Universität Stuttgart
Institut für Formale Methoden der Informatik
Abteilung Sichere und Zuverlässige Softwaresysteme
Fakultät Informatik, Elektrotechnik und Informationstechnik
Universitätsstr. 38, 70569 Stuttgart

Diplomarbeit No. 21/92

Verifikation dynamischer Systeme: Reguläre Ausdrücke zur Spezifikation verbotener Pfade

Nicolas Relange

Studiengang:	Informatik
Prüfer:	Prof. V. Diekert
Betreuerin:	Dr. B. König
begonnen am:	23. Januar 2004
beendet am:	19. September 2004
CR-Klassifikation:	F.3.1, F.3.2, F.4.2, G.2.2

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Grundlagen 1

1.1.1 Programmanalyse 1

1.1.2 Werkzeuge 2

1.1.3 Graphtransformations-Systeme 2

1.2 Definitionen 2

1.3 Beispiel 5

1.4 Ziele 5

2 Problembeschreibung **9**

2.1 Der Petri-Graph 9

2.1.1 Darstellung im Petri-Graphen 9

2.1.2 Markierung 9

2.1.3 Approximationen 11

2.2 Eigenschaften des Systems 13

2.2.1 Die Formel im Petri-Netz-Analysetool 13

2.2.2 Reguläre Ausdrücke im Graphen 13

2.2.3 Zustand des Systems 13

2.2.4 Erhaltung und Reflektion von Eigenschaften 14

2.2.5 Reguläre Ausdrücke in einem Hypergraphen 15

2.3 Darstellung des Anfangsproblems 18

2.4 Eine (unendliche) Formel 19

3 Lösungsansatz **21**

3.1 Die minimale Überdeckung 21

3.2 Zur Formel 25

3.2.1 Eine endliche Darstellung der Formel 25

3.2.2 Warum ist die endliche Formel relevant 26

4 Implementierung der Lösung **27**
4.1 Algorithmus 27
4.2 Korrektheit des Algorithmus 29

5 Anwendung an einem Beispiel **43**
5.1 Beispiel – Version 1 43
5.1.1 Ziel 43
5.1.2 `aunfold` 43
5.1.3 `sponge` 47
5.1.4 `bra` 47
5.1.5 Schlussbemerkung 48
5.2 Beispiel – Version 2 48
5.2.1 Überblick von `-depth` Option 48
5.2.2 `aunfold` 48
5.2.3 `sponge` 51
5.2.4 `bra` 51
5.2.5 Schlussbemerkung 51
5.3 Rot-Schwarz-Bäume 52
5.3.1 Definition 52
5.3.2 Rot-Schwarz-Bäume als GTS 52
5.3.3 Verifikation des Systems 53
5.3.4 Berechnung 53
5.3.5 Schlussbemerkung 55

Schlussbemerkungen **57**

A XML-Dateien (Auszüge) **61**
A.1 Öffentliche-Private Server 61
A.1.1 `pub_priv_server.xml` (Auszug) 61
A.1.2 `pub_priv_server2.gxl` (Auszug) 62
A.2 Rot-Schwarz-Bäume 66

B man page of sponge **69**

Literaturverzeichnis **73**
Index **75**

Kapitel 1

Einleitung

Das Internet wird immer größer und besteht immer mehr aus dynamischen Systemen. In diesen Systemen werden Objekte dynamisch erzeugt, bewegt und gelöscht. Da es auch einen unendlichen Zustandraum gibt, sind diese Systeme sehr komplex und schwierig zu analysieren. Trotzdem ist es sehr interessant zu wissen, ob ein dynamisches System eine besondere Eigenschaft besitzt. Zum Beispiel könnte sich die Frage stellen, ob ein System sicher ist.

In dieser Arbeit wird eine besondere Verifikation behandelt: die Regulären Ausdrücke zur Spezifikation verbotener Pfade. Behandelt werden nur die Systeme, die in einem Graphtransformations-System modelliert werden können.

1.1 Grundlagen

1.1.1 Programmanalyse

Es handelt sich hier um *Abstrakte Interpretation* [CC77][Cou96], eine Technik zur Approximation von Programmverhalten. Man will nicht wissen, ob das ganze System korrekt ist, sondern interessiert sich für eine bestimmte Eigenschaften des Systems. Diese Vorgehensweise führt zur *Programmanalyse und -Verifikation* [NNH99].

Anderseits nähert sich diese Arbeit auch dem *Model-Checking*. [CGL99] zeigt, wie das Konzept der Abstrakten Interpretation in Model-Checking benutzt werden kann.

Hier wird das Problem im folgendenen Rahmen[1] behandelt:

- das System wird durch ein Graphtransformations-System dargestellt
- die Eigenschaften werden mit regulären Ausdrücken beschrieben

Die Verifikation eines Systems besteht hier aus den folgenden Schritten:

1. Modellierung des Systems durch ein Graphtransformations-System (siehe Abschnitt 1.1.3 auf der nächsten Seite)

[1] Man findet auch in der Literatur [BKK03] den Fall, wo die Eigenschaften in der monadischen Logik zweite Stufe beschrieben werden

2. Approximation anhand eines Petri-Netzes, so dass es möglich ist, in vertretbarer Zeit Verifikationen zu berechnen [BCK01][BK02]

3. Verifikation auf die die gesuchte Eigenschaften hin

4. Verfeinerung der Approximation, falls die Anforderungen in Schritt 3 nicht gefüllt wurden

5. Überführen des Ergebnisses in das ursprüngliche System

Die Abbildung 1.1 auf der nächsten Seite zeigt diesen Prozess.

1.1.2 Werkzeuge

Vorhanden waren bereits folgende Werkzeuge zur Verifikation dynamischer Systeme:

aunfold Zur Approximation des Graphtransformations-Systems in einem Petri-Graphen

bra Zur Berechnung der Überdeckbarkeit in Petri-Netzen

Was fehlte, war ein Werkzeug, welches mit Hilfe eines regulären Ausdruckes zur Spezifikation verbotener Pfade, für das approximierte System Beziehungen auf Markierungen der Petri-Netz berechnen kann. Der reguläre Ausdruck wird in eine Formel auf Petri-Netz-Markierungen umwandelt. Diese gilt als erfüllt, wenn ein solchen Pfad existiert.

