

# Optimierung der Sweep-Line-Methode

Diplomarbeit



Humboldt-Universität zu Berlin  
Mathematisch-Naturwissenschaftliche Fakultät II  
Institut für Informatik

Robert Prüfer

Gutachter:  
Prof. Dr. Wolfgang Reisig  
Prof. Dr. Karsten Wolf

Berlin, den 6. April 2010



## Zusammenfassung

Software wird in der heutigen Zeit oft mit formalen Methoden, denen mathematische Strukturen zugrunde liegen, modelliert. Solche Methoden bieten die Möglichkeit, mit ihnen spezifizierte Systeme automatisch auf bestimmte Eigenschaften zu prüfen. Häufig geschieht dies auf der Basis des Zustandsraumes des Systems, also der Menge aller Zustände mit entsprechenden Übergängen zwischen diesen, die das System annehmen kann. Hierbei tritt das Problem der *Zustandsraumexplosion* [Val90] auf: Schon für kleine Systeme kann der Zustandsraum zu groß werden, um vollständig im Speicher des vorhandenen Rechners repräsentiert werden zu können. Aus diesem Grund wurden einige Techniken zur Reduktion des Zustandsraumes entwickelt. Eine dieser Techniken ist die Sweep-Line-Methode. Mit ihr werden wir uns in dieser Arbeit befassen. Um die Sweep-Line-Methode automatisiert nutzen zu können, ist es nötig, eine Berechnungsvorschrift für das mit der Sweep-Line-Methode verbundene Fortschrittsmaß anzugeben. In [Sch04] wurde eine solche Berechnungsvorschrift für Petrinetze entwickelt, die einige Freiheitsgrade lässt. Der Autor nennt in [Sch04] einige Optimierungsziele, die auf Grundlage dieser Freiheitsgrade erreicht werden sollen. Mit dieser Optimierung wird eine verbesserte Leistung der Sweep-Line-Methode angestrebt. Ziel dieser Arbeit ist es, eine Heuristik zur Umsetzung dieser Optimierungsansätze zu entwickeln und für einige Petrinetze zu testen, ob die Optimierung die Leistung der Sweep-Line-Methode verbessert.



## **Danksagung**

Allen voran möchte ich meinen Eltern danken, die mich während des Studiums konsequent unterstützt und mir stets Rückhalt geboten haben.

Großer Dank gebührt auch Daniela Weinberg für die engagierte Betreuung meiner Studienarbeit und Diplomarbeit. Ihre zahlreichen Hinweise und konstruktiv-kritischen Anmerkungen waren mir eine große Hilfe.

Des Weiteren danke ich Karsten Wolf für sein aufmerksames Interesse an dieser Arbeit und für die Möglichkeit, diese auf Grundlage seiner vorangegangenen Publikation zur automatischen Berechnung von Progress-Werten verwirklichen zu können.

Zudem möchte ich Christian Stahl danken, der mein Interesse am Thema dieser Arbeit geweckt hat und so die Initialzündung für meine Studien- und Diplomarbeit gab.

Danke!



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>9</b>
<b>2</b>	<b>Grundlagen</b>	<b>11</b>
2.1	Definitionen und Terminologie . . . . .	11
2.2	Petrinetze . . . . .	13
2.3	Die Sweep-Line-Methode . . . . .	14
2.4	Algebraische Berechnung von Progress-Werten . . . . .	17
2.4.1	Geometrische Interpretation . . . . .	18
2.4.2	Ansätze zur Optimierung . . . . .	18
2.5	Das Maximum Feasible Subsystem Problem . . . . .	20
2.5.1	Algorithmische Lösungsmethoden . . . . .	20
2.5.2	LP-basierte Heuristik zur Lösung von MIN IIS COVER-Instanzen . . .	21
2.6	DAS ALL PAIRS SHORTEST PATHS PROBLEM auf Graphen . . . . .	24
<b>3</b>	<b>Geometrisch basierter Ansatz zur Berechnung der Offset-Werte</b>	<b>27</b>
<b>4</b>	<b>Optimierung der Berechnung der Offset-Werte</b>	<b>31</b>
4.1	Minimierung der Anzahl negativer Offset-Werte . . . . .	31
4.1.1	Mögliche Beschleunigung der Berechnung . . . . .	37
4.1.2	Heuristik zur Berechnung mehrerer MIN IIS COVER-Lösungen . . . .	39
4.2	Vermeidung von Ketten von Regresstransitionen . . . . .	41
4.2.1	Aktivierungsgraph . . . . .	42
4.2.2	Heuristik zur Vermeidung von Ketten von Regresstransitionen . . . .	44
4.3	Implementierung . . . . .	50
4.4	Fallstudie . . . . .	52
<b>5</b>	<b>Bewertung der Optimierungsansätze</b>	<b>59</b>
5.1	Minimierung der Anzahl negativer Offset-Werte . . . . .	61
5.2	Vermeidung von Ketten von Regresstransitionen . . . . .	64
<b>6</b>	<b>Schlussbemerkungen</b>	<b>67</b>
6.1	Zusammenfassung . . . . .	67
6.2	Ausblick . . . . .	67



# 1 Einleitung

In der heutigen Welt sind Computer aus dem Alltag vieler Menschen kaum noch wegzudenken. Digitale Systeme werden in immer mehr Bereichen eingesetzt, nicht nur zur Verarbeitung von Informationen wie in Wirtschaft und Behörden, sondern beispielsweise auch als eingebettete Systeme in vielen verschiedenen Geräten wie Fahrzeugen, ebenso wie zur Konstruktion von Bauwerken oder technischen Geräten und zur Simulation und Berechnung komplexer Abläufe. Auch die Komplexität von Software und Hardware hat mit der vermehrten Nutzung von Computern deutlich zugenommen. Mit steigender Bedeutung von Computern für die Gesellschaft wurde die Notwendigkeit erkannt, Software und Hardware vor der eigentlichen Erstellung zu modellieren, da komplexe Systeme anderenfalls kaum noch beherrschbar sind und bereits kleine Fehler unter Umständen fatale Auswirkungen haben können; mag der Ausfall der haus-eigenen Waschmaschine noch zu verkraften sein, kann fehlerhafte Software in einem Flugzeug oder einem Atomkraftwerk zu einer ernsthaften Gefahr werden.

Für die Modellierung diskreter Systeme bieten sich *formale Methoden* an. Diese haben den Vorteil, dass ihre Semantik mathematisch eindeutig spezifiziert ist. Daher bieten sie die Möglichkeit, mit ihnen erstellte Modelle automatisch auf bestimmte Eigenschaften zu überprüfen. Bei der *expliziten Zustandsraumverifikation* [Sch02] werden dafür alle möglichen Zustände, die in einem bestimmten System auftreten können, durchmustert. Ziel ist es, zu überprüfen, ob einer oder mehrere Zustände mit der gewünschten Eigenschaft existieren oder nicht. Oft verhält es sich allerdings so, dass der gesamte Zustandsraum deutlich mehr Zustände enthält, als im Speicher des Rechners repräsentiert werden können; man spricht in diesem Zusammenhang vom Problem der *Zustandsraumexplosion* [Val90]. Um die Anzahl der Zustände, die im Speicher gehalten werden müssen, zu verringern, wurden verschiedene Reduktionsmethoden entwickelt. Die Sweep-Line-Methode ist eine dieser Reduktionsmethoden. Mit ihr werden wir uns in dieser Arbeit beschäftigen. Sie kann bei der Überprüfung von Sicherheitseigenschaften wie z.B. der Existenz von Deadlocks oder der Erreichbarkeit eines bestimmten Zustands angewandt werden. Die grundlegende Idee der Sweep-Line-Methode ist es, Zustände, die für die weitere Durchmusterung des Zustandsraumes nicht mehr benötigt werden, aus dem Speicher zu löschen. Um zu wissen, welche Zustände gelöscht werden können, ist ein Fortschrittsmaß, die sogenannte *Progress Measure*, nötig. Allerdings wird in den Originalpublikationen zur Sweep-Line-Methode keine Aussage darüber getroffen, wie die Progress Measure automatisch berechnet werden kann. In [Sch04] wird solch eine automatische Berechnung der Progress Measure für Petrinetze, eine formale Methode, die insbesondere zur Modellierung verteilter Algorithmen und reaktiver Systeme geeignet ist, vorgeschlagen. Die Progress Measure wird hierbei aus *Offset-Werten* kombiniert, welche den einzelnen Transitionen des Petrinetzes vor der Zustandsraumdurchmusterung zugewiesen werden. Die Berechnung der Offset-Werte lässt allerdings an zwei Stellen Freiheitsgrade zu, so dass es möglich ist, sie an diesen Stellen im Hinblick auf den Speicherplatzbedarf oder das Laufzeitverhalten zu optimieren. In [Sch04] werden Optimierungsziele vorgeschlagen, von deren Erreichen eine Steigerung der Leistung der Sweep-Line-Methode erwartet wird. Ziel dieser Arbeit ist es, geeignete Heuristiken zum

## *1 Einleitung*

Erreichen dieser Optimierungsziele zu finden und anhand einer Fallstudie zu überprüfen, inwiefern die vorgestellten Heuristiken tatsächlich die Leistung der Sweep-Line-Methode verbessern. Der Fokus wird hierbei auf der Verringerung des Speicherplatzbedarfs liegen; die Laufzeit spielt nur eine untergeordnete Rolle.

Die Arbeit ist neben dieser Einleitung in fünf weitere Kapitel gegliedert. Im folgenden Kapitel werden wir zunächst einige zum Verständnis dieser Arbeit notwendige Grundlagen betrachten. Anschließend wird der geometrisch basierte Ansatz zur Berechnung der Offset-Werte beschrieben, bevor wir uns in Kapitel 4 der Optimierung der Berechnung der Offset-Werte zuwenden können; in diesem Kapitel ist auch eine Fallstudie enthalten. Als Konsequenz aus den Ergebnissen der Fallstudie werden die Optimierungsansätze in Kapitel 5 einer erneuten Bewertung unterzogen. Mit einer Zusammenfassung und einem Ausblick auf offene Fragen wird die Arbeit abgeschlossen.

## 2 Grundlagen

### 2.1 Definitionen und Terminologie

An dieser Stelle werden wir einige notwendige Begriffe, Definitionen und Konventionen einführen. Insbesondere für die in dieser Arbeit entwickelten Heuristiken werden Definitionen aus verschiedenen Teilgebieten der Mathematik benötigt, die an dieser Stelle zunächst nur eingeführt werden sollen, ohne dass wir auf ihre genauere spätere Verwendung eingehen. Die Definitionen sind an gängige Konventionen aus dem jeweiligen Bereich angepasst.

**Zahlenmengen.** Wir bezeichnen mit  $\mathbb{N}$  die Menge der natürlichen Zahlen, mit  $\mathbb{Q}$  die Menge der rationalen Zahlen, mit  $\mathbb{R}$  die Menge der reellen Zahlen und mit  $\mathbb{Z}$  die Menge der ganzen Zahlen. Es gilt  $0 \in \mathbb{N}$ .

**Vektoren und Matrizen.** Einen  $n$ -dimensionalen Vektor  $v$  schreiben wir als  $v = (v_0, \dots, v_{n-1})$ , wobei wir insbesondere bei der Vektormultiplikation Zeilenvektoren durch die Schreibweise  $v^T$  von Spaltenvektoren abgrenzen, sofern nicht eindeutig erkennbar ist, dass es sich um Zeilen einer Matrix handelt. Die Indizierung der Elemente eines Vektors beginnt immer bei 0. Ebenso verfahren wir mit Matrizen: Eine Matrix  $M \in A^{k \times l}$  mit  $A \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$  schreiben wir als

$$M = \begin{pmatrix} m_{0,0} & \cdots & m_{0,l-1} \\ \vdots & \ddots & \vdots \\ m_{k-1,0} & \cdots & m_{k-1,l-1} \end{pmatrix}.$$

Einen einzelnen Zeilenvektor von  $M$  schreiben wir auch  $m_{i,*}$  mit  $0 \leq i \leq k-1$ , für einen Spaltenvektor entsprechend  $m_{*,j}$  mit  $0 \leq j \leq l-1$ . Eine Matrix  $M$  befindet sich in der *reduzierten Zeilenstufenform*, wenn folgende Bedingungen erfüllt sind (vgl. z.B. [May00]):

- Alle Zeilen, die nur aus Nullen bestehen, stehen unter allen Zeilen, die nicht ausschließlich aus Nullen bestehen.
- Das *Pivotelement* (der erste Eintrag von links aus gesehen, der ungleich Null ist) jeder Zeile, die nicht ausschließlich Nullen enthält, ist 1.
- Das Pivotelement jeder Zeile, die nicht ausschließlich Nullen enthält, steht immer rechts vom Pivotelement der darüberstehenden Zeile.
- Eine Spalte, die das Pivotelement einer Zeile enthält, enthält keine weiteren von Null verschiedenen Einträge.

**Graphen- und Automatentheorie.** Einen *Graph* beschreiben wir als Paar  $G = (V, E)$ , wobei  $V$  die Menge der Knoten und  $E \subseteq (V \times V)$  die Menge der gerichteten Kanten des Graphen bezeichnet. Grafisch werden die Knoten des Graphen als Kreise und die Kanten als Pfeile dargestellt, wobei der Pfeil einer Kante  $(v_0, v_1) \in E$  von  $v_0$  nach  $v_1$  verläuft.

Einen *Weg* im Graphen von  $v_0$  nach  $v_n$  mit  $v_0, \dots, v_n \in V$  beschreiben wir durch  $\mathcal{W} = (v_0, \dots, v_n)$  als Folge von Knoten, so dass für alle  $v_k, v_{k+1}$  mit  $0 \leq k \leq n-1$  gilt:  $(v_k, v_{k+1}) \in E$ . Die Länge des Weges richtet sich nach der Anzahl der Kanten, die benötigt werden, um ihn zu durchlaufen, und beträgt dementsprechend  $|\mathcal{W}| = n$ . Die Folge  $((v_0, v_1), \dots, (v_{n-1}, v_n))$  der Kanten, welche die Knoten des Weges miteinander verbinden, geben wir nicht mit an, da sie in unseren Betrachtungen keine Rolle spielt.

Ein *Transitionssystem* beschreiben wir als Quadrupel  $\mathcal{T} = (S, I, A, E)$ , wobei  $S$  die Menge der Zustände,  $I \subseteq S$  eine nichtleere Menge von Anfangszuständen,  $A$  eine Menge von Aktionen sowie  $E \subseteq S \times A \times S$  die Zustandsübergangsrelation bezeichnet.

**Lineare Optimierung.** Ein *Lineares Programm* (LP), auch als lineares Optimierungsproblem bezeichnet, besteht aus einer Zielfunktion sowie Nebenbedingungen und Variablenbeschränkungen. Die Variablen bezeichnen wir gewöhnlicherweise mit dem Vektor  $x \in \mathbb{R}^n$ . Je nachdem, ob die Zielfunktion maximiert oder minimiert werden soll, wird sie als  $\max c^T x$  bzw.  $\min c^T x$  geschrieben mit  $c \in \mathbb{R}^n$ . Nebenbedingungen haben die Form  $Ax \leq b$ , wobei  $A \in \mathbb{R}^{m \times n}$  und  $b \in \mathbb{R}^m$  gilt und  $R$  eine Relation bezeichnet mit  $R \in \{\geq, \leq, =\}$ . Eine einzelne Nebenbedingung kann gemäß oben genannter Konvention auch als  $a_{i,*} x R b_i$  mit  $0 \leq i \leq m-1$  geschrieben werden. Einem Linearen Programm  $\mathcal{L}$  der Form

$$\begin{aligned} & \max c^T x \\ & \text{s.t. } Ax \leq b \\ & \quad x \geq 0 \end{aligned}$$

ist das *duale Problem*

$$\begin{aligned} & \min b^T y \\ & \text{s.t. } A^T y \geq c \\ & \quad y \geq 0 \end{aligned}$$

zugeordnet. Mit „s.t.“ (für „subject to“) werden hierbei die Nebenbedingungen und Variablenbeschränkungen eingeleitet. Der Lösungsvektor eines Linearen Programms ist der Vektor, für den die Zielfunktion ihr Optimum annimmt; wir bezeichnen ihn mit  $x^*$ . Die Lösungswerte  $y^*$  des dualen Problems werden auch als *reduzierte Kosten* der Nebenbedingungen bezeichnet.

**Weitere Bezeichnungen.** Ein *Mengenüberdeckungsproblem* (vgl. z.B. [Chv79]) definieren wir folgendermaßen: Sei  $M$  eine Menge und  $S_0, \dots, S_k$  Teilmengen von  $M$ . Dann wird eine Menge  $L \subseteq \{0, \dots, k\}$  gesucht mit  $\bigcup_{i \in L} S_i = M$ , so dass  $|L|$  minimal ist.

Eine *Partition* einer Menge  $M$  ist ein Mengensystem  $\mathcal{P} = \{M_0, \dots, M_n\}$ , für das folgende Eigenschaften gelten: Für alle  $i$  mit  $0 \leq i \leq n$  gilt  $M_i \subseteq M$  sowie  $M_i \neq \emptyset$ . Des Weiteren gilt  $\bigcup_{0 \leq i \leq n} M_i = M$  und  $M_i \cap M_j = \emptyset$  für  $i \neq j$ . Wir schreiben  $x \sim y$ , wenn ein  $M_i \in \mathcal{P}$  existiert, so dass  $\{x, y\} \subseteq M_i$  gilt.

Mit  $[x]$  bezeichnen wir das Runden einer Zahl  $x \in \mathbb{R}$  zur nächsten ganzen Zahl  $z \in \mathbb{Z}$ ; es gilt  $[x] = \max \{z \in \mathbb{Z} \mid z \leq x + 0.5\}$ .

## 2.2 Petrinetze

In diesem Abschnitt werden wir Petrinetze in dem Maße definieren, wie es für die vorliegende Arbeit notwendig ist. Für eine ausführliche Einführung in Petrinetze und deren Analyse sei auf [Rei91] sowie [Sta90] verwiesen.

Wir beschreiben ein *Petrinetz*  $N$  als 5-Tupel  $N = (P, T, F, W, s_0)$ , wobei

- $P = \{p_0, \dots, p_n\}$  und  $T = \{t_0, \dots, t_k\}$  disjunkte endliche Mengen sind.  $P$  enthält die Plätze und  $T$  die Transitionen des Petrinetzes.
- die Relation  $F \subseteq (P \times T) \cup (T \times P)$  die Kanten zwischen Plätzen und Transitionen enthält.
- die Funktion  $W : F \rightarrow \mathbb{N} \setminus \{0\}$  den Kanten Gewichte zuordnet. Für  $(x, y) \notin F$  wird  $W(x, y) = 0$  gesetzt.
- $s_0$  die Anfangsmarkierung des Petrinetzes repräsentiert. Die Markierung eines Petrinetzes beschreiben wir durch die Funktion  $s : P \rightarrow \mathbb{N}$ . Wir schreiben eine Markierung  $s$  auch als Vektor der Form  $s = (s(p_0), \dots, s(p_n))$ .

Die Menge  $X = P \cup T$  bezeichnen wir als die Menge aller *Netzknoten*. Zu jedem Netzknoten  $x \in X$  können wir den *Vorbereich*  $\bullet x = \{y \in X \mid (y, x) \in F\}$  sowie den *Nachbereich*  $x^\bullet = \{y \in X \mid (x, y) \in F\}$  bestimmen. Ist  $x$  ein Platz, bezeichnen wir die Elemente dieser Mengen als Vor- bzw. Nachtransitionen. Handelt es sich bei  $x$  um eine Transition, nennen wir die Elemente dementsprechend Vor- und Nachplätze. Ein Platz ist ein *geteilter Vorplatz*, wenn er mehr als eine Nachtransition hat.

Eine Transition  $t_i$  ist *aktiviert*, wenn für alle Vorplätze  $p_j$  von  $t_i$  gilt:  $s(p_j) \geq W(p_j, t_i)$ . Eine Transition, die aktiviert ist, kann *schalten*. Beim Schalten einer Transition werden für jeden Vorplatz  $p_j$  von  $t_i$   $W(p_j, t_i)$  Marken verbraucht und für jeden Nachplatz  $p_j$  von  $t_i$   $W(t_i, p_j)$  Marken erzeugt. Kann eine Transition bei keiner Markierung eines Netzes schalten, bezeichnet man sie als *tot*. Eine Markierung, bei der keine Transition aktiviert ist, wird als *Deadlock* bezeichnet. Der *Transitionsvektor*  $\Delta t_i$  zu einer Transition  $t_i$  gibt an, wieviele Marken  $t_i$  beim Schalten auf den einzelnen Plätzen in der Summe verbraucht bzw. erzeugt werden. Er besteht dementsprechend aus  $n+1$  Komponenten  $\Delta t_{i,j}$ , wobei der Index  $j$  dem des Platzes  $p_j$  entspricht. Der Wert von  $\Delta t_{i,j}$  wird durch  $\Delta t_{i,j} = W(t_i, p_j) - W(p_j, t_i)$  berechnet. Aus der Markierung  $s$  kann die *Folgemarkierung*, die durch das Schalten der Transition  $t_i$  bei der Markierung  $s$  entsteht, demgemäß durch  $s' = s + \Delta t_i$  angegeben werden.

Eine *Schaltsequenz*  $\mathcal{S} = (t_0 \dots t_m)$  ist eine Folge von Transitionen, die nacheinander schalten können, wobei die Transitionen nicht paarweise verschieden sein müssen. Die Länge der Schaltsequenz beträgt  $|\mathcal{S}| = m + 1$ . Die *Inzidenzmatrix* eines Petrinetzes ist die Matrix  $C$ , deren Spalten die Transitionsvektoren bilden und in deren  $i$ -ter Spalte der Transitionsvektor  $\Delta t_i$  zu finden ist. Es gilt also  $c_{*,i} = \Delta t_i$ .

Eine *T-Invariante* (kurz für *Transitionsinvariante*) eines Petrinetzes ist eine nicht-triviale, ganzzahlige Lösung  $x$  des Gleichungssystems  $C \cdot x = 0$ . Eine T-Invariante wird *echte T-Invariante* genannt, wenn alle Komponenten von  $x$  nichtnegativ sind. Mit *supp*( $x$ ) (*Träger* von  $x$ ) bezeichnen wir die Menge, die die Indizes aller Komponenten  $x_i$  von  $x$  enthält, welche positiv sind. Eine echte T-Invariante  $x$  wird als *minimal* bezeichnet, wenn keine echte T-Invariante  $y$  mit  $\text{supp}(y) \subset \text{supp}(x)$  existiert.

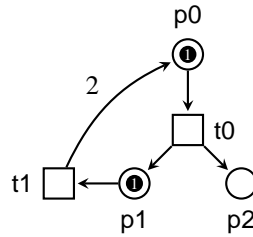


Abbildung 2.1: Petrinetz

Als *Erreichbarkeitsgraph*<sup>1</sup> zu einem Petrinetz bezeichnen wir das Transitionssystem  $\mathcal{T} = (S, I, A, E)$ , wobei  $S$  der Menge aller erreichbaren Markierungen entspricht und  $I = \{s_0\}$  sowie  $A = T$  und folglich  $E \subseteq S \times T \times S$  gilt. Wenn eine Transition  $t_i$  bei der Markierung  $s \in S$  aktiviert ist und das Schalten dieser Transition zum Zustand  $s'$  führt, gilt  $(s, t_i, s') \in E$ .  $s'$  wird dementsprechend als *Folgezustand* von  $s$  bezeichnet. Alle Zustände, die im Erreichbarkeitsgraphen von  $s$  aus erreichbar sind, bezeichnen wir als *transitive Folgezustände*.

Petrinetze werden grafisch dargestellt, indem Plätze als Kreise, Transitionen als Quadrate und Elemente der Kantenrelation als Pfeile gezeichnet werden. Kantengewichte, die ungleich 1 sind, werden an den jeweiligen Kanten notiert. Die Anfangsmarkierung wird, so wie sie für unsere Betrachtungen benötigen, als schwarzer Kreis mit der entsprechenden Anzahl der Marken innerhalb des jeweiligen Platzes dargestellt. Erreichbarkeitsgraphen stellen wir dar, indem wir Zustände als Kreise und Zustandsübergänge als mit Transitionsnamen beschriftete Pfeile zeichnen. Der Anfangszustand wird immer mit  $s_0$  bezeichnet. In allen Abbildungen entfällt die Tiefstellung der Indizes.

**Beispiel 1.** In Abb. 2.1 ist ein Petrinetz mit den Plätzen  $P = \{p_0, p_1, p_2\}$ , den Transitionen  $T = \{t_0, t_1\}$  und den Kanten  $F = \{(p_0, t_0), (t_0, p_1), (t_0, p_2), (p_1, t_1), (t_1, p_0)\}$  dargestellt. Es gilt  $W(t_1, p_0) = 2$ ; für alle anderen Kanten  $(x, y) \in F$  gilt  $W(x, y) = 1$ . Die Anfangsmarkierung des Netzes ist  $s_0 = (1, 1, 0)$ .

\*

## 2.3 Die Sweep-Line-Methode

Bei der Sweep-Line-Methode handelt es sich um eine Technik zur Reduktion von Speicherplatzbedarf bei der Durchmusterung von Transitionssystemen. Sie wurde in der in dieser Arbeit verwendeten Form erstmals in [KM02] beschrieben. Sie kann bei der Überprüfung von Sicherheitseigenschaften wie z.B. der Existenz von Deadlocks, der Erreichbarkeit einer gegebenen Markierung und der Existenz toter Transitionen verwendet werden. Wir wollen die Sweep-Line-Methode auf Erreichbarkeitsgraphen von Petrinetzen anwenden. Da jeder Er-

<sup>1</sup>Die Bezeichnung ist gängigen Konventionen geschuldet. Der Erreichbarkeitsgraph ist nicht als Graph im Sinne der Definition in Kapitel 2.1 zu betrachten.

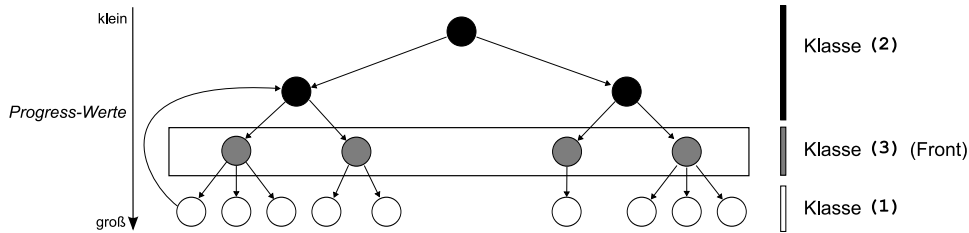


Abbildung 2.2: Eine Momentaufnahme der Zustandsraumdurchmusterung durch Breitensuche mit der Sweep-Line-Methode [Pru09]

reichbarkeitsgraph ein Transitionssystem ist, werden wir im Folgenden mit Ausnahme von Satz 1 auf den Begriff des Transitionssystems verzichten und stattdessen nur den Begriff des Erreichbarkeitsgraphen verwenden. Der Grundgedanke der Sweep-Line-Methode ist, dass jedem Erreichbarkeitsgraphen eine Art Fortschritt zugrunde liegt. Folglich können Zustände, die für die weitere Durchmusterung nicht mehr benötigt werden, aus dem Speicher gelöscht werden. Die Zustände, die gelöscht werden können, sind genau diejenigen, deren Folgezustände bereits alle durchmustert wurden. Wir können deshalb in einem System drei Klassen von Zuständen unterscheiden: (1) Zustände, die noch nicht gefunden wurden; (2) Zustände, die gefunden wurden und deren Folgezustände alle durchmustert wurden; (3) Zustände, die gefunden wurden, deren Folgezustände aber noch nicht alle durchmustert wurden. Die Zustände der Klasse (3) werden im Folgenden auch als *Front* bezeichnet. In Abb. 2.2 sind die drei Klassen anhand eines Beispiels dargestellt.

Um den Fortschritt bei der Durchmusterung des Erreichbarkeitsgraphen bewerten zu können, benötigen wir ein Maß, das jedem Zustand einen Wert zuweist, der diesen Fortschritt angibt. Dieses Maß wird als *Progress Measure* bezeichnet.

