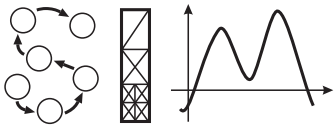
FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
INSTITUT FÜR INFORMATIK (MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG)

Lehrstuhl für Informatik 10 (Systemsimulation)



Beschleunigung der Simulation von Petri-Netzen durch Anwendung und Erweiterung von Methoden aus der Zustandsraumsimulation

Christian Schulz
geb. am 10. Februar 1978 in Dresden

Betreuer:
Prof. Graham Horton,
Dipl.-Inf. Stefan Heller (DaimlerChrysler AG)

Beginn der Arbeit: 21.02.2002
Abgabe der Arbeit: 20.11.2002

Studienarbeit im Fach Informatik

Erklärung:

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 20. November 2002 .....

Abstract

The high description mightiness of extended generalized stochastic petri-nets led to the problem, that in general the behaviour of the modelled system could no longer be calculated numerically. Instead it had to be obtained through simulation, which involved high computing times. Therefore the acceleration of the simulation of petri-nets became important. There already exist methods for the solution of the similar task in state space, with which nice results are achieved. The attempt, to transfer the acceleration method of transformation from state space to petri-nets led to the new technique of switching between transformed and original petri-net.
Up to now, this approach can only be applied to special topologies with as the same conditions as in state space. In this thesis this method is extended in order to increase the amount of topologies, to which this method can be applied. Therefore the condition whether to switch from the transformed to the original petri-net or not is changed and gets an additional part. This even enables the transformation of nested topologies. The experiments demonstrate, that a clear acceleration of the simulation is possible but that it depends on the petri-net.
Each switching from the transformed to the original petri-net and vica versa costs additional computing time. Therefore a technique is presented, that uses already known learning algorithms in order to reduce the amount of unprofitable switching operations. For the experiments in this thesis the sarsa-algorithm from reinforcement learning is used.
An additional new method for the acceleration of the simulation of petri-nets is presented at the end of this thesis. This approach allows the transformation of special single-server topologies without the need of switching to the original petri-net later.

Kurzfassung

Aufgrund der hohen Mächtigkeit von erweiterten generalisierten stochastischen Petri-Netzen kann man das Verhalten des modellierten Systems im Allgemeinen nicht mehr numerisch berechnen, sondern muss es mit Hilfe der Simulation ermitteln. Durch die damit verbundenen hohen Rechenzeiten gewinnt die Beschleunigung der Simulation von Petri-Netzen an Bedeutung. Für die Lösung der ähnlichen Aufgabenstellung im Zustandsraum existieren bereits Verfahren, mit denen gute Resultate erzielt werden. Bei dem Versuch, die Beschleunigungsmethode der Transformation aus dem Zustandsraum auf Petri-Netze zu übertragen, wurde ein Verfahren des dauernden Wechsels zwischen umgeformten und originalem Petri-Netz entwickelt.
Diese Methode ist bisher allerdings nur auf Figuren anwendbar, bei denen exakt die gleichen Bedingungen wie im Zustandsraum gelten. Um die Menge der umformbaren Topologien zu vergrößern, wird in dieser Arbeit dieses Verfahren erweitert. Dabei wird die ursprüngliche Bedingung, welche besagt, wann von dem transformierten auf das original Petri-Netz geschaltet werden muss, entsprechend abgeändert und durch eine Zusatzbedingung ergänzt. Durch diese neue Schaltbedingung wird sogar die Umformung von ineinander geschachtelten Figuren möglich. Die Experimente zeigen, dass sich so je nach Beschaffenheit des Petri-Netzes eine deutliche Beschleunigung der Simulation erzielen lässt.
Jeder Wechsel von dem transformierten auf das originale Petri-Netz und umgekehrt kostet Rechenaufwand und damit auch Rechenzeit. Aus diesem Grund wird ein Verfahren vorgestellt, dass es mit Hilfe von bereits bekannten Lernverfahren erlaubt, die Anzahl an unrentablen Schaltvorgängen zu verringern. In den entsprechenden Experimenten wird dabei das Sarsa-Verfahren des Reinforcement Learning verwendet.
Gegen Ende der Arbeit wird außerdem noch ein neuer Ansatz zur Beschleunigung der Simulation von Petri-Netzen präsentiert. Dieser ermöglicht die Transformation von bestimmten Single-Server Topologien, ohne dass ein späteres Zurückschalten auf das originale Petri-Netz notwendig ist.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Um das Verhalten von Systemen wie z. B. den Ausstoß einer Produktionsanlage für Autos untersuchen zu können, ist es zuerst einmal notwendig, diese Systeme und ihre Eigenschaften formal korrekt und einheitlich zu beschreiben. Deshalb erfolgt die Darstellung dieser Systeme nach wohldefinierten Regeln, zusammengehörige Regeln bilden eine Modellierungssprache. Beispiele für solche Sprachen sind Automaten, Zustandsräume und Petri-Netze. Die verschiedenen, bereits existierenden Modellierungssprachen bieten unterschiedliche Vor- und Nachteile. Ein großer Vorteil von erweiterten generalisierten stochastischen Petri-Netzen ist ihre hohe Mächtigkeit, d.h. mit ihnen lassen sich viele unterschiedliche Systeme darstellen. Aus diesem Grund werden bei der DaimlerChrysler AG auch erweiterte generalisierte stochastische Petri-Netze zur Analyse verschiedenster, teilweise sicherheitsrelevanter Systeme verwendet.

Petri-Netze wurden bereits 1962 von Carl Adam Petri in dessen Dissertationsarbeit vorgestellt [Bau90]. Mit Hilfe dieser Petri-Netze lassen sich unter anderem Konkurrenz-, Ausschluß- und Synchronisationsprozesse modellieren. Das Verhalten eines durch ein solches Petri-Netz dargestellten Systems lässt sich direkt aus dem Petri-Netz durch eine numerische Analyse berechnen. Allerdings ist die Mächtigkeit dieser noch relativ einfachen Petri-Netze begrenzt. So lassen sich z. B. keine zeitlichen Verzögerungen in einem System darstellen. Im Laufe der Jahre wurde von verschiedenen Personen und Gruppen die Mächtigkeit der Petri-Netze vergrößert, was unter anderem zu den von der DaimlerChrysler AG verwendeten erweiterten generalisierten stochastischen Petri-Netzen führte. Mit dieser Erweiterung der Mächtigkeit ist aber auch ein bedeutender Nachteil verbunden: Das Verhalten eines durch ein erweitertes generalisiertes stochastisches Petri-Netz modellierten Systems lässt sich im allgemeinen Fall nicht mehr numerisch berechnen. Deshalb muss auf die Alternative, die Simulation eines Petri-Netzes, zurückgegriffen werden. Dabei bildet ein sogenannter Simulator die in dem Petri-Netz eintretenden Ereignisse über die Zeit nach. In dem Petri-Netz für eine Produktionsanlage für Autos kann ein solches Ereignis beispielsweise das Fertigstellen des Einbaus der Sitze durch Roboter A sein.

Die Tatsache, dass die nachgebildeten Ereignisse in einem Petri-Netz zufällig, entsprechend gewisser Wahrscheinlichkeiten auftreten, bewirkt, dass das während der Simulation beobachtete Verhalten ebenfalls zufällig sein kann. Um daher zuverlässige Aussagen über das Verhalten eines Systems treffen zu können, muss das entsprechende Petri-Netz mehrmals simuliert und die einzelnen Ergebnisse anschließend gemittelt werden. Damit erhöht sich aber auch die für die Gewinnung verlässlicher Ergebnisse benötigte Rechenzeit. Des-

weiteren ist für die DaimlerChrysler AG insbesondere die Analyse von neu entwickelten Systemen von Interesse. Dabei soll häufig das Verhalten des Systems unter verschiedenen Parametereinstellungen untersucht werden. Dies bedeutet aber wiederum, dass für jede interessante Kombination von Parameterwerten das entsprechende Petri-Netz mehrmals simuliert werden muss. Dadurch gewinnt die dafür benötigte Zeitdauer immens an Bedeutung, welche je nach Komplexität des Systems mehrere Stunden, Tage, teilweise sogar Wochen dauern kann.

Diese doch hohen Rechenzeiten führen zu dem Wunsch, den für die Simulation eines erweiterten generalisierten stochastischen Petri-Netzes notwendigen Rechenaufwand und damit die benötigte Zeitdauer zu reduzieren. In [Lub00] ist es gelungen, die Simulation von Zustandsräumen doch merklich zu beschleunigen. Allerdings ist die Mächtigkeit von Zustandsräumen nicht für alle zu analysierenden Systeme ausreichend. Dadurch wird es interessant, die in [Lub00] entwickelten Methoden auf die Simulation von erweiterten generalisierten stochastischen Petri-Netzen zu übertragen.

1.1 Bisherige Arbeiten und Eingliederung dieser Arbeit

Wie bereits erwähnt, ist es in [Lub00] gelungen, die Simulation von Zustandsräumen zu beschleunigen. Die dabei verwendeten Verfahren lassen sich in strukturerhaltende und strukturverändernde einteilen. Bei den strukturverändernden Methoden erfolgt eine Transformation des Zustandsraumes. Aufgrund der guten Ergebnisse wurde bereits in [Lub01] versucht, diese Methoden auf Petri-Netze zu übertragen. Die direkte Anwendung gelang allerdings nur für die strukturverändernden Verfahren und auch hier nur für die Transformation von Infinite-Server-Figuren, welche zudem nur selten vorkommen. Die Probleme bei nicht Infinite-Server-Figuren führten zu der Idee, nur die Topologien umzuformen, bei denen zustandsraum-analoge Bedingungen herrschen. Die Sicherstellung dieser Bedingungen bewirkt einen immer wiederkehrenden Wechsel zwischen dem transformierten und dem originalen Petri-Netz.

In dieser Arbeit wird diese Vorgehensweise so erweitert, dass auch Figuren transformiert werden können, die nicht exakt zustandsraum-analoge Bedingungen haben. Desweiteren wird die Umformung von ineinander geschachtelten Topologien ermöglicht. Außerdem wird mit Hilfe von bereits existierenden Lernverfahren versucht, die Anzahl sich nicht lohnender Wechsel zwischen dem transformierten und originalen Petri-Netz zu verringern. Gegen Ende der Arbeit wird außerdem noch eine neue Vorgehensweise für die Umformung von bestimmten Single-Server-Topologien vorgestellt, bei der kein Wechsel mehr notwendig ist.

Die Implementierungen in dieser Arbeit wurden in den ebenfalls in [Lub01] in Java entwickelten Simulator für Petri-Netze integriert. Die Experimente wurden auf einem AMD Athlon 1GHz mit 512MB Hauptspeicher durchgeführt.

1.2 Aufbau der Arbeit

In Kapitel 2 werden der Zustandsraum sowie die Simulation desselben vorgestellt. Anschließend werden die in [Lub00] gefundenen Beschleunigungsmethoden für die Simulation von Zustandsräumen erläutert. Desweiteren werden Petri-Netze eingeführt und ihre Simulation behandelt. Das Kapitel endet mit der Vorstellung der Probleme, die bei dem Versuch entstehen, die Beschleunigungsmethoden aus dem Zustandsraum direkt auf Petri-Netze zu übertragen. Der aufgrund dieser Schwierigkeiten gewählte Ansatz des immer wiederkehrenden Wechsels zwischen transformiertem und originalem Petri-Netz wird in Kapitel 3 dargelegt und erweitert. Ebenso wird die Umformung von geschachtelten Figuren behandelt und außerdem wird das Potential des mit Hilfe dieser Methode zu erzielenden Gewinns untersucht. Das Kapitel 4 stellt schließlich die Transformation einzelner Figuren vor, zeigt deren Korrektheit und untersucht den damit möglichen Gewinn an Rechenzeit. Die Verwendung bereits vorhandener Lernverfahren zur Verminderung der Anzahl an unrentablen Schaltvorgängen wird in Kapitel 5 erläutert. Den neu entwickelten Ansatz zur Transformation von Single-Server-Topologien präsentiert das Kapitel 6 und durchleuchtet ihn auf Korrektheit und erzielbaren Gewinn hin. In Kapitel 7 erfolgt der Ausblick auf sinnvolle Erweiterungsmöglichkeiten. Einige wichtige Details in Bezug auf die Implementierung werden im Anhang A beschrieben. Der Anhang B schließlich enthält die Abbildungen für die Untersuchung der Korrektheit der verschiedenen Transformationen.

Kapitel 2

Zustandsraum und Petri-Netz

Der Wunsch nach einer Beschleunigung der Simulation von Petri-Netzen führte zu der Idee, bereits bekannte Methoden aus dem Zustandsraum auf Petri-Netze zu übertragen. In diesem Kapitel werden zunächst in 2.1 der Zustandsraum sowie in 2.2 die in [Lub00] für diesen entwickelten Beschleunigungsmethoden vorgestellt. Anschließend werden in 2.3 Petri-Netze eingeführt, deren Simulation in 2.4 behandelt wird. In 2.5 wird auf die Probleme eingegangen, die sich bei der Anwendung der Beschleunigungsmethoden aus dem Zustandsraum auf Petri-Netze ergeben.

2.1 Zustandsraum

Ein Zustandsraum wird durch einen gerichteten Graphen dargestellt. Dieser besteht aus einer Knotenmenge V (vertices), einer Kantenmenge E (edges) sowie einem Startzustand $s \in V$. Abbildung 2.1 zeigt einen Beispiel-Zustandsraum mit v_1 als Startzustand. In dem Graphen repäsentiert jeder Knoten einen Zustand und jede Kante einen möglichen Zustandsübergang. Eine Kante $e_i = (v_{i,1}, v_{i,2}, X_i)$ beginnt bei dem Knoten $v_{i,1}$ und endet bei dem Knoten $v_{i,2}$. Außerdem bekommt jede Kante e_i eine Zufallsvariable X_i zugewiesen, deren Wert durch eine beliebige Wahrscheinlichkeitsverteilung bestimmt wird. Hierbei sind lediglich Verteilungen erlaubt, die ausschließlich positive Zahlen liefern. In der späteren Simulation entspricht die jeweilige Zeitdauer, die dieser Zustandswechsel benötigt, dem jeweiligen Wert dieser Zufallsvariablen.

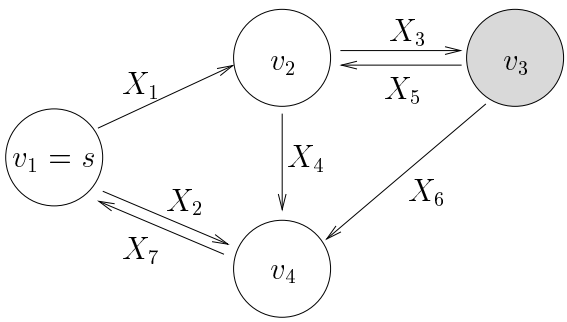


Abbildung 2.1: Beispiel für einen Zustandsraum; an jeder Kante e_i steht ihre Zufallsvariable X_i

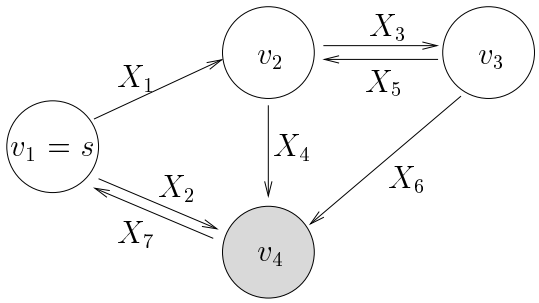


Abbildung 2.2: Der Beispiel-Zustandsraum nach einem möglichen Zustandsübergang

Der in Abbildung 2.1 dargestellte Zustandsraum enthält vier Zustände und sieben Zustandsübergänge, der aktuelle Zustand ist v_3: $v_{active} = v_3$ und an jeder Kante e_i steht ihre Zufallsvariable X_i.
Die Simulation des Zustandsraumes beginnt bei dem Zeitpunkt $\tau = 0$, der aktive Zustand ist der Startzustand $v_{active} \in V = s$. Der anschließende Algorithmus ist während der ganzen Simulation derselbe. Es werden alle n Zustandsübergänge, die vom aktuellen Zustand aus möglich sind $(e_i = (v_{active}, v_{i,2}, X_i))$, sowie deren benötigte Zeitdauer t_i ermittelt. Dabei wird ein neuer Wert für die Zufallsvariable X_i berechnet, die Dauer t_i entspricht diesem neuen Wert. Anschließend wird die Kante $e_{shortest}$ mit der kürzesten Zeit $t_{shortest} = \min\{t_i\};\ i = 1, \ldots, n$ bestimmt. Die globale Zeit wird erhöht $\tau = \tau + t_{shortest}$, der Zielzustand der Kante ist der neue aktive Zustand $v_{active} = v_{shortest,2}$ und der Algorithmus beginnt von vorn.
In Abbildung 2.1 ist $v_{active} = v_3$. Die möglichen Zustandsübergänge sind $e_5(v_3, v_2, X_5)$ und $e_6(v_3, v_4, X_6)$. Angenommen, es gelte $t_6 < t_5$. Damit gilt $e_{shortest} = e_6$ und $t_{shortest} = t_6$. Daraufhin erhöht sich die Gesamtzeit $\tau = \tau + t_6$ und es ändert sich der aktuelle Zustand $v_{active} = v_4$. Abbildung 2.2 zeigt den Zustandsraum nach diesem Übergang.
Dieser Algorithmus wiederholt sich solange, bis ein vorher definiertes Abbruchkriterium erfüllt ist. Ein solches Abbruchkriterium kann z. B. das Erreichen eines bestimmten Endzustandes sein, eine festgesetzte Gesamtsimulationszeit, eine Anzahl an gesamten oder bestimmten Zustandsübergängen oder eine Anzahl, die ein bestimmter Zustand maximal besucht werden darf. Welches Kriterium auch gewählt wird, es muss stets darauf geachtet werden, dass keine unendliche Wiederholung des Algorithmus entstehen kann.
Damit am Ende der Simulation eine Statistik über das Verhalten des Zustandsraumes erstellt werden kann, müssen die entsprechenden Werte bereits während der Simulation berechnet werden. Die dabei ermittelten Werte können je nach Zweck der Simulation verschieden sein, z. B. die durchschnittliche Aufenthaltsdauer in einem Zustand, die mittlere Zeitdauer eines Zustandsüberganges oder die Häufigkeit eines bestimmten Zustandsüberganges.

2.2 Beschleunigung im Zustandsraum

Da die Simulation von Zustandsräumen wie in 2.1 dargestellt sehr zeitaufwendig werden kann, wurden in [Lub00] unter anderem die im Folgenden beschriebenen Möglichkeiten zur Beschleunigung entwickelt und getestet.

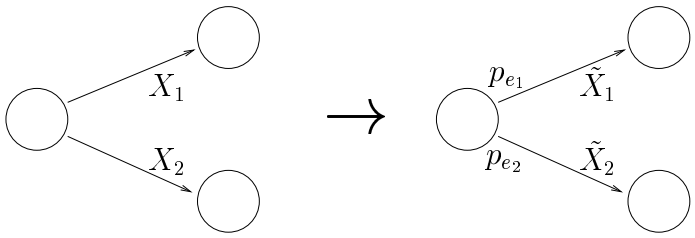


Abbildung 2.3: Berechnung der Verzweigungswahrscheinlichkeiten und Anpassung der Verteilungen der Zufallsvariablen

2.2.1 Verzweigungswahrscheinlichkeiten

Diese Idee beruht auf der Tatsache, dass von jedem Zustand immer die gleichen n Zustandsübergänge möglich sind. Anstatt alle n Zeitdauern zu berechnen und anschließend zu vergleichen, wäre es weniger rechenintensiv, wenn sich zuerst ein Zustandsübergang auswählen ließe und danach lediglich für diesen die Zeitdauer berechnet würde. Dafür ist es notwendig anhand der Verteilungsfunktionen der Zufallsvariablen der einzelnen Übergänge die Wahrscheinlichkeiten zu berechnen mit der ein Zustandswechsel vorgenommen wird. Da die Verteilungen konstant sind, lassen sich diese in einem Vorverarbeitungsschritt vor der Simulation errechnen.
Von einem Zustand v_k sind n Zustandsübergänge möglich. Damit bezeichnet die Verzweigungswahrscheinlichkeit p_{e_i} einer Kante e_i die Wahrscheinlichkeit, dass $t_i < t_j$, $j = 1, \ldots, i-1, i+1, \ldots, n$. Dafür gilt [Lub00]

$$p_{e_i} = P((t_i < t_1) \wedge \ldots \wedge (t_i < t_{i-1}) \wedge (t_i < t_{i+1}) \wedge \ldots \wedge (t_i < t_n)) \quad (2.1)$$

$$p_{e_i} = \int_0^\infty f_{X_i}(t) \cdot (1 - F_{X_1}(t)) \cdot \ldots \cdot (1 - F_{X_{i-1}}(t)) \cdot (1 - F_{X_{i+1}}(t)) \cdot \ldots \cdot (1 - F_{X_n}(t))\, dt \quad (2.2)$$

mit f_{X_i} als Dichte- und F_{X_i} als Verteilungsfunktion der Wahrscheinlichkeitsverteilung von X_i der Kante e_i.
Zusätzlich zur Berechnung der Wahrscheinlichkeit, muss der Tatsache Rechnung getragen werden, dass im Simulationsschritt die Zeitdauer eines Zustandsüberganges erst nach der Auswahl einer Kante ermittelt wird. Demzufolge müssen die Verteilungsfunktionen angepasst werden. Für die geänderte Dichtefunktion $f_{\tilde{X}_i}$ von X_i gilt [Lub00]:

$$f_{\tilde{X}_i}(t) = f_{X_i}(t) \mid ((t_i < t_1) \wedge \ldots \wedge (t_i < t_{i-1}) \wedge (t_i < t_{i+1}) \wedge \ldots \wedge (t_i < t_n)) \quad (2.3)$$

$$f_{\tilde{X}_i}(t) = f_{X_i}(t) \frac{(1 - F_{X_1}(t)) \cdot \ldots \cdot (1 - F_{X_{i-1}}(t)) \cdot (1 - F_{X_{i+1}}(t)) \cdot \ldots \cdot (1 - F_{X_n}(t))}{p_{e_i}} \quad (2.4)$$

Abbildung 2.3 zeigt das Ergebnis der Berechnung der Verzweigungswahrscheinlichkeiten.

2.2.2 Transformation des Zustandsgraphen

Eine weitere Möglichkeit die Simulation zu beschleunigen ist die Reduzierung des Zustandsraumes mittels Ersetzung uninteressanter Bereiche. Dabei bedeutet uninteressant, dass über diesen Teil keine Statistik-Berechnung nötig ist. Die folgenden Verfahren arbeiten stets auf solchen Bereichen.

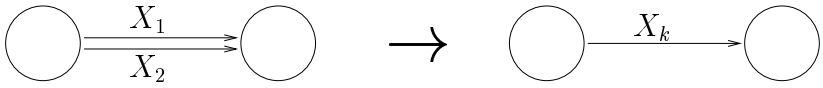


Abbildung 2.4: Vereinigung zweier Parallelkanten

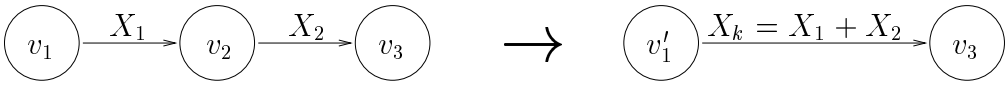


Abbildung 2.5: Verkürzung einer Sequenz

Bei der *Vereinigung von Parallelkanten* werden n parallele Kanten e_i, d.h. mit gleichem Start- und Endzustand, also $v_{1,1} = v_{j,1}$ und $v_{1,2} = v_{j,2}$ mit $j = 2, \ldots, n$, durch eine Kante e_k mit $v_{k,1} = v_{1,1}$ und $v_{k,2} = v_{1,2}$ ersetzt. Für die Zufallsvariable X_k der neuen Kante gilt [Lub00]

$$X_k = \min\{X_i\}; \qquad i = 1, \ldots, n \tag{2.5}$$

und damit für die Verteilungsfunktion $F_{X_k}(t)$ [Lub00]

$$F_{X_k}(t) = 1 - \prod_{i=1}^{n} (1 - F_{X_i}(t)). \tag{2.6}$$

Abbildung 2.4 zeigt eine Vereinigung von zwei Parallelkanten.
Durch die Einsparung eines kompletten Zustandes, verspricht die *Sequenzverkürzung* eine gute Beschleunigung. Eine Sequenz ist wie in Abbildung 2.5 gegeben, wenn von einem Zustand v_1 genau eine Kante e_1 weggeht und von dem Endzustand dieser Kante v_2 wiederum exakt eine Kante e_2 startet. Der Knoten v_2 ist der mittlere und v_3 der letzte Zustand der Sequenz. In den Knoten v_2 darf ausschließlich die Kante e_1 enden. Ist der erste Knoten der Sequenz der aktive Zustand $v_{active} = v_1$, so ist lediglich der Übergang zu v_2 möglich. Von v_2 wiederum ist der einzig mögliche Folgezustand v_3. Demzufolge gilt immer, dass nachdem v_1 der aktive Zustand des Zustandsraumes war, v_3 nach einer gewissen Zeitdauer der aktuelle Zustand wird. Diese entspricht genau der Zeit, die die Zustandsübergänge e_1 und e_2 benötigen. Dank diese Tatsache lässt sich die Sequenz auch einfacher durch eine einzelne, neue Kante e_k darstellen, welche bei $v_{k,1} = v_1$ beginnt und bei $v_{k,2} = v_3$ endet. Für die Zufallsvariable X_k der neuen Kante gilt [Lub00]

$$X_k = X_1 + X_2, \tag{2.7}$$

und somit ergibt sich die Dichtefunktion $f_{X_k}(t)$ aus der Faltung der Funktionen $f_{X_1}(t)$ und $f_{X_2}(t)$ [Lub00].

$$f_{X_k}(t) = f_{X_1}(t) * f_{X_2}(t) = \int_0^t f_{X_1}(\lambda) f_{X_b}(t - \lambda) d\lambda \tag{2.8}$$

Abbildung 2.5 zeigt das Ergebnis einer Sequenzverkürzung. Durch die erhöhte Zeitdauer des einzig möglichen Zustandsüberganges von v_1 ändert sich die Aufenthaltswahrscheinlichkeit in v_1, dieser wird deshalb nach der Transformation als v_1' bezeichnet.

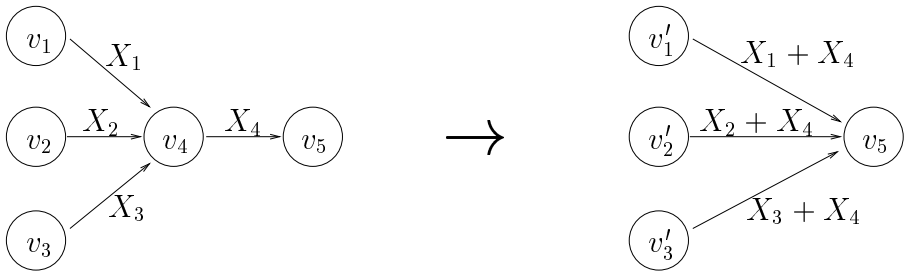


Abbildung 2.6: Verkürzung einer Merge-Figur

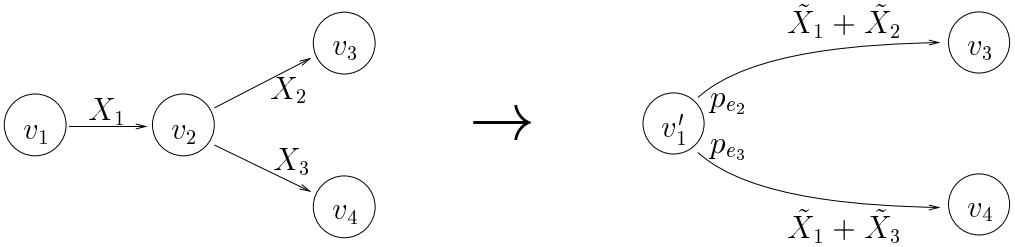


Abbildung 2.7: Verkürzung einer Split-Figur

Eine ganz analoge Ersetzung lässt sich bei einer *Merge-Figur* wie in Abbildung 2.6 vornehmen. Hierbei können beliebig viele (n) Kanten in dem mittleren Zustand enden, von welchem aus wieder genau eine Kante e_4 startet. Bei der Verkürzung wird jede der n Kanten einzeln wie bei der Seqeunz durch Faltung mit der Kante e_4 zusammengefügt und es entstehen n neue Kanten.

Bei der *Split-Figur* (Abbildung 2.7) dürfen im Gegensatz zur Sequenz n Kanten vom mittleren Zustand aus beginnen. Diese können wieder mit der Kante e_1, die im mittleren Zustand endet, zusammengefasst werden. Hierbei muss jedoch darauf geachtet werden, dass sich die Wahrscheinlichkeit für einen bestimmten Zustandsübergang nicht ändern darf. Werden lediglich die Dichtefunktionen wie bei der Sequenz gefaltet, ist dies nicht gegeben. Allerdings wurde bereits in 2.2.1 die Auswahl einer Kante von der Berechnung der Zeitdauer getrennt. Werden die Verzweigungswahrscheinlichkeiten kombiniert mit der Faltung der angepassten Wahrscheinlichkeitsverteilungen der Zufallsvariablen, ergibt sich damit eine korrekte Ersetzung für die Split-Figur. Ein Beispiel zeigt Abbildung 2.7.

Durch diese Kombination ist es auch möglich die Bedingung, dass nur die Kante e_1 vom vorderen Zustand starten darf, aufzuheben. Damit erhält man eine Ersetzung einer ganz allgemeinen erweiterten Split-Figur. Abbildung 2.8 stellt ein Beispiel dar.

Durch Berechnung der Verzweigungswahrscheinlichkeiten und anschließendes Falten lassen sich die verschiedenen Figuren miteinander kombinieren und dadurch viele Bereiche des Zustandsraumes vereinfachen. Ein bisher unbetrachtetes Problem ist eine Schleife, also wenn der Start- und der Endzustand einer Kante identisch sind. Die in [Lub00] gefundene Lösung basiert auf einer unendlichen Summe von Faltungen, wodurch sich ein Problem mit der Genauigkeit ergibt. Zusätzlich besteht ein hoher Rechenaufwand für Faltungen, weswegen auf eine weitere Untersuchung verzichtet wurde.

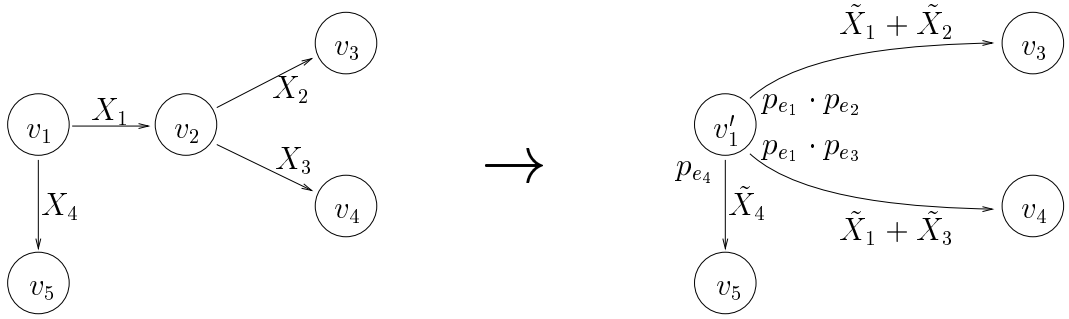


Abbildung 2.8: Verkürzung einer erweiterten Split-Figur

2.3 Petri-Netz

Mit Zustandsräumen lassen sich einfache Abläufe gut modellieren. Problematisch ist jedoch die Nachbildung des Verhaltens von Systemen mit parallelen, synchronisierten Abläufen. So ist z.B. die Modellierung eines Boxenstops in der Formel 1 mit Hilfe eines Zustandsgraphen sehr kompliziert. Bei dem Stop finden der Wechsel der vier Reifen und das Auftanken parallel statt, das Auto kann aber erst weiter fahren wenn alle fünf Vorgänge abgeschlossen sind. In einem möglichen Zustandsraum für den Boxenstop sind von dem Zustand ,,Ankunft in der Box" die Übergänge in den Zustand ,,Tanken fertig" oder ,,Reifen X fertig" möglich. Wechselt der Graph allerdings in den Zustand ,,Tanken fertig", so sind nun vier Übergänge in vier komplett neue Zustände möglich, für jeden Reifen einer. Da schon Zeit für das Tanken vergangen ist, müssen diese Übergänge andere Verteilungsfunktionen haben als die Übergänge für die Reifen von dem Zustand ,,Ankunft in der Box" aus. Außerdem wurde die Gesamtzeit für jeden Reifenwechsel eigentlich schon einmal berechnet. Demzufolge wird der Vorgang des Boxenstops nicht ganz korrekt nachgebildet. Desweiteren ist der Zustandsgraph ziemlich groß (über 30 Zustände) und die Bestimmung der Verteilungen für die Übergänge recht kompliziert.
Die Modellierung solcher Synchronisationsprozesse ist dagegen mit Hilfe von Petri-Netzen relativ einfach. Abbildung 2.9 zeigt das zum Boxenstop gehörige Petri-Netz, welches im Gegensatz zu dem Zustandsraum anschaulich ist.
In diesem Abschnitt werden erweiterte generalisierte stochastische Petri-Netze, im Folgenden einfach als Petri-Netze bezeichnet, vorgestellt. Tiefergehende Informationen findet der interessierte Leser in [Bau90] sowie [MBC+95].

Ein Petri-Netz ist ein gerichteter Graph mit zwei verschiedenen Knotentypen und Kanten. Als Knoten gibt es *Stellen* und *Transitionen*, eine Kante darf nur von einer Stelle zu einer Transition oder umgekehrt gehen. Zusätzlich können Stellen sogenannte *Token* enthalten. Stellen werden als Kreise, Transitionen als Rechtecke oder Balken und Token als Punkte dargestellt. Ein Token darf nur in Stellen vorkommen, eine Stelle kann vorerst beliebig viele Token enthalten, wobei alle Token ununterscheidbar sind.
Das Petri-Netz des Boxenstops enthält die Stellen P_1 bis P_{10} und P_{racing} sowie die Transitionen $T_{boxstop}$, T_{tireX}, T_{fuel} und T_{go}. Die Stelle P_{racing} enthält einen Token.
In einem Petri-Netz gibt es verschiedene Arten von Kanten. *Inhibitorische Kanten*, die im Gegensatz zu den ,,normalen" eine Transition deaktivieren, laufen stets von einer Stelle zu

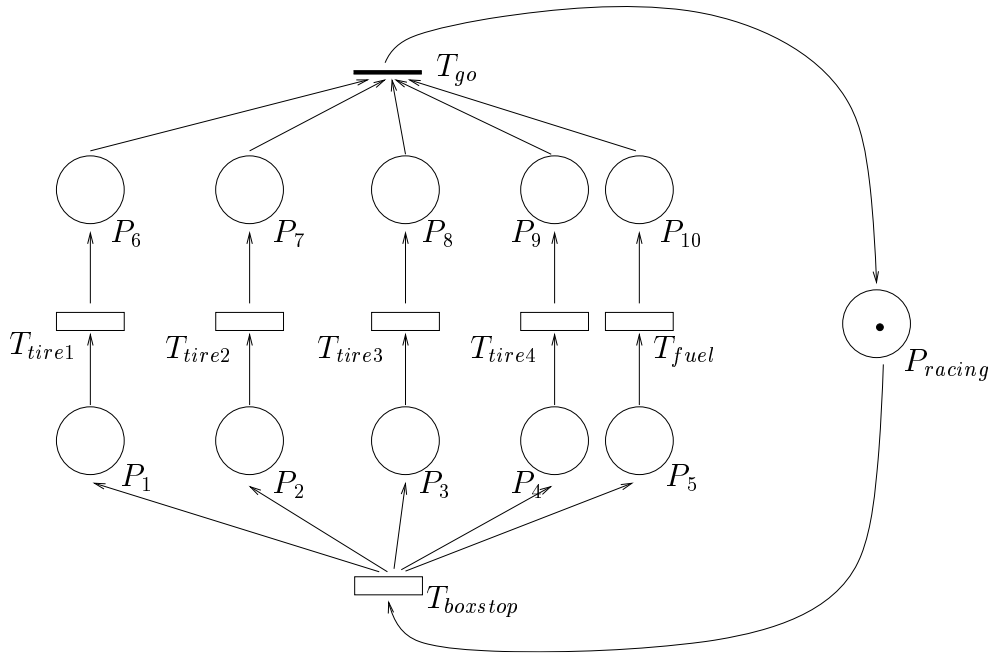


Abbildung 2.9: Petri-Netz für einen Boxenstop in der Formel 1

einer Transition und werden als Linie mit einem kleinen Kreis am Ende dargestellt. Alle anderen, ,,normalen" Kanten werden als Pfeil dargestellt. ,,Normale" Kanten, die von einer Stelle zu einer Transition laufen, werden im Folgenden als *Eingangskanten*, ,,normale" Kanten von Transitionen zu Stellen als *Ausgangskanten* bezeichnet. Alle Kanten haben eine *Vielfachheit* m_i, die durch eine kleine Zahl unter der Kante dargestellt wird. Bei Kanten mit Vielfachheit $m_i = 1$ kann auf die Zahl verzichtet werden. In dem Petri-Netz des Boxenstops befinden sich nur Eingangs- und Ausgangskanten mit der Vielfachheit eins und keine inhibitorischen Kanten.
Bei den Transitionen wird zwischen zeitbehafteten und zeitlosen Transitionen unterschieden. Die zeitlosen Transitionen werden als Balken, die zeitbehafteten als Rechtecke dargestellt. In dem Boxenstop Petri-Netz ist T_{go} eine zeitlose, alle anderen sind zeitbehaftete Transitionen. Eine zeitbehaftete Transition T_i steht für einen Vorgang, der Zeit benötigt. Die entsprechende Dauer wird durch die zu T_i gehörige Wahrscheinlichkeitsverteilung X_i beschrieben. Es sind nur Verteilungen erlaubt, die keine negativen Zahlen liefern. Eine zeitlose Transition T_j steht für ein unmittelbar, mit der Wahrscheinlichkeit p_j eintretendes Ereignis.
In Abbildung 2.10 werden die einzelnen Elemente eines Petri-Netzes veranschaulicht. Dabei sind T_{Enter} und T_{Leave} zeitbehaftete, T_{Free} eine zeitlose Transition, P_{Queue} und $P_{Service}$ sind Stellen. In der Stelle P_{Queue} sind drei Token enthalten. Die Ausgangskante von Transition T_{Enter} zur Stelle P_{Queue} hat die Vielfachheit drei, alle anderen Kanten die Vielfachheit eins. Die Transition T_{Enter} hat die Verteilung X_{Enter}, T_{Leave} die Verteilung X_{Leave} und die Transition T_{Free} die Wahrscheinlichkeit $p_{Free} = 1$.
Die Stellen, bei denen die Eingangskanten einer Transition T_i beginnen, werden im Folgenden auch als *Eingangsstellen* der Transition T_i, die Stellen, in die die Ausgangskanten einer Transition enden, auch als *Ausgangsstellen* der Transition T_i bezeichnet.

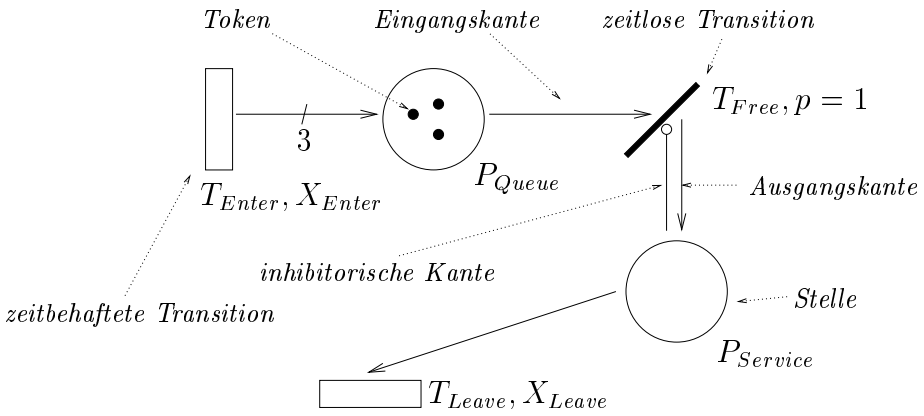


Abbildung 2.10: Veranschaulichung eines Petri-Netzes anhand einer Art Warteschlange

Eine Transition T_i heißt *aktiviert*, wenn folgende Bedingungen erfüllt sind:

1. In jeder Eingangsstelle P_j sind **mindestens** m_j Token enthalten, wobei m_j die Vielfachheit der Eingangskante von P_j ist.

2. In jeder Stelle P_k, von der aus eine inhibitorische Kante zu T_i geht, sind **weniger** als m_k Token enthalten, wobei m_k die Vielfachheit der inhibitorischen Kante von P_k ist.

Die Bedingung 2 lässt sich einfacher verständlich auch folgendermaßen ausdrücken: Eine Transition T_i ist **nicht** aktiviert, wenn in mindestens einer Stelle P_k, von der aus eine inhibitorische Kante zu T_i geht, $\geq m_k$ Token enthalten sind (m_k ist wieder die Vielfachheit der inhibitorischen Kante von P_k). Wird eine Transition, die bisher nicht aktiviert war, aktiviert, heißt dies im folgenden *Aktivierung*.
Ist eine Transition aktiviert, so *feuert* sie eventuell. Eine zeitlose Transition feuert sofort nachdem sie aktiviert wurde, bei einer zeitbehafteten muss erst eine zufällige, entsprechend ihrer Wahrscheinlichkeitsverteilung bestimmte Zeitdauer vergehen. Grundsätzlich gilt: Eine zeitlose Transition feuert immer vor einer zeitbehafteten. Eine Transition T_f entfernt aus jeder ihrer Eingangsstellen P_i beim Feuern m_i Token, wobei m_i die Vielfachheit der Eingangskante von P_i ist. Desweiteren erzeugt sie in jeder Ausgangsstelle P_j m_j Token. Hierbei ist m_j die Vielfachheit der Ausgangskante zu P_j.
In dem Petri-Netz des Boxenstops befindet sich ein Token in der Stelle P_{racing}. Damit ist die Transition $T_{boxstop}$ aktiviert. Die Zeitdauer, die vergeht, bis sie feuert, steht für die Zeit, die das Auto auf der Rennstrecke seine Runden dreht, bis es an die Box muss. Nach dem Schalten dieser Transition befindet sich in den Stellen P_1 bis P_5 jeweils ein Token und in P_{racing} keiner mehr. Damit sind T_{tire1} bis T_{tire4} sowie T_{fuel} aktiviert. Auch dies sind zeitbehaftete Transitionen. Die einzelnen Zeitdauern bis zum Feuern stehen hier für die Zeit, die der entsprechende Vorgang benötigt. Erst wenn alle vier Reifen gewechselt sind und das Auto aufgetankt ist, d.h. die entsprechenden Transitionen gefeuert haben, befindet sich in jeder der Stellen P_6 bis P_{10} jeweils ein Token. Damit ist T_{go} aktiviert und feuert sofort, das Auto kann wieder auf die Rennstrecke.
Durch das Feuern einer Transition kann es passieren, dass eine andere, aktivierte Transition auf einmal nicht mehr aktiviert ist. Dies wird auch als *deaktivieren* bezeichnet. Wird

eine deaktivierte Transition wieder aktiviert, so wird eine neue Zeitdauer bis zum Feuern bestimmt.
Der Zustand eines Petri-Netzes ist durch die Verteilung der Token über die Stellen gegeben. Das Petri-Netz in Abbildung 2.10 hat den Zustand $(3;0)$, wobei die erste Zahl die Anzahl an Token in P_{Queue} und die zweite die Anzahl in $P_{Service}$ angibt.

