Kombination von stochastischen Prozeßalgebren mit Phasentyp–Verteilungen

Diplomarbeit

von

M. Gajewsky

Betreuung

Dr.-Ing. Reinhard German   Prof. Dr.-Ing. Günter Hommel

Fachgebiet Prozeßdatenverarbeitung und Robotik
Institut für Technische Informatik
Fachbereich Informatik
Technische Universität Berlin

Mai 1996

Inhaltsverzeichnis

Kapitel 1

Einleitung

Bei der Bewertung komplexer technischer Systeme, wie z.B. fehlertoleranter Echtzeitsysteme, spielen neben qualitativen auch quantitative Kriterien eine wichtige Rolle. Quantitative Kriterien betreffen vor allem die Leistungsfähigkeit und die Zuverlässigkeit eines Systems. Die Bewertung dieser quantitativen Kriterien kann entweder durch Messung an einem realen System oder aber modellgestützt erfolgen. Dabei bieten modellgestützte Ansätze gegenüber Messungen den wesentlichen Vorteil, daß hier Aussagen über noch nicht fertiggestellte Systeme getroffen werden können. Die gewonnenen Ergebnisse können dann schon bei der Erstellung solcher Systeme Eingang finden, und somit eventuelle Korrekturen in einem frühen Stadium des Software–Entwurfs vorgenommen werden. Die Auswertung solcher Modelle erfolgt entweder durch eine numerische Analyse, d.h. der Lösung eines Gleichungssystems, oder durch Simulation, d.h. Abschätzung der quantitativen Größen des modellierten Systems anhand von simulativen Durchläufen. Welche dieser beiden Auswertungsmöglichkeiten verwandt werden kann, hängt von dem Modell ab. Formalismen, die für beide Methoden geeignet sind, sind einerseits *stochastische Petri–Netze* (SPN), andererseits lassen sich aber auch *stochastische Prozeßalgebren* (SPA) sowohl durch Analyse wie auch Simulation auswerten.

Da Prozeßalgebren im Vergleich zu Petri–Netzen bessere Eigenschaften bezüglich Kompositionalität bieten[1], besteht die Hoffnung, daß sich diese auch auf stochastische Prozeßalgebren übertragen lassen. Dies ist die Motivation, zuerst einmal die grundliegende Theorie von SPA weiterzuentwickeln, obwohl die Theorie der SPN zur Zeit weiter ausgebaut und damit besser für praktische Anwendungen geeignet

[1] Der Box–Kalkül für Petri–Netze von E. Best [BDH92],[BFF$^+$95] gestattet die Verknüpfung und damit evtl. Modularisierung von Netzen

ist.

Die Theorie von SPA[2] wurde in den letzten Jahren auf der Grundlage klassischer Prozeßalgebren wie *Calculus of Communicating Systems* (CCS) von Milner [Mil89] und *Communicating Sequential Processes* (CSP) von Hoare [Hoa85] entwickelt. Das Ziel ist die Erweiterung dieser Formalismen um stochastische und zeitliche Aspekte, um diese für die modellbasierte Leistungs– und Zuverlässigkeitsbewertung einzusetzen. In den meisten stochastischen Prozeßalgebren wird die Dauer von Aktionen durch eine Exponentialverteilung beschrieben. So läßt sich die einem Modell zugrundeliegende operationale Semantik auf eine Markow–Kette abbilden und dann mit numerischen Standardverfahren analysieren.
Die Annahme der Exponentialverteilung für die Dauer einer Aktion ist oft unrealistisch. Eine Möglichkeit zur Beschreibung anderer Verteilungen ist durch die Klasse der Phasentypverteilungen gegeben, die zudem jede Verteilung beliebig genau approximieren können. Ein wichtiger Spezialfall ist die Erlang–Verteilung, die aus einer Sequenz von Phasen gleicher Dauer besteht. Sie wird zur Approximation von deterministischen Zeiten verwandt.

Aus diesem Grund stellt die hier vorgestellte Erweiterung des Formalismus der SPA um die Klasse von Phasentypverteilungen einen sinnvollen Ausbau der Theorie dar. Dieser Ansatz erlaubt es, beliebiges zeitliches Verhalten nachzubilden bei gleichzeitig einfacher Analysierbarkeit. Damit kann das zeitliche Verhalten eines Systems zugleich realistischer modelliert wie auch mit Standardmethoden analysiert werden. Folglich liefert das Modell zum einen aussagekräftigere Analyseergebnisse im Hinblick auf das reale System, als es in den meisten bisherigen stochastischen Erweiterungen von Prozeßalgebren möglich ist. Zum anderen können Standardverfahren bei der Auswertung eingesetzt werden. In dieser Arbeit stützt sich die Erweiterung auf eine SPA, in der nur Exponentialverteilungen zugelassen sind. Diese Grundlage ist nicht nur im Hinblick auf die Verwendung bekannter Formalismen sinnvoll, sondern auch in mathematischer Hinsicht. Phasentypverteilungen sind komplexe Verteilungen, die sich aus Exponentialverteilungen zusammensetzen.

Die Möglichkeit der anschaulichen Repräsentation dieser Verteilungen als Graphen sowie auch die enge Verwandtschaft von Graphen mit stochastischen Formalismen legt dabei die explizite Nutzung von Graphen und Graphgrammatiken nahe. Des-

[2] Siehe [Hil93], [Buc94], [BDG94] und [HHM95]

halb wird zunächst im Falle der Exponentialverteilungen ein Bestandteil der SPA durch Graphgrammatiken ersetzt. Darauf aufbauend wird durch eine Erweiterung der Graphregelmenge der Ausbau um Phasentypverteilungen realisiert. Diese Vorgehensweise hat zudem den Effekt, daß sowohl das Verständnis als auch die algorithmische Umsetzung der Erweiterung erleichtert wird.

Da Phasentypverteilungen bereits in SPN integriert sind, werden stochastische Petri–Netze zur Überprüfung der Plausibilität der SPA–Erweiterung herangezogen. Bei gegebenen SPN– und SPA–Modellen desselben Systems lassen sich Analyseergebnisse in den beiden Formalismen vergleichen und gestatten so Rückschlüsse auf die Adäquatheit der Modellierung wie auch auf die korrekte Vorgehensweise in dem entsprechenden Formalismus. Gerade bei der Erweiterung von SPA können so stochastische Petri–Netze als Referenz genutzt werden:
Für verschiedene Teilschritte der SPA–Erweiterung wird daher ein Vergleich mit einem analogen SPN–Modell einer gepufferten Warteschlange angestellt und ihre zugrundeliegende Markow–Kette miteinander in Beziehung gesetzt. Eine kurze Einführung in stochastische Petri–Netze ist im Anhang zu finden.

1.1 Einleitendes Beispiel mit Petri–Netzen

Ein zentrales Forschungsgebiet des Operation Research ist die *Warteschlangentheorie* (siehe auch [Kle75], [Ger95]). Eine Warteschlange modelliert ein System, in dem ankommende Aufträge von Bedieneinheiten (Servern) bearbeitet werden. Dabei spielen sowohl die Zwischenankunftszeiten von Aufträgen (A), die Bedienzeiten durch die Bedieneinheit (S) und die Anzahl der verfügbaren Server (m) eine entscheidende Rolle. Andere Faktoren sind die Pufferkapazität des Systems (B), die maximale Anzahl von Aufträgen (K) und die Bedienstrategie (SD). Solche Warteschlangen notiert man in der Reihenfolge „A/S/m/B/K/SD“, wobei eine konkrete Warteschlange durch konkrete Werte für die Parameter gegeben ist. „A“ und „S“ sind die wahrscheinlichkeitstheoretischen Beschreibungen des Ankunftsprozesses der Aufträge (Arrivalprocess) respektive die Verteilung der Bedienzeit (Serviceprocess). „m“, „B“ und „K“ sind natürliche Zahlen für die Anzahl der Bedieneinheiten, die Pufferkapazität und die maximale Anzahl von Aufträgen, „SD“ beschreibt eine Strategie wie z.B. „first–come, first–served“(FCFS). In der Warteschlangentheorie wurden hierzu viele Untersuchungen in Bezug auf die Auswirkungen verschiedener

Parameter auf das Verhalten solcher Systeme angestellt, die in der entsprechenden Literatur zu finden sind.

Warteschlangen können auch innerhalb des Formalismus der SPN formuliert werden. Als laufendes Beispiel wird in dieser Arbeit eine *M/D/1/K/K/FCFS–Warteschlange* modelliert, d.h. die Zwischenankunftszeiten haben eine Dauer, die durch eine Exponentialverteilung (M) gegeben ist, die Bedienzeiten sind deterministisch (D). Es gibt einen verfügbaren Server und K Plätze im Puffer für ankommende Aufträge. Falls diese Kapazität überschritten wird, gehen Aufträge verloren und werden somit nicht bearbeitet. Ist noch Kapazität frei, wird der angekommene Auftrag in eine Liste eingereiht (FCFS) und wartet auf seine Bearbeitung. Das Petri–Netz in Abbildung 1.1 bildet diese Konstellation nach.

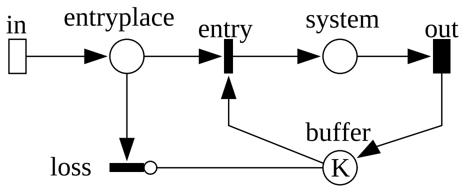

Abbildung 1.1: Petri–Netz–Modell einer Warteschlange

Dabei repräsentiert die Stelle *entryplace* die Ankunft eines Auftrags, der mit einer exponentiell verteilten Zwischenzeit (*in*) ankommt. Die Stelle *system* modelliert die Liste, in die der Auftrag sofort eingereiht wird (*entry*), falls noch genügend Kapazität im Puffer ($buffer$) vorhanden ist. Anderenfalls geht der Auftrag umgehend verloren (*loss*). Ein eingereihter Auftrag wird in einer deterministischen Zeit (*out*) bearbeitet, wodurch wiederum Kapazität frei wird.

1.2 Beschreibung der Kapitel

Die Kapitel 2, 3 und 4 geben jeweils eine kurze Einführung in die verwendeten Formalismen. Dabei werden in Kapitel 2 Grundlagenkenntnisse der Stochastik vorausgesetzt. Darauf aufbauend werden dort Phasentyp–Verteilungen eingeführt. Dabei wird in besonderem Maße auf ihre graphische Repräsentation eingegangen.

In Kapitel 3 wird eine stochastische Prozeßalgebra vorgestellt. Zu ihrer Grundlage gehören Prozeßalgebren, die in der Einleitung des Kapitels informell beschrieben werden. Die Modellierung eines Systems innerhalb dieses Formalismus erfolgt durch eine abstrakte Sprache, sein Verhalten wird durch die operationale Semantik beschrieben. Anhand einer Beispielmodellierung werden die vorgestellten Konzepte veranschaulicht. Zudem wird das Eingangsbeispiel – das Petri–Netz–Modell einer Warteschlange – übersetzt in einen stochastischen Prozeß, der das gleiche zeitliche Verhalten hat.

Eine Möglichkeit der einheitlichen Behandlung von Phasentyp–Verteilungen und der Repräsentation des Verhaltens eines stochastischen Prozesses ist durch Graphgrammatiken gegeben. In Kapitel 4 wird dieser Zusammenhang motiviert und eine kurze Einführung in den Formalismus gegeben.

Die Integration von stochastischen Prozeßalgebren und Graphgrammatiken findet in Kapitel 5 statt. Dort wird ein Bestandteil der stochastischen Prozeßalgebren ersetzt durch eine Menge von Graphregeln. Dabei bleibt die operationale Semantik eines Prozesses invariant unter dieser Ersetzung, was am Beispiel verdeutlicht wird.

Das Kapitel 6 ist das Kernstück dieser Arbeit. Dort werden stochastische Prozeßalgebren um Phasentyp–Verteilungen erweitert. Dazu erfolgt zunächst ein Überblick über verschiedene Strategien im Hinblick auf die Ausführung von komplexen Aktionen. Diesen Strategien folgend werden dann Graphregeln zur Ableitung des Verhaltens eines Prozesses angegeben. Wiederum wird am Beispiel gezeigt, welchen Effekt diese Regeln haben.

Ein weiterer Ausbau der stochastischen Prozeßalgebren wird in Kapitel 7 vorgenommen. Dort wird das Verhalten eines Prozesses mit zeitlosen Aktionen spezifiziert.

Ein komplexeres Beispiel wird in Kapitel 8 vorgestellt: Ein Tokenring. Dieser wird einerseits mit stochastischen Petri–Netzen, andererseits mit stochastischen Prozeßalgebren modelliert.

Kapitel 9 gibt eine Zusammenfassung der gewonnenen Ergebnisse. Darüberhinaus werden offene Fragen formuliert, die sich als Konsequenz aus der Erweiterung ergeben.

Da einige der Beispiele auch als stochastische Petri–Netze modelliert und im Laufe dieser Arbeit als Referenz genutzt werden, befindet sich im Anhang A die Definition dieser Netze und ihres Schaltverhaltens.

Kapitel 2

Phasentypverteilungen

Phasentypverteilungen (PH–Verteilungen)[1] sind eine Klasse von Wahrscheinlichkeitsverteilungen, die jede beliebige allgemeine Verteilung annähern können. Zugleich sind sie auf eine Markow–Kette abbildbar, was die Analyse dieser Verteilung erheblich vereinfacht. Im Zusammenhang mit dieser Arbeit stellen sie gerade die Klasse von Wahrscheinlichkeitsverteilungen dar, um die stochastische Prozeßalgebren erweitert werden sollen. Dieser Ansatz erlaubt, das zeitliche Verhalten von Aktionen mit einer allgemeinen Verteilung durch eine PH–Verteilung zu approximieren bei gleichzeitiger Verwendbarkeit von Standardanalysemethoden.

Die Grundbegriffe der Wahrscheinlichkeitstheorie wie *Zufallsexperiment, Elementarereignis, Ereignisraum, Wahrscheinlichkeit, Zufallsvariable, Verteilungsfunktion* usw. (siehe [Fel66]) werden in dieser Arbeit vorausgesetzt. Deshalb wird im folgenden nur eine kurze Beschreibung von Phasentypverteilungen und ihrer graphischen Darstellung gegeben. Beispielhaft werden zwei Verteilungen vorgestellt, die sich als PH–Verteilungen darstellen lassen.

2.1 Mathematische Definition

Eine Familie $(X(t))_{t \in I}$ von Zufallsvariablen, d.h. eine Zufallsvariable für jeden Wert der Indexvariablen t, wird als *stochastischer Prozeß* bezeichnet. Er erlaubt die Modellierung eines dynamischen Ablaufgeschehens. Im Modellzusammenhang wird der Wert der Zufallsvariablen als Zustand und der Indexwert als Zeit interpretiert, wobei beide Werte diskret oder kontinuierlich sein können. Stochastische Prozesse,

[1] Siehe [Neu89], Seite 231 bis 248

deren zukünftiger Verlauf nur vom aktuellen Zustand abhängt, bezeichnet man als Markow–Prozesse[2]. Die einzige sog. „gedächtnislose" zeitkontinuierliche Verteilung ($t \in I\!R^+$), die dieser Anforderung genügt, ist die *Exponentialverteilung*. Dabei ist die Wahrscheinlichkeit $P(X \leq t)$, daß die Zufallsvariable X einen Wert t nicht überschreitet, durch eine Exponentialfunktion $F(t)$ gegeben.

Definition 2.1.1 (Exponentialverteilung)
Sei $\lambda \in I\!R^+$, S ein Ereignisraum und $X : S \to I\!R$ eine Zufallsvariable, dann wird eine Exponentialverteilung durch folgende Zuordnung beschrieben:

$$P(X \leq t) = F(t) = \begin{cases} 1 - e^{-\lambda t} & t \geq 0, \\ 0 & t < 0 \end{cases}$$

□

Beispiel einer Markow–Kette aus [Ger95]

Betrachtet man ein redundantes Zweikomponentensystem mit identischen Komponenten S_1 und S_2, so kann man vier verschiedene Zustände dieses Systems (Abbildung 2.1) unterscheiden.

1. Beide Komponenten sind intakt (Zustand :„1,2 i").
2. Die Komponente S_1 ist ausgefallen, die andere intakt (Zustand :„2 i").
3. Die Komponente S_2 ist ausgefallen, die andere intakt (Zustand :„1 i").
4. Beide Komponenten sind ausgefallen (Zustand :„0 i").

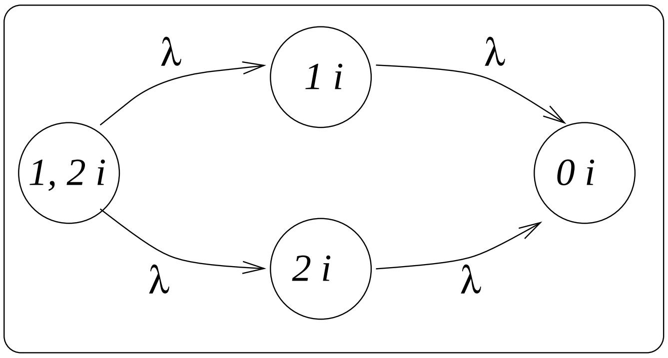

Abbildung 2.1: Markow–Kette eines redundanten Zweikomponentensystems

[2]Markow–Prozesse mit einem diskreten Zustandsraum werden auch *Markow–Ketten* genannt.

Angenommen, die Lebensdauer beider Komponenten ist gegeben durch eine Exponentialverteilung mit der Rate λ. Dann können sie als Zufallsvariablen X_1, X_2 mit Verteilung:

$$F_{X_1}(t) = F_{X_2}(t) = 1 - e^{-\lambda t}$$

modelliert werden. Der dazugehörige Zustandsgraph (Markow–Kette) ist in Abbildung 2.1 zu sehen. Aus der Markow–Kette kann nun die Generatormatrix $\mathbf{Q}$ ermittelt werden, die die Grundlage für die stationäre und transiente Analyse des Systems ist: Sie ist quadratisch mit der Dimension $m \times m$, wenn die Markow–Kette m Zustände (Knoten) hat. Das Element $q_{i,j}$ wird bestimmt durch die Rate des Zustandsübergangs vom Zustand i in den Zustand j für $i \neq j$. Die Elemente der Hauptdiagonale $q_{i,i}$ sind als negative Summe der sonstigen Einträge dieser Zeile i gegeben.
In diesem Fall hat sie also folgende Gestalt:

$$\mathbf{Q} = \begin{pmatrix} -2\lambda & \lambda & \lambda & 0 \\ 0 & -\lambda & 0 & \lambda \\ 0 & 0 & -\lambda & \lambda \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Wie schon aus der Problemstellung leicht folgt, ist der Zustand „0 i“ derjenige, der — ließe man das System ohne Eingriff von außen unendlich lange laufen —mit Sicherheit eintritt. Danach ist das System (ohne Eingriff von außen) nicht mehr in der Lage, seinen Zustand zu verändern. Einen solchen Zustand nennt man *absorbierend*. Nicht–absorbierende Zustände heißen dagegen *transient*. Das redundante Zweikomponentensystem besteht demnach aus drei transienten und einem absorbierenden Zustand. Zustandsübergänge haben eine Dauer, die durch eine Exponentialverteilung gegeben ist. Insofern erfüllt dieses System die folgenden Bedingungen an eine Phasentypverteilung und ist so auch als ein System modellierbar, dem eine solche Phasentypverteilung zugrundeliegt.

Sei ein Markow–Prozeß mit einem endlichen Zustandsraum $\{1, \ldots, m+1\}$ gegeben. Die Zustände 1 bis m seien transient, d.h. sie werden nach Ablauf einer gewissen Zeit wieder verlassen, der Zustand $m+1$ sei absorbierend. Eine Zufallsvariable X hat eine Phasentypverteilung $F(\cdot)$, falls X als die Zeit angesehen werden kann, bis der absorbierende Zustand in dem Markow–Prozeß erreicht ist.

Definition 2.1.2 (Phasentypverteilung (nach [Neu89]))
Eine kontinuierliche Wahrscheinlichsverteilung $F(\cdot)$ auf $[0, \infty)$ heißt Phasentypverteilung, falls sie die Verteilung der Zeit ist bis zur Absorption in einem Markow–Prozeß mit einem einzigen absorbierenden Zustand. Das heißt, es existiert ein Wahrscheinlichkeitsvektor (α, α_{m+1}) und eine Generatormatrix der Form

$$\mathbf{Q} = \begin{bmatrix} \mathbf{T} & t_0 \\ 0 & 0 \end{bmatrix}$$

so daß

$$F(x) = 1 - \alpha \cdot e^{\mathbf{T}x} \cdot \mathbf{e}, \text{ für } x \geq 0$$

gilt, wobei der Vektor $\mathbf{e}$ der m-dimensionale Spaltenvektor ist, dessen Einträge alle „1“ sind. Zusätzlich sollen die folgenden Bedingungen erfüllt sein:

1. Der Wahrscheinlichkeitvektor $(\alpha, \alpha_{m+1}) = ((\alpha_1, \dots, \alpha_m), \alpha_{m+1})$ erfüllt die Gleichung: $\sum_{i=1}^{m+1} \alpha_i = 1$. Dieser Vektor enthält die Wahrscheinlichkeiten α_i, mit der der Markow–Prozeß in der i–ten Phase beginnt. Sie heißen daher auch *Eingangswahrscheinlichkeiten*.

2. Die Generatormatrix $\mathbf{Q}$, resp. ihre Untermatrix $\mathbf{T}$ müssen dabei mehrere Bedingungen erfüllen für alle $1 \leq i, j \leq m + 1$:

   (a) $\mathbf{T}$ ist regulär, d.h. invertierbar.

   (b) Die Hauptdiagonalelemente der Untermatrix $\mathbf{T}$ sind negativ: $\mathbf{T}_{ii} < 0$

   (c) Sonstige Elemente sind positiv: $q_{ij} \geq 0$ für $i \neq j$

   (d) Die Zeilensumme von $\mathbf{Q}$ ist Null: $\sum_{j=1}^{m+1} q_{ij} = 0$. Diese Eigenschaft wird auch als *Normierungseigenschaft* bezeichnet.

□

Den Spaltenvektor t_0 nennt man auch Ausgangsratenvektor, da er die Raten der Exponentialverteilungen enthält, die in den absorbierenden Zustand führen. Die transienten Zustände werden auch als *Phase* bezeichnet.
Die PH–Verteilung wird durch die Matrix $\mathbf{T}$ und den Wahrscheinlichkeitsvektor α eindeutig repräsentiert und als Paar $(\mathbf{T}, \alpha)$ notiert. Wegen der Normierungseigenschaften sowohl der Generatormatrix als auch des Wahrscheinlichkeitsvektors lassen sich alle anderen Größen berechnen durch: $\alpha_{m+1} = 1 - \sum_{i=1}^{m} \alpha_i$ und $t_{0i} = -\sum_{j=1}^{m} \mathbf{T}_{ij}$.

Demnach sind PH–Verteilungen komplexe Verteilungen, die sich aus Exponentialverteilungen zusammensetzen. Die Gedächtnislosigkeit der Exponentialverteilungen ermöglicht es, auch PH–Verteilungen auf eine zeitkontinuierliche Markow–Kette abzubilden.

2.2 Repräsentation

In Umkehrung des oben beschriebenen Verfahrens, aus der graphischen Repräsentation eines Systems die Generatormatrix $\mathbf{Q}$ zu ermitteln, kann auch aus der Repräsentation einer PH–Verteilungen durch die Matrix $\mathbf{T}$ und den Wahrscheinlichkeitsvektor α die graphische Darstellung erschlossen werden.
Sei eine PH–Verteilung durch $(\mathbf{T}, \alpha)$ mit $\dim \mathbf{T} = m$ gegeben. Der Graph G mit der Knotenmenge V und Kantenmenge E, der diese Verteilung darstellt, besteht dann aus $m+2$ Knoten, $V = \{K_0, K_1, \ldots, K_{m+1}\}$. Die Knoten K_i für $i \neq 0$ repräsentieren die Zustände. Eine Kante zwischen den Knoten K_i und K_j (für $1 \leq i, j \leq m+1$) existiert, falls $q_{ij} \neq 0$ und wird in diesem Fall mit der Rate q_{ij} beschriftet. Ihre Interpretation ist ein Zustandsübergang vom i–ten in den j-ten Zustand mit der Rate q_{ij}. Die Eingangswahrscheinlichkeiten α_i von K_0 in den i-ten Zustand werden mit gepunkteten Kanten notiert: dabei werden nur Wahrscheinlichkeitskanten gezeichnet, für die $\alpha_i \neq 0$ ist. α_{m+1} errechnet sich aus $1 - \sum_{j=1}^{m} \alpha_j$. Der Knoten K_0 dient in diesem Falle also nur dazu, keine „hängenden“ Kanten ohne Quelle für die Eingangswahrscheinlichkeiten zu haben.

Ein bedeutendes Beispiel für Phasentypverteilungen ist die k–phasige Erlang–Verteilung ($\mathbf{E}_k$). Sie setzt sich aus einer Sequenz von Exponentialverteilungen mit der Rate λ zusammen. Ihre Repräsentation ist gegeben durch den Wahrscheinlichkeitsvektor

$$\alpha = \underbrace{(1, 0, \ldots, 0)}_{k}$$

und der Matrix $\mathbf{T}$ der Dimension $k \times k$ aus Abbildung 2.2
Die Besonderheit der Erlang–Verteilung liegt darin, daß sie eine deterministische Schaltzeit approximiert. Die k–phasige Erlang–Verteilung mit der Rate $k \cdot \mu$ für den Übergang zwischen zwei Phasen nähert die deterministische Schaltzeit $\frac{1}{\mu}$ bei gegen

$$\mathbf{T} = \begin{bmatrix} -\lambda & \lambda & 0 & \cdots & 0 & 0 \\ 0 & -\lambda & \lambda & \cdots & 0 & 0 \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & -\lambda & \lambda \\ 0 & 0 & 0 & \cdots & 0 & -\lambda \end{bmatrix}$$

Abbildung 2.2: **T**–Matrix der Erlang–Verteilung der Ordnung k ($\mathbf{E}_k$)

„Unendlich" strebendem k an. Das bedeutet, daß der Erwartungswert dieser Verteilung $\frac{1}{\mu}$ ist und die Varianz für k gegen „Unendlich" gegen „Null" strebt. Genauer: Die Varianz errechnet sich (nach [Kle75]) als

$$\sigma(X) = \frac{1}{\sqrt{k} \cdot \mu}$$

Dem oben angegebenen Algorithmus zufolge erhält man die graphische Darstellung dieser Verteilung wie in Abbildung 2.3 gezeigt.

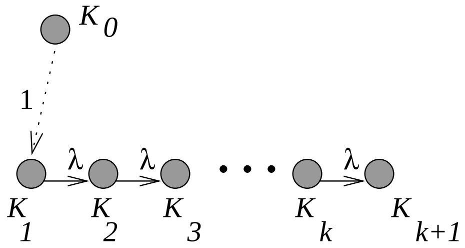

Abbildung 2.3: Graphische Darstellung der Erlang–Verteilung E_k

Ein anderes Beispiel für die Ausdrucksmächtigkeit solcher PH–Verteilungen ist die hyperexponentielle Verteilung H_k der Ordnung k, die ebenfalls durch eine PH–Verteilung dargestellt werden kann:

$$\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_k)$$

und die Matrix der Dimension $k \times k$, die auf den Ausgangsvektor t_0 schließen läßt:

$$\mathbf{T} = \begin{bmatrix} -\lambda_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & -\lambda_2 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & -\lambda_{k-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & -\lambda_k \end{bmatrix}, t_0 = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{k-1} \\ \lambda_k \end{pmatrix}$$

Ihre graphische Entsprechung ist in Abbildung 2.4 zu sehen. Weitere Beispiele sind in [Lan] zu finden.

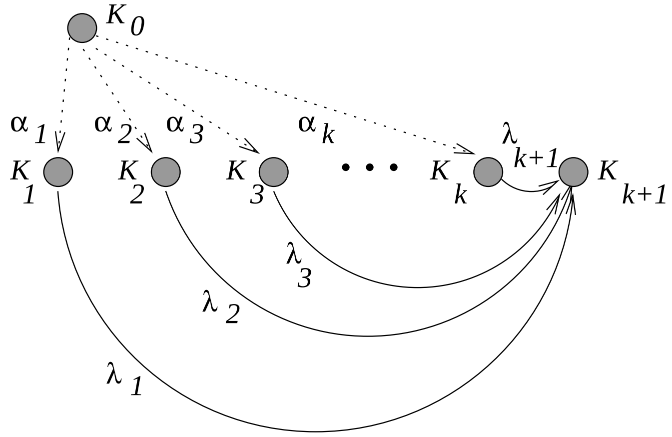

Abbildung 2.4: Graphische Darstellung der hyperexponentiellen Verteilung der Ordnung k

Kapitel 3

Stochastische Prozeßalgebren

In diesem Kapitel wird zunächst die grundlegende Idee von Prozeßalgebren beschrieben. Darauf basierend werden die stochastischen Prozeßalgebren (SPA) anhand einer ausgewählten SPA eingeführt. Am Beispiel eines Modells einer Warteschlange werden die Konzepte erläutert. Dieses Modell wird als Übersetzung eines stochastischen Petri–Netzes (SPN) gewonnen, wobei das Verfahren informell beschrieben wird. Die beiden Modellen zugrundeliegende Markow–Kette ist identisch.

3.1 Einführung in Prozeßalgebren

Prozeßalgebren (PA) sind genau wie Petri–Netze ein Mittel zur Spezifikation und Modellierung nebenläufiger und verteilter Systeme. Eine der ersten PA war der in den achtziger Jahren von Milner entwickelte *Calculus of Communicating Systems (CCS)* [Mil89], und die zugrundeliegende Idee diente im weiteren Verlauf als Grundlage für ähnliche Formalismen. Die Problemstellung ist es, ein komplexes nebenläufiges und verteiltes System zu erfassen, zu modellieren und Aussagen über sein Verhalten zu treffen. Die wesentliche Idee besteht darin, ein solches System als ein Netzwerk von verschiedenen identifizierbaren Teilen (Agenten) zu betrachten, die miteinander kommunizieren können. Diese Sicht führt in natürlicher Weise dazu, daß sich ein System modular zusammensetzt und daher auch modular aufgebaut werden kann. Grundelemente von Agenten sind Aktionen, die entweder interaktiv mit ihrer Umgebung (handshake communication) oder unabhängig und damit nebenläufig (concurrent) ablaufen. Selbst unabhängige Aktionen können als interne Kommunikation angesehen werden, die nach außen unsichtbar (silent) sind.
Milner definiert nun das Verhalten eines Systems als seine Fähigkeit zur Kommu-

nikation mit der Umgebung, d.h. das Verhalten eines Systems ist einzig und allein sein beobachtbares Verhalten. Über diesem Verhalten werden Äquivalenzrelationen definiert, die *starke (und schwache) Bisimulation*, bei der zwei Agenten genau dann äquivalent sind, wenn sie das gleiche beobachtbare Verhalten bezüglich aller (oder nur bezüglich der sichtbaren) Aktionen zeigen. Die Bisimulation ist eine zentrale Komponente in dieser Modellierungsweise, da sie einen Austausch äquivalenter Agenten erlaubt, und zusammen mit der Modularisierbarkeit einen wichtigen Aspekt im Hinblick auf den Softwareentwurf darstellt.

Die Realisierung der grundlegenden Idee geschieht folgendermaßen: PA basieren auf einer formalen textuellen Beschreibung. Genauer ist die Syntax von PA durch eine induktiv erzeugte *abstrakte Sprache* über einem Alphabet von atomaren Aktionen gegeben. Wörter dieser Sprache sind Prozesse (Agenten), die die Fähigkeit haben, untereinander zu kommunizieren und somit zu kooperieren. Komplexe Systeme werden durch Komposition solcher Aktionen gebildet, wobei die Art der Komposition durch die Operatoren der Sprache gegeben ist. Im wesentlichen sind das Sequentialisierung, Parallelisierung mit Synchronisation über eine Aktionenmenge und nicht–deterministische Auswahl. Daneben gibt es einen abstrahierenden Operator, der eine Informations–Kapselung (information–hiding) realisiert.
Die Möglichkeiten eines Prozesses, sich weiterzuentwickeln, d.h. sein Verhalten oder seine *operationale Semantik*, wird mittels eines Regelkalküls abgeleitet. Untersuchungen des modellierten Systems können nun durch Untersuchungen der operationalen Semantik des Prozesses angestellt werden. Dabei sind zwei Prozesse mit der gleichen operationalen Semantik verhaltensäquivalent und somit wechselseitig austauschbar. Daneben kann Verhaltensäquivalenz zweier Agenten schon auf der syntaktischen Ebene durch eine Menge von Gleichungen beschrieben werden. Bei aller Vielfältigkeit verschiedener Varianten, gehört somit zu einer PA immer die Grammatik der abstrakten Sprache zusammen mit dem Kalkül für die operationale Semantik. Außerdem gelten zeichnen sich alle diese Versionen durch Modularisierbarkeit, Begriff der Verhaltensäquivalenz und Austauschbarkeit von äquivalenten Subsystemen aus.
Da die Kenntnis von *CCS* nicht Voraussetzung zum Verständnis der stochastischen PA ist, wird an dieser Stelle auf eine Vorstellung dieses Formalismus verzichtet. Ausführlich beschrieben ist er in [Mil89] zu finden.

Zum Abschluß dieser allgemeinen Einführung noch eine Bemerkung zum Vergleich mit Petri–Netzen: Der induktive Aufbau von Agenten aus Aktionen, hat eine wichtige Implikation auf den Zustandsbegriff: Ein Zustand des Systems ist dadurch gekennzeichnet, welche Aktionen ausgeführt werden können. Also ist der Zustand nicht direkt aus der Spezifikation ersichtlich, oder wie bei Petri–Netzen durch die Markierung gegeben, sondern steckt implizit in der möglichen Weiterentwicklung des Systems.

3.2 Erweiterung um Stochastik

Um neben der funktionalen Analyse, die in Prozeßalgebren durchgeführt werden kann, auch Leistungsbewertung mit diesem Formalismus vornehmen zu können, wurden Prozeßalgebren um Stochastik erweitert. So wird es ermöglicht, die einem System zugrundeliegende Markow–Kette zu finden und zu analysieren. Dies erfordert eine Erweiterung des Begriffes der Bisimulation, in die die temporalen Aspekte mit eingeflochten werden müssen: Die *Markow'sche Bisimulation*. Ihre Definition hängt wesentlich von der spezifischen Erweiterung ab, die in verschiedenen Versionen stochastischer Prozeßalgebren leicht unterschiedlich ist.
Der Formalismus, der hier beispielhaft erläutert werden soll, stammt von der Erlanger Gruppe um Prof. Herzog [HHM95], [GHR93], [GHR94]. Sie entwickelte eine Familie von stochastischen Prozeßalgebren mit dem Namen *Timed Processes and Performability Evaluation* (TIPP), zu der verschiedene Ausprägungen, bzw. Restriktionen erforscht wurden:

1. TIPP befaßt sich mit allgemeinen Verteilungen.
2. Markovian TIPP dagegen läßt nur Aktionen zu, deren Dauer durch eine Exponentialverteilung gegeben ist.
3. Eine andere Variante wird um den *probabilistic choice operator* erweitert.
4. *Zeitlose Aktionen* mit einer eingeschränkten Operatormenge werden in [HRW95] untersucht.

Somit bietet TIPP eine große Vielseitigkeit, wobei sich bei Kenntnis des Basisformalismus Ergebnisse in verschiedenen Erweiterungen leichter nachvollziehen lassen

als in voneinander unabhängigen Formalismen. Von besonderem Interesse ist TIPP im Hinblick auf die Behandlung zeitloser Aktionen, da sie eine der wenigen SPA ist, die solche Aktionen betrachtet.
Weiterhin bietet TIPP den Vorteil, daß die zugrundeliegende Markow–Kette direkt aus der operationalen Semantik eines Prozesses abgeleitet werden kann[1]. Die Handhabung der Raten von Exponentialverteilungen bei Synchronisation ist weniger restriktiv als in anderen stochastischen Erweiterungen und erlaubt, unterschiedliches Antwortverhalten eines Systems zu modellieren. Außerdem wurden hier Prozeßgleichungen (Kommutativität von $+$ und $||$, Expansion von $||$) aufgestellt, die es ermöglichen, die Markow'sche Bisimularität auf der syntaktischen Ebene zu erkennen.

3.2.1 MTIPP – eine Stochastische Prozeßalgebra

Die Grundelemente von *Markovian Timed Process and Performance Evaluation* (MTIPP) als einer speziellen Version von Prozeßalgebren sind zunächst Aktionen und Operatoren zur Zusammensetzung von Prozessen aus Aktionen. Aktionen beschreiben die relevanten Aktivitäten des Systems und werden als atomar angesehen. Aktionen können aktiv sein oder passiv. Eine Aktion a, deren Dauer exponential verteilt mit der Rate λ ist, wird notiert mit (a, λ). Passive Aktionen $(a, 1)$ warten auf Kommunikationswünsche ihrer Umgebung. Es gibt eine ausgezeichnete Aktion τ, die nach außen funktional unsichtbar ist und somit interne Aktivitäten eines Systems darstellt. Allerdings ist sie von außen als Verzögerung entsprechend der dazugehörigen Rate bemerkbar. Die Menge der Aktionen Act setzt sich zusammen aus regulären Aktionen (Com) und der unsichtbaren (silent) Aktion τ, also $Act = Com \uplus \{\tau\}$
Mit Hilfe von Operatoren können kleinere Systeme zu komplexeren zusammengesetzt werden. Formal wird dies durch eine Grammatik beschrieben.

[1] Die operationale Semantik stimmt mit der Markow–Kette bis auf Mehrfachkanten und Schlingen überein.

Grammatik der abstrakten stochastischen Prozeß–Sprache

Seien Com, und Var gegeben. Die Menge $\mathcal{L}$ von wohlgeformten stochastischen Prozeßbeschreibungen, kurz Prozesse, ist durch die Menge der Wörter zu folgender Grammatik gegeben, wobei $a \in Act$, $\lambda \in I\!R^+$, $S \subseteq Com, X \in Var, (Act = Com \uplus \{\tau\})$

$$P := Stop \mid (a, \lambda).P \mid P + P \mid P||_S P \mid P\backslash a \mid X \mid recX : P$$

Die intuitive Bedeutung dieser Operatoren ist die folgende:

- $Stop$ ist ein Prozeß, der keine Aktionen ausführt.
  (Termination)
- $(a, \lambda).P$ ist ein Prozeß, der zunächst die Aktion a mit Rate λ ausführt und sich dann verhält wie der Prozeß P.
  (Sequenz)
- $P+Q$ verhält sich entweder wie P oder wie Q, wobei die Wahl nicht von außen beeinflußt werden kann.
  (Nichtdeterminstische Auswahl)
- $P||_S Q$ führt nebenläufig die Prozesse P und Q aus, die sich aber bei Aktionen aus S synchronisieren müssen.
  (Parallelität)
- $P\backslash a$ verhält sich genau wie P, außer daß die Aktion a nach außen unsichtbar ist.
  (Funktionale Abstraktion)
- X ist eine Prozeßvariable und dient der Benennung von Prozessen.
- $recX : P$ steht für $X := P$, wobei X frei[2] in P vorkommt. Im folgenden werden wir diesen Operator oft alternativ in der zweiten Fassung notieren.
  (Rekursion)

Als kleines Beispiel modellieren wir eine M/M/1/1/1/FCFS–Warteschlange mit Ausfall und Reparatur. Aufgrund der Restriktion in MTIPP ist die Dauer aller

[2]Eine Variable X ist frei in einem Wort P, wenn sie außerhalb von allen Subprozessen der Form $recX : B$ vorkommt.