**Definition 1** (Progress Measure, Progress-Wert). Sei  $\mathcal{T} = (S, I, A, E)$  ein Erreichbarkeitsgraph. Mit **Progress Measure** bezeichnen wir die Funktion  $p : S \rightarrow \{\mathbb{N}, \mathbb{Q}\}$ . Den Wert  $p(s)$  eines bestimmten Zustandes  $s \in S$  bezeichnen wir als **Progress-Wert** von  $s$ .

In [Mai03] wird die Progress Measure für beliebige partiell geordnete Zielmenge definiert. Wir haben sie hier an unsere Zwecke angepasst. In den Originalpublikationen zur Sweep-Line-Methode [KM02, Mai03] bleibt es die Aufgabe des Anwenders, eine Progress Measure anzugeben. In Abschnitt 2.4 werden wir eine Vorgehensweise zur automatischen Berechnung von Progress-Werten im Kontext von Petrinetzen vorstellen.

Nun können wir die Funktionsweise der Sweep-Line-Methode beschreiben:

Gehen wir zunächst davon aus, dass die Progress Measure monoton ist. Für einen Erreichbarkeitsgraphen  $\mathcal{T} = (S, I, A, E)$  mit  $(s, t_i, s') \in E$  soll also gelten:  $p(s) \leq p(s')$ . Ist der Progress-Wert eines Zustandes kleiner als der minimale Progress-Wert der Zustände der Front, gehört der entsprechende Zustand zur Klasse (2) und kann aufgrund der Monotonie der Progress Measure gelöscht werden. Je weiter die Durchmusterung des Zustandsraumes voranschreitet, desto größer wird der minimale Progress-Wert der Zustände der Front und desto mehr Zustände können gelöscht werden. Diese Variante wurde bereits in [CKM01] beschrieben und wird auch als „Basisvariante“ der Sweep-Line-Methode bezeichnet.

Allerdings funktioniert die Basisvariante nicht für Erreichbarkeitsgraphen, die Kanten enthal-

## 2 Grundlagen

ten, welche von Zuständen der Klasse (1) oder (3) zu Zuständen der Klasse (2) führen (siehe Abb. 2.2). Da der Zielzustand einer solchen Kante bereits aus dem Speicher gelöscht wurde, existiert keine Information mehr darüber, dass er bereits gefunden wurde. Somit wird dieser Zielzustand sowie alle seine Folgezustände und transitiven Folgezustände erneut durchmustert. Weil die Durchmusterung auf diese Weise wiederum den betrachteten Zielzustand als neuen Zustand finden wird, terminiert die Basisvariante für solche Erreichbarkeitsgraphen nicht.

Damit die Sweep-Line-Methode auf alle Erreichbarkeitsgraphen angewendet werden kann, ohne dass sie in eine Endlosschleife gerät, wurde die Basisvariante zur „verallgemeinerten Sweep-Line-Methode“ weiterentwickelt. Bei dieser Variante muss die Progress Measure nicht mehr monoton sein. Eine Kante, deren Zielzustand einen geringeren Progress-Wert hat als ihr Ausgangszustand, wird als *Regresskante* bezeichnet. Aufgrund dieser Eigenschaft können Zielzustände von Regresskanten während der Durchmusterung als solche erkannt werden. Sie werden als *persistent* markiert. Persistente Zustände werden im aktuellen Durchlauf nicht weiter durchmustert, werden aber im Speicher gehalten. Sofern im aktuellen Durchlauf neue persistente Zustände gefunden wurden, wird eine neue Iteration mit genau diesen Zuständen als Startzuständen gestartet. Auf diese Weise wird sichergestellt, dass alle Zustände des Erreichbarkeitsgraphen mindestens einmal durchmustert werden. Es kann jedoch vorkommen, dass Zustände mehrmals durchmustert werden. Satz 1 kann entnommen werden, dass die Anzahl der maximalen Iterationen der Sweep-Line-Methode proportional zur Anzahl der Regresskanten steigt.

**Satz 1** ([Mai03]). *Sei  $s$  ein Zustand in einem Transitionssystem und  $reach(s)$  die Menge aller Zustände, die von  $s$  aus erreichbar sind. Wenn in einem Transitionssystem alle erreichbaren Zustände mit dem Durchlaufen von maximal  $n$  Regresskanten erreicht werden können, dann terminiert der Algorithmus nach  $n + 2$  Iterationen. Dabei hat er höchstens  $(n + 2) \cdot |reach(s)|$  Zustände durchmustert.*

Die verallgemeinerte Sweep-Line-Methode ist in Algorithmus 1 als Pseudocode angegeben. Hierbei werden in *Roots* die Ausgangszustände der jeweiligen Iteration gespeichert und in *Persistent* die persistenten Zustände. In *U* befinden sich alle gefundenen Zustände, die während der aktuellen Iteration noch zu durchmustern sind. Die zu durchmusternden Zustände werden aufsteigend nach ihrem Progress-Wert aus *U* ausgewählt. Wurde ein Zustand  $s$  durchmustert, wird er solange in der Menge *R* gespeichert, bis kein Zustand aus *U* mehr existiert, dessen Progress-Wert kleiner oder gleich dem Progress-Wert von  $s$  ist. Jeder Zustand, der während einer Iteration neu als persistent markiert wird, wird in *Roots* gespeichert und ist somit Ausgangszustand der nächsten Iteration.

Die Sweep-Line-Methode kann mit *Stubborn Sets*, einer weiteren Methode zur Reduktion von Erreichbarkeitsgraphen, kombiniert werden. Wir werden die Kombination beider Methoden in dieser Arbeit anhand der Fallstudie in Abschnitt 4.4 betrachten. Für eine Einführung in Stubborn Sets sei auf [Val96] verwiesen. Eine Berechnungsmethode für Petrinetze, die in Kombination mit der Sweep-Line-Methode genutzt werden kann, findet man in [Sch99].

---

**Algorithmus 1** Die verallgemeinerte Sweep-Line-Methode (vgl. [Mai03])

---

Setze  $R = \{s_0\}$ ,  $Roots = \{s_0\}$ ,  $Persistent = \emptyset$

Solange  $Roots \neq \emptyset$ :

Führe  $SWEEP(Roots, R, Persistent)$  aus.

Prozedur  $SWEEP(Roots, R, Persistent)$ :

Setze  $U = Roots$ .

Setze  $Roots = \emptyset$ .

Solange  $U \neq \emptyset$ :

Wähle  $s \in U$ , so dass  $\neg \exists s' \in U : p(s') < p(s)$ .

Setze  $U = U \setminus \{s\}$ .

Für alle  $s', t_i$ , für die  $s' = s + \Delta t_i$  gilt:

Wenn  $s' \notin R$ :

Setze  $R = R \cup \{s'\}$ .

Wenn  $p(s) \leq p(s')$ :

Setze  $U = U \cup \{s'\}$ .

sonst:

Setze  $Persistent = Persistent \cup \{s'\}$ .

Setze  $Roots = Roots \cup \{s'\}$ .

Setze  $R := \{s \in R \mid \exists s' \in U : p(s') \leq p(s)\} \cup Persistent$ .

---

## 2.4 Algebraische Berechnung von Progress-Werten

Da die Erzeugung einer Progress Measure in den Originalpublikationen zur Sweep-Line-Methode [Mai03, KM02] dem Anwender überlassen wird, ist es nötig, eine automatische Berechnung von Progress-Werten bereitzustellen, wenn die Sweep-Line-Methode vollständig automatisiert genutzt werden soll. In [Sch04] wird eine solche automatische Berechnung für die Anwendung der Sweep-Line-Methode auf Erreichbarkeitsgraphen von Petrinetzen vorgestellt. Diese wollen wir hier betrachten, da sie die Grundlage unserer weiteren Ausführungen bildet. In aktuellen Versionen des Model-Checking-Tools LoLA [Sch00] ist diese Methode bereits implementiert.

Der Grundgedanke der hier vorgestellten Berechnung ist es, jeder Transition  $t_i$  bereits vor dem Aufbau des Graphen einen festen Wert, den *Offset-Wert*  $o(t_i)$ , zuzuweisen, um dann die Progress-Werte der Zustände des Erreichbarkeitsgraphen während der Zustandsraumdurchmusterung aus diesen Offset-Werten zu kombinieren.

**Definition 2** (Offset-Wert). Sei  $N = (P, T, F, W, s_0)$  ein Petrinetz. Der **Offset-Wert** einer Transition  $t_i \in T$  ist der Wert der Funktion  $o : T \rightarrow A$  mit  $A \in \{\mathbb{Z}, \mathbb{Q}\}$ .

Zur Berechnung der Offset-Werte wird zunächst eine größte Menge von Transitionen bestimmt, deren Transitionsvektoren linear unabhängig sind. Diese bezeichnen wir mit  $U$ . Weil die zu den Elementen aus  $U$  zugehörigen Transitionsvektoren auch als Basis eines Vektorraumes aufgefasst werden können, bezeichnen wir diese Menge im Folgenden auch als *Basis*. Für jede Transition  $t_j$ , die in  $U$  enthalten ist, setzen wir nun  $o(t_j) = 1$ . Da  $U$  maximal gewählt

wurde, kann jeder Transitionsvektor  $\Delta t_i$ , für den  $t_i \notin U$  gilt, durch die Linearkombination  $\Delta t_i = \lambda_0 \Delta t_0 + \dots + \lambda_n \Delta t_n$  mit  $U = \{t_0, \dots, t_n\}$  berechnet werden. Folglich wird der Offset-Wert für alle Transitionen  $t_i$  mit  $t_i \notin U$  durch  $o(t_i) = \lambda_0 o(t_0) + \dots + \lambda_n o(t_n)$  berechnet. Es ist offensichtlich, dass mit diesem Verfahren stets ganzzahlige Offset-Werte berechnet werden. In Kapitel 3 werden wir eine Abwandlung dieser Berechnungsmethode vorstellen, mit der auch nicht-ganzzahlige Offset-Werte berechnet werden können.

Der Progress-Wert  $p(s)$  bezüglich eines Zustands  $s$  des Erreichbarkeitsgraphen eines gegebenen Petrinetzes wird nun folgendermaßen berechnet:

Für die Anfangsmarkierung  $s_0$  des Petrinetzes gilt  $p(s_0) = 0$ . Für jeden Folgezustand  $s' = s + \Delta t_i$ , der durch das Schalten der Transition  $t_i$  im Zustand  $s$  entsteht, gilt  $p(s') = p(s) + o(t_i)$ . Durch die hier beschriebene Berechnungsmethode wird gewährleistet, dass ein für einen beliebigen Zustand  $s$  einmal berechneter Progress-Wert  $p(s)$  auch dann konstant ist, wenn  $s$  über verschiedene Schaltsequenzen erreichbar ist und  $p(s)$  demnach durch verschiedene Terme berechnet werden kann. Wir bezeichnen die Progress Measure daher als *konsistent*.

### 2.4.1 Geometrische Interpretation

In [Sch04] wird eine geometrische Betrachtungsweise der Berechnung erläutert, die wir an dieser Stelle wiedergeben wollen.

Die Markierungen können als Punkte im euklidischen Raum  $\mathbb{Q}^{|P|}$  betrachtet werden. Für jede Transition  $t_i$  ist dann ihr zugehöriger Transitionsvektor  $\Delta t_i$  ein Vektor in diesem Raum. Ist  $s$  eine Markierung und schaltet die Transition  $t_i$  bei  $s$ , entsteht die Markierung  $s'$  durch die Verschiebung von  $s$  um  $\Delta t_i$ . Dies entspricht der oben genannten Gleichung  $s' = s + \Delta t_i$ .

Die Transitionen aus  $U$  lassen sich ebenfalls als Punkte im Raum  $\mathbb{Q}^{|P|}$  betrachten, nämlich als Verschiebung des Punktes  $\underline{0}$  um ihren Transitionsvektor. Die kleinste Hyperebene  $E$ , die diese Punkte enthält, wird genau durch die entsprechenden Transitionsvektoren aufgespannt.

Nun kann der Progress-Wert für eine gegebene Markierung  $s$  geometrisch interpretiert werden: Sei  $d$  der Punkt der Hyperebene  $E$ , der den kleinsten Abstand zu  $\underline{0}$  besitzt und  $g$  die Gerade, die durch  $\underline{0}$  und  $d$  verläuft. Sei  $E'$  die zu  $E$  parallele Hyperebene, die  $s$  enthält. Jetzt kann der Schnittpunkt  $i$  der Geraden  $g$  mit der Hyperebene  $E'$  definiert werden. Der Progress-Wert von  $s$  ist der Abstand von  $\underline{0}$  zum Punkt  $i$ , wobei der Progress-Wert 1 durch den Abstand zwischen  $\underline{0}$  und  $d$  definiert ist.

In Abb. 2.3 ist die geometrische Interpretation beispielhaft für ein Netz mit 6 Transitionen und 2 Plätzen dargestellt.

### 2.4.2 Ansätze zur Optimierung

In [Sch04] werden einige Möglichkeiten zur Optimierung der vorgestellten Berechnungsmethode erwähnt, die wir in diesem Abschnitt betrachten wollen.

Zunächst kann festgestellt werden, dass zwei Freiheitsgrade existieren: Die Wahl der Basis  $U$  sowie die Zuweisung eines Offset-Wertes für jede Transition aus  $U$ . Wie oben erwähnt wurde, wird bisher jeder Transition aus  $U$  der Offset-Wert 1 zugewiesen – dies ist jedoch eine recht willkürliche Festlegung, da die Progress Measure nach dem genannten Berechnungsverfahren auch dann konsistent ist, wenn man den Transitionen aus  $U$  beliebige Offset-Werte zuweist.

Auf Grundlage dieser Freiheitsgrade ist es möglich, die beschriebene Berechnungsmethode zu optimieren. In [Sch04] werden zwei Optimierungsziele vorgeschlagen, die die Leistung der



Iteration noch einmal durchmustert, jetzt befinden sich aber bereits zu Beginn dieser Iteration  $k$  Zustände im Speicher – für die erneute Durchmusterung des Zustandsraumes von  $s_0$  aus steht nun weniger ungenutzter Speicher zur Verfügung als beim ersten Durchlauf. Man kann an diesem Beispiel erkennen, dass im schlechtesten Fall bei der erneuten Durchmusterung eines Teils des Zustandsraumes der verfügbare Speicherplatz überschritten werden kann; dies gilt es zu vermeiden.

Nachdem wir bisher die Sweep-Line-Methode und die algebraische Berechnung der Offset-Werte beschrieben haben, werden in den folgenden beiden Abschnitten zwei Probleme eingeführt, die zunächst keinen Bezug zur Sweep-Line-Methode haben, aber innerhalb der in dieser Arbeit entwickelten Heuristik zur Optimierung der Berechnung der Offset-Werte eine Rolle spielen. Es handelt sich hierbei um das *Maximum Feasible Subsystem Problem*, das eng mit linearer Optimierung verknüpft ist, sowie das *All Pairs Shortest Paths Problem* aus der Graphentheorie.

### 2.5 Das Maximum Feasible Subsystem Problem

Das *Maximum Feasible Subsystem Problem* (MAXFS) hat die Frage zum Gegenstand, wie aus einem unzulässigen System von Ungleichungen  $Ax \leq b$  mit  $A \in \mathbb{R}^{m \times n}$  und  $b \in \mathbb{R}^m$  ein zulässiges Teilsystem gefunden werden kann, das so viele Ungleichungen wie möglich enthält. Sucht man die minimale Anzahl an Nebenbedingungen, die aus einem System von Ungleichungen entfernt werden müssen, um ein zulässiges System zu erhalten, wird das Problem als *Minimum Unsatisfied Linear Relation Problem* (MINULR) [AK94] bezeichnet. Auf Grundlage folgender Definition kann eine weitere Betrachtungsweise auf das Problem gewonnen werden.

**Definition 3** (IIS). Seien  $N$  die Nebenbedingungen eines Linearen Programms  $\mathcal{L}$  und gelte  $I \subseteq N$ .  $I$  ist eine **irreduzible, unzulässige Menge von Nebenbedingungen** (IIS für „irreducible infeasible set“), wenn  $I$  eine unzulässige Menge von Nebenbedingungen bildet und jede echte Teilmenge von  $I$  eine Menge zulässiger Nebenbedingungen ist.

Eine Menge unzulässiger Nebenbedingungen kann in Bezug auf ihre Kardinalität eine exponentielle Anzahl an IIS enthalten [Cha94].

Wird aus einem IIS eine beliebige Nebenbedingung entfernt, erhält man ein zulässiges System von Ungleichungen. Versucht man nun, eine Menge von Ungleichungen minimaler Kardinalität zu finden, so dass in ihr aus jedem IIS des betrachteten Systems mindestens eine Ungleichung enthalten ist, bezeichnet man das Problem als *Minimum-Cardinality IIS Set Covering Problem* (MIN IIS COVER) [Chi96]. Dieses Problem ist identisch zu MINULR. Da MAXFS NP-schwer ist [AK95, Cha94], wurden spezielle Algorithmen entwickelt, um MAXFS-Instanzen exakt oder näherungsweise zu lösen. Der folgende Abschnitt soll einen Überblick über einige dieser Methoden geben.

#### 2.5.1 Algorithmische Lösungsmethoden

Zur exakten sowie zur approximativen Lösung von MAXFS-Instanzen wurden einige Methoden vorgeschlagen, über die wir an dieser Stelle einen Überblick geben wollen, wobei wir uns auf den möglichen Nutzen bezüglich unserer Anwendung konzentrieren werden. Für eine detailliertere Einführung in MAXFS-Lösungsmethoden sei auf [Chi08] verwiesen.

MAXFS-Instanzen können als gemischt-ganzzahliges Programm formuliert und exakt gelöst werden [GM91]. Da jedoch bei der Formulierung dieses Optimierungsproblems gewisse Freiheitsgrade existieren, kann es zu Ungenauigkeiten und numerischen Problemen bei der Lösung des gemischt-ganzzahligen Programmes kommen.

In [Pfe08] wird ein auf einem speziellen Branch-and-Cut-Algorithmus [CF04] basierendes Verfahren entwickelt, das exakte Lösungen in kürzerer Zeit als die eben genannte Methode liefert. Die in [Pfe08] aufgeführten Messungen lassen jedoch den Schluss zu, dass dieses Verfahren für große Systeme trotzdem eine für unsere Zwecke unangemessen hohe Zeitspanne zum Auffinden einer exakten Lösung benötigt.

Viele Algorithmen wurden für die Lösung von MIN IIS COVER-Instanzen entwickelt. In [PR96] wird vorgeschlagen, zunächst alle IISs eines Systems aufzuzählen und anschließend mittels einer Heuristik ein Mengenüberdeckungsproblem über diesen zu lösen.

Eine Zwei-Phasen-Heuristik, bei der zunächst eine Relaxierung des MAXFS-Problems gelöst wird und anschließend versucht wird, die Lösung durch eine exakte Formulierung des Problems zu verbessern, wird in [ABC08] vorgestellt. Sowohl die in [PR96] als auch die in [ABC08] vorgestellte Lösungsmethode könnten für unsere Anwendung geeignet sein.

In [Chi96] und [Chi01] wird eine LP-basierte Heuristik zur Lösung von MAXFS-Instanzen entwickelt, die trotz kurzer Laufzeiten relativ genaue Resultate liefert. Aufgrund dieser Eigenschaften erscheint sie uns für unser Problem sehr gut geeignet. Diese Methode ist Bestandteil unserer Implementation und wird im nächsten Abschnitt näher betrachtet. Wir haben sie den in [ABC08] und [PR96] beschriebenen Methoden zunächst auch deshalb vorgezogen, da anhand des Algorithmus' unmittelbar erkennbar ist, dass der Speicherplatzbedarf dieser Methode auch bei der Anwendung auf große MAXFS-Instanzen relativ gering ist. Da sie ihren Zweck sehr gut erfüllt, wie wir in der Fallstudie in Abschnitt 4.4 sehen werden, haben wir keine Veranlassung gesehen, zum Vergleich andere Methoden zu implementieren.

Des Weiteren wurden Heuristiken zur Lösung von sehr großen MAXFS-Instanzen entwickelt. Die in [ABH05] entwickelte Heuristik, welche Relaxierung mit probabilistischen Algorithmen verbindet, lieferte in einer von uns vorgenommenen prototypischen Implementierung zu ungenaue Ergebnisse. In [MWE02] wurde im Kontext der Berechnung von Proteinfaltungspotentialen eine auf Innere-Punkte-Verfahren basierende Heuristik entwickelt, die jedoch aus methodischen Gründen für die Ungleichungssysteme, die bei unserer Anwendung auftreten, ungeeignet erscheint und keine brauchbaren Resultate liefern dürfte.

### 2.5.2 LP-basierte Heuristik zur Lösung von MIN IIS COVER-Instanzen

Im vorherigen Abschnitt haben wir bereits die LP-basierte MIN IIS COVER-Heuristik von Chinneck [Chi01] erwähnt. Diese wollen wir hier näher betrachten.

Der Autor stellt in [Chi01] mehrere Heuristiken zur näherungsweise Lösung von MIN IIS COVER-Instanzen vor. Wir werden uns hier auf eine dieser Heuristiken konzentrieren. Diese wurde bereits in [Chi96] entwickelt und in [Chi01] noch einmal in einer überarbeiteten Version vorgestellt. Sie bildet die Grundlage für alle weiteren in [Chi01] entwickelten Heuristiken. Diese verbessern zwar die Laufzeit des Algorithmus', führen jedoch oft zu ungenaueren Resultaten. Aus diesem Grund haben wir uns für die Verwendung der Heuristik entschieden, die nun vorgestellt werden soll.

## 2 Grundlagen

Bevor wir mit der Beschreibung der Heuristik beginnen, müssen wir zunächst den Begriff des *elastischen Linearen Programms* klären.

**Definition 4** (Elastisches Lineares Programm). *Gegeben sei ein lineares Programm  $\mathcal{L}$ . Das zugehörige **elastische Lineare Programm** (eLP)  $\mathcal{EL}$  wird nach folgenden Regeln erstellt:*

1. Für jede Nebenbedingung  $a_{i,*}x \leq b_i$  in  $\mathcal{L}$  tue folgendes:

- Wenn  $R = \geq$ : Füge die Nebenbedingung  $a_{i,*}x + e_i \geq b_i$  in  $\mathcal{EL}$  ein.
- Wenn  $R = \leq$ : Füge die Nebenbedingung  $a_{i,*}x - e_i \geq b_i$  in  $\mathcal{EL}$  ein.
- Wenn  $R = =$ : Füge die Nebenbedingung  $a_{i,*}x + e'_i - e''_i \geq b_i$  in  $\mathcal{EL}$  ein.

2. Für alle  $e_i, e'_i, e''_i$  soll gelten:  $e_i \geq 0 \wedge e'_i \geq 0 \wedge e''_i \geq 0$ .

3. Die Zielfunktion von  $\mathcal{EL}$  ist  $\min \sum_i e_i + e'_i + e''_i$ .

Die Nebenbedingungen des eLPs gleichen also bis auf einen Unterschied den Nebenbedingungen des LPs: In jeder Nebenbedingung des eLPs wird eine *elastische Variable*  $e_i \in \mathbb{R}$  eingefügt, falls es sich um eine Ungleichung handelt und zwei elastische Variablen  $e'_i, e''_i \in \mathbb{R}$ , falls es sich um eine Gleichung handelt. Dies geschieht genau so, dass die Menge der Nebenbedingungen des eLPs in jedem Fall zulässig ist. Sollte die Menge der Nebenbedingungen des LPs unzulässig sein, wird die Menge der Nebenbedingungen des eLPs dann zulässig, wenn die Werte der elastischen Variablen hinreichend groß gewählt werden.

Die Summe der elastischen Variablen soll nun minimiert werden, weil aus dem Ergebnis der Optimierung Rückschlüsse auf die Zulässigkeit des LPs gezogen werden können. Gilt nach der Optimierung des eLPs  $\sum_i x_i^* = 0$ , ist das LP zulässig. Gilt hingegen  $\sum_i x_i^* \neq 0$ , ist das LP unzulässig. Da die Zielfunktion in diesem Fall nicht zu 0 minimiert werden konnte, gibt es mindestens eine Nebenbedingung im LP, die es unzulässig werden lässt. Der Wert  $\sum_i x_i^*$  wird im Folgenden als *SINF* (*sum of infeasibilities*) bezeichnet. Die Anzahl der elastischen Variablen, deren Wert nicht zu 0 minimiert werden konnte, wird *NINF* (*number of infeasibilities*) genannt.

Mit Hilfe von elastischen Programmen kann nun die MIN IIS COVER-Heuristik entwickelt werden. Das Berechnungsverfahren ist in Algorithmus 2 dargestellt. Ziel ist es, in der Menge *CoverSet* alle Nebenbedingungen zu speichern, die zur MIN IIS COVER-Menge gehören.

Als Eingabe für den Algorithmus werden die Nebenbedingungen des LPs erwartet, zu dem die MIN IIS COVER-Menge berechnet werden soll. Zu Beginn des Algorithmus wird das zu diesem LP zugehörige eLP gelöst. Konnte der Wert von genau einer elastischen Variablen nicht zu 0 minimiert werden, ist die dieser Variablen zugeordnete Nebenbedingung die einzige, die zur *CoverSet*-Menge gehört. Der Algorithmus kann dann bereits an dieser Stelle terminieren. Gilt hingegen  $NINF \neq 1$ , werden alle Nebenbedingungen, deren reduzierte Kosten ungleich 0 sind (deren Entfernen also einen Einfluss auf den Wert der Zielfunktion hätte), als potentielle Elemente der MIN IIS COVER-Menge in der Variable *HoldSet* gespeichert.

Die Menge *CandidateSet* erhält nun zunächst alle Elemente der Menge *HoldSet*. Nun wird für jede Nebenbedingung  $N$ , die in *CandidateSet* gespeichert wurde, getestet, wie stark sich der *SINF*-Wert des eLPs verringert, wenn  $N$  aus dem elastischen Programm entfernt wird. Dazu wird  $N$  temporär aus dem eLP entfernt und das eLP noch einmal gelöst. Sollte der Wert von *SINF* hierdurch auf 0 reduziert werden, muss außer  $N$  keine weitere Nebenbedingung entfernt werden, um eine MIN IIS COVER-Lösung zu erhalten; Der Algorithmus kann dann terminieren.

---

**Algorithmus 2** Chinneck's MIN IIS COVER-Heuristik (vgl. [Chi01])

---

EINGABE: Nebenbedingungen eines (unzulässigen) LPs.

Setze  $CoverSet = \emptyset$ .  
 Erstelle das zum LP zugehörige eLP.  
 Löse das eLP.  
 Wenn  $NINF = 1$  gilt:  
     Füge die Nebenbedingung, die das LP unzulässig werden lässt, zu  $CoverSet$  hinzu.  
     Terminiere.  
 Setze  $HoldSet = \{ \text{alle Nebenbedingungen, deren reduzierte Kosten ungleich 0 sind} \}$ .

1: Setze  $CandidateSet = HoldSet$ .  
 Setze  $MINSINF = \emptyset$ .  
 Für jede Nebenbedingung  $N$  in  $CandidateSet$ :  
     Lösche  $N$  aus dem eLP.  
     Löse das eLP.  
     Wenn  $SINF = 0$  gilt:  
         Füge  $N$  zu  $CoverSet$  hinzu.  
         Terminiere.  
     Wenn  $SINF < MINSINF$  gilt:  
         Setze  $winner = N$ .  
         Setze  $MINSINF = SINF$ .  
         Setze  $HoldSet = \{ \text{alle Nebenbedingungen, deren reduzierte Kosten ungleich 0 sind} \}$ .  
         Wenn  $NINF = 1$ :  
             Setze  $nextwinner = \text{die Nebenbedingung, die das LP unzulässig werden lässt}$ .  
         sonst:  
             Setze  $nextwinner = \emptyset$ .  
     Füge  $N$  wieder in das eLP ein.  
 Füge  $winner$  zu  $CoverSet$  hinzu.  
 Wenn  $nextwinner \neq \emptyset$  gilt:  
     Füge  $nextwinner$  zu  $CoverSet$  hinzu.  
     Terminiere.  
 Lösche die  $winner$ -Nebenbedingung permanent aus dem eLP.  
 Gehe zu 1.

