prozessoren dar und benutzt eine explizite Darstellung der beiden Prozessoren durch zwei Subnetze, während in Abbildung 2.10 wiederum die kompakte Darstellung gewählt wurde. Beim Vergleich dieser beiden Abbildungen fällt zunächst die wesentlich bessere Übersichtlichkeit des kompakten Modells auf.

Wir wollen uns auf eine Erläuterung des kompakten Modells in Abbildung 2.10 beschränken (es hat Ähnlichkeit mit einem Modell aus [Ajmone Marsan und Chiola, 1987]). Die sich in der Stelle *P1* befindlichen Tokens repräsentieren arbeitende Prozessoren (in der Abbildung sind dies drei Prozessoren), d.h. dieses Modell ist im Gegensatz zu dem aus Abb. 2.11 auf einfache Weise skalierbar bezüglich der Anzahl der Prozessoren. Nach dem Schalten der Transition *work* ist ein Prozessor bereit, auf den Bus zuzugreifen. Dieser wird durch das Token in *P4* repräsentiert, welches für das Schalten der zeitlosen Transition *getbus* benötigt wird. Nach Beendigung des exklusiven Zugriffs, d.h. nach dem Schalten der Transition *access* kehrt der Prozessor wieder in die Ausgangsposition *P1* zurück, und das Bus-Token liegt wieder in der Stelle *P4*. Der Ausfall eines arbeitenden Prozessors wird durch das Schalten der Transition *breakdown* modelliert. Sofort nach dem Ausfall wird der freie Bus belegt (Transition *idlecase*), oder — falls gerade ein Prozessor den Bus benutzt — dem Zugreifer der Bus entzogen

und dieser in die Warteposition *P2* zurückversetzt (Transition *busycase*). Darauf folgt das Umkonfigurieren des Systems während der Schaltzeit der Transition *downgrade*. Anschließend wird der Bus wieder freigegeben, so daß das System während der Reparatur des ausgefallenen Prozessors weiterarbeiten kann. Für das Rückkonfigurieren (Transition *upgrade*) wird ebenfalls der Bus benötigt.

Man erkennt in diesem Beispiel die typische Verwendung der zeitlosen Transitionen. Zum einen werden sie als Ausgangstransitionen von Wartepositionen verwendet (Transitionen *getbus* und *getbus-upgrade*), zum anderen repräsentieren sie Entscheidungs- bzw. Verzweigungsmöglichkeiten (Transitionen *idlecase* und *busycase*). Deutlich ist an diesem Beispiel auch zu sehen, daß Petrinetze zunächst wenig Möglichkeiten zur Strukturierung einer Modellbeschreibung bieten. Die Prozessoren, der Bus-Arbitrator und der Repairman sind in dem Modell von Abb. 2.10 nicht als in sich selbst abgeschlossene Komponenten identifizierbar, sondern vollkommen miteinander verwoben.

Zu den neueren Entwicklungen im Bereich der Petrinetze gehören farbige Netze, allgemein auch als High-level Petri Nets bezeichnet [Jensen und Rozenberg, 1991], die eine kompaktere Darstellung erlauben und durch die Verwendung von typisierten Tokens zur Strukturierung von Modellen beitragen. Zeitbehaftete Vertreter dieser Klasse von Netzen sind die Stochastic Well Formed Nets, für die auch spezielle, teilweise sehr effiziente Analysemethoden existieren (siehe Abschnitte 3.3 und 3.6).

Unter den zahlreichen petrinetzbasierten Tools für die Leistungsbewertung seien hier nur GreatSPN [Chiola, 1992], SPNP [Ciardo und Muppala, 1991] und DSPNexpress [Lindemann, 1992] genannt. Letzteres unterstützt Netze, die neben zeitlosen und exponentiellen Transitionen auch Transitionen mit deterministischer Verzögerungszeit enthalten, wobei die Auswertealgorithmen momentan auf solche Netze beschränkt sind, bei denen in keiner erreichbaren Markierung mehr als eine deterministische Transition schaltfähig ist.

2.2.4 Warteschlangenmodelle

Warteschlangenmodelle gehören zu den ältesten Modellierungsmethoden und wurden bereits in den sechziger Jahren für die Leistungsbewertung von Rechensystemen eingesetzt. Klassische Anwendungen sind neben der Betrachtung einzelner Bedienstationen z.B. Time-sharing Computersysteme, insbesondere das Central-Server-Modell. Eine allgemeine Einführung findet man z.B. in [Bolch, 1989] und in [Kleinrock, 1975], Anwendungen u.a. aus dem Bereich der Multiuser Computersysteme und der Kommunikationsnetze sind in [Kleinrock, 1976] beschrieben.

Die Grundelemente, aus denen eine Station besteht, sind eine oder mehrere Bedieneinheiten (server) und der Warteraum (queue). Die Bedienstrategie (service discipline) gibt an, in welcher Reihenfolge die in der Warteschlange befindlichen Aufträge (customer) bedient werden. Die Ankunft von Aufträgen an einer Station, d.h. die statistische Verteilung der Zwischenankunftszeiten, wird durch den Ankunftsprozeß beschrieben. Falls die Zwischenankunftszeiten exponentiell verteilt sind, spricht man von einem Poissonprozeß. Auch die Bedienzeit für die verschiedenen Klassen von Aufträgen wird in Form einer Verteilung angegeben. Je nachdem, ob man Warteschlangen mit endlicher oder

unendlicher Population betrachtet ist auch der zugrundeliegende Zustandsraum endlich oder unendlich.

Komplexe Systeme werden mit Hilfe von Warteschlangennetzen (WSN) modelliert, die aus mehreren Stationen bestehen, und in denen neben der Beschreibung der Stationen die Wegewahl (routing) der Aufträge eine wichtige Rolle spielt [Gelenbe und Mitrani, 1980]. Man unterscheidet zwischen offenen und geschlossenen Netzen, wobei sich bei geschlossenen eine konstante endliche Anzahl von Aufträgen im System befindet, also keine externen Ankünfte und Abgänge erlaubt sind (Systeme mit externen Ankünften und Blockierverhalten, können manchmal im Grenzfall als geschlossene Netze dargestellt werden). Wichtige Arbeiten auf diesem Gebiet wurden von Jackson [Jackson, 1957; Jackson, 1963] (offene Netze), Gordon und Newell [Gordon und Newell, 1967] (geschlossene Netze, erfordert die Bestimmung einer Normalisierungskonstanten) und den Autoren des berühmten BCMP-Theorems [Baskett *et al.*, 1975] geleistet.

Für die Analyse von sogenannten Produktformnetzen ist es nicht notwendig, den Zustandsraum explizit aufzubauen. Produktform bedeutet, daß sich die Gesamtlösung aus dem Produkt der Einzellösungen für jede Station zusammensetzt, die man durch Betrachtung der einzelnen Stationen in Isolation gewinnt, wobei lediglich vom Gesamtsystem abhängige Parameter berücksichtigt werden müssen. Die wichtigsten effiziente Algorithmen für die Analyse von Produktformnetzen sind der Algorithmus von Buzen zur Bestimmung der Normalisierungskonstanten in geschlossenen Netzen [Buzen, 1973], die parametrische Analyse [Chandy *et al.*, 1975b; Chandy *et al.*, 1975a] und die Mittelwertanalyse (mean value analysis, MVA) [Reiser und Lavenberg, 1980]. Eine gute Übersicht über diese und weitere Verfahren gibt [Conway und Georganas, 1989]. Dort wird auch gezeigt, daß alle derartigen Algorithmen auf dem allgemeinen Prinzip von Dekomposition und Aggregierung nach Simon und Ando [Simon und Ando, 1961] bzw. Courtois [Courtois, 1977] beruhen, siehe auch Abschnitt 3.5.1.

Es existiert eine große Anzahl von Tools für die computergestützte Modellierung mit Warteschlangennetzen. Davon seien exemplarisch QNAP2 [Veran und Potier, 1984], RESQ [Gordon *et al.*, 1988] und PEPSY [Bolch *et al.*, 1995] erwähnt, welche neben analytischen Lösungsverfahren auch Simulationstechniken zur Verfügung stellen.

Das Beispiel des Multiprozessorsystems mit Repair und Breakdown, welches bereits in den vorangegangenen Abschnitten diskutiert wurde, soll nun auch im Zusammenhang mit Warteschlangenmodellen betrachtet werden. Ein einfaches Modell des Multiprozessorsystems ist in Abb. 2.12 dargestellt, in welcher die N Prozessoren durch Stationen ohne Warteschlange repräsentiert sind. Der Zugriff über den Bus auf den Speicher wird durch die Station *Memory Access* modelliert. Ein Prozessor, der den Bus zum Zugriff auf den Speicher benutzen möchte, muß sich in die Warteschlange dieser Station einreihen, welche nach der FIFO-Disziplin abgearbeitet wird. Nach erfolgter Bedienung in der Zugriffsstation kehrt der Prozessor zu seiner Ausgangsstation zurück. Auf entsprechende Weise wird auch der Ausfall eines Prozessors und seine Reparatur in der Station *Repair* nachgebildet. Ein wichtiger Aspekt des realen Systems, nämlich die Benutzung des Busses für die Rekonfigurierung des Multiprozessorsystems am Anfang und am Ende einer Reparatur und das dabei eventuell auftretende Entziehen des Busses, wird mit dem Warteschlangenmodell in Abb. 2.12 nicht erfaßt. Allgemein kann gesagt

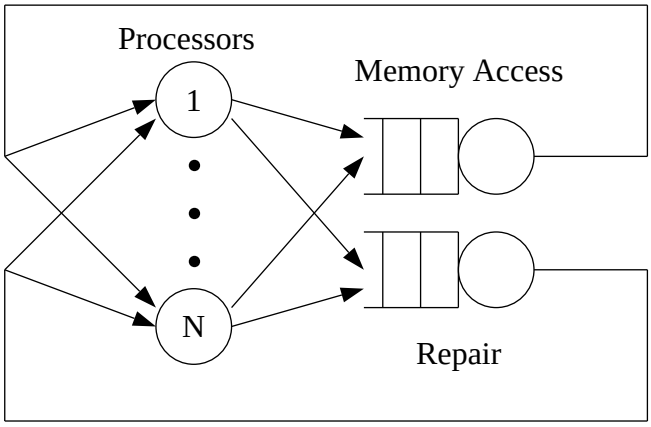


Abbildung 2.12: Warteschlangenmodell des Multiprozessorsystems mit Breakdown und Repair (ohne Berücksichtigung von Buszugriffen durch den Repairman)

werden, daß es grundsätzlich nur schwer möglich ist, mit Warteschlangenmodellen die gemeinsame Nutzung von Ressourcen zu modellieren.

Mit einem zweiten Warteschlangenmodell, welches in Abb. 2.13 dargestellt ist, sollen die Mängel des einfachen Warteschlangenmodells behoben werden. In diesem erweiterten Modell werden zunächst zwei Warteschlangen, eine für die Speicherzugriffswünche und eine für die Reparaturwünsche, von einer gemeinsamen Bedieneinheit, welche den Bus darstellt, bearbeitet. Die Warteschlange für die Reparaturaufträge wird dabei vorrangig bedient (Priorität 2), und Reparaturauftäge sind sogar in der Lage, die Zugriffsaufträge zu verdrängen. Ein Reparaturauftag begibt sich nach der ersten Bearbeitung in der Bedieneinheit *Bus* zur eigentlichen Reparaturstation *Repair*. Danach kehrt er in einen dritten Warteraum vor der Station *Bus* zurück, welcher mit höchster Priorität (Priorität 3) bedient wird, allerdings ohne die Möglichkeit, daß dabei ein Speicherzugriffsauftrag verdrängt wird, da Verdrängung nur beim Downgrade, nicht jedoch beim Upgrade stattfindet. Erst nach diesem zweiten Durchlaufen der Station *Bus* kehrt der Reparaturauftag zu der auslösenden Prozessorstation zurück.

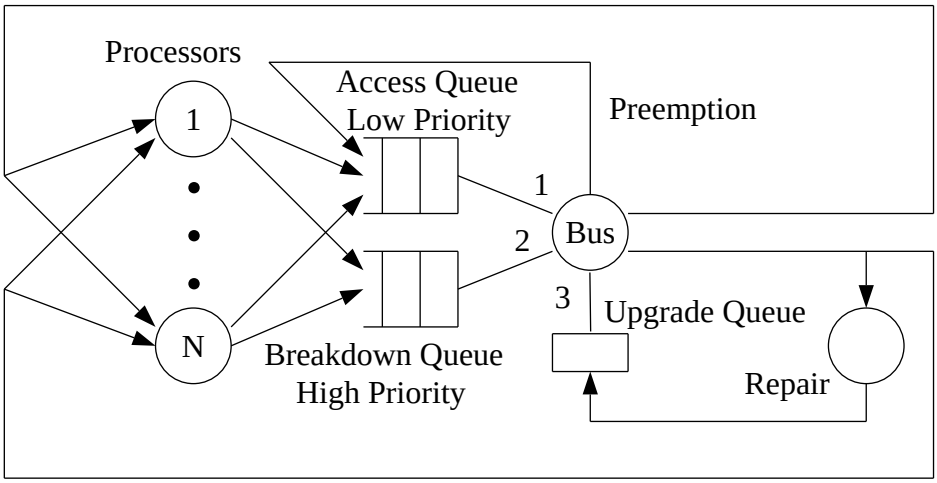


Abbildung 2.13: Warteschlangenmodell des Multiprozessorsystems mit Breakdown und Repair

Es gibt beim Verdrängen von Aufträgen zwei Regeln, je nachdem ob die vor dem Entzug des Betriebsmittels bereits geleistete Bedienzeit einem Auftrag angerechnet wird (preemption resume) oder nicht (preemption restart). Bei exponentiellen Bedienzeiten besteht aber wegen der Gedächtnislosigkeit der Exponentialverteilung kein Unterschied zwischen diesen beiden Strategien.

Die Tatsache, daß ein Reparaturauftrag genau einmal die Schleife mit der *Repair*-Station durchläuft, bevor er zum auslösenden Prozessor zurückkehrt, kann man leicht über verschiedene Auftragsklassen, entsprechende Klassenwechsel und Routingwahrscheinlichkeiten ausdrücken. Schwierigkeiten ergeben sich allerdings durch die Forderung, daß zu jedem Zeitpunkt nur ein Reparaturauftrag bearbeitet werden soll, d.h. Aufträge in der *Breakdown-Queue* sollen erst dann berücksichtigt werden, wenn die Repair-Schleife leer ist. Dieser Sachverhalt läßt sich mit einem klassischen Warteschlangenmodell nicht ausdrücken, sondern erfordert die Einführung von Zusatzkonstrukten, wie z.B. die Verwendung einer globalen Statusvariablen.

Wir stellen also fest, daß die graphische Darstellungsform der Warteschlangennetze lediglich eine semi-formale Beschreibung von Systemen erlaubt. Es gibt keine einheitliche Notation, die in der Lage ist, alle relevanten Aspekte des Systems formal auszudrücken. So müssen in vielen Fällen Zusatzangaben z.B. über Prioritäten, Synchronisationen oder spezielle Vorschriften bezüglich des Routing von Aufträgen gemacht werden.

Zu den Vorzügen der Warteschlangenmodelle gehören in erster Linie die sehr effizienten Analysetechniken für Produktformnetze. Außerdem haben Warteschlangenmodelle aufgrund ihrer langen Tradition eine größere Verbreitung gefunden als dies bei den anderen hier diskutierten Modellbeschreibungsmethoden bisher der Fall ist. Dies spiegelt sich auch in einer Fülle von Werkzeugen wider.

2.2.5 Weitere Methoden für die Modellbeschreibung

Die in den vorangegangenen Abschnitten diskutierten Formalismen sind nur ein Ausschnitt aus dem umfangreichen Spektrum von Beschreibungsmethoden für Markovmodelle. Exemplarisch für alle anderen Methoden seien nun noch kurz stochastische Graphmodelle und sprachliche Beschreibungen erwähnt.

Stochastische Graphmodelle beschreiben Vorgänger-Nachfolgerbeziehungen in sich abgeschlossener Teilaufgaben und sind daher gut für die Modellierung paralleler Programme in Form von Taskgraphen geeignet. Sie bieten aber wenig Flexibilität bei der Modellierung anderer Aspekte allgemeiner dynamischer Abläufe, wie z.B. Ressourcenbelegung oder bedingte Verzweigungen. Für stochastische Graphmodelle wurden an der Universität Erlangen-Nürnberg effiziente exakte und approximative Auswerteverfahren entwickelt und im Tool PEPP [Hartleb und Quick, 1993] implementiert. Diese Modelle werden zur Leistungsbewertung von parallelen Programmen eingesetzt [Fleischmann und Werner, 1989; Sötz und Werner, 1990].

Zur Beschreibung von Markovmodellen auf einer zustandsnahen Ebene wurden verschiedene spezialisierte Sprachen entwickelt. Als Beispiele seien hier die Eingabesprachen für die Werkzeuge MARCA [Stewart, 1991], MOSES [Bolch *et al.*, 1995; Jung, 1991] und USENUM [Sczittnick, 1987] genannt.

2.3 Vergleich der Methoden für die Modellbeschreibung

In diesem Abschnitt werden wir die verschiedenen Modellierungsmethoden einander gegenüberstellen. Das Ziel des Vergleichs ist es, neben den Gemeinsamkeiten vor allem die Unterschiede, also die Vor- und Nachteile der einzelnen Methoden zu beleuchten. Insbesondere soll herauspräpariert werden, wo Ansatzpunkte für ein effizientes Vorgehen bei der Modellerstellung und auch im Hinblick auf die nachfolgende Modellanalyse liegen. Aus dem Vergleich sollen sich Anforderungen an neue Modellierungsmethoden bzw. Erweiterungen von existierenden Methoden um neue Techniken ableiten.

Eine Übersicht und Vergleich von Spezifikationsmethoden für Markov-Reward Modelle haben bereits Haverkort und Trivedi gegeben [Haverkort und Trivedi, 1993]. In [Herzog, 1990] werden verschiedene Modellierungstechniken für verteilte Systeme in einem allgemeineren Kontext als dem der Markovmodelle verglichen. Übersichten über Werkzeuge, also nicht Methoden, findet man z.B. in [Schroetter und de Meer, 1991] und [Kauer, 1992].

2.3.1 Vergleichskriterien

Zum Vergleich wollen wir die folgenden Kriterien heranziehen, die sich in vier grundlegende (g_i) und zwei praktisch orientierte Kriterien (p_i) untergliedern lassen:

g_1 Abstraktionsgrad: Auf welcher Ebene findet die Modellbeschreibung statt? Nahe am Bewertungsproblem oder nahe an der Markovkette? Wenn die Beschreibung nahe an der Markovkette angelehnt ist, wollen wir von einem geringen Abstraktionsgrad sprechen. Ein hoher Abstraktionsgrad verdeckt die zugehörige Markovkette, zeichnet sich also durch größere Problemorientiertheit aus, welche die Benutzernähe eines Modells fördert. Ein zweites Kriterium, das eng mit dem Abstraktionsgrad zusammenhängt, ist die Frage, ob die Modellbeschreibung zustandsorientiert (state oriented) erfolgt, oder ob zur Beschreibung andere, nicht direkt mit dem Zustandsbegriff zusammenhängende Mittel zur Verfügung stehen.

g_2 Strukturiertheit: Hier betrachten wir vier Unterpunkte:

a) Unterstützt die Methode eine modulare Systembeschreibung, also den Aufbau eines Gesamtmodells aus in sich abgeschlossenen Submodellen?
b) Ist eine hierarchische Modellierung möglich?
c) Im Hinblick auf moderne parallele und verteilte Systeme mit unter Umständen vielen gleichartigen Komponenten stellt sich die Frage, ob die Replikation von Komponenten auf einfache Weise ohne großen Aufwand beschreibbar ist. Besonders für die Modellierung skalierbarer Systeme (z.B. Multiprozessorsysteme mit variabler Anzahl von Prozessoren) ist dies von Bedeutung.
d) Schließlich interessiert, soweit wir dies an dieser Stelle schon beurteilen können, ob eventuell vorhandene Strukturmerkmale auch bei der anschließenden Modellanalyse ausgenutzt werden können. Gibt es effiziente Analyseverfahren, die von der Strukturiertheit Gebrauch machen?

g_3 Formalisierungsgrad: Kann ein Modell mit der jeweiligen Modellierungsmethode vollständig formal beschrieben werden, oder sind informelle Zusatzangaben not-

wendig? Dieser Punkt hängt eng mit dem folgenden zusammen, da eine formale Basis Voraussetzung für die Entwicklung eines Kalküls ist.

g_4 Vorhandensein eines Kalküls: Für welche der Modellierungsmethoden existiert ein Kalkül, bestehend aus einer Menge von Axiomen und Arbeitsregeln, der es erlaubt, die Äquivalenz zwischen Modellen festzustellen und Äquivalenzumformungen an Modellen oder Teilmodellen vorzunehmen? Im Zusammenhang damit stellt sich auch die Frage, ob außer der Leistungsbewertung auch die qualitative Analyse von Modellen unterstützt wird.

g_5 Universalität: Sind alle gebräuchlichen in parallelen und verteilten Systemen auftretenden Phänomene auf adäquate Weise beschreibbar? Werden alle wünschenswerten Leistungsmaße unterstützt? Ist die Modellierungsmethode allgemein anwendbar, also nicht auf ein spezielles Anwendungsgebiet spezialisiert?

p_1 Verbreitung: Die Verbreitung einer Modellierungsmethode hat sicherlich zum Teil damit zu tun, wie lang diese schon bekannt ist. Sie läßt aber auch auf die Akzeptanz durch die Anwender (Systementwickler, Systemanalytiker) und damit auf die Benutzerfreundlichkeit einer Methode schließen.

p_2 Werkzeugunterstützung: Dieser Punkt ist nah verwandt mit dem vorhergehenden. Er ist für die Praxis von entscheidender Bedeutung, da Leistungsbewertung ohne die Unterstützung durch hochentwickelte Werkzeuge nicht durchführbar ist.

2.3.2 Gegenüberstellung der Methoden

Eine Zusammenfassung der in den folgenden Abschnitten enthaltenen Vergleichsergebnisse zeigt Tab. 2.1.

Abstraktionsgrad (g_1)

Bei stochastischen Automaten stehen zur Modellbeschreibung nur Zustände und Zustandsübergänge zur Verfügung. Sie gehören damit ganz eindeutig zu den zustandsorientierten Modellbeschreibungsmethoden. Die Modellbeschreibung auf höherer Ebene ist bei SANs sehr stark an der durch sie beschriebenen Markovkette orientiert, eine starke Abstraktion davon und somit eine hohe Problemorientiertheit ist mit ihnen nicht möglich. Besteht ein Modell z.B. aus einem einzigen Automaten, so entspricht dieser der graphischen Darstellung der Markovkette.

Trotz der im Beispiel bereits festgestellten Nähe der beiden Beschreibungsmethoden erlauben stochastische Prozeßalgebren im Vergleich mit stochastischen Automaten eine Darstellung, die stärker von der Markovkette abstrahiert. Statt einer zustandsorientierten Betrachtung wird hier eine verhaltensorientierte gewählt. Es wird nicht beschrieben, in welchem Zustand sich ein System zu einem Zeitpunkt befindet, sondern wie sich das System verhalten kann. Es stehen also statt der Zustände die Zustandsübergänge im Vordergrund des Interesses. Die Wahl einer sprachlichen Beschreibung, im Gegensatz zur graphischen Darstellung bei den stochastischen Automaten, verstärkt zusätzlich den Abstraktionseffekt.

Stochastische Petrinetze erlauben eine sehr abstrakte Beschreibung des zugehörigen stochastischen Prozesses. So ist z.B. selbst bei einfachen Petrinetzen auf den ersten Blick nicht ohne weiteres festzustellen, ob es wenige, viele oder gar unendlich viele

	g_1 Abstraktions-grad	g_2 Strukturiertheit		g_3 Formalisie-rungsgrad	g_4 Kalkül	g_5 Universalität	p_1 Verbreitung	p_2 Werkzeug-unterstützung
SAN 1982	niedrig zustands-orientiert	modular	+	formal	+	ja	niedrig	erst ein existierendes Werkzeug
		hierarchisch	–					
		Replikation	+					
		strukt. Analyse	+					
SPA 1990	mittel verhaltens-orientiert	modular	+	sehr formal	++	ja	niedrig, Tendenz zunehmend	wenig
		hierarchisch	+					
		Replikation	+					
		strukt. Analyse	+[1)]					
SPN 1981	hoch einheits-orientiert (Marken)	modular	–	formal	++	ja	hoch	viele
		hierarchisch	–					
		Replikation	+[2)]					
		strukt. Analyse	–[3)]					
WSN 1957	hoch einheits-orientiert (Aufträge)	modular	+	eher informell	–	nein	sehr hoch	viele
		hierarchisch	+					
		Replikation	–					
		strukt. Analyse	++					

Tabelle 2.1 Vergleich der Modellierungsmethoden

1) bisher keine entsprechenden Arbeiten, aber prinzipiell möglich, 2) falls über Anfangsmarkierung auszudrücken, 3) nur für stark eingeschränkte Netzklassen bzw. erweiterte Formalismen

erreichbare Markierungen, also Zustände, gibt. Zur Bestimmung der einem Petrinetz zugehörigen Markovkette ist die Abarbeitung eines komplexen Algorithmus erforderlich. Den statischen Teil eines Petrinetzmodells bildet ein bipartiter Graph. Ein dynamischer Ablauf entsteht durch Marken, die nach wohldefinierten Regeln durch die Stellen des Graphen wandern, wobei sich ihre Zahl dabei vermehren oder verringern kann. Die Marken werden im Englischen oft Entity (Einheit) genannt, weshalb wir Petrinetze auch als einheits-orientierte (entity-oriented) Modellbeschreibungsmethode bezeichnen.

Ähnlich liegen die Dinge bei den Warteschlangennetzen, einer Darstellung die ebenfalls sehr stark vom zugehörigen Markovprozeß abstrahiert. Die Zustände entsprechen hier der Verteilung von Aufträgen auf die einzelnen Stationen, wobei zusätzliche Bedingungen, wie z.B. die aktuelle Bedienphase in einer Bedieneinheit mit Phasenverteilung, eine Rolle spielen. Hier sind es die Aufträge, die als Einheiten durch die statische Netzstruktur wandern, und damit liegt auch hier eine einheits-orientierte Beschreibungsmethode vor. Eine Besonderheit der Produktformnetze ist es, daß für die Analyse die zugehörige Markovkette gar nicht explizit bestimmt werden muß, da hier statt der Durchführung einer allgemeinen Markovanalyse die Lösung mit Hilfe geschlossener analytischer Formeln (bzw. mit auf solchen Formeln aufbauenden Algorithmen) berechnet werden kann.

Strukturiertheit (g_2)

Die Frage nach einer Tendenz zur Strukturiertheit bei den einzelnen Methoden stellen wir vorausschauend auf Kapitel 3, wo sich dieses Kriterium als eine der zentralen Voraussetzungen für effiziente Analysemethoden erweisen wird.

Ein Modell, das als SAN beschrieben ist, hat naturgemäß einen modularen Aufbau. Jeder stochastische Automat stellt ein Modul des Gesamtmodells dar, welches von seiner Umwelt eindeutig abgegrenzt ist und mit dieser über synchronisierende Ereignisse interagiert. Hierarchische Modelle lassen sich mit SANs nicht beschreiben, da hierfür kein entsprechender Mechanismus vorhanden ist. Die Darstellung von Modellen mit replizierten Komponenten ist dagegen leicht möglich, indem der diese Komponente beschreibende Automat entsprechend oft kopiert wird. Das in Abschnitt 2.2.1 erwähnte Auswerteverfahren von Plateau (siehe Abschnitt 3.5.2) beruht auf einer besonderen Matrixstruktur, die sich aufgrund der Strukturiertheit, also der Aufgliederung des Gesamtmodells in die einzelnen stochastischen Automaten ergibt.

Ähnliches läßt sich für eine prozeßalgebraische Modellbeschreibung feststellen. Prozeßvariablen (wie *Processor* oder *Multiprocessor* im Beispiel aus Abschnitt 2.2.2) bezeichnen abgegrenztes Teilverhalten, welches ausschließlich durch seine Interaktion mit der Umgebung charakterisiert ist. Über die modularen Eigenschaften der stochastischen Automaten hinausgehend ist bei Prozeßalgebren auch eine hierarchische Verknüpfung von Teilverhalten möglich und wird in den meisten Modellbeschreibungen auch benutzt (in obigem Beispiel enthält die Prozeßbeschreibung von *Multiprocessor* mehrfach das Modul *Processor*). Auch die Darstellung von Replikation ist durch die mehrfache Benutzung des Parallelоperators mit gleichen Operanden leicht möglich. Uns sind bisher keine Publikationen über spezielle Auswerteverfahren für stochastische Prozeßalgebren bekannt, was sicher damit zu tun hat, daß es sich hier um ein recht neues Gebiet handelt. Es ist jedoch klar, daß tensorielle Ansätze wie das Plateau'sche Verfahren nach

entsprechenden Anpassungen auch für prozeßalgebraische Modelle effizient eingesetzt werden können.

Im Gegensatz dazu muß man feststellen, daß herkömmliche stochastische Petrinetze kaum Möglichkeiten der Strukturierung bieten. Die Tatsache, daß beliebige Stellen und Transitionen eines Netzes durch Kanten verbunden sein können, erlaubt große Freiheiten wie die Formulierung komplexer globaler Synchronisationen und Schaltbedingungen. Diese Möglichkeit erkauft man sich allerdings mit dem Verzicht auf lokale Abgeschlossenheit. Die Darstellung von Modellen mit replizierten Teilmodellen ist grundsätzlich durch die Replikation von Subnetzen möglich, kann jedoch sehr aufwendig sein. In vielen Fällen läßt sich Replikation aber auch durch einen Parameter in der Anfangsmarkierung auf kompakte Weise ausdrücken. Strukturbasierte Auswertemethoden sind für allgemeine SPNs kaum bekannt. Da die Analyse komplexer Petrinetze schnell zu praktisch nicht mehr beherrschbaren Zustandsräumen führt, gibt es aber Arbeiten, in denen versucht wurde, trotz der geringen Strukturiertheit strukturelle Einheiten in stochastischen Petrinetzen herauszupräparieren. In diesem Zusammenhang ist der Ansatz von Ciardo und Trivedi zur Dekomposition von SPNs zu erwähnen [Ciardo und Trivedi, 1993] (siehe auch Abschnitt 3.5.1). Effiziente Auswerteverfahren für die erweiterten Netzklassen der Stochastic Well Formed Nets und der Stochastic Activity Networks werden in Abschnitt 3.6 angesprochen. Effiziente Lösungsverfahren gibt es auch für die eingeschränkte Klasse der stochastischen Petrinetze mit Produktform [Henderson und Lucic, 1993].

Warteschlangennetze kann man grundsätzlich als strukturiert bezeichnen, wenn man die einzelnen Stationen als Module des Gesamtmodells auffaßt. Sie erlauben den Aufbau von hierarchischen Modellen (siehe z.B. [Kobayashi, 1981]). Replikationen sind nur durch die explizite mehrfache Spezifikation von Stationen bzw. Subnetzen darstellbar, außer eventuell in Einzelfällen bei geschlossenen Netzen durch eine variierende Auftragszahl. Warteschlangennetze zeichnen sich gegenüber den anderen Modellierungsmethoden dadurch aus, daß für ihre Analyse eine große Zahl sehr effizienter Verfahren vorhanden ist. Von den verschiedenen Algorithmen für Produktformnetze wird die Strukturiertheit auf unterschiedliche Arten ausgenutzt: Die meisten Algorithmen beruhen auf der Untergliederung des Gesamtmodells in die einzelnen Stationen, aber es gibt auch andere Zerlegungsmöglichkeiten, z.B. nach Auftragsklassen bzw. -ketten.

Formalisierungsgrad (g_3) und Vorhandensein eines Kalküls (g_4)

Die ersten drei Modellierungsmethoden SANs, SPAs und SPNs stehen jeweils auf einer bereits aus dem Gebiet der zeitlosen Modelle bekannten formalen Basis. Da die stochastischen Automaten durch Erweiterungen aus dem Formalismus der endlichen Automaten hervorgegangen sind, lassen sich die aus diesem Bereich bekannten Regeln und Ergebnisse übertragen.

Stochastische Prozeßalgebren bieten einen sehr formalen Rahmen, der es ermöglicht, neben Leistungsbewertung auch qualitative Untersuchungen von Modellen durchzuführen. Zur Lösung des Zustandsraumproblems bieten SPAs einen interessanten Ansatzpunkt: Das Vorhandensein eines Äquivalenzbegriffes für Zustände auf prozeßsprachlicher Ebene und die darauf basierende Axiomatisierung erlauben vor der Übersetzung

in das zugehörige Transitionssystem eine Vereinfachung von Prozeßtermen und damit eine potentielle Reduktion des Zustandsraums (siehe Abschnitt 3.2.2). Untersuchungen über die Effizienz dieses Vorgehens sind zur Zeit jedoch noch laufend.

Für zeitlose Petrinetze existiert eine umfassende Theorie, von der stochastische Netzmodelle im Hinblick auf den Nachweis qualitativer Eigenschaften voll profitieren können. Wie auch Netztransformationen in den Bereich stochastischer Netze übertragen werden können, zeigt Abschnitt 3.2.1.

Die Modellbeschreibung von Warteschlangennetzen ist im Vergleich zu den anderen drei untersuchten Methoden relativ wenig formalisiert. Das liegt sicher daran, daß Warteschlangenmodelle von Anfang an ausschließlich als kompakte Repräsentation eines Leistungsbewertungsproblems gedacht waren, wobei aus Gründen der Anschaulichkeit stets auch graphische Darstellungen benutzt wurden. Für die Beantwortung anderer, also z.B. qualitativer Fragestellungen sind diese Modelle daher wenig geeignet.

Universalität (g_5)

Grundsätzlich stellt man fest, daß SANs, SPAs und SPNs sehr allgemeine Formalismen sind, mit denen sich fast beliebige dynamische Abläufe darstellen lassen, wie sie in Rechensystemen auftreten. Dies beruht auf der Tatsache, daß diese drei Methoden jeweils nur über wenige, sehr elementare Konstrukte und Operatoren verfügen. Sie haben dadurch aber auch den Nachteil, daß sämtliche Teile eines Modells explizit modelliert werden müssen, da außer den Grundkonstrukten keinerlei höherwertigen Module als Bausteine angeboten werden. Das kann die Beschreibung eines Modells unter Umständen sehr aufwendig machen. Z.B. ist bereits die Modellierung einer einfachen Warteschlange mit diesen Methoden zwar durchaus möglich aber entsprechend aufwendig.

Im Vergleich damit sind Warteschlangennetze weniger universell, z.B. nicht in der Lage allgemeine Synchronisationen oder Bedingungen auszudrücken. In Abschnitt 2.2.4 haben wir beobachtet, daß es schwierig ist, die Tatsache zu beschreiben, daß nicht mehr als ein Prozessorausfall gleichzeitig bearbeitet werden soll. Da bei Warteschlangen aber implizit Annahmen getroffen werden, wie z.B, daß der Warteraum ein FIFO-Puffer ist, lassen sich bestimmte Dinge viel leichter und unkomplizierter damit ausdrücken.

Stellt man die in Abschnitt 2.2 exemplarisch betrachteten Multiprozessormodelle einander gegenüber, so stellt man fest, daß es durchaus semantischen Unterschiede zwischen den vier Modellen gibt. Betrachtet man z.B. die Vergabe des Busses, so ist diese beim SAN-Modell und beim SPA-Modell mit einer exponentiell verteilten Zeit behaftet, während beim GSPN-Modell die Busvergabe durch eine zeitlose Transition dargestellt wird. Dies bedeutet aber nicht notwendigerweise einen Unterschied zwischen den Modellbeschreibungsmethoden, da keine prinzipiellen Argumente dagegen sprechen, die Formalismen SAN und SPA entsprechend um zeitlose Ereignisse bzw. Aktionen zu erweitern. Im Rahmen der stochastischen Prozeßalgebren wird zur Zeit intensiv an einer solchen Erweiterung gearbeitet [Rettelbach, 1995]. In bezug auf SANs sei auf die in Abschnitt 5.1.1 beschriebene Erweiterung der Modellwelt aus Kapitel 4 um unendlich große Raten verwiesen. Beim Warteschlangenmodell wird die Busvergabe nicht expli-

zit modelliert. Sie kann daher als zeitlos oder als Bestandteil der Speicherzugriffszeit angesehen werden.

Verbreitung (p_1) und Werkzeugunterstützung (p_2)

Nur bei stochastischen Petrinetzen und Warteschlangennetzen kann man von einer weiten Verbreitung sprechen, die sich auch in der großen Fülle angebotener Werkzeuge widerspiegelt. Es wäre ohne weiteres möglich, für jede dieser beiden Methoden neben den in Abschnitt 2.2 erwähnten Werkzeugen noch viele andere aufzulisten, wobei die Petrinetzwerkzeuge eher akademischen, die Warteschlangennetze teilweise auch schon kommerziellen Charakter haben. Letzteres liegt daran, daß Warteschlangenmodelle seit vielen Jahren Gegenstand theoretischer Untersuchungen sind und auch schon viel im industriellen Bereich, vor allem bei der Planung von Kommunikationsnetzen eingesetzt werden.

Für SANs ist uns nur ein einziges Werkzeug bekannt [Plateau *et al.*, 1988] und für SPAs existieren nur die in Abschnitt 2.2.2 erwähnten Prototypen. Im akademischen Bereich läßt sich aber im Moment ein stark zunehmendes Interesse an stochastischen Prozeßalgebren feststellen, welches sicher auch zur Entwicklung von Werkzeugen für die Modellbeschreibung und -auswertung führen wird. Es ist zu erwarten, daß in Zukunft formaler Softwareentwurf und Leistungsbewertung immer stärker integriert sein werden, und daß deshalb Modellierungsmethoden mit formaler Basis an Bedeutung gewinnen werden. Zum Teil ist dies heute schon sichtbar an der Verwendung formaler Spezifikationsmethoden wie SDL (automatenbasiert) und LOTOS (prozeßalgebrabasiert), und der Bemühung, Werkzeuge für diese Methoden um Leistungsaspekte zu erweitern.

Zusammenfassung des Vergleichs

Man muß einsehen, daß es kaum eine allgemeingültige Metrik geben kann, mit der ein wertender Vergleich zwischen den diskutierten Modellierungsmethoden möglich ist. Die Methoden unterscheiden sich in einigen wesentlichen Punkten recht stark. Insgesamt wird man aber erst dann, wenn es um eine konkrete Anwendung geht, in der Lage sein, die dafür geeignetste Methode auszuwählen. Da jede Methode Vorteile bietet, aber auch Schwächen aufweist, kann erst im Zusammenhang mit einem konkreten Modellierungsproblem eine Entscheidung über die zu benutzende Modellierungsmethode getroffen werden. Liegt z.B. ein durch ein Warteschlangennetz beschreibbares Problem vor, und ist man vornehmlich an Leistungsgrößen wie mittleren Wartezeiten und Pufferbelegung interessiert, so wird man ein Warteschlangennetz den anderen Beschreibungsmethoden vorziehen, auch wenn es z.B. die Nachprüfung qualitativer Eigenschaften nicht unterstützt. Jede der Methoden hat und behält daher ihre Existenzberechtigung.

Man stellt aber bei der Modellierung von Synchronisationsmechanismen und wechselseitigen Abhängigkeiten in verteilten Systemen oder Multiprozessoren fest, daß Warteschlangenmodelle hierfür wenig geeignet sind. Daneben muß man auch anerkennen, daß das Fehlen einer formalen Basis für die Modellbeschreibung bei den Warteschlangennetzen einen gravierenden prinzipiellen Nachteil darstellt. Auch wenn Methoden wie z.B. stochastische Prozeßalgebren heute noch nicht über vergleichbar effiziente Auswerteverfahren und unterstützende Werkzeuge verfügen, so liegt in ihnen dennoch ein

großes Potential zur Lösung zukünftiger Probleme des methodischen, Leistungsbewertung einbeziehenden Systementwurfs.

2.4 Anwachsen des Zustandsraums für das Beispielmodell

Zum Schluß dieses Kapitels wollen wir drei der vier behandelten Beschreibungsmethoden aus der Sicht des Auswerteaufwands, ausgedrückt durch das Wachstum des Zustandsraums für das Beispielmodell aus Abschnitt 2.1, betrachten.

Zunächst wurden mit Hilfe des TIPP-Tools die Zustandszahlen für das in Abb. 2.7 dargestellte Modell ermittelt. Die Zustandszahlen für dieses SPA-Modell gelten wegen der bereits festgestellten Übereinstimmung auch für das SAN-Modell aus Abb. 2.5. In Tabelle 2.2 sind die Zustandszahlen für variierenden Parallelitätsgrad, d.h. für eine unterschiedliche Anzahl von Prozessoren $N = 1 \ldots 7$ angegeben. Außerdem enthält die Tabelle die Zeiten für den Aufbau des Zustandsraums und die anschließende numerische Analyse der Markovkette.

# Prozessoren N	**Zeit [min:s:ms]**	**# Zustände**
1	0:02:490	9
2	0:02:450	51
3	0:04:710	243
4	0:20:650	1053
5	2:02:370	4293
6	17:05:330	16767
7	117:04:860	63423

Tabelle 2.2 Zustandszahlen für das Multiprozessorbeispiel in seiner Formulierung als stochastische Prozeßalgebra gemäß Abb. 2.7 (bzw. SAN gemäß Abb. 2.5)

Diese Werte wurden auf einer HP 9000/710 Workstation mit 32MB Hauptspeicher gemessen. Da die Parameter für die einzelnen Raten so gewählt waren, daß sich eine “gutartige” (nicht steife) Markovkette ergab, wurden für die numerische Analyse nur wenige Iterationen mit dem SOR-Verfahren benötigt. Die angegebenen Rechenzeiten wurden also hauptsächlich für das Aufstellen des Zustandsraums verbraucht.

Dann wurde für das Multiprozessorbeispiel in seiner Formulierung als stochastisches Petrinetz vom Typ GSPN gemäß Abb. 2.10 die Anzahl der stabilen Markierungen, also ebenfalls die Zustandszahl der zugeordneten Markovkette, ermittelt. Dies geschah mit Hilfe des Werkzeugs DSPNexpress [Lindemann, 1992] für verschiedene Prozessorzahlen, d.h. verschiedene Anfangsmarkierungen der Stelle *P1*. Die hier gewonnenen Zustandszahlen sind zusammen mit dem Zeitbedarf für die Berechnung des Erreichbarkeitsgraphen und die Analyse der zugehörigen Markovkette des jeweiligen Netzes in der Tabelle 2.3 angegeben.

Bevor wir zur Interpretation der ermittelten Zustandszahlen kommen sei darauf hingewiesen, daß für das stochastische Petrinetzmodell gemäß Abb. 2.11 die Zustandszahlen

für Prozessorzahlen größer als zwei nicht ermittelt werden konnten. Das liegt daran, daß durch die explizite graphische Darstellung eines Subnetzes für jeden Prozessor das Netz bereits bei kleinen Prozessorzahlen äußerst komplex wird und daher schon beim Zeichenvorgang nicht mehr handhabbar ist. Das in Abb. 2.11 dargestellte Netz ($N = 2$ Prozessoren) besitzt 32 stabile Markierungen, also eine im Vergleich mit den 17 stabilen Markierungen des kompakten Netzes von Abb. 2.10 bereits deutlich größere Zahl.

# Proz. N	**Zeit [min:s:ms]**	**# Zust.**	**# Zust. (mod.)**
1	0:17:030	5	8
2	0:17:390	17	30
3	0:17:030	44	80
4	0:17:480	95	175
5	0:17:670	181	336
6	0:17:870	315	588
7	0:18:570	512	960
8	0:19:180	789	1485
9	0:20:530	1165	2200
10	0:23:400	1661	3146
...		...	
15	1:07:510	6816	13056
...		...	
30	17:29:390	86831	?[1)]

Tabelle 2.3 Zustandszahlen für das Multiprozessorbeispiel in seiner Formulierung als stochastisches Petrinetz gemäß Abb. 2.10 (kompakte Darstellung). Die letzte Spalte enthält die Zustandszahlen für ein modifiziertes Modell mit zeitbehafteter Busvergabe. 1) Zustandszahl wegen Speichermangels nicht ermittelbar

Man bemerkt, daß die Anzahl der Zustände zwar auch beim kompakten GSPN-Modell auf sehr große Werte anwächst, allerdings bei weitem nicht so schnell, wie beim SPA-Modell. Für die Praxis bedeutet dies, daß sich für dieses Beispiel in seiner Darstellung als kompaktes GSPN-Modell keine wesentlichen Schwierigkeiten durch die Größe des Zustandsraums ergeben werden, da bei einer busgekoppelten Multiprozessorarchitektur mit nur einem einzigen Bus kaum Prozessorzahlen der Größenordung 30 interessieren werden. Das SPA- bzw. SAN-Modell erlaubt dagegen Auswertungen nur für eine sehr begrenzte Prozessorenzahl, und wird daher manche interessierende Fragestellung unbeantwortet lassen.

Um die Ursachen für das unterschiedlich starke Anwachsen der Zustandsräume herauszufinden, folgt nun ein Vergleich der beiden Modellbeschreibungen. Dabei stellt man im wesentlichen zwei Unterschiede fest:

1. Im SPA-Modell hat jeder Prozessor eine eigene Identität, während im kompakten GSPN-Modell die Prozessoren durch ununterscheidbare Tokens repräsentiert werden. So ist beim SPA-Modell z.B. die folgende Aussage möglich: “Der erste und der dritte Prozessor sind ausgefallen”. Durch das kompakte GSPN-Modell kann dagegen lediglich beschrieben werden, daß zwei Prozessoren ausgefallen sind, nicht aber, welche.
2. Die Busvergabe hat im SPA-Modell eine exponentielle Bearbeitungszeit, während sie im GSPN-Modell ohne Verzögerung stattfindet.

Um die Einflüsse der beiden Punkte auf das unterschiedlich starke Anwachsen des Zustandsraums zu bewerten und damit einen fairen Vergleich zu ermöglichen, wurde zusätzlich ein modifiziertes (ebenfalls kompaktes) GSPN-Modell betrachtet, bei dem auch die Busvergabe durch Transitionen mit exponentieller Schaltzeit modelliert wurde, das sich also im zweiten Punkt nicht mehr vom SPA- bzw. SAN-Modell unterscheidet. Die Zustandszahlen für dieses Netz sind in der letzten Spalte der Tabelle 2.3 angegeben. Gegenüber dem ursprünglichen Petrinetz kommt es erwartungsgemäß zu einer Vergrößerung des Zustandsraums, da die zuvor den zeitlosen Transitionen entsprechenden verschwindenden Markierungen nun auch stabile Markierungen sind und daher als Zustände der Markovkette in Erscheinung treten. Trotz dieser Angleichung sind die Zustandsräume des kompakten GSPN-Modells immer noch viel kleiner als die des SPA-Modells bei entsprechender Prozessorenzahl. Offensichtlich sind die Auswirkungen des zweiten Punktes nicht so dramatisch wie die des ersten.