Aktionen exponentiell verteilt.
Aufträge kommen mit einer exponentiell verteilten Zwischenankunftszeit in das System und werden wiederum mit einer exponentiell verteilten Dauer von einem Server bearbeitet. Solange ein Auftrag bearbeitet wird, werden keine weiteren angenommen. Der Server wiederum kann in untätigen Phasen mit einer Wahrscheinlichkeit ausfallen, die ihrerseits durch eine Exponentialfunktion gegeben ist. In diesem Fall wird zuerst der Fehler ermittelt, danach wird die Bedieneinheit mit einer exponentiell verteilten Dauer repariert. Nach der Reparatur kann der Server erneut Aufträge annehmen oder wieder ausfallen.
In Abbildung 3.1 ist das Modell dieses Systems zu sehen. Der Name des Ankunftsprozesses ist „$Poisson$“, der der Bedieneinheit „$Server$“.

$$
\begin{array}{lcl}
Poisson & := & (in, \lambda).Poisson \\
\text{(bzw. alternativ} & & recPoisson : (in, \lambda).Poisson \text{)} \\
\hline
Server & := & (in, 1).(out, \mu).Server + (fail, \nu).Repair \\
Repair & := & (diagnose, \delta_1).(repair, \delta_2).Server
\end{array}
$$

$$M/M/1/1/1/FCFS - System := Poisson||_{\{in\}}Server$$

Abbildung 3.1: M/M/1/1/1/FCFS–System mit Ausfall und Reparatur

Operationale Semantik

Die Repräsentation des Verhaltens eines Prozeßes, der durch ein Wort zu obiger Grammatik beschrieben wird, wird nun durch ein *labelled transition system* (LTS) erzeugt. Üblicherweise ist ein LTS über einer Labelmenge Lab eine Struktur

$$(S, s_0, \rightarrow)$$

mit einer Menge S von Zuständen, dem initialen Zustand s_0 und einer Relation $\rightarrow \subseteq S \times Lab \times S$, die die Menge der Übergänge beschreibt. Dabei bedeutet $(s, l, s') \in \rightarrow$ einen Übergang von s nach s' mit der Kennzeichnung l (label) und wird notiert als $s \xrightarrow{l} s'$.
Die operationale Semantik eines Prozesses P in MTIPP wird als das folgende LTS

$$\mathcal{T}(P) = (\mathcal{L}, P, \rightarrow),$$

definiert, wobei die Menge der Zustände die Menge $\mathcal{L}$ aller Wörter zur Grammatik ist, der initiale Zustand ist P selbst und die Übergänge werden durch die Ableitungsregeln in Abbildung 3.2 definiert[3]. Die Kante wird mit dem Aktionsnamen, der Rate und einer Zeichenkette w beschriftet, die aus der Menge der Wörter $w \in W^* = \{+_l, +_r, ||_l, ||_r, ||, \backslash_{yes}, \backslash_{no}\}^*$ stammt. Jede solche Kante ($P \xrightarrow{a, \lambda, w} Q$) bedeutet, daß sich der Prozeß P durch Ausführen der Aktion a mit Rate λ zum Prozeß Q weiterentwickeln kann. Die Zeichenkette w, die die „Geschichte" der Ableitung beschreibt, hat hierbei mehr technische Bedeutung, die später noch erläutert werden wird. Aus der operationalen Semantik eines Prozesses P kann die ihm zugrundeliegende Markow–Kette direkt abgeleitet werden: Sie stimmt mit dem LTS bis auf Schlingen und Mehrfachkanten überein. Mehrfachkanten werden dabei aufgrund der Additivität der Raten zu einer einzigen verschmolzen, indem die Raten aufsummiert werden. Schlingen gehören nicht zur Markow–Kette.

Die Deduktionsregeln in Abbildung 3.2 sollen folgendermaßen gelesen werden: Im Zähler stehen die Prämissen, im Nenner die Konklusion. Das heißt, zur Ableitung der Konklusion müssen zuerst die Prämissen abgeleitet werden. Bei komplexeren Prozeßen müssen also zunächst die Teilprozesse abgeleitet werden. Rekursiv gliedert man wieder die Teilprozesse auf, bis die prämissenlose Regel $\langle . \rangle$ angewandt werden kann. Anders ausgedrückt, muß der durch die Operatoren zusammengesetzte Prozeß zunächst in seine sequentiellen Teilprozesse zerlegt werden und danach den Regeln folgend die Ableitung gemacht werden[4]. Besondere Beachtung verdient die Regel $\langle || \rangle$: Sie kann so interpretiert werden, daß bei der gemeinsamen Aktion zweier Prozesse, jeder seine spezifische „Eigenart" einbringt. Handelt es sich z.B. um einen Auftragsprozeß und einen Serverprozeß, ist die Rate im Auftrag lesbar als Umfang des zu bewältigenden Auftrags, während sie beim Server als Geschwindigkeit deutbar ist. Die Multiplikation der Raten bedeutet somit, daß die Dauer der gemeinsamen Aktion sowohl abhängt von der Größe des Auftrags als auch von der Arbeitsgeschwindigkeit der Bedieneinheit.

[3] Nach dieser Definition ist also die Zustandsmenge jedes Prozesses gleich und ihre Kardinalität unendlich. Wir sprechen im folgenden von einem endlichen LTS, falls die Menge der Übergänge endlich ist. Auch wird in dieser Arbeit von den „irrelevanten" Zuständen abstrahiert, also solchen, die nicht durch Übergänge aus $\rightarrow$ vom initialen Zustand aus erreichbar sind.

[4] $A\{recX : A/X\}$ bedeutet dabei, daß im Rumpf A der Rekursion jedes Auftreten der Variablen X durch den Ausdruck $recX : A$ ersetzt wird.

$$\frac{}{(a,\lambda).A \xrightarrow{a,\lambda,\epsilon} A}\langle.\rangle$$

$$\frac{A \xrightarrow{a,\lambda,w} A'}{A+B \xrightarrow{a,\lambda,+_l w} A'}\langle +_l \rangle \qquad \frac{B \xrightarrow{a,\lambda,w} B'}{A+B \xrightarrow{a,\lambda,+_r w} B'}\langle +_r \rangle$$

$$\frac{A \xrightarrow{a,\lambda,w} A'}{A\|_S B \xrightarrow{a,\lambda,\|_l w} A'\|_S B}\langle \|_l \rangle \quad (a \notin S) \qquad \frac{B \xrightarrow{a,\lambda,w} B'}{A\|_S B \xrightarrow{a,\lambda,\|_r w} A\|_S B'}\langle \|_r \rangle \quad (a \notin S)$$

$$\frac{A \xrightarrow{a,\lambda,w} A' \quad B \xrightarrow{a,\mu,v} B'}{A\|_S B \xrightarrow{a,\lambda\mu,(w,v)} A'\|_S B'}\langle \| \rangle \quad (a \in S)$$

$$\frac{A \xrightarrow{b,\lambda,w} A'}{A\backslash a \xrightarrow{b,\lambda,w} A'\backslash a}\langle \backslash_{no} \rangle \ (a \neq b) \qquad \frac{A \xrightarrow{a,\lambda,w} A'}{A\backslash a \xrightarrow{\tau,\lambda,w} A'\backslash a}\langle \backslash_{yes} \rangle$$

$$\frac{A\{recX : A/X\} \xrightarrow{a,\lambda,w} A'}{recX : A/X \xrightarrow{a,\lambda,w} A'}\langle rec \rangle$$

Abbildung 3.2: Ableitungsregeln der operationalen Semantik von MTIPP

Am Beispiel der M/M/1/1/1/FCFS–Warteschlange mit Ausfall und Reparatur aus Abbildung 3.1 stellt sich die Frage, welche Aktionen und Folgezustände abgeleitet werden können:

$$\frac{? \qquad ?}{Poisson||_{\{in\}}Server \xrightarrow{?} \quad ?}\langle ? \rangle$$

Diese Frage läßt sich folgendermaßen beantworten, wobei P für $Poisson$, S für $Server$ und R für $Repair$ steht:

$$\frac{\dfrac{\dfrac{}{(in,\lambda).(recP:(in,\lambda).P) \xrightarrow{in,\lambda,\epsilon} recP:(in,\lambda).P}\langle.\rangle}{recP:(in,\lambda).P \xrightarrow{in,\lambda,\epsilon} recP:(in,\lambda).P}\langle rec \rangle \qquad \dfrac{\dfrac{}{(in,1).(out,\mu).S \xrightarrow{in,1,\epsilon} (out,\mu).S}\langle.\rangle}{(in,1).(out,\mu).S+(fail,\nu).R \xrightarrow{in,1,+_l} (out,\mu).S}\langle +_l \rangle}{recP:(in,\lambda).P||_{\{in\}}S \xrightarrow{in,\lambda,(\epsilon,+_l)} recP:(in,\lambda).P||_{\{in\}}(out,\mu).S}\langle \| \rangle$$

Aber auch eine andere Ableitung ist möglich:

$$\dfrac{\dfrac{\dfrac{}{(fail,\nu).R \xrightarrow{fail,\nu,\epsilon} R}\langle . \rangle}{S \xrightarrow{fail,\nu,+_r} R}\langle +_r \rangle}{recP:(in,\lambda).P||_{\{in\}}\, S \xrightarrow{fail,\nu,\|_r +_r} recP:(in,\lambda).P||_{\{in\}}\, R}\langle \|_r \rangle$$

Dieses Verfahren muß nun für alle Aktionen und alle Nachfolger durchgeführt werden. Dabei wird zunächst eine anwendbare Regel gesucht bezüglich des äußersten Operators und der Prozeß in seine Subprozesse zerlegt. Dann versucht man eine Ableitung der Subprozesse herzuleiten. Falls mehrere Regeln anwendbar sind, entscheidet man sich für eine und kann bei weiteren Ableitungen eine der anderen anwendbaren Regeln verwenden. Dieses Verfahren setzt man solange fort, bis man alle Subprozesse in sequentieller Form gefunden hat, die dann prämissenlos abgeleitet werden können.
Das LTS eines Prozesses findet man, indem man dieses Verfahren auf jeden neu abgeleiteten Prozeß anwendet.
Die M/M/1/1/1/FCFS–Warteschlange mit Ausfall und Reparatur hat demnach die operationale Semantik, die in Abbildung 3.3 zu sehen ist, wobei die dritte Komponente der Kantenmarkierung zur Vereinfachung weggelassen wurde.

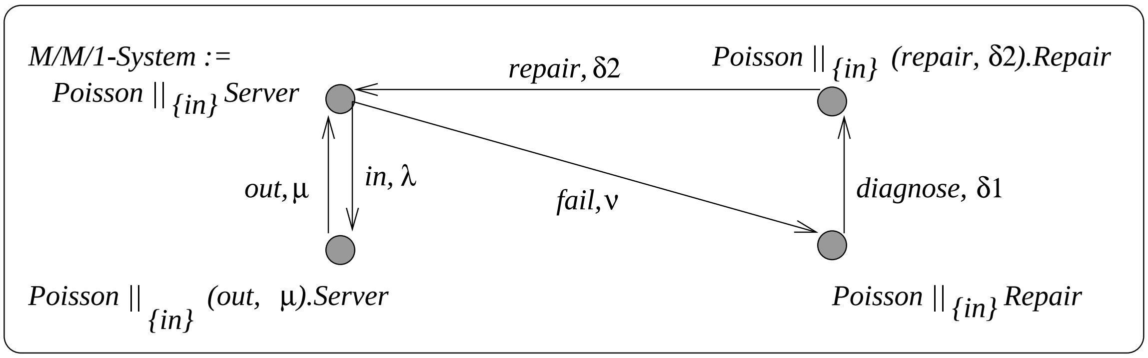

Abbildung 3.3: Operationale Semantik des M/M/1/1/1/FCFS-Systems

Bisimulation

Da SPA zwei Aspekte eines Systems modellieren, nämlich das funktionale wie auch das zeitliche Verhalten, muß der Begriff der Verhaltensäquivalenz, die Bisimulation (siehe Definition 3.2.1.1), im Vergleich zu *CCS* erweitert werden.
Betrachten wir zunächst das funktionale Verhalten, so nennen wir zwei Systeme

äquivalent, wenn sie bei (potentiell unendlicher) Beobachtung von außen in der Abfolge der ausgeführten Aktionen nicht unterscheidbar sind. Formal faßt man dies in folgender Definition:

Definition 3.2.1.1 (Starke Bisimulation ($\sim_F$) aus [HRW95])
Eine symmetrische Relation $\mathcal{R} \subseteq \mathcal{L} \times \mathcal{L}$ über Prozessen ist eine starke Bisimulation falls $(P, Q) \in \mathcal{R}$ impliziert für alle $a \in Act$:

1. $P \xrightarrow{a} P' \implies Q \xrightarrow{a} Q'$ für ein Q' und es gilt $(P', Q') \in \mathcal{R}$
2. $Q \xrightarrow{a} Q' \implies P \xrightarrow{a} P'$ für ein P' und es gilt $(P', Q') \in \mathcal{R}$

□

Bemerkung: Dies ist exakt die Definition von starker Bisimulation, die für *CCS* gilt. Sie wird manchmal auch *funktionale Bisimulation* genannt, da sie nur Aktionen, nicht deren Dauer, zum Gegenstand hat.

Eine mehr informelle Beschreibung von starker Bisimulation kann man folgendermaßen formulieren: Angenommen ein Prozeß P kann sich durch Ausführen der Aktionen $(a_1, a_2, \ldots a_n), n \in I\!N$ zu einem Prozeß P^* entwickeln, ($P \xrightarrow{(a_1, a_2, \ldots a_n)} P^*$). P^* seinerseits kann die Aktion a ausführen ($P^* \xrightarrow{a} P'$). Der Prozeß Q ist dann funktional äquivalent zu P, ($P \sim_F Q$), wenn

1. er auch die Aktionenfolge $(a_1, a_2, \ldots a_n)$ ausführen kann.
2. in jedem Zustand Q^*, den er mit $(a_1, a_2, \ldots a_n)$ erreicht, auch die Aktion a ausführen kann.
3. dies für jede Aktionenfolge (insbesondere die leere) und wechselseitig für P und Q gilt.

Die Definition der Starken Bisimulation vergleicht zwei Prozesse indirekt über ihr zugeordnetes LTS. Einen direkten Vergleich kann man schon auf der syntaktischen Ebene anstellen durch eine Menge von Prozeßgleichungen. Eine vollständige und korrekte Axiomatisierung für die starke Bisimulation ist in [Mil89] angegeben. Vollständigkeit bedeutet in diesem Zusammenhang, daß alle äquivalenten Prozesse entsprechend den Gleichungen identifiziert werden. Korrektheit heißt, daß nur solche

Prozesse identifiziert werden können, die äquivalent sind.

Ausgehend von der funktionalen Bisimulation, sollen nun auch die zeitlichen Aspekte eines Systems erfaßt werden. Es genügt nicht, daß zwei Prozesse die gleichen Aktionen ausführen können, sondern die Rate der Übergänge muß ebenso berücksichtigt werden. Dies wird durch die folgende Definition beschrieben. Wir benötigen dazu eine Funktion γ, die die Summe aller Raten aller Transitionen mit Namen a aus einem Zustand P in eine Menge von Zuständen S liefert.

Definition 3.2.1.2 (Ratensumme)
$\gamma : \mathcal{L} \times Act \times \mathcal{P}(\mathcal{L}) \to I\!R^+$ wird definiert als

$$\gamma(P, a, S) := \sum_{(\lambda,w)\in I} \lambda,$$

mit $I := \{(\lambda, w) | P \xrightarrow{a, \lambda, w} P' \wedge P' \in S\}$ □

Definition 3.2.1.3 (Markow'sche Bisimulation ($\sim_M$) aus [HRW95])
Sei eine Äquivalenzrelation $\mathcal{B} \subseteq \mathcal{L} \times \mathcal{L}$ über der Menge der abstrakten stochastischen Prozesse gegeben. Sei zudem $\mathcal{C} \in \mathcal{L}/_{\mathcal{B}}$ eine beliebige Äquivalenzklasse aus der Quotientenmenge $\mathcal{L}/_{\mathcal{B}}$.
Die Äquivalenzrelation $\mathcal{B}$ ist eine Markow'sche Bisimulation, falls für beliebige Aktionen $a \in Act$ und beliebige Äquivalenzklassen $\mathcal{C}$ gilt:

$$(P, Q) \in \mathcal{B} \text{ impliziert } \gamma(P, a, \mathcal{C}) = \gamma(Q, a, \mathcal{C})$$

Zwei Prozesse P und Q sind Markow-äquivalent, $P \sim_M Q$, falls $(P, Q) \in \mathcal{B}$ für eine beliebige Markow'sche Bisimulation $\mathcal{B}$. □

Informell kann diese Definition folgendermaßen formuliert werden: Angenommen, man hätte eine Menge $\mathcal{A} = \{C_1, \dots, C_k\} (k \in I\!N^+)$ von stochastischen Prozessen, die untereinander Markow–äquivalent sind. Der Prozeß P kann sich durch Ausführung der Aktion a zu Prozessen C_i aus $\mathcal{A}$ weiterentwickeln. Das bedeutet für das LTS, von P führen n Kanten mit der Bezeichnung (a, λ_i, v_i) in die C_i. Analoges gelte für Q: d.h. von dem Prozeß Q gehen m Kanten in beliebige Prozesse C_j mit dem label (a, μ_j, w_j). Identifiziert man nun die Markow–äquivalenten Prozesse zu einem Prozeß $\mathcal{C}$, was der Faktorisierung entspricht mit $\mathcal{C}$ einem Repräsentanten der

Äquivalenzklasse, so führen alle Kanten von P und Q nun in diesen ausgezeichneten Prozeß $\mathcal{C}$. Wegen der Additivität der Raten kann man die Kantenbüschel von P, resp. Q zu einer Kante verschmelzen, wobei die Raten aufsummiert werden. $P \xrightarrow{a, \sum_{i=1}^{n} \lambda_i} \mathcal{C}$ und $Q \xrightarrow{a, \sum_{i=1}^{m} \mu_i} \mathcal{C}$. Gilt nun (für beliebige Aktionen a)

$$\gamma(P, a, \mathcal{A}) = \sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{m} \mu_i = \gamma(Q, a, \mathcal{A})$$

so sind P und Q Markow–äquivalent (unter der Voraussetzung, daß $\mathcal{A}$ eine Äquivalenzklasse ist, d.h. alle zu den C_i Markow–äquivalenten Prozesse enthält).

Bemerkung: Eine wesentliche Eigenschaft beider Bisimulationsbegriffe ist, daß sie nicht nur Äquivalenzrelationen, sondern sogar Kongruenzen sind. Das heißt, falls $P \sim_M P'$ und $Q \sim_M Q'$, dann gilt auch, daß $P \circ Q \sim_M P' \circ Q'$ für beliebige binäre Operatoren $\circ$. Erst durch diese Eigenschaft macht es Sinn, von der Möglichkeit des Austausches äquivalenter Prozesse zu sprechen.
Eine vollständige und korrekte Axiomatisierung für die Markow'sche Bisimulation ist das Gleichungskalkül in Abbildung 3.4 (siehe auch[HHM95]). Die Existenz eines solchen Kalküls stellt eine erhebliche Vereinfachung dar bei der Untersuchung von Prozessen auf Äquivalenz, da sie schon auf der syntaktischen Ebene durchgeführt werden kann.

Die letzte Gleichung $\langle E \rangle$ aus Abbildung 3.4 expandiert einen aus parallelen Komponenten bestehenden Prozeß zu der Summe, also Nichtdeterministischer Auswahl, von Teilprozessen[5]. Das bedeutet auch, daß diesem Formalismus das Konzept der „interleaving concurrency“ zugrundeliegt, in dem Parallelität von Prozessen dadurch ausgedrückt wird, daß sie in beliebiger Reihenfolge sequentialisiert werden können.

Eine letzte Bemerkung zur Zeichenkette w, die zusammen mit dem Aktionennamen und der Rate an abgeleitete Kanten geschrieben wird. Angenommen wir hätten einen Prozeß $P := (a, \lambda).Stop + (a, \lambda).Stop$. P führt die Aktion a mit der Rate 2λ aus. Würde die Zeichenkette, die die „Geschichte“ der Ableitung dokumentiert, nicht notiert werden, wäre das LTS und damit die Markowkette des Prozeßes P

$$P \xrightarrow{a, \lambda} Stop$$

[5]Dabei ist die leere Summe der Prozeß, der nichts tut, also *Stop*.

$$
\begin{array}{|rcll|rcll|}
\hline
P + Stop & = & P & \langle +_0 \rangle & P ||_\emptyset Stop & = & P & \langle \|_0 \rangle \\
P + (Q + R) & = & (P + Q) + R & \langle +_A \rangle & P ||_S (Q ||_S R) & = & (P ||_S Q) ||_S R & \langle \|_A \rangle \\
P + Q & = & Q + P & \langle +_K \rangle & P ||_S Q & = & Q ||_S P & \langle \|_K \rangle \\
\hline
Stop \backslash a & = & Stop & \langle \backslash_0 \rangle & ((b, \lambda).P) \backslash a & = & (b, \lambda).P \backslash a & \langle \backslash_{no} \rangle \\
(P + Q) \backslash a & = & P \backslash a + Q \backslash A & \langle \backslash_+ \rangle & ((a, \lambda).P) \backslash a & = & (\tau, \lambda).P \backslash a & \langle \backslash_{yes} \rangle \\
\hline
\multicolumn{7}{|c}{(a, \lambda).P + (a, \mu).P \quad = \quad (a, \lambda + \mu).P} & \langle \lambda\mu \rangle \\
\hline
P ||_S Q & = & \multicolumn{5}{l}{\sum_{a_i \notin S} (a_i, \lambda_i).(P_i ||_S Q) +} & \\
 & & \multicolumn{5}{l}{\sum_{b_j \notin S} (b_j, \mu_j).(P ||_S Q_i) +} & \\
 & & \multicolumn{5}{l}{\sum_{a_i \in S} \sum_{b_j = a_i} (a_i, \lambda_i \mu_j).(P_i ||_S Q_j)} & \langle E \rangle \\
 & & \multicolumn{5}{c}{\text{für } P \equiv \sum_i (a_i, \lambda_i).P_i \text{ und } Q \equiv \sum_j (b_j, \mu_j).Q_j} & \\
\hline
\end{array}
$$

Abbildung 3.4: Axiomatisierung der Markow'schen Bisimulation

was nicht dem zeitlichen Verhalten entspicht. Mit der Zeichenkette erhält man folgende Semantik

$$P \underset{a, \lambda, +_r}{\overset{a, \lambda, +_l}{\Longrightarrow}} Stop$$

Wegen der Additivität der Raten ist also die zugrundeliegende Markowkette

$$P \xrightarrow{2\lambda} Stop$$

Solange keine derartigen Konflikte vorliegen, werden wir im folgenden diese Zeichenkette zur Vereinfachung weglassen.

3.2.2 Beispiel: Warteschlange

Unter Berücksichtigung der Tatsache, daß an dieser Stelle noch keine deterministischen oder zeitlose Dauer von Aktionen eingeführt sind, wird das Eingangsbeispiel der M/D/1/K/K/FCFS–Warteschlange zunächst vereinfacht, indem alle Transitionen aus dem Petri–Netz durch solche mit exponentialverteilter Schaltzeit ersetzt werden. Um dennoch einen endlichen Erreichbarkeitsgraphen und damit endliche Markow–Kette zu erhalten, wird weiterhin die Tokenzahl auf der Stelle *entryplace* auf 1 beschränkt, was durch die zusätzliche, hemmende Kante realisiert wird.

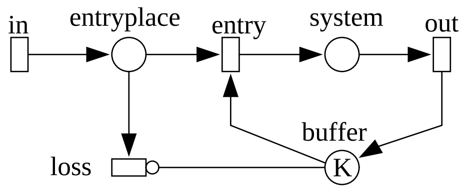

Abbildung 3.5: Warteschlange mit exponentiell verteilten Schaltzeiten

Das Petri–Netz modelliert so ein System, in das Aufträge mit exponentiell verteilter Zwischenankunftszeit eintreffen. Sie warten zunächst eine durch eine Exponentialfunktion gegebene Zeitspanne, bis sie in Abhängigkeit der zur Verfügung stehenden Kapazität ($buffer$) entweder in eine Liste eingereiht werden ($entry$) oder gelöscht werden ($loss$). Während dieser Wartezeit wird kein weiterer Auftrag akzeptiert, was durch die hemmende Kante ausgedrückt wird. Durch den Server werden eingereihte Aufträge mit exponentiell verteilter Dauer abgearbeitet.

$$
\begin{array}{lll l}
Poisson & := & (in,\lambda).Poisson & \\
\hline
Entryplace_0 & := & (in,1).Entryplace_1 & \\
Entryplace_1 & := & (entry,\delta).Entryplace_0 + (loss,\delta).Entryplace_0 & \\
\hline
System_0 & := & (entry,1).System_1 & \\
System_j & := & (entry,1).System_{j+1} + (out,\mu).System_{j-1} & (0 < j) \\
\hline
Buffer_0 & := & (out,1).Buffer_1 + (loss,1).Buffer_0 & \\
Buffer_i & := & (out,1).Buffer_{i+1} + (entry,1).Buffer_{i-1} & (0 < i < K) \\
Buffer_K & := & (entry,1).Buffer_{K-1} &
\end{array}
$$

$$
\begin{aligned}
Warteschlange :=& (Poisson||_{\{in\}}Entryplace_0)\,||_{\{entry,loss\}} \\
& (System_0||_{\{entry,out\}}Buffer_K)
\end{aligned}
$$

Abbildung 3.6: Warteschlange mit exponentiell verteilten Zeiten

Die vereinfachte $Warteschlange$ soll nun im SPA–Formalismus modelliert werden. Dabei ist die Modellierung in Abbildung 3.6 nur eine von vielen möglichen, deren Vorteil aber darin besteht, sehr nahe an dem Petri–Netz–Modell zu sein: jede Stelle ist ein Subprozeß, wobei der Index den Füllstand der Stelle anzeigt. Aktionen entsprechen Transitionen; sie sind passiv in Teilprozessen, bei denen die entsprechende Stelle im Nachbereich dieser Transition liegen, aktiv mit der zur Transition gehörenden Rate, wenn die Stelle im Vorbereich dieser Transition liegt. Hemmende Kanten im Petri–Netz werden in positive Bedingungen übersetzt, d.h. die Transition kann schalten, wenn die „hemmende Stelle“ leer ist (bzw. bei Kantengewicht $g \neq 1$ weniger als g Token beherbergt). Die so definierten Subprozesse werden durch den Paralleloperator verbunden. Dabei synchronisieren sich Subprozesse, deren entsprechenden Stellen durch Transitionen verbunden sind, über die den Transitionen entsprechenden Aktionen.

Betrachtet man den Subprozeß $(System_0||_S Buffer_K)$, wobei $S = \{entry, out\}$, so kann er gemäß den Gleichungen auf Seite 31 äquivalent umgeformt werden (siehe Abbildung 3.7).

$$
\begin{array}{lcll}
System_0||_S Buffer_K & = & Stop + Stop + (entry, 1).(System_1||_S Buffer_{K-1}) & \langle E \rangle \\
 & = & (entry, 1).(System_1||_S Buffer_{K-1}) + Stop + Stop & \langle +_K \rangle \\
 & = & (entry, 1).(System_1||_S Buffer_{K-1}) + Stop & \langle +_0 \rangle \\
 & = & (entry, 1).(System_1||_S Buffer_{K-1}) & \langle +_0 \rangle \\
System_i||_S Buffer_{K-i} & = & Stop + Stop + (out, \mu).(System_{i-1}||_S Buffer_{K-(i-1)}) + & \\
 & & (entry, 1).(System_{i+1}||_S Buffer_{K-(i+1)}) & \langle E \rangle \\
 & = & (out, \mu).(System_{i-1}||_S Buffer_{K-(i-1)}) + & \\
 & & (entry, 1).(System_{i+1}||_S Buffer_{K-(i+1)}) & \text{s.o.} \\
System_K||_S Buffer_0 & = & Stop + (loss, 1).(System_K||_S Buffer_0) + & \\
 & & (out, \mu).(System_{K-1}||_S Buffer_1) + & \langle E \rangle \\
 & = & (out, \mu).(System_{K-1}||_S Buffer_1) + & \\
 & & (loss, 1).(System_K||_S Buffer_0) & \text{s.o.}
\end{array}
$$

Abbildung 3.7: Äquivalenzumformungen eines Subprozesses

Definiert man nun

$$Server_i := System_i||_S Buffer_{K-i} \quad (0 \leq i \leq K)$$

so erhält man folgenden Markow-äquivalenten Prozeß (siehe Abb.3.8):

$$
\begin{array}{lcll}
Server_0 & := & (entry, 1).Server_1 & \\
Server_i & := & (entry, 1).Server_{i+1} + (out, \mu).Server_{i-1} & (0 < i < K) \\
Server_K & := & (out, \mu).Server_{K-1} + (loss, 1).Server_K &
\end{array}
$$

Abbildung 3.8: Markow-äquivalenter Subprozeß

Andererseits ist die Markow'sche Bisimulation auch anhand der zugehörigen LTS erkennbar, wie in Abbildung 3.9 zu sehen ist.:

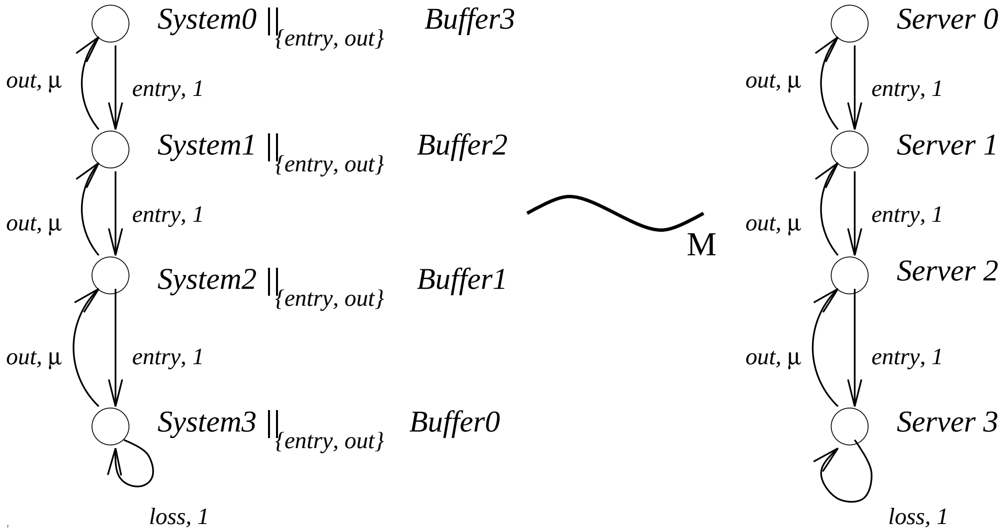

Abbildung 3.9: Operationale Semantik zweier Markow-äquivalenter Prozesse

Der Subprozeß $(System_0||_{\{entry,out\}}Buffer_K)$ kann also durch den Markow-äquivalenten Prozeß $Server_0$ ersetzt werden, ohne das LTS des Gesamtprozesses zu verfälschen. Dies entspricht einer Vereinfachung des Prozesses und einer Reduzierung der Ableitungsschritte. Die operationale Semantik der $Warteschlange$ für $K = 3$ ist in Abbildung 3.10 dargestellt. Dabei sind die zugrundeliegenden Markow–Ketten des Petri–Netzes und des stochastischen Prozesses identisch.

3.2.3 Weitere stochastische Prozeßalgebren

Bei der Erweiterung zu stochastischen PA gibt es verschiedene Möglichkeiten. Einerseits kann die Operatormenge mehr oder weniger umfangreich sein und die Bedeutung der Operatoren ist ebenfalls variabel. Andererseits kann man die Grundelemente von Prozeßalgebren, die Aktionen, unterteilen in aktive/passive oder zeitbehaftete/zeitlose, oder diese Unterscheidung entfällt. Selbst bei den Aktionen zugeordneten Raten ist die Interpretation nicht „à priori“ gegeben. Die Wahl, die hier getroffen hat, hat ihre Auswirkungen sowohl auf die Mächtigkeit des Formalismus in Bezug auf eine realistische Modellierung eines gegebenen Systems als auch auf den Äquivalenzbegriff.

Neben der im vorigen Abschnitt beschriebenen SPA gibt es im wesentlichen drei weitere Formalismen. *Performance Enhanced Process Algebra* (PEPA) von J. Hillston [Hil93], *Markovian Process Algebra* (MPA) von P. Buchholz [Buc94] und *Markovian Process Algebra* (MPA) von Bernardo, Donatiello, Gorrieri [BDG94] stellen weitere um Stochastik angereicherte PA dar. Einen ausführlichen Vergleich hat M. Ribaudo in [Rib95] in ihrer Dissertation gezogen, woraus hier einige der wichtigsten Resultate kurz zitiert werden.

	PEPA	MTIPP	MPA (Buchholz)	MPA (Bernardo)
Stop	Nein	Ja	Ja	Ja
Auswahl	Innerster Operator für parallele Komponenten	Bewachte Auswahl für Normalform	Bewachte Auswahl	Keine Restriktion
Rekursion	Konstante	Explizit	Explizit	Explizit
Synchronisation	Minimum der Raten	Produkt der Raten	Produkt der Invokationen	Maximum der Raten
Raten (a, r_1) und (a, r_2)	$r_1 \neq r_2$	$r_1 \neq r_2$	$r_1 = r_2 = r_\alpha$	$r_1 \neq r_2$
Passive Aktionen	Ja	Ja	Nein	Ja
Zeitlose Aktionen	Nein	in $\mathcal{L}_{IM}$	Nein	Ja
Gleichungen	Ja	Ja	Ja	Nein
Toolunterstützung	Ja	Ja	Nein	Nein

Die Tabelle soll folgendermaßen verstanden werden:

- „*Stop*“ bedeutet, daß es in dem Formalismus eine explizite Termination gibt.
- In der Zeile „Auswahl“ stehen Bedingungen an das Auftreten des Operators „Nichtdeterministische Auswahl“ innerhalb eines Wortes. So kann es aus Kongruenzgründen erforderlich sein, daß dieser Operator nicht als oberster auftritt (MTIPP) und (MPA) von Buchholz oder sogar nur im Inneren eines Wortes (PEPA).
- Die Darstellung von zyklischem Verhalten mittels eines eigenen Operators wird in der Zeile „Rekursion“ angegeben.
- Die Zeile „Synchronisation“ beschreibt den arithmetischen Umgang mit den zu einer Aktion a gehörenden Rate im Falle der Kooperation.

- Die folgende Zeile gibt an, ob eine Rate fest einem Aktionsnamen zugeordnet ist.
- Die Möglichkeit der Unterscheidung in passive/aktive bzw. zeitbehaftete/zeitlose Aktionen ist Gegenstand der folgenden Zeilen.

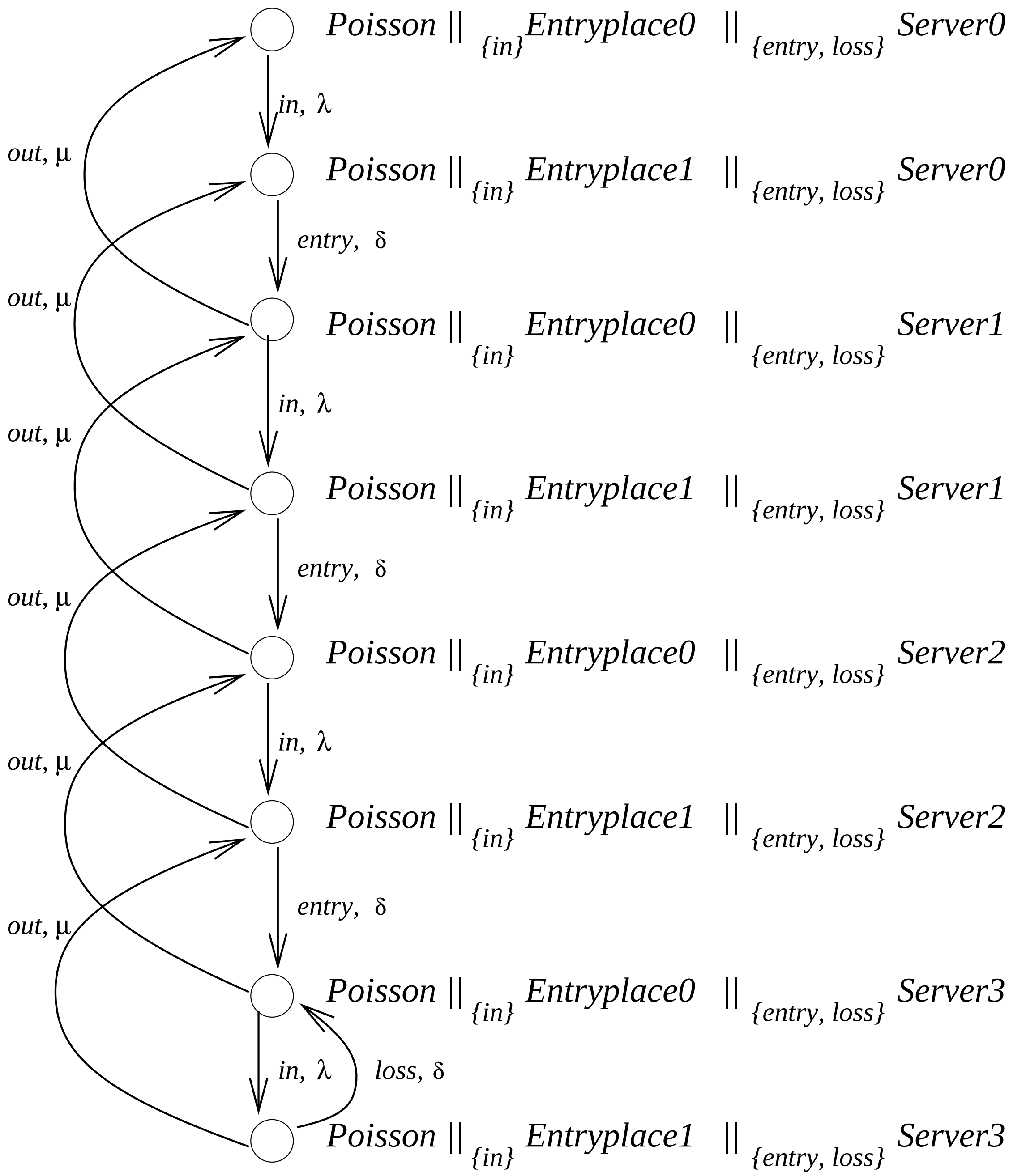

Abbildung 3.10: Operationale Semantik der Warteschlange

Kapitel 4

Einführung in Graphgrammatiken

Graphen haben in dieser Arbeit einen besonderen Stellenwert. Nicht nur Markow–Ketten werden durch sie repräsentiert, die Semantik von Petri–Netzen ist durch Erreichbarkeitsgraphen definiert, und ein LTS aus dem vorigen Kapitel ist auch als Graph darstellbar. Das allein könnte allerdings noch keine ausreichende Motivation sein, die formale Grundlage für Graphen hier einzuführen, da obige Formalismen wie gesehen sehr gut verständlich sind ohne eine genaue Kenntnis von Graphersetzungssystemen. Aber auch Phasentypverteilungen, um die stochastische Prozeßalgebren in dieser Arbeit erweitert werden sollen, können selbst wieder durch Graphen dargestellt werden (siehe auch Kapitel 2 bzw. [Lan]). Aktionen, deren Dauer phasentypverteilt ist, sind also im Gegensatz zu den bisher zulässigen Aktionen *nicht atomar*, weder funktional noch zeitlich. Die Atomarität ist aber gerade eine wesentliche Voraussetzung in den Formalismen der stochastischen Prozeßalgebren. Das vorgestellte Kalkül ist auch nur für solche Aktionen geeignet. Will man phasentypverteilte Aktionen in den Formalismus integrieren, müßte der Ableitungskalkül dahingehend erweitert werden, daß in einem Schritt ein Menge von Kanten und Nachfolgezuständen, die zudem untereinander verbunden sein können, abgeleitet werden kann. Dies ist unverträglich mit dem Konzept von Ableitungsregeln, die aus einer Menge von Prämissen eine *einzige* Konklusion folgern. Versucht man andererseits solch eine Phasentypverteilung nun in ihre atomaren Bestandteile aufzugliedern und durch einen Prozeß der SPA zu ersetzen, erschwert sich die Verständlichkeit erheblich. Zum einen ist für den Benutzer nicht mehr ersichtlich, welche Aktionen das funktionale Verhalten des Systems spezifizieren und welche „Dummies“ sind, um das zeitliche Verhalten zu modellieren. Zum anderen müßte man weiterhin den Umgang mit Verzweigungswahrscheinlichkeiten, die zu einer Phasentypverteilung gehören, im

Kalkül regeln. Führt man dazu, was zunächst naheliegt, Wahrscheinlichkeitskanten im LTS ein, müßten diese spätestens zum Erzeugen der Markow–Kette gelöscht werden. Da eine Ableitungsregel aber nur erzeugenden Charakter hat, wäre auf jeden Fall die Markow–Kette nicht mehr direkt aus dem LTS ersichtlich.
Aber nicht nur der starke Zusammenhang der verschiedenen Formalismen mit Graphen läßt eine explizite Behandlung von Graphen und Graphgrammatiken sinnvoll erscheinen. Daneben können wegen des konstruktiven und regelbasierten Charakters von Graphgrammatiken die Ersetzungsschritte als ein Algorithmus aufgefaßt werden. Aufgrund der mathematischen Fundierung dieser Theorie ist es zudem möglich, die Korrektheit der Regeln bezüglich des erzeugten LTS nachzuweisen und damit den Algorithmus zumindest plausibel zu machen. Dabei wird das Verständnis durch die graphische Anschauung unterstützt.
Aus diesen Gründen erscheint es sinnvoll, das Ableitungskalkül durch eine Menge von Graphregeln zu ersetzen. Somit wird es dann gleichzeitig möglich, die herkömmlichen exponentiellen Verteilungen wie auch allgemeine Phasentypverteilungen innerhalb eines Formalismus behandeln zu können wie auch leicht in ein Programm zur Erzeugung der zugrundeliegenden Markow–Kette umsetzen zu können.
Im folgenden werden diejenigen Konzepte aus der Theorie der Graphgrammatiken ([EPS73], [Ehr79], [Löw93], [KLTW93]) vorgestellt, die in dieser Arbeit zur Anwendung kommen sollen.
Allgemein beschrieben sind Graphgrammatiken eine Ausprägung von Ersetzungssystemen. Dabei werden Graphen regelbasiert ersetzt, d.h. eine Regel $r : L \to R$ angewandt auf einen Graphen G bedeutet intuitiv, daß der Teilgraph L in G ersetzt wird durch den Teilgraphen R. Man kann jetzt sowohl den Graphbegriff als auch die Anwendung einer Regel genauer spezifizieren.