Um die Menge der Systeme zu erweitern, die sich mit Petri-Netzen modellieren lassen, wurden in [MBC+95], [Lub01] noch folgende Erweiterungen eingeführt.
Alle Parameter im Petri-Netz können durch Funktionen ersetzt werden, welche den Netzzustand auslesen können.
Als zusätzliche Aktivierungsbedingung kann eine Transition eine *Guardfunktion* bekommen. Dies ist eine Funktion, die entweder *wahr* oder *falsch* zurück gibt. Dabei darf sie, wie alle Funktionen, die Anzahl an Token in den Stellen abfragen. Eine Transition ist **nicht** aktiviert, wenn sie eine Guardfunktion besitzt und diese den Wert falsch zurückliefert.
Ebenso optional können Stellen eine *Stellenkapazität* C erhalten: Eine Transition T_j wird deaktiviert bzw. ist nicht aktiviert, wenn durch ihr Feuern in einer Stelle P_i mehr als C_i Token enthalten wären.
Alle Transitionen bekommen eine Priorität p zugeordnet. Wären nach den bisherigen Regeln zwei Transitionen unterschiedlicher Priorität aktiviert, so ist nur die mit höherer Priorität aktiviert. Verschiedene Prioritäten zwischen zeitlosen und zeitbehafteten Transitionen haben keine Auswirkungen auf die Aktivierung sowie die Feuer-Reihenfolge.
Der *Aktivierungsgrad* E_i einer Transition T_i besagt, wie oft T_i aktiviert ist. Ist $E_i = k$ bedeutet dies, dass in jeder Eingangsstelle P_j von T_i mindestens $k \cdot m_j$ Token enthalten sind, m_j ist dabei die Vielfachheit der Eingangskante von P_j zu T_i. War bisher der Aktivierungsgrad einer zeitbehafteten Transition größer als eins, so wurde bisher erst nach dem Feuern eine neue Zeitdauer bestimmt. Eine solche Transition heißt *Single-Server*. Alternativ dazu ist es möglich, jedem einzelnen Aktivierungsgrad quasi eine eigene Transition zur Verfügung zu stellen. Das bedeutet Folgendes: Steigt der Aktivierungsgrad von einer Transition T_i, so wird eine neue Zeitdauer entsprechend X_i ermittelt. Eine neue Transition T_k erhält diese Zeitdauer bis zum Feuern und die gleichen Kanten wie T_i. Nachdem T_k gefeuert hat, wird sie wieder aus dem Netz entfernt. Die Transition T_i stellt demzufolge jedem Aktivierungsgrad eine unabhängig laufende Transition mit einer entsprechend durch X_i ermittelten Zeitdauer bis zum Feuern zur Verfügung. Eine solche Transition T_i heißt *Infinite-Server*. Sinkt durch Deaktivierung der Aktivierungsgrad eines solchen Infinite-Servers, so wird eine der eingefügten Transitionen zufällig ausgewählt und aus dem Netz entfernt. Eine weitere Möglichkeit ist es, nicht unendlich, sondern begrenzt viele Transitionen zur Verfügung zu stellen. Eine solche Transition T_l heißt *Multi-Server* und die maximale Anzahl an Transitionen ist bestimmt durch die *Vielfachheit* $M_{l,max}$.
Bei einer Aktivierung einer zeitbehafteten Transition nach einer Deaktivierung wurde bisher eine neue Verzögerungszeit entsprechend der Wahrscheinlichkeitsverteilung ermittelt. Diese Strategie wird als *race enable* bezeichnet, es gibt noch weitere solcher Strategien (*race policies*). Bei *race age* wird der Wert der Verzögerungszeit bei der Aktivierung auf den Restwert gesetzt, den die letzte Zeitdauer zum Zeitpunkt der Deaktivierung hatte. Mit einer Transition dieser Strategie lässt sich demzufolge ein Prozess modellieren, der nach einer Unterbrechung weiter läuft. Bei *race repeat* wird bei der Aktivierung nach einer Deaktivierung die neue Zeitdauer auf den ursprünglichen Wert der alten Verzögerungszeit

gesetzt. Ein Vorgang, der nach einer Unterbrechung wieder von vorn begonnen werden muss, lässt sich demnach mit Hilfe einer Transition dieser Strategie darstellen. Eine weitere Strategie ist *race reset*. Bei einer Transition mit dieser Strategie wird nach jedem Feuern einer Transition im Petri-Netz eine neue Zeitdauer bis zum Feuern erzeugt.
Jede Transition bekommt eine optionale *Reset-Liste*. Feuert eine Transition, so werden die Zeitdauern bis zum Feuern bei jeder der in der Reset-Liste enthaltenen Transitionen verworfen und anschließend zufällig eine neue bestimmt, sofern die entsprechende Transition aktiviert ist. Dies geschieht bei jeder der verschiedenen race policies mit Ausnahme von race repeat. Hier wird die neue Zeitdauer auf den ursprünglichen Wert der alten gesetzt (analog zur policy).
Schließlich wird noch ein neues Element zum Petri-Netz hinzugefügt: Kostenfunktionen (*rewards*). Diese bieten ein Hilfsmittel zur Beobachtung des Verhaltens des Petri-Netzes. Sie selbst greifen nicht in das Verhalten des Petri-Netzes ein und ändern die bisher vorgestellten Regeln nicht. Eine Kostenfunktion enthält eine Funktion, die direkt vor dem Feuern einer bestimmten oder beliebigen Transition ausgewertet wird. Die berechneten Werte werden je nach Art der Kostenfunktion addiert oder gemittelt.

2.4 Simulation von Petri-Netzen

Wie bereits in der Einleitung erwähnt, lassen sich Petri-Netze mit der in 2.3 vorgestellten Funktionalität nur schwer oder nicht mehr numerisch lösen. Aus diesem Grund wird die Simulation dieser Petri-Netze interessant. Die Funktionsweise eines solchen Simulators soll in dem folgenden Abschnitt vorgestellt werden.
Der Simulator imitiert das Verhalten des Petri-Netzes. Dafür benötigt er eine Gesamtzeit τ. Ein Petri-Netz wie in Abbildung 2.9 oder 2.10 stellt den Zustand eines Petri-Netzes zu einem bestimmten Zeitpunkt, meistens den *Initial Zustand* ($\tau = 0$) dar. Dabei wird eine abstrakte Zeit verwendet. Eine Zeiteinheit der Simulationszeit τ steht dabei je nach Petri-Netz für eine oder mehrere Sekunden, Stunden oder Jahre oder andere physikalische Zeiteinheiten.
Der Simulations-Algorithmus ist während der gesamten Simulation derselbe, er wird in Abbildung 2.11 dargestellt. Zuerst wird kontrolliert, ob zeitlose Transitionen aktiviert sind. Ist dies der Fall, so muss eine davon ausgewählt werden. Anhand der Wahrscheinlichkeiten p_i der Transition T_i wird die Auswahlwahrscheinlichkeit $p_{i,choose}$ bestimmt:

$$p_{i,choose} = \frac{p_i}{\sum\limits_{aktivierte} p_a} \tag{2.9}$$

Entsprechend der verschiedenen $p_{i,choose}$ wird anschließend eine zeitlose Transition zufällig ausgewählt. Diese feuert, d.h. der Simulator entfernt und erzeugt die Token in den entsprechenden Stellen. Dieses Suchen und Schalten von zeitlosen Transitionen findet solange statt, wie mindestens eine zeitlose Transition aktiviert ist.
Der nächste Schritt ist die Suche nach allen aktivierten zeitbehafteten Transitionen. Dabei erzeugt der Simulator für alle zeitbehafteten Transitionen, die neu aktiviert wurden, die entsprechenden Zeitdauern. Diese Zeiten werden im Folgenden als *Laufzeiten* bezeichnet.

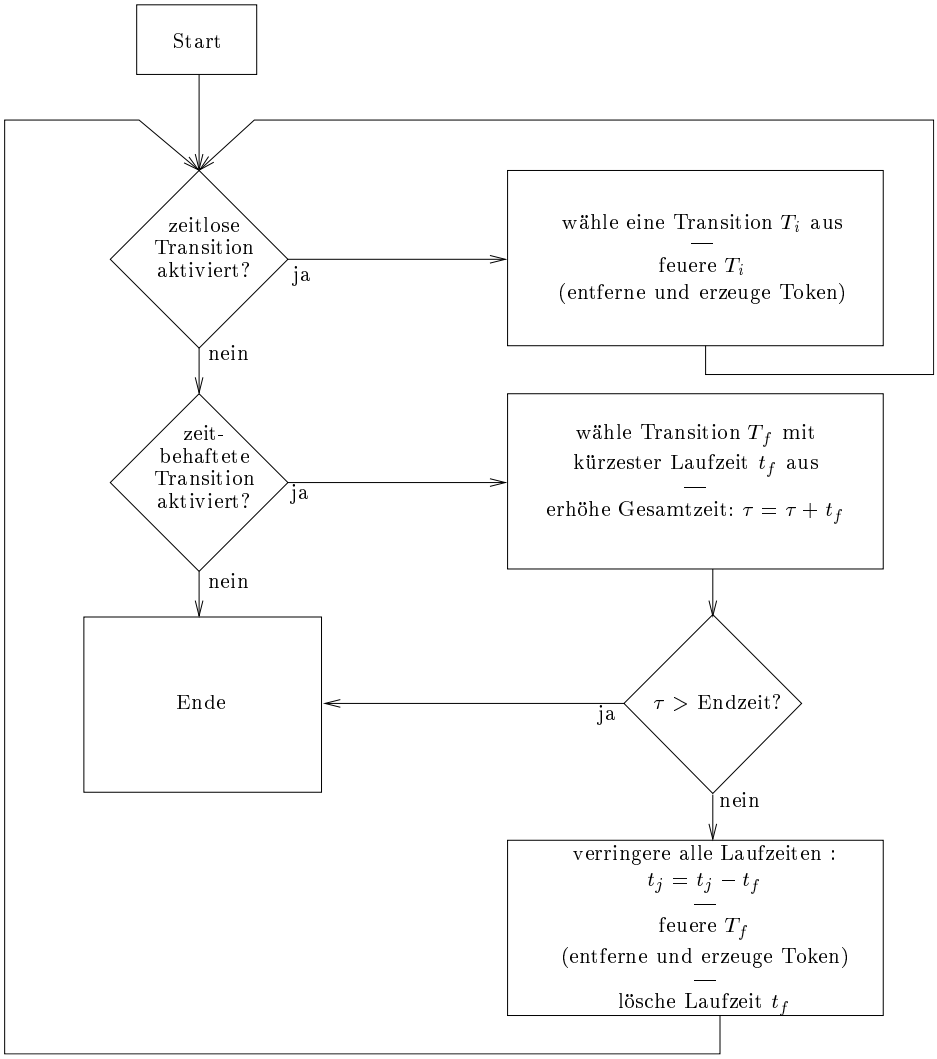


Abbildung 2.11: Der Simulations-Algorithmus für eine Replikation

Aus allen aktivierten zeitbehafteten Transitionen wählt der Simulator dann die Transition mit der kürzesten Laufzeit aus. Haben mehrere die gleiche, so wird eine von diesen zufällig ausgewählt, wobei alle gleichberechtigt sind. Die ausgewählte Transition T_f habe die Laufzeit t_f. Der Simulator erhöht die Gesamtzeit $\tau = \tau + t_f$ und verringert alle Laufzeiten der ebenfalls aktivierten Transitionen um t_f: $t_j = t_j - t_f$. Anschließend feuert T_f, wodurch die Token erzeugt und entfernt und die Laufzeit t_f gelöscht werden. Danach beginnt der Algorithmus von vorn.

In dem Beispiel von Abbildung 2.10 wird eine leicht modifizierte Warteschlange, z. B. für ein Geldinstitut (Bank), dargestellt. Es sei die Gesamtzeit $\tau = 0$. Die Transitionen T_{Enter} und T_{Free} sind aktiviert: T_{Enter} hat keine Eingangskante und keine inhibitorische Kante und ist damit immer aktiviert. In der Stelle P_{Queue} ist mehr als ein und in $P_{Service}$ kein Token enthalten und damit ist T_{Free} aktiviert. Da T_{Free} eine zeitlose Transition ist, feuert sie sofort. Aus P_{Queue} wird ein Token gelöscht und in $P_{Service}$ einer erzeugt (sobald der Schalter der Bank frei ist, kann eine Person aus der Schlange an ihn herantreten). Die

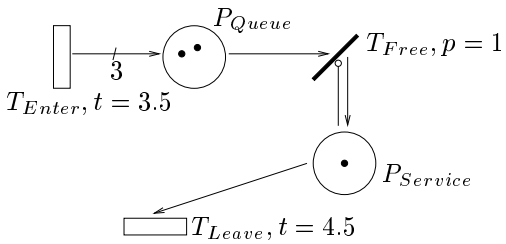


Abbildung 2.12: Das Beispiel-Petri-Netz nach einem Feuern; $\tau = 0$

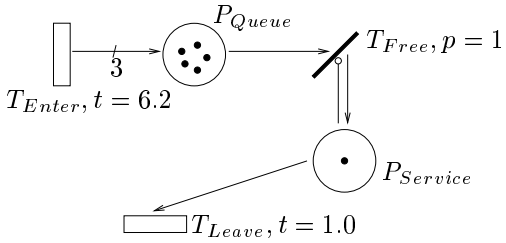


Abbildung 2.13: Das Beispiel-Petri-Netz nach dem zweitem Feuern; $\tau = 3.5$

Gesamtzeit bleibt null. Nun ist zwar in P_{Queue} noch immer mehr als ein Token enthalten, allerdings befindet sich auch in $P_{Service}$ ein Token. Damit ist T_{Free} nun nicht mehr aktiviert. T_{Enter} ist noch immer aktiviert. Da sie keine aktuelle Laufzeit hat, wird eine neue zufällig bestimmt: $t_{Enter} = 3.5$. Außerdem ist nun T_{Leave} aktiviert, es wird die Laufzeit bestimmt: $t_{Leave} = 4.5$. Abbildung 2.12 zeigt das entsprechende Petri-Netz. In diesem Zustand des Petri-Netzes $(2; 1)$ sind ausschließlich zeitbehaftete Transitionen aktiviert, die kürzeste Laufzeit beträgt 3.5. Demzufolge feuert als nächstes die Transition T_{Enter} (drei Personen betreten die Bank), dabei wird die alte Laufzeit von T_{Enter} gelöscht. Nach dem Feuern ist keine zeitlose Transition aber T_{Enter} aktiviert, es muss also eine neue Laufzeit berechnet werden: $t_{Enter} = 6.2$. Das Petri-Netz nach dem Feuern zeigt die Abbildung 2.13.

Der Simulationsalgorithmus wird solange wiederholt, bis entweder keine Transition mehr aktiviert ist oder bis die Simulationszeit eine vorher festgelegte Endzeit erreicht hat.

Auch die vorgestellten Erweitererungen für Petri-Netze sind in einem Simulator realisierbar. Es wird hier nur kurz auf die Multi-Server hingewiesen. Ein solcher lässt sich realisieren, in dem entsprechend viele Laufzeiten für eine Transition erzeugt und in einer Liste verwaltet werden. Bei der Suche nach der nächsten feuernden Transition muss logischerweise die kürzeste dieser Laufzeiten betrachtet werden. Im Gegensatz dazu müssen alle Laufzeiten eines Multi-Servers bei der Verringerung der Laufzeiten geändert werden. Mit der Anzahl der Laufzeiten für eine Transition nimmt damit der Rechenaufwand zu.

Bei der Simulation werden die Laufzeiten zufällig erzeugt. Damit ist aber auch das Verhalten des Petri-Netzes zufällig. Um aussagekräftige Ergebnisse zu erhalten, muss deshalb ein Petri-Netz mehrmals simuliert werden. Dabei wird die Simulation von $\tau = 0$ bis zum Endzeitpunkt (bzw. bis keine Transitionen mehr aktiviert sind) als eine Replikation bezeichnet. Für das Endergebnis einer Simulation werden die Ergebnisse der einzelnen Replikationen gemittelt. Die Ergebnisse, im Folgenden auch *Statistik* genannt, enthalten Aussagen über das Verhalten des Petri-Netzes . Dies können z. B. die maximale oder

mittlere Anzahl an Token in einer Stelle, die Feuerfrequenz einer Transition oder andere Größen sein.

Ein Simulator, der die volle hier vorgestellte Komplexität enthält, wurde in [Lub01] in Java entwickelt. In den Parameter-Funktionen wird die Anzahl der Token in einer Stelle mit Hilfe einer mark-Funktion abgefragt. In dieser steht dem Benutzer die volle Mächtigkeit der Programmiersprache zur Verfügung. Daher ist eine Realisierung bereits bekannter Rechenaufwand-sparender Verfahren wie z. B. die neighbourhood-Berechnung so komplex, dass in [Lub01] darauf verzichtet wurde. Bei der neighbourhood-Berechnung werden für jede Transition T_i die Transitionen ermittelt, deren Aktivierungsgrad sich durch das Feuern von T_i ändern kann. In der Simulation müssen damit nach dem Feuern von T_i lediglich diese Transitionen neu auf Aktiviertheit überprüft werden.
Die Implementierungen in dieser Studienarbeit wurden in diesen Simulator eingebaut. Dabei wurden nur die benötigten Teilbereiche geändert und der Rest unverändert gelassen.

2.5 Anwendbarkeit der Beschleunigungsmethoden des Zustandsraumes auf Petri-Netze

Der hohe Rechenaufwand für die Simulation von Petri-Netzen führt zu dem Wunsch, die benötigte Rechenzeit zu reduzieren. Da für den Zustandsraum die Lösung des analogen Problems gute Resultate erzielt, ergibt sich die Idee, diese Methoden auch auf Petri-Netze anzuwenden. Die dabei auftretenden Probleme werden im Folgenden vorgestellt.

Im Zustandsraum gibt es einen aktiven Zustand, von dem aus weitere Zustandsübergänge möglich sind. Nach einem Zustandswechsel sind die ehemals möglichen nicht mehr von Bedeutung. Das Geschehen im Zustandsraum beschränkt sich demzufolge lokal auf den aktuellen Zustand. Im Petri-Netz dagegen sind viele, parallele Abläufe möglich, verschiedene Token in verschiedenen Stellen können getrennt und unabhängig voneinander unterschiedliche Transitionen aktivieren. Damit ist das Geschehen im Petri-Netz nicht einfach lokal begrenzt und übersichtlich.
Angenommen mit einer Stelle sind zwei zeitbehaftete Transitionen durch Eingangskanten verbunden. In der Stelle befindet sich ein Token, beide Transitionen sind aktiviert. Damit haben beide eine Laufzeit. Nun wird ein zweiter Token in der Stelle erzeugt. Nach dem Feuern einer der Transitionen ist die andere immer noch aktiviert, die Laufzeit wird nicht neu berechnet. Damit hängt aber auch die Wahrscheinlichkeit, welche der beiden Transitionen anschließend feuert, mit dem vorherigen Feuern zusammen. Die Berechnung von Verzweigungsmöglichkeiten wie im Zustandsraum wird durch diese Tatsache sehr kompliziert. Daher wurde auf eine genauere Untersuchung dieser Methode für Petri-Netze zugunsten der Transformation verzichtet.
Im Zustandsraum ließ sich eine Sequenz relativ einfach durch eine Faltung verkürzen. In [Lub01] wurde daher versucht, dieses Prinzip auf eine Sequenz im Petri-Netz zu übertragen. Dabei ergab sich folgendes Problem: Eine Single-Server Sequenz besteht beispielsweise wie in Abbildung 2.14 aus zwei Single-Server Transitionen T_1 und T_2, alle Kanten sind Einfachkanten, d.h. sie haben eine Vielfachheit von eins. Ein Token wird in der Stelle P_1 erzeugt und $t_1 = 1$ berechnet. Nach einer Zeiteinheit feuert T_1, T_2 bekommt eine

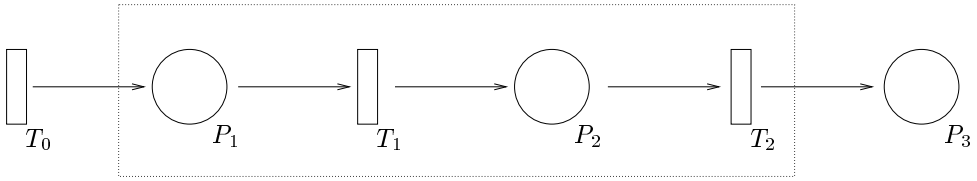


Abbildung 2.14: Eine Single-Server Sequenz

Laufzeit von $t_2 = 2$. Angenommen ein zweiter Token wird in P_1 zwei Zeiteinheiten nach dem ersten generiert. Zu diesem Zeitpunkt hat T_1 keine Laufzeit und T_2 eine Restlaufzeit von $t_2 = 1$. Demzufolge kann sofort eine neue Laufzeit für T_1 bestimmt werden. Wären durch Faltung T_1 und T_2 zu einer neuen Single-Server Transition T_3 umgeformt worden, hätte diese noch eine Laufzeit. Demzufolge würde erst eine neue Laufzeit für T_3 nach dem Feuern von T_3 berechnet werden. Dies würde aber im originalen Petri-Netz bedeuten, dass T_1 erst eine Laufzeit bekäme, wenn T_2 feuert. Ein solches Verhalten ist aber nicht korrekt. Eine denkbare Alternative wäre es, statt der Single-Server Transition einen Infinite-Server T_3 zu verwenden. Hier ergibt sich aber ein genau umgekehrtes Problem. Wird ein Token in P_1 erzeugt und T_1 hat noch eine Laufzeit, wird erst nach dem Feuern von T_1 eine neue berechnet. Bei dem Infinite-Server hingegen würde sofort nach Erzeugen eine neue Laufzeit bestimmt, was wiederum nicht dem Verhalten des Petri-Netzes entspräche. Demzufolge lassen sich Single-Server Figuren nicht mittels Faltung ersetzen. Gleiches gilt für Multi-Server Topologien [Lub01].
Keine Probleme dagegen ergeben sich bei der Zusammenfassung einer Infinite-Server Sequenz. Die neue Infinite-Server Transition verhält sich exakt wie die Sequenz. Die Verteilungsfunktion der neuen Transition entsteht dabei analog zum Zustandsraum durch Faltung. Demzufolge lassen sich die Methoden aus 2.2.2 auf die entsprechenden Figuren im Petri-Netz anwenden, wenn diese ausschließlich aus Infinite-Servern bestehen [Lub01].

Kapitel 3

Adaptive Simulation

Die Anwendung der Beschleunigungsmechanismen aus dem Zustandsraum auf Petri-Netze verursacht bei nicht Infinite-Server Figuren das Problem, dass sich die umgeformten Topologien während der Simulation nicht korrekt verhalten. Die Ursache des Problems liegt in der Tatsache, dass im Petri-Netz verschiedene Token teilweise unterschiedliche Übergänge ermöglichen, welche sich gegenseitig beeinflussen können. Im Zustandsraum existiert diese Schwierigkeit nicht, da es nur einen aktiven Zustand gibt, von dem aus Übergänge möglich sind. Dies ergibt die Idee, in den umzuformenden Figuren gleiche Bedingungen wie im Zustandsraum sicherzustellen, was, wie in 3.1 gezeigt wird, zu einem immer wiederkehrenden Wechsel zwischen originalen und transformierten Topologien führt. Durch die in 3.2 vorgestellte Erweiterung der Bedingung, wann ein solcher Wechsel stattfinden muss, kann in 3.3 diese Methode des Wechselns auch für eine Hierarchie von Figuren angewendet werden. Die Frage der Überprüfung dieser Bedingung wird in 3.4 behandelt, ebenso wie das Erzeugen einer Hierarchie, in der bereits im Initial-Zustand Token enthalten sind. In 3.5 wird schließlich das Potential zur Zeitersparnis bei dieser Strategie untersucht.

3.1 Konzept, Idee

In [Lub01] wurde gezeigt, dass umgeformte Infinite-Server Topologien ein korrektes Verhalten zeigen. In Petri-Netzen kommen solche Figuren jedoch recht selten vor, weshalb in diesem Abschnitt ein Weg gesucht wird, auch andere Figuren zu transformieren. Im Folgenden wird von Petri-Netzen bzw. Topologien ausgegangen, die lediglich zeitbehaftete Transitionen aber nicht ausschließlich Infinite-Server enthalten.
Die Probleme bei der Umformung von Petri-Netz-Figuren entstehen unter anderem, wenn eine Transition mehrmals oder wenn mehrere voneinander unabhängige Transitionen aktiviert sind. Dies entspricht mehreren sich behindernden bzw. parallelen, unabhängigen Vorgängen. In dem Zustandsraum wird ein Ereignis durch einen Zustandsübergang repräsentiert. Es sind nur die Übergänge erlaubt, die vom aktuellen Zustand aus möglich sind. Diese sind aber weder unabhängig voneinander noch behindern sie sich gegenseitig. Durch die Sicherstellung analoger Verhältnisse in den umzuformenden Figuren des Petri-Netzes, müssten diese sich nach der Transformation korrekt verhalten. Der aktive Zustand eines Zustandsraumes könnte auch durch einen Token dargestellt werden, der bei einem Zustandswechsel in einen anderen Zustand ,,geschoben" wird. Auf das Petri-Netz angewendet, dürfen damit alle Kanten in einer Figur nur die Vielfachheit $m = 1$ und alle

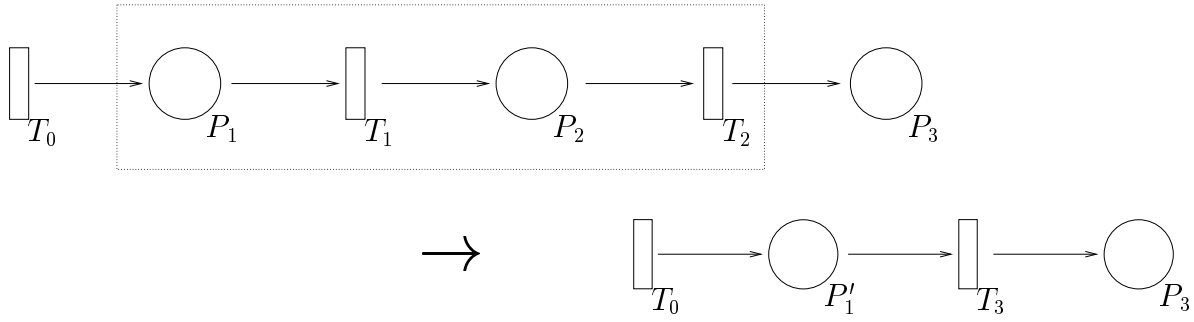


Abbildung 3.1: Verkürzung einer Beispiel-Sequenz

Transitionen lediglich eine Eingangs- und Ausgangskante besitzen. Diese Einschränkungen beziehen sich nur auf die Topologie, sind demzufolge einfach zu überprüfen. Ebenso dürfen die in 2.3 gemachten Erweiterungen nicht in der Figur enthalten sein, da es für diese keine Analogie im Zustandsraum gibt. Ausgenommen sind hiervon die Vielfachheiten der Transitionen, da jede, die sich in einer Zustandsraum analogen Figur befindet, sowieso nur maximal einmal aktiviert sein kann. Die Tatsache, dass es im Zustandsgraphen nur einen aktiven Zustand gibt, lässt sich durch die Bedingung, dass maximal ein Token in einer Figur enthalten sein darf, auf Petri-Netze transferieren. Die entscheidende Frage ist, wie stellt man diese Voraussetzung sicher? Eine Möglichkeit wäre, vor der Simulation durch eine Analyse des Petri-Netzes alle Figuren herauszufinden, in denen während der Simulation maximal ein Token enthalten ist. Diese Untersuchung ist nicht trivial und wird mit zunehmender Komplexität des Petri-Netzes schwerer. Im Gegensatz dazu ist es einfach, während der Simulation zu überprüfen, wie viele Token aktuell in einer Figur enthalten sind. Damit ergibt sich folgende Idee [Lub01]: Ist in einer Topologie maximal ein Token enthalten, so kann die transformierte Figur verwendet werden. Befindet sich aber durch das Feuern einer Transition auf einmal mehr als ein Token in der Figur, muss auf die originale Topologie umgeschaltet werden. Da sich die Anzahl an Token im Laufe der Simulation dauernd ändern kann, führt dies zu einem immer wiederkehrenden Wechsel zwischen originaler und umgeformter Figur. Im folgendem Abschnitt wird diese Strategie und die damit verbundenen Probleme an dem Beispiel einer Single-Server Sequenz behandelt.

Die Beispiel Sequenz besteht wie in Abbildung 3.1 zu sehen ist, aus zwei Single-Server Transitionen T_1 und T_2. Entsprechend den Einschränkungen haben beide Transitionen jeweils exakt eine Eingangs- und eine Ausgangskante, alle Kanten haben die Vielfachheit eins. Damit eine korrekte Sequenz gegeben ist, hat die mittlere Stelle P_2 die Bedingung, mit exakt einer eingehenden und einer ausgehenden Kante verbunden zu sein. Ebenfalls darf von der ersten Stelle P_1 nur eine Kante weggehen. Diese Vorraussetzungen sind in der Beispiel-Sequenz alle erfüllt. Damit überhaupt umgeformt werden darf, müssen analog zum Zustandsraum die Stellen P_1, P_2 und die Transitionen T_1 und T_2 als uninteressant markiert sein, d.h. es braucht keine Statistik über sie berechnet zu werden. Bei der Umformung der Sequenz wird eine neue Single-Server Transition T_3 eingefügt mit einer Eingangskante von P_1 und einer Ausgangskante zu P_3. Abbildung 3.1 zeigt das Ergebnis der Transformation. Analog zum Zustandsgraphen ändert sich die mittlere Anzahl an To-

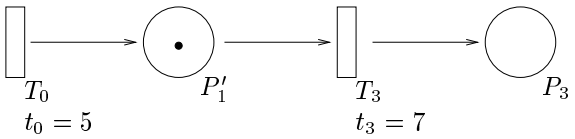


Abbildung 3.2: Aktivierte verkürzte Sequenz, $\tau = 10$

ken in P_1, deshalb wird diese Stelle als P_1' bezeichnet. Die Statistik von P_3 bleibt gleich, da sich die neue Transition T_3 genauso wie die ursprüngliche Sequenz verhalten muss.

Im Zustandsraum wurde die Zeitdauer für den neuen Übergang durch eine neue Wahrscheinlichkeitsverteilung beschrieben, die aus der Faltung der beiden Verteilungen der ersetzten Kanten entstand. In der umgeformten Beispiel-Sequenz kann es allerdings passieren, dass noch während die Transition T_3 eine Laufzeit hat, ein zweiter Token in der Stelle P_1 erzeugt wird. Tritt dieser Fall ein, muss auf die originale Sequenz geschaltet werden. Dabei müssen aus der Restlaufzeit t_3 die Restlaufzeiten t_1 und t_2 berechnet werden können. Dies ist aber nicht möglich, wenn t_3 zufällig anhand einer eigenen, durch Faltung entstandenen Verteilungsfunktion, erzeugt wurde. Stattdessen wird die Berechnung der Laufzeit von T_3 auf die Laufzeiten von T_1 und T_2 zurückgeführt, sie entspricht einfach der Addition der beiden. In einer Beispiel-Simulation wird dies ebenso wie ein Schaltvorgang anschaulich.

Vor der Beispiel-Sequenz befindet sich eine Single-Server Transition T_0. Diese hat keine Eingangskante, ist demzufolge immer aktiviert. Sie hat die Laufzeit $t_0 = 10$ und ist als einzige aktiviert, feuert demzufolge zum Zeitpunkt $\tau = 10$. Damit sind nun T_0 und T_3 aktiviert, die Laufzeit von T_0 sei $t_0 = 5$. Die Laufzeit von T_3 wird auf die Addition der Laufzeiten von T_1 und T_2 zurückgeführt, diese seien $t_1 = 3$ und $t_2 = 4$. Für T_3 bedeutet dies, dass $t_3 = t_1 + t_2 = 7$. Diese Situation zeigt die Abbildung 3.2.

Die derzeit kürzeste Laufzeit ist die von T_0 mit $t_0 = 5$. Damit feuert T_0 zum Zeitpunkt $\tau = 15$ und es befinden sich zwei Token in der Stelle P_1'. Diese Stelle gehört aber zu dem Bereich der umgeformten Sequenz, in der sich maximal ein Token aufhalten darf. Damit muss wieder auf die originale Sequenz geschaltet werden. Die Restlaufzeit von T_3 zu diesem Zeitpunkt beträgt $t_3 = 2$, die ursprüngliche, totale Laufzeit betrug 7, demzufolge sind bereits 5 ,,abgelaufen". Die in t_3 enthaltene totale Laufzeit von T_1 betrug 3, T_1 hätte also zu diesem Zeitpunkt bereits gefeuert. Damit befindet sich ein Token eigentlich bereits in P_2. Beim Schalten wird nun T_3 wieder aus der Figur entfernt und die ursprünglichen Transitionen T_1 und T_2 werden eingefügt. Ein Token wird aus P_1 entfernt und einer in P_2 erzeugt. Die Restlaufzeit von T_2 ergibt sich aus der Restlaufzeit von T_3: $t_2 = t_3 = 2$. Die Transition T_1 hatte in der umgeformten Figur quasi schon gefeuert, ihre alte, totale Laufzeit kann damit verworfen werden. Es befindet sich immer noch ein Token in P_1, demzufolge ist T_1 erneut aktiviert, es wird eine neue Laufzeit von $t_1 = 4$ erzeugt. Schließlich ist auch noch T_0 aktiviert, sie erhält eine neue Laufzeit von $t_0 = 50$. Abbildung 3.3 zeigt die Sequenz nach dem Schalten.

Zu diesem Zeitpunkt ist t_2 die kürzeste Laufzeit, T_2 feuert demzufolge als nächstes bei einer Gesamtzeit von $\tau = 17$. Nach diesem Feuern befindet sich nur noch ein Token in der Sequenz, da die Stelle P_3 nicht dazu gerechnet wird. Es kann also wieder auf die transformierte Sequenz geschaltet werden. Damit wird T_3 wieder eingefügt und T_1 und

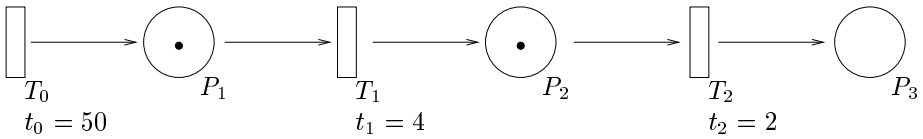


Abbildung 3.3: Die Sequenz nach dem Zurückschalten, $\tau = 15$

T_2 werden entfernt. Der Token in P_1 bleibt, in P_2 befindet sich keiner. Die neue Laufzeit t_3 wird aus der Addition der Restlaufzeit t_1 und einer neu erzeugten Laufzeit t_2 berechnet.

In dem Beispiel ist erkennbar, dass diese Strategie zu immer wiederkehrenden Wechseln zwischen originaler und transformierter Figur führt. Das Schalten wird dabei dynamisch je nach Zustand der Figur vorgenommen, weshalb diese Vorgehensweise im Folgenden auch als *adaptive Simulation* bezeichnet wird. In [Lub01] wurde gezeigt, dass sich damit die in 2.2.2 vorgestellten Figuren transformieren lassen.

3.2 Erweiterung

Ein Vorteil von Petri-Netzen gegenüber dem Zustandsraum ist z. B. die Möglichkeit, die Synchronisation von mehreren Abläufen wesentlich einfacher darzustellen, was durch eine Transition mit mehreren Eingangskanten realisiert werden kann. Es wäre demzufolge wünschenswert, die adaptive Simulation auch auf Figuren anzuwenden, die solche Transitionen enthalten. Bevor jedoch genauer darauf eingegangen wird, werden der Einfachheit halber ein paar neue Begriffe und Annahmen eingeführt. In dieser Arbeit wird stets von Figuren ausgegangen, die ausschließlich zeitbehaftete Transitionen enthalten. Analog zum Zustandsraum, bei dem eine Topologie transformiert durch Kanten dargestellt wurde, wird angenommen, dass die zu vereinfachenden Petri-Netz-Figuren nach der Umformung nur durch zeitbehaftete Transitionen und deren Eingangs- und Ausgangskanten dargestellt werden. Bei der Umformung fallen demzufolge alle enthaltenen Stellen weg. Die Transitionen, die die originale Figur repräsentieren, werden im folgenden auch *Ersatz-Transitionen* genannt (sie ,,ersetzen" die originale Topologie). Die originale Figur selbst wird als *Sub-Netz* bezeichnet. Alle Eingangsstellen der Ersatz-Transitionen zusammen bilden die Eingangsstellen des Sub-Netzes, gleiches gilt für die Ausgangsstellen. Desweiteren wird im Folgenden das Schalten von den Ersatz-Transitionen auf das Sub-Netz als *Herunterschalten*, der Wechsel von dem Sub-Netz auf die Ersatz-Transitionen als *Hochschalten* bezeichnet. Zur Unterscheidung, ob gerade die originale oder die verkürzte Figur verwendet wird, wird der Term *benutzt* eingeführt: Nach dem Herunterschalten wird das Sub-Netz benutzt, nach dem Hochschalten die Ersatz-Transitionen.
Die Bedingung, dass heruntergeschaltet werden muss, wenn sich mehr als ein Token in der umgeformten Topologie befindet, kann mit den so vorgenommenen Annahmen folgendermaßen umformuliert werden: Es muss heruntergeschaltet werden, wenn sich mehr als ein Token in den Eingangsstellen der Ersatz-Transitionen befindet. Diese Bedingung wird im Folgenden auch als *Nutzungs-Bedingung* bezeichnet (sie besagt, ob die Ersatz-Transitionen weiterhin genutzt werden können).

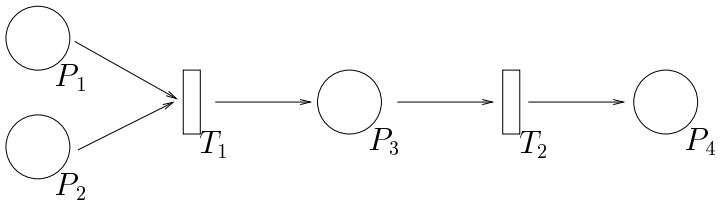


Abbildung 3.4: Eine Sequenz mit zwei Eingangsstellen

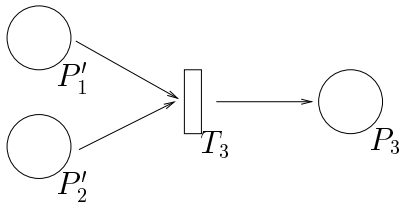


Abbildung 3.5: Die verkürzte Sequenz mit zwei Eingangsstellen

In einer weiteren Beispiel-Sequenz hat die erste Transition zwei Eingangsstellen, wie in Abbildung 3.4 zu sehen ist. Die Idee der adaptiven Simulation lässt sich auch für diese Sequenz anwenden: Die Transitionen T_1 und T_2 werden durch eine neue Transition T_3 ersetzt, die wiederum zwei Eingangskanten hat, je eine von P'_1 und P'_2. Abbildung 3.5 zeigt das Ergebnis der Umformung. Die Laufzeit von T_3 wird wie vorher durch Addition der originalen Laufzeiten bestimmt: $t_3 = t_1 + t_2$. Allerdings ist in dieser Situation die Aussage, dass heruntergeschaltet werden muss, wenn sich mehr als ein Token in den Eingangsstellen von T_3 befindet, nicht sinnvoll und auch nicht notwendig. Ist kein Token in der Stelle P'_1, so können beliebig viele Token in der Stelle P'_2 sein, ohne dass heruntergeschaltet werden muss. Die Transition T_3 ist in diesem Falle nicht aktiviert, ebenso wären in der originalen Figur weder T_1 noch T_2 bei dieser Tokenverteilung aktiviert, weshalb T_3 für diese Situation das Verhalten korrekt nachbildet. Heruntergeschaltet werden muss jedoch, wenn sowohl in P'_1 als auch in P'_2 mindestens zwei Token vorhanden sind. Dann ist der Aktivierungsgrad von T_3 mindestens zwei, T_3 hat aber nur eine Laufzeit, eine neue wird erst nach dem Feuern bestimmt. In der originalen Figur würde aber eine neue Laufzeit für T_1 bereits vor dem Feuern von T_2 erzeugt werden. In dieser Situation ist das Verhalten von T_3 deshalb nicht korrekt. Durch die eben vorgestellten Fälle wird klar, dass die Anzahl an Token in den Eingangsstellen der Ersatz-Transition kein geeignetes Kriterium zur Feststellung, ob heruntergeschaltet werden muss, ist. Einmal sind beliebig viele, also z. B. auch 50 Token unkritisch, ein anderes mal muss bereits bei vier Token geschaltet werden. Ebenfalls im Beispiel erkennbar ist, dass eigentlich der Aktivierungsgrad von T_3 darüber entscheidet, ob T_3 sich korrekt verhält. Ist T_3 einmal oder weniger aktiviert, so wäre auch in der ursprünglichen Sequenz nur eine der beiden Transitionen aktiviert. Ist der Aktivierungsgrad von T_3 zwei oder höher, so könnten in der originalen Sequenz auch beide Transitionen gleichzeitig aktiviert sein und das Verhalten von T_3 entspricht nicht dem der originalen Figur.

Aufgrund dieser Erkenntnis lautet die Nutzungs-Bedingung für die Ersatz-Transitionen nun folgendermaßen: Es muss auf das Sub-Netz geschaltet werden, wenn eine Ersatz-Tran-

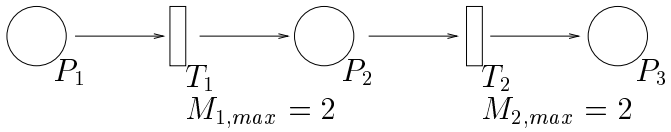


Abbildung 3.6: Eine Sequenz aus Multi-Servern

sition einen höheren Aktivierungsgrad als eins besitzt. Damit können auch Figuren ersetzt werden, die Transitionen enthalten, die mit mehr als einer Eingangsstelle des Sub-Netzes verbunden sind. Die bisherige Einschränkung der Vielfachheit eins der Kanten kann damit ebenfalls fallen gelassen werden. Allerdings kann durch ein Feuern einer Ersatz-Transition T_f lediglich ein einmaliges Feuern der durch T_f repräsentierten Folge von Transitionen aus dem Sub-Netz imitiert werden. Wenn die Kantenvielfachheiten in dieser Folge aber so gewählt sind, dass das Feuern einer der Transitionen bewirkt, dass die letzte Transition nicht exakt einmal feuert, dann kann dieses Verhalten von einer Ersatz-Transition nicht korrekt nachgebildet werden. Damit ergibt sich wieder eine Einschränkung der erlaubten Vielfachheiten für die Kanten. Wie genau diese allerdings aussieht, hängt immer von der jeweiligen zu ersetzenden Figur ab. In Kapitel 4 wird unter anderem auf die spezifischen Bedingungen für die Kantenvielfachheiten einzelner Topologien eingegangen.
Eine neue Sequenz besteht wie in Abbildung 3.6 aus zwei Multi-Server Transitionen T_1 und T_2. Angenommen, die entsprechende Ersatz-Transition T_3 ist ein Infinite-Server. In der Eingangsstelle P_1 werden nun zwei Token erzeugt. Demzufolge ist der Aktivierungsgrad von T_3 gleich zwei, nach der bisherigen Nutzungs-Bedingung muss heruntergeschaltet werden. Die originalen Transitionen haben allerdings eine Vielfachheit von zwei, weshalb T_1 sofort zwei Laufzeiten erzeugt. Nach einem Feuern von T_1 bekommt T_2 sofort eine neue Laufzeit, da auch T_2 eine Vielfachheit von zwei hat und der maximal mögliche Aktivierungsgrad in dieser Token-Konstellation ebenfalls zwei ist. Damit beeinflussen sich die beiden Laufzeiten von T_1 bzw. T_2 nicht gegenseitig, solange der insgesamte Aktivierungsgrad von T_1 und T_2 kleiner gleich zwei bleibt. Die Ersatz-Transition T_3 ist ein Infinite-Server, demzufolge wird das Verhalten des Sub-Netzes bis zu einem Aktivierungsgrad von zwei korrekt nachgebildet.
An diesem Beispiel ist deutlich zu sehen, dass die bisherige Grenze von einem Aktivierungsgrad eins nicht notwendig ist, solange alle Ersatz-Transitionen Infinite-Server sind. Stattdessen wird die Grenze von der jeweiligen Figur und den Vielfachheiten der enthaltenen Transitionen bestimmt. Ist der Aktivierungsgrad einer Ersatz-Transition höher als dieser Grenzwert, so muss heruntergeschaltet werden. Damit ist es aber auch nicht notwendig, dass die Ersatz-Transitionen Infinite-Server sind, solange ihre Vielfachheit größer gleich dem Grenzwert ist. Eine Vielfachheit größer als der Grenzwert ist ebenfalls nicht erforderlich. Demzufolge wird die Vielfachheit einer Ersatz-Transition genau gleich dem Grenzwert des Aktivierungsgrades gesetzt, bis zu dem sie das Verhalten des Sub-Netzes korrekt nachbildet. Die Nutzungs-Bedingung für die Ersatz-Transitionen lässt sich dann in folgender Weise formulieren: Es muss heruntergeschaltet werden, sobald der Aktivierungsgrad einer Ersatz-Transition höher als ihre Vielfachheit ist.
Durch die Formulierung der Nutzungs-Bedingung in Abhängigkeit von dem Aktivierungsgrad einer Ersatz-Transition ergibt sich ein bisher unbetrachtetes Problem bei Figuren, die durch mehrere Ersatz-Transitionen dargestellt werden. Eine solche Topologie ist ei-

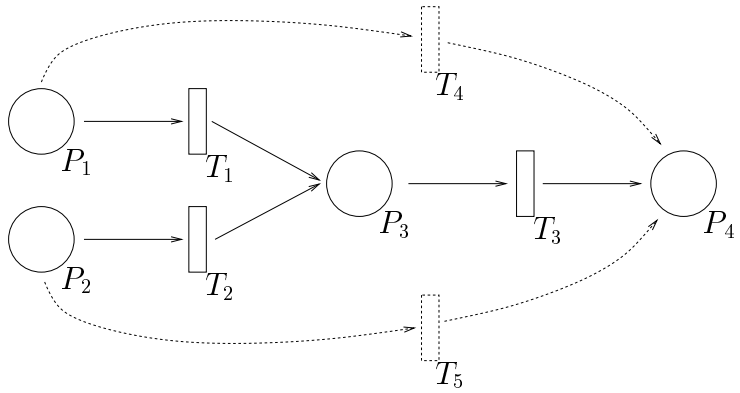


Abbildung 3.7: Eine Merge-Figur mit angedeuteten Ersatz-Transitionen

ne Merge-Figur, wie sie in Abbildung 3.7 zu sehen ist. Die Ersatz-Transition T_4 steht für die Transitionen T_1 und T_3 und die Ersatz-Transition T_5 für T_2 und T_3, alle originalen Transitionen sind Single-Server. Da sich eine Transformation nicht lohnt, wenn eine Ersatz-Transition eine Vielfachheit von null hat, haben beide Ersatz-Transitionen mindestens eine Vielfachheit von eins. Aus diesem Grund sind T_4 und T_5 Single-Server Transitionen. Befindet sich jeweils ein Token in den Eingangsstellen P_1 und P_2, haben beide Ersatz-Transitionen demnach den Aktivierungsgrad eins. Die jeweilige Laufzeit wird aus den Laufzeiten der originalen Transitionen berechnet. Allerdings repräsentieren beide Ersatz-Transitionen die Transition T_3. T_3 müsste demzufolge zwei verschiedene Laufzeiten gleichzeitig haben. T_3 ist aber ein Single-Server, wodurch die Ersatz-Transitionen das Verhalten des Sub-Netzes nicht korrekt nachbilden. Es muss heruntergeschaltet werden, obwohl beide Ersatz-Transitionen einen Aktivierungsgrad kleiner gleich ihrer Vielfachheit haben.
Repräsentieren mehrere Ersatz-Transitionen ein Sub-Netz, so spielt auch die Summe aller Aktivierungsgrade eine Rolle. Die bisherige Nutzungs-Bedingung ist demzufolge nicht ausreichend, sie muss erweitert werden. Es ergibt sich folgende neue Nutzungs-Bedingung:

Von den Ersatz-Transitionen muss auf das Sub-Netz geschaltet werden, wenn

1. der Aktivierungsgrad einer Ersatz-Transition höher als ihre Vielfachheit ist (*lokale Nutzungs-Bedingung*)

oder

2. die Summe aller Aktivierungsgrade der Ersatz-Transitionen zusammen höher als ein Grenzwert B ist (*globale Nutzungs-Bedingung*).