Wir stellen zusammenfassend fest, daß der Zustandsraum des SAN- bzw. SPA-Modells hauptsächlich deshalb wesentlich größer ist als der des kompakten GSPN-Modells, weil statt einer Repräsentation der Prozessoren durch ununterscheidbare Tokens diese als individuelle Einheiten modelliert werden. Wir wollen schon an dieser Stelle die allgemeine Bedeutung dieser Beobachtung herausstellen, die für die Betrachtungen in Kapitel 3 und Kapitel 4 eine große Rolle spielen wird. Man hat natürlich den Wunsch, auch bei strukturierten Modellen wie SAN- und SPA-Modellen mit weniger Zuständen auszukommen. Eine kompakte Modellbeschreibung mit ununterscheidbaren Prozessoren ist zwar prinzipiell auch mit Hilfe von SAN- bzw. SPA-Modellen möglich, ihre Formulierung wäre im allgemeinen aber sehr aufwendig und nicht so leicht intuitiv verständlich wie es die Modelle der Abbildungen 2.5 und 2.7 sind. Zur Lösung dieses Dilemmas wird in dieser Arbeit vorgeschlagen, strukturierte Modelle wie SAN- und SPA-Modelle auf naheliegende Weise mittels mehrerer parallel geschalteter Teilmodelle zu spezifizieren, anschließend jedoch einen Reduktionsschritt durchzuführen, mit dem die gewünschte Verkleinerung des Zustandsraums errreicht wird. Ein solcher Weg wird in der Modellwelt von Kapitel 4 beschritten.

3 Methoden für die effiziente Modellierung großer Systeme

In diesem Kapitel wird ein Überblick über Methoden gegeben, deren Ziel es ist, komplexe Markovmodelle für die Analyse handhabbar zu machen. Mit komplexen Modellen sind dabei solche gemeint, die einen so großen Zustandsraum haben, daß sie mit herkömmlichen Methoden wegen Speichermangels oder zu langer Rechenzeiten nicht mehr beherrschbar sind. Es wird ein Einblick in den derzeitigen Stand der Technik gegeben, indem publizierte Methoden untersucht werden. Da sich bekannte Methoden für die Beherrschung großer Modelle oft noch als unzulänglich erweisen, werden aus der Übersicht Anforderungen an neue Techniken aufgestellt.

Das Kapitel beschränkt sich aber nicht auf eine Sichtung der Bemühungen anderer Autoren. Eigene Arbeiten gehen bei der in Abschnitt 3.2.1 dargestellten Vereinfachung von GSPNs über die in der Literatur beschriebenen Methoden hinaus. Die in Abschnitt 3.4 diskutierte Matrix-Semantik für eine stochastische Prozeßalgebra ist ebenfalls ein neuer, eigener Ansatz.

3.1 Übersicht über Ansatzpunkte zur Überwindung des Zustandsraumproblems

Das allgemeine Vorgehen bei der Modellierung mit Markovmodellen gliedert sich in mehrere aufeinanderfolgende Phasen, die bei jedem Modellierungsvorgang durchlaufen werden, vgl. Tab. 3.1, links.

	Modellierungsphase	Ansatzpunkte
I	Erstellung der Modellbeschreibung auf höherer Ebene	Transformation der Modellbeschreibung
		Strukturierte Beschreibung
II	Übersetzung	“Geschicktes” Übersetzen
III	Eventuelle Manipulation der Beschreibung auf niedriger Ebene	Suche nach zusammenfaßbaren Zuständen
IV	Numerische Analyse	“Intelligente” numerische Verfahren

Tabelle 3.1 Vier Phasen bei der Markovmodellierung und Übersicht über mögliche Ansatzpunkte zur Überwindung des Zustandsraumproblems

Die Modellierung beginnt stets mit der Erstellung einer Modellbeschreibung auf höherer Ebene durch den Modellierer. Der nächste Schritt besteht aus der Übersetzung

der Beschreibung auf höherer Ebene in die zugehörige Markovkette, welche in der Tabelle als Beschreibung auf niedriger Ebene bezeichnet wird. Schließlich folgt die numerische Analyse der Markovkette mit Hilfe eines (meist iterativen) numerischen Lösungsverfahrens, welche als Resultat den Vektor der stationären bzw. transienten Zustandswahrscheinlichkeiten liefert.

Die Analyseergebnisse in Form von Zustandswahrscheinlichkeiten müssen dann zur Modellbeschreibung auf höherer Ebene in Beziehung gebracht werden, um eine adäquate Interpretation dieser Ergebnisse sicherzustellen. Darunter versteht man zum Beispiel die Berechnung des Durchsatzes einer Transition in einem stochastischen Petrinetz, welche sich als das Produkt aus der Aktivierungswahrscheinlichkeit und der Schaltrate der Transition ergibt.

Entsprechend der Modellierungsphasen kann man nun Ansatzpunkte für den Umgang mit dem Zustandsraumproblem lokalisieren, vgl. Tab. 3.1, rechts. Es stellt sich heraus, daß in allen vier Phasen Ansätze möglich sind. Eine Beschreibung und Diskussion spezieller Verfahren findet sich in den folgenden Abschnitten. Die Abschnitte 3.2 und 3.3 befassen sich mit Ansatzpunkten in der Phase I, der Abschnitt 3.4 mit Ansatzpunkten in der Phase II. Ansätze in Phase IV werden in Abschnitt 3.5 diskutiert.

Allgemein kann gesagt werden, daß die Ansätze zwei generellen Leitlinien folgen:

1. Reduktion der Zustandszahl durch Elimination redundanter Zustände bzw. durch das Zusammenfassen von Teilmengen von Zuständen zu einem Makrozustand.
2. Ausnutzung von Modelleigenschaften (z.B. Strukturiertheit) für intelligente Auswerteverfahren.

Im ersten Fall wird also versucht, ein zu starkes Anwachsen des Zustandsraums durch Zusammenfassen von Zuständen zu verhindern, während im zweiten Fall die Größe des Zustandsraums zunächst hingenommen und unverändert belassen wird, man dann jedoch versucht, adäquate Analyseverfahren für große Zustandsräume anzugeben. Von Trivedi wurden diese beiden Ansätze mit den Schlagwörtern “largeness avoidance” und “largeness tolerance” umrissen [Trivedi und Malhotra, 1993, S. 40].

Das Zusammenfassen von Zuständen auf der Ebene der Markovkette (Beschreibung auf niedriger Ebene) ist zwar vom mathematischen Gesichtspunkt durch das Konzept der Zusammenfaßbarkeit (lumpability) ein gelöstes Problem (siehe Anhang A.2). In der Praxis ist es jedoch nur mit unerhörtem Rechenaufwand möglich, auf dieser Ebene Teilmengen zusammenfaßbarer Zustände zu identifizieren. Denkbare Suchalgorithmen für unstrukturierte Zustandsräume sind von hoher Komplexität und daher für große Zustandszahlen nicht praktikabel, weshalb solche Algorithmen, also Ansatzpunkte in der Phase III, in dieser Arbeit nicht näher betrachten werden. Wir stellen vielmehr fest, daß die Information über Teilmengen zusammenfaßbarer Zustände aus der Modellbeschreibung auf höherer Ebene extrahiert werden muß, wenn ein solches Vorgehen erfolgreich sein soll.

3.2 Transformation der Modellbeschreibung auf höherer Ebene

In diesem Abschnitt diskutieren wir eine Möglichkeit zur Analyse der Modellbeschreibung auf höherer Ebene, die darin besteht, die Modellbeschreibung geeignet zu trans-

formieren. Dabei wird das Ziel verfolgt, die nachfolgende Erzeugung der zugehörigen Markovkette und deren Auswertung zu vereinfachen. Uns beschäftigt in diesem Abschnitt also *nicht* die Analyse auf der Ebene der Markovkette, sondern eine frühere Phase der Modellanalyse.

Die Grundidee der Modelltransformation ist die folgende: Man stellt fest, daß sich bestimmte Muster von Modellkonstrukten, bestimmte immer wieder in verschiedenen Variationen auftretende Szenarien, in eine einfachere Struktur transformieren lassen. Es wird also die Modellbeschreibung modifiziert, ohne daß sich dadurch die Semantik des Modells (das Modellverhalten) — zumindest die für das Ziel der Leistungsbewertung relevanten Aspekte — ändern soll.

Um allgemein anwendbare Vorschriften für die Modelltransformation angeben zu können, sind die beiden folgenden Aufgaben zu lösen:

1. Bestimmung einer Menge von geeigneten transformierbaren Musterkonstrukten und Angabe der dazugehörigen Vorschriften zur Transformation in einfachere Ersatzkonstrukte.
2. Entwicklung effizienter Suchverfahren zum Auffinden vordefinierter Musterkonstrukte im Modell.

Modelltransformationen sind im allgemeinen methodenspezifisch. D.h., daß eine Transformationsvorschrift immer für eine spezielle Modellbeschreibungsmethode angegeben wird. Meist sind die Transformationsvorschriften nicht ohne weiteres auf andere Beschreibungsmethoden übertragbar. Die hinter einer Transformationsvorschrift steckende Grundidee kann sich jedoch oft auch im Rahmen anderer Modellierungsmethoden als anwendbar erweisen.

In den beiden folgenden Abschnitten werden Transformationsregeln im Kontext spezieller Modellierungsmethoden vorgestellt.

3.2.1 Vereinfachung verallgemeinerter stochastischer Petrinetze

Die qualitative Analyse zeitloser Petrinetze wirft zum Teil ähnliche Probleme in bezug auf die Größe des Zustandsraums auf wie die Analyse stochastischer Petrinetze. Bestimmte Eigenschaften zeitloser Petrinetze lassen sich zwar mit Hilfe von Netzinvarianten nachweisen [Starke, 1990], in vielen Fällen ist jedoch auch hier die Betrachtung des dynamischen Netzverhaltens, also die Erzeugung und Untersuchung der gesamten Erreichbarkeitsmenge notwendig. Diese kann, ebenso wie bei stochastischen Petrinetzen, von gigantischem Umfang und deshalb in der Praxis schwierig zu handhaben sein.

Man hat daher andere Methoden zum Nachweis von qualitativen Eigenschaften komplexer zeitloser Petrinetze gesucht. Ein erfolgreicher Weg ist in diesem Zusammenhang die Entwicklung von Netztransformationen, die es gestatten, ein vorgegebenes Netz zu vereinfachen, ohne dabei seine interessierenden Eigenschaften, wie z.B. Deadlockfreiheit oder Lebendigkeit, zu verändern. In Arbeiten von Berthelot sind Reduktionsregeln für zeitlose Netze dargestellt [Berthelot, 1986; Berthelot, 1987]. Es erfolgte auch eine Übertragung dieser Reduktionsregeln in den Kontext farbiger zeitloser Petrinetze [Haddad, 1990]. Dabei ging es stets um die Verifikation von Verhaltenseigenschaften zeitloser Petrinetze, also deren qualitative Analyse.

Wir werden im folgenden untersuchen, ob diese Reduktionstechnik auf Netze der Klasse GSPN übertragbar ist und wie dies gegebenenfalls geschehen kann.

Es sei hier auf parallel zu unseren eigenen Arbeiten verlaufene Bemühungen hingewiesen, die in [Chiola *et al.*, 1991b] veröffentlicht wurden. Dort wird ein Algorithmus für die Elimination zeitloser Transitionen auf Netzebene im Kontext von sog. GSPN-Familien, d.h. parametrisierter Netze angegeben. Weitergehende, von dem Konzept des beobachtbaren Verhaltens (exhibited behaviour) eines Netzes ausgehende Untersuchungen finden sich in [Simone und Marsan, 1992]. Dort werden zwei GSPNs zunächst genau dann als äquivalent bezeichnet, wenn die beiden zugehörigen Markovketten zueinander isomorph sind. Es folgt dann eine Ausweitung des Äquivalenzbegriffes auf zwei Markovketten, deren eine die durch Anwendung des Konzepts der Zusammenfaßbarkeit (lumpability) aus der anderen entstandene ist.

Redundante Stellen

Eine Stelle, welche in jeder erreichbaren Markierung genügend Tokens enthält, um das Schalten ihrer Ausgangstransitionen nicht zu verhindern, heißt nach Berthelot *redundante Stelle* (redundant place). Manchmal wird eine solche Stelle auch als *implizit* bezeichnet. Eine redundante Stelle kann aus dem Netz entfernt werden, ohne daß dadurch sein Verhalten beeinflußt wird. Dieser Gedanke ist ohne weiteres auf GSPN-Netze übertragbar. Als Illustration dient das Beispiel in Abb. 3.1.

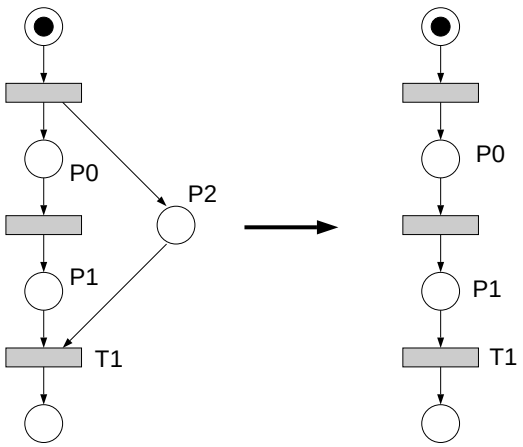


Abbildung 3.1: Elimination der redundanten Stelle $P2$

Die Netze in Abb. 3.1 und in den folgenden Abbildungen sind als GSPNs zu interpretieren, wobei für alle grau unterlegten Transitionen nicht spezifiziert ist, ob es sich um zeitbehaftete oder zeitlose Transitionen handelt.

Zum Schalten der Transition $T1$ ist jeweils ein Token in den Stellen $P1$ und $P2$ notwendig. Es ist sichergestellt, daß immer dann, wenn $P1$ ein Token enthält auch $P2$ mit einem solchen versehen ist. Das Erreichen einer Markierung M mit $M(P1) = 1$ und $M(P2) = 0$ ist nicht möglich. Daher ist $P2$ eine redundante Stelle. Die Reduktion des Netzes besteht darin, sämtliche redundante Stellen und die zu ihnen gehörenden Kanten zu entfernen, siehe Abb. 3.1 (rechts).

Im Zusammenhang mit der Definition redundanter Stellen begegnen wir einem Problem, welches auch bei der Definition anderer Reduktionsregeln auftreten wird: Die eben gegebene Charakterisierung einer redundanten Stelle basiert auf einer Verhaltensbedingung für das Netz (ein bestimmter Markierungstypus ist erreichbar / nicht erreichbar). Diese wird man in der Praxis nicht nachprüfen wollen, da dafür die gesamte Erreichbarkeitsmenge generiert werden müßte. Man ist daher bestrebt, die Anwendungskriterien für eine Reduktionsregel ausschließlich mit Hilfe struktureller Bedingungen zu formulieren.

Es gibt neben der Charakterisierung durch Verhaltensbedingungen die zweite Möglichkeit, eine redundante Stelle P durch Strukturbedingungen zu charakterisieren. Dazu fordert man, entsprechend der von Berthelot angegebenen Bedingung, für eine redundante Stelle die Existenz einer Referenzmenge von Stellen $I \subseteq S$ und einer Gewichtungsfunktion $V : I \cup \{P\} \rightarrow I\!N - \{0\}$, so daß folgende drei Bedingungen erfüllt sind:

1. In der Anfangsmarkierung M_0 ist die gewichtete Markierung von P um $b_{M_0} \geq 0$ größer als die Summe der gewichteten Markierungen der zu I gehörenden Stellen.
2. Für das Schalten einer beliebigen Transition des Netzes werden höchstens b_{M_0} mehr Tokens von der Stelle P als aus der Menge I benötigt.
3. Beim Schalten einer beliebigen Transition wächst die gewichtete Markierung von P mindestens genauso stark wie die gewichtete Markierung der zu I gehörenden Stellen.

Also sind auch diese Strukturbedingungen ohne Einschränkung von zeitlosen Petrinetzen auf GSPNs übertragbar. Im Beispiel in Abb. 3.1 besteht die Referenzmenge bezüglich der redundanten Stelle $P2$ aus den Stellen $P0$ und $P1$ und als Gewichtsfunktion kann hier die konstante Funktion mit dem Wert eins gewählt werden.

Es ist leider festzustellen, daß das Entfernen redundanter Stellen zwar eine Vereinfachung der graphischen Darstellung eines GSPNs bewirkt, die Größe der Erreichbarkeitsmenge und die Struktur des Erreichbarkeitsgraphen sich jedoch nicht ändern. Man erhält also aus dem ursprünglichen und aus dem reduzierten GSPN dieselbe Markovkette. Der einzige Gewinn für die Analyse besteht in der Tatsache, daß beim Erzeugen des Erreichbarkeitsgraphen die interne Repräsentation eines Zustands geringfügig kompakter sein kann, da für die Markierung der redundanten Stellen kein Speicherplatz vorgesehen werden muß.

Verdoppelte Stellen

Enthält ein Netz zwei Stellen P und P', deren Tokens in keiner erreichbaren Markierung miteinander "verwechselt" werden können, so nennt man sie *verdoppelte Stellen* (doubled places). Dies wird formal durch die drei folgenden Bedingungen ausgedrückt:

1. Keine Transition hat gleichzeitig P und P' als Eingangsstellen.
2. Jede Transition, die P (P') als Eingangsstelle hat, hat mindestens eine weitere Eingangsstelle.
3. In jeder erreichbaren Markierung und für jede Transition T, die P als Eingangsstelle hat, gilt: Enthalten alle Eingangsstellen von T außer P genügend Tokens, um das Schalten von T zu ermöglichen, so ist P' leer (und umgekehrt).

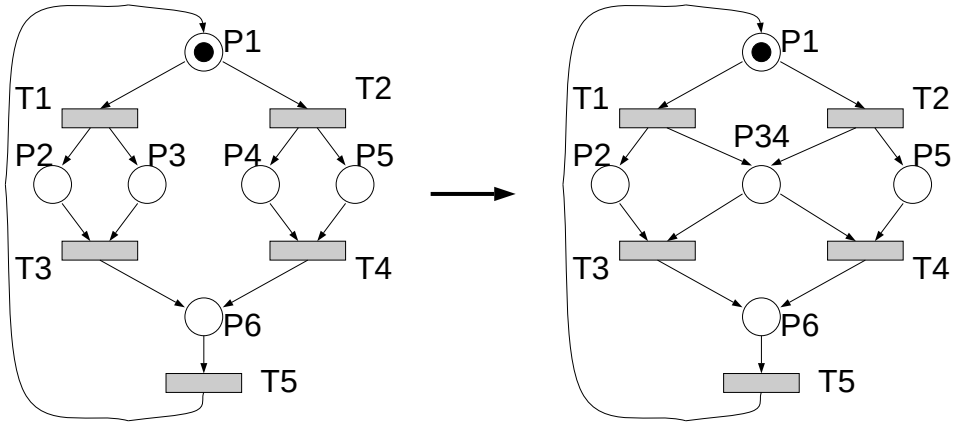


Abbildung 3.2: Fusion der verdoppelten Stellen $P3$ und $P4$

Der Reduktionsschritt besteht in der Fusion der beiden Stellen zu einer einzigen. Als Beispiel betrachte man Abb. 3.2, wo es sich bei $P3$ und $P4$ um verdoppelte Stellen handelt. In diesem einfachen Beispielnetz gibt es aber auch andere Paare verdoppelter Stellen, z.B. könnte man $P3$ und $P5$ fusionieren.

Die dritte Bedingung ist eine Verhaltensbedingung, die aber im Einzelfall auch durch eine Netzinvariante nachgeprüft werden kann, welche ausdrückt, daß bestimmte Typen von Markierungen ausgeschlossen sind. Eine Strukturbedingung für das Beispielnetz in Abb. 3.2 kann in Form der folgenden S-Invarianten [Starke, 1990] formuliert werden. Dabei ist C die Inzidenzmatrix des GSPN.

$$yC = y \begin{bmatrix} -1 & -1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & -1 \end{bmatrix} = 0$$

Einer der Lösungsvektoren dieses linearen Gleichungssystems ist $y_1 = [1\,1\,0\,1\,0\,1]$. Zusammen mit der Anfangsmarkierung (ein Token in $P1$) besagt diese Lösung, daß in jeder erreichbaren Markierung die Stellen $P1, P2, P4$ und $P6$ zusammen genau ein Token enthalten. Also können $P2$ und $P4$ in keiner erreichbaren Markierung gleichzeitig markiert sein, womit die dritte Bedingung erfüllt ist.

Im Beispiel läßt sich das Paar verdoppelter Stellen noch durch eine weitere S-Invariante, den Lösungsvektor $y_2 = [1\,0\,1\,1\,0\,1]$ charakterisieren. Die Stellen $P3$ und $P4$ können also nie gleichzeitig markiert sein, was zusammen mit den beiden ersten Bedingungen intuitiv klar macht, daß Tokens in diesen beiden Stellen nicht miteinander verwechselt werden können.

Untersucht man den Gewinn, den man durch die Fusion verdoppelter Stellen erzielt, so muß man wiederum — wie bei den redundanten Stellen — feststellen, daß sich die Menge erreichbarer Zustände nicht reduziert, sich also lediglich die Komplexität der graphischen Repräsentation des Netzes verringert.

Fusion von Transitionen

Wir wollen als nächstes die Fusion von Transition betrachten. Es war Berthelots Anliegen, im Rahmen zeitloser Petrinetze bestimmte dicht aufeinanderfolgende Schaltfolgen von Transitionen ununterscheidbar zu machen, die betreffenden Transitionen also zu verschmelzen. Für die Übertragung der so motivierten Reduktionsregeln in den Bereich der Netzklasse GSPN kommt eine Fusion von Transitionen nur dann in Frage, wenn mindestens einer der zu verschmelzenden Partner eine zeitlose Transition ist, da die Fusion zweier exponentieller Transitionen zu einer neuen zeitbehafteten Transition mit Phasenverteilung führen müßte, welche in GSPNs nicht zur Verfügung steht.

Die Fusion mehrerer ausschließlich zeitloser Transitionen wollen wir hier nicht näher betrachten, da dies ein Spezialfall der Reduktion zeitloser Subnetze ist, welche sich stets auf einen sog. free-choice Konflikt reduzieren lassen [Simone und Marsan, 1992]. Es bleibt also die Verschmelzung einer Menge von zeitlosen mit einer Menge von exponentiellen Transitionen zu behandeln. Dabei orientieren wir uns an den Konzepten Prä-Fusion und Post-Fusion von Berthelot.

Eine der Prä-Fusion von Transitionen entsprechende Reduktionsregel für GSPN-Netze läßt sich für Szenarien angeben, in denen nach dem Schalten einer zeitlosen Transition eine Konfliktsituation zwischen zwei (oder mehreren) exponentiellen Transitionen auftritt. Die Reduktion besteht dann darin, die zeitlose Transition mit den darauffolgenden exponentiellen derart zu verschmelzen, daß die Vorbedingungen für das Schalten der zeitlosen Transition auf die nachfolgenden exponentiellen Transitionen übertragen werden. Eine Beispielsituation ist in Abb. 3.3 abgebildet.

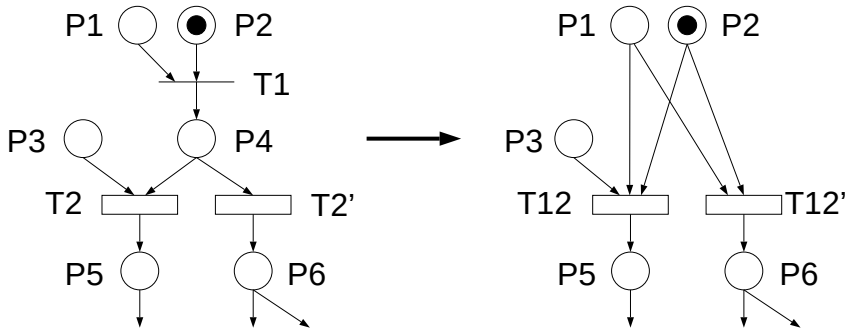


Abbildung 3.3: Prä-Fusion von Transitionen

Das Pendant zur Prä-Fusion stellt die Post-Fusion von Transitionen dar, welche den Konflikt zweier (oder mehrerer) zeitloser Transitionen mit dem Schalten einer vorangegangenen exponentiellen Transition fusioniert, indem diese dupliziert (vervielfacht) wird. Die Schaltrate der exponentiellen Transition wird entsprechend der Wahrscheinlichkeiten für das Schalten der einen oder der anderen zeitlosen Transition auf die beiden Duplikate verteilt. Dieser Sachverhalt ist in Abb. 3.4 veranschaulicht.

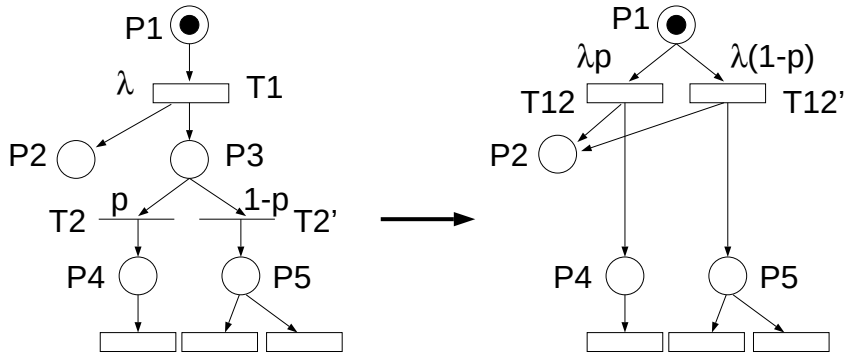


Abbildung 3.4: Post-Fusion von Transitionen

Der durch die Fusion von Transitionen erzielte Gewinn für die Netzanalyse läßt sich wie folgt veranschlagen: Bei beiden diskutierten Varianten werden eine bzw. mehrere zeitlose Transitionen bereits auf Netzebene eliminiert. Dies resultiert in einer Reduzierung der Anzahl verschwindender Markierungen des Netzes. Während durch die Post-Fusion genau eine verschwindende Markierung eingespart wird, erreicht man durch die Prä-Fusion, je nach Gestalt der Vorbedingungen für die zu eliminierende zeitlose Transition unter Umständen eine Einsparung mehrerer verschwindender Markierungen.

Eine Reduzierung der Zahl verschwindender Markierungen bedeutet eine Speicherplatzersparnis während der Konstruktion der zugehörigen Markovkette. Besonders für solche Netze vom GSPN-Typ, die eine große Anzahl zeitloser Markierungen haben, bringt die Reduktion Vorteile. Was die Anzahl der stabilen Markierungen des Netzes betrifft, also die Zustandszahl der zugehörigen Markovkette, so bleibt diese durch die Fusion von Transitionen unverändert.

Vertauschen von zeitlosem mit zeitbehaftetem Konflikt

Eine wichtige Reduktionsregel, die auch in [Simone und Marsan, 1992] Erwähnung findet, besteht aus dem Vertauschen zweier Konfliktstellungen zwischen zeitlosen und zeitbehafteten Transitionen. Diese Regel ist genau dann anwendbar, wenn auf den Konflikt von n zeitlosen Transitionen in jedem Zweig ein Konflikt m exponentieller Transitionen folgt, und zwar so, daß diese m Transitionen in jedem Zweig mit demselben Satz von Schaltraten ausgestattet sind. Ein typischer Fall, in dem eine solche Situation auftreten kann, ist die Abarbeitung verschiedener Auftragsklassen auf verscheidenen Prozessortypen. Ein solches Szenario ist in Abb. 3.5 für den Spezialfall $m = 2$ dargestellt. An die Nachfolgeverhalten der $n \times m$ exponentiellen Transitionen werden im allgemeinen keine Bedingungen gestellt, in der Abbildung wurde jedoch aus Gründen der Einfachheit angenommen, daß sie durch eine weitere exponentielle Transition mit Schaltrate α_{ij} bestimmt sind.

Die Reduktion besteht nun aus dem Vertauschen der beiden Konfliktebenen, so daß der (in jedem Zweig zu durchlaufende) Konflikt zwischen den m exponentiellen Transitionen dem Konflikt unter den n zeitlosen Transitionen vorangestellt wird. Um für jede Schaltkombination aus exponentieller und zeitloser Transition zu demselben Folgeverhalten zu gelangen wie im Originalnetz, müssen die das Nachfolgeverhalten repräsen-

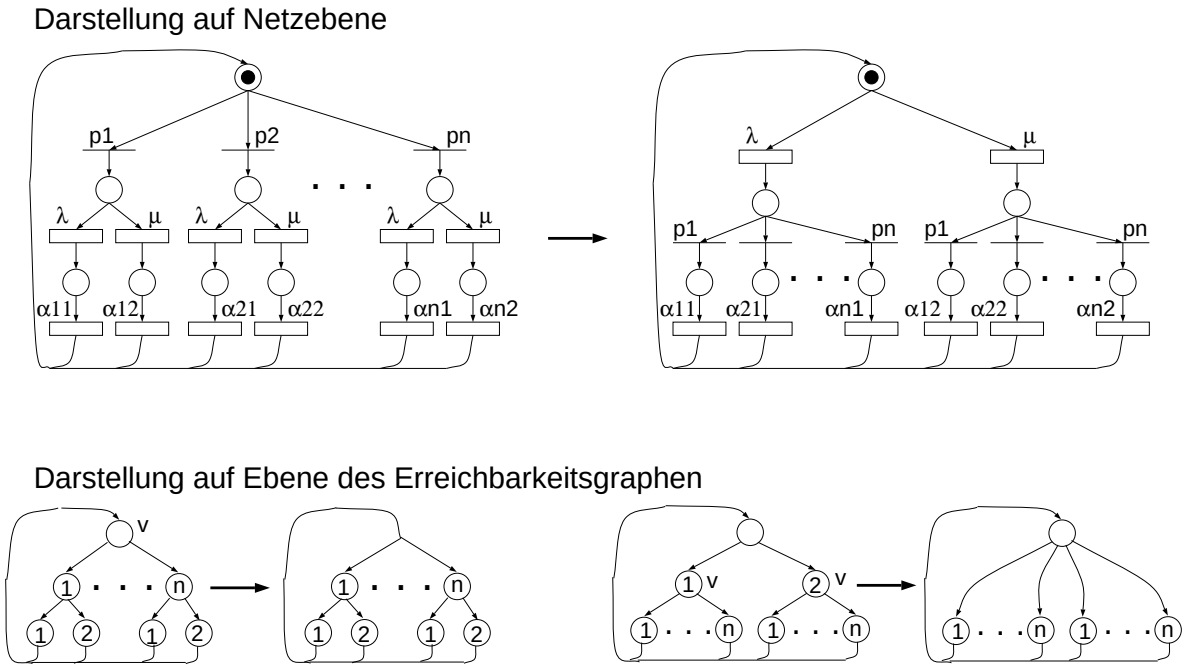


Abbildung 3.5: Vertauschen von zeitlosem mit zeitbehaftetem Konflikt

tierenden Subnetze entsprechend permutiert werden, wie dies in der Abbildung für die Transitionen mit den Schaltraten α_{ij} angedeutet ist.

Im unteren Teil von Abb. 3.5 sind die Erreichbarkeitsgraphen des Ausgangsnetzes und des reduzierten Netzes sowohl vor als auch nach dem Entfernen der verschwindenden Markierungen dargestellt (verschwindende Markierungen sind durch ein "v" gekennzeichnet). Wir beobachten, daß die Anzahl stabiler Netzmarkierungen, die ja der Zustandszahl der zugehörigen Markovkette entspricht, von $n + mn = n + 2n$ auf $1 + mn = 1 + 2n$ reduziert wird. Mit Hilfe dieser Reduktionsregel wird also erstmals die zu einem GSPN gehörende Markovkette reduziert, und zwar um $n - 1$ Zustände.

Zusammenfassende Wertung der Reduktionsregeln für verallgemeinerte stochastische Petrinetze

Wir stellen fest, daß mit den in diesem Abschnitt betrachteten Reduktionsregeln eine graphische Vereinfachung des Netzes bzw. eine Elimination von verschwindenden Zuständen auf Netzebene erreicht wird. Während eine einfachere graphische Darstellung "nur" dem Benutzer die Übersicht über das Modell erleichtert, bringt der zweite Punkt unter Umständen wertvolle Vorteile für die Analyse: Gerade bei Modellen mit einer großen Anzahl verschwindender Markierungen ist deren frühzeitige Eliminierung (auf Netzebene) wünschenswert, da dann im Rahmen der Erreichbarkeitsanalyse für die verschwindenden Markierungen kein Speicherplatz benötigt wird. Wir haben außerdem eine Reduktionsregel vorgestellt, die auch eine Verringerung der Anzahl der Zustände der zugehörigen Markovkette bewirkt. Diese "starke" Reduktionsregel ist zwar nur in einem recht speziellen Kontext anwendbar, kann jedoch im konkreten Einzelfall zu einer wertvollen Reduktion des Zustandsraums beitragen.

Um den tatsächlichen Gewinn, der durch die Anwendung der vorgestellten Reduktionsregeln entsteht, zu bewerten, ist die folgende Abwägung notwendig: Wie groß ist der

Aufwand für die Netztransformation? In welcher Relation stehen der Aufwand für die Anwendung einer bestimmten Reduktionsregel und die dadurch erzielte Einsparung? Nur wenn der Aufwand ein bestimmtes Maß nicht übersteigt, wird sich die Anwendung einer Reduktionsregel lohnen. Um diese Frage positiv zu beantworten, müssen noch Algorithmen entwickelt werden, die systematisch und effizient nach reduzierbaren Substrukturen eines GSPNs suchen.

Die Menge der in diesem Abschnitt betrachteten Reduktionsregeln stellt keinen Anspruch auf Vollständigkeit. Wie die Beispiele zeigen, fällt der durch die Netzreduktionstechnik zu erzielende Gewinn aber in der Regel eher bescheiden aus, so daß man nicht erwarten kann, mit dieser Methode das Zustandsraumproblem ganz in den Griff zu bekommen.

Da die Vereinfachungen speziell auf verallgemeinerte stochastische Petrinetze zugeschnitten sind, kann man die Regeln nicht ohne weiteres auf andere Modellbeschreibungsmethoden übertragen. Hinter manchen Reduktionsregeln, wie z.B. hinter der Vertauschung von einem zeitlosen mit einem zeitbehafteten Konflikt, stecken aber durchaus allgemeine Überlegungen (Ausnutzung von Symmetrien), die sich auch im Zusammenhang mit anderen Modellierungsmethoden als tragfähig erweisen.

3.2.2 Vereinfachung von Prozeßtermen

In der Dissertation von Hillston [Hillston, 1994] wird die stochastische Prozeßalgebra PEPA entwickelt. Die Definition und Diskussion verschiedener Äquivalenzbegriffe (Isomorphie, Bisimulation, starke Äquivalenz) für diese Sprache legt den Grundstein für die Vereinfachung von Prozeßtermen. Es wird vorgeschlagen, Prozeßterme zu vereinfachen, indem eine Komponente durch eine äquivalente ersetzt wird, welche in einem kleineren Markovprozeß resultiert. Leistungsmaße sind in Form sogenannter Rewards angegeben und bleiben von der Vereinfachung unberührt.

Wesentlicher Bestandteil einer jeden Algebra sind Rechenregeln, mit deren Hilfe Umformungen an Termen vorgenommen werden können, so daß bestimmte Eigenschaften der Terme invariant bleiben. Ein solcher Kalkül sollte also auch für stochastische Prozeßalgebren zur Verfügung stehen. Er wurde mit der Axiomatisierung der Sprache TIPP erstmals realisiert [Hermanns und Rettelbach, 1994]. Für die stochastische Prozeßalgebra TIPP wurde dazu ein Axiomensystem angegeben, von dem die Autoren zeigten, daß es korrekt und vollständig ist. Die Korrektheit wurde über die Bisimulationsäquivalenz der den Termen zugehörigen Transitionssysteme nachgewiesen. Unter Anwendung der Axiome lassen sich nun Prozeßterme auf sprachlicher Ebene umformen und dadurch vereinfachen. Auch hier ist es das Ziel, äquivalente Prozeßterme zu erzeugen, die in einem möglichst kleinen Markovprozeß resultieren. Wie bei den Reduktionsregeln für verallgemeinerte stochastische Petrinetze stellt sich hier ebenfalls das Problem der effizienten algorithmischen Suche nach zu vereinfachenden Modellkonstrukten.

3.3 Möglichkeiten der strukturierten Modellbeschreibung

3.3.1 Modularität bei stochastischen Automaten und stochastischen Prozeßalgebren

Eine naheliegende Möglichkeit, Modelle strukturiert zu beschreiben, ist das modulare Zusammensetzen des Gesamtmodells aus Submodellen. Ein solcher modularer Aufbau des Gesamtmodells bringt für den Modellierer gegenüber einer unstrukturierten Modellbeschreibung u.a. folgende Vorteile:

- Die *Übersichtlichkeit* des Modells wird erhöht. Beim Modellierungsvorgang können Submodelle getrennt und unabhängig voneinenader spezifiziert werden.
- Bei der Betrachtung des Gesamtmodells kann eine *Abstraktion* vom internen Verhalten der einzelnen konstituierenden Module erfolgen. Auch dadurch werden globale Zusammenhänge für den Modellierer leichter verständlich, da er nicht mehr alle Details des Modells auf einmal überschauen muß.
- Die Modularisierung bringt eine leichtere *Modifizierbarkeit* des Modells mit sich. Teile des Gesamtmodells können durch einfaches Ersetzen von Modulen geändert werden, ohne daß dadurch die Gesamtarchitektur des Modells in Frage gestellt wird.
- Bestimmte oft wiederkehrende Konstrukte können dem Modellierer in Form *vordefinierter Module* bereitgestellt werden (z.B. in einer Bibliothek), wodurch die Modellerstellung erleichtert wird.

Von den Modellbeschreibungsmethoden, die in Abschnitt 2.2 besprochen wurden, erfüllen die stochastischen Automaten (SANs) und die stochastischen Prozeßalgebren die Anforderungen an eine modulare Modellbeschreibung. Bei ersteren besteht das Gesamtmodell aus mehreren stochastischen Automaten, welche über synchronisierende Ereignisse interagieren. Es kommt so zu einer natürlichen Unterscheidung zwischen internen Ereignissen eines Submodells und synchronisierenden, globalen Ereignissen, welche alle Automaten betreffen, die einen Übergang mit dem betreffenden Ereignisnamen beinhalten. Synchronisierende Ereignisse drücken die Abhängigkeiten zwischen Submodellen aus.

Ähnlich liegen die Dinge bei den stochastischen Prozeßalgebren. Hier entsteht das komplexe Gesamtmodell durch systematisches Zusammenschalten von einfachen Prozeßtermen mit Hilfe von Kompositionsoperatoren, wird also das Konzept der Kompositionalität unterstützt. Beim Parallelоperator $||_S$ entscheidet die Synchronisationsmenge S darüber, ob eine bestimmte Aktion in dem gegebenen Kontext ein internes oder ein externes Ereignis bedeutet. Die an einer Synchronisation beteiligten Partner sind dabei frei bestimmbar, d.h. ein und dieselbe Aktion kann an verschieden Stellen des Prozeßterms einmal ein internes, das andere mal ein synchronisierendes Ereignis darstellen. Stochastische Prozeßalgebren verfügen somit in diesem Punkt über eine größere Flexibilität als stochastische Automaten. Außerdem bieten sie über ein modulares Vorgehen hinaus die Möglichkeit der hierarchischen Modellierung.

Es ist wichtig, ein modulares Vorgehen bei der Modellierung gegenüber der Dekomposition von Modellen scharf abzugrenzen. Die Dekomposition eines fertig vorgefundenen Modells kann große Schwierigkeiten bereiten, da auf dieser Ebene nicht bekannt ist,

an welchen Stellen mögliche Trennlinien zwischen Teilen des Modells zu finden sind. Ferner ist es schwierig, Abhängigkeiten zwischen einmal bestimmten Teilen zu formulieren (siehe z.B. [Ciardo und Trivedi, 1993]). Im Gegensatz dazu kann Modularität beim Modellentwurf auf einfache Weise ingenieurmäßig unterstützt werden und bringt dabei die oben genannten Vorteile mit sich (vgl. [Siegle, 1994c]).

3.3.2 Hierarchische Multi-Paradigmen-Modelle

Die Methode der hierarchischen Modellierung wurde in einigen Arbeiten mit der Idee verbunden, Teile des Gesamtmodells mit unterschiedlichen Methoden zu beschrieben. Man kann sich leicht vorstellen, daß je nachdem, was ein Submodell darstellen soll, dafür die eine oder die andere Modellierungsmethode zu bevorzugen ist.

Eine Kombination von Produktform-Warteschlangennetzen und Petrinetzen (GSPNs) wird in [Balbo *et al.*, 1986] und [Balbo *et al.*, 1988] beschrieben. Hier wird ein hierarchisches Modell aufgebaut, dessen Submodelle zunächst isoliert mit fest vorgegebenen Populationen analysiert werden. Es folgt ein Aggregierungsschritt, in dem Submodelle durch flußäquivalente Server (flow-equivalent server) ersetzt werden. Daran schließt sich die Analyse des auf diese Weise reduzierten Gesamtmodells an. Durch die nachfolgende Berücksichtigung der bedingten Wahrscheinlichkeiten in den Submodellen gelangt man im allgemeinen zu einer Näherung der exakten Lösung, die umso besser ist, je stärker die einzelnen Submodelle entkoppelt sind. Hat das Gesamtmodell Produktformeigenschaften, so ist die gewonnene Lösung sogar exakt.

In den Arbeiten von Buchholz [Buchholz, 1991; Buchholz, 1992b] werden ebenfalls hierarchische Modelle verwendet, deren Submodelle mit unterschiedlichen Modellierungsmethoden spezifiziert sein können. Betrachtet man der Einfachheit halber ein zweistufiges Modell, so wird zuerst der Zustandsraum für die obere Ebene generiert, d.h. alle vom Ausgangszustand aus erreichbaren Populationsverteilungen über die Submodelle. Jeder solche Populationsvektor entspricht einem Makrozustand. Dann werden für jedes Submodell und jede Population die zugehörigen Mikrozustände und Matrizen erzeugt. Dies kann für alle Submodelle unabhängig voneinander und daher parallel geschehen. Aus diesen Matrizen wird dann die Generatormatrix des Gesamtmodells mit Hilfe von Tensoroperationen aufgebaut, wodurch viel Speicherplatz eingespart wird. Eine solche Darstellung der Generatormatrix nennt man Tensordeskriptor (siehe Abschnitt 4.4.1).

Es handelt sich also um einen ähnlichen Ansatz wie derjenige von Plateau für stochastische Automaten [Plateau, 1985]. Die Unterschiede liegen darin, daß bei Buchholz eine asynchrone Interaktion der Submodelle vorliegt (Tokens/Aufträge werden in Stellen/Warteschlangen plaziert). Im Gegensatz dazu liegen bei den stochastischen Automaten synchronisierende Ereignisse vor, also der gleichzeitige Zustandsübergang in mehreren Automaten. Für die numerische Lösung werden in beiden Fällen gängige Iterationsverfahren so angepaßt, daß sie auf der Tensordarstellung arbeiten können (siehe Abschnitt 3.5.2).

Im Zusammenhang mit Multiparadigmenmodelle ist auch das Multiparadigmentool SHARPE zu erwähnen, das eine hierarchische hybride Modellierung ermöglicht [Sahner und Trivedi, 1987].

3.3.3 Strukturierung erweiterter stochastischer Petrinetze

Stochastic Well Formed Coloured Nets

Farbige (bzw. High Level) Petrinetze basieren auf folgender Grundidee: Wenn Subnetze eines Petrinetzes gleichartiges oder ähnliches Verhalten repräsentieren, werden sie aufeinander gefaltet. Um dennoch zwischen den Einzelvorgängen in den ursprünglichen Subnetzen unterscheiden zu können, führt man farbige Tokens ein, d.h. man nimmt eine Einteilung der Tokens in Klassen vor. Man erreicht durch diese Vorgehensweise eine kompaktere graphische Repräsentation des Netzes. Farbige Tokens und aus solchen aufgebaute Tupel werden im folgenden auch *Objekte* genannt.

Von Chiola et al. wurde ein Formalismus für die Leistungsbewertung mit vollständig durchstrukturierten und durchtypisierten farbigen Petrinetzen entwickelt, die Stochastic Well Formed Coloured Nets (SWCN) [Chiola und Franceschinis, 1989; Chiola *et al.*, 1990a; Chiola *et al.*, 1990b]. Diesen Arbeiten liegen Vorarbeiten von Dutheillet und Haddad über nicht zeitbehaftete reguläre farbige Petrinetze zugrunde [Dutheillet und Haddad, 1989a; Dutheillet und Haddad, 1989b; Dutheillet, 1992]. SWCNs entstehen aus allgemeinen Stochastic High Level Nets (SHN) durch Restriktionen in der Syntax, wodurch die algorithmische Erkennung und Ausnutzung von speziellen Netz-, Markierungs- und Prädikatstrukturen ermöglicht wird. Sie haben aber nachweisbar dieselbe Modellierungsmächtigkeit wie die allgemeine Klasse der SHN. Nachfolgend werden die wichtigsten Elemente und Eigenschaften dieser Modellbeschreibungsmethode genannt:

- Farben sind in Klassen und Unterklassen eingeteilt. Verschiedene Objekte werden sich i.a. gleichartig verhalten, falls sie zur selben Klasse gehören. Unterklassen können geordnet sein, so daß zwischen ihren Elementen eine zyklische Vorgänger-Nachfolgerbeziehung besteht. Diese Eigenschaft kann zur Modellierung von Reihenfolgebeziehungen (z.B. Paketnummern) verwendet werden.
- Jeder Stelle und jeder Transition ist ein Farbenbereich (colour domain) zugeordnet. Er bestimmt, welche Farben in einer Stelle zulässig sind, bzw. für welche Farbe von Objekten eine Transition schalten kann.
- Transitionen können mit Wächter-Prädikaten (guards) versehen sein, die komplexe Schaltbedingungen ausdrücken.
- Kanten sind mit Farbfunktionen (colour functions) beschriftet. Diese geben an, welche Typen von Objekten beim Schalten einer Transition von einer Stelle abgezogen bzw. einer Stelle hinzugefügt werden. Sie sind aus Standardfunktionen aufgebaut, mit denen spezielle Objekte ausgewählt werden können.