4.1 Einfache Graphen und Graphtransformation

In diesem Abschnitt werden einfache Graphen eingeführt, ihre Struktur, Abbildungen zwischen Graphen, d.h. *Graphmorphismen*, Regeln und ihre Anwendung auf einen „Wirtsgraphen“ im *SPO–Ansatz* (siehe [Löw93]). Das Verständnis dieser Begriffe ist die Voraussetzung für die folgenden Abschnitte, in denen die Struktur der Graphen erweitert sowie die Anwendbarkeit von Regeln eingeschränkt wird. Anhand von Beispielen werden die Begriffe veranschaulicht. Weiterhin wird hier *sequentielle* *und parallele Unabhängigkeit* von Graphableitungen erklärt.

Unter *einfachen Graphen*[1] sind getypte, gerichtete Graphen mit Mehrfachkanten zu verstehen. Ein solcher Graph besteht aus einer (getypten) Knotenmenge und einer (getypten) Kantenmenge, wobei jeder Kante ein Quell- und ein Zielknoten zugeordnet wird. Die Struktur eines solchen Graphen G wird folgendermaßen beschrieben $G = (E, V, s, t)$:

$$E \overset{s}{\underset{t}{\rightrightarrows}} V$$

wobei E die Kantenmenge *(Edges)*, V die Knotenmenge *(Vertices)*, s *(source)* und t *(target)* Quell– resp. Zielfunktion. Die Knoten werden graphisch ● oder ○ dargestellt, Kanten von unterschiedlichen Typen $\longrightarrow$ oder $\dashrightarrow$.
Eine *Graphtransformation* findet durch die Anwendung einer Regel statt. Eine Regel $r : L \to R$ besteht aus zwei Graphen L und R, wobei r ihre „Beziehung“ angibt, d.h. r ist ein *Graphmorphismus*.

Definition 4.1.1 (Graphmorphismus)
Ein (totaler) Graphmorphismus $m : G \to H$ zwischen zwei einfachen Graphen G und H besteht aus zwei (totalen) Abbildungen, je zwischen Knoten– und Kantenmengen von G und H, so daß

- diese mit Quell- und Zielfunktion verträglich sind, d.h. der Quellknoten der abgebildeten Kante wird auf den Quellknoten der Bildkante abgebildet und analoges gilt für den Zielknoten.
- diese mit Knoten– und Kantentypen verträglich sind, d.h. Knoten und Bildknoten haben den gleichen Typ und analoges für Kanten[2].

Dementsprechend besteht ein partieller Graphmorphismus f aus zwei partiellen verträglichen Abbildungen auf Knoten– und Kantenmengen. Zusätzlich wird gefordert, daß der Definitionsbereich des partiellen Graphmorphismus ($dom(f)$) selbst wieder ein Graph ist.
□

[1] Alle formalen Definitionen der Begriffe dieses Abschnitts sind in [KLTW93] zu finden. Anhand einer Spezifikation eines Grapheditors sind diese Konzepte in [CGHB+95] motiviert.

[2] Dies ist eine relativ simple Auffassung, da die Abbildung auf den Typen die identische Abbildung ist. Man könnte auch eine beliebige Abbildung auf den Typen haben. Dann fordert die Verträglichkeit, daß der Typ der Bildkante gleich dem Bildtypen der Urkante ist. Analoges für Knoten.

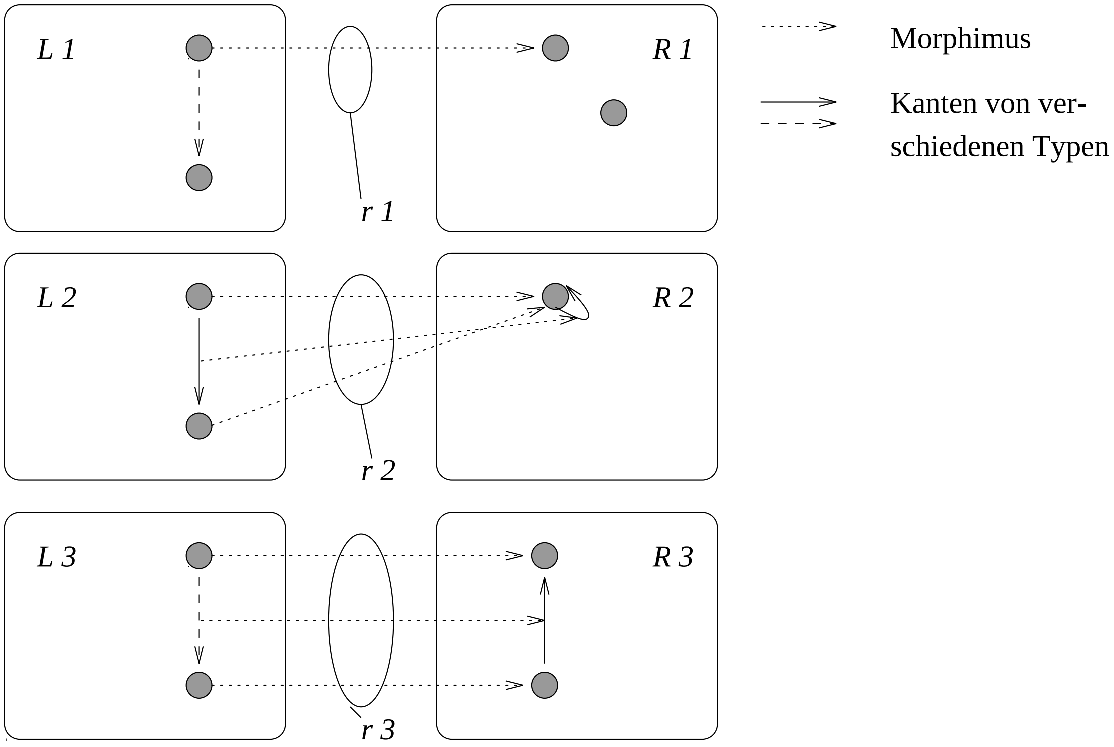

Abbildung 4.1: Beispiele für Graphmorphismen auf einfachen Graphen

Eine besondere Bedeutung kommt hier den nicht–injektiven Graphmorphismen zu, die auch *verklebende* Graphmorphismen genannt werden. Diejenigen Objekte, die durch den Morphismus auf ein einziges Objekt abgebildet werden, werden *identifiziert* oder auch *verklebt*.
Beispiele: In Abbildung 4.1 ist $r1$ ein partieller, $r2$ ein totaler, nicht–injektiver, also verklebender Graphmorphismus und $r3$ ist *kein* Graphmorphismus, da er die Verträglichkeitsbedingung verletzt.
Ausgehend von der Definition des Morphismus sind wir nun in der Lage, eine Regelanwendung zu charakterisieren. Hierbei verwenden wir das Konzept des *Single–Pushouts*, (SPO), das in [Löw93] eingeführt wird.

- Eine *Regel* r ist ein partieller Graphmorphismus im oben beschriebenen Sinne. Dabei liegen gerade diejenigen Knoten und Kanten nicht im Definitionsbereich von r, die gelöscht werden sollen. Gibt es also solche Objekte, heißt r auch *löschende Regel*. Im Gegensatz dazu sind nicht löschende und nicht surjektive Regeln *erzeugende Regeln*.

- Eine Regel $r : L \longrightarrow R$ ist *anwendbar* auf einen Graphen G, wenn es einen totalen Graphmorphismus $m : L \longrightarrow G$ gibt. Ein solcher Graphmorphismus heißt *Ansatz* der Regel r im Graphen G. Die Knoten und Kanten, die nicht im Bild von m liegen, werden *Kontext* genannt.

- Zu jeder Regel $r : L \longrightarrow R$ mit Ansatz m in G gibt es genau einen Graphen H, der das *Ergebnis der Regelanwendung* repräsentiert (und charakterisiert ist als sogenannter Pushoutgraph in der Kategorie der Graphen).

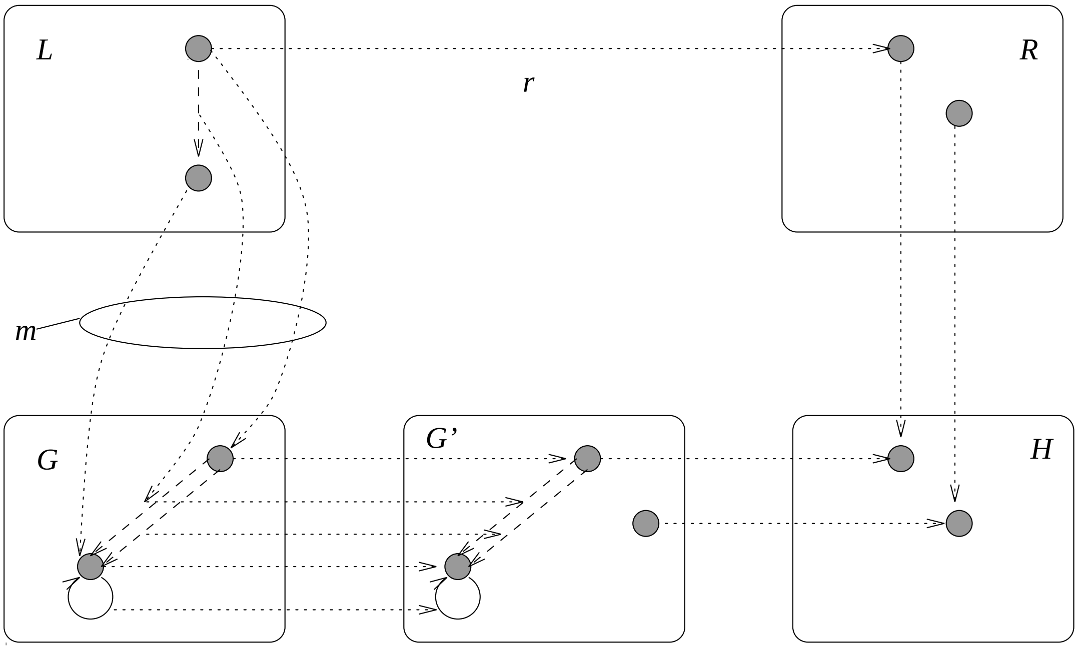

Abbildung 4.2: Beispiel einer SPO-Graphtransformation

Ist eine Regel $r : L \longrightarrow R$ mit einem Ansatz m in einen Graphen G gegeben, so erfolgt die *Konstruktion des Ergebnisgraphen H* bei der SPO-Graphtransformation in zwei Schritten (siehe auch Abbildung 4.2):

1. Verkleben und Hinzufügen
   Zunächst wird nur der Definitionsbereich $dom(r)$ des Regelmorphismus r betrachtet; das sind genau diejenigen Knoten und Kanten aus L, die über r nach R abgebildet werden. Über den Ansatzmorphismus m hat nun jedes dieser Objekte eine Entsprechung in G. Der Teilgraph C in G, der durch den Ansatz des Definitionsbereiches $dom(r)$ getroffen wird, wird nun durch die rechte Regelseite ersetzt. Bei diesem Schritt werden r-Bildobjekte *verklebt*, wenn

der Regelmorphismus m ihre Urbilder aus L verklebt. *Hinzugefügt* werden Objekte, die im rechten Teil der Regel stehen, aber nicht von r erreicht werden. Indem diese Ersetzung ausgeführt wird, entsteht als Zwischenergebnis ein Graph G', in den alle Objekte aus dem Kontext in G einfach unverändert übernommen werden.

2. Löschen
   In der Transformationsregel sind die zu löschenden Objekte diejenigen, die über r nicht nach R abgebildet werden, die also *nicht* in $dom(r)$ liegen. Alle diese Objekte haben über den Ansatz m eine Entsprechung in G. Da bei dem Zwischenschritt von G nach G' nichts gelöscht wurde, können sie alle auch in G' wiedergefunden und gelöscht werden. Dabei können „hängende Kanten“ entstehen, das sind Kanten, die keine Quelle oder kein Ziel mehr haben; solche Kanten werden mitgelöscht, sonst wäre das Ergebnis kein Graph mehr. Der so entstandene Graph H ist das Ergebnis der SPO-Ableitung.

Aus dieser Konstruktionsweise ergibt sich, daß *Konflikte* einfach durch Löschen „gelöst“ werden, was typisch für den SPO-Ansatz ist. Ein Konflikt entsteht, wenn zwei Objekte aus der linken Regelseite, von denen das eine durch die Regel erhalten, das andere gelöscht wird, durch einen verklebenden Ansatz auf dasselbe Objekt in G abgebildet werden.

Parallele Unabhängigkeit

Im letzten Teil dieses Abschnitts wollen wir uns kurz der parallelen und sequentiellen Unabhängigkeit von Regelanwendungen zuwenden. Wenn wir an Graphregeln als einer Art von Programmcode denken, der auf einen Graphen G angewandt wird, ist es häufig ausschlaggebend, in welcher Reihenfolge diese Programmstücke angewandt werden. So müssen die Regelanwendungen sequentialisiert werden, um einen gewünschten Ergebnisgraphen zu erhalten. In einigen Fällen aber kann diese Sequentialisierung überflüssig sein, da die Anwendung der Regeln in einer beliebigen Reihenfolge erfolgen kann, immer mit dem gleichen Ergebnis. Diese Eigenschaft wird durch parallele und sequentielle Unabhängigkeit beschrieben, und erlaubt somit eine nichtdeterministische Auswahl der nächsten Regel oder sogar eine verteilte parallele Anwendung.

Gegeben seien im folgenden zwei Regeln $r_1 : L_1 \rightarrow R_1$ und $r_2 : L_2 \rightarrow R_2$. Man nennt die Ableitung eines Graphen G mit Regel r_2 am Ansatz m_2 *schwach parallel unabhängig* von einer anderen Ableitung des Graphen G mit Regel r_1 am Ansatz m_1, wenn sie kausal von ihr unabhängig ist, d.h. beliebig verzögert ausgeführt werden kann. Intuitiv bedeutet das, daß r_2 nicht auf solchen Elementen von G operiert, die von r_1 unter m_1 gelöscht werden. Das heißt, die Elemente des Ausgangsgraphen, die im Ansatz der zweiten Ableitung liegen, dürfen nur im Kontext oder im bewahrten Teil der ersten Ableitung liegen. Zu beachten sind hier auch die bei der ersten Ableitung implizit gelöschten Kanten. Zwei Ableitungen, die wechselseitig schwach parallel unabhängig sind, werden *(stark) parallel unabhängig* genannt.
Neben der Unabhängigkeit von Ableitungen läßt sich auch die Unabhängigkeit von Regeln untersuchen: Zwei Regeln sind voneinander unabhängig, wenn alle möglichen Ableitungen (an beliebigen Graphen mit beliebigen Ansätzen) mit diesen Regeln voneinander unabhängig sind.

Sequentielle Unabhängigkeit

Eine Ableitung ist (schwach) sequentiell unabhängig von einer anderen, wenn sie keine Elemente benutzt, die von der anderen erst erzeugt werden. (Starke) sequentielle Unabhängigkeit und Unabhängigkeit von Regelableitungen sind analog zum parallelen Fall definiert.

4.2 Attributierte Graphen und Graphtransformation

In diesem Abschnitt wird der Graphbegriff und somit auch der Morphismenbegriff erweitert. Attributierung von Knoten und Kanten ist eine Möglichkeit, zusätzliche Informationen in den Graphen zu kodieren und Berechnungen durchzuführen. Graphtransformation modifiziert sich dahingehend, daß diese Attribute die Anwendbarkeit von Regeln einschränken. So wird, aufbauend auf der vorigen Einleitung, die Anwendung einer Regel auf einen attributierten Graphen definiert und am Beispiel erläutert. Zusätzlich können in die linke Regelseite Bedingungen an den Ansatz, in die rechte Regelseite Belegungen der Attribute im Ergebnisgraphen aufgenommen werden. Die formalen Definitionen und Sätze finden sich in [HMTW95].

Attributierte Graphen bestehen aus zwei Komponenten: dem Graphanteil und dem Attributanteil, die miteinander verknüpft werden. Der Graphteil sind einfache getypte Graphen wie sie im vorigen Abschnitt beschrieben wurden. Attribute sind Variablen oder Werte, die den Kanten und Knoten zugeordnet werden. Dabei können jedem Typen von Kanten resp. Knoten Attribute aus unterschiedlichen Bereichen, d.h. Mengen, zugeordnet werden. Solche Mengen sind z.B. ganze Zahlen, reelle Zahlen oder Zeichenketten, die zudem näher beschrieben werden können (z.B. durch eine reguläre Grammatik). Der Bereich, aus der ein Attribut kommt, nennen wir im folgenden *Attributtyp*, die Gesamtheit aller Attributtypen *Attributanteil*.

In dieser Arbeit werden nur Graphen über dem gleichen Attributanteil betrachtet, was die allgemeine Definition aus [HMTW95] vereinfacht. Somit setzen sich (partielle) *Attributierte Graphmorphismen* aus (partiellen) einfachen Graphmorphismen auf dem Graphanteil und totalen Abbildungen auf den Attributtypen zusammen, die wiederum die Verträglichkeitsbedingung erfüllen müssen.

- Eine *Regel* $r : L \to R$ ist ein Graphmorphismus zwischen attributierten Graphen, wobei die Attribute auch Variablen zu dem entsprechenden Attributtypen sein können. Sinnvollerweise sollten die Variablen in R schon in L vorkommen, um gebunden zu werden.

- Ein *Ansatz* $m : L \longrightarrow G$ ist ein totaler Graphmorphismus, der die Variablen aus L belegt. Gleiche Variablen aus L müssen auch gleich belegt werden, was eine indirekte Bedingung an den Ansatz stellt. Konkrete Attributwerte hingegen müssen identisch abgebildet werden, was eine direkte Bedingung an den Ansatz stellt. Terme, die sowohl Variablen als auch Operationssymbole des Attributtypen enthalten, werden durch „pattern matching“ belegt, d.h. die Variablen werden so belegt, daß der Term aus L mit dem Wert aus G unifiziert wird. Implizit werden mit dem Ansatz auch die Variablen aus R belegt.

- Eine Regel $r : L \longrightarrow R$ ist *anwendbar* auf einen Graphen G, wenn es einen Ansatz $m : L \longrightarrow G$ gibt.

- Zu jeder Regel $r : L \longrightarrow R$ mit Ansatz m in G gibt es genau einen Graphen H, der das *Ergebnis der Regelanwendung* repräsentiert. Die Belegung der

Variablen aus L mit Werten aus G setzt sich in den Ergebnisgraphen H fort.

Graphisch stellen wir Attribute dar, indem wir sie in der Nähe des Knotens/ der Kante plazieren, zu dem sie gehören, z.B. ● $X + X$ und $\xrightarrow{<a,\, \lambda\,,\, w>}$. Ein Beispiel der Regelanwendung in einer attributierten Graphgrammatik ist in Abbildung 4.3 zu sehen.

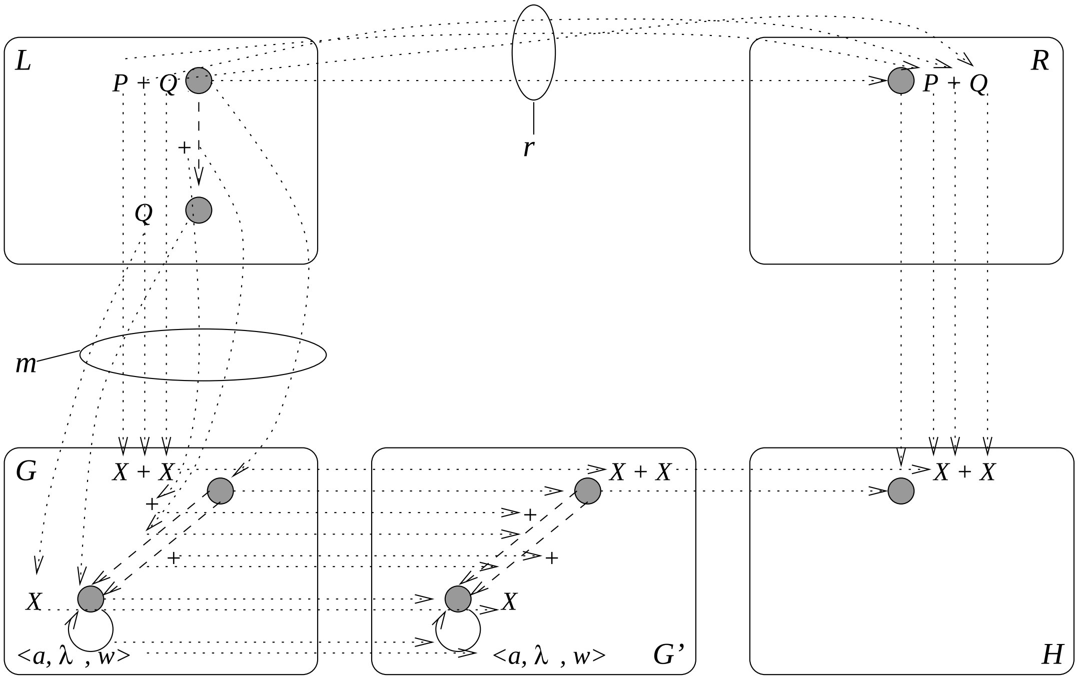

Abbildung 4.3: Regelanwendung bei attributierten Graphen

Erläuterung derr Abbildung: Knotenattribute sind Prozesse im Sinne des vorigen Kapitels, der Kantentyp $\longrightarrow$ *hat eine Aktion, also Aktionsname mit zugehöriger Rate, und ein Wort w als Wert und der Kantentyp* $\dashrightarrow$ *hat einen zweistelligen Operator der abstrakten Sprache zum Attribut. Dabei steht „X" für den Prozeß* $(a, \lambda).Stop$*, dementprechend* $X + X$ *für* $(a, \lambda).Stop + (a, \lambda).Stop$.
Der Zwischengraph G' ergibt sich aus dem Konstruktionsschritt 1 „Verkleben und Hinzufügen" im SPO–Konzept.

4.2.1 Bedingungen auf dem Attributanteil

Durch die Erweiterungen der Graphen um Attribute ist man nun in der Lage, zusätzliche Informationen in den Graphen zu kodieren. Sie stellen aber auch eine Möglichkeit dar, bestimmte Bedingungen an einen Ansatz zu stellen. Dies wird implizit durch die Attribute in der linken Regelseite realisiert. Sie stellen insofern eine Bedingung dar, als sie durch pattern–matching mit den Attributen des Wirtsgraphen unifizierbar sein müssen.

Häufig möchte man aber Bedingungen an einen Ansatz formulieren, die sich nur auf die Attribute beziehen. Diese Bedingungen können in Form von Gleichungen oder logischen Aussagen in die linke Regelseite mitaufgenommen werden. Ein Ansatz $m : L \to G$ in den Graphen G ist dann nur gegeben, falls er erstens ein totaler attributierter Graphmorphismus ist und zweitens die Auswertung des logischen Ausdrucks wahr ergibt. Ein Beispiel, in dem eine solche bedingte Regel nicht anwendbar ist, d.h. es keinen Ansatz in den Graphen G gibt, ist in Abbildung 4.4 zu sehen. Sucht man nach einem Ansatz $m : L \to G$, so muß dieser auf dem graphischen Teil die Identität sein. Auf dem Attributanteil ergeben sich wegen der Verträglichkeitsbedingung somit folgende Belegungen der Variablen:
$P \mapsto Server$, $a \mapsto fail$, $b \mapsto fail$, $r \mapsto \mu$. Somit werden a und b gleich belegt, was die Bedingung $a \neq b$ verletzt. Daher ist diese Regel auf den Graphen G nicht anwendbar.

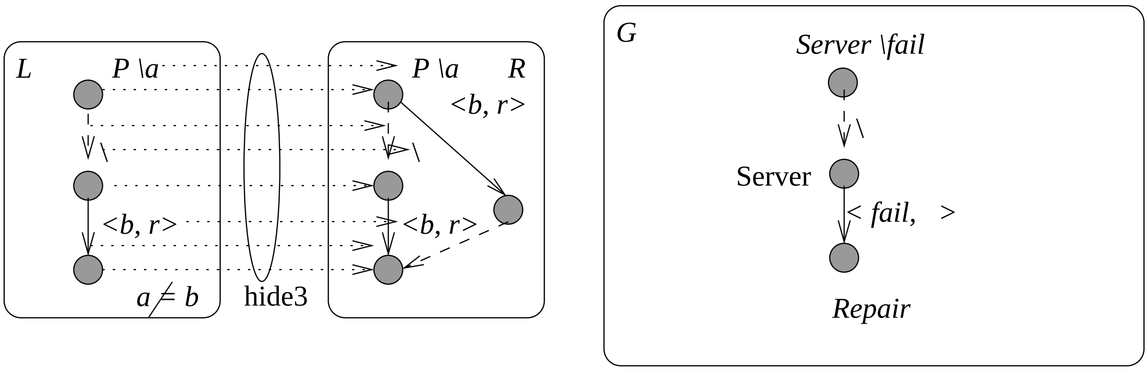

Abbildung 4.4: Nicht anwendbare Regel

Zusätzlich soll hier eine weitere Konvention eingeführt werden, die es erlaubt, eine Menge von Regeln, die sich nur auf dem Attributanteil, nicht aber auf dem Graphanteil unterscheiden, durch eine einzige Graphregel darzustellen.

Konvention 4.2.1
In der rechten Regelseite wird die Belegung mit Attributen angegeben, die in Abhängigkeit von der Variablenbelegung durch den Ansatz erfolgt.

Dies stellt im wesentlichen eine Schreibvereinfachung dar, da man ohne diese Konvention durch eine komplette Fallunterscheidung denselben Effekt erzielen könnte.

Als Beispiel siehe die Regel $para2$ aus Abbildung 5.8 auf der Seite 61. Dort soll das Attribut des neu erzeugten Knotens entweder „$P''||_S P'''$“ oder „$P'''||_S P''$“ sein in Abhängigkeit davon, wie die Variable „x“ durch den Ansatz im Graphen belegt wurde.

4.3 Beschränkte Regelanwendung und Transaktionskonzept

Dieser Abschnitt beschäftigt sich mit der Modifizierung der Anwendung von Regeln. Zum einen wird die Anwendbarkeit von Regeln eingeschränkt, indem zusätzliche Anwendungbedingungen gestellt werden. Zum anderen wird ein Transaktionskonzept eingeführt, das die Reihenfolge der Regelanwendung und die Häufigkeit betrifft. Insbesondere wird durch dieses Konzept ein Algorithmus beschrieben.

4.3.1 Beschränkte Regelanwendung

Die einzige Voraussetzung für die Anwendbarkeit einer Regel $r : L \to R$ auf G war bisher die Existenz eines totalen Graphmorphismus $m : L \to G$, was eine relativ schwache Bedingung ist. Die Existenz eines solchen Morphismusses stellt eine positive Bedingung dar und bedeutet, daß die Struktur L „irgendwie“ in G zu finden ist.[3] Insbesondere rein erzeugende Regeln, d.h. totale injektive Regeln, sind damit unendlich oft anwendbar. Die Menge aller aus einem Startgraphen mittels einer Regelmenge ableitbaren Graphen, die *Graphsprache*, wird demnach sofort unendlich, falls nur eine erzeugende Regel anwendbar wird. Aber nicht nur aus Gründen der Termination ist es wünschenswert, *negative Bedingungen* ausdrücken zu können, sondern auch um die Modellierungsmächtigkeit dieses abstrakten Beschreibungsmittels im Allgemeinen zu erhöhen. Bei Graphregeln heißt das, daß bestimmte Strukturen verboten werden sollen für die Anwendbarkeit einer Regel. Zusätzlich zu den positiven Bedingungen werden nun negative zugelassen, was folgendermaßen notiert wird:

$$nab^4 : \quad N \xleftarrow{n} L \xrightarrow{r} R$$

[3]Wären hier partielle Morphismen zugelassen, könnte „man sich die Mühe des Zeichnens von L sparen“, da es von jedem Graphen mindestens einen partiellen Morphismus in jeden anderen gibt. Die Regel soll ja gerade ausdrücken, daß bestimmte Strukturen gefordert werden.

Dabei ist n ein totaler Graphmorphismus, meistens sogar die Inklusion. N stellt also gewissermaßen den Kontext von L dar. Wollen wir nab auf einen Graphen G anwenden, müssen wir, intuitiv beschrieben, wiederum L in G finden, aber nicht gleichzeitig auch N. Dabei kann es passieren, daß sich positive und negative Bedingungen widersprechen (z.B. falls $N = L$ und $n = id$), und die Regel somit nie anwendbar ist. Solche Widersprüche sollten vermieden werden.
Etwas genauer kann man die Anwendbarkeit einer nab-Regel durch ein Diagramm[5] folgendermaßen beschreiben:

$$\begin{tikzcd}
N & L \arrow[l, "n"'] \arrow[r, "r"] \arrow[d, "m"] \arrow[dl, phantom, "="] & R \\
 & G &
\arrow[from=1-1, to=2-2, "f"', "/" marking]
\end{tikzcd}$$

Abbildung 4.5: Morphismendiagramm einer nab-Regel

Die Regel $nab : N \overset{n}{\leftarrow} L \overset{r}{\rightarrow} R$ ist anwendbar auf den Graphen G am Ansatz $m : L \rightarrow G$, falls kein totaler Graphmorphismus $f : N \rightarrow G$ existiert, sodaß $m = f \circ n$ gilt.
Ein häufig verwandter Spezialfall der nab-Regeln mit einer negativen Bedingung ist, wenn $N = R$ und $n = r$ gilt. Die Regel darf also nur angewandt werden, wenn sie nicht schon vorher an diesem Ansatz angewandt wurde (siehe auch Abbildung 4.6 und 4.7). Zur Vereinfachung schreiben wir diesen Fall im folgenden:

$$r : L \overset{\leftrightarrows}{\rightarrow} R$$

In Verallgemeinerung dieses Konzeptes sind auch mehrere negative Bedingungen zugelassen. Man notiert die Regel dann folgendermaßen:

$$nab : N_i \overset{n_i}{\leftarrow} L \overset{r}{\rightarrow} R$$

mit $N_i, i \in I := \{1, \ldots, k\}, k \in \mathbb{N}$ einer endlichen Familie von Graphen und $n_i, i \in I$ einer Familie von Graphmorphismen. Eine nab–Regel mit mehreren negativen

[4] nab steht für Nicht$A$nwendbarkeits$B$edingung.
Neben solchen nab-Regeln sind noch weitere Einschränkungen in [HMTW95] untersucht worden.
[5] Diagramm in der Kategorie der Graphen mit partiellen Graphmorphismen

Anwendungsbedingungen verbietet also alle Strukturen, die durch die N_i bezeichnet sind. Formaler ausgedrückt:
Die Regel $nab : N_i \overset{n_i}{\leftarrow} L \overset{r}{\rightarrow} R$ ist anwendbar auf den Graphen G am Ansatz $m : L \rightarrow G$, falls kein einziger Graphmorphismus $f_i : N_i \rightarrow G$ existiert, sodaß $m = f_i \circ n_i$ gilt.
Falls von den negativen Bedingungen N_i wieder für ein i gilt: $N_i = R$ und $n_i = r$, notiert man diesen Fall analog zu obigem Fall mit einer einzigen negativen Bedingung:

$$N_j \overset{n_j}{\leftarrow} L \rightleftarrows R \text{ mit } j \in I \backslash \{i\}$$

Alle (attributierten) Graphregeln sind ein Spezialfall von *nab*–Regeln, wobei keine Struktur für die Anwendbarkeit verboten wird.

Konvention 4.3.1
Um die Lesbarkeit der Graphen im Folgenden zu erhöhen, werden die

- *Morphismenpfeile auf dem Graphanteil weggelassen unter der Annahme, daß Objekte mit gleicher Position aufeinander abgebildet werden. Nur bei Ausnahmen oder Mißverständlichkeiten werden sie gezeichnet. Die Existenz des Morphismus wird durch einen einzigen Pfeil in die entsprechende Richtung angedeutet.*
- *Variablen in linker (rechter) Regelseite weggelassen, sofern sie nicht mehr als einmal in dem Graphen vorkommen und somit völlig frei belegt werden können.*
- *Morphismenpfeile auf dem Attributanteil weggelassen. Die Verträglichkeit liefert, daß das Attribut eines Objektes auf das des Bildobjektes abgebildet wird.*

4.3.2 Transaktionskonzept

Das letzte Konzept, mit dem Graphgrammatiken angereichert werden sollen, ist das *Transaktionskonzept*, das in ähnlicher Weise in Datenbankanwendungen benutzt wird. In diesem Zusammenhang bedeutet es einerseits, daß die Regeln in einer bestimmten Reihenfolge angewandt werden sollen. Zwischenschritte sind dabei nicht von außen zugreifbar. Außerdem soll eine Transaktion scheitern, wenn sie nicht komplett ausgeführt werden kann, und der Graph soll in seinen ursprünglichen Zustand versetzt werden.

Ein *Transaktionselement* E ist eine endliche, nichtleere Menge von *nab*-Regeln. Dabei wird die übliche Mengenschreibweise mit geschweiften Klammern verwendet.

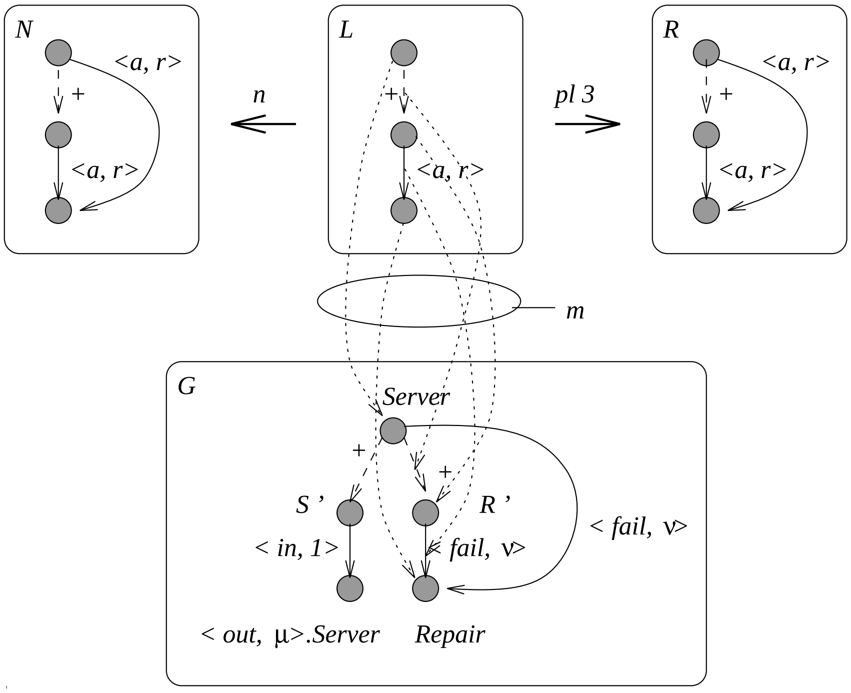

Abbildung 4.6: Scheitern einer nab-Regel

Erläuterung derr Abbildung: Hier wird das Beispiel der M/M/1/1/1/FCFS-Warteschlange aus dem vorigen Kapitel aufgegriffen (siehe Abbildung 3.1). S' steht hierbei für $(in, 1).(out, \mu).Server$ *und R' für* $(fail, \nu).Repair$. *An diesem Ansatz m darf die Regel pl3 nicht angewandt werden.*

In Erweiterung des Anwendbarkeitsbegriffes für einzelne Regeln heißt ein Transaktionselement E *anwendbar*, wenn mindestens eine der Regeln aus E anwendbar ist. Ist keine Regel aus E anwendbar, heißt E *nicht anwendbar*. Die einmalige *Anwendung* von E auf einen Graphen G besteht aus der Anwendung einer (beliebigen) Regel aus E auf G. In diesem Fall handelt es sich also um eine nichtdeterministische Auswahl, wenn mehrere Regeln aus E auf G anwendbar sind. Weiterhin kann vorgegeben werden, wie oft ein bestimmtes Transaktionselement angewendet werden soll (*Anwendungshäufigkeiten*). Möglich ist, daß ein Transaktionselement einmal, n-mal oder maximal oft, d.h. solange es anwendbar ist, angewendet werden soll. Anwendungshäufigkeiten werden folgendermaßen notiert:

- $t^n \quad (n \in I\!N^+)$ für n-malige Anwendung des Transaktionselementes t
- $t^\Box$ für wiederholte Anwendung des Transaktionselementes t, solange es anwendbar ist.

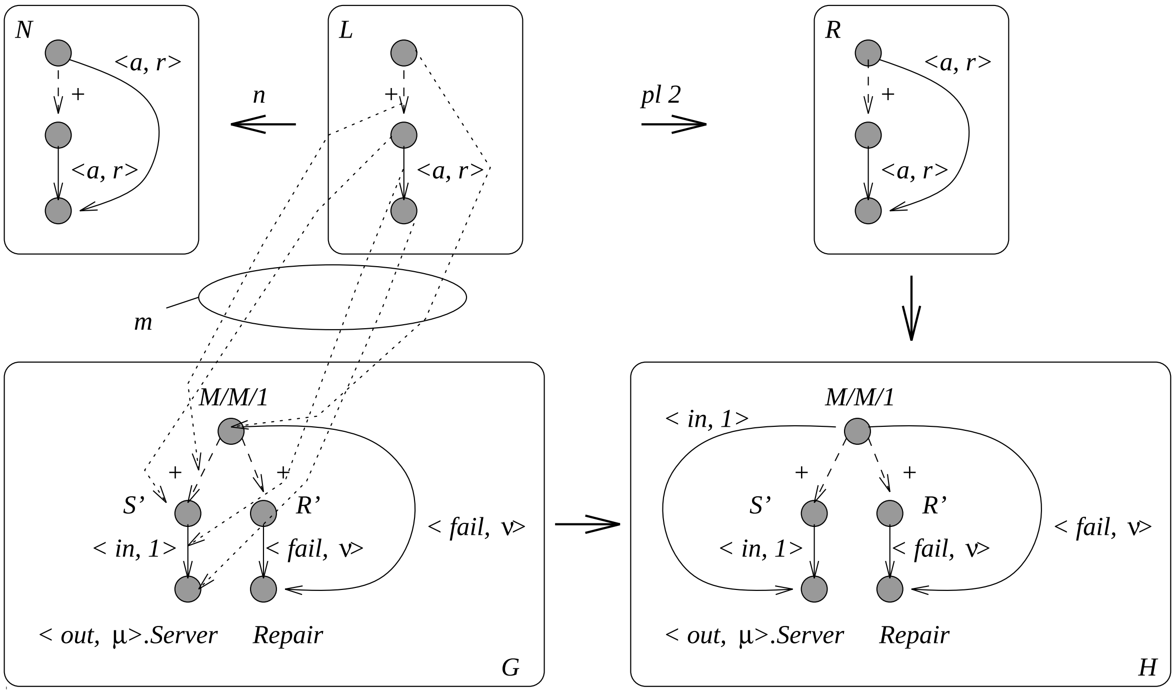

Abbildung 4.7: Anwendung einer *nab*-Regel

Erläuterung derr Abbildung: S' steht hierbei für $(in, 1).(out, \mu).Server$ *und R' für* $(fail, \nu).Repair$. *Am Ansatz m ist die Regel anwendbar und führt zum Ergebnis H.*

Die Mehrfachanwendung besteht dann aus der sequentiellen Ausführung des entsprechenden Transaktionselementes in der geforderten Häufigkeit. Ist eine Ausführung eines Transaktionselementes in der angegebenen Häufigkeit nicht möglich, schlägt die gesamte Transaktion fehl, d.h. der Graph wird nicht verändert. Bei wiederholter Anwendung eines Transaktionselementes, solange es anwendbar ist, besteht natürlich die Gefahr, daß die Transaktion nicht terminiert. Eine Folge von Transaktionselementen mit Anwendungshäufigkeiten bezeichnen wir als *Transaktion*. Die Transaktionselemente werden dabei in der gewünschten Reihenfolge notiert, durch Semikolons getrennt und durch || eingeschlossen. Hat die „Abarbeitung" mit der Anwendung einer Regel des ersten Transaktionselementes erst einmal begonnen, soll - Fehlschlagen der Transaktion und Endlosschleifen ausgenommen – die gesamte Regelfolge ohne Unterbrechung abgearbeitet werden.