Die Höhe des Grenzwertes B hängt wie die Vielfachheiten der Ersatz-Transitionen von der Topologie selbst und den Vielfachheiten der originalen Transitionen ab. Der Einfachheit halber wird die erste Teilbedingung im Folgenden als *lokale Nutzungs-Bedingung* und die zweite als *globale Nutzungs-Bedingung* bezeichnet.
Ist der Grenzwert B gleich oder höher der Summe aller Vielfachheiten der Ersatz-Transitionen, so kann er ignoriert werden: Ein Überschreiten dieses Wertes hätte logischerweise

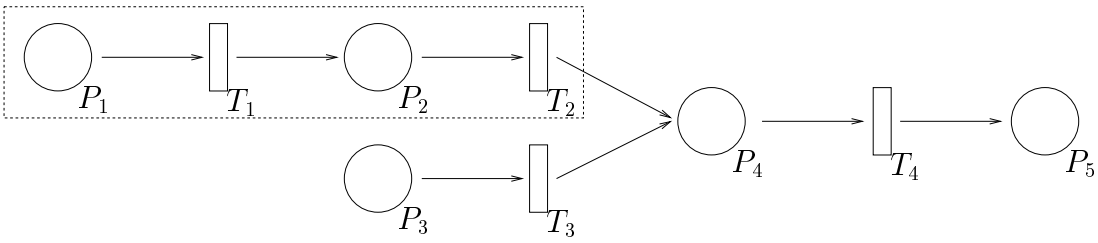


Abbildung 3.8: Eine Merge-Figur mit enthaltener Sequenz

auch immer ein Überschreiten der ersten Teilbedingung zur Folge. Damit lässt sich die Menge aller ersetzbaren Figuren in zwei Untermengen aufteilen: in Figuren mit ausschließlich lokaler und in Figuren mit lokaler und globaler Nutzungs-Bedingung.
In der adaptiven Simulation werden die Laufzeiten der Ersatz-Transitionen aus den Laufzeiten der Transitionen des Sub-Netzes berechnet. Die Einschränkung, dass nur Transitionen mit race-enable Strategie im Sub-Netz vorkommen dürfen, kann demnach fallengelassen werden. Hat eine Transition eine andere policy, so wird dies bereits in der Laufzeit berücksichtigt, die zur gesamten Laufzeit der Ersatz-Transitionen beiträgt. Eventuelle Einschränkungen in Bezug auf die Anzahl an Aktivierungsgraden, für die sich die Ersatz-Transitionen korrekt verhalten, lassen sich durch die bereits eingeführten Vielfachheiten der Ersatz-Transitionen sowie den Grenzwert B realisieren.

In der ursprünglichen Idee der adaptiven Simulation konnte von dem Sub-Netz wieder auf die Ersatz-Transitionen geschaltet werden, sobald sich maximal ein Token in der Figur befand. Diese Bedingung ist zwar weiterhin korrekt, aber durch die Erweiterung der Nutzungs-Bedingung kann auch diese verändert werden. Beim Hochschalten werden die Token aus dem Sub-Netz entfernt und entsprechend viele in den Eingangsstellen der Ersatz-Transitionen erzeugt. Die daraus resultierenden Aktivierungsgrade der Ersatz-Transitionen müssen der Nutzungs-Bedingung entsprechen, da ansonsten gleich wieder heruntergeschaltet werden müsste. Die Bedingung, die besagt, ob von dem Sub-Netz hochgeschaltet werden darf, muss demzufolge überprüfen, ob nach einem eventuellen Hochschalten die Nutzungs-Bedingung eine Benutzung der Ersatz-Transitionen erlaubt. Diese Bedingung wird im Folgenden auch als *Ersetzungs-Bedingung* bezeichnet (sie besagt, ob das Sub-Netz in dem derzeitigen Zustand ersetzt werden kann).

3.3 Hierarchische Zusammenhänge

Ein weiteres Problem ergibt sich, wenn die verschiedenen Figuren ineinander geschachtelt vorkommen. Abbildung 3.8 zeigt hierfür ein Beispiel: In einem Merge kommt eine Sequenz vor. Im Zustandsraum war dies kein Problem, da eine umgeformte Figur einfach wieder neu transformiert werden konnte. Bei der adaptiven Simulation für Petri-Netze entsprechen die Ersatz-Transitionen aber nicht ,,normalen" Transitionen, es muss auf die Korrektheit ihres Verhaltens geachtet werden. Allerdings lässt sich auch für ,,normale" Transitionen eine Nutzungs-Bedingung definieren, die besagt, dass nicht geschaltet werden muss. Diese ist eine ausschließlich lokale Nutzungs-Bedingung. Betrachtet man das Sub-Netz von den

Ersatz-Transitionen aus, so besitzen alle darin enthaltenen Transitionen eine Nutzungs-Bedingung. Damit ist es auch möglich, dass ein Sub-Netz andere Ersatz-Transitionen enthält, da sie nicht mehr von den ,,normalen" Transitionen unterscheidbar sind. So kann eine Hierarchie von Sub-Netzen entstehen.
Die Nutzungs-Bedingung von Ersatz-Transitionen muss dieser veränderten Situation entsprechend angepasst werden. Im Folgenden wird die bisherige Nutzungs-Bedingung, wie sie in 3.2 entwickelt wurde, als *direkte Nutzungs-Bedingung* bezeichnet, da sie sich direkt auf das korrekte Verhalten der Ersatz-Transitionen ohne Betrachtung der Hierarchie bezieht. Die neue, *hierarchische Nutzungs-Bedingung* von Ersatz-Transitionen eines Sub-Netzes lautet dann folgendermaßen: Es muss auf das Sub-Netz heruntergeschaltet werden, wenn die direkte Nutzungs-Bedingung dies erfodert oder wenn die hierarchische Nutzungs-Bedingung einer der Transition im Sub-Netz ein Schalten auf deren Sub-Netz erforderlich macht.
Es stellt sich die Frage, ob die Überprüfung der Nutzungs-Bedingungen der Transitionen des Sub-Netzes immer ein korrektes Ergebnis liefert. In den folgenden Überlegungen wird von einem Sub-Netz N_k ausgegangen, das durch die Ersatz-Transitionen T_r ersetzt ist. Die Ersatz-Transitionen T_r werden benutzt und ihre Nutzungs-Bedingung soll überprüft werden. Bei der Frage der Korrektheit der hierarchischen Nutzungs-Bedingung der Sub-Netz-Transitionen ist nur die Abfrage der direkten Nutzungs-Bedingung von Bedeutung, da die absteigende Überprüfung der hierarchischen Bedingung genau wieder das Problem ist.
Die direkte Nutzungs-Bedingung einer Transition T_s des Sub-Netzes N_k besteht aus der lokalen und der globalen Nutzungs-Bedingung. Die lokale vergleicht den Aktivierungsgrad der Transition mit ihrer Vielfachheit. Diese Bedingung lässt sich in die Nutzungs-Bedingung der Ersatz-Transitionen T_r einbauen: Die Vielfachheiten der Ersatz-Transitionen T_r und der Grenzwert B werden so gewählt, dass Folgendes gilt: Immer wenn die Anzahl an Laufzeiten einer Sub-Netz-Transition T_i größer würde als ihre Vielfachheit $M_{i,max}$, erzwingt die Nutzungs-Bedingung von T_r ein Schalten auf das Sub-Netz N_k. Genau dieses Verhalten ist aber bereits durch die Wahl der Vielfachheit von T_r und die bisherige Definition der direkten Nutzungs-Bedingung sichergestellt. Damit erübrigt sich eine extra Überprüfung der lokalen Nutzungs-Bedingungen der Sub-Netz-Transitionen.
Im Gegensatz zur lokalen überprüft die globale Nutzungs-Bedingung die Summe aller Aktivierungsgrade der Ersatz-Transitionen eines Sub-Netzes. In dem Sub-Netz N_k befinden sich die Transitionen T_1 und T_2, die zusammen wieder ein darunter liegendes Sub-Netz repräsentieren. Bei der Überprüfung der globalen Nutzungs-Bedingung von T_1 muss diese den Aktivierungsgrad von T_1 und T_2 kennen. Beide befinden sich aber mitten in N_k, ihre Eingangsstellen sind keine Eingangsstellen von N_k und enthalten demnach keine Token. Damit kann der Aktivierungsgrad nicht anhand der Tokenzahl bestimmt werden. Die Anzahl an Laufzeiten von T_1 und T_2 gibt nur den alten, nicht aber den neuen Aktivierungsgrad an, da das Herunterschalten stets vor der Berechnung der Laufzeiten erfolgen muss. Ansonsten kann durch die verschiedenen race policies ein falsches Verhalten entstehen (z. B. beim Split). Es ist also nicht möglich, den neuen Aktivierungsgrad von T_1 und T_2 festzustellen.
Diese Erkenntnis führt zu folgender Einschränkung: Es dürfen nur Figuren ersetzt werden, bei denen alle Transitionen, die eine globale Nutzungs-Bedingung haben, *Anfangs-Transitionen* sind. Anfangs-Transitionen sind dabei die Transitionen, deren gesamten

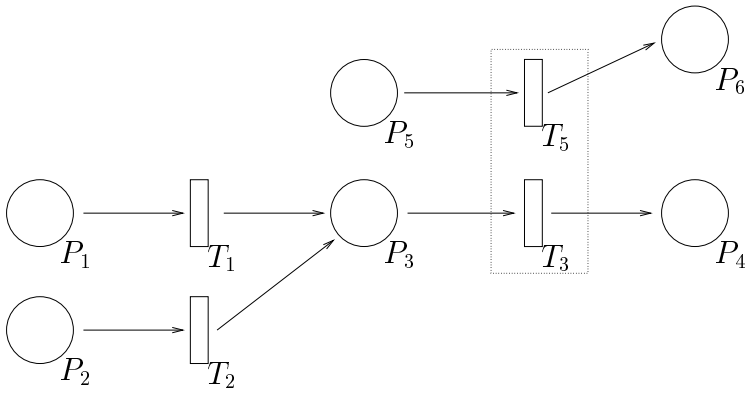


Abbildung 3.9: Die Ersetzung dieser Merge-Figur ist verboten

Eingangsstellen ebenfalls Eingangsstellen des Sub-Netzes sind. Werden die Ersatz-Transitionen der Figur benutzt, so beinhalten die Eingangsstellen der Topologie alle Token. Demzufolge lässt sich der aktuelle Aktivierungsgrad der Anfangs-Transitionen anhand der Tokenzahl bestimmen, die globalen Nutzungs-Bedingungen liefern ein korrektes Ergebnis.
Ein weiteres Problem ergibt sich, wenn wie in Abbildung 3.9, eine Transition einer Figur mit einer Transition außerhalb der Figur ein Sub-Netz darstellt. In dem Fall, der in der Abbildung zu sehen ist, kann Folgendes passieren: Die Transition T_3 besitzt ausschließlich eine lokale Nutzungs-Bedingung. Wird nun der Merge ersetzt, so ist die Überprüfung dieser Bedingung automatisch in der Nutzungs-Bedingung der Ersatz-Transitionen des Merges enthalten. Allerdings kann die Nutzungs-Bedingung der Transition T_5 ein Herunterschalten auf das Sub-Netz von T_3 und T_5 notwendig machen, auch wenn die lokale Nutzungs-Bedingung von T_3 kein Schalten verursachet. Dieser Fall kann demzufolge nicht von der direkten Nutzungs-Bedingung der Ersatz-Transitionen des Merges abgefangen werden. Damit müsste die hierarchische Nutzungs-Bedingung die Nutzungs-Bedingung von T_3 abfragen, die den Aktivierungsgrad von T_5 überprüft. Problematisch wird das Ganze, wenn auch noch T_5 ersetzt ist. Wird nun die Nutzungs-Bedingung des Merges vor der Nutzungs-Bedingung der Ersatz-Transitionen von T_5 abgefragt, so ist T_5 zu diesem Zeitpunkt noch immer ersetzt. Damit lässt sich aber der Aktiverungsgrad von T_5 im Allgemeinen nicht ermitteln. Die Nutzungs-Bedingung des Merges müsste also ein weiteres Mal nach dem Schalten auf T_5 überprüft werden. Dies bedeutet aber viel Aufwand, da nach jedem Schalten wieder die Nutzungs-Bedingungen aller benutzten Ersatz-Transitionen erneut kontrolliert werden müssen. Die adaptive Simulation ist allerdings zur Beschleunigung der Simulation gedacht. Deshalb ist im Folgenden die Ersetzung einer Figur, die nicht alle Ersatz-Transitionen eines Sub-Netzes enthält, verboten.
Mit diesen Einschränkungen lässt sich die hierarchische Nutzungs-Bedingung folgendermaßen formulieren: Es muss auf das Sub-Netz heruntergeschaltet werden, wenn die direkte Nutzungs-Bedingung dies erfodert oder die hierarchische Nutzungs-Bedingung einer der Anfangs-Transitionen T_a im Sub-Netz ein Schalten von T_a auf das Sub-Netz von T_a erforderlich macht.
In dieser Arbeit wird der Einfachheit halber noch eine Einschränkung gemacht. Besitzt eine Figur ausschließlich eine lokale Nutzungs-Bedingung , so dürfen alle enthaltenen Transitionen ausschließlich eine lokale Nutzungs-Bedingung haben. Damit lässt sich die

hierarchische Nutzungs-Bedingung für diese Topologien einfacher ausdrücken: Es muss auf das Sub-Netz heruntergeschaltet werden, wenn die direkte, lokale Nutzungs-Bedingung dies erfodert. Die Überprüfung dieser Bedingung ist relativ einfach und schnell.

Die bisherige Ersetzungs-Bedingung kontrolliert, ob nach einer eventuellen Ersetzung die Nutzungs-Bedingung ein Schalten veranlassen würde oder nicht. Bei der neuen, *hierarchischen Ersetzungs-Bedingung* kommt noch ein weiterer Punkt hinzu: Damit auf die Ersatz-Transitionen geschaltet werden kann, müssen alle Transitionen des Sub-Netzes benutzt werden und deren hierarchische Nutzungs-Bedingung darf kein Schalten verursachen.
Die Überprüfung der Nutzungs-Bedingungen der einzelnen Sub-Netz-Transitionen kann man einsparen, wenn immer zuerst alle notwendigen Herunterschaltungen vorgenommen werden, bevor die Möglichkeit eines Hochschaltens geprüft wird. In diesem Fall sind immer alle Nutzungs-Bedingungen der benutzten Transitionen erfüllt. Damit ist nur noch die Überprüfung der Benutzung aller enthaltenen Transitionen notwendig.

3.4 Umgang mit und Erzeugung von einer Hierarchie

In den bisherigen Abschnitten dieses Kapitels wurde gezeigt, dass eine Ersetzung von ineinander geschachtelten Figuren im Petri-Netz möglich ist. Desweiteren wurde untersucht, wann von den Ersatz-Transitionen heruntergeschaltet werden muss und wann wieder hochgeschaltet werden kann. Im Folgenden soll kurz auf die Frage eingegangen werden, ob nach jedem Feuern immer alle Nutzungs-Bedingungen und Ersetzungs-Bedingungen überprüft werden müssen. Anschließend wird gezeigt, wie man im Vorverarbeitungsschritt eine Hierarchie erzeugen kann, bei der bereits im Initial-Zustand Token enthalten sind.

Die Nutzungs-Bedingung gibt an, wann von den Ersatz-Transitionen auf das Sub-Netz geschaltet werden muss. Dabei spielt der Aktivierungsgrad der Ersatz-Transitionen die entscheidende Rolle. Dieser verändert sich, wenn sich die Anzahl an Token in den Eingangsstellen ändert. Demzufolge brauchen nach einem Feuern nur die Nutzungs-Bedingungen überprüft zu werden, die zu Ersatz-Transitionen gehören, in deren Eingangsstellen durch das Feuern neue Token erzeugt wurden.
Von einer Stelle P_1 gehen zwei Ersatz-Transitionen T_1 und T_2 weg, die verschiedene Sub-Netze repräsentieren. Beide Ersatz-Transitionen werden aktiviert. Die Laufzeiten der Ersatz-Transitionen stehen für eine Reihe von Laufzeiten in den Sub-Netzen und bestimmen außerdem, welche der beiden feuert. In dem Sub-Netz von T_1 geht die Transition T_{11} direkt von P_1 weg, in dem Sub-Netz von T_2 die Transition T_{22}. Der Unterschied der Laufzeiten von T_{11} und T_{22} stimmt im Allgemeinen nicht mit dem Unterschied der Laufzeiten von T_1 und T_2 überein. Damit ist aber auch die Wahrscheinlichkeit, dass T_1 vor T_2 feuert anders als die Wahrscheinlichkeit, dass T_{11} vor T_{22} feuert. Das Verhalten des originalen Petri-Netzes wird nicht korrekt nachgebildet. Ersatz-Transitionen, die von derselben Stelle weggehen, müssen deshalb ein und dasselbe Sub-Netz repräsentieren und haben demnach dieselbe Nutzungs-Bedingung.
Von einer Stelle, in der durch das Feuern einer Transition neue Token erzeugt wurden, gehen n Ersatz-Transitionen weg. Dank der obrigen Erkenntnis genügt es, die Nutzungs-Bedingung einer dieser n Ersatz-Transitionen zu überprüfen.

Nach dem Herunterschalten auf das Sub-Netz N_k werden die Token auf die Stellen des gesamten Sub-Netzes N_k verteilt. Es muss demzufolge eine erneute Überprüfung für alle Transitionen stattfinden, in deren Eingangsstellen Token erzeugt wurden. Dies gilt auch für die Anfangs-Transitionen von N_k, falls deren Eingangsstellen nach dem Schalten noch Token beinhalten: Im Normalfall wurden zumindest ein paar der Token aus den Eingangsstellen entfernt, so dass eine eventuelle vorherige Überprüfung der Nutzungs-Bedingung der Anfangs-Transitionen ein anderes Resultat ergeben haben kann.

Die Ersetzungs-Bedingung kontrolliert, ob von dem Sub-Netz auf die Ersatz-Transitionen geschaltet werden kann. Dabei werden die Token aus dem Sub-Netz entfernt und entsprechend viele in den Eingangsstellen des Sub-Netzes erzeugt. Wird ein Sub-Netz benutzt, so kann sich das Ergebnis der Überprüfung der Ersetzungs-Bedingung nur ändern, wenn mindestens ein Token aus dem Sub-Netz entfernt wurde. Damit muss die Ersetzungs-Bedingung nur von dem Sub-Netz überprüft werden, bei dem die gerade feuernde Transition eine *End-Transition* ist und das gerade benutzt wird. End-Transition ist eine Transition, wenn ihre Ausgangsstellen auch Ausgangsstellen des Sub-Netzes sind oder sie gar keine Ausgangskanten hat.
Wird ein Hochschalten veranlasst, so wird die feuernde Transition durch mindestens eine Ersatz-Transition T_r repräsentiert. Eine anschließende Kontrolle der Ersetzungs-Bedingung der Figur, zu der T_r gehört, ist aus dem gleichen Grund nur dann notwendig, wenn T_r von dieser Topologie eine End-Transition ist.

Bisher wurde bei allen Figuren angenommen, dass sie im Vorverarbeitungsschritt keine Token enthalten und damit die maximal mögliche Hierarchie erzeugt werden kann. Durch eine Aufspaltung des Vorverarbeitungsschrittes in zwei Teile, lässt sich auch eine Hierarchie von Figuren ersetzen, in der bereits im Initial-Zustand Token enthalten sind. Im ersten Teil wird einfach davon ausgegangen, dass keine Token im gesamten Petri-Netz existieren. Damit kann die maximal mögliche Hierarchie generiert werden. Anschließend werden alle Ersatz-Transitionen auf unbenutzt gesetzt und das originale Petri-Netz auf benutzt. Dies ist korrekt, da kein Token bisher beachtet wurde und damit keine Laufzeiten existieren. Im nun folgenden zweiten Schritt wird die Ersetzungs-Bedingung für alle Sub-Netze Schritt für Schritt überprüft und wenn möglich hochgeschaltet. Dies geschieht in der Reihenfolge, in der die Sub-Netze in der Hierarchie vorkommen: erst die Sub-Netze im originalen Petri-Netz, dann die darüber liegenden und so weiter. Am Ende dieses zweiten Teils befindet sich die Hierarchie in einem korrekten Zustand und die Simulation kann beginnen.

3.5 Untersuchung des Potentials zur Zeitersparnis bei adaptiver Simulation

Nachdem in den vorherigen Abschnitten deutlich wurde, dass die adaptive Simulation nicht nur für zustandsraumanaloge Petri-Netz-Figuren anwendbar ist, soll in diesem gezeigt werden, wo bei der adaptiven Simulation der Gewinn liegt, d.h. wo Rechenaufwand eingespart wird.

1. Suche nach den aktivierten zeitbehafteten Transitionen
2. Berechnung und Löschung von Laufzeiten wo nötig
3. Wahl der Transition, die als nächstes feuert
4. Erhöhen der Gesamtzeit τ
5. Feuern der Transition:
   (a) Token aus den Eingangsstellen entfernen
   (b) Token in den Ausgangsstellen erzeugen
   (c) Alle Laufzeiten ändern
   (d) Die entsprechende Laufzeit der Transition, die gefeuert hat, löschen

Abbildung 3.10: Die Teilschritte der Behandlung zeitbehafteter Transitionen

Bei der in dieser Arbeit untersuchten adaptiven Simulation kann eine Einsparung nur bei der Behandlung der zeitbehafteten Transitionen auftreten, da lediglich Figuren umgeformt werden, die keine zeitlosen Transitionen enthalten. In Abbildung 3.10 werden die einzelnen Teilschritte dieser Behandlung dargestellt.

Die Schritte 2 und 5d finden bei der adaptiven Simulation zwar zu einem andern Zeitpunkt aber dennoch statt, da die Ersatz-Transitionen ihre Laufzeiten aus denen des Sub-Netzes berechnen. Die anderen Schritte jedoch müssen nur einmal beim bzw. nach dem Feuern einer Ersatz-Transition durchgeführt werden. Repräsentiert eine solche Transition n Transitionen aus dem Sub-Netz , werden die Schritte 1 3, 4, 5a, 5b sowie 5c demzufolge $(n-1)$-mal eingespart. Im Folgenden wird der Einfluss des Petri-Netzes auf die für die einzelnen Schritte benötigte Rechenzeit untersucht.

Aufgrund der hohen Komplexität der verwendeten Petri-Netze muss der in dieser Arbeit benutzte Simulator bei der Suche nach den aktivierten Transitionen (Schritt 1) stets alle zeitlosen Transitionen überprüfen. Damit spielt die Gesamtzahl an zeitbehafteten Transitionen eine Rolle für den Zeitaufwand, welcher für Schritt 1 benötigt wird. Dieser Zusammenhang ist in Tabelle 3.1 erkennbar. Die Tabelle zeigt die für die Schritte 1 3, 5a, 5b, 5c sowie die insgesamt für eine Abarbeitung der zeitbehafteten Transitionen benötigte Rechenzeit in Form von Konfidenz Intervallen. Die Zeit für das Erhöhen der Gesamtzeit (4) wurde nicht mitgemessen, da dieser Schritt eine einzelne Operation und der Rechenaufwand damit von der Art des Petri-Netzes unabhängig ist. Für jeden Schritt zeigt die Zeile mit dem „$\uparrow$" die obere und die Zeile mit dem „$\downarrow$" die untere Grenze des Intervalls an. In der Zeile mit dem „$-$" steht der Mittelwert der jeweiligen Messung. Aus programmiertechnischen Gründen ist leider in der Zeitdauer für Schritt 1 auch die Dauer für die Berechnung des Schrittes 2 und in dem Aufwand für die Token (Schritte 5a und 5b) der Aufwand zum Löschen der Laufzeit (Schritt 5d) enthalten. Die Experimente werden daher mit Hilfe einer Single-Server Sequenz durchgeführt, welche aus n Transitionen

		5	10	20	50	500
Suche der aktivierten Transitionen	↑	0.008165	0.009939	0.015727	0.032806	1.427041
	–	0.007240	0.008800	0.014530	0.031210	1.400640
	↓	0.006315	0.007661	0.013333	0.029614	1.374239
Auswahl der feuernden Transition	↑	0.003627	0.003690	0.003274	0.003512	0.008477
	–	0.003000	0.003030	0.002840	0.002990	0.007440
	↓	0.002373	0.002370	0.002406	0.002468	0.006403
Token erzeugen/ löschen	↑	0.006934	0.005368	0.005197	0.005433	0.007025
	–	0.004960	0.004350	0.004310	0.004450	0.005870
	↓	0.002986	0.003332	0.003423	0.003467	0.004715
Laufzeiten anpassen	↑	0.002448	0.001926	0.002367	0.002741	0.003394
	–	0.002030	0.001600	0.001870	0.002120	0.002880
	↓	0.001612	0.001274	0.001373	0.001499	0.002366
gesamte Zeit	↑	0.025702	0.025984	0.031523	0.049034	1.458543
	–	0.023260	0.024060	0.029680	0.046870	1.432100
	↓	0.020818	0.022136	0.027837	0.044706	1.405657

Tabelle 3.1: Untersuchung des Einflusses der Anzahl von Transitionen auf die Zeitdauern der einzelnen Schritte einer Simulation. Jedes Konfidenz-Intervall wurde aus 100000 Messwerten mit einem $\alpha = 0.01$ berechnet. Alle Zeitangaben in Millisekunden.

und Stellen besteht. Die Verteilungsfunktionen der Transitionen sind deterministisch und so gewählt, dass immer nur die erste (hat keine Eingangskante) und eine weitere, innere Transition aktiviert sind. Feuert die erste Transition, so müssen zwei neue Laufzeiten bestimmt werden, feuert die letzte, muss keine neue Laufzeit ermittelt werden, so dass im Schnitt nach jedem Feuern eine neue Laufzeit bestimmt wird. Der zu viel gemessene Aufwand für Schritt 2 ist daher von der Anzahl an Transitionen n unabhängig und während der gesamten Messungen ein konstanter Summand. Jede Spalte zeigt die Messergebnisse für ein n, in der ersten Zeile der Spalte steht die Anzahl an Transitionen. Mit steigendem n ist ein deutlicher Anstieg der benötigten Rechenzeit für Schritt 1 zu erkennen. Der Aufwand für Schritt 2 kann maximal den bei kleinstem n ($n = 5$) gemessenen Wert betragen, die Zeiten bei größeren n (≥ 20) betragen aber ein Vielfaches davon. Demzufolge steigt der Anteil der Zeitdauer für den Schritt 1 an der gesamten Zeitdauer für einen Zustandswechsel mit der Anzahl an zeitbehafteten Transitionen im Petri-Netz und damit ebenfalls der zu erwartende Gewinn bei der adaptiven Simulation. Dieser Anstieg erfolgt erst ziemlich stark, fällt dann jedoch ab bis der Anteil und demnach auch der Gewinn ungefähr gleich bleiben.

Die für die Wahl der zu feuernden Transition (Schritt 3) benötigte Rechenzeit hängt mit der Menge der aktivierten Transitionen zusammen. Tabelle 3.2 verdeutlicht diesen Einfluss. Als Petri-Netz wird ein Ring aus 50 Single-Server Transitionen gewählt (Ring = geschlossene Sequenz) mit verschieden hoher Anzahl n an gleichzeitig aktivierten Transitionen. Dabei sind die deterministisch erzeugten Laufzeiten sowie die Verteilung der Token so gewählt, dass immer n Transitionen aktiviert sind. Der Anstieg der Zeitdauer für Schritt 3 ist klar erkennbar. Mit wachsender Anzahl an aktivierten Transitionen steigt auch die Anzahl der anzupassenden Laufzeiten (Schritt 5c), was sich in einem Anstieg der Rechenzeit für diesen Schritt niederschlägt. Der Aktivierungsgrad verbunden mit der Vielfachheit der aktivierten Transitionen ist ein weiterer Faktor für die Anzahl

		1	2	5	10	20
Suche der aktivierten Transitionen	↑	0.035251	0.032599	0.034381	0.036264	0.047791
	–	0.032000	0.030930	0.032700	0.034570	0.042960
	↓	0.028749	0.029261	0.031019	0.032876	0.038129
Auswahl der feuernden Transition	↑	0.004010	0.004011	0.005856	0.008593	0.017152
	–	0.003300	0.003330	0.004990	0.007640	0.014180
	↓	0.002590	0.002649	0.004124	0.006687	0.011208
Token erzeugen/ löschen	↑	0.010462	0.005176	0.005358	0.004748	0.013357
	–	0.005920	0.004220	0.004420	0.003980	0.006690
	↓	0.001378	0.003264	0.003482	0.003212	2.264940E-5
Laufzeiten anpassen	↑	0.001919	0.003100	0.004035	0.006484	0.019856
	–	0.001530	0.002470	0.003330	0.005600	0.012970
	↓	0.001141	0.001840	0.002625	0.004716	0.006084
gesamte Zeit	↑	0.054930	0.049033	0.053636	0.060491	0.094420
	–	0.049230	0.046790	0.051300	0.058120	0.083270
	↓	0.043530	0.044547	0.048964	0.055749	0.072120

Tabelle 3.2: Untersuchung des Einflusses der Anzahl von aktivierten Transitionen auf die Zeitdauern der einzelnen Schritte einer Simulation. Jedes Konfidenz-Intervall wurde aus 100000 Messwerten mit einem $\alpha = 0.01$ berechnet. Alle Zeitangaben in Millisekunden.

an Laufzeiten. Dieser Zusammenhang wird in Tabelle 3.3 untersucht. Das entsprechende Petri-Netz besteht aus einer Infinite-Server Transition und einer Stelle, die gleichzeitig Eingangs- und Ausgangsstelle ist. Die Anzahl x an Token in der Stelle und damit der Aktivierungsgrad wird bei jedem neuen Experiment erhöht. In der ersten Zeile ist die Anzahl der anzupassenden Laufzeiten $n = x - 1$ angegeben. Die Spalte mit $n = 0$ zeigt demzufolge den Aufwand, der allein der Test, ob Laufzeiten verändert werden müssen, benötigt. Mit steigendem n ist ein deutlicher Anstieg der Rechenzeit für Schritt 5c zu erkennen. Bei der adaptiven Simulation werden die Schritte 3 und 5c eingespart, weshalb mit steigender Anzahl an aktivierten Transitionen und/oder Laufzeiten der zu erwartende Gewinn wächst. Allerdings kann es durch die vielen möglichen Feuervorgänge schnell passieren, dass von den Ersatz-Transitionen auf das Sub-Netz heruntergeschaltet werden muss, was wiederum die adaptive Simulation verlangsamt.

Als letztes soll der Einfluss der Anzahl an Eingangs- und Ausgangsstellen der feuernden Transition auf die Zeitdauer der Tokenentfernung und -erzeugung untersucht werden. Dafür wird folgendes Petri-Netz gewählt: Eine Single-Server Transition besitzt n Eingangs- und Ausgangsstellen. In den Eingangsstellen sind genügend Token enthalten, so dass die Transition während der gesamten Simulation aktiviert ist. Bei dem Petri-Netz wird nach jedem Feuern eine Laufzeit gelöscht, daher ist der in dem Messungen für die Token enthaltene Aufwand für Schritt 5d nur ein konstanter Summand, der die Entwicklung über die Anzahl an Stellen nicht beeinflusst. Tabelle 3.4 zeigt das Ergebnis. Es ist erst bei einer größeren Anzahl an Stellen ein Anstieg zu erkennen ($n \geq 5$). Demzufolge ist bei weniger Stellen der Aufwand für Schritt 5d ausschlaggebend für die Rechenzeit. In den umzuformenden Figuren werden wahrscheinlich die Transitionen mit eher weniger als mehr Stellen verbunden sein, woraus sich vermuten lässt, dass der eingesparte Aufwand für Schritt 5a und 5b in dem zu erwartenden Gewinn vernachlässigbar ist.

In der Spalte $n = 0$ der Tabelle 3.3 ist der Aufwand für die Suche der aktivierten Transi-

		0	1	5	10	50
Suche der aktivierten Transitionen	↑	0.005894	0.005441	0.005274	0.005125	0.005069
	–	0.004970	0.004470	0.004460	0.004300	0.004280
	↓	0.004046	0.003499	0.003646	0.003475	0.003491
Auswahl der feuernden Transition	↑	0.002988	0.003279	0.003166	0.004125	0.012403
	–	0.002400	0.002660	0.002740	0.003590	0.011420
	↓	0.001812	0.002041	0.002314	0.003055	0.010437
Token erzeugen/ löschen	↑	0.005147	0.005296	0.004733	0.005807	0.004715
	–	0.004230	0.004330	0.003960	0.004750	0.003870
	↓	0.003313	0.003364	0.003187	0.003693	0.003025
Laufzeiten anpassen	↑	0.002406	0.002444	0.004828	0.007003	0.026657
	–	0.001850	0.001880	0.003980	0.005910	0.024920
	↓	0.001294	0.001316	0.003132	0.004817	0.023183
gesamte Zeit	↑	0.022223	0.022040	0.023348	0.027038	0.053336
	–	0.020430	0.020190	0.021620	0.025000	0.050910
	↓	0.018637	0.018340	0.019892	0.022962	0.048484

Tabelle 3.3: Untersuchung des Einflusses des Aktivierungsgrades eines Infinite-Servers auf die Zeitdauern der einzelnen Schritte einer Simulation. Jedes Konfidenz-Intervall wurde aus 100000 Messwerten mit einem $\alpha = 0.01$ berechnet. Alle Zeitangaben in Millisekunden.

tionen minimal, da es nur eine gibt. Demzufolge wird angenommen, der gemessene Aufwand ist durch das Erzeugen der neuen Laufzeiten zustande gekommen. Ebenso gibt es nur eine aktivierte Transition und eine Eingangs- und Ausgangsstelle. Die Zeitdauer für die Token wird daher hauptsächlich durch das Löschen der Laufzeit verursacht. Der Zeitaufwand für die Änderung der Laufzeiten ist minimal, da es keine Laufzeiten zu modifizieren gibt. Damit ist der gesamte Aufwand für die Bearbeitung der zeitbehafteten Transitionen ebenfalls im kleinst möglichen Bereich. Die bei der adaptiven Simulation nicht eingesparten Schritte ergeben ungefähr die Hälfte der gesamten Zeitdauer. Die andere Hälfte wird jedoch eingespart. Die adaptive Simulation enthält also Beschleunigungspotential.
Allerdings benötigt die adaptive Simulation auch zusätzliche Schritte. Es müssen zusätzlich zu den originalen Laufzeiten die Laufzeiten der Ersatz-Transitionen erzeugt und gelöscht werden. Desweiteren muss immer, wenn sich die Anzahl an Token in der ersetzten Figur ändert, überprüft werden, ob ein Schalten auf die originale Topologie notwendig ist. Gleiches gilt für den umgekehrten Fall. Außerdem benötigt der eigentliche Schaltvorgang an sich zusätzlichen Rechenaufwand.
Die adaptive Simulation hat demzufolge beschleunigende und verlangsamende Faktoren. Die Höhe dieser Faktoren hängt mit dem Aufbau des zu transformierenden Petri-Netzes zusammen. Eine allgemeine Aussage, ob sich die adaptive Simulation lohnt, kann aus diesen Gründen nicht getroffen werden, es hängt immer von dem jeweiligen Petri-Netz ab.

		1	2	5	10
Suche der aktivierten Transitionen	↑	0.005191	0.006428	0.007137	0.008823
	–	0.004390	0.005400	0.006140	0.007880
	↓	0.003589	0.004372	0.005143	0.006937
Auswahl der feuernden Transition	↑	0.003510	0.002665	0.002769	0.003588
	–	0.002810	0.002210	0.002320	0.002890
	↓	0.002110	0.001755	0.001871	0.002192
Token erzeugen/ löschen	↑	0.004791	0.004950	0.006049	0.010665
	–	0.003970	0.003950	0.004990	0.006890
	↓	0.003149	0.002950	0.003931	0.003115
Laufzeiten anpassen	↑	0.002501	0.001883	0.001852	0.001745
	–	0.001840	0.001470	0.001470	0.001320
	↓	0.001179	0.001057	0.001088	8.946297E-4
gesamte Zeit	↑	0.021243	0.021409	0.023808	0.030732
	–	0.019500	0.019540	0.021930	0.026650
	↓	0.017757	0.017671	0.020052	0.022568

Tabelle 3.4: Untersuchung des Einflusses der Anzahl von Eingangs- und Ausgangsstellen auf die Zeitdauern der einzelnen Schritte einer Simulation. Jedes Konfidenz-Intervall wurde aus 100000 Messwerten mit einem $\alpha = 0.01$ berechnet. Alle Zeitangaben in Millisekunden.

Kapitel 4

Verschiedene Topologien

Im 2. Kapitel wurde für verschiedene Figuren eine Umformung des Zustandsraumes vorgestellt. Die Idee, diese Transformation auch auf Petri-Netze anzuwenden, führte zur adaptiven Simulation, welche im Kapitel 3 dargestellt und erweitert wurde. Durch diese Ausweitung ist es möglich geworden, die Figuren aus Kapitel 2 bei Petri-Netzen auch dann zu transformieren, wenn sie nicht exakt analog zu denen im Zustandsraum sind. Stattdessen ergeben sich neue Bedingungen, die sie erfüllen müssen. In diesem Kapitel werden diese Voraussetzungen, sowie die Topologien der Ersatz-Transitionen und eventuelle globale Nutzungs-Bedingungen untersucht und es wird auf die Berechnung der Laufzeiten und auf die verschiedenen Ersetzungs-Bedingungen eingegangen. Desweiteren wird gezeigt, dass sich die erweiterte adaptive Simulation für die verschiedenen Figuren korrekt verhält sowie eine Analyse des Gewinnpotentials vorgenommen. Am Ende des Kapitels wird auf die Korrektheit sowie das Potential des Gewinns der adaptiven Simulation für eine Hierarchie von Figuren eingegangen.

4.1 Allgemein

Dieser Abschnitt behandelt noch einmal ausführlich die allgemeinen Bedingungen, die die verschiedenen Figuren erfüllen müssen, damit sie umgeformt werden können. Insbesondere wird dabei auf die Gründe eingegangen, warum einige der Erweiterungen aus 2.3 nicht in das in dieser Arbeit behandelte Konzept der adaptiven Simulation übernommen wurden.

Bei der adaptiven Simulation wird wie bei der Transformation des Zustandsgraphen die Statistik der Transitionen und Stellen in der umgeformten Figur geändert. Werden die Ersatz-Transitionen benutzt, so befinden sich die Token, die ansonsten in dem Sub-Netz sind, in den Eingangsstellen der Topologie, deren Statistik demzufolge auch verändert wird. Mit Statistik sind die Informationen über das Verhalten der Stelle oder Transition gemeint, z. B. die mittlere Anzahl an Token in der Stelle oder die Feuerfrequenz einer Transition. Die Transitionen und Stellen der zu transformierenden Figur sowie deren Eingangsstellen müssen demzufolge als uninteressant markiert sein, dass bedeutet, dass die Statistik von ihnen nicht von Bedeutung ist. Die Ausgangsstellen des Sub-Netzes müssen dies nicht, da ihre Statistik nicht verändert wird, sogar nicht verändert werden darf!

In 2.3 wurde die Ersetzung einzelner fester Parameter durch Funktionen erlaubt. In einer Figur mit solchen Funktionen kann das Verhalten bei jedem Feuern anders sein. Damit ist es aber auch nicht oder nur schwer möglich, dieses mit Hilfe von Ersatz-Transitionen zu imitieren. In einer Topologie, die umgeformt werden soll, dürfen demzufolge keine Parameter durch Funktionen ersetzt sein.
In dem in dieser Arbeit erweitertem Konzept der adaptiven Simulation wird die Laufzeit der Ersatz-Transitionen aus den Laufzeiten der Transitionen des Sub-Netzes ermittelt. Ein Herunterschalten wird dann notwendig, wenn dies nicht mehr möglich ist, also wenn eine Laufzeit im Sub-Netz erst nach dem ,,ablaufen" einer anderen ermittelt werden kann. Dies kann allerdings nur eintreten, wenn in einer der Eingangsstellen Token erzeugt wurden. Eine ständige, zeitaufwendige Abfrage der Nutzungs-Bedingung ist damit nicht notwendig. Der Fall, dass eine Transition im Sub-Netz aufgrund von Einflüssen deaktiviert wird, die nichts mit der Figur selbst zu tun haben, zerstört aber dieses Konzept. Alle Topologien, wo dieser Fall eintreten kann, dürfen demzufolge bei der in dieser Arbeit behandelten adaptiven Simulation nicht transformiert werden. Damit ergeben sich für eine umzuformende Figur folgende Einschränkungen: Die Transitionen dürfen keine Guardfunktion besitzen und in keiner Reset-Liste irgendeiner Transition vorkommen und sie müssen alle dieselbe Priorität haben.
Die letzte Einschänkung bewirkt, dass alle Transitionen in einem Sub-Netz die gleiche Priorität haben, die dann auch die Ersatz-Transitionen bekommen. Werden die Ersatz-Transitionen benutzt und sind aktivert, so kann es passieren, dass eine andere Transition mit höherer Priorität aktiviert wird und die Ersatz-Transitionen deaktiviert. Bei einer späteren Aktivierung können die Ersatz-Transitionen allerdings die Laufzeiten nicht einfach komplett neu berechnen. Ein paar der originalen Laufzeiten waren zu dem Zeitpunkt der Deaktivierung bereits ,,abgelaufen", dürfen demzufolge nicht noch einmal mit einberechnet werden. Desweiteren muss auf die race-policies der einzelnen Sub-Netz-Transitionen Rücksicht genommen werden. Ein weiteres Problem tritt ein, wenn von den so deaktivierten Ersatz-Transitionen auf das Sub-Netz geschaltet werden muss. Die Lösung, bei einer solchen Deaktivierung der Ersatz-Transitionen gleich auf das Sub-Netz zu schalten, passt nicht in das in dieser Arbeit verwendete Konzept der Zeitpunkte, wann geschaltet wird: Die Tatsache, dass die Deaktivierung aufgrund einer Transition mit höherer Priorität stattfindet, wird erst bei der Suche nach den aktivierten Transitionen erkannt. Die Alternative, eine spezielle Deaktivierungs-Berechnung, ist aufwendig und verlangsamt die adaptive Simulation. Aus diesem Grund wird in dieser Arbeit auf eine solche Behandlung des Falles verzichtet. Damit müssen alle Transitionen eines Sub-Netzes die höchste Priorität besitzen, die im gesamten Petri-Netz vorkommt.
Die in 2.3 eingeführten Stellenkapazitäten können das Feuern einer Transition verhindern. Besitzt eine Ausgangsstelle einer Ersatz-Transition eine solche Kapazität, kann sie wegen der enthaltenen Anzahl an Token die Transition deaktivieren. Im Sub-Netz dagegen wären nur die direkt mit dieser Stelle verbundenen Transitionen deaktiviert, die anderen könnten weiterlaufen. Damit müsste heruntergeschaltet werden, da das Verhalten der Ersatz-Transition nicht korrekt ist. Dieser Fall ist in dem in dieser Arbeit entwickeltem Konzept ebenfalls ausgeschlossen, womit keine Ausgangsstelle eines Sub-Netzes eine Stellenkapazität haben darf.
Besitzt eine Stelle in einer Figur eine Stellenkapazität, so kann der Fall eintreten, dass eine Laufzeit für eine Transition, bei der sie eine Ausgangsstelle ist, erst erzeugt werden kann,

```
int sum = 0;
for(int i = 0; i < 10; i++)
   sum += mark("P"+i);
return sum;
```

Abbildung 4.1: Beispiel für die Verwendung der mark-Funktion

wenn eine andere Transition, bei der die Stelle eine Eingangsstelle ist, gefeuert hat. In dieser Situation müsste auf das Sub-Netz geschaltet werden. Die Abfrage, ob ein solcher Fall eintritt, ist allerdings kompliziert und aufwendig. Um diese zu sparen, ist es bei dem in dieser Arbeit vorgestelltem Konzept verboten, dass eine innere Stelle einer Figur eine Stellenkapazität besitzt.
Ein ganz anderes Problem ergibt sich, wenn durch das Feuern einer Transition aus dem Sub-Netz eine beliebige Transition außerhalb der Figur deaktiviert wird oder deren Laufzeit neu berechnet werden muss. Die Ersatz-Transitionen feuern immer zu den Zeitpunkten, bei denen auch eine End-Transition des Sub-Netzes gefeuert hätte. Allerdings passiert zu den Zeiten, bei denen innere Transitionen gefeuert hätten, gar nichts. Der Effekt der Deaktivierung einer anderen Transition kann demzufolge nicht durch die Ersatz-Transitionen nachgebildet werden. Damit dürfen aber alle Transitionen eines Sub-Netzes, die keine End-Transitionen sind, keine Reset-Liste haben und jede Ersatz-Transition bekommt dieselbe Reset-Liste wie die End-Transition, die sie repräsentiert. Aus dem gleichen Grund dürfen auch keine Kostenfunktionen bei dem Feuern einer der inneren Transitionen ausgewertet werden. Wird eine Kostenfunktion beim Feuern einer End-Transition ausgewertet, so wird sie das auch beim Feuern der entsprechenden Ersatz-Transition.
Ein ähnlicher Fall tritt ein, wenn eine Transition T_i im Petri-Netz die race-reset-policy hat. In diesem Fall wird für die Transition T_i bei jedem Feuern einer anderen Transition eine neue Laufzeit berechnet. Existiert aber eine Ersatz-Transition, so fehlen die Zeitpunkte des Feuerns für die Transitionen aus dem Sub-Netz und damit auch einige der Zeitpunkte, bei denen die Laufzeit von T_i eigentlich neu erzeugt werden müsste. Hat demzufolge mindestens eine Transition im gesamten Petri-Netz die race-reset-policy, so kann die adaptive Simulation bei diesem Petri-Netz nicht angewendet werden.
Ein analoges Problem verursacht eine Kostenfunktion, die direkt vor dem Feuern einer beliebigen Transition ausgewertet wird. In diesem Fall fehlen bei der Benutzung einer Ersatz-Transition einige der Zeitpunkte, zu denen eigentlich die Kostenfunktion ausgewertet werden müsste. Befindet sich daher eine solche Kostenfunktion im Petri-Netz, so darf die adaptive Simulation ebenfalls nicht angewendet werden.
Die Möglichkeit, in einer Parameter-Funktion die Anzahl an Token in einer bestimmten Stelle abzufragen, verursacht ein weiteres Problem. Die Stellen in einem ersetzten Sub-Netz enthalten keine Token, wenn die Ersatz-Transitionen benutzt werden. Wird die Anzahl an Token in einer solchen Stelle abgefragt, so ist diese null, was nicht dem Wert entspricht, den sie hätte, wenn das Sub-Netz benutzt werden würde. Demzufolge dürfen keine Figuren transformiert werden, die Stellen enthalten, deren Anzahl an Token von einer Parameter-Funktion zu irgendeinem Zeitpunkt abgefragt wird.