Die Vorteile dieser Beschreibungsmethode liegen in der Netzanalyse, also beim Aufbau des Erreichbarkeitsgraphen und der Generierung der zugehörigen Markovkette: Die Strukturinformation (insbesondere Information über Symmetrien) ist impliziter Bestandteil der Modellsyntax in Form der Farbbereiche und Kantenfunktionen, sie muß also nicht vom Modellierer “hineininterpretiert” werden. Daher kann diese Information bei der Analyse durch das Konzept des symbolischen Erreichbarkeitsgraphen (siehe Abschnitt 3.6) automatisch gewinnbringend genutzt werden.

Stochastic Activity Networks

Von Sanders und Meyer wurde eine Erweiterung stochastischer Petrinetze entwickelt, die sogenannten Stochastic Activity Networks [Sanders und Meyer, 1991]. Außer Stellen (places) und Transitionen (activities) stehen in diesen Netzen noch die beiden Primitive Eingangstor (input gate) und Ausgangstor (output gate) zur Verfügung. Mit ihnen lassen sich auf bequeme Weise komplexe Schaltbedingungen und Schaltaktivitäten von Transitionen beschreiben.

Die Modellwelt der Stochastic Activity Networks beinhaltet auch einen formalen Rahmen zur Definition gewünschter Leistungsindizes. Dazu wird das allgemeine Konzept der Reward-Strukturen herangezogen und zu einem speziellen, auf die Modellwelt der Stochastic Activity Networks zugeschneiderten Reward-Mechanismus angepaßt. Neben den üblichen zustandsorientierten Rewards werden auch aktivitätsorientierte Maße berücksichtigt.

Die Konstruktion eines Gesamtmodells geschieht durch Anwendung der beiden Operationen Replikation (replicate) und Zusammenfügen (join). Bei der Replikation wird ein Netz mehrfach repliziert, wobei eine ausgezeichnete Teilmenge von Stellen nicht repliziert wird, also allen Replikaten gemeinsam bleibt, und so die Verbindung zwischen den Submodellen herstellt. Das Zusammenfügen zweier oder mehrerer Subnetze geschieht ebenfalls über das Verschmelzen von Stellen. Mit Hilfe dieser beiden Operationen können Subnetze über mehrere Ebenen hinweg hierarchisch miteinander verknüpft werden, wodurch das Gesamtmodell eine Baumstruktur erhält.

Das unter Ausnutzung der Modell- und Rewardstruktur entwickelte Verfahren zur Generierung des reduzierten Zustandraums eines Stochastic Activity Networks wird in Abschnitt 3.6 angesprochen.

3.4 Umgang mit dem Zustandsraumproblem beim Übersetzungsvorgang

In den beiden folgenden Abschnitten wird beschrieben, wie während des Übersetzungsvorgangs von der Modellbeschreibung auf höherer Ebene in die zugehörige Markovkette darauf geachtet werden kann, daß sowohl Zwischendarstellungsformen (wie z.B. der Erreichbarkeitsgraph eines stochastischen Petrinetzes) als auch der Zustandsraum der schließlich entstehenden Markovkette möglichst klein bleiben. In Abschnitt 3.4.1 wird dazu auf die frühzeitige Elimination zeitloser Transitionen bei verallgemeinerten stochastischen Petrinetzen eingegangen. In Abschnitt 3.4.2 wird eine neuartige Semantik für stochastische Prozeßalgebren vorgestellt, mit deren Hilfe Prozeßterme direkt in die Transitionsratenmatrix der zugehörigen Markovkette überführt werden. Dieser Abschnitt geht insofern über die Betrachtung des reinen Übersetzungsproblems hinaus, als in ihm auch eine wesentliche Erweiterung der Prozeßalgebra TIPP, nämlich um einen Operator zur Darstellung von replizierten Teilmodellen, eingeführt wird.

3.4.1 Elimination zeitloser Transitionen bei GSPN während der Erreichbarkeitsanalyse

Bei solchen Petrinetzen vom Typ GSPN, die eine große Zahl verschwindender Markierungen haben, ist es sinnvoll, diese im Verlauf der stationären oder transienten Analyse so früh wie möglich zu eliminieren, da sie ohnehin die Wahrscheinlichkeit Null zugeordnet bekommen. Gewöhnlich geschieht das Eliminieren erst nach Aufstellen des gesamten Erreichbarkeitsgraphen. Möglich ist aber auch die Eliminierung der zeitlosen Transitionen bereits auf Netzebene. Wir haben im Abschnitt 3.2 gesehen, daß die Vereinfachung von GSPN-Netzen derartige Vereinfachungen leisten kann.

Schließlich ist als dritte Möglichkeit eine Elimination *während* des Aufbauens des Erreichbarkeitsgraphen denkbar [Ciardo *et al.*, 1991]. Dabei kann man wie folgt vorgehen: Wird bei der Erzeugung neuer Markierungen eine verschwindende Markierung gefunden, so benutzt man die Schaltwahrscheinlichkeiten aller in dieser Markierung schaltbaren zeitlosen Transitionen, um die Rate in die verschwindende Markierung auf ihre Nachfolgermarkierungen aufzuteilen. Dies ist in Abb. 3.6 (a) dargestellt, wobei stabile Markierungen durch ein “s”, verschwindende Markierungen durch ein “v” gekennzeichnet sind.

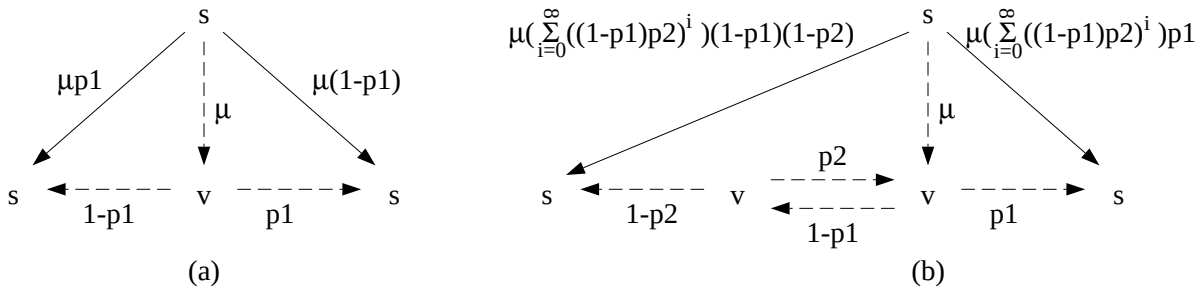


Abbildung 3.6: Die durchgezogenen Pfeile zeigen das Aufteilen der Rate in eine verschwindende Markierung auf die Nachfolgermarkierungen.
(a) einfacher Fall, (b) Schleifen zwischen verschwindenden Markierungen.

Bei diesem Verfahren werden lediglich die stabilen Markierungen permanent gespeichert, während die Information über verschwindende Markierungen so bald wie möglich gelöscht wird.

Die Vorteile dieser Methode liegen in dem reduzierten Speicherbedarf für den Erreichbarkeitsgraphen. Man erkauft sich dies durch einen unter Umständen höheren Rechenaufwand, da ein und dieselbe verschwindende Markierung von verschiedenen stabilen Markierungen aus erreichbar sein kann, im Verlauf der Berechnung also mehrfach auftauchen kann. In diesem Fall müßte die Ermittlung der Nachfolgezustände der verschwindenden Markierung und die entsprechende Aufteilung der Rate mehrmals erfolgen. Ein weiterer Nachteil sind Schwierigkeiten, die sich ergeben wenn Schleifen zwischen verschwindenden Markierungen existieren, siehe Abb. 3.6 (b). Dann sind komplizierte Datenstrukturen notwendig.

3.4.2 Matrix-Semantik für stochastische Prozeßalgebren

Für TIPP^{MS}, eine modifizierte Version der in Abschnitt 2.2.2 erwähnten Prozeßalgebra TIPP, wurde von Rettelbach und Siegle eine sogenannte Matrix-Semantik entwickelt [Rettelbach und Siegle, 1994]. Die Modifikation der Sprache gegenüber TIPP besteht aus der Hinzunahme eines Replikationsoperators und aus dem Verzicht auf einen allgemeinen Paralleloperator und Ausblendeoperator. Um die Notation der im folgenden diskutierten semantischen Regeln so einfach wie möglich zu halten, sei zunächst angenommen, daß die Sprache nur über einen einzigen Aktionstyp, die Aktion a, verfügt. Die Erweiterung auf mehr als einen Aktionstyp ist auf einfache Weise möglich und wird am Ende dieses Abschnitts angesprochen, siehe Seite 63. Der Sprachumfang von TIPP^{MS} wird also durch die folgende Grammatik festgelegt

$$P \ ::= \ 0 \ \mid \ X \ \mid \ (a,r).P \ \mid \ P + P \ \mid \ recX : P \ \mid \ !_S^n P$$

Der neu in die Spache TIPP^{MS} aufgenommene Replikationsoperator $!_S^n$ beschreibt die n-fache Parallelschaltung eines Prozesses mit sich selbst, wobei S die Menge der synchronisiert stattfindenden Aktionen angibt. Er kann also mit Hilfe eines gewöhnlichen Paralleloperators $||_S$ wie folgt ausgedrückt werden.

$$!_S^n A = \underbrace{A \,||_S A \,||_S \ldots ||_S A}_{n \text{ mal}}$$

Eine Erweiterung des Sprachumfangs von TIPP^{MS} ist geplant.

Die neue Matrix-Semantik erlaubt ein direktes Übersetzen von Prozeßtermen in die Darstellung der zugehörigen Markovkette als Transitionsratenmatrix. Damit wird die aufwendige Erreichbarkeitsanalyse vermieden, welche auf dem üblichen Weg zur Gewinnung der Markovkette bei der Berechnung des beschrifteten Transitionssystems notwendig ist. Bei der Abbildung von Prozeßtermen auf die zugehörigen Ratenmatrizen wird sichergestellt, daß eine erzeugte Matrix folgende wichtigen Eigenschaften erfüllt:

- Laut Konvention wird der erste Zustand als Startzustand angenommen, und alle Zustände müssen vom Startzustand aus *erreichbar* sein. Dies wird gefordert, um zu vermeiden, daß für unerreichbare Zustände Speicherplatz vergeudet wird.
- Es wird nicht vorausgesetzt, daß die Matrizen eine irreduzible Markovkette beschreiben. Vielmehr sind *reduzible* Markovketten, also solche mit absorbierenden Zuständen bzw. absorbierenden Teilmengen von Zuständen erlaubt.
- Die Matrix beschreibt eine Markovkette, die nicht weiter zusammenfaßbar ist (siehe Abschnitt A.2), die also in diesem Sinne *minimal* ist.
- Da stochastische Prozeßalgebren auch die qualitative Analyse unterstützen sollen, sind Schleifentransitionen, d.h. Transitionen von einem Zustand zurück zu sich selbst, sinnvoll. Daher sind die verwendeten Matrizen nicht Generatormatrizen, sondern haben nichtnegative Diagonalelemente.

Die Matrix-Semantik enthält die beiden folgenden Definitionen, die die Zuweisung von den elementaren Prozeßtermen 0 (gestoppter Prozeß) und X (Prozeßvariable) zu elementaren 1×1-Matrizen beschreiben:

$$Q_0 := (0)$$
$$Q_X := (X)$$

Mit Q_A wird allgemein die dem Prozeßterm A durch die Matrix-Semantik zugeordnete Matrix bezeichnet.

Die komplexen Prozeßtermen entsprechenden Transitionsmatrizen werden aus diesen elementaren Matrizen mit Hilfe von semantischen Regeln für die Operatoren der Sprache aufgebaut. Die Semantik eines Operators der Sprache beschreibt also die Art der Verknüpfung der den Operanden zugeordneten Matrizen. Daher spricht man auch von einer *kompositionellen Semantik*. Die Regeln für Präfixoperator, Auswahloperator, Rekursion und Replikation seien hier jedoch nicht ausführlich und nur exemplarisch dargestellt.

Präfixoperator

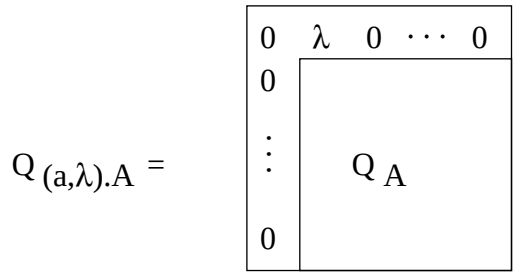


Abbildung 3.7: Matrix-Semantik des Präfixoperators

Die Semantik für den Präfixoperator, also das Vorschalten einer Aktion (a, λ) vor einen Prozeß A, ist in Abb. 3.7 dargestellt. Dabei wird angenommen, daß die zum Prozeßterm A gehörende Matrix Q_A schon bekannt ist, d.h. in vorausgegangenen Schritten ermittelt wurde. Die Matrix für den Term $(a, \lambda).A$ ergibt sich aus Q_A durch Erweiterung um eine Zeile und eine Spalte, deren einziges von Null verschiedenes Element ein λ in der Position $(1, 2)$ ist. Dieses λ repräsentiert die Rate vom neuen Anfangszustand in den Nachfolgezustand, den alten Anfangszustand des Prozesses A. Wegen der momentanen Beschränkung der Sprache TIPP^{MS} auf einen einzigen Aktionstyp (die Aktion a) tritt der Name der Aktion in den Matrizen nicht explizit auf, sondern steht implizit fest.

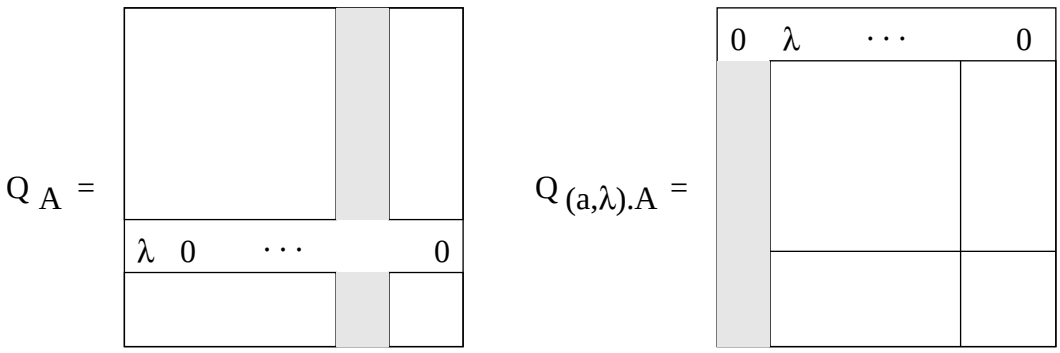


Abbildung 3.8: Semantik des Präfixoperators bei zusammenfaßbaren Zuständen

Es ist möglich, daß die nach dieser einfachen Regel konstruierte Matrix eine zusammenfaßbare Markovkette beschreibt. Man kann leicht zeigen, daß dies genau dann der Fall

ist, wenn der Prozeß A einen Zustand enthält, von dem aus ein Übergang nur in den Anfangszustand von A möglich ist, und zwar mit derselben Rate λ. Dieser Zustand kann dann mit dem neu hinzukommenden Zustand zusammengefaßt werden. In diesem Fall muß zur Erzeugung der Matrix $Q_{(a,\lambda).A}$ lediglich die Matrix Q_A permutiert werden, so daß der betreffende Zustand an die erste Stelle gelangt. Dies ist in Abb. 3.8 dargestellt.

Auswahloperator

Dieser Operator beschreibt eine Auswahl zwischen zwei alternativen Verhalten A und B. Die erste stattfindende Aktion bestimmt, ob sich der Prozeß in der Folge wie A oder wie B verhält. Der Grundgedanke für die Matrix-Semantik des Auswahloperators ist in Abb. 3.9 dargestellt.

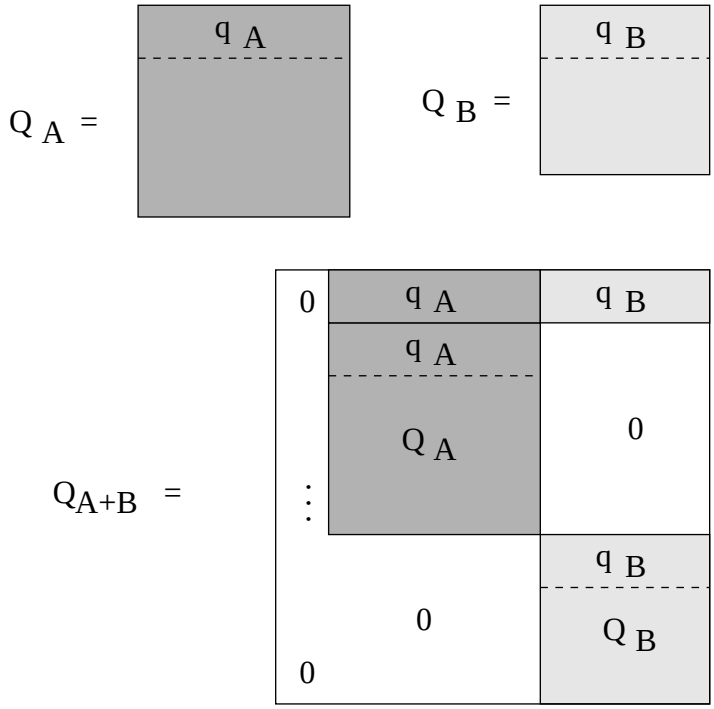


Abbildung 3.9: Grundidee der Matrix-Semantik für den Auswahloperator

Im Anfangszustand sind alle Transitionen möglich, die in A oder in B stattfinden können. Daher ist die erste Zeile von Q_{A+B} aus den ersten Zeilen q_A und q_B der Operandenmatrizen zusammengesetzt. Die gesamte erste Spalte ist mit Nullelementen besetzt, da der Prozeß sich nach einmal erfolgter Entscheidung in der Folge entweder wie A oder wie B verhält und niemals zum Ausgangszustand zurückkehren kann.

Damit ist jedoch nur das Grundprinzip der Semantik für den Auswahloperator beschrieben. Die so erzeugte Matrix Q_{A+B} erfüllt nicht notwendigerweise die oben geforderten Eigenschaften der Erreichbarkeit und Minimalität. Es zeigt sich, daß die Anfangszustände der Prozesse A und B in der resultierenden Matrix Q_{A+B} unerreichbar sein können. Dies ist genau dann der Fall, wenn die erste Spalte der Matrix Q_A (Q_B) nur Nullelemente enthält. Trifft dies zu, so kann der vormalige Anfangszustand von A (B) eliminiert werden.

Schwerer wiegt die Tatsache, daß Q_{A+B} Teilmengen von zusammenfaßbaren Zuständen beinhalten kann. Dieser Fall tritt ein, wenn A und B gemeinsames Teilverhalten aufweisen. Um dieses Problem in den Griff zu bekommen, wird eine Normalform für Transitionsratenmatrizen definiert, die auf der "Inselstruktur" der Matrizen basiert. Als Inseln werden dabei Teilmengen gegenseitig erreichbarer Zustände bezeichnet. Die Verwendung dieser Normalform erleichtert das Lokalisieren gemeinsamen Teilverhaltens wesentlich. Für das Erkennen und Zusammenfassen des gemeinsamen Verhaltens von A und B wird ein entsprechender Algorithmus angegeben, welcher die Normalform-eigenschaften ausnutzt, aber trotzdem recht aufwendig ist.

Rekursionsoperator

Wir wollen an dieser Stelle nur den Grundgedanken der Matrix-Semantik für den Rekursionsoperator beschreiben, da dieser Abschnitt ja nur exemplarisch das Prinzip der Matrix-Semantik zeigen soll. Um die zum Prozeßterm $recX : A$ gehörende Matrix zu bestimmen, wird angenommen, daß die Matrix Q_A für den Prozeß A bereits vorliegt. Uns interessieren hier nur die Fälle, in denen in A die Prozeßvariable X vorkommt, da andernfalls der Rekursionsoperator keinen Effekt hat. Die allgemeine Struktur der Matrix Q_A ist in Abb. 3.10 (links) dargestellt. Ohne Beschränkung der Allgemeinheit kann man annehmen, daß sich der Eintrag X in der letzten Position befindet. Mehr als einen X-Eintrag kann eine Matrix in Normalform nicht enthalten, da mehrere X-Einträge beim Aufbau von Q_A als identisches Verhalten erkannt und daher zusammengefaßt würden.

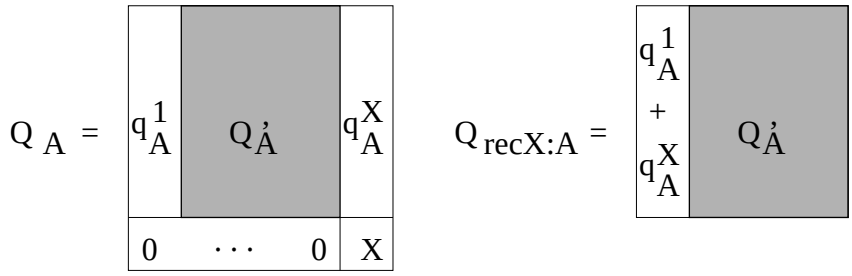


Abbildung 3.10: Grundidee der Matrix-Sematik für Rekursion

Der Rekursionsoperator bedeutet informell, daß immer dann, wenn innerhalb des Prozesses A die Prozeßvariable X auftaucht, der Prozeß in seinen Anfangszustand zurückspringt. Deshalb entsteht die in der rechten Hälfte der Abbildung wiedergegebene Matrix $Q_{recX:A}$ aus Q_A durch Aufaddieren der letzten Spalte q_A^X auf die erste Spalte q_A^1 und anschließendes Streichen der letzten Zeile und Spalte. Dies läßt sich so interpretieren, daß ein Übergang in den Zustand X gleichbedeutend ist mit einem Übergang in den Anfangszustand des Prozesses.

Auch hier sind über das dargestellte Grundprinzip hinaus weitergehende Überlegungen in bezug auf Minimalität und Normalform der entstehenden Matrix $Q_{recX:A}$ notwendig, worauf an dieser Stelle aber nicht eingegangen werden soll.

Replikationsoperator

Zur Ermittlung der Transitionsratenmatrix $Q_{!^n_S A}$ für den Replikationsoperator macht man sich dieselben Zusammenhänge zunutze, die später in Abschnitt 4.6 zur Reduktion des kombinierten Zustandsraums replizierter Submodelle verwendet werden.

Nimmt man zunächst an, daß keine Synchronisation stattfindet, daß also gilt $S = \emptyset$, so wird das Verhalten des n-fach replizierten Prozesses A durch die Matrix

$$Q = \underbrace{Q_A \oplus Q_A \oplus \ldots \oplus Q_A}_{n \text{ mal}}$$

beschrieben. Dabei bezeichnet der Operator $\oplus$ die Tensorsumme von Matrizen (siehe Anhang B). Die so definierte Matrix Q beschreibt für $n > 1$ eine zusammenfaßbare Markovkette, ist also in diesem Sinne nicht minimal. Unter der Annahme, daß Q_A die Dimension s hat, folgt, daß Q von der Dimension s^n ist.

Zur Identifizierung äquivalenter Zustände betrachten wir nun eine Tupeldarstellung. Die s^n Zustände von Q können wie folgt mit n-Tupeln aus den Ziffern $\{0, 1, \ldots, s-1\}$ in aufsteigender lexikographischer Reihenfolge durchnumeriert werden:

$$\begin{array}{rcl}
\text{Zustand } 0 & = & (\ 0\ ,\ 0\ ,\ \ldots\ ,\ 0\ ) \\
\text{Zustand } 1 & = & (\ 0\ ,\ 0\ ,\ \ldots\ ,\ 1\ ) \\
 & \vdots & \\
\text{Zustand } s^n - 1 & = & (\ s-1\ ,\ s-1\ ,\ \ldots\ ,\ s-1\ )
\end{array}$$

Im allgemeinen bezeichnet das Tupel $(i_1, i_2, \ldots, i_n)$ einen Zustand, in dem sich das k-te Replikat des Prozesses A im Zustand i_k befindet, $k = 1, \ldots, n$. Aus Symmetriebetrachtungen ist bekannt, daß alle Zustände, deren Tupeldarstellungen auseinander durch beliebige Permutation hervorgehen, eine Teilmenge äquivalenter (und deshalb zusammenfaßbarer) Zustände darstellen. Insgesamt existieren $\binom{n+s-1}{s-1}$ solche Teilmengen äquivalenter Zustände.

Daher hat die minimale Matrix $Q_{!^n_\emptyset A}$, die das Verhalten von $!^n_\emptyset A$ beschreibt, $\binom{n+s-1}{s-1}$ Zustände. Jeder Zustand von $Q_{!^n_\emptyset A}$ repräsentiert eine Teilmenge von äquivalenten, also zusammenfaßbaren Zuständen der Matrix Q. Als Repräsentant für eine solche Teilmenge wird das lexikographisch kleinste Element ausgewählt. Zum Beispiel ist das Tupel 012 der Repräsentant für die Teilmenge $\{012, 021, 102, 120, 201, 210\}$. Man kann sich zwei verschiedene Wege zur Konstruktion der Matrix $Q_{!^n_\emptyset A}$ aus Q_A vorstellen,

siehe Abb. 3.11.

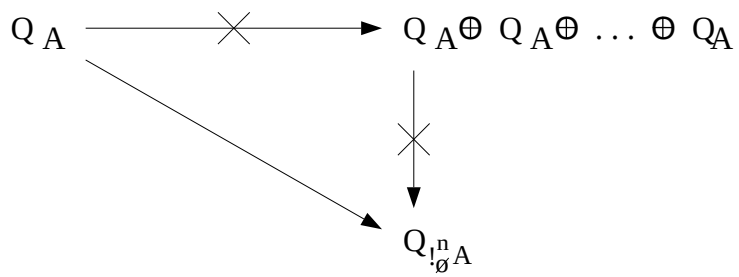


Abbildung 3.11: Zwei Wege zur Konstruktion der Matrix $Q_{!^n_\emptyset A}$

Das explizite Aufstellen der Tensorsumme und anschließendes Zusammenfassen der äquivalenten Zustände wäre extrem aufwendig. Wir haben eben gesehen, daß die Zwischenmatrix Q die Dimension s^n hat! Daher schlagen wir einen Algorithmus vor, der die reduzierte Matrix $Q_{!^n_\emptyset A}$ direkt aus der Matrix Q_A berechnet. Dieser Weg entspricht dem direkten Pfeil in Abb. 3.11.

An dieser Stelle erläutern wir den Algorithmus anhand eines Beispiels. Die formale Beschreibung eines entsprechenden Algorithmus für die Modellwelt von Kapitel 4 befindet sich in Abschnitt 4.6. In Abb. 3.12 ist eine Matrix Q_A der Dimension 3 vorgegeben. Außerdem sind auch die Matrizen $Q_{!^2_\emptyset A}$ und $Q_{!^3_\emptyset A}$ dargestellt. Man kann beobachten, daß die Einträge der beiden letztgenannten Matrizen auf einfache Weise aus den Einträgen von Q_A entstehen. Zur Verdeutlichung des Vorgehens wurden für die von Null verschiedenen Einträge der Ausgangsmatrix Q_A verschiedene natürliche Zahlen verwendet, so daß für die Einträge der zusammengesetzten Matrizen $Q_{!^2_\emptyset A}$ und $Q_{!^3_\emptyset A}$ ersichtlich wird, aus welchen Einträgen der Ausgangsmatrix sie entstanden sind.

Um die Elemente der Matrix $Q_{!^n_\emptyset A}$ allgemein formal zu spezifizieren, definieren wir zunächst die Hilfsfunktion ndiff(t_1, t_1), welche die Anzahl der Ziffern zurückgibt, in denen sich zwei Tupel t_1 und t_2 unterscheiden. Die Position einer Ziffer innerhalb des Tupels ist bei dieser Betrachtung irrelevant, da Tupel, die durch Permutation auseinander hervorgehen, äquivalente Zustände bezeichnen. Genau hier vollzieht sich also der Übergang von der Betrachtung der individuellen Replikate zu einer abstrakteren Sicht, die nur die Anzahl der sich in einem bestimmten Zustand befindlichen Replikate berücksichtigt.

Als Beispiel für die Funktion ndiff ergibt sich ndiff$(000, 001) = 1$, ndiff$(011, 112) = 1$ und ndiff$(011, 122) = 2$. Außerdem definieren wir die Gewichtsfunktion $w_i(t)$, welche die Häufigkeit des Vorkommens der Ziffer i im Tupel t angibt. Z.B. ergibt sich $w_0(001) = 2$ und $w_1(001) = 1$.

Seien nun o und t die beiden n-Tupel, welche den Ausgangs- und den Zielzustand eines Übergangs bezeichnen. Dann läßt sich die Transitionsrate $r_{o,t}$ vom Ausgangs- in den Zielzustand wie folgt allgemein schreiben:

$$r_{o,t} = \begin{cases} \sum_{i \in o} w_i(o) a_{ii} & \text{falls ndiff}(o,t) = 0 \\ w_i(o) a_{ij} & \text{falls ndiff}(o,t) = 1 \\ 0 & \text{sonst} \end{cases} \tag{3.1}$$

$Q_A =$

	0	1	2
0	1		2
1		3	4
2		5	

$Q_{!^2_{\emptyset}A} =$

	00	01	02	11	12	22
00	2x1		2x2		0	
01		1+3	4		2	
02		5	1			2
11				2x3	2x4	
12		0		5	3	4
22					2x5	

$Q_{!^3_{\emptyset}A} =$

	000	001	002	011	012	022	111	112	122	222
000	3x1		3x2		0			0		
001		2x1 +3	4		2x2			0		
002		5	2x1			2x2				
011				1+ 2x3	2x4			2		
012				5	1+3	4			2	
022					2x5	1				2
111							3x3	3x4		
112		0					5	2x3	2x4	
122								2x5	3	4
222									3x5	

Abbildung 3.12: Beispiel für die Konstruktion der Matrix $Q_{!^n_{\emptyset}A}$

Der erste der drei Fälle bezieht sich auf die Einträge auf der Diagonalen von $Q_{!^n_{\emptyset}A}$. Diese beschreiben den Fall einer Schleifentransition eines Replikats vom Zustand i in den Zustand i. Dafür wird die Rate a_{ii} mit dem Gewicht der Ziffer i im Ausgangszustand, also der Anzahl der sich im Zustand i befindlichen Replikate, multipliziert.

Im zweiten Fall unterscheiden sich Ausgangs- und Zielzustand um genau eine Ziffer (wobei wiederum die Position einer Ziffer innerhalb des Tupels irrelevant ist). Wir nehmen an, daß eine Ziffer i durch ein j ersetzt wurde. Dann ergibt sich die Transitionsrate als a_{ij} multipliziert mit dem Gewicht der Ziffer i im Ausgangszustand.

Im dritten Fall unterscheiden sich Ausgangs- und Zielzustand um mehr als eine Ziffer. Da simultane Transitionen unter Markovschen Bedingungen mit Wahrscheinlichkeit 0 auftreten, ergibt sich die Transitionsrate in diesem Fall als 0.

Bei den eben angestellten Betrachtungen waren wir davon ausgegangen, daß keine Synchronisation zwischen den Replikaten existierte, daß also $S = \emptyset$. Was geschieht

nun im Fall $S \neq \emptyset$? Unter der Annahme, daß alle durch die Matrix Q_A beschriebenen Aktionen vom Typ a sind und daß über a synchronisiert wird, ist in obiger Definition für die Matrix Q die Tensorsumme durch das Tensorprodukt zu ersetzen, man erhält also

$$Q = \underbrace{Q_A \otimes Q_A \otimes \ldots \otimes Q_A}_{n \text{ mal}}$$

Die Teilmengen zusammenfaßbarer Zustände sind hier dieselben wie im unsynchronisierten Fall und es läßt sich ein entsprechender Algorithmus angeben (siehe Abschnitt 4.6).

Wenn die Ausgangsmatrix Q_A reduzibel ist, müssen die Zustände des replizierten Modells in einer Reihenfolge entsprechend der Inselstruktur von Q_A generiert werden, damit die resultierende Matrix $Q_{!^n_\emptyset A}$ wiederum Normalform besitzt.

Mehrere Aktionstypen

Um zwischen verschiedenen Aktionen unterscheiden zu können ist es notwendig, die Sprache TIPP^{MS} so zu erweitern, daß neben der Aktion a auch andere Aktionstypen zulässig sind. Das in diesem Abschnitt dargestellte Vorgehen kann dazu wie folgt ausgebaut werden: Die Matrixeinträge der einem Prozeßterm A zugeordneten Matrix Q_A werden durch Tupel ersetzt, und zwar so, daß jede Komponente des Tupels die Übergänge bezüglich eines bestimmten Aktionstyps beschreibt. Beim Überprüfen der Äquivalenz von Zuständen muß nun die Äquivalenz bezüglich aller Aktionstypen überprüft werden, d.h. zwei Zustände können nur dann als äquivalent betrachtet werden, wenn äquivalentes Verhalten bezüglich aller Aktionstypen vorliegt.

Bei der Anwendung der semantischen Regeln für den Replikationsoperator ist ein differenziertes Vorgehen notwendig, wenn von der Replikation sowohl unsynchronisierte als auch synchronisierte Aktionen betroffen sind. Für jeden Aktionstyp, d.h. für jede einzelne Komponente eines Tupels, ist dann zu entscheiden, ob die Regel für unsynchronisierte oder die Regel für synchronisierte Aktionen anzuwenden ist. Ein solches differenziertes Vorgehen für unterschiedliche Ereignisse wird auch in der Modellwelt von Kapitel 4 benutzt.

Vorteile der Matrix-Semantik

Die bedeutenden Vorteile der Matrix-Semantik für die Sprache TIPP^{MS} gegenüber dem herkömmlichen Vorgehen bei der Generierung der Markovkette sind nicht zu übersehen. Die normalerweise notwendige aufwendige dynamische Erreichbarkeitsanalyse, bei der die in den Prozeßtermen steckende Strukturinformation nicht genutzt werden kann, entfällt zugunsten eines strukturierten Verfahrens, welches direkt zur minimalen Darstellung der zugehörigen Markovkette führt. Während das Auffinden und Zusammenfassen äquivalenter Zustände in einem unter Umständen riesigen Zustandsraum praktisch kaum durchführbar ist, leistet die neue Matrix-Semantik dies durch ein geordnetes kompositionelles Vorgehen, bei dem schrittweise minimale Matrizen aufgebaut werden.

Die eingeschränkte Sprache TIPP^{MS}, anhand derer das Konzept der Matrix-Semantik eingeführt wurde, erfüllt die Anforderungen und Wünsche der Benutzer noch nicht.

Beispielsweise ist eine Erweiterung um einen allgemeinen Parallelоperator für die Praxis unabdingbar. Ebenso wünscht man sich zur Abstraktion vom internen Verhalten eines Prozesses einen Ausblendeoperator. Die grundsätzliche Vorgehensweise bei der Matrix-Semantik, die Vorteile und die zu lösenden Probleme sind jedoch klar erkennbar.

3.5 "Intelligente" numerische Verfahren

3.5.1 Dekomposition/Aggregation

Die Idee, einen komplexen Zustandsraum durch Dekomposition beherrschbar zu machen, wurde zum ersten Mal von Simon und Ando für Systeme aus dem Bereich der Wirtschaftswissenschaften vorgeschlagen [Simon und Ando, 1961]. Einige Jahre später benutzte Courtois dieselbe Vorgehensweise für die Leistungsbewertung von Rechensystemen [Courtois, 1977]. Bei diesen Ansätzen ist statt der aufwendigen Analyse *eines komplexen* Systems lediglich die Analyse *mehrerer einfacher* Systeme erforderlich.

Grundsätzlich sind Dekompositionsverfahren für die Analyse von Markovketten dann erfolgreich einsetzbar, wenn der Zustandsraum in M Teilmengen "kommunizierender" Zustände partitioniert werden kann. Innerhalb einer jeden Teilmenge herrscht eine starke Kopplung zwischen den Zuständen. Umgekehrt sollte die Kopplung zwischen Zuständen aus unterschiedlichen Teilmengen möglichst schwach sein. Systeme, die diese Eigenschaft besitzen, bezeichnet man als *fast vollständig zerlegbar* (nearly completely decomposable, kurz NCD). Drückt man diesen Sachverhalt in bezug auf die Generatormatrix der Markovkette aus, so bedeutet dies, daß diese — entsprechend der Partitionierung des Zustandsraums in Teilmengen — eine Blockstruktur aufweisen sollte, wobei die Einträge in den Diagonalblöcken möglichst groß gegenüber den Einträgen in den Nichtdiagonalblöcken sein sollten.

Das Vorgehen der Dekomposition und Aggregation nach Simon und Ando besteht aus den drei folgenden Schritten:

1. *Dekompositionsschritt:* Die Elemente aus den Nichtdiagonalblöcken werden gestrichen und auf die von Null verschiedenen Elemente in den Diagonalblöcken aufgeteilt. Die Güte der Approximation hängt in entscheidender Weise von der Art der Aufteilung ab. Durch diesen Schritt wird das System reduzibel, es zerfällt in M unabhängige Systeme. Anschließend werden die stationären Verteilungsvektoren $v_i, i = 1, \ldots, M,$ dieser Systeme bestimmt.
2. *Aggregationsschritt:* Man betrachtet das sogenannte reduzierte System, welches die Übergänge zwischen den M Teilmengen beschreibt. Die Übergangsrate von i nach j in diesem System ergibt sich als die mit v_i gewichtete Summe der ursprünglichen Raten aus den zur Teilmenge i gehörenden Zuständen in Zustände aus der Teilmenge j. Dann bestimmt man den stationären Verteilungsvektor u des reduzierten Systems.
3. *Kombinationsschritt:* Man erhält die Approximation der Lösung des ursprünglichen Systems durch Kombination der in den beiden ersten Schritten berechneten Teillösungen als $(u_1 v_1, \ldots, u_M v_M)$. Die stationäre Wahrscheinlichkeit eines Zustands z errechnet sich also aus dem Produkt der Wahrscheinlichkeit, daß sich das System in einer bestimmten Teilmenge befindet (u_i) und der Wahrscheinlichkeit für den Zustand innerhalb der Teilmenge ($v_i(z)$).

Dieses Approximationsverfahren liefert im allgemeinen umso bessere Ergebnisse, je schwächer die Kopplung zwischen den Teilmengen ist. Für die Klasse der sogenannten reversiblen Markovprozesse liefert das Verfahren sogar exakte Ergebnisse [Conway und Georganas, 1989, S. 119], und zwar unabhängig von der gewählten Partitionierung, d.h. unabhängig davon, ob Zustände stark oder schwach gekoppelt sind. Conway und Georganas zeigen, daß dieser Sachverhalt die theoretische Grundlage vieler effizienter Analyseverfahren für Warteschlangennetze mit Produktformlösung bildet (z.B. Buzen's Algorithmus zur Bestimmung der Normalisierungskonstanten, MVA [Conway und Georganas, 1989, Chapter 4]).

Lediglich zur Gegenüberstellung und Abgrenzung sei an dieser Stelle der orthogonale Ansatz von Ciardo und Trivedi zur Dekomposition von stochastischen Petrinetzen erwähnt [Ciardo und Trivedi, 1993]. Sie identifizieren typische Netzstrukturen (wie Rendezvous, Processor Sharing, Parbegin-Parend, . . .) welche eine bestimmte Dekomposition des Netzes nahelegen. Da die entstehenden Teilnetze nicht unabhängig voneinander sind, ist ein iteratives Vorgehen notwendig, wobei in jedem Iterationsschritt jedes Teilnetz Eingabeparameter benötigt, die sich aus der Lösung anderer Teilnetze im vorangegangenen Iterationsschritt ergeben. Es werden also Netze bestehend aus fast unabhängigen Subsystemen (near independent subsystems, kurz NIS) mit Hilfe eines iterativen Verfahrens analysiert. Der zugehörige stochastische Prozeß entspricht im wesentlichen der Kombination zweier unabhängiger Markovprozesse, seine Generatormatrix ist daher fast gleich der Tensorsumme zweier kleinerer Generatormatrizen. Interessant im Hinblick auf Abschnitt 3.5.2 ist die Tatsache, daß auch hier genau solche Netzaufteilungen betrachtet werden, die sogenannten *near-independent* Markovketten entsprechen, also Markovketten, deren Generatormatrix annähernd als Tensorsumme von kleineren Matrizen dargestellt werden kann.

3.5.2 Strukturierte Analyse auf strukturierter Beschreibung der Generatormatrix

In diesem Abschnitt wird das gemeinsame Grundprinzip einer Klasse von effizienten numerischen Lösungsverfahren dargestellt, welche auf einer strukturierten Modellbeschreibung auf höherer Ebene aufbauen. Bei den hier betrachteten Verfahren wird das explizite Aufstellen der Generatormatrix des Gesamtmodells vermieden. Statt dessen benutzt man zur iterativen Berechnung der Lösung eine Darstellung der Generatormatrix in Form eines Tensordeskriptors.

Unter einem *Tensordeskriptor* versteht man eine strukturierte, speicherplatzsparende Darstellung einer Generatormatrix. Tensordeskriptoren werden mit Hilfe der Operatoren Tensorprodukt $\otimes$ und Tensorsumme $\oplus$ (siehe Anhang B) aus kleineren Matrizen aufgebaut. Neben dem Vorteil der Speicherplatzersparnis ermöglichen Tensordeskriptoren effiziente numerische Analyseverfahren, die Gegenstand des vorliegenden Abschnitts sind. Die numerischen Lösungsverfahren in den Methoden von Plateau [Plateau, 1985] und Buchholz [Buchholz, 1992b] sind beide Ausprägungen des hier in einem allgemeinen Kontext dargestellten Vorgehens. Tensordeskriptoren spielen auch in der Modellwelt von Kapitel 4 eine wichtige Rolle, siehe Abschnitt 4.4.1 und Abschnitt 4.4.2, insbesondere Gl (4.1).

Grundsätzlich ist das Vorgehen bei der Bestimmung des Vektors der stationären Zustandswahrscheinlichkeiten π mit Hilfe iterativer Verfahren so, daß die Matrixgleichung

$$\pi Q = 0$$

in eine Fixpunktgleichung der Form

$$\pi H = \pi$$

überführt wird, welche man auch als Eigenvektorproblem auffassen kann. Die Matrix H hängt funktional von der Generatormatrix Q ab, d.h. man kann schreiben $H = f(Q)$. Aus dieser Fixpunktgleichung leitet sich dann die folgende Iterationsvorschrift ab:

$$\pi^{(n)} = \pi^{(n-1)} H$$

Daher wird die Matrix H als *Iterationsmatrix* bezeichnet.

Als Beispiel betrachten wir die Iterationsmatrix für den Spezialfall der sogenannten Power-Methode. Sie ist durch den Ausdruck $H_{pow} = Q\Delta t + I$ gegeben, wobei $\Delta t < 1/(\max_i(|q_{ii}|))$ sein muß. Andere Verfahren wie das Jacobi-Verfahren, das Gauß-Seidel-Verfahren oder das SOR-Verfahren arbeiten ebenfalls mit einer aus Q abgeleiteten Iterationsmatrix.

Wir nehmen nun an, daß die Generatormatrix Q in Form eines Tensordeskriptors aus den Matrizen $A_1, A_2, \ldots, A_n$ aufgebaut ist. Wir schreiben kurz

$$Q = g(A_1, A_2, \ldots, A_n)$$

Die Funktion g stellt also die Matrix Q auf der Basis der Operanden $A_1, A_2, \ldots, A_n$ unter Verwendung der Operatoren Tensorprodukt $\otimes$ und Tensorsumme $\oplus$ dar. Um ein iteratives Verfahren so anzupassen, daß es auf der Tensordarstellung der Generatormatrix Q basiert, wird zunächst aus der Tensordarstellung von Q eine Tensordarstellung der Iterationsmatrix H gewonnen. Das Ergebnis dieses ersten Schrittes ist eine Darstellung von H als Kombination von Tensorsummen und -produkten.

$$H = f(Q) = f(g(A_1, A_2, \ldots, A_n))$$

Für die Power-Methode z.B. bereitet dieser Schritt keine Schwierigkeiten. Dagegen müssen nach [Buchholz, 1991, S. 41] die anderen genannten Verfahren teilweise modifiziert werden, um diesen Schritt vollziehen zu können.

Die in jedem Iterationsschritt auszuführende Vektor-Matrix Multiplikation $\pi^{(n-1)}H$ erfordert dann die Ausführung der beiden folgenden Operationen:

$$\pi \bigoplus_{i=1}^{n} A_i \quad \text{und} \quad \pi \bigotimes_{i=1}^{n} A_i$$

Im folgenden wird dargestellt, daß man Ausdrücke von dieser Form auf sehr effiziente Weise berechnen kann.

Zunächst betrachten wir das Produkt aus dem Vektor π und einer verallgemeinerten Tensorsumme. Für seine praktische Berechnung benutzt man die in der folgenden Gleichung dargestellten Äquivalenzen.

$$\begin{aligned}\pi \bigoplus_{i=1}^{n} A_i &= \pi \sum_{i=1}^{n} I_{l_i} \otimes A_i \otimes I_{u_i} \\ &= \pi \sum_{i=1}^{n} P_{\sigma_i}^T (I_{l_i u_i} \otimes A_i) P_{\sigma_i} \\ &= \sum_{i=1}^{n} \pi P_{\sigma_i}^T \begin{bmatrix} A_i & & & \\ & A_i & & \\ & & \ddots & \\ & & & A_i \end{bmatrix} P_{\sigma_i}\end{aligned} \tag{3.2}$$

Die verallgemeinerte Tensorsumme läßt sich zunächst als gewöhnliche Matrixsumme von Tensorprodukten der Form $I_{l_i} \otimes A_i \otimes I_{u_i}$ schreiben (siehe Anhang B, Gl. (B.2)). Die Summanden werden dann als Permutation von Matrizen das Typs $I_{l_i u_i} \otimes A_i$ dargestellt, wobei die zugehörigen Permutationsmatrizen mit $P_{\sigma_i}^T$ (Zeilenpermutation) und P_{σ_i} (entsprechende Spaltenpermutation) bezeichnet sind. Zieht man den Vektor π unter das Summationszeichen, so gelangt man zu der endgültigen Darstellung.

Das Produkt des permutierten Vektors $\pi^{\sigma_i} = \pi P_{\sigma_i}^T$ mit der Matrix $I_{l_i u_i} \otimes A_i$ ist deshalb so effizient berechenbar, weil letztere die oben angedeutete Form hat: Die Matrix besteht aus $(l_i u_i)^2$ Blöcken, wobei alle Diagonalblöcke gleich der Matrix A_i sind und alle Nichtdiagonalblöcke verschwinden. Matrizen diese Typs nennt man auch *Normalfaktoren* (normal factor). Bei der Multiplikation eines Vektors mit einem Normalfaktor sind also jeweils Teile des Vektors mit der Matrix A_i zu multiplizieren. Dieser Multiplikationsvorgang kann daher sehr einfach und parallel durchgeführt werden. Auf der äußeren Ebene der Summe können alle n Summanden unabhängig und daher parallel berechnet werden.