AUSGABE:  $CoverSet$  ist eine kleine Menge von Nebenbedingungen, die alle IIS abdeckt.

---

Ist dies nicht der Fall, wird getestet, ob das Löschen von  $N$  den  $SINF$ -Wert mehr reduziert als das temporäre Löschen der bisher aus  $CandidateSet$  ausgewählten Nebenbedingungen. Ziel dieses Vergleiches ist es, letztendlich die Nebenbedingung, deren Löschen den  $SINF$ -Wert um den größten Betrag reduziert, zu  $CoverSet$  hinzuzufügen. Reduziert das Entfernen von  $N$  den  $SINF$ -Wert tatsächlich mehr als das Löschen jeder vorherigen Nebenbedingung, werden die Nebenbedingungen, die auch nach dem Entfernen von  $N$  einen Einfluss auf den Wert der Zielfunktion haben, in der neu instantiierten Variable  $HoldSet$  gespeichert. Ist der  $SINF$ -Wert des LPs ohne  $N$  gleich 1, kann sofort ermittelt werden, welche Nebenbedingung noch zusätzlich als letzte zu  $CoverSet$  hinzugefügt werden muss.

Wurde die Nebenbedingung gefunden, deren Löschen den  $SINF$ -Wert um den größten Betrag reduziert, kann sie zur  $CoverSet$ -Menge hinzugefügt und dauerhaft aus dem LP gelöscht werden. Anderenfalls wird die im vorherigen Absatz beschriebene Vorgehensweise mit dem nun um eine Nebenbedingung reduzierten eLP und der neuen  $HoldSet$ -Menge wiederholt, um eine weitere Nebenbedingung zu  $CoverSet$  hinzuzufügen. Dies wird solange getan, bis so viele Nebenbedingungen aus dem LP gelöscht und zu  $CoverSet$  hinzugefügt wurden, dass das LP zulässig ist.  $CoverSet$  enthält dann wie gewünscht eine möglichst kleine Menge von Nebenbedingungen, die aus dem LP entfernt werden muss, um es zulässig werden zu lassen.

Wir werden die kleinste durch den hier vorgestellten Algorithmus gefundene Menge in unseren folgenden Betrachtungen als minimal bezeichnen, obwohl sie, da es sich bei dem Algorithmus um eine Heuristik handelt, keine exakte Lösung der MIN IIS COVER-Instanz sein muss – sie ist lediglich minimal in dem Sinn, dass sie die kleinste Menge ist, die mit diesem Algorithmus aufgefunden werden kann.

## 2.6 Das ALL PAIRS SHORTEST PATHS PROBLEM auf Graphen

Das ALL PAIRS SHORTEST PATHS PROBLEM hat zum Gegenstand, die kürzesten Wege zwischen allen Paaren von Knoten eines Graphen zu finden. Hierfür verwenden wir aufgrund seiner Einfachheit den Floyd-Warshall-Algorithmus. Es existieren jedoch Methoden, die abhängig von der Knoten- und Kantenanzahl des Graphen ein besseres Laufzeitverhalten aufweisen können als der hier vorgestellte Algorithmus, wie beispielsweise in [KV08] dargestellt wird.

**Floyd-Warshall-Algorithmus.** Der *Floyd-Warshall-Algorithmus* löst das ALL PAIRS SHORTEST PATH PROBLEM auf Graphen. Er besteht im Wesentlichen aus drei ineinander verschachtelten Schleifen, die jeweils die Menge der Knoten des Graphen durchlaufen. Die Laufzeit des Algorithmus ist demnach kubisch in Abhängigkeit von der Knotenanzahl. Algorithmus 3 zeigt eine Pseudocode-Notation des Floyd-Warshall-Algorithmus'.

Der Algorithmus funktioniert folgendermaßen:

Zunächst wird die Länge des kürzesten Weges für alle Paare von Knoten, zwischen denen eine Kante existiert, auf das Gewicht dieser Kante gesetzt. Knotenpaare, zwischen denen keine Kante existiert, erhalten den Wert  $\infty$ . Der Wert für den Weg von einem Knoten zu sich selbst wird auf 0 gesetzt.

Nun werden in der äußeren Schleife der Knoten  $v_j$  und in den beiden inneren Schleifen die Knoten  $v_i$  und  $v_k$  ausgewählt, wobei  $j \neq i \neq k$  gilt und die Knoten aufsteigend nach der Knotenindizierung von 0 bis  $n$  gewählt werden. Man betrachtet dann zum einen den direkten kürzesten bisher gefundenen Weg  $W_1$  von  $v_i$  nach  $v_k$  und zum anderen den kürzesten Weg

---

**Algorithmus 3** Der Floyd-Warshall-Algorithmus (vgl. [KV08])

---

INGABE: Ein Graph  $G = (V, E)$  mit  $V = \{v_0, \dots, v_n\}$  sowie eine Kantengewichtsfunktion  $w : E \rightarrow \mathbb{Q}$ .

Setze  $l_{i,j} = w((v_i, v_j))$  für alle  $(v_i, v_j) \in E$ .

Setze  $l_{i,j} = \infty$  für alle  $(v_i, v_j) \in (V \times V) \setminus E$  mit  $i \neq j$ .

Setze  $l_{i,i} = 0$  für alle  $v_i \in V$ .

Für  $j := 0$  bis  $n$ :

    Für  $i := 0$  bis  $n$ : Wenn  $i \neq j$ :

        Für  $k := 0$  bis  $n$ : Wenn  $k \neq j$ :

            Wenn  $l_{i,k} > l_{i,j} + l_{j,k}$ , dann setze  $l_{i,k} = l_{i,j} + l_{j,k}$ .

AUSGABE: Eine Matrix  $L$ , wobei  $l_{i,j}$  die Länge des kürzesten Weges von  $v_i$  nach  $v_j$  angibt.

---

$\mathcal{W}_2$  von  $v_i$  nach  $v_k$ , der den Knoten  $v_j$  enthält. Ist  $\mathcal{W}_1$  kürzer als  $\mathcal{W}_2$ , so ist dies bereits der kürzeste Weg, der von  $v_i$  nach  $v_k$  über Knoten führt, deren Index kleiner oder gleich  $j$  ist. Anderenfalls ist offensichtlich  $\mathcal{W}_2$  kürzer als  $\mathcal{W}_1$ . Somit muss nun der Wert für die Länge des kürzesten bisher gefundenen Weges, also von  $\mathcal{W}_1$ , auf den Wert für die Länge von  $\mathcal{W}_2$  gesetzt werden. Da der Algorithmus für alle möglichen Knotenpaare  $v_i, v_k$  und alle möglichen von ihnen verschiedenen Knoten  $v_j$  ausgeführt wird, entspricht die Länge von  $\mathcal{W}_1$  der Länge des kürzesten Weges zwischen  $v_i$  und  $v_k$ , sobald der Algorithmus terminiert.



### 3 Geometrisch basierter Ansatz zur Berechnung der Offset-Werte

Bereits in [Pru09] haben wir den geometrisch basierten Ansatz zur Berechnung der Offset-Werte vorgestellt, auf den wir in diesem Kapitel noch einmal eingehen wollen.

Eine Hyperebene  $E$  kann nicht nur wie in Abschnitt 2.4.1 durch die Vektoren, die sie aufspannen, beschrieben werden, sondern auch durch einen einzelnen Vektor  $a^T$ , der orthogonal zu ihr steht und dessen Ursprung auf der Hyperebene liegt (vgl. [BV04]):

$$E = \{y \mid a^T y = 0\}$$

Eine Hyperebene teilt einen Raum in zwei Halbräume, nämlich

$$H_{>} = \{y \mid a^T y > 0\}$$

und

$$H_{<} = \{y \mid a^T y < 0\} .$$

Betrachten wir nun, wie in Abschnitt 2.4.1 beschrieben, die Transitionen als Punkte im Raum  $\mathbb{Q}^{|P|}$ . Dann wird durch den Vektor  $a^T \in \mathbb{Q}^{|P|}$  bestimmt, welche Transition in welchem Halbraum liegt.

In Satz 2 nutzen wir diese Betrachtungen für eine im Vergleich zu Abschnitt 2.4 abgeänderte Berechnung der Offset-Werte.

**Satz 2.** Sei  $C = |P| \times |T|$  die Inzidenzmatrix eines Petrinetzes  $N = (P, T, F, W, s_0)$  und  $a \in \mathbb{Q}^{|P|}$  ein Vektor mit dem Ursprung  $\mathcal{Q}$ . Dann gilt: Die Komponente  $x_i$  des Vektors  $x = a^T C$  ergibt den Offset-Wert  $o(t_i)$  der Transition  $t_i$ .

Wir werden nun zeigen, warum die Verwendung der mit dieser Berechnungsmethode erhaltenen Offset-Werte zu einer konsistenten Progress Measure führt.

*Beweis.* Sei  $N = (P, T, F, W, s_0)$  ein Petrinetz und  $t_j \in T$  eine beliebig gewählte Transition.

*Fall 1:* Es besteht keine lineare Abhängigkeit zwischen  $t_j$  und anderen Transitionen des Petrinetzes.

Würden wir die Offset-Werte nach dem algebraischen Ansatz berechnen, wäre  $t_j$  in jeder möglichen Basis  $U$  enthalten, da es keine Möglichkeit gibt,  $\Delta t_j$  aus anderen Transitionsvektoren zu kombinieren. Die Offset-Werte für Transitionen aus  $U$  können nach [Sch04] beliebig gewählt werden, ohne dass die Konsistenz der Progress Measure beeinträchtigt wird.

*Fall 2:*  $t_j$  ist von anderen Transitionen linear abhängig.

Dann gilt  $\Delta t_j = \lambda_0 \Delta t_0 + \dots + \lambda_n \Delta t_n$  für eine linear unabhängige Menge von Transitionen  $U = \{t_0, \dots, t_n\}$  mit  $n < |T| - 1$  und  $t_j \notin U$ . Da der Vektor  $a^T$  mit jedem Transitionsvektor multipliziert wird, gilt offensichtlich auch  $a^T \cdot \Delta t_j = \lambda_0 a^T \cdot \Delta t_0 + \dots + \lambda_n a^T \cdot \Delta t_n$  und somit

### 3 Geometrisch basierter Ansatz zur Berechnung der Offset-Werte

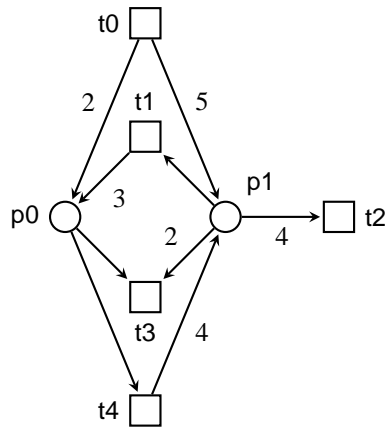


Abbildung 3.1: Beispiel-Petrinetz zur geometrisch basierten Berechnung der Offset-Werte [Pru09]

$o(t_j) = \lambda_0 o(t_0) + \dots + \lambda_n o(t_n)$ . Da die linearen Abhängigkeiten zwischen den Transitionen bei der Berechnung der Offset-Werte erhalten bleiben, wird auch in diesem Fall die Konsistenz der Progress Measure nicht beeinflusst.  $\square$

Es ist also möglich, einen beliebigen Vektor  $a^T$  zu wählen, um die Offset-Werte zu berechnen. Es kann beispielsweise auch  $a^T = (0, \dots, 0)$  gewählt werden; dann gilt  $x = (0, \dots, 0)$ , jeder Transition wird also der Offset-Wert 0 zugewiesen. Dies ist zwar nicht wünschenswert, da hierdurch jedem Zustand des Erreichbarkeitsgraphen der Progress-Wert 0 zugewiesen werden würde und die Anwendung der Sweep-Line-Methode somit keinen Effekt hätte; die berechnete Progress Measure ist dennoch offensichtlich konsistent.

Geometrisch lässt sich diese Berechnung der Offset-Werte folgendermaßen interpretieren: Der Vektor  $a^T$  hat seinen Ursprung im Punkt  $\underline{0}$ . Er beschreibt die Hyperebene  $E$ , zu der orthogonal steht. Auch  $E$  verläuft dementsprechend durch  $\underline{0}$ . Somit werden durch  $a^T$  auch die Halbräume  $H_>$  und  $H_<$  definiert. Offensichtlich existiert folgender Zusammenhang zwischen den mit Hilfe von  $a^T$  berechneten Offset-Werten und der Lage der entsprechenden Transitionen:

- Wenn  $t_i$  in  $E$  liegt, dann folgt  $o(t_i) = 0$ .
- Wenn  $t_i$  in  $H_>$  liegt, dann folgt  $o(t_i) > 0$ .
- Wenn  $t_i$  in  $H_<$  liegt, dann folgt  $o(t_i) < 0$ .

**Beispiel 2.** Mit Hilfe des in Abbildung 3.1 dargestellten Petrinetzes kann die geometrische Interpretation veranschaulicht werden. Die Inzidenzmatrix dieses Netzes ist

$$C = \begin{pmatrix} 2 & 3 & 0 & -1 & -1 \\ 5 & -1 & -4 & -2 & 4 \end{pmatrix}.$$

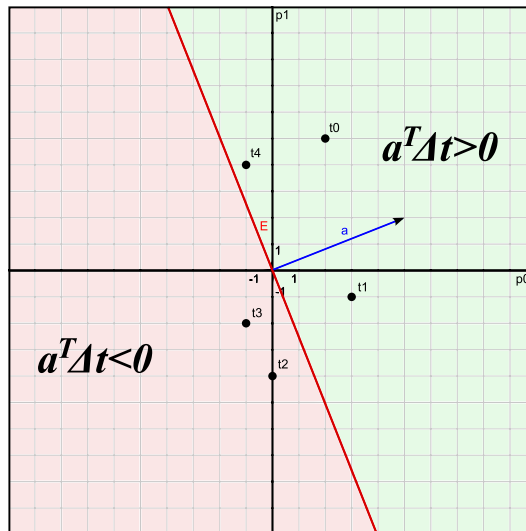


Abbildung 3.2: Darstellung der Transitionen im Raum  $\mathbb{Q}^{|P|}$  und der durch  $a$  definierten Halbräume (vgl. [Pru09])

In Abbildung 3.2 sind die Transitionen  $t_0, \dots, t_4$  im Raum  $\mathbb{Q}^{|P|}$  dargestellt; dieser ist zweidimensional, da das Petrinetz über zwei Plätze verfügt. Der orthogonal zur Hyperebene  $E$  stehende Vektor  $a^T = (5, 2)$  teilt den Raum so in zwei Halbräume, dass  $t_0, t_1$  und  $t_4$  im Halbraum  $H_>$  liegen, während sich  $t_2$  und  $t_3$  im Halbraum  $H_<$  befinden. Die Offset-Werte der Transitionen ergeben sich durch  $a^T C = (o(t_0), o(t_1), o(t_2), o(t_3), o(t_4)) = (20, 13, -8, -9, 3)$ .

\*

Vergleichen wir nun den geometrisch basierten Ansatz zur Berechnung der Offset-Werte mit dem algebraisch basierten Ansatz und dessen geometrischer Interpretation aus Abschnitt 2.4.1. Angenommen, man hat zu einem gegebenen Netz Offset-Werte nach der algebraischen Methode berechnet. Dann wurde hierbei eine bestimmte Basis  $U$  gewählt. Die Hyperebene  $E$ , die durch die zu den Transitionen aus  $U$  zugehörigen Vektoren aufgespannt wird, definiert den Offset-Wert 1 für alle Punkte, die sie enthält. Würde man die Offset-Werte nach der in diesem Abschnitt beschriebenen Methode mit einem Vektor  $a^T$  berechnen, der genau die Richtung der Geraden  $g$  (siehe Abschnitt 2.4.1) und eine bestimmte Länge besitzt, würde man für jede Transition den gleichen Offset-Wert erhalten wie nach der algebraischen Berechnungsmethode. Ändert man die Länge des Vektors  $a^T$ , nicht jedoch seine Richtung, ändert sich zwar die Größe der Offset-Werte, ihr Verhältnis zueinander bleibt jedoch gleich.

Der Hauptunterschied der in diesem Abschnitt beschriebenen Methode gegenüber der algebraischen Methode ist also, dass die Hyperebene  $E$  nicht mehr durch eine Menge  $U$  von linear unabhängigen Transitionsvektoren beschrieben wird, sondern nur noch durch einen beliebig zu wählenden Vektor  $a^T$  der Dimension  $|P|$ . Der Offset-Wert 1 wird jetzt nicht mehr durch die Menge  $U$  und die Festlegung, dass alle in ihr enthaltenen Transitionen diesen Offset-Wert zu erhalten haben, definiert, sondern durch Richtung und Länge des Vektors  $a^T$ . Es existiert nun also statt zwei Freiheitsgraden, nämlich der Wahl der Basis  $U$  und der Festlegung der Offset-Werte für die in ihr enthaltenen Transitionen, nur noch der Vektor  $a^T$ , der gewählt werden

### *3 Geometrisch basierter Ansatz zur Berechnung der Offset-Werte*

muss.

Im folgenden Kapitel werden wir beschreiben, wie die geometrische Methode zur Berechnung der Offset-Werte für die Minimierung der Anzahl negativer Offset-Werte genutzt werden kann.

## 4 Optimierung der Berechnung der Offset-Werte

In diesem Kapitel werden wir eine Heuristik entwickeln, die zum Ziel hat, die Berechnung der Offset-Werte insbesondere im Hinblick auf den Speicherplatzbedarf der Zustandsraumdurchmusterung mit der Sweep-Line-Methode zu optimieren. Wir orientieren uns hierbei an den beiden in Abschnitt 2.4.2 genannten Optimierungsansätzen: der Minimierung der Anzahl negativer Offset-Werte sowie der Vermeidung von Ketten von Regresstransitionen. Die Heuristik ist so aufgebaut, dass zunächst mehrere kleinste Mengen von Regresstransitionen berechnet werden und anschließend von diesen die Menge ausgewählt wird, die die bestmögliche Vermeidung von Ketten von Regresstransitionen verspricht. Wir betrachten daher zunächst die Minimierung der Anzahl negativer Offset-Werte und wenden uns anschließend der Vermeidung von Ketten von Regresstransitionen zu.

### 4.1 Minimierung der Anzahl negativer Offset-Werte

In [Pru09] haben wir uns bereits mit der Minimierung der Anzahl an negativen Offset-Werten beschäftigt. Dies wollen wir an dieser Stelle weiter vertiefen.

Grundsätzlich sehen wir zwei Möglichkeiten, um die Anzahl an Regresstransitionen bei der Berechnung der Offset-Werte zu minimieren:

1. Wir könnten zunächst alle minimalen T-Invarianten erstellen; diese seien indiziert. Gegeben sei ein Petrinetz  $N = (P, T, F, W, s_0)$ . Wir bezeichnen eine T-Invariante mit  $x_j$ . Dann erstellen wir für jede Transition  $t_i \in T$  eine Menge  $S_i$ , und zwar so, dass  $j \in S_i$  gilt, wenn  $i \in \text{supp}(x_j)$ . Anschließend würden wir ein Mengenüberdeckungsproblem über der Menge  $M = \{0, \dots, |T| - 1\}$  und deren Teilmengen  $S_i$  mit  $0 \leq i \leq |T| - 1$  lösen. Alle Transitionen, deren Indizes in der Überdeckung  $L$  enthalten sind, wären Regresstransitionen. Hierbei muss jedoch beachtet werden, dass die Mengenüberdeckung nicht so gewählt sein darf, dass eine Invariante  $x_j$  existiert mit  $L \supseteq \text{supp}(x_j)$ .

Diese Berechnungsmethode hat folgenden Hintergrund:

In jedem Kreis im Erreichbarkeitsgraphen muss genau einer der beiden folgenden Fälle eintreten:

- Alle Transitionen im Kreis haben den Offset-Wert 0.
- Mindestens eine Transition im Kreis hat einen negativen Offset-Wert und mindestens eine Transition im Kreis hat einen positiven Offset-Wert.

Eine T-Invariante gibt an, welche Transition wie oft schalten muss, damit ein Kreis im Erreichbarkeitsgraph entsteht: Sei  $x_j = (x_{j_0}, \dots, x_{j_k})$  eine T-Invariante. Dann muss für alle  $i$  mit  $0 \leq i \leq k$  die Transition  $t_i$  genau  $x_{j_i}$  mal schalten, damit ein Kreis im Erreichbarkeitsgraph

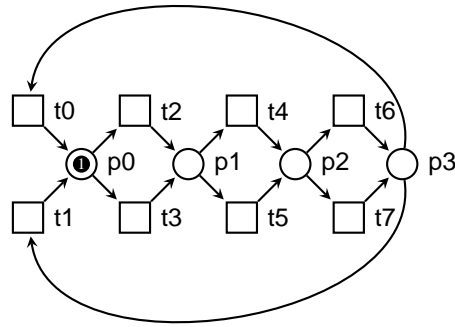


Abbildung 4.1: Petrinetz mit 8 Transitionen und 16 T-Invarianten (vgl. [Sta90])

entstehen kann. Durch die Lösung des Mengenüberdeckungsproblems wird nun eine kleinste Menge von Transitionen gesucht, so dass im Träger jeder T-Invariante mindestens eine dieser Transitionen (präziser: deren Index) enthalten ist. Diese Menge bildet dann eine minimale Menge von Regresstransitionen, wenn für keine der T-Invarianten alle Elemente ihres Trägers in der Mengenüberdeckung enthalten sind. Wenn doch eine solche T-Invariante  $x_j$  existiert, können nicht alle Transitionen, deren Index in  $supp(x_j)$  enthalten ist, Regresstransitionen sein, da sonst alle Transitionen der durch  $x_j$  bestimmten Kreise im Erreichbarkeitsgraphen einen negativen Offset-Wert erhalten würden und die Progress Measure somit inkonsistent werden würde.

Möchte man diese Methode praktisch nutzen, treten jedoch einige Schwierigkeiten auf: So ist das Problem, alle minimalen T-Invarianten zu berechnen, exponentiell. Zu einem Netz mit  $n$  Plätzen und  $2n$  Transitionen kann es  $2^n$  minimale T-Invarianten geben (vgl. [Sta90]). Ein solches Netz ist für  $n = 4$  in Abb. 4.1 dargestellt. Hinzu kommt, dass das Mengenüberdeckungsproblem NP-schwer ist (vgl. z.B. [Kar72]). Somit kann man davon ausgehen, dass es für große Probleminstanzen sehr zeitaufwändig ist, eine Lösung zu finden, die unseren Anforderungen entspricht. Es scheint also wenig ratsam zu sein, diesen Ansatz praktisch umzusetzen. Deshalb werden wir nun einen zweiten Ansatz betrachten, der uns praktikabler erscheint und den wir schließlich in der Implementation umgesetzt haben.

**2.** Eine andere Möglichkeit ist, die Minimierung der Anzahl negativer Offset-Werte wie in [Pru09] beschrieben als MaxFS-Instanz aufzufassen.

Nutzen wir den in Kapitel 3 beschriebenen geometrischen Ansatz zur Berechnung der Offset-Werte, kann das Problem der Minimierung der Anzahl an Regresstransitionen folgendermaßen aufgefasst werden: Gesucht wird ein Vektor  $a^T$ , der den Raum  $Q^{|P|}$  so in die beiden Halbräume  $H_<$  und  $H_>$  aufteilt, dass möglichst viele Transitionen in  $H_>$  liegen. Liegt eine Transition  $t_i$  im Halbraum  $H_>$ , gilt für sie die Ungleichung  $a^T \cdot \Delta t_i > 0$ . Diese Ungleichung soll nun für möglichst viele Transitionen erfüllt sein. Stellen wir die Ungleichung für jede Transition auf, erhalten wir ein Ungleichungssystem der Form

#### 4.1 Minimierung der Anzahl negativer Offset-Werte

$$\begin{aligned} a_0 \cdot \Delta t_{0_0} + \dots + a_n \cdot \Delta t_{0_n} &> 0 \\ \dots + \dots + \dots &> 0 \\ a_0 \cdot \Delta t_{k_0} + \dots + a_n \cdot \Delta t_{k_n} &> 0 \end{aligned}$$

mit  $n = |P| - 1$  und  $k = |T| - 1$ . Wir erhalten zu einem gegebenen Petrinetz  $N = (P, T, F, W, s_0)$  mit der Inzidenzmatrix  $C$  somit ein Ungleichungssystem der Form  $C^T a > \underline{0}$ . Sucht man aus einem Ungleichungssystem der allgemeinen Form  $AxRb$  mit  $A \in \mathbb{R}^{m \times n}$ ,  $b \in \mathbb{R}^m$ ,  $x \in \mathbb{R}^n$  und  $R \in \{=, \geq, >, \neq\}$  die größte zulässige Teilmenge von Ungleichungen, bezeichnet man dieses Problem als  $\text{MAXFLS}^R$  [AK95]. Da unser Ungleichungssystem dieser Form mit  $A = C^T$ ,  $x = a$ ,  $R = '>'$  und  $b = \underline{0}$  entspricht, nennen wir die eben beschriebenen Ungleichungen  $\text{MAXFLS}^>$ -Nebenbedingungen. Es ist leicht zu sehen, dass jede  $\text{MAXFS}$ -Instanz wegen  $Ax \leq b \Leftrightarrow -Ax \geq -b$  als  $\text{MAXFLS}^{\geq}$ -Instanz formuliert werden kann.

Unser Problem ist  $\text{MAXFS}$  also sehr ähnlich. Wenn wir die Ungleichungen in die Form  $a^T \cdot \Delta t_i \leq 0$  bringen können, kann das System als  $\text{MAXFS}$ -Instanz aufgefasst werden. Würden wir nun in den Ungleichungen die „>“-Relation durch „ $\geq$ “ ersetzen, um Ungleichungen der Form  $a^T \cdot \Delta t_i \geq 0$  zu erhalten, würde folgendes Problem auftreten:  $a^T = (0, \dots, 0)$  wäre nun eine zulässige Lösung für jedes Ungleichungssystem dieser Form. Somit wäre immer das gesamte Ungleichungssystem als sein größtes zulässiges Teilsystem die Lösung der  $\text{MAXFS}$ -Instanz. Aus  $a^T = (0, \dots, 0)$  folgt unmittelbar, dass jeder Transition der Offset-Wert 0 zugewiesen werden würde. Wie bereits oben erwähnt, ist es nicht sinnvoll, jeder Transition den Offset-Wert 0 zuzuweisen. Um dieses Problem zu umgehen, ersetzen wir die 0 auf der rechten Seite jeder Ungleichung durch einen möglichst kleinen positiven Wert  $\varepsilon \in \mathbb{Q}$ . Somit wird  $a^T = (0, \dots, 0)$  als zulässiger Lösungsvektor durch eine Verschärfung der Nebenbedingungen ausgeschlossen. Jetzt erhält man dementsprechend folgende Nebenbedingungen eines LP, auf die man zur Lösung unseres Problems eine  $\text{MAXFS}$ -Heuristik anwenden kann:

$$\begin{aligned} -a_0 \cdot \Delta t_{0_0} - \dots - a_n \cdot \Delta t_{0_n} &\leq \varepsilon \\ \dots - \dots - \dots &\leq \varepsilon \\ -a_0 \cdot \Delta t_{k_0} - \dots - a_n \cdot \Delta t_{k_n} &\leq \varepsilon \end{aligned}$$

Wir werden diese Nebenbedingungen im Folgenden als  $\text{MAXFS}$ -Nebenbedingungen eines Petrinetzes bezeichnen.

Jede  $\text{MAXFLS}^>$ -Nebenbedingung wurde in eine  $\text{MAXFS}$ -Nebenbedingung umgewandelt. Da die  $\text{MAXFS}$ -Nebenbedingung eine Verschärfung der jeweiligen  $\text{MAXFLS}^>$ -Nebenbedingung darstellt, ist die zulässige Menge der  $\text{MAXFS}$ -Nebenbedingungen eine Teilmenge der durch die  $\text{MAXFLS}^>$ -Nebenbedingungen beschriebenen zulässigen Menge. Wählt man also einen beliebigen zulässigen Vektor  $a^T$  aus der zulässigen Menge der  $\text{MAXFS}$ -Nebenbedingungen, befindet sich dieser auch in der zulässigen Menge der  $\text{MAXFLS}^>$ -Nebenbedingungen.