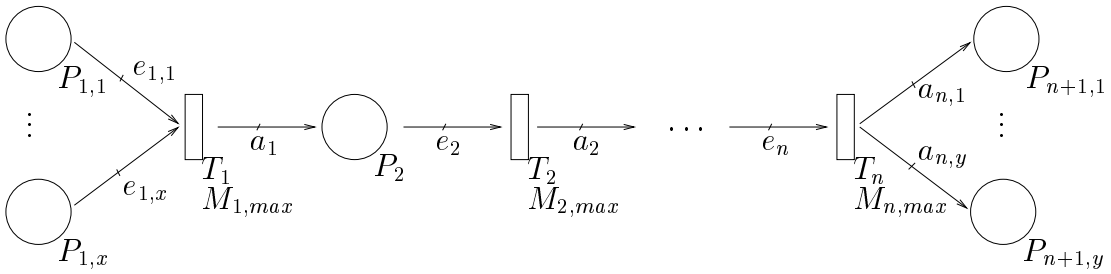


Abbildung 4.2: Eine allgemeine, ersetzbare Sequenz

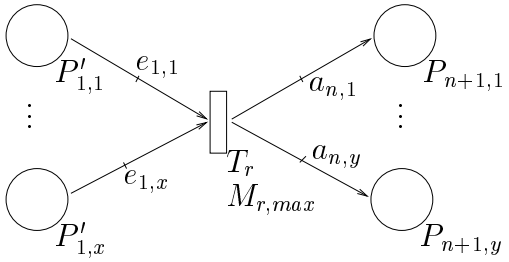


Abbildung 4.3: Die ersetzte Sequenz

In dem realisierten Simulator, in den das in dieser Arbeit vorgestellte Konzept der adaptiven Simulation eingebaut wurde, wird die Abfrage der Anzahl an Token mit Hilfe einer mark(P_i)-Funktion ermöglicht, die die Anzahl der Token in der Stelle P_i zurückliefert. Desweiteren steht in den Parameter-Funktionen dem Benutzer die volle Mächtigkeit der Programmiersprache Java zur Verfügung. Eine Verwendung der mark-Funktion kann daher wie in Abbildung 4.1 aussehen. Eine Analyse aller verwendeten Parameter-Funktionen nach der Menge der abgefragten Stellen wird durch diese Mächtigkeit sehr kompliziert und aufwendig. Deshalb ist in dieser Arbeit die Verwendung der adaptiven Simulation verboten, wenn irgendeine Funktion die mark-Funktion verwendet.
Die Bedingungen bezüglich der Topologie, also wie viele Eingangs- und Ausgangskanten erlaubt sind etc., sind je nach betrachteter Figur unterschiedlich, weswegen sie auch in den entsprechenden Abschnitten behandelt werden.

4.2 Sequenz

Eine allgemeine Sequenz, die mit den in dieser Arbeit gewonnenen Erkenntnissen und Einschränkungen mit Hilfe der adaptiven Simulation verkürzt werden kann, ist in Abbildung 4.2 zu sehen. Sie besteht aus n Transitionen, von denen die Transitionen T_1 bis T_{n-1} exakt eine Ausgangskante und die Transitionen T_2 bis T_n genau eine Eingangskante besitzen. Die Stellen P_2 bis P_n sind ebenso mit exakt einer hineinlaufenden und einer ausgehenden Kante verbunden. Die Stellen $P_{1,i}$ bilden die Eingangsstellen der Sequenz, von ihnen darf jeweils nur eine Kante weggehen, die Ausgangsstellen der Sequenz sind die Stellen $P_{n+1,j}$. Alle Transitionen sowie die Stellen $P_{1,i}$ bis P_n dürfen mit keinen inhibitorischen Kanten verbunden sein.
Damit eine Ersetzung der Sequenz möglich ist, muss die Anfangs-Transition T_1 mindestens

eine Eingangsstelle besitzen. Ist dies nicht der Fall, so müsste nach jedem Feuern von T_1 für T_1 eine neue Laufzeit bestimmt werden. Bei einer Ersatz-Transition T_r würde zu diesen Feuerzeitpunkten allerdings nichts passieren, T_r würde erst nach dem eigenen Feuern eine neue Laufzeit bekommen, was nicht dem korrekten Verhalten entsprechen würde.
Die Eingangskanten von T_1 dürfen ebenso wie die Ausgangskanten von T_n jede beliebige Vielfachheit haben. Für die restlichen Kanten muss folgender Zusammenhang zwischen den Kantenvielfachheiten gelten: Die Vielfachheit der Ausgangskante von T_i muss gleich der Vielfachheit der Eingangskante von T_{i+1} sein:

$$m_{ausgang,i} = a_i = e_{i+1} = m_{eingang,i+1} \tag{4.1}$$

Durch diese Bedingung ist gegeben, dass immer, wenn eine Transition der Sequenz feuert, die nachfolgende Transition genau einmal aktiviert wird. Das Feuern einer Transition ,,wandert" quasi durch die Sequenz, eine Zeitspanne nach der Aktivierung von T_1 feuert T_n exakt einmal. Damit kann die Sequenz durch eine Ersatz-Transition T_r dargestellt werden, welche in Abbildung 4.3 zu sehen ist.
Die Ersatz-Transition T_r hat die gleichen Eingangskanten wie T_1 und die gleichen Ausgangskanten wie T_n mit jeweils den gleichen Vielfachheiten. Für die Vielfachheit von T_r gilt:

$$M_{r,max} = \min(M_{i,max}); \qquad i = 1, \ldots, n \tag{4.2}$$

Für $M_{r,max}$ Aktivierungsgrade kann jede der n Transitionen der Sequenz eine eigene Laufzeit generieren, damit behindern sich $M_{r,max}$ Laufzeiten in der Sequenz nicht, für sie bildet die Ersatz-Transition das Verhalten korrekt nach. Zusätzlich ist durch die in 4.1 gemachten Einschränkungen sichergestellt, dass keine der n Transitionen nach einer Aktivierung wieder deaktiviert werden kann. Damit ist es unwichtig, ob die Transitionen die race-enable, race-age oder race-repeat Strategie haben, in diesem Fall verhalten sich alle wie race-enable. Die race-policies der Transitionen haben demzufolge keinen Einfluss auf die Korrektheit des Verhaltens der Ersatz-Transition. Dank der Tatsache, dass keine der n Transitionen je deaktiviert werden kann, kann auch T_r nie deaktiviert werden, die race-policy von T_r ist demnach egal, es wird die race-enable Strategie gewählt. Ein Grenzwert B für die globale Nutzungs-Bedingung ist hier nicht notwendig, da die Sequenz lediglich durch eine Transition ersetzt wird.
Die Berechnung einer neuen Laufzeit t_r von T_r erfolgt durch die Addition der Laufzeiten der Transitionen der Sequenz:

$$t_r = \sum_{i=1}^{n} t_i \tag{4.3}$$

Bei der Überprüfung, ob von der Sequenz wieder hochgeschaltet werden kann, muss die Ersetzungs-Bedingung lediglich die Summe der Aktivierungsgrade der n Transitionen ermitteln. Ist diese kleiner als $M_{r,max}$, so kann von der Sequenz auf die Ersatz-Transition geschaltet werden.
Die Laufzeiten der Ersatz-Transition werden beim Hochschalten folgendermaßen berechnet. Hat eine Transition T_i eine Restlaufzeit, so wird für die Transitionen T_{i+1} bis T_n eine neue Laufzeit generiert und die Ersatz-Transition erhält die Summe dieser Zeiten als eine neue Laufzeit. Außerdem werden e_i Token aus der Eingangsstelle P_i von T_i entfernt und $e_{1,j}$ Token in den Eingangsstellen $P_{1,j}$ erzeugt. Nach einem Schaltvorgang befinden sich demzufolge keine Token mehr in den inneren Stellen der Sequenz.

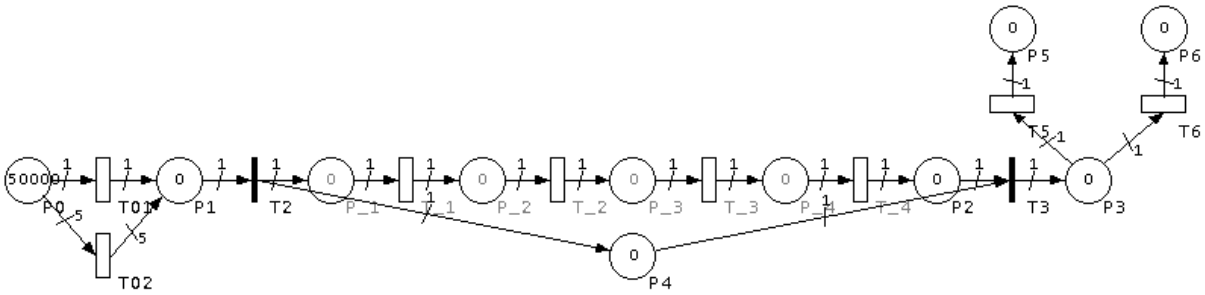


Abbildung 4.4: Das Petri-Netz, mit dem die Korrektheit überprüft wird

Bei dem Herunterschalten auf die Sequenz wird für jede Laufzeit t_i der Ersatz-Transition die Transition T_j in der Sequenz bestimmt, bei der sich die Laufzeit gerade „befindet". Diese Transitionen sind aus den gespeicherten originalen Laufzeiten und den Restlaufzeiten von T_r berechenbar, ebenso wie die neue Laufzeit von T_j. In der Eingangsstelle P_j von T_j werden e_j Token hinzugefügt und aus jeder Eingangsstelle $P_{1,l}$ werden $e_{1,l}$ Token entfernt. Die originalen Laufzeiten der Transitionen T_{j+1} bis T_n, die zu der Laufzeit t_i der Ersatz-Transition beigetragen haben, müssen in den jeweiligen Transitionen zwischengespeichert und bei der nächsten Aktivierung verwendet werden. Sollten bei einer Transition der Sequenz nach dem Schalten mehrere originale Zeiten gespeichert sein, so müssen diese nacheinander bei den nächsten Aktivierungen verwendet werden. Die dabei verwendete Reihenfolge der Laufzeiten darf nicht von ihren Werten abhängen, da ansonsten der Unterschied zweier aufeinanderfolgenden Laufzeiten nicht mehr zufällig ist.

Um die Korrektheit der hier vorgestellten Ersatz-Transition sowie der Schaltbedingungen und des Schaltvorgangs für die Sequenz zu zeigen, werden im Folgenden verschieden Experimente anhand des in Abbildung 4.4 dargestellten Petri-Netzes durchgeführt, indem die Multi-Server Transitionen T_i eine ersetzbare Sequenz bilden. Dabei haben die Transitionen T_2 und T_4 die race-enable Strategie, T_1 hat race-age und T_3 die race-repeat Strategie. In der Stelle P4 wird die Zeitdauer, die ein Token durchschnittlich in der Sequenz verbringt, gemessen. Durch die Multi-Server Transitionen T5 und T6 wird die Ankunftscharakteristik der Token untersucht, d.h. in welchem zeitlichen Abstand sie aus der Sequenz kommen. Demzufolge ist die Statistik dieser Stellen und Transitionen von besonderem Interesse.
Bei der adaptive Simulation ergeben sich zwei verschiedene Kenngrößen für ein Sub-Netz: V_f, die das Verhältnis der Anzahl an Feuervorgängen der Ersatz-Transitionen zu denen des Sub-Netzes beinhaltet, sowie V_s, die das Verhältnis der Anzahl an Feuervorgängen der Ersatz-Transitionen zu der Anzahl an Schaltungen auf das Sub-Netz beschreibt. Es werden drei verschiedene Versuche mit jeweils anderen Verteilungsfunktionen für die Transitionen T01 und T02 und damit anderen Kenngrößten V_f und V_s durchgeführt. Im folgenden Versuch wird die Transition T02 weggelassen und die Verteilung von T01 so gewählt, dass nie von der Ersatz-Transition heruntergeschaltet werden muss, $V_f = 1:0$ und $V_s = 1:0$. Desweiteren wird das Verhalten des Petri-Netzes über 10 Replikationen beobachtet.
Für jede Stelle werden in jeder Replikation die mittlere Tokenzahl sowie die mittlere Auslastung, für jede Transition die mittlere Feuerrate und ebenfalls die mittlere Auslastung

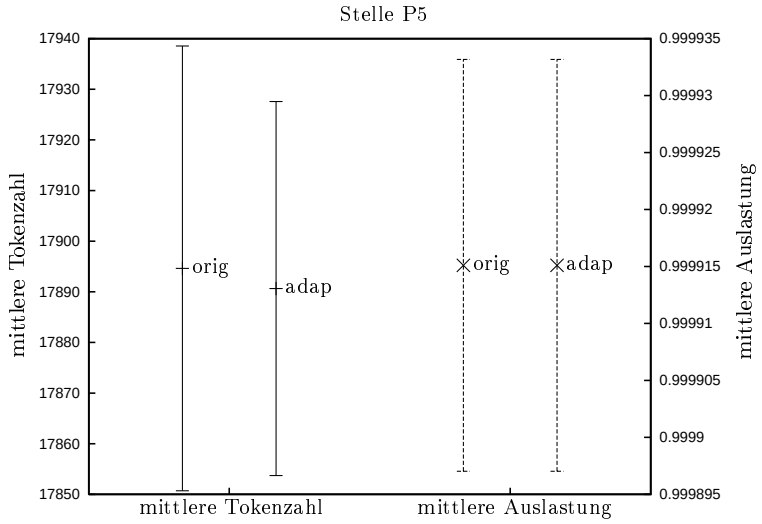


Abbildung 4.5: Versuch 1: Statistik-Vergleich für die Stelle P5 bei 10 Replikationen, $V_f = 1 : 0$, $V_s = 1 : 0$

gemessen und anschließend werden der Mittelwert

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^{n} Y_i; \qquad Y_i : \text{Meßwert in Replikation i von n} \tag{4.4}$$

sowie die Standardabweichung

$$\hat{\sigma}^2 \left(\hat{\theta}\right) = \frac{S^2}{n} = \frac{\sum_{i=1}^{n} \frac{\left(Y_i - \hat{\theta}\right)^2}{n-1}}{n} \tag{4.5}$$

ermittelt. Aus diesen lässt sich mit Hilfe eines Konfidenz Intervalls der reale Mittelwert θ abschätzen [BCNN01].

$$\hat{\theta} - t_{\frac{\alpha}{2},f} \hat{\sigma} \left(\hat{\theta}\right) \leq \theta \leq \hat{\theta} + t_{\frac{\alpha}{2},f} \hat{\sigma} \left(\hat{\theta}\right) \tag{4.6}$$

Zum Vergleich der Ergebnisse wird jeweils der Mittelwert und die Standardabweichung graphisch dargestellt, wodurch Unterschiede leichter zu erkennen sind. Da für $\alpha \leq 0.1$ immer gilt, dass $t_{\frac{\alpha}{2},f} > 1$ [BCNN01] und $t_{\frac{\alpha}{2},f}$ die gemessene Standardabweichung lediglich skaliert, wurde auf diesen Faktor verzichtet. In Abbildung 4.5 ist die Statistik der adaptiven Simulation („adap") für die Stelle P5 der der normalen, originalen Simulationsmethode („orig") gegenübergestellt. In jedem Balken ist in der Mitte der Mittelwert gekennzeichnet, der Abstand eines Endpunktes des Balken von der Mitte entspricht der ermittelten Standardabweichung. Der Vergleich aller Statistiken ist in Abbildung B.1 und Abbildung B.2 im Anhang B zu sehen.
Um einen aussagekräftigeren Vergleich der Ergebnisse zu erhalten, wird bei diesem wie in allen folgenden Versuchen die Methode des „correlated sampling" [BCNN01] angewendet, d.h. dass die den Zufallszahlen zugrunde liegenden Zahlenströme bei beiden Simulationsarten identisch sind. Allerdings werden bei der adaptiven Simulation die Laufzeiten der Transitionen zu anderen Zeitpunkten als bei der normalen Simulation bestimmt. Bei Multi-Server Ersatz-Transitionen kann dies dazu führen, dass andere Zufallszahlen als bei

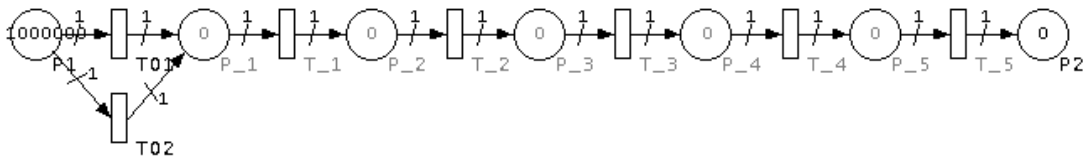


Abbildung 4.6: Petri-Netz für die Untersuchung des Zusammenhangs zwischen Gewinn und Schalthäufigkeit

Schalthäufigkeit	Rechenzeit		Gewinn		V_f	V_s	Zeit
	orig	adap	abs	%			Vorv.
nie	108.882	46.278	62.604	57.497	1 : 0.000	1 : 0.000	<0.001
selten	135.787	86.246	49.541	36.484	1 : 0.577	1 : 0.336	0.001
mittel	139.280	116.378	22.902	16.443	1 : 2.104	1 : 0.900	0.001
oft	141.164	131.610	9.554	6.768	1 : 5.342	1 : 1.902	0.001
immer	121.507	131.370	-9.863	-8.117	0 : 1.000	0 : 1.000	<0.001

Tabelle 4.1: Ergebnisse der Untersuchung des Zusammenhangs zwischen Gewinn und Schalthäufigkeit; alle Zeitangaben in Sekunden

der originalen Methode zu der Verweildauer eines Token in der Sequenz beitragen. Demnach kann es zu Unterschieden in den Ergebnissen kommen. Diese Differenzen müssten jedoch mit steigender Replikationenzahl abnehmen.
Es zeigt sich, dass die bei der adaptiven Simulation gewonnene Statistik in etwa der der originalen Simulationsmethode entspricht. Allerdings ist insbesondere bei den interessanten Stellen und Transitionen noch eine deutliche Abweichung vorhanden. Dies kann daran liegen, dass die Ergebnisse lediglich über 10 Replikationen ermittelt wurden. Erhöht man die Anzahl an Replikationen, müsste sich der Unterschied verringern. Deshalb wird ein zweiter Versuch mit exakt demgleichen Petri-Netz über 40 Replikationen durchgeführt. In Abbildung B.3 und Abbildung B.4 wird die Statistik der verschiedenen Simulationsmethoden miteinander verglichen.
Es ist eine deutliche Verbesserung zu erkennen. Die Statistik der adaptiven Simulation entspricht bis auf geringe Unterschiede der der originalen Simulationsmethode. Aus diesem Ergebnis lässt sich schließen, dass die Ersatz-Transition der Sequenz das Verhalten des originalen Petri-Netzes korrekt nachbildet, wenn nicht geschaltet werden muss.
Um auch das richtige Schaltverhalten der Ersatz-Transition zu überprüfen, werden zwei weitere Versuche durchgeführt. In dem ersten muss relativ wenig von der Ersatz-Transition heruntergeschaltet werden, $V_f = 1 : 0.537$ und $V_s = 1 : 0.137$, in dem zweiten häufiger $V_f = 1 : 1.948$, $V_s = 1 : 0.400$. Die Statistik wird jeweils über 40 Replikationen ermittelt.
Auch bei diesen beiden Versuchen gibt es nur geringe Unterschiede in den beiden Statistiken. Aus den durchgeführten Experimenten ist demzufolge erkennbar, dass die Ersatz-Transition einer Sequenz, die Schaltbedingungen und der Schaltvorgang, so wie sie in diesem Abschnitt vorgestellt wurden, das Verhalten der Sequenz korrekt nachbilden.

In den nun folgenden Versuchen soll der Gewinn an Rechenzeit, der sich mit der Benutzung der Ersatz-Transition der Sequenz erzielen lässt, untersucht werden. Dabei besitzt die

n	Rechenzeit		Gewinn		V_f	V_s	Zeit
	orig	adap	abs	%			Vorv.
2	89.378	89.037	0.341	0.382	1 : 0.000	1 : 0.000	0.002
5	88.880	84.383	4.497	5.060	1 : 0.000	1 : 0.000	0.002
10	88.169	76.073	12.096	13.719	1 : 0.000	1 : 0.000	0.001
50	78.191	27.896	50.295	64.323	1 : 0.000	1 : 0.000	0.004
100	73.056	1.750	71.306	97.605	1 : 0.000	1 : 0.000	0.008

Tabelle 4.2: Ergebnisse der Untersuchung des Zusammenhangs zwischen Gewinn und Anzahl n von Transitionen in der Sequenz; alle Zeitangaben in Sekunden

adaptive Simulation auch verlangsamende Faktoren, zu denen der Schaltvorgang gehört. Muss von der Ersatz-Transition mehrmals heruntergeschaltet werden, so müsste der Gewinn sinken. Dieses Verhalten wird in einem Experiment untersucht, für das verschiedene Petri-Netze verwendet werden, die analog zu dem in Abbildung 4.6 dargestellten sind. In diesem feuern die Transitionen T01 und T02 zusammen sowie jede anderen genau 1000000 mal. Die Transitionen T_i bilden eine ersetzbare Sequenz, wird die Ersatz-Transition benutzt, werden dadurch vier von sechs Feuervorgängen eingespart. In den verschiedenen Experimenten werden die Verteilungsfunktionen der Transitionen so geändert, dass sich die Schalthäufigkeit ändert. Bei einigen der Versuche wurde die Transition T02 weggelassen, was aber nichts an der Gesamtsituation ändert. Die Ergebnisse des Versuchs zeigt die Tabelle 4.1. Musste nie von der Ersatz-Transition heruntergeschaltet werden, so liegt der Gewinn bei 57.5 Prozent, was in etwa den Erwartungen entspricht, da bei der adaptiven Simulation nicht alle Schritte für einen Feuervorgang eingespart werden (siehe 3.5). Mit steigender Schalthäufigkeit sinkt wie vermutet der erzielte Gewinn. Bei dem letzten Versuch wurde der worst-case Fall untersucht: Immer wenn die End-Transition des Sub-Netzes feuert, wird auf die Ersatz-Transition geschaltet, von der aber unmittelbar danach wieder auf das Sub-Netz geschaltet werden muss. Demzufolge wird fast die ganze Simulationszeit über die Sequenz benutzt, aber dennoch nach jedem Feuern geschaltet. Damit wird durch die adaptive Simulation kein Feuervorgang eingespart, es entsteht nur zusätzlicher Aufwand für das Schalten und Überprüfen der Schaltbedingungen und die adaptive Simulation wird langsamer als die originale Simulationsmethode. Der Versuch zeigt, dass die adaptive Simulation für eine Sequenz eine spürbare Reduzierung der benötigten Rechenzeit bewirkt, solange die Ersatz-Transition ab und zu feuert. Besonders eindrucksvoll ist die Tatsache, dass selbst dann noch ein Gewinn erzielt wird, wenn häufig geschaltet werden muss.

Zusätzlich ist in der Tabelle auch die Rechenzeit enthalten, die für den Vorverarbeitungsschritt notwendig ist. Der mit der adaptiven Simulation erzielte absolute Gewinn oder Verlust wächst mit der Simulationszeit, während der Vorverarbeitungsschritt lediglich einmal ausgeführt werden muss. Außerdem bewegt sich die Rechenzeit für diesen in dem Bereich von Millisekunden. Aus diesen Gründen ist der Aufwand für den Vorverarbeitungsschritt in der Betrachtung des erzielten Gewinns an Rechenzeit vernachlässigbar.

Wird bei der Sequenz die Anzahl n der ersetzten Transitionen erhöht, so steigt damit die Anzahl an eingesparten Feuervorgängen ebenso wie der Aufwand zum Schalten und zum Überprüfen der Ersetzungs-Bedingung gleichermaßen. Da bereits in dem vorherigen Experiment der Gewinn bei unterschiedlichen Kennzahlen V_f und V_s untersucht wurde,

# inaktive Trans.	Rechenzeit		Gewinn		V_f	V_s	Zeit
	orig	adap	abs	%			Vorv.
0	9.202	3.675	5.527	60.063	1 : 0.000	1 : 0.000	0.047
5	10.620	4.222	6.398	60.245	1 : 0.000	1 : 0.000	0.046
10	12.129	4.785	7.344	60.549	1 : 0.000	1 : 0.000	0.001
50	26.500	10.435	16.065	60.623	1 : 0.000	1 : 0.000	0.001
100	48.275	18.828	29.447	60.998	1 : 0.000	1 : 0.000	0.001
250	121.875	46.048	75.827	62.217	1 : 0.000	1 : 0.000	0.004
500	892.394	297.883	594.511	66.620	1 : 0.000	1 : 0.000	0.008

Tabelle 4.3: Ergebnisse der Untersuchung des Zusammenhangs zwischen Gewinn und Aufwand für die Suche der aktivierten Transitionen; alle Zeitangaben in Sekunden

kann in den folgenden darauf verzichtet werden. In dem nächsten Versuch wird eine Sequenz von 100 Transitionen betrachtet und die Anzahl n der ersetzbaren erhöht. Vor der Sequenz befindet sich eine weitere Transition, die immer aktiviert ist und so feuert, dass immer maximal ein Token in der Sequenz enthalten ist. Mit steigendem n müsste der erzielte Gewinn ansteigen, was auch deutlich in Tabelle 4.2 zu erkennen ist. Werden alle 100 Transitionen ersetzt, so liegt er sogar bei knappen 98 Prozent!
In dem Abschnitt 3.5 wurde gezeigt, dass mit steigender Anzahl an Transitionen der Aufwand zur Suche der aktivierten Transitionen wächst. Demzufolge müsste ebenfalls der erzielte Gewinn bei der adaptiven Simulation ansteigen, bis der Aufwand der Suche fast dem ganzen Aufwand für einen Feuervorgang entspricht. Um diesen Zusammenhang zu untersuchen, wird ein Petri-Netz mit einer Sequenz von fünf Transitionen verwendet, vor der sich eine weitere Transition befindet, die immer aktiviert ist und so feuert, dass maximal ein Token in der Sequenz ist. Zusätzlich wird in das Petri-Netz eine Anzahl an inaktiven Transitionen eingefügt, die nie aktiviert sind und demzufolge lediglich den Suchaufwand beeinflussen. Die Ergebnisse des Versuchs zeigt die Tabelle 4.3. Es ist mit steigender Anzahl an inaktiven Transitionen auch ein Anwachsen des prozentualen Gewinns zu erkennen. Allerdings ist der Einfluss allgemein eher gering, so beträgt der Unterschied zwischen keiner und 500 inaktiven Transitionen lediglich knappe sieben Prozent, während der Gewinn sich in dem Bereich von 60 Prozent bewegt. Bei mehr als 500 Transitionen ist bei dem verwendeten Petri-Netz keine Steigerung des relativen Gewinns zu erwarten, da bei dieser Anzahl der Aufwand für die Suche bereits fast den ganzen Aufwand für einen Feuervorgang verursacht, wie in 3.5 zu sehen war.

4.3 Merge

In Abbildung 4.7 ist eine allgemeine, umformbare Merge-Figur zu sehen. Sie besteht aus n Anfangs-Transitionen $T_{1,i}$, von denen jede beliebig viele Eingangsstellen $P_{1,i,j}$ haben kann, aber mindestens eine haben muss. Die Stelle P_2 ist die einzige Ausgangsstelle aller Transitionen $T_{1,i}$ und ist mit exakt einer ausgehenden Kante verbunden, welche zu einer weiteren Transition T_2 führt. Diese hat nur diese eine Eingangsstelle, kann aber beliebig viele Ausgangskanten besitzen. Die Eingangsstellen des Merges sind die Stellen $P_{1,i,j}$, sie müssen exakt eine ausgehende Kante haben. Die Stelle P_2 darf ausschließlich mit den Kanten verbunden sein, die von den Transitionen $T_{1,i}$ kommend in sie hineinlaufen, sie

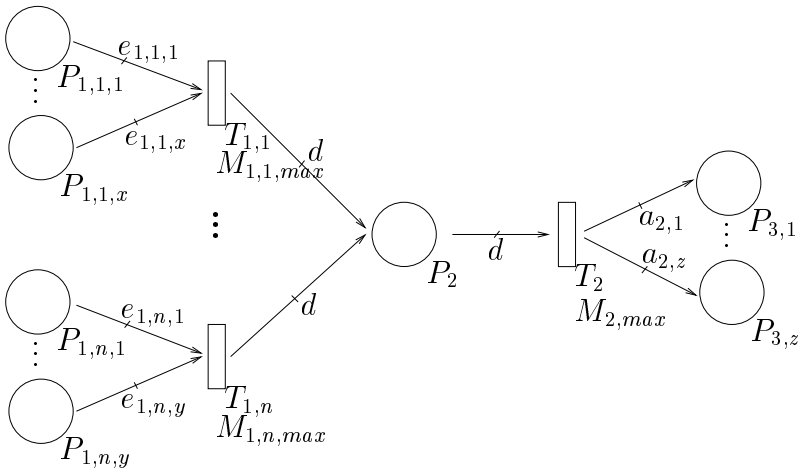


Abbildung 4.7: Ein allgemeiner, ersetzbarer Merge

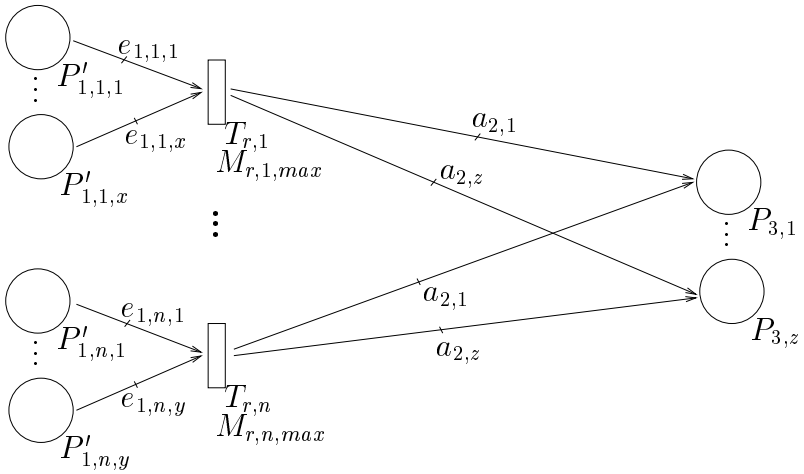


Abbildung 4.8: Der ersetzte Merge

hat demzufolge exakt n eingehende Kanten. Keine der Transitionen des Merges, keine der Eingangsstellen und auch nicht die mittlere Stelle dürfen mit inhibitorischen Kanten verbunden sein.
Die Eingangskanten der n Anfangs-Transitionen dürfen ebenso wie die Ausgangskanten von T_2 jede beliebige Vielfachheit besitzen. Die Vielfachheit der Ausgangskanten der n Anfangs-Transitionen muss gleich der Vielfachheit der Eingangskante der Transition T_2 sein. Damit ist sichergestellt, dass durch das Feuern einer Transition $T_{1,i}$ die Transition T_2 genau einmal aktiviert wird und die Merge-Figur sich durch n Ersatz-Transitionen darstellen lässt. Abbildung 4.8 zeigt die umgeformte Merge-Figur.
Jede Ersatz-Transition $T_{r,i}$ steht dabei für die Transitionen $T_{1,i}$ und T_2, hat demzufolge die gleichen Eingangskanten wie $T_{1,i}$ und die gleichen Ausgangskanten wie T_2 mit denselben Vielfachheiten. Für die Vielfachheit einer Ersatz-Transition gilt:

$$M_{r,i,max} = \min(M_{1,i,max}, M_{2,max}) \tag{4.7}$$

Ebenso wie bei der Sequenz kann eine Transition des Merges nicht aufgrund eines Ereig-

nisses in der Figur deaktiviert werden. Demzufolge ist durch die in 4.1 gemachten Einschränkungen sichergestellt, dass keine der Transitionen jemals deaktiviert werden kann. Analog zur Sequenz spielt damit die race-policy der Transitionen des Merges keine Rolle, die Ersatz-Transitionen bekommen die race-enable Strategie.
Bei der Umformung der Merge-Figur repräsentieren alle Ersatz-Transitionen die Transition T_2. Die Summe aller Aktivierungsgrade der Ersatz-Transitionen entspricht der Anzahl an gleichzeitigen Laufzeiten in T_2, womit diese durch die Vielfachheit von T_2 begrenzt werden muss. Dies wird durch die globale Nutzungs-Bedingung sichergestellt:

$$B = M_{2,max} \tag{4.8}$$

Die Berechnung der Laufzeit einer Ersatz-Transition erfolgt wie bei der Sequenz durch Addition der originalen Laufzeiten :

$$t_{r,i} = t_{1,i} + t_2 \tag{4.9}$$

Bei der Überprüfung, ob auf die Ersatz-Transitionen geschaltet werden kann, muss die Ersetzungs-Bedingung die Summe aller Aktivierungsgrade der n Anfangs- und der mittleren Transitionen errechnen. Diese muss kleiner gleich der Vielfachheit von T_2 und kleiner gleich der Summe der Vielfachheiten der Ersatz-Transitionen sein, damit hochgeschaltet werden darf. Zusätzlich muss der Aktivierungsgrad einer Transition $T_{1,i}$ kleiner gleich der Vielfachheit der Ersatz-Transition $T_{r,i}$ sein. Beim Schalten an sich müssen die Restlaufzeiten von T_2 so über die Ersatz-Transitionen verteilt werden, dass nach dem Schaltvorgang Folgendes gilt: Der Aktivierungsgrad einer Ersatz-Transition muss kleiner gleich ihrer Vielfachheit sein.
Durch die globale Nutzungs-Bedingung der Merge-Figur ist es in einer Hierarchie von Sub-Netzen erlaubt, dass auch die Transitionen $T_{1,i}$ eine globale Nutzungs-Bedingung haben. Damit ergibt sich aber ein Problem mit dem oben erwähnten Vorgehen beim Hochschalten. Angenommen die Transitionen $T_{1,3}$ bis $T_{1,5}$ repräsentieren zusammen wieder ein Sub-Netz mit einer globalen Nutzungs-Bedingung mit einem Grenzwert von $B_{3,4,5} = 2$. Nach einem Hochschalten auf die Ersatz-Transitionen des Merges darf die Summe der Aktivierungsgrade von $T_{r,3}$ bis $T_{r,5}$ demnach maximal gleich zwei sein. Die Ersetzungs-Bedingung muss auf diese besonderen Randbedingungen Rücksicht nehmen. Um dennoch die Überprüfung relativ einfach zu halten, wird folgendes Verfahren verwendet.
Alle Anfangs-Transitionen müssen benutzt werden, damit auf die Ersatz-Transitionen des Merges geschaltet werden kann. Zu dem Augenblick des Schaltens sind demzufolge alle globalen Nutzungs-Bedingungen, die diese eventuell haben, erfüllt, da auf ein Hochschalten erst geprüft wird, wenn nicht mehr heruntergeschaltet werden muss. In der Merge-Figur werden die Anfangs-Transitionen in zwei Mengen aufgeteilt: G mit und L ohne globale Nutzungs-Bedingung. Von beiden Mengen wird die Summe der Aktivierungsgrade gebildet:

$$g = \sum_{T_{1,i} \in G} E_{T_{1,i}} \qquad\qquad l = \sum_{T_{1,i} \in L} E_{T_{1,i}} \tag{4.10}$$

Ist $g > 0$, so darf keiner der Aktivierungsgrade von T_2 auf die Transitionen der Menge G verteilt werden, da dann eine der globalen Nutzungs-Bedingungen nicht mehr erfüllt sein könnte. Ist $g = 0$, so kann zumindest einer der Aktivierungsgrade von T_2 auf diese Menge aufgeteilt werden, da ein Grenzwert kleiner eins bei einer globalen Nutzungs-Bedingung

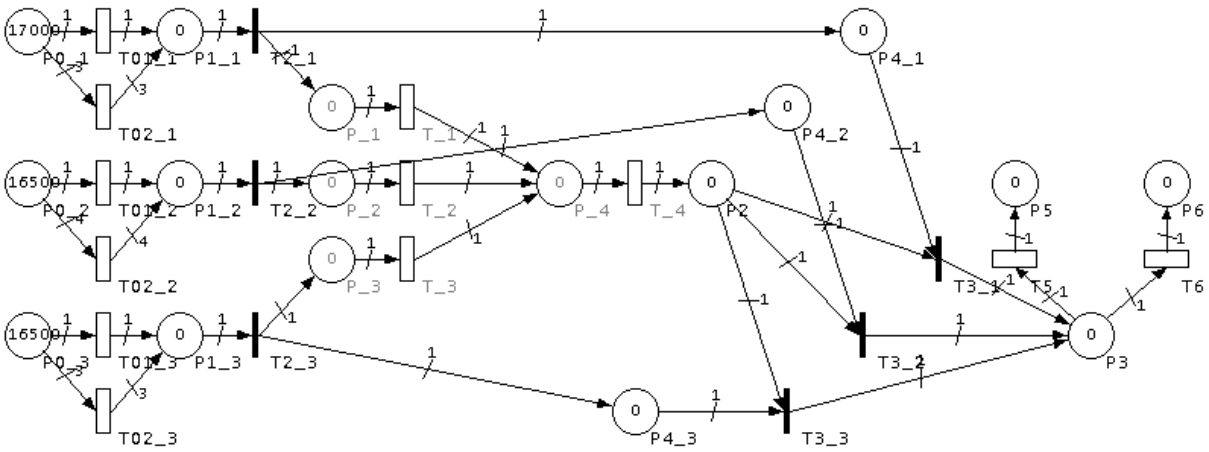


Abbildung 4.9: Das Petri-Netz, mit dem die Korrektheit überprüft wird

nicht vorkommen kann. Aus der Summe l lässt sich die Anzahl c der Aktivierungsgrade errechnen, die die Ersatz-Transitionen $T_{r,i}$ der Transitionen aus der Menge L maximal von T_2 aufnehmen können. Der Aktivierungsgrad von T_2 muss dann, damit hochgeschaltet werden kann, kleiner gleich c bzw. kleiner gleich $c + 1$ sein. Zusätzlich muss natürlich weiterhin gelten, dass die Summe der Aktivierungsgrade aller Transitionen des Merges kleiner gleich der Vielfachheit von T_2 und der Aktivierungsgrad einer Transition $T_{1,i}$ kleiner gleich der Vielfachheit der Ersatz-Transition $T_{r,i}$ ist.
Ein weiterer Vorteil dieser Aufteilung in zwei Mengen ist, dass bei der Überprüfung der hierarchischen Nutzungs-Bedingung der Ersatz-Transitionen des Merges lediglich die Nutzungs-Bedingungen der Transitionen in der Menge G kontrolliert werden müssen.

Zur Überprüfung der Korrektheit der eben vorgestellten Ersatz-Transitionen sowie der Schaltbedingungen der Merge-Figur wird das in Abbildung 4.9 dargestellte Petri-Netz verwendet. Die Multi-Server Transitionen T_i und ihre Eingangsstellen bilden eine umformbaren Merge-Figur, T_1 hat die race-age, T_3 race-repeat und T_2 und T_4 die race-enable Strategie.
Die Verteilungsfunktionen der einzelnen Transitionen werden so gewählt, dass von den Ersatz-Transitionen ab und zu, aber nicht zu oft, auf das Sub-Netz geschaltet werden muss: $V_f = 1 : 0.337$, $V_s = 1 : 0.152$. So wird gleichzeitig die Korrektheit der Berechnung der Laufzeiten sowie die des Schaltverhaltens überprüft. Die Ergebnisse werden über 40 Replikationen ermittelt und sind in Abbildung B.9 und Abbildung B.10 dargestellt. Die Unterschiede zwischen der adaptiven und der ,,normalen" Simulationsmethode sind so gering, dass sich aus dem Vergleich schließen lässt, dass die Ersatz-Transitionen das Verhalten desSub-Netzes korrekt imitieren.