Sämtliche in Gl. (3.2) benutzten Permutationen müssen nicht explizit ausgeführt werden, sondern werden durch geschickte Adreßtransformationen realisiert. Ebensowenig wird man in der Praxis die Matrizen $I_{l_i u_i} \otimes A_i$ explizit aufstellen.

Auf ähnliche Weise läßt sich eine effiziente Berechnung des Produkts aus einem Vektor und einem verallgemeinerten Tensorprodukt wie folgt erreichen:

$$\begin{aligned}\pi \bigotimes_{i=1}^{n} A_i &= \pi \prod_{i=1}^{n} I_{l_i} \otimes A_i \otimes I_{u_i} \\ &= \pi P_{\sigma_1}^T (I_{l_1 u_1} \otimes A_1) P_{\sigma_1} P_{\sigma_2}^T (I_{l_2 u_2} \otimes A_2) P_{\sigma_2} \ldots P_{\sigma_n}^T (I_{l_n u_n} \otimes A_n) P_{\sigma_n} \\ &= \pi^{\sigma_1} (I_{l_1 u_1} \otimes A_1) P_{\sigma_{1,2}} (I_{l_2 u_2} \otimes A_2) P_{\sigma_{2,3}} \ldots P_{\sigma_{n-1,n}} (I_{l_n u_n} \otimes A_n) P_{\sigma_n}\end{aligned} \tag{3.3}$$

Zunächst wird das verallgemeinerte Tensorprodukt als Matrixprodukt von Matrizen das Typs $I_{l_i} \otimes A_i \otimes I_{u_i}$ geschrieben (siehe Anhang B, Gl. (B.1)). Dann folgt durch die Anwendung von Permutationen eine Umwandlung dieser Matrizen in Normalfaktoren. Schließlich wird die Hintereinanderausführung zweier Permutationen P_{σ_i} und $P_{\sigma_{i+1}}^T$ durch eine einzige Permutation $P_{\sigma_{i,i+1}}$ ersetzt.

Im Gegensatz zur oben beschriebenen Multiplikation eines Vektors mit einer verallgemeinerten Tensorsumme ist hier — da es sich um die Berechnung eines Produkts handelt — auf der äußeren Ebene ein sequentielles Vorgehen notwendig: Die beiden Schritte Adreßtransformation (zur Realisierung einer Permutation) und Multiplikation mit einem Normalfaktor werden insgesamt n mal wiederholt.

Da auch bei den numerischen Verfahren für die transiente Analyse von Markovprozessen die Vektor-Matrix Multiplikation eine wichtige Rolle spielt, können auch solche Verfahren vorteilhaft auf einer Darstellung der Generatormatrix in Form eines Tensordeskriptors aufbauen.

3.6 Ausnutzung von Modellsymmetrien in strukturierten Modellen

Moderne parallele oder verteilte Systeme sind oft aus einer großen Zahl gleichartiger oder zumindest ähnlicher Komponenten aufgebaut. Man denke z.B. an hochparallele Rechensysteme mit Hunderten oder Tausenden von identischen Prozessoren. Ein weiteres Beispiel sind lokale Rechnernetze, an welche eine Vielzahl von Workstations angeschlossen sind. Hinzu kommt, daß gleichartige Komponenten oft auch gleichartige Aufgaben wahrnehmen und daher in einem Modell durch dasselbe stochastische Verhalten nachgebildet werden können. Es kommt also oft vor, daß ein strukturiertes Modell teilweise gleichartige Komponenten (z.B. Submodelle) beinhaltet. Wir bezeichnen diese Eigenschaft als *Symmetrie*.

Symmetrien treten nicht nur bei der Modellbeschreibung auf höherer Ebene auf, sie spiegeln sich auch in der Struktur des zugehörigen Zustandsraums wider. Die sich aus den Symmetrien ergebende Strukturierung des Zustandsraums resultiert oft darin, daß dieser dann mit Hilfe des Konzepts der Zusammenfaßbarkeit (siehe Anhang A.2) reduziert werden kann.

Die Idee der Zustandsraumreduktion durch die Ausnutzung von Symmetrien ist nicht neu. Diese Technik wurde bereits in einer Vielzahl veröffentlichter Fallstudien angewendet, jedoch meist in einer auf die spezielle Anwendung zugeschnittenen Weise (z.B. [Bhandarkar, 1975; Kant, 1988]). Das bedeutet, daß in jedem Einzelfall die Intuition und Erfahrung des Modellierers benötigt wird, um Symmetrien in der Modellbeschreibung zu erkennen, die Symmetrien im Zustandsraum wiederzufinden und schließlich das Prinzip der Zusammenfaßbarkeit auszunutzen. Eine automatisierbare und dadurch auch durch Werkzeuge unterstützte Ausnutzung von Symmetrien ist auf diese Weise nicht möglich.

Erst in jüngeren Arbeiten wurde versucht, das Erkennen und die Ausnutzung von Symmetrien bereits als festen Bestandteil in das methodische Vorgehen bei der Modellierung zu integrieren. Im folgenden sind fünf solche Ansätze für ein allgemeines Vorgehen bei der Modellierung genannt, in denen die Ausnutzung von Symmetrien eine wesentliche Rolle spielt:

- In [Atif, 1992] wird eine Verbesserung der iterativen Lösungsmethode aus [Plateau, 1985] für Modelle mit Symmetrien beschrieben. Dabei wird insbesondere die Reevaluierung symmetrischer Teile während der Multiplikation eines Vektors mit

dem Tensordeskriptor vermieden. Die Darstellung des Tensordeskriptors selbst bleibt jedoch unberührt.

- In den beiden in [Balbo *et al.*, 1988] angegebenen Beispielen werden identische GSPN-Submodelle zusammengefaltet. Da keine farbigen Petrinetze verwendet werden, also lediglich eine einzige Klasse von Tokens zur Verfügung steht, führt dies zum Verlust der Identität der Aufträge im System. Der durch das Falten erzielte Gewinn besteht aber in einer Reduktion des Zustandsraums. In dieser Arbeit geschieht das Falten ad hoc, ist also nicht grundsätzlich Bestandteil der Methode und bleibt dem Modellierer überlassen.

  In [Buchholz, 1992a] wird beschrieben, wie eine automatisierbare Symmetrieausnutzung in den Kontext von [Buchholz, 1992b] eingebaut werden kann. Der in Kapitel 4 der vorliegenden Arbeit vorgestellte Ansatz einer strukturierten Modellbeschreibung mit besonderer Berückichtigung von Modellsymmetrien ist mit dieser Symmetrieausnutzung verwandt. Da sich jedoch unsere Modellwelt von der Buchholz'schen unter anderem dadurch unterscheidet, daß sie die Spezifikation von Modellen mit replizierten Submodellen vorsieht, ist es uns möglich, eine effiziente Prozedur für die Berechnung der Matrizen des reduzierten Modells anzugeben (siehe Abschnitt 4.6).
- Im Rahmen der Stochastic Well Formed Nets [Chiola und Franceschinis, 1989; Chiola *et al.*, 1990a; Chiola *et al.*, 1990b] treten Symmetrien auf natürliche Weise in Erscheinung und sind als solche automatisch erkennbar. Die Methode zur Ausnutzung symmetrischer Netzmarkierungen basiert auf der Definition einer Äquivalenzrelation auf der Menge der farbigen Netzmarkierungen. Markierungen die durch zulässige Permutation ineinander überführt werden können, werden als äquivalent betrachtet. Zulässige Permutationen sind dabei solche, die Farben auf Farben derselben Klasse (bzw. Unterklasse) abbilden. Bei geordneten Unterklassen sind nur Rotationen zulässig, da hier die relative Reihenfolge der Objekte beibehalten werden muß.

  Auf diese Weise gelangt man zu dem Konzept der symbolischen Markierung. Eine solche bildet eine Äquivalenzklasse von gewöhnlichen Markierungen und wird durch einen aufgrund der lexikographischen Ordnung eindeutig bestimmten Repräsentanten (die kanonische Form der symbolischen Markierung) repräsentiert. Aufbauend darauf wird eine symbolische Schaltregel (symbolic firing rule) definiert, welche den Übergang zwischen den symbolischen Markierungen beschreibt. Man kann mit Hilfe dieser symbolischen Schaltregel einen symbolischen Erreichbarkeitsgraphen (symbolic reachability graph, SRG) direkt erzeugen und aus diesem eine Markovkette ableiten. Von dieser wird gezeigt, daß sie isomorph zu der Markovkette ist, die durch Zusammenfassen der über den gewöhnlichen Erreichbarkeitsgraphen erzeugbaren Markovkette entstehen würde. Es ist mit Hilfe des symbolischen Erreichbarkeitsgraphen also möglich, ohne den Umweg über den gewöhnlichen Erreichbarkeitsgraphen zu einer Reduktion des Zustandsraums des Modells zu gelangen! Dies stellt ein wichtiges Ergebnis dar, da gerade der gewöhnliche Erreichbarkeitsgraph — obwohl es sich nur um eine Zwischendarstellungsform handelt — aufgrund seiner Größe die praktische Analyse eines Modells unmöglich machen kann. Dieses Vorgehen ist voll automatisierbar, da die syntak-

tische Modellbeschreibung die gesamte Information über Symmetrien enthält. Die Autoren machen jedoch keine Aussage über die Effizienz des Verfahrens.
Für dieselbe Modellbeschreibungsmethode wurde ein weiteres Verfahren zur Reduktion des Zustandsraums entwickelt, die sogenannte Decolourization [Chiola *et al.*, 1991a]. Es ist dann anwendbar, wenn Farben redundant sind, da sie sich gleich verhalten. Die Methode leistet das Erkennen und Eliminieren solcher Farben und ist unabhängig von der Methode des symbolischen Erreichbarkeitsgraphen. Beide Techniken können also kombiniert werden und sich so ergänzen.

- Wie bereits in Abschnitt 3.3.3 erwähnt, ist in der Modellwelt der Stochastic Activity Networks ein spezieller Operator zur Spezifikation replizierter Submodelle vorgesehen. Für diese Methode wurde ein Algorithmus für die direkte Berechnung eines reduzierten Zustandsraums aus der Modellspezifikation entwickelt [Sanders und Meyer, 1991]. Der direkte Weg von der Modellbeschreibung auf höherer Ebene zum reduzierten Zustandsraum hat den wichtigen Vorteil, daß ein unter Umständen sehr großer und daher wegen Speicherbeschränkungen nicht realisierbarer Zustandsraum niemals, auch nicht als Zwischenergebnis, aufgebaut werden muß. Bei der Erzeugung des reduzierten stochastischen Prozesses wird zum einen die Netzstruktur, insbesondere das Vorhandensein replizierter Subsysteme, und zum anderen die Reward-Struktur ausgenutzt. Diese Methode wird in dem Werkzeug UltraSAN benutzt [Couvillion *et al.*, 1991; Sanders *et al.*, 1994].
- Die im Abschnitt 3.4 erläuterte Matrix-Semantik für die Prozeßsprache TIPP^{MS} stellt einen Replikationsoperator zur Verfügung. Die Semantik gewährleistet eine Abbildung von Prozeßtermen auf eine Ratenmatrix, die die vollständig zusammengefaßte zugehörige Markovkette beschreibt.

4 Eine Modellwelt für die strukturierte Modellierung von Systemen mit Symmetrien

In diesem Kapitel wird ein neuer Ansatz vorgestellt, eine Modellwelt, die eine strukturierte Modellbeschreibung und -auswertung erlaubt. Sie zeichnet sich dadurch aus, daß sie nicht nur die *Beschreibung* von Systemen mit replizierten Teilsystemen unterstützt, sondern auch effiziente Verfahren für die *Analyse* solcher Modelle bereitstellt. Insbesondere ist ein Verfahren zur Reduktion des Zustandsraums in Gegenwart replizierter Teilmodelle fester Bestandteil der Modellwelt. Diese Zustandsraumreduktion fügt sich nahtlos in die strukturierte Analyse auf der Basis eines Tensordeskriptors ein und ergänzt diese dadurch in idealer Weise.

Die hier entwickelten Ideen und Resultate haben allgemeine Gültigkeit. Zur Modellbeschreibung auf höherer Ebene beschränkt sich unsere Modellwelt zunächst auf den Formalismus der stochastischen Automaten als einer zustandsorientierten Beschreibungsmethode. Zur Erläuterung der Zustandsraumreduktion wird nämlich auf eine matrixorientierte Darstellung zurückgegriffen, da die Argumentation über Zusammenfaßbarkeit von Zuständen nur auf dieser Ebene zu führen ist. Obwohl es die Anschaulichkeit etwas beeinträchtigt, hat dieses Vorgehen den Vorteil, sehr allgemein zu sein. Bereits in Kapitel 2 wurde die enge Verwandtschaft zwischen stochastischen Automaten und stochastischen Prozeßalgebren festgestellt. Dies bedeutet, daß in die hier vorgestellte Modellwelt durchaus auch mit Hilfe einer SPA beschriebene (Teil-)Modelle hineinpassen. Etwas schwieriger, aber trotzdem denkbar ist die Integration von stochastischen Petrinetzmodellen oder Warteschlangenmodellen. Es ist also ohne weiteres möglich, innerhalb ein und desselben Gesamtmodells unterschiedliche Modellierungsmethoden zur Spezifizierung der einzelnen Submodelle zu verwenden, d.h. die hier vorgestellte Modellwelt unterstützt den Multi-Paradigmen-Ansatz.

In diesem Kapitel wird bewußt eine minimale Modellwelt beschrieben. Dies geschieht aus Gründen der Verständlichkeit und um die Notation übersichtlich zu halten. Mögliche Verallgemeinerungen und Erweiterungen der Modellwelt finden sich in Kapitel 5.

Die Darstellung einer strukturierten Modellwelt ist auf zwei Weisen möglich: Man kann “bottom-up” vorgehen, ausgehend von den einzelnen Modulen und darauf aufbauend zu immer größeren Einheiten gelangend, oder aber “top-down”, beginnend mit der Gesamtsicht und anschließender Verfeinerung. Wir entscheiden uns hier für den letzteren Weg.

4.1 Das Gesamtmodell

Unsere Modellwelt sieht einen modularen, strukturierten Modellaufbau vor. Auf oberster Ebene wird das sogenannte *Gesamtmodell* (overall model) definiert, welches eine abstrahierte Sicht auf das gesamte modellierte System darstellt. Eine solche Sicht erlaubt es dem Benutzer, ein System als Ganzes zu betrachten, ohne durch die Einbeziehung von momentan unnötigen Details die Übersicht zu verlieren.

Das Gesamtmodell besteht aus Modulen, den sogenannten *Submodellen*. Sie können untereinander in mehr oder weniger enger Beziehung stehen. Wir sprechen daher von interagierenden Submodellen. Für die Anzahl der Submodelle, die das Gesamtmodell ausmachen, wird die Variable n verwendet.

Ein wichtiges Merkmal der strukturierten Modellierung ist die Tatsache, daß Submodelle unabhängig voneinander spezifiziert werden. Dies erlaubt es dem Modellierer, sich bei der Modellspezifizierung auf Teile des Gesamtsystems zu konzentrieren.

Das stochastische Verhalten des Gesamtmodells wird durch einen Markovprozeß beschrieben, welcher sich durch die Überlagerung der den Submodellen zugehörigen Markovprozesse unter Berücksichtigung möglicher Abhängigkeiten zwischen diesen Prozessen ergibt. Für die quantitative Analyse eines Modells ist es notwendig, die dem Gesamtmodell zugehörige Generatormatrix zu erzeugen. Als erster Schritt auf diesem Weg wird für jedes Submodell aus seiner Beschreibung auf höherer Ebene eine Menge von Matrizen generiert, durch welche das Verhalten des Submodells eindeutig charakterisiert ist.

Auf die Techniken zur Erzeugung der einem Submodell zugehörigen Matrizen aus seiner Beschreibung auf höherer Ebene wird im Rahmen dieses Kapitels nicht mehr eingegangen. Die Verfahren zur Erzeugung des Zustandsraums und der Zustandsübergänge aus den gängigen Modellbeschreibungsmethoden sind bekannt. Man denke z.B. an die Konstruktion der Markovkette aus einem stochastischen Petrinetz über die Zwischendarstellung des Erreichbarkeitsgraphen, vgl. Abschnitt 2.2.3, oder an die Erzeugung der Markovkette aus einer prozeßalgebraischen Modellbeschreibung über ein beschriftetes Transitionssystem, vgl. Abschnitt 2.2.2.

Mit dem hier vorgestellten Ansatz soll eine möglichst allgemeine Modellwelt erfaßt werden. Daher wird für alle Submodelle, ganz gleich mit welcher Modellierungsmethode sie formuliert sind, eine zustandsorientierte Sicht favorisiert. Im Gegensatz dazu wurde in anderen Arbeiten (z.B. in [Buchholz, 1992b]) eine Auftrags- bzw. Token-orientierte Sicht verwendet, die eine teilweise knappere Form der Darstellung zuläßt, gleichzeitig aber den Nachteil hat, daß die Interaktion zwischen Submodellen auf einige wenige Primitive eingeschränkt ist. Diese Primitive beschreiben den Übergang eines Teils (Auftrag oder Token, im Englischen allgemein als “entity” bezeichnet) von einem Submodell in ein anderes. Durch die zustandsorientierte Sicht sind in unserer Modellwelt beliebige Synchronisationen zwischen zwei oder mehr Submodellen möglich. Diesen Vorteil der Allgemeinheit der zustandsorientierten Sicht erkauft man sich allerdings mit dem Nachteil einer geringeren Anschaulichkeit.

4.2 Submodelltypen

In Abschnitt 3.6 wurde bereits auf die zunehmend wichtige Rolle von Systemen hingewiesen, die aus sich gleich oder ähnlich verhaltenden replizierten Komponenten bestehen. Der erste Schritt auf dem Weg zu einer effizienten Modellierung solcher Systeme ist die Identifizierung identischer (replizierter) Komponenten des realen Systems, die sich im Modell auf natürliche Weise als eine Reihe identischer Submodelle widerspiegeln werden.

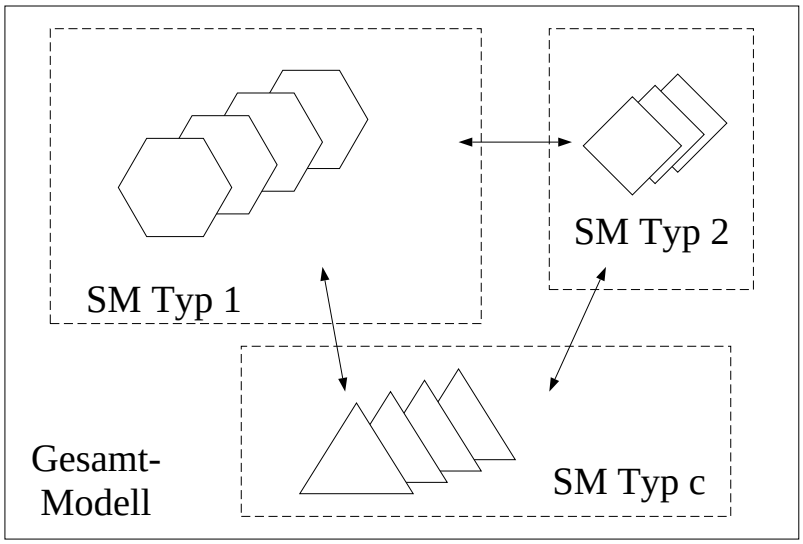


Abbildung 4.1: Allgemeiner Aufbau des Gesamtmodells aus zu Gruppen desselben Typs zusammengefaßten Submodellen

Diesem Phänomen wird in der hier diskutierten Modellwelt durch Vorkehrungen für die Spezifizierung replizierter Komponenten Rechnung getragen, mit dem Ziel, im nachfolgenden Analyseschritt einen automatisierbaren Weg zur Zustandsraumreduktion zu ermöglichen. Dazu werden bei der Spezifikation des Gesamtmodells gleichartige Submodelle zu Gruppen zusammengefaßt. Gleichartige Submodelle werden Submodelle desselben *Submodelltyps* genannt und oft auch als *symmetrisch* bezeichnet.

Der allgemeine Aufbau eines jeden Gesamtmodells ist in Abb. 4.1 schematisch dargestellt. Man erkennt, daß sich das Gesamtmodell aus vielen Submodellen zusammensetzt, von denen jeweils einige denselben Typ haben. Graphisch wird der Typ eines Submodells durch seine geometrische Form angegeben. Alle Submodelle eines Typs liegen innerhalb eines gestrichelten Kastens, d.h. in derselben Gruppe.

Wir wollen hier den Begriff des Submodelltyps sehr eng fassen: Zwei Submodelle sind genau dann vom selben Typ, wenn das eine ein genaues Replikat des andern darstellt, beide also identisch sind. Andere, allgemeinere Möglichkeiten, d.h. eine Lockerung der Forderung nach exakter Gleichheit, werden in Kapitel 5 diskutiert.

Die Anzahl unterschiedlicher Submodelltypen in einem Gesamtmodell wird mit der Variable c benannt. Die Anzahl der Submodelle vom Typ i wird mit n_i bezeichnet, $i = 1 \ldots c$. Daraus ergibt sich für die gesamte Anzahl n von Submodellen die Beziehung

$$n = \sum_{i=1}^{c} n_i$$

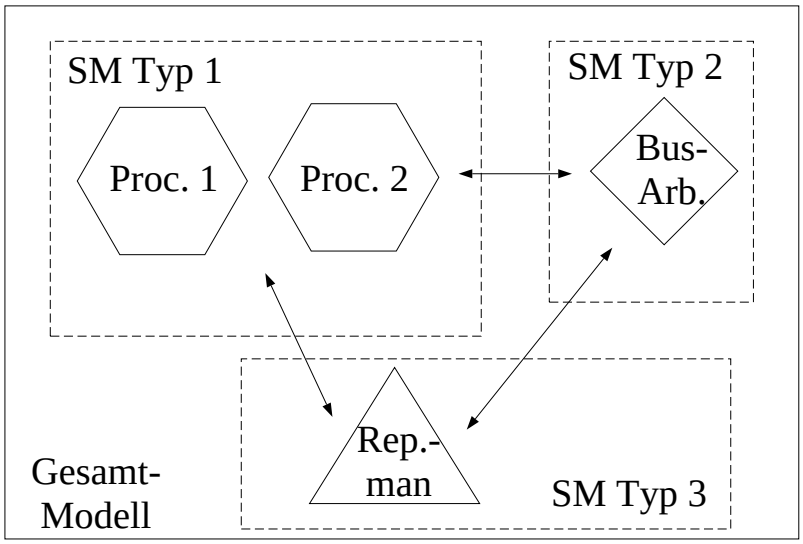
Unter der Annahme, daß die Submodelle vom Typ i jeweils s_i mögliche Zustände annehmen können, und aus der Tatsache, daß der Zustandsraum des Gesamtmodells gleich dem cartesischen Produkt der Zustandsräume der Submodelle ist, folgt, daß die Anzahl der Zustände des Gesamtmodells durch

$$s = \underbrace{s_1 \ldots s_1}_{n_1} \underbrace{s_2 \ldots s_2}_{n_2} \ldots \underbrace{s_c \ldots s_c}_{n_c} = \prod_{i=1}^{c} s_i^{n_i}$$

gegeben ist. Offensichtlich können auf diese Weise bereits bei der Kombination einiger weniger Submodelle moderater Größe enorme Zustandsräume erzeugt werden!

Es ist an dieser Stelle wichtig, darauf hinzuweisen, daß aufgrund der Bedingungen, die sich aus der Synchronisierung von Submodellen ergeben, nicht alle s Zustände tatsächlich erreichbar sein müssen. Darin steckt ein potentielles Problem, da unter Umständen für unerreichbare Zustände Systemressourcen (Speicherplatz und/oder Rechenzeit) verschwendet werden, wodurch sich das Problem der Zustandsraumexplosion weiter verschlimmert.

Beispiel:
Als Beispiel verwenden wir in diesem Kapitel das bereits aus Kapitel 2 bekannte Modell eines Multiprozessors mit Breakdown und Repair.



Die Abbildung zeigt, daß das Gesamtmodell Submodelle dreier unterschiedlicher Typen beinhaltet, nämlich *Processor*, *Bus-Arbitrator* und *Repairman*. Wir betrachten einen Fall mit $(N =)n_1 = 2$ Prozessoren.

4.3 Beschreibung der Submodelle

Für die Beschreibung des Verhaltens der Submodelle ist der Begriff des Ereignisses von zentraler Bedeutung. Ein *Ereignis* (event) ist der zeitlose Zustandsübergang eines Submodells oder der gleichzeitige zeitlose (synchronisierte) Übergang von mehreren Submodellen. Es ist in diesem Zusammenhang wichtig, den Ereignisbegriff nicht mit dem Begriff der Aktivität zu verwechseln. Eine *Aktivität* wird durch zwei Ereignisse begrenzt und ist mit einer Zeitdauer versehen, während Ereignisse als instantane Zustandsübergänge zeitlos sind.

Es gibt verschiedene Arten von Ereignissen. Im einfachsten Fall unterscheidet man zwischen *internen* und *synchronisierenden* Ereignissen. Interne Ereignisse betreffen nur ein einziges Submodell, während ein synchronisierendes Ereignis den gleichzeiti-

gen Zustandswechsel von mindestens zwei Submodellen bewirkt. Synchronisierende Ereignisse sind demzufolge dafür geeignet, Abhängigkeiten zwischen Submodellen zu spezifizieren.

Das Eintreten eines Ereignisses setzt voraus, daß bestimmte *Vorbedingungen* für dieses Ereignis erfüllt sind. Die Vorbedingung für ein lokales Ereignis ist dann erfüllt, wenn sich das betreffende Submodell in einem bestimmten Zustand befindet. Vorbedingung für ein synchronisierendes Ereignis ist i.a. eine bestimmte Zustandskombination der an diesem Ereignis beteiligten Submodelle.

Die *Verzögerungszeit* eines Ereignisses ist die Zeit zwischen der Erfüllung der Vorbedingungen für dieses Ereignis und dem Eintreten des Ereignisses. Obwohl das Ereignis selbst zeitlos ist, kann man ihm also eine Verzögerungszeit größer Null zuordnen. Die Verzögerungszeit eines Ereignisses im Modell repräsentiert beispielsweise eine Bearbeitungszeit oder eine Zwischenankunftszeit im realen System. Da der Moment, in dem die Vorbedingungen eines Ereignisses erfüllt sind, stets durch ein anderes Ereignis markiert wird, entspricht die Verzögerungszeit eines Ereignisses immer der Dauer einer durch zwei Ereignisse begrenzten Aktivität.

Im Kontext zeitkontinuierlicher Markovmodelle wird mit Raten operiert, welche die Parameter negativ exponentieller Verteilungen darstellen. Jedes Ereignis ist deshalb mit einer Rate assoziiert, welche die Verteilung der Verzögerungszeit eindeutig bestimmt. Raten von Ereignissen können aus unterschiedlichen Bereichen gewählt werden. Im Rahmen der grundsätzlichen Beschreibung der Modellwelt wollen wir uns zunächst auf den einfachsten Fall konstanter endlicher Raten beschränken, d.h. einer Rate $r \in I\!R^+$, die ein Ereignis mit exponentiell verteilter Verzögerungszeit kennzeichnet. Mögliche Erweiterungen, z.B. Ereignisse mit einer verschwindenden Verzögerungszeit, sind in Abschnitt 5.1.1 behandelt.

Bei gleichzeitigem Erfülltsein der Vorbedingungen für verschiedene (interne oder synchronisierende) Ereignisse tritt ein Wettbewerb (race policy) ein. Man kann sich diesen als ein Ablaufen von Stoppuhren (timer) vorstellen: Jedem Ereignis, dessen Vorbedingung erfüllt ist, ist eine Stoppuhr zugeordnet, die eine Stichprobe der stochastisch verteilten Verzögerungszeit für dieses Ereignis repräsentiert. Alle Stoppuhren werden gleichzeitig gestartet. Die Stoppuhr mit der kürzesten Laufzeit bestimmt, welches Ereignis als nächstes eintritt. Wegen der Gedächtnislosigkeit der Exponentialverteilung können beim Eintreten des Ereignisses alle Stoppuhren zurückgesetzt werden um daraufhin gemeinsam mit möglicherweise neu hinzukommenden Stoppuhren neu gestartet zu werden.

4.3.1 Interne Ereignisse

Der unabhängige Zustandsübergang in einem einzigen Submodell wird als internes (lokales) Ereignis bezeichnet. Andere Submodelle sind von einem internen Ereignis nur insofern berührt, als sich durch den neuen Zustand nach Eintreten des internen Ereignisses die Vorbedingungen für synchronisierende Ereignisse ändern können.

Ein Submodell vom Typ i kann s_i verschiedene Zustände annehmen. Daher wird das interne Verhalten von Submodellen des Typs i, d.h. die durch interne Ereignisse möglichen Zustandsübergänge, durch eine Matrix $Q_l^{(i)}$ der Dimension s_i beschrieben.

Diese Matrix ist dieselbe für alle Submodelle des Typs i, sie enthält die den internen Ereignissen von Submodellen dieses Typs zugehörigen Raten. Können in Submodellen vom Typ i keine internen Ereignisse stattfinden, so gilt $Q_l^{(i)} = 0$.

Beispiel:
Für die Zustände und Zustandsübergänge orientieren wir uns an der Darstellung des Multiprozessorbeispiels als SAN-Modell gemäß Abb. 2.5, S. 15. Die Matrizen $Q_l^{(i)}$ haben daher folgende Darstellung:

$Q_l^{(\text{Processor})} = Q_l^{(1)} =$

	0	1	2	3	4
work 0		λ		λ	
w. for bus 1					
access 2					
down 3					
u. repair 4					

$Q_l^{(\text{Bus Arb.})} = Q_l^{(2)} = 0$

$Q_l^{(\text{Repairman})} = Q_l^{(3)} =$

	0	1	2	3	4	5
idle 0						
w. for bus 1 1						
downgrade 2						
repair 3					λ	
w. for bus 2 4						
upgrade 5						

Die Matrizen für die internen Ereignisse sind für dieses Beispiel extrem dünn besetzt, da fast alle Zustandsübergänge mit synchronisierenden Ereignissen assoziiert sind. Die sich daraus ergebende Problematik werden wir weiter unten in der Fortsetzung des Beispiels diskutieren.

4.3.2 Synchronisierende Ereignisse

Das Eintreten eines synchronisierenden (globalen) Ereignisses bewirkt den gleichzeitigen Zustandsübergang von zwei oder mehr Submodellen. Die Variable E bezeichnet die Menge der synchronisierenden Ereignisse im Gesamtmodell. Für ein synchronisierendes Ereignis $e \in E$ wird mit λ_e die ihm zugehörige Rate angegeben.

Die Matrix $Q_e^{(i)}$ beschreibt die möglichen synchronisierten Zustandsübergänge für Submodelle des Typs i. Sie gibt an, welche Vorbedingungen bezüglich Submodellen vom Typ i für das Eintreten des synchronisierenden Ereignisses e erfüllt sein müssen, und

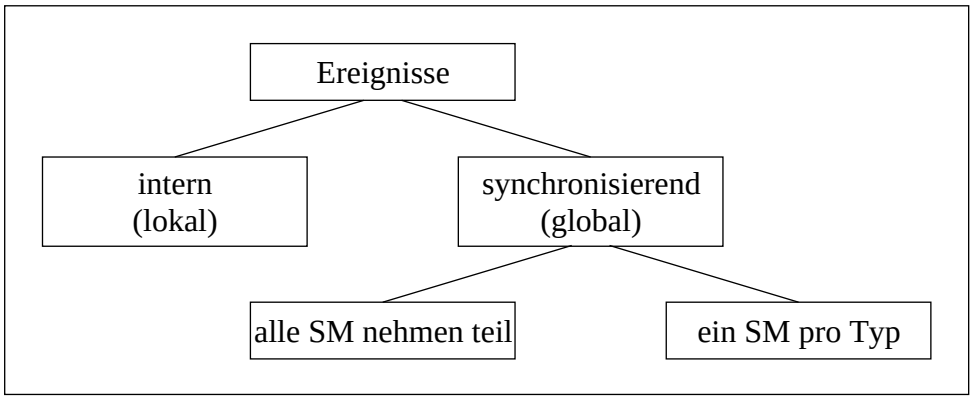


Abbildung 4.2: Grundlegende Ereignistypen

wie diese Submodelle von dem synchronisierenden Ereignis e beeinflußt werden. Wie die Matrix $Q_l^{(i)}$, so hat auch $Q_e^{(i)}$ die Dimension s_i. Nehmen die Submodelle eines bestimmten Typs nicht an dem synchronisierenden Ereignis teil, so ist die entsprechende Matrix eine Identitätsmatrix, d.h. $Q_e^{(i)} = I$.

Im einfachsten Fall sind für ein synchronisierendes Ereignis bezüglich der Typzugehörigkeit von Submodellen zwei mögliche Semantiken vorgesehen: Entweder nehmen *alle* Submodelle eines Typs an dem synchronisierenden Ereignis teil, oder aber es nimmt nur *ein einziges*, beliebiges (daher nicht von vornherein eindeutig bestimmbares) Submodell des Typs an dem Ereignis teil. Im ersten Fall verlangt die Vorbedingung, daß alle Submodelle eines Typs zur Teilnahme an dem synchronisierenden Ereignis bereit sind, im zweiten Fall muß lediglich eines bereit sein. Mit diesen beiden Möglichkeiten ist man bereits in der Lage, die wichtigsten Synchronisationsmuster wie z.B. Barrier Synchronisation oder das Rendezvous zwischen einem Submodell und genau einem Submodell eines anderen Typs auf elegante Weise zu modellieren.

Die bisher beschriebenen grundlegenden Ereignistypen sind in Abb. 4.2 dargestellt. Aufbauend darauf sind verschiedene Erweiterungen denkbar, siehe Abschnitt 5.1.3.

Es stellt sich an dieser Stelle die Frage nach der Lokalität der Rate. Dabei können zwei unterschiedliche Standpunkte vertreten werden: Zum einen ist es möglich, immer eines der an einem synchronisierenden Ereignis teilnehmenden Submodelle als Aktor auszuzeichnen und alle anderen teilnehmenden Submodelle als passiv zu betrachten. In diesem Fall müssen die Einträge in der Matrix des Aktors die Rate λ_e des synchronisierenden Ereignisses enthalten, während die in den Matrizen der passiven Teilnehmer möglichen Zustandsübergänge jeweils durch eine 1 eingetragen werden. Der Wert 1 wird verwendet, da die Raten später multipliziert werden. Diese Sichtweise führt im Zusammenhang mit typisierten Submodellen jedoch zu Schwierigkeiten. Sollen sich z.B. alle n_i Submodelle eines Typs durch das Ereignis e synchronisieren, so müßte eines davon als Aktor ausgezeichnet werden, was der perfekten Symmetrievorstellung widerspricht. Ein anderer möglicher Fall, der Schwierigkeiten macht, ist die Synchronisation von genau einem Submodell vom Typ i mit genau einem Submodell vom Typ j, wobei beide Teilnehmer nicht eindeutig vorherbestimmt sind.

Deshalb wird von uns ein zweiter Standpunkt bevorzugt: Wir betrachten alle an einem synchronisierenden Ereignis teilnehmenden Submodelle als gleichberechtigt, d.h. eine

Unterscheidung zwischen Aktor und passiven Teilnehmern entfällt. Alle Matrizen vom Typ $Q_e^{(i)}$ enthalten demzufolge lediglich 1-Einträge, und die Rate λ_e des synchronisierenden Ereignisses e wird in einer separaten Tabelle festgehalten. Für solche synchronisierenden Ereignisse, bei denen nicht von vornherein feststeht, welche Submodelle tatsächlich teilnehmen werden, hat dieses Verfahren den Vorteil, auf die schwierige a priori Festlegung eines Aktors zu verzichten. Auf diese Weise sind also auch flexible Synchronisationsmuster modellierbar.

Beispiel:
Die folgende Tabelle gibt einen Überblick über die synchronisierenden Ereignisse im Multiprozessorbeispiel:

Ereignisname	Beteiligte Submodelle
grant bus *l*	(Proc. 1 $\oplus$ Proc. 2) $\otimes$ Bus Arbitrator
release bus *l*	(Proc. 1 $\oplus$ Proc. 2) $\otimes$ Bus Arbitrator
start repair	(Proc. 1 $\oplus$ Proc. 2) $\otimes$ Repairman
grant bus *h*	Repairman $\otimes$ Bus Arbitrator
preempt	(Proc. 1 $\oplus$ Proc. 2) $\otimes$ Repairman
release bus *h*	Repairman $\otimes$ Bus Arbitrator
up	(Proc. 1 $\oplus$ Proc. 2) $\otimes$ Bus Arbitrator $\otimes$ Repairman

Um die Art der Interaktion zwischen den teilnehmenden Submodellen anzudeuten, werden in der rechten Spalte der Tabelle die Symbole $\otimes$ und $\oplus$ verwendet, je nachdem, ob alle beide oder nur eines der *Processor*-Submodelle an der Synchronisation mit den Submodellen der anderen Typen teilnimmt. Diese Symbole spiegeln die weiter unten verwendete Tensorbeschreibung wider, siehe Abschnitt 4.4. An den beiden Ereignissen *grant bus h* und *release bus h* sind die *Processor*-Submodelle gar nicht beteiligt.

Als Beispiel für Matrizen des Typs $Q_e^{(i)}$ sind nachfolgend die Matrizen angegeben, die sich auf das synchronisierende Ereignis *grant bus l* beziehen. Da der Repairman nicht an diesem Ereignis teilnimmt, ist die zugehörige Matrix eine Identitätsmatrix.

$$Q^{(\text{Processor})}_{\text{grant bus } l} = Q^{(1)}_{\text{grant bus } l} =$$

	0	1	2	3	4
work 0					
w. for bus 1			1		
access 2					
down 3					
u. repair 4					

$$Q^{(\text{Bus Arb.})}_{\text{grant bus } l} = Q^{(2)}_{\text{grant bus } l} =$$

	0	1
bus idle 0		1
bus used 1		

$$Q^{(\text{Repairman})}_{\text{grant bus } l} = Q^{(3)}_{\text{grant bus } l} = I_6$$

Da es in diesem Beispiel drei Submodelltypen und sieben synchronisierende Ereignisse gibt, werden insgesamt $3 \times 7 = 21$ Matrizen des Typs $Q_e^{(i)}$ erzeugt. Man beobachtet, daß zwischen den einzelnen Submodellen eine sehr starke Abhängigkeit herrscht. Diese drückt sich —wie bereits weiter oben festgestellt — zum einen dadurch aus, daß die Matrizen für die internen Ereignisse sehr dünn besetzt sind. Zum andern zeigt sich das in der Tatsache, daß von den insgesamt $5 \times 5 \times 2 \times 6 = 300$ Zuständen des Produktzustandsraums aufgrund der durch die Synchronisationen gegebenen Einschränkungen nur 51 tatsächlich erreichbar sind, vgl. Tab. 2.2, S. 37.

Daraus muß man schließen, daß die Methode für das vorliegende Multiprozessorbeispiel wenig geeignet ist, da zu viele unerreichbare Zustände vorliegen, im Verhältnis zur Zustandszahl der einzelnen Submodelltypen also zuviel Interaktion zwischen Submodellen existiert.

Allgemein ist festzustellen, daß die Anwendung der hier diskutierten Methode der modularen Modellbeschreibung und des im folgenden beschriebenen Tensordeskriptors nur dann vorteilhaft ist, wenn Submodelle ein im wesentlichen in sich abgeschlossenes Verhalten aufweisen und nur relativ schwach mit ihrer Umwelt interagieren, also bei größeren Beispielen mit vielen internen Ereignissen. Es fällt daher schwer, ein nichttriviales und trotzdem noch auf dem Papier darstellbares Beispiel zu finden, welches einerseits verschiedene Submodelltypen, andererseits einige synchronisierende Ereignisse beinhaltet.

4.4 Die Generatormatrix des Gesamtmodells

4.4.1 Tensordeskriptoren

In diesem Abschnitt wird das Prinzip der Generierung der Generatormatrix eines strukturierten Markovmodells aus Matrizen, die das Verhalten von Submodellen beschreiben, erläutert. Dazu werden die Operatoren der Tensoralgebra benötigt, deren Grundlagen in Anhang B.1 dargestellt sind. Ausgangspunkt für dieses Vorgehen sind die folgenden grundlegenden Fakten:

Angenommen, es liegen zwei unabhängige *zeitdiskrete* Markovprozesse X_1 und X_2 vor, die durch die stochastischen Übergangsmatrizen P_1 und P_2 spezifiziert sind. Wir wollen den kombinierten stochastischen Prozeß X betrachten, dessen Zustandsraum als das cartesische Produkt der beiden Zustandsräume von P_1 und P_2 gegeben ist. Dann besitzt der kombinierte stochastische Prozeß X die stochastische Übergangsmatrix P, die sich als das Tensorprodukt von P_1 und P_2 schreiben läßt:

$$P = P_1 \otimes P_2$$

Für den Fall zweier unabhängiger *zeitkontinuierlicher* Markovprozesse mit den Generatormatrizen Q_1 und Q_2 existiert ein analoger Zusammenhang. Hier ist die Generatormatrix des kombinierten stochastischen Prozesses Q gegeben als

$$Q = Q_1 \oplus Q_2$$

Rein intuitiv kann man sich vorstellen, daß das Tensorprodukt ein Operator ist, der den *gleichzeitigen* Zustandswechsel zweier (oder mehrerer) unabhängiger Markovprozesse ausdrückt. Im Fall zweier unabhängiger zeitdiskreter Markovprozesse geschieht zu jedem Zeitschritt in beiden Prozessen ein Zustandsübergang (der allerdings mit Wahrscheinlichkeit p_{ii} zum vorigen Zustand zurück führen kann). Daher wird hier der Operator Tensorprodukt verwendet. Umgekehrt drückt der Operator Tensorsumme den (asynchronen) Zustandswechsel nur eines der beiden beteiligten Markovprozesse aus. Aufgrund dieses Sachverhaltes werden im folgenden die beiden Tensoroperatoren benutzt, um die Generatormatrix des Gesamtmodells als Tensorausdruck unter Verwendung der Submodellmatrizen darzustellen.

4.4.2 Der Tensordeskriptor für das Gesamtmodell

In diesem Abschnitt wird ein sehr allgemeiner Ausdruck für den Tensordeskriptor Q eines strukturierten Markovmodells in unserer Modellwelt angegeben und diskutiert.

Aus den im vorigen Abschnitt angestellten Überlegungen folgt, daß sich die Generatormatrix des Gesamtmodells auf die in Gl. (4.1) dargestellte Weise konstruieren läßt.

$$Q = \underbrace{\bigoplus_{i=1}^{c}\left(\bigoplus_{j=1}^{n_i} Q_l^{(i)}\right)}_{\text{Teil 1}} + \sum_{e\in E}\lambda_e\left(\underbrace{\bigotimes_{i=1}^{c}\left(\bigodot_{j=1}^{n_i} Q_e^{(i)}\right)}_{\text{Teil 2}} - \underbrace{\bigotimes_{i=1}^{c}\left(\bigodot_{j=1}^{n_i} Q_{e,n}^{(i)}\right)}_{\text{Teil 3}}\right) \tag{4.1}$$

An der Form von Q erkennt man, daß die Generatormatrix des Gesamtmodells aus drei Teilen besteht:

Der erste Teil bezieht sich auf die internen Ereignisse in den n Submodellen. Er ist als Tensorsumme aus Matrizen vom Typ $Q_l^{(i)}$ aufgebaut, da die lokalen Ereignisse einzelne unabhängige Zustandsübergänge auf dem Produktzustandsraum bewirken. Die Tensorsumme ist nach Submodelltypen strukturiert. Es handelt sich also um eine zweistufige Tensorsumme, wobei im Inneren die Tensorsumme über die identischen Matrizen der zum Submodelltyp i gehörigen Submodelle (davon gibt es n_i Stück), und im Äußeren die Tensorsumme über alle Submodelltypen gebildet wird (es gibt c unterschiedliche Submodelltypen). Man beachte, daß die Operanden der inneren Tensorsumme unabhängig vom Index j sind, d.h. die innere Tensorsumme stellt folgenden Ausdruck dar:

$$\bigoplus_{j=1}^{n_i} Q_l^{(i)} = \underbrace{Q_l^{(i)} \oplus Q_l^{(i)} \oplus \ldots \oplus Q_l^{(i)}}_{n_i \text{ mal}}$$

Der zweite Teil bezieht sich auf die synchronisierenden Ereignisse $e \in E$. Auch hier ist wie beim ersten Teil die Zweistufigkeit der Schreibweise bemerkbar. Auf der äußeren Ebene wird das Tensorprodukt gebildet, da synchronisierende Ereignisse einen gleichzeitigen Zustandsübergang aller teilnehmenden Submodelltypen bewirken. Wir verwenden in Gl. (4.1) auf der inneren Ebene für die Verknüpfung der Matrizen $Q_e^{(i)}$ die Notation $\odot$, um synchronisierende Ereignisse unabhängig von ihrer Semantik mit einer einheitlichen Schreibweise zu behandeln. Das Symbol $\odot$ ist entweder als $\otimes$ oder

als $\oplus$ zu interpretieren, in Abhängigkeit von der Semantik des Ereignisses e: Falls alle Submodelle des Typs i am synchronisierenden Ereignis e teilnehmen, ist $\odot$ durch $\otimes$ zu ersetzen (gemeinsamer Zustandsübergang). Wenn nur ein Submodell vom Typ i am Ereignis e teilnimmt, wird $\odot$ durch den Operator $\oplus$ ersetzt.

Da wir — wie oben dargelegt — von einer Synchronisation unter gleichberechtigten Partnern ausgehen, wird die dem Ereignis e zugehörige Rate λ_e dem Tensorausdruck für die synchronen Übergänge vorangestellt.

Dem dritten Teil, der aus den Matrizen vom Typ $Q_{e,n}^{(i)}$ aufgebaut ist, kommt keine eigenständige Bedeutung zu. Dieser Teil wird vielmehr benötigt, um die Diagonalelemente der Generatormatrix des Gesamtmodells bezüglich der synchronisierenden Ereignisse so anzupassen, daß alle Zeilensummen der entstehenden Matrix Q gleich Null sind. Matrizen des Typs $Q_{e,n}^{(i)}$ enthalten nur auf der Hauptdiagonalen von Null verschiedene Einträge. Das k-te Diagonalelement einer Matrix $Q_{e,n}^{(i)}$ ist gleich der k-ten Zeilensumme der entsprechenden Matrix $Q_e^{(i)}$.