Auf die  $\text{MAXFS}$ -Nebenbedingungen kann nun die LP-basierte Heuristik aus Abschnitt 2.5.2 angewandt werden. Als Lösung erhält man eine Teilmenge  $L$  dieser Nebenbedingungen. Da es sich bei der Heuristik aus Abschnitt 2.5.2 um eine  $\text{MIN IIS COVER}$ -Heuristik handelt, bildet  $L$  die Lösung der zugehörigen  $\text{MIN IIS COVER}$ -Instanz. Die Lösung  $L'$  der  $\text{MAXFS}$ -Instanz besteht dementsprechend aus allen  $\text{MAXFS}$ -Nebenbedingungen, die nicht in  $L$  enthalten sind.

#### 4 Optimierung der Berechnung der Offset-Werte

Jede Transition, deren zugehörige Ungleichung in  $L'$  enthalten ist, kann einen positiven Offset-Wert erhalten. Alle Transitionen, deren Ungleichungen nicht in dieser Menge enthalten sind (dies sind genau die Ungleichungen aus  $L$ ), müssen einen negativen Offset-Wert erhalten, weil die ihnen zugeordneten Ungleichungen nicht gleichzeitig mit den in  $L'$  enthaltenen erfüllbar sind. Aus der durch die in  $L'$  enthaltenen Ungleichungen beschriebenen Lösungsmenge kann nun der Vektor  $a^T$  beliebig gewählt werden, um mit dem geometrisch basierten Ansatz gültige Offset-Werte, von denen eine minimale Anzahl negativ ist, zu berechnen.

Um die geometrische Interpretation des MAXFS-Problems zu veranschaulichen, betrachten wir an dieser Stelle noch einmal das Netz aus Abbildung 3.1 (S. 28). Aus der Inzidenzmatrix

$$C = \begin{pmatrix} 2 & 3 & 0 & -1 & -1 \\ 5 & -1 & -4 & -2 & 4 \end{pmatrix}$$

des Petrinetzes ergeben sich die MAXFS-Nebenbedingungen

$$\begin{aligned} -2a_1 - 5a_2 &\leq \varepsilon \\ -3a_1 + a_2 &\leq \varepsilon \\ 4a_2 &\leq \varepsilon \\ a_1 + 2a_2 &\leq \varepsilon \\ a_1 - 4a_2 &\leq \varepsilon. \end{aligned}$$

Jede MAXFS-Nebenbedingung beschreibt selbst wiederum einen Halbraum. Dieser besteht aus allen Punkten im Raum  $\mathbb{Q}^{|P|}$ , die die Nebenbedingung erfüllen. Betrachtet man den Schnitt aller durch die MAXFS-Nebenbedingungen beschriebenen Halbräume, erhält man ein *Polyeder*. Dieses ist unbeschränkt und konvex. Wenn zwei Punkte  $x, y \in \mathbb{Q}^{|P|}$  innerhalb des Polyeders liegen, liegen also auch alle Punkte aus der Menge  $\{\lambda x + (1 - \lambda)y \mid \lambda \in \mathbb{Q}, 0 \leq \lambda \leq 1\}$  innerhalb des Polyeders. Das Polyeder beschreibt die zulässige Lösungsmenge des LPs. Ist das Polyeder nichtleer, ist die Lösungsmenge zulässig; sonst ist das LP unzulässig. In Abbildung 4.2 sind die Halbräume dargestellt, die durch die den Transitionen zugeordneten MAXFS-Nebenbedingungen beschrieben werden. Für blau dargestellte Geraden liegt der beschriebene Halbraum oberhalb dieser, für rot dargestellte darunter. Dies ist zusätzlich durch Pfeile gekennzeichnet. Es ist erkennbar, dass der Schnitt dieser Halbräume leer ist, da der Punkt  $\underline{0}$  als Lösung ausgeschlossen ist. Somit ist das LP nicht zulässig. Durch das Lösen der zugehörigen MAXFS-Instanz sucht man nun nach einem nichtleeren Polyeder, das durch möglichst viele Ungleichungen beschrieben wird. In Abbildung 4.3 ist ein solches Polyeder für das Beispiel dargestellt. Die den Transitionen  $t_0, t_1$  und  $t_4$  zugeordneten Ungleichungen bilden eine zulässige Lösung der durch alle Nebenbedingungen beschriebenen MAXFS-Instanz.

Die hier beschriebene Berechnungsmethode hat gegenüber der algebraischen Methode offensichtliche Vorteile bei Netzen, für die eine monotone Progress Measure existiert. Wir wollen dies anhand eines Beispiels verdeutlichen.

#### 4.1 Minimierung der Anzahl negativer Offset-Werte

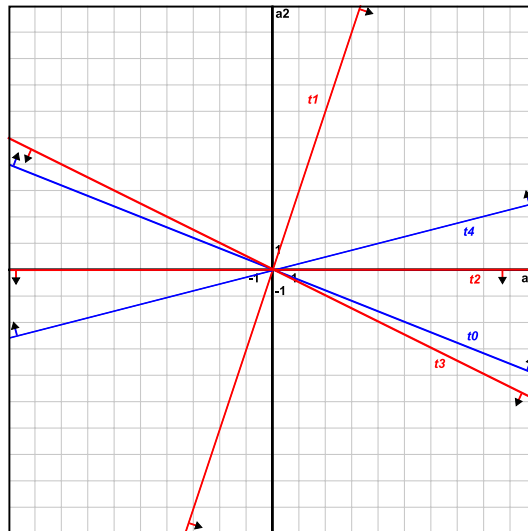


Abbildung 4.2: Darstellung der den Transitionen zugeordneten Halbräume [Pru09]

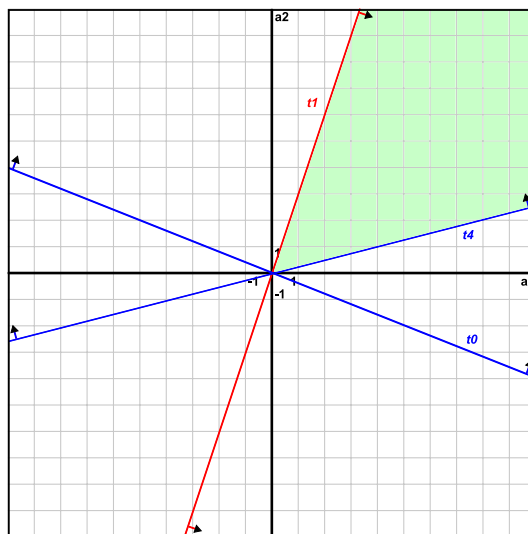


Abbildung 4.3: Darstellung des aus den Transitionen  $t_0$ ,  $t_1$  und  $t_4$  ableitbaren Polyeders [Pru09]

#### 4 Optimierung der Berechnung der Offset-Werte

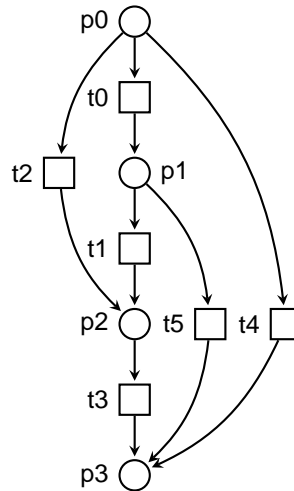


Abbildung 4.4: Ein Petrinetz, bei dem Regresstransitionen verhindert werden können [Pru09]

**Beispiel 3.** Gegeben sei das Petrinetz aus Abbildung 4.4, das wir bereits in [Pru09] betrachteten. Die Inzidenzmatrix des Netzes ist

$$C = \begin{pmatrix} -1 & 0 & -1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Im Erreichbarkeitsgraphen dieses Netzes existieren unabhängig von der Anfangsmarkierung keine Kreise. Berechnet man die Offset-Werte nach der algebraischen Methode, kann beispielsweise die Basis  $U = \{t_0, t_3, t_4\}$  gewählt werden. Die Transitionen aus  $U$  erhalten den Offset-Wert  $o(t_0) = o(t_3) = o(t_4) = 1$ , die Offset-Werte der restlichen Transitionen ergeben sich durch die Gleichungen  $o(t_1) = o(t_4) - o(t_3) - o(t_0) = -1$ ,  $o(t_2) = o(t_4) + o(t_3) = 2$  und  $o(t_5) = o(t_4) - o(t_0) = 0$ . Somit ist  $t_1$  eine Regresstransition.

Berechnet man die Offset-Werte jedoch nach dem geometrisch basierten Ansatz, erhält man die MaxFS-Nebenbedingungen

$$\begin{aligned} a_1 - a_2 &\leq \varepsilon \\ a_2 - a_3 &\leq \varepsilon \\ a_1 - a_3 &\leq \varepsilon \\ a_3 - a_4 &\leq \varepsilon \\ a_1 - a_4 &\leq \varepsilon \\ a_2 - a_4 &\leq \varepsilon. \end{aligned}$$

Diese beschreiben zulässige Nebenbedingungen eines LP, in dessen Lösungsmenge beispielsweise der Vektor  $a^T = (1, 2, 3, 4)$  liegt. Somit können wir die Offset-Werte  $x = a^T C = (o(t_0), \dots, o(t_5)) = (1, 1, 2, 1, 3, 2)$  berechnen. Mit dieser Berechnung erhält man keinen negativen Offset-Wert, weshalb eine monotone Progress Measure berechnet werden kann.

\*

### 4.1.1 Mögliche Beschleunigung der Berechnung

An dieser Stelle soll eine Idee zur Beschleunigung der vorgestellten Berechnungsmethode skizziert werden. Ausgangspunkt ist die Beobachtung, dass man die Menge aller Transitionen für einige Petrinetze offensichtlich so partitionieren kann, dass alle Transitionen innerhalb einer Partitionsmenge unabhängig von allen anderen Transitionen des Netzes sind. Dann ist es möglich, die Menge von Regresstransitionen für jede Partitionsmenge einzeln zu berechnen. Wir können eine solche Partition  $\mathcal{P} = \{M_0, \dots, M_k\}$  folgendermaßen erstellen:

Gegeben sei eine Inzidenzmatrix  $C$ . Wir berechnen zunächst die reduzierte Zeilenstufenform  $R$  von  $C$ . Nun werden die einzelnen Zeilen  $z$  von  $R$  betrachtet. Wenn für zwei Transitionen  $t_i$  und  $t_j$   $R(z, i) \neq 0$  und  $R(z, j) \neq 0$  gilt, dann existiert eine gültige Gleichung  $\Delta t_i = \lambda_0 \Delta t_0 + \dots + \lambda_j \Delta t_j + \dots + \lambda_n \Delta t_n$  mit  $\lambda_j \neq 0$  und  $n = |T| - 2$  (da  $t_i$  auf der rechten Seite der Gleichung nicht vorkommt).  $t_i$  und  $t_j$  sind also bei einer bestimmten Wahl der Basis  $U$  voneinander linear abhängig. Somit müssen  $t_i$  und  $t_j$  in der gleichen Menge  $M_l$  mit  $0 \leq l \leq k$  liegen.

---

#### Algorithmus 4 Algorithmus zur Beschleunigung der Berechnung einer MaxFS-Lösung

---

EINGABE: Inzidenzmatrix  $C$  eines Petrinetzes  $N = (P, T, F, W, s_0)$ .

Berechne die reduzierte Zeilenstufenform  $R$  von  $C$ .

Setze  $\mathcal{P} = \emptyset$ ,  $\mathcal{Z} = \emptyset$ ,  $f : \mathcal{Z} \rightarrow \mathcal{P}$ .

Für alle  $t_i \in T$ :

Setze  $Y = \{j \mid r_{j,i} \neq 0\}$ .

Für alle  $Z_l \in \mathcal{Z}$ :

Falls ein  $y \in Y$  existiert mit  $y \in Z_l$ :

Setze  $Z_l = Z_l \cup Y$  und  $f(Z_l) = f(Z_l) \cup \{t_i\}$ .

Falls  $\mathcal{Z} = \emptyset$  oder kein  $y \in Y$  existiert mit  $y \in Z_l$  für beliebige  $Z_l \in \mathcal{Z}$ :

Setze  $M = \{t_i\}$ .

Setze  $\mathcal{Z} = \mathcal{Z} \cup \{Y\}$ ,  $\mathcal{P} = \mathcal{P} \cup \{M\}$  und  $f(Y) = M$ .

Wiederhole:

Falls  $M_i, M_j \in \mathcal{P}$  existieren mit  $M_i \cap M_j \neq \emptyset$ :

Setze  $M_i = M_i \cup M_j$  und  $\mathcal{P} = \mathcal{P} \setminus M_j$ .

sonst:

terminiere.

AUSGABE: Partition  $\mathcal{P}$ , die  $T$  in voneinander unabhängige Mengen aufteilt.

---

Der von uns entworfene Algorithmus 4 erzeugt eine solche Partition. Sie wird in  $\mathcal{P}$  gespeichert, wobei  $\mathcal{P}$  anfänglich leer ist; die einzelnen Mengen  $M_l$  werden im Verlauf des Algorithmus' eingefügt. Jede Menge  $M_l$  aus  $\mathcal{P}$  soll über die bijektive Funktion  $f$  genau einer Menge  $Z_l$  des Mengensystems  $\mathcal{Z}$  zugeordnet werden. In  $Z_l$  sollen die Indizes aller Zeilen aus  $R$  gespeichert werden, an deren Stelle in mindestens einer der den Transitionen aus  $M_l$  zugeordneten Spalten ein Wert ungleich 0 vorkommt. Nun werden alle Transitionen  $t_i \in T$  nacheinander betrachtet: In der Menge  $Y$  wird zunächst temporär der Index  $j$  jeder Zeile aus  $R$  gespeichert, für die  $r_{j,i} \neq 0$  gilt. Falls bereits eine Transition  $t_j \in T$  betrachtet wurde, die in  $R$  in derselben Zeile

#### 4 Optimierung der Berechnung der Offset-Werte

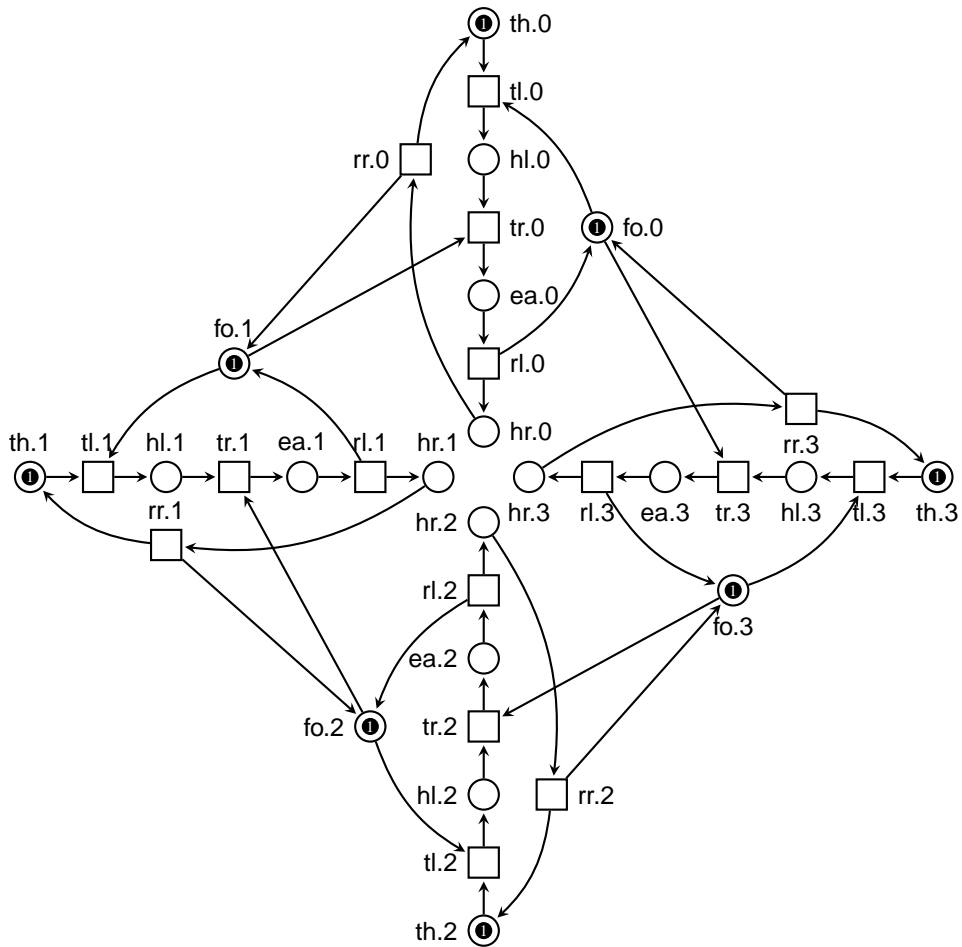


Abbildung 4.5: Petrinetz für 4 „speisende Philosophen“ (vgl. z.B. [Sch04])

wie  $t_i$  einen Eintrag mit einem Wert ungleich 0 hat, muss  $t_i$  in derselben Menge  $M_l$  wie  $t_j$  liegen und wird daher  $M_l$  hinzugefügt. Ebenso werden die Indizes aller Zeilen, an deren Stelle in der  $t_i$  zugeordneten Spalte ein Eintrag ungleich 0 steht (also genau die Indizes aus  $Y$ ), zu der Menge  $Z_l$  hinzugefügt. Wurde hingegen noch keine Transition  $t_j$  mit der genannten Eigenschaft betrachtet, werden die neuen Mengen  $M = \{t_i\}$  zu  $\mathcal{P}$  sowie  $Y$  zu  $\mathcal{Z}$  hinzugefügt.

Da in  $\mathcal{P}$  nach dem Ausführen dieser Schritte mehrere Mengen vorkommen können, die dieselbe Transition enthalten, müssen solche Mengen anschließend vereinigt werden, damit  $\mathcal{P}$  eine Partition bildet.

Betrachten wir diesen Ansatz nun am Beispiel der „speisenden Philosophen“ aus Abb. 4.5. Führt man Algorithmus 4 aus, erhält man die Partition

$\mathcal{P} = \{\{tl.0, tr.0, rl.0, rr.0\}, \{tl.1, tr.1, rl.1, rr.1\}, \{tl.2, tr.2, rl.2, rr.2\}, \{tl.3, tr.3, rl.3, rr.3\}\}$ . Die Regresstransitionen können also für jeden „Philosophen“ einzeln berechnet werden. Wird dafür die in Abschnitt 2.5.2 beschriebene LP-basierte Heuristik verwendet, müssen insgesamt nur 4 eLPs gelöst werden, da jede Menge der Partition nur eine Regresstransition enthält, welche nach dem Lösen von einem eLP identifiziert werden kann. Im Übrigen kann innerhalb der einzelnen Mengen jede beliebige Transition als Regresstransition gewählt werden, was je-

doch an dieser Stelle nicht weiter von Bedeutung ist. Wendet man die Heuristik jedoch auf das gesamte Netz an, ohne vorher die Partition zu bilden, müssen  $\sum_{k=2}^4 4k = 36$  eLPs gelöst werden, da mit jeder Regresstransition, die gefunden wird, drei weitere Transitionen als potentielle Regresstransitionen ausscheiden, nämlich alle restlichen, die zum gleichen „Philosophen“ wie die Regresstransition gehören. Verallgemeinert man diese Rechnung für ein System mit  $n$  Philosophen, kann man feststellen, dass ohne die hier vorgeschlagene Beschleunigungsmethode insgesamt  $\sum_{k=2}^n 4k$  eLPs, mit der Beschleunigungsmethode jedoch nur  $n$  eLPs gelöst werden müssen. Obwohl für die Partitionierung noch zusätzlich Algorithmus 4 ausgeführt werden muss, sollte die hier vorgestellte Vorgehensweise für hinreichend große Systeme von „speisenden Philosophen“ gegenüber der Berechnung von Regresstransitionen ohne Partitionierung deutliche Laufzeitvorteile erbringen.

Offensichtlich hängt aber der Erfolg der Beschleunigungsmethode stark von der Struktur des jeweiligen Petrinetzes ab. So ergibt sich bei Netzen, für die die Partition  $\mathcal{P}$  nur eine Menge enthält, eine Laufzeitverschlechterung, da gegenüber der Berechnung ohne Partitionierung zusätzlich Algorithmus 4 ausgeführt wird und sonst keine Veränderung auftritt.

Die hier vorgestellte Idee ist nicht in unsere Implementation mit eingeflossen. Mit Hilfe einer von uns vorgenommenen prototypischen Implementierung stellten wir fest, dass für viele Netze der Fallstudie aus Abschnitt 4.4 nur eine Menge pro Partition berechnet wird und insgesamt keine signifikante Beschleunigung zu erwarten ist.

#### 4.1.2 Heuristik zur Berechnung mehrerer MIN IIS COVER-Lösungen

Wir haben bisher beschrieben, wie eine einzelne Menge von Regresstransitionen minimaler Kardinalität berechnet werden kann. Da aber zwischen mehreren solcher Mengen eine bezüglich weiterer Heuristiken optimale ausgewählt werden soll, benötigen wir eine Methode, mit der mehrere Mengen von Regresstransitionen berechnet werden können. Deshalb haben wir das hier beschriebene Berechnungsverfahren entwickelt, um auf Basis der in Abschnitt 2.5.2 beschriebenen LP-basierten MIN IIS COVER-Heuristik mehrere Mengen von Regresstransitionen berechnen zu können.

Zunächst wird, wie in Abschnitt 2.5.2 beschrieben, eine einzelne minimale Menge  $M$  von Regresstransitionen berechnet. Anschließend wird folgendermaßen verfahren:

Sei  $\mathcal{L}$  ein LP, dessen Nebenbedingungen den MAXFS-Nebenbedingungen des betrachteten Petrinetzes  $N = (P, T, F, W, s_0)$  entsprechen und  $\mathcal{E}\mathcal{L}$  das aus  $\mathcal{L}$  entstehende eLP. Es gelte  $M = \{t_0, \dots, t_m\}$ . Für jede Transition  $t_i \in M$  wird nun das LP  $\mathcal{L}'_i$  erstellt, das entsteht, wenn die Ungleichungen, die allen Transitionen aus  $M \setminus \{t_i\}$  zugeordnet sind, aus  $\mathcal{L}$  entfernt werden. Es genügt folglich, eine Nebenbedingung aus dem LP  $\mathcal{L}'_i$  zu entfernen, um es zulässig werden zu lassen, nämlich die zu  $t_i$  zugehörige MAXFS-Nebenbedingung. Nun wird das zu  $\mathcal{L}'_i$  zugehörige eLP  $\mathcal{E}\mathcal{L}'_i$  gelöst. Wie schon bei der ursprünglichen LP-basierten Heuristik werden wieder die reduzierten Kosten der Nebenbedingungen betrachtet. Sind die reduzierten Kosten einer Nebenbedingung ungleich 0, hat das Löschen dieser Nebenbedingung Einfluss auf den Wert der Zielfunktion des eLPs (also den *SINF*-Wert). Unsere Annahme ist nun folgende: Da es genügt, die zu  $t_i$  zugehörige Nebenbedingung aus dem  $\mathcal{L}'_i$  zu entfernen, um es zulässig werden zu lassen, könnte auch das Entfernen einer anderen einzelnen Nebenbedingung, deren reduzierte Kosten ungleich 0 sind,  $\mathcal{L}'_i$  zulässig werden lassen. Alle Transitionen, deren zugehörige Ungleichungen diese Eigenschaft aufweisen, werden deshalb in der Menge  $R_i$  gespeichert. Es gilt immer  $t_i \in R_i$ , weil ja  $t_i$  in Kombination mit allen Transitionen aus  $M \setminus \{t_i\}$  eine minimale

#### 4 Optimierung der Berechnung der Offset-Werte

Menge von Regresstransitionen bildet und somit das Entfernen der zu  $t_i$  zugehörigen Ungleichung den *SINF*-Wert von  $\mathcal{E}\mathcal{L}'_i$  reduziert.

Im nächsten Schritt werden die Transitionen aus den einzelnen Mengen  $R_i$  miteinander kombiniert: Wir bilden alle möglichen Mengen  $M'_j$  von Transitionen, so dass aus jeder Menge  $R_i$  genau eine Transition in  $M'_j$  enthalten ist (da  $M'_j$  eine Menge im mathematischen Sinn ist, ist des Weiteren ausgeschlossen, dass eine Transition in  $M'_j$  mehrfach enthalten ist). Wir erhoffen uns, dass von diesen Mengen möglichst viele gültige Regresstransitionsmengen bilden. Weil jede Transition  $t_i \in M$  in der jeweiligen Menge  $R_i$  enthalten ist, gilt  $M'_j = M$  für genau ein  $j$ . Insgesamt erhalten wir maximal  $|R_0| \cdot \dots \cdot |R_m|$  Mengen von Transitionen mit der Kardinalität  $|M|$ . Da die Mengen  $R_0, \dots, R_m$  nicht für jedes Netz paarweise disjunkt sind, verringert sich die Anzahl der Mengen  $M'_j$  in einem solchen Fall dementsprechend. Abhängig vom Netz kann so eine große Anzahl an unterschiedlichen Mengen entstehen. Betrachtet man das System der 4 „speisenden Philosophen“ aus Abb. 4.5, erhält man mit der Methode insgesamt  $4^4 = 256$  verschiedene Mengen von Regresstransitionen. Es ist erkennbar, dass in diesem Fall alle möglichen kleinsten Mengen von Regresstransitionen erzeugt wurden, da von den 4 Transitionen pro „Philosoph“ jeweils genau eine, aber jede beliebige Transition Regresstransition sein kann. Für ein System mit  $n$  speisenden Philosophen ergeben sich dementsprechend  $4^n$  verschiedene Mengen von Regresstransitionen. Es ist also grundsätzlich nicht ratsam, für jedes Netz alle möglichen Mengen von Regresstransitionen zu berechnen. Für die im Rahmen der Fallstudie in Abschnitt 4.4 getesteten Netze erwies sich der Ansatz aber als brauchbar; hier trat für kein Netz eine übermäßig große Anzahl an verschiedenen Regresstransitionsmengen auf.

Im letzten Schritt testen wir nun für jede Menge  $M'_j$ , ob sie eine gültige Menge von Regresstransitionen ist. Hierzu erstellen wir das lineare Programm  $\mathcal{L}''_j$ , das entsteht, wenn alle zu den Transitionen aus einer Menge  $M'_j$  zugeordneten Nebenbedingungen aus  $\mathcal{L}$  entfernt werden. Ist  $\mathcal{L}''_j$  zulässig, bildet  $M'_j$  eine minimale Menge von Regresstransitionen, weil in diesem Fall offensichtlich keine Nebenbedingungen mehr vorhanden sind, die das System unzulässig werden lassen und wegen  $|M'_j| = |M|$  mit der LP-basierten Heuristik keine kleinere Menge an Regresstransitionen als  $M'_j$  auffindbar war.