Wir beschreiben den Aufbau einer solchen Transaktion durch folgende BNF–Darstellung:

<TRANSAKTION> := ‖<T-ELEMENT-FOLGE>‖
<T-ELEMENT-FOLGE> := <T-ELEMENT>
|<T-ELEMENT>; <T-ELEMENT-FOLGE>
<T-ELEMENT> := {<REGELMENGE>} <ANW.–HÄUFIGKEIT>
<ANW.–HÄUFIGKEIT> := $\Box \mid^{1} \mid^{2} \mid \ldots \mid^{n}$
<REGELMENGE> := <REGEL>
|<REGEL>, <REGELMENGE>
<REGEL> := < *nab*-REGEL>

Kapitel 5

Stochastische Prozeßalgebren mit Graphgrammatik

In diesem Kapitel sollen die beiden vorgestellten Konzepte der stochastischen Prozeßalgebren und der Graphgrammatiken zusammengeführt werden. Wie schon im vorigen Kapitel erwähnt, bietet sich diese Vorgehensweise aus mehreren Gründen an: Die operationale Semantik eines Prozesses ist durch ein labelled transition system gegeben, das durch einen Graphen repräsentierbar ist. Desgleichen sind Phasentyp–Verteilungen graphisch darstellbar, womit also die wesentlichen Teile dieser Arbeit durch Graphen einheitlich beschreibbar sind. Zudem ist das herkömmliche Ableitungskalkül der operationalen Semantik ungeeignet für phasentypverteilte Aktionen, da diese nicht atomar sind. Die Darstellung des zeitlichen Verhaltens einer solchen komplexen Aktion läßt sich also nicht in einem Schritt ableiten, da sie durch eine Menge von Zustandsübergängen und Zuständen gegeben ist, die außerdem mit einer gewissen Wahrscheinlichkeit betreten werden. Somit bedeutet das Ableitungskalkül eine Einschränkung auf atomare Aktionen, sofern man die komplexe phasentypverteilte Aktion nicht in ihre Bestandteile aufgliedern will und damit das Verständnis erheblich erschwert. Demgegenüber kann dieses Problem der Ableitung des Verhaltens einer komplexen Aktion mit Graphgrammatiken gelöst werden, und sie bieten zusätzlich den Vorteil der Anschaulichkeit, guten mathematischen Fundierung und Konstruktivität.

Deshalb soll der Formalismus der stochastischen Prozeßalgebren dahingegend modifiziert werden, daß das Ableitungskalkül durch eine Graphgrammatik ersetzt wird. Dabei soll die Syntax der stochastischen Prozesse unverändert bleiben und nur die Ableitung des LTS eines Prozesses mittels Graphregeln erfolgen. Ausgehend von dem abzuleitenden Prozeß als Startgraphen soll eine Regelmenge auf diesen angewandt werden, deren Anwendung das LTS des Prozesses erzeugt. Dazu werden zunächst die Graphklasse und die Regeln konkret angegeben. Zur Generierung des LTS muß sowohl die Reihenfolge der Regelanwendungen als auch deren Häufigkeit näher spezifiziert werden, was durch eine Transaktionsvorschrift realisiert wird. Anhand eines Beispiels wird dann die Korrektheit der angegebenen Graphgrammatik bzgl. des generierten LTS veranschaulicht.

5.1 Graphklasse

Das Ziel, die operationale Semantik eines Prozesses durch Anwendung von Graphregeln zu erzeugen, impliziert, daß die Graphen zumindest die Labels des LTS beinhalten. Das bedeutet, daß Knoten mit Prozessen der abstrakten Sprache beschriftet werden und Kanten mit einem Tripel aus Aktionsnamen, der Rate der Exponentialverteilung und der Zeichenkette aus Regelnamen. Es handelt sich also um attributierte Graphen. Zur Erinnerung hier noch einmal die Definitionen: Die Syntax eines stochastischen Prozesses ist durch die folgende Grammatik gegeben für $a \in Act$, $\lambda \in Rate = I\!R^{+}$, $S \subseteq Com, X \in Var, (Act = Com \cup \{\tau\})$

$$P := Stop \mid (a, \lambda).P \mid P + P \mid P||_S P \mid P\backslash a \mid recX : P \mid X$$

Analog sind die Kantenattribute Tripel

$$t \in Act \times Rate \times W^{*} \text{ mit } W = \{+_l, +_r, ||_l, ||_r, ||, \backslash_{yes}, \backslash_{no}\}$$

Im folgenden werden Knoten mit dem Attribut P synonym auch als Prozeß P bezeichnet, die Kante als *Aktionskante* oder auch *reguläre Kante*.
Dem konstruktiven Charakter der Graphgrammatiken Rechnung tragend braucht man zusätzlich *Hilfskanten*. Zum einen soll nur dann die operationale Semantik erzeugt werden, wenn sie es noch nicht wurde (Hilfskante *not expanded*), zum anderen muß die Dekomposition in Subprozesse dokumentiert werden. Dazu werden weitere Hilfskanten eingeführt, die im wesentlichen mit einem Regelnamen des Ableitungskalküls (siehe Abbildung 3.2) attributiert werden. Die Attribute dieser Hilfskanten sind Elemente der Menge $H = \{+_l, +_r, ||_l, ||_r, rec, \backslash, not\, expanded, root\}$. Variablen zu dieser Menge werden z.B. $+_x$ notiert und bedeuten dann, daß das Attribut entweder $+_l$ oder $+_r$ ist.

Mit der Unterscheidung zwischen Aktionskanten und Hilfskanten erhält man also zwei Typen von Kanten. Reguläre Kanten werden graphisch $\xrightarrow{< a,\, \lambda\, ,\, w>}$ dargestellt, Hilfskanten sind gestrichelt und beschriftet mit einem Element der Menge H, z.B. $\overset{+1}{\dashrightarrow}$. Die Graphklasse besteht also aus attributierten getypten Graphen und attributierten Graphmorphismen mit den oben angegebenen Attributtypen.

5.2 Regeln

Ruft man sich die Funktionsweise der Ableitungsregeln in Erinnerung, so fällt auf, daß die Prozesse rekursiv zurückverfolgt werden bis die prämissenlose Regel $\langle . \rangle$ angewandt werden kann. Diese Vorgehensweise soll nun analog auf Graphen angewandt werden. Dabei macht man sich die Tatsache zunutze, daß das LTS kompositional ist, d.h. das LTS eines zusammengesetzten Prozesses sich aus der Komposition der LTS seiner Subprozesse ergibt. Die Idee ist also: Ausgehend von dem Prozeß P als Startgraphen, dessen LTS erzeugt werden soll, werden rekursiv die LTS der Subprozesse generiert und daraufhin diese in geeigneter Weise zusammendesetzt. Der so entstandene Graph enthält somit das LTS des Prozesses wie auch seiner Subprozesse.

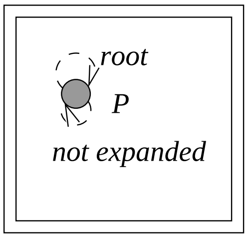

Abbildung 5.1: Startgraph, dessen LTS erzeugt werden soll

Dieser Idee zufolge kann man zwei Klassen von Regeln unterscheiden: die *dekomponierenden*, d.h. aufspaltenden Regeln und die *komponierenden* Regeln, die der Anwendung einer Deduktionsregel des Kalküls entsprechen. Zu jedem Operatoren der abstrakten Sprache gibt es demnach eine Menge von Regeln, solche, die die Herleitung der Prämissen realisieren und solche, die die Ableitung der Konklusion umsetzen. Einen Sonderfall stellen hier die prämissenlosen Operatoren „*Stop*“ und „.“ dar. Sie werden durch eine einzige Graphregel umgesetzt, da keine Bedingungen für die Ableitung zu beachten sind.

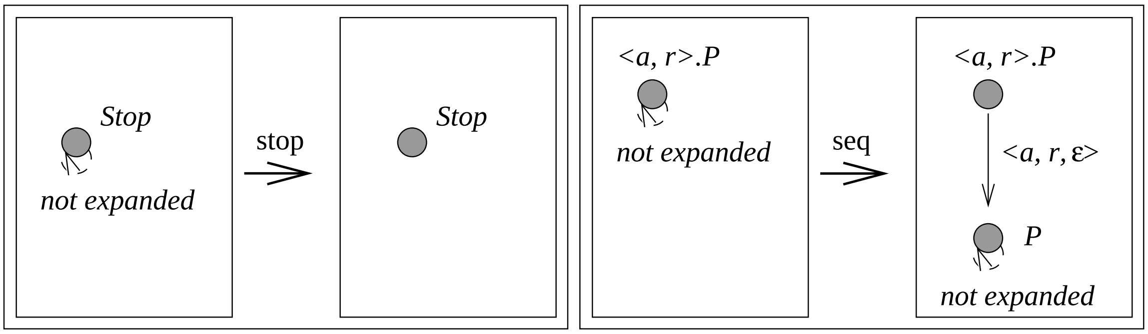

Abbildung 5.2: Stop– und Sequenz–Regel

Die *not expanded*–Schlinge dient dazu, festzustellen, ob mit der Generierung des LTS dieses Prozesses schon begonnen wurde oder nicht. Dementsprechend löschen alle dekomponierenden Regeln diese Kante und hängen eine *not expanded*–Schlinge an die Nachfolgeknoten. Einen Sonderfall behandelt die *stop*–Regel, die so interpretierbar ist, daß das LTS des Prozesses *Stop* „leer" ist, der Prozeß also keine Aktionen ausführen kann. Die *seq*–Regel dagegen generiert eine Kante des LTS und signalisiert, daß der Nachfolgeknoten wiederum expandiert werden muß (siehe Abbildung 5.2).
Bei den restlichen Operatoren gemäß der Syntax in Abschnitt 5.1 hingegen muß der Prozeß zunächst in seine Subprozesse zerlegt und später die generierten LTS in bestimmter Weise zusammengefügt werden. So braucht man für solch einen Operator

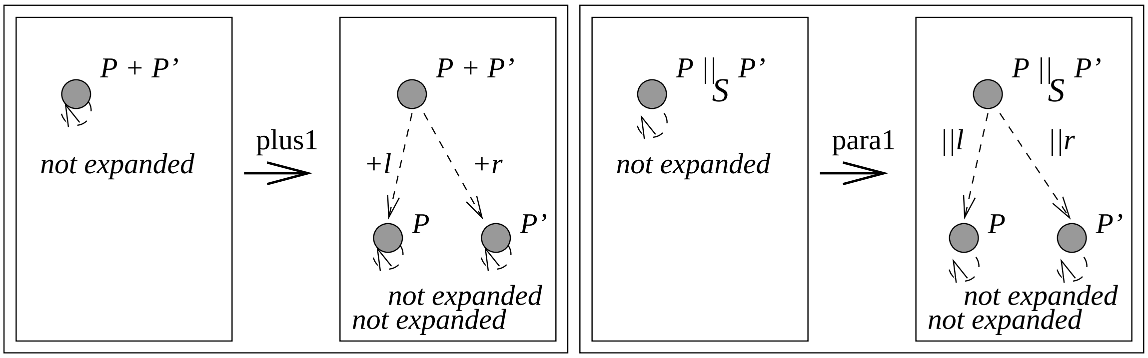

Abbildung 5.3: Dekomposition Auswahl– und Parallel–Regel

eine Menge von Graphregeln. Der Name der Graphregel gibt den Operator an, die Zahl spiegelt die Reihenfolge der Regeln zu einem Operator wider. Die jeweils erste Regel zu einem dieser Operatoren spaltet einen zusammengesetzten Prozeß in seine Subprozesse auf. Dabei werden Hilfskanten von dem Knoten des abzuleitenden

Prozesses zu den Knoten der Subprozesse gezogen. Diese ermöglichen es später, die Vorgehensweise beim Erzeugen des LTS richtig auszuwählen, sind somit gewissermassen das „Gedächtnis", wie der abzuleitende Prozeß zusammengesetzt war (vgl. Abbildung 5.3 für „+" und „$||_S$").
In ähnlicher Weise arbeiten die *hide*– und die *rec*–Regel aus Abbildung 5.4. Im Unterschied zu den obigen Regeln erzeugen sie jeweils einen Nachfolgeknoten.

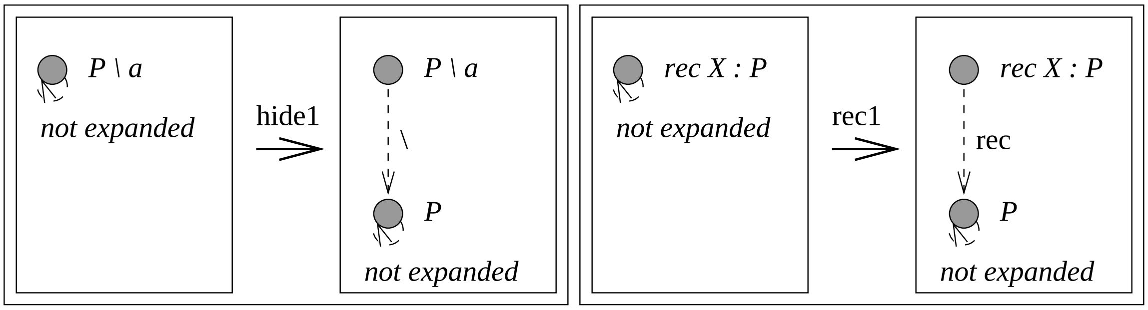

Abbildung 5.4: Abstraktions– und Rekursions–Regel–Dekomposition

Ist die Aufspaltung in die Subprozesse erst einmal in die Wege geleitet, kann der rekursive Prozeß P wieder mit $recX : P$ identifiziert werden. Die Regel $rec2$ (siehe Abbildung 5.5) kann angewandt werden, wenn die *not expanded*–Schlinge gelöscht ist. Die Hilfs– oder Aktionskanten, die an dem Knoten von Prozeß P hingen, befinden sich nun an dem Prozeß $recX : P$.

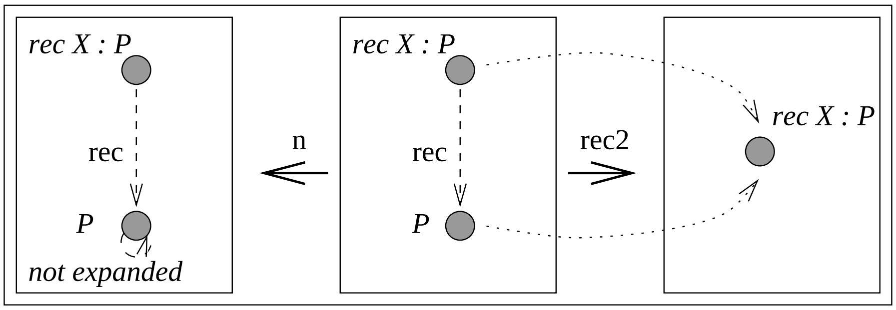

Abbildung 5.5: Rekursion-Regel–Komposition

Entsprechend der Grundidee soll auch in der Rekursion zunächst das LTS des inneren Prozesses schrittweise abgeleitet werden. Dies wird bei einem rekursiven Prozeß nicht vollständig möglich sein, da in seinem Inneren die freie Variable X vorkommt. In dem Fall wird sie mit dem Prozeß $recX : P$ identifiziert, d.h. die entsprechenden Knoten werden verklebt. Die Regel $rec3$ in Abbildung 5.6 realisiert die Verklebung.

Zusammen mit der Anwendung der Regel $rec2$ wird in dem Graphen ein Zyklus erzeugt.

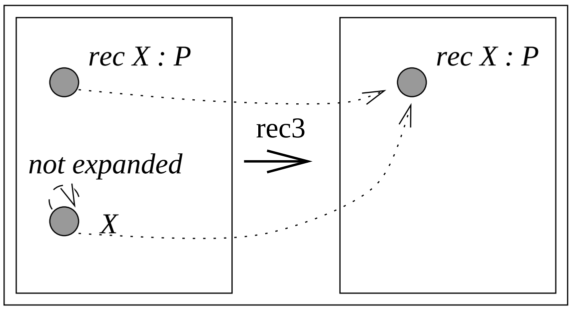

Abbildung 5.6: Rekursion-Regel–Komposition

Das Verfahren der Dekomposition kann in beliebiger Reihenfolge angewandt werden. So wird zunächst ein Baum von Subprozessen erzeugt. An den untersten Blättern hängen die *not expanded*–Schlingen. Kann man auf einen dieser Subprozesse S die *seq*–Regel anwenden, schafft diese Regelanwendung die Voraussetzung dafür, eine der folgenden komponierenden Regeln anzuwenden. Alle komponierenden Regeln der +– und ||–Operatoren fordern nämlich in ihrer linken Regelseite die Existenz (mindestens) einer Aktionskante, was bedeutet, daß das LTS dieses Subprozesses zumindest teilweise schon erzeugt wurde. In Abhängigkeit davon, durch welche Regel dieser Knoten S erzeugt wurde, kann man mit einer komponierenden Regel zu dem gleichen Operator fortfahren.

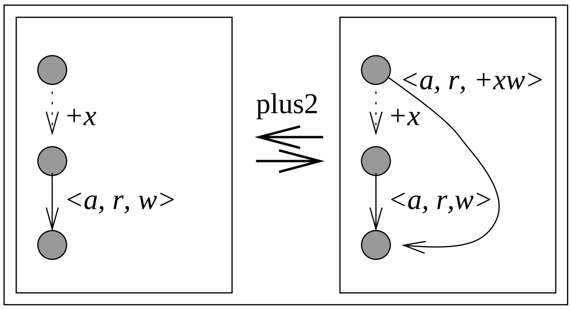

Abbildung 5.7: Auswahl–Regel–Komposition

In Abbildung 5.7 ist die Komposition des LTS zu einem Prozeß mit nicht-deterministischer Auswahl zu sehen. Diese Regel ist nur an einem Ansatz anwendbar, an dem sie noch nicht angewandt wurde, um zu verhindern, daß mehrere Kanten mit dem label gezogen werden (siehe dazu auch die Erklärung auf Seite 50). Dies würde das LTS und damit die zugrundeliegende Markow–Kette verfälschen.

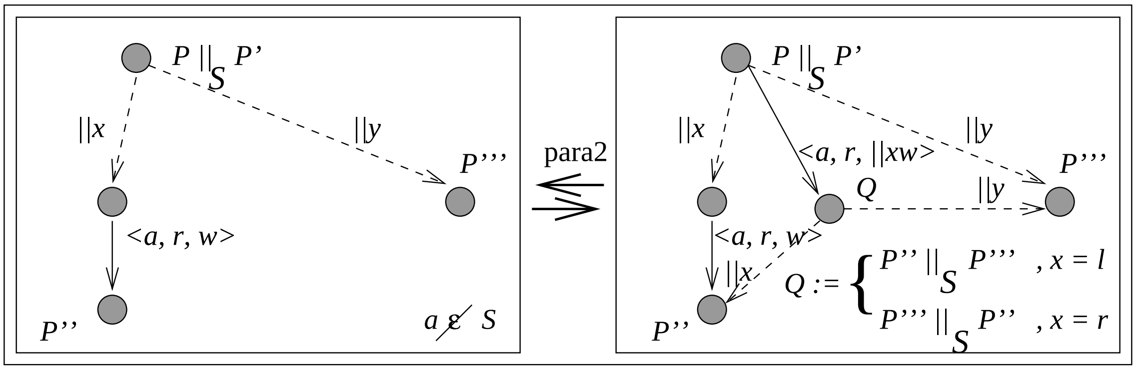

Abbildung 5.8: Parallel–Regel–Komposition ohne Synchronisation

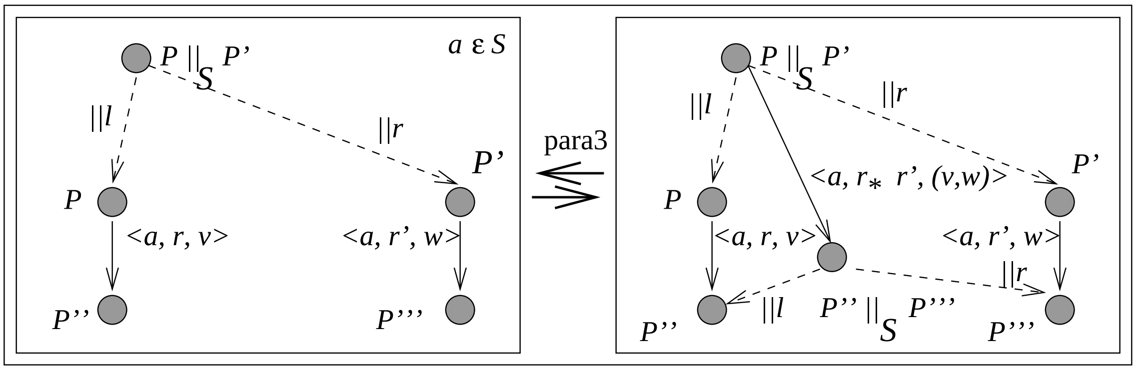

Abbildung 5.9: Parallel–Regel–Komposition mit Synchronisation

Die Regeln zum Paralleloperator in Abbildung 5.8 und 5.9 spiegeln die unterschiedlichen Fälle der Synchronisation wider: $para2$ wird angewandt, wenn die parallelen Prozesse nicht miteinander kommunizieren, $para3$ in dem Fall, in dem sie gemeinsam eine Aktion ausführen. Beides sind nab–Regeln, da sie ohne negative Bedingung unendlich oft anwendbar sind, was wiederum die zugrundeliegende Markow–Kette verfälschen würde.

In Abbildung 5.10 ist die Vorgehensweise bei funktionaler Abstraktion in zwei Graphregeln umgesetzt. Regel $hide3$ wird in dem Falle $a \neq b$ angewandt, $hide2$ sonst.

Durch wahlweises Anwenden der dekomponierenden und komponierenden Regeln wird schrittweise das LTS des Startgraphen generiert. Im folgenden bezeichnet „R“ die Menge der in diesem Abschnitt vorgestellten Regeln:

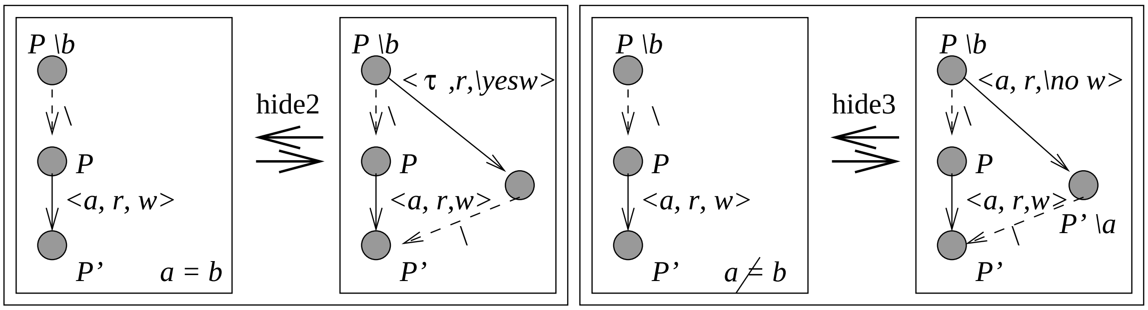

Abbildung 5.10: Abstraktions-Regel–Komposition

$$\begin{aligned} R_{dekomp} &:= \{plus1, para1, hide1, rec1\} \\ R_{komp} &:= \{stop, seq, plus2, para2, para3, hide2, hide3, rec2, rec3\} \\ R &:= R_{dekomp} \cup R_{komp} \end{aligned}$$

5.2.1 Sequentielle Unabhängigkeit

Die komponierenden Regeln zu einem Operatoren sind sequentiell abhängig von der dekomponierenden desselben Operators. Letztere erzeugen die Hilfskanten, die für einen Ansatz der komponierenden Regeln benötigt werden. Somit ist die Reihenfolge dieser Regeln vorgegeben.

5.2.2 Parallele Unabhängigkeit

Hier müssen lediglich die löschenden Regeln untersucht werden, da erzeugende Regeln untereinander nicht im Konflikt stehen können.
Nur die jeweils ersten Regeln zu einem Operator löschen ein Objekt aus ihrem Ansatz (die *not expanded*–Schlinge). Sie sind aber untereinander trotzdem parallel unabhängig wegen der Bedingungen an attributierte Graphmorphismen. Die in den linken Regelseiten vorkommenden Knotenattribute setzen sich aus Variablen und einem Terminalsymbol der abstrakten Sprache zusammen. Ein Ansatz auf dem Attributteil wird durch Unifizierung der Terme gefunden („pattern matching“), wodurch ausgeschlossen ist, daß dekomponierende Regeln zu unterschiedlichen Operatoren einen gleichen Ansatz im Graphen haben können. Sie löschen also nicht aus dem Ansatz der anderen Regel, da sie keinen gleichen Ansatz haben.
Zu den erzeugenden Regeln sind sie unabhängig, da sie etwas löschen, was für

einen Ansatz einer komponierenden Regel nicht gefordert wird: die *not expanded*–Schlinge. Somit ist gezeigt, daß alle Regeln untereinander parallel unabhängig sind, da erzeugende Regeln diese Eigenschaft natürlicherweise besitzen und die restlichen (dekomponierenden) aus oben genannten Gründen.

Für die Anwendung der Graphregeln bedeutet das, daß ihre Reihenfolge beliebig ist sofern sie gleichzeitig anwendbar sind. Man kann somit wahlweise per Breitensuche, Tiefensuche oder unstrukturiert vorgehen, z.B. die Generierung des LTS eines Subprozesses vorziehen, der von besonderem Interesse ist.
Eine weitere Konsequenz aus der parallelen Unabhängigkeit und der Tatsache, daß die LTS aller Subprozesse generiert werden, ist, daß die Anwendung der Regeln aus R verteilt stattfinden kann. So könnte man die Erzeugung des LTS von Subprozessen z.B. auf unterschiedliche Rechner aufteilen und später diese in geeigneter Weise komponieren.

5.2.3 Optimierung des Verfahrens

Im Laufe dieses Verfahrens kann es vorkommen, daß Knoten erzeugt werden, die mit demselben Attribut versehen sind. Zum Beispiel bei dem Prozeß $P + P$, wobei P für einen beliebigen Prozeß steht. Aus Optimierungsgründen bietet es sich in einem solchen Fall an, die Knoten mit gleicher Attributierung zu verkleben. Die Regel $kontroll$ realisiert diese Verklebung und löscht dabei eine *not expanded*–Kante. Sie darf nicht mit einem verklebenden Ansatz angewandt werden, was durch die *nab*–Regel in Abbildung 5.11 ausgedrückt wird. Sie soll also nur an einem injektiven Ansatz angewandt werden, d.h. nur wenn wirklich zwei verschiedene Knoten mit der gleichen Attributierung existieren. Ohne diese Einschränkung könnte sie mit nicht–injektivem Ansatz an einem einzelnen Knoten mit der *not expanded*–Schlinge angewandt werden, womit die Ableitung des LTS dieses Prozesses fehlerhaft unterbrochen wäre.
Diese Identifizierung hat zur Folge, daß nun mehrere Hilfskanten von verschiedenen Prozessen auf diesen Knoten zeigen. Da komponierende Graphregeln aber nicht löschen, beeinflussen sie sich nicht untereinander. Somit gibt es durch diese Verklebung keine ungewollten Nebeneffekte, sondern die Ableitungen bleiben voneinander parallel unabhängig.

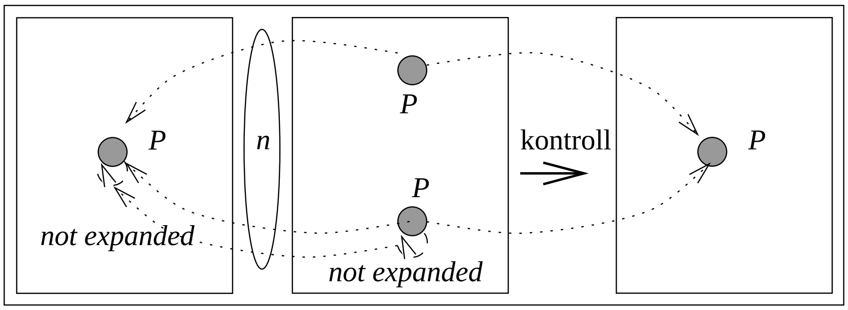

Abbildung 5.11: Kontrollfluß-Regel

Eine Knotenidentifizierung wird auch durch die Regel $new-kontroll$ aus Abbildung 5.12 beschrieben. Im Gegensatz zu obiger Regel fordert sie nicht die *not expanded*–

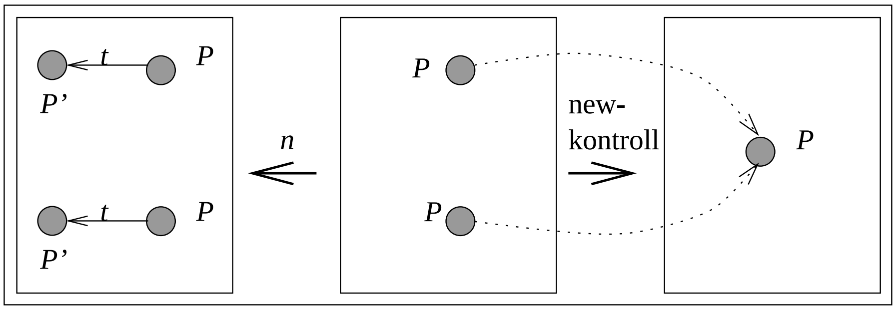

Abbildung 5.12: Kontrollfluß-Regel

Schlinge. Betrachten wir die komponierenden Regeln nochmals genauer, so werden in $para2$, $para3$, $hide2$ und $hide3$ in der rechten Regelseite neue Knoten erzeugt, an denen daher selbstverständlich keine Schlinge hängt. Es ist aber nicht ausgeschlossen, daß weitere Knoten mit demselben Attribut wie die erzeugten Knoten schon in dem Graphen existieren. Einerseits aus „Platzersparnis“ bzw. kompakterer Darstellung sollte man solche Knoten identifizieren, andererseits aber auch, um nicht endlos immer den gleichen Prozeßknoten zu expandieren.
Hier liegt aber auch eine Gefahr: Erfolgt die Verklebung „zu spät“, verfälscht sie das LTS dieses Prozesses. Ein Gegenbeispiel veranschaulicht dieses:

> Angenommen, man hätte in einem Graphen den Teilgraphen $P \xrightarrow{a, \lambda, w} Q$ zweimal, da man zweimal den Knoten P abgeleitet hätte. Eine Identifikation der Knoten von P und der Knoten von Q liefert den Teilgraphen $P \overset{a, \lambda, w}{\underset{a, \lambda, w}{\rightrightarrows}} Q$ mit einer Vereinigung des Kontextes der beiden P–Knoten

und Q–Knoten. Ermittelt man nun die zugrundeliegende Markow–Kette führt das zu dem falschen Ergebnis $P \xrightarrow{2\lambda} Q$.

Eine Lösung dieses Problems ergibt sich, wenn man Knoten mit dem gleichen Attribut frühestmöglich identifiziert, also sofort nach der Erzeugung des zweiten bzw. bevor ein neuer Knoten erzeugt wird. Das wird in der Transaktionsvorschrift im kommenden Abschnitt 5.3 berücksichtigt. Die negativen Anwendungsbedingungen der Regeln stellen dann sicher, daß ein oben geschilderter Fehler nicht eintreffen kann.

Unter Beachtung voriger Überlegungen hat die Anwendung von Graphregeln mindestens einen großen Vorteil gegenüber der Anwendung von Ableitungsregeln des herkömmlichen Kalküls: die explizite Optimierungsmöglichkeit. Bei der Ableitung des LTS durch das Kalkül könnte man zwar den Programmcode in ähnlicher Weise optimieren, was aber versteckt bleibt, da über die Generierung nichts ausgesagt wird durch Deduktionsregeln. Bei Graphregeln ist die Optimierung hingegen aufgrund des konstruktiven Ansatzes explizit darstellbar.

5.2.4 Labelled Transition System

Diese Konstruktivität von Graphregeln hat allerdings ihren Preis: Bei den Ableitungen werden Hilfskanten gezogen und auch die Zustandsübergänge der Subprozesse erzeugt. Das heißt, daß durch die Anwendung der Regeln ein Graph generiert wird, von dem nur ein Teil die relevanten Informationen beinhaltet. Diesen Graphen nennen wir im folgenden den *LTS–Graphen*. Das LTS[1] des Startprozesses muß nun extrahiert werden. Nur der über Aktionskanten mit dem Startknoten verbundene Teilgraph ist das LTS des Prozesses. Hilfskanten sind dabei irrelevant. Man „wandert“ sozusagen ausgehend vom Startknoten auf den Aktionskanten durch den Graphen. Alle Knoten, die so erreicht werden, nimmt man als Zustände in das LTS auf, die Aktionskanten als Zustandsübergänge des LTS.
Eine formale Beschreibung zum Herausfiltern des LTS des Prozesses P_0 aus einem Graphen $G = (E, V_A, V_H, s_A, t_A, s_H, t_H)$ läßt sich induktiv formulieren. Dabei steht G für den durch die Regeln erzeugten Graphen, V_A sind Aktionskanten mit Quelle s_A und Ziel t_A, analog sind V_H Hilfskanten. Die Attribute von Knoten $w(e)$ sind stochastische Prozesse der abstrakten Sprache, das Attribut $w(v_A)$ einer Aktionskante v_A

[1] Das LTS steht hier kurz für die relevanten Zustände und Zustandsübergänge.

ist das Tripel $t \in Com \times I\!R^+ \times W^*$. Das LTS des Prozesses P, $\mathcal{T}(P) = (\mathcal{L}, P, \rightarrow)$, genauer seine Zustandsübergänge und relevanten Zustände, die vom initialen Zustand aus erreichbar sind, ergibt sich wie unten. Die Hilfsmenge Z_0 enthält dabei die erreichten Zustände:

1. Zustandsmenge Z_0 : $P_0 \in Z_0$. Der abzuleitende Prozeß ist erreichbar durch den leeren Zustandsübergang.

2. Zustandsübergänge aus $\rightarrow$ ergeben sich folgenderdermaßen:
   Falls $S := w(s_A(v_A)) \in Z_0$, dann $Z := w(t_A(v_A)) \in Z_0$ und $\langle S, w(v_A), Z \rangle \in \rightarrow$. Das bedeutet, falls von einem Element (Knoten S) der Zustandsmenge ein Zustandsübergang (Aktionskante) in einen anderen Zustand (Knoten Z) möglich ist, dann gehört dieser Zielzustand (Z) zu der Zustandsmenge und der Übergang in die Kantenmenge.

3. Das sind alle relevanten Zustände und Zustandsübergänge.

5.3 Transaktionsvorschrift

Wie bei der Erklärung der Graphregeln schon erwähnt, sollen die Regeln in einer bestimmten Reihenfolge angewandt werden. Zuerst die dekomponierenden, dann die komponierenden Regeln. Diese Vorgehensweise ergibt sich aus der sequentiellen Abhängigkeit der Regeln zu einem Operatoren. Insofern muß man hier keine Reihenfolge „künstlich“ vorgeben.
Mit den Überlegungen zur parallelen Unabhängigkeit ist aber auch eine feste Reihenfolge der Regeln aus R nicht erforderlich. Anders verhält es sich, wenn die optimierenden Regeln auch zur Anwendung kommen sollen: Zwischen je zwei Regelanwendungen aus R soll, falls möglich, immer eine Optimierung stattfinden. Damit ergibt sich folgende Transaktionsvorschrift:

$$\text{step} = || R; \{kontroll\}^{\square}; \{new - kontroll\}^{\square} ||$$

Diese Vorschrift besagt, daß eine beliebige anwendbare Regel aus R angewandt werden soll. Danach die Regel *kontroll* „so häufig wie möglich“. Sie kann entweder keinmal, einmal oder zweimal angewandt werden, da im vorigen Schritt maximal zwei neue Knoten mit einer *not expanded*–Schlinge erzeugt wurden, die mit einem bestehenden identifiziert werden können. Zuletzt soll die Regel $new - kontroll$ ebenfalls maximal oft angewandt werden: das ist keinmal oder einmal, da eine Regel aus R maximal einen Knoten ohne *not expanded*–Schlinge erzeugt. Pro Transaktion können also rein rechnerisch maximal vier Regelanwendungen durchgeführt werden. Kann keine Regel aus R zur Anwendung kommen, schlägt die Transaktion fehl, und das Verfahren zur Generierung des LTS–Graphen terminiert.

Der LTS–Graph wird durch maximal häufige Anwendung von „step“ generiert. Das bedeutet auch, daß diese Generierung eventuell nicht terminiert, dann nämlich, wenn das LTS des Prozesses unendlich ist. Wenn aber die Zustandsmenge eines Systems unendlich ist, wird eine Analyse und Leistungsbewertung in allen Formalismen äußerst problematisch. So muß man das Verfahren nach einer vertretbaren Anzahl von Berechnungsschritten abbrechen, die sich aus der Komplexität ergibt. Selbst bei einem endlichen Zustandsgraphen kann die Grenze der Berechenbarkeit überschritten werden, was ebenfalls zum Abbruch des Verfahrens führt.
Ein hinreichendes Kriterium für die Endlichkeit des Zustandsraumes eines Modells ist bei stochastischen Prozessen die Endlichkeit der Prozeßdeklarationen. Das bedeutet, daß z.B. für einen indizierten Prozeß eine untere und eine obere Schranke für den Index angegeben ist. Es ist allerdings kein notwendiges Kriterium, was anhand des Beispiels der Warteschlange deutlich wird. Dort ist der Index des Subprozesses $System_i$ in der ersten Version des Modells der Warteschlange nach oben unbeschränkt, das LTS hingegen endlich.

Hier soll es einzig und allein darum gehen, daß das Verfahren der Graphersetzung eine beliebig nahe Approximation des LTS liefert. Im Falle eines endlichen LTS wird es exakt gefunden, wenn nämlich „step“ nicht mehr angewandt werden kann.

5.4 Beispiel: Warteschlange

Nachdem nun die Graphklasse und die Regeln konkret angegeben sind, soll als Beispiel die vereinfachte M/D/1/K/K/FCFS–Warteschlange aus dem Abschnitt 3.2.2 aufgegriffen werden. Dort war auch das LTS dieses Prozesses angegeben, das nun mittels Graphersetzung generiert werden soll. Ausgehend von dem Prozeß

$$Warteschlange := (Poisson||_{\{in\}}Entryplace_0)\,||_{\{entry,loss\}}Server_0$$

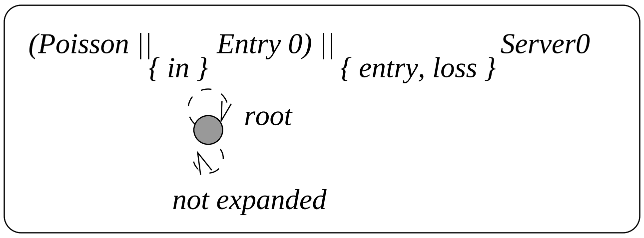

Abbildung 5.13: Startgraph der Warteschlange

als Startgraphen (siehe Abbildung 5.13) soll die Transaktionsvorschrift „step“ solange auf diesen angewandt werden, wie sie anwendbar ist[2]. Die Abbildung 5.14 zeigt die ersten Ersetzungsschritte. Hier können nur dekomponierende Regeln angewandt werden. Es entsteht ein Baum von Subprozessen mit *not expanded*–Schlingen an den Blättern.