Falls die Submodelle des Typs i von einem bestimmten synchronisierenden Ereignis e nicht betroffen sind, so handelt es sich bei den betreffenden Matrizen $Q_e^{(i)}$ und $Q_{e,n}^{(i)}$ um Identitätsmatrizen.

Beispiel:
Die Darstellung der Generatormatrix für das Multiprozessorbeispiel in Form eines Tensordeskriptors Q erfolgt wegen des Umfangs des entsprechenden Ausdrucks nur unvollständig. Es sind nur die sich auf die internen Ereignisse und auf das synchronisierende Ereignis *grant bus l* beziehenden Teile angegeben. Die Terme für die anderen sechs synchronisierenden Ereignisse sind entsprechend aufgebaut.

$$Q = \underbrace{\bigoplus_{i=1}^{3}\left(\bigoplus_{j=1}^{n_i} Q_l^{(i)}\right)}_{\text{Teil 1}} + \sum_{e\in E}\lambda_e\left(\underbrace{\bigotimes_{i=1}^{3}\left(\bigodot_{j=1}^{n_i} Q_e^{(i)}\right)}_{\text{Teil 2}} - \underbrace{\bigotimes_{i=1}^{3}\left(\bigodot_{j=1}^{n_i} Q_{e,n}^{(i)}\right)}_{\text{Teil 3}}\right)$$

$$\begin{aligned}
&= \left(Q_l^{(Proc.)} \oplus Q_l^{(Proc.)}\right) \oplus Q_l^{(Bus\,Arb.)} \oplus Q_l^{(Repairman)} && \text{Teil 1}\\
&\quad + \lambda_{grant\,bus\,l}\left(\left(Q_{grant\,bus\,l}^{(Proc.)} \oplus Q_{grant\,bus\,l}^{(Proc.)}\right) \otimes Q_{grant\,bus\,l}^{(Bus\,Arb.)} \otimes I_6\right) && \text{Teil 2}\\
&\quad + \ldots\\
&\quad - \lambda_{grant\,bus\,l}\left(\left(Q_{grant\,bus\,l,\,n}^{(Proc.)} \oplus Q_{grant\,bus\,l,\,n}^{(Proc.)}\right) \otimes Q_{grant\,bus\,l,\,n}^{(Bus\,Arb.)} \otimes I_6\right) && \text{Teil 3}\\
&\quad - \ldots
\end{aligned}$$

Wir wollen nun noch eine duale Sicht auf die internen Ereignisse erwähnen: Betrachtet man interne Ereignisse als eine spezielle Art synchronisierender Ereignisse, an denen genau ein einziges Submodell teilnimmt, so kann man alle Ereignisse als synchronisie-

rend auffassen. Man gelangt so zu folgender vereinheitlichter Darstellung:

$$Q = \sum_{e \in E'} \lambda_e \left(\bigotimes_{i=1}^{c} \left(\bigodot_{j=1}^{n_i} Q_e^{(i)} \right) - \bigotimes_{i=1}^{c} \left(\bigodot_{j=1}^{n_i} Q_{e,n}^{(i)} \right) \right) \tag{4.2}$$

Die Menge E' entsteht aus E durch Hinzunahme der internen Ereignisse. Diese Schreibweise ist zwar auf den ersten Blick etwas prägnanter, für die Praxis ist jedoch die in Gl. (4.1) angegebene Form zu bevorzugen, da dort alle internen Ereignisse eines Submodells in *einer* Matrix $Q_l^{(i)}$ zusammengefaßt sind, während bei der zweiten Schreibweise für *jedes* interne Ereignis e' zwei Matrizen $Q_{e'}^{(i)}$ und $Q_{e',n}^{(i)}$ benötigt werden.

Wir haben eine Modellwelt für die strukturierte Modellierung vorgestellt, die voll von der Speicherplatzersparnis des Tensordeskriptor-Ansatzes (vgl Abschnitt 3.5.2) profitiert. Die numerische Analyse des Gesamtmodells kann direkt auf der Repräsentation der Generatormatrix des Gesamtmodells Q als Tensordeskriptor durchgeführt werden. Daher kann auf die Generierung des Gesamtzustandsraums und das explizite Aufstellen von Q verzichtet werden. Es kommt also nicht zu dem enormen Speicherplatzaufwand für die Generatormatrix des Gesamtmodells.

Im Vergleich zu früheren Ansätzen weist die hier beschriebene Modellwelt bereits an dieser Stelle einige bemerkenswerte Erweiterungen auf. An erster Stelle steht die Einführung von Submodelltypen, wodurch verschiedene Synchronisationsmuster unterstützt werden, und die in der Folge eine signifikante Reduktion des Zustandsraums ermöglichen werden. Es ist aber hier auch darauf hinzuweisen, daß die bisher vorgestellte Modellwelt lediglich eine erste, minimale Konfiguration darstellt, die auf verschiedene Weise erweitert werden kann.

4.5 Generierung eines reduzierten Gesamtmodells

Strukturierte Beschreibung und Analyse auf der Basis von Tensordeskriptoren tragen in hohem Maße dazu bei, umfangreiche Modelle handhabbar zu machen. Die Analyse großer Zustandsräume bleibt aber trotz Anwendung dieser Technik weiterhin sehr aufwendig in bezug auf die für die Lösung des linearen Gleichungssystems benötigte Rechenzeit. Daher muß nach wie vor das Ziel verfolgt werden, den Zustandsraum zu reduzieren.

Es wurde in dieser Arbeit bereits mehrfach darauf hingewiesen, daß das Vorhandensein replizierter Teilmodelle innerhalb eines Gesamtmodells die Anwendung von Reduktionsmechanismen zur Verringerung der Anzahl der Zustände gestattet. In diesem Abschnitt wird gezeigt, daß sich in der oben definierten Modellwelt das Prinzip der Zusammenfaßbarkeit nicht nur auf das Gesamtmodell, sondern bereits auf der Ebene der Submodelltypen anwenden läßt. Wir werden nachweisen, daß sich dadurch die größtmögliche Reduktion des Gesamtzustandsraums erreichen läßt. Diese Methode hat auch den Vorteil, daß sie sich nahtlos in die speicherplatzsparende Technik der Tensordeskriptoren einfügt.

Innerhalb eines Submodelltyps lassen wir per Definition nur solche Submodelle zu, die exakte Replikate voneinander sind. Daher wird im folgenden vom Konzept der *strikten*

Zusammenfaßbarkeit (siehe Anhang A.2) Gebrauch gemacht. Wie man durch Lockerung der Symmetriebedingungen auch schwächere Formen der Zusammenfaßbarkeit ausnutzen kann, wird in Abschnitt 5.1.4 angesprochen.

4.5.1 Symmetrische Zustände auf Ebene der Submodelltypen

Die Vorstellung von Submodelltypen erlaubt es, Symmetrieeigenschaften auf automatisierbare Weise bereits zu einem frühen Zeitpunkt, zu dem die Teile des Gesamtmodells unabhängig voneinander (und daher auch parallel zueinander) behandelt werden können, auszunutzen.

Dem Ansatz liegt folgende Grundidee zugrunde: Da das Gesamtmodell eine bestimmte Anzahl replizierter Submodelle desselben Typs enthält, gibt es Zustände, die *symmetrisch* zueinander sind. Es stellt sich heraus, daß solche Zustände dieselbe Gleichgewichtswahrscheinlichkeit haben, und daß es daher nicht notwendig ist, sie individuell zu behandeln. Um den Zustandsraum zu reduzieren, werden alle Submodelle eines Typs zusammengefaßt und durch ein einziges reduziertes Modell ersetzt. Während der Konstruktion des reduzierten Modells wird für jede Menge symmetrischer Zustände ein Zustand als Repräsentant ausgewählt.

Wir bezeichnen mit Z den Zustandsraum des Gesamtmodells. Ein Zustand $z \in Z$ wird durch das folgende Tupel charakterisiert:

$$z = (z_{11}, z_{12}, \ldots, z_{1n_1}, z_{21}, z_{22}, \ldots, z_{2n_2}, \ldots, z_{c1}, z_{c2}, \ldots, z_{cn_c})$$

In diesem Tupel bezeichnet z_{ik} den Zustand des k-ten Submodells vom Typ i. Zwei Zustände x und y gelten genau dann als symmetrisch, wenn y aus x durch Anwendung einer bestimmten Art von Permutation hervorgeht, d.h. falls eine Permutationsmatrix P der Dimension $\sum_{i=1}^{c} n_i$ existiert, so daß gilt $y = xP$, unter der zusätzlichen Bedingung, daß P eine Blockdiagonalstruktur aufweist, wie sie in Gl. (4.3) dargestellt ist.

$$y = xP = x \begin{bmatrix} P_1 & & & 0 \\ & P_2 & & \\ & & \ddots & \\ 0 & & & P_c \end{bmatrix} \tag{4.3}$$

Dabei sind die Blöcke P_i wiederum Permutationsmatrizen der Dimension n_i. Umgangssprachlich ausgedrückt bedeutet das, daß die Permutation mit der Matrix P eine Position innerhalb des Zustandsvektors auf eine andere Position, die innerhalb desselben Submodelltyps liegt, abbildet.

Beispiel:
Im Multiprozessorbeispiel sind die beiden folgenden Zustände symmetrisch:

(work , wait for bus , bus idle , idle) = (0,1,0,0)
(wait for bus , work , bus idle , idle) = (1,0,0,0)

Die ersten beiden Komponenten geben den Zustand der Submodelle vom Typ *Processor* an, können also permutiert werden. Neben diesem Paar gibt es für dieses Beispiel unter den 51 erreichbaren Zuständen weitere 23 Paare symmetrischer Zustände.

Der Zustandsraum Z des Gesamtmodells wird nun so in disjunkte Untermengen ω_i partitioniert, daß alle zueinander symmetrischen Zustände in derselben Untermenge liegen. Eine solche Partitionierung definiert eine Äquivalenzrelation auf dem Gesamtzustandsraum. Durch die Art der Konstruktion dieser Partition ist garantiert, daß die Summe der Raten aus einem gegebenen Zustand i heraus in alle Zustände einer anderen Untermenge ω_J für alle Ausgangszustände innerhalb einer Untermenge ω_I identisch ist:

$$
\begin{aligned}
q(i,J) &= \sum_{j \in \omega_J} q(i,j) \\
&= \sum_{j \in \omega_J} q\left(\left(\underbrace{i_{11},\ldots,i_{1n_1}}_{\text{Perm. mit } P_1},\ldots,\underbrace{i_{c1},\ldots,i_{cn_c}}_{\text{Perm. mit } P_c} \right), (j_{11},\ldots,j_{1n_1},\ldots,j_{c1},\ldots,j_{cn_c}) \right) \\
&= \sum_{j \in \omega_J} q\left(\left(\overbrace{i'_{11},\ldots,i'_{1n_1}}^{}\ldots\overbrace{i'_{c1},\ldots,i'_{cn_c}}^{} \right), \overbrace{(j_{11},\ldots,j_{1n_1},\ldots,j_{c1},\ldots,j_{cn_c})}^{\text{Zielzustand entspr. permutiert}} \right) \\
&= q\Big(i',J\Big) = const \ \ \forall i \in \omega_I
\end{aligned}
$$

Dies ist genau die Bedingung für gewöhnliche Zusammenfaßbarkeit. Außerdem ist gewährleistet, daß die Summe der Raten aus einer Untermenge ω_I in einen gegebenen Zustand j hinein für alle Zielzustände innerhalb einer Untermenge ω_J identisch ist:

$$
\begin{aligned}
q(I,j) &= \sum_{i \in \omega_I} q(i,j) \\
&= \sum_{i \in \omega_I} q\left((i_{11},\ldots,i_{1n_1},\ldots,i_{c1},\ldots,i_{cn_c}), \left(\underbrace{j_{11},\ldots,j_{1n_1}}_{\text{Perm. mit } P_1},\ldots,\underbrace{j_{c1},\ldots,j_{cn_c}}_{\text{Perm. mit } P_c} \right) \right) \\
&= \sum_{i \in \omega_I} q\left(\overbrace{(i_{11},\ldots,i_{1n_1},\ldots,i_{c1},\ldots,i_{cn_c})}^{\text{Ausgangszust. entspr. permutiert}}, \left(\overbrace{j'_{11},\ldots,j'_{1n_1}}^{}\ldots\overbrace{j'_{c1},\ldots,j'_{cn_c}}^{} \right) \right) \\
&= q\Big(I,j'\Big) = const \ \ \forall j \in \omega_J
\end{aligned}
$$

Damit ist die Bedingung für exakte Zusammenfaßbarkeit bezüglich der definierten Partition ebenfalls erfüllt. Da sowohl die Eigenschaften gewöhnliche Zusammenfaßbarkeit und exakte Zusammenfaßbarkeit vorliegen, folgt also insgesamt die Eigenschaft der strikten Zusammenfaßbarkeit.

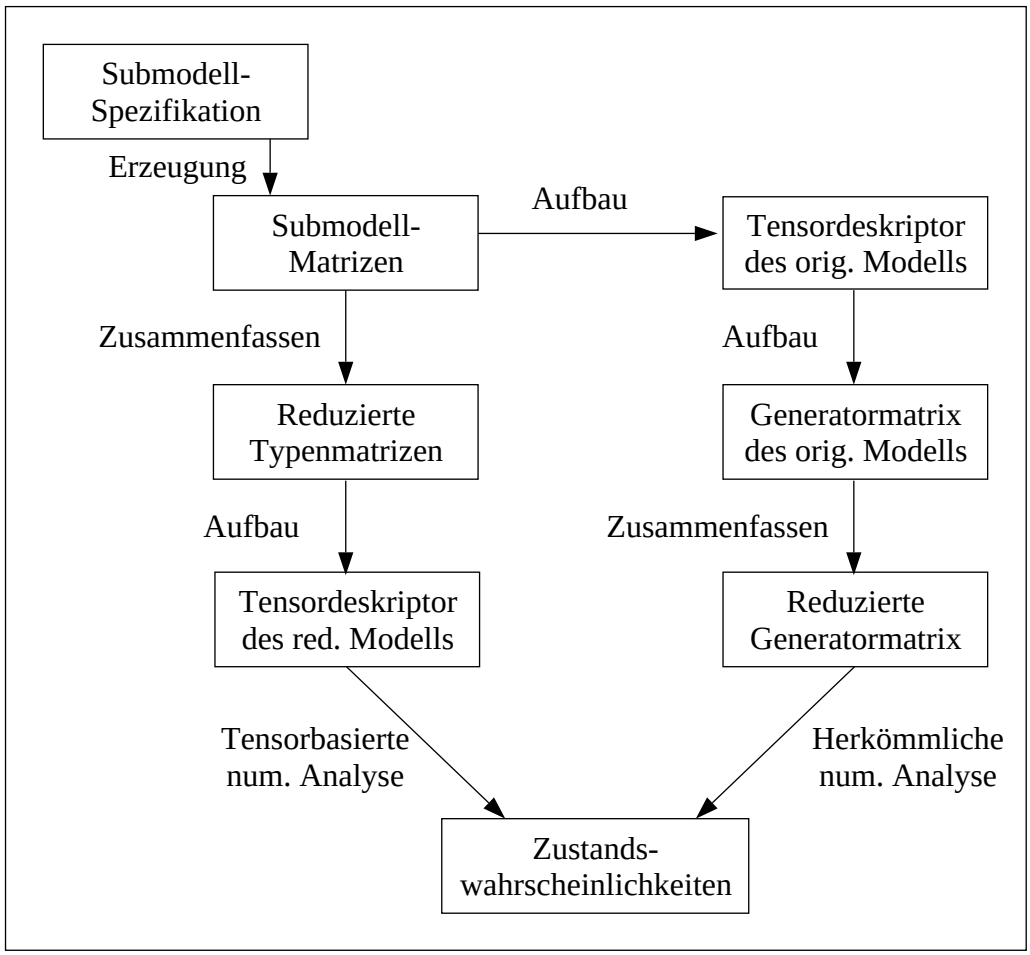


Abbildung 4.3: Zwei Wege bei der Erzeugung des reduzierten Modells

Aus diesen Überlegungen folgt, daß es zur Ermittlung exakter Analyseergebnisse genügt, ein reduziertes Modell zu betrachten, in welchem jede Äquivalenzklasse (jede Untermenge symmetrischer Zustände) durch genau einen einzigen Zustand repräsentiert wird. Als Repräsentant wird aus jeder Menge symmetrischer Zustände derjenige Zustand z ausgewählt, für den das folgende Kriterium erfüllt ist:

$$\forall k \in \{1, \ldots, c\} : \; z_{k1} \leq \ldots \leq z_{kn_k} \tag{4.4}$$

Es wird also aus jeder Untermenge symmetrischer Zustände das jeweils lexikographisch kleinste Element als Repräsentant ausgewählt.

Im Anhang A.2 wird erläutert, daß sich die Generatormatrix $\hat{Q}$ des aggregierten Markovprozesses als das Matrixprodukt

$$\hat{Q} = UQV$$

schreiben läßt, wobei V die Funktion einer Projektionsmatrix und U die Funktion einer Selektionsmatrix hat.

Wie Abb. 4.3 zeigt, gibt es jedoch in der hier vorgestellten Modellwelt *zwei* denkbare Wege zur Erzeugung des reduzierten Modells. Der naive Ansatz führt über die explizite Berechnung der Generatormatrix Q des Gesamtmodells aus ihrem Tensordeskriptor,

wie er in Gl. (4.1) gegeben ist und die anschließende Multiplikation mit den auf Basis der Symmetriebedingungen bestimmbaren Matrizen U und V. Diese Vorgehensweise entspräche dem rechten Pfad durch das Schema in Abb. 4.3. Wegen der Dimension der Matrix Q wäre ein solches Vorgehen unpraktisch und in vielen Fällen aufgrund von Speicherplatzbeschränkungen sogar unmöglich.

Wir benutzen stattdessen die über die Submodelltypen zur Verfügung stehende Information und führen die Aggregierung von Zuständen auf Submodelltypenebene *vor* dem Aufbau des Tensordeskriptors durch. Dieses Vorgehen ist im linken Pfad durch Abb. 4.3 dargestellt und bringt die folgenden praktischen Vorteile:

- Das Zusammenfassen von Zuständen geschieht nicht auf der Ebene des Gesamtmodells, sondern findet auf der überschaubaren Ebene von Submodellen desselben Typs statt.
- Das Zusammenfassen kann auf effiziente Weise durchgeführt werden. In der Praxis muß nicht auf die allgemeine Methode der Multiplikation $\hat{Q} = UQV$ zurückgegriffen werden, sondern es ist möglich, einen effizienten Algorithmus für das Zusammenfassen von Zuständen anzugeben (siehe Abschnitt 4.6).
- Für die anschließende numerische Analyse des reduzierten Modells kann der speicherplatzsparende Tensoransatz benutzt werden.

Wir richten daher unsere Aufmerksamkeit auf die zum Typ i gehörenden Submodelle. Der kombinierte Zustandsraum Z_i aller Submodelle vom Typ i besitzt $s_i^{n_i}$ Zustände, wobei jeder Zustand durch ein n_i-Tupel beschrieben ist. Hier gilt wiederum, daß zwei Zustände $x = (x_1, \ldots, x_{n_i})$ und $y = (y_1, \ldots, y_{n_i})$ genau dann symmetrisch sind, wenn y eine Permutation von x ist, d.h. falls eine Permutationsmatrix P_i der Dimension n_i existiert, so daß gilt $y = xP_i$. Es handelt sich dabei um dieselben Matrizen P_i aus denen die Matrix P in Gl. (4.3) aufgebaut ist.

Wegen der Eigenschaften der Tensoroperatoren sind die Zustände innerhalb des kombinierten Zustandsraums der Submodelle vom Typ i lexikographisch geordnet. Dies hat zur Folge, daß die bei der Zusammenfassung dieser Zustände benötigten Matrizen U und V lediglich von zwei Parametern abhängen. Diese Parameter sind die Anzahl n_i der Submodelle dieses Typs und die Kardinalität s_i des Zustandsraums eines Submodells dieses Typs. Es ist also (erstaunlicherweise) möglich, die bei der Reduktion verwendeten Matrizen U und V anzugeben, ohne das konkrete Verhalten der Submodelle, welches durch die Übergänge zwischen ihren Zuständen spezifiziert ist, zu kennen. Die Matrizen U und V sind bei gegebenem n_i und s_i a priori bekannt und wir werden sie in der Folge mit $U_{s_i}^{n_i}$ und $V_{s_i}^{n_i}$ bezeichnen.

Wir können nun sofort die Elemente der Matrizen $U_{s_i}^{n_i}$ und $V_{s_i}^{n_i}$ unter Verwendung von Tupelbezeichnungen für den Ausgangs- und den Zielzustand angegeben. Für die Matrix $V_{s_i}^{n_i}$ ergibt sich

$$V_{s_i}^{n_i}((k_1, \ldots, k_{n_i}), (l_1, \ldots, l_{n_i})) = \begin{cases} 1 & \text{falls } (l_1, \ldots, l_{n_i}) = Perm(k_1, \ldots, k_{n_i}) \\ 0 & \text{sonst} \end{cases} \quad (4.5)$$

wobei das l-Tupel die Bedingung $l_1 \leq \ldots \leq l_{n_i}$ zu erfüllen hat, da es in gleicher Weise wie die Bedingung in Gl. (4.4) einen repräsentativen Zustand bezeichnet.

Die Matrix $U_{s_i}^{n_i}$ kann auf entsprechende Weise charakterisiert werden:

$$U_{s_i}^{n_i}((k_1,\ldots,k_{n_i}),(l_1,\ldots,l_{n_i})) = \begin{cases} 1 & \text{falls } (l_1,\ldots,l_{n_i}) = (k_1,\ldots,k_{n_i}) \\ 0 & \text{sonst} \end{cases} \tag{4.6}$$

Für das k-Tupel muß die Bedingung $k_1 \leq \ldots \leq k_{n_i}$ erfüllt sein.

Die im Anhang A.2 erläuterten Freiheitsgrade für die Selektionsmatrix U (Zustandsgewichte) wurden hierbei so gewählt, daß auf genau einen Zustand (den Repräsentanten) das gesamte Gewicht entfällt. Es wird sich herausstellen, daß die Konzentration der gesamten Masse auf einen Zustand die effizienteste Wahl ist, da sie erlaubt, vom Matrixprodukts QV nur jeweils eine Zeile pro Menge äquivalenter Zustände zu berechnen.

Es soll schon an dieser Stelle kurz darauf hingewiesen werden, daß es bei einer expliziten Berechnung von Matrixprodukten der Form $U_{s_i}^{n_i}\tilde{Q}^{(i)}V_{s_i}^{n_i}$ (vgl. Abschnitt 4.5.4, S. 93) effizienter ist, die linke Matrixmultiplikation vor der rechten auszuführen, da $U_{s_i}^{n_i}$ weniger von Null verschiedene Elemente enthält als $V_{s_i}^{n_i}$. Anders ausgedrückt heißt das, daß bei umgekehrter Wahl der Reihenfolge im Produkt $\tilde{Q}^{(i)}V_{s_i}^{n_i}$ auch solche Zeilen berechnet würden, welche anschließend von der Selektionsmatrix $U_{s_i}^{n_i}$ gar nicht ausgewählt werden. Diese Überlegung ist wichtig im Hinblick auf Abschnitt 4.6, wo gezeigt wird, daß für die Berechnung der Matrixprodukte ein effizienter Algorithmus angegeben werden kann, der unter anderem diesen Sachverhalt berücksichtigt.

4.5.2 Aufbau und Eigenschaften der Projektionsmatrizen

In der folgenden Diskussion werden der Aufbau und einige Eigenschaften der Matrizen vom Typ $V_{s_i}^{n_i}$ und $U_{s_i}^{n_i}$ untersucht. Zur Vereinfachung der Schreibweise wird in diesem Abschnitt der Index i fallen gelassen, so daß wir die Projektionsmatrix mit V_s^n und die Selektionsmatrix mit U_s^n bezeichnen werden, ohne dabei zu vergessen, daß diese Matrizen zur Zusammenfassung von Zuständen auf Ebene der Submodelle des Typs i dienen.

Zunächst soll die Ausgangssituation noch einmal kurz allgemein beschrieben werden:

Ausgangspunkt: Gegeben ist ein Markovprozeß mit Zustandsraum $Z = \{(z_1, z_2, \ldots, z_n) \mid \forall i:\ 0 \leq z_i \leq s-1\}$. Die Zustände sind n-Tupel, wobei jede Komponente s verschiedene Werte annehmen kann.
Ferner ist eine Äquivalenzrelation $\equiv$ auf Z auf folgende Weise definiert
$\forall a, b \in Z:\ a \equiv b \iff (a \text{ ist Permutation von } b)$

Es folgt daraus unmittelbar, daß der Zustandsraum $R_s^n = |Z| = s^n$ verschiedene Elemente beinhaltet. Aufgrund kombinatorischer Überlegungen beträgt die Anzahl der Äquivalenzklassen $C_s^n = \binom{n+s-1}{s-1}$. Wir wollen die Matrizen U und V ermitteln, die für das Zusammenfassen dieses stochastischen Prozesses benötigt werden. Es geht also darum, eine Projektionsmatrix zu bestimmen, welche eine Abbildung der Zustände auf Äquivalenzklassen leistet. Außerdem ist eine Selektionsmatrix U_s^n anzugeben. Dabei sei stets vorausgesetzt, daß die Zustände $(z_1, z_2, \ldots, z_n)$ innerhalb der Generatormatrix des Markovprozesses in lexikographisch aufsteigender Reihenfolge angeordnet sind. Diese Voraussetzung ist für die Submodell-Tensordeskriptoren wegen der Eigenschaften der

Tensoroperatoren erfüllt. Durch die Zusammenfassung von symmetrischen Zuständen wird die Größe des Zustandsraums des Prozesses von $R_s^n = s^n$ auf C_s^n Zustände verringert.

Problemstellung: Bestimmung der Projektionsmatrix $V_s^n \in \mathbb{R}^{R_s^n \times C_s^n}$ mit Elementen aus $\{0, 1\}$, durch die die Zustände auf ihre Äquivalenzklassen abgebildet werden. Bestimmung der Selektionsmatrix $U_s^n \in \mathbb{R}^{C_s^n \times R_s^n}$.
Wie kann man bei gegebenem s, n und Indexpaar (k, l) entscheiden, ob $V_s^n(k, l)$ bzw. $U_s^n(l, k)$ gleich 0 oder gleich 1 ist?
Dabei gelte $0 \leq k \leq s^n - 1 \wedge 0 \leq l \leq C_s^n - 1$.

Die folgende Betrachtung wird sich auf V_s^n-Matrizen konzentrieren, aber es lassen sich leicht analoge Überlegungen für die entsprechenden U_s^n-Matrizen anstellen. Beispiele für Matrizen des Typs V_s^n sind in Abb. 4.4 angegeben.

$V_2^3 =$

	000	001	011	111
000	1			
001		1		
010		1		
011			1	
100		1		
101			1	
110			1	
111				1

$V_4^2 =$

	00	01	02	03	11	12	13	22	23	33
00	1									
01		1								
02			1							
03				1						
10		1								
11					1					
12						1				
13							1			
20			1							
21						1				
22								1		
23									1	
30				1						
31							1			
32									1	
33										1

$V_3^2 =$

	00	01	02	11	12	22
00	1					
01		1				
02			1			
10		1				
11				1		
12					1	
20			1			
21					1	
22						1

Abbildung 4.4: Drei Beispiele für Projektionsmatrizen V_s^n

Folgender Hilfssatz ist für die nachfolgenden Überlegungen nützlich:

Hilfssatz: C_s^n läßt sich als Summe schreiben:

$$C_s^n = C_s^{n-1} + C_{s-1}^{n-1} + \ldots + C_2^{n-1} + C_1^{n-1} = \sum_{i=0}^{s-1} C_{s-i}^{n-1}$$

Diesen Satz kann man leicht über die Beziehung zwischen Binomialkoeffizienten beweisen, bzw. sich anhand des Pascal'schen Dreiecks veranschaulichen.

Es ist möglich, für einen gegebenen Repräsentanten einer Äquivalenzklasse, d.h. für einen Zustand $z = (z_1, z_2, \ldots, z_n)$ mit $z_1 \leq z_2 \leq \ldots \leq z_n$, die Position dieses Zustands zu ermitteln. Diese Position bezeichnet die dem Zustand entsprechende Spaltennummer innerhalb der Matrix V_s^n bzw. die Zeilennummer innerhalb der Matrix U_s^n (Die erste Spalte bzw. Zeile erhält die Nummer 0). Die dafür wichtigen Zusammenhänge kann man sich anhand des Schemas in Abb. 4.5 klarmachen.

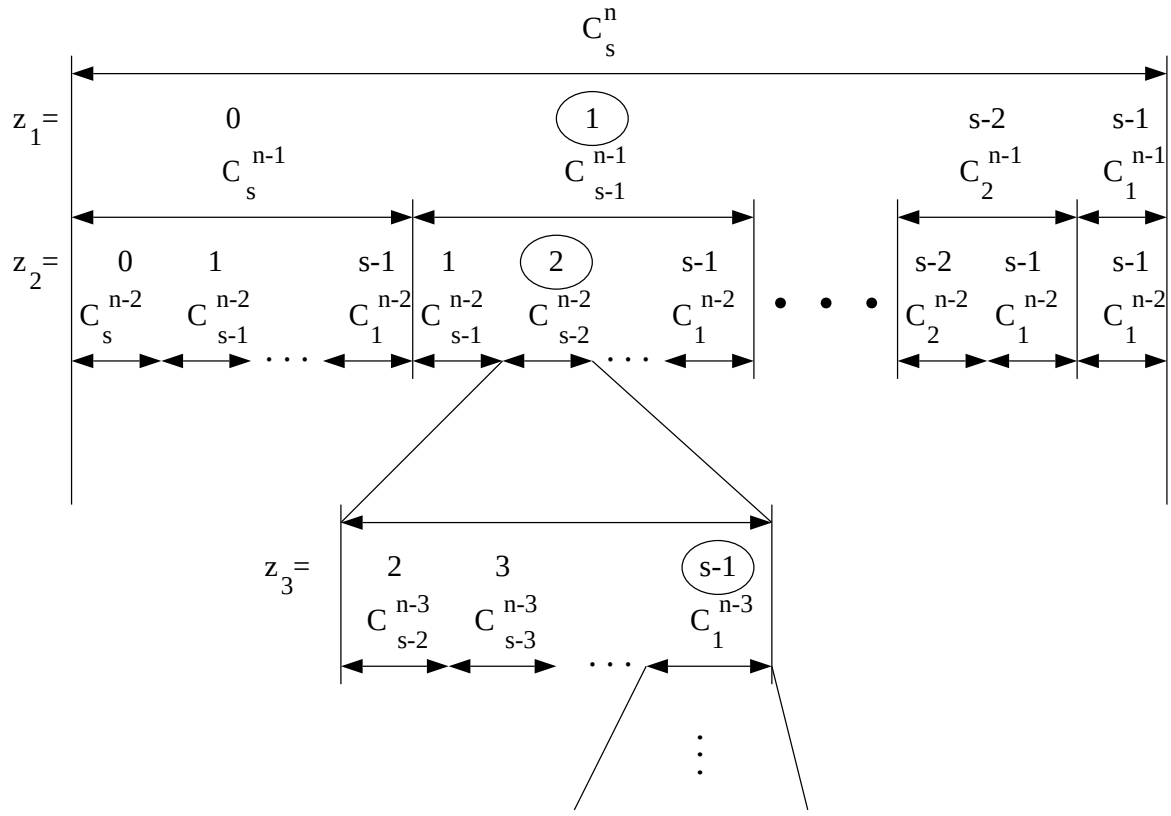


Abbildung 4.5: Schema zur Bestimmung der Position des repräsentativen Zustands $z = (z_1, z_2, \ldots, z_n)$. Eingekreiste Ziffern heben den Zustand $z = (1, 2, s-1, \ldots)$ hervor.

Das Schema veranschaulicht die Anordung der repräsentativen Zustände in den Spalten der Matrix V_s^n. Diese Anordung erhält man aus der lexikographischen Ordnung aller Tupel $(z_1, z_2, \ldots, z_n)$ durch Streichen derjenigen Zustände, die die Bedingung für einen Repräsentanten nicht erfüllen. Im Schema ist eine geschachtelte Unterteilung in Gruppen von Zuständen mit gleicher führender Ziffer dargestellt. Mit den C_x^y ist jeweils die Anzahl der zu den Gruppen bzw. Untergruppen gehörenden Zustände angegeben.

Der Zusammenhang zwischen der Größe einer Gruppe und der Summe der Größen ihrer Untergruppen gilt aufgrund des oben angegeben Hilfssatzes. Im Schema ist aus Platzgründen nur ein Teil vollständig ausgeführt. Mit den eingekreisten Ziffern ist das Auffinden der Position eines Zustands mit der führenden Ziffernfolge $(1, 2, s-1, \ldots)$ angedeutet.

Für einen repräsentativen Zustand z bezeichne die Funktion $\mathrm{Pos}_s^n(z)$ seine Position innnerhalb der Spalten der Matrix V_s^n. Aus dem Schema der Abb. 4.5 kann man die folgende Beziehung ablesen (man definiert zusätzlich $z_0 = 0$):

$$\mathrm{Pos}_s^n(z) = \sum_{j=1}^{n} \sum_{i=z_{j-1}}^{z_j-1} C_{s-i}^{n-j}$$

Bevorzugt man eine rekursive Schreibweise, so ergibt sich

$$\mathrm{Pos}_s^n(z_1, z_2, \ldots, z_n) = \sum_{i=0}^{z_1-1} C_{s-i}^{n-1} + \mathrm{Pos}_{s-z_1}^{n-1}(z_2 - z_1, z_3 - z_1, \ldots, z_n - z_1)$$

Beispiel:
Seien $s = 3$ und $n = 2$. Die Spaltenposition des Zustands $z = (1, 2)$ berechnet sich wie folgt

$$\begin{aligned}
\mathrm{Pos}_3^2(1, 2) &= \sum_{i=0}^{0} C_{3-i}^{2-1} + \sum_{i=1}^{1} C_{3-i}^{2-2} \\
&= C_3^1 + C_2^0 \\
&= \binom{1+2}{2} + \binom{0+1}{1} \\
&= 3 + 1 = 4
\end{aligned}$$

Ein Vergleich mit der Matrix V_3^2 in Abb. 4.4 zeigt, daß der Zustand $z = (1, 2)$ tatsächlich die Spaltenposition 4 einnimmt (die Spaltenzählung beginnt mit 0).

Mit diesem Zwischenergebnis sind wir unserem Ziel, für ein beliebiges Element (k, l), $0 \leq k \leq s^n - 1 \wedge 0 \leq l \leq C_s^n - 1$, der Matrix V_s^n zu entscheiden, ob es gleich Null oder gleich eins ist, bedeutend näher gekommen. Es sind dazu die drei folgenden Schritte auszuführen:

1. Umwandlung von k in eine n-stellige Zahl zur Basis s.
2. Durch Sortieren der Ziffern von k erhält man den Repräsentanten der Äquivalenzklasse, zu welcher k gehört.
3. Auf den Repräsentanten wird die Funktion $\mathrm{Pos}_s^n()$ angewendet.

Für die Durchführung dieser Schritte werden die beiden zusätzlichen Funktionen $\mathrm{digit}()$ und $\mathrm{sort}()$ definiert. Die Funktion $\mathrm{digit}()$ überführt eine Zahl k in die Darstellung zur Basis s, d.h. in ein Tupel $(k_1, k_2, \ldots, k_n)$ mit $0 \leq k_i \leq s - 1$. Die Funktion $\mathrm{sort}()$ permutiert (sortiert) dieses Tupel so, daß gilt $k_1 \leq k_2 \leq \ldots \leq k_n$, d.h. damit ist der Repräsentant der Äquivalenzklasse von k bestimmt. Unter Zuhilfenahme dieser beiden

Funktionen lassen sich die Elemente der Matrix V_s^n ähnlich wie in Gl. (4.5), aber ohne die Verwendung von Tupelbezeichnungen, schreiben als

$$V_s^n(k,l) = \begin{cases} 1 & \text{falls } \mathrm{Pos}_s^n(\mathrm{sort}(\mathrm{digit}(k))) = l \\ 0 & \text{sonst} \end{cases}$$

4.5.3 Beziehung zwischen den Projektionsmatrizen

Innerhalb der hier vorgestellten Modellwelt ist ein weiterer wichtiger Punkt von Interesse: Wir stellen fest, daß die Projektionsmatrix V für das Gesamtmodell in einer engen Beziehung zu den Projektionsmatrizen $V_{s_i}^{n_i}$ der Submodelltypen steht. Eine gleiche Beobachtung gilt für die Beziehung zwischen der Selektionsmatrix U und den Matrizen $U_{s_i}^{n_i}$.
Wir zeigen daher die beiden für den folgenden Abschnitt 4.5.4 wichtigen Beziehungen

$$U = \bigotimes_{i=1}^{c} U_{s_i}^{n_i} \quad , \quad V = \bigotimes_{i=1}^{c} V_{s_i}^{n_i} \tag{4.7}$$

die für den Nachweis von Gl. (4.12) benötigt werden.
Der Nachweis wird über eine Inspektion der Elemente der Matrizen U, V und $U_{s_i}^{n_i}, V_{s_i}^{n_i}$ geführt. Wir konzentrieren uns auf die Betrachtung der Matrix V. Diese hat die Dimension $\left(\prod_{i=1}^{c} R_{s_i}^{n_i}\right) \times \left(\prod_{i=1}^{c} C_{s_i}^{n_i}\right)$, und ihre Elemente sind (in Tupelschreibweise) gegeben durch

$$\begin{aligned} &V((i_{11},\ldots,i_{1n_1},\ldots,i_{c1},\ldots,i_{cn_c}),(j_{11},\ldots,j_{1n_1},\ldots,j_{c1},\ldots,j_{cn_c})) \\ =&\begin{cases} 1 & \text{if } (j_{11},\ldots,j_{1n_1},\ldots,j_{c1},\ldots,j_{cn_c}) = P(i_{11},\ldots,i_{1n_1},\ldots,i_{c1},\ldots,i_{cn_c}) \\ 0 & \text{else} \end{cases} \end{aligned} \tag{4.8}$$

Dabei gilt für die j-Indizes wieder die Bedingung

$$\forall k \in \{1,\ldots,c\}: \; j_{k1} \leq \ldots \leq j_{kn_k} \tag{4.9}$$

welche denselben Sachverhalt ausdrückt wie die Bedingung in Gl. (4.4). Man beachte, daß P in Gl. (4.8) dieselbe Permutationsmatrix bezeichnet wie in Gl. (4.3).
Wir vergleichen nun die Elemente von V mit den Elementen der Matrix $W = \bigotimes_{i=1}^{c} V_{s_i}^{n_i}$. Letztere lassen sich in Tupelschreibweise wie folgt darstellen:

$$\begin{aligned} &W((i_{11},\ldots,i_{1n_1},\ldots,i_{c1},\ldots,i_{cn_c}),(j_{11},\ldots,j_{1n_1},\ldots,j_{c1},\ldots,j_{cn_c})) \\ =&V_{s_1}^{n_1}((i_{11},\ldots,i_{1n_1}),(j_{11},\ldots,j_{1n_1})) \cdot \ldots \cdot V_{s_c}^{n_c}((i_{c1},\ldots,i_{cn_c}),(j_{c1},\ldots,j_{cn_c})) \\ =&\begin{cases} 1 & \text{if } (j_{11},\ldots,j_{1n_1}) = P_1(i_{11},\ldots,i_{1n_1}) \wedge \ldots \wedge (j_{c1},\ldots,j_{cn_c}) = P_c(i_{c1},\ldots,i_{cn_c}) \\ 0 & \text{else} \end{cases} \end{aligned} \tag{4.10}$$

Der erste Umformungsschritt in dieser Gleichung folgt direkt aus der Definition des Tensorprodukts. Die Bedingung für die j-Indizes aus Gl. (4.9) muß auch hier erfüllt sein.

Beim Vergleich der 1-Bedingungen in Gl. (4.8) und Gl. (4.10) stellen wir fest, daß diese wegen der Beziehung zwischen P und den Matrizen P_i identisch sind. Damit ist die Äquivalenz zwischen den Matrizen V und W bewiesen, es gilt also $V = \bigotimes_{i=1}^{c} V_{s_i}^{n_i}$.

Beispiel:
Im Multiprozessorbeispiel hat die Matrix $V_{s_1}^{n_1} = V_5^2$ bereits die beträchtliche Dimension 25×15. Die Dimension der Projektionsmatrix V für das Gesamtmodell ist sogar 300×180. Um ein Beispiel für $V = \bigotimes_{i=1}^{c} V_{s_i}^{n_i}$ aufzuzeigen, verlassen wir daher dieses Beispiel zugunsten eines sehr einfachen, welches sich noch leicht darstellen läßt: Wir wollen annehmen, daß im Gesamtmodell nur Submodelle von zwei Typen enthalten sind und daß gilt: $n_1 = 2$, $s_1 = 2$, $n_2 = 1$ und $s_2 = 3$.

$V_2^2 =$

	00	01	11
00	1		
01		1	
10		1	
11			1

$V_3^1 =$

	0	1	2
0	1		
1		1	
2			1

$V = V_2^2 \otimes V_3^1 =$

	000	001	002	010	011	012	110	111	112
000	1								
001		1							
002			1						
010				1					
011					1				
012						1			
100				1					
101					1				
102						1			
110							1		
111								1	
112									1

Die Matrix V_3^1 ist eine Identitätsmatrix der Dimension 3, da bei nur einem Submodell pro Typ natürlich keine Zustandsraumreduktion auf Typenebene möglich ist. Man beachte, daß die Tupelbezeichnungen der Spalten automatisch in aufsteigend lexikographischer Reihenfolge angeordnet sind. Wichtig ist ferner, daß z.B. die Zustandspaare $(0,0,1)$ und $(0,1,0)$ keine äquivalenten Zustände bezeichnen, da die ersten beiden Positionen den Zustand von Submodellen des Typs 1, die dritte Position dagegen den Zustand des Submodells vom Typ 2 bezeichnen.

4.5.4 Zusammenfaßbarkeit auf Ebene der Submodelltypen

In diesem Abschnitt wird der Beweis geführt, daß die Aggregierung von Zuständen, das Zusammenfassen, auf Typenebene durchgeführt werden kann. Es wird also formal gezeigt, daß die beiden in Abb. 4.3 dargestellten Wege, zu einem reduzierten Gesamtmodell zu gelangen, zu ein und demselben Ergebnis führen. Bei dem Beweis werden Rechenregeln der Tensoralgebra sowie Eigenschaften der Projektions- und Selektionsmatrizen, die in den Abschnitten 4.5.2 und 4.5.3 erarbeitet wurden, angewendet.

Zunächst werden die drei folgenden Typen von Matrizen definiert, die den kombinierten stochastischen Prozeß der Submodelle vom Typ i in seiner ausführlichen Form, d.h. vor einer eventuellen Zustandsraumreduktion, beschreiben. Matrizen dieses Typs werden in dieser Arbeit durch eine Tilde ˜ gekennzeichnet.

$$\tilde{Q}_l^{(i)} = \bigoplus_{j=1}^{n_i} Q_l^{(i)}, \quad \tilde{Q}_e^{(i)} = \bigodot_{j=1}^{n} Q_e^{(i)}, \quad \tilde{Q}_{e,n}^{(i)} = \bigodot_{j=1}^{n} Q_{e,n}^{(i)}$$

In der Matrix $\tilde{Q}_l^{(i)}$ sind alle lokalen Ereignisse der Submodelle vom Typ i zusammengefaßt, während die Matrizen $\tilde{Q}_e^{(i)}$ und $\tilde{Q}_{e,n}^{(i)}$ dazu dienen, die Art der Teilnahme von Submodellen des Typs i an dem synchronisierenden Ereignis e zu beschreiben. Die Interpretation des Symbols $\bigodot$ ist wie im Abschnitt 4.4.2.

Die Matrizen für den reduzierten stochastischen Prozeß der Submodelle vom Typ i kann man nun mit Hilfe dieser Definition, der Projektionsmatrix $V_{s_i}^{n_i}$ und der Selektionsmatrix $U_{s_i}^{n_i}$ wie folgt angeben:

$$\hat{Q}_l^{(i)} = U_{s_i}^{n_i} \tilde{Q}_l^{(i)} V_{s_i}^{n_i}, \quad \hat{Q}_e^{(i)} = U_{s_i}^{n_i} \tilde{Q}_e^{(i)} V_{s_i}^{n_i}, \quad \hat{Q}_{e,n}^{(i)} = U_{s_i}^{n_i} \tilde{Q}_{e,n}^{(i)} V_{s_i}^{n_i}$$

Das Dach ˆ kennzeichnet in dieser Arbeit Matrizen und Konstanten (vgl. $\hat{d}_j$ unten in Gl. (4.12)), die sich auf den reduzierten Prozeß beziehen.