Wie bei der Sequenz ist es auch bei der Merge-Figur sinnvoll, den mit Hilfe der adaptiven Simulation zu erzielenden Gewinn in Bezug auf die Schalthäufigkeit zu untersuchen. Die dafür verwendeten Petri-Netze entsprechen dem in Abbildung 4.10, in dem die Transitionen T0X_i sowie T_1-3 jeweils zusammen exakt 1500000 mal feuern, genauso wie die Transition T_4. Letztere und damit ein Drittel der Feuervorgänge wird durch die Verwen-

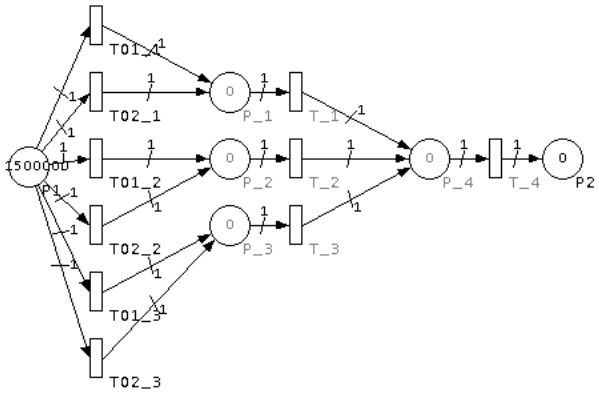


Abbildung 4.10: Petri-Netz für die Untersuchung des Zusammenhangs zwischen Gewinn und Schalthäufigkeit

Schalthäufigkeit	Rechenzeit		Gewinn		V_f	V_s	Zeit
	orig	adap	abs	%			Vorv.
nie	98.908	81.827	17.081	17.270	1 : 0.000	1 : 0.000	0.002
selten	128.034	118.782	9.252	7.226	1 : 0.568	1 : 0.371	0.001
mittel	131.194	134.742	-3.548	-2.704	1 : 2.360	1 : 0.967	0.001
oft	131.875	139.493	-7.618	-5.777	1 : 4.256	1 : 1.757	0.001
immer	98.041	112.438	-14.397	-14.685	0 : 1.000	0 : 1.000	0.001

Tabelle 4.4: Ergebnisse der Untersuchung des Zusammenhangs zwischen Gewinn und Schalthäufigkeit; alle Zeitangaben in Sekunden

dung der Ersatz-Transitionen eingespart. Allerdings ist die Überprüfung der Schaltbedingungen bei einem Merge aufwendiger als bei einer Sequenz, weshalb auch eine niedrigerer Gewinn zu erwarten ist. Insbesondere ist die Frage, bis zu welcher Schalthäufigkeit noch ein Gewinn an Rechenzeit zu erzielen ist, von Interesse. Die Ergebnisse in Tabelle 4.4 zeigen, dass die adaptive Simulation schneller als die herkömmliche Methode ist, wenn die Ersatz-Transitionen öfter feuern als von ihnen heruntergeschaltet wird. Werden mehr Schaltvorgänge notwendig, so ist die adaptive Simulation doch merklich langsamer.
Bei der Merge-Figur wächst der Aufwand für das Schalten wie auch für die Überprüfung der Schaltbedingungen mit der Anzahl n der Anfangs-Transitionen. Im Gegensatz dazu wird bei jedem Feuern einer Ersatz-Transition nur ein Feuervorgang eingespart, un-

n	Rechenzeit		Gewinn		V_f	V_s	Zeit
	orig	adap	abs	%			Vorv.
2	78.322	71.114	7.208	9.203	1 : 0.474	1 : 0.474	0.044
3	79.413	71.946	7.467	9.403	1 : 0.474	1 : 0.474	0.002
5	80.401	74.804	5.597	6.961	1 : 0.474	1 : 0.474	0.002
10	83.347	82.869	0.478	0.574	1 : 0.474	1 : 0.474	0.006
20	87.979	97.906	-9.927	-11.283	1 : 0.474	1 : 0.474	0.006

Tabelle 4.5: Ergebnisse der Untersuchung des Zusammenhangs zwischen Gewinn und Anzahl n von Anfangs-Transitionen in der Merge-Figur; alle Zeitangaben in Sekunden

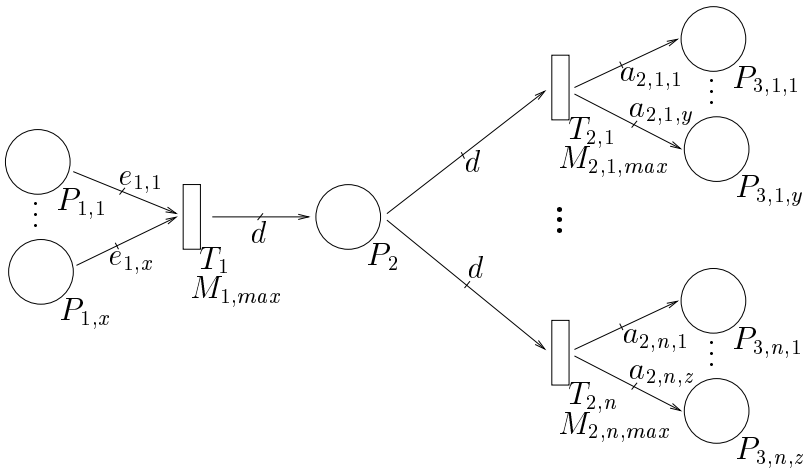


Abbildung 4.11: Ein allgemeiner, ersetzbarer Split

abhängig wie viele Anfangs-Transitionen der Merge hat. Demzufolge müsste mit steigender Anzahl an Anfangs- und somit Ersatz-Transitionen für den Merge der erzielte Gewinn an Rechenzeit sinken, bis die adaptive Simulation sogar langsamer wird. Dieser Zusammenhang soll in dem folgenden Experiment untersucht werden. In diesem wird die Anzahl an Anfangs-Transitionen des Merges variiert. Um dennoch die gleichen Rahmenbedingungen zu haben, befinden sich bei allen Versuchen immer gleich viele Transitionen und Stellen in dem Petri-Netz. Desweiteren werden alle Token in derselben Eingangsstelle des Merges erzeugt, damit ist immer dieselbe Anfangs- bzw. Ersatz-Transition aktiviert, während die anderen inaktiv sind. Aus dem vorherigen Versuch ist bereits bekannt, dass für eine Merge-Figur nur Gewinn erzielt werden kann, wenn relativ wenig geschaltet werden muss, deshalb sind die Verteilungsfunktionen so gewählt, dass $V_f = 0.474$ und $V_s = 0.474$ ist. Tabelle 4.5 zeigt die gemessenen Rechenzeiten. Es zeigt sich deutlich, dass mit steigender Anzahl an Anfangs-Transitionen der Gewinn sinkt, bis die adaptive Simulation sogar langsamer als die originale Methode wird. Aus diesen Ergebnissen ist klar erkennbar, dass sich die adaptive Simulation für eine Merge-Figur nur lohnt, wenn sie relativ wenig Anfangs-Transitionen besitzt.

4.4 Split

Abbildung 4.11 zeigt eine allgemeine, umformbare Split-Figur. Sie besteht aus einer Anfangs-Transition T_1 und n End-Transitionen $T_{2,i}$. Die Transition T_1 kann beliebig viele Eingangsstellen $P_{1,j}$ haben, mindestens jedoch eine. Sie hat exakt eine Ausgangsstelle P_2, von welcher aus n Kanten zu den n End-Transitionen weggehen. Diese dürfen ausschließlich P_2 als Eingangsstelle haben, können dafür mit beliebig vielen Ausgangskanten verbunden sein. In die Stelle P_2 darf nur exakt die eine Kante von T_1 hineinlaufen und auch nur die n Kanten zu den End-Transitionen hinaus. Die Eingangsstellen des Splits sind die Stellen $P_{1,j}$, von ihnen darf nur exakt eine Kante weggehen. Alle Transitionen sowie die Stellen $P_{1,j}$ und P_2 dürfen mit keiner inhibitorischen Kante verbunden sein.
Die Eingangskanten von T_1 sowie die Ausgangskanten von $T_{2,i}$ können jede beliebige Viel-

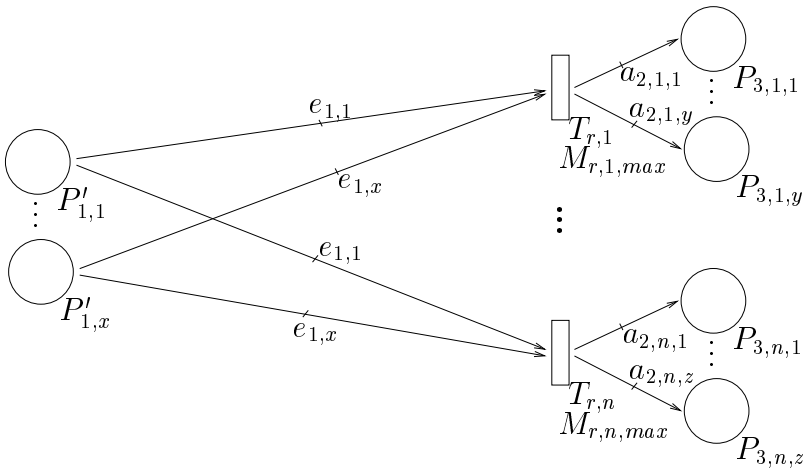


Abbildung 4.12: Der ersetzte Split

fachheit haben. Die Vielfachheit der Eingangskanten von $T_{2,j}$ muss gleich der Vielfachheit der Ausgangskante von T_1 sein. Ein Feuern der Transition T_1 aktiviert demzufolge jede der n End-Transitionen genau einmal. Die dadurch mögliche Umformung des Splits wird in Abbildung 4.12 dargestellt, der Split wird durch n Ersatz-Transitionen repräsentiert.
Eine Ersatz-Transition $T_{r,i}$ stellt die Transitionen T_1 und $T_{2,i}$ dar und hat demnach die gleichen Eingangskanten wie T_1 und die gleichen Ausgangskanten wie $T_{2,i}$, mit jeweils denselben Vielfachheiten. Allerdings ist die Vielfachheit einer Ersatz-Transition gleich eins:

$$M_{r,i,max} = 1 \tag{4.11}$$

Dies hat mehrere Gründe. Einer davon ist die Tatsache, dass die Feuerreihenfolge zweier Transitionen, die zum selben Zeitpunkt feuern, eine Rolle spielt. Angenommen der Split bestände aus zwei End-Transitionen $T_{2,1}$ und $T_{2,2}$ mit einer Vielfachheit von zehn und dem Aktivierungsgrad zwei und folgenden Laufzeiten : $t_{2,1,1} = 0$, $t_{2,1,2} = 5$, $t_{2,2,1} = 3$ und $t_{2,2,2} = 4$ und die Transition T_1 hätte eine Laufzeit von $t_{1,1} = 0$.
Unter diesen Bedingungen sind die beiden folgenden Varianten möglich. Wenn T_1 als erstes feuert, wird deren Laufzeit gelöscht und die End-Transitionen bekommen eine weitere Laufzeit dazu. Anschließend feuert $T_{2,1}$, deren Laufzeit $t_{2,1,1}$ wird ebenfalls gelöscht. Durch das Feuern von $T_{2,1}$ sinkt allerdings der Aktivierungsgrad von $T_{2,2}$, eine der Laufzeiten muss ebenfalls entfernt werden. Dabei wird zufällig eine aus den drei möglichen ausgewählt, wobei jede eine Entfernungswahrscheinlichkeit von $\frac{1}{3}$ hat, auch die gerade neu erzeugte.
In der zweiten Möglichkeit feuert von derselben Ausgangssituation aus $T_{2,1}$ als erstes. Dadurch muss eine der beiden Laufzeiten von $T_{2,2}$ entfernt werden, diesmal hat allerdings jede eine Wahrscheinlichkeit von $\frac{1}{2}$. Nach dem anschließenden Feuern von T_1 wird dann die neue Laufzeit erzeugt, die nun eine Entfernungswahrscheinlichkeit von null hat.
Aus diesem Beispiel ist ersichtlich, dass es eine Rolle spielt, in welcher Reihenfolge die Transitionen feuern. Dieses Verhalten wäre zwar mit den Ersatz-Transitionen ebenfalls nachbildbar, würde aber einen zusätzlichen, komplexeren Rechenschritt erfordern. Außerdem gibt es bei Multi-Server Ersatz-Transitionen noch ein weiteres Problem, welches von den verschiedenen möglichen race-policies für die End-Transitionen herrührt.

Angenommen, der Split wird benutzt und die End-Transitionen haben eine Laufzeit von $t_{2,1,1} = 3$ und $t_{2,2,1} = 5$ und die Transition $T_{2,2}$ habe die race-age Strategie. Zu diesem Zeitpunkt erhöht sich der Aktivierungsgrad von T_1 von null auf eins, es wird eine neue Laufzeit von $t_{1,1} = 10$ erzeugt. Anschließend feuert die Transition $T_{2,1}$ und die Restlaufzeit $t_{2,2,1} = 2$ wird gespeichert und nach dem Feuern von T_1 als neue Laufzeit wieder verwendet. Nun werden bei derselben Ausgangssituation die Ersatz-Transitionen benutzt. Deren Aktivierungsgrad steigt von eins auf zwei und es wird nicht heruntergeschaltet. Demzufolge wird sofort eine Laufzeit für T_1 bestimmt, aber auch eine für $T_{2,2}$. Diese wird neu erzeugt, obwohl in der originalen Figur die Restlaufzeit verwendet wird, das Verhalten ist damit nicht korrekt.
Aus diesem Grunde wäre eine höhere Vielfachheit für die Ersatz-Transitionen nur möglich, wenn als Strategie für die End-Transitionen lediglich race-enable zugelassen würde und selbst dann wäre ein zusätzlicher, komplexer Rechenschritt nötig. Deshalb wird die Vielfachheit von eins gewählt.
Durch die Tatsache, dass alle Ersatz-Transitionen des Splits dieselben Eingangsstellen haben, haben sie stets denselben Aktivierungsgrad. In Verbindung mit der Vielfachheit eins der Ersatz-Transitionen ist damit kein Grenzwert B erforderlich. Außerdem ist der Aktivierungsgrad der Transition T_1 gleich dem der Ersatz-Transitionen und nicht die Summe davon wie beim Merge.
Im Gegensatz zur Sequenz und zum Merge kann eine Ersatz-Transition des Splits aber deaktiviert werden. Dank der in 4.1 gemachten Einschränkungen kann dies allerdings nur passieren, wenn eine andere der Ersatz-Transitionen gefeuert hat. Die Laufzeit von T_1 ist demnach zu diesem Zeitpunkt stets abgelaufen. Eine Ersatz-Transition $T_{r,i}$ muss bei einer Deaktivierung demzufolge lediglich die Laufzeit der Sub-Netz-Transition $T_{1,i}$ entsprechend der Strategie von $T_{1,i}$ behandeln. Damit ist auch das korrekte Verhalten des Splits bezüglich der einzelnen race-policies sichergestellt.
Die Berechnung der Laufzeit einer Ersatz-Transition erfolgt dann einfach durch Addition,

$$t_{r,i} = t_1 + t_{1,i} \tag{4.12}$$

wobei beachtet werden muss, dass die Laufzeit t_1 für die Transition T_1 nur einmal erzeugt werden darf.
Bei der Überprüfung, ob hochgeschaltet werden darf, muss lediglich überprüft werden, ob die Aktivierungsgrade von T_1 und einer der End-Transitionen zusammen kleiner gleich eins sind. Ist dies der Fall, darf auf die Ersatz-Transitionen geschaltet werden.

Durch das folgenden Experiment soll gezeigt werden, dass die Schaltbedingungen ebenso wie der Aufbau der Ersatz-Transitionen für eine Split-Figur korrekt sind. Abbildung 4.13 zeigt das dafür verwendete Petri-Netz, indem die Multi-Server Transitionen T_i einen ersetzbaren Split bilden. Desweiteren wird für die Transitionen T_1 und T_3 die race-enable, für T_2 race-age und für T_4 die race-repeat Strategie ausgesucht. Die verschiedenen Verteilungsfunktionen sind so gewählt, dass von den Ersatz-Transition ab und zu auf das Sub-Netz heruntergeschaltet werden muss: $V_f = 1 : 0.296$, $V_s = 1 : 0.284$.
Die Ergebnisse werden über 40 Replikationen ermittelt und sind in Abbildung B.11 und Abbildung B.12 dargestellt. Beim Vergleich der Statistiken fällt auf, dass überhaupt keine Abweichung vorhanden ist. Dies liegt an der Tatsache, dass die Ersatz-Transitionen die

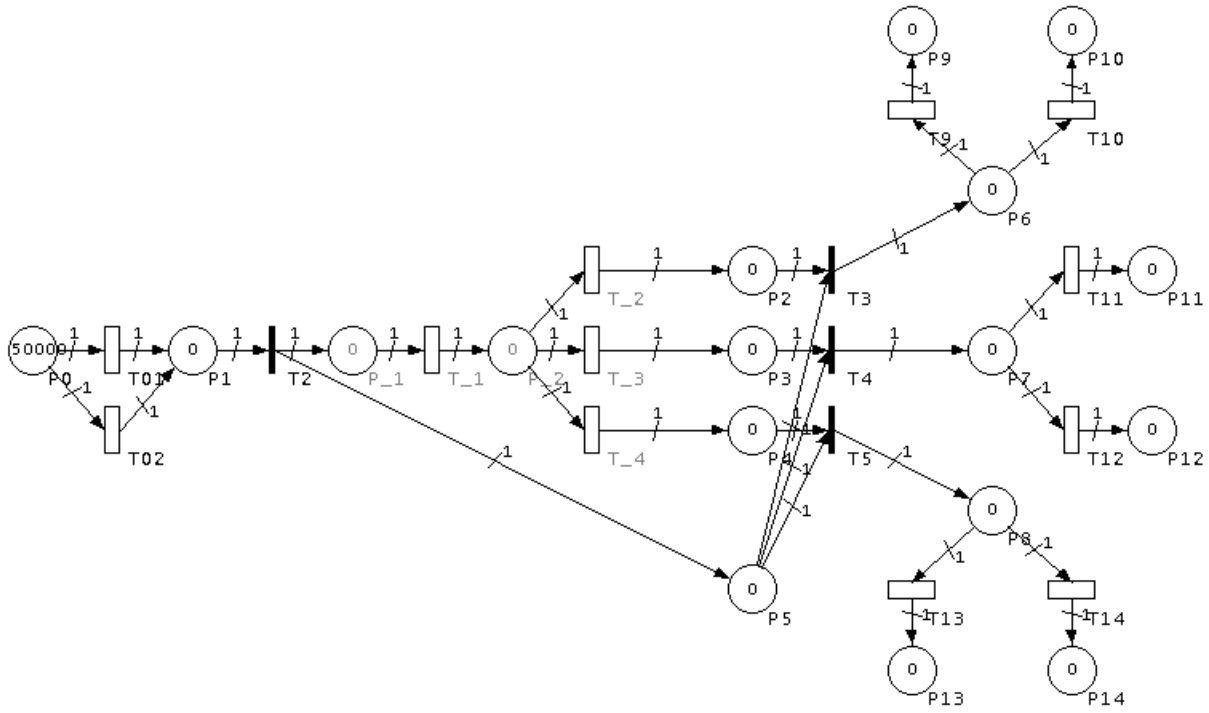


Abbildung 4.13: Das Petri-Netz, mit dem die Korrektheit überprüft wird

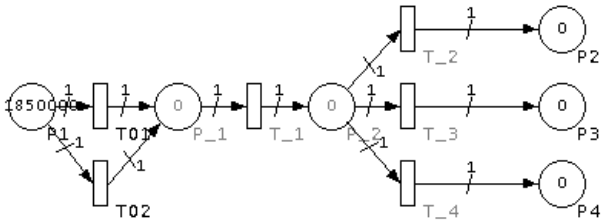


Abbildung 4.14: Petri-Netz für die Untersuchung des Zusammenhangs zwischen Gewinn und Schalthäufigkeit

Vielfachheit von eins haben und damit keine Unterschiede in den Verweildaurern der Token in dem Split zwischen adaptiver und ,,normaler" Simulation entstehen können. Die Gegenüberstellung zeigt eindeutig, dass die Ersatz-Transitionen das Verhalten der Split-Figur korrekt nachbilden.

Auch bei der Split-Figur wird der zu erwartende Gewinn durch die Häufigkeit, wie oft von den Ersatz-Transitionen heruntergeschaltet werden muss, beeinflusst. In dem folgenden Experiment wird dieser Zusammenhang untersucht und versucht, die Frage zu klären, bis zu welcher Häufigkeit die adaptive Simulation für die Split-Figur einen Gewinn an Rechenzeit bringt. Die dabei verwendeten Petri-Netze entsprechen dem in Abbildung 4.14, in dem durch das Verwenden der Ersatz-Transitionen ein Drittel der Feuervorgänge eingespart wird. Die in Tabelle 4.6 enthaltenen Ergebnisse zeigen, dass die adaptive Simulation schneller als die herkömmliche Methode ist, solange die Ersatz-Transitionen nicht viel sel-

Schalthäufigkeit	Rechenzeit		Gewinn		V_f	V_s	Zeit
	orig	adap	abs	%			Vorv.
nie	96.496	82.136	14.360	14.881	1 : 0.000	1 : 0.000	0.001
selten	119.752	113.087	6.665	5.566	1 : 0.410	1 : 0.386	0.001
mittel	121.726	118.092	3.634	2.985	1 : 0.962	1 : 0.878	0.001
oft	123.265	126.451	-3.186	-2.585	1 : 2.047	1 : 1.812	0.001
immer	97.539	125.424	-27.885	-28.589	0 : 1.000	0 : 1.000	0.001

Tabelle 4.6: Ergebnisse der Untersuchung des Zusammenhangs zwischen Gewinn und Schalthäufigkeit; alle Zeitangaben in Sekunden

n	Rechenzeit		Gewinn		V_f	V_s	Zeit
	orig	adap	abs	%			Vorv.
2	84.600	76.358	8.242	9.742	1 : 0.607	1 : 0.607	0.009
3	85.239	77.741	7.498	8.796	1 : 0.569	1 : 0.569	0.002
5	86.606	80.505	6.101	7.045	1 : 0.526	1 : 0.526	0.002
10	87.651	87.617	0.034	0.039	1 : 0.483	1 : 0.483	0.007
20	85.567	99.270	-13.703	-16.014	1 : 0.457	1 : 0.457	0.006

Tabelle 4.7: Ergebnisse der Untersuchung des Zusammenhangs zwischen Gewinn und Anzahl n von End-Transitionen in der Split-Figur; alle Zeitangaben in Sekunden

tener feuern als von ihnen heruntergeschaltet wird.
Wie bei der Merge-Figur nimmt auch bei dem Split der Aufwand für das Schalten sowie das Überprüfen der Schaltbedingungen mit der Anzahl n an End-Transitionen zu, während bei jedem Feuern einer Ersatz-Transition nur ein Feuervorgang eingespart wird. Demzufolge müsste der Gewinn mit der Anzahl an End-Transitionen abnehmen. Dieser Einfluss soll in dem folgenden Versuch überprüft werden, bei dem die Anzahl an End-Transitionen variiert wird. In dem dabei verwendeten Petri-Netzen befinden sich insgesamt immer gleich viele Transitionen und Stellen, es sind immer gleich viele aktiviert und es gibt immer gleich viele Feuervorgänge. Dadurch sind die gleichen Rahmenbedingungen sichergestellt. Trotzdem ändern sich die Kennzahlen V_f und V_s mit der Anzahl an End-Transitionen. Allerdings nimmt die Anzahl an Schaltungen im Vergleich zu dem Feuern der Ersatz-Transitionen mit steigendem n ab, V_s wird kleiner, was aber eher einen beschleunigenden Effekt hat. Dennoch ist in den Ergebnissen in Tabelle 4.7 ein deutlicher Rückgang des erzielten Gewinns zu erkennen bis die adaptive Simulation sogar langsamer als die herkömmliche Methode wird. Demzufolge lohnt sich die adaptive Simulation für eine Split-Figur nur, wenn diese relativ wenig End-Transitionen besitzt.

4.5 Parallel-Transitionen

Eine besondere Figur sind die Parallel-Transitionen, die in Abbildung 4.15 zu sehen sind. Sie bestehen aus n Transitionen T_i, die alle dieselbe Eingangsstelle P_1 und nur diese haben. Ebenso besitzen sie alle dieselbe Ausgangsstelle P_2 und nur diese. Von der Eingangsstelle dürfen ausschließlich die Kanten zu den n Transitionen weggehen und keine der Transitionen darf mit einer inhibitorischen Kante verbunden sein.
Die Parallel-Transitionen werden durch eine Ersatz-Transition T_r dargestellt, welche in

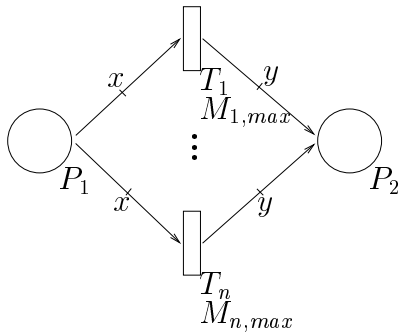


Abbildung 4.15: Allgemeine, ersetzbare Parallel-Transitionen

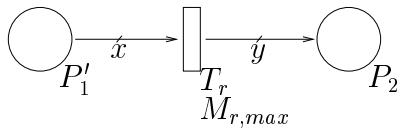


Abbildung 4.16: Die ersetzten Parallel-Transitionen

Abbildung 4.16 zu sehen ist. Durch die Tatsache, dass alle ersetzten Transitionen parallel sind und keine Folge von Transitionen enthalten ist, wird durch T_r kein Feuervorgang eingespart: Ein Feuern von T_r entspricht nur einem Feuern im Sub-Netz. Dafür nimmt bei der Benutzung von T_r die Anzahl an insgesamt aktivierten Transitionen ab. Dadurch ist der Aufwand für einen Feuervorgang während T_r aktiviert ist etwas geringer als im Sub-Netz. Allerdings ist fraglich, ob die so gesparte Rechenzeit den zusätzlichen Aufwand für die adaptive Simulation ausgleichen kann, denn oft bestehen Parallel-Transitionen nur aus zwei oder drei Transitionen, die Einsparung ist in diesem Fall sehr gering. Betrachtet man die Figur aber in einer Hierarchie, so ist auch in dieser Situation eine Umformung sinnvoll, da die in den vorherigen Abschnitten behandelten Topologien meistens eine Transition mit exakt einer Eingangs- oder Ausgangskante benötigen. Sollten an dieser Stelle im originalen Petri-Netz mehrere Parallel-Transitionen stehen, so ist bisher eine Transformation nicht möglich. Werden aber zuerst die Parallel-Transitionen durch eine Transition mit einer Eingangs- und einer Ausgangsstelle ersetzt, kann anschließend auch die entsprechende Figur umgeformt werden.
Durch die Tatsache, dass immer wenn eine der Parallel-Transitionen feuern würde auch die Ersatz-Transition feuert, ändert sich die Statistik der Eingangsstelle nicht, sie muss demzufolge nicht als uninteressant markiert sein und von ihr darf eine inhibitorische Kante ausgehen. Dafür dürfen die Parallel-Transitionen, obwohl sie alle End-Transitionen sind, keine Reset-Liste haben: Da jedes Feuern der Ersatz-Transition für das Feuern einer anderen der Parallel-Transitionen stehen kann, müsste die Ersatz-Transition jedesmal eine andere Reset-Liste haben. Ebenso darf keine Kostenfunktion bei dem Feuern einer der Parallel-Transitionen ausgewertet werden.
Alle Eingangskanten der Parallel-Transitionen T_i müssen dieselbe Vielfachheit besitzen. Würden sie dies nicht, so könnte die Ersatz-Transition T_r das korrekte Verhalten nur mit Hilfe einer Eingangskante mit Funktions-abhängiger Vielfachheit nachbilden. Eine anschließende Ersetzung von T_r wäre dadurch aber nicht mehr möglich. Aus dem gleichen

Grund müssen auch alle Ausgangskanten dieselbe Vielfachheit haben.
Die Parallel-Transitionen werden durch eine Ersatz-Transition T_r dargestellt. Sie hat exakt eine Eingangskante von P_1 mit genau der Vielfachheit der Eingangskanten und eine Ausgangskante zu P_2 mit derselben Vielfachheit wie die Ausgangskanten der Parallel-Transitionen. Für ihre eigene Vielfachheit gilt

$$M_{r,max} = 1. \tag{4.13}$$

Angenommen zwei Parallel-Transitionen werden benutzt, haben eine Vielfachheit von zehn, den Aktivierungsgrad eins und folgende Laufzeiten : $t_{1,1} = 5$ und $t_{2,1} = 7$. In dieser Situation würde in fünf Zeiteinheiten die nächste der Parallel-Transitionen feuern. Nun erhöhe sich der Aktivierungsgrad auf zwei. Es werden folgende neue Laufzeiten generiert: $t_{1,2} = 2$ und $t_{2,2} = 3$. Nach zwei Zeiteinheiten feuert demnach T_1, zu diesem Zeitpunkt haben die Transitionen folgende Restlaufzeiten : $t_{1,1} = 3$, $t_{1,2} = 0$, $t_{2,1} = 5$ sowie $t_{2,2} = 1$. Es wird die Laufzeit $t_{1,2}$ entfernt und auch von T_2 wird eine der Laufzeiten zufällig ausgewählt und gelöscht, in dem Beispiel sei dies $t_{2,1} = 5$. Nach dem Feuern haben die Transition demzufolge die Laufzeiten $t_{1,1} = 3$ und $t_{2,1} = 1$ und als nächstes feuert T_2 nach einer Zeiteinheit. Von dem Anfangszeitpunkt des Beispiels aus gesehen, haben die Parallel-Transitionen damit zu den Zeitpunkten 2 und 3 gefeuert und nicht nach fünf Zeiteinheiten, wie es ursprünglich den Anschein hatte. Für eine Ersatz-Transition T_r mit einer höhern Vielfachheit als eins bedeutet dies, dass sich bereits berechnete Laufzeiten ändern können. Dadurch kann T_r aber nicht mehr ersetzt werden, da sich in dem in dieser Arbeit vorgestelltem Konzept der adaptiven Simulation eine einmal berechnete Laufzeit aus dem Sub-Netz nicht mehr ändern darf. Aus diesem Grund darf die Ersatz-Transition der Parallel-Transitionen nur eine Vielfachheit von eins haben.
Eine neue Laufzeit von T_r wird dann folgendermaßen berechnet:

$$t_r = \min(t_i); \qquad i = 1, \ldots, n \tag{4.14}$$

Bei der Überprüfung ob hochgeschaltet werden kann, muss die Ersetzungs-Bedingung lediglich kontrollieren, ob der Aktivierungsgrad der Parallel-Transitionen kleiner gleich eins ist. Dies entspricht der Abfrage, ob in der Eingangsstelle weniger als $2 \cdot x$ Token enthalten sind. Ist dies der Fall, so darf auf die Ersatz-Transition geschaltet werden.

Durch die besondere Situation der Parallel-Transitionen dürfte sich die Statistik des Petri-Netzes durch die Ersatz-Transition T_r nicht ändern (abgesehen von den Parallel-Transitionen selbst), was durch das folgende Experiment gezeigt werden soll. Das dafür verwendete Petri-Netz ist in Abbildung 4.17 zu sehen. Die Multi-Server Transition T_1 hat in diesem die race-age, T_3 die race-repeat, T_2 und T_4 die race-enable Strategie. Wieder werden die Verteilungsfunktionen so gewählt, dass ab und zu auf das Sub-Netz geschaltet werden muss: $V_f = 1 : 0.292$, $V_s = 1 : 0.275$.
Anhand des in Abbildung B.13 und Abbildung B.14 dargestellten Vergleichs der Statistiken ist eindeutig erkennbar, dass die Ersatz-Transition das Verhalten der Parallel-Transitionen korrekt nachbildet.

In dem folgenden Experiment wird der zu erwartende Gewinn bzw. Verlust in Anhängigkeit von der Schalthäufigkeit untersucht. In dem dabei verwendeten Petri-Netz aus Abbildung 4.18 befinden sich wenig Transitionen, der durch die Ersatz-Transition gesparte

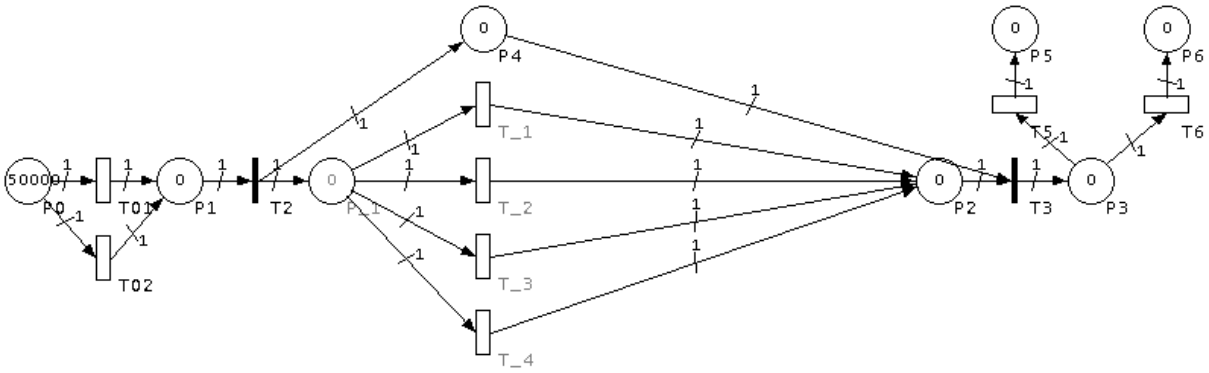


Abbildung 4.17: Das Petri-Netz, mit dem die Korrektheit überprüft wird

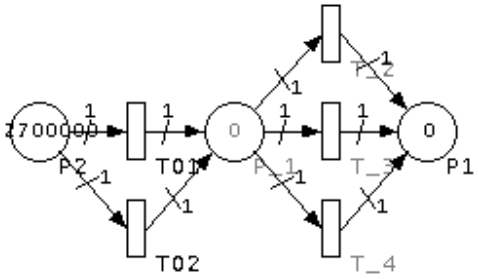


Abbildung 4.18: Petri-Netz für die Untersuchung des Zusammenhangs zwischen Gewinn und Schalthäufigkeit

Aufwand ist demzufolge sehr gering. Aus diesem Grund ist es wahrscheinlich, dass die adaptive Simulation für dieses Petri-Netz selbst dann langsamer ist, wenn wenig geschaltet werden muss. Die Ergebnisse in Tabelle 4.8 bestätigen dies. Desweiteren ist ein Anstieg des Verlustes mit wachsender Schalthäufigkeit zu erkennen.
Durch die Parallel-Transitionen wird die Rechenzeit für das Ändern der Laufzeiten sowie das Auswählen einer der aktivierten Transitionen verkürzt. Mit steigender Anzahl n an Parallel-Transitionen müsste demnach der Verlust sinken, bis irgendwann sogar ein Gewinn erzielt werden könnte. In dem dazugehörigen Versuch wird ein Petri-Netz verwendet, in dem immer gleich viele Transitionen enthalten sind. Es wird jeweils die Anzahl

Schalthäufigkeit	Rechenzeit		Gewinn		V_f	V_s	Zeit
	orig	adap	abs	%			Vorv.
nie	96.357	101.842	-5.485	-5.692	1 : 0.000	1 : 0.000	0.001
selten	121.539	129.072	-7.533	-6.198	1 : 0.440	1 : 0.388	0.001
mittel	124.584	134.471	-9.887	-7.936	1 : 1.172	1 : 0.948	<0.001
oft	130.081	140.352	-10.271	-7.896	1 : 3.301	1 : 1.887	0.001
immer	93.101	105.661	-12.560	-13.491	0 : 1.000	0 : 1.000	0.001

Tabelle 4.8: Ergebnisse der Untersuchung des Zusammenhangs zwischen Gewinn und Schalthäufigkeit; alle Zeitangaben in Sekunden

n	Rechenzeit		Gewinn		V_f	V_s	Zeit
	orig	adap	abs	%			Vorv.
2	54.487	56.022	-1.535	-2.817	1 : 0.429	1 : 0.429	0.001
3	55.598	56.270	-0.672	-1.209	1 : 0.429	1 : 0.429	0.001
5	57.733	55.679	2.054	3.558	1 : 0.429	1 : 0.429	0.001
10	60.990	53.539	7.451	12.217	1 : 0.429	1 : 0.429	0.001
20	61.456	44.835	16.621	27.045	1 : 0.429	1 : 0.429	0.002

Tabelle 4.9: Ergebnisse der Untersuchung des Zusammenhangs zwischen Gewinn und Anzahl n von Parallel-Transitionen; alle Zeitangaben in Sekunden

der Parallel-Transitionen verändert, es sind aber immer gleich viele Transitionen aktiviert und die Anzahl der Feuervorgänge bleibt ebenfalls konstant. In Tabelle 4.9 sind die gemessenen Rechenzeiten enthalten. Es zeigt sich deutlich eine Verringerung des Verlustes bei wachsendem n, bei einem n größer gleich fünf wird sogar ein Gewinn erzielt. Aus diesen Ergebnissen lässt sich schließen, dass die Verwendung einer Ersatz-Transition für die Parallel-Transitionen bei großen n sowie einer großen Anzahl an aktivierten Transitionen eine Beschleunigung verspricht.

4.6 Hierarchische Experimente

In diesem Abschnitt soll gezeigt werden, dass sich die in 3.3 und 3.4 entwickelte adaptive Simulation für eine Hierarchie von Figuren entsprechend dem originalen Petri-Netz verhält. Außerdem wird auf den Gewinn an Rechenzeit eingegangen.

In Abbildung 4.19 ist das Petri-Netz dargestellt, dass zur Überprüfung der Korrektheit verwendet wird. Es stellt ein System einer Fabrik mit zwei Händlern sowie den Haushalten als Abnehmern dar. Im Zuge des Recycling wird der Abfall der Haushalte wieder als Rohstoff zur Produktion genommen, wodurch ein geschlossener Kreislauf entsteht. In dem Petri-Netz wird die Möglichkeit der hierarchischen Modellierung verwendet, bei der durch ein bestimmtes Symbol, hier ein Achteck, ein komplettes Sub-Petri-Netz dargestellt wird. Demzufolge steht das Achteck „Fabrik" für das Petri-Netz der Produktionsanlage, welches in Abbildung 4.20 zu sehen ist. Diese Anlage ist so aufgebaut, dass eine komplett ersetzbare Hierarchie von Sub-Netzen entsteht. Dabei werden verschiedene Multi-Server und race-policies verwendet und die einzelnen Wahrscheinlichkeitsverteilungen der Transitionen sind so gewählt, dass jedes Sub-Netz mindestens einmal verwendet werden muss, meistens aber die Ersatz-Transitionen oder deren Ersatz-Transitionen benutzt werden können. Damit wird sowohl die Korrektheit der Berechnung von Laufzeiten als auch die der Schaltvorgänge in einer Hierarchie überprüft.
Der Vergleich der Ergebnisse ist in Abbildung B.15 und Abbildung B.16 dargestellt. Die Unterschiede sind dabei so gering, dass sich aus dem Versuch schließen lässt, dass die adaptive Simulation auch für eine Hierarchie von Sub-Netzen das Verhalten des originalen Petri-Netzes korrekt nachbildet.

In der Betrachtung der einzelnen Topologien wurde deutlich, dass der mit Hilfe der adaptiven Simulation zu erzielende Gewinn an Rechenzeit immer von dem jeweiligen Petri-Netz

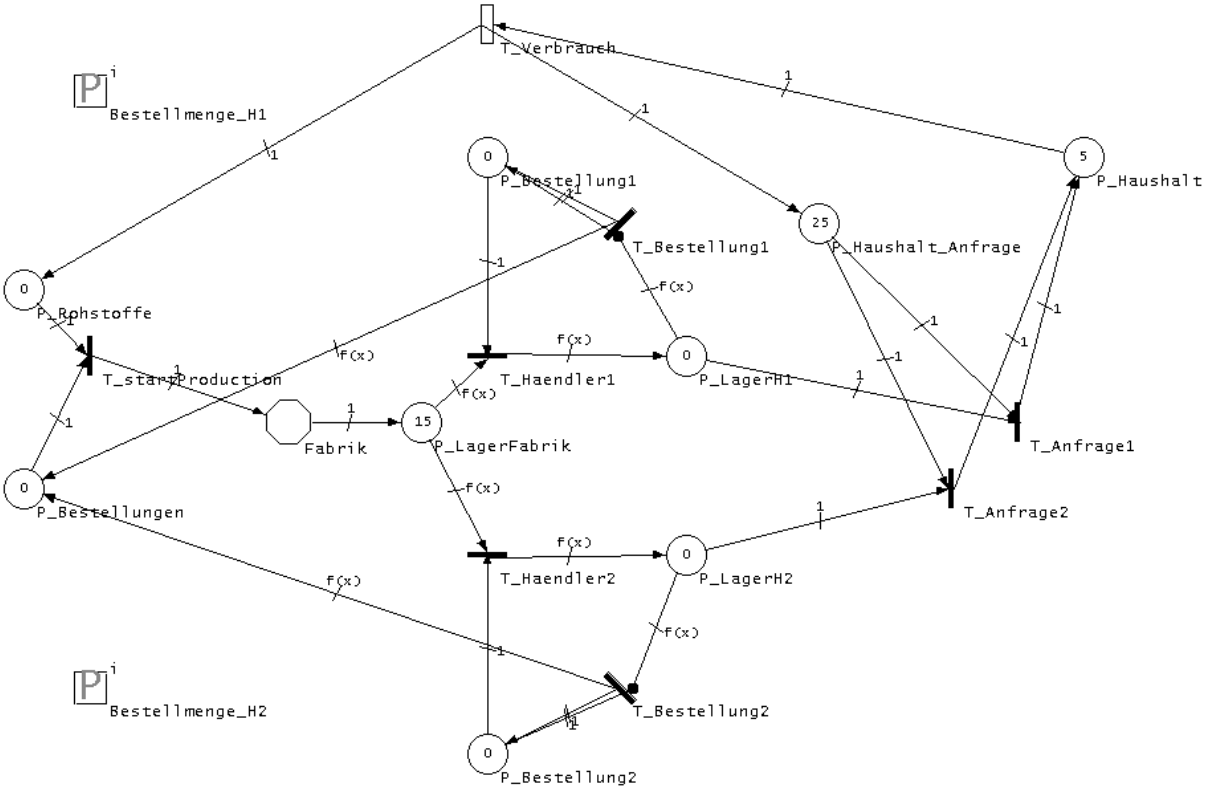


Abbildung 4.19: Ein Petri-Netz, dass eine Hierarchie von Sub-Netzen enthält

Rechenzeit	original	482.082
	adaptiv	315.180
Gewinn	absolut	166.902
	%	34.621
Rechenzeit Vorv.		0.011

Tabelle 4.10: Vergleich der Rechenzeit für die herkömmliche, originale Simulationsmethode und die adaptive Simulation; alle Zeitangaben in Sekunden

abhängt. Aus dieser Erkenntnis lässt sich auch für ein komplexeres Petri-Netz mit mehreren ersetzbaren Figuren oder einer Hierarchie von Sub-Netzen schließen, dass der zu erzielende Gewinn immer mit den Eigenschaften der einzelnen Topologien zusammenhängt. In Tabelle 4.10 sind die Rechenzeiten für die adaptive Simulation sowie für die originale Methode für das Petri-Netz, welches auch zur Überprüfung der Korrektheit verwendet wurde, enthalten. Aus ihnen geht klar hervor, dass mit Hilfe der adaptiven Simulation eine Beschleunigung erzielt werden konnte. Dieses Ergebniss zeigt, dass die adaptive Simulation auch für komplexere Petri-Netze eine deutliche Verringerung der benötigten Rechenzeit bewirken kann.

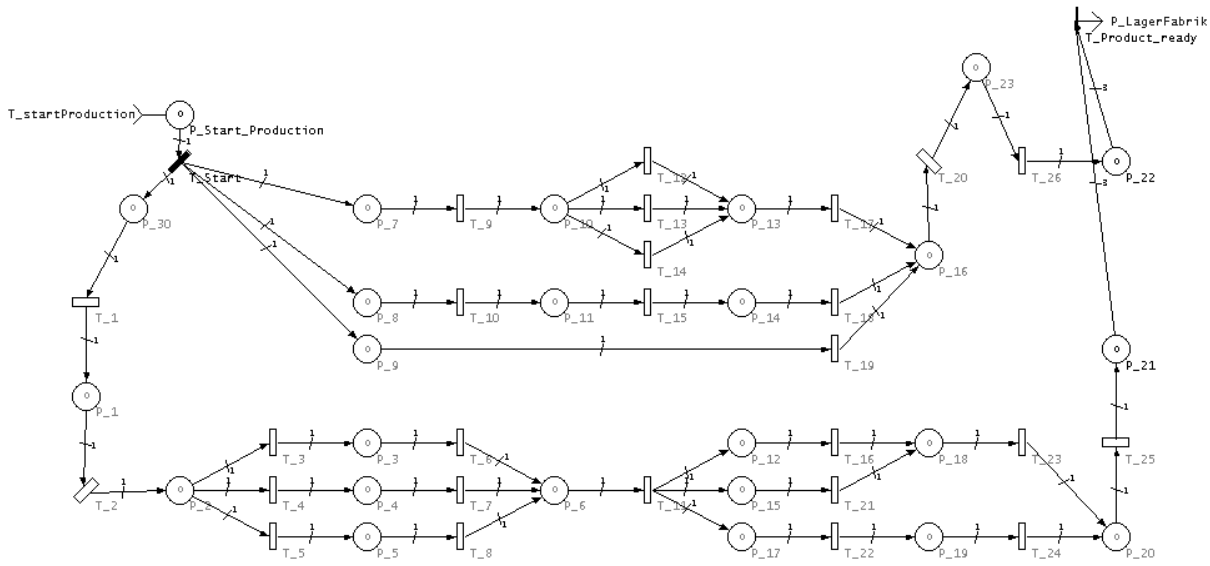


Abbildung 4.20: Die komplett ersetzbare Hierarchie von Sub-Netzen

Kapitel 5

Lernverhalten

In Kapitel 4 wurde deutlich, dass bei einem zu häufigen Schalten zwischen Ersatz-Transitionen und Sub-Netz die für die adaptive Simulation benötigte Rechenzeit höher werden kann als bei der herkömmlichen, in 2.4 erklärten Simulation. Daraus entsteht der Wunsch, während der Simulation erkennen zu können, ob sich das Schalten für eine bestimmte Figur lohnt oder nicht. In diesem Kapitel wird ein Ansatz vorgestellt, der genau dies mit Hilfe von bereits existierenden Lernverfahren ermöglicht. Anschließend wird kurz das in dieser Arbeit verwendete Lernverfahren vorgestellt und der Einfluss des Lernens auf den mit Hilfe der adaptiven Simulation erzielten Gewinn an Rechenzeit untersucht.

5.1 Konzept, Idee

Die adaptive Simulation führt zu einem immer wiederkehrenden Wechsel zwischen Ersatz-Transitionen und Sub-Netz. Nimmt die Anzahl an solchen Schaltvorgängen im Vergleich zu dem Feuern der Ersatz-Transitionen zu, so sinkt der durch diese Simulationsmethode erzielte Zeitgewinn. Ab einem bestimmten Verhältnis ist die adaptive Simulation sogar langsamer als die herkömliche, in 2.4 behandelte Simulation. Leider ist aus dem Petri-Netz selbst nur sehr schwer zu erkennen, ob bei einer bestimmten Figur die adaptive Simulation einen Gewinn an Rechenzeit bringt. Demzufolge wäre es sinnvoll, während der Simulation selbst aus dem Schalt- und Feuerverhalten einer Topologie erkennen zu können, ob sich die adaptive Simulation für sie lohnt oder nicht.
In dem Bereich der Informatik existieren bereits verschiedene, allgemeine Lernverfahren, wie z. B. das Reinforcement Learning [SB98], die für diese Aufgabe mehr oder weniger geeignet sind. Daher wird in diesem Abschnitt kein neues Verfahren entwickelt, sondern eine Methode vorgestellt, die es ermöglicht, die bereits vorhandenen verwenden zu können. Um die Einbettung dieser Verfahren möglichst einfach zu halten, wird den Ersatz-Transitionen eines jeden Sub-Netzes ein sogenannter *Entscheider* zugeordnet. Dieser soll während der Simulation erlernen, ob ein Hochschalten auf die Ersatz-Transitionen sinvoll ist. Dieses Lernziel ist hier vollkommen ausreichend: Werden die Ersatz-Transitionen benutzt und es muss heruntergeschaltet werden, so kann auf diesen Schaltvorgang nicht verzichtet werden. Desweiteren ist es unsinnig, von den Ersatz-Transitionen auf das Sub-Netz zu schalten, wenn dies nicht notwendig ist, da die Benutzung der Ersatz-Transitionen ja gerade den Gewinn bei der adaptiven Simulation erbringt. Von dem Sub-Netz auf die Ersatz-Transitionen darf wiederum nur geschaltet werden, wenn es die Ersetzungs-Bedingung erlaubt.