**Beispiel 4.** Betrachten wir noch einmal das Petrinetz aus Abbildung 3.1 (S. 28). Als erste minimale Menge von Regresstransitionen kann beispielsweise  $\{t_4, t_3\}$  berechnet werden. Demnach werden die Linearen Programme  $\mathcal{L}'_4$  mit den Nebenbedingungen

$$\begin{aligned} -2a_1 - 5a_2 &\leq \varepsilon \\ -3a_1 + a_2 &\leq \varepsilon \\ 4a_2 &\leq \varepsilon \\ a_1 - 4a_2 &\leq \varepsilon \end{aligned}$$

und  $\mathcal{L}'_3$  mit den Nebenbedingungen

$$\begin{aligned} -2a_1 - 5a_2 &\leq \varepsilon \\ -3a_1 + a_2 &\leq \varepsilon \\ 4a_2 &\leq \varepsilon \\ a_1 + 2a_2 &\leq \varepsilon \end{aligned}$$

erstellt. Die Nebenbedingungen aus  $\mathcal{L}'_4$  sind hierbei den Transitionen  $T \setminus (M \setminus \{t_4\}) = \{t_0, t_1, t_2, t_4\}$  zugeordnet. Die Nebenbedingungen aus  $\mathcal{L}'_3$  entsprechen den MAXFS-Nebenbedingungen der Transitionen  $T \setminus (M \setminus \{t_3\}) = \{t_0, t_1, t_2, t_3\}$ . Zu beiden Linearen Programmen werden nun die zugehörigen elastischen Linearen Programme  $\mathcal{E}\mathcal{L}'_3$  und  $\mathcal{E}\mathcal{L}'_4$  gelöst und jeweils die Lösungen der den Nebenbedingungen zugeordneten dualen Variablen betrachtet. Die Werte aller dualen Variablen aus  $\mathcal{E}\mathcal{L}'_4$  sind ungleich 0. Daher sind alle vier Transitionen, deren MAXFS-Nebenbedingungen zu  $\mathcal{L}'_4$  gehören, in der Menge  $R_4 = \{t_0, t_1, t_2, t_4\}$  enthalten. Das Lösen von  $\mathcal{E}\mathcal{L}'_3$  ergibt, dass die den MAXFS-Nebenbedingungen von  $t_0, t_2$  und  $t_3$  zugeordneten dualen Variablen ungleich 0 sind; dementsprechend ergibt sich  $R_3 = \{t_0, t_2, t_3\}$ . Nun können die Mengen  $M'_j$  als Kombinationen aller Transitionen  $t_i \in R_4$  und  $t_j \in R_3$  gebildet werden:  $M'_0 = \{t_0, t_2\}$ ,  $M'_1 = \{t_0, t_3\}$ ,  $M'_2 = \{t_1, t_0\}$ ,  $M'_3 = \{t_1, t_2\}$ ,  $M'_4 = \{t_1, t_3\}$ ,  $M'_5 = \{t_2, t_3\}$ ,  $M'_6 = \{t_4, t_0\}$ ,  $M'_7 = \{t_4, t_2\}$  und  $M'_8 = \{t_4, t_3\}$ . Für jede Menge  $M'_j$  wird jetzt das LP  $\mathcal{L}''_j$  erstellt, das die MAXFS-Nebenbedingungen aller Transitionen aus  $T \setminus M$  enthält.  $\mathcal{L}''_0$  enthält beispielsweise die Nebenbedingungen

$$\begin{aligned} -3a_1 + a_2 &\leq \varepsilon \\ a_1 + 2a_2 &\leq \varepsilon \\ a_1 - 4a_2 &\leq \varepsilon. \end{aligned}$$

Für jedes LP  $\mathcal{L}''_j$  wird nun getestet, ob es zulässig ist; da dies für  $\mathcal{L}''_2, \mathcal{L}''_3, \mathcal{L}''_5, \mathcal{L}''_6$  und  $\mathcal{L}''_8$  zutrifft, bilden die Mengen  $M'_2, M'_3, M'_5, M'_6$  und  $M'_8$  minimale Regresstransitionsmengen.

\*

Das eben beschriebene Verfahren garantiert nicht, dass zu jedem Petrinetz tatsächlich mehrere Regresstransitionsmengen generiert werden, zumal für einige Netze nur eine kleinste Menge von Regresstransitionen existiert. Des Weiteren erstellt es, da es sich um eine Heuristik handelt, für viele Netze nur einen Teil aller möglichen Regresstransitionsmengen. Im Rahmen unserer Fallstudie aus Abschnitt 4.4 lieferte der Algorithmus brauchbare Resultate. Da wir nun dazu in der Lage sind, mehrere minimale Mengen von Regresstransitionen zu ermitteln, können wir im nächsten Abschnitt eine Heuristik entwickeln, um aus diesen Mengen eine auszuwählen, mit der Ketten von Regresstransitionen möglichst vermieden werden.

## 4.2 Vermeidung von Ketten von Regresstransitionen

In Abschnitt 2.4.2 haben wir gezeigt, dass Ketten von Regresstransitionen möglichst vermieden werden sollten. Um einen Ansatz zum Erreichen dieses Ziels entwickeln zu können, müssen wir zunächst betrachten, unter welchen Gegebenheiten solche Ketten auftreten können.

Eine Kette von Regresstransitionen im Erreichbarkeitsgraphen tritt immer dann auf, wenn zwei oder mehr Regresstransitionen direkt nacheinander schalten können. Grundsätzlich hängt dies von der Markierung des Petrinetzes ab; wenn auf allen Plätzen eines Netzes hinreichend viele Marken liegen, kann zunächst jede mögliche Kombination von Transitionen sequentiell geschaltet werden.

Für zwei Regresstransitionen  $t$  und  $t'$  sollte gelten, dass  $t$  durch das Schalten von  $t'$  nicht aktiviert wird und ebenso  $t'$  nicht durch das Schalten von  $t$  aktiviert wird. Eine gegenseitige Aktivierung hätte unmittelbar zur Folge, dass beide Transitionen nacheinander schalten können und somit wahrscheinlich lange Ketten im Erreichbarkeitsgraphen bilden. Um dies zu

verhindern, haben wir das Konzept des *Aktivierungsgraphen* entwickelt, das wir im nächsten Abschnitt vorstellen werden.

Weiterhin kann es zur Vermeidung von Ketten wichtig sein, Transitionen mit geteilten Vorplätzen zu betrachten, da sich solche Transitionen unter bestimmten Umständen gegenseitig deaktivieren können.

Auf Grundlage dieser Feststellungen werden wir eine Heuristik entwickeln, die beim Vermeiden von Ketten von Regresstransitionen helfen soll. Da wir hierfür unter anderem den eben erwähnten Aktivierungsgraphen verwenden wollen, werden wir nun zunächst dieses Konzept betrachten.

### 4.2.1 Aktivierungsgraph

An dieser Stelle führen wir den Aktivierungsgraphen ein, der bei der Vermeidung von Ketten von Regresstransitionen helfen soll.

**Definition 5** (Aktivierungsgraph). *Gegeben sei ein Petrinetz  $N = (P, T, F, W, s_0)$ .  $G = (V, E)$  ist der **Aktivierungsgraph** von  $N$  mit*

- $V = T$ ,
- $(t, t') \in E$  gdw.  $\exists p \in P : t \in \bullet p, t' \in p \bullet$ .

Ist ein Platz  $p$  Nachplatz einer Transition  $t$  und gleichzeitig Vorplatz einer Transition  $t'$ , so existiert im Aktivierungsgraphen eine Kante von  $t$  nach  $t'$ . Dies bedeutet, dass durch das Schalten von  $t$  eine Marke auf einem Vorplatz von  $t'$  erzeugt wird. Somit kann  $t'$  durch das Schalten von  $t$  potentiell aktiviert werden.

Betrachten wir nun genauer, unter welchen Umständen durch das Schalten von  $t$  die Transition  $t'$  garantiert aktiviert wird: Seien  $t, t'$  Transitionen und sei  $t'$  nicht aktiviert. Damit  $t'$  durch das Schalten von  $t$  aktiviert wird, muss für alle Vorplätze  $p$  von  $t'$  gelten:

$W(t, p) - W(p, t) + s(p) \geq W(p, t')$ . Hierbei sei noch einmal darauf hingewiesen, dass die Funktion  $W$  den Wert 0 erhält, falls keine Kante zwischen dem jeweiligen Platz und der jeweiligen Transition existiert. Nach dem Schalten der Transition  $t$  müssen also mehr Marken auf allen Vorplätzen von  $t'$  liegen, als  $t'$  beim Schalten vom jeweiligen Platz verbraucht. Es ist offensichtlich, dass es zur Erfüllung dieser Bedingung nicht hinreichend ist, wenn  $p$  Nachplatz von  $t$  und Vorplatz von  $t'$  ist. Wenn eine Kante im Aktivierungsgraphen von  $t$  nach  $t'$  existiert, hat dies demnach nicht notwendigerweise zur Folge, dass  $t'$  durch das Schalten von  $t$  aktiviert wird. Dadurch, dass eine Marke auf einem Vorplatz von  $t'$  erzeugt wird, wird allerdings die Wahrscheinlichkeit einer Aktivierung von  $t'$  durch das Schalten von  $t$  erhöht.

Unsere weiteren Betrachtungen beziehen sich vor allem auf spezielle Schaltsequenzen, die wir *zusammenhängende Schaltsequenzen* nennen wollen.

**Definition 6** (Zusammenhängende Schaltsequenz). *Sei  $N = (P, T, F, W, s_0)$  ein Petrinetz und  $t, t_1, \dots, t_n, t' \in T$ . Eine **zusammenhängende Schaltsequenz** ist eine Schaltsequenz  $S = (tt_1 \dots t_n t')$ , für die  $t \bullet \cap \bullet t_1 \neq \emptyset, \dots, t_n \bullet \cap \bullet t' \neq \emptyset$  gilt.*

In Satz 3 soll nun die Grundlage für die Anwendung des Aktivierungsgraphen zur Vermeidung von Ketten von Regresstransitionen gelegt werden.

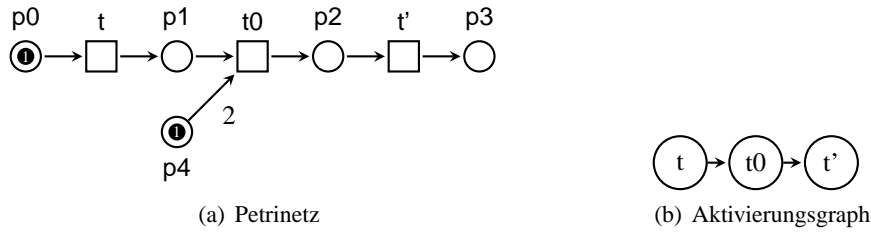


Abbildung 4.6: Beispiel 5

**Satz 3.** Sei  $N = (P, T, F, W, s_0)$  ein Petrinetz,  $G = (V, E)$  sein Aktivierungsgraph sowie  $\mathcal{W} = (t, t_1, \dots, t_n, t')$  ein kürzester Weg von  $t$  nach  $t'$  in  $G$ . Weiterhin sei  $\mathcal{S} = (tt'_1 \dots t'_m t')$  die kürzeste zusammenhängende Schaltsequenz zwischen  $t$  und  $t'$  im Petrinetz. Dann gilt  $|\mathcal{S}| > |\mathcal{W}|$ .

*Beweis.* Existiert im Aktivierungsgraphen eine Kante  $(t, t_1) \in E$ , so gilt  $t^\bullet \cap {}^\bullet t_1 \neq \emptyset$ . Es folgt  $t^\bullet \cap {}^\bullet t_1 \neq \emptyset, \dots, t_n^\bullet \cap {}^\bullet t' \neq \emptyset$  für die Transitionen im Weg  $\mathcal{W}$ . Dies entspricht genau der Bedingung, die auch für die zusammenhängende Schaltsequenz  $\mathcal{S}$  gilt. Folgt in dieser auf eine Transition  $t'_1$  die Transition  $t'_2$ , so muss mindestens einer der Nachplätze von  $t'_1$  Vorplatz von  $t'_2$  sein. Wenn also eine Schaltsequenz  $\mathcal{S}' = (tt_1 \dots t_n t')$  existiert, die dem Weg  $\mathcal{W}$  entspricht, ist sie die kürzeste zusammenhängende Schaltsequenz. Dann gilt  $|\mathcal{S}| = |\mathcal{W}| + 1$ , da wir als Länge des Weges  $\mathcal{W}$  die Anzahl der seiner Kanten angeben und als Länge der Schaltsequenz  $\mathcal{S}$  die Anzahl der in ihr enthaltenen Transitionen, welche in diesem Fall den Knoten des Weges  $\mathcal{W}$  entsprechen. Wenn solch eine Schaltsequenz nicht existiert, ist die Ungleichung  $|\mathcal{S}| > |\mathcal{W}|$  für die kürzeste zusammenhängende Schaltsequenz  $\mathcal{S}$  trotzdem erfüllt, da  $\mathcal{W}$  der kürzeste Weg mit den Bedingungen ist, die auch für  $\mathcal{S}$  gelten. Somit gilt in jedem Fall  $|\mathcal{S}| > |\mathcal{W}|$ .  $\square$

Wir können nun mit Hilfe der Länge des kürzesten Weges zwischen zwei Transitionen  $t$  und  $t'$  im Aktivierungsgraphen Aussagen darüber treffen, wie das Schalten von  $t$  die Aktivierung von  $t'$  beeinflusst. Existiert kein Weg der Länge 1 zwischen  $t$  und  $t'$ , gilt  $t^\bullet \cap {}^\bullet t' = \emptyset$ . Somit wird durch das Schalten von  $t$  keine Marke auf einem Vorplatz von  $t'$  erzeugt. Wenn die Transition  $t'$  bei einer beliebigen Markierung  $s$  nicht aktiviert ist, wird sie also auch durch das Schalten von  $t$  nicht aktiviert. Wenn  $t$  nicht aktiviert ist und ein Weg der Länge  $n$  zwischen  $t$  und  $t'$  existiert, bedeutet dies, dass das Schalten von  $t$  erst dann eine Aktivierung von  $t'$  zur Folge hat, wenn zwischen  $t$  und  $t'$  noch mindestens  $n - 1$  andere Transitionen schalten.

Die Schwächen des Konzepts liegen darin, dass einige wichtige Eigenschaften eines Petrinetzes in ihm nicht repräsentiert werden. So ist anhand des Aktivierungsgraphen nicht erkennbar, wie viele Vorplätze eine Transition besitzt. Zudem existieren im Aktivierungsgraphen keine Kantengewichte. Beide Dinge sind entscheidend, um genauer beurteilen zu können, unter welchen Umständen eine Transition eine andere aktiviert. So ist es beispielsweise möglich, dass zwischen zwei Transitionen  $t$  und  $t'$  ein Weg  $\mathcal{W} = (t, \dots, t'', \dots, t')$  im Aktivierungsgraphen existiert, obwohl  $t'$  tot ist.

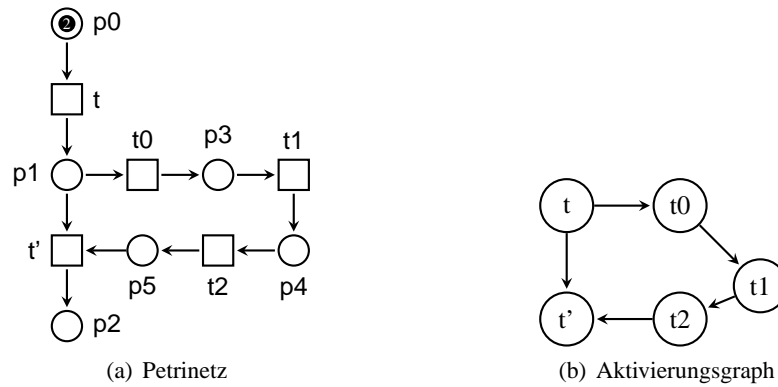


Abbildung 4.7: Beispiel 6

**Beispiel 5.** Für das Netz aus Abb. 4.6(a) existiert im Aktivierungsgraphen (Abb. 4.6(b)) ein Weg von  $t$  über  $t_0$  nach  $t'$ . Allerdings ist  $t_0$  tot, da es keine erreichbare Markierung gibt, bei der auf  $p_4$  zwei Marken liegen. Somit hat das Schalten von  $t$  bei keiner erreichbaren Markierung des Netzes einen Einfluss auf die Aktivierung von  $t'$ .

\*

Es ist auch möglich, dass die Anzahl der Transitionen, die tatsächlich zwischen zwei Transitionen  $t$  und  $t'$  geschaltet werden müssen, damit das Schalten von  $t$  Einfluss auf die Aktivierung von  $t'$  hat, deutlich größer ist als der Weg zwischen diesen Transitionen im Aktivierungsgraphen.

**Beispiel 6.** In dem in Abb. 4.7(a) dargestellten Netz existiert im zugehörigen Aktivierungsgraphen (Abb. 4.7(b)) zwar die Kante  $(t, t')$ ; dennoch müssen nach dem Schalten von  $t$  erst die Transitionen  $t_0, t_1$  und  $t_2$  schalten, damit  $t'$  aktiviert ist. Obwohl im Aktivierungsgraphen ein Weg der Länge 1 existiert, muss die Schaltsequenz  $S = (tt_0t_1t_2)$  geschaltet werden, damit  $t'$  aktiviert wird.

\*

Trotz dieser Schwächen kann das Konzept des Aktivierungsgraphen seinem Zweck, zur Vermeidung von Ketten von Regresstransitionen beizutragen, sehr dienlich sein. Wir werden im folgenden Abschnitt zeigen, wie der Aktivierungsgraph zum Erreichen dieses Ziels eingesetzt werden kann.

#### 4.2.2 Heuristik zur Vermeidung von Ketten von Regresstransitionen

In diesem Abschnitt werden wir eine Heuristik zur Vermeidung von Ketten von Regresstransitionen vorstellen. Sie besteht aus vier Bewertungskriterien, auf die wir nun eingehen werden. In den folgenden Betrachtungen gehen wir stets davon aus, dass aus mehreren minimalen Mengen von Regresstransitionen eine Menge  $R$  ausgewählt werden kann.

**(I)** Die Summe der kürzesten Wege zwischen jedem Paar  $(t, t')$  von Regresstransitionen im Aktivierungsgraph sollte möglichst lang sein.

Dies soll vermeiden, dass Regresstransitionen sich gegenseitig aktivieren. Seien  $\mathcal{W}_0, \dots, \mathcal{W}_n$

kürzeste Wege im Aktivierungsgraph zwischen allen Paaren  $(t, t')$  mit  $t, t' \in R$ . Wir ordnen jedem Weg  $\mathcal{W}_i$  eine zusammenhängende Schaltsequenz  $\mathcal{S}_i = (tt_1 \dots t_m t')$  zu. Nach Satz 3 gilt: Wenn eine solche Schaltsequenz  $\mathcal{S}_i$  existiert, dann gilt  $|\mathcal{S}| > |\mathcal{W}|$ . Wird nun die Summe der Länge aller Wege  $\mathcal{W}_i$  maximiert, wird dadurch wahrscheinlich auch die Summe der Länge aller Schaltsequenzen  $\mathcal{S}_i$  vergrößert. Somit kann dieses Kriterium in vielen Fällen bei Vermeidung von Ketten von Regresstransitionen helfen.

**(II)** Seien  $\mathcal{W}_0, \dots, \mathcal{W}_n$  kürzeste Wege im Aktivierungsgraph zwischen allen Paaren  $(t, t')$  mit  $t, t' \in R$ . Dann sollte der Wert  $\min \{|\mathcal{W}_i| \mid 0 \leq i \leq n\}$  möglichst groß sein.

Im Gegensatz zu **(I)** beziehen wir uns hier nicht auf die Summe der kürzesten Wege, sondern auf den kürzesten Weg unter diesen. Hat der kürzeste Weg von einer Transition  $t$  zu einer Transition  $t'$  die Länge 1, bedeutet dies, dass durch das Schalten von  $t$  eine Marke auf einem Vorplatz von  $t'$  erzeugt wird. Somit kann  $t'$  durch das Schalten von  $t$  aktiviert werden, was zur Folge hätte, dass beide Transitionen unmittelbar nacheinander schalten können. Analog dazu würden zwischen zwei Transitionen  $t$  und  $t'$ , zwischen denen ein relativ kurzer Weg im Aktivierungsgraphen existiert, wahrscheinlich nur wenige Transitionen schalten, bevor nach dem Schalten von  $t$  auch  $t'$  aktiviert ist. Genau dies soll mit diesem Bewertungskriterium vermieden werden.

**(III)** Die Summe der kürzesten Wege zwischen jedem Paar  $(t, t')$  von Transitionen, bei dem  $t$  eine bei der Anfangsmarkierung aktivierte Transition und  $t'$  eine Regresstransition ist, sollte möglichst groß sein.

Ziel dieses Bewertungskriteriums ist es, zu verhindern, dass bereits kurz nach Beginn der Durchmusterung des Zustandsraumes Regresskanten im Erreichbarkeitsgraphen durchlaufen werden und die erste Iteration der Sweep-Line-Methode somit frühzeitig endet. Sei  $A$  die Menge der Transitionen, die bei der Anfangsmarkierung aktiviert ist. Es gelte  $t \in A$  und  $t' \in R$ . Sei  $\mathcal{W}$  der kürzeste Weg von  $t$  nach  $t'$  im Aktivierungsgraphen. Wenn eine zusammenhängende Schaltsequenz  $\mathcal{S} = (tt_1 \dots t_m t')$  existiert, gilt nach Satz 3  $|\mathcal{S}| > |\mathcal{W}|$ . Analog zu **(I)** kann also dadurch, dass die Summe der Länge der Wege zwischen allen Paaren  $(t, t')$  maximiert wird, in vielen Fällen auch die Länge der entsprechenden Schaltsequenzen vergrößert werden. Hierdurch wird wahrscheinlich erreicht, dass zunächst ein möglichst großer Teil des Zustandsraums in der ersten Iteration der Sweep-Line-Methode durchmusterung werden kann, während der sich im Vergleich zu den folgenden Durchläufen die wenigsten persistenten Zustände im Speicher befinden. Dies vermeidet zwar nicht unmittelbar Ketten von Regresstransitionen, kann sich aber dennoch positiv auf den maximalen Speicherplatzbedarf während der Zustandsraumdurchmusterung auswirken.

**(IV)** Die Anzahl der Vorplätze, die eine Transition  $t \in R$  mit anderen Transitionen aus  $R$  teilt, sollte möglichst gering sein. Hingegen sollte die Anzahl der Vorplätze, die eine Transition  $t \in R$  mit Transitionen  $t' \in T \setminus R$  teilt, möglichst groß sein, sofern für einen solchen Vorplatz  $p$  die Ungleichung  $W(p, t') > W(t', p)$  gilt.

Hintergrund dieses Bewertungskriteriums ist die Tatsache, dass Transitionen, die einen gemeinsamen Vorplatz haben, sich gegenseitig deaktivieren können. Grundsätzlich hängt dies von der Markierung des betrachteten Platzes ab. Um zu erfahren, wie viele Marken maximal auf einem Platz liegen können, müsste er jedoch auf Beschränktheit getestet werden, wofür wiederum der Erreichbarkeitsgraph aufgebaut werden müsste. Daher ist vor der Zustands-

#### 4 Optimierung der Berechnung der Offset-Werte

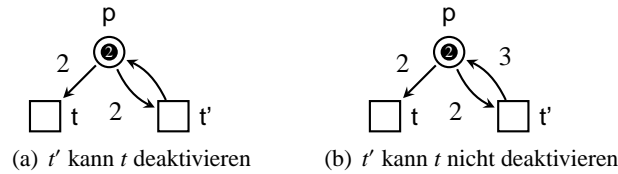


Abbildung 4.8: Illustrationen zu Bewertungskriterium (IV)

raumdurchmusterung nicht bekannt, wieviele Marken maximal auf einem Platz liegen können. Betrachten wir zunächst den Fall, in dem eine Regresstransition einen Vorplatz mit anderen Regresstransitionen teilt. Liegen zu wenige Marken auf dem Platz, um beide Transitionen nacheinander schalten zu lassen, würde sich dies für unsere Zwecke positiv auswirken, da sich die Regresstransitionen in diesem Fall gegenseitig deaktivieren würden. Liegen jedoch so viele Marken auf dem Platz, dass beide Transitionen nacheinander schalten können, würde dies zu einer Kette von Regresstransitionen im Erreichbarkeitsgraphen führen. Da unsere Priorität hier darauf liegt, solche Ketten zu vermeiden, soll verhindert werden, dass eine Regresstransition einen Vorplatz mit einer anderen Regresstransition teilt.

Teilt eine Regresstransition  $t$  hingegen einen Vorplatz  $p$  mit einer Transition  $t'$ , die nicht Regresstransition ist, müssen zunächst die Kantengewichte zwischen  $p$  und  $t'$  betrachtet werden. Sei  $s' = s + \Delta t'$  ein Folgezustand, der durch das Schalten von  $t'$  eintritt. Gilt  $W(p, t') > W(t', p)$ , dann erzeugt die Transition  $t'$  beim Schalten weniger Marken auf  $p$ , als sie von diesem Platz verbraucht (siehe Abb. 4.8(a)). Dementsprechend ist es möglich, dass das Schalten von  $t'$  die Transition  $t$  deaktiviert. Falls  $t$  mehrmals nacheinander schalten kann, wird die Länge dieser Kette zwar hierdurch nicht reduziert, da im Erreichbarkeitsgraphen alle möglichen Schaltkombinationen betrachtet werden. Wenn  $t$  aber durch das Schalten von  $t'$  deaktiviert wird, sinkt die Wahrscheinlichkeit, dass  $t$  bei den transitiven Folgezuständen von  $s'$  aktiviert ist, weil dafür zunächst wieder Marken auf  $p$  erzeugt werden müssen. Solange  $t$  deaktiviert ist, kann diese Transition in keiner Kette von Regresstransitionen auftreten. Somit kann es sich positiv auf die Vermeidung von Ketten von Regresstransitionen auswirken, wenn  $t$  einen Vorplatz mit  $t'$  teilt. Gilt hingegen  $W(p, t') \leq W(t', p)$ , werden beim Schalten von  $t'$  mindestens genauso viele Marken auf  $p$  erzeugt wie verbraucht (siehe Abb. 4.8(b)). Somit wird  $t$  durch das Schalten von  $t'$  keinesfalls deaktiviert. Deshalb wirkt sich der geteilte Vorplatz in diesem Fall nicht positiv auf die Vermeidung von Ketten von Regresstransitionen aus.

In der Praxis können die Bewertungskriterien (I) bis (III) so umgesetzt werden, dass zunächst die kürzesten Wege zwischen allen Knoten im Aktivierungsgraphen berechnet werden. Dies entspricht der Lösung einer ALL PAIRS SHORTEST PATHS PROBLEM-Instanz. Das Problem kann also beispielsweise mit dem Floyd-Warshall-Algorithmus gelöst werden. Da der Aktivierungsgraph keine Kantengewichte besitzt, wird für die im Floyd-Warshall-Algorithmus benötigte Kantengewichtsfunktion  $w : E \rightarrow \mathbb{Q}$  für alle Kanten  $e$  des Aktivierungsgraphen  $w(e) = 1$  gesetzt. Als Ergebnis erhält man die Matrix  $L$ , wobei  $l_{i,j}$  die Länge des kürzesten Weges von  $i$  nach  $j$  angibt. Über den entsprechenden Eintrag kann nun der kürzeste Weg von  $t_i$  nach  $t_j$  und somit auch leicht die Summe von kürzesten Wegen ermittelt werden. Hierbei stellt sich die Frage, wie der Eintrag  $l_{i,j} = \infty$  repräsentiert wird, der auftritt, wenn kein Weg von  $t_i$  nach  $t_j$  im Aktivierungsgraphen existiert. Grundsätzlich bietet es sich für alle drei Bewertungskri-

terien an, eine Menge  $R$  von Regresstransitionen zu bevorzugen, bei der zwischen möglichst vielen der betrachteten Paare  $(t, t')$  kein Weg im Aktivierungsgraphen existiert. Dafür müssen wir anhand der Summe der Wege erkennen können, wie viele der möglichen Wege zwischen allen betrachteten Transitionspaaren nicht existieren. Deshalb wird der Wert  $l_{i,j} = \infty$  für die genannten Bewertungskriterien folgendermaßen kodiert:

Wir betrachten ein Petrinetz  $N = (P, T, F, W, s_0)$  und eine zugehörige minimale Menge  $R$  von Regresstransitionen. Der zugehörige Aktivierungsgraph  $G$  enthält dementsprechend  $|T|$  Knoten. Folglich hat der längste Weg, der zwischen zwei Transitionen in  $G$  existieren kann, die Länge  $l_{max} = |T| - 1$ . Für die Kriterien **(I)** und **(II)** folgt nun: Es existieren  $\binom{|R|}{2} = \frac{|R| \cdot (|R| - 1)}{2}$  verschiedene Paare von Regresstransitionen. Wenn für alle  $t, t' \in R$  Wege in  $G$  sowohl von  $t$  nach  $t'$  als auch von  $t'$  nach  $t$  existieren, kann die Summe dieser Wege nicht größer sein als  $2 \cdot l_{max} \cdot \binom{|R|}{2} = (|T| - 1) \cdot |R| \cdot (|R| - 1)$ . Deshalb kann jeder Eintrag  $l_{i,j} = \infty$  durch  $l_{i,j} = (|T| - 1) \cdot |R| \cdot (|R| - 1) + 1$  ersetzt werden. Seien  $R, R'$  zwei Regresstransitionsmengen für das Netz  $N$  und  $\mathcal{W}_1, \dots, \mathcal{W}_n$  sowie  $\mathcal{W}'_1, \dots, \mathcal{W}'_m$  die kürzesten Wege zwischen allen Paaren von Transitionen aus  $R$  bzw.  $R'$ . Es gilt nicht immer  $n = m$ , da nicht zwischen allen Paaren von Transitionen der jeweiligen Regresstransitionsmenge Wege existieren müssen. Seien nun  $S = \sum_{t_i, t_j \in R} l_{i,j}$  und  $S' = \sum_{t_i, t_j \in R'} l_{i,j}$ . Aufgrund des Wertes, der für  $l_{i,j} = \infty$  eingesetzt wird, gilt  $n > m \Rightarrow S > S'$  und  $n < m \Rightarrow S < S'$ . Bei Kriterium **(I)** werden also Regresstransitionsmengen bevorzugt, zwischen deren Transitionen im Aktivierungsgraphen möglichst viele Wege nicht existieren. Welche Länge die existierenden Wege haben, ist erst entscheidend, wenn  $n = m$  gilt.