Für den Beweis werden wir nun die Äquivalenz zweier Ausdrücke für die Generatormatrix der reduzierten Markovkette zeigen. Wir vergleichen den Ausdruck, den man erhält, wenn die Aggregierung nach Aufbau der Generatormatrix des Gesamtmodells, wie sie in Gl. (4.1) dargestellt ist, durchgeführt wird, mit dem Ausdruck, der sich ergibt, wenn man die Generatormatrix des Gesamtmodells aus den bereits auf Klasssenebene

reduzierten Submodellmatrizen erzeugt. Es ist also die folgende Äquivalenz zu zeigen:

$$\begin{aligned}\hat{Q} &= U\left(\bigoplus_{i=1}^{c}\tilde{Q}_l^{(i)} + \sum_{e\in E}\lambda_e\left(\bigotimes_{i=1}^{c}\tilde{Q}_e^{(i)} - \bigotimes_{i=1}^{c}\tilde{Q}_{e,n}^{(i)}\right)\right)V \\ &\stackrel{!}{=} \quad \bigoplus_{i=1}^{c}\hat{Q}_l^{(i)} + \sum_{e\in E}\lambda_e\left(\bigotimes_{i=1}^{c}\hat{Q}_e^{(i)} - \bigotimes_{i=1}^{c}\hat{Q}_{e,n}^{(i)}\right)\end{aligned} \tag{4.11}$$

Beide in Gl. (4.11) einander gegenübergestellten Ausdrücke sind aus drei Teilen aufgebaut. Es wird sich herausstellen, daß zwischen den einander entsprechenden Teilen Äquivalenz herrscht, so daß der Beweis in drei voneinander unabhängige Teile zerfällt. Am kompliziertesten gestaltet sich der Nachweis der Äquivalenz für den ersten Teil, der sich auf die lokalen Ereignisse bezieht und auf Matrizen des Typs $\tilde{Q}_l^{(i)}$ aufbaut. Wir beginnen mit dem Term $U\left(\bigoplus_{i=1}^{c}\tilde{Q}_l^{(i)}\right)V$ der linken Seite und zeigen durch Umformungen, daß er äquivalent zum Term $\bigoplus_{i=1}^{c}\hat{Q}_l^{(i)}$ der rechten Seite ist.

$$\begin{aligned}&U\left(\bigoplus_{i=1}^{c}\tilde{Q}_l^{(i)}\right)V \\ =&\left(\bigotimes_{i=1}^{c}U_{s_i}^{n_i}\right)\left(\sum_{i=1}^{c}I_{d_1}\otimes\ldots\otimes I_{d_{i-1}}\otimes\tilde{Q}_l^{(i)}\otimes I_{d_{i+1}}\otimes\ldots\otimes I_{d_c}\right)\left(\bigotimes_{i=1}^{c}V_{s_i}^{n_i}\right) \\ =&\sum_{i=1}^{c}\left(\left(\bigotimes_{i=1}^{c}U_{s_i}^{n_i}\right)\left(I_{d_1}\otimes\ldots\otimes I_{d_{i-1}}\otimes\tilde{Q}_l^{(i)}\otimes I_{d_{i+1}}\otimes\ldots\otimes I_{d_c}\right)\left(\bigotimes_{i=1}^{c}V_{s_i}^{n_i}\right)\right) \\ =&\sum_{i=1}^{c}\left(\bigotimes_{j=1}^{i-1}\left(U_{s_j}^{n_j}I_{d_j}V_{s_j}^{n_j}\right)\right)\otimes\left(U_{s_i}^{n_i}\tilde{Q}_l^{(i)}V_{s_i}^{n_i}\right)\otimes\left(\bigotimes_{j=i+1}^{c}\left(U_{s_j}^{n_j}I_{d_j}V_{s_j}^{n_j}\right)\right) \\ =&\sum_{i=1}^{c}\left(\bigotimes_{j=1}^{i-1}I_{\hat{d}_j}\right)\otimes\left(U_{s_i}^{n_i}\tilde{Q}_l^{(i)}V_{s_i}^{n_i}\right)\otimes\left(\bigotimes_{j=i+1}^{c}I_{\hat{d}_j}\right) \\ =&\bigoplus_{i=1}^{c}\left(U_{s_i}^{n_i}\tilde{Q}_l^{(i)}V_{s_i}^{n_i}\right) \\ =&\bigoplus_{i=1}^{c}\hat{Q}_l^{(i)}\end{aligned} \tag{4.12}$$

Im einzelnen werden in Gl. (4.12) folgende Umformungen vorgenommen: Zunächst werden die Matrizen U und V in ihrer Form als Tensorprodukte dargestellt, die aus Abschnitt 4.5.3 bekannt ist (vgl. Gl. (4.7)). Die Tensorsumme über die Matrizen $\tilde{Q}_l^{(i)}$ wird als Matrixsumme von Tensorprodukten geschrieben (siehe Anhang B.1, S. 133, Gl. (B.2)). Daraufhin wird die Matrixsumme nach außen gestellt und anschließend die Regel der Gl. (B.3) angewendet. Dabei ergibt sich die besondere Behandlung des i-ten Faktors innerhalb des Tensorprodukts, da dieser der einzige Faktor ist, bei dem

keine Identitätsmatrix auftritt. Wir machen die Feststellung, daß die leicht nachprüfbare Beziehung

$$U_{s_j}^{n_j} I_{d_j} V_{s_j}^{d_j} = I_{\hat{d}_j}$$

gilt. Dabei bezeichnet d_j die Anzahl der Zustände des kombinierten Prozesses des Submodelltyps j *vor* der Reduktion, und $\hat{d}_j$ die entsprechende Zustandszahl *nach* der Reduktion, d.h. es gelten die Beziehungen $d_j = R_{s_j}^{n_j}$ und $\hat{d}_j = C_{s_j}^{n_j}$. Als letzter Schritt erfolgt dann die Rückumwandlung der Matrixsumme in eine Tensorsumme.

Für den zweiten Teil der Gl. (4.11), wo es um die Matrizen des Typs $\tilde{Q}_e^{(i)}$ geht, weist man die Äquivalenz wiederum unter Anwendung der Eigenschaft von Gl. (B.3) aus Anhang B.1 und mit Hilfe der Tensordarstellungen für die Matrizen U und V durch folgende Betrachtung nach:

$$\begin{aligned}
&U\left(\sum_{e\in E}\lambda_e\left(\bigotimes_{i=1}^{c}\tilde{Q}_e^{(i)}\right)\right)V\\
=&\sum_{e\in E}\lambda_e U\left(\bigotimes_{i=1}^{c}\tilde{Q}_e^{(i)}\right)V\\
=&\sum_{e\in E}\lambda_e\left(\bigotimes_{i=1}^{c}U_{s_i}^{n_i}\right)\left(\bigotimes_{i=1}^{c}\tilde{Q}_e^{(i)}\right)\left(\bigotimes_{i=1}^{c}V_{s_i}^{n_i}\right) \qquad (4.13)\\
=&\sum_{e\in E}\lambda_e\left(\bigotimes_{i=1}^{c}\left(U_{s_i}^{n_i}\tilde{Q}_e^{(i)}V_{s_i}^{n_i}\right)\right)\\
=&\sum_{e\in E}\lambda_e\left(\bigotimes_{i=1}^{c}\hat{Q}_e^{(i)}\right)
\end{aligned}$$

Eine analoge Überlegung wie für den zweiten Teil kann auch für den dritten Teil der Gl. (4.11), an welchem die Matrizen $\tilde{Q}_{e,n}^{(i)}$ beteiligt sind, angestellt werden, so daß insgesamt die Gültigkeit der Gl. (4.11) und damit die Behauptung bewiesen ist.

4.5.5 Quantifizierung der Reduktion des Zustandsraums

Das Anwenden des Prinzips der Zusammenfaßbarkeit auf der Ebene von Submodellen desselben Typs hat den bedeutenden Vorteil, daß die Generatormatrix Q des Gesamtmodells für die Ausnutzung von Symmetrien niemals explizit aufgebaut werden muß. Das heißt also, daß das Zusammenfassen durchgeführt werden kann, während gleichzeitig — durchgehend durch den gesamten Modellierungsablauf — voll von den Speicherplatzersparnissen des Tensordeskriptoransatzes profitiert wird.

Die Matrizen, welche den aggregierten Markovprozeß beschreiben, werden in einem Vorverarbeitungsschritt für jeden Submodelltyp unabhängig berechnet. Im Vergleich mit der Dimension von Q sind die Matrizen, mit denen bei der Aggregierung auf Submodelltypenebene gearbeitet wird, sehr klein. Im folgenden Abschnitt 4.6 wird ein neuer, effizienter Algorithmus für die Aggregierung replizierter identischer Submodelle vorgestellt.

Wir sind nun in der Lage, die Größe des ursprünglichen und des aggregierten Zustandsraums zu vergleichen. Allgemein gilt, daß sich die Größe des Zustandsraums in

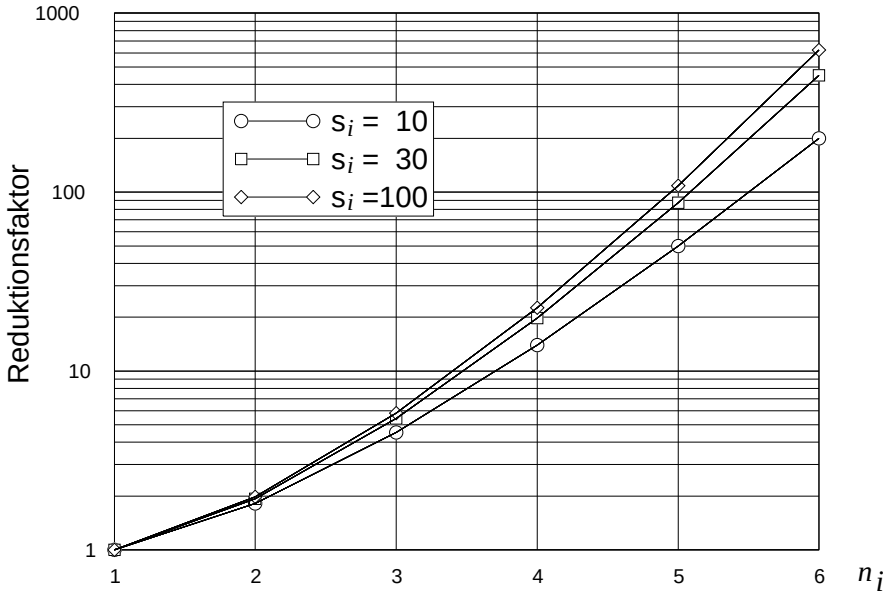


Abbildung 4.6: Reduktionsfaktor des kombinierten Zustandsraums von Submodellen desselben Typs in Abhängigkeit von n_i für einige Parameter s_i

bezug auf einen Submodelltyp um den Faktor $s_i^{n_i}/C_{s_i}^{n_i}$ reduziert. Dieser Faktor ist in Abb. 4.6 gegenüber der Anzahl n_i der zu einem Typ gehörenden Submodelle aufgetragen (vgl. [Siegle, 1994b]). Dabei wird zusätzlich der Parameter s_i, d.h. die Größe des Zustandsraums eines einzelnen Submodells dieses Typs, variiert, wodurch sich eine Kurvenschar ergibt. Man erkennt in der Abbildung, daß der Gewinn durch die Zustandsraumreduktion in der Tat enorm ist! Die Reduktion lohnt sich umso mehr, je größer die Anzahl n_i der replizierten Submodelle ist.

Nach der Durchführung des Zusammenfassens auf Submodelltypenebene wird der Tensordeskriptor $\hat{Q}$ des reduzierten Gesamtmodells aus den reduzierten Submodellmatrizen $\hat{Q}_l^{(i)}, \hat{Q}_e^{(i)}$ und $\hat{Q}_{e,n}^{(i)}$ aufgebaut. Das Gesamtmodell profitiert auf diese Weise voll von der Reduktion auf Submodelltypenebene. Sein Zustandsraum wird um den Faktor $\prod_{i=1}^{c} s_i^{n_i}/C_{s_i}^{n_i}$, also um das Produkt der Reduktionsfaktoren auf Typenebene, verringert.

Beispiel:
Der Reduktionsfaktor für das Multiprozessormodell beträgt

$$\frac{5^2}{C_5^2} \times \frac{2^1}{C_2^1} \times \frac{6^1}{C_6^1} = \frac{25}{15} \times \frac{2}{2} \times \frac{6}{6} = \frac{300}{180}$$

Selbst in diesem Beispiel, wo lediglich von *einem* Submodelltyp zwei Duplikate vorliegen, wird ein Reduktionsfaktor von etwa 1.7 erreicht.

Von den nach der Reduktion verbliebenen 15 Zuständen des kombinierten Zustandsraums der beiden *Processor*-Submodelle sind zwei nicht erreichbar, nämlich genau die beiden, die den Paaren $(access, access)$ und $(under\ repair, under\ repair)$ entsprechen. Diese Information ist jedoch bei der Reduktion des Zustandsraums auf Submodelltypenebene leider noch nicht bekannt und kann daher nicht genutzt werden.

Man darf jedoch angesichts der beeindruckenden Zahlen von Abb. 4.6 nicht versucht sein, zu glauben, daß mit der dargestellten Methode große Zustandsräume grundsätzlich vermieden werden können. Bei der Modellierung komplexer Systeme kann auch die Größe des reduzierten Zustandsraums, gegeben durch $\prod_{i=1}^{c} C_{s_i}^{n_i}$, beträchtlich sein. Daher bleiben effiziente numerische Methoden (siehe z.B. [Stewart *et al.*, 1993]) für die anschließende Matrixanalyse unerläßlich.

Weiterhin muß festgestellt werden, daß das bisher für das Zusammenfassen dargestellte Vorgehen für die Praxis noch verbessert werden muß. Auch auf Typenebene ist die Ausführung des Tensorprodukts bzw. der Tensorsumme und die anschließende Durchführung der Multiplikation mit einer Projektions- und einer Selektionsmatrix noch aufwendig. Diese Tatsache haben wir schon weiter oben in Abschnitt 3.4.2 festgestellt, vgl. auch Abb. 3.11. Daher wird im folgenden Abschnitt ein schneller Reduktionsalgorithmus vorgestellt, der diese beiden Matrixmultiplikationen effizient vereint.

4.6 Ein schneller Reduktionsalgorithmus

In diesem Abschnitt geht es um praktische Gesichtspunkte der Zustandsraumreduktion. Die Fragestellung lautet dabei: Wie kann das reduzierte Modell auf effiziente Weise generiert werden? Zur Beantwortung dieser Frage stellen wir einen neuen Algorithmus vor, mit dessen Hilfe die Matrizen, die den reduzierten stochastischen Prozeß eines Submodelltyps i beschreiben, auf effiziente Weise berechnet werden (vgl. [Siegle, 1994a]). Anschließend wird die Komplexität dieses Algorithmus analysiert und mit der Komplexität eines analogen Algorithmus für die Berechnung des nicht reduzierten Tensordeskriptors verglichen.

4.6.1 Beschreibung des Algorithmus

Es folgt nun die Beschreibung eines neuen Algorithmus, mit dessen Hilfe Matrizen vom Typ

$$\hat{Q}^{(i)} = U_{s_i}^{n_i} \tilde{Q}^{(i)} V_{s_i}^{n_i} = U_{s_i}^{n_i} \left(\bigodot_{j=1}^{n_i} Q^{(i)} \right) V_{s_i}^{n_i} \tag{4.14}$$

geschickt in einem einzigen Arbeitsschritt, d.h. direkt aus der mit einem Submodell assoziierten Matrix $Q^{(i)}$, berechnet werden können, ohne dabei jemals die Tensoroperation innerhalb der Klammern explizit auszuführen. Wir interpretieren dabei das Symbol $\odot$ entweder als Tensorprodukt oder als Tensorsumme, je nachdem, ob die beteiligten Matrizen synchronisierte oder unsynchronisierte Übergänge in den beteiligten Submodellen beschreiben. Die Matrizen $U_{s_i}^{n_i}$ und $V_{s_i}^{n_i}$ werden in dem neuen Algorithmus nicht als solche benötigt. Ihre Rolle wird implizit berücksichtigt.

Die folgende Abb. 4.7 enthält eine Beschreibung des allgemeinen Rahmens des Algorithmus in Pseudocode. Die rechte Hälfte der Abbildung wird für die in Abschnitt 4.6.2 dargestellte Komplexitätsanalyse benötigt.

(1)	row $= 0$	1
(2)	$(o_1, \ldots, o_{n_i}) = (0, \ldots, 0)$	$+ n_i$
(3)	**while** $((o_1, \ldots, o_{n_i}) \neq (s_i - 1, \ldots, s_i - 1))$	$+ C_{s_i}^{n_i} \times$
(4)	{col $= 0$	$\{1$
(5)	$(t_1, \ldots, t_{n_i}) = (0, \ldots, 0)$	$+ n_i$
(6)	**while** $((t_1, \ldots, t_{n_i}) \neq (s_i - 1, \ldots, s_i - 1))$	$+ C_{s_i}^{n_i} \times$
(7)	{Bestimmung von $\hat{Q}^{(i)}(row, col)$	$\{K$
(8)	increment_ordered$(t_1, \ldots, t_{n_i})$	$+ n_i$
(9)	col++	$+ 1$
(10)	}	$\}$
(11)	increment_ordered$(o_1, \ldots, o_{n_i})$	$+ n_i$
(12)	row++	$+ 1$
(13)	}	$\}$

Abbildung 4.7: Rahmen für den schnellen Reduktionsalgorithmus (Pseudocode und Anzahl elementarer Rechenoperationen)

Die Grundidee des Algorithmus ist es, nur diejenigen Zeilen der Matrix $\tilde{Q}^{(i)}$ zu berechnen, welche durch die anschließende Links-Multiplikation mit der Selektionsmatrix $U_{s_i}^{n_i}$ ausgewählt werden. Innerhalb jeder Zeile wird unmittelbar die Projektion aller symmetrischen Zustände vollzogen, die normalerweise durch die Rechts-Multiplikation mit der Projektionsmatrix $V_{s_i}^{n_i}$ erreicht wird. Um ein solches Vorgehen zu ermöglichen, merkt sich der Algorithmus für jedes Matrixelement die Tupeldarstellung der Zustandsbeschreibung des Ausgangszustands (originating state) $(o_1, \ldots, o_{n_i})$ und die des Zielzustands (target state) $(t_1, \ldots, t_{n_i})$.

Das Inkrementieren der Zustandsbeschreibung in Tupelform wird durch die Funktion increment_ordered() bewerkstelligt, welche Zustände in lexikographisch aufsteigender Reihenfolge generiert, dabei jedoch solche Zustände $(k_1, \ldots, k_{n_i})$ überspringt, für die die Bedingung $k_1 \leq \ldots \leq k_{n_i}$ nicht erfüllt ist. Es werden von der Funktion increment_ordered() also nur die repräsentativen Zustände erzeugt.

Nachfolgend wird nun die noch fehlende Zuweisungsvorschrift für das Matrixelement $\hat{Q}^{(i)}(row, col) = \hat{Q}^{(i)}((o_1, \ldots, o_{n_i}), (t_1, \ldots, t_{n_i}))$ in Zeile (7) des Algorithmus für die beiden Fälle Tensorsumme und Tensorprodukt entwickelt.

Zunächst behandeln wir den Fall, daß das in Gl. (4.14) benutzte Symbol $\odot$ die Tensorsumme $\oplus$ repräsentiert. Betrachtet man die Elemente der nicht reduzierten Matrix $\tilde{Q}^{(i)}$ für diesen Fall, so haben diese die folgende Darstellung:

$$\tilde{Q}^{(i)}((o_1, \ldots, o_{n_i}), (t_1, \ldots, t_{n_i})) = \sum_{j=1}^{n_i} Q^{(i)}(o_j, t_j) 1(\forall m \neq j : o_m = t_m) \qquad (4.15)$$

Ein solches Matrixelement entspricht der Rate eines einzelnen — also asynchronen — Zustandsübergangs in genau einem der n_i identischen Submodelle, was durch die Bedingung $1(\ldots)$ ausgedrückt wird. Der Übergang von der Matrix $\tilde{Q}^{(i)}$ zur reduzierten Matrix $\hat{Q}^{(i)}$ erfordert in jeder Zeile ein Aufsummieren über alle äquivalenten (symmetrischen) Zustände, also über alle Zustände, die Permutationen voneinander sind. Dieses Aufsummieren wird normalerweise durch die Multiplikation mit der

Projektionsmatrix $V_{s_i}^{n_i}$ erreicht, wir schreiben sie aber zunächst in folgender Form:

$$\hat{Q}^{(i)}((o_1,\ldots,o_{n_i}),(t_1,\ldots,t_{n_i})) = \sum_{(k_1,\ldots,k_{n_i})=Perm(t_1,\ldots,t_{n_i})} \sum_{j=1}^{n_i} Q^{(i)}(o_j,k_j)1(\forall m \neq j : o_m = k_m) \quad (4.16)$$

Diese Zuweisungsvorschrift könnte man nun in die Zeile (7) des Algorithmus einsetzen. Da das Aufsummieren über alle möglichen Permutationen einer Zustandsbeschreibung in der Praxis jedoch umständlich ist, geben wir noch eine zweite Schreibweise an, die wie in Abschnitt 3.4.2, Gl. (3.1) wieder die Funktion ndiff() benutzt. In diesem Fall handelt es sich um einen entsprechenden Algorithmus wie derjenige für die Matrix-Semantik des unsynchronisierten Replikationsoperators. Unter Verwendung der dort eingeführten Funktion ndiff() erhält man folgende äquivalente Darstellung zur Berechnung der Matrixelemente in Zeile (7) des Algorithmus:

$$\hat{Q}^{(i)}((o_1,\ldots,o_{n_i}),(t_1,\ldots,t_{n_i})) = \begin{cases} w_z(o)q_{zy} & \text{falls ndiff } (o,t) = 1 \\ 0 & \text{sonst} \end{cases} \quad (4.17)$$

Dabei bezeichnet $w_z(o)$ wieder das Gewicht der Ziffer z im Tupel o, und q_{zy} die Rate vom Zustand z in den Zustand y in der Matrix $Q^{(i)}$. Die Beschreibung des Zielzustands entsteht aus der Beschreibung des Ausgangszustands durch das Ersetzen genau einer Ziffer z durch die Ziffer y, wobei die Position der Ziffern keine Rolle spielt (d.h. die Tupel werden als Multimengen angesehen).

Beispiel: Asynchroner Übergang

$s_i = 3, n_i = 2$

$Q^{(i)} =$

	0	1	2
0		1	
1	2		3
2			

$\widehat{Q}^{(i)} =$

	00	01	02	11	12	22
00		2x1				
01	2		3	1		
02					1	
11		2x2			2x3	
12			2			3
22						

$\hat{Q}^{(i)}((0,0),(0,0)) = 0 \quad \text{da ndiff}((0,0),(0,0)) = 0$

$\hat{Q}^{(i)}((0,0),(0,1)) = 2 \times q_{01} = 2 \times 1 \quad \text{da ndiff}((0,0),(0,1)) = 1 \text{ und } w_0(0,0) = 2$

...

Der bisher dargestellte Fall umfaßt asynchrone Übergänge, findet in der Modellwelt also Anwendung bei den internen Ereignissen. Außerdem werden auch die sich auf ein synchronisierendes Ereignis beziehenden Matrizen so behandelt, falls *lediglich ein* Submodell des betreffenden Submodelltyps an der Synchronisation teilnimmt.

Als zweites betrachten wir den verbliebenen Fall, daß nämlich in Gl. (4.14) das Symbol $\odot$ als Tensorprodukt $\otimes$ zu interpretieren ist. Innerhalb unserer Modellwelt findet dieser Fall bei synchronisierenden Ereignissen Anwendung, wenn *alle* Submodelle des betrachteten Submodelltyps an der Synchronisation teilnehmen. Dann haben die Elemente der noch nicht reduzierten Matrix $\tilde{Q}^{(i)}$ aufgrund der Definition der Tensormultiplikation folgende Darstellung:

$$\tilde{Q}^{(i)}((o_1,\ldots,o_{n_i}),(k_1,\ldots,k_{n_i})) = \prod_{j=1}^{n_i} Q^{(i)}(o_j,k_j) \tag{4.18}$$

Die Elemente der reduzierten Matrix $\hat{Q}^{(i)}$ können wieder durch Aufsummieren über alle äquivalenten Zustände angegeben werden:

$$\hat{Q}^{(i)}((o_1,\ldots,o_{n_i}),(t_1,\ldots,t_{n_i})) = \sum_{(k_1,\ldots,k_{n_i})=Perm(t_1,\ldots,t_{n_i})} \prod_{j=1}^{n_i} Q^{(i)}(o_j,k_j) \tag{4.19}$$

Es werden also wieder die Einträge für alle symmetrischen Zustände innerhalb einer Zeile aufsummiert, was der Projektion mit Hilfe der Matrix $V_{s_i}^{n_i}$ entspricht. Da die Matrizen, die die synchronisierenden Ereignisse beschreiben, nur Null- und Eins-Einträge haben, ist das Produkt in Gleichung (4.19) besonders leicht zu realisieren. Offensichtlich ist das Produkt nur dann verschieden von Null, wenn *alle* Faktoren $Q^{(i)}(o_j,k_j)$ verschieden von Null sind, die entsprechenden Übergänge also in allen Submodellen des betreffenden Typs möglich sind. Wenn mindestens eines der Elemente $Q^{(i)}(o_j,k_j)$ gleich Null ist, kann die Multiplikation vorzeitig abgebrochen werden.

Beispiel: Synchroner Übergang

$s_i = 3, n_i = 2$

$Q^{(i)} =$

	0	1	2
0		1	
1	1		1
2			

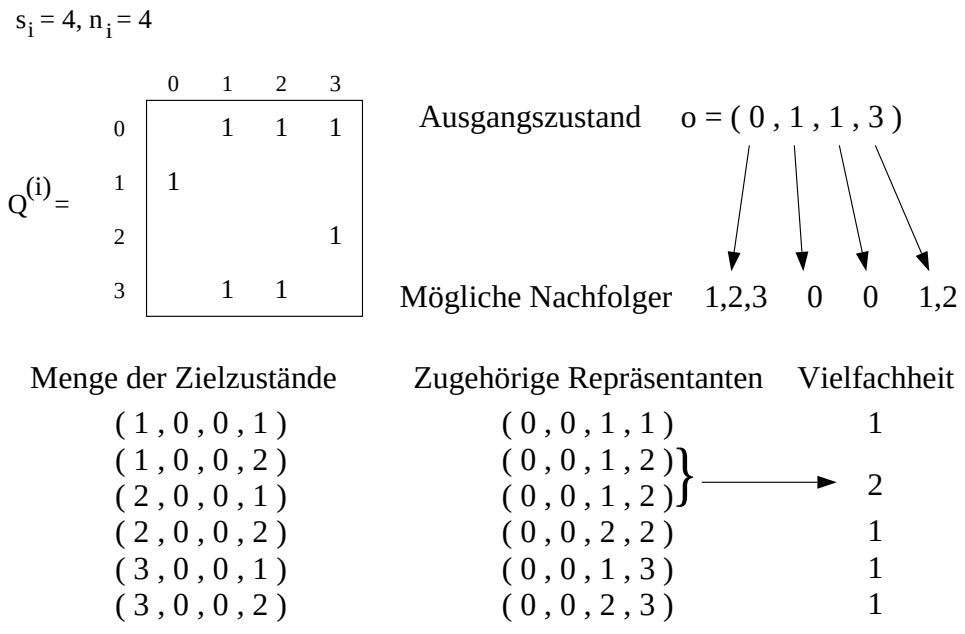
$\hat{Q}^{(i)} =$

	00	01	02	11	12	22
00				1		
01		1			1	
02						
11	1		2x1			1
12						
22						

Im Vergleich zum Beispiel für den unsynchronisierten Übergang enthält die Matrix $Q^{(i)}$ keine Raten, sondern lediglich 1-Einträge an den Stellen, wo synchronisierte Übergänge möglich sind. Auf den ersten Blick mag es vielleicht erstaunen, daß hier positive Diagonalelemente entstehen können. Wenn es um die Ermittlung von zustandsorientierten Leistungsmaßen geht, kann man diese Diagonalelemente vernachlässigen, da sie die Zustandswahrscheinlichkeiten nicht beeinflussen. Für funktionale Betrachtungen spielen sie aber durchaus eine Rolle.

Eine zweite Darstellung, die das Aufsummieren über alle Permutationen vermeidet, kann hier nicht so leicht in geschlossener Form angegeben werden wie im Fall des asynchronen Übergangs bei der Darstellung in Gl. (4.17). Das liegt daran, daß die Anzahl der Ziffern, in denen sich die Tupeldarstellungen des Ausgangs- und das Zielzustands unterscheiden, hier nicht a priori bekannt ist. Um trotzdem ohne die lästige Berechnung der Permutationen in Gl. (4.19) auszukommen, kann am Algorithmus aus Abb. 4.7 eine leichte Modifikation vorgenommen werden. Dazu berechnet man in der äußeren Schleife, etwa vor Zeile (4), die Menge aller mittels eines synchronen Übergangs aus dem Ausgangszustand erreichbaren Zielzustände samt zugehöriger Vielfachheit. In Zeile (7) findet dann keine eigentliche Berechnung mehr statt, sondern es genügt, festzustellen, ob und gegebenenfalls in welcher Vielfachheit der Zielzustand in dieser (Multi-) Menge enthalten ist. Diese Vielfachheit ergibt nämlich genau das Matrixelement $\hat{Q}(row, col)$.

Beispiel: Berechnung der Menge der Nachfolgezustände



4.6.2 Komplexitätsanalyse

Für die nun folgende Analyse der Komplexität des Algorithmus wird das Schema in der rechten Spalte von Abb. 4.7 herangezogen. Es gibt die Anzahl elementarer Rechenoperationen an, die in der jeweiligen Zeile des Algorithmus ausgeführt werden. Unter elementaren Rechenoperationen verstehen wir dabei Vergleiche, Zuweisungen, Additionen und Multiplikationen.

Für die allgemeine Darstellung des Algorithmus werden insgesamt

$$\begin{aligned} & 1 + n_i + C_{s_i}^{n_i}\left(2 + 2n_i + C_{s_i}^{n_i}(1 + n_i + K)\right) \\ =& 1 + n_i + (2 + 2n_i)C_{s_i}^{n_i} + (1 + n_i + K)\left(C_{s_i}^{n_i}\right)^2 \end{aligned}$$

Operationen benötigt, wobei für die Berechnung in Zeile (7) der für die verschiedenen Fälle noch zu bestimmende Wert K angenommen wurde.

Wir wollen zunächst exemplarisch eine Analyse für den Fall des Tensorprodukts in der Darstellung nach Gl. (4.19) vornehmen. Dafür wird das Wissen verwendet, daß für einen gegebenen Zielzustand $(t_1, \ldots, t_{n_i})$ im Durchschnitt $s_i^{n_i}/C_{s_i}^{n_i}$ Permutationen betrachtet werden müssen. Das heißt, daß die Summe im Durchschnitt $s_i^{n_i}/C_{s_i}^{n_i}$ Summanden hat, wobei für die Berechnung jedes Summanden n_i Multiplikationen notwendig sind. Für die Bewertung von Zeile (7) ergibt sich also $K = n_i \times s_i^{n_i}/C_{s_i}^{n_i}$, so daß insgesamt

$$\begin{aligned}&1 + n_i + (2 + 2n_i)C_{s_i}^{n_i} + \left(1 + n_i + n_i s_i^{n_i}/C_{s_i}^{n_i}\right)\left(C_{s_i}^{n_i}\right)^2\\ =&1 + n_i + \left(2 + 2n_i + n_i s_i^{n_i}\right)C_{s_i}^{n_i} + (1 + n_i)\left(C_{s_i}^{n_i}\right)^2\end{aligned}$$

Operationen benötigt werden. Der dominante Term in diesem Ausdruck lautet $n_i s_i^{n_i} C_{s_i}^{n_i}$, da der Wert von $s_i^{n_i}$ ja viel schneller anwächst als der von $C_{s_i}^{n_i}$.

Die nicht reduzierte Generatormatrix $\tilde{Q}^{(i)}$ kann durch einen Algorithmus berechnet werden, der dieselbe Form wie der in Abb. 4.7 dargestellte hat, jedoch statt der Funktion increment_ordered() die gewöhnliche Inkremetierungsfunktion und in Zeile (7) den Ausdruck von Gl. (4.18) verwendet. Dafür ergibt sich folgende Anzahl von elementaren Rechenoperationen

$$\begin{aligned}&1 + n_i + s_i^{n_i}\left(2 + 2n_i + s_i^{n_i}(1 + n_i + n_i)\right)\\ =&1 + n_i + (2 + 2n_i)s_i^{n_i} + (1 + n_i + n_i)s_i^{2n_i}\end{aligned}$$

mit dem dominanten Term $n_i s_i^{2n_i}$.

Ein Vergleich der dominanten Terme der beiden ermittelten Komplexitäten ergibt, daß die Konstruktion der reduzierten Generatormatrix mit Hilfe des neuen Algorithmus mindestens um den Faktor $\left(n_i s_i^{2n_i}\right)/\left(n_i s_i^{n_i} C_{s_i}^{n_i}\right) = s_i^{n_i}/C_{s_i}^{n_i}$ billiger ist als die Generierung der nicht reduzierten Generatormatrix.

Dieses Ergebnis entspricht in der Tat dem aufgrund folgender Überlegungen zu erwartenden Gewinn: In die Berechnung einer Zeile der reduzierten Matrix $\hat{Q}^{(i)}$ gehen alle in der entsprechenden Zeile stehenden Elemente der Matrix $\tilde{Q}^{(i)}$ ein. Insbesondere wird über die Permutation jedes solche Element genau einmal benötigt. Daher kann innerhalb einer Zeile kein Gewinn erwartet werden. Da jedoch statt $s_i^{n_i}$ Zeilen lediglich $C_{s_i}^{n_i}$ Zeilen berechnet werden müssen, ist tatsächlich der oben ermittelte Gewinn zu erwarten. Für eine Illustration des Gewinns für verschiedene Parameter s_i und n_i kann wieder auf Abb. 4.6 verwiesen werden.

Für den Fall der Tensorsumme legen wir der Komplexitätsanalyse die Darstellung nach Gl. (4.17) zugrunde. Die Funktion ndiff() kann mit lediglich n_i Vergleichsoperationen berechnet werden, da die zu vergleichenden Tupel bereits sortiert, d.h. mit aufsteigenden Ziffern vorliegen. Beim Vergleich fällt als Nebenprodukt auch sofort das Gewicht $w_z(o)$ ab. Der Wert von K ergibt sich in diesem Fall also zu $K = n_i$, so daß insgesamt

$$\begin{aligned}&1 + n_i + C_{s_i}^{n_i}\left(2 + 2n_i + C_{s_i}^{n_i}(1 + n_i + n_i)\right)\\ =&1 + n_i + (2 + 2n_i)C_{s_i}^{n_i} + (1 + n_i + n_i)\left(C_{s_i}^{n_i}\right)^2\end{aligned}$$

Rechenoperationen benötigt werden. Der dominante Term lautet in diesem Fall $n_i(C_{s_i}^{n_i})^2$ und liegt daher noch unter dem für die Tensormultiplikation.

4.6.3 Bewertung des Algorithmus

Die Komplexitätsanalyse belegt, daß die Berechnung der Matrizen $\hat{Q}^{(i)}$ mittels des neuen Algorithmus bedeutend billiger ist als die Berechnung der Matrizen $\tilde{Q}^{(i)}$. Dies ist ein sehr wichtiges Ergebnis. Es besagt zum einen, daß kein zusätzlicher Preis bezahlt werden muß, um für die folgende Analyse der Markovkette von der Zustandsraumreduktion zu profitieren. Im Gegenteil, schon beim Aufbau der Generatormatrix wird viel Arbeit eingespart. Zum anderen wird deutlich, daß der vorgestellte Algorithmus die Matrix $\hat{Q}^{(i)}$ viel effizienter berechnet als das mit dem naiven Ansatz $\hat{Q}^{(i)} = U_{s_i}^{n_i} \tilde{Q}^{(i)} V_{s_i}^{n_i}$ möglich wäre, also durch explizite Ausführung der Tensoroperation und anschließende Matrixmultiplikation mit den Matrizen $U_{s_i}^{n_i}$ und $V_{s_i}^{n_i}$.

Der vorgestellte algorithmische Rahmen zur Berechnung der reduzierten Matrizen auf Submodelltypenebene ist ein wichtiger Baustein für ein automatisiertes Vorgehen bei der Markovanalyse in unserer Modellwelt. Er hat den Vorteil, daß mit seiner Hilfe die Reduktion des Zustandsraums effizient durchgeführt werden kann.

5 Erweiterung der Modellwelt, Zusammenfassung und Ausblick

5.1 Erweiterung und Verallgemeinerung der Modellwelt

In diesem Abschnitt werden sinnvolle Erweiterungen und Verallgemeinerungen der Modellwelt aus Kapitel 4 angesprochen. Wie bereits in der Einleitung zu Kapitel 4 gesagt, wurde dort eine minimale Modellwelt beschrieben. Auf den nachfolgenden Seiten soll dem Leser zumindest eine intuitive Vorstellung davon gegeben werden, wie diese ausgebaut werden kann und welche zusätzlichen Möglichkeiten der Modellierung sich damit eröffnen. Die Erweiterungen werden anhand von Beispielen erläutert.

5.1.1 Verallgemeinerte Raten

Die Verzögerungszeit eines jeden Ereignisses ist in der beschriebenen Modellwelt exponentiell verteilt. Für eine eindeutige Parametrierung der Verteilung genügt daher die Angabe einer Rate. Bisher wurden nur konstante, endliche Raten betrachtet. Darüberhinaus sind aber verschiedene Erweiterungen denkbar und möglich:

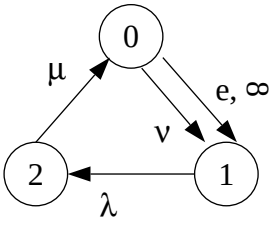
1. Als Grenzfall werden unendlich große Raten, d.h. *Raten mit dem Wert* $r = \infty$ *zugelassen*. Ereignisse mit einer solchen Rate heißen zeitlose Ereignisse. Sie haben eine Verzögerungszeit gleich Null, treten also ohne Verzögerung auf und haben daher stets Vorrang (Priorität) vor Ereignissen mit exponentiell verteilter, also streng positiver Verzögerungszeit.
2. Im Zusammenhang mit synchronisierenden Ereignissen ist es von Interesse, mit *probabilistischen Raten* zu arbeiten, um Phänomene wie z.B. Routingwahrscheinlichkeiten zu modellieren. Man muß dazu für jede Kombination aus Submodell und Ereignis ein Gewicht definieren, das festlegt, mit welcher Wahrscheinlichkeit ein Submodell an einem bestimmten synchronisierenden Ereignis teilnimmt.
3. Es ist möglich, Ereignisse mit *funktionalen Raten* zu versehen. Dies sind Raten, deren konkreter Wert vom globalen Zustand des Gesamtmodells abhängt. Die Rate, mit der ein bestimmter (lokaler oder globaler) Zustandsübergang stattfindet, kann also vom aktuellen Zustand eines oder mehrerer Submodelle abhängen. Ein solcher Ansatz wurde von Plateau verfolgt (z.B. in [Plateau *et al.*, 1988]). Er wird daher hier nicht näher erläutert.

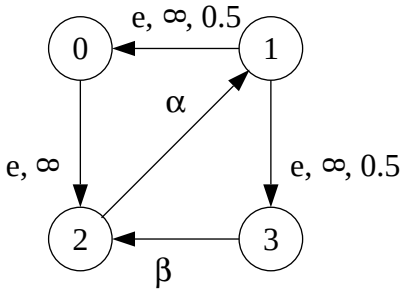
Von diesen drei möglichen Verallgemeinerungen soll lediglich der erste Punkt, die Erweiterung um zeitlose Ereignisse, in der nachfolgenden Diskussion vertieft werden. Es wird sich herausstellen, daß ein grundlegender Aspekt des zweiten Punkts, nämlich die Festlegung von Wahrscheinlichkeiten durch Gewichte, dabei auch schon eine Rolle spielt.

Zeitlose Ereignisse erleichtern die Formulierung bestimmter Synchronisations- und Wartesituationen, ohne die zugehörige Markovkette unnötig aufzublähen. Man kann bei der

Analyse von Modellen mit zeitlosen Ereignissen ein analoges Grundvorgehen wie bei der Analyse verallgemeinerter stochastischer Petrinetze (GSPN) mit zeitlosen und exponentiellen Transitionen anwenden. Auch hier erfordert das Aufstellen der zugrundeliegenden Markovkette eine Vorverarbeitung, d.h. die Elimination der verschwindenden Zustände. Das sind diejenigen Zustände, in denen mindestens ein zeitloses Ereignis stattfinden kann, vgl. Abschnitt 2.2.3. Anhand eines einfachen Beispiels soll dies nun verdeutlicht werden.

Beispiel:
Es liegen zwei Submodelle vor, die über ein synchronisierendes Ereignis e kommunizieren. Dieses synchronisierende Ereignis besitzt die Rate ∞, hat also die Verzögerungszeit Null. Unter den beiden Submodellen sind die zugehörigen Matrizen sowohl für die lokalen Ereignisse als auch für das synchronisierende Ereignis e angegeben. Zur Hervorhebung sind alle Matrixeinträge, die sich auf das synchronisierende Ereignis e beziehen, mit einem Kästchen versehen. Einträge auf der Hauptdiagonalen — die negativen Zeilensummen — sind der Einfachheit halber nicht berücksichtigt.





$$Q_l^{(1)} = \begin{array}{c|ccc} & 0 & 1 & 2 \\ 0 & & \boxed{\nu} & \\ 1 & & & \boxed{\lambda} \\ 2 & \boxed{\mu} & & \end{array}$$

$$Q_l^{(2)} = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ 0 & & & & \\ 1 & & & & \\ 2 & & & \boxed{\alpha} & \\ 3 & & & & \boxed{\beta} \end{array}$$

$$Q_e^{(1)} = \begin{array}{c|ccc} & 0 & 1 & 2 \\ 0 & & \boxed{1} & \\ 1 & & & \\ 2 & & & \end{array}$$

$$Q_e^{(2)} = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ 0 & & & \boxed{1} & \\ 1 & \boxed{0.5} & & & \boxed{0.5} \\ 2 & & & & \\ 3 & & & & \end{array}$$

Im linken Submodell ist ein Übergang von Zustand 0 nach Zustand 1 auf zwei Arten möglich: Entweder als internes Ereignis mit der Rate ν oder als synchronisierendes Ereignis e mit der Rate ∞.

Im rechts dargestellten Submodell ist aus dem Zustand 0 mittels des synchronisierenden Ereignisses e ein Übergang in den Zustand 2 möglich. Zustand 1 hat zwei mögliche Folgezustände: Sowohl Zustand 0 als auch Zustand 3 sind über das synchronisierende Ereignis e erreichbar. Die Wahrscheinlichkeit beträgt dabei in beiden Fällen 0.5.

Anschließend ist die Matrix $Q_l + Q_e$ des Gesamtmodells dargestellt, die sich aus den Submodellmatrizen wie folgt zusammensetzt:

$$Q_l + Q_e = Q_l^{(1)} \oplus Q_l^{(2)} + Q_e^{(1)} \otimes Q_e^{(2)}$$

Es handelt sich bei dieser Matrix nicht um eine Generatormatrix, da sie neben Übergangsraten auch Übergangswahrscheinlichkeiten enthält. In der Matrix $Q_l + Q_e$ bezeichnen 00 und 01 verschwindende Zustände, da in ihnen zeitlose Zustandsübergänge möglich sind. Die beiden zeitbehafteten Zustandsübergänge 00→10 und 01→11, beide mit Rate ν, können aufgrund der Priorität des zeitlosen Ereignisses e niemals stattfinden.

$Q_l + Q_e =$

	00	01	02	03	10	11	12	13	20	21	22	23
00					ν		1					
01					0.5	ν		0.5				
02		α					ν					
03			β					ν				
10									λ			
11										λ		
12						α					λ	
13							β					λ
20	μ											
21		μ										
22			μ							α		
23				μ							β	

Um die beiden verschwindenden Zustände 00 und 01 zu eliminieren, sind alle Zustandsübergänge, die den Zustand 00 oder 01 als Zielzustand haben, geeignet "umzulenken". So ist z.B. der Übergang 20→00 in 20→12 abzuändern. Der Übergang 21→01 führt zu den beiden neuen Übergängen 21→10 und 21→13, wobei die Rate μ gemäß der Wahrscheinlichkeiten aufgespalten wird. Dieses Vorgehen führt schließlich zu der folgenden Matrix Q.

$$Q = \begin{array}{c|cccccccccc} & 02 & 03 & 10 & 11 & 12 & 13 & 20 & 21 & 22 & 23 \\ \hline 02 & & & 0.5\alpha & & \nu & 0.5\alpha & & & & \\ 03 & \beta & & & & & \nu & & & & \\ \hline 10 & & & & & & & \lambda & & & \\ 11 & & & & & & & & \lambda & & \\ 12 & & & & \alpha & & & & & \lambda & \\ 13 & & & & & \beta & & & & & \lambda \\ \hline 20 & & & & & \mu & & & & & \\ 21 & & & 0.5\mu & & & 0.5\mu & & & & \\ 22 & \mu & & & & & & & \alpha & & \\ 23 & & \mu & & & & & & & \beta & \end{array}$$

Bei dieser Matrix handelt es sich nun wieder um eine ganz normale Generatormatrix einer zeitkontinuierlichen Markovkette, die das Verhalten der Kombination der beiden Submodelle beschreibt.

Es sei noch darauf hingewiesen, daß es durchaus sinnvoll ist, einen “doppelten” Zustandsübergang wie den von Zustand 0 nach Zustand 1 im linken Submodell zu spezifizieren, da sich erst auf der Ebene des Produktzustandsraums entscheidet, ob eine Zustandskombination 0x (hierbei bezeichen “x” den Zustand des rechten Submodells) einen verschwindenden oder einen stabilen Zustand bezeichnet.

Im allgemeinen können sowohl interne als auch synchronisierende Ereignisse mit einer unendlichen Rate versehen werden. Der erste Fall ist verhältnismäßig einfach zu handhaben, da die zeitlosen Zustandsübergänge bereits vor dem Zusammenschalten der Submodelle eliminiert werden können. Bei synchronisierenden Ereignissen mit unendlicher Rate können potentiell entstehende verschwindende Zustände dagegen erst nach dem Aufbau des Produktzustandsraums identifiziert und eliminiert werden. In solchen Fällen, in denen mehrere zeitlose Zustandsübergänge in Konflikt stehen, müssen die Übergangswahrscheinlichkeiten mit Hilfe von Gewichten festgelegt werden, was bei synchronisierenden Ereignissen recht komplex werden kann.

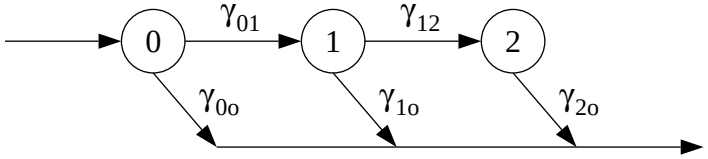
Die Verwendung zeitloser Ereignisse ist auch mit der Zustandsraumreduktion auf Ebene der Submodelltypen verträglich. Man kann z.B. ohne weiteres obiges Beispiel um ein zusätzliches Submodell des ersten Typs erweitern, dann zunächst eine Zustandsraumreduktion für die beiden symmetrischen Submodelle durchführen, um anschließend die Verknüpfung mit dem Submodell des zweiten Typs durchzuführen und die verschwindenden Zustände zu eliminieren. Damit kann man zusammenfassend feststellen, daß sich zeitlose Ereignisse gut in die Modellwelt integrieren lassen.

5.1.2 Verallgemeinerte Verteilungen

Markovanalyse setzt voraus, daß die Verweilzeit in jedem Zustand exponentiell verteilt ist. Da reale Laufzeiten diese Voraussetzung oft nicht erfüllen, stellt dies eine Einschränkung für den praktischen Einsatz von Markovmodellen dar. Bei den Anwendern besteht häufig der Wunsch, Laufzeiten mit anderen Verteilungen, im Grenzfall auch deterministische Laufzeiten zu verwenden. Eine Möglichkeit, dies zu erreichen ohne den Kontext der Markovmodelle zu verlassen, ist die Verwendung von aus exponentiellen Phasen zusammengesetzten Verteilungen, kurz Phasenverteilungen genannt. Mit Phasenverteilungen lassen sich allgemeine Verteilungen beliebig genau approximieren. In [Bux und Herzog, 1977] wurden sie beispielsweise für die Approximation gemessener Verteilungen benutzt. Eine Methode zur Schätzung der Parameter einer Phasenverteilung nach der Maximum-Likelihood Methode auf der Basis von Meßdaten wird in [Bobbio und Cumani, 1991] beschrieben. Auch in [Schmickler, 1992] geht es um die Approximation empirischer Verteilungsfunktionen, und zwar mit Hilfe gemischter Erlangverteilungen. Der mit der Verwendung von Phasenverteilungen verbundene Nachteil besteht jedoch in einem zusätzlichen Anwachsen des Zustandsraums wegen der unter Umständen großen Zahl von benötigten Phasen.