Die neue Anwendung – `sponge` – ist dann in der Programmanalyse zwischen `aunfold` und `bra` einzuordnen.

Die Abbildung 1.2 auf Seite 4 beschreibt, wie die verschiedene Komponenten untereinander kommunizieren.

Im Folgenden werden diese verschiedenen Strukturen beschrieben.

1.1.3 Graphtransformations-Systeme

Die meisten verteilten Systeme können durch Graphtransformations-Systeme dargestellt werden.

In einem Graphtransformations-System wird die Anfangskonfiguration zuerst durch einen Graphen beschrieben. Desweiteren gibt es Regeln, die alle möglichen Änderungen des Systems spezifizieren.

Dies führt zu den folgenden formalen Definitionen:

1.2 Definitionen

Definition 1 (Graph) Ein *Graph* G ist ein Tupel $(V_G, E_G, s_G, t_G, l_G)$, wobei V_G eine endliche Menge von Knoten, E_G eine endliche Menge von Kanten ist, $s_G, t_G : E_G \to V_G$ die Ausgangsfunktion und die Eingangsfunktion darstellen und $l_G : E_G \to \Lambda$ die Labellingfunktion für Kanten ist. □

Nach [BKK03] hat man auch die folgende Definition:

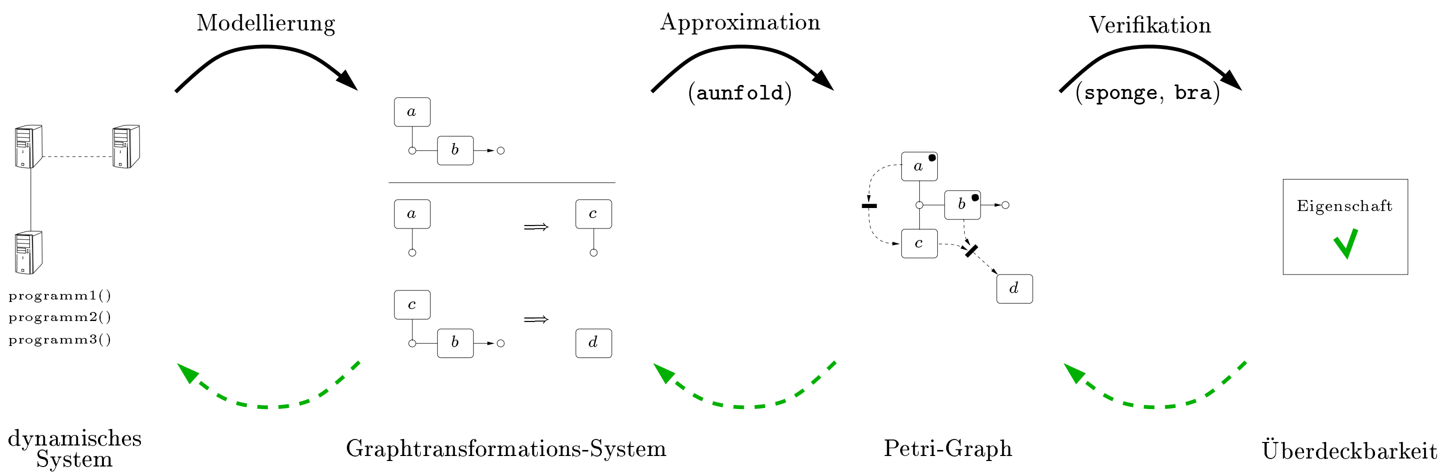


Abbildung 1.1: Programmanalyse

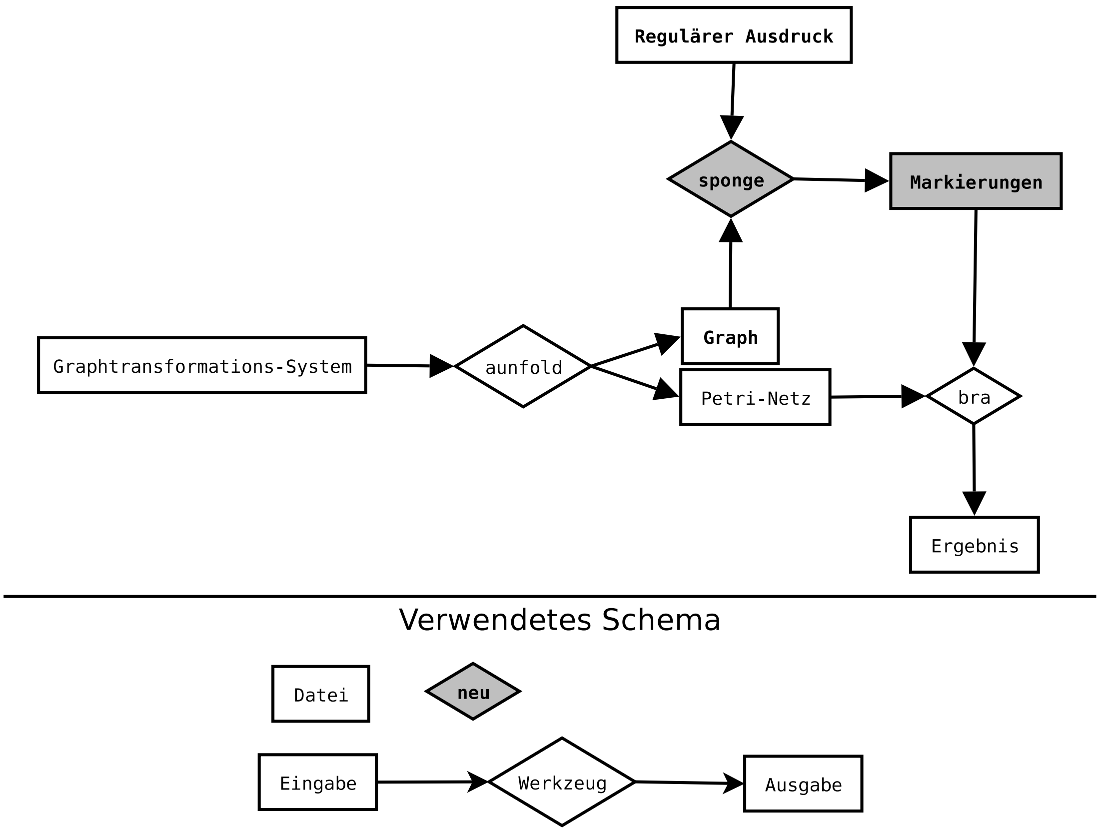


Abbildung 1.2: Programmanalyse und ihre Komponenten

Definition 2 (Graphmorphismus) Ein *Graphmorphismus* $\varphi : G_1 \to G_2$ ist ein Tupel von Abbildungen $\varphi_V : V_{G_1} \to V_{G_2}$ und $\varphi_E : E_{G_1} \to E_{G_2}$, wobei $\varphi_V \circ s_{G_1} = s_{G_2} \circ \varphi_E$, $\varphi_V \circ t_{G_1} = t_{G_2} \circ \varphi_E$ und $l_{G_1} = l_{G_2} \circ \varphi_E$ für alle $e \in E_{G_1}$ gilt.