Die Aufgabe des Entscheiders ist demnach folgende: Wird das Sub-Netz benutzt und die Ersetzungs-Bedingung ermöglicht ein Hochschalten, so muss der Entscheider wählen, ob dieser Schaltschritt erfolgen soll oder nicht. Diese Entscheidung trifft er aufgrund des bisherigen Schalt- und Feuerverhaltens der entsprechenden Figur. Im Laufe einer Replikation kann sich dieses Verhalten ändern, so dass ein Hochschalten einmal sinnvoll sein kann und einmal nicht. Aus diesem Grund muss der Entscheider bei jedem möglichen Hochschaltvorgang neu festlegen, ob dieser erfolgen soll oder nicht.
Damit der Entscheider diese Aufgabe erfüllen kann, muss er das Schalt- und Feuerverhalten „seiner" Ersatz-Transitionen und des Sub-Netzes kennen, jeder Schaltvorgang und jedes Feuern muss ihm mitgeteilt werden. Für den Entscheider sind also folgende Situationen von Interesse:

1. Das Sub-Netz wird benutzt und eine der End-Transitionen feuert.
2. Die Ersatz-Transitionen werden benutzt und eine von ihnen feuert.
3. Es wird von den Ersatz-Transitionen auf das Sub-Netz geschaltet.
4. Es wird vom Sub-Netz auf die Ersatz-Transitionen geschaltet.

Aufgrund einer möglichen Hierarchie von Sub-Netzen kann es passieren, dass die Ersatz-Transitionen des Entscheiders wiederum ersetzt werden. Diese Ersatz-Transitionen der Ersatz-Transitionen werden im Folgenden auch als *Super-Transitionen* bezeichnet. Dadurch erweitern sich aber auch die für den Entscheider interessanten Ereignisse:

5. Die Super-Transitionen werden benutzt und eine von ihnen feuert.
6. Es wird von den Ersatz-Transitionen auf die Super-Transitionen geschaltet.
7. Es wird von den Super-Transitionen auf die Ersatz-Transitionen geschaltet.

Der Rechenaufwand für das Schalten auf die Super-Transitionen und umgekehrt ist allerdings für den Entscheider nicht von Bedeutung, da es die Aufgabe des Entscheiders der Super-Transitionen ist, festzustellen ob sich dieses Schalten lohnt oder nicht. Dennoch sind diese Vorgänge für den Entscheider wichtig, da durch die Super-Transitionen folgendes Problem entstehen kann.
Angenommen, eine Sequenz besteht aus drei Transitionen T_1, T_2 und T_3, sie wird benutzt und T_3 ist einmal aktiviert. Außerdem ist T_2 wiederum eine Ersatz-Transition mit einem Entscheider D_2. Nun wird auf die Ersatz-Transition T_r der Sequenz geschaltet. Dabei werden die Token in der Eingangsstelle von T_3 entfernt und neue Token in den Eingangsstellen der Sequenz erzeugt. Anschließend feuert T_r, für den Entscheider D_2 bedeutet dies, dass die Super-Transition gefeuert hat. Allerdings hat der für dieses Feuern verantwortliche Aktivierungsgrad die Transition T_2 schon einmal vor dem Schalten auf T_r zum Feuern veranlaßt. Der Entscheider D_2 beachtet demzufolge dieses Feuern, dass beim Schalten quasi durch T_2 hindurchgeschoben wurde, zweimal.
Um solche mehrfache Berücksichtigung eines Feuervorgangs zu vermeiden, erhält der Entscheider einen Zähler F_s, welcher besagt, wie viele Feuervorgänge bei dem Schalten auf die Super-Transitionen durch die Ersatz-Transitionen hindurchgeschoben wurden. Bei einem

anschließenden Feuern einer der Super-Transitionen kann er aus diesem Zähler schließen, ob dieser Feuervorgang schon einmal beachtet wurde.
In der obrigen Beispiel-Sequenz ist zum Zeitpunkt des Schaltens auf T_r die Transition T_3 aktiviert. Beim darauffolgenden Feuern von T_r wird demzufolge weniger Rechenzeit eingespart, als wenn T_r bereits bei der Aktivierung der Sequenz benutzt worden wäre. Diese Information kann aber für den Entscheider von T_r von Bedeutung sein. Deshalb wird ein weiterer Zähler F_r eingeführt, der die Anzahl an Laufzeiten , die die Sub-Netz-Transitionen zum Zeitpunkt des Hochschaltens hatten, enthält.
Auch der umgekehrte Fall, wenn zum Zeitpunkt der Aktivierung der Figur die Ersatz-Transitionen benutzt werden, aber vor dem Feuern heruntergeschaltet wird und anschließend das Sub-Netz feuert, ist eventuell für den Entscheider von Interesse. Diese Information bekommt der Entscheider von einem dritten Zähler F_n, der die Anzahl an Laufzeiten aller Ersatz-Transitionen zum Zeitpunkt des Herunterschaltens beinhaltet.
Dank dieser Zähler erhält der Entscheider alle Informationen, die er zum Erkennen, ob sich die adaptive Simulation für seine Figur lohnt oder nicht, benötigt. Hierbei kann er ein beliebiges Lernverfahren verwenden. Der Umgang mit den Zählern erfolgt dabei nach folgenden Regeln, in denen die abstrakte Simulationszeit, die von der Aktivierung einer Figur bis zu dem letzten dadurch ermöglichten Feuern einer End-Transition vergeht, als *Figur-Laufzeit* bezeichnet wird.

Situation 1: Das Sub-Netz feuert

Hier gilt immer $F_r = 0$ und $F_s = 0$.
Ist $F_n > 0$, so wurden während einem Teil der Figur-Laufzeit die Ersatz-Transitionen benutzt. Der neue Wert des Zählers ist $F_n = F_n - 1$.
Ist $F_n = 0$, so wurde das Sub-Netz während der gesamten Figur-Laufzeit benutzt.

Situation 2: Eine der Ersatz-Transitionen feuert

In dieser Situation ist $F_n = 0$ und $F_s = 0$.
Ist $F_r > 0$, so wurde das Sub-Netz während der Figur-Laufzeit teilweise verwendet, der neue Wert ist $F_r = F_r - 1$.
Bei $F_r = 0$ wurden die Ersatz-Transitionen während der gesamten Figur-Laufzeit benutzt.

Situation 3: Ersatz-Transitionen → Sub-Netz

$F_r = 0$, $F_s = 0$, $F_n =$ Anzahl an Laufzeiten der Ersatz-Transitionen.

Situation 4: Sub-Netz → Ersatz-Transitionen

$F_n = 0$, $F_s = 0$, $F_r =$ Anzahl an Laufzeiten der Sub-Netz-Transitionen.

Situation 5: Eine der Super-Transitionen feuert

$F_n = 0$ ist in diesem Fall immer gegeben.
Ist $F_s > 0$, so hat der Entscheider diesen Feuervorgang schon einmal beachtet. Für den neuen Wert gilt $F_s = F_s - 1$.

Für $F_s = 0$ hat der Entscheider diesen Feuervorgang bisher nicht beachtet. Es gibt folgende zwei Möglichkeiten:

$F_r > 0$ bedeutet, dass während der Figur-Laufzeit das Sub-Netz zumindest teilweise verwendet wurde, der Zähler wird um eins erniedrigt: $F_r = F_r - 1$.

$F_r = 0$ bedeutet, dass die ganze Figur-Laufzeit durch die Ersatz- bzw. die Super-Transitionen benutzt wurden.

Situation 6: Ersatz-Transitionen → Super-Transitionen

$F_n = 0$, F_r bleibt unverändert und F_s = Anzahl an Feuervorgängen, die durch alle Ersatz-Transitionen des Entscheiders ,,hindurchgeschoben" werden.

Situation 7: Super-Transitionen → Ersatz-Transitionen

F_m sei die Anzahl an Feuervorgängen, die beim Schalten durch die Ersatz-Transitionen geschoben werden. Dabei gilt immer $F_m \geq F_s$.
Von den F_m Feuervorgängen hat der Entscheider F_s schon einmal beachtet.
Von den restlichen $F_m - F_s$ Feuervorgängen aber erfährt der Entscheider zum ersten Mal. Für jeden von ihnen gilt dabei wieder, dass wenn $F_r > 0$ das Sub-Netz während der Figur-Laufzeit zumindest teilweise verwendet wurde, bei $F_r = 0$ dagegen die Ersatz-Transitionen die ganze Figur-Laufzeit durch benutzt wurden.
Der Wert von F_r muss entsprechend geändert werden, die restlichen Zähler sind $F_n = 0$ und $F_s = 0$.

5.2 Informationsfluß

In dem vorherigen Abschnitt wurden die für einen Entscheider interessanten Situationen behandelt. In diesem wird auf die Frage eingegangen, welcher Entscheider bei einem Feuer- oder Schaltvorgang in einer Hierarchie von Sub-Netzen informiert werden muss. Um dabei nicht extra auf ,,normale" Transitionen getrennt eingehen zu müssen, erhalten auch diese einen Entscheider, der nichts lernen muss und immer ein Hochschalten erlaubt.

Eine Transition T_f feuert. Damit müssen alle Entscheider, bei denen entweder T_f oder eine ihrer mehrfachen Ersatz-Transitionen eine End-Transition ist, darüber informiert werden, dass ihr Sub-Netz gefeuert hat. Mehrfache Ersatz-Transitionen sind dabei die Ersatz-Transitionen der Ersatz-Transitionen der Ersatz-Transitionen usw. .
Desweiteren hat für alle Entscheider der Sub-Netz-Transitionen von T_f die Super-Transition gefeuert. Dies gilt auch für die Transitionen, die zu dem Feuern nichts beigetragen haben. Dies ist deshalb sinnvoll, da z. B. beim Merge Folgendes der Fall sein kann. Eine Transition T_i der n Transitionen beim Merge ist wieder eine Ersatz-Transition. Für sich allein betrachtet, ist die adaptive Simulation für diese Transition langsamer als die normale. Allerdings wird T_i nur sehr selten aktiviert, während der Merge sehr oft aktiviert ist und beim Merge sich die adaptive Simulation lohnt. Würde der Entscheider von T_i nicht mitbekommen, dass sich der Merge rentiert, würde er stets das Sub-Netz von T_i als benutzt wählen. Damit könnten aber auch die Ersatz-Transitionen des Merges nicht mehr benutzt werden. Bei der Verteilung der Information, dass die Super-Transition gefeuert

hat, ist außerdem zu beachten, dass alle Ersatz-Transitionen eines Sub-Netzes immer nur einen Entscheider haben und die Nachricht jedem Entscheider nur einmal mitgeteilt werden darf.

Die Entscheider der Sub-Netz-Transitionen können auf diese Mitteilung verschieden reagieren. Hat der Entscheider einer Sub-Netz-Transition T_n von diesem Feuern zum ersten Mal erfahren, kann den Entscheidern der Sub-Netz-Transitionen von T_n ebenfalls mitgeteilt werden, dass ihre Super-Transition gefeuert hat. Anders ist die Situation, wenn der Entscheider von T_n dieses Feuern schon einmal beachtet hat. In diesem Fall haben auch alle Entscheider der Sub-Netz-Transitionen von T_n dieses Feuern bereits beachtet. Es besteht also die Möglichkeit, diesen keine Information zukommen zu lassen. Alternativ dazu kann den Entscheidern der Sub-Netz-Transitionen von T_n auch mitgeteilt werden, dass ein Feuervorgang ihre Super-Transition ,,ein zweites Mal" durchlaufen hat. Da diese Information vielleicht für einige Lernverfahren von Interesse sein kann, wurde die zweite Alternative gewählt. Damit ergibt sich eine weitere interessante Situation für einen Entscheider:

8. Ein Feuervorgang hat die SuperTransition ,,ein zweites Mal" durchlaufen.

Bekommt ein Entscheider diese Information, so beeinflußt dies nicht die Werte der in 5.1 eingeführten Zähler. Desweiteren teilt er wiederum den Entscheidern seiner Sub-Netz-Transitionen mit, dass ein Feuervorgang deren Super-Transition ,,ein zweites Mal" durchlaufen hat.

Nach dem auf das Feuern einer Transition in einer Hierarchie von Sub-Netzen eingegangen wurde, wird zunächst der Hochschaltvorgang betrachtet. Von dem Sub-Netz N_i wird auf die Ersatz-Transitionen T_k geschaltet. Für die Entscheider der Transitionen in dem Sub-Netz N_i bedeutet dies, dass von den Ersatz- auf die Super-Transitionen und für den Entscheider der Ersatz-Transitionen T_k, dass von dem Sub-Netz auf die Ersatz-Transitionen geschaltet wird. Für alle anderen Entscheider ist dieser Vorgang nicht von Interesse.

Als letztes wird das Herunterschalten von den Ersatz-Transitionen T_k auf das Sub-Netz N_i behandelt. Für alle Entscheider, deren Transitionen in der Hierarchie über den Ersatz-Transitionen liegen, also mehrfach Ersatz-Transitionen von T_k sind, ist dieser Vorgang uninteressant. Für den Entscheider der Transitionen T_k wurde von den Ersatz-Transitionen auf das Sub-Netz geschaltet, für die Entscheider der Transitionen des Sub-Netzes N_i von den Super-Transitionen auf die Ersatz-Transitionen. Letzteres bewirkt aber, dass durch die Transitionen des Sub-Netzes N_i Feuervorgänge geschoben werden, entweder zum ersten oder zum zweiten Mal. Demzufolge müssen die Entscheider der Transitionen des Sub-Netzes N_i wiederum entsprechend oft den Entscheidern ihrer Sub-Netz-Transitionen mitteilen, dass deren Super-Transition gefeuert oder dass ein Feuervorgang deren Super-Transition ,,ein zweites Mal" durchlaufen hat.

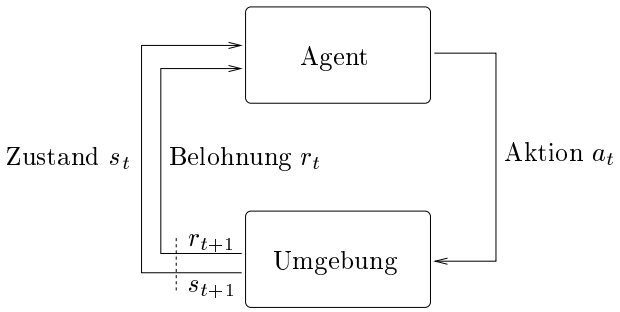


Abbildung 5.1: Interaktion zwischen Agent und Umgebung

5.3 Experimente

In diesem Abschnitt wird kurz das verwendete Lernverfahren vorgestellt und der Einfluss des Lernens auf den Zeitgewinn der adaptiven Simulation beschrieben.

In dieser Arbeit wird das Sarsa-Verfahren des Reinforcement Learning [SB98] als Lernverfahren verwendet. Beim Reinforcement Learning interagiert zu diskreten Zeitschritten t ein Agent mit einer Umgebung. Zu jedem dieser Zeitschritte bekommt der Agent von der Umgebung eine Information über den Zustand $s_t \in S$, indem sie sich gerade befindet, S ist die Menge aller Zustände. Daraufhin wählt der Agent eine Aktion a_t aus der Menge der in diesem Zustand möglichen Aktionen $A(s_t)$ aus und teilt diese der Umgebung mit. Im nächsten Zeitschritt erhält der Agent die Information über den Folgezustand s_{t+1} und die damit verbundene Belohnung r_{t+1}. Abbildung 5.1 zeigt das System von Agent und Umgebung.
Der Agent versucht über lange Zeit gesehen eine insgesamt möglichst maximale Belohnung zu erhalten. Dabei wird zwischen episodischen und kontinuierlichen Problemen unterschieden. Bei episodischen gibt es einen Endzustand, nach dessen Erreichen die Umgebung wieder in einen Startzustand übergeht. Das Ziel des Agenten ist es demzufolge, eine möglichst hohe Gesamtbelohnung in einer Episode zu bekommen. Dagegen laufen kontinuierliche Probleme quasi endlos, der Agent versucht, seine zukünftigen Belohnungen zu maximieren.
Der Agent wählt die Aktion a_t nur aufgrund des aktuellen Zustandes s_t aus. Alle Zustände der Umgebung müssen deshalb Markov-Zustände sein, d.h. sie müssen die Markov Eigenschaft haben. Diese besagt, dass die Wahrscheinlichkeit des Folgezustandes s_{t+1} und der damit verbundenen Belohnung r_{t+1} unter der ausschließlichen Beachtung von s_t und a_t die gleiche sein muss, wie wenn alle bisherigen Zustände und Aktionen beachtet werden würden. Hierbei ist es sinnvoll, auch Zustände, die nicht exakt die Markov Eigenschaft erfüllen, näherungsweise als Markov-Zustände zu betrachten.
In dem verwendeten Sarsa-Verfahren hat jedes Zustands-Aktions-Paar einen Wert $Q(s, a)$, welcher besagt, wie „gut" die Aktion a im Zustand s ist. Der Agent wählt in dem Zustand s_t die Aktion a_t aufgrund einer ϵ-greedy Strategie aus. Das bedeutet, die Aktion mit dem höchsten Zustands-Aktionswert erhält die Wahrscheinlichkeit $1 - \epsilon + \frac{\epsilon}{|A(s_t)|}$ und alle anderen $\frac{\epsilon}{|A(s_t)|}$, wobei $|A(s_t)|$ die Anzahl an möglichen Aktionen im Zustand s_t ist. Im nächsten Zeitschritt erhält der Agent die Belohnung r_{t+1} und wählt ebenfalls nach der

Initialisiere $Q(s,a)$ beliebig

Wiederhole für jede Episode

Initialisiere s

Wähle Aktion a bzgl. s mittels der ϵ-greedy Strategie aus

Wiederhole für jeden Zeitschritt

Führe Aktion a aus, beobachte Belohnung r und Folgezustand s'

Wähle a' bzgl. s' mittels der ϵ-greedy Strategie aus

$Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma Q(s',a') - Q(s,a)]$

$s \leftarrow s'$; $a \leftarrow a'$;

bis s der Endzustand ist

Abbildung 5.2: Der verwendete Lern-Algorithmus: Sarsa – Ein ,,On-Policy Temporal Difference Control" - Algorithmus

ϵ-greedy Strategie die neue Aktion a_{t+1} aus. Anschließend wird der Wert des ,,letzten" Zustands-Aktions-Paares verändert:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \tag{5.1}$$

Der sich daraus ergebende Algorithmus ist in Abbildung 5.2 dargestellt. Für tiefergehende Informationen über Reinforcement Learning und den Sarsa-Algorithmus, sowie warum dieser korrekt lernt, wird der Leser auf [SB98] verwiesen.
In der adaptiven Simulation ist das Lernen, ob sich ein Schaltvorgang lohnt oder nicht, ein Zusatz zu der Grundidee, welche zur Beschleunigung der Simulation gedacht ist. Aus diesem Grund darf das Lernen an sich keinen hohen Aufwand benötigen, da ansonsten der Rechenaufwand für das Lernen den Gewinn des Erkennens von sich lohnenden Schaltvorgängen wieder zunichte macht. Für den verwendeten Sarsa-Algorithmus bedeutet dies, dass die Anzahl der Zustands-Aktions-Paare möglichst klein gehalten werden muss. Die gewählten Zustände sowie die darin möglichen Aktionen und daraus resultierende Folgezustände und Belohnungen sind in Tabelle 5.1 dargestellt. Um Zustands-Aktions-Paare zu sparen, wird auf die gesonderte Betrachtung, ob das Sub-Netz oder die Ersatz-Transitionen die gesamte Figur-Laufzeit durch benutzt wurden, verzichtet. Der Folgezustand ,,Ersatz" auf das Zustands-Aktions-Paar (Ersatz, Nichts) kann außer dem Feuern einer Ersatz-Transition auch bedeuten, dass eine Super-Transition gefeuert hat und der Entscheider von diesem Feuern zum ersten Mal erfährt.
Die Schalthäufigkeit und damit die Antwort auf die Frage, ob sich ein Schaltvorgang lohnt oder nicht, kann sich bei der adaptiven Simulation über die Simulationszeit verändern. Damit ist aber der Wert des entsprechenden Zustands-Aktions-Paares am Ende einer Replikation nicht der optimale Wert für den Beginn der nächsten. Daher wird das Problem des Lernens bei der adaptiven Simulation als kontinuierliches Problem betrachtet und der Wert aller Zustands-Aktions-Paare am Anfang einer jeden Replikation neu initialisiert.

Zustand	Aktion	Folgezustand	Belohnung	Kommentar
Ersatz	Nichts	Ersatz	100	Eine Ersatz-Transition hat gefeuert.
		Sub-Netz-Aktiv	-10	Es wurde Heruntergeschaltet.
Sub-Netz-Aktiv	Nichts	Sub-Netz-Aktiv	0	Eine End-Transition des Sub-Netzes hat gefeuert.
		Sub-Netz-Wahl	0	Eine End-Transition des Sub-Netzes hat gefeuert und es kann Hochgeschaltet werden.
Sub-Netz-Wahl	Bleiben	Sub-Netz-Aktiv	0	Es wurde nicht geschaltet.
	Schalten	Ersatz	-10	Es wurde Hochgeschaltet.

Tabelle 5.1: Zustands-Aktions-Paare mit möglichen Folgezuständen und Belohnungen

	Rechenzeit
original	96.496
adaptiv immer schaltend	82.136
adaptiv lernend	83.230

Tabelle 5.2: Abschätzung des Aufwandes für das Lernen; alle Zeitangaben in Sekunden

Dabei erhält jedes Paar den Wert 0 bis auf das Zustands-Aktions-Paar (Sub-Netz-Wahl, Schalten), welches den Wert 100 erhält. Dies geschieht, damit der Entscheider zu Beginn einer Replikation möglichst oft schaltet, um festzustellen, ob sich ein Schaltvorgang lohnt oder nicht. Die restlichen Parameter sind folgendermaßen gewählt: $\alpha = 0.2$, $\gamma = 0.8$ und $\epsilon = 0.1$.

Das Lernen, ob sich das Hochschalten lohnt oder nicht, benötigt zusätzlichen Rechenaufwand. Muss nie von den Ersatz-Transitionen heruntergeschaltet werden, so verlangsamt dieser Aufwand die adaptive Simulation. Der Vergleich der adaptiven Simulation einmal mit Lernen und einmal ohne für ein solches Petri-Netz zeigt demzufolge den für das Lernen benötigten Aufwand. Für diesen Versuch wird das Petri-Netz der Split-Figur aus 4.4 verwendet, bei dem nie von den Ersatz-Transitionen heruntergeschaltet werden muss. Die Rechenzeiten in Tabelle 5.2 zeigen, dass die adaptive Simulation mit einem lernenden Entscheider nur minimal langsamer ist, als wenn immer geschaltet wird. Der Aufwand für das Lernen ist demnach durchaus vertretbar.
Der Vorteil des Lernens zeigt sich, wenn man ein Petri-Netz betrachtet, bei dem sich die Antwort auf die Frage, ob sich ein Hochschalten lohnt oder nicht, immer wieder während der Simulationszeit ändert. In dem folgenden Experiment wird ein solches Petri-Netz verwendet. Es enthält eine ersetzbare Split-Figur, bei der sich das Hochschalten während der Simulationszeit $20000 \cdot k \leq \tau \leq 20000 \cdot k + 7530$ mit $k = 1, 2, \ldots$ nicht rentiert, da nach einem Hochschalten immer sofort wieder heruntergeschaltet werden muss. Zu den restlichen Zeiten jedoch lohnt sich das Schalten auf die Ersatz-Transitionen, da nur selten von ihnen wieder auf das Sub-Netz geschaltet werden muss. Außerdem befinden sich in dem Petri-Netz noch 50 inaktive Transitionen, die eine gewisse Größe des Petri-Netzes simulieren sollen. Wird bei der adaptiven Simulation dieses Petri-Netzes immer geschaltet, so ist die adaptive Simulation langsamer als die originale Simulationsmethode,

		original	adaptiv	
			immer schaltend	lernend
# Feuervorgängen	Sub-Netz	—	453449	459379
	Ersatz-Transitionen	—	187051	181121
# Herunterschaltungen		—	452406	34929
V_f		—	1 : 2.424	1 : 2.536
V_s		—	1 : 2.419	1 : 0.193
Rechenzeit		103.836	107.032	99.689
Gewinn	absolut	—	-3.196	4.147
	%	—	-3.078	3.994
Rechenzeit Vorv.		—	0.002	0.002

Tabelle 5.3: Vergleich der adaptiven Simulation mit und ohne Lernen; alle Zeitangaben in Sekunden

wie in Tabelle 5.3 zu sehen ist. Wird allerdings der nach dem Sarsa-Verfahren lernende Entscheider verwendet, so erkennt dieser die Zeitabschnitte, in denen sich ein Schalten lohnt bzw. nicht lohnt. Aus den Ergebnissen in Tabelle 5.3 geht hervor, dass die Anzahl an Schaltvorgängen mit der lernenden adaptiven Simulation deutlich geringer ist, als bei der immer schaltenden. Die Anzahl an Feuervorgängen der Ersatz-Transitionen ist dagegen fast gleich geblieben, woraus sich schließen lässt, dass der Entscheider korrekt erkennt, wann sich das Hochschalten lohnt und wann nicht. Dadurch ist die lernende adaptive Simulation für dieses Petri-Netz schneller als die herkömmliche Methode. Im Vergleich zu der immer schaltenden adaptiven Simulation ist sie sogar um sieben Prozent in Bezug auf die originale Rechenzeit schneller. Dieses Resultat zeigt, dass die Verwendung eines lernenden Entscheiders durchaus sinnvoll ist.

Kapitel 6

Berechnung von Feuerzeitpunkten

Während der Implementierung der adaptiven Simulation entstand eine neue Idee für die Beschleunigung der Simulation. Anstatt aus den Laufzeiten der Sub-Netz-Transitionen die Laufzeiten der Ersatz-Transitionen zu berechenen, ist es unter bestimmten Umständen möglich, gleich die Zeitpunkte zu ermitteln, bei denen eine der End-Transitionen des Sub-Netzes feuern würde. Aus diesen lassen sich dann einfach die Laufzeiten der Ersatz-Transitionen errechnen. Der Vorteil dieser Methode ist es, dass die Berechnung der Feuerzeitpunkte gleich den gegenseitigen Einfluss verschiedener Laufzeiten berücksichtigen kann. Demzufolge wird ein Herunterschalten von den Ersatz-Transitionen unnötig. In diesem Kapitel soll untersucht werden, inwieweit eine solche Berechnung möglich und mit welchem Aufwand sie verbunden ist und welcher Gewinn sich mit der neuen Idee erzielen lässt.

6.1 Konzept, Idee

Bei einer Single-Server Sequenz, wie sie in Abbildung 6.1 zu sehen ist, ist es möglich, gleich bei der Aktivierung der ersten Transition den Zeitpunkt zu berechnen, zu dem die letzte Transition der Sequenz feuern wird. Zur Simulationszeit $\tau = 10$ ist die Transition T_2 aktiviert und hat eine Restlaufzeit von $t_{2,1} = 2$ und feuert demzufolge zu dem Zeitpunkt $\tau = 12$. Desweiteren wird in P_1 ein neuer Token erzeugt und damit T_1 aktiviert, sie bekommt eine Laufzeit von $t_{1,1} = 1$ und feuert zu dem Zeitpunkt $\tau = 11$. Zu dieser Simulationszeit hat die zweite Transition noch eine Restlaufzeit, sie feuert erst bei $\tau = 12$, weshalb die neue Laufzeit von $t_{2,2} = 3$ erst bei $\tau = 12$ zu „laufen" beginnt. Es ist also bereits zum Zeitpunkt der Aktivierung der ersten Transition T_1 der Zeitpunkt $\tau = 15$ berechenbar, bei dem die End-Transition der Sequenz feuern wird.
In der Abbildung 6.1 steht $t_{2,2} = 3|15$. Dies bedeutet, dass die Transition T_2 eine Laufzeit

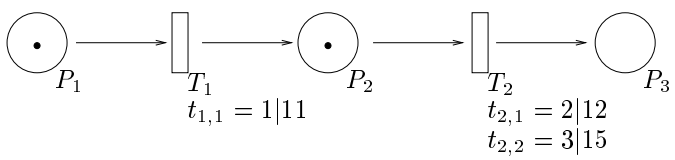


Abbildung 6.1: Eine Single-Server Sequenz, $\tau = 10$

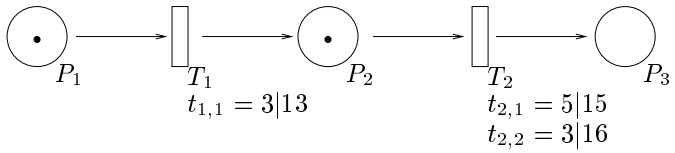


Abbildung 6.2: Eine Multi-Server-Sequenz, alle Transitionen haben die Vielfachheit $M_{i,max} = 2$; $\tau = 10$

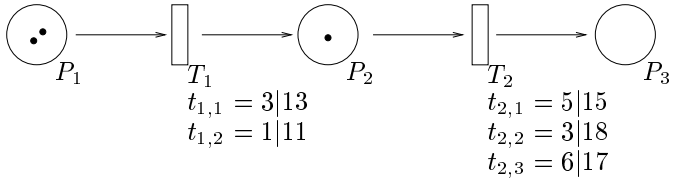


Abbildung 6.3: Die Multi-Server-Sequenz nach dem Erzeugen eines neuen Token in P_1, alle Transitionen haben die Vielfachheit $M_{i,max} = 2$; $\tau = 10$

von $t_{2,2} = 3$ hat, welche die Transition in der derzeitigen Konstellation zum Zeitpunkt $\tau = 15$ zum Feuern veranlassen wird.

Diese Tatsache führt zu der Idee, gleich bei der Aktivierung einer Topologie die Zeitpunkte zu berechnen, bei denen die End-Transitionen feuern werden und aus diesen wiederum die Laufzeiten für die Ersatz-Transitionen zu bestimmen. Somit lässt sich eine Topologie, bei der eine solche Berechnung möglich ist, durch Infinite-Server Ersatz-Transitionen darstellen, von denen im Gegensatz zur adaptiven Simulation nie heruntergeschaltet werden muss. Im Folgenden wird untersucht, welche Einschränkungen eine Topologie erfüllen muss, damit eine Ermittlung der Feuerzeitpunkte möglich ist.

Ein Problem ergibt sich bereits bei der Anwendung dieser Idee auf eine Multi-Server Sequenz. In Abbildung 6.2 wird eine Sequenz gezeigt, bei der jede Transition die Vielfachheit $M_{i,max} = 2$ hat. Zu dem dargestellten Zeitpunkt ist die Sequenz zweimal aktiviert, die Feuerzeitpunkte von T_2 sind bereits berechnet. Allerdings wird nun ein zweiter Token in der Eingangsstelle P_1 der Sequenz erzeugt. Für diesen werden sofort die Laufzeiten beider Transitionen der Sequenz bestimmt: $t_{1,2} = 1$ und $t_{2,3} = 6$. Die Transition T_1 hat zum Zeitpunkt der Aktivierung nur eine Laufzeit, die neue fängt demnach sofort an zu ,,laufen" und T_1 feuert zu der Zeit $\tau = 11$. Zu dieser Simulationszeit hat aber auch T_2 nur eine Laufzeit, da $t_{2,2}$ erst zu einer Gesamtzeit von $\tau = 13$ zu ,,laufen" beginnt. Demzufolge kann die Laufzeit $t_{2,3}$ sofort ,,loslaufen". Dies verursacht aber, dass zum Zeitpunkt $\tau = 13$, an dem bisher $t_{2,2}$ gleich beginnen konnte, T_2 bereits zwei Laufzeiten besitzt. Damit muss $t_{2,2}$ bis zum nächsten Feuern von T_2 warten und die alte berechnete Feuerzeit von T_2 von $\tau = 16$ ist nicht mehr korrekt. Stattdessen feuert T_2 nun zu den Zeitpunkten $\tau = 15$, $\tau = 17$ und $\tau = 18$.

Aus diesem Beispiel ist ersichtlich, dass es bei der Verwendung von Multi-Server-Transitionen zu Situationen kommen kann, bei denen nicht alle der bisher berechneten Feuerzeitpunkte der End-Transitionen korrekt sind. Die inkorrekten Zeiten müssen gelöscht und neu berechnet werden. Betrachtet man aber eine große Topologie mit vielen Transitionen, so kann das Löschen und neu Berechnen von Feuerzeiten einen beträchtlichen Aufwand mit sich bringen, weshalb die neue Idee nur auf Single-Server-Topologien angewendet

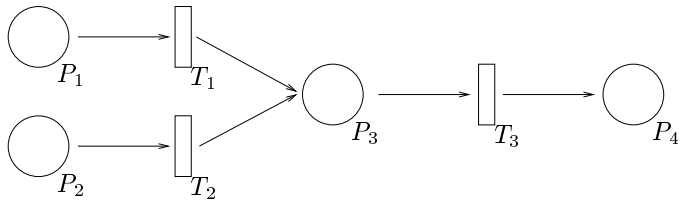


Abbildung 6.4: Diese Topologie kann nicht mit der neuen Idee transformiert werden

werden darf.
Die Bedingung, dass einmal berechnete Feuerzeitpunkte der End-Transitionen immer korrekt bleiben müssen, bedeutet aber auch, dass keine Transition der Topolgie von außen, d.h. aus Gründen, die nicht direkt aus der Topologie selbst ersichtlich sind, deaktiviert werden darf. Demzufolge müssen für alle Kanten, Stellen und Transitionen der Topologie die in 4.1 eingeführten Einschränkungen gelten.
Der hier vorgestellte, neue Ansatz basiert auf der Idee, dass man in einer Topologie nach und nach für alle Transitionen aus dem Aktivierungszeitpunkt, den zu diesem Zeitpunkt vorhandenen Laufzeiten sowie einer neu erzeugten, den neu hinzukommenden Feuerzeitpunkt berechnen kann. Damit ist dieses Konzept aber auch auf eine allgemeine Single-Server-Topologie anwendbar und nicht nur auf bestimmte Figuren. Allerdings bewirkt die oben erwähnte Bedingung, dass diese allgemeine Topologie trotzdem gewisse Enschränkungen erfüllen muss.
Zum einen muss der neue Aktivierungszeitpunkt immer hinter dem letzten liegen. Ist dies nicht der Fall, so könnte die neue Laufzeit eher ,,loslaufen" als die zuletzt ermittelte und damit deren Feuerzeitpunkt verschieben. Dies wird an einem Beispiel deutlich. In Abbildung 6.4 ist ein Beispiel-Netz zu sehen. Zu dem Zeitpunkt $\tau = 10$ wird in der Stelle P_1 ein Token erzeugt, die generierten Laufzeiten sind $t_{1,1} = 5$ und $t_{3,1} = 4$. Die Transition T_3 wird demzufolge zur Simulationszeit $\tau = 15$ aktiviert und feuert um $\tau = 19$. Nun wird aber um $t = 11$ in P_2 ebenfalls ein Token erzeugt und folgende Laufzeiten werden generiert: $t_{2,1} = 2$ und $t_{3,2} = 5$. Die Transition T_2 feuert um $\tau = 13$ und aktiviert demnach T_3 früher als es der zuletzt ermittelte Aktivierunszeitpunkt von $\tau = 15$ tat. Dies hat zur Folge, dass um $\tau = 13$ die Laufzeit $t_{3,2}$,,losläuft" und damit den Zeitpunkt, zu dem $t_{3,1}$ beginnt, verschiebt. Es enstehen folgende, neue Feuerzeitpunkte für T_3: $\tau = 18$ und $\tau = 22$. Der alte Feuerzeitpunkt ist demzufolge nicht korrekt.
Der Fall, dass der neue Aktivierunszeitpunkt vor dem letzten liegt, kann in einer Single-Server Topologie allerdings nur eintreten, wenn eine Stelle wie in Abbildung 6.4 mehr als eine eingehende Kante hat. Aus diesem Grund dürfen alle Stellen in der Topologie maximal eine eingehende Kante besitzen.
Damit bereits zum Aktivierungszeitpunkt der Feuerzeitpunkt berechnet werden kann, muss schon bei der Aktivierung klar erkennbar sein, ob die Transition überhaupt feuert oder deaktiviert wird. Die Frage der Deaktivierung darf demnach nur direkt von den Eingangsstellen einer Transition abhängen, es dürfen keine inhibitorischen Kanten in der Topologie vorkommen. Desweiteren müssen alle Kanten, die von einer Stelle weggehen, dieselbe Vielfachheit haben. Ist dies nicht der Fall, so kann eine Stelle P_1 zwei hineinlaufende Kanten e_1 und e_2 mit den Vielfachheiten $m_1 = 1$ und $m_2 = 2$ haben. Zum Zeitpunkt $\tau = 10$ wird ein Token in der Stelle erzeugt. Damit ist die Transition der Kante e_1 akti-

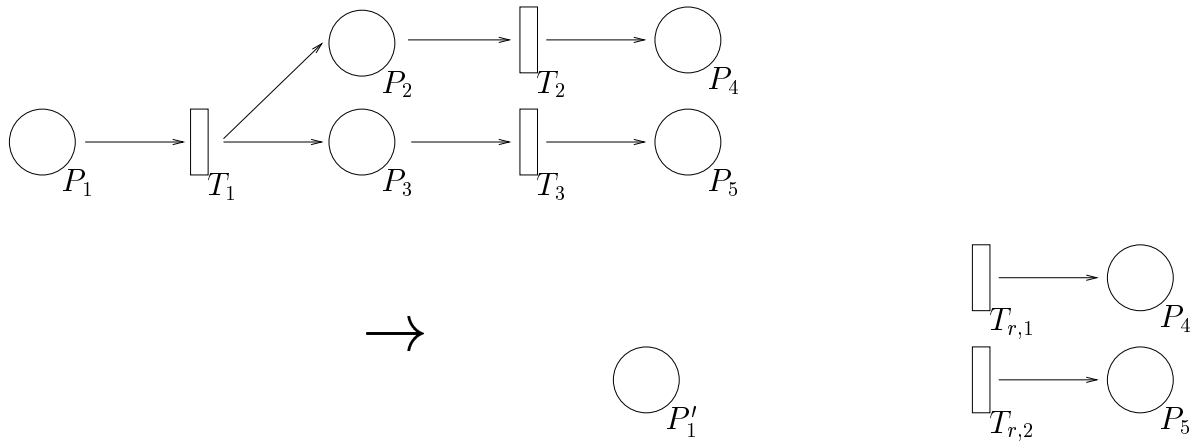


Abbildung 6.5: Ersetzung einer allgemeinen Topologie

viert, sie bekommt eine Laufzeit von $t_1 = 5$ und feuert zum Zeitpunkt $\tau = 15$. Nun wird aber zu der Simulationszeit $\tau = 11$ ein zweiter Token in P_1 erzeugt und damit auch die Transition von e_2 aktiviert. Diese erhält eine Laufzeit von $t = 1$ und feuert demzufolge um $\tau = 12$. Dieses Feuern deaktiviert allerdings die Transition von e_1, wodurch deren vorher berechneter Feuerzeitpunkt nicht mehr korrekt ist.
Aufgrund der Frage der Deaktivierung ergibt sich noch eine weitere Einschränkung. Hat eine Stelle P_i mehr als eine ausgehende Kante, so dürfen die Transitionen dieser Kanten nur P_i als einzige Eingangsstelle haben. Im anderen Fall kann es wieder passieren, dass bei der Erzeugung von Token in P_i nicht alle Transitionen der ausgehenden Kanten aktiviert sind, es aber zu einem späteren Zeitpunkt werden. Dies kann die gleichen Folgen haben, wie wenn die Eingangskanten verschiedene Vielfachheiten hätten. Analog gilt demzufolge auch, dass, wenn eine Transition mehrere Eingangsstellen besitzt, diese Stellen nur mit einer ausgehenden Kante verbunden sein dürfen.
Dank all der bisher gemachten Einschränkungen ist sichergestellt, dass ein einmal berechneter Feuerzeitpunkt immer korrekt bleibt. Eine Topologie, die diese Bedingungen erfüllt, kann demzufolge durch Infinite-Server-Transitionen ersetzt werden. Dabei wird die komplette Topologie aus dem Petri-Netz entfernt und für jede End-Transition wird eine Ersatz-Transition eingefügt, die dieselben Ausgangskanten hat. Eine solche Ersetzung ist in Abbildung 6.5 zu sehen. Die Ersatz-Transitionen besitzen keine Eingangskanten, da es wie in Abbildung 6.5 der Fall sein kann, dass mehrere Ersatz-Transitionen durch denselben Token aktiviert werden, ihn aber nicht beide beim Feuern entfernen können. Damit entsteht aber durch die Ersetzung kein korrektes Petri-Netz: Die Ersatz-Transitionen haben keine Eingangsstellen und sind trotzdem nicht immer aktiviert. Dennoch lässt sich dieses Netz simulieren: Jedesmal, wenn ein Token in einer der Eingangsstellen der Topologie erzeugt wird, werden die neuen Feuerzeitpunkte der End-Transitionen und aus denen die neuen Laufzeiten der Ersatz-Transitionen berechnet. Der Aktivierungsgrad einer Ersatz-Transition entspricht demnach immer der Anzahl an Laufzeiten und nicht umgekehrt! Außerdem werden die Token aus den Eingangsstellen einfach gelöscht. Die Ersatz-Transitionen feuern ganz „normal" zu den Zeiten, bei denen eine ihrer Laufzeiten abgelaufen ist. Da sie Infinite-Server sind, können sie unendlich viele Laufzeiten besitzen und damit das Verhalten der Topologie korrekt nachbilden.

Diese Art der Transformation kann relativ schnell zu einer großen Anzahl an Laufzeiten bei den Ersatz-Transitionen führen. Dadurch steigt der Aufwand für das Ändern der Laufzeiten (siehe 3.5), was wiederum die Simulation verlangsamt. Allerdings lässt sich dieses Problem leicht beheben: Die Ersatz-Transitionen speichern intern die Zeitpunkte, zu denen sie feuern müssen und berechnen lediglich die Laufzeit für den nächsten Feuerzeitpunkt. Damit haben die Ersatz-Transitionen immer maximal eine Laufzeit und bilden das Verhalten der Topologie trotzdem korrekt nach.

6.2 Berechnung der Feuerzeitpunkte

Bei der Single-Server-Sequenz aus 6.1 war die Ermittlung der Feuerzeitpunkte der End-Transition relativ einfach. In diesem Abschnitt wird ein Algorithmus vorgestellt, der die Berechnung aller Feuerzeitpunkte aller Transitionen in einer allgemeinen Topologie ermöglicht. Dabei werden zur besseren Verständlichkeit die Transitionen T_i, die eine Stelle P_j als Eingangsstelle haben, im Folgenden auch als *Nachfolge-Transitionen* von P_j bezeichnet.