Für das Bewertungskriterium **(III)** muss neben der Menge  $R$  auch die Menge  $A$  der bei der Anfangsmarkierung aktivierten Transitionen betrachtet werden. Die maximale Anzahl aller Wege von Transitionen aus  $A$  zu Transitionen aus  $R$  im Aktivierungsgraphen beläuft sich auf  $|A| \cdot |R|$ . Für dieses Kriterium kann der Eintrag  $l_{i,j} = \infty$  dementsprechend durch  $l_{i,j} = |A| \cdot |R| \cdot l_{max} + 1$  ersetzt werden.

Wir definieren nun eine Funktion  $f$ , die für jede Menge  $R$  von Regresstransitionen angibt, wie stark sie den vier Bewertungskriterien entspricht. Am Ende soll die Menge  $R$  ausgewählt werden, für die  $f$  den größten Wert zurückgibt. Die einzelnen Bewertungskriterien liefern hierfür folgende numerische Werte:

- **(I)** liefert die Summe der Längen der kürzesten Wegen zwischen allen Paaren von Transitionen aus  $R$ .
- **(II)** liefert den Wert  $\min \{|\mathcal{W}_i| \mid 1 \leq i \leq n\}$  für alle kürzesten Wege  $\mathcal{W}_1, \dots, \mathcal{W}_n$  zwischen den Paaren von Transitionen aus  $R$ .
- **(III)** liefert die Summe der Längen der kürzesten Wege von allen Transitionen, die bei der Anfangsmarkierung aktiviert sind zu allen Transitionen aus  $R$ .
- **(IV)** liefert die Anzahl der Vorplätze, die eine Transition  $t \in R$  mit anderen Transitionen aus  $R$  bzw. Transitionen aus  $T \setminus R$  teilt.

Die Werte, die für die einzelnen Bewertungskriterien für eine bestimmte Menge an Regresstransitionen zurückgegeben werden, bezeichnen wir entsprechend der Nummern der Kriterien mit  $h_1$  bis  $h_4$ . Die Argumente der Funktion  $f$  sind folglich  $h_1, h_2, h_3$  und  $h_4$ .

Während die Bewertungen **(I)** bis **(III)** einen einzelnen Wert zurückgeben und somit unmittelbar den Wert für  $h_1, h_2$  und  $h_3$  liefern, muss aus den zwei Werten, die **(IV)** zurückliefert, ein

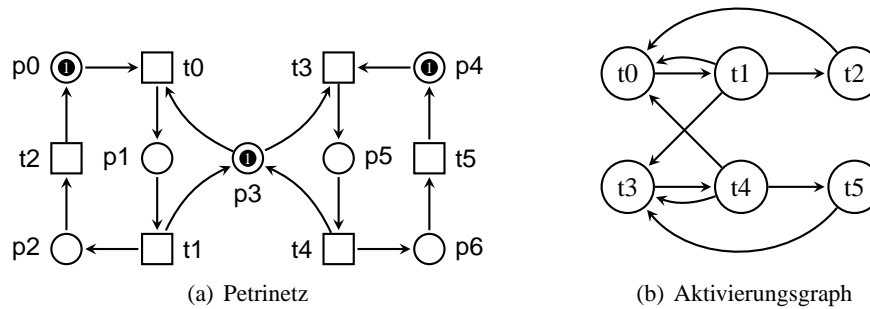


Abbildung 4.9: Mutex-Algorithmus

einzelner kombiniert werden. Dies kann folgendermaßen umgesetzt werden: Als Ausgangswert dient  $h_4 = 0$ . Für jeden Vorplatz, den eine Transition  $t \in R$  mit anderen Transitionen aus  $R$  teilt, wird der Wert von  $h_4$  um 1 verringert. Für jeden Vorplatz  $p$ , den  $t$  hingegen nur mit Transitionen aus  $t' \in T \setminus R$  teilt und für mindestens ein  $t'$  die Ungleichung  $W(p, t') > W(t', p)$  gilt, wird der Wert von  $h_4$  um 1 erhöht.

Alle vier Werte sollen nun gewichtet werden. Deshalb soll die Funktion  $f$  den Wert  $f(h_1, h_2, h_3, h_4) = \frac{h_1}{u_1} + \frac{h_2}{u_2} + \frac{h_3}{u_3} + \frac{h_4}{u_4}$  erhalten, wobei die Werte der Variablen  $u_1$  bis  $u_4$  für ein bestimmtes Petrinetz konstant sein müssen. Wir wollen nun geeignete Werte für die Variablen  $u_1$  bis  $u_4$  finden. Für die Bewertungskriterien **(I)** und **(III)** können wir hierbei die Werte nutzen, die wir für Einträge der Form  $l_{i,j} = \infty$  im jeweiligen Kriterium eingesetzt haben; wir setzen  $u_1 = (|T| - 1) \cdot |R| \cdot (|R| - 1) + 1$  und  $u_3 = |A| \cdot |R| \cdot (|T| - 1) + 1$ . Die Werte für  $\frac{h_1}{u_1}$  und  $\frac{h_3}{u_3}$  werden demnach größer als 1, sobald im Aktivierungsgraph ein Weg zwischen den betrachteten Transitionen nicht existiert.

Für das Kriterium **(II)** setzen wir  $u_2 = l_{max} = |T| - 1$ , weil wir davon ausgehen, dass in den meisten Fällen ein Weg zwischen den Regresstransitionen innerhalb einer Menge  $R$  existiert; dann gilt  $0 < \frac{h_2}{u_2} < 1$ . Sollte dies nicht der Fall sein, gilt  $h_2 = u_1$ . Der Wert für  $\frac{h_2}{u_2}$  wird also dann, wenn im Aktivierungsgraphen kein Weg zwischen den Transitionen aus  $\bar{R}$  existiert, deutlich größer als in allen anderen Fällen. Dies ist durchaus so gewollt, da solche Mengen bevorzugt ausgewählt werden sollen.

Da bei Kriterium **(IV)** Vorplätze betrachtet werden, erscheint es sinnvoll, den Wert von  $u_4$  an die Anzahl der Plätze  $|P|$  des betrachteten Petrinetzes zu koppeln. Wir haben uns für  $u_4 = |P|$  entschieden, da  $|P|$  die maximale Anzahl an Vorplätzen einer Transition ist.

Als Funktionswert für die Heuristik erhalten wir also insgesamt

$$f(h_1, h_2, h_3, h_4) = \frac{h_1}{(|T| - 1) \cdot |R| \cdot (|R| - 1) + 1} + \frac{h_2}{|T| - 1} + \frac{h_3}{|A| \cdot |R| \cdot (|T| - 1) + 1} + \frac{h_4}{|P|}.$$

**Beispiel 7.** Betrachten wir die Berechnung der Funktion  $f$  am Beispiel des Petrinetzes für den Mutex-Algorithmus aus Abb. 4.9. In diesem können unter anderem die Regresstransitions-

mengen  $R_1 = \{t_2, t_5\}$  und  $R_2 = \{t_1, t_3\}$  gewählt werden. Des Weiteren wird

$$L = \begin{pmatrix} 0 & 1 & 2 & 2 & 3 & 4 \\ 1 & 0 & 1 & 1 & 2 & 3 \\ 1 & 2 & 0 & 3 & 4 & 5 \\ 2 & 3 & 4 & 0 & 1 & 2 \\ 1 & 2 & 3 & 1 & 0 & 1 \\ 3 & 4 & 5 & 1 & 2 & 0 \end{pmatrix}$$

für die kürzesten Wege zwischen allen Paaren von Knoten im Aktivierungsgraphen berechnet. In diesem Petrinetz gilt  $l_{max} = |T| - 1 = 5$ ,  $|R| = 2$  sowie  $|A| = 2$ , da bei der Anfangsmarkierung zwei Transitionen aktiviert sind, nämlich  $t_0$  und  $t_3$ . Die Werte für  $u_1$  bis  $u_4$  ergeben sich durch  $u_1 = (|T| - 1) \cdot |R| \cdot (|R| - 1) + 1 = 11$ ,  $u_2 = l_{max} = 5$ ,  $u_3 = |A| \cdot |R| \cdot (|T| - 1) + 1 = 21$  sowie  $u_4 = |P| = 7$ . Da zwischen allen Transitionen im Aktivierungsgraphen Wege existieren, tritt der Eintrag  $l_{i,j} = \infty$  nicht auf. Würde für mindestens ein  $i$  und ein  $j$   $l_{i,j} = \infty$  gelten, müsste dieser Eintrag durch  $l_{i,j} = (|T| - 1) \cdot |R| \cdot (|R| - 1) + 1 = 11$  ersetzt werden.

Nun können wir die Werte  $h_1$  bis  $h_4$  jeweils für Regresstransitionsmengen  $R_1$  und  $R_2$  berechnen.

Für  $R_1$  wird zunächst die Länge der kürzesten Wege von  $t_2$  nach  $t_5$  und von  $t_5$  nach  $t_2$  summiert:  $h_1 = l_{2,5} + l_{5,2} = 5 + 5 = 10$ . Beide Wege haben die Länge 5, daher folgt  $h_2 = 5$ . Für  $h_3$  müssen die kürzesten Wege von allen Transitionen aus  $A = \{t_0, t_3\}$  zu allen Transitionen aus  $R_1 = \{t_2, t_5\}$  summiert werden; es folgt  $h_3 = l_{0,2} + l_{0,5} + l_{3,2} + l_{3,5} = 2 + 4 + 4 + 2 = 12$ . Da weder  $t_5$  noch  $t_2$  geteilte Vorplätze haben, gilt  $h_4 = 0$ . Man erhält also für  $R_1$  den Funktionswert

$$f = (h_1, h_2, h_3, h_4) = \frac{h_1}{u_1} + \frac{h_2}{u_2} + \frac{h_3}{u_3} + \frac{h_4}{u_4} = \frac{10}{11} + \frac{5}{5} + \frac{12}{21} + \frac{0}{7} \approx 2,48.$$

Berechnen wir nun die Werte  $h_1$  bis  $h_4$  für die Regresstransitionsmenge  $R_2 = \{t_1, t_3\}$ . Es gilt  $h_1 = l_{1,3} + l_{3,1} = 1 + 3 = 4$ . Der kürzere dieser beiden Wege hat die Länge 1; daher gilt  $h_2 = 1$ . Für  $h_3$  folgt analog zur eben für  $R_1$  geführten Berechnung:

$h_3 = l_{0,1} + l_{0,3} + l_{3,1} + l_{3,3} = 1 + 2 + 3 + 0 = 6$ . Um einen Wert für  $h_4$  zu erhalten, müssen die geteilten Vorplätze der Transitionen aus  $R_2$  betrachtet werden: Als Ausgangswert wird  $h_4 = 0$  gesetzt.  $t_1$  hat keine geteilten Vorplätze; daher hat diese Transition keinen Einfluss auf den Wert von  $h_4$ .  $t_3$  teilt sich den Vorplatz  $p_3$  mit  $t_0$ . Weil  $t_0$  nicht in  $R_2$  enthalten ist und  $W(p_3, t_0) > W(t_0, p_3)$  gilt, wird der Wert von  $h_4$  um 1 erhöht. Insgesamt folgt demnach  $h_4 = 1$ . Jetzt kann für  $R_2$  der Funktionswert

$$f = (h_1, h_2, h_3, h_4) = \frac{h_1}{u_1} + \frac{h_2}{u_2} + \frac{h_3}{u_3} + \frac{h_4}{u_4} = \frac{4}{11} + \frac{1}{5} + \frac{6}{21} + \frac{1}{7} \approx 0,99$$

berechnet werden.

Da der Wert von  $f$  für  $R_1$  höher ist als für  $R_2$ , wird  $R_1$  als bevorzugte Menge von Regresstransitionen ausgewählt.

Insgesamt können mit der in Abschnitt 4.1.2 beschriebenen Heuristik für das Mutex-Netz 9 minimale Regresstransitionsmengen berechnet werden. Von all diesen ist  $R_1$  die Menge mit dem höchsten Funktionswert  $f$ . Um Ketten von Regresstransitionen zu vermeiden, wird beim Mutex-Netz durch die in diesem Abschnitt beschriebene Heuristik demnach die Regresstransitionsmenge  $R_1 = \{t_2, t_5\}$  ausgewählt.

\*

### 4.3 Implementierung

Wir haben die in dieser Arbeit vorgeschlagenen Optimierungsansätze zur Minimierung der Anzahl an negativen Offset-Werten sowie zur Vermeidung von Ketten von Regresstransitionen durch eine Modifikation des Model-Checking-Tools LoLA [Sch00] praktisch umgesetzt. Hierbei haben wir die algebraische Berechnung der Offset-Werte durch die geometrische Berechnung ersetzt. Die Variable  $\varepsilon$  der MAXFS-Nebenbedingungen (siehe S. 33) erhielt den Wert 0,001. Implementiert wurden zudem die in Abschnitt 2.5.2 beschriebene LP-basierte MIN IIS COVER-Heuristik sowie die Heuristik zur Erstellung mehrerer MIN IIS COVER-Lösungen aus Abschnitt 4.1.2 und die Ansätze zur Vermeidung von Ketten von Regresstransitionen aus Abschnitt 4.2.2. Zur Lösung von Linearen Programmen haben wir das Tool lp\_solve [BEN09] in LoLA integriert. Der Aufbau von elastischen Linearen Programmen wurde von uns implementiert, da lp\_solve selbst nicht über solche Methoden verfügt.

**Berechnung der Offset-Werte zu einer gegebenen MIN IIS COVER-Lösung.** In Abschnitt 4.1 haben wir beschrieben, dass aus der zulässigen Lösungsmenge, die man als Ergebnis der Optimierung erhält, ein beliebiger Vektor  $a^T$  gewählt werden kann, um die Offset-Werte zu berechnen. Um die geometrische Berechnungsmethode leicht in LoLA integrieren zu können und um mögliche Probleme beim Rechnen mit Gleitkommazahlen zu vermeiden, wollen wir ganzzahlige Offset-Werte berechnen. Es bietet sich an, dafür einen ganzzahligen Vektor  $a^T$  zu wählen. Zudem sollten die Komponenten des Vektors  $a^T$  einen möglichst kleinen Betrag aufweisen, um auch den Betrag der Offset-Werte gering zu halten. Je größer die Offset-Werte der Transitionen sind, desto größer werden auch die Progress-Werte. Somit steigt mit großen Offset-Werten auch die Wahrscheinlichkeit, dass irgendwann im Verlauf der Zustandsraumdurchmusterung die obere Grenze des Wertebereichs einer Variable, welcher ein Progress-Wert zugewiesen ist, überschritten wird. Um bezüglich des Betrags möglichst kleine Offset-Werte zu erhalten, kann man folgendes Optimierungsproblem formulieren:

Gegeben sei ein Petrinetz und eine zugehörige minimale Menge an Regresstransitionen  $M$ . Zunächst bilden wir ein Optimierungsproblem  $\mathcal{L}$ , dessen Nebenbedingungen genau die MAXFS-Nebenbedingungen des gegebenen Petrinetzes sind, deren zugehörige Transitionen nicht in  $M$  enthalten sind. In  $\mathcal{L}$  befinden sich dementsprechend alle Nebenbedingungen, deren zugeordnete Transitionen einen positiven Offset-Wert erhalten werden. Um nun einen Lösungsvektor für  $\mathcal{L}$  zu erhalten, dessen Komponenten bezüglich des Betrags möglichst klein sind, definieren wir für  $\mathcal{L}$  die Zielfunktion  $\min \sum_j |x_j|$ , wobei mit  $x_j$  die einzelnen Variablen von  $\mathcal{L}$  bezeichnet werden. Da diese Zielfunktion nicht linear ist, ist  $\mathcal{L}$  kein Lineares Programm. Wir werden nun eine Möglichkeit betrachten, Optimierungsprobleme mit dieser Zielfunktion zu linearisieren. Hierbei orientieren wir uns an der in [Gas03] dargestellten Vorgehensweise.

Sei  $\mathcal{L}'$  das LP, das durch Linearisierung aus  $\mathcal{L}$  gebildet werden soll. Für jede Variable  $x_j$  aus  $\mathcal{L}$  existieren in  $\mathcal{L}'$  zwei Variablen, nämlich  $x'_j$  und  $x''_j$ . Der Wert für jede dieser Variablen soll größer oder gleich 0 sein. Während die Nebenbedingungen in  $\mathcal{L}$  die Form  $Ax \leq \varepsilon$  hatten, haben sie nun in  $\mathcal{L}'$  die Form  $Ax' - Ax'' \leq \varepsilon$ . Hieraus wird der Hauptgedanke dieser Linearisierung ersichtlich: Statt der unbeschränkten Variable  $x_j$  aus  $\mathcal{L}$  existieren jetzt in  $\mathcal{L}'$  die nichtnegativen Variablen  $x'_j$  und  $x''_j$ , die sich durch die Gleichung  $x_j = x'_j - x''_j$  wiederum zu

einer unbeschränkten Variable kombinieren lassen. Um nun den minimalen Wert für  $\sum_j |x_j|$  zu erhalten, muss die Summe aller  $x'_j$  und  $x''_j$  minimiert werden. Das Optimierungsproblem  $\mathcal{L}$  der Form

$$\begin{aligned} \min \quad & \sum_j |x_j| \\ \text{s.t.} \quad & Ax \leq \varepsilon \end{aligned}$$

wird also in das LP  $\mathcal{L}'$  der Form

$$\begin{aligned} \min \quad & \sum_j x'_j + x''_j \\ \text{s.t.} \quad & Ax' - Ax'' \leq \varepsilon \\ & x' \geq 0, x'' \geq 0 \end{aligned}$$

umgewandelt. Wurden die optimalen Lösungsvektoren  $x'^*$  und  $x''^*$  gefunden, wird  $x^* = x'^* - x''^*$  berechnet, so dass jede Vektorkomponente  $x_j^*$  wieder einer Transition des Petrinetzes zugeordnet werden kann.

Wir haben nun den zulässigen Vektor  $x$  erhalten, dessen Komponenten innerhalb der zulässigen Lösungsmenge der MAXFS-Nebenbedingungen den kleinsten Betrag besitzen. Dieser liegt nach unseren Betrachtungen aus Abschnitt 4.1 auch in der Lösungsmenge der MAXFLS<sup>></sup>-Nebenbedingungen. Es bleibt noch die Aufgabe, einen ganzzahligen Vektor aus der zulässigen Menge der MAXFS-Nebenbedingungen zu finden. Wir haben versucht, das LP  $\mathcal{L}'$  durch `lp_solve` als ganzzahliges Lineares Programm lösen zu lassen. Allerdings nahm dies abhängig vom Netz und insbesondere von der Größe des Netzes sehr viel Zeit in Anspruch, so dass uns dies nicht als geeignete Methode zur Lösung dieses Problems erscheint. Da die Lösung ganzzahliger Linearer Programme im Allgemeinen recht zeitaufwändig ist, ist nicht davon auszugehen, dass der Lösungsvorgang durch die Verwendung eines anderen Tools als `lp_solve` wesentlich beschleunigt wird. Somit muss eine ganzzahlige Lösung ausgehend vom nicht-ganzzahligen Optimum  $x^*$  gesucht werden. Da die zulässige Menge der MAXFLS<sup>></sup>-Nebenbedingungen ein unbeschränktes konvexes Polyeder bildet und  $x^*$  innerhalb dieser Menge liegt, können wir dabei folgendermaßen vorgehen:

Zunächst ermitteln wir den größten Betrag, der innerhalb der Komponenten  $x_j^*$  von  $x^*$  vorkommt:  $m = \max\{|x^*(0)|, \dots, |x^*(|P| - 1)|\}$ . Anschließend bestimmen wir für jede Vektorkomponente  $x_j^*$  den Wert  $x_j^{*'} = x_j^*/m$ . Es gilt nun für alle Komponenten  $x_j^{*'}$ :  $-1 \leq x_j^{*'}$ .

Jetzt werden folgende Schritte ausgeführt, um einen Vektor  $a^T$  mit gültigen Offset-Werten zu erhalten:

1. Seien  $a^T$  und  $a'^T$  Vektoren der Dimension  $|P|$ . Für alle  $j$  mit  $0 \leq j \leq |P| - 1$ : Setze  $a_j = 0$  und  $a'_j = 0$ .
2. Für alle  $j$  mit  $0 \leq j \leq |P| - 1$ : Setze  $a'_j = a'_j + x_j^{*'}$ .
3. Für alle  $j$  mit  $0 \leq j \leq |P| - 1$ : Setze  $a_j = [a'_j]$ .
4. Falls  $a^T$  ein gültiger Lösungsvektor ist, terminiere.
5. Gehe zu 2. .

#### 4 Optimierung der Berechnung der Offset-Werte

Wir addieren also den Vektor  $x^{*T}$  sooft zum Vektor  $a'^T$  dazu, bis die gerundeten Komponenten von  $a'^T$  eine zulässige Lösung ergeben. Eine andere Möglichkeit wäre, statt des einmaligen Rundens in Schritt 3 für jede Vektorkomponente  $a'_j$  die Werte  $[a'_j-0.5]$  und  $[a'_j+0.5]$  zu bestimmen (also jeweils auf- und abzurunden) und für alle Vektoren, die sich aus diesen Komponenten kombinieren lassen, zu testen, ob sie eine zulässige Lösung bilden. Diese Vorgehensweise würde zwar die Wahrscheinlichkeit zum Auffinden von Offset-Werten mit möglichst kleinen Beträgen erhöhen; für ein Netz mit  $|P|$  Plätzen müssten jedoch  $2^{|P|}$  Vektoren auf Zulässigkeit geprüft werden. Aufgrund der exponentiellen Zeitkomplexität ist diese Variante nicht zu empfehlen. Deshalb haben wir uns dafür entschieden, die Vektorkomponenten  $a'_j$  in Schritt 3 nur jeweils einmal zu runden.

Ob  $a^T$  ein gültiger Lösungsvektor ist (also innerhalb der zulässigen  $\text{MAXFLS}^>$ -Menge liegt), wird getestet, indem für jede Transition  $t$  der Offset-Wert  $o(t) = a^T \cdot \Delta t$  berechnet wird. Gilt  $o(t) < 0$  genau dann, wenn auch  $t \in M$  gilt, ist  $a^T$  ein gültiger Lösungsvektor.

Somit erhält man einen ganzzahligen Vektor  $a^T$ , der zur Berechnung der Offset-Werte nach der geometrischen Methode genutzt werden kann.

### 4.4 Fallstudie

In diesem Abschnitt wird anhand von Messungen überprüft, inwiefern die Leistung der Sweep-Line-Methode durch die vorgestellten Optimierungsansätze tatsächlich verbessert wird. Die Messwerte für die Sweep-Line-Methode mit der optimierten Offset-Wert-Berechnung, welche die geometrisch basierte Berechnungsmethode zur Grundlage hat, werden mit den Messwerten verglichen, die die Anwendung der Sweep-Line-Methode mit dem algebraischen Ansatz ohne weitere Optimierung liefert. Hierzu wurde die von uns modifizierte Version des Tools LoLA verwendet, wobei die algebraische Berechnungsmethode der Offset-Werte durch die in Abschnitt 4.3 genannten Methoden, also die geometrische Berechnungsmethode und die Heuristik, ersetzt wurde. Um Werte zum Vergleich mit der algebraischen Berechnungsmethode zu erhalten, wurden die Zustandsräume derselben Netze zusätzlich mit einer nicht modifizierten LoLA-Version durchmustert. Werte, die mit der modifizierten LoLA-Version erstellt wurden, sind im Folgenden mit (H) (für „Heuristik“) gekennzeichnet; Werte, die mit der nicht modifizierten Version erstellt wurden, kennzeichnen wir mit (O) (für „Original“).

Alle Berechnungen wurden auf einem PC mit Intel-Prozessor (3.0 GHz) und 3.5 GB RAM unter Linux ausgeführt.

Die Heuristik wurde an insgesamt 32 Petrinetzen eines GALS-Projekts [SRK05] getestet. Die Grundidee von GALS-Techniken zum Schaltkreisentwurf (*Globally Asynchronous Locally Synchronous*) ist, dass lokal synchron arbeitende Blöcke miteinander über asynchrone *Wrapper* kommunizieren. Bei der Konstruktion des Wrappers muss darauf geachtet werden, dass keine *Hazards* auftreten. Dies sind Zustände, bei denen das Ausgangssignal einen undefinierten Wert annimmt. Um potentielle Hazards zu erkennen, muss ein formales Modell des Wrappers erstellt werden. In [SRK05] wird eine Methode zur Modellierung eines GALS-Wrappers mit Petrinetzen vorgestellt. Das Auftreten eines Hazards kann so auf ein Model-Checking-Problem reduziert werden, nämlich die Erreichbarkeit einer bestimmten Markierung des Petrinetzes.

In Tabelle 4.1 sind die Messergebnisse für die einzelnen Netze aufgeführt. Ziel der Berechnungen war es, den Erreichbarkeitsgraphen unter Verwendung der Sweep-Line-Methode in

Netz	Heuristik					Original				peak(H) peak(O)
	Zeit	RT(H)	Mengen	sweeps(H)	peak(H)	RT(O)	sweeps(O)	peak(O)		
pausable_clock_generator_01	00:02,3	4	3	15	1.959	15	14	2.176	90,0%	
pausable_clock_generator_02	<00:01	3	24	4	17	3	4	12	141,7%	
pausable_clock_generator_03	<00:01	4	108	3	8	4	2	8	100,0%	
timeout_generator_01	00:52,6	4	1	119	578.112	19	92	798.366	72,4%	
timeout_generator_02	<00:01	4	6	127	10	11	10	135	94,1%	
timeout_generator_03	<00:01	4	54	4	11	4	4	11	100,0%	
output_port_01	00:15,8	4	1	77	47.697	25	15	28.986	164,6%	
output_port_02	<00:01	2	1	16	3.547	19	17	6.995	50,7%	
output_port_03	<00:01	2	1	13	111	8	10	118	94,1%	
output_port_04	<00:01	2	3	13	80	4	15	96	83,3%	
output_port_05	<00:01	2	6	5	23	2	5	30	76,7%	
clock_control_01	<00:01	4	1	21	259	8	27	381	68,0%	
clock_control_02	<00:01	4	1	15	154	15	13	180	85,6%	
clock_control_03	<00:01	4	36	6	110	8	17	94	117,0%	
clock_control_04	<00:01	4	53	3	9	4	8	11	81,8%	
input_port_01	07:45,4	4	12	OOM		21	OOM			
input_port_02	<00:01	3	12	13	2.863	11	17	4.644	61,6%	
input_port_03	<00:01	1	2	14	101	3	28	255	39,6%	
input_port_04	<00:01	1	2	10	95	3	31	259	36,7%	
wrapper_01	20:25,9	2	2	OOM		50	OOM			
wrapper_01b	04:55,0	1	2	OOM		47	OOM			
wrapper_02	07:55,0	1	3	OOM		18	OOM			
wrapper_02b	09:02,5	1	3	OOM		16	OOM			
wrapper_03	10:38,0	1	3	OOM		4	OOM			
wrapper_04	03:04,8	1	3	14	328.129	4	18	428.542	76,6%	
wrapper_05	00:21,6	1	3	11	61.843	5	13	109.213	56,6%	
wrapper_06	00:23,6	1	3	10	64.642	5	14	115.006	56,2%	
wrapper_07	00:14,1	1	3	8	132.135	4	11	159.635	82,8%	
wrapper_08	00:30,5	1	3	8	263.070	1	8	260.118	101,1%	
wrapper_09	00:05,8	1	3	8	51.902	2	6	41.500	125,1%	
wrapper_10	00:14,5	1	3	10	101.347	1	10	95.000	106,7%	
wrapper_11	00:03,5	1	3	11	30.931	1	10	25.600	120,8%	

Tabelle 4.1: Messwerte für Petrinetze eines GALs-Projekts unter Verwendung der Sweep-Line-Methode und Stubborn Sets

#### 4 Optimierung der Berechnung der Offset-Werte

Kombination mit Stubborn Sets (die wir bereits in Abschnitt 2.3 erwähnten) zu erstellen. Die einzelnen Spalten der Tabelle stehen hierbei für folgende Werte:

- *Netz*: das jeweilige getestete Petrinetz
- *Zeit*: die vom LoLA-Prozess genutzte *user*-Zeit in Minuten
- *RT*: Anzahl der berechneten Regresstransitionen
- *Mengen*: Anzahl der Mengen von Regresstransitionen, die durch die Heuristik erstellt wurden
- *sweeps*: Anzahl der Iterationen der Sweep-Line-Methode
- *peak*: maximale Anzahl der Zustände, die während der Zustandsraumdurchmusterung im Speicher repräsentiert wurden
- $\frac{peak(H)}{peak(O)}$  in Prozent

Der Tabelleneintrag „OOM“ (für *out of memory*) besagt, dass der Speicherplatz für das jeweilige Petrinetz mit der entsprechenden Offset-Wert-Berechnungsmethode nicht ausreichte, um die Zustandsraumdurchmusterung komplett durchzuführen. Dieser Fehler trat in allen Fällen erst während der Zustandsraumdurchmusterung selbst und nicht während der Berechnung der Offset-Werte auf.

Anhand der Messwerte in der Spalte  $\frac{peak(H)}{peak(O)}$  kann man erkennen, dass die Heuristiken nicht grundsätzlich den gewünschten Effekt der Einsparung von zu speichernden Zuständen erzielen. Bei 16 der 32 Netze wurde die Anzahl der *peak*-Zustände durch die Verwendung der Heuristik reduziert; bei 7 Netzen wurde sie allerdings erhöht. Bei allen Netzen, deren Erreichbarkeitsgraph bei Verwendung der algebraischen Methode nicht in den Speicher passte, war der Erreichbarkeitsgraph auch bei Verwendung der Heuristik zu groß für den Speicher. Da allerdings Stubborn Sets in Kombination mit der Sweep-Line-Methode verwendet wurden, können hieraus nur bedingt Schlussfolgerungen über die Wirksamkeit der Heuristik gezogen werden. Die Berechnung der Stubborn Sets, welche mit der Sweep-Line-Methode kombiniert werden, erfolgt in LoLA offensichtlich in Abhängigkeit der Offset-Werte aller Transitionen. Führt man die Zustandsraumdurchmusterung für ein Netz zweimal mit unterschiedlichen Offset-Werten aus, so kann sich die Anzahl der persistenten Zustände auch dann unterscheiden, wenn in beiden Durchmusterungen die gleichen Transitionen als Regresstransitionen gewählt werden. Somit ist es nicht möglich, aus Messwerten, die unter gleichzeitiger Verwendung beider Reduktionsmethoden erstellt wurden, Rückschlüsse über einen Zusammenhang zwischen den gewählten Regresstransitionen und der Anzahl an persistenten Zuständen zu ziehen. Aus diesem Grund werden wir jetzt die Messwerte näher betrachten, die man erhält, wenn man die Sweep-Line-Methode als einzige Reduktionsmethode anwendet.

Die Messwerte sind in Tabelle 4.2 aufgeführt. Zusätzlich zu den Spalten der Tabelle 4.1 enthält diese Tabelle jeweils für die Heuristik und die algebraische Berechnungsmethode die Spalte *persistent*, welche die Anzahl der persistenten Zustände enthält, sowie die Spalte  $|T|$ , welche die Anzahl der Transitionen des jeweiligen Petrinetzes angibt.

Zunächst kann man anhand der Tabelle feststellen, dass bei Verwendung der unmodifizierten LoLA-Version im Vergleich zur Durchmusterung mit Stubborn Sets bei 4 weiteren Netzen der

Netz	T	Heuristik						Original					
		Zeit(H)	RT(H)	Mengen	sweeps(H)	peak(H)	persistent(H)	RT(O)	sweeps(O)	peak(O)	persistent(O)	peak(H) peak(O)	
pausable_clock_generator_01	53	00:05,3	4	3	14	60.808	60.148	15	11	52.218	50.176	116,45%	
pausable_clock_generator_02	13	<00:01	3	24	4	145	94	3	4	141	104	102,84%	
pausable_clock_generator_03	13	<00:01	4	108	5	188	154	4	5	202	162	93,07%	
timeout_generator_01	93	05:50,5	4	1	OOM			19	OOM				
timeout_generator_02	37	00:02,5	4	6	10	5.524	5.368	11	9	5.825	5.756	94,83 %	
timeout_generator_03	16	<00:01	4	54	4	679	648	4	5	679	648	100,00 %	
output_port_01	102	02:08,8	4	1	17	2.047.368	2.027.520	25	10	2.003.646	1.966.512	102,18%	
output_port_02	92	00:05,8	2	1	11	53.204	40.560	19	6	82.257	75.248	64,68%	
output_port_03	54	<00:01	2	1	13	2.700	2.688	8	9	2.017	1.968	133,86%	
output_port_04	25	<00:01	2	3	13	2.700	2.688	4	12	2.540	2.480	106,30%	
output_port_05	14	<00:01	2	6	6	246	240	2	6	202	184	121,78%	
clock_control_01	38	<00:01	4	1	17	8.640	8.640	8	17	8.556	8.496	100,98%	
clock_control_02	150	<00:01	4	1	14	2.160	2.160	15	12	2.088	2.064	103,45%	
clock_control_03	26	<00:01	4	36	5	1.706	1.404	8	14	2.168	2.168	78,69%	
clock_control_04	16	<00:01	4	53	4	1.542	1.400	4	8	1.716	1.616	89,86%	
input_port_01	107	04:55,8	4	12	OOM			21	OOM				
input_port_02	80	06:42,5	3	12	4	2.702.627	2.388.144	11	9	4.174.454	4.157.568	64,74%	
input_port_03	28	00:28,1	1	2	4	194.187	147.456	3	17	516.324	516.096	37,61%	
input_port_04	30	00:30,1	1	2	4	199.925	151.552	3	17	530.660	530.432	37,67%	
wrapper_01	379	18:54,4	2	2	OOM			50	OOM				
wrapper_01b	379	03:24,1	1	2	OOM			47	OOM				
wrapper_02	176	09:58,6	1	3	OOM			18	OOM				
wrapper_02b	176	08:38,7	1	3	OOM			16	OOM				
wrapper_03	145	10:15,6	1	3	OOM			4	OOM				
wrapper_04	93	10:49,6	1	3	OOM			4	OOM				
wrapper_05	81	08:40,1	1	3	5	1.986.613	891.846	5	OOM				
wrapper_06	83	08:55,1	1	3	5	2.008.500	904.614	5	OOM				
wrapper_07	60	06:26,0	1	3	8	1.919.438	1.872.948	4	10	5.341.781	5.335.488	35,93%	
wrapper_08	56	04:20,9	1	3	8	1.259.150	1.196.660	1	8	1.231.650	1.196.660	102,23%	
wrapper_09	49	00:23,0	1	3	6	146.919	103.084	2	5	250.081	229.142	58,75%	
wrapper_10	49	01:17,2	1	3	8	402.764	170.956	1	8	420.105	170.956	95,87%	
wrapper_11	46	00:10,8	1	3	9	74.884	59.636	1	9	62.944	59.636	118,97%	

Tabelle 4.2: Messwerte für Petrinetze eines GALs-Projekts unter Verwendung der Sweep-Line-Methode

#### 4 Optimierung der Berechnung der Offset-Werte

Speicher nicht ausreichte, um die Zustandsraumdurchmusterung komplett auszuführen. Durch Anwendung der Heuristik konnte jedoch für 2 dieser Netze der Zustandsraum komplett durchgemustert werden.

Die Anzahl der Zustände, die maximal im Speicher gehalten wurden, konnte bei 11 der getesteten Netze durch Anwendung der Heuristik verringert werden; bei 10 hingegen wurde sie hierdurch erhöht. Durch die Heuristik wird dieser Wert also nicht wie erhofft bei der deutlichen Mehrzahl der Netze verringert. Grundsätzlich hängt der *peak*-Wert nicht nur von der gewählten Menge an Regresstransitionen ab, sondern von den Offset-Werten aller Transitionen. Beim Netz `pausable_clock_generator_02` wurde die Anzahl der persistenten Zustände durch die Verwendung der Heuristik reduziert, der *peak*-Wert jedoch erhöht. Bei allen anderen Netzen, bei denen die Anzahl der persistenten Zustände durch die Verwendung der Heuristik verringert oder erhöht wurde, wurde auch der *peak*-Wert entsprechend dieser Veränderung verringert oder erhöht. Um die Zusammenhänge zwischen der gewählten Menge an Regresstransitionen und den Messwerten unabhängig von den tatsächlich berechneten Offset-Werten betrachten zu können, werden wir uns dennoch bei den folgenden Betrachtungen auf die Anzahl der persistenten Zustände statt auf den *peak*-Wert beziehen.

Bei 24 der 32 Netze konnte die Anzahl der Regresstransitionen durch die Heuristik im Vergleich zur algebraischen Methode verringert werden; bei den restlichen 8 Netzen veränderte sie sich nicht. Die größte relative Verringerung der Anzahl an Regresstransitionen bezüglich der Gesamtanzahl der Transitionen eines Netzes wurde beim Netz `pausable_clock_generator_01` erreicht; hier konnte der Anteil der Regresstransitionen um knapp 20,8% von rund 28,3% auf rund 7,5% gesenkt werden. Im Schnitt konnte die Anzahl der Regresstransitionen gemessen an der Gesamtanzahl an Transitionen von etwa 14,7% auf rund 6,8% reduziert werden. Die größte absolute Verringerung der Anzahl an Regresstransitionen ist beim Netz `wrapper_01b` zu verzeichnen; wurden mit der algebraischen Berechnungsmethode noch 47 Regresstransitionen berechnet, konnte deren Anzahl durch die Heuristik auf 1 reduziert werden.

Insgesamt wurde bei 14 Netzen, deren Erreichbarkeitsgraph vollständig berechnet werden konnte, die Anzahl der Regresstransitionen reduziert. Allerdings wurden nur bei 8 dieser Netze auch die Anzahl der persistenten Zustände reduziert. Somit kann man aus dieser Fallstudie nicht schlussfolgern, dass die Minimierung der Anzahl an Regresstransitionen eine Verringerung der Anzahl der persistenten Zustände zur Folge hätte. So konnte die Anzahl der Regresstransitionen beispielsweise beim Netz `output_port_01` von 25 auf 4 reduziert werden; dennoch hat sich die Anzahl der persistenten Zustände durch die Verwendung der Offset-Werte mit der geringeren Anzahl an Regresstransitionen geringfügig vergrößert.

Des Weiteren hatten wir vermutet, dass durch eine Verringerung der Anzahl der persistenten Zustände auch die Anzahl der Iterationen der Sweep-Line-Methode sinkt, da nach Satz 1 die maximale Anzahl der Iterationen von der Anzahl der persistenten Zustände abhängt. Bei 11 der von uns getesteten Netze konnte die Anzahl der persistenten Zustände verringert werden; jedoch verringerte sich nur bei 6 von diesen auch die Anzahl der Iterationen der Sweep-Line-Methode. Hieraus lässt sich kein signifikanter Zusammenhang aus der Anzahl der persistenten Zustände und der Anzahl der Iterationen der Sweep-Line-Methode ableiten.

Zudem lässt sich beobachten, dass die Auswahl der Regresstransitionen einen entscheidenden Einfluss auf die Anzahl der persistenten Zustände hat. Betrachten wir das Netz `wrapper_11`: Hier existiert minimal eine Regresstransition, wobei drei verschiedene Transitionen als solche ausgewählt werden können. Je nachdem, welche Transition ausgewählt wird, beläuft sich die

Anzahl der persistenten Zustände auf 36.271, 59.636 oder 90.532. Wie man an der Tabelle erkennen kann, wird durch die Heuristik nicht die bezüglich der Anzahl der persistenten Zustände beste Transition ausgewählt.

An den vorhergehenden Betrachtungen wird deutlich, dass unsere Kombination aus Heuristiken zur Minimierung der Anzahl an Regresstransitionen und Vermeidung von Ketten von Regresstransitionen im Allgemeinen nicht die gewünschten Effekte zeigt. In weiteren Versuchen ist es uns nicht gelungen, einen Zusammenhang zwischen den in Abschnitt 4.2.2 beschriebenen einzelnen Bewertungskriterien der Heuristik zur Vermeidung von Ketten von Regresstransitionen und der Anzahl an persistenten bzw. *peak*-Zuständen, die man durch die Wahl der entsprechenden Regresstransitionen erhält, zu erkennen – unabhängig davon, ob die Zustandsraumdurchmusterung mit oder ohne Stubborn Sets ausgeführt wurde. Wir haben zudem getestet, ob eine Veränderung der in 4.2.2 genannten Gewichtung oder das Weglassen einzelner Kriterien signifikante Änderungen der Messwerte zur Folge hat; natürlich erhielten wir in diesen Versuchen für die einzelnen Netze zum Teil andere Messwerte als die oben aufgeführten, am Gesamtbild der Resultate änderte sich hierdurch jedoch nichts. Somit kann man davon ausgehen, dass die Heuristik zur Vermeidung von Ketten von Regresstransitionen nicht zu einer Verbesserung des Speicherplatzbedarfs bei der Zustandsraumdurchmusterung mit der Sweep-Line-Methode führt.

Aufgrund der nicht zufriedenstellenden Resultate dieser Fallstudie werden wir die Optimierungsansätze im folgenden Kapitel einer erneuten Bewertung unterziehen.



## 5 Bewertung der Optimierungsansätze

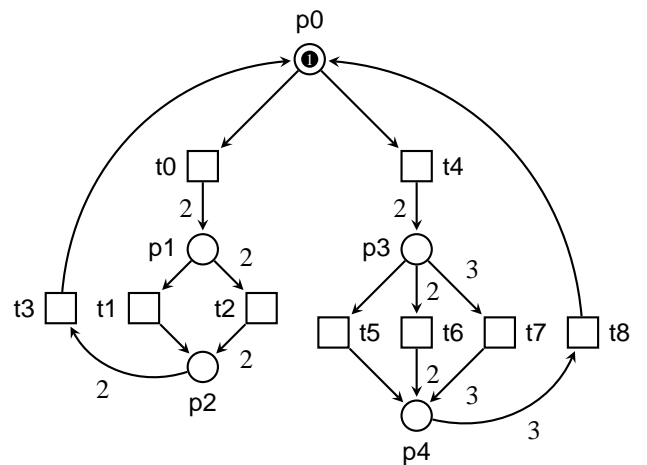
An den Messergebnissen der Fallstudie in Abschnitt 4.4 ist erkennbar, dass die Minimierung der Anzahl an Regresstransitionen und die Vermeidung von Ketten von Regresstransitionen in der mittels unserer Heuristik umgesetzten Form nicht die gewünschte Speicherplatzeinsparung zur Folge haben. Dies veranlasst uns, die beiden Optimierungsansätze in diesem Kapitel noch einmal einer kritischen Betrachtung zu unterziehen. Zunächst wollen wir aber an dieser Stelle zeigen, dass die Wahl der Offset-Werte aller Transitionen eines Netzes (also auch derer, die nicht Regresstransitionen sind) Einfluss auf den Speicherplatzbedarf bei der Zustandsraumdurchmusterung mit der Sweep-Line-Methode hat.

Wir betrachten das in Abb. 5.1(a) dargestellte Petrinetz. Bei diesem Netz kann  $R = \{t_3, t_8\}$  als Regresstransitionsmenge gewählt werden. Den Transitionen können beispielsweise die Offset-Werte  $(o(t_0), \dots, o(t_8)) = (1, 1, 2, -3, 1, 1, 2, 3, -4)$  zugewiesen werden. Anhand des in Abbildung 5.1(b) dargestellten Erreichbarkeitsgraphen des Petrinetzes können wir beschreiben, wie die Zustandsraumdurchmusterung mit diesen Offset-Werten verläuft:

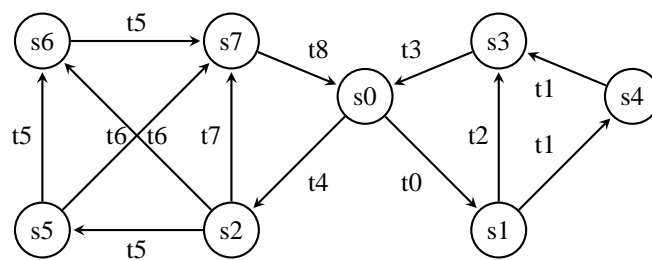
Da sowohl  $t_0$  als auch  $t_4$  bei der Anfangsmarkierung  $s_0$  aktiviert sind, werden von der Anfangsmarkierung aus die Zustände  $s_1 = s_0 + \Delta t_0$  und  $s_2 = s_0 + \Delta t_4$  erkundet. Da  $o(t_0) = o(t_4) = 1$  gilt, besitzen diese beiden Zustände den Progress-Wert 1. Deshalb werden anschließend die Folgezustände sowohl von  $s_1$  als auch von  $s_2$  durchgemustert, bevor beide Zustände aus dem Speicher gelöscht werden; dann befinden sich folgende fünf Zustände im Speicher:  $s_3 = s_1 + \Delta t_2$ ,  $s_4 = s_1 + \Delta t_1$ ,  $s_5 = s_2 + \Delta t_5$ ,  $s_6 = s_2 + \Delta t_6$  und  $s_7 = s_2 + \Delta t_7$ . In dieser Iteration der Sweep-Line-Methode ist dies die größte auftretende Anzahl an Zuständen im Speicher. Am Ende dieser Iteration wird  $s_0$  als persistent markiert, weil  $s_0$  der Zustand ist, der durch das Schalten der Regresstransitionen  $t_3$  bzw.  $t_8$  erreicht wird. Deshalb wird der gesamte Zustandsraum in der zweiten Iteration der Sweep-Line-Methode nochmals von der Anfangsmarkierung aus durchgemustert. Somit befinden sich bei der Durchmusterung mit diesen Offset-Werten während der zweiten Iteration maximal 6 Zustände (die fünf genannten und  $s_0$  als persistenter Zustand) im Speicher.

Durch eine andere Wahl der Offset-Werte für die gleiche Regresstransitionsmenge  $R$  kann die Anzahl der *peak*-Zustände bei diesem Netz reduziert werden. Die Offset-Werte für  $t_0$ ,  $t_1$ ,  $t_2$  und  $t_3$  können hierfür beibehalten werden. Anhand von Abb. 5.1(a) kann man erkennen, dass die Transitionen  $t_0$ ,  $t_1$  und  $t_2$ , sobald zwei Marken auf  $p_2$  liegen, nicht mehr aktiviert sind, bevor die Regresstransition  $t_3$  schaltet. In dem Netz existiert nur ein Zustand, für den  $s(p_2) \geq 2$  gilt, nämlich  $s_3$ . Diesem Zustand ist der Progress-Wert  $p(s_3) = o(t_0) + o(t_2) = 1 + 2 = 3$  zugewiesen<sup>1</sup>. Diese Information können wir nun nutzen, um die Offset-Werte für alle restlichen Transitionen zu bestimmen. Wir wissen, dass die Zustandsraumdurchmusterung des Petrinetzes vom Zustand  $s_3$  aus in der aktuellen Iteration nicht fortgesetzt wird, sobald dieser erreicht ist, da bei diesem nur die Regresstransition  $t_3$  aktiviert ist. Die Transitionen  $t_4$ ,  $t_5$ ,  $t_6$ ,  $t_7$  und  $t_8$  sind zudem unabhängig von den restlichen Transitionen des Netzes. Erhält

<sup>1</sup>Aufgrund der linearen Abhängigkeiten zwischen den Transitionsvektoren gilt  $o(t_0) + o(t_2) = o(t_0) + 2 \cdot o(t_1)$ . Der Progress-Wert kann dementsprechend auch über die zweite Summe ermittelt werden.



(a) Petrinetz



(b) Erreichbarkeitsgraph

Abbildung 5.1: Beispiel für die Auswirkungen der Zuweisung unterschiedlicher positiver Offset-Werte auf den Speicherplatzbedarf

nun  $t_4$  den Offset-Wert  $o(t_4) = p(s_3) + 1 = 4$ , werden die Folgezustände von  $s_2$  erst dann durchmustert, wenn  $s_3$  bereits erreicht wurde, da die Sweep-Line-Methode die Durchmusterung immer von dem nicht durchmusterten Zustand mit dem niedrigsten Progress-Wert aus weiterführt. Somit werden alle transitiven Folgezustände von  $s_2$  erst dann durchmustert, wenn die Durchmusterung aller transitiven Folgezustände von  $s_1$  bis hin zu  $s_3$  abgeschlossen ist und diese bereits wieder aus dem Speicher gelöscht wurden. Um eine konsistente Progress Measure zu erhalten, können wir wie bisher  $o(t_5) = 1$ ,  $o(t_6) = 2$  und  $o(t_7) = 3$  setzen; für  $o(t_8)$  ergibt sich dann  $o(t_8) = -o(t_4) - o(t_7) = -7$ . Wir haben nun die Offset-Werte  $(o(t_0), \dots, o(t_8)) = (1, 1, 2, -3, 4, 1, 2, 3, -7)$  erhalten. Die Zustandsraumdurchmusterung verläuft mit diesen Offset-Werten folgendermaßen: Wieder werden von  $s_0$  aus die Zustände  $s_1$  und  $s_2$  durchmustert. Im nächsten Schritt werden dann nur die beiden Folgezustände von  $s_1$ , nämlich  $s_3$  und  $s_4$ , durchmustert. Neben diesen befindet sich nach wie vor  $s_2$  im Speicher. Bis der Zustand  $s_8$  erkundet wurde, kommen keine weiteren Zustände hinzu. Nachdem  $s_0$ , der Folgezustand von  $s_3$ , als persistent markiert wurde, werden nun die drei Folgezustände von  $s_2$  durchmustert, die sich dann zusätzlich zum persistenten Zustand im Speicher befinden.  $s_2$  wurde zu diesem Zeitpunkt bereits wieder aus dem Speicher gelöscht. Da keine weiteren Zustände mehr hinzukommen können, befinden sich in dieser Iteration also maximal 4 Zustände im Speicher. Wie oben beschrieben, wird auch hier der gesamte Zustandsraum in einer zweiten Iteration durchmustert; allerdings erhöht sich die Anzahl der *peak*-Zustände in dieser nicht mehr.

Durch eine geeignete Wahl der Offset-Werte aller Transition bei gleichbleibender Regresstransitionsmenge ist es uns also gelungen, die Anzahl der *peak*-Zustände von 6 auf 4 zu reduzieren. Hieran wird deutlich, dass die Wahl der Offset-Werte auch für die Transitionen, die nicht als Regresstransitionen gewählt werden, einen Einfluss auf den maximalen Speicherplatzbedarf der Sweep-Line-Methode haben kann.

Vor diesem Hintergrund können nun die beiden ursprünglichen Optimierungsansätze einer erneuten Bewertung unterzogen werden.

## 5.1 Minimierung der Anzahl negativer Offset-Werte

In Abschnitt 2.4.2 wurde festgestellt, dass die Minimierung der Anzahl an Regresstransitionen zu einer geringeren Anzahl an persistenten Zuständen führen sollte. Dies ist offensichtlich nicht bei allen Netzen der Fall. Wir wollen an dieser Stelle ein Beispiel-Netz mit nur 7 Transitionen und 5 Plätzen angeben, bei dem die erhoffte Speicherplatzreduktion durch Minimierung der Anzahl an Regresstransitionen nicht eintritt. Für das in Abbildung 5.2 dargestellte Netz liefert die Minimierung der Anzahl an Regresstransitionen genau eine Menge, nämlich  $\{t_1\}$ . Werden die Offset-Werte mit der nicht modifizierten LoLA-Version berechnet, erhält man  $\{t_3, t_6\}$  als Menge von Regresstransitionen. Erstellt man den Erreichbarkeitsgraphen sowohl mit der modifizierten LoLA-Version, in der die Heuristik implementiert ist, als auch mit einer nicht modifizierten LoLA-Version, erhält man die in Tabelle 5.1 aufgeführten Ergebnisse. Die Spalten wurden analog zur Fallstudie in Abschnitt 4.4 benannt, wobei hier anhand der Bezeichnung der verwendeten LoLA-Version in den jeweiligen Zeilen erkennbar ist, ob die algebraische Berechnungsmethode (bezeichnet als „original“) oder die geometrische Berechnungsmethode mit der Heuristik (bezeichnet als „Heuristik“) verwendet wurde. „SWEEP“ steht für die Verwendung der Sweep-Line-Methode ohne weitere Reduktionstech-

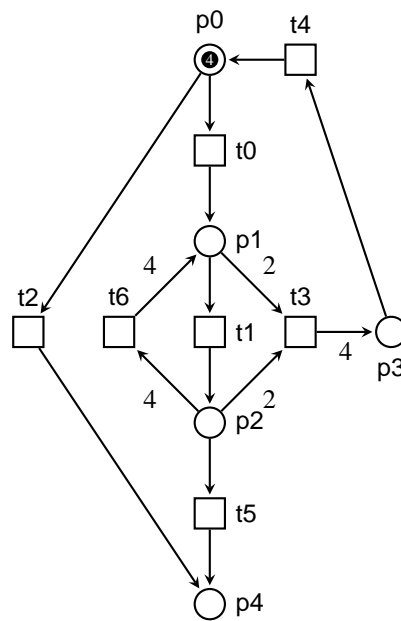


Abbildung 5.2: Petrinetz mit minimal einer Regresstransition

<i>LoLA-Version</i>	<i>Regresstransitionen</i>	<i>sweeps</i>	<i>peak</i>	<i>persistent</i>
SWEEP original	$t_3, t_6$	2	25	2
SWEEP Heuristik	$t_1$	6	62	56
SWEEP+STUBBORN original	$t_3, t_6$	2	8	2
SWEEP+STUBBORN Heuristik	$t_1$	6	16	13

Tabelle 5.1: Messwerte für das Beispiel aus Abbildung 5.2

niken, „SWEEP+STUBBORN“ für die Verwendung der Sweep-Line-Methode in Kombination mit Stubborn Sets.

Obwohl die algebraische Berechnungsmethode für zwei Transitionen negative Offset-Werte zurückliefert, während mit der Heuristik nur ein negativer Offset-Wert berechnet wird, ist die Anzahl der maximal im Speicher gehaltenen Zustände (*peak*) sowohl ohne Verwendung von Stubborn Sets als auch mit deren Verwendung mit der algebraischen Methode geringer als mit der Heuristik. Gleiches gilt für die Anzahl der persistenten Zustände und die Anzahl der Iterationen der Sweep-Line-Methode.

Auch anhand der Fallstudie in Abschnitt 4.4 wird wie erwähnt beispielsweise am Netz `output_port_01` deutlich, dass selbst eine deutliche Verringerung der Anzahl an Regresstransitionen eine Vergrößerung der Anzahl an persistenten bzw. *peak*-Zuständen zur Folge haben kann.

Wir betrachten nun, warum die Minimierung der Anzahl an Regresstransitionen nicht immer die gewünschte Speicherplatzeinsparung zur Folge haben muss.

Wenn eine Regresstransition  $t$  bei einem Zustand  $s$  im Erreichbarkeitsgraphen aktiviert ist, dann ist der Zustand  $s' = s + \Delta t$  ein persistenter Zustand. Die Menge der persistenten Zustände ist daher die Menge aller Zustände, die durch das Schalten der Regresstransitionen entstehen. Würde man tatsächlich die Anzahl der persistenten Zustände minimieren wollen, müsste man also vor der Zustandsraumdurchmusterung abschätzen, wie oft die Regresstransitionen während der Zustandsraumdurchmusterung aktiviert sind und wieviele Zustände Zielzustand mehrerer Regreskanten sind.

Am Beispiel aus Abb. 5.2 kann man erkennen, dass die Transitionen  $t_3$  und  $t_6$  in der Summe deutlich seltener aktiviert sind als  $t_1$ :  $t_3$  ist nur bei der Markierung  $s_1 = (0, 2, 2, 0, 0)$  aktiviert und  $t_6$  nur bei der Markierung  $s_2 = (0, 0, 4, 0, 0)$ . Da das Schalten dieser beiden Transitionen zu unterschiedlichen Folgemarkierungen führt, werden während der Zustandsraumdurchmusterung mit diesen beiden Transitionen als Regresstransitionen genau 2 Zustände als persistent markiert, nämlich  $s'_1 = (0, 0, 0, 4, 0)$  und  $s'_2 = (0, 4, 0, 0, 0)$ . Wird die Zustandsraumdurchmusterung hingegen mit  $t_1$  als Regresstransition durchgeführt, treten deutlich mehr persistente Zustände auf, da  $t_1$  bei jeder Markierung aktiviert ist, für die  $s(p_1) > 0$  gilt.

Beim Netz aus Abb. 5.2 spielen die Kantengewichte eine wesentliche Rolle. Würde für alle Kanten  $(x, y)$  des Petrinetzes  $W(x, y) = 1$  gelten, wären  $t_3$  und  $t_6$  bei einer größeren Anzahl von Markierungen aktiviert. Somit würde bei einer Zustandsraumdurchmusterung mit diesen beiden Transitionen als Regresstransitionen dann auch eine größere Anzahl an Zuständen als persistent markiert werden. Dennoch kann man anhand der Kantengewichte nur bedingt Rückschlüsse darauf ziehen, wie oft eine Transition während der Zustandsraumdurchmusterung aktiviert ist bzw. ob sie andere Zielzustände hat als andere Regresstransitionen. Für alle Kanten  $(x, y)$  der Netze der Fallstudie aus Kapitel 4.4 gilt  $W(x, y) = 1$ . Trotzdem erzielt man auch hier bei einigen Netzen trotz deutlicher Verringerung der Anzahl an Regresstransitionen keine Speicherplatzeinsparung bei der Zustandsraumdurchmusterung.

Wie wir bereits in Abschnitt 4.1 festgestellt haben, bringt die Minimierung der Anzahl an Regresstransitionen besonders dann Vorteile, wenn alle Regresstransitionen eliminiert werden können, weil dann während der Zustandsraumdurchmusterung keine persistenten Zustände mehr auftreten. Da Petrinetze aber in der Praxis häufig zur Modellierung reaktiver Systeme verwendet werden, dürfte dies nur bei wenigen Netzen zu erreichen sein. Für das Beispiel der „speisenden Philosophen“ aus Abbildung 4.5 verhält es sich sogar so, dass mit der algebraischen Methode bereits Offset-Werte mit der minimalen Anzahl an Regresstransitionen

berechnet werden (nämlich, wie bereits erwähnt, eine pro Philosoph). Für ein entsprechend großes System von Philosophen ist der Zeitaufwand für die Berechnung der Regresstransitionen mit den vorgestellten LP-basierten MIN IIS COVER-Heuristiken (ohne die vorgeschlagene mögliche Beschleunigung) relativ hoch, da ein Viertel aller Transitionen als Regresstransitionen identifiziert werden müssen und  $4^n$  verschiedene Mengen an Regresstransitionen berechnet werden können – die Anzahl der Regresstransitionen wird jedoch im Vergleich zur algebraischen Berechnungsmethode trotz des hohen Zeitaufwands nicht verringert. Dennoch erscheint es empfehlenswert, für jedes Netz grundsätzlich zu testen, ob eine monotone Progress Measure existiert. Die zusätzliche Laufzeit sowie der zusätzliche Speicherplatzbedarf hierfür sind im Regelfall sehr gering, da lediglich das durch die zum Netz zugehörigen MAXFS-Nebenbedingungen beschriebene Ungleichungssystem auf Zulässigkeit geprüft werden muss.

Insgesamt kann man schlussfolgern, dass die Minimierung der Anzahl an Regresstransitionen vor allem bei Netzen Speicherplatzeinsparungen bringen wird, bei denen entweder eine monotone Progress Measure existiert oder alle Transitionen während der Zustandsraumdurchmusterung in etwa gleich oft aktiviert sind. Bei allen anderen Netzen müsste darauf geachtet werden, dass Regresstransitionen gewählt werden, die möglichst selten aktiviert sind und möglichst wenige unterschiedliche Zielzustände haben – sonst kann sich auch die Wahl einer kleinen Menge von Regresstransitionen negativ auf den Speicherplatzbedarf bei der Zustandsraumdurchmusterung auswirken.

### 5.2 Vermeidung von Ketten von Regresstransitionen

In Abschnitt 2.4.2 wurde festgestellt, dass durch das Auftreten von Ketten von Regresstransitionen große bereits durchmusterter Teile des Zustandsraumes erneut durchmusterter werden, was zu einer Erhöhung des maximalen Speicherplatzbedarfs während der Durchmusterung (also einer Erhöhung der *peak*-Zahl) führen kann. Wir sind beim Versuch, solche Ketten zu vermeiden, so vorgegangen, dass wir aus mehreren kleinsten Mengen von Regresstransitionen die Menge auswählen, deren Transitionen unter anderem möglichst keine zusammenhängende Schaltsequenz bilden. Tatsächlich ist es uns jedoch nicht gelungen, ein Beispiel zu finden, bei dem die Auswahl einer minimalen Menge, mit der Ketten von Regresstransitionen auftreten, zu einer höheren Anzahl an *peak*-Zuständen führt als die Wahl aller anderen Mengen. Schauen wir uns deshalb an dieser Stelle ein Beispiel an, bei dem eine Kette von Regresstransitionen auftritt, ohne dass sich dies negativ auf den Speicherplatzbedarf auswirkt.

Für das Petrinetz aus Abbildung 5.3 können mit der in Abschnitt 4.1.2 beschriebenen Heuristik insgesamt 12 verschiedene Regresstransitionsmengen berechnet werden. Eine davon ist die Menge  $R = \{t_8, t_9, t_{11}\}$ . Aus den in ihr enthaltenen Transitionen kann man eine zusammenhängende Schaltsequenz bilden: Von der erreichbaren Markierung  $s = (0, 2, 0, 0, 0, 0, 0, 1, 0)$  aus existiert die zusammenhängende Schaltsequenz  $S = (t_9 t_8 t_{11} t_{11})$ .  $R$  ist jedoch nicht die Menge, für die sich während der Zustandsraumdurchmusterung die größte Anzahl an Zuständen im Speicher befindet, wie man an Tabelle 5.2 erkennen kann. Wir beziehen uns im Folgenden zunächst auf die Durchmusterung des Zustandsraumes ohne Stubborn Sets. Da die persistenten Zustände genau die Zustände sind, die sich durch das Schalten der Regresstransitionen ergeben, hat die Vermeidung von Ketten von Regresstransitionen im Allgemeinen keine Verringerung der Anzahl an persistenten Zuständen zur Folge. In unserem Beispiel ist die Anzahl der

## 5.2 Vermeidung von Ketten von Regresstransitionen

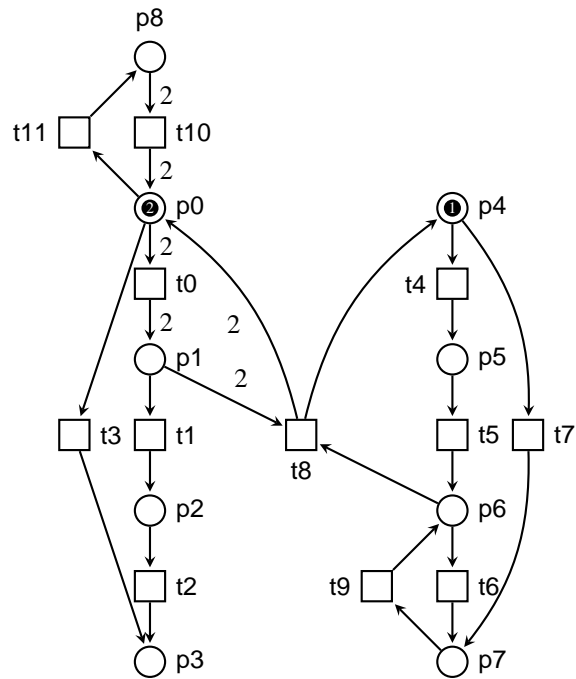


Abbildung 5.3: Petrinetz mit der Regresstransitionskette ( $t_9t_8t_{11}t_{11}$ )

<i>Regresstransitionen</i>	SWEEP			SWEEP+STUBBORN		
	<i>sweeps</i>	<i>peak</i>	<i>persistent</i>	<i>sweeps</i>	<i>peak</i>	<i>persistent</i>
$t_0, t_6, t_{10}$	3	27	17	3	9	6
$t_0, t_6, t_{11}$	3	29	23	3	10	8
$t_0, t_9, t_{10}$	3	28	17	3	11	6
$t_0, t_9, t_{11}$	3	30	23	3	11	8
$t_4, t_9, t_{10}$	2	36	24	3	13	8
$t_4, t_9, t_{11}$	4	35	28	3	14	10
$t_5, t_9, t_{10}$	2	32	14	3	11	5
$t_5, t_9, t_{11}$	4	31	20	4	12	8
$t_6, t_8, t_{10}$	2	28	14	2	10	5
$t_6, t_8, t_{11}$	3	30	21	3	12	8
$t_8, t_9, t_{10}$	3	27	17	2	10	5
$t_8, t_9, t_{11}$	3	32	21	3	12	9

Tabelle 5.2: Messwerte für das Beispiel aus Abbildung 5.3

persistenten Zustände für die Regresstransitionsmenge  $R = \{t_8, t_9, t_{11}\}$  geringer als bei 4 anderen der berechneten Regresstransitionsmengen. Zudem ist die Differenz zwischen der Anzahl der persistenten Zustände und der Anzahl der *peak*-Zustände bei 5 anderen Mengen mindestens genauso groß wie bei der Menge  $R$ . Insbesondere ist dies auch bei der Menge  $\{t_6, t_8, t_{10}\}$  der Fall, obwohl von den in dieser Menge enthaltenen Transitionen nur zwei Transitionen bei einer einzigen erreichbaren Markierung nacheinander schalten können, nämlich  $t_6$  und  $t_{10}$  bei der Markierung  $s = (0, 0, 0, 0, 0, 1, 0, 2)$ . Wie wir bereits erwähnt haben, hängt der *peak*-Wert jedoch auch von den Offset-Werten aller Transitionen und nicht nur von der Menge der Regresstransitionen ab. Somit kann man nur bedingt Rückschlüsse von dem Auftreten von Ketten von Regresstransitionen auf die Differenz zwischen dem *peak*-Wert und dem *persistent*-Wert ziehen. Der niedrigste *peak*- und *persistent*-Wert wird bei der Wahl der Regresstransitionsmengen  $\{t_0, t_6, t_{10}\}$  bzw.  $\{t_8, t_9, t_{10}\}$  erzielt.

Wendet man bei der Zustandsraumdurchmusterung zusätzlich zur Sweep-Line-Methode Stubborn Sets an, kann zur maximalen Speicherplatzreduktion ebenfalls die Menge  $\{t_0, t_6, t_{10}\}$  gewählt werden. Wie bei der Fallstudie in Abschnitt 4.4 können die Ergebnisse der Zustandsraumdurchmusterung mit Stubborn Sets ohne genauere Betrachtung der Implementation in LoLA nicht interpretiert werden.

Auch wenn die Messergebnisse aus Tabelle 5.2 einen anderen Eindruck erwecken mögen, konnten wir in unseren Tests nicht feststellen, dass ein Zusammenhang zwischen den Messwerten ohne Stubborn Sets und mit Stubborn Sets für die einzelnen Regresstransitionsmengen besteht. Eine Regresstransitionsmenge, deren Wahl bei der Zustandsraumdurchmusterung ohne Stubborn Sets zu einem relativ niedrigen *peak*- und *persistent*-Wert führt, kann bei der Zustandsraumdurchmusterung mit Stubborn Sets durchaus ungünstigere Messwerte aufweisen als viele andere Regresstransitionsmengen. Für das Netz aus Abbildung 5.3 genügt es bereits, die Kantengewichte  $W(p_8, t_{10})$ ,  $W(t_{10}, p_0)$ ,  $W(p_0, t_0)$  und  $W(t_0, p_1)$  jeweils auf 1 zu setzen, um Messwerte zu erhalten, bei denen ein entsprechender Zusammenhang nicht erkennbar ist.

Unsere Betrachtungen legen nahe, dass es für den Speicherplatzbedarf bei der Zustandsraumdurchmusterung nicht relevant zu sein scheint, ob eine Kette von Regresstransitionen existiert oder nicht. Da Ketten von Regresstransitionen in keinem Zusammenhang mit der Anzahl an persistenten Zuständen stehen, kann die Auswahl einer Menge von Regresstransitionen, welche keine Ketten bilden, durchaus zu einem hohen *persistent*-Wert während der Zustandsraumdurchmusterung führen. Im Allgemeinen wurde bei den getesteten Beispielen der *peak*-Wert durch den Versuch, Ketten zu vermeiden, nicht reduziert. Somit scheint die Vermeidung von Ketten von Regresstransitionen kein günstiger Ansatz zu sein, wenn man den Speicherplatzbedarf bei der Zustandsraumdurchmusterung reduzieren möchte.

# 6 Schlussbemerkungen

## 6.1 Zusammenfassung

Das Ziel der vorliegenden Arbeit war es, geeignete Heuristiken zur Umsetzung zweier Ansätze für die Optimierung der automatischen Berechnung von Offset-Werten für die Sweep-Line Methode zu finden. Beide Ansätze, die Minimierung der Anzahl an Regresstransitionen und die Vermeidung von Ketten von Regresstransitionen, waren durch [Sch04] vorgegeben.

Nach der Betrachtung einiger Grundlagen haben wir zunächst den bereits in [Pru09] entwickelten geometrisch basierten Ansatz zur Berechnung der Offset-Werte beschrieben, auf dessen Grundlage wir die Optimierungsansätze wie anschließend beschrieben umgesetzt haben. Um die Anzahl an Regresstransitionen minimieren zu können, haben wir zwei Möglichkeiten identifiziert: Die Formulierung als Mengenüberdeckungsproblem im Zusammenhang mit T-Invarianten und die bereits größtenteils in [Pru09] vorgestellte Formulierung als MAXFS-Instanz. Da uns der zweite Ansatz zur praktischen Umsetzung deutlich geeigneter erschien, haben wir diesen näher betrachtet. Insbesondere haben wir ihn im Vergleich zu [Pru09] zur praktischen Anwendbarkeit hin weiterentwickelt. Des Weiteren gaben wir eine Möglichkeit zur Beschleunigung der Berechnung an und haben eine Methode entwickelt, um mehrere Lösungen zu einer MAXFS- bzw. MIN IIS COVER-Instanz zu finden.

Im Anschluss wurde eine Heuristik zur Vermeidung von Ketten von Regresstransitionen vorgestellt, mit deren Hilfe aus mehreren Mengen von Regresstransitionen eine ausgewählt werden sollte, so dass möglichst wenige solcher Ketten bei der Zustandsraumdurchmusterung auftreten. Hierzu wurde das Konzept des Aktivierungsgraphen eingeführt und anschließend die vier Bewertungskriterien vorgestellt, aus denen sich die Heuristik zusammensetzt.

Nach der Beschreibung der Heuristiken sind wir auf die Implementierung eingegangen, mit deren Hilfe die anschließend präsentierte Fallstudie vorgenommen wurde. Hierbei haben wir festgestellt, dass die Heuristiken im Allgemeinen nicht zur gewünschten Speicherplatzeinsparung bei der Zustandsraumdurchmusterung führen. Dies hat uns veranlasst, beide Optimierungsansätze noch einmal einer kritischen Betrachtung zu unterziehen. Dabei haben wir zunächst gezeigt, dass nicht nur die Wahl der Regresstransitionen, sondern die Wahl der Offset-Werte für alle Transitionen eines Petrinetzes Einfluss auf den Speicherplatzbedarf bei der Zustandsraumdurchmusterung mit der Sweep-Line-Methode haben kann. Weiterhin haben wir betrachtet, warum eine geringe Anzahl an Regresstransitionen nicht immer zu einem verringerten Speicherplatzbedarf führen muss und warum die Vermeidung von Ketten von Regresstransitionen ein eher weniger geeignetes Kriterium zum Erreichen dieses Ziels ist.

## 6.2 Ausblick

Da die in dieser Arbeit betrachteten Optimierungsansätze im Allgemeinen nicht zu einer Reduzierung des Speicherplatzbedarfs bei der Zustandsraumdurchmusterung führen, stellt sich

## 6 Schlussbemerkungen

die Frage, wie diese erreicht werden kann.

Grundsätzlich wäre es wichtig, genau zu analysieren, wie sich die Sweep-Line-Methode in Kombination mit Stubborn Sets verhält, da die Kombination beider Methoden in vielen Fällen zu einer deutlich größeren Reduktion des Zustandsraumes führt als die alleinige Anwendung der Sweep-Line-Methode. Dies wirft weiterhin die Frage auf, ob die Optimierung ähnlich wie in [Sch99] für Stubborn Sets vom Ziel der Zustandsraumdurchmusterung (z.B. Suche nach Deadlocks, Prüfen der Erreichbarkeit einer bestimmten Markierung) abhängig gemacht werden sollte.

Um die Anzahl der persistenten Zustände zu reduzieren, muss abgeschätzt werden können, wie oft die gewählten Regresstransitionen im Verlauf der Zustandsraumdurchmusterung aktiviert werden und wieviele Zustände Ziel mehrerer Regresskanten sind. Hierzu müsste eine geeignete Heuristik gefunden werden.

Des Weiteren wurde in dieser Arbeit nicht geklärt, wie die Offset-Werte aller Transitionen eines Netzes zu einer gegebenen Menge von Regresstransitionen automatisch so berechnet werden können, dass die Anzahl der *peak*-Zustände bei der Zustandsraumdurchmusterung möglichst gering ist. Auch hierfür kann möglicherweise eine Heuristik entwickelt werden.

Insgesamt bleiben also noch einige Fragen zu klären, damit durch eine optimierte Berechnung der Offset-Werte eine signifikante Leistungssteigerung der Sweep-Line-Methode erzielt werden kann.

# Literaturverzeichnis

- [ABC08] AMALDI, Edoardo ; BRUGLIERI, Maurizio ; CASALE, Giuliano: A two-phase relaxation-based heuristic for the maximum feasible subsystem problem. In: *Comput. Oper. Res.* 35 (2008), Nr. 5, S. 1465–1482
- [ABH05] AMALDI, Edoardo ; BELOTTI, Pietro ; HAUSER, Raphael: *Randomized relaxation methods for the maximum feasible subsystem problem*. Jünger, Michael (Hrsg.) et al., Integer programming and combinatorial optimization. 11th international IP-CO conference, Berlin, Germany, June 8–10, 2005. Proceedings. Berlin: Springer. Lecture Notes in Computer Science 3509, S. 249-264, 2005
- [AK94] AMALDI, Edoardo ; KANN, Viggo: On the approximability of removing the smallest number of relations from linear systems to achieve feasibility. / Department of Mathematics, Swiss Federal Institute of Technology, Lausanne and Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm. 1994. – Forschungsbericht
- [AK95] AMALDI, Edoardo ; KANN, Viggo: The complexity and approximability of finding maximum feasible subsystems of linear relations. In: *Theor. Comput. Sci.* 147 (1995), Nr. 1-2, S. 181–210
- [BEN09] BERKELLAR, Michel ; EIKLAND, Kjell ; NOTEBAERT, Peter: *lp\_solve. Open source (Mixed-Integer) Linear Programming system, version 5.5.0.14*. <http://lpsolve.sourceforge.net>, 2009
- [BV04] *Kapitel 2*. In: BOYD, Stephen ; VANDENBERGHE, Lieven: *Convex Optimization*. Cambridge University Press, 2004, S. 27–29
- [CF04] CODATO, Gianni ; FISCHETTI, Matteo: *Combinatorial Benders' cuts*. Bienstock, Daniel (Hrsg.) et al., Integer programming and combinatorial optimization. 10th international IPCO conference, New York, NY, USA, June 7–11, 2004. Proceedings. Berlin: Springer. Lecture Notes in Computer Science 3064, S. 178-195, 2004
- [Cha94] CHAKRAVARTI, Nilotpal: Some results concerning post-infeasibility analysis. In: *Eur. J. Oper. Res.* 73 (1994), Nr. 1, S. 139–143
- [Chi96] CHINNECK, John W.: An effective polynomial-time heuristic for the minimum-cardinality IIS set-covering problem. In: *Ann. Math. Artif. Intell.* 17 (1996), Nr. 1-2, S. 127–144
- [Chi01] CHINNECK, John W.: Fast Heuristics for the Maximum Feasible Subsystem Problem. In: *INFORMS J. Comput.* 13 (2001), Nr. 3, S. 210–223

## Literaturverzeichnis

- [Chi08] *Kapitel 7.* In: CHINNECK, John W.: *Feasibility and Infeasibility in Optimization*. Springer Science+Business Media, LLC, 2008, S. 159–193
- [Chv79] CHVATAL, Vašek: A greedy heuristic for the set-covering problem. In: *Math. Oper. Res.* 4 (1979), Nr. 3, S. 233–235
- [CKM01] CHRISTENSEN, Søren ; KRISTENSEN, Lars M. ; MAILUND, Thomas: A Sweep-Line Method for State Space Exploration. In: MARGARIA, Tiziana (Hrsg.) ; YI, Wang (Hrsg.): *TACAS Bd. 2031*, Springer, 2001 (Lecture Notes in Computer Science). – ISBN 3–540–41865–2, S. 450–464
- [Gas03] *Kapitel 11.* In: GASS, Saul I.: *Linear Programming: Methods and Applications*. 5. Auflage. Dover Publications, Inc., Mineola, N.Y., 2003, S. 395
- [GM91] GREENBERG, Harvey J. ; MURPHY, Frederic H.: Approaches to diagnosing infeasible linear programs. In: *ORSA J. Comput.* 3 (1991), Nr. 3, S. 253–261
- [Kar72] KARP, Richard M.: Reducibility among combinatorial problems. In: MILLER, R.E. (Hrsg.) ; THATCHER, J.W. (Hrsg.): *Complexity of Computer Computations*, Plenum, New York, 1972, S. 83–103
- [KM02] KRISTENSEN, Lars M. ; MAILUND, Thomas: A Generalised Sweep-Line Method for Safety Properties. In: ERIKSSON, Lars-Henrik (Hrsg.) ; LINDSAY, Peter A. (Hrsg.): *FME Bd. 2391*, Springer, 2002 (Lecture Notes in Computer Science). – ISBN 3–540–43928–5, S. 549–567
- [KV08] *Kapitel 7.* In: KORTE, Bernhard ; VYGEN, Jens: *Combinatorial Optimization. Theory and Algorithms*. 4. Auflage. Springer-Verlag Berlin Heidelberg, 2008, S. 158
- [Mai03] MAILUND, Thomas: *Sweeping the State Space. A Sweep-Line State Space Exploration Method*, Department of Computer Science, University of Aarhus, Diss., 2003
- [May00] *Kapitel 2.* In: MAYER, Carl D.: *Matrix Analysis and Applied Linear Algebra*. Philadelphia : Society for Industrial and Applied Mathematics, 2000, S. 44–47
- [MWE02] MELLER, Jaroslaw ; WAGNER, Michael ; ELBER, Ron: Maximum feasibility guideline in the design and analysis of protein folding potentials. In: *Journal of Computational Chemistry* 23 (2002), Nr. 1, S. 111–118
- [Pfe08] PFETSCH, Marc E.: Branch-and-Cut for the Maximum Feasible Subsystem Problem. In: *SIAM J. Optim.* 19 (2008), Nr. 1, S. 21–38
- [PR96] PARKER, Mark ; RYAN, Jennifer: Finding the minimum weight IIS cover of an infeasible system of linear inequalities. In: *Ann. Math. Artif. Intell.* 17 (1996), Nr. 1-2, S. 107–126
- [Pru09] PRUEFER, Robert: *Optimierung der Sweeplinemethode*, Humboldt-Universität zu Berlin, Studienarbeit, März 2009
- [Rei91] REISIG, Wolfgang: *Petrinetze. Eine Einführung*. 2. Auflage. Springer-Verlag Berlin Heidelberg New York Tokyo, 1991

- [Sch99] SCHMIDT, Karsten: Stubborn Sets for Standard Properties. In: DONATELLI, Susanna (Hrsg.) ; KLEIJN, H. C. M. (Hrsg.): *ICATPN* Bd. 1639, Springer, 1999 (Lecture Notes in Computer Science). – ISBN 3–540–66132–8, S. 46–65
- [Sch00] SCHMIDT, Karsten: LoLA: A Low Level Analyser. In: MOGENS NIELSEN AND DAN SIMPSON (Hrsg.): *Application and Theory of Petri Nets, 21st International Conference (ICATPN 2000)*, Springer-Verlag, Juni 2000 (Lecture Notes in Computer Science 1825). – ISBN 3–540–67693–7, S. 465–474
- [Sch02] SCHMIDT, Karsten: *Explicit state space verification*, Humboldt-Universität zu Berlin, Diss., 2002
- [Sch04] SCHMIDT, Karsten: *Automated generation of a progress measure for the sweep-line method*. Jensen, Kurt (Hrsg.) et al., Tools and algorithms for the construction and analysis of systems. 10th international conference, TACAS 2004, held as part of the joint conferences on theory and practice of software, ETAPS 2004, Barcelona, Spain, March 29 – April 2, 2004. Proceedings. Berlin: Springer. Lecture Notes in Computer Science 2988, S. 192-204, 2004
- [SRK05] STAHL, Christian ; REISIG, Wolfgang ; KRSTIC, Milos: Hazard Detection in a GALS Wrapper: A Case Study. In: DESEL, Jörg (Hrsg.) ; WATANABE, Y. (Hrsg.): *Proceedings of the Fifth International Conference on Application of Concurrency to System Design (ACSD'05)*. St. Malo, France : IEEE Computer Society, Juni 2005, 234-243
- [Sta90] STARKE, Peter H.: *Analyse von Petri-Netz-Modellen*. B.G. Teubner Stuttgart, 1990
- [Val90] VALMARI, Antti: A Stubborn Attack On State Explosion. In: CLARKE, Edmund M. (Hrsg.) ; KURSHAN, Robert P. (Hrsg.): *CAV* Bd. 531, Springer, 1990 (Lecture Notes in Computer Science). – ISBN 3–540–54477–1, S. 156–165
- [Val96] VALMARI, Antti: The State Explosion Problem. In: REISIG, Wolfgang (Hrsg.) ; ROZENBERG, Grzegorz (Hrsg.): *Petri Nets* Bd. 1491, Springer, 1996 (Lecture Notes in Computer Science). – ISBN 3–540–65306–6, S. 501–520



### **Selbständigkeitserklärung**

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 6. April 2010

Unterschrift

### **Einverständniserklärung**

Ich erkläre hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Institutes für Informatik der Humboldt-Universität zu Berlin ausgestellt werden darf.

Berlin, den 6. April 2010

Unterschrift