Ein Graphmorphismus φ gilt als *kanten-bijektiv*, wenn φ_E eine Bijektion ist. □

Ein Hypergraph ist eine Verallgemeinerung eines Graphen. Er erlaubt die Nutzung von Hyperkanten, wobei mehrere Knoten (in einer bestimmten Ordnung) verbunden werden.

Die Definition ist wie folgt:

Definition 3 (Hypergraph) Ein *Hypergraph* G ist ein Tupel (V_G, E_G, c_G, l_G), wobei V_G eine endliche Menge von Knoten, E_G eine endliche Menge von Kanten, $c_G : E_G \to V_G^*$ die Verbindungsfunktion und Eingangsfunktion sind und $l_G : E_G \to \Lambda$ die Labellingfunktion für Kanten ist. □

Der einzige Unterschied zwischen dem Graphen und dem Hypergraphen ist die Verbindungsfunktion. Ein Graph ist ein Hypergraph, bei dem $c_G(e) = \big(s_G(e), t_G(e)\big)$ gilt.

Definition 4 (Graphtransformations-System) Ein *Graphtransformations-System* besteht aus einem Anfangsgraphen G_0 und einer Menge von Regeln $\mathcal{R}$. Eine Regel $r \in \mathcal{R}$ ist ein Tupel (L, R, α), wobei L und R Hypergraphen sind und $\alpha : V_L \to V_R$ eine injektive Funktion. □

Um eine Regel $r = (L, R, \alpha)$ in einem Hypergraphen G anzuwenden, wird zuerst der Teilgraph L im Hypergraphen gesucht. Dann werden die alten Kanten von L gelöscht und alle Knoten des Musters L V_L werden durch α ersetzt. Die neuen Knoten werden danach mit den Kanten aus R verbunden.

Das nächste Beispiel und Abbildung 1.3 auf der nächsten Seite erläutern diesen Prozess.

1.3 Beispiel

Um das Ganze zu veranschaulichen, folgendes Beispiel:

Das zu verifizierende System besteht aus privaten und öffentlichen Servern. Am Anfang gibt es nur einen privaten Server und einen Generator, der öffentliche Server herstellen kann. Ein privater Server kann auch zu einem öffentlichen Server werden.

Die Herstellung von Verbindungen zwischen den Servern ist auch mit der Einschränkung möglich, dass die Öffentlichen nicht die Privaten kennen.

Jeder Server stellt Prozesse her: die Privaten erstellen interne Prozesse und die Öffentlichen Externe. Die Prozesse können sich danach durch die Verbindungen bewegen.

Dieses System kann durch ein Graphtransformations-System beschrieben werden (siehe Abbildung 1.3 auf der nächsten Seite).

1.4 Ziele

Angestrebt wird, dass bestimmte Situationen nicht in dem Graphtransformations-System stattfinden. Diese Situationen werden durch reguläre Ausdrücke beschrieben, deren Alphabet aus den Labels der verschiedenen Kanten besteht.