Eine Phasenverteilung wird beschrieben durch eine quadratische Ratenmatrix T und einen Wahrscheinlichkeitsvektor α derselben Dimension [Neuts, 1989]. Die Matrix T hat negative Diagonalelemente und nichtnegative Elemente außerhalb der Hauptdiagonalen. Mit ihr ist zugleich ein nicht negativer Vektor $T^o = -Te$ (Vektor der negativen Zeilensummen) der Austrittsraten aus der Phasenverteilung bestimmt. Der Vektor α legt die Wahrscheinlichkeiten dafür fest, daß die Bearbeitung in einer bestimmten Phase begonnen wird.

Beispiel:
Ein typisches Beispiel für eine Phasenverteilung ist die Cox-Verteilung. Die Bearbeitung beginnt in der ersten Phase. Aus jeder Phase ist neben dem Übergang in die Folgephase (mit Rate $\gamma_{i,i+1}$) auch die Beendigung der Bearbeitung (mit Rate $\gamma_{i,o}$) möglich. Im folgenden ist eine 3–phasige Cox-Verteilung dargestellt.



$$\alpha = [1,0,0] \qquad T = \begin{bmatrix} -(\gamma_{01}+\gamma_{0o}) & \gamma_{01} & \\ & -(\gamma_{12}+\gamma_{1o}) & \gamma_{12} \\ & & -\gamma_{2o} \end{bmatrix} \qquad \left(T^o = \begin{bmatrix} \gamma_{0o} \\ \gamma_{1o} \\ \gamma_{2o} \end{bmatrix} \right)$$

Die Verteilungsfunktion einer durch (α, T) gegebenen Phasenverteilung lautet in geschlossener Form

$$F(t) = 1 - \alpha \exp(Tt)e \qquad t \geq 0$$

Dabei ist der Matrixexponent $\exp(Tt)$ gleich der unendlichen Summe $\sum_{i=0}^{\infty} (Tt)^i/i!$ und e bezeichnet einen mit lauter Einsen besetzten Spaltenvektor.

Es stellt sich nun die Frage, wie sich Phasenverteilungen geschickt in die Modellwelt aus Kapitel 4 integrieren lassen.

Eine naheliegende Möglichkeit bestünde darin, die Phasenverteilung als eigenständiges Hilfs-Submodell zu spezifizieren und über ein Eintritts- und ein Austrittsereignis mit dem eigentlichen Modell, in dem eine Phasenverteilung benutzt werden soll, zu synchronisieren. Dieser einfache Ansatz hat aber den schwerwiegenden Nachteil, daß viele unerreichbare Zustände entstehen, weil im Produktzustandsraum jeder Zustand des Modells mit jeder Phase kombiniert auftritt. Außerdem wäre diese Vorgehensweise problematisch bei Phasenverteilungen mit mehreren Ein- und Austrittspunkten sowie bei Spezialfällen, in denen möglicherweise nur eine Phase durchlaufen wird, wo also Ein- und Austritt zusammenfallen können (wie es z.B. bei Cox-Verteilungen der Fall ist).

Ein anderes Verfahren, bei dem diese Probleme vermieden werden, besteht in der direkten Ersetzung eines Zustands mit exponentieller Verweilzeit durch mehrere, den einzelnen Phasen einer Phasenverteilung entsprechenden Unterzustände. Auf Matrixebene bedeutet dies eine Modifikation einer vorliegenden Matrix Q in eine Matrix Q' gemäß des folgenden Vorgehens (es wird angenommen, daß die Verweilzeit im i-ten Zustand durch eine Phasenverteilung ersetzt wird):

1.a) Das i-te Diagonalelement der Matrix Q wird durch die Matrix T der Phasenverteilung ersetzt.

$$Q'_{ii} = T$$

2. Um die Eintrittswahrscheinlichkeiten in die Phasenverteilung zu realisieren, müssen die Zustandsübergänge in den i-ten Zustand hinein mit dem Vektor α multipliziert werden.

$$Q'_{ji} = q_{ji}\alpha \qquad j \neq i$$

3.a) Die Austrittsraten aus der Phasenverteilung (Vektor T^o) werden entsprechend der ursprünglichen Zustandsübergangswahrscheinlichkeiten aus dem i-ten Zustand heraus gewichtet.

$$Q'_{ij} = p_{ij}T^o \qquad j \neq i$$

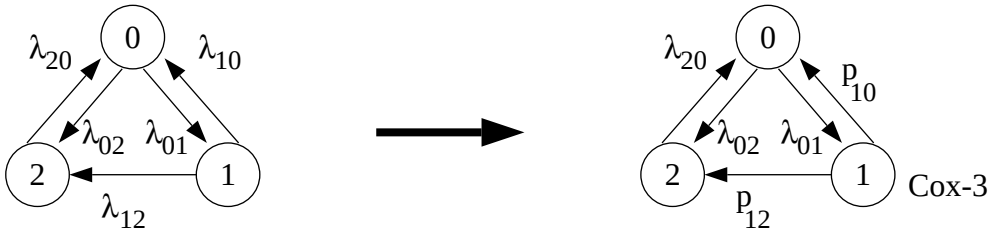
Die Zustandsübergangswahrscheinlichkeiten sind dabei gegeben durch

$$p_{ij} := q_{ij}\left(\sum_{j \neq i} q_{ij}\right)^{-1} \qquad j \neq i$$

4. Alle anderen Zustandsübergänge bleiben unverändert.

$$q'_{jk} = q_{jk} \qquad j, k \neq i$$

Beispiel:
Die exponentielle Verweilzeit in Zustand 1 des linken Modells, deren Erwartungswert $1/(\lambda_{10} + \lambda_{12})$ beträgt, wird durch die dreiphasige Cox-Verteilung aus dem letzten Beispiel ersetzt.



$$Q = \begin{array}{c|ccc} & 0 & 1 & 2 \\ 0 & -\Sigma & \lambda_{01} & \lambda_{02} \\ 1 & \lambda_{10} & -\Sigma & \lambda_{12} \\ 2 & \lambda_{20} & & -\Sigma \end{array}$$

$$Q' = \begin{array}{c|ccccc} & 0 & 10 & 11 & 12 & 2 \\ 0 & -\Sigma & \lambda_{01} & & & \lambda_{02} \\ 10 & p_{10}\gamma_{0o} & -\Sigma & \gamma_{01} & & p_{12}\gamma_{0o} \\ 11 & p_{10}\gamma_{1o} & & -\Sigma & \gamma_{12} & p_{12}\gamma_{1o} \\ 12 & p_{10}\gamma_{2o} & & & -\Sigma & p_{12}\gamma_{2o} \\ 2 & \lambda_{20} & & & & -\Sigma \end{array}$$

$$p_{10} = \lambda_{10} / (\lambda_{10} + \lambda_{12}) \qquad p_{12} = \lambda_{12} / (\lambda_{10} + \lambda_{12})$$

Dieses Verfahren der Ersetzung eines Zustands durch mehrere, den Phasen entsprechenden Unterzustände erlaubt noch folgende differenziertere Variante: Statt der Verweilzeit in einem Zustand i soll nur ein bestimmter Zustandsübergang aus diesem Zustand heraus, z.B. der von Zustand i nach Zustand k mit phasenverteilter Verzögerungszeit erfolgen. Andere mögliche Zustandsübergänge aus dem Zustand i heraus sollen jedoch nach wie vor mit exponentieller Verzögerungszeit stattfinden. Mit dieser Variante wird also eine größere Flexibilität bei der Modellspezifikation erreicht: Anstatt grundsätzlich alle Ereignisse, die in einem bestimmten Zustand möglich sind, mit einer phasenverteilten Verzögerungszeit zu versehen, kann dies bei Bedarf nur für ein bestimmtes Ereignis geschehen. Die Berechnungsvorschriften in den Punkten 2. und 4. der voranstehenden Liste bleiben davon unberührt. Lediglich die erste und dritte Berechnungsvorschrift sind wie folgt zu modifizieren:

1.b) Vor der Ersetzung des i-ten Diagonalelements der Matrix Q durch die Matrix T ist von allen Diagonalelementen der Matrix T der Wert $\sum_{j \neq i,k} q_{ij}$ zu subtrahieren.

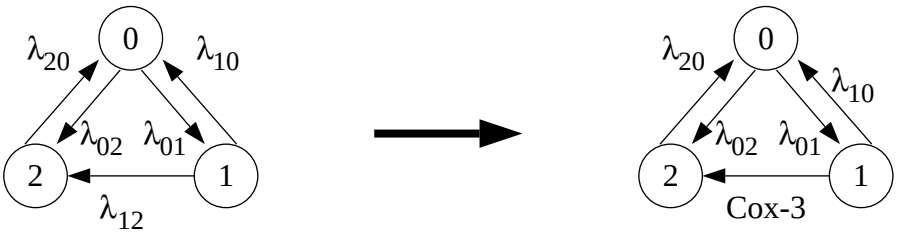
Damit werden auf der Hauptdiagonalen der resultierenden Matrix Q' auch die Zustandsübergänge mit exponentieller Verzögerungszeit berücksichtigt.

3.b) Die Austrittsraten aus der Phasenverteilung in den Zielzustand k sind durch den Vektor T^0 bestimmt. Aus allen Phasen gelten außerdem die exponentiellen Austrittsraten in andere mögliche Zielzustände.

$$Q'_{ik} = T^o$$
$$Q'_{ij} = q_{ij}e \qquad j \neq i, k$$

Bei dieser Variante sind in jeder Phase der Phasenverteilung die parallel laufenden Exponentialverteilungen unverändert gültig, d.h. die Phasenverteilung steht im Wettbewerb mit einer oder mehreren Exponentialverteilungen (race policy, vgl. S. 21).

Beispiel:
Im Gegensatz zum vorigen Beispiel wird nun lediglich die Verzögerungszeit des Übergangs von Zustand 1 nach Zustand 2 des linken Modells durch die dreiphasige Cox-Verteilung ersetzt. Der Verzögerungszeit des Übergangs von Zustand 1 nach Zustand 0 bleibt unverändert exponentiell verteilt mit Rate λ_{10}.



Q =

	0	1	2
0	-Σ	λ_{01}	λ_{02}
1	λ_{10}	-Σ	λ_{12}
2	λ_{20}		-Σ

Q' =

	0	10	11	12	2
0	-Σ	λ_{01}			λ_{02}
10	λ_{10}	-Σ	γ_{01}		γ_{0o}
11	λ_{10}		-Σ	γ_{12}	γ_{1o}
12	λ_{10}			-Σ	γ_{2o}
2	λ_{20}				-Σ

Die Vorteile bei beiden Varianten dieser Ersetzungsstrategie bestehen darin, daß keine unerreichbaren Zustände entstehen. Die Dimension der resultierenden Matrix Q' ist genau um die Anzahl der Phasen (minus eins) größer als die Ausgangsmatrix Q, so daß also die Größe der entstehenden Matrix minimal gehalten wird.

Bisher haben wir nur die Ersetzung der Verzögerungszeit lokaler Ereignisse durch eine Phasenverteilung diskutiert. Im Rahmen unserer Modellwelt ist es darüberhinaus wichtig, zu untersuchen, wie sich Submodelle über Ereignisse mit phasenverteilter Verzögerungszeit synchronisieren lassen. Dabei können die beiden folgenden Ansätze verfolgt werden:

1. Zunächst wird mit exponentiell verteilten Verzögerungszeiten gearbeitet. Synchronisierende Ereignisse werden daher so behandelt wie in Kapitel 4 beschrieben. Anschließend wird für diejenigen Submodelle, die über Ereignisse mit phasenverteilter Verzögerungszeit synchronisiert werden sollen, der Produktzustandsraum explizit aufgebaut. Erst danach werden die exponentiellen Verteilungen der betreffenden Ereignisse durch Phasenverteilungen ersetzt. Dabei kann man genauso vorgehen wie oben für lokale Ereignisse beschrieben. Der Nachteil dieses Ansatzes besteht darin, daß der Produktzustandsraum der über Ereignisse mit phasenverteilter Verzögerungszeit synchronisierten Submodelle explizit aufgebaut werden muß, was teilweise zum Verlust der Vorteile des Tensordeskriptor-Ansatzes für das Gesamtmodell führt.
2. Die Ersetzung der exponentiellen Verzögerungszeit durch eine phasenverteilte wird in *genau einem* der an der Synchronisation beteiligten Submodelle vorgenommen. Die Synchronisation mit den übrigen an dem synchronisierenden Ereignis teilnehmenden Submodellen geschieht über eine Synchronisation der letzten Phase der Phasenverteilung, d.h. über ein Ereignis mit exponentieller Verzögerungszeit. Mit diesem Ansatz werden die Vorteile des Tensordeskriptors nicht geschmälert. Er ist jedoch weniger flexibel was die möglichen Synchronisationsmuster, d.h. die an der Synchronisation teilnehmenden Submodelle betrifft, da ein Submodell — dasjenige, in dem die Ersetzung vorgenommen wird — als Teilnehmer a priori feststehen muß. Sollen mehrere gleichartige Submodelle synchronisiert werden, so wird bei diesem Ansatz die vollständige Symmetrie durchbrochen, da in eines der Submodelle die Phasenverteilung eingebaut werden muß.

Zum Schluß dieses Abschnitts soll noch eine Bemerkung zum Anwachsen des Zustandsraums bei der Verwendung von Phasenverteilungen erfolgen. Aufgrund des vergrößerten Zustandsraums eines einzelnen Submodells wächst natürlich auch der Zustandsraum des Gesamtmodells. Die Zustandsraumreduktion, die anwendbar ist, wenn mehrere Submodelle eines Typs vorliegen, wirkt sich aber auch günstig auf die Zahl der durch die Phasenverteilung hinzukommenden Unterzustände aus. Kommen in einem Submodell mit s Zuständen durch die Verwendung einer Phasenverteilung m Zustände hinzu, so steigt die Zustandszahl des Gesamtmodells bei n Submodellen nicht um den erwarteten Faktor $((s+m)/s)^n$. Man kann den Anstiegsfaktor folgendermaßen abschätzen

$$
\begin{aligned}
&\frac{\binom{n+s+m-1}{s+m-1}}{\binom{n+s-1}{s-1}} \\
=&\frac{(n+s+m-1)!\,(s-1)!\,n!}{(s+m-1)!\,n!\,(n+s-1)!} \\
=&\frac{(n+s+m-1)(n+s+m-2)\ \ldots\ (s+m+1)(s+m)}{(n+s-1)\qquad(n+s-2)\quad\ldots\quad\qquad(s+1)\qquad s} \\
<&\left(\frac{s+m}{s}\right)^n
\end{aligned}
$$

Damit ist gezeigt, daß der Anstiegsfaktor echt kleiner als $((s+m)/s)^n$ ist.

5.1.3 Weitere Synchronisationsmuster

Bisher stehen in der vorgestellten Modellwelt zwei Synchronisationsmuster zur Verfügung: Enweder nehmen *alle* Submodelle eines Typs an einem synchronisierenden Ereignis teil (wir assoziieren diesen Fall mit dem Operator $\otimes$), oder *genau ein* Submodell eines Typs nimmt an der Synchronisation teil (Operator $\oplus$).

Es stellt sich nun die Frage, ob und gegebenenfalls wie das Schema von Abb 4.2, S. 77 erweitert werden kann. Es gibt beispielsweise Fälle, in denen eine bestimmte Anzahl k (mit $1 < k < n_i$) zufällig ausgewählter Submodelle eines Typs an einem synchronisierenden Ereignis teilnehmen sollen. Die Darstellung dieses Sachverhalts mit der Methode des Tensordeskriptors in geschlossener Form ist zwar möglich aber umständlich. Möchte man z.B. erreichen, daß genau zwei Submodelle des Typs i an der Synchronisation teilnehmen, so erhält man einen Ausdruck der Form

$$\begin{aligned}\tilde{Q}_e^{(i)} = & Q_e^{(i)} \otimes Q_e^{(i)} \otimes I_{s_i} \otimes \ldots \otimes I_{s_i} \\ & + Q_e^{(i)} \otimes I_{s_i} \otimes Q_e^{(i)} \otimes I_{s_i} \otimes \ldots \otimes I_{s_i} \\ & + \ldots \\ & + I_{s_i} \otimes \ldots \otimes I_{s_i} \otimes Q_e^{(i)} \otimes Q_e^{(i)}\end{aligned}$$

Für andere Fälle, wie z.B. das Prädikat “es sollen mindestens k Submodelle des i-ten Typs an der Synchronisation teilnehmen”, werden die geschlossenen Formeln sogar noch aufwendiger.

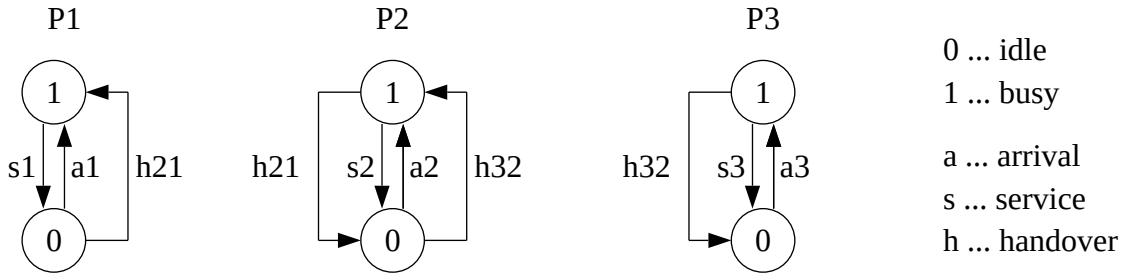
Trotz dieser Komplexität der geschlossenen Darstellung ist erfreulicherweise festzustellen, daß der Reduktionsalgorithmus mit relativ wenig Aufwand so erweitert werden kann, daß die Teilnahme einer spezifizierten Anzahl von Submodellen eines Typs an einem synchronisierenden Ereignis beherrscht wird. Dazu ist der Algorithmus so zu modifizieren, daß für jeden Ausgangszustand unter Berücksichtigung des vorgegebenen Prädikats eine Berechnung der mittels des synchronisierten Ereignisses erreichbaren Zielzustände durchgeführt wird.

5.1.4 Lockerung der Symmetriebedingungen

In Anhang A.2 sind die drei Arten von Zusammenfaßbarkeit, nämlich gewöhnliche, exakte und strikte Zusammenfaßbarkeit, erläutert. Im Kontext unserer Modellwelt wurden bisher lediglich Beispiele für *strikte Zusammenfaßbarkeit* angegeben: Wir stellten in Kapitel 4 fest, daß die Reduktion des Zustandsraums auf Ebene der Submodelltypen aufgrund des Vorliegens strikter Zusammenfaßbarkeit zulässig ist. Eine interessante Fragestellung lautet nun: Wie könnten die strengen Symmetriebedingungen der Modellwelt abgeschwächt werden, damit gewöhnliche (ohne exakte) oder exakte (ohne gewöhnliche) Zusammenfaßbarkeit vorliegt?

Wir betrachten zunächst ein Beispiel, in dem ausschließlich *gewöhnliche Zusammenfaßbarkeit* vorliegt.

Beispiel:
Drei Prozessoren, die durch die Submodelle P1, P2 und P3 repräsentiert werden, bearbeiten ankommende Aufträge. Die Prozessoren befinden sich entweder im Zustand 0 (idle) oder im Zustand 1 (busy). Die Prozessoren P2 und P3 versuchen nach Möglichkeit, einen Auftrag, den sie erhalten haben, an den links von ihnen liegenden Prozessor weiterzureichen. Dies geschieht über ein synchronisierendes Ereignis, das "handover" genannt wird.



Im folgenden ist die Matrix Q des Gesamtmodells dargestellt, wobei der Einfachheit halber für die Raten dieselben Bezeichner wie für die zugehörigen Ereignisse gewählt wurden. Außerdem wurde von der Annahme ausgegangen, daß die Ankunfts- und Bedienraten bei allen drei Prozessoren identisch sind, daß also gilt $a1 = a2 = a3 = a$ und $s1 = s2 = s3 = s$.

Die Matrix Q besitzt nicht die Eigenschaft der exakten Zusammenfaßbarkeit (bezüglich der eingezeichneten Partition), da innerhalb der Diagonalblöcke das Teilspaltenkriterium nicht erfüllt ist. Dagegen ist das Teilzeilenkriterium erfüllt, sogar unabhängig davon, ob die Raten $h21$ und $h32$ identisch sind oder nicht. Es liegt also gewöhnliche Zusammenfaßbarkeit ohne exakte Zusammenfaßbarkeit vor.

Q =

	000	001	010	100	011	101	110	111
000	Σ	a	a	a				
001	s	Σ	h32		a	a		
010	s		Σ	h21	a		a	
100	s			Σ		a	a	
011		s	s		Σ	h21		a
101		s		s		Σ	h32	a
110			s	s			Σ	a
111					s	s	s	Σ

$\widehat{Q}$ =

	000	001	011	111
000	Σ	3a		
001	s	Σ	2a	
011		2s	Σ	a
111			3s	Σ

Die rechts dargestellte Matrix $\hat{Q}$ erhält man durch Zusammenfassen der Zustände zu Makrozuständen, wozu man den Reduktionsalgorithmus aus Kapitel 4 verwenden kann. Beim Zusammenfassen geht erwartungsgemäß die Information über das "handover" verloren, d.h. das Zusammenfassen kommt einem Ausblenden der "handover"-Ereignisse gleich.

Offensichtlich ist die beobachtete Asymmetrie zwischen den drei Prozessoren im Beispiel verantwortlich für das Fehlen exakter Zusammenfaßbarkeit. Die Prozessoren verhalten sich dagegen gegenüber der Umwelt identisch (arrival und service), daher die festgestellte gewöhnliche Zusammenfaßbarkeit. Allgemein wird man feststellen, daß gewöhnliche Zusammenfaßbarkeit ohne exakte Zusammenfaßbarkeit bei asymmetrischem Verhalten innerhalb eines Submodelltyps, jedoch bei identischem Verhalten gegenüber den Submodellen anderer Typen (der Umwelt) vorliegt.

Es ist uns bisher nicht gelungen, einfache Beispiele für *exakte Zusammenfaßbarkeit* (ohne gleichzeitige gewöhnliche Zusammenfaßbarkeit) zu finden, die im Rahmen unserer Modellwelt sinnvoll sind.

5.2 Zusammenfassung und Ausblick

In dieser Arbeit wurden Methoden zur Beschreibung und Analyse von Markovmodellen mit großem Zustandsraum untersucht. Wir wollen zum Abschluß noch einmal kurz die inhaltlichen Schwerpunkte der Arbeit rekapitulieren und die wichtigsten Ergebnisse zusammenfassen.

Kapitel 2 enthält eine vergleichende Untersuchung verschiedener Modellbeschreibungsmethoden, die zunächst anhand eines gemeinsamen Beispielproblems eingeführt und anschließend bezüglich verschiedener Kriterien verglichen werden. Bei diesem Vergleich stellten wir fest, daß die untersuchten Modellbeschreibungsmethoden wesentliche Unterschiede aufweisen, die Wahl der geeigneten Modellbeschreibungsmethode also entscheidenden Einfluß auf den Erfolg eines Leistungsbewertungsprojekts nehmen kann.

In Kapitel 3 wurden unterschiedliche Ansätze für die effiziente Modellierung großer Systeme untersucht. Bei den Modelltransformationen mit dem Ziel der Modellvereinfachung wurde insbesondere die Vereinfachung verallgemeinerter stochastischer Petrinetze näher untersucht. Anschließend wurden Methoden der strukturierten Modellbeschreibung und -analyse betrachtet. Im Rahmen eines geschickten Übersetzens von der Modellbeschreibung auf höherer Ebene in die darunterliegende Markovkette stand eine Matrixsemantik für stochastische Prozeßalgebren im Vordergrund.

Die mehr analytischen Arbeiten der Kapitel 2 und 3 münden in Kapitel 4 in die Beschreibung einer neuartigen Modellwelt. Diese zeichnet sich durch eine strukturierte Beschreibung des Gesamtmodells aus, welches aus interagierenden Submodellen aufgebaut wird. Eine Besonderheit dieser Modellwelt ist ihre Eignung zur Beschreibung von Systemen mit replizierten Komponenten durch replizierte Submodelle. Dies führt dazu, daß mit Hilfe eines schnellen Reduktionsalgorithmus der Zustandsraum des Gesamtmodells unter Umständen dramatisch reduziert werden kann. Für die Generatormatrix des Gesamtmodells wird eine Darstellung in Form eines Tensordeskriptors angegeben, auf deren Basis eine effiziente verteilte numerische Analyse durchgeführt wird.

Aus Sicht der vorliegenden Arbeit ist für die erfolgreiche Durchführung einer Modellierungsstudie zunächst die Auswahl einer geeigneten Modellbeschreibungsmethode wichtig. Ab einer gewissen Größe und Kompliziertheit des zu modellierenden Systems sollte unbedingt eine strukturierte Modellbeschreibung gewählt werden. Anschließend wird man versuchen, im Rahmen einer Voranalyse Teilmodelle mit Hilfe von Transformatio-

nen soweit wie möglich zu vereinfachen. Danach kann bei Modellen mit replizierten Submodellen eine Zustandsraumreduktion auf Ebene der Submodelltypen durchgeführt werden, woran sich die strukturierte numerische Analyse auf der Basis des Tensordeskriptors anschließt.

Durch eine solche Kombination von strukturierter Modellbeschreibung, Modelltransformation, Ausnutzung von Symmetrien und strukturierter Analyse kommt man zu einem Gesamtvorgehen, mit dem man ein möglicherweise auftretendes Zustandsraumproblem bestmöglich in den Griff bekommt.

Trotz der positiven Ergebnisse der Arbeit gibt es natürlich nach wie vor eine Reihe schwieriger Probleme, von denen exemplarisch das der unerreichbaren Zustände im Zusammenhang mit dem Ansatz des Tensordeskriptors noch einmal hervorgehoben sei. Diese und die in diesem Kapitel nur angerissenen möglichen Erweiterungen der Modellwelt bilden Ausgangspunkte für aufbauende Arbeiten.

Anhang A Markovprozesse

A.1 Mathematische Grundlagen

In diesem Abschnitt wird eine kurze Einführung in die Thematik der stochastischen Prozesse gegeben, wobei insbesondere Markovprozesse im Vordergrund stehen. Dies geschieht, um aus der Fülle der Literatur mit nicht immer einheitlicher Nomenklatur den in dieser Arbeit benötigten begrifflichen Apparat bereitzustellen. Ausführlichere Abhandlungen zu diesem Thema sind unter anderem in [Çinlar, 1975; Kleinrock, 1975; Kemeny und Snell, 1976; Trivedi, 1982; Stewart, 1989] enthalten.

Der Begriff des stochastischen Prozesses ist für die Leistungsbewertung im allgemeinen und für die Markovmodellierung im besonderen von zentraler Bedeutung. Ein *stochastischer Prozeß* ist definiert als eine Familie von Zufallsvariablen.

$$\{X(t) \mid t \in T\}$$

Die Menge T wird als Index- oder Parametermenge bezeichnet. Normalerweise ist mit dem Parameter t eine Zeitvorstellung verbunden, so daß $X(t)$ den Zustand eines Systems zum Zeitpunkt t beschreibt.

Anhand des Wertebereichs der Zufallsvariablen $X(t)$ lassen sich stochastische Prozesse in zwei Gruppen einteilen. Zum einen gibt es stochastische Prozesse, die Werte aus einem kontinuierlichen Wertebereich annehmen können. Uns interessieren im Rahmen dieser Arbeit jedoch stochastische Prozesse mit diskretem (endlichem oder abzählbar unendlichem) Zustandsraum. Ein diskreter Wertebereich kann bijektiv auf eine Teilmenge der natürlichen Zahlen abgebildet werden, so daß im folgenden ohne Beschränkung der Allgemeinheit angenommen wird, daß gilt

$$X(t) \in \{0, 1, 2, \ldots\}$$

Stochastische Prozesse mit diskretem Zustandsraum werden auch als *Ketten* bezeichnet. Anhand der Indexmenge T kann eine weitere, analoge Einteilung stochastischer Prozesse vorgenommen werden. Wird der stochastische Prozeß nur zu diskreten Zeitpunkten (Zeitschritten) t_n, $n \in \{0, 1, 2, \ldots\}$ betrachtet, so spricht man von einem *zeitdiskreten stochastischen Prozeß* und verwendet häufig die Bezeichnung $X(t_n) = X_n$. Ist dagegen $X(t)$ für jeden Wert $t \in \mathbb{R}$ bzw. $t \in \mathbb{R}_0^+$ definiert, so handelt es sich um einen *zeitkontinuierlichen stochastischen Prozeß*.

Für eine weitere Klassifikation stochastischer Prozesse ist die stochastische Abhängigkeit zwischen den Zufallsvariablen $X(t)$ für verschiedene Werte des Parameters t entscheidend. Je nach der Art der Abhängigkeit erhält man Prozesse aus der Klasse der Markovprozesse, Birth-Death Prozesse, Random Walks, Erneuerungsprozesse (renewal processes), Semi-Markovprozesse, usw., wobei sich diese Klassen zum Teil überlappen.

Wir wollen die besonderen Eigenschaften der für diese Arbeit wichtigen Markovprozesse betrachten und dabei mit der Definition zeitdiskreter Markovketten beginnen.

Definition: Eine Familie von Zufallsvariablen $\{X_0, X_1, X_2, \ldots\}$ bildet eine *zeitdiskrete Markovkette*, falls für alle $n \in \mathbb{N}$, für alle möglichen Werte $i_0, i_1, \ldots, i_{n-1}$ und für alle j gilt

$$\begin{aligned} &Prob[X_n = j \mid X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}] \\ =&Prob[X_n = j \mid X_{n-1} = i_{n-1}] \end{aligned} \tag{A.1}$$

Umgangssprachlich ausgedrückt heißt das, daß die zukünftige Entwicklung einer zeitdiskreten Markovkette lediglich vom gegenwärtigen Zustand, nicht jedoch von der Vorgeschichte abhängig ist. Diese Eigenschaft bezeichnet man auch als *Markoveigenschaft*.

Wir betrachten nun die Wahrscheinlichkeit für den Übergang von einem Zustand i in einen Zustand j in n Schritten. Diese n-Schritt Übergangswahrscheinlichkeit $p_{ij}(n)$ ist definiert als

$$p_{ij}(n) = Prob[X_{m+n} = j \mid X_m = i]$$

Im allgemeinen kann diese bedingte Wahrscheinlichkeit abhängig vom Zeitschritt m sein. Wir wollen hier jedoch nur den *homogenen* Fall betrachten, d.h. den Fall daß $p_{ij}(n)$ unabhängig von m ist. Für $n = 1$ erhält man speziell

$$p_{ij} = p_{ij}(1) = Prob[X_{m+1} = j \mid X_m = i]$$

und in Matrixform

$$P = [p_{ij}] = \begin{bmatrix} p_{11} & p_{12} & \cdots \\ p_{21} & & \\ \vdots & & \ddots \\ & & \end{bmatrix}$$

Die Dimension der Übergangsmatrix P kann sowohl endlich als auch unendlich sein, je nachdem, ob die Markovkette endlich oder unendlich viele verschiedene Zustände annehmen kann. Aus den beiden offensichtlichen Bedingungen $0 \leq p_{ij} \leq 1$ und $\sum_j p_{ij} = 1$, sowie der Zusatzbedingung $\forall j : \sum_i p_{ij} > 0$ (d.h. jede Spalte der Matrix P muß mindestens ein von 0 verschiedenes Element enthalten) folgt, daß P eine stochastische Matrix ist [Varga, 1962; Stewart, 1989].

Betrachten wir nun den Zusammenhang zwischen den n-Schritt und den 1-Schritt Übergangswahrscheinlichkeiten: Um in $m + n$ Schritten vom Zustand i in den Zustand j zu gelangen, muß nach den ersten m Schritten ein Zwischenzustand k durchlaufen werden. Daher gilt

$$p_{ij}(m + n) = \sum_k p_{ik}(m) p_{kj}(n)$$

Diese Beziehung ist als Chapman-Kolmogorov Gleichung für zeitdiskrete Markovketten bekannt, mit deren Hilfe sich die n-Schritt Übergangswahrscheinlichkeiten rekursiv berechnen lassen als

$$p_{ij}(n) = \sum_k p_{ik} p_{kj}(n - 1)$$

In Matrixschreibweise erhält man

$$P(n) = [p_{ij}(n)] = P \cdot P(n-1) = \ldots = P^n$$

Nun sei $\pi_j^{(n)}$ definiert als die Wahrscheinlichkeit, daß sich die Markovkette im n-ten Zeitschritt im Zustand j befindet

$$\pi_j^{(n)} = Prob[X_n = j]$$

Damit gelangt man zu der wichtigen Beziehung

$$\pi^{(n)} = \left[\pi_j^{(n)}\right] = \pi^{(n-1)}P = \ldots = \pi^{(0)}P^n$$

Die Zustandswahrscheinlichkeiten im n-ten Zeitschritt sind also eindeutig bestimmt durch die Übergangsmatrix P und die Ausgangsverteilung $\pi^{(0)}$.

Man nennt eine Markovkette *irreduzibel*, falls jeder Zustand j ausgehend von einem beliebigen, fest vorgegebenen Zustand i in endlich vielen Schritten erreichbar ist. Ist ausgehend vom Zustand i die Rückkehr zum selben Zustand i nur nach γ, 2γ, 3γ, ... Schritten möglich ($\gamma > 1$), so nennt man den Zustand i *periodisch*. Anderfalls heißt i *aperiodisch*.

Mit Hilfe dieser Begriffe können wir nun die Voraussetzungen angeben, unter denen eine zeitdiskrete Markovkette eine stationäre Verteilung (Gleichgewichtsverteilung) besitzt [Trivedi, 1982, S. 320; Kleinrock, 1975, S. 29]. Ist eine homogene Markovkette irreduzibel und aperiodisch, so existieren die Grenzwerte

$$\pi_j = \lim_{n\to\infty} \pi_j^{(n)}$$

für alle j und sind unabhängig von der Ausgangsverteilung. Als zusätzliche Bedingung für die Existenz einer stationären Verteilung ist noch zu fordern, daß die mittleren Rekurrenzzeiten der Zustände, d.h. die mittlere Anzahl der Schritte bis zum Wiedereintritt in denselben Zustand, endlich sind. Markovketten, die neben Irreduzibilität und Aperiodizität diese Bedingung erfüllen, nennt man *ergodisch*.

Aus der Beziehung

$$\pi_j^{(n)} = \sum_i \pi_i^{(n-1)} p_{ij}$$

erhält man — unter Voraussetzung der Existenz der Grenzwerte — durch Grenzübergang und Vertauschung von Grenzwert und Summation die folgende Identität:

$$\lim_{n\to\infty} \pi_j^{(n)} = \lim_{n\to\infty} \sum_i \pi_i^{(n-1)} p_{ij} = \sum_i \lim_{n\to\infty} \pi_i^{(n-1)} p_{ij}$$

Daraus folgt der für die Bestimmung der stationären Verteilung wichtige Zusammenhang

$$\pi_j = \sum_i \pi_i p_{ij}$$

mit der Nebenbedingung

$$\sum_j \pi_j = 1$$

In Matrixform lautet diese Beziehung

$$\begin{aligned} \pi P &= \pi \\ \pi e &= 1 \end{aligned} \tag{A.2}$$

Dabei ist e ein Vektor, dessen Elemente alle gleich eins sind. Es handelt sich hierbei also um ein (singuläres) lineares Gleichungssystem zur Bestimmung der stationären Zustandswahrscheinlichkeiten der zeitdiskreten Markovkette.

Bevor wir zur Betrachtung zeitkontinuierlicher Markovketten übergehen, wollen wir noch die Verteilung der Verweilzeit in einem Zustand einer zeitdiskreten Markovkette angeben. Sei N_i definiert als die Anzahl der Zeitschritte, die die Markovkette in einem Zustand verweilt, bevor eine Transition in einen anderen Zustand erfolgt. N_i ist eine Zufallsvariable, deren Verteilung nun leicht angegeben werden kann als

$$Prob[N_i = n] = p_{ii}^{n-1}(1 - p_{ii}), \quad n = 1, 2, \ldots$$

Man sieht, daß sich als direkte Folge aus der Markoveigenschaft für die Zufallsvariable N_i die einzige diskrete Verteilung mit der Eigenschaft der Gedächtnislosigkeit, also eine geometrische Verteilung, ergibt.

Wir gehen nun vom zeitdiskreten zum zeitkontinuierlichen Fall über.

Definition: Der stochastische Prozeß $X(t)$ bildet eine *zeitkontinuierliche Markovkette*, falls für alle $n \in \mathbb{N}$ und für alle möglichen Werte $t_0 < t_1 < \ldots < t_n$ gilt

$$\begin{aligned} &Prob[X(t_n) = j \mid X(t_0) = i_0, X(t_1) = i_1, \ldots, X(t_{n-1}) = i_{n-1}] \\ =&Prob[X(t_n) = j \mid X(t_{n-1}) = i_{n-1}] \end{aligned} \tag{A.3}$$

Diese Bedingung für zeitkontinuierliche Markovketten ist offensichlich analog zu der oben dargestellten für den zeitdiskrteten Fall, vgl. Gl. (A.1).

Die Übergangswahrscheinlickeit $p_{ij}(\Delta t)$ vom Zustand i in den Zustand j über ein Zeitintervall der Länge Δt ist definiert als

$$p_{ij}(\Delta t) = Prob[X(s + \Delta t) = j \mid X(s) = i]$$

Die Matrix $P(\Delta t)$ enthält alle solche Übergangswahrscheinlichkeiten zwischen zwei Zuständen

$$P(\Delta t) = [p_{ij}(\Delta t)]$$

Während $p_{ij}(\Delta t)$ im allgemeinen vom absoluten Zeitpunkt s abhängig sein kann, wollen wir auch hier nur den homogenen Fall betrachten, also die Unabhängigkeit von s voraussetzen. Für die Übergangswahrscheinlichkeit ist dann nur die Zeitdifferenz Δt maßgebend.

Wir definieren die Wahrscheinlichkeit, daß sich die Markovkette zum Zeitpunkt t im Zustand j befindet

$$\pi_j(t) = Prob[X(t) = j]$$

und können damit die Zustandswahrscheinlichkeit zum Zeitpunkt $t + \Delta t$ wie folgt schreiben

$$\pi_j(t + \Delta t) = \sum_i \pi_i(t) p_{ij}(\Delta t)$$

bzw. in Matrixform

$$\pi(t + \Delta t) = \pi(t) P(\Delta t)$$

Zur Bestimmung des Vektors der Zustandswahrscheinlichkeiten zum Zeitpunkt t leitet man über eine Grenzwertbetrachtung folgende Differentialgleichung her:

$$\begin{aligned}
\frac{d\pi(t)}{dt} &= \lim_{\Delta t \to 0} \frac{\pi(t + \Delta t) - \pi(t)}{\Delta t} \\
&= \lim_{\Delta t \to 0} \pi(t) \frac{P(\Delta t) - I}{\Delta t} \\
&= \pi(t) \lim_{\Delta t \to 0} \frac{P(\Delta t) - I}{\Delta t}
\end{aligned}$$

Hierbei bezeichnet I eine Identitätsmatrix passender Dimension. Wir definieren die Ratenmatrix Q, die auch als *infinitesimale Generatormatrix* der zeitkontinuierlichen Markovkette bezeichnet wird, folgendermaßen:

$$Q = \lim_{\Delta t \to 0} \frac{P(\Delta t) - I}{\Delta t}$$

Im einzelnen ergeben sich für die Elemente der Matrix Q die Beziehungen

$$\begin{aligned}
q_{ij} &= \lim_{\Delta t \to 0} \frac{p_{ij}(\Delta t)}{\Delta t} \quad , \quad i \neq j \\
q_{ii} &= \lim_{\Delta t \to 0} \frac{p_{ii}(\Delta t) - 1}{\Delta t}
\end{aligned}$$

Die Differentialgleichung läßt sich damit kurz schreiben als

$$\frac{d\pi(t)}{dt} = \pi(t) Q$$

Unter entsprechenden Voraussetzungen wie im zeitdiskreten Fall kann auch bei zeitkontinuierlichen Markovketten auf die Existenz einer stationären Verteilung π geschlossen werden. Dann gilt also

$$\pi = \lim_{t \to \infty} \pi(t)$$

Da in diesem Fall der Differentialquotient für große Werte von t verschwinden muß, können wir schreiben

$$\lim_{t \to \infty} \frac{d\pi(t)}{dt} = 0$$

Damit gelangt man zu folgendem linearen Gleichungssystem für die Bestimmung der stationären Zustandswahrscheinlichkeiten:

$$\begin{aligned} \pi Q &= 0 \\ \pi e &= 1 \end{aligned} \tag{A.4}$$

Anhand folgender Überlegung kann man leicht zeigen, daß sich eine zeitkontinuierliche Markovkette in eine zeitdiskrete überführen läßt und umgekehrt. Ausgehend von der soeben gewonnenen Beziehung $\pi Q = 0$ erreicht man durch Multiplikation mit einem kleinen Zeitintervall Δt und anschließendes Addieren des Vektors π auf beiden Seiten der Gleichung folgende Darstellung

$$\pi Q \Delta t + \pi = \pi(Q\Delta t + I) = \pi$$

Diesen Vorgang nennt man Diskretisierung, Δt ist der Diskretisierungsparameter. Ist Δt klein genug, d.h. gilt

$$\Delta t < \frac{1}{\max\limits_i |q_{ii}|} \tag{A.5}$$

so ist $P = Q\Delta t + I$ eine stochastische Matrix. Damit wurde eine Darstellung der der folgenden Form gewonnen

$$\pi P = \pi, \quad \pi e = 1$$

Diese Darstellung entspricht genau der Problemdarstellung einer zeitdiskreten Markovkette in Gl. (A.2).

Auch für den Fall einer zeitkontinuierlichen Markovkette wollen wir abschließend noch die Verteilung der Verweilzeit in einem Zustand betrachten. Aus der Markovbedingung kann unmittelbar gefolgert werden, daß die Verweildauer T_i in einem Zustand exponentiell verteilt ist [Kleinrock, 1975, S. 45; Çinlar, 1975, S. 242]. Ausgehend von der Gleichung

$$Prob[T_i > s + t \mid T_i > s] = \frac{Prob[T_i > s + t, T_i > s]}{Prob[T_i > s]} = \frac{Prob[T_i > s + t]}{Prob[T_i > s]}$$

erhält man

$$\begin{aligned} Prob[T_i > s + t] &= Prob[T_i > s + t \mid T_i > s] \cdot Prob[T_i > s] \\ &= Prob[T_i > t] \cdot Prob[T_i > s] \end{aligned}$$

Die zuletzt benutzte Äquivalenz gilt aufgrund der Markoveigenschaft. Die einzige kontinuierliche Verteilung, die diese Gleichung erfüllt ist die Exponentialverteilung.

A.2 Zusammenfassen von Zuständen

In diesem Abschnitt werden die Grundlagen für das Zusammenfassen von Zuständen einer Markovkette dargestellt. Dieses Prinzip wird in der englischsprachigen Literatur als *Lumpability* bezeichnet, und wir benutzen dafür den deutschen Ausdruck *Zusammenfaßbarkeit*. Es existieren diverse Kategorien von Zusammenfaßbarkeit, nämlich gewöhnliche (ordinary), exakte (exact), strikte (strict), schwache (weak) und annähernde (near), von denen wir hier nur die drei ersten diskutieren wollen, da sie für die vorliegende Arbeit relevant sind. Eine ausführlichere Darstellung der Grundlagen der Zusammenfaßbarkeit wird in [Kemeny und Snell, 1976] gegeben, und eine gute Übersicht findet man in dem neueren Papier [Buchholz, 1994].

Das Zusammenfassen von Zuständen ermöglicht eine Reduktion des Zustandsraums durch die Aggregierung von Zuständen, wodurch eine vergröberte Sicht auf die Markovkette entsteht. Der Aufwand für die Analyse der zusammengefaßten Markovkette ist dann geringer als für die Analyse der ursprünglichen. Für viele Betrachtungen ist diese vergröberte Sicht aber immer noch ausreichend zur Beantwortung interessierender Fragen.

Die Grundidee der Zusammenfaßbarkeit ist es, den Zustandsraum der ursprünglichen Markovkette nach bestimmten Kriterien in disjunkte Zustandsmengen zu partitionieren und jede dieser Zustandsmengen als Makrozustand zu interpretieren. Man beobachtet, daß das Maskieren von Zustandsübergangen innerhalb eines Makrozustands einen reduzierten stochastischen Prozeß ergibt, der unter bestimmten Voraussetzungen ebenfalls die Markoveigenschaft besitzt, man also wiederum eine Markovkette erhält. Dabei stehen die stationären und transienten Zustandswahrscheinlichkeiten der ursprünglichen und der zusammengefaßten Markovkette in engem Zusammenhang.

Wir wollen nun den mathematischen Hintergrund für die Zusammenfaßbarkeit von zeitdiskreten Markovketten betrachten, entsprechende Zusammenhänge gelten aber auch für zeitkontinuierliche Markovketten. Sei also X eine zeitdiskrete Markovkette mit dem endlichen, n-elementigen Zustandsraum Z, d.h. $\forall i \in \mathbb{N} : X_i \in Z$. Diese Markovkette ist eindeutig spezifiziert durch eine stochastische Übergangsmatrix P. Sei $\Omega = \{\omega_1, \ldots, \omega_N\}$ eine Partition des Zustandsraums Z in N disjunkte Teilmengen, wobei $N \leq n$. Zwei Zustände, die in derselben Teilmenge liegen, bezeichnet man als im Sinne der Partition äquivalent. Dann wird die zugehörige *Projektionsmatrix* (Buchholz nennt sie *Kollektormatrix*) $V \in \mathbb{R}^{n \times N}$ definiert als diejenige Matrix, deren Elemente gegeben sind durch

$$V(i, J) = \begin{cases} 1 & i \in \omega_J \\ 0 & \text{sonst} \end{cases}$$

Die Matrix V gibt an, welche Zustände i in welcher Teilmenge ω_J liegen. Ferner ist $U = \overline{V^T} \in \mathbb{R}^{N \times n}$ die zugehörige *Distributormatrix*, die aus V durch Transposition und Normalisierung entsteht. Die Schreibweise $\overline{V^T}$ bedeutet also eine Normalisierung der Zeilen der Matrix V^T auf den Wert 1. Die Rechts-Multiplikation von P mit V bewirkt ein Aufsummieren aller derjenigen Spalten von Q, die bezüglich der Partition äquivalent sind. Die Linksmultiplikation mit U bewirkt eine Mittelwertbildung über diejenigen Zeilen, die äquivalente Zustände repräsentieren.