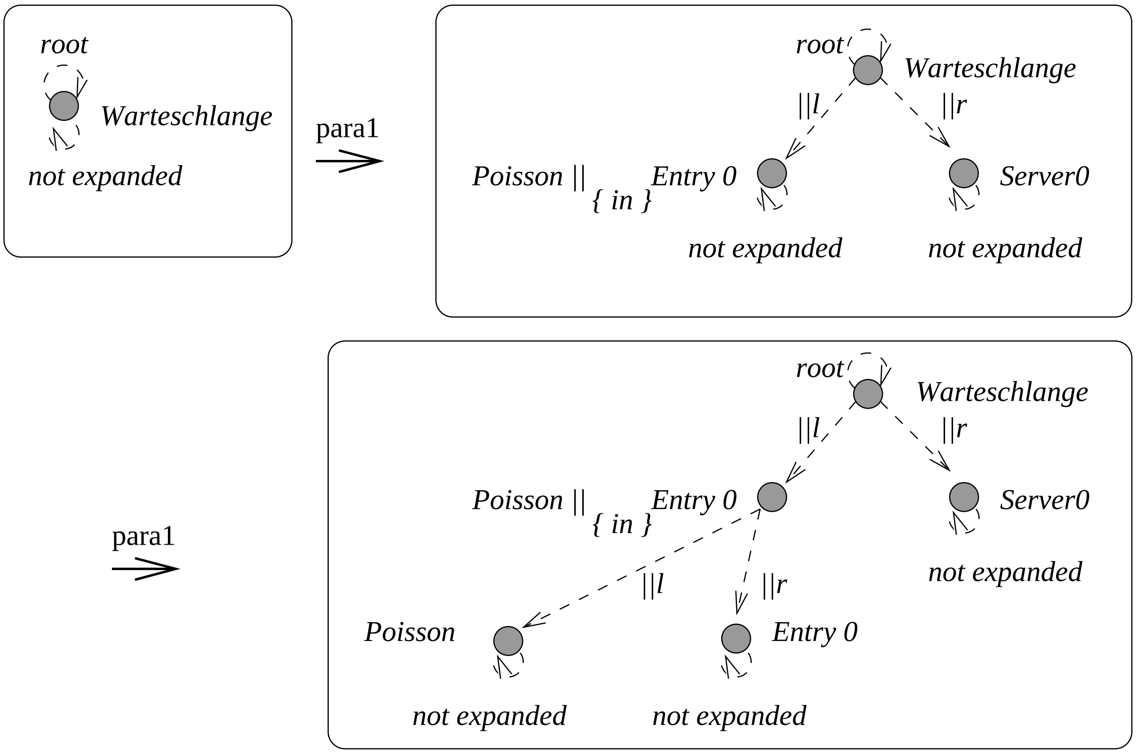

Abbildung 5.14: Warteschlange

[2]In den folgenden graphischen Darstellungen wird der Name „*Entryplace*“ abgekürzt zu „*Entry*“

Gemäß der Transaktionsvorschrift kann man sich nun „zufällig" einen Subprozeß heraussuchen, dessen LTS als nächstes abgeleitet werden soll. Zum Beispiel den Prozeß $recPoisson : (in, \lambda).Poisson$, dessen Ableitung in Abbildung 5.15 zu sehen ist. Keine weiteren Regeln aus R sind auf den letzten Graphen anwendbar, womit auch die Transaktionsvorschrift nicht mehr anwendbar ist. Hier entspricht der LTS–Graph genau dem LTS des Prozesses: Keine nicht zum LTS gehörenden Kanten und Knoten sind in ihm enthalten.

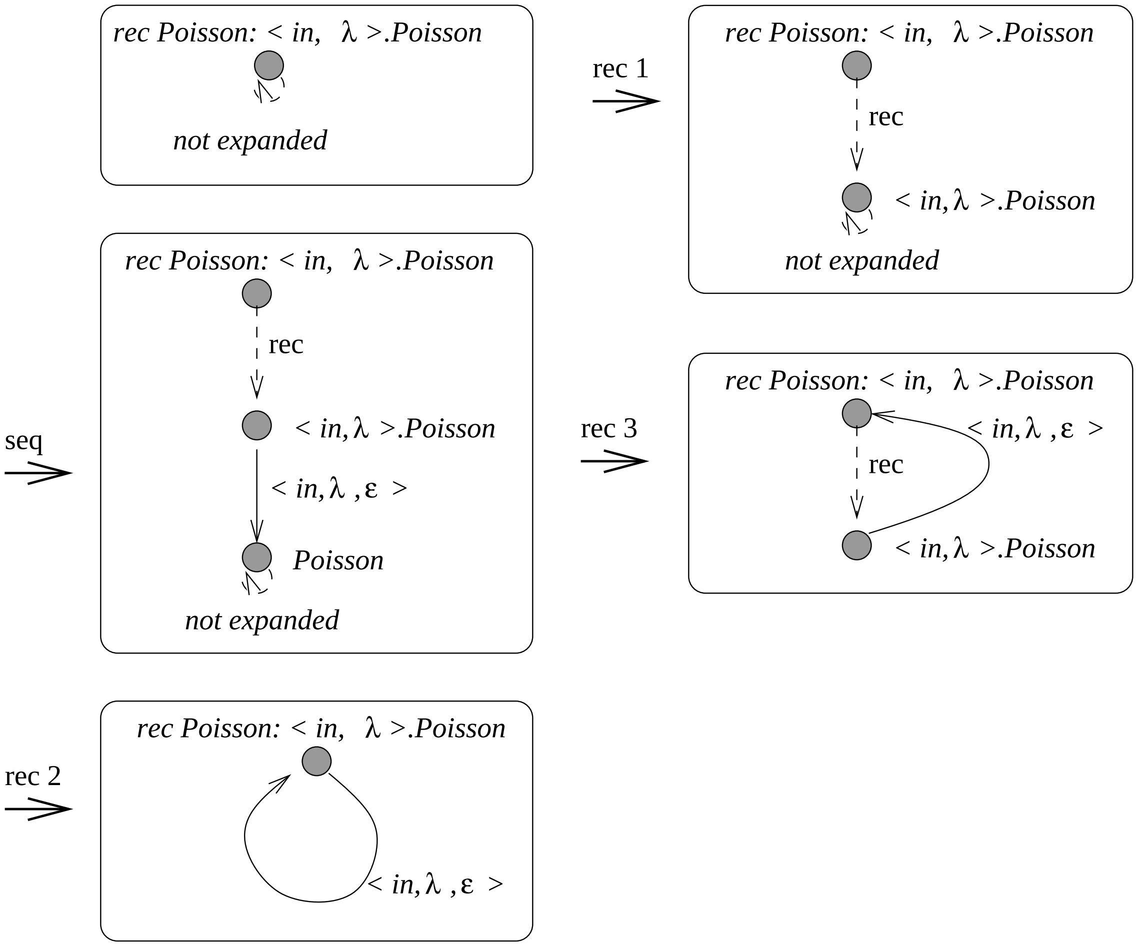

Abbildung 5.15: Ableitung des Prozesses $Poisson$

Analog kann der Subprozeß $Entry_0$ in Abbildung 5.17 abgeleitet werden. Auch hier ist die Ableitungssequenz endlich. Auf den rechts unten stehenden Graphen in Abbildung 5.17 ist keine der Regeln aus R mehr anwendbar. Der entstandene LTS–Graph enthält Hilfskanten und auch Aktionskanten, die nicht zum LTS des

Subprozesses gehören, z. B.:

$$(entry, \delta).Entry_0 \xrightarrow{< entry, \delta, \epsilon >} Entry_0$$

Bevor man nun die komponierenden Regeln weiter anwendet, kann man gemäß der Strategie zuerst das LTS des Subprozesses $Entry_0$ extrahieren. In Abbildung 5.16 ist das geschehen.

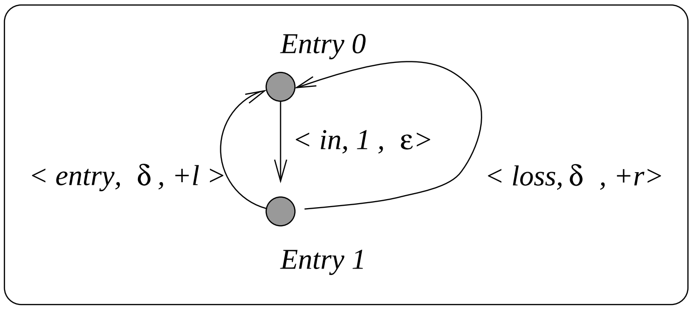

Abbildung 5.16: LTS des Prozesses $Entry_0$

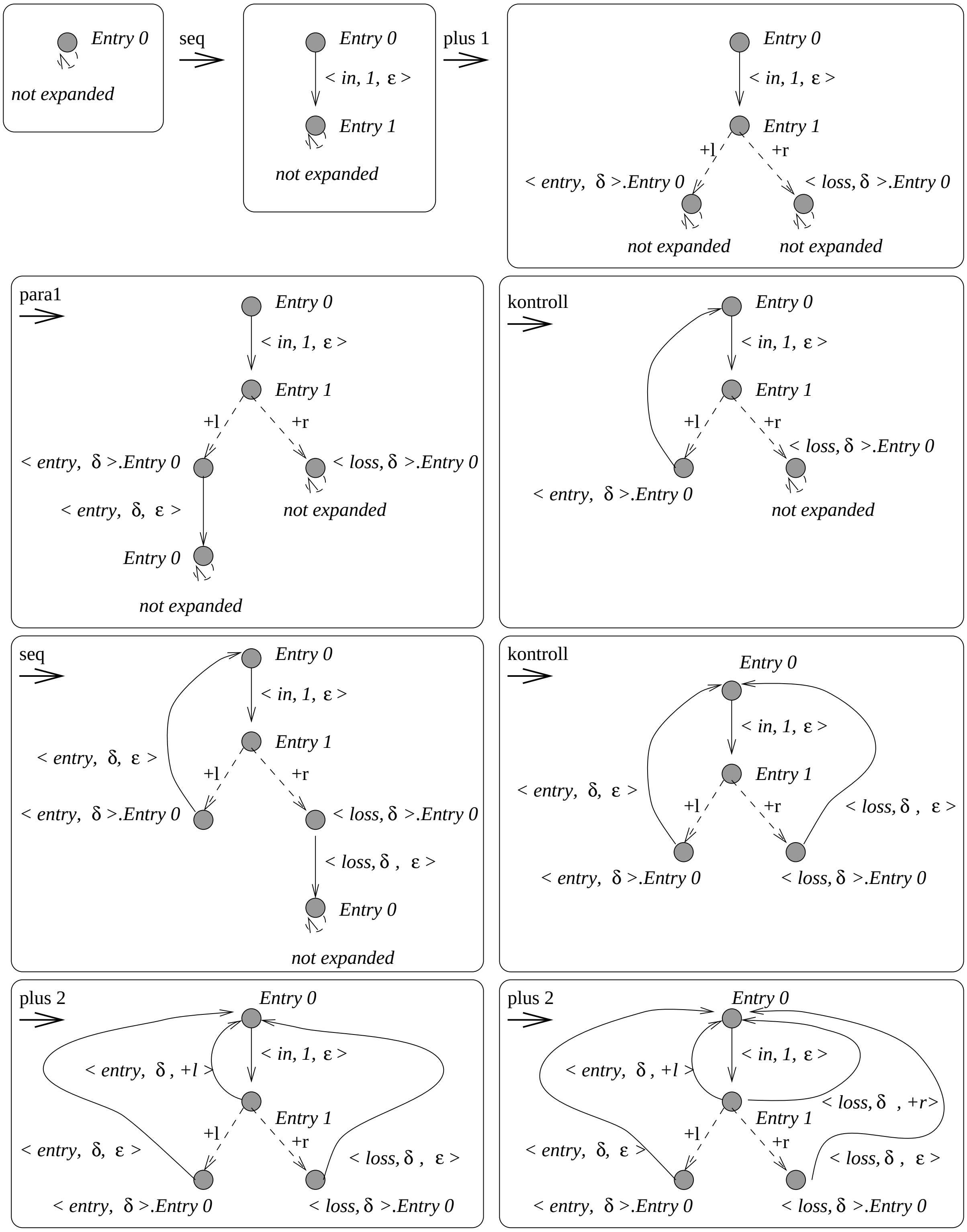

Abbildung 5.17: Ableitung des Prozesses $Entry_0$

Analog verfährt man mit dem dritten Prozeß $Server_0$ und erhält das LTS, das auch in Kapitel 3 in Abbildung 3.9 zu sehen ist.

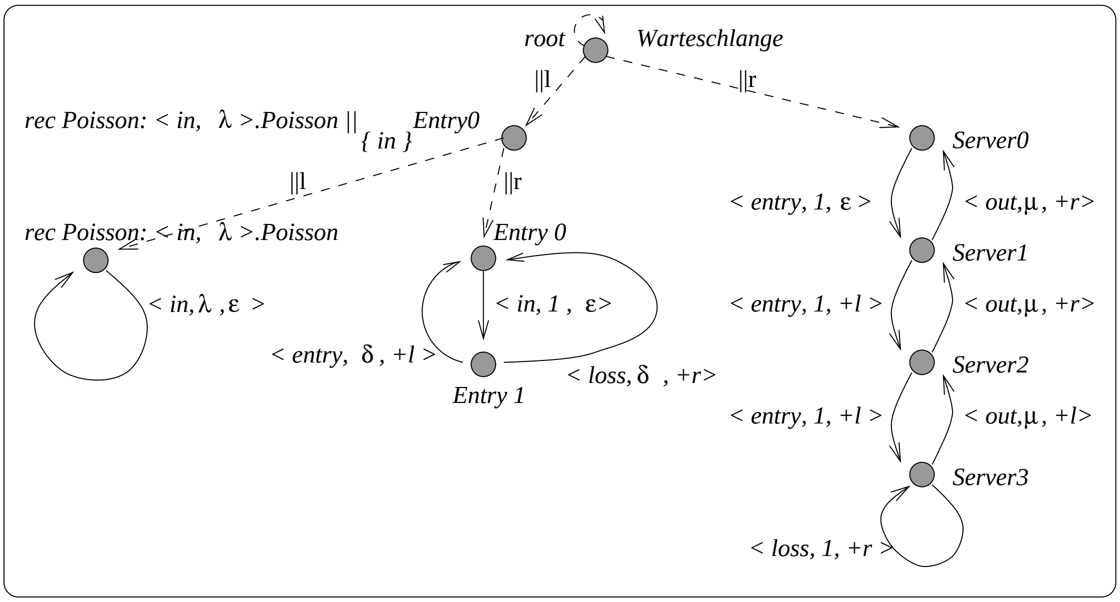

Abbildung 5.18: Warteschlange expandiert

Hat man das LTS aller Subprozesse generiert, kann das LTS des Gesamtprozesses $Warteschlange$ abgeleitet werden. (Eine „unstrukturierte“ Ableitung ist wie erwähnt auch möglich.) Wendet man auf den Graphen in Abbildung 5.18 die Regeln $para2$, $para3$ und $new - kontroll$ der Transaktionsvorschrift entsprechend an, so entsteht der LTS–Graph der $Warteschlange$ (siehe Abbildung 5.19). Auf diesen sind wiederum keine weiteren Regelanwendungen aus R möglich. Das Verfahren terminiert somit und als letzter Schritt erfolgt das Herausfiltern des LTS aus dem LTS–Graphen aus Abbildung 5.19. Das so gefundene LTS des Prozesses $Warteschlange$ ist identisch mit dem in Abbildung 3.10 auf Seite 37, das mit dem herkömmlichen Ableitungskalkül erzeugt wurde.

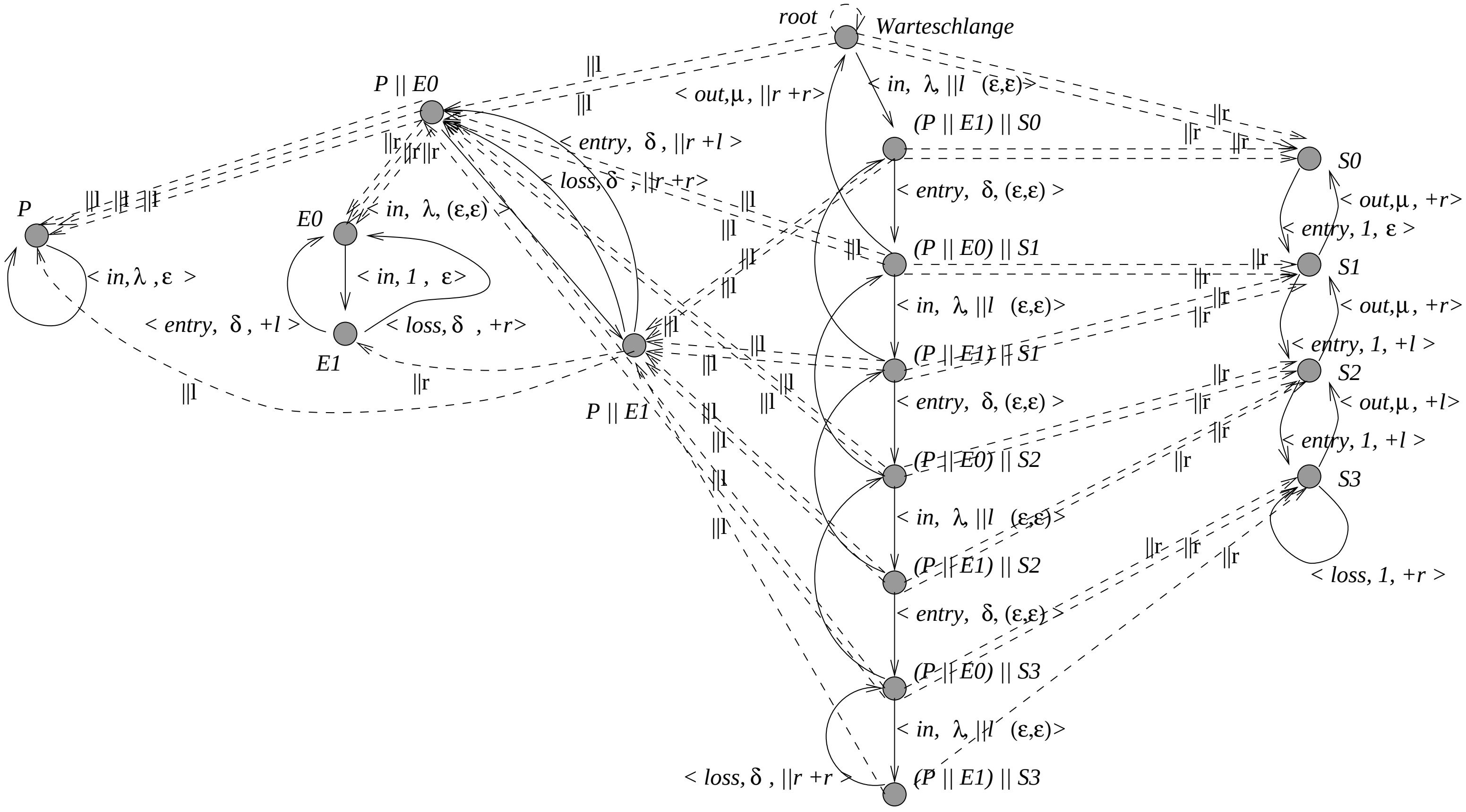

Abbildung 5.19: LTS–Graph der *Warteschlange*

Kapitel 6

Phasentyp–Verteilungen mit Graphgrammatik

In diesem Kapitel werden stochastische Prozeßalgebren um Phasentyp–Verteilungen (PH–Verteilungen) erweitert. Die Generierung des LTS–Graphen soll ähnlich erfolgen zu derjenigen aus Kapitel 5: durch dekomponierende und komponierende Graphregeln. Zunächst folgt eine kurze Einführung in die Problematik, wie Aktionen mit einer PH–Verteilung mit anderen Aktionen interagieren. Dann wird die Graphklasse erweitert, so daß PH–Verteilungen dargestellt werden können. Die Menge der komponierenden Regeln wird erweitert um solche, die den *expandierten LTS–Graphen* erzeugen, der das Verhalten eines Prozesses mit phasentypverteilten Aktionen repräsentiert. Dabei bleibt die Menge der dekomponierenden Regeln unverändert. An einem Beispiel wird dann die Generierung des expandierten LTS–Graphen gezeigt.

6.1 Problematik

Das LTS eines Prozesses bzw. der Erreichbarkeitsgraph eines Petri–Netzes veranschaulicht, welche Aktionen in welchen Zuständen prinzipiell ausführbar sind. In dem modellierten System kommt eine dieser Alternativen dann real zur Ausführung. Wie hier die Auswahl getroffen wird, hängt von bestimmten Strategien ab. Die für diese Arbeit relevanten Strategien werden im folgenden Abschnitt kurz erläutert. Die Expansion einer phasentypverteilten Aktion muß dann diesen Strategien entsprechen.

Zunächst einige Vorüberlegungen, die auch im Bereich der SPN mit Phasentypverteilungen relevant sind: Sowohl dem SPN– als auch dem SPA–Formalismus liegt eine sogenannte *„race policy"* zugrunde. Diese bestimmt, welche Aktion ausgeführt wird in dem Fall, daß die Ausführung mehrerer Aktionen potentiell möglich wäre. „Race policy" besagt, daß in einem solchen Fall diejenige ausgeführt wird, deren Dauer am kürzesten ist. Weil die Dauer einer Aktion hier über eine Exponentialverteilung gegeben ist, kann also nur die Wahrscheinlichkeit angegeben werden, mit der eine Aktionsdauer kürzer als eine andere ist. Die Wahrscheinlichkeit für die Ausführung einer Aktion a mit der Rate λ, die in einem Zustand zusammen mit der Aktion (b, μ) ausführbar ist, beträgt $\frac{\lambda}{\lambda+\mu}$. Allgemein bei n ausführbaren Aktionen (a_i, λ_i) : $\frac{\lambda_i}{\sum_{i=1}^{n} \lambda_i}$

Da PH–Verteilungen komplexe, aus Exponentialverteilungen zusammengesetzte sind, muß geklärt werden, wann eine solche Aktion als abgeschlossen betrachtet werden kann. Die Dauer der Aktion bis zu ihrer Vollendung ist wegen der obigen „race policy" maßgeblich für das Verhalten des Systems. Gemäß der Definition von PH-Verteilungen (siehe Definition 2.1.2) ist es sinnvoll, eine phasentypverteilte Aktion als vollendet anzusehen, wenn der absorbierende Zustand erreicht ist. Bis dahin ist sie unterbrechbar.

Eine weitere Frage, die in diesem Kontext geklärt werden muß, ist, welchen Einfluß die Vollendung einer Aktion a auf vordem auch mögliche (phasentypverteilte) Aktionen $a_i, i \in I\!N$ hat. Hier unterscheidet man drei verschiedene Strategien (siehe auch [Cum85]):

- *„Resampling policy"* bricht die Ausführung aller anderen Aktionen a_i ab, d.h. die Zeit wird sozusagen zurückgesetzt auf den Punkt vor der Ausführbarkeit der a_i. Falls Aktionen auch in dem neuen Zustand ausführbar sind, beginnen sie wieder von vorn.

- *„Age memory policy"* im Gegensatz unterbricht nur die Ausführung der Aktionen a_i. Werden sie erneut ausführbar, setzen sie an dem Unterbrechungspunkt wieder auf.

- *„Enabling memory policy"* stellt eine Mischung aus beiden vorangegangenen Strategien dar: Ist eine Aktion a_i in dem neuen Zustand wieder ausführbar, wird am Unterbrechungspunkt wieder aufgesetzt. Anderenfalls wird sie zurückgesetzt auf den Anfangspunkt.

Unter Verwendung der enabling policy ergeben sich damit zwei verschiedene Fälle für die gleichzeitige Ausführbarkeit z.B. einer exponentiell verteilten und einer phasentypverteilten Aktion. Entweder sind sie „im Konflikt“ zueinander oder nebenläufig. Dabei sind Aktionen a und b im Konflikt, falls die Auswahl von a die Ausführung von b verhindert und umgekehrt[1]. In den Abbildungen 6.1 und 6.2 sind diese veranschaulicht. Ist also z.B. eine Aktion (a, λ) mit exponentiell verteilter Dauer im

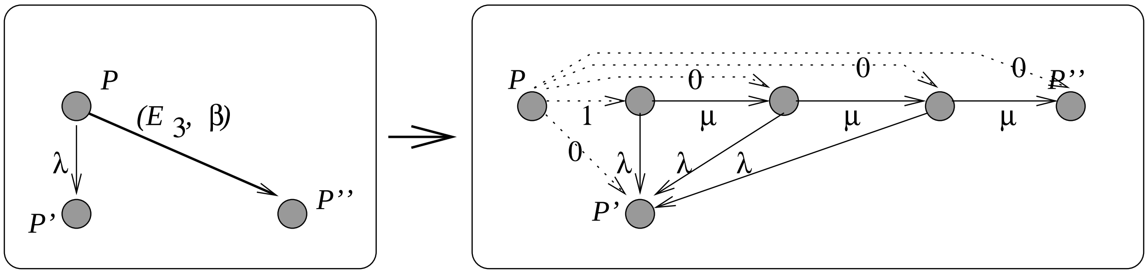

Abbildung 6.1: Konflikt zwischen einer Erlang–verteilten und einer exponentiell verteilten Aktion

Konflikt mit einer phasentypverteilten Aktion $(b, (E_3, \beta))$, kann in jeder transienten Phase der Erlang–Verteilung die Aktion a ausgeführt werden. Damit wird der weitere Durchlauf der Phasen abgebrochen (siehe Abbildung 6.1). Hingegen bei Nebenläufigkeit kann in jeder Phase, also auch der absorbierenden, die parallele Aktion ausgeführt werden, was aber nicht zum Abbruch der phasentypverteilten Aktion führt. Im Gegenteil werden nach Ausführung von a die restlichen Phasen weiter durchlaufen. Der rechte Graph in Abbildung 6.2 veranschaulicht dieses zeit-

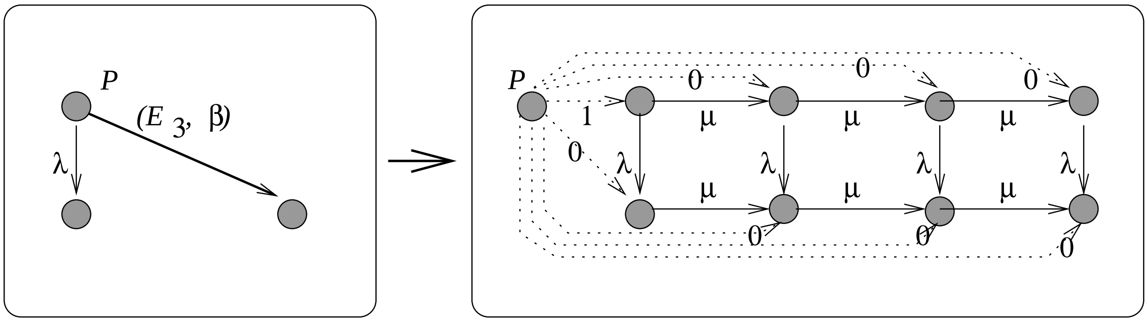

Abbildung 6.2: Nebenläufigkeit zwischen einer Erlang–verteilten und einer exponentiell verteilten Aktion

[1]In dem SPA–Formalismus ist eine Aktion a im Konflikt zu einer Aktion b, wenn die Subprozesse, in denen sie ausführbar sind, durch den „+“–Operator verbunden sind.

liche Verhalten.
Die gepunkteten Kanten in den beiden Abbildungen sind dabei Wahrscheinlichkeitskanten, die zur Repräsentation der Erlang–Verteilung gehören. Sie zeigen an, mit welcher Wahrscheinlichkeit der Prozeß in den entsprechenden Zustand (in Nullzeit) eintritt. Somit weicht die graphische Darstellung einer phasentypverteilten Aktion insofern von der im Kapitel 2 ab, als in diesen Abbildungen auch Wahrscheinlichkeitskanten mit dem Wert „0“ gezeichnet werden.

Aus dieser Problematik wird ersichtlich, daß in einem Zustand, in dem zwei verschiedene Aktionen möglich sind, festgestellt werden muß, ob diese nebenläufig oder im Konflikt zueinander sind. In Abhängigkeit davon soll – mit der enabling Strategie – das LTS unterschiedlich expandiert und so verschiedene Markow–Ketten generiert werden.

Ein weiteres Problem betrifft die („konfliktreiche“ und nebenläufige) Kombination von zwei phasentypverteilten Aktionen. Allgemein gilt für eine Kombination von zwei phasentypverteilten Aktionen, daß ihre Phasen sozusagen miteinander kombiniert werden ($\oplus$). In solch einer kombinierten Phase $A_i \oplus B_j$ sind die Zustandübergänge möglich, die in der Phase A_i möglich sind und diejenigen, die in der Phase B_j möglich sind. Genauer:

$$A_i \oplus B_j \xrightarrow{\quad t_{i,k} \quad} A_k \oplus B_j \qquad \text{für } A_i \xrightarrow{\quad t_{i,k} \quad} A_k$$

beziehungsweise

$$A_i \oplus B_j \xrightarrow{\quad t'_{j,k} \quad} A_i \oplus B_k \qquad \text{für } B_j \xrightarrow{\quad t'_{j,k} \quad} B_k$$

Dabei sind $t_{i,k}$ resp. $t'_{j,k}$ die Raten des Übergangs vom i–ten resp. j–ten in den k–ten Zustand der Phasentypverteilung, die der Aktion a resp. b zugrundeliegt. Die Eingangswahrscheinlichkeit in den Zustand $A_i \oplus B_j$ ergibt sich als Produkt der Eingangswahrscheinlichkeiten α_i und β_j.
Konflikt und Nebenläufigkeit sind analog zu dem oben diskutierten Fall. Sind zwei phasentypverteilte Aktionen im Konflikt zueinander, ist die kombinierte Aktion dann beendet, wenn eine der beiden Aktionen den absorbierenden Zustand erreicht haben. Im Falle der Nebenläufigkeit kann in einem beliebigen Nachfolgezustand ein Phasenübergang stattfinden. Dies bedeutet auch, daß lange nach Beendigung der

einen Aktion – mit beliebig vielen Aktionen dazwischen – die andere (ist der absorbierende Zustand noch nicht erreicht) einen Phasenübergang machen kann. Somit wächst der Zustandsraum drastisch an (um den Faktor der Dimension der Matrix **T** der Phasentypverteilung multipliziert mit der Anzahl der Nachfolgezustände).

6.1.1 Vorgehensweise bei der Erweiterung

Die Frage bei der Erweiterung von SPA um PH–Verteilungen ist also, wann solche phasentypverteilten Aktionen behandelt werden sollen. Dabei ergeben sich folgende Alternativen:

1. Es werden schon bei der Generierung des LTS die speziellen Eigenschaften von phasentypverteilten Aktionen berücksichtigt.

2. Das LTS wird zunächst „symbolisch" abgeleitet, wobei alle Aktionen wie exponentiell verteilte behandelt werden. Danach versucht man dann, diese symbolischen Kanten zu expandieren.

Im zweiten Fall ergeben sich folgende Probleme: Ein Problem ist die Unterscheidung zwischen konfliktreichen und nebenläufigen Aktionen, die aus einem Zustand herausführen. Diese ließe sich anhand des Wortes w der Geschichte der Ableitungen realisieren.
Ein weiteres Problem hängt mit dem veränderten Begriff von Nebenläufigkeit zusammen: Betrachtet man z.B. den Prozeß P:

$$P := (a, \lambda).Stop||_{\emptyset}(b, (\mathbf{T}, \beta)).Stop$$

Dieser Prozeß führt nebenläufig die Aktionen a und b aus, wobei die Dauer von b durch eine PH–Verteilung gegeben ist. Sein symbolisches LTS ist in Abbildung 6.3 zu sehen. Die Expansion müßte allerdings ähnlich zu derjenigen aus Abbildung 6.2 sein. Von jeder Phase müßte ein Zustandsübergang via a möglich sein in den Zustand, in dem die Phasenübergänge äquivalent möglich sind.
Daraus wird ersichtlich, daß eine symbolische Kante nicht nur durch den Graphen der PH–Verteilung ersetzt werden muß, sondern der „Kontext" dieser Kante ist von entscheidender Bedeutung. Im obigen Fall muß man zweimal die b–Aktionskante zu

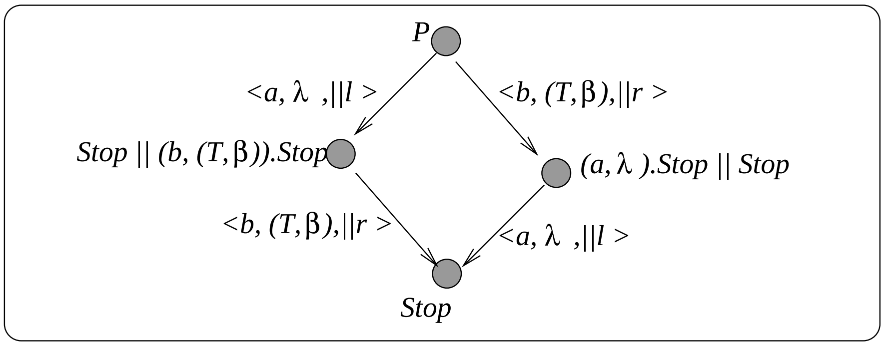

Abbildung 6.3: Symbolisches LTS von P

dem Graphen expandieren und identische Phasen durch eine a–Aktionskante verbinden. Dabei reicht es nicht, nur einen Knoten mit seinen ausgehenden Kanten zu betrachten, sondern auch zumindest die Nachfolgerknoten mit ihren Kanten. Für komplexere Prozesse wird also die Expansion beliebig kompliziert, so daß diese Vorgehensweise für das Verständnis nicht geeignet erscheint.

Daher sollen schon bei der Generierung des LTS eines Prozesses phasentypverteilte Aktionen expandiert werden, was in den folgenden Abschnitten erläutert wird.

6.2 Graphklasse

Hier werden die nötigen Erweiterungen im Hinblick auf phasentypverteilte Aktionen definiert. Die Operatoren der abstrakten Sprache (Sequenz, Nichtdeterministische Auswahl, usw.) bleiben unverändert gegenüber der Definition 3.2.1.
Um Aktionen, deren Dauer durch eine PH–Verteilung gegeben ist, überhaupt kennzeichnen zu können, muß die Ratenmenge um ein Tupel aus Matrix und Wahrscheinlichkeitsvektor vergrößert werden, durch die eine PH–Verteilung eindeutig repräsentiert wird (siehe Kapitel 2).

$$Rate := \mathbb{R}^+ \cup (\mathbb{R}^{k,k} \times [0,1]^k) \qquad k \geq 2$$

Dabei ist $\mathbf{M} \in \mathbb{R}^{k,k}$ eine quadratische Matrix der Dimension $k \geq 2$ mit reellen Einträgen, $\beta \in [0,1]^k$ ein k–dimensionaler Spaltenvektor mit reellen Zahlen aus dem Intervall $[0,1]$.
Eine Aktion mit phasentypverteilter Dauer wird also $(a, (\mathbf{T}, \alpha))$ notiert, wobei $(\mathbf{T}, \alpha)$ die Verteilung repräsentiert. Das „passive Gegenstück“ zu solch einer phasentypver-

teilten Aktion, also eine Aktion, mit der sich eine phasentypverteilte synchronisieren kann, soll wieder als $(a, 1)$ notiert werden.

Abgesehen von passiven Aktionen muß zusätzlich geregelt werden, wie phasentypverteilte Aktionen untereinander und mit anderen kooperieren können. Hier erscheint es weder sinnvoll, daß diese komplexe Aktion mit einer atomaren noch mit einer anderen phasentypverteilten Aktion kooperiert. Denn es ist unklar, was solch eine Synchronisation für die Dauer der gemeinsam ausgeführten Aktion bedeuten sollte. Insofern genügt es, eine phasentypverteilte Aktion $(a, (\mathbf{T}, \alpha))$ nur mit einer passiven Aktion $(a, 1)$ zu synchronisieren. Die passive Aktion soll dabei die Dauer der phasentypverteilten Aktion „erben", d.h. die gemeinsam ausgeführte Aktion hat die Dauer der PH–Verteilung. Alle anderen Kombination werden hier nicht zugelassen.

Die Erweiterung von SPA um PH–Verteilungen wird nun folgendermaßen realisiert. Eine Aktion mit der Rate einer PH–Verteilung soll durch den Graphen mit Eingangswahrscheinlichkeiten dargestellt werden, der die PH–Verteilung repräsentiert. Dazu muß zunächst die Graphklasse erweitert werden: Ein neuer Kantentyp wird dazu eingeführt (mit gepunkteter Linie $\overset{\alpha}{\cdots\cdots\triangleright}$), dessen Attribut eine Wahrscheinlichkeit ist: $\alpha \in [0, 1]$

Auch bietet es sich an, weitere Hilfskanten einzuführen: Eine Schlinge, ($\overset{phase}{- - - - - \triangleright}$), kennzeichnet den Prozeß, der eine phasentypverteilte Aktion ausführen kann. Eine weitere Schlinge, ($\overset{absorb\,(a)}{- - - - - \triangleright}$), markiert den absorbierenden Zustand einer Aktion a. Außerdem noch zwei weitere Kennzeichnungen (ph_l, ph_r), die für die richtige Komposition bei konfliktreichen phasentypverteilten Aktionen benötigt werden. „$sync$" ist eine Kennzeichnung für die Synchronisation. Mit $a \in Com$ erweitert sich die Attributmenge für Hilfskanten zu H':

$$\begin{aligned} H' &:= H \cup \{phase, absorb(a), ph_l, ph_r, sync\} \\ &= \{+_l, +_r, ||_l, ||_r, rec, \backslash, not\ expanded, root, phase, absorb(a), ph_l, ph_r, sync\} \end{aligned}$$

Neben dieser Erweiterung des Attributtyps für Hilfskanten, soll auch die Attributmenge für Knoten vergrößert werden. Zur Benennung von transienten Phasen einer Aktion a wird die Variablenmenge erweitert: Für jede phasentypverteilte Aktion $(a, (\mathbf{T}, \alpha))$ wird eine Menge von Prozeßvariablen aufgenommen. Ihr Name ergibt sich aus dem Aktionsnamen a, der aber als Variablenname mit großem Anfangsbuchstaben geschrieben wird. Außerdem werden sie mit der Nummer der Phase indiziert, also $\{A_1, A_2, \ldots, A_k\} \subseteq Var'$ für $\dim \mathbf{T} = k$.

Somit ergibt sich mit „Var" der herkömmlichen Prozeßvariablenmenge, folgende induktive Definition der erweiterten Variablenmenge Var':

- $Var \subseteq Var'$
- $\{A_1, A_2, \ldots, A_k\} \subseteq Var'$ für $(a, (\mathbf{T}, \alpha)) \in Act \times Rate$ und $\dim \mathbf{T} = k$.

Zusätzlich wird zur Kombination der Phasen von alternativen Aktionen ein „Hilfsoperator" $\oplus : Var' \times Var' \to Var'$ eingeführt. Dieser Hilfsoperator dient nicht zur Zusammensetzung von Prozessen, deren operationale Semantik durch das Gleichnungskalkül abgeleitet werden kann, sondern kombiniert lediglich Prozeßvariablen. Zur Kombination der Phasen von nebenläufigen Aktionen wird der „$||$"–Operator verwandt.

Ein weiterer Hilfsoperator($[\cdot]$) wird zur Kennzeichnung eines Prozeßknotens benötigt, um die Synchronisation korrekt zu behandeln. Die Menge der Knotenattribute „$Vert$" ergibt sich somit aus folgender induktiven Definition:

- $W \in Vert$ für $W \in \mathcal{L}$, d.h. Knoten werden mit Wörtern der abstrakten Sprache attributiert.
- $X \in Vert$ für $X \in Var'$.
- $X \oplus Y \in Vert$ für $X, Y \in Vert$
- $[X] \in Vert$ für $X \in Vert$

Auch die Menge der Aktionsnamen erweitert sich. Für eine phasentypverteilte Aktion $(a, (\mathbf{T}, \alpha))$ wird für jeden Eintrag $t_{i,j} > 0$ in der Matrix $\mathbf{T}$ ein neuer Aktionsname $a_{i,j}$ eingeführt. Es ist der Name der phasentypverteilten Aktion, der indiziert wird mit i, j, falls von der Phase i ein Übergang in die Phase j möglich ist. So kennzeichnet dieser neue Name einer Komponente der Aktion a einen den Indizes entsprechenden Phasenübergang.

Zuletzt soll dem dokumentarischen Charakter des LTS insofern Rechnung getragen werden, als ein weiteres Symbol zur Aktionskantenbeschriftung eingeführt wird (ph). Dieses dient dazu, die Geschichte der Ableitung auch für phasentypverteilte Aktionen festzuhalten. Somit erweitert sich die Menge der „Ableitungswörter“ W'^*, der dritten Komponente der Aktionskante:

$$\begin{aligned} W' &:= W \cup \{ph\} \\ &= \{+_l, +_r, ||_l, ||_r, ||, \backslash_{yes}, \backslash_{no}, ph\} \end{aligned}$$

6.3 Graphregeln für phasentypverteilte Aktionen

Mit dieser Erweiterung um die zulässigen Verteilungen und unter Berücksichtigung der zugelassenen Synchronisationen für phasentypverteilte Aktionen wird das *expandierte LTS* des Prozesses abgeleitet. Ausgehend von dem Startgraphen mit einer *not expanded* und einer *root*–Schlinge, der als Atrribut den abzuleitenden Prozeß trägt, wird wieder schrittweise ein Baum von Subprozessen erzeugt. Die dekomponierenden Regeln R_{decomp} sind auch mit der Erweiterung anwendbar, da sie sich jeweils nur auf den (unveränderten) komponierenden Operatoren beziehen. Da sich außerdem lediglich die Ratenmenge vergrößert hat, sind natürlich alle Regeln auf der bisherigen Ratenmenge $Rate = I\!R^+$ anwendbar. Deshalb müssen nur neue komponierende Regeln für phasentypverteilte Aktionen angegeben werden.
Wir unterscheiden hier mehrere Fälle, die zum Teil schon in der Schilderung der Problematik angesprochen wurden:

1. Eine phasentypverteilte Aktion a ist die einzige, die in einem Prozeß P ausgeführt werden kann.

2. Eine phasentypverteilte Aktion a läuft alternativ zu einer phasentypverteilten oder exponentiell verteilten Aktion ab.

3. Eine phasentypverteilte Aktion a läuft nebenläufig zu einer phasentypverteilten oder exponentiell verteilten Aktion ab.

4. Eine phasentypverteilte Aktion a muß sich mit ihrem passiven Gegenstück $(a, 1)$ synchronisieren.

5. Eine phasentypverteilte Aktion a wird gekapselt, d.h. ist nach außen unsichtbar.

Im folgenden werden diese Fälle behandelt.

6.3.1 Sequenz

Der einfachste Fall ist derjenige, in dem eine phasentypverteilte Aktion die einzige in einem Zustand P ausführbare Aktion ist. Diese Situation wird in der linken Regelseite von $phase1$ erfaßt.

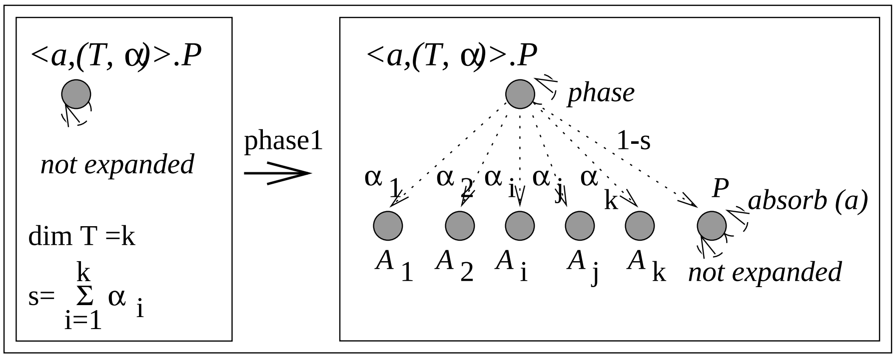

Abbildung 6.4: Expansion einer phasentypverteilten Aktion

Die Regeln $phase1 \ldots phase3$ in Abbildung 6.4 und 6.5 sollen die graphische Repräsentation der PH–Verteilung $(\mathbf{T}, \alpha)$ erzeugen wie sie in Kapitel 2 spezifiziert wurde. Mit $phase1$ in Abbildung 6.4 werden k Knoten erzeugt, falls die Dimension $\dim \mathbf{T} = k$ ist. Diese Knoten stellen die transienten Phasen der Verteilung dar. Das Knotenattribut ist eine Prozeßvariable aus der erweiterten Variablenmenge Var' und bedeutet eine Benennung der Phase. Der Knoten trägt den Namen der Aktion a, der zusätzlich indiziert wird mit der Nummer der Phase, den er repräsentiert. Der absorbierende Zustand hingegen wird mit dem Prozeß P attributiert, der auf die phasentypverteilte Aktion folgt. Zusätzlich wird er als nicht expandiert und als absorbierender Zustand gekennzeichnet, was für spätere Regeln hilfreich ist. Eine Kennzeichnung, eine $phase$–Schlinge, erhält auch der Zustand, in dem die phasentypverteilte Aktion ausführbar ist. Zugleich wird zu jedem dieser Knoten eine Wahrscheinlichkeitskante gezogen, die mit der Eingangswahrscheinlichkeit α_i in die i–te Phase attributiert wird. Auch zum absorbierenden Zustand wird solch eine Wahrscheinlichkeitskante gezogen, wobei sich die Eingangswahrscheinlichkeit aus

$1 - \sum_{i=1}^{k} \alpha_i$ berechnet. Hier werden also auch Wahrscheinlichkeitskanten mit der Attributierung „0" zugelassen. Somit sind alle Phasen, transiente wie auch die absorbierende, die zur Aktion a gehören, mit dem entsprechenden Knoten $(a, (\mathbf{T}, \alpha)).P$ verbunden.

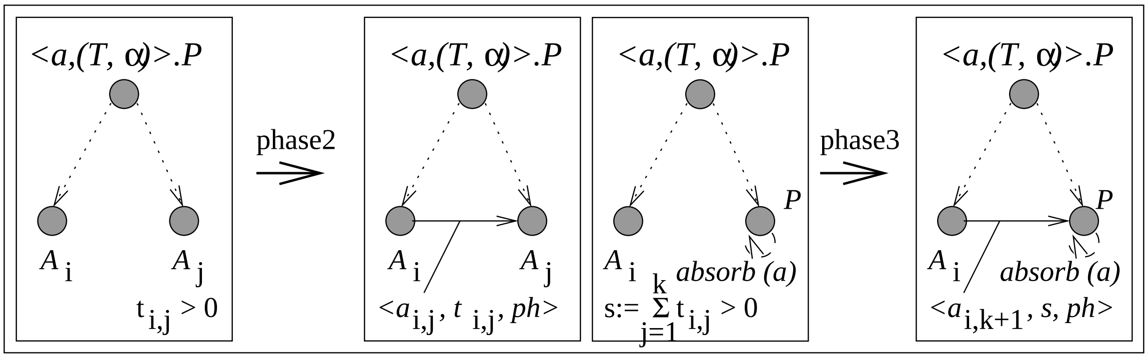

Abbildung 6.5: Expansion einer phasentypverteilten Aktion

Mit den Regeln aus Abbildung 6.5 werden dann die Zustandsübergänge zwischen transienten Phasen (*phase*2) und zwischen transienten und absorbierenden Zustand (*phase*3) behandelt. Dabei werden nur Aktionskanten gezogen, bei denen die Rate $t_{i,j} > 0$ ist, also bei denen wirklich ein Phasenübergang stattfinden kann. Der Name der Aktion, mit dem eine solche Kante beschriftet wird, ergibt sich aus dem Namen der phasentypverteilten Aktion zusammen mit einem doppelten Index i, j für den Zustandsübergang von der i–ten in die j–te Phase.

Bemerkung: Genauer betrachtet, geben die Regeln *phase*1 ... *phase*3 ein Schema für eine Menge von Regeln an, da in Abhängigkeit von der Dimension der Matrix $\mathbf{T}$ und ihren Einträgen der rechte Graph unterschiedlich sein kann. So kann man sie als mit der Matrix und dem Wahrscheinlichkeitsvektor parametrisierte Regeln auffassen.

Beispiel

Als kleines Beispiel wird in Abbildung 6.6 die Semantik des Prozeßes P abgeleitet mit

$$P := (out, \mathbf{E}_3, \alpha).Stop \quad \text{mit } \mathbf{E}_3 := \begin{bmatrix} -3\lambda & 3\lambda & 0 \\ 0 & -3\lambda & 3\lambda \\ 0 & 0 & -3\lambda \end{bmatrix} \text{ und } \alpha := (1, 0, 0)$$

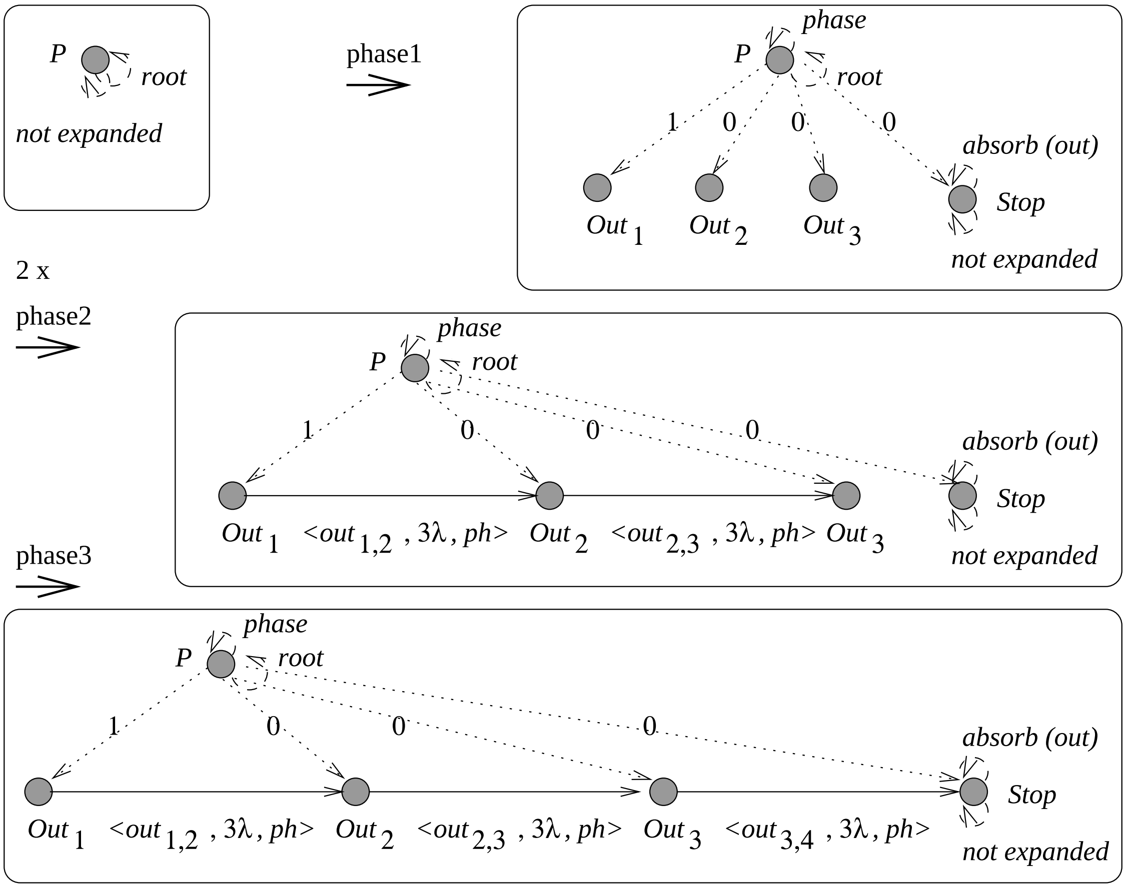

Abbildung 6.6: Expansion einer Erlang–Verteilung $\mathbf{E}_3$

Das Ergebnis der Ableitung ist also ein Graph, der die Erlang–verteilte Aktion *out* mit drei Phasen darstellt.

Komposition allgemein

Zunächst einmal kann man feststellen, daß sich Phasentypverteilungen in gewisser Hinsicht „vererben“. Die Wurzel einer Phasentypverteilung, also der Knoten, der in Kapitel 2 als K_0 bezeichnet wird, ist kein Zustand, in dem Aktionen ausgeführt werden können. Vielmehr ist er ein „Pseudoprozeß“, der für eine Menge von Prozessen steht, nämlich alle diejenigen, in die eine Wahrscheinlichkeitskante führt, deren Attribut $\alpha \neq 0$ ist. Anders ausgedrückt hat die Wurzel einer PH–Verteilung lediglich Symbolcharakter. Diese Eigenschaft, symbolisch für eine Menge von Prozessen zu stehen, pflanzt sich auf Prozesse fort, die einen solchen Pseudoknoten als Subprozeß haben. Es handelt sich also um eine „Vererbung nach oben“. Das soll nicht heißen, daß der zusammengesetzte Prozeß selbst eine phasentypverteilte Aktion ausführt, nur in Bezug auf Eingangswahrscheinlichkeiten in einen Zustand „erbt“ er diese Eigenschaften. Zur Kennzeichnung der Knoten mit obiger Eigenschaft dient die

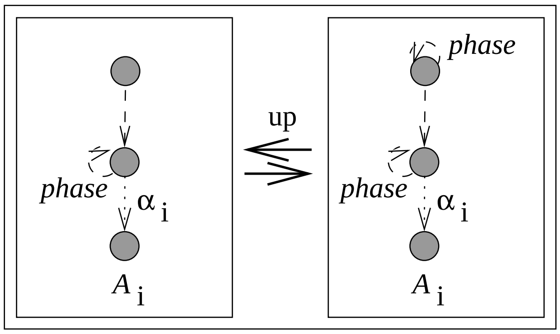

Abbildung 6.7: „Vererbung“ von phasentypverteilten Aktionen

phase–Schlinge. Die *nab*–Regel *up* aus Abbildung 6.7 kennzeichnet einen Prozeß, der mindestens einen Pseudoprozeß als Subprozeß hat, mit einer *phase*–Schlinge. Da die wiederholte Anwendung dieser Regel keinen Informationsgewinn bedeutet, soll sie an einem Ansatz maximal einmal angewandt werden. Diese *phase*–Schlinge gelangt später zur Errechnung der Eingangswahrscheinlichkeiten noch zu weiterer Bedeutung: Von einem so gekennzeichneten Knoten kann zwar eine von den Graphregeln erzeugte Aktionskante ausgehen, sie gehört aber nicht zum LTS des Prozesses.

6.3.2 Nicht–Deterministische Auswahl

Hier werden Graphregeln definiert, die das zeitliche Verhalten zweier Aktionen bei einem „Konflikt“ ableiten. Dabei bedeutet ein Konflikt, daß die Vollendung zweier Aktionen sich gegenseitig ausschließt (race policy).
Die Regel in Abbildung 6.8 behandelt den Fall, in dem zwei phasentypverteilte Aktionen miteinander konkurrieren. Sie erzeugt einen neuen Knoten, für den die Eingangswahrscheinlichkeit als Produkt der Eingangswahrscheinlichkeiten seiner Komponenten ergibt. Er mit einem weiteren „Hilfsattribut“ für Knoten beschriftet wird. Der Operator $\oplus$ zwischen den Knotennamen der Komponenten dieser zusammengesetzten Phase läßt auf die Herkunft dieses Knotens schließen. Um hier eine einheitliche Notation zu gewähren, werden in Abhängigkeit von der Position (links oder rechts) der phasentypverteilten Prozesse in dem zusammengesetzten Prozeß

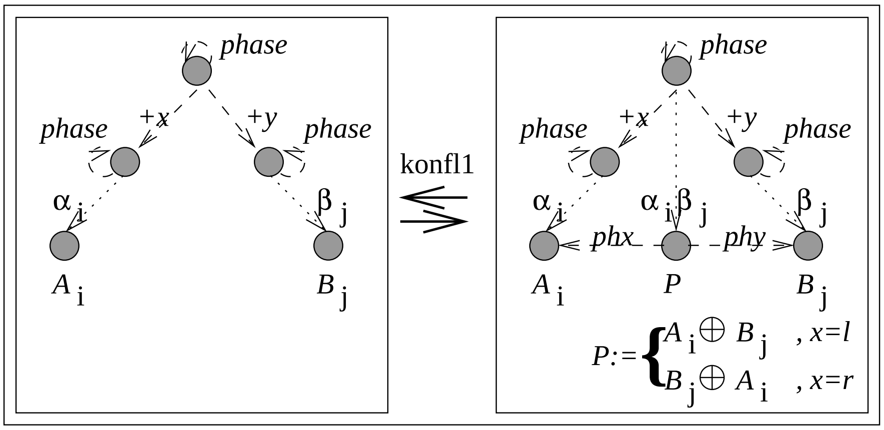

Abbildung 6.8: Phasentypverteilte Aktionen im Konflikt

(P+Q) die Phasennamen an die gleiche Stelle neben den Operatoren $\oplus$ geschrieben. Wesentlich ist hierbei die Möglichkeit, die Komponenten dieser zusammengesetzten Phase wiederzuerkennen. Dieses wird auch durch die neu gezogenen Hilfskanten ph_x und ph_y ermöglicht. So kann in den folgenden (symmetrischen) Regeln $konkl2$ und $konfl3$ ein Zustandsübergang zwischen zusammengesetzten Phasen erkannt und entsprechend behandelt werden (siehe Abbildung 6.9). Solche zusammengesetzten Phasen $A_i \oplus B_j$ können folgende Übergänge vollziehen:

$$A_i \oplus B_j \xrightarrow{(a_{i,k}, \lambda, +_l w)} A_k \oplus B_j \qquad \text{für } A_i \xrightarrow{(a_{i,k}, \lambda, w)} A_k$$

beziehungsweise

$$A_i \oplus B_j \xrightarrow{(b_{j,k}, \lambda, +_r w)} A_i \oplus B_k \qquad \text{für } B_j \xrightarrow{(b_{j,k}, \lambda, w)} B_k$$

Abschließend werden die absorbierenden Zustände der enabling Strategie entsprechend behandelt. Ist von einer Komponente der zusammengesetzten Phase ein Übergang in den absorbierenden Zustand möglich, so kann auch die zusammengesetzte Phase genau diesen Zustand durch den gleichen Übergang erreichen. Graphisch wird dieser Fall in der Regel $konfl4$ behandelt in Abbildung 6.10.

Ein Konflikt nun zwischen einer phasentypverteilten Aktion und einer exponentiell verteilten, kann auf den Konflikt zwischen zwei phasentypverteilten Aktionen zurückgeführt werden, indem die exponentiell verteilte Aktion als phasentypverteilte dargestellt wird. Diese Möglichkeit ergibt sich mathematisch folgendermaßen:

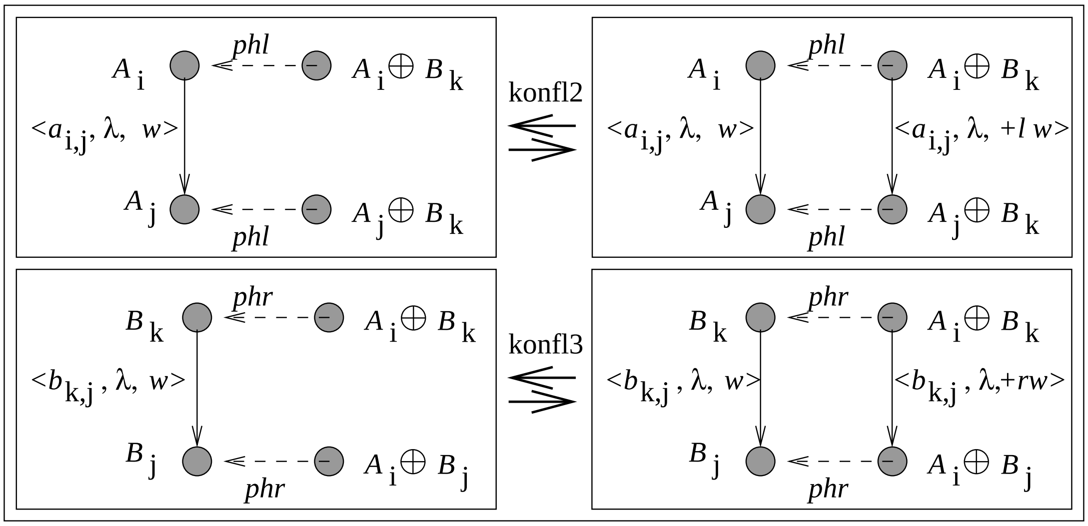

Abbildung 6.9: Phasentypverteilte Aktionen im Konflikt

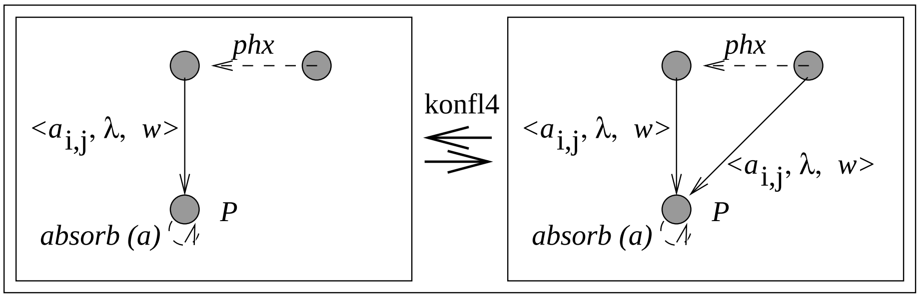

Abbildung 6.10: Phasentypverteilte Aktionen im Konflikt

Eine Exponentialverteilung mit der Rate λ ist äquivalent zu einer Phasentypverteilung mit der Darstellung $\mathbf{T} = [-\lambda]$ und dem Vektoren $\alpha = (1)$ (siehe auch [Lan]). Deshalb genügt hier eine einzige Regel, die eine exponentiell verteilte Aktion durch eine phasentypverteilte darstellt (siehe Abbildung 6.11). Danach sollen dann die Regeln $konfl1, \ldots, konfl4$ angewandt werden.

Gemäß der enabling policy muß festgestellt werden, ob nach Ausführung einer Aktion a, die „alternativ“ zu einer phasentypverteilten Aktion b ist, die zweite wieder ausführbar ist. Dann sollen die durchlaufenen Phasen nicht zurückgesetzt werden. Diesen Fall behandelt die Regel *enable* in Abbildung 6.12. Diese Regel erkennt die Wiederausführbarkeit der Aktion b daran, daß ein zu b gehöriger Phasenübergang in dem neuen Zustand ausführbar ist. So wird mit a ein Übergang vollzogen in die

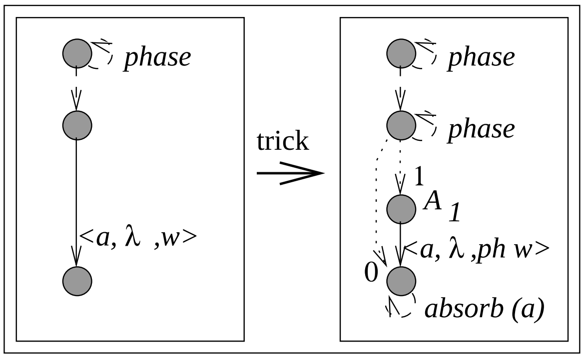

Abbildung 6.11: Umwandlung einer exponentiell verteilten Aktion in eine phasentypverteilte Aktion

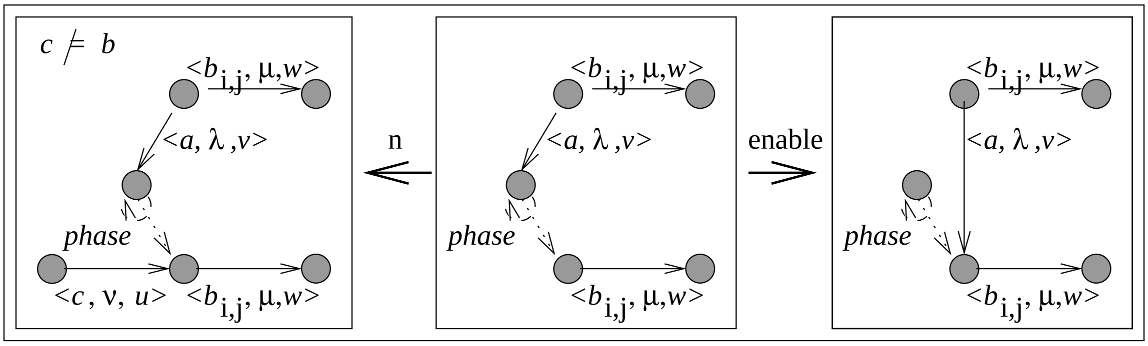

Abbildung 6.12: Aufsetzen nach einer Unterbrechung

entsprechende Phase von b. Dabei ist wieder zu beachten, daß ein Phasenübergang von b im Inneren eines Prozeßes P (z.B. $P := (b, (\mathbf{T}_b, \alpha_b)).P' + (c, (\mathbf{T}_c, \alpha_c)).P''$) stattfinden kann. Deshalb wird mit der negativen Bedingung sichergestellt, daß nur in diejenige zusammengesetzte Phase ein Übergang durch die Aktion a stattfindet, die nicht zugleich durch einen Übergang mittels der Aktion c erreicht werden kann. Sonst würde diese Aktion c sozusagen „übersprungen“ werden.

6.3.3 Parallelität

Die Regeln $neben1$ und $neben2$ behandeln den Fall, daß die phasentypverteilte Aktion nicht kooperiert. Dieser Fall ist ganz analog zu der Regel $para2$ aus Kapitel 5. Falls der Prozeß Q selbst nun wieder mit einer $phase$–Schlinge versehen ist, ist die gleiche Regel $neben$ aus Abbildung 6.13 nochmals anwendbar. Dabei entsteht dann eine Sequenz von Wahrscheinlichkeitskanten, die später (in Abschnitt 6.3.6) geeignet behandelt wird.

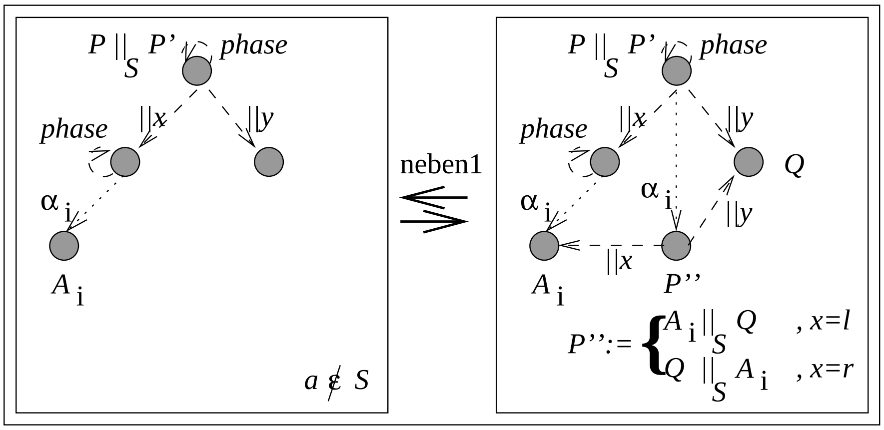

Abbildung 6.13: Nebenläufige phasentypverteilte Aktionen: transiente Zustände

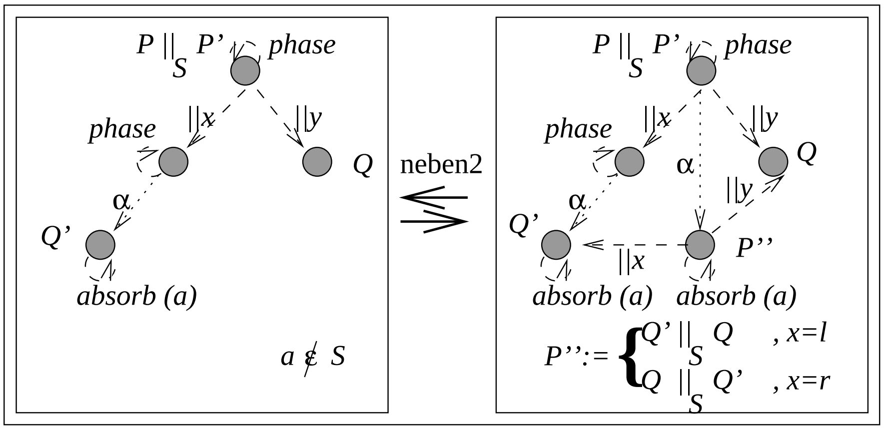

Abbildung 6.14: Nebenläufige phasentypverteilte Aktionen: absorbierender Zustand

Für absorbierende Zustände wird die Regel $neben2$ angewandt aus Abbildung 6.14. Der erzeugte Knoten repräsentiert zwar keinen absorbierenden Zustand, wird aber dennoch mit einer $absorb$–Schlinge versehen zur Kennzeichnung, daß die komplexe Aktion a in diesem Zustand vollendet ist.

6.3.4 Synchronisation

Im Falle der Synchronisation der phasentypverteilten Aktion a mit ihrem passiven Gegenstück werden die folgenden Regeln angewandt.

Synchronisation soll hier bedeuten, daß die passive Aktion solange wartet, bis die aktive den entsprechenden absorbierenden Zustand erreicht hat. Der absorbierende Knoten der Aktion a ist durch eine $absorb(a)$–Schlinge gekennzeichnet. Dabei ist zu beachten, daß die aktive phasentypverteilte Aktion a evtl. im Inneren eines Prozesses P ausgeführt wird. Dann werden aber die Knotenattribute den Operatoren entsprechend kombiniert. Anhand der Knotennamen ist also nicht ohne vollständige Fallunterscheidung erkenntlich, ob die Aktion a im Inneren ausgeführt wird[2]. Daher bietet die $absorb(a)$–Schlinge eine elegante Möglichkeit, eine phasentypverteilte Aktion im Inneren eines Prozesses zu erkennen. Zu den Knoten mit dieser Schlinge führt eine Wahrscheinlichkeitskante. In der linken Regelseite von $phsync1$ sind diese Bedingungen gefordert. Ziel ist es, den Graphen, in den Wahrscheinlichkeitskanten von Q' gehen, zu kopieren mit all seinen Knoten und Kanten. Die Kopie wird von den Regeln $sync1 \ldots 4$ aus den Abbildungen 6.15 und 6.16 realisiert. Dabei markiert

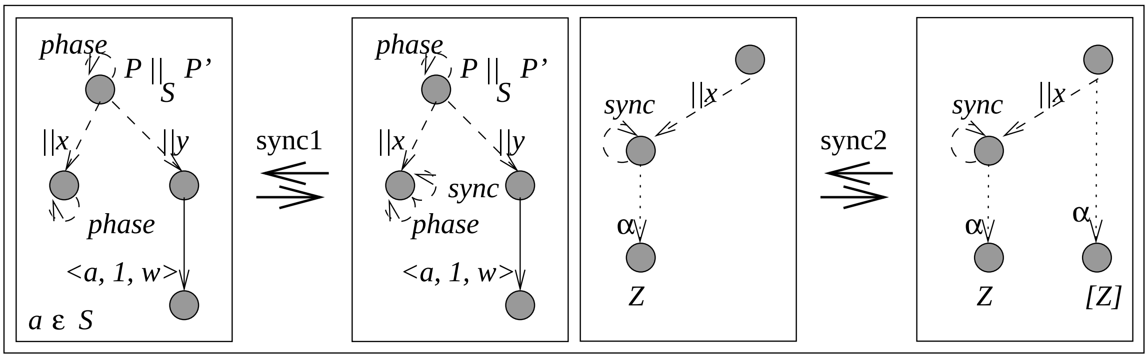

Abbildung 6.15: Synchronisation mit einer phasentypverteilten Aktion

die Regel $sync1$ den Knoten mit einer $sync$–Schlinge, der die Wurzel des zu kopierenden Graphen ist. In Regel $sync2$ werden dann die Knoten kopiert mit den zu ihnen führenden Wahrscheinlichkeitskanten. Dabei weicht der Name der Kopie von dem des Originals ab: Er wird in eckige Klammern gesetzt. Dies verhindert, daß eine der Optimierungsregeln Kopie und Original sofort wieder verklebt. Die dritte Regel $sync3$ kopiert die Kennzeichnung $absorb(x)$, $sync4$ die Aktionskanten. Durch maximal häufige Anwendung dieser Regeln an dem mit der $sync$–Schlinge markierten Knoten, wird der gesamte Graph kopiert. Jetzt müssen die notwendigen Modifikationen vorgenommen werden: In allen Zuständen, in denen die phasentypverteilte Aktion a vollendet ist, können auch die Aktionen ausgeführt werden, die nach der passiven Aktion a ausführbar sind. Dazu werden die absorbierenden Zustände von a mit dem Nachfolger der passiven Aktion parallel kombiniert. Aktionen, die vordem

[2]Hierzu gehören alle Kombinationen $A_i \oplus X, X \oplus A_i, A_i||_S X, X||_S A_i$ bzw. allgemein $X \oplus Y$ bzw. $X||_S Y$, wobei X oder Y eine Kombination ist, in der A_i vorkommt

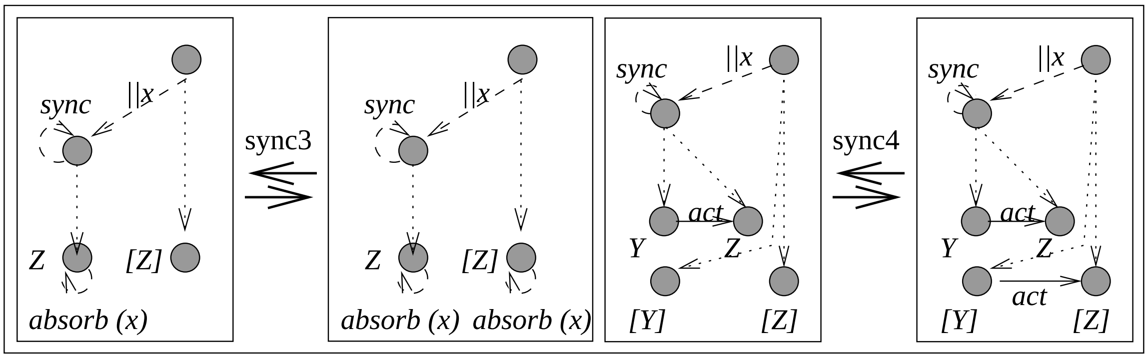

Abbildung 6.16: Synchronisation mit einer phasentypverteilten Aktion

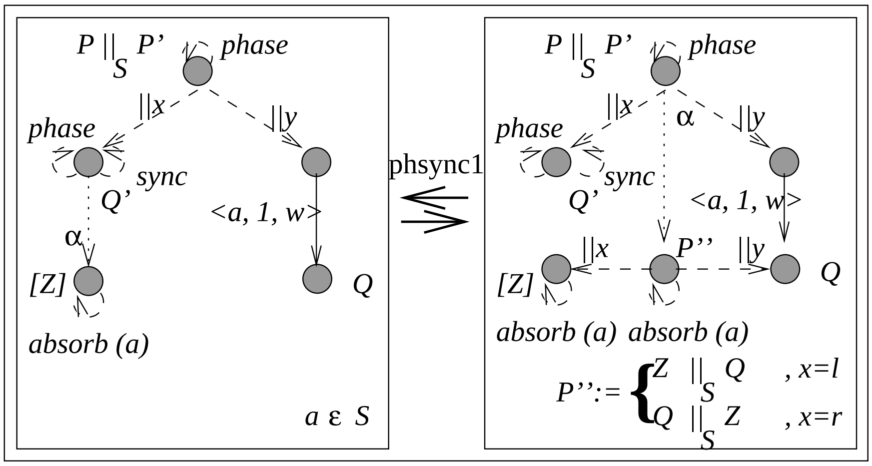

Abbildung 6.17: Synchronisation mit einer phasentypverteilten Aktion

in den absorbierenden Zustand von a führten, führen nun in diesen nebenläufigen Prozeß. Das bedeutet, daß die entsprechenden Kanten umgehängt werden müssen. Die Regel $sync6$ realisiert dieses. Daraufhin müssen die Regeln für den Paralleloperator angewandt werden, so daß diese Synchronisation Auswirkungen auf den gesamten Teilgraphen hat, dessen Wurzel durch eine $absorb(a)$–Schlinge gekennzeichnet ist.

6.3.5 Informationskapselung

Im Falle, daß eine phasentypverteilte Aktion a in einem Prozeß $P\backslash a$ „versteckt“ werden soll, muß jede Aktionskante, die einen Phasenübergang dieser Aktion repräsentiert, umbenannt werden. Wiederum kann die Aktion im Inneren von P

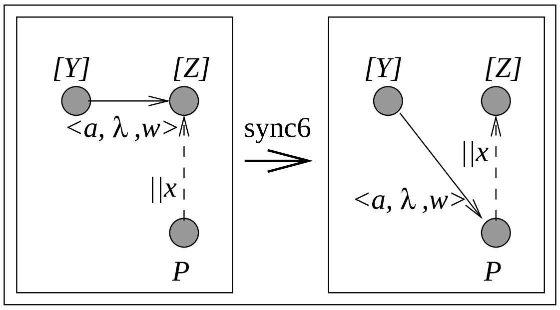

Abbildung 6.18: Synchronisation mit einer phasentypverteilten Aktion

vorkommen. Zunächst sollen dazu die Knoten kopiert werden, in die Wahrscheinlichkeitskanten von P laufen (siehe Regel $phhide1$ aus Abbildung 6.19). Dabei

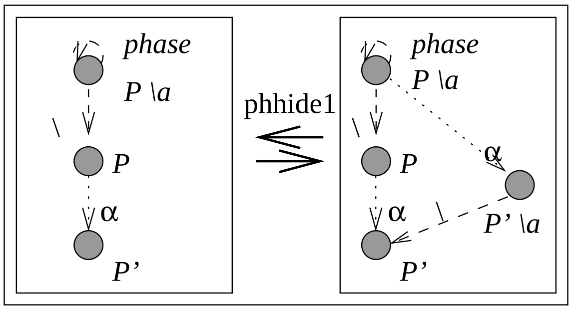

Abbildung 6.19: Informationskapselung einer phasentypverteilten Aktion: Knoten kopieren

werden die Knoten entsprechend umbenannt. Schlingen, die einen absorbierenden Zustand kennzeichnen, werden kopiert und ihr Attribut gegebenenfalls umbenannt. Das hat den gewünschten Effekt, daß diese Aktion a sich nicht mehr synchronisieren kann, da dazu eine $absorb(a)$–Schlinge vonnöten ist, die nun in $absorb(\tau)$ umbenannt ist (siehe Regel $phhide2$ in Abbildung 6.20).
Zustandsübergänge werden ebenfalls modifizierend kopiert: Falls der Übergang mit „$a_{i,j}$“ beschriftet ist, bedeutet das einen Phasenübergang der komplexen Aktion a. Alle diese Teilaktionen werden umbenannt in „τ“. Die Rate hingegen wird übernommen und dem Ableitungswort ein „$\backslash_{yes}$“ vorangestellt. Aktionen, die nicht durch die Informationskapselung betroffen sind, werden kopiert. Die Regeln $phhide3$ und $phhide4$ realisieren diese Fälle.

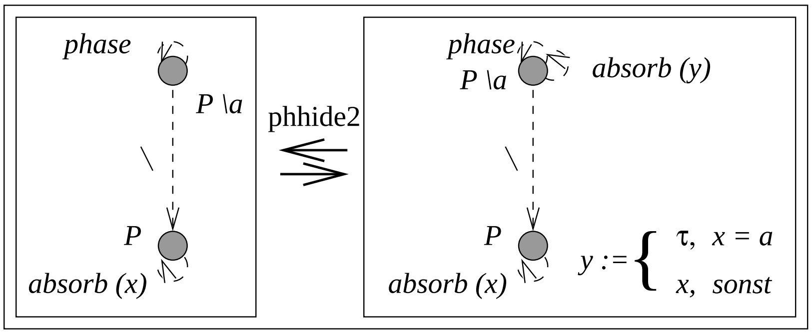

Abbildung 6.20: Informationskapselung einer phasentypverteilten Aktionen: Absorbierende Zustände mit Schlinge

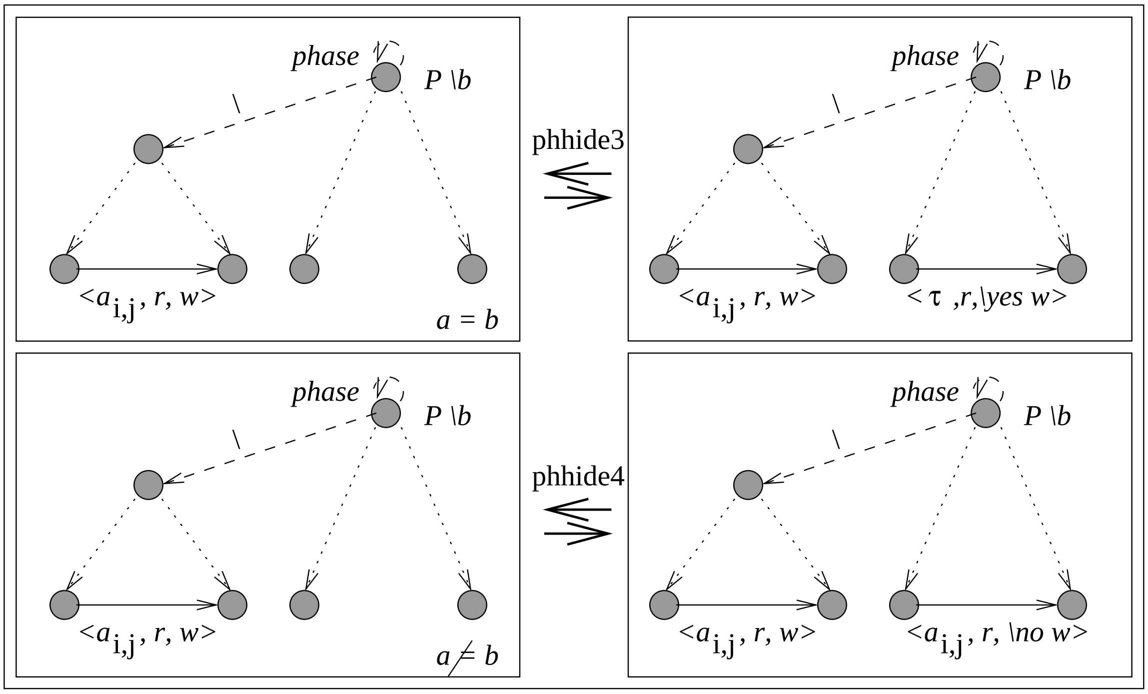

Abbildung 6.21: Informationskapselung einer phasentypverteilten Aktionen: Kanten kopieren

6.3.6 Wahrscheinlichkeiten

Hat man nun mit den vorangegangenen Regeln einen Graphen erzeugt, so unterscheidet der sich insofern von einem LTS–Graphen, als er zwei verschiedene Typen von Kanten enthält, Aktionskanten und Wahrscheinlichkeitskanten. Die Wahrscheinlichkeit eines Eintritts von einem Zustand P in einen Zustand Q wird nun mit der Rate der Exponentialverteilung des Zustandsüberganges in den Zustand P „verrechnet“. Mathematisch ergibt sich die Rate eines Zustandsüberganges über einen Pseudozustand hinweg als das Produkt der Rate mit der Wahrscheinlichkeit. Das heißt, es gilt

$$P \xrightarrow{\ \alpha\ } Q \text{ und } V \xrightarrow{(a,\lambda,w)} P \implies V \xrightarrow{\quad (a,\alpha\lambda,w) \quad} Q$$

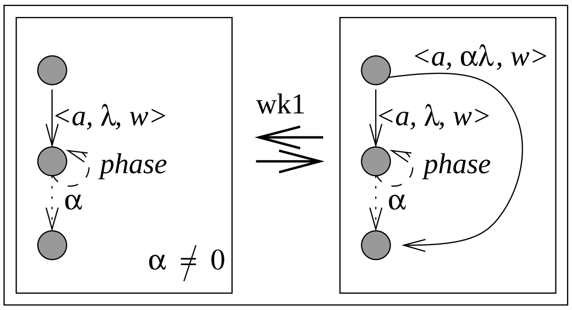

Abbildung 6.22: „Verrechnung“ von Wahrscheinlichkeitskanten

Die Regel $wk1$ in Abbildung 6.22 realisiert diese Berechnung.
Sie erfaßt nicht den Fall, in dem der Wurzelknoten des Graphen mit einer *phase*–Schlinge ausgestattet ist. Dann gibt es keine Aktion, die vor dem initialen Zustand, dem Startknoten, ausgeführt wird. In diesem Fall geben die Knoten, in die vom Wurzelknoten aus Wahrscheinlichkeitskanten führen, die Zustände an, in denen das System beginnt. Die Attribute der Wahrscheinlichkeitskanten gehen dann in den Anfangsvektor $p(0)$[3] ein. Hier kann ein Spezialfall auftreten, wenn nämlich die Parallelregel $neben1$ aus Abbildung 6.13 mehrmals angewandt wurde. Dann sind die Zustände, in die die Wahrscheinlichkeitskanten führen, selber „Pseudozustände“ in dem Sinne, als sie für eine Menge von Zuständen stehen, die mit einer gewissen Wahrscheinlichkeit betreten werden. Für diesen Fall, also einer Sequenz von Wahrscheinlichkeitskanten, müssen die Regeln aus Abbildung 6.23 angewandt werden. Dabei werden die Wahrscheinlichkeiten der aufeinanderfolgenden Kanten multipliziert. Die Summe aber aller Werte der Eingangswahrscheinlichkeiten muß „1“ sein, weshalb die Regel $wk2$ einen Trick anwendet: sie dreht die Kante um, die aus einem Pseudoknoten in einen Pseudoknoten (solche mit einer *phase*–Schlinge) hereinführen. Danach wird das Produkt der Wahrscheinlichkeit der umgedrehten Kante mit der Wahrscheinlichkeit der (vordem folgenden) Kante berechnet und an die

[3] Siehe auch Anhang A. Dieser Vektor geht in die numerische Analyse der Markow–Kette ein.

neue Kante geschrieben (Regel $wk3$). Betrachtet man den oberen Knoten, so ist die Summe der Werte der aus ihm herausführenden Wahrscheinlichkeitskanten wieder „1“. Die Regel $wk2$ ist nur am Ende einer Sequenz von Wahrscheinlichkeitskanten anwendbar, was durch die zusätzliche negative Anwendungsbedingung ausgedrückt wird. Anderenfalls wären die Berechnungen fehlerhaft.

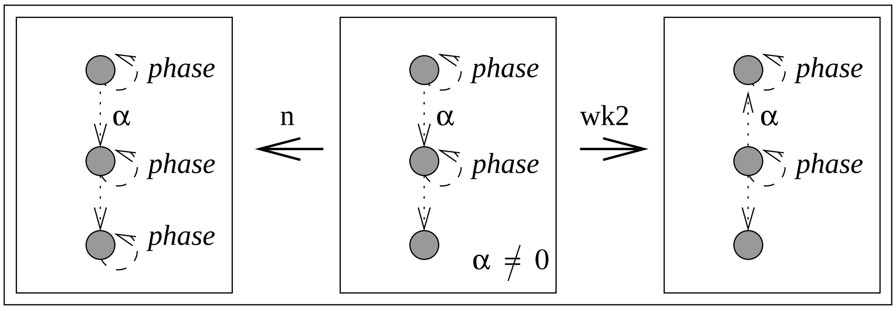

Abbildung 6.23: „Verrechnung“ von Wahrscheinlichkeitskanten

Danach wird das Produkt mit den Werten der Wahrscheinlichkeitskanten gebildet (Regel $wk3$ in Abbildung 6.24). Dabei wird bei der Regel $wk3$ die Konvention aus Kapitel 4 verwandt: sie hat zwei negative Anwendungsbedingungen: Weder darf der unterste Knoten mit einer *phase*–Schlinge gekennzeichnet sein, noch soll die Produktwahrscheinlichkeitskante schon gezogen worden sein.

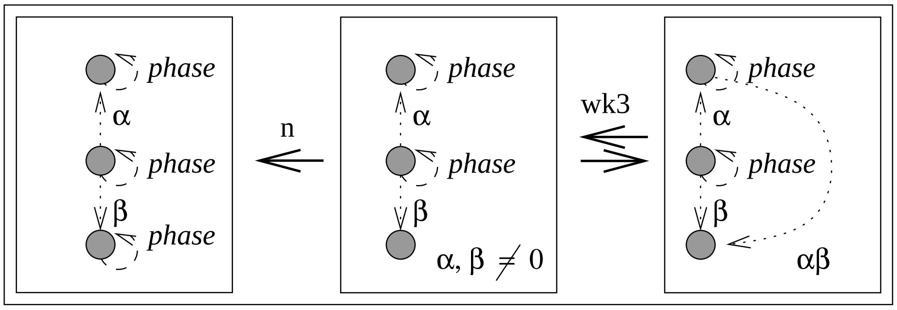

Abbildung 6.24: „Verrechnung“ von Wahrscheinlichkeitskanten

6.3.7 Expanded Labelled Transition System

Abschließend werden alle Knoten, die mit einer *phase*-Schlinge gekennzeichnet sind, mit Ausnahme des Wurzelknotens gelöscht, Dabei werden implizit (im SPO–Ansatz) die ein– und ausgehenden Kanten dieser Knoten mitgelöscht. Die Regel *phdel* realisiert dieses Vorgehen.

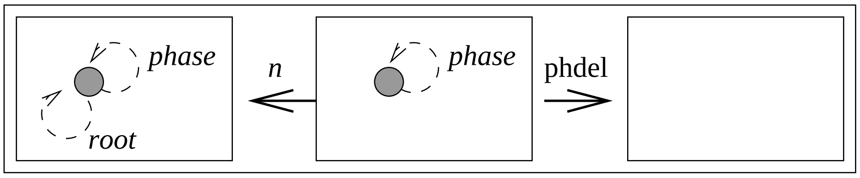

Abbildung 6.25: Löschung der „Pseudozustände“

Das expandierte LTS des Startprozesses ist wieder derjenige Graph, der durch ausgehende Aktionskanten mit dem entsprechenden *root*–Knoten verbunden ist. Falls dieser Knoten ein „Pseudozustand“ ist, also der Knoten mit einer *phase*–Schlinge versehen ist, werden die Zielknoten Z_i der von ihm ausgehenden Wahrscheinlichkeitskanten als Zustände mit in das expandierte LTS aufgenommen. Die Werte der Wahrscheinlichkeitskanten gehen in den Anfangsvektor $p(0)$ ein. Die Graphen, die mit den Z_i durch Aktionskanten verbunden sind, sind das expandierte LTS des Prozesses.

6.4 Transaktionsvorschrift

Die Graphregeln für die Erweiterung aus den vorangegangenen Abschnitten lassen sich in drei Klassen unterteilen:

1. Regeln für die Kombination von phasentypverteilten Aktionen mit anderen durch die Operatoren der abstrakten Sprache ($R_{exp.-comp}$).

2. Spezialregeln für Wahrscheinlichkeitskanten (R_{prop})

3. Regel für Pseudozustand (R_{phase})

$$\begin{aligned} R_{exp.-comp} &= \{phase1, \ldots, phase3, up, konfl1, \ldots, konfl4, trick, neben1, \\ &\quad neben2, phhide1, \ldots, phhide4, sync1, \ldots sync6, wk1\} \\ R_{prop} &= \{wk2, wk3\} \\ R_{phase} &= \{phdel\} \end{aligned}$$

Zusammen mit den schon früher (im Kapitel 5) eingeführten Regeln R_{decomp} und R_{comp}, die den LTS–Graphen eines Prozesses generieren, kann man die Menge R_{lts} der auf die Erzeugung von Aktionskanten ausgerichteten Regeln folgendermaßen definieren.

$$R_{lts} := R_{decomp} \cup R_{comp} \cup R_{exp.-comp}$$

Die Transaktionsvorschrift aus Kapitel 5 erweitert sich dahingehend, daß in einem Schritt eine beliebige Regel aus der erweiterten Regelmenge gefolgt von den verklebenden Optimierungsregeln angewandt werden soll. Diese verklebenden Regeln sollen allerdings nicht auf Phasen A_i angewandt werden, da sich damit die Anzahl der Kanten verfälschen würde wie auch die Eindeutigkeit des absorbierenden Zustands verlorenginge. Daher wird in die linke Regelseite von $kontroll$ und $new - kontroll$ eine Bedingung an den Ansatz gestellt: $P \notin Var' \backslash Var$. Die so modifizierten Regeln werden „$kontroll'$“ resp. „$new - kontroll'$“ genannt.

$$\text{exp.-step} = || R_{lts}; \{kontroll'\}^{\Box}; \{new - kontroll'\}^{\Box} ||$$

Falls diese Transaktionsvorschrift nicht mehr anwendbar ist, werden die Berechnungen auf den Wahrscheinlichkeitskanten durchgeführt: Die Regelmenge R_{prop} soll maximal oft angewandt werden. Zum Schluß werden die Pseudozustände gelöscht mit der Regelmenge R_{phase}.

6.5 Beispiel: Warteschlange

Das Eingangsbeispiel der Warteschlange kann nun im erweiterten SPA–Formalismus insofern realistischer modelliert werden, als mit der Erweiterungen um PH–Verteilungen auch eine deterministische Dauer approximiert werden kann. So wird die deterministische Dauer durch eine Erlangverteilung (der Ordnung 3) angenähert.

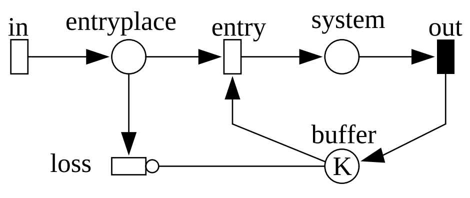

Abbildung 6.26: Warteschlange mit exponentiell verteilten und deterministischen Schaltzeiten

Das Petri–Netz modelliert so ein System, in das Aufträge mit exponentiell verteilter Zwischenankunftszeit eintreffen. Sie warten zunächst eine durch eine Exponentialfunktion gegebene Zeitspanne, bis sie in Abhängigkeit der zur Verfügung stehenden Kapazität ($buffer$) entweder in eine Liste eingereiht ($entry$) oder gelöscht ($loss$)

werden. Während dieser Wartezeit wird kein weiterer Auftrag akzeptiert, was durch die hemmende Kante ausgedrückt wird. Durch den Server werden eingereihte Aufträge mit deterministischer Dauer abgearbeitet.

Analog zur Modellierung im Kapitel 3 wird dieses Petri–Netz als Prozeß *Warteschlange* modelliert. Der Unterschied zur Definition in jenem Kapitel liegt in der Rate der Aktion *out*. Diese ist jetzt durch eine Erlang–Verteilung der Ordnung 3 ($\mathbf{T} = \mathbf{E}_3, \alpha = (1,0,0)$) gegeben.

$$\begin{array}{lll}
Poisson & := & (in, \lambda).Poisson \\
\hline
Entryplace_0 & := & (in, 1).Entryplace_1 \\
Entryplace_1 & := & (entry, \delta).Entryplace_0 + (loss, \delta).Entryplace_0 \\
\hline
Server_0 & := & (entry, 1).Server_1 \\
Server_j & := & (entry, 1).Server_{j+1} + (out, (\mathbf{T}, \alpha)).Server_{j-1} \qquad (0 < j) \\
Server_K & := & (out, (\mathbf{T}, \alpha)).Server_{K-1} + (loss, 1).Server_K
\end{array}$$

$$Warteschlange := (Poisson||_{\{in\}}Entryplace_0)\,||_{\{entry,loss\}}Server_0$$

Abbildung 6.27: Warteschlange mit deterministischer Bediendauer

Hier tritt nun eine der Besonderheiten bei der Modellierung mit komplexen Aktionen auf. Es kann sein, daß man mit dem +–Operatoren nicht einen Konflikt modellieren, sondern ausdrücken wollte, daß zwei Aktionen a und b potentiell ausführbar sind. Ein solcher Fall liegt zum Beispiel bei der Modellierung des Subprozesses $Server_j$ vor. Dieser kann Aufträge entgegennehmen $(entry, 1).Server_{j+1}$ oder bearbeiten $(out, (\mathbf{T}, \alpha)).Server_{j-1}$. Wenn während der Bearbeitung ein neuer Auftrag ankommt, soll die Bearbeitung nicht abgebrochen, sondern nur unterbrochen werden (enabling policy). Während bei exponentiell verteilten Aktionen aufgrund der Atomarität nicht zwischen Abbruch und Unterbrechung unterschieden werden kann, ist dies sehr wohl der Fall bei phasentypverteilten Aktionen. Diese modifizierte Bedeutung des +–Operators wird durch die Regel *enable* aus Abbildung 6.12 umgesetzt.

In Abbildung 6.28 wird der $Server_1$–Prozeß schrittweise expandiert. Dabei ist die Expansion einer 3–phasigen Erlang–Verteilung aus Abbildung 6.6 benutzt worden. In Abbildung 6.29 ist dann die Regel *enable* aus Abbildung 6.12 angewandt worden.

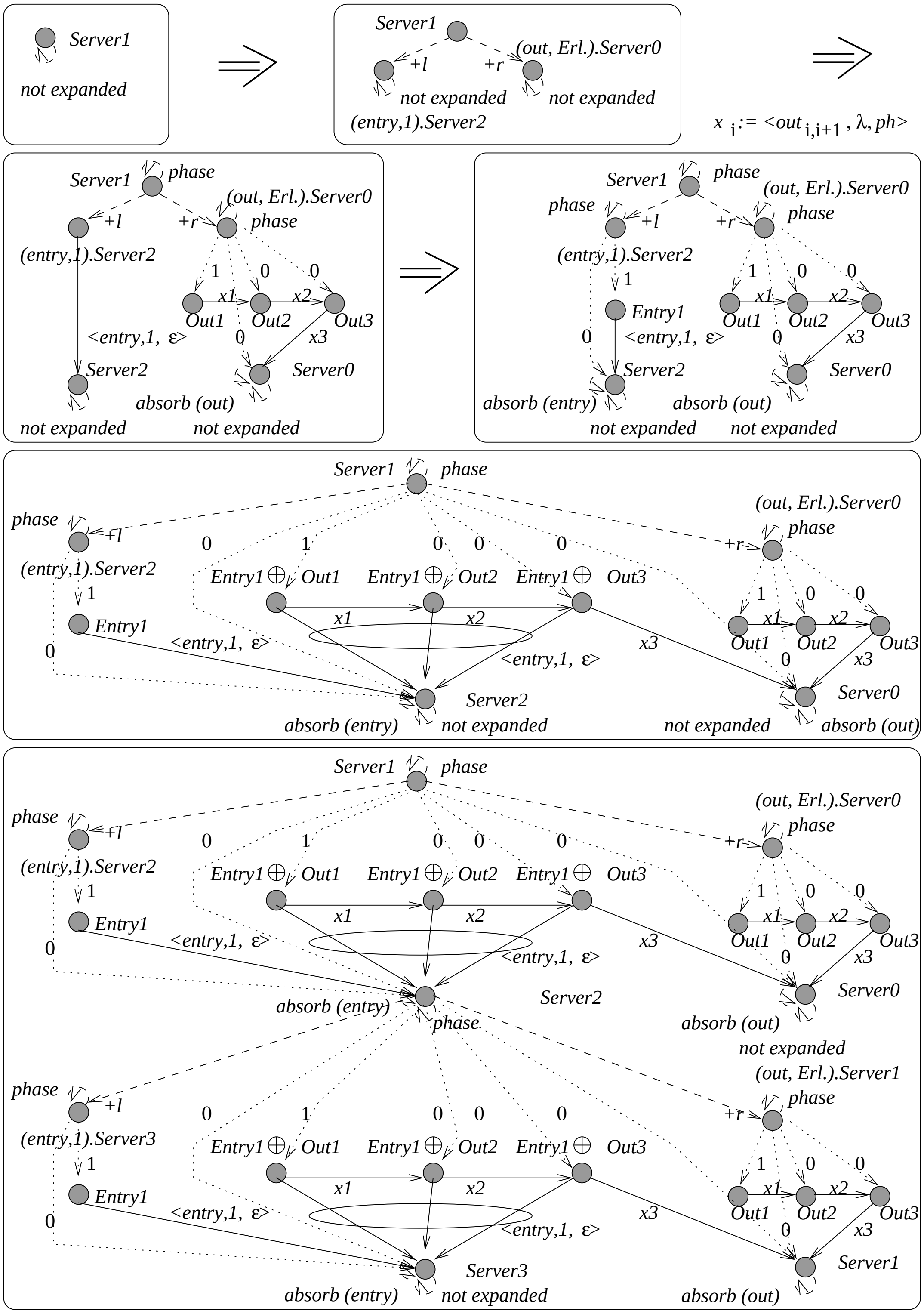

Abbildung 6.28: Expansion des $Server_1$–Prozeßes

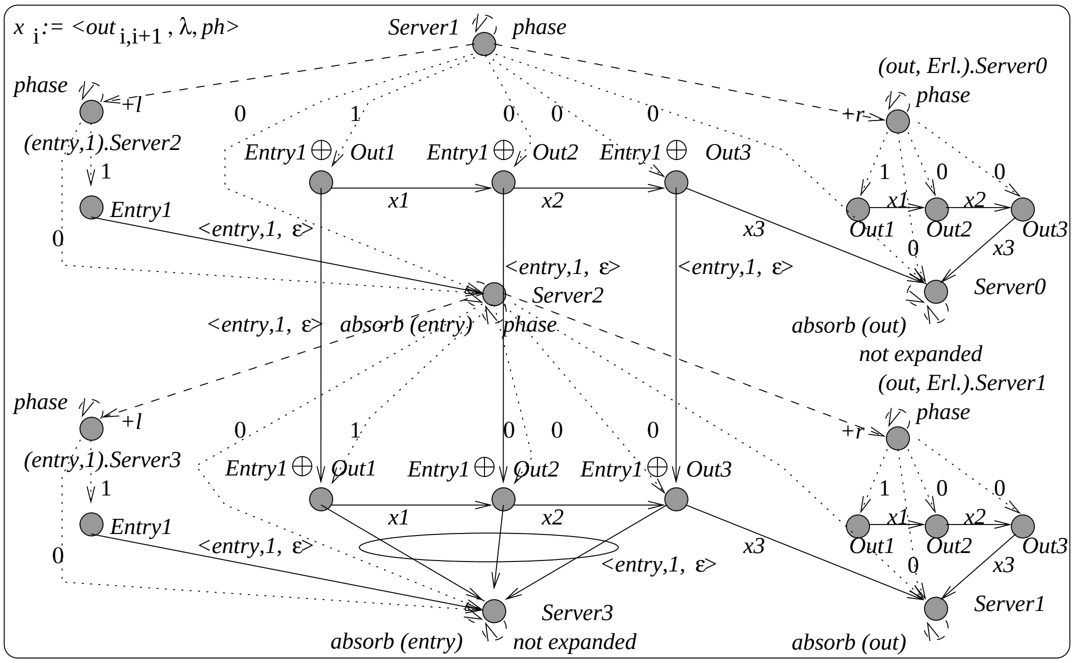

Abbildung 6.29: Expansion des $Server_1$–Prozesses

In der Abbildung 6.30 ist der expandierte Prozeßes $Server_0$ zu sehen, der der einzige veränderte Subprozeß ist gegenüber der Modellierung aus Kapitel 5. Dabei ist die Expansion in Abbildung 6.6 und Abbildung 6.28 angewandt worden. Zur besseren Lesbarkeit sind Knotenattribute, Hilfskanten und Wahrscheinlichkeitskanten mit dem Attribut „0“ weggelassen worden. Im rechten Graphen der Abbildung 6.30 sind zudem die Verrechnungen von Aktionskanten und Wahrscheinlichkeitskanten dargestellt. Bis auf die Attribute stellt der Graph die zugrundeliegende Markow–Kette des Subprozeßes $server_0$ dar. Sie entspricht der Markow–Kette des Teil–Petri–Netzes aus Abbildung 6.31.

In der Abbildung 6.32 ist die Komposition der Subprozesse $Poisson||_{\{in\}}Entryplace_0$ und $Server_0$ zu sehen. Sie synchronisieren sich über die Aktionen $entry$ und $loss$. In der Darstellung wurde weitgehend auf Hilfskanten verzichtet, auch wurden schon teilweise die Wahrscheinlichkeitskanten mit Aktionskanten in dem Prozeß $Server_0$ verrechnet und Pseudoknoten und Wahrscheinlichkeitskanten gelöscht. Dies entspricht zwar nicht der Transaktionsvorschrift, wurde aber aus Gründen der Übersichtlichkeit so dargestellt. Der mittlere Teilgraph repräsentiert das zeitliche Verhalten dieser Warteschlange. Zum ihrem expandierten LTS gelangt man, wenn die Wahrscheinlichkeitskante mit der Aktionskante verrechnet wird, und der mit der

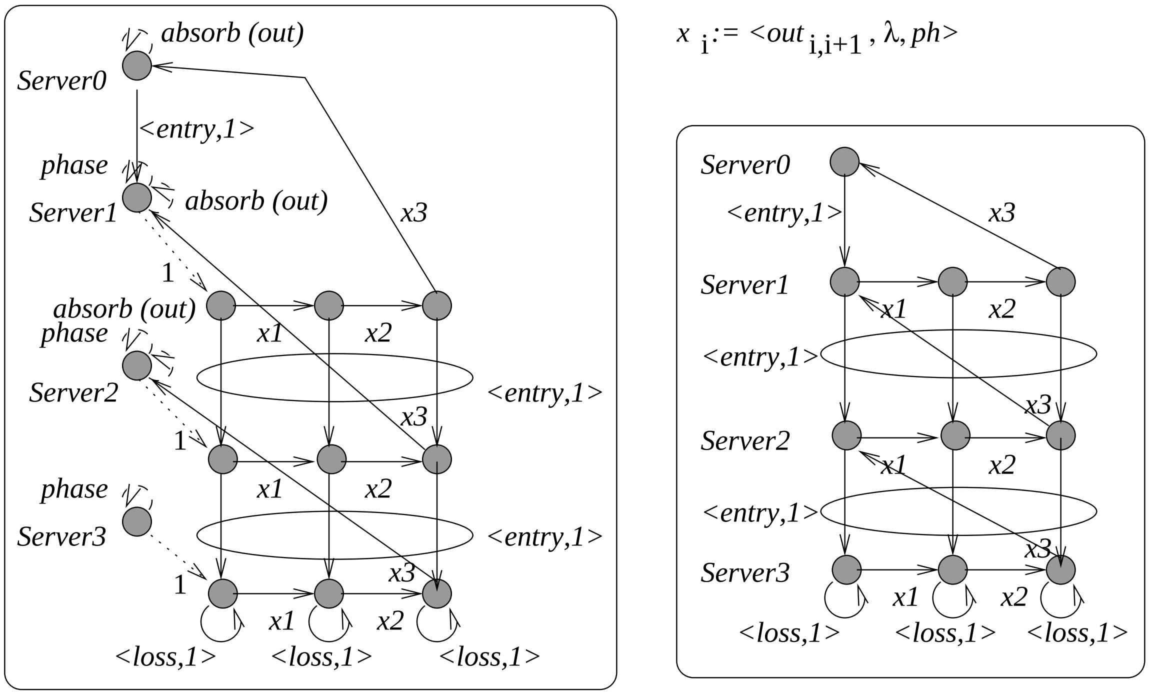

Abbildung 6.30: Expansion des $Server_0$–Prozesses

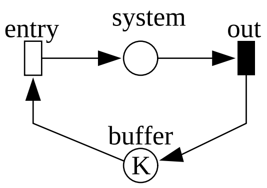

Abbildung 6.31: Teilnetz der Warteschlange

phase–Schlinge gekennzeichnete Knoten gelöscht wird.Das expandierte LTS ist hier wieder der mit dem *root*–Knoten über Aktionskanten verbundene Graph.

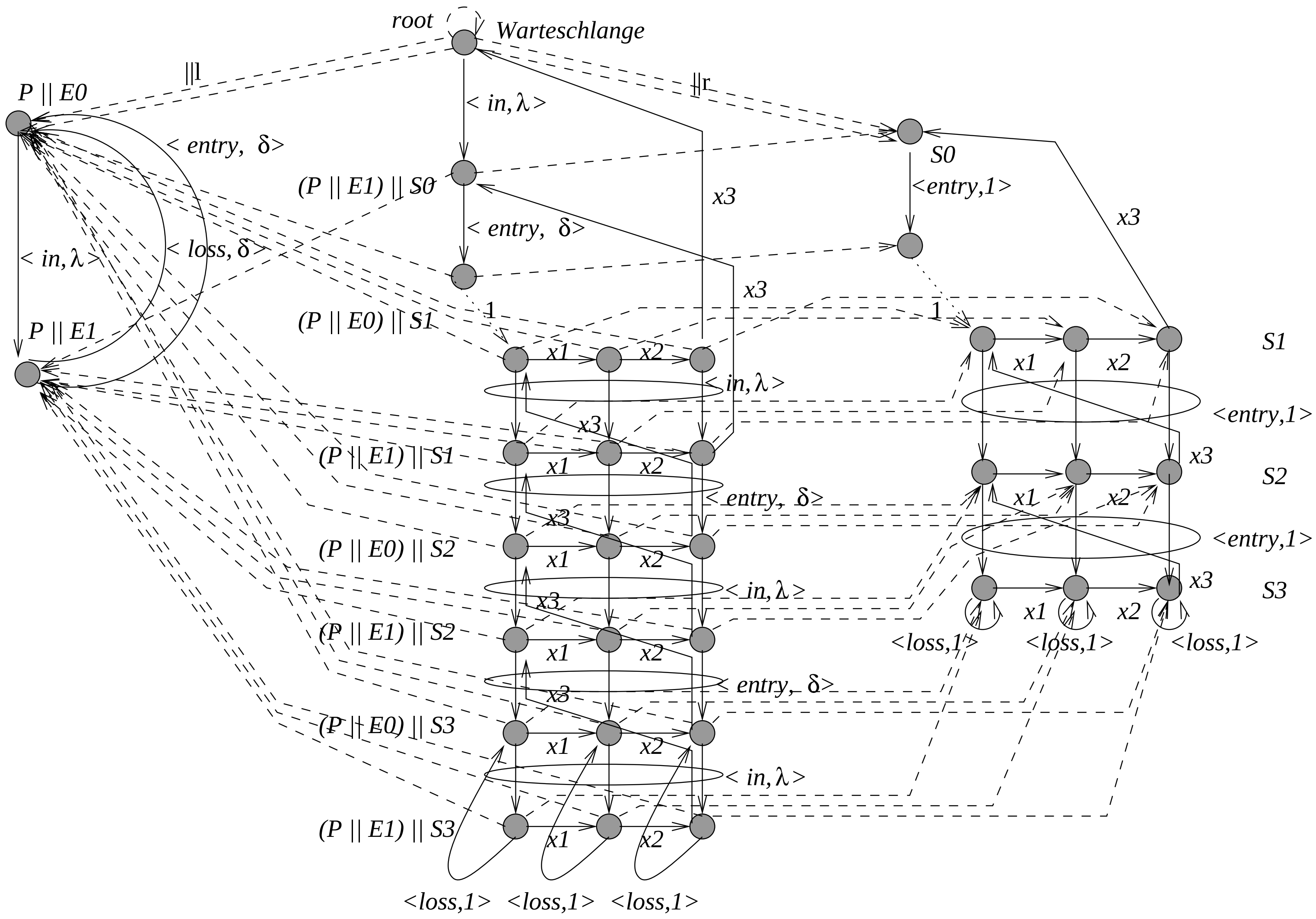

Abbildung 6.32: Warteschlange mit phasentypverteilter Aktion „*out*“

Kapitel 7

Zeitlose Aktionen

Eine zusätzliche sinnvolle Erweiterung im Hinblick auf die Dauer von Aktionen ist der Ausbau um *zeitlose* Aktionen, deren Ausführung ohne Verzögerung geschieht. Damit ergibt sich eine Unterscheidung in *flüchtige* und *andauernde* Zustände. In Analogie zur Vorgehensweise bei SPN wird das *reduzierte LTS* generiert, in das nur Zustandsübergänge zwischen andauernden Zuständen eingehen. Für den Ausbau des Formalismus werden wieder Graphregeln definiert. Anhand des Eingangsbeispiels der Warteschlange wird die Vorgehensweise zur Erzeugung des reduzierten LTS veranschaulicht.

Aus dem LTS eines Prozesses wird seine zugrundeliegende Markow–Kette ermittelt, die dann mit numerischen Standardmethoden analysiert wird. Dazu wird die Generatormatrix $\mathbf{Q}$ der Markow–Kette (siehe auch Anhang A) bestimmt und u.a. potenziert. Dabei kann ein numerisches Problem auftreten, wenn die Einträge der Matrix, d.h. die Raten der Aktionen, um mehrere Größenordnungen differieren: Das numerische Verfahren ist dann unstabil, was die Genauigkeit der Analyseergebnisse stark beeinträchtigen kann.
Dieses Problem wird vermieden, wenn Aktionen, deren mittlere Dauer im Vergleich mit anderen sehr kurz ist, als zeitlos angesehen werden. Im folgenden werden also stochastische Prozeßalgebren um zeitlose Aktionen erweitert. Aufbauend auf der graphischen SPA mit PH–Verteilungen werden im folgenden die Graphregeln vorgestellt, die diese Erweiterung realisieren.

7.1 Graphklasse

Um zeitlose Aktionen beschreiben zu können, muß zunächst die Menge der zulässigen Raten erweitert werden. Die zeitlose Aktion a soll als (a, ∞) notiert werden, dementsprechend vergrößert sich die Ratenmenge:

$$Rate := I\!R^+ \cup (I\!R^{k,k} \times [0,1]^k) \cup \{\infty\} \qquad k \geq 2$$

Orientiert an der Vorgehensweise im SPN–Formalismus soll die Generierung der zugrundeliegenden Markow–Kette eines Prozesses mit zeitlosen Aktionen in zwei Schritten erfolgen:

1. Generierung des expandierten LTS mit flüchtigen Zuständen, von denen Kanten mit dem Attribut $\langle a, \infty, w \rangle$ ausgehen.

2. Reduktion des LTS auf andauernde Zustände.

Die Generierung des expandierten LTS mit flüchtigen Zuständen erfolgt analog zu der zuvor beschriebenen Generierung ohne zeitlose Aktionen. Im Falle der Synchronisation muß nur zusätzlich erklärt werden, welche Dauer gemeinsam ausgeführte Aktionen haben. Vernünftig erscheint, daß die Dauer einer Aktion, die von einem Prozeß potentiell in Nullzeit ausgeführt wird, bei der er aber mit einem anderen Prozeß kooperieren muß, sich ergibt als die vom zweiten Prozeß benötigte. Das heißt, man erklärt die Multiplikation der Raten (siehe Parallelregel aus Abbildung 5.9) folgendermaßen für $\lambda \in I\!R^+ \cup (I\!R^{k,k} \times [0,1]^k)$:

$$\lambda \cdot \infty := \infty \cdot \lambda := \lambda$$

Demgegenüber sollen passive Aktionen $(a, 1)$, die sich mit einer zeitlosen Aktion synchronisieren müssen, die „Dauer der aktiven Aktion erben“. Also soll gelten:

$$1 \cdot \infty := \infty \cdot 1 := \infty$$

Die Zweideutigkeit der Rate „1“ einmal als Rate der Exponentialfunktion, andererseits aber auch als Kennzeichnung passiver Aktionen, macht hier eine restriktive Vorgehensweise nötig, da aus einer Prozeßdeklaration nicht hervorgeht, wie diese Rate zu verstehen ist. So ist eine konsistente Definition der Multiplikation mit „∞“ die folgende:

$$\lambda \cdot \infty := \infty \cdot \lambda := \begin{cases} \infty & \text{falls } \lambda = 1 \\ \lambda & \text{sonst} \end{cases}$$

Zur Kennzeichnung flüchtiger Zustände wird ein weiteres Attribut für Hilfskanten eingeführt: die $vanish$–Schlinge.
Außerdem soll es ermöglicht werden, eine Liste von Aktionennamen (Act^+) anstatt einer einzigen Aktion an die Zustandsübergangskanten zu schreiben . So ergibt sich die Kennzeichnung t der Übergänge als ein Tripel

$$t \in Act^+ \times Rate \times W'^* \text{ mit } W' = \{+_l, +_r, ||_l, ||_r, ||, \backslash_{yes}, \backslash_{no}, ph\}$$

7.2 Graphregeln

Mit diesen Erweiterungen kann der LTS–Graph wie vordem gezeigt durch die Regelmenge R_{lts} abgeleitet werden. Das LTS wird extrahiert, und danach setzt die zweite Phase der Reduktion ein.
Die Unterteilung in flüchtige und andauernde Zustände wird durch die Regel $mark$ (in Abbildung 7.1) realisiert. Hier wird ein Zustand, in dem eine zeitlose Aktion

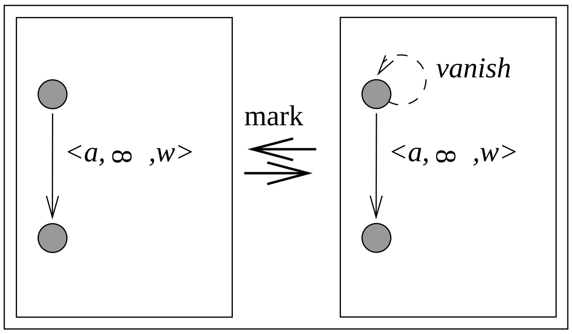

Abbildung 7.1: Markierung der flüchtigen Zustände

ausführbar ist, als flüchtig gekennzeichnet. Da eventuell mehrere zeitlose Aktionen ausführbar sind, die wiederholte Kennzeichnung aber keinen Informationsgewinn darstellen würde, hat diese Regel aus Optimierungsgründen eine negative Anwendungsbedingung. Schon gekennzeichnete Knoten sollen keine weiteren Schlingen erhalten.
Ausgehend von einem andauernden Zustand werden dann implizite Übergänge, also Übergangssequenzen mit Durchlaufen beliebig vieler flüchtiger Zustände, gefunden. Das iterierte Anwenden der Regel $concat$ aus Abbildung 7.2 erzeugt letzendlich eine Kante von einem andauernden in einen andauernden Zustand mit der Rate der

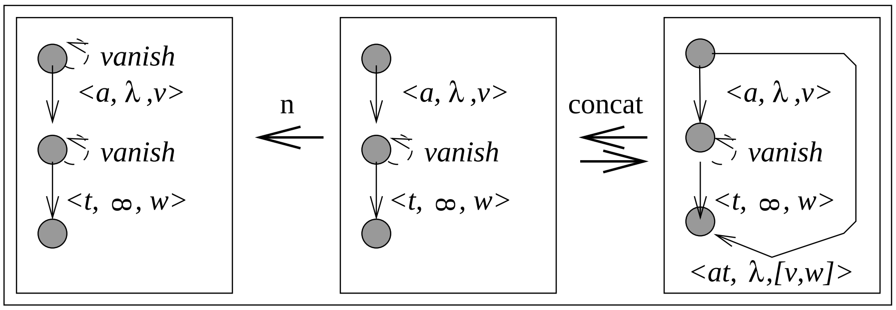

Abbildung 7.2: Aktionenfolge mit zeitlosen Aktionen

ersten Aktion unter der Voraussetzung, daß in dem LTS keine Zyklen von zeitlosen Aktionen existieren.

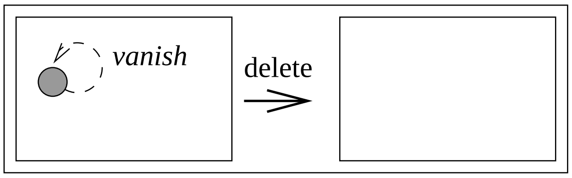

Abbildung 7.3: Löschen der flüchtigen Zustände

Zum Schluß werden die flüchtigen Zustände gelöscht, was durch die Regel *delete* realisiert wird. Alle Kanten, die diesen Prozeß zum Ziel oder als Quelle haben, werden implizit mitgelöscht.

Aus den obigen Erklärungen geht die Transaktionsvorschrift hervor:

$$\text{reduce} = || \{mark, concat\}^{\square}; \{delete\}^{\square} ||$$

Die Regeln *mark* und *concat* sollen zunächst maximal oft angewandt werden. Untereinander ist keine Reihenfolge vorgeschrieben, sie ergibt sich allerdings teilweise aus der sequentiellen Abhängigkeit der *concat*–Regel von der *mark*–Regel. Die von der *mark*–Regel erzeugte *vanish*–Schlinge wird für einen Ansatz der *concat*–Regel gebraucht. Erst, wenn keine der beiden Regeln mehr anwendbar ist, soll die *vanish*–Schlinge gelöscht werden. Die Löschung wird somit bis zum Schluß hinausgezögert, auch wenn sie schon früher ohne Verfälschung des LTS angewandt werden könnte. Es reicht eigentlich aus, wenn alle eingehenden Kanten in einen flüchtigen Zustand

mit allen ausgehenden so „verknüpft“ worden sind wie durch die Regel *concat* beschrieben. Sobald diese Regel an einem Ansatz nicht mehr anwendbar ist, könnte der flüchtige Zustand gelöscht werden. Dies ist leider mit den vorgestellten Konzepten nicht ausdrückbar, und wird daher restriktiver gehandhabt.
Das Ergebnis dieser Regelanwendungen ist das reduzierte LTS, in dem nur andauernde Zustände vorkommen, die durch Kanten verbunden sind. Die Kanten tragen eine Liste von Aktionen, die Rate der Exponentialverteilung der ersten Aktion dieser Liste und die „Geschichte der Ableitung“ als Attribut. Der so erzeugte LTS–Graph repräsentiert also bis auf Mehrfachkanten, Schlingen und Atrributierung die zugrundeliegende Markow–Kette.

7.3 Beispiel: Warteschlange

Mit der Einführung von zeitlosen Aktionen ist man nun in der Lage, das Eingangsbeispiel der M/D/1/K/K/FCFS–Warteschlange auch unter Berücksichtigung der Dauer der Aktionen in einen stochastischen Prozeß übersetzen. Die Aktionen *loss* und *entry* sind zeitlos, die Dauer der deterministischen Bedienzeit *out* wird wie im Kapitel 6 durch eine Erlang–Verteilung der Ordnung 3 angenähert. Das Petri–Netz–Modell aus Abbildung 7.4 ist jenes aus dem einleitenden Kapitel 1, das nun mit der erweiterten Fassung von SPA modelliert werden kann.

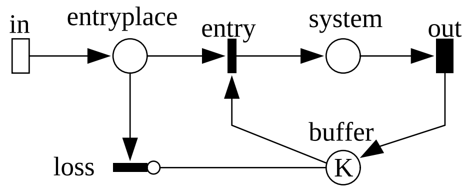

Abbildung 7.4: Warteschlange mit exponentiell verteilter, deterministischer und zeitloser Dauer von Zustandsübergängen

$Poisson$	$:=$	$(in, \lambda).Poisson$	
$Entryplace_0$	$:=$	$(in, 1).Entryplace_1$	
$Entryplace_1$	$:=$	$(entry, \infty).Entryplace_0 + (loss, \infty).Entryplace_0$	
$Server_0$	$:=$	$(entry, 1).Server_1$	
$Server_i$	$:=$	$(entry, 1).Server_{i+1} + (out, (\mathbf{E}_3, (1,0,0))).Server_{i-1}$	$(0 < i < K)$
$Server_K$	$:=$	$(out, (\mathbf{E}_3, (1,0,0))).Server_{K-1} + (loss, 1).Server_K$	

$$Warteschlange := (Poisson||_{\{in\}}Entryplace_0) \quad ||_{\{entry,loss\}}Server_0$$

Gemäß der obigen Erläuterung wird zunächst das LTS des Prozesses abgeleitet. Es ist links oben in der Abbildung 7.5 auf Seite 111. Auf dieses wird die Transaktionsvorschrift „reduce" angewandt. Eine mögliche Ableitung ist in Abbildung 7.5 zu sehen, wobei erst *mark*, dann *concat* und zum Schluß *delete* jeweils maximal oft angewandt wurden. Das Attribut „x_i" steht hierbei wieder für „$\langle out_{i,i+1}, 3\mu, w\rangle$", wobei 3μ die Rate eines Phasenüberganges der Erlang–Verteilung ist.

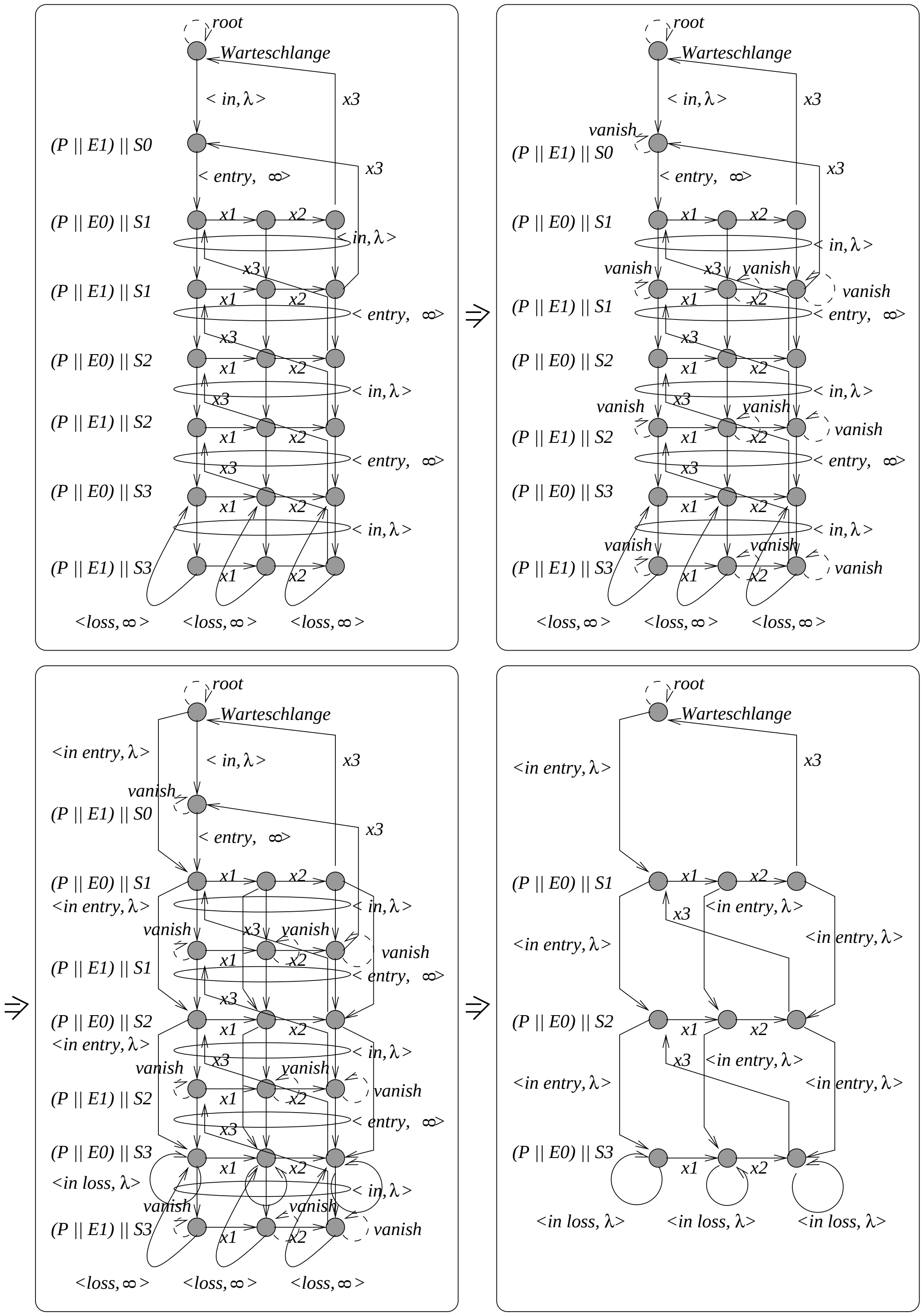

Abbildung 7.5: Reduzierung des LTS der Warteschlange

Kapitel 8

Modellierung eines Tokenrings

Kommunikationssysteme sind ein zentrales Gebiet der Informatik, in dem Leistungsbewertung relevant ist. Grob umrissen sind das Systeme, in denen Nachrichten zwischen den verschiedenen Teilnehmern (Stationen) gesendet und empfangen werden können. Die Wahrscheinlichkeit, daß eine Nachricht seine Empfänger erreicht wie auch, mit welcher durchschnittlichen Dauer man hierbei rechnen muß, sind unter anderen Gegenstand der Leistungsbewertung.

Ein solches Kommunikationssystem ist durch verschiedene Merkmale charakterisiert: Die Geschwindigkeit, mit der Nachrichten auf dem physikalischen Medium übertragen werden (Signalgeschwindigkeit), und diejenige, mit der Informationen auf dieses Medium gelegt werden (Bitrate); Die Möglichkeit, weitere Teilnehmer in das System aufzunehmen (offenes vs. geschlossenes Kommunikationssystem); Die Konvention des Nachrichtenaustausches zwischen Stationen (Protokoll); Die Zugriffsstrategien auf das Übertragungsmedium usw. (siehe auch [Tan92]). Ein weiteres Charakteristikum ist die *Topologie*, die die Struktur der Verbindung der Teilnehmer solcher Kommunikationssysteme beschreibt. U.a. unterscheidet man zwischen:

- Vollständiger Vernetzung, d.h jeder Teilnehmer ist physikalisch mit jedem anderen verbunden.
- Sternförmiger Verbindungsstrukturen, in denen jede Nachricht von einem zentralen Rechner weitervermittelt wird.
- Ringstrukturen, in denen jeder Teilnehmer einen Vorgänger und einen Nach-

folger hat.

- Unstrukturierter Vernetzung, bei der kein Bildungsgesetz erkennbar ist.

Ein Kommunikationssystem, das ringförmig verbunden ist, ist der *Tokenring*. Hier zirkuliert ein spezielles Bitmuster, ein *Token*, durch das Netz, das den Teilnehmern ihre Berechtigung signalisiert, Nachrichten zu verschicken. Hat also eine Station einen Sendewunsch, muß sie zunächst auf den Empfang des Tokens warten. Erhält sie es, hat sie für eine festgelegte maximale Dauer Sendeerlaubnis. Innerhalb dieser Zeitspanne legt sie die zuvor gepufferten Nachrichten auf den Ring, wobei die zulässige Dauer nicht voll ausgeschöpft werden muß. Das Token wird einbehalten, während die Nachrichten weiterzirkulieren. Jede Station, an der sie vorbeikommen, prüft, ob sie selbst Adressat einer der Nachrichten ist und kopiert den für sie bestimmten Teil. Sie modifiziert die Nachricht leicht, was einer Empfangsbestätigung entspricht und leitet sie weiter. Wieder beim Absender angekommen, entfernt dieser die Nachrichten vom Ring und generiert ein neues Token, das weiterzirkuliert. Er prüft die Bestätigungen für seine Nachrichten und wiederholt gegebenenfalls den Sendevorgang beim nächsten Erhalt des Tokens.

8.1 Stochastisches Petri–Netz

Das Petri–Netz aus Abbildung 8.1 ist [Ger95] entnommen und modelliert den Aspekt der Kommunikation über einen Tokenring, der die Versendung von Nachrichten betrifft. Den Mechanismus der Adressatenprüfung wie auch der Empfangsbestätigung erfaßt dieses Modell nicht.
Das Modell ist folgendermaßen zu verstehen: Es gibt N Stationen, die ringförmig miteinander verbunden sind. Eine Station i befindet sich initial in einem Zustand, in dem sie „denkt“, also für das Kommunikationssystem unsichtbare Aktivitäten ausführt. Dieser Zustand wird durch die Stelle $Think_i$ modelliert. Aus diesem Zustand „Denken“ resultiert nach einer exponentiell verteilten Dauer die Erzeugung einer Nachricht (Transition $generate_i$), die in einem Puffer abgelegt wird (Stelle $buffer_i$). Dieser Puffer kann maximal k Nachrichten aufnehmen, was dadurch modelliert wird, daß initial k Token auf der Stelle $Think_i$ liegen. Bekommt die Station i nun die Sendeerlaubnis, sprich liegt ein Token auf der Stelle $token_i$, so gibt es zwei

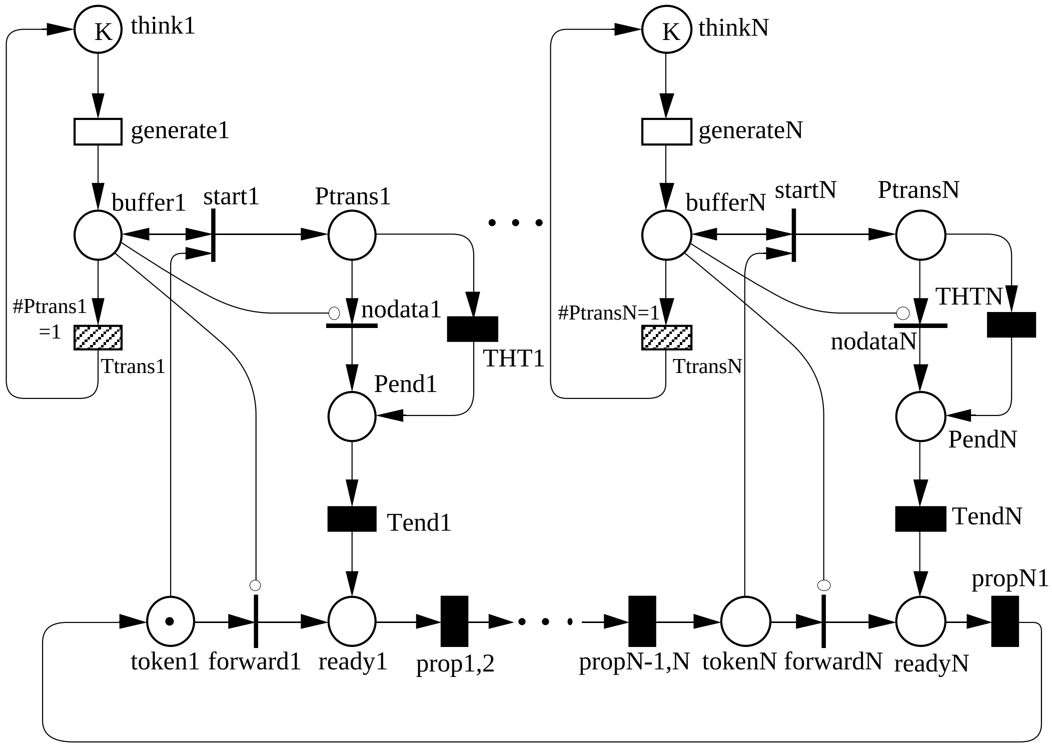

Abbildung 8.1: Petri–Netz–Modell eines Tokenrings

verschiedene Möglichkeiten: Entweder hat die Station noch keine Nachrichten generiert, d.h. die Stelle $buffer_i$ ist leer. In diesem Fall schaltet die Transition $forward_i$ in Nullzeit, was durch die hemmende Kante von der Stelle $buffer_i$ ausgedrückt wird. Anderenfalls beginnt unverzüglich der Sendevorgang (Transition $start_i$). Ein Token in der Stelle $Ptrans_i$ bedeutet also, daß die Station gerade sendet. In Abhängigkeit von der Länge der Nachricht und den Charakteristika des Kommunikationsnetzes wird pro Nachricht eine gewisse Sendezeit benötigt. Dies spiegelt sich im Netz so wieder, daß von der Stelle $buffer_i$ nach einer durch eine allgemeine Verteilung gegebenen Dauer eine Nachricht entfernt wird. Die Schaltdauer modelliert so die Länge der Nachricht. Gibt es keine zu verschickenden Nachrichten mehr, sprich ist die Stelle $buffer_i$ leer, kann die Transition $nodata_i$ in Nullzeit schalten und die Übertragung ist beendet (Stelle $Pend_i$). Erreicht die Übertragungsdauer eine gewisse Grenze, wird ebenfalls der Sendevorgang abgebrochen (Stelle $Pend_i$). In Modell ist dieser „timeout" dadurch nachgebildet, daß die deterministische Transition THT_i (Token Holding Time) nach Ablauf dieser Zeit das Übertragungsende (Stelle $Pend_i$) erzwingt. Unabhängig von der Art des Übertragungsschluß muß nach dem Senden ein neues Token generiert und auf das Übertragungsmedium gelegt werden, was durch die deterministische Transition $Tend_i$ zum Ausdruck kommt. Die Dauer dieser Transition entspricht also der Bitrate. Wieder unabhängig davon, ob gesendet wurde oder nicht, wird zuletzt das Token weitergegeben (Stelle $ready_i$). Die Dauer der Tokenübertragung ist festgelegt durch die Signalgeschwindigkeit des Mediums sowie den konstanten Abstand zur nächsten Station, weshalb die Transition $prop_{i,i+1}$ eine deterministische Schaltzeit besitzt.

8.2 Stochastischer Prozeß

Die Umsetzung des Petri–Netz–Modells in einen stochastischen Prozeß erfolgt genauso wie in dem Abschnitt 3.2.2 beschrieben: Jede Transition entspricht einer Aktion. Jede Stelle wird in einen Prozeß übersetzt, der passiv Aktionen ausführt, falls die Stelle im Nachbereich der entsprechenden Transition liegt. Aktiv kann er eine Aktion ausführen, in deren Vorbereich die Stelle liegt. Gibt es mehr als eine Stelle im Vorbereich einer Transition, wählen wir eine aus, bei deren Modellierung als Prozeß die entsprechende Aktivität aktiv ist. Entspechend ist die gleiche Aktion der anderen Vorbereichsstellen dann passiv.
Die Schaltdauer der Transitionen wird für die entsprechenden Aktionen übernommen. Dabei nutzen wir hier, daß die Dauer von Aktionen nicht auf Exponentialverteilungen beschränkt ist. Deterministische Transitionen werden also in Aktionen mit k–phasiger Erlang–Verteilung (det) übersetzt, allgemeine Transitionen in Aktionen mit einer PH–Verteilung, die diese allgemeine Verteilung approximiert (g).
Das Gesamtsystem erhält man, indem die einzelnen Prozesse parallel kombiniert werden. Dabei kooperieren sie bei der Ausführung einer Aktion, deren korrespondierende Transition die entprechenden Stellen „verbindet“. Daher listen wir zunächst solche Verbindungen auf, d.h. wir stellen fest, welche Transitionen mit welchen Stellen zusammenhängen. Das sind natürlich einerseits die Stellen des Vor– und Nachbereichs. Aber auch Stellen, von denen eine hemmende Kante zur Transition führt, beeinflussen die Aktivierung dieser Transition. Und auch diejenigen Stellen, über deren Füllstand eine Schaltbedingung formuliert ist (siehe **Ptrans**). Das Zusammentragen des „Wirkungsradius“ einer Transition erleichtert später die Erkenntnis, welche Prozesse sich über welchen Aktionen synchronisieren müssen.

Anstatt alle Prozesse einzeln zu dem Gesamtsystem zu kombinieren, werden sie zur besseren Lesbarkeit zunächst „gruppiert", d.h. sie werden über einer Menge von zu synchronisierenden Aktionen parallel verbunden und der kombinierte Prozeß mit einer Variablen benannt. Bei einer Gruppierung müssen sich natürlich alle Prozesse innerhalb dieser Gruppe wie oben angegeben synchronisieren. Das Synchronisationsverhalten der Gruppe nach außen mit einem Prozeß P ergibt sich aus der Vereinigung der Aktionenmengen, über die sich Gruppenmitglieder mit dem Prozeß P synchronisieren müssen.

Die Menge der Aktionen ergibt sich also aus der Menge der Transitionen des Netzes, wobei der Index die Station angibt, innerhalb derer diese Aktion stattfindet.

$Act = \{generate_i, ttrans_i, start_i, nodata_i, tht_i, tend_i, forward_i, prop_{i,i+1}\}$, $0 < i \leq N$

Bei der Liste der zu einer Transition gehörenden Stellen werden zur Vereinfachung die Indizes weggelassen.

generate	Zusammenhang mit	**Think**	
generate	– " –	**Buffer**	
ttrans	Zusammenhang mit	**Buffer**	
ttrans	– " –	**Buffer**	
ttrans	– " –	**Ptrans**	#(Ptrans) = 1
start	Zusammenhang mit	**Token**	
start	– " –	**Buffer**	
start	– " –	**Ptrans**	
forward	Zusammenhang mit	**Token**	
forward	– " –	**Buffer**	
forward	– " –	**Ready**	
nodata	Zusammenhang mit	**Ptrans**	
nodata	– " –	**Buffer**	
nodata	– " –	**Pend**	
tend	Zusammenhang mit	**Pend**	
tend	– " –	**Ready**	
tht	Zusammenhang mit	**Ptrans**	
tht	– " –	**Pend**	

Konvention 8.2.1
Zur Vereinfachung reduzieren wir die doppelte Indizierung „i, j“, die einmal die j-te Station und den Füllstand i auf der entsprechenden Stelle bedeutet, zunächst auf den Index i. Damit erhält man für einen Tokenring mit N Stationen die im folgenden angegebenen Prozesse N-fach, jeweils mit dem zweiten Index für die Nummer der Station.
Prozeßnamen werden groß-, Aktivitätennamen kleingeschrieben.

Think

$$\begin{array}{lcll} Think_0 & := & (ttrans, 1).Think_1 & \\ Think_i & := & (generate, \lambda).Think_{i-1} + (ttrans, 1).Think_{i+1} & (0 < i < K) \\ Think_K & := & (generate, \lambda).Think_{K-1} & \end{array}$$

Buffer

$$\begin{array}{lcll} Buffer_0 & := & (forward, 1).Buffer_0 + (nodata, 1).Buffer_0 + & \\ & & (generate, 1).Buffer_1 & \\ Buffer_i & := & (start, det).Buffer_i + (ttrans, g).Buffer_{i-1} + & \\ & & (generate, 1).Buffer_{i+1} + (tht, 1).Buffer_i & (0 < i < K) \\ Buffer_K & := & (start, det).Buffer_K + (ttrans, g).Buffer_{K-1} + & \\ & & (tht, 1).Buffer_K & \end{array}$$

Diese beiden Prozesse werden zu einem zusammengefaßt, indem man sie parallelschaltet und dabei über *generate* und *ttrans* synchronisiert.

$$Message_K := Buffer_0 ||_{\{generate, ttrans\}} Think_K$$

Ptrans

$$\begin{array}{lcll} Ptrans_0 & := & (start, 1).Ptrans_1 & \\ Ptrans_1 & := & (nodata, \infty).Ptrans_0 + (tht, det).Ptrans_0 + (ttrans, 1).Ptrans_1 + & \\ & & (start, 1).Ptrans_2 & \\ Ptrans_i & := & (start, 1).Ptrans_{i+1} + (nodata, \infty).Ptrans_{i-1} + & \\ & & (tht, det).Ptrans_{i-1} & (1 < i) \end{array}$$

Pend

$$\begin{array}{lcl} Pend_0 & := & (nodata, 1).Pend_1 + (tht, 1).Pend_1 \\ Pend_i & := & (nodata, 1).Pend_{i+1} + (tht, 1).Pend_{i+1} + (tend, det).Pend_{i-1} \end{array}$$

Ready

$$\begin{array}{lcll} Ready_0 & := & (tend, 1).Ready_1 + (forward, 1).Ready_1 & \\ Ready_i & := & (tend, 1).Ready_{i+1} + (forward, 1).Ready_{i+1} & \\ & & +(prop_{1-2}, det).Ready_{i-1} & (0 < i) \end{array}$$

Token

$$\begin{array}{lcl} Token_0 & := & (prop_{i-1}, i).Token_1 \\ Token_1 & := & (forward, \infty).Token_0 + (start, 1).Token_0 \end{array}$$

Die Stellen, die mit der Weitergabe des Tokens direkt befaßt sind, sind $Token_i$ und $Ready_i$. Diese werden wieder zu einem Prozeß zusammengefaßt. Initial sind beide

Stellen leer bei denjenigen Stationen, die nicht als erste das Token zur Verfügung haben, so daß der erste Index hier „0“ ist.:

$$Connect_i := Token_{0,i} ||_{\{forward_i\}} Ready_{0,i}$$

Der Prozeß $Connect_i$ stellt die Anbindung der Station i an den Ring bzgl. der Tokenweitergabe her. Für den initialen Zustand soll ein Token auf der Stelle $Token_1$ liegen; deshalb wird zudem der Prozeß $Connect_{init}$ benötigt.

$$Connect_{init} := Token_{1,1} ||_{\{forward_1\}} Ready_{0,1}$$

Werden jetzt alle zuvor definierten Prozesse mit einem zweiten Index für die Station, zu der sie gehören, ausgestattet, kann man die i–te Station folgendermaßen definieren:

$$\begin{aligned} Station_i := & \; Message_{K,i} \; ||_{\{start_i, nodata_i, ttrans_i\}} Ptrans_{0,i} \; ||_{\{nodata_i, tht_i\}} Pend_{0,i} \\ & ||_{\{tend_i, forward_i\}} Connect_i \end{aligned}$$

Den Tokenring erhält man, indem man die Stationen parallelschaltet und eine Synchronisation über die Aktion *prop* fordert, die die Tokenweitergabe modelliert.

$$Tokenring := Station_1 ||_{\{prop_{1,2}\}} Station_2 \cdots ||_{\{prop_{N-1,N}\}} Station_N ||_{\{prop_{N,1}\}} Tokenring$$

In diesem Ring ist kein zirkulierendes Token vorhanden. Man fügt es zu, indem bei Station_1 der $Connect$–Prozeß angepaßt wird:

$$Station_{1,init} := \; \ldots \; ||_{\{tend_1, forward_1\}} Connect_{init}$$

So wird dann ein Tokenring mit Token durch folgende Definition modelliert.

$$Init := Station_{1,init} ||_{\{prop_{1,2}\}} Station_2 \cdots ||_{\{prop_{N-1,N}\}} Station_N ||_{\{prop_{N,1}\}} Tokenring$$

Kapitel 9

Fazit

9.1 Ergebnisse

Der Formalismus der stochastischen Prozeßalgebren wurde in dieser Arbeit im Hinblick auf die stochastische Verteilung für die Dauer von Aktionen erweitert. Neben den herkömmlichen Exponentialverteilungen kann nun die Dauer einer Aktion auch durch eine Phasentyp–Verteilung gegeben oder zeitlos sein. Damit ist man in der Lage, quantitative Eigenschaften eines Systems für alle möglichen zeitlichen Verhalten zu ermitteln, da jedes beliebige zeitliche Verhalten entweder direkt modelliert oder durch eine Phasentyp–Verteilung approximiert werden kann. Zum Beispiel kann eine deterministische Dauer durch eine Erlang–Verteilung angenähert werden.
Die Erweiterung von Exponentialverteilungen auf Phasentyp–Verteilungen wurde im Rahmen dieser Arbeit durch die Verwendung von Graphgrammatiken realisiert. Die herkömmlichen Ableitungsregeln für stochastische Prozeßalgebren wurden durch eine Menge von Graphregeln ersetzt, wobei die daraus resultierende operationale Semantik eines stochastischen Prozesses in beiden Fällen übereinstimmt: Bei der Modellierung der Dauer von Systemaktivitäten durch Exponentialverteilungen wird durch die Graphregelmenge ein Graph erzeugt, der anschließend auf genau das *labelled transition system* (LTS) reduziert wird, das auch durch die Ableitungsregeln hergeleitet wird. Während in den meisten stochastischen Prozeßalgebren die Dauer von Aktivitäten auf Exponentialverteilungen eingeschränkt ist, ermöglicht die Verwendung von Graphgrammatiken nun auch die Modellierung von Phasentyp–Verteilungen. Darüberhinaus kann jede andere Verteilung approximiert werden durch Modellierung der Dauer der Aktion durch Phasentyp–Verteilungen.
Die Verwendung von Graphgrammatiken hat den weiteren Vorteil, daß durch Graph-

regeln ein abstrakter Algorithmus beschrieben wird und somit die Voraussetzungen für die Übertragung in ein Programm zur Generierung des LTS eines stochastischen Prozesses geschaffen sind. Dieses Programm muß nicht notwendigerweise sequentiell sein, da wegen der parallelen Unabhängigkeit der Regeln keine Reihenfolge vorgegeben wird.
Um die Angemessenheit der Vorgehensweise exemplarisch zu überprüfen, wurde das erzeugte LTS eines stochastischen Prozesses für eine Warteschlange mit dem Erreichbarkeitsgraphen des analogen Petri–Netz–Modells verglichen. Dazu wurde ein allgemeines Verfahren entwickelt, das ein Petri–Netz in einen stochastischen Prozeß übersetzt. Für das Beispiel zeigte sich, daß der Erreichbarkeitsgraph des Petri–Netzes und das LTS des stochastischen Prozeses übereinstimmen.

9.2 Ausblick

Die Korrektheit der Graphregeln bzgl. des erzeugten LTS ist im Falle von exponentiell verteilten Aktionen anhand eines Beispiels demonstriert worden. Sie müßte zudem formal nachgewiesen werden. Dazu muß gezeigt werden, daß immer, wenn durch die Ableitungsregeln ein Zustandsübergang a von einem Prozeß P in einen Prozeß P' hergeleitet werden kann, auch eine Aktionskante mit dem Attribut a vom Knoten P in den Knoten P' durch Anwendung der Graphregeln abgeleitet werden kann. Dieser Beweis könnte per Induktion über die Anzahl der komponierenden Operatoren geführt werden.
Die Korrektheit der Graphregeln bzgl. phasentypverteilter und zeitloser Aktionen könnte man unter Verwendung der Übersetzungsvorschrift[1] von M. Ribaudo in [Rib95] für einen stochastischen Prozeß in ein stochastisches Petri–Netz folgendermaßen nachweisen: Die Generierung der Markow–Kette eines stochastischen Prozesses mittels Graphgrammatiken und die Generierung der Markow-Kette des korrespondierenden Petri–Netz–Modells führen zu dem gleichen Ergebnis.

Ein wesentliches Charakteristikum der stochastischen Prozesse mit Exponentialverteilungen ist ihre Kompositionalität (siehe z.B. [HHM95]). Es ist wünschenswert, wenn diese Kompositionaltität nun auch für Prozesse mit phasentypverteilten und

[1]Diese Übersetzungsvorschrift muß entsprechend den Erweiterungen von SPA um phasentypverteilte und zeitlose Aktionen um die Übersetzung solcher Aktionen erweitert werden.

zeitlosen Aktionen nachgewiesen werden kann. Dabei muß der Begriff der Markow'schen Bisimulation dahingehend erweitert werden, daß die Wahrscheinlichkeiten für einen direkten (zeitlosen) Zustandsübergang z.B. in eine Phase einer Phasentyp–Verteilung in den Äquivalenzbegriff mit eingehen. Danach muß überprüft werden, ob die so definierte Äquivalenzrelation auch eine Kongruenz bzgl. der komponierenden Operatoren der abstrakten Sprache ist.
Das Gleichungskalkül zur Identifizierung funktional und temporal äquivalenter Prozesse auf der syntaktischen Ebene ist nun allerdings nicht mehr vollständig, da phasentypverteilte Aktionen nicht berücksichtigt sind. Es ergibt sich die Schwierigkeit, alle äquivalenten Prozesse über Gleichungen zu identifizieren. Dies ist wahrscheinlich kaum möglich ohne die Phasentyp–Verteilung genau zu untersuchen. Das Beispiel eines redundanten Zweikomponentensystems aus Kapitel 2 kann man mit zwei syntaktisch verschiedenen Prozessen insofern äquivalent modellieren, als die zugrundeliegende Markow–Kette identisch ist:

$$(fail, \lambda).Stop \,||_{\emptyset}\, (fail, \lambda).Stop \quad \sim_M \quad (fail, (\begin{pmatrix} -2\lambda & \lambda & \lambda \\ 0 & -\lambda & \lambda \\ 0 & 0 & -\lambda \end{pmatrix}, (1,0,0)).Stop$$

Diese Äquivalenz ist möglicherweise auf der syntaktischen Ebene nicht mehr entscheidbar, so daß die Vollständigkeit des Gleichungskalküls eingebüßt wäre.

Die Umsetzung der Erweiterung in ein Programm zur Erzeugung des LTS des modellierten Systems soll vorzugsweise durch ein Graphableitungstool erfolgen, wobei die Parallelität der Regelableitungen dadurch voll ausgenutzt werden kann, daß die Generierung auf unterschiedliche Rechner verteilt wird. Ansonsten scheint eine Umsetzung in eine parallele Programmiersprache empfehlenswert.

Anhang A

Einführung in stochastische Petri–Netze

Stochastische Petri–Netze sind eine Erweiterung von zeitlosen Petri-Netzen [Pet62], die zur Modellierung nebenläufiger und verteilter Systeme verwendet werden. Als eine erste stochastische Erweiterung werden Transitionen mit einer Schaltzeit ausgestattet, der eine Exponentialverteilung unterliegt. Daneben gibt es Transitionen, die in Null–Zeit schalten, d.h. zeitlos sind. Außerdem werden Transitionen mit einer Priorität ausgestattet, wobei zeitlose über den zeitbehafteten Transitionen stehen. Diese Erweiterung führt zu *Generalized Stochstic Petri–Nets* (GSPN). Die folgende Einführung ist [Ger95] entnommen und wird hier im wesentlichen zitiert.

Definition A.0.1 (Generalized Stochastic Petri–Net)
Ein GSPN ist ein 8-Tupel, $GSPN = (P, T, I, O, H, \Pi, M_0, W)$ mit

- $P = \{p_1, \ldots, p_{|P|}\}$ ist eine endliche Menge von Stellen. Eine Stelle wird graphisch als Kreis gezeichnet. ($\bigcirc$)

- $T = \{t_1, \ldots, t_{|T|}\}$ ist eine endliche Menge von Transitionen. Dabei gilt $P \cap T = \emptyset$. Diese Menge ist unterteilt in zeitlose (immediate: $\rule{2pt}{10pt}$) und zeitbehaftete ($\Box$) Transitionen, $T = T_{imm} \cup T_{exp}$.

- $I : P \times T \to I\!N_0$ sind Eingangskanten, ($\bigcirc\!\!\rightarrow\!\Box$). Die Stelle, von der eine Eingangskante zur Transition führt, heißt Eingangsstelle. Die Gesamtheit aller Eingangsstellen einer Transition heißt Vorbereich der Transition.

- $O : T \times P \to \mathbb{N}_0$ sind Ausgangskanten, ( ) Ausgangsstellen und Nachbereich einer Transition sind analog zum Eingang definiert.

- $H : P \times T \to \mathbb{N}_0$ sind hemmende Kanten, ( ).

- $\Pi : T \to \mathbb{N}_0$ gibt die Prioritäten der Transitionen an.

- $M_0 \in \mathbb{N}_0^{|P|}$ ist die initiale Markierung eines Petri–Netzes, also die Anfangsbelegung der Stellen mit Token.

- $W : T \to \mathbb{R}^+$ ist das Gewicht einer Transition. Ist die Transition zeitlos, so gibt das Gewicht die Schaltwahrscheinlichkeit an, anderenfalls die mittlere Schaltzeit. In diesem Fall liegt der Schaltzeit eine Exponentialverteilung zugrunde, deren Rate sich aus dem reziproken der mittleren Schaltzeit ergibt.

$\square$

In dem an der Technischen Universität Berlin entwickelten Tool für stochastische Petri–Netze (siehe auch [GHKZ95]) „TimeNET“ sind zusätzlich Schaltbedingungen an Transitionen (guards) und auch markierungsabhängiges Schaltverhalten implementiert.

Definition A.0.2 (Schaltverhalten)
Eine Transition t_0 kann in einer Markierung M schalten, wenn

1. gemäß der Anzahl der Eingangskanten $I(p, t_0) =: i$ auf jeder Stelle p des Vorbereichs mindestens die Anzahl i von Token liegt.

2. weniger als $H(p, t_0)$ Token auf jeder Stelle p liegen, von der eine hemmende Kante zur Transition t_0 führt.

3. Bei einem Konflikt, also gleichzeitiger Aktivierung zweier konkurrierender Transitionen, schaltet die höher priorisierte. Bei gleicher Priorität ergibt sich die Schaltwahrscheinlichkeit aus dem Gewicht: Die Schaltwahrscheinlichkeit einer Transition ergibt sich aus dem Quotienten ihrer Rate und der Summe der Raten der insgesamt aktivierten Transitionen.

4. Beim Schalten einer aktivierten Transition t_0 werden die Token aller Eingangsstellen p_I entsprechend der Funktion $I(p_I, t_0)$ entfernt und auf die Ausgangsstellen p_O werden Token gemäß der Funktion $O(t_0, p_O)$ hinzugefügt. Dieser Vorgang ist ununterbrechbar.

$\square$

Die Analyse von GSPN erfolgt in drei Schritten:

1. Abbildung des GSPN auf eine zeitkontinuierliche Markow–Kette
2. Analyse der Markow–Kette
3. Berechnung der Bewertungsgrößen des GSPN

Bei der Abbildung des GSPN auf eine zeitkontinuierliche Markow–Kette wird zunächst zwischen zwei Klassen von Markierungen unterschieden: den flüchtigen und den andauernden. Flüchtige Markierungen sind solche, in denen nur zeitlose Transitionen aktiviert sind; in andauernden sind nur zeitbehaftete Transitionen aktiviert. Die Markow–Kette entspricht dem *reduzierten Erreichbarkeitsgraphen* (*RRG*), mit $RRG = (S, A)$. Die Knoten S sind die andauernden Markierungen, Kanten $A \subseteq S \times S \times T^+$ führen von andauernden in andauernde Markierungen mit einer Schaltsequenz von Transitionen. Diese Schaltsequenz besteht aus einer zeitbehafteten Transition, gefolgt von beliebig vielen zeitlosen Transitionen. Die Markow–Kette hat den Zustandsraum S mit Zustandsübergängen, deren Rate sich aus den Kanten des RRG errechnet: Es ist die Rate der zeitbehafteten Transition multipliziert mit den Wahrscheinlichkeiten der zeitlosen Transitionen. Kanten zwischen gleichen Zuständen werden zusammengefaßt, indem die Raten addiert werden.

Zur Analyse wird die Generatormatrix $\mathbf{Q}$ mit $\dim \mathbf{Q} = |S|$ ermittelt. Die Berechnung der transienten Zustandswahrscheinlichkeiten erfolgt durch Lösung eines Differentialgleichungssystems 1.Ordnung:

$$p(t) = (p_0(t), p_1(t), \ldots, p_{|S|-1}(t)) = p(0) \cdot e^{\mathbf{Q}t}$$

Die stationäre Zustandswahrscheinlichkeit (zeitunabhängig) ergibt sich als Vektor $\pi = (\pi_0, \pi_1, \ldots, \pi_{|S|-1})$ mit der Eigenschaft:

$$\pi \cdot \mathbf{Q} = 0, \quad \text{mit} \sum_{i=0}^{|S|-1} \pi_i = 1$$

In diesem Fall ist ein lineares Gleichungssystem zu lösen.

Aus den Zustandswahrscheinlichkeiten können Bewertungsgrößen bestimmt werden. Die Wahrscheinlichkeit einer Bedingung A ist die Summe der Wahrscheinlichkeiten der Zustände, in denen A erfüllt ist.

Deterministic and Stochastic Petri–Nets (DSPN) [GHKZ95] sind eine Erweiterung von GSPN, wobei den zeitbehafteten Transitionen eine exponentiell verteilte oder eine deterministische Schaltzeit zugeordnet werden kann. Darüberhinaus können Transitionen in *Extended DSPN* (EDSPN) [GHKZ95] mit einer beliebig verteilten Zeit schalten. Daneben gibt es einen Formalismus, der Phasentypverteilungen als Dauer von Transitionen zuläßt (auf der Grundlage von [Cum85]). Dieser wird in der vorliegenden Arbeit als Referenz genutzt, wie die Expansion des LTS um phasentypverteilte Aktionen realisiert wird.

Literaturverzeichnis

[BDG94] M. Bernardo, L. Donatiello, and R. Gorrieri. MPA: a stochastic Process Algebra. Technical report, University of Bologna, Laboratory of Computer Science, May 1994.

[BDH92] E. Best, R. Devillers, and J. Hall. The Box Calculus: a new causal algebra with multi-label communication. volume 609 of *Lecture Notes in Computer Science*, pages 21–69. 1992.

[BFF+95] E. Best, H. Fleischhack, W. Fraczak, R.P. Hopkins, H. Klaudel, and E. Pelz. An M-Net Semantics of $b(pn)^2$. In J. Desel, editor, *Structures in Concurrency Theory*, pages 85–100. Springer-Verlag, 1995.

[Buc94] P. Buchholz. On a Markovian Process Algebra. Technical report, University of Dortmund, 1994.

[CGHB+95] M. Conrad, M. Gajewsky, R. Holl-Biniasz, M. Rudolf, J. Demuth, S. Weber, R. Heckel, J. Müller, G. Taentzer, and A. Wagner. Graphische Spezifikation ausgewählter Teile von AGG - einem algebraischen Graphgrammatiksystem. Technical Report 95-7, TU-Berlin, 1995.

[Cum85] A. Cumani. ESP - A Package for the Evaluation of Stochastic Petri Nets with Phase-Type Distributed Transition Times. In *Proc. 1st Int. Workshop Timed Petri Nets*, pages 144–151, Torino, Italy, 1985.

[Ehr79] H. Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *1st Graph Grammar Workshop, Lecture Notes in Computer Science 73*, pages 1–69. Springer Verlag, 1979.

[EPS73] H. Ehrig, M. Pfender, and H.J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973.

[Fel66] W. Feller. *An Introduction to Probability Theory and its Applications.* John Wiley, New York, NY, 1966.

[Ger95] R. German. Modellierung moderner Kommunikationssysteme. TU-Berlin, Prozeßdatenverarbeitung und Robotik, 1995. Skript zur gleichnamigen Lehrveranstaltung WS95/96.

[GHKZ95] R. German, G. Hommel, C. Kelling, and A. Zimmermann. TimeNET: A Toolkit for Evaluating Non-Markovian Stochastic Petri Nets. *Performance Evaluation*, 24:69–87, 1995.

[GHR93] N. Götz, U. Herzog, and M. Rettelbach. TIPP-Introduction and Application to Protocol Performance Analysis. *Formale Beschreibungstechniken für verteilte Systeme, FOKUS series, Saur publishers, Munich*, 1993.

[GHR94] 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 *Tutorial Proc of the PERFORMANCE '93, Rome, volume 729 of LNCS.* Springer Verlag, 1994.

[HHM95] H. Herrmanns, U. Herzog, and V. Mertsiotakis. Stochastic Process Algebras as a Tool for Performance and Dependability Modelling. In *Proc. IEEE Int. Computer Performance and Dependability Symp. Erlangen*, Germany, 1995.

[Hil93] J. Hillston. PEPA: Performance Enhanced Process Algebra. Technical report, The University of Edinburgh, March 1993.

[HMTW95] R. Heckel, J. Müller, G. Taentzer, and A. Wagner. Attributed graph transformations with controlled application of rules. In G. Valiente and

F. Rossello Llompart, editors, *Proc. Colloquium on Graph Transformation and its Application in Computer Science*. Technical Report B - 19, Universitat de les Illes Balears, 1995.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HRW95] H. Herrmanns, M. Rettelbach, and T. Weiss. Formal Characterisation of Immediate Actions in SPA with Nondeterministic Branching. *The Computer Journal*, 1995. to appear.

[Kle75] L. Kleinrock. *Queuing Systems Volume I: Theory*. John Wiley, New York, NY, 1975.

[KLTW93] M. Korff, M. Löwe, G. Taentzer, and A. Wagner. *Algebraische Graphgrammatiken — zwischen Theorie und Praxis — Skript zur Lehrveranstaltung "Graphtransformation und Petrinetze"(in german)*. Technical University of Berlin, Department of Computer Science, 1993.

[Lan] A. Lang. Parameter estimation for phase-type Distributions, Part 1: Fundamentals and existing methods. Master's thesis, Department of Statistics, Oregon State University.

[Löw93] M. Löwe. Algebraic approach to single-pushout graph transformation. *TCS*, 109:181–224, 1993.

[Mil89] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, London, 1989.

[Neu89] M.F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and their Applications*. Probability: Pure and Applied. Marcel Dekker, INC, New York and Basel, 1989.

[Pet62] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Schriften des Institutes für Instrumentelle Mathematik, Bonn, 1962.

[Rib95] M. Ribaudo. *On the Relationship between Stochastic Petri Nets and Stochastic Process Algebras*. PhD thesis, University of Torino, 1995.

[Tan92] A.S. Tanenbaum. *Computer-Netzwerke*. Wolfram's Verlag, Attenkirchen, 1992.