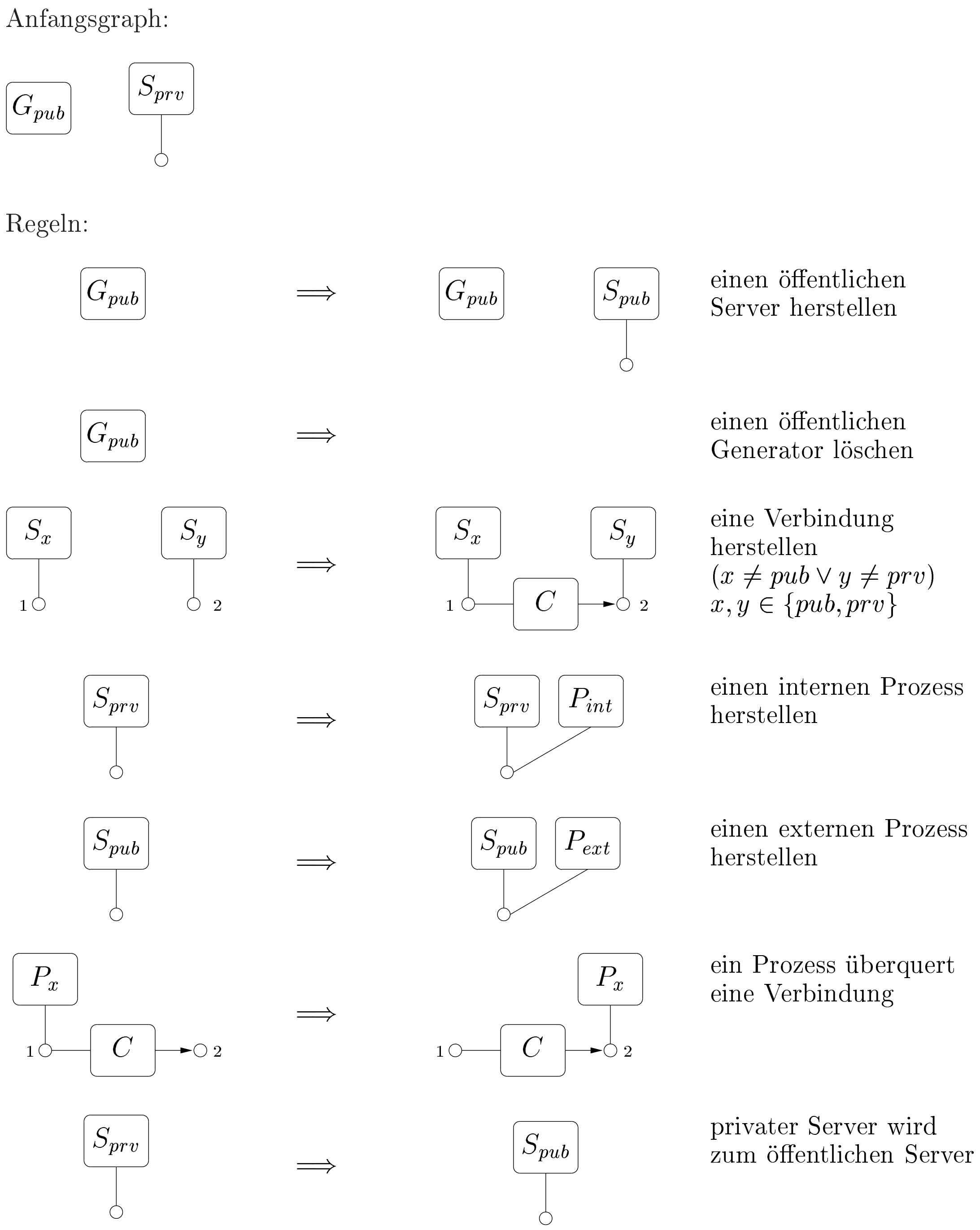


Abbildung 1.3: Grammatik des Beispiels *öffentlich-private Server*

Beispielsweise wird aus Sicherheitsgründen verlangt, dass sich ein öffentlicher Server nie mit einem privaten verbindet. Dieser verbotene Pfad kann durch den folgenden regulären Ausdruck gegeben werden:

$$S_{pub}C^*S_{prv}$$

wobei $\Sigma = \{S_{prv}, S_{pub}, C, P_{int}, P_{ext}, G_{pub}\}$ gilt.

Eine andere Konfiguration, die nicht erwünscht ist, wäre dass ein privater Server einen externen Prozess besitzt:

$$S_{prv}P_{ext}$$

Kapitel 2

Problembeschreibung

2.1 Das Graphtransformations-System approximiert durch einen Petri-Graphen

2.1.1 Darstellung im Petri-Graphen

Das Graphtransformations-System, das in Definition 4 definiert wurde, kann auch durch eine Kombination von einem Graphen, einem Petri-Netz (den Petri-Graphen) und einer initialen Markierung ausgedrückt werden.

Dies ist die approximierte Struktur, die von `aunfold` erzeugt wird.

Der Prozess, zum Aufbau des Petri-Graphen:

- zuerst werden alle Graphen, die das Graphtransformations-System bilden, zusammengesetzt
- der Anfangsgraph wird durch eine Markierung dargestellt
- das Verhalten der verschiedenen Regeln wird durch ein Petri-Netz gegeben

Die Abbildung 2.1 auf der nächsten Seite zeigt das Ergebnis der obenstehenden Prozesse für das Graphtransformations-System, das in Abschnitt 1.3 auf Seite 5 beschrieben wird.

2.1.2 Markierung

Die Markierung des Petri-Netzes hat auch eine Bedeutung in dem Graphen: sie ist wichtig, um das dynamische Verhalten des Systems zu darstellen.

Definition 5 (Markierung) Eine *Markierung* m über dem Graphen G ist eine Funktion, die zu allen Kanten eine ganze Zahl zuordnet. Dann: $m \in E_G^{\oplus}$ ($M^{\oplus}$ ist die Menge aller Multimengen über M, auch dargestellt durch Abbildung $M \to \mathbb{N}$). □

Beispiel 1 $m : e_1 \mapsto 2, e_2 \mapsto 1, e_3 \mapsto 0, e_4 \mapsto 3, e_5 \mapsto 1$

Siehe Abbildung 2.2 auf der nächsten Seite □

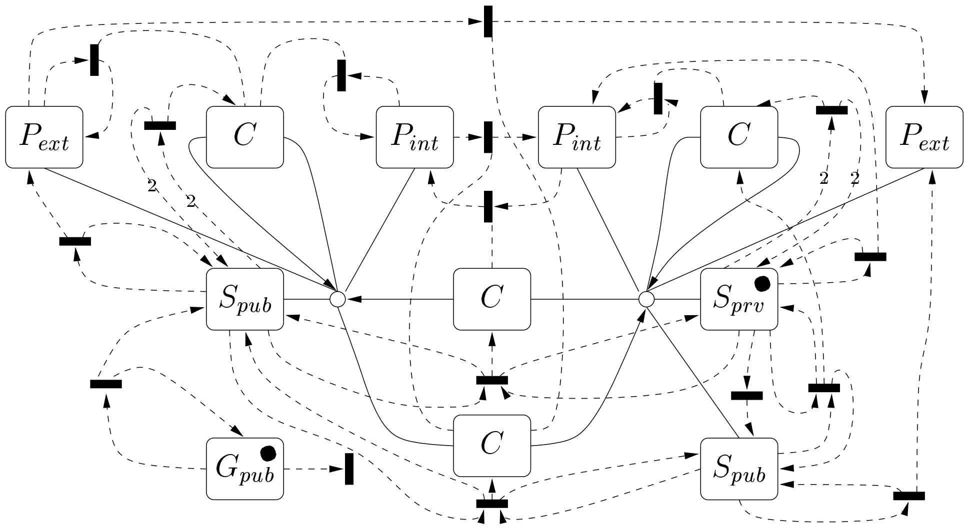
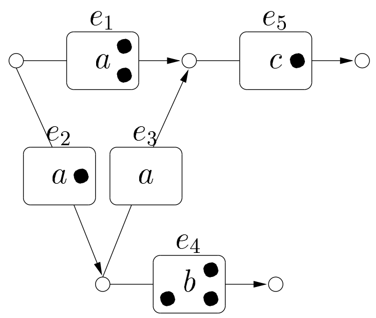


Abbildung 2.1: Der Petri-Graph



G

Abbildung 2.2: Ein Graph G mit Markierung

Die Markierungen stellen die Anzahl der Instanzen einer Kante dar. So existiert eine Kante ohne Markierung in der Realität nicht.

Nach [BKK03] kann man dieses Verhalten formal beschreiben:

Definition 6 ($graph_G$) Sei G ein Hypergraph und $m \in E_G^{\oplus}$ eine Markierung.

Der Graph $H = graph_G(m)$, der von m *erzeugt* wird, ist mit den folgenden Gleichungen definiert:

$$\begin{aligned} V_H &= \big\{v \in V_G \mid (\exists e \in m)\ v \in c_G(v)\big\} \\ E_H &= \big\{(e,i) \mid e \in m\ \wedge\ 1 \leq i \leq m(e)\big\} \\ c_H\big((e,i)\big) &= c_G(e) \\ l_H\big((e,i)\big) &= l_G(e) \end{aligned}$$

Abbildung 2.3 zeigt ein Beispiel. □

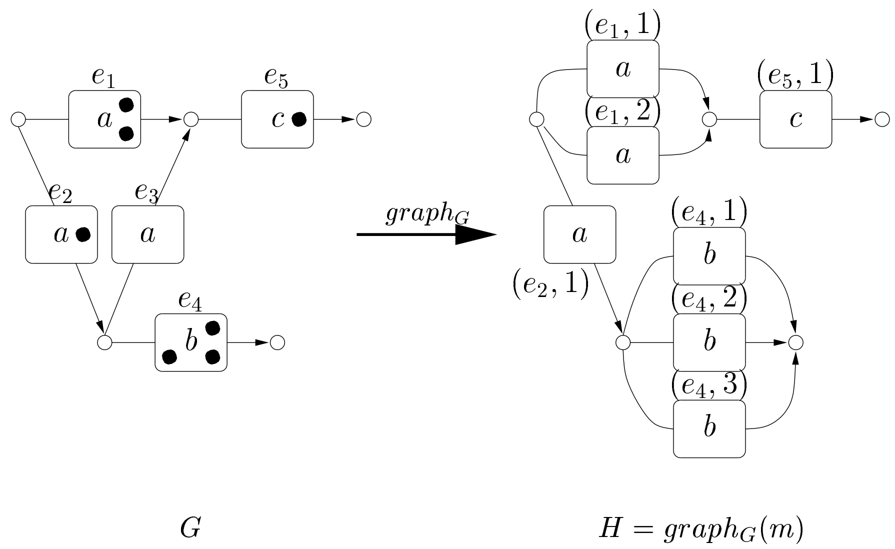


Abbildung 2.3: Bedeutung der Markierungen

2.1.3 Approximationen

$H = graph_G(m)$ ist nur ein möglicher Graph, der erzeugt werden kann. Eigentlich stellt H eine ganze Menge von Graphen dar.

Für jeden erreichbaren Graphen H' gibt es eine erreichbare Markierung m, sodass ein Graphmorphismus (siehe Definition 2) $\varphi : H' \to H$ existiert, wobei φ kanten-bijektiv ist.

Die Abbildung 2.4 auf der nächsten Seite zeigt einen der möglichen Graphen.

Während der Approximation in `aunfold` können nämlich zwei Knoten verschmelzen (siehe Abbildung 2.5 auf der nächsten Seite).

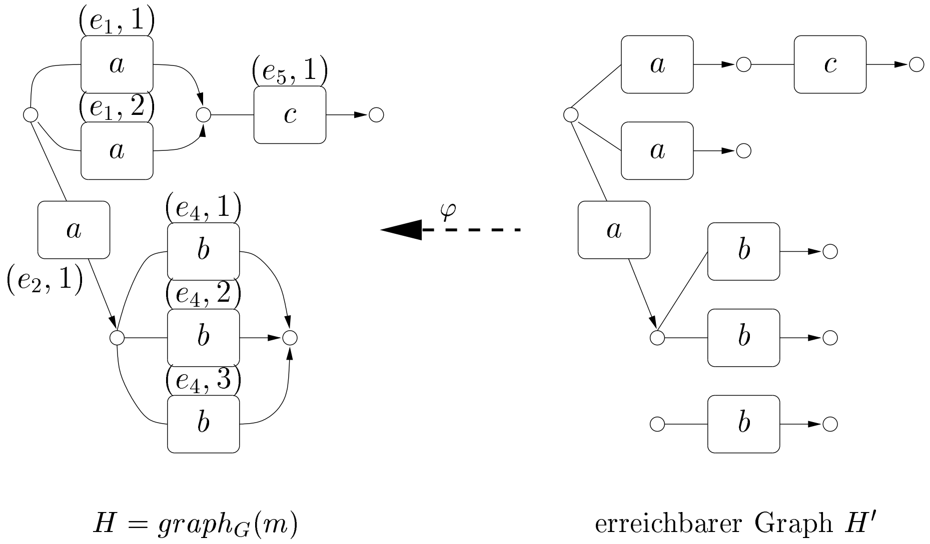


Abbildung 2.4: Erreichbarer Graph

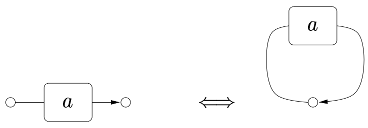


Abbildung 2.5: Zwei Knoten verschmelzen

2.2 Eigenschaften des Systems

2.2.1 Die Formel im Petri-Netz-Analysetool

Wenn man die Abbildung 1.2 auf Seite 4 beobachtet, bemerkt man, dass das Petri-Netz am Ende durch `bra` analysiert wird. `bra` ist eine Anwendung, die Markierungen in einem Petri-Netz auf ihre Überdeckbarkeit überprüft.

Das in dieser Arbeit entwickelte Werkzeug stellt daher Beziehungen auf Markierungen her, die für `bra` verständlich sind. Die Markierungen, die überprüft werden sollen, sind durch die Formel aus Definition 7 gegeben. Die Formel betrifft Markierungen von Petri-Netzen.

Definition 7 (Formel) Seien $e \in E$ und $c \in \mathbb{N}$.

Die Syntax der Formel M ist:

$$M ::= \#e \leq c \mid \neg M \mid M \vee M' \mid M \wedge M'$$

Man schreibt $m \models M$, wenn eine Markierung m die Formel M erfüllt.

$\#e$ stellt die Anzahl der Tokens in der Hyperkante e dar. Die Semantik von $\#e \leq c$ ist dann:

$$m \models (\#e \leq c) \Leftrightarrow m(e) \leq c$$

$\neg$, $\vee$ und $\wedge$ sind die übliche logische Negation, Disjunktion und Konjunktion. □

2.2.2 Reguläre Ausdrücke im Graphen

In diesem Abschnitt wird definiert, wann ein Graph ein Wort aus dem regulären Ausdruck enthält. Wir betrachten hier noch nicht die Dynamik des Graphtransformations-Systems.

Definition 8 ($\boldsymbol{Reg'(G,r)}$) $Reg'(G,r)$ ist wahr, wenn r als kanten-disjunkter Pfad[1] in G vorkommt.

$$\begin{aligned} Reg'(G,r) \quad \Leftrightarrow \quad & (\exists p = e_1 e_2 \dots e_n \in {E_G}^n) \\ & \Big((\forall i \in \mathbb{N}_{n-1})\ t_G(e_i) = s_G(e_{i+1}) \Big) \ \wedge \\ & \Big(\left(\forall (i,j) \in {\mathbb{N}_n}^2\right)\ e_i = e_j \Rightarrow i = j \Big) \ \wedge \\ & l_G(e_1) l_G(e_2) \dots l_G(e_n) \in L(r) \end{aligned}$$

mit der Konvention, dass $\mathbb{N}_n$ gleich $\{1, \dots n\}$ ist.

Die ersten zwei Zeilen beschreiben die Existenz eines Pfades. Die dritte Zeile beschreibt die Disjunktheit eines Pfades. In der letzten Zeile kommt r in diesem Pfad vor. □

2.2.3 Zustand des Systems zu einem bestimmten Zeitpunkt

Der Zustand des Systems (Graph und Petri-Netz) wird nach einer bestimmten Zeit durch seine Markierung gegeben (siehe 2.1.2 auf Seite 9).

Die folgende Definition bestimmt, ob es möglich ist, einen Pfad der diesem regulären Ausdruck entspricht, in einem Graphen mit seiner entsprechenden Markierung zu finden.

[1] ein Knoten darf durchaus mehrmals durchlaufen werden (analog zu Euler-Pfaden)

Definition 9 (Reg) $Reg(G, m, r)$ ist wahr, wenn $graph_G(m)$ einen Pfad enthält, der r entspricht.

$$\begin{array}{rcl} Reg(G,m,r) & \Leftrightarrow & (\exists p = e_1 e_2 \ldots e_n \in {E_G}^n) \\ & & (\forall i \in \mathbb{N}_{n-1})\ t_G(e_i) = s_G(e_{i+1}) \wedge \\ & & \#_{e_i} p \leq m(e_i) \wedge \\ & & l_G(e_1) l_G(e_2) \ldots l_G(e_n) \in L(r) \end{array}$$

□

Satz 1 *Es gilt sofort:*

$$Reg(G,m,r) \Leftrightarrow Reg'\big(graph_G(m), r\big)$$

Das heißt: $Reg(G,m,r) = false \Leftrightarrow graph_G(m)$ *enthält keinen Pfad, der dem regulären Ausdruck r entspricht.*

Beweis Mit der Definition 6 von $graph_G(m)$. ■

2.2.4 Erhaltung und Reflektion von Eigenschaften

Eine Eigenschaft wird von kanten-bijektiven Morphismen erhalten, genau dann, wenn für jeden kanten-bijektiven Morphismus $\phi : H \to H'$ gilt: falls H eine diese Eigenschaft hat (z.B. es gibt einen bestimmten Pfad), so hat auch H' die Eigenschaft.

Die Eigenschaft wird reflektiert, wenn aus der Tatsache, dass die Eigenschaft für H' gilt, auch folgt, dass die Eigenschaft für H gilt.

Zuerst hat man den folgenden Satz:

Satz 2 *Sei* $\varphi : H' \mapsto H$ *der kanten-bijektive Graphmorphismus vom Abschnitt 2.1.3 auf Seite 11, der auf einen erreichbaren Graphen* H' *den Graphen* $H = graph_G(m)$ *abbildet.*

Dann gilt:

$$Reg'(H', r) \Rightarrow Reg'(H, r)$$

oder:

$$\neg Reg'(H, r) \Rightarrow \neg Reg'(H', r)$$

Beweis Sei H' ein Graph und r ein regulärer Ausdruck, wobei H gleich $\varphi(H')$ ist und $Reg'(H', r)$ wahr ist.

Es gibt dann nach der Definition 8 von Reg' einen Pfad $p' = e'_1 e'_2 \ldots e'_n \in {E_{H'}}^n$, der die folgende Eigenschaften überprüft:

- $(\forall i \in \mathbb{N}_{n-1})\ t_{H'}(e'_i) = s_{H'}(e'_{i+1})$
- $\big(\forall (i,j) \in {\mathbb{N}_n}^2\big)\ e'_i = e'_j \Rightarrow i = j$
- $l_{H'}(e'_1) l_{H'}(e'_2) \ldots l_{H'}(e'_n) \in L(r)$

Seien die n Kanten $e_i = \varphi_E(e'_i) \in E_H$ für $i \in \mathbb{N}_n$.

Nach Definition 2 gilt für alle $i \in \mathbb{N}_{n-1}$:

$$\begin{aligned} t_H(e_i) &= (t_H \circ \varphi_E)\left(e'_i\right) \\ &= (\varphi_V \circ t_{H'})\left(e'_i\right) \\ &= (\varphi_V \circ s_{H'})\left(e'_{i+1}\right) \\ &= (s_H \circ \varphi_E)\left(e'_{i+1}\right) \\ &= s_H\left(e_{i+1}\right) \end{aligned}$$

Da φ kanten-bijektiv ist, gilt außerdem:

$$\left(\forall (i,j) \in {\mathbb{N}_n}^2\right)\; e_i = e_j \Rightarrow e'_i = e'_j \Rightarrow i = j$$

Zuletzt ergibt $l_{H'} = l_H \circ \varphi_e$ das folgende Ergebniss:

$$l_H(e_1)l_H(e_2)\ldots l_H(e_n) \in L(r)$$

Es gibt also einen Pfad $p = e_1e_2\ldots e_n \in {E_H}^n$, der die drei Eigenschaften von Definition 8 überprüft. Daraus folgt, dass $\mathit{Reg}'(H,r)$ auch wahr ist. ■

Es gilt dann:

$$\neg \mathit{Reg}(G,m,r) \Leftrightarrow \neg \mathit{Reg}'(\mathit{graph}_G(m),r) \Rightarrow \neg \mathit{Reg}'(H',r)$$

wobei $H = \mathit{graph}_G(m)$ und $H = \varphi(H')$.

Wenn man beweist, dass für alle erreichbaren Markierungen m $\neg \mathit{Reg}(G,m,r)$ gilt, enthalten alle erreichbaren Graphen keinen Pfad aus r.

Die Aufgabe besteht nun darin, diese Eigenschaft als Formel auf automatisch überprüfbaren Petri-Netz-Markierungen (siehe Definition 7) zu beschreiben.

2.2.5 Interpretation von regulären Ausdrücken in einem Hypergraphen

Nach Definition 3 auf Seite 5 des Hypergraphen machen $s_G, t_G : E_G \to V_G$ für Kanten der Kardinalität größer zwei keinen Sinn. Deshalb interpretiert man in der Praxis die Hyperkanten als eine Menge von normalen Kanten.

Dafür erstellt man eine Abbildung zwischen einem Hypergraphen und einem binären Graphen, welche im Folgenden erläutert wird.

Definition 10 (Hypergraph zu binären Graphen) Sei $G = (V_G, E_G, c_G, l_G)$ ein Hypergraph, der nach Definition 3 definiert wurde.

Man kann nun eine Abbildung ψ aufbauen, die den Hypergraphen G in einen binären Graphen $\psi(G) = G' = (V_{G'}, E_{G'}, s_{G'}, t_{G'}, l_{G'})$ (siehe Definition 1) umwandelt:

$$\begin{aligned} V_{G'} &= V_G \cup \left\{ v_e \mid e \in E_G \text{ und } c_G(e) = \epsilon \right\} \\ E_{G'} &= \left\{ e_{ij} \mid e \in E_G,\; c_G(e) = v_0v_1\ldots v_{n-1} \text{ mit } n \geq 1,\; (i,j) \in \{0,\ldots,n-1\}^2 \right\} \cup \\ &\quad \left\{ e \mid e \in E_G \text{ und } c_G(e) = \epsilon \right\} \end{aligned}$$

Für $c_G(e) = \epsilon$, gilt:

$$
\begin{aligned}
s_{G'}(e) &= v_e \\
t_{G'}(e) &= v_e \\
l_{G'}(e) &= 0\, l_G(e)\, 0
\end{aligned}
$$

Für $c_G(e) \neq \epsilon$, gilt:

$$
\begin{aligned}
s_{G'}(e_{ij}) &= v_i \\
t_{G'}(e_{ij}) &= v_j \\
l_{G'}(e_{ij}) &= i\, l_G(e)\, j
\end{aligned}
$$

Die Zahlen geben die Ordnung an, in der die Knoten gelesen werden sollen. □

Abbildung 2.6 zeigt die Funktionsweise des Morphismus ψ bei Kanten mit 0, 1 oder 2 Knoten.

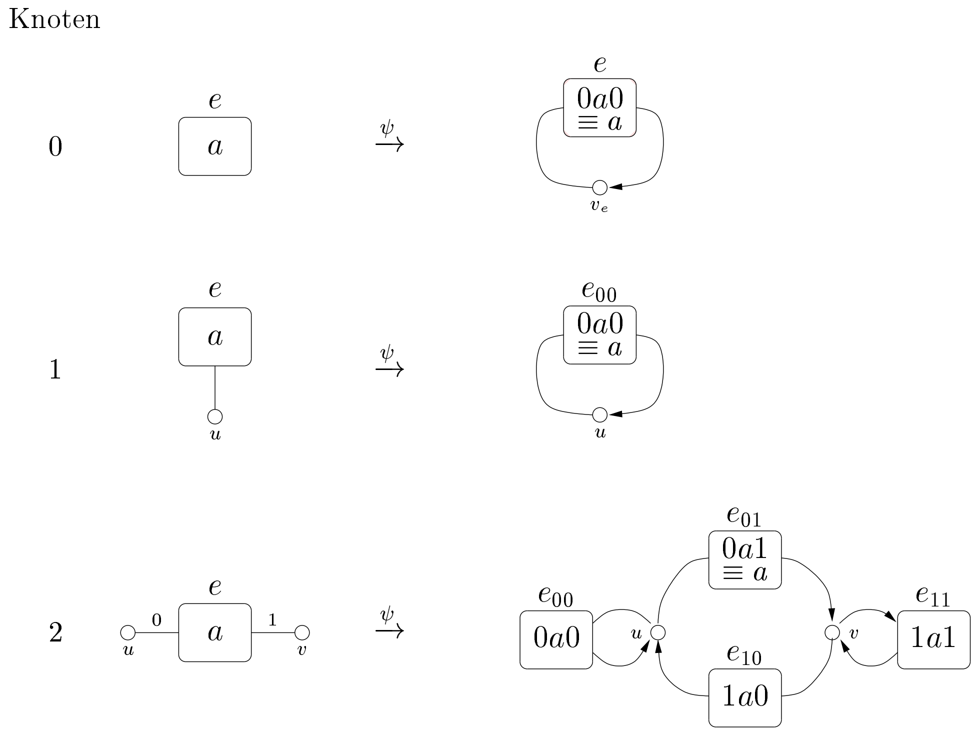


Abbildung 2.6: ψ: von Hypergraphen zu binären Graphen

Um die Syntax leichter zu machen, gibt es auch Konventionen, wenn keine Zahl für die Label gegeben wurde. Zum Beispiel reicht es für eine Kante mit nur einem Knoten a anstatt $0a0$ zu schreiben. Die Regeln ergeben sich aus der Anzahl der Knoten mit denen sie verbunden wurden:

0 Knoten: ihre Auffassung ist gleich der einer isolierten binären Kante: $a \equiv 0a0$.

1 Knoten: eine Hyperkante mit einem Knoten verhält sich wie eine binäre Kante, die den gleichen Knoten mit sich selbst verbindet: $a \equiv 0a0$.

2 Knoten: eine Hyperkante mit zwei Knoten ist eine binäre Kante: $a \equiv 0a1$.

$\geq$ **3 Knoten:** die Auffassung einer Hyperkante mit mehr als drei Knoten ist abhängig vom Werkzeug. Zwei Modi erlauben, ihre Bedeutung abzukürzen.

In *mode 0* ist die Hyperkante gleich einer Menge von binären Kanten, deren Anfangsknoten der Knoten 0 ist: $a \equiv 0ai$, wobei $i \geq 1$.

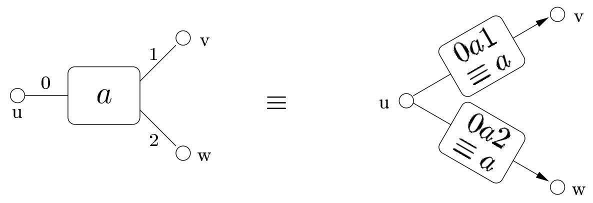


Abbildung 2.7: *mode 0* für Hyperkanten[1]

In *mode 1* ist die Hyperkante die Menge aller binären Kanten, deren Anfangsknoten eine kleinere Zahl als der Zielknoten hat: $a \equiv iaj$, wobei $i < j$.

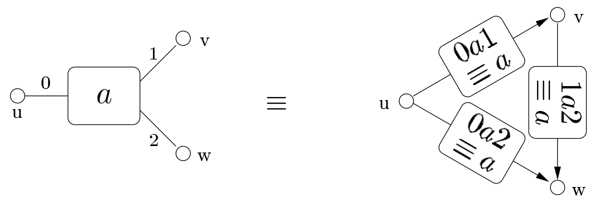


Abbildung 2.8: *mode 1* für Hyperkanten[1]

[1]die binäre Kanten, deren Label nicht a ist, werden nicht dargestellt

2.3 Darstellung des Anfangsproblems

Definition 11 Sei $P_G(r) \subseteq {E_G}^*$ die Menge aller Pfade, die ein Wort aus $L(r)$ enthalten. $P_G(r)$ wird dann mit den folgenden Gleichungen definiert:

$$\begin{aligned} P_G(r) &= \big\{p = e_1 e_2 \ldots e_n \in {E_G}^* \mid n \in \mathbb{N}, \\ &\quad (\forall i \in \mathbb{N}_n)\, e_i \in E_G, \\ &\quad (\forall i \in \mathbb{N}_{n-1})\, t_G(e_i) = s_G(e_{i+1}) \text{ und} \\ &\quad l_G(e_1) l_G(e_2) \ldots l_G(e_n) \in L(r)\big\} \end{aligned}$$

□

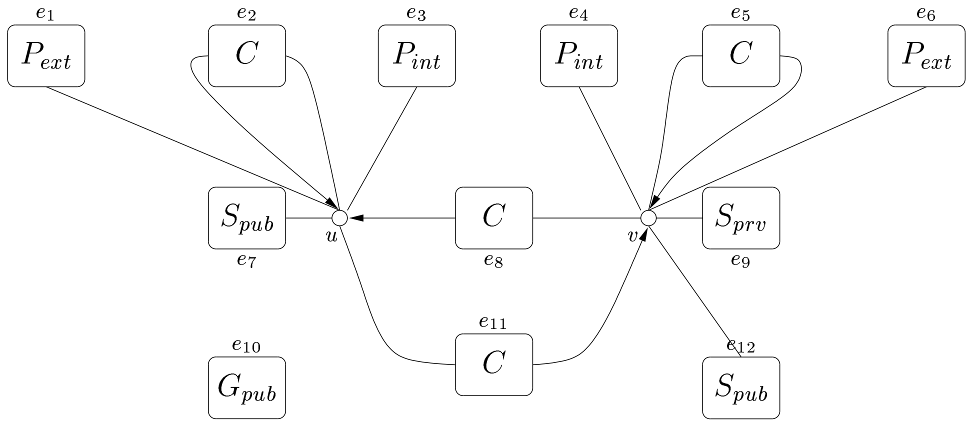


Abbildung 2.9: Hypergraph des Beispiels *öffentliche-private Server*

Das nächste Beispiel illustriert diese Definition.

Beispiel 2 Mit dem Hypergraphen des vorhergehenden Beispiels (siehe Abbildung 2.3) hat man folgende Ergebnisse:

$$\begin{aligned} P_G(\epsilon) &= \{\epsilon_{E_G}\} \\ P_G(C) &= \{e_2, e_5, e_8, e_{11}\} \\ P_G(C^2) &= \{e_2e_2, e_2e_{11}, e_5e_5, e_5e_8, e_8e_2, e_8e_{11}, e_{11}e_5, e_{11}e_8\} \\ P_G(P_{int}P_{ext}) &= \{e_3e_1, e_4e_6\} \\ P_G(P_{int} + P_{ext}) &= \{e_1, e_3, e_4, e_6\} \\ P_G(C^*) &= \{\epsilon_{e_G}, e_2, e_5, e_8, e_{11}, e_2e_2, e_2e_{11}, e_5e_5, e_5e_8, e_8e_2, e_8e_{11}, \\ &\quad e_{11}e_5, e_{11}e_8 \ldots\} \\ &= \bigcup_{i=0}^{\infty} P_G(C^i) \end{aligned}$$

Man bemerkt, dass $P_G(C^*)$ eine unendliche Menge ist. □

Definition 12 Sei $\Pi_G(r) \subseteq V^2 \times E^{\oplus}$ eine Menge, die die verschiedenen Pfade $P_G(r)$ charakterisiert. Dabei gilt:

$$\begin{aligned}\Pi_G(r) \quad = \quad & \big\{\pi = (u,v,\zeta) \mid (u,v,\zeta) \in V^2 \times E^{\oplus}, \\ & p = e_1 e_2 \dots e_n \in P_G(r), \\ & s_G(e_1) = u \text{ und } t_G(e_n) = v, \\ & (\forall e \in E_G)\, \zeta(e) = \#_e p\big\}\end{aligned}$$

□

Beispiel 3 Das gleiche Beispiel wird benutzt, um $\Pi_G(r)$ zu illustrieren.

Hier sind verschiedene Werte für $\Pi_G(r)$ gegeben, wobei $e_i \mapsto k_1$, $e_j \mapsto k_2$ den folgenden Sinn hat: es ist die Funktion $\zeta \in E^{\oplus}$, die $\zeta(e_i) = k_1$, $\zeta(e_j) = k_2$, $(\forall n)\; n \notin \{i,j\} \Rightarrow \zeta(e_n) = 0$ überprüft.

$$\begin{aligned}
\Pi_G(\epsilon) &= \big\{(u,u,0),(v,v,0)\big\} \\
\Pi_G(