Um in einer allgemeinen Topologie die Feuerzeitpunkte der End-Transitionen berechnen zu können, müssen Schritt für Schritt für alle Transitionen die Zeitpunkte des Feuerns berechnet werden. Dafür ist eine Art Mini-Simulation mit einer eigenen Simulationszeit S innerhalb der Topologie notwendig. Zu Beginn wird diese Simulationszeit der Topologie auf die Simulationszeit des eigentlichen Simulators gesetzt $S = \tau$. Anschließend werden die Feuerzeitpunkte der Nachfolge-Transitionen der Eingangsstellen der Topologie ermittelt. Die genaue Berechnung von Feuerzeitpunkten der Nachfolge-Transitionen einer beliebigen Stelle wird später erläutert. Die so erhaltenen Feuerzeiten werden zusammen mit der jeweiligen Transition als zweier-Tupel in einer Menge F gespeichert. Im nächsten Schritt wird das Tupel mit dem kleinsten Zeitpunkt des Feuerns aus der Menge F entfernt, S auf diesen Zeitpunkt gesetzt und die Transition des Tupels feuert. Durch dieses Feuern können nur die Nachfolge-Transitionen der Ausgangsstellen von T_f neu aktiviert werden. Demzufolge genügt es, für diese Transitionen die Feuerzeitpunkte zu berechnen und sie zu der Menge F hinzuzufügen. Anschließend wiederholt sich der Algorithmus ab der Auswahl des Tupels.
Diese Mini-Simulation endet, wenn die Menge F leer ist. Es kann aber auch passieren, dass die Simulationszeit S die Endzeit des eigentlichen Simulators überschreitet. Tritt dieser Fall ein, so kann die Mini-Simulation abgebrochen werden.
Für diesen Mini-Simulator ist die Ermittlung der Feuerzeitpunkte der Nachfolge-Transitionen T_i einer beliebigen Stelle P_j essenziell. Dank der in 6.1 gemachten Einschränkungen ist der Aktivierungsgrad der Transitionen T_i immer gleich. Demzufolge werden zu dem Zeitpunkt S_l, bei dem eine der Transitionen T_i feuert und der Aktivierungsgrad auf null sinkt, alle anderen dieser Transitionen deaktiviert. Die Transitionen T_i speichern diese Simulationszeit S_l und ob sie feuern oder deaktiviert werden. Werden die Transitionen T_i nun zu einem neuen Zeitpunkt S_a wieder aktiviert, so kann eine Transition T_k aus den gespeicherten Informationen und S_a berechnen, zu welcher Gesamtzeit S_f sie feuern würde:
Wurde bzw. wird T_k bei S_l deaktiviert, so gibt es folgende Möglichkeiten. Ist $S_a < S_l$ so

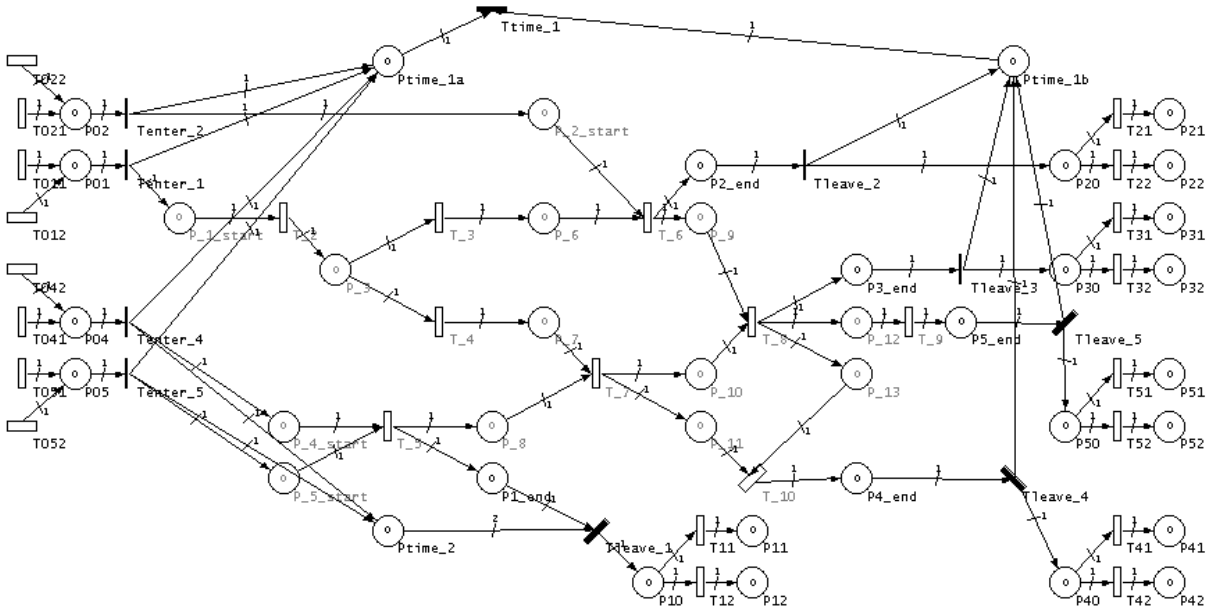


Abbildung 6.6: Das Petri-Netz, mit dem die Korrektheit überprüft wird

kann bei S_l die Laufzeit weiter ,,laufen", da T_k nun bei S_l nicht mehr deaktiviert wird. Ist dagegen $S_a > S_l$, so muss eine neue Laufzeit für T_k entsprechend ihrer race-policy erzeugt werden, welche bei S_a ,,losläuft". Bei dem Fall, dass $S_a = S_l$ muss beachtet werden, dass es nicht egal ist, ob T_k zuerst deaktiviert wird und dann der Aktivierungsgrad steigt oder umgekehrt. Je nachdem entspricht diese Situation einer der beiden vorherigen.
Hat T_k aber zu S_l gefeuert oder wird bei S_l feuern, so muss immer eine neue Laufzeit erzeugt werden. Diese beginnt bei einer Simulationszeit von $\max(S_l, S_a)$ zu ,,laufen".
Aus den so ermittelten, potentiellen Feuerzeiten S_f wird die kürzeste ausgewählt. Deren Transition bekommt damit S_f als einen neuen Feuerzeitpunkt, die restlichen Transitionen werden zu dieser Zeit deaktiviert.
Sollten die Transitionen T_i nach einem Feuern mehrmals aktiviert sein, so werden die entsprechenden Feuerzeitpunkte nacheinander mit Hilfe des eben vorgestellten Verfahren berechnet.

6.3 Experimente

Nachdem in den vorherigen Abschnitten der neue Ansatz erläutert wurde, soll in diesem die Korrektheit gezeigt und eine Analyse des Gewinns an Rechenzeit vorgenommen werden.

Bei dem in diesem Kapitel vorgestellten Ansatz werden die einzelnen Zufallszahlen in exakt der gleichen Reihenfolge verwendet, wie in der originalen Simulationsmethode. Bei der Verwendung der Methode des ,,correlated sampling" dürfte es demzufolge keine Unterschiede in der Statistik geben. Um dies und damit die Korrektheit des Ansatzes zu zeigen, wird das in Abbildung 6.6 dargestellte Petri-Netz verwendet, welches einen komplexen er-

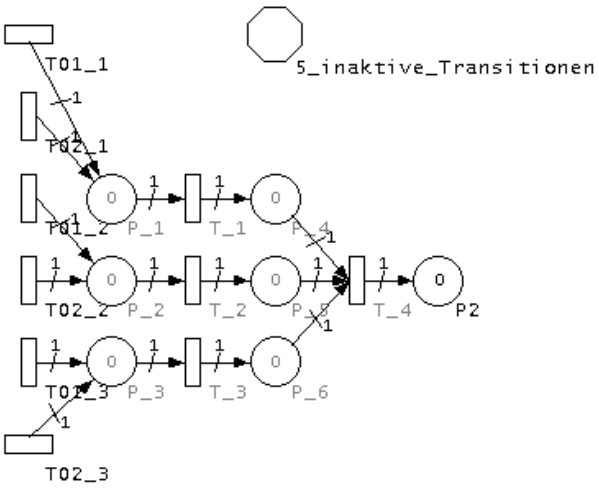


Abbildung 6.7: Das Petri-Netz für die Join-Figur

setzbaren Bereich enthält. In Abbildung B.17 sowie Abbildung B.18 werden die über 10 Replikationen gemittelten Ergebnisse des neuen Ansatzes denen der originalen Methode gegenübergestellt. Bei den Transitionen T0XX haben beide Statistiken denselben Mittelwert für die mittlere Auslastung, allerdings ist bei der originalen Methode zusätzlich eine Standardabweichung vorhanden. Betrachtet man diese Transitionen jedoch genauer, so stellt sich heraus, dass sie während der gesamten Simulationszeit aktiviert sind. Die Standardabweichung in der Statistik der originalen Methode resultiert daher ausschließlich aus Rechenungenauigkeit. Sieht man darum von dieser Abweichung ab, so sind die Ergebnisse des neuen Ansatzes mit denen der originalen Methode identisch. Der in diesem Kapitel vorgestellte Ansatz stellt demzufolge eine korrekte Simulationsmethode für Petri-Netze dar.

Bei dem in diesem Kapitel vorgestellten, neuen Ansatz wird wie bei der adaptiven Simulation eine Topologie von zeitbehafteten Transitionen durch Ersatz-Transitionen dargestellt. Für die Ermittlung der Feuerzeitpunkte werden die originalen Laufzeiten benötigt, der Mini-Simulator muss die als nächstes feuernde Transition auswählen, die Gesamtzeit erhöhen sowie Token entfernen und erzeugen. Dafür müssen bei der Suche nach den aktivierten Transitionen nicht alle zeitbehafteten Transitionen kontrolliert werden, da nach einem Feuervorgang im Mini-Simulator nur die Nachfolge-Transitionen der Ausgangsstellen der feuernden Transition aktiviert sein können. Mit dem neuen Ansatz wird demzufolge der Großteil der Suche nach den aktivierten Transitionen sowie das Ändern der Laufzeiten eingespart. Dafür entsteht zusätzlicher Aufwand für die Verwaltung des Mini-Simulators sowie die nicht ganz einfache Berechnung der Feuerzeitpunkte.

Durch die Einsparung der Überprüfung aller zeitbehafteten Transitionen bei der Suche nach den aktivierten, hängt der mit diesem Ansatz erzielbare Gewinn direkt mit der Anzahl an zeitbehafteten Transitionen zusammen. In dem folgenden Experiment wird diese Beziehung anhand verschiedener Topologien untersucht. Abbildung 6.7 zeigt das Petri-Netz einer *Join-Figur*, welches zusätzlich noch fünf inaktive Transitionen enthält. Die Anzahl der inaktiven Transitionen wird variiert, um einen verschieden hohen Suchauf-

Figur	# inaktive Transitionen	Rechenzeit orig	Rechenzeit adap	Gewinn abs	Gewinn %	Zeit Vorv.
Sequenz	5	2.181	1.431	0.750	34.388	0.057
	50	4.669	2.348	2.321	49.711	0.185
	500	143.271	46.679	96.592	67.419	0.920
Split	5	2.567	2.681	-0.114	-4.441	0.034
	50	5.685	4.962	0.723	12.718	0.149
	500	154.311	126.599	27.712	17.959	0.927
Join	5	3.086	3.547	-0.461	-14.938	0.049
	50	6.277	5.647	0.630	10.037	0.165
	500	175.995	110.362	65.633	37.293	0.912

Tabelle 6.1: Ergebnisse der Untersuchung des Zusammenhanges zwischen Gewinn und Suchaufwand; alle Zeitangaben in Sekunden

# uninteressante Transitionen	Rechenzeit orig	Rechenzeit adap	Gewinn abs	Gewinn %	Zeit Vorv.
9	75.490	100.793	-25.303	-33.518	0.130
6	76.981	103.460	-26.479	-34.397	0.147
5	76.973	93.131	-16.158	-20.992	0.126
4	77.962	83.459	-5.497	-7.051	0.118
3	78.502	75.193	3.309	4.215	0.111

Tabelle 6.2: Ergebnisse der Untersuchung des Zusammenhanges zwischen Gewinn und Komplexität des ersetzten Bereiches; alle Zeitangaben in Sekunden

wand zu erhalten. Die Ergebnisse des Versuchs sind in Tabelle 6.1 abgebildet. Es zeigt sich, dass der Gewinn wie erwartet mit steigender Anzahl an inaktiven Transitionen und damit wachsendem Suchaufwand größer wird. Zusätzlich fällt auf, dass sowohl bei der Split wie auch bei der Join-Figur der neue Ansatz bei kleinen Petri-Netzen langsamer ist als die originale Simulationsmethode, der zusätzliche Aufwand also größer ist als der eingesparte.

Wie bei der adaptiven Simulation muss auch in dem in diesem Kapitel behandelten Ansatz der Vorverarbeitungsschritt nur einmal ausgeführt werden, weshalb er ebenfalls nicht in die Betrachtung des erzielten Gewinns an Rechenzeit einfließt. Allerdings ist aus den Zeiten in Tabelle 6.1 klar erkennbar, dass der Aufwand für den Vorverarbeitungsschritt bei diesem neuen Ansatz höher ist als bei der adaptiven Simulation.

Die Tatsache, dass der neue Ansatz bei kleinen Petri-Netzen für die Join und Split-Figur langsamer als die originale Methode ist, könnte an der höheren Komplexität dieser Figuren liegen. Dies würde aber bedeuten, dass der mit dem neuen Ansatz zu erzielende Gewinn von der Komplexität des ersetzen Bereiches abhängig ist. Um diesen Zusammenhang zu untersuchen, wird das Petri-Netz aus Abbildung 6.6 verwendet, mit dem auch schon die Korrektheit des Ansatzes gezeigt wurde. In dem folgenden Experiment wird die Anzahl der uninteressanten Stellen und Transitionen so geändert, dass die Komplexität des ersetzten Bereiches immer mehr abnimmt. Dabei bedeutet eine hohe Anzahl an ersetzten Transitionen eine hohe Komplexität, eine kleine Anzahl an ersetzten Transitionen eine geringe. Die Ergebnisse des Versuchs sind in Tabelle 6.2 enthalten. Aus diesen geht deutlich hervor, dass der erzielbare Gewinn mit steigender Komplexität abnimmt bzw.

der Verlust zunimmt.
Aus den Ergebnissen der beiden Experimente lässt sich schließen, dass der in diesem Kapitel vorgestellte Ansatz nur für relativ einfache Figuren in großen Petri-Netzen geeignet ist.

Kapitel 7

Ausblick

Die Experimente in Kapitel 4 haben gezeigt, dass mit Hilfe der adaptiven Simulation eine deutliche Verkürzung der benötigten Rechenzeit erzielt werden kann. Der größte Teil des Gewinns wird dabei durch die Einsparung von einzelnen Feuervorgängen erreicht. Wäre man in der Lage, bei einem Petri-Netz noch mehr Feuervorgänge einzusparen, so müsste damit eine noch höhere Beschleunigung erreichbar sein. Eine Möglichkeit, mehr Feuervorgänge einzusparen, ist es, weitere ersetzbare Topologien zu finden. Dies wird insbesonders dann reizvoll, wenn durch neue Figuren in einer Hierarchie von Sub-Netzen auf höherer Ebene wiederum bereits bekannte Topologien entstehen. Damit würde es gelingen, eine Hierarchie mit noch weniger Ersatz-Transitionen darzustellen und damit mehr Feuervorgänge einzusparen. Ein weiterer Vorteil neuer Figuren wäre es, dass sich durch sie die Menge der Petri-Netze, bei denen die adaptiven Simulation angewendet werden könnte, vergrößern würde.

Wäre man in der Lage, einige der in dieser Arbeit verwendeten Bedingungen für die transformierbaren Figuren fallen zu lassen, so würde sich diese Menge ebenfalls erweitern. Eine Möglichkeit hierfür bieten z. B. die Prioritäten: In dieser Arbeit musste jede Transition eines Sub-Netzes die höchste Priorität im Petri-Netz besitzen. Lässt man diese Vorraussetzung fallen, so bleibt nur noch die Bedingung, dass alle dieselbe Priorität haben müssen, übrig. Allerdings entsteht dann das Problem, dass die entsprechenden Ersatz-Transitionen aufgrund einer Transition mit höherer Priorität deaktivert werden können. Bei der dadurch notwendigen speziellen Deaktivierungs-Berechnung muss ähnlich dem Herunterschalten überprüft werden, bei welcher Sub-Netz-Transition eine Laufzeit stehen geblieben ist, damit sie bei der anschließenden Aktivierung genau dort wieder beginnen kann. Desweiteren muss ebenfalls auf den Fall, dass vor einer neuen Aktivierung heruntergeschaltet werden muss, Rücksicht genommen werden. Außerdem ist nicht klar, ob dieser Aufwand den erhofften Gewinn nicht wieder zunichte macht.

In dieser Arbeit wurde anhand eines Beispiels gezeigt, dass die Verwendung eines lernenden Entscheiders durchaus seine Vorteile mit sich bringt. Das dabei verwendete Sarsa-Lernverfahren muss allerdings nicht das optimale Lernverfahren für das Problem des Erkennens von sich lohnenden Schaltvorgängen sein, ebenso wie die gewählten Parameter und Zustands-Aktions-Paare und Belohnungen nicht die optimalen Werte haben müssen. Es ist daher sinnvoll, in einer weiteren Untersuchung aus den vielen verschiedenen Lernverfahren das für die adaptive Simulation am besten geeignete herauszufinden. Dabei kann es durchaus passieren, dass für verschiedene Figuren unterschiedliche Lernmethoden optimal

sind. Ebenso ist es möglich, dass keine Aussage über ein optimales Verfahren getroffen werden kann, da immer eine Entscheidung zwischen Qualität und Aufwand einer Lernmethode gefällt werden muss. Ein Verfahren, dass zwar einwandfrei sich nicht lohnende Schaltvorgänge identifizieren kann, für diese Erkenntnis aber mehr Rechenzeit benötigt, als die gesparten Schaltvorgänge brauchen würden, ist für die adaptive Simulation nicht geeignet.
Der Entscheider erhält aus den Zählern F_r und F_n beim Feuern einer Ersatz-Transition bzw. des Sub-Netzes die Information, ob während der Figur-Laufzeit zumindest teilweise das Sub-Netz bzw. die Ersatz-Transitionen benutzt wurden. Angenommen in einer Figur befindet sich ein Token, dieser aktiviert eine der Anfangs-Transitionen und es wird auf die Ersatz-Transitionen geschaltet. Bevor eine der Ersatz-Transitionen feuert, muss wieder auf das Sub-Netz heruntergeschaltet werden, allerdings ist nun eine End-Transition aktiviert. Demzufolge wurde mindestens ein Feuervorgang eingespart. Wäre dagegen sofort nach dem Hochschalten wieder auf das Sub-Netz geschaltet worden, so wäre kein Feuervorgang eingespart worden. In beiden Fällen erhält der Entscheider aber die gleiche Information von den Zählern. Es wäre demnach eine sinnvolle Erweiterung, dem Entscheider beim Schalten mitzuteilen, wie viele Feuervorgänge während der Benutzung der Ersatz-Transition eingespart wurden bzw. wie viele Feuervorgänge bei der Benutzung des Sub-Netzes stattgefunden haben. Aus dieser Information könnte er dann genauer lernen. Allerdings wird dadurch das Lernen wieder aufwendiger, weshalb zu untersuchen wäre, ob sich dieser extra Aufwand lohnt, also ob ein genügend schnelles Lernverfahren diese Information überhaupt berücksichtigen kann.
Aus den Experimenten in Kapitel 6 geht hervor, dass sich der in diesem Kapitel vorgestellte Ansatz nur für relativ einfache Figuren in großen Petri-Netzen rentiert. Damit ist es aber nicht unbedingt notwendig, eine beliebige Topologie von Single-Server-Transitionen ersetzen zu können. Stattdessen wäre es vielleicht sinnvoller, wie bei der adaptiven Simulation lediglich einzelne Figuren umzuformen. Demnach wäre auch der beschriebene Mini-Simulator nicht mehr notwendig, da sich in einer Figur mit festgelegter Topologie die Ermittlung der Feuerzeitpunkte einfacher gestaltet. Dies hätte den Vorteil, dass weitere Teilschritte der Behandlung der zeitbehafteten Transitionen eingespart werden könnten, womit wiederum der zu erzielende Gewinn wachsen würde. Unklar ist allerdings, ob bei dieser Vorgehensweise auch eine Hierarchie von Figuren ersetzt werden kann.

Anhang A

Implementierungs-Details

In dem Konzept der adaptiven Simulation ist der korrekte Umgang mit den Laufzeiten auch in einer Hierarchie von Sub-Netzen von grundlegender Wichtigkeit. Im Gegensatz zu der Nutzungs- und Ersetzungs-Bedingung sowie den Schaltvorgängen, ist dessen Implementation nicht direkt aus dem Konzept ersichtlich, weshalb in diesem Anhang darauf eingegangen wird. Außerdem wird für den in Kapitel 6 vorgestellten Ansatz eine weitere Einschränkung der umformbaren Topologien vorgenommen, die aber ausschließlich auf computertechnischen Gründen beruht.

Bevor auf den Umgang mit den Laufzeiten eingegangen wird, wird die verwendete Klassenhierarchie für die Transitionen vorgestellt. Alle ,,normalen, herkömmlichen" Transitionen sind Objekte der Klasse `SimTrans`. Von dieser ist die Klasse `SimTransAdaptive` abgeleitet, welche die Oberklasse für die verschiedenen Typen von Ersatz-Transitionen bildet. Für jede implementierte Figur gibt es eine eigene Unterklasse von `SimTransAdaptive`, von welcher die jeweiligen Ersatz-Transitionen instanziiert werden.

Beim Herunterschalten auf das Sub-Netz kann es passieren, dass die Laufzeit einer Sub-Netz-Transition noch gar nicht angefangen hat zu ,,laufen". Demzufolge muss sie gespeichert und später wieder verwendet werden. Dies geschieht über die `store`-Methode. Der Aufruf der `store`-Methode bei einer ,,normalen" Transition bewirkt, dass diese die betreffende Laufzeit intern speichert, so dass sie entsprechend wieder verwendet werden kann. Wird die Methode auf einer Ersatz-Transition aufgerufen, so ruft diese die `store`-Methode wieder bei all den Sub-Netz-Transitionen auf, deren Laufzeit zu der zu speichernden Laufzeit beigetragen hat. Dieses Weiterleiten des Speicherns bewirkt, dass Laufzeiten nur auf der Ebene des originalen Petri-Netzes, also nur in den originalen Transitionen, zwischengespeichert werden.
Das Löschen einer Laufzeit erfolgt mittels einer `remove`-Methode. Für das Entfernen einer Laufzeit kann es allerdings verschiedene Gründe geben. Zum einen kann die Transition gefeuert haben oder sie wurde deaktiviert oder die Laufzeit wurde ,,zu früh" berechnet und deshalb zwischengespeichert. Als erstes wird der Aufruf der `remove`-Methode auf einer ,,normalen" Transition behandelt. Hat die Transition gefeuert oder wurde die Laufzeit zwischengespeichert, so kann die entsprechende Laufzeit einfach entfernt werden. Wurde die Transition jedoch deaktiviert, so muss entsprechend der race-policy reagiert werden. Bei einer Transition mit der race enable Strategie kann die Laufzeit ebenfalls einfach gelöscht

werden. Bei der race age Strategie dagegen merkt sich die Transition die Restlaufzeit, bei der race repeat Strategie jedoch die ursprünglich erzeugte Laufzeit. Diese gemerkte Laufzeit wird dann bei dem nächsten Aktivieren entsprechend der race policy verwendet. Der Aufruf der `remove`-Methode auf einer Ersatz-Transition bewirkt Folgendes. Zunächst wird die entsprechende Laufzeit der Ersatz-Transition immer gelöscht. Hat die Ersatz-Transition gefeuert, so haben dies auch alle Sub-Netz-Transitionen, die zu der entsprechenden Laufzeit beigetragen haben. Bei diesen wird daher ebenfalls die `remove`-Methode mit dem Grund, dass die Transition gefeuert hat, aufgerufen. Gleiches passiert, wenn die Laufzeit zwischengespeichert wurde. Interessant wird die Situation, wenn die Ersatz-Transition deaktiviert wurde, was bei den in dieser Arbeit vorgestellten Topologien allerdings nur bei der Split-Figur passieren kann. Wenn die Ersatz-Transition einer Split-Figur deaktiviert wird, hat dank der gemachten Einschränkungen immer eine andere Ersatz-Transition derselben Split-Figur gefeuert, die allen gemeinsame Sub-Netz-Transition hat daher bereits gefeuert. Falls dies nicht schon eine andere der Ersatz-Transitionen des Splits getan hat, wird auf der gemeinsamen Sub-Netz-Transition die `remove`-Methode mit der Begründung des Feuerns aufgerufen. Bei der ersetzten End-Transition wird zuerst die Laufzeit auf den Wert der Restlaufzeit der Ersatz-Transition gesetzt und anschließend die `remove`-Methode aufgrund der Deaktivierung aufgerufen.
Für das Erzeugen einer Laufzeit ist eine extra `add`-Methode zuständig. Bei einer ,,normalen" Transition überprüft diese als erstes, ob Laufzeiten zwischengespeichert wurden. Ist dies der Fall, wird eine davon als neue Laufzeit verwendet. Nur wenn keine gespeicherte Laufzeit existiert, wird eine neue Laufzeit entsprechend der race-policy erzeugt. Bei der race-age und race-repeat Strategie kontrolliert die Methode, ob sich die Transition aufgrund vorheriger Deaktivierungen Restlaufzeiten bzw. ursprüngliche Laufzeiten gemerkt hat. Wenn ja, so wird entsprechend der race policy eine von ihnen als neue Laufzeit verwendet. Ist dies nicht der Fall bzw. immer bei der race-enable Strategie, wird eine neue Laufzeit entsprechend der Wahrscheinlichkeitsverteilung der Transition generiert. Wird die `add`-Methode bei einer Ersatz-Transition aufgerufen, so braucht diese nur die `add`-Methoden bei den Sub-Netz-Transitionen aufrufen, um deren korrekt erzeugte Laufzeiten zu erhalten, aus denen sie dann die Laufzeit der Ersatz-Transition berechnen kann.
Diese Methoden werden ebenfalls bei einem Schaltvorgang verwendet. Dadurch ist sichergestellt, dass sowohl beim, als auch nach einem Schalten korrekt mit den Laufzeiten umgegangen wird.

Bei dem in Kapitel 6 vorgestellten Ansatz können in die Eingangsstellen einer ersetzbaren Topologie beliebig viele Kanten hineinlaufen. Eine dadurch mögliche Eingangsstelle eines umzuformenden Bereiches ist in Abbildung A.1 dargestellt, in der T_1 und T_2 die race-enable Strategie haben. In dieser speziellen Konstellation bewirkt das Feuern von T_2 kein Deaktivieren von T_1, die Laufzeit von T_1 ,,läuft" ganz normal weiter [Lub01].
Wird der Bereich, der mit P_1 beginnt, ersetzt, so muss für T_2 eine Ersatz-Transition eingefügt werden. Rein theoretisch bilden die Ersatz-Transitionen das Verhalten der transformierten Topologie auch korrekt nach. Aufgrund der diskreten Darstellung der Fließpunktzahlen im Computer kann es allerdings zu einem inkorrekten Verhalten kommen. Wird ein Token in der Stelle P_1 erzeugt, so werden bei dem in Kapitel 6 vorgestellten Ansatz zunächst die Feuerzeitpunkte τ_1 für T_1 und τ_2 für T_2 ermittelt. Angenommen es gilt $\tau_2 < \tau_1$, wodurch T_1 zu dem Zeitpunkt τ_2 deaktiviert wird. Nun feuert theoretisch

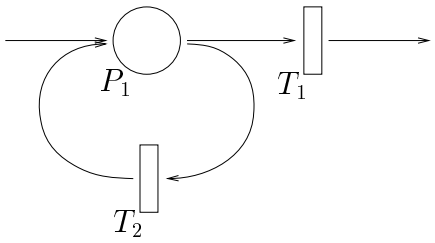


Abbildung A.1: Diese Stelle darf keine Eingangsstelle einer mit Hilfe des in Kapitel 6 vorgestellten Ansatzes zu transformierenden Topologie sein

die Ersatz-Transition von T_2 zu τ_2, T_1 wird demnach zu dem gleichen Zeitpunkt wieder aktiviert, zu dem sie auch deaktiviert wurde. In diesem Fall kann die Laufzeit von T_1 wie in der originalen Topologie „weiterlaufen", die Ersatz-Transitionen bilden das Verhalten korrekt nach. Jedoch kann es aufgrund von Rundungsfehlern passieren, dass die Ersatz-Transition von T_2 nicht exakt um τ_2 sondern kurz danach um $\tau_3 > \tau_2$ feuert. Damit wird T_1 zu einem späteren Zeitpunkt wieder aktiviert, als sie deaktiviert wurde. Demzufolge wird eine neue Laufzeit generiert, anstatt dass die alte „weiterläuft". Dieser Fall kann aber in der originalen Topologie nie eintreten. Im Laufe einer Replikation ist es durchaus wahrscheinlich, dass dieses inkorrekte Verhalten mehrmals eintritt, wodurch sich das Verhalten des gesamten Petri-Netzes ändern kann. Aus diesem Grund darf eine Eingangsstelle einer mit Hilfe des in Kapitel 6 vorgestellten Ansatzes zu transformierenden Topologie nie sowohl Eingangs- wie auch Ausgangsstelle einer Transition sein.

Anhang B

Validierungs-Ergebnisse

In diesem Anhang sind die gesamten Vergleiche der Ergebnisse der adaptiven Simulation mit denen der originalen Methode zur Überprüfung der Korrektheit dieses Verfahrens enthalten, wie auch die Gegenüberstellung der Statistik des in Kapitel 6 behandelten Ansatzes zu der Statistik der originalen Simulationsmethode.

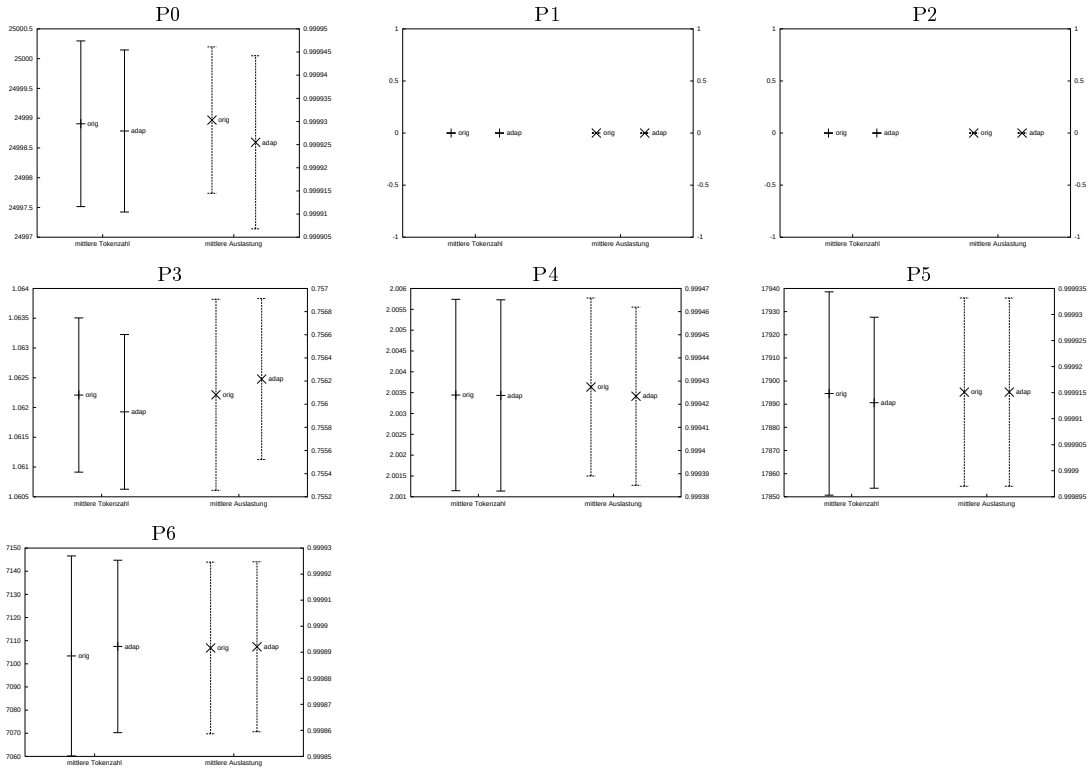


Abbildung B.1: Sequenz — Versuch 1: Statistik-Vergleich für die Stellen bei 10 Replikationen, $V_f = 1:0$, $V_s = 1:0$

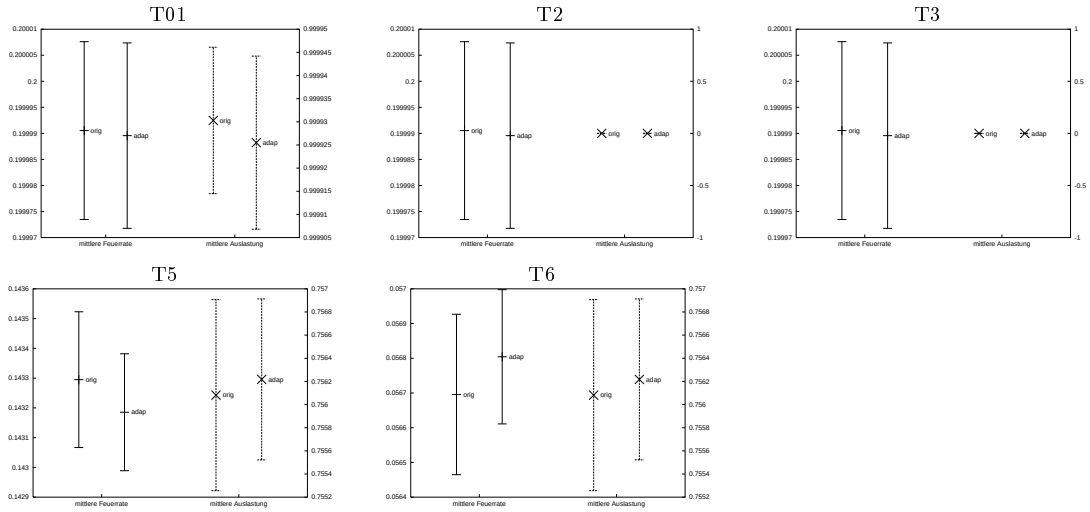


Abbildung B.2: Sequenz — Versuch 1: Statistik-Vergleich für die Transitionen bei 10 Replikationen, $V_f = 1:0$, $V_s = 1:0$

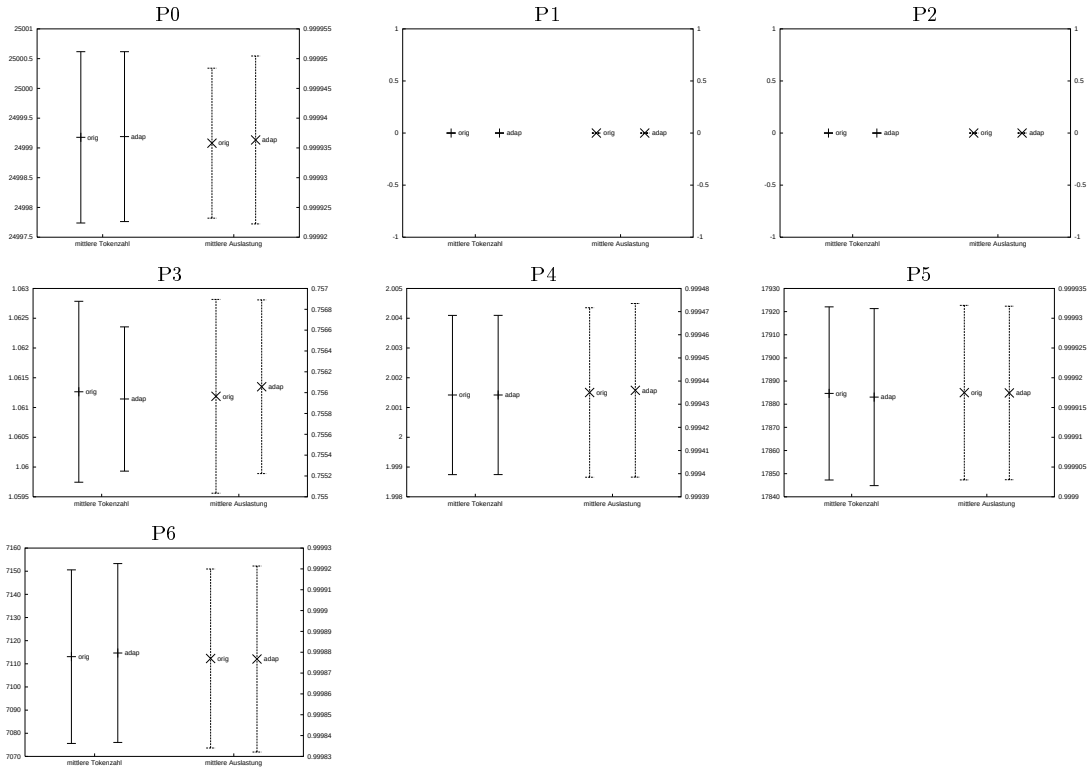


Abbildung B.3: Sequenz — Versuch 2: Statistik-Vergleich für die Stellen bei 40 Replikationen, $V_f = 1:0$, $V_s = 1:0$

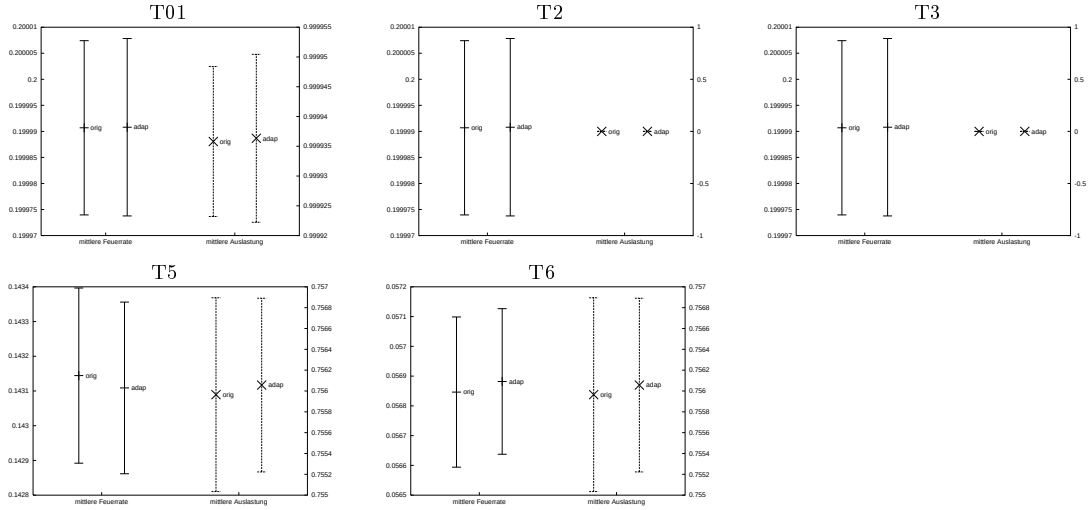


Abbildung B.4: Sequenz — Versuch 2: Statistik-Vergleich für die Transitionen bei 40 Replikationen, $V_f = 1:0$, $V_s = 1:0$

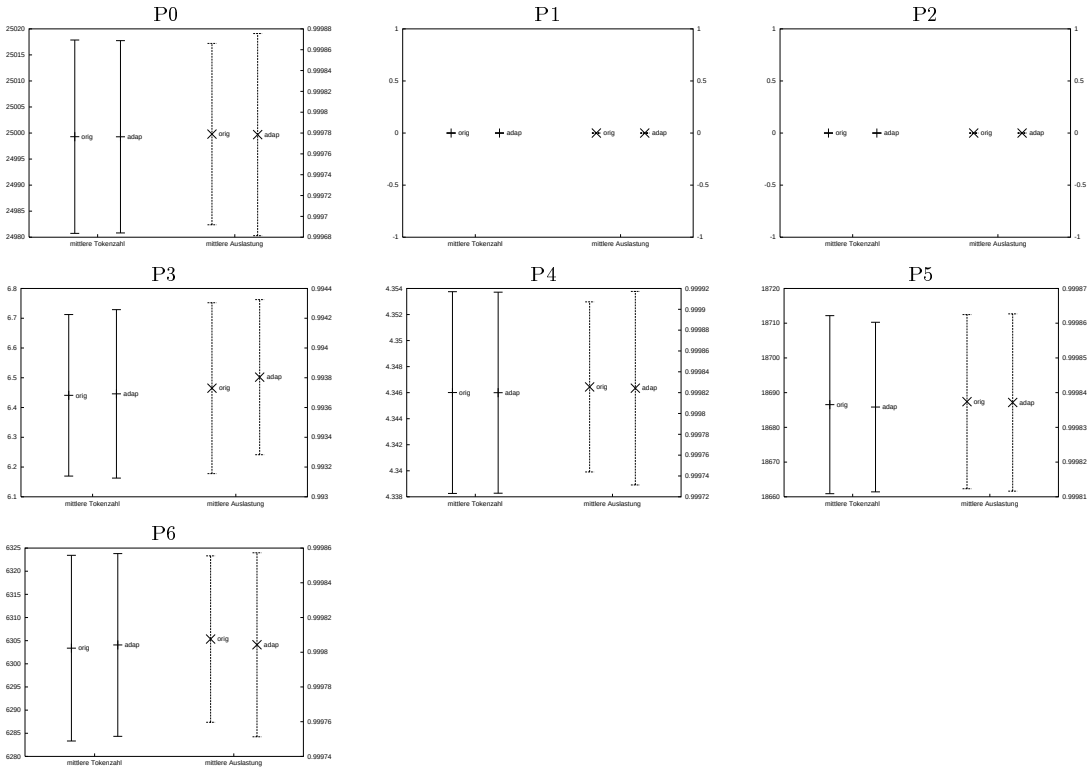


Abbildung B.5: Sequenz — Versuch 3: Statistik-Vergleich für die Stellen bei 40 Replikationen, $V_f = 1 : 0.537$, $V_s = 1 : 0.137$

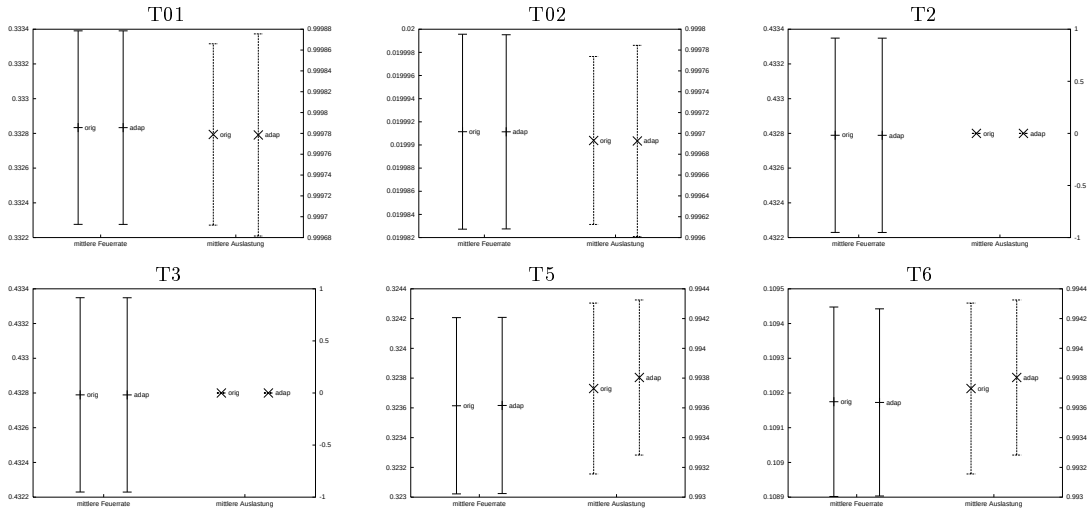


Abbildung B.6: Sequenz — Versuch 3: Statistik-Vergleich für die Transitionen bei 40 Replikationen, $V_f = 1 : 0.537$, $V_s = 1 : 0.137$

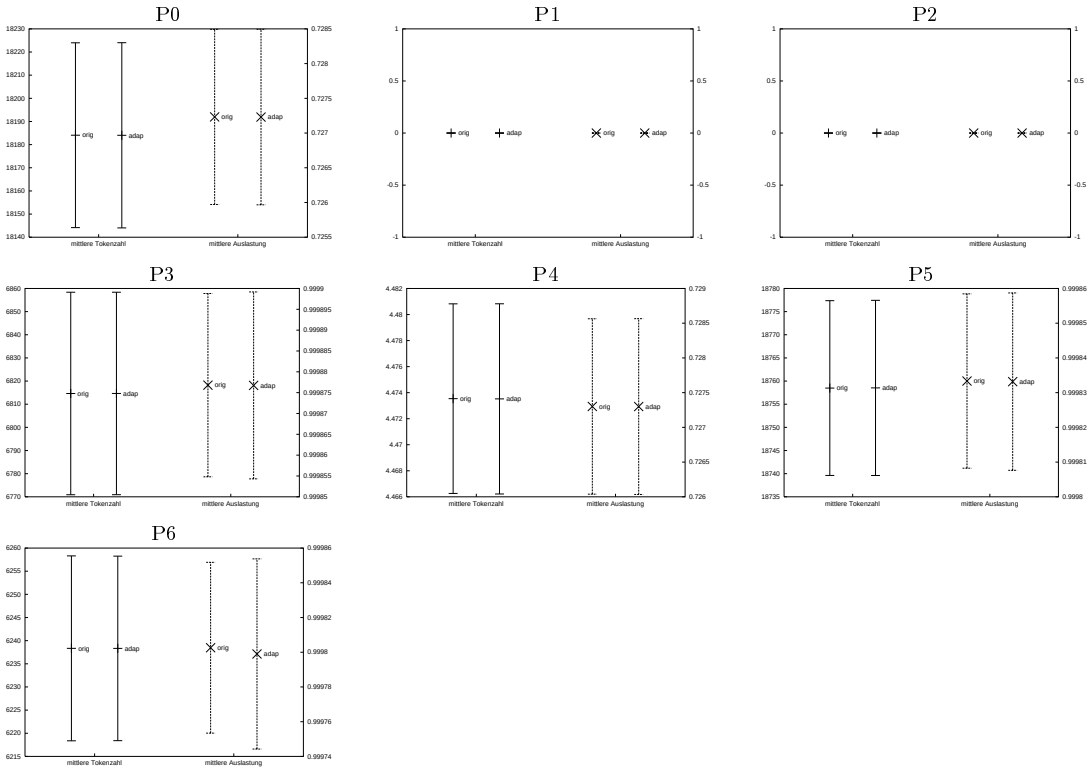


Abbildung B.7: Sequenz — Versuch 4: Statistik-Vergleich für die Stellen bei 40 Replikationen, $V_f = 1 : 1.948$, $V_s = 1 : 0.400$