Definition: Seien P, Ω, U und V definiert wie oben. Die Markovkette X heißt

- *gewöhnlich zusammenfaßbar (ordinarily lumpable) bezüglich* Ω gdw. $\forall J \in \{1, \ldots, N\}$ und $\forall i, j \in \omega_J$ gilt: $(e_i - e_j)PV = 0$. (Teilzeilenkriterium)
- *exakt zusammenfaßbar (exactly lumpable) bezüglich* Ω gdw. $\forall J \in \{1, \ldots, N\}$ und $\forall i, j \in \omega_J$ gilt: $(e_i - e_j)P^T V = 0$. (Teilspaltenkriterium)
- *strikt zusammenfaßbar (strictly lumpable)* gdw. X sowohl gewöhnlich als auch exakt zusammenfaßbar ist.

Dabei bezeichnet e_i einen Zeilenvektor passender Länge mit 1 in Position i und 0 in allen anderen Positionen.

Die Analyse der aufgrund *gewöhnlicher Zusammenfaßbarkeit* entstandenen reduzierten Markovkette erlaubt die exakte Bestimmung der Wahrscheinlichkeiten der Makrozustände, nicht aber der Einzelwahrscheinlichkeiten der Zustände. Die Wahrscheinlichkeit für den Makrozustand Π_J ist gleich der Summe aller Einzelwahrscheinlichkeiten der Zustände, die zum Makrozustand J gehören:

$$\Pi_J = \sum_{i \in \omega_J} \pi_i \tag{A.6}$$

Liegt gewöhnliche Zusammenfaßbarkeit vor, so sind im Produkt QV alle Zeilen, die äquivalenten Zuständen entsprechen, identisch. Es ist daher nicht notwendig, bei der Linksmultiplikation mit U eine Mittelwertbildung vorzunehmen und deshalb vorteilhaft, eine modifizierte Distributormatrix U' zu benutzen, welche aus V^T entsteht, indem man in jeder Zeile alle 1-Elemente außer dem ersten durch 0 ersetzt. Dies bewirkt, daß durch die Links-Multiplikation mit U' jeweils genau eine (die erste) dieser identischen Zeilen ausgewählt wird. Aus diesem Grund bezeichnen wir U' in diesem Fall als *Selektionsmatrix*. Für die Einzelwahrscheinlichkeiten lassen sich bei gewöhnlicher Zusammenfaßbarkeit nur (mehr oder weniger ungenaue) Schranken angeben

Im Falle von *exakter Zusammenfaßbarkeit*, gilt ebenfalls Gl. (A.6), die Einzelwahrscheinlichkeiten sind jedoch nun exakt bestimmbar durch die Beziehung

$$\pi_i = \frac{\Pi_J}{|\omega_J|} \quad , \quad i \in \omega_J$$

Es haben also alle zu einem Makrozustand gehörenden Zustände dieselbe Wahrscheinlichkeit.

Allgemein gilt für Zusammenfaßbarkeit, daß die Übergangsmatrix $\hat{P}$ der zusammengefaßten Markovkette bezüglich einer vorgegebenen Partition Ω als das Matrixprodukt

$$\hat{P} = UPV \tag{A.7}$$

berechnet werden kann [Kemeny und Snell, 1976]. Entsprechendes gilt für zeitkontinuierliche Markovketten, wenn man die Übergangsmatrix P durch die Generatormatrix Q ersetzt.

Beispiel:
Wir betrachten eine zeitdiskrete Markovkette mit der Übergangsmatrix P welche gegeben ist durch

$$P = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1/4 & 1/4 & 1/2 \\ 0 & 5/8 & 3/8 \end{bmatrix}$$

Die stationäre Verteilung π dieser Markovkette wird ermittelt aus der Gleichung $\pi(P - I) = 0$ und führt zu dem Wahrscheinlichkeitsvektor

$$\pi = (1/9, 4/9, 4/9)$$

Betrachten wir nun die Partition $\Omega_1 = \{\{1, 2\}, \{3\}\}$, welche die ersten beiden Zustände in Relation setzt. Wir stellen fest, daß die Markovkette bezüglich dieser Partition gewöhnlich zusammenfaßbar ist, da das Teilzeilenkriterium erfüllt ist. Es gilt nämlich

$$P(1, 1) + P(1, 2) = P(2, 1) + P(2, 2) = 1/2$$

und

$$P(1, 3) = P(2, 3) = 1/2$$

Wir berechnen die Übergangsmatrix der zusammengefaßten Markovkette als

$$\hat{P} = U'PV = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1/4 & 1/4 & 1/2 \\ 0 & 5/8 & 3/8 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \\ 5/8 & 3/8 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ 5/8 & 3/8 \end{bmatrix}$$

Die stationäre Lösung Π der zusammengefaßten Markovkette ergibt sich aus der Bestimmungsgleichung $\Pi(\hat{P} - I) = 0$ und wir erhalten

$$\Pi = (5/9, 4/9)$$

Offensichtlich gilt wie erwartet $\Pi_1 = \pi_1 + \pi_2$
Die Markovkette unseres Beispiels ist bezüglich der Partition Ω_1 nicht exakt zusammenfaßbar, da das Teilspaltenkriterium nicht erfüllt ist, wie man leicht überprüfen kann.
Betrachten wir jedoch eine andere Partition $\Omega_2 = \{\{1\}, \{2, 3\}\}$, so stellen wir fest, daß P bezüglich dieser exakt zusammenfaßbar ist. Das Teilspaltenkriterium lautet nämlich in diesem Fall

$$P(2, 2) + P(3, 2) = P(2, 3) + P(3, 3) = 7/8$$

und

$$P(1, 2) = P(1, 3) = 1/2$$

Hier erhält man die Übergangsmatrix der zusammengefaßten Markovkette wie folgt:

$$\hat{P} = UPV = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1/4 & 1/4 & 1/2 \\ 0 & 5/8 & 3/8 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1/4 & 3/4 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1/8 & 7/8 \end{bmatrix}$$

und die stationäre Lösung aus $\Pi(\hat{P} - I) = 0$ als

$$\Pi = (1/9, 8/9)$$

Auch hier gilt, daß die Makrozustände die Wahrscheinlichkeiten der ihnen zugehörigen Einzelwahrscheinlichkeiten auf sich vereinigen, d.h. $\Pi_1 = \pi_1$ und $\Pi_2 = \pi_2 + \pi_3$. Zusätzlich weiß man hier, daß sich die Wahrscheinlichkeit $\Pi_2 = 8/9$ *gleichmäßig* auf π_2 und π_3 verteilt, daß also gilt $\pi_2 = \pi_3 = \Pi_2/2 = 4/9$.

A.3 Numerische Analyseverfahren

Zur Bestimmung der stationären Zustandswahrscheinlichkeiten einer Markovkette ist im allgemeinen eine Matrixanalyse erforderlich, d.h. die Lösung eines Problems in einer der beiden folgenden dualen Darstellungen:

$$\pi Q = 0 \quad , \quad \pi e = 1$$
$$\pi P = \pi \quad , \quad \pi e = 1$$

Man kann das Problem also zum einen als ein homogenes, singuläres lineares Gleichunsgssystem, zum andern als ein Eigenvektorproblem einer unsymmetrischen, oft sehr dünn besetzten Matrix zum vorgegebenen Eigenwert eins ansehen.

Einen Überblick über numerische Verfahren zur Analyse von Markovketten geben z.B. [Krieger, 1990] und [Philippe *et al.*, 1992]. Grundsätzlich kann man eine Einteilung in direkte Verfahren und iterative Verfahren vornehmen.

Grundlage der *direkten Verfahren* ist das bekannte Gauß'sche Eliminationsverfahren, zu dem es verschiedene Varianten gibt, z.B. die LU-Zerlegung. Die direkten Verfahren können nur bei kleinen, dicht besetzten Matrizen vorteilhaft eingesetzt werden. Das liegt daran, daß im Zusammenhang mit direkten Verfahren nicht von Speichertechniken für dünnbesetzte Matrizen profitiert werden kann. Bei solchen "sparse-matrix"-Techniken werden nur die von Null verschiedenen Elemente der Matrix gespeichert. Beim Gauß-Verfahren kann sich jedoch die Nicht-Null-Struktur der Matrix dynamisch ändern, da bei der Elimination des führenden Elements einer Zeile Einträge in solchen Positionen entstehen können, die zuvor mit Null besetzt waren (fill-in).

Dagegen können Speichertechniken für dünnbesetzte Matrizen bei den *iterativen Verfahren* gut eingesetzt werden, denn hier ist die Grundoperation eine Vektor-Matrix-Multiplikation, bei der die Matrixstruktur der Iterationsmatrix erhalten bleibt. Um zu einem Iterationsschema zu gelangen, wird das System $\pi Q = 0$ umgeformt zu $\pi H = \pi$ und daraus das Iterationsschema

$$\pi^{(n)} = \pi^{(n-1)} H$$

abgeleitet (vgl. Abschnitt 3.5.2). Beispielsweise benutzt man für die Power-Methode die Iterationsmatrix $H_{pow} = Q\Delta t + I$, wobei Δt wieder die Bedingung aus Gl. (A.5) zu erfüllen hat. Auf ähnliche Weise entstehen die Iterationsvorschriften des Jacobi-Verfahrens, des Gauß-Seidel-Verfahrens und des SOR-Verfahrens [Buchholz, 1991, S. 34]. Ein Problem bei den iterativen Verfahren kann ein schlechtes Konvergenzverhalten sein. So ist z.B. bei der Power-Methode der Konvergenzfaktor durch den Quotienten λ_2/λ_1 bestimmt, wobei λ_2 den subdominanten Eigenwert der Iterationsmatrix H_{pow} darstellt. In Fällen, bei denen sich die Eigenwerte in der Nähe des dominanten Eigenwerts $\lambda_1 = 1$ häufen, kommt es zu einem sehr schlechten Konvergenzverhalten. Dies ist z.B. bei fast vollständig zerlegbaren Systemen (nearly completely decomposable systems, NCD) [Courtois, 1977] der Fall. Abhilfe kann in diesem Fall eine Vorkonditionierung schaffen.

Im Rahmen der iterativen Verfahren ist auch auf die sogenannten Projektionstechniken hinzuweisen, zu denen das Verfahren der konjugierten Gradienten (CG-Verfahren), das Arnoldi-Verfahren und das GMRES-Verfahren gehören. Die Grundidee bei diesen Verfahren ist es, die exakte Lösung durch eine Folge von Näherungen aus einem Unterraum kleinerer Dimension zu approximieren.

Auf Dekompositions/Aggregationsverfahren wurde bereits in Abschnitt. 3.5.1 kurz eingegangen, als weitere Literatur sei auf [Koury *et al.*, 1984] und [Cao und Stewart, 1985] verwiesen. Ein neueres solches experimentelles Verfahren, das von den zur Lösung von partiellen Differentialgleichungen eingesetzten Mehrgitterverfahren inspiriert wurde, ist das Multi-Level-Verfahren [Horton und Leutenegger, 1993].

Neben den bisher erwähnten Verfahren gibt es eine Reihe besonderer Verfahren für Markovmodelle mit unendlichem Zustandsraum [King und Mitrani, 1980], z.B. die matrixgeometrische Methode [Neuts, 1981] und die Methode der Spektralanalyse [Chakka und Mitrani, 1992]. Modelle mit unendlichem Zustandsraum werden in dieser Arbeit nicht behandelt.

Neben der Ermittlung der stationären Zustandswahrscheinlichkeiten ist es für die Beantwortung vieler Fragestellungen wichtig, transiente Leistungsmaße zu ermitteln, also die Markovkette über einen endlichen Zeitraum zu betrachten, ausgehend von einem vorgegebenen Ausgangszustand. Die transiente Analyse spielt besonders bei Anwendungen aus den Gebieten Zuverlässigkeitsbewertung und Performability [Meyer, 1980; Smith *et al.*, 1988] eine große Rolle. Für die transiente Analyse ist die Lösung der folgenden, im Anhang A.1 hergeleiteten Differentialgleichung erforderlich:

$$\frac{d\pi(t)}{dt} = \pi(t)Q$$

Klassische Verfahren wie z.B. das Verfahren der numerischen Integration nach Runge-Kutta führen zwar zu einer Lösung, nutzen aber nicht die speziellen probabilistischen Eigenschaften der Generatormatrix Q und des Vektors π. Daher werden i.a. sogenannte Randomisierungsverfahren eingesetzt, die die zeitkontinuierliche Markovkette in eine zeitdiskrete überführen, dafür die Zustandswahrscheinlichkeiten im i-ten Schritt berechnen und schließlich zur Wiedereinführung der kontinuierlichen Zeit die diskrete Markovkette in einen Poissonprozeß einbetten [Jensen, 1953; Grassmann, 1977; Gross und Miller, 1984; Ramesh und Trivedi, 1993].

Anhang B Tensoralgebra

B.1 Mathematische Grundlagen

Als wichtiges mathematisches Handwerkszeug werden in Abschnitt 3.5.2 und in Kapitel 4 Methoden der Tensoralgebra benötigt. Daher werden in diesem Abschnitt die Grundlagen der Tensoralgebra bereitgestellt. Für eine vertiefende Behandlung dieses Stoffes sei auf die Arbeiten [Davio, 1981] und [Massey, 1984] hingewiesen.

Zunächst werden die neben der Matrixmultiplikation und Matrixaddition wichtigen Operatoren Tensorprodukt und Tensorsumme definiert. Diese beiden Verknüpfungen erlauben es, größere Matrizen aus kleineren aufzubauen, und so große Matrizen in strukturierter Form darzustellen. Diese Operatoren werden wir benutzen, um Übergangs- bzw. Generatormatrizen komplexer Markovprozesse, d.h. solcher Markovprozesse, die das cartesische Produkt meherer, teilweise unabhängiger Markovprozesse sind, anzugeben.

Definition: Das *Tensorprodukt* (oft auch als Kroneckerprodukt bezeichnet) zweier Matrizen $A \in \mathbb{R}^{r_A \times c_A}$ und $B \in \mathbb{R}^{r_B \times c_B}$ ist definiert als die Matrix $C \in \mathbb{R}^{r_A r_B \times c_A c_B}$ mit

$$C = A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1c_A}B \\ a_{21}B & & & \\ \vdots & & & \vdots \\ a_{r_A 1}B & & \dots & a_{r_A c_A}B \end{bmatrix}$$

Die Definition für das Tensorprodukt stellt also keinerlei Anforderungen an die Dimension der beiden beteiligten Matrizen A und B. Man veranschaulicht sich die Definition des Tensorprodukts leicht an folgendem Beispiel:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} & a_{13}b_{11} & a_{13}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} & a_{13}b_{21} & a_{13}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} & a_{23}b_{11} & a_{23}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} & a_{23}b_{21} & a_{23}b_{22} \end{bmatrix}$$

Wir wollen noch auf eine zweite, duale Sicht hinweisen: Stellt man sich die aus dem Tensorprodukt von A und B resultierende Matrix C in Blöcke der Größe $r_B \times c_B$ strukturiert vor, und bezeichnet man mit $C((i_1, i_2), (j_1, j_2))$ das (i_2, j_2) Element von Block (i_1, j_1), so gilt

$$C((i_1, i_2), (j_1, j_2)) = A(i_1, j_1)B(i_2, j_2)$$

Das verallgemeinerte Tensorprodukt von mehr als zwei Matrizen $C = \bigotimes_{k=1}^{n} A_k$ wird induktiv definiert als $\bigotimes_{k=1}^{n} A_k = \left(\bigotimes_{k=1}^{n-1} A_k \right) \otimes A_n$. Man erhält damit eine Gesamtmatrix

C, die aus geschachtelten Blöcken aufgebaut ist, wobei ein bestimmtes Element gleich dem folgenden Produkt ist:

$$C((i_1,\ldots,i_n),(j_1,\ldots,j_n)) = \prod_{k=1}^{n} A_k(i_k,j_k)$$

Das verallgemeinerte Tensorprodukt kann auch als Matrixprodukt von Tensorprodukten wie folgt geschrieben werden.

$$\bigotimes_{k=1}^{n} A_k = \prod_{k=1}^{n} I_{l_k} \otimes A_k \otimes I_{u_k} \tag{B.1}$$

Dabei ist A_k eine Matrix der Dimension d_k, und I_{l_k} und I_{u_k} bezeichnen Identitätsmatrizen der Dimension $\prod_{i=1}^{k-1} d_i$ bzw. $\prod_{i=k+1}^{n} d_i$.
Die zweite wichtige Verknüpfung ist die Tensorsumme, deren Definition auf der des Tensorprodukts basiert.

Definition: Die *Tensorsumme* ist ein Operator zur Verknüpfung zweier *quadratischer* Matrizen $A \in \mathbb{R}^{d_A \times d_A}$ und $B \in \mathbb{R}^{d_B \times d_B}$. Sie ist mit Hilfe des Tensorprodukts definiert als

$$A \oplus B = A \otimes I_{d_B} + I_{d_A} \otimes B$$

wobei I_d eine Identitätsmatrix der Dimension d bezeichnet.

Die Definition für die Tensorsumme verlangt also, daß die beiden beteiligten Matrizen A und B quadratisch sind. Ihre Dimensionen d_A bzw. d_B können beliebig gewählt werden. Zur Veranschaulichung der Tensorsumme dient das folgende Beispiel

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \oplus \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & b_{12} & a_{12} & & a_{13} & \\ b_{21} & a_{11}+b_{22} & & a_{12} & & a_{13} \\ a_{21} & & a_{22}+b_{11} & b_{12} & a_{23} & \\ & a_{21} & b_{21} & a_{22}+b_{22} & & a_{23} \\ a_{31} & & a_{32} & & a_{33}+b_{11} & b_{12} \\ & a_{31} & & a_{32} & b_{21} & a_{33}+b_{22} \end{bmatrix}$$

In der dualen Repräsentation (Blocksicht) ergibt sich nun für ein ausgewähltes Element der Matrix $C = A \oplus B$ die Formel

$$C((i_1,i_2),(j_1,j_2)) = A(i_1,j_1)1(i_2=j_2) + B(i_2,j_2)1(i_1=j_1)$$

Dabei hat die Funktion $1()$ entweder den Wert 0 oder 1, abhängig davon, ob die in Klammern angegebene Bedingung wahr oder falsch ist.
Die verallgemeinerte Tensorsumme von n Matrizen ist analog zum verallgemeinerten Tensorprodukt induktiv definiert als $\bigoplus_{k=1}^{n} A_k = \left(\bigoplus_{k=1}^{n-1} A_k\right) \oplus A_n$. Für die Elemente der Ergebnismatrix C gelangt man dann zu der Beziehung

$$C((i_1,\ldots,i_n),(j_1,\ldots,j_n)) = \sum_{k=1}^{n} A_k(i_k,j_k)1(\forall m \neq k : i_m = j_m)$$

Die verallgemeinerte Tensorsumme kann auch als Matrixsumme von Tensorprodukten wie folgt geschrieben werden.

$$\bigoplus_{k=1}^{n} A_k = \sum_{k=1}^{n} I_{l_k} \otimes A_k \otimes I_{u_k} \tag{B.2}$$

Dabei ist A_k eine quadratische Matrix der Dimension d_k, und I_{l_k} und I_{u_k} bezeichnen wie oben Identitätsmatrizen adäquater Dimension.

Neben den Definitionen für das Tensorprodukt und die Tensorsumme sind schließlich für diese Arbeit die im folgenden Satz zusammengefaßten Regeln für das Rechnen mit Tensoroperatoren wichtig.

Satz: Für das Rechnen mit Tensoroperatoren gilt:

a. Assoziativität des Tensorprodukts

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C$$

b. Distributivität des Tensorprodukts bezüglich der Matrixaddition

$$A \otimes (B + C) = (A \otimes B) + (A \otimes C)$$
$$(A + B) \otimes C = (A \otimes C) + (B \otimes C)$$

c. Für die Transposition gilt

$$(A \otimes B)^T = A^T \otimes B^T$$

d. Kompatibilität des Tensorprodukts mit der Matrixinversion

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$$

e. Ein weiterer wichtiger Zusammenhang, den wir in Abschnitt 4.5.4 benötigen, ist die Kompatibilität zwischen Tensorprodukt und Matrixmultiplikation, gegeben durch

$$(A_1 B_1) \otimes (A_2 B_2) = (A_1 \otimes A_2)(B_1 \otimes B_2)$$

Dieser Zusammenhang kann durch Induktion leicht verallgemeinert werden zu

$$\bigotimes_{i=1}^{n} (A_i B_i) = \left(\bigotimes_{i=1}^{n} A_i \right) \left(\bigotimes_{i=1}^{n} B_i \right) \tag{B.3}$$

f. Es ist zu beachten, daß für das Tensorprodukt die Kommutativität nicht gilt, wie man sich leicht anhand eines einfachen Beispiels veranschaulicht. Allerdings stehen $A \otimes B$ und $B \otimes A$ in der Beziehung

$$A \otimes B = P_r (B \otimes A) P_c$$

Dabei sind P_r und P_c zwei Permutationsmatrizen, die eine Zeilen- bzw. Spaltenpermutation in Form eines Perfect Shuffle bewirken.

Mit den Definitionen der Tensoroperatoren und dem abschließenden Satz sind die für diese Arbeit benötigten Grundlagen der Tensoralgebra bereitgestellt.

Literatur

[Ajmone Marsan *et al.*, 1984] M. Ajmone Marsan, G. Balbo, and G. Conte. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, May 1984.

[Ajmone Marsan *et al.*, 1986] M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems*. MIT Press, Computer System Series, 1986.

[Ajmone Marsan und Chiola, 1987] M. Ajmone Marsan and G. Chiola. On Petri Nets with Deterministic and Exponentially Distributed Firing Times. In G. Rosenberg, editor, *Advances in Petri Nets*, pages 132–45. Springer Verlag, 1987.

[Atif, 1992] K. Atif. *Modelisation du Parallelisme et de la Synchronisation*. PhD thesis, Institut National Polytechnique de Grenoble, 1992. (in French).

[Balbo *et al.*, 1986] G. Balbo, S.C. Bruell, and S. Ghanta. Combining Queueing Networks and Generalized Stochastic Petri Net Models for the Analysis of Some Software Blocking Phenomena. *IEEE Transactions on Software Engineering*, 12(4):561–576, 1986.

[Balbo *et al.*, 1988] G. Balbo, S.C. Bruell, and S. Ghanta. Combining Queueing Networks and Generalized Stochastic Petri Nets for the Solution of Complex Models of System Behaviour. *IEEE Transactions on Computers*, 37(10):1251–1268, Oct. 1988.

[Baskett *et al.*, 1975] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios. Open Closed and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22(2):248–260, 1975.

[Bause und Buchholz, 1993] F. Bause and P. Buchholz. Qualitative and Quantitative Analysis of Timed SDL Specifications. In *Kommunikation in Verteilten Systemen*, pages 486–500, München, March 1993. Springer, Informatik aktuell.

[Berthelot, 1986] G. Berthelot. Checking Properties of Nets Using Transformations. In G. Rozenberg, editor, *Advances in Petri Nets 1985, LNCS 222*, pages 19–40. Springer, 1986.

[Berthelot, 1987] G. Berthelot. Transformations and Decompositions of Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1986, LNCS 254*, pages 359–376. Springer, 1987.

[Bhandarkar, 1975] D.P. Bhandarkar. Analysis of Memory Interference in Multiprocessors. *IEEE Transactions on Computers*, C-24(9):897–908, September 1975.

[Bobbio und Cumani, 1991] A. Bobbio and A. Cumani. ML estimation of the parameters of a PH distribution in triangular canonical form. In G. Balbo and G. Serazzi, editors, *Modelling Techniques and Tools for Computer Performance Evaluation*, pages 31–44, Torino, February 1991.

[Bode und Händler, 1980] A. Bode und W. Händler. *Rechnerarchitektur*. Springer-Verlag, Berlin, 1980.

[Bolch *et al.*, 1995] G. Bolch, M. Roessler, R. Zimmer, H. Jung und S. Greiner. Leistungsbewertung mit PEPSY-QNS und MOSES. *Informationstechnik und Technische Informatik*, (3/95):28–33, Juni 1995.

[Bolch, 1989] G. Bolch. *Leistungsbewertung von Rechensystemen mittels analytischer Warteschlangenmodelle*. Leitfäden und Monographien der Informatik. B.G. Teubner Stuttgart, 1989.

[Brand und Zafiropulo, 1983] D. Brand and P. Zafiropulo. On Communicating Finite-State Machines. *JACM*, 30:323–342, April 1983.

[Buchholz, 1991] P. Buchholz. *Die strukturierte Analyse Markovscher Modelle*. Dissertation, Universität Dortmund, 1991.

[Buchholz, 1992a] P. Buchholz. Hierarchical Markovian Models - Symmetries and Reduction. In R. Pooley and J. Hillston, editors, *6th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 305–319, Edinburgh, September 1992.

[Buchholz, 1992b] P. Buchholz. Numerical Solution Methods Based on Structured Descriptions of Markovian Models. In G. Balbo and G. Serazzi, editors, *Proceedings of the 5th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 242–258. Elsevier Science Publisher B.V., 1992.

[Buchholz, 1994] P. Buchholz. Exact and Ordinary Lumpability in Finite Markov Chains. *Journal of Applied Probability*, (31):59–75, 1994.

[Bux und Herzog, 1977] W. Bux and U. Herzog. The Phase Concept: Approximation of Measured Data and Performance Analysis. In *Int. Symp. on Computer Performance, Measurements, and Evaluation*, 1977.

[Buzen, 1973] J.P. Buzen. Computational Algorithms for Closed Queueing Networks with Exponential Servers. *Communications of the ACM*, 16:527–531, 1973.

[Cao und Stewart, 1985] W.L. Cao and W.J. Stewart. Iterative Aggregation/-Disaggregation Techniques for Nearly Uncoupled Markov Chains. *Journal of the ACM*, 32(3):702–719, July 1985.

[Çinlar, 1975] E. Çinlar. *Introduction to Stochastic Processes*. Prentice-Hall, 1975.

[CCITT, 1992] CCITT. *Recommendation Z.100: Specification and Description Language SDL, Blue Book*. ITU General Secreteriat — Sales Section, Place des Nations, CH-1211 Geneva 20, 1992.

[Chakka und Mitrani, 1992] R. Chakka and I. Mitrani. A Numerical Solution Method for Multiprocessor Systems with General Breakdowns and Repairs. In *Proceedings of the Sixth International Conference on Modelling Techniques and Tools*, pages 289–304, Edinburgh, September 1992.

[Chandy *et al.*, 1975a] K.M. Chandy, U. Herzog, and L. Woo. Approximate Analysis of General Queuing Networks. *IBM Journal of Research and Development*, 19(1):43–49, Jan. 1975.

[Chandy *et al.*, 1975b] K.M. Chandy, U. Herzog, and L. Woo. Parametric Analysis of Queuing Models. *IBM Journal of Research and Development*, 19(1):36–42, Jan. 1975.

[Chiola *et al.*, 1990a] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic Well-Formed Coloured Nets and Multiprocessor Modelling Applications. IBP Tech. Report 90/41, Universite Paris 6, Oct. 1990. Reprinted in High-level Petri Nets, K. Jensen, G. Rozenberg, eds.

[Chiola *et al.*, 1990b] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On Well-Formed Coloured Nets and their Symbolic Reachability Graph. In *Proceedings of the 11th International Conference on Application and Theory of Petri Nets*, pages 387–410, Paris, June 1990. Reprinted in High-level Petri Nets, K. Jensen, G. Rozenberg, eds.

[Chiola *et al.*, 1991a] G. Chiola, , and G. Franceschinis. A Structural Color Simplification in Well-Formed Colored Nets. In *Proceedings of the 4th International Workshop on Petri Nets and Performance Models*, pages 144–153, Melbourne, December 1991. IEEE.

[Chiola *et al.*, 1991b] G. Chiola, S. Donatelli, and G. Franceschinis. GSPNs versus SPNs: what is the actual role of immediate transitions? In *Proceedings of the 4th International Workshop on Petri Nets and Performance Models*, pages 20–31, Melbourne, December 1991. IEEE.

[Chiola und Franceschinis, 1989] G. Chiola and G. Franceschinis. Colored GSPN Models and Automatic Symmetry Detection. In *3rd Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, Dec. 1989. IEEE-CS Press.

[Chiola, 1992] G. Chiola. GreatSPN 1.5 Software Architecture. In G. Balbo and G. Serazzi, editors, *Proceedings of the 5th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Torino, Feburary 1991*, pages 117–132. Elsevier Science Publisher B.V., 1992.

[Ciardo *et al.*, 1991] G. Ciardo, J. Muppala, and K.S. Trivedi. On the Solution of GSPN Reward Models. *Performance Evaluation*, 12(4):237–254, 1991.

[Ciardo und Muppala, 1991] G. Ciardo and J.K. Muppala. *Manual for the SPNP Package Version 3.1*. Duke University, October 1991.

[Ciardo und Trivedi, 1993] G. Ciardo and K.S. Trivedi. A decomposition approach for stochastic reward net models. *Performance Evaluation*, 18(1):37–59, July 1993.

[Conway und Georganas, 1989] A.E. Conway and N.D. Georganas. *Queueing Networks – Exact Computational Algorithms*. MIT Press, 1989.

[Courtois, 1977] P.J. Courtois. *Decomposability, queueing and computer system applications*. ACM monograph series, 1977.

[Couvillion *et al.*, 1991] J. Couvillion, R. Freire, R. Johnson, W.D. Obal II, M.A. Qureshi, M. Rai, W.H. Sanders, and J. Tvedt. Performability modeling with UltraSAN. *IEEE Software*, 8(5):69–80, September 1991.

[Davio, 1981] M. Davio. Kronecker Products and Shuffle Algebra. *IEEE Transactions on Computers*, C-30(2):116–125, February 1981.

[Dutheillet und Haddad, 1989a] C. Dutheillet and S. Haddad. Aggregation and Disaggregation of States in Colored Stochastic Petri Nets: Application to a Multiprocessor

Architecture. In *3rd Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, Dec. 1989. IEEE-CS Press.

[Dutheillet und Haddad, 1989b] C. Dutheillet and S. Haddad. Regular Stochastic Petri Nets. In *10th Intern. Conf. on Application and Theory of Petri Nets, Advances in Petri Nets 1990, LNCS 483*, pages 186–210, Bonn Germany, June 1989. reprinted in High-level Petri Nets.

[Dutheillet, 1992] C. Dutheillet. *Symétries dans les Réseaux Colorés: Définition, Analyse et Application à l'Evaluation de Performance*. PhD thesis, Université Paris VI, CNRS, 1992. (in French).

[Fleischmann und Werner, 1989] G. Fleischmann und G. Werner. Beschreibung paralleler Softwarestrukturen durch stochastische Graphen und ihre Bewertung mit Hilfe von Markovketten. Interner Bericht, Universität Erlangen–Nürnberg, IMMD IV, 1989.

[Gelenbe und Mitrani, 1980] E. Gelenbe and I. Mitrani. *Analysis and Synthesis of Computer Systems*. Academic Press, 1980.

[Gilmore und Hillston, 1994] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-Based Approach to Performance Modelling. In G. Haring and G. Kotsis, editors, *7th Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 353–368, Wien, May 1994.

[Gordon *et al.*, 1988] R.F. Gordon, E.A. MacNair, K.J. Gordon, and J.F. Kurose. Higher Level Modelling in RESQME. In *Proc. of the European Simulation Multiconference 1988, Nice, France*, pages 52–57, 1988.

[Gordon und Newell, 1967] W.J. Gordon and G.F. Newell. Closed Queueing Systems with Exponential Servers. *Operations Research*, 15:254–265, 1967.

[Götz *et al.*, 1993] N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis Using Stochastic Process Algebras. In *Proc. of the 16th International Symposium on Computer Performance Modelling, Measurement and Evaluation, PERFORMANCE 1993, Tutorial*. Springer LNCS 729, 1993.

[Grassmann, 1977] W. Grassmann. Transient Solutions in Markovian Queues. *European Journal of Operational Research*, 1:396–402, 1977.

[Gross und Miller, 1984] D. Gross and D.R. Miller. The Randomization Technique as a Modeling Tool and Solution Procedure for Transient Markov Processes. *Operations Research*, 32(2):343–361, March-April 1984.

[Haddad, 1990] S. Haddad. A Reduction Theory for Coloured Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1989, LNCS 424*, pages 209–235. Springer, 1990. reprinted in High-level Petri Nets.

[Hartleb und Quick, 1993] F. Hartleb and A. Quick. Performance Evaluation of Parallel Programms – Modeling and Monitoring with the Tool PEPP. In B. Walke and O. Spaniol, editors, *Proceedings der 7. GI–ITG Fachtagung "Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen", Aachen, 21.–23. September 1993*, pages 51–63. Informatik Aktuell, Springer, 1993.

[Haverkort und Trivedi, 1993] B.R. Haverkort and K.S. Trivedi. Specification Techniques for Markov Reward Models. *Discrete Event Systems: Theory and Applications*, 3:219–247, 1993.

[Henderson und Lucic, 1993] W. Henderson and D. Lucic. Aggregation and Disaggregation Through Insensitivity in Stochastic Petri Nets. *Performance Evaluation*, 17(2):91–114, March 1993.

[Hermanns und Rettelbach, 1994] H. Hermanns and M. Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, pages 71–88, Erlangen-Regensberg, July 1994. IMMD, Universität Erlangen-Nürnberg.

[Herzog, 1990] U. Herzog. Formal Description, Time and Performance Analysis. A Framework. In T. Härder, H. Wedekind, and G. Zimmermann, editors, *Entwurf und Betrieb Verteilter Systeme*, pages 172–190. Springer Verlag, Berlin, IFB 264, 1990.

[Hillston, 1994] J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994.

[Hoare, 1985] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ, 1985.

[Hogrefe, 1989] Dieter Hogrefe. *Estelle, LOTOS und SDL*. Springer, Berlin, 1989.

[Horton und Leutenegger, 1993] G. Horton and S. Leutenegger. A Multi-Level Solution Algorithm for Steady-State Markov Chains. Technical Report 9/93, Universität Erlangen-Nürnberg, IMMD III, 1993.

[Jackson, 1957] J.R. Jackson. Networks of Waiting Lines. *Operations Research*, 5:518–521, 1957.

[Jackson, 1963] J.R. Jackson. Jobshop-like Queueing Systems. *Management Science*, 10(1):131–142, 1963.

[Jensen und Rozenberg, 1991] K. Jensen and G. Rozenberg, editors. *High-level Petri Nets*. Springer, 1991.

[Jensen, 1953] A. Jensen. Markoff Chains as an Aid in the Study of Markoff Processes. *Skand. Aktuarietiedskr.*, 36:87–91, 1953.

[Jung, 1991] H. Jung. *Leistungsbewertung UNIX-basierter Betriebssysteme für Multiprozessoren mit globalem Speicher*. Dissertation, Universität Erlangen–Nürnberg, September 1991.

[Kant, 1988] K. Kant. Application Level Modeling of Parallel Machines. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 83–93, Santa Fe, NM, May 1988.

[Kauer, 1992] T. Kauer. Untersuchung von ausgewählten Softwarewerkzeugen zur funktionalen und temporalen Modellierung und Analyse verteilter Systeme. Studienarbeit, Universität Erlangen–Nürnberg, IMMD VII, Juli 1992.

[Kemeny und Snell, 1976] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer, 1976.

[King und Mitrani, 1980] P.J.B. King and I. Mitrani. Numerical Methods for Infinite Markov Processes. In *Proc. Performance 80, ACM Sigmetrics*, pages 277–282, 1980.

[Kleinrock, 1975] L. Kleinrock. *Queueing Systems*, volume 1: Theory. John Wiley & Sons, 1975.

[Kleinrock, 1976] L. Kleinrock. *Queueing Systems*, volume 2: Applications. John Wiley & Sons, 1976.

[Kobayashi, 1981] H. Kobayashi. *Modeling and Analysis — An Introduction to System Performance Evaluation Methodology*. Addison–Wesley, reprinted and corrected edition, October 1981.

[Koury *et al.*, 1984] J.R. Koury, D.F. McAllister, and W.J. Stewart. Iterative Methods for Computing Stationary Distributions of Nearly Completely Decomposable Markov Chains. *SIAM Journal on Algebraic and Discrete Methods*, 5(2):164–186, June 1984.

[Krieger, 1990] U. Krieger. Computational Methods for Markovian Queueing Models - A Survey. Technical report, Forschungsinstitut der Deutschen Bundespost, Darmstadt, 1990.

[Kritzinger, 1985] P.S. Kritzinger. Analyzing the time efficiency of a communication protocol. In Y. Yemini, R. Strom, and S. Yemini, editors, *Protocol Specification, Testing, and Verification, IV*, pages 527–539. North-Holland, 1985.

[Lindemann, 1992] C. Lindemann. DSPNexpress: A Software Package for the Efficient Solution of Deterministic and Stochastic Petri Nets. In *Proceedings of the 6th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 15–29, Edinburgh, September 1992.

[Linn, 1985] R.J. Linn. The Features and Facilities of Estelle: A Formal Description Technique Based upon an Extende Finite State Machine Model. In *Protocol Specification, Testing, and Verification 5*, pages 271–296. IFIP WG 6.1, 1985.

[Liu, 1989] M.T. Liu. Protocol Engineering. *Advances in Computers*, 29:79–195, 1989.

[Markov, 1907] A.A. Markov. Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain. *The Notes of the Imperial Academy of Sciences of St. Petersburg, VIII Series, Physio-Mathematical College*, XXII(9), December 5 1907.

[Massey, 1984] W.A. Massey. Open Networks of Queues: Their Algebraic Structure and Estimating their Transient Behavior. *Adv. Appl. Prob.*, 16:176–201, March 1984.

[Meyer, 1980] J.F. Meyer. On Evaluating the Performability of Degradable Computing Systems. *IEEE Transactions on Computer Systems*, C-29(8):720–731, August 1980.

[Milner, 1989] R. Milner. *A Calculus of Communicating Systems*. Prentice Hall, London, 1989.

[Molloy, 1981] M.K. Molloy. *On the integration of delay and throughput measures in distributed processing models*. PhD thesis, University of California, Los Angeles, 1981.

[Molloy, 1982] M.K. Molloy. Performance Analysis Using Stochastic Petri Nets. *IEEE Trans on Computers*, C-31:913–917, September 1982.

[Murata, 1989] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proc. of the IEEE*, 77(4):541–580, April 1989.

[Natkin, 1980] S. Natkin. *Reseaux de Petri Stochastiques*. PhD thesis, CNAM-Paris, 1980.

[Neuts, 1981] M.F. Neuts. *Matrix–Geometric Solutions in Stochastic Models*. John Hopkins Series in Mathematical Sciences. Johns Hopkins University Press, 1981.

[Neuts, 1989] M.F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Probability: Pure and Applied. Marcel Dekker, New York, Basel, 1989.

[Petri, 1962] C.A. Petri. *Kommunikation mit Automaten*. Dissertation, Universität Bonn, 1962. Schriften des Instituts für Instrumentelle Mathematik Nr. 3.

[Philippe *et al.*, 1992] B. Philippe, Y. Saad, and W.J. Stewart. Numerical Methods in Markov Chain Modeling. *Operations Research*, 40(6):1156–1179, December 1992.

[Plateau *et al.*, 1988] B. Plateau, J.-M. Fourneau, and K.-H. Lee. PEPS: A Package for Solving Complex Markov Models of Parallel Systems. In *Proceedings of the 4th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 341–360, Palma (Mallorca), September 1988.

[Plateau und Atif, 1991] B. Plateau and K. Atif. Stochastic Automata Network for Modeling Parallel Systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[Plateau und Fourneau, 1991] B. Plateau and J.-M. Fourneau. A Methodology for Solving Markov Models of Parallel Systems. *Journal of Parallel and Distributed Computing*, 12:370–387, 1991.

[Plateau, 1985] B. Plateau. On the Synchronization Structure of Parallelism and Synchronization Models for Distributed Algorithms. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 147–154, Austin, TX, August 1985.

[Ramesh und Trivedi, 1993] A. V. Ramesh and K. Trivedi. On the Sensitivity of Transient Solutions of Markov Models. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 122–134, Santa Clara, CA, May 1993.

[Reiser und Lavenberg, 1980] M. Reiser and S. Lavenberg. Mean Value Analysis of Closed Multichain Queueing Networks. *Journal of the ACM*, 27(2):313–322, 1980.

[Rettelbach und Siegle, 1994] M. Rettelbach and M. Siegle. Compositional Minimal Semantics for the Stochastic Process Algebra TIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, pages 89–106, Regensberg/Erlangen, July 1994. Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg.

[Rettelbach, 1991] M. Rettelbach. Leistungsbewertung mit Prozeßalgebren. Diplomarbeit, Universität Erlangen–Nürnberg, IMMD VII, Februar 1991.

[Rettelbach, 1995] M. Rettelbach. Probabilistic Branching in Markovian Process Algebras. *The Computer Journal*, 38(7):590–599, December 1995. Special issue: Proc. of the 3rd Workshop on Process Algebras and Performance Modelling.

[Rudin, 1983] H. Rudin. From Protocol Speciication towards Automated Perfromance Prediction. In *Proc. Int. Workshop on Protocol Specification, Testing and Verification III*, pages 257–269, North Holland, 1983. IFIP 1983, Elsevier Science Publishers.

[Rudin, 1985] H. Rudin. Time in Formal Protocol Specification. In *Proc. Kommunikation in Verteilten Systemen, Informatik Fachberichte No. 95*, Seite 575–587. Springer Verlag, 1985.

[Sahner und Trivedi, 1987] R. Sahner and K. Trivedi. Reliability Modeling using SHARPE. *IEEE Transactions on Reliability*, R-36(2), 1987.

[Sanders *et al.*, 1994] W.H. Sanders, W.D. Obal II, M.A. Qureshi, and F.K. Widjanarko. UltraSAN V2.0 Overview. In G. Haring and H. Wabnig, editors, *Short Papers and Tool Descriptions of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 31–34, Vienna, May 1994.

[Sanders und Meyer, 1991] W.H. Sanders and J.F. Meyer. Reduced Base Model Construction Methods for Stochastic Activity Networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, January 1991.

[Schmickler, 1992] L. Schmickler. MEDA: Mixed Erlang Distributions as Phase-Type Representations of Empirical Distribution Functions. *Commun. Statist.-Stochastic Models*, 8(1):131–156, 1992.

[Schroetter und de Meer, 1991] M. Schroetter und H. de Meer. Tools und Expertensysteme zur Modellierung von Rechensystemen — Ein Überblick. Interner Bericht 1/91, Universität Erlangen-Nürnberg, IMMD IV, 1991.

[Sczittnick, 1987] M. Sczittnick. Techniken zur funktionalen und quantitativen Analyse von Markoffschen Rechensystemmodellen. Diplomarbeit, Universität Dortmund, August 1987.

[Siegle, 1994a] M. Siegle. Reduced Markov Models of Parallel Programs with Replicated Processes. In *2nd EUROMICRO Workshop on "Parallel and Distributed Processing"*, pages 126–133, Malaga, Spain, January 1994.

[Siegle, 1994b] M. Siegle. Structured Markovian Performance Modelling with Automatic Symmetry Exploitation. In G. Haring and H. Wabnig, editors, *Short Papers and Tool Descriptions Proc. of the 7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 77–81, Vienna, Austria, May 1994.

[Siegle, 1994c] M. Siegle. Using Structured Modelling for Efficient Performance Prediction of Parallel Systems. In G.R. Joubert, D. Trystram, F.J. Peters, and D.J. Evans, editors, *Parallel Computing: Trends and Applications, Proceedings of the International Conference ParCo93*, pages 453–460. North-Holland, 1994.

[Sifakis, 1977] J. Sifakis. Use of Petri Nets for Performance Evaluation. In *Proc. of the Third Int. Workshop on Modeling and Performance Evaluation of Computer Systems, Amsterdam*, 1977.

[Simon und Ando, 1961] H.A. Simon and A. Ando. Aggregation of Variables in Dynamic Systems. *Econometrica*, 29:111–138, 1961.

[Simone und Marsan, 1992] C. Simone and M. Ajmone Marsan. The Application of EB-Equivalence Rules to the Structural Reduction of GSPN Models. *Journal of Parallel and Distributed Computing*, 15(3):296–302, July 1992.

[Smith *et al.*, 1988] R.M. Smith, K.S. Trivedi, and A.V. Ramesh. Perfomability Analysis: Measures, an Algorithm, and a Case Study. *IEEE Transactions on Computers*, 37(4):406–417, April 1988.

[Sötz und Werner, 1990] F. Sötz und G. Werner. Lastmodellierung mit stochastischen Graphen zur Verbesserung paralleler Programme auf Multiprozessoren mit Fallstudie. In *11. ITG/GI–Fachtagung Architektur von Rechensystemen*, Seite 231–243. vde–Verlag, Berlin und Offenbach, Januar 1990.

[Starke, 1990] P.H. Starke. *Analyse von Petri-Netz-Modellen*. Teubner, Stuttgart, 1990.

[Stewart *et al.*, 1993] W. Stewart, K. Atif, and B. Plateau. The Numerical Solution of Stochastic Automata Networks. Rapport Apache 6, Institut IMAG, LGI, LMC, Grenoble, November 1993.

[Stewart, 1989] W.J. Stewart. Markov Chains and Stochastic Matrices. Technical Report TR-89-01, North Carolina State University, 1989.

[Stewart, 1991] W.J. Stewart. MARCA: Markov Chain Analyzer, A Software Package for Markov Modeling. In W.J. Stewart, editor, *Numerical Solution of Markov Chains*. Marcel Dekker, 1991.

[Studt, 1995] R. Studt. Konzeption und Implementierung eines Werkzeuges zur syntaktischen und semantischen Analyse von TIPP-Prozessen. Studienarbeit, Universität Erlangen-Nürnberg, IMMD VII, Februar 1995.

[Trivedi und Malhotra, 1993] K.S. Trivedi and M. Malhotra. Reliability and Performability Techniques and Tools: A Survey. In B. Walke and O. Spaniol, editors, *Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen*, pages 27–48, Aachen, September 1993. Springer.

[Trivedi, 1982] Kishor S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Prentice-Hall, 1982.

[Varga, 1962] R.S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, 1962.

[Veran und Potier, 1984] M. Veran and D. Potier. QNAP2: A Portable Environment for Queueing Systems Modelling. In *Proceedings of the First International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Paris, May 1984.