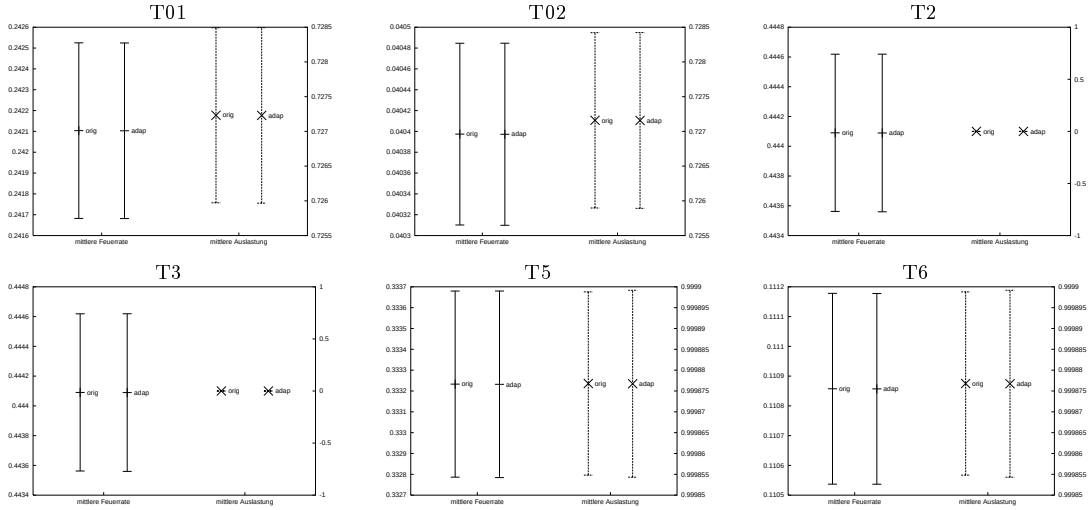


Abbildung B.8: Sequenz — Versuch 4: Statistik-Vergleich für die Transitionen bei 40 Replikationen, $V_f = 1 : 1.948$, $V_s = 1 : 0.400$

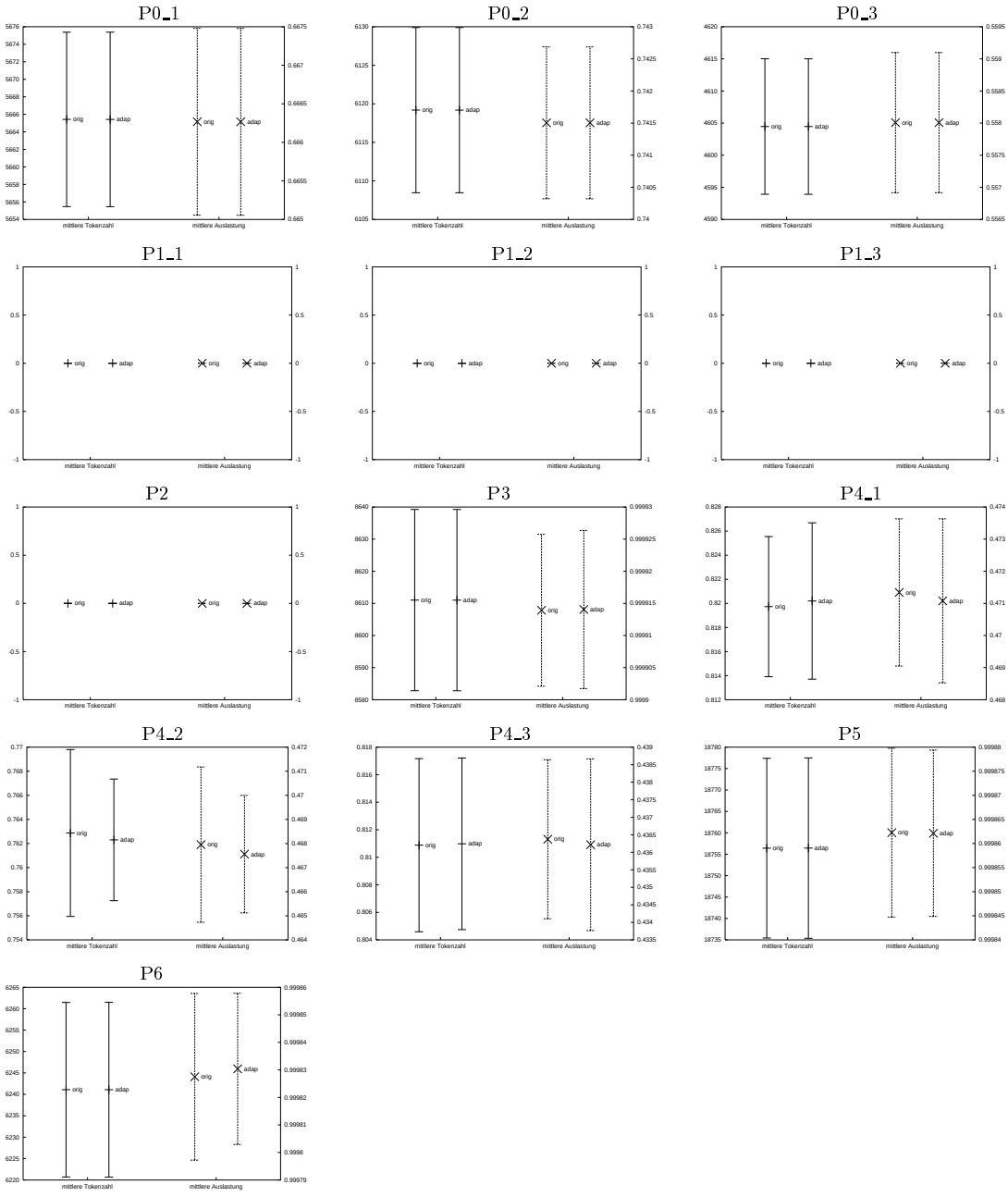


Abbildung B.9: Merge: Statistik-Vergleich für die Stellen bei 40 Replikationen, $V_f = 1 : 0.337$, $V_s = 1 : 0.152$

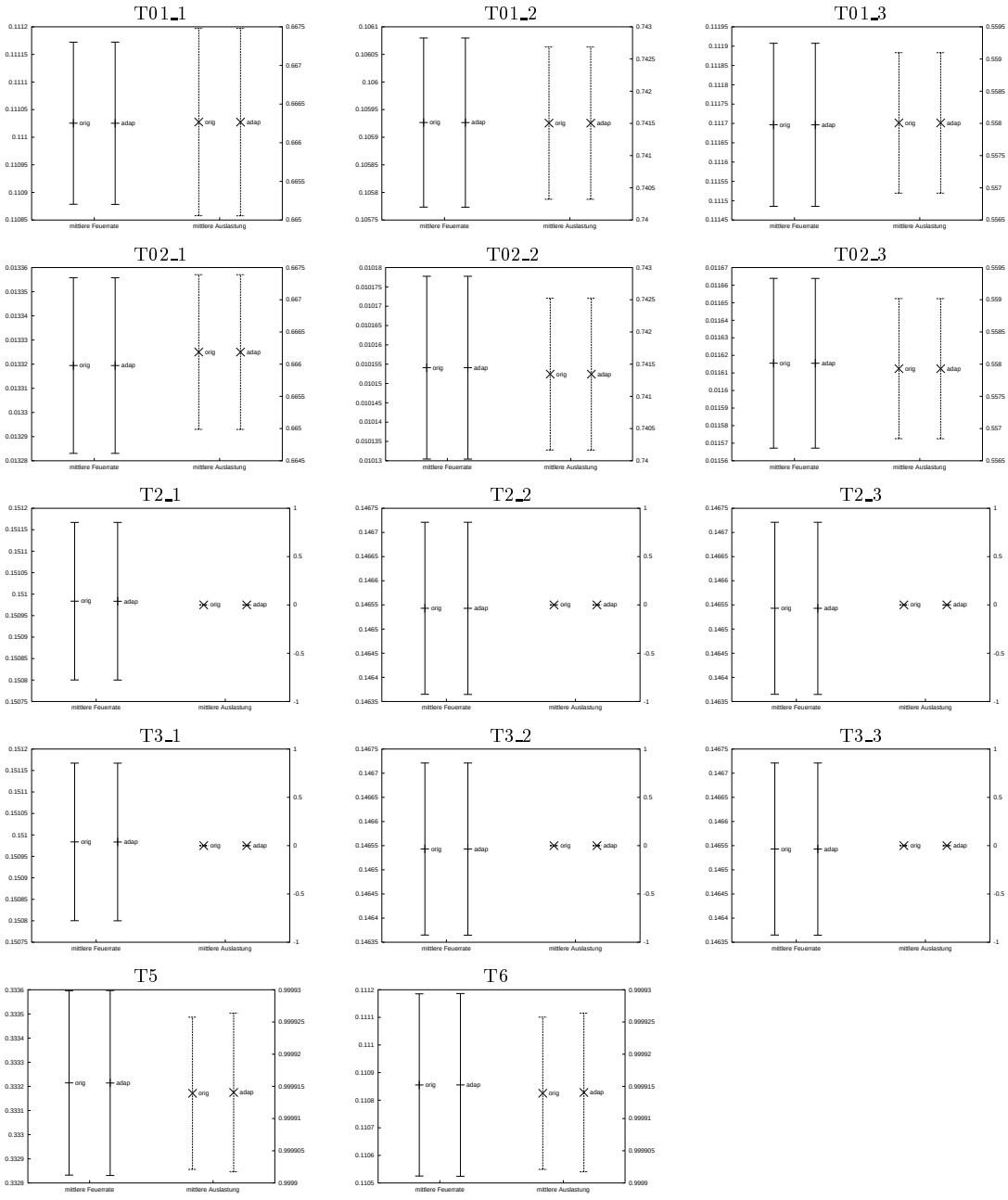


Abbildung B.10: Merge: Statistik-Vergleich für die Transitionen bei 40 Replikationen, $V_f = 1 : 0.337$, $V_s = 1 : 0.152$

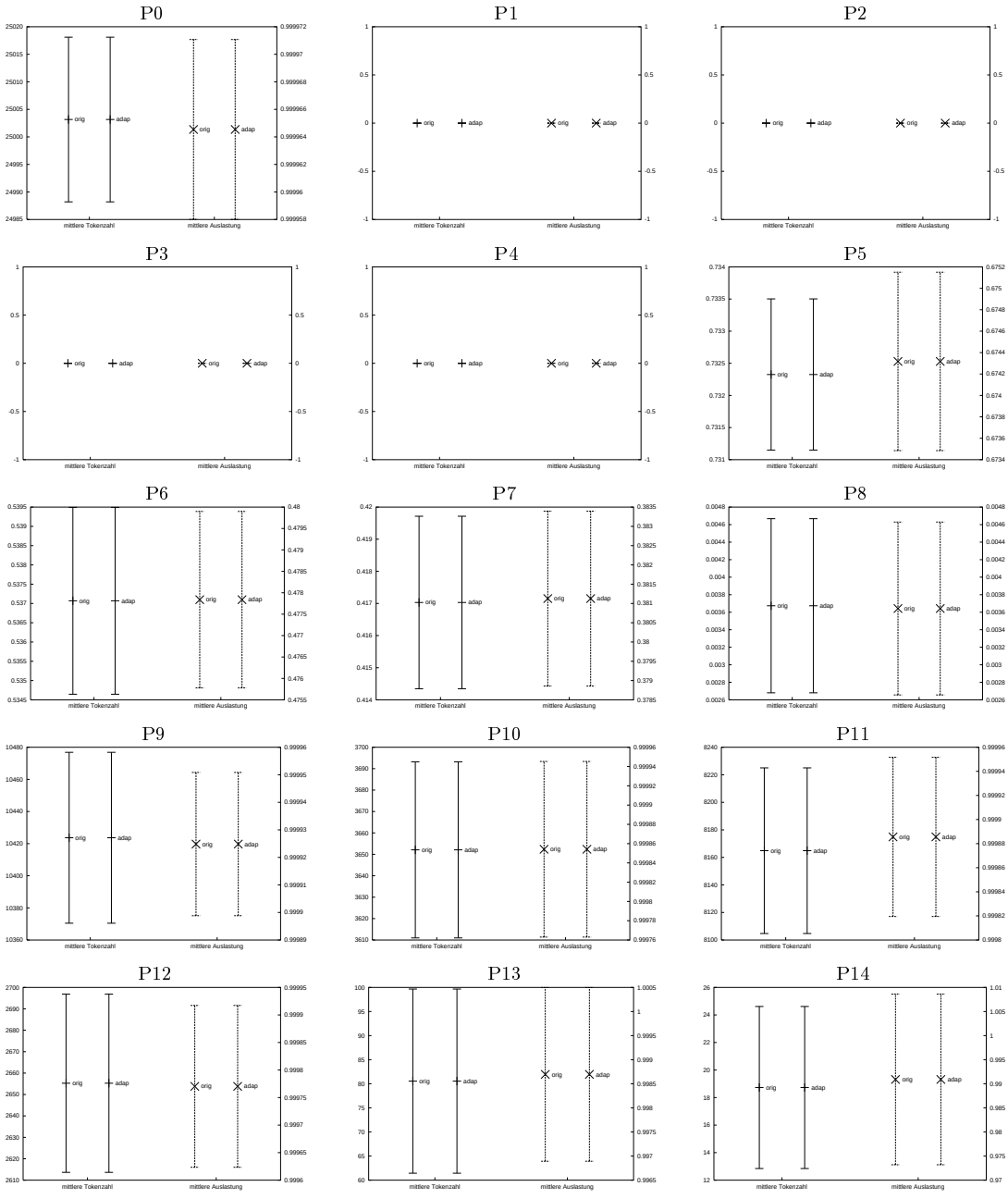


Abbildung B.11: Split: Statistik-Vergleich für die Stellen bei 40 Replikationen, $V_f = 1 : 0.296$, $V_s = 1 : 0.284$

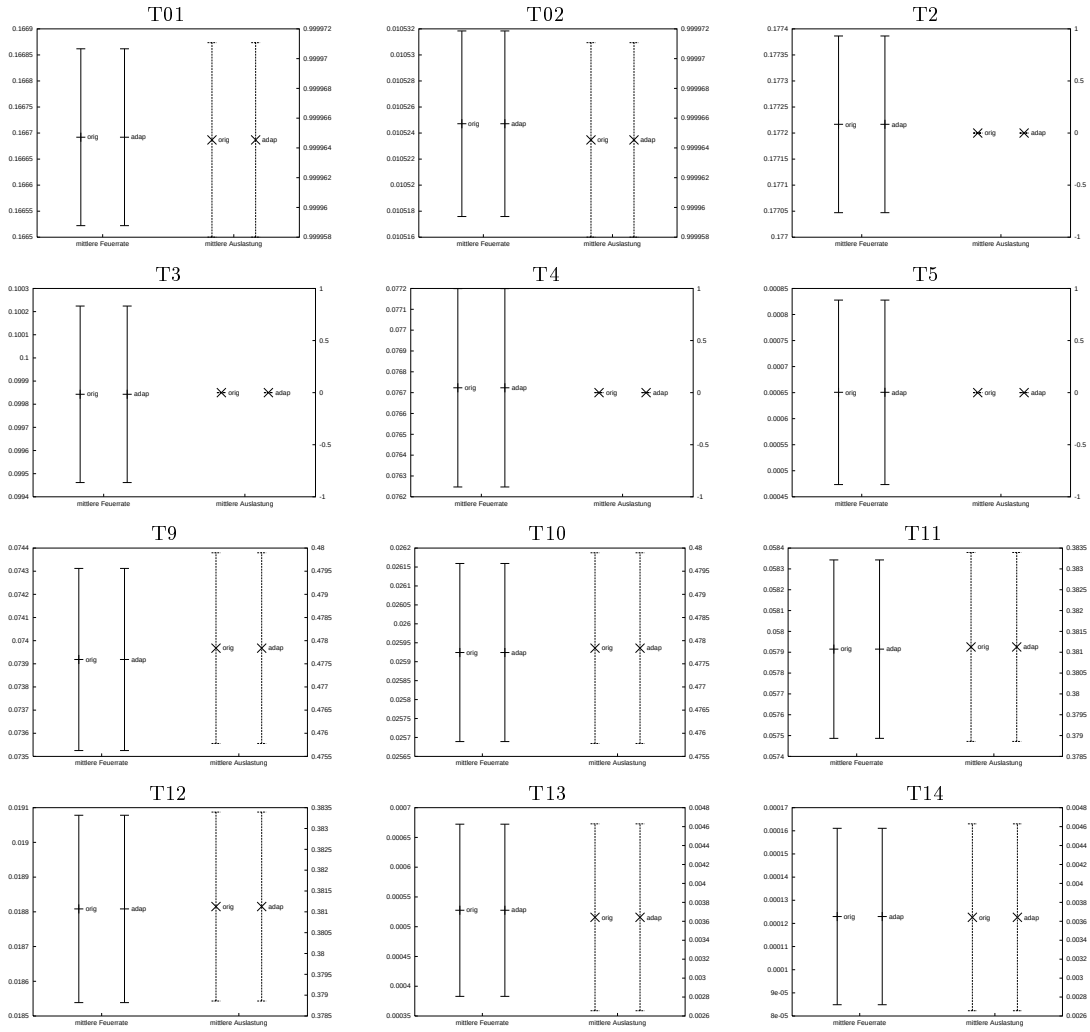


Abbildung B.12: Split: Statistik-Vergleich für die Transitionen bei 40 Replikationen, $V_f = 1 : 0.296$, $V_s = 1 : 0.284$

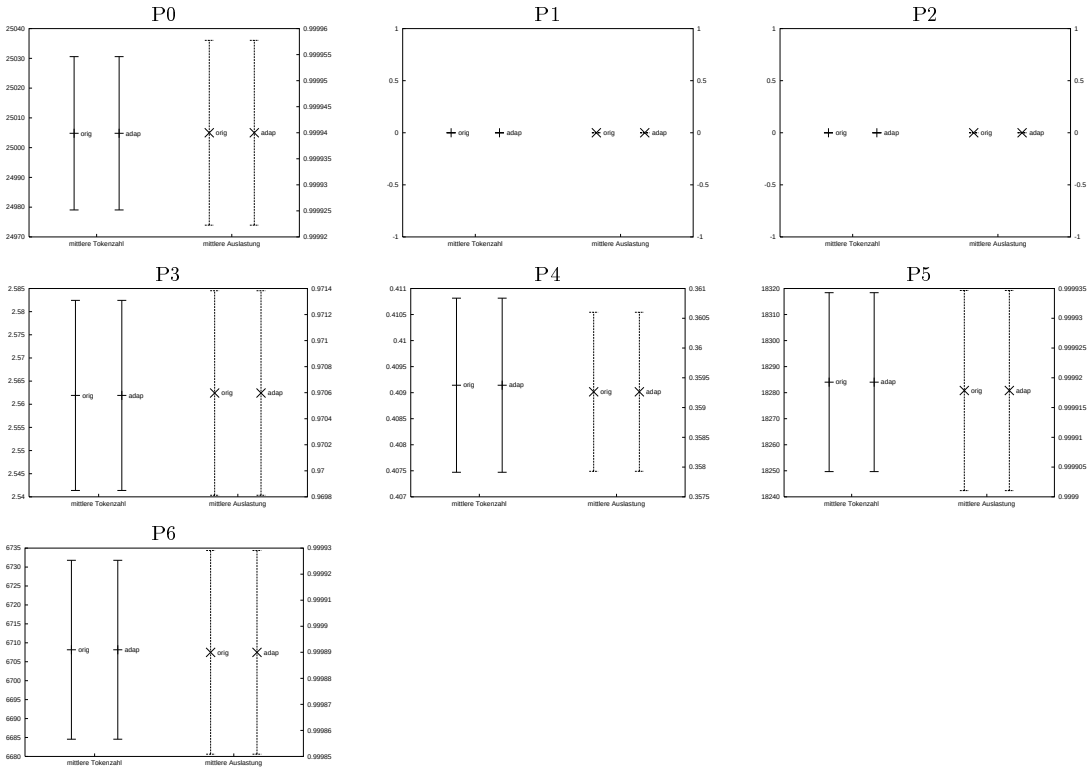


Abbildung B.13: Parallel-Transitionen: Statistik-Vergleich für die Stellen bei 40 Replikationen, $V_f = 1 : 0.292$, $V_s = 1 : 0.275$

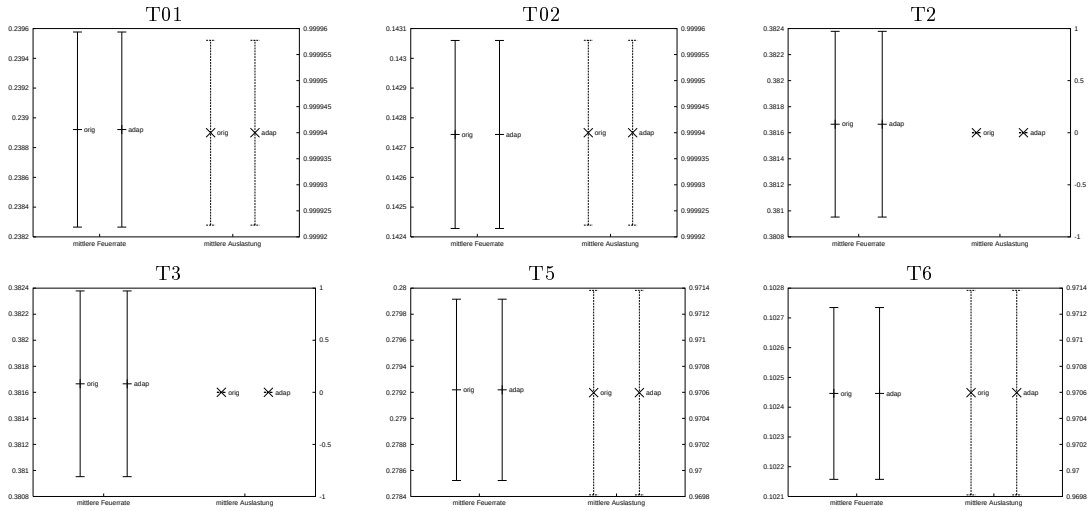


Abbildung B.14: Parallel-Transitionen: Statistik-Vergleich für die Transitionen bei 40 Replikationen, $V_f = 1 : 0.292$, $V_s = 1 : 0.275$

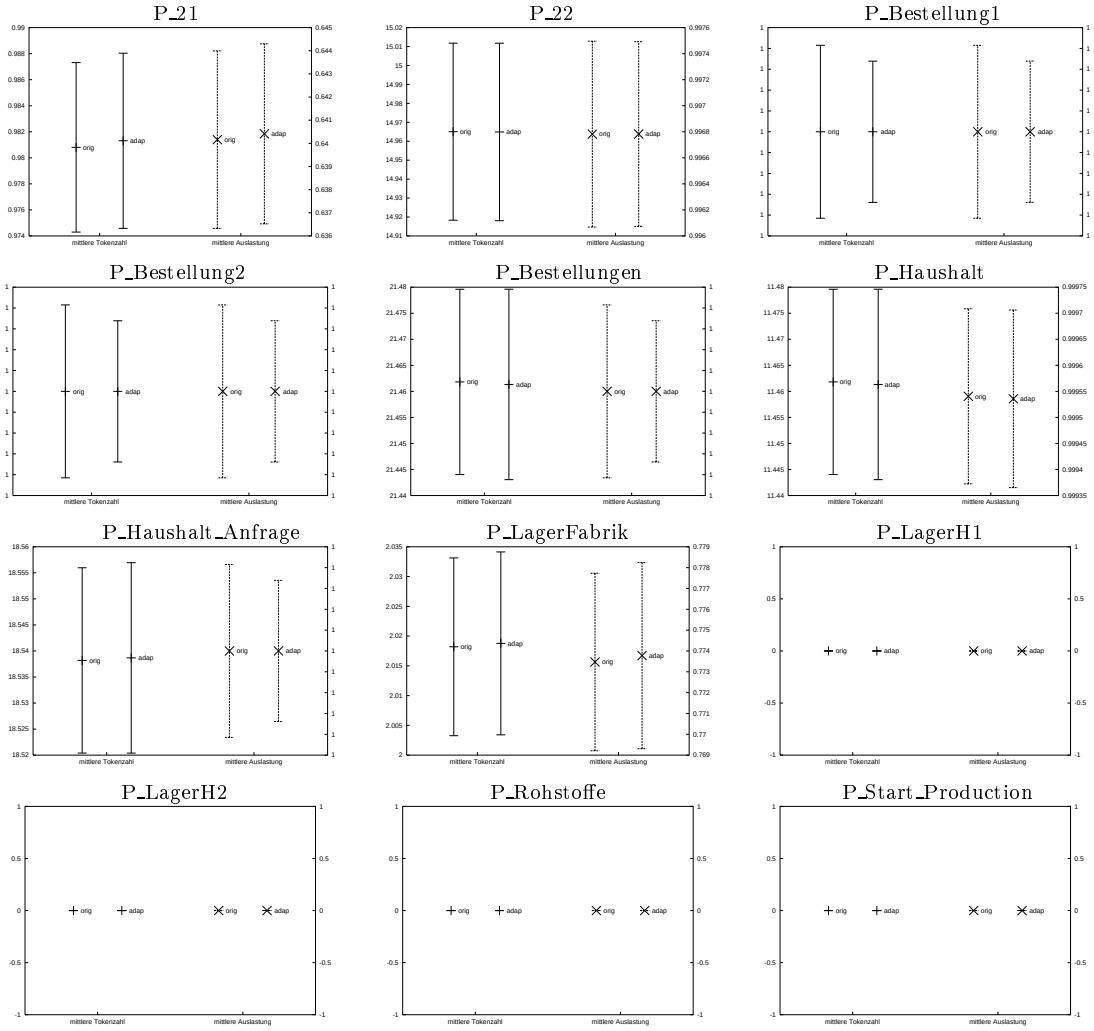


Abbildung B.15: Hierarchie: Statistik-Vergleich für die Stellen bei 100 Replikationen

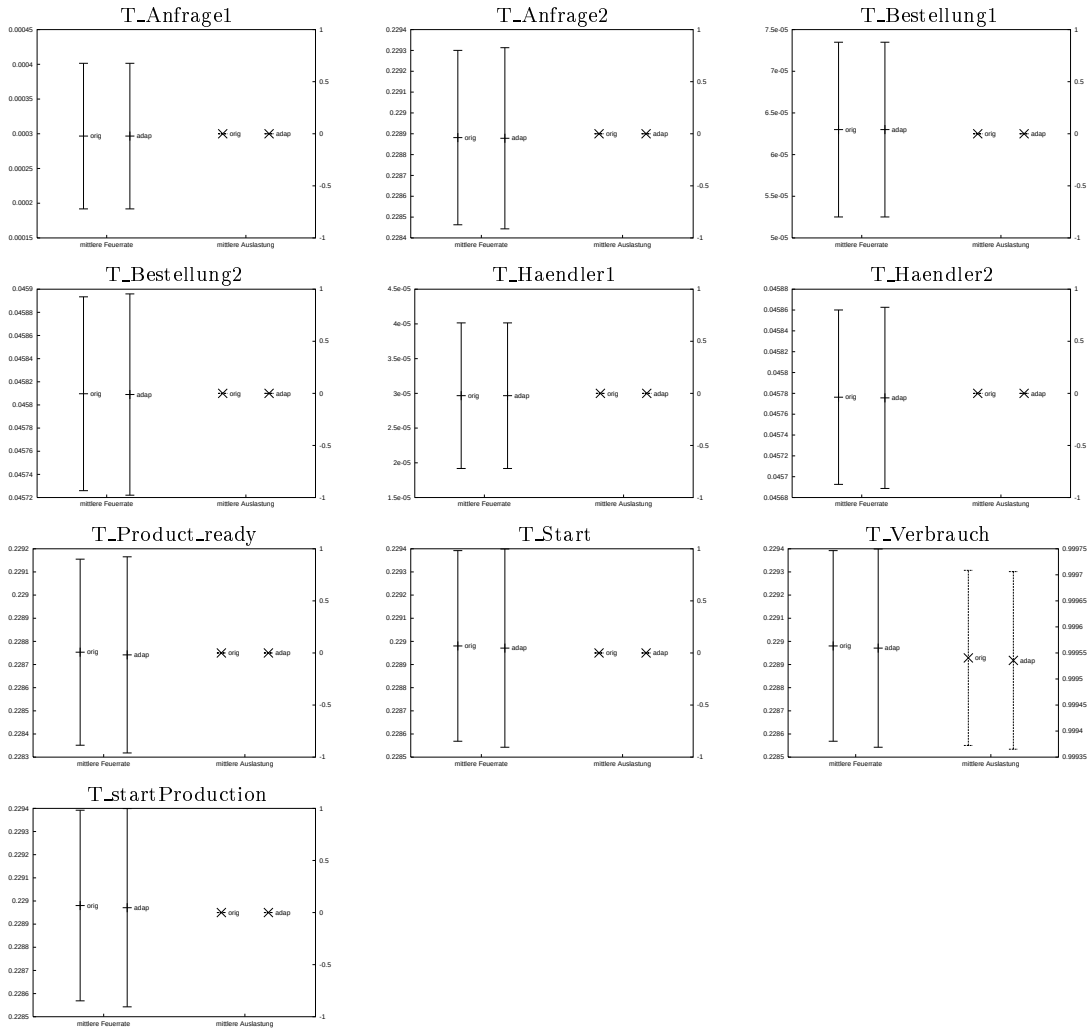
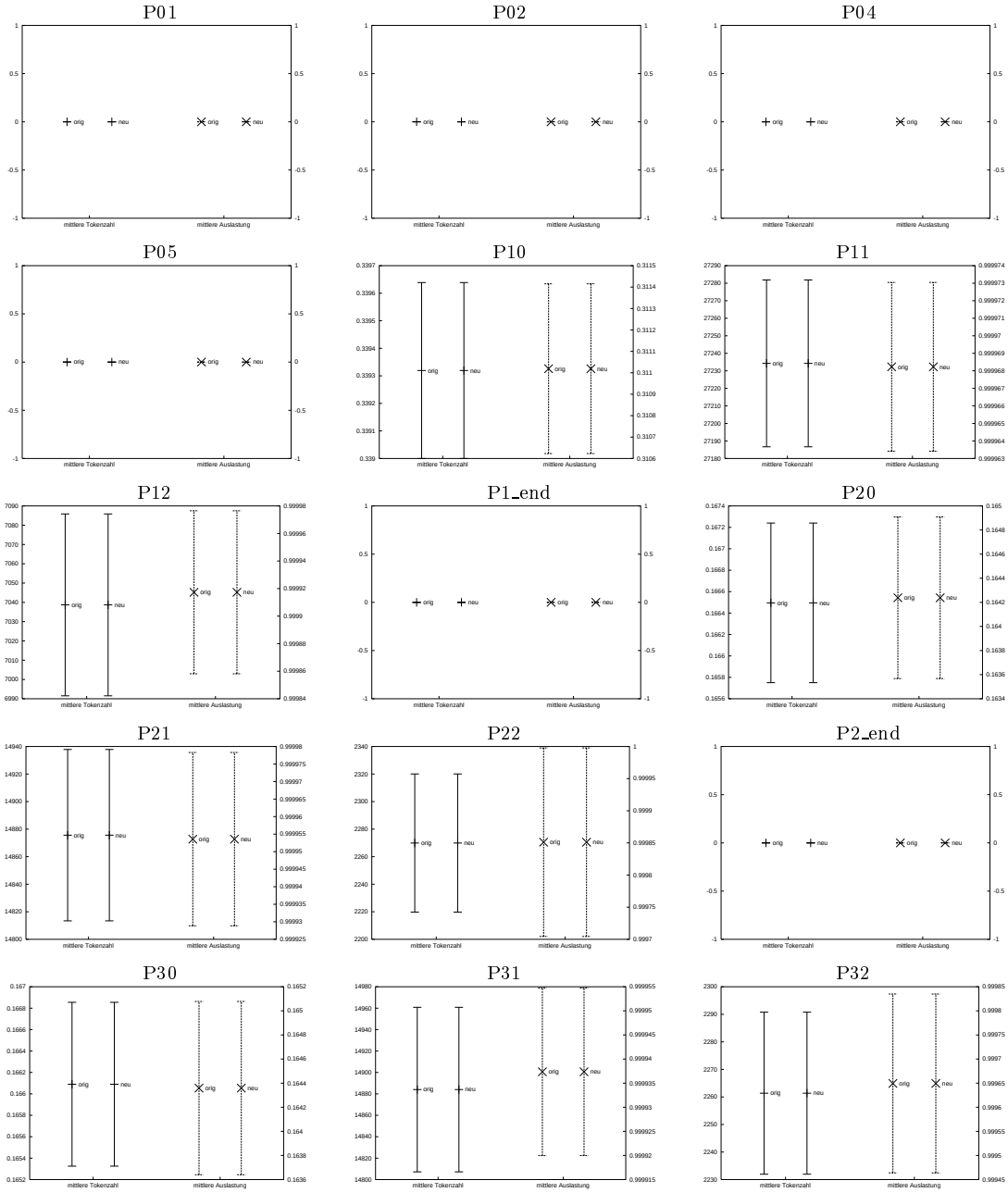


Abbildung B.16: Hierarchie: Statistik-Vergleich für die Transitionen bei 100 Replikationen


P01
P02
P04
P05
P10
P11
P12
P1_end
P20
P21
P22
P2_end
P30
P31
P32
orig
neu
mittlere Tokenzahl
mittlere Auslastung

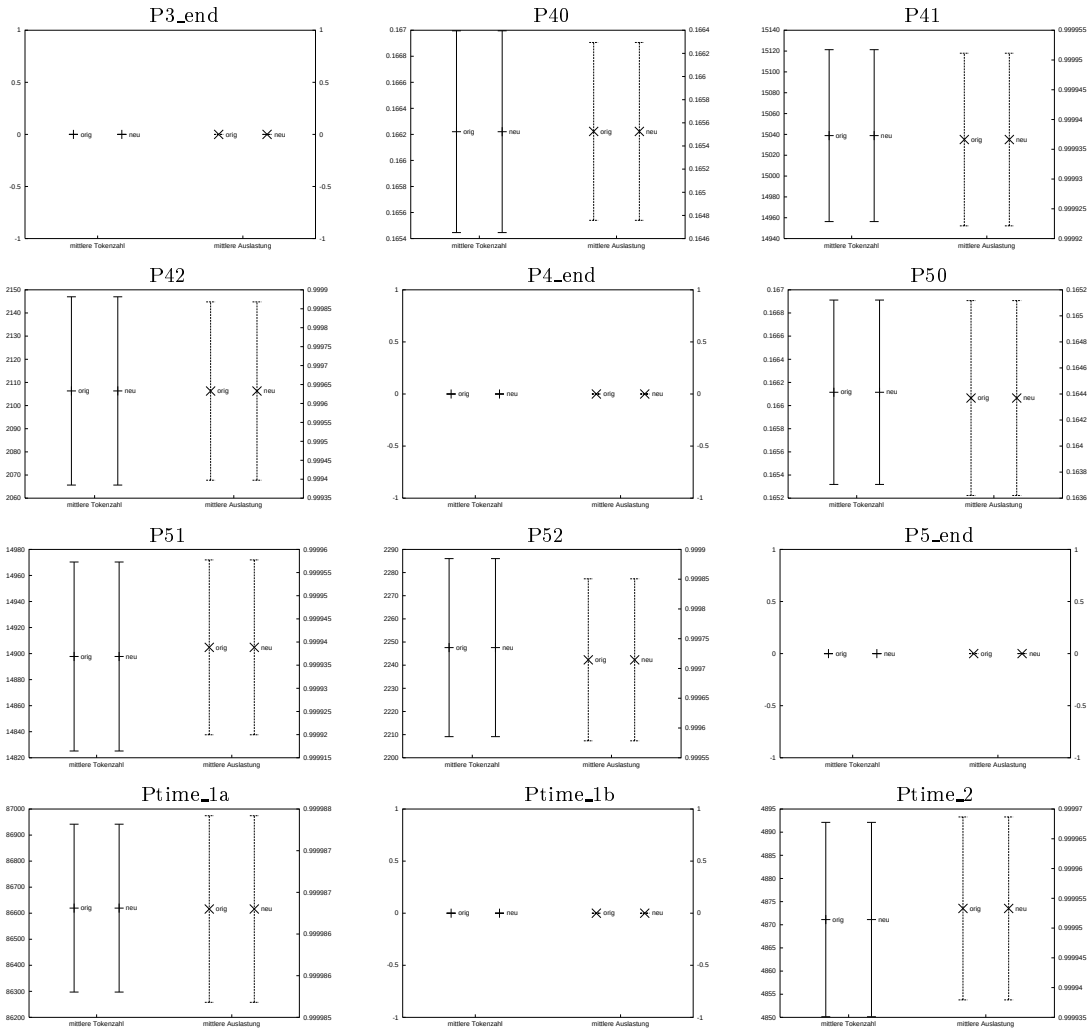
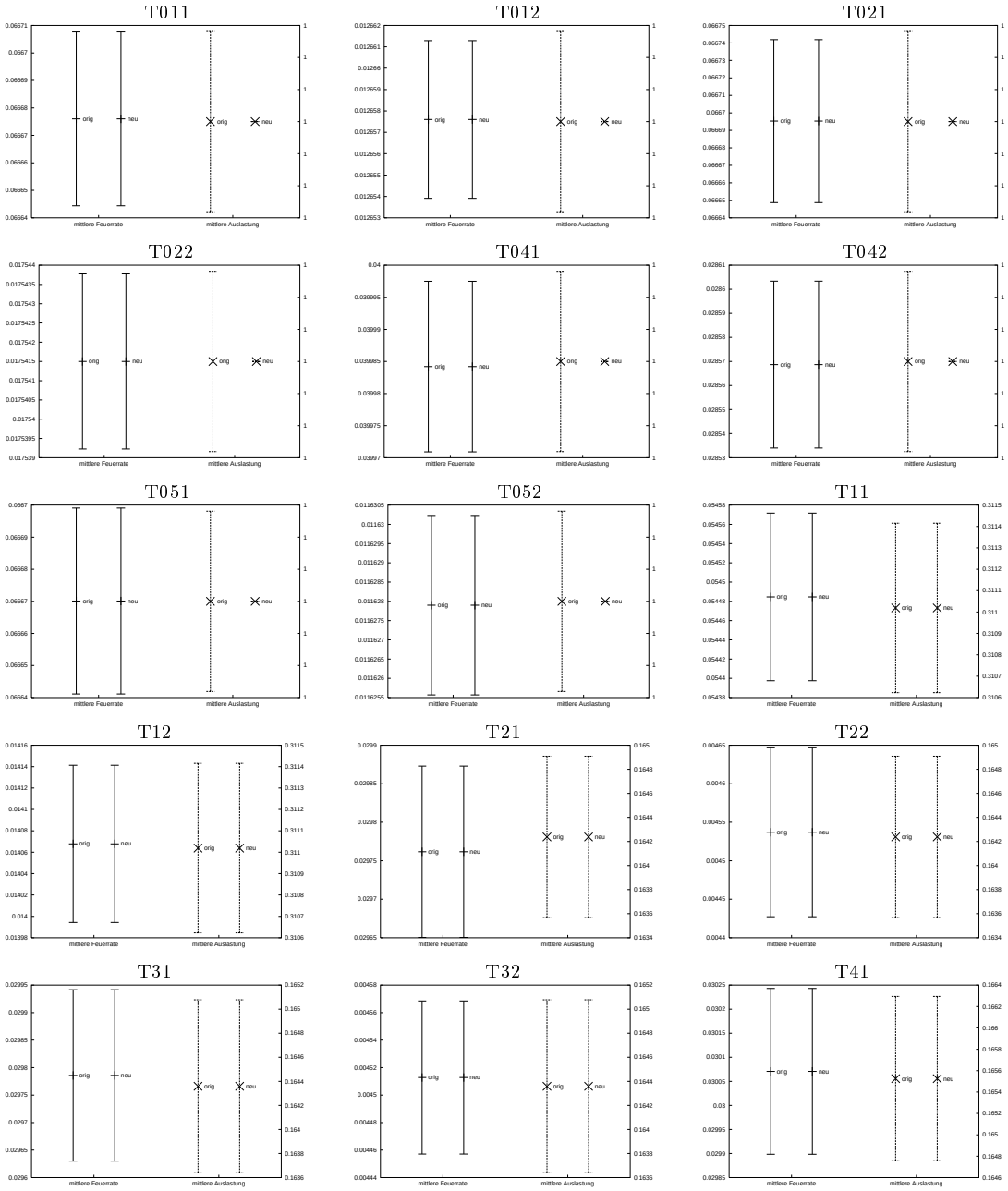


Abbildung B.17: Berechnung der Feuerzeitpunkte: Statistik-Vergleich für die Stellen bei 10 Replikationen


T011
T012
T021
T022
T041
T042
T051
T052
T11
T12
T21
T22
T31
T32
T41
orig
neu
mittlere Feuerrate
mittlere Auslastung

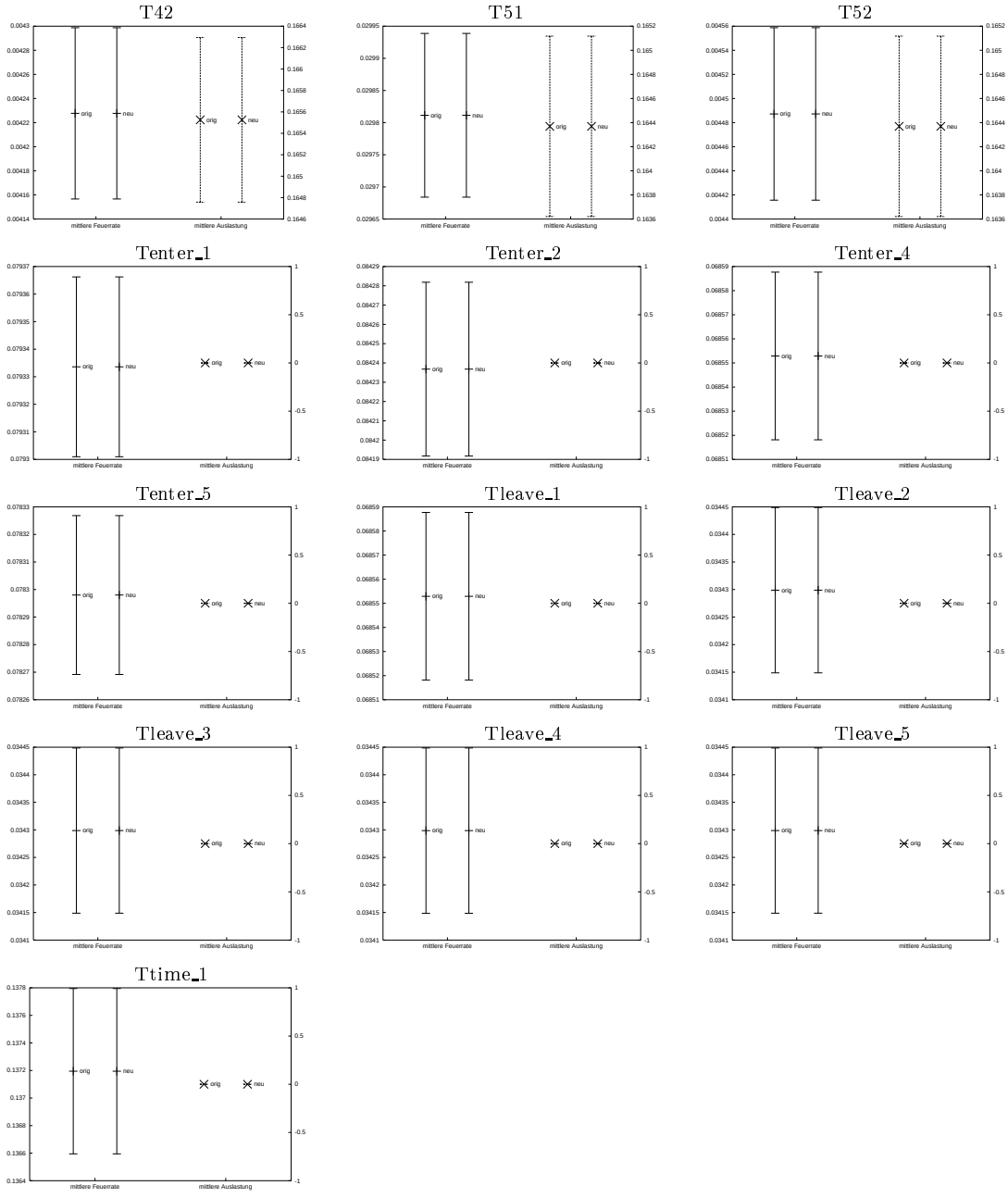


Abbildung B.18: Berechnung der Feuerzeitpunkte: Statistik-Vergleich für die Transitionen bei 10 Replikationen

Literaturverzeichnis

[Bau90] Bernd Baumgarten. *Petri-Netze, Grundlagen und Anwendungen.* Bibliographisches Institut & F.A. Brockhaus AG, Mannheim, 1990.

[BCNN01] Jerry Banks, John S. Carson II, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation.* Prentice-Hall, Upper Saddle River, NJ 07458, 3rd edition, 2001.

[Lub00] Michael Luber. Schnellere Simulation durch Zustandsraumreduktion. Studienarbeit, Lehrstuhl für Systemsimulation, Informatik, Universität Erlangen-Nürnberg, 2000.

[Lub01] Michael Luber. Entwurf und Implementierung effizienzsteigender Methoden zur Simulation stochastischer Petrinetze. Diplomarbeit, Lehrstuhl für Systemsimulation, Informatik, Universität Erlangen-Nürnberg, 2001.

[MBC+95] M. Ajmone Marsan, G. Balb, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* John Wiley & Sons, Inc, New York, 1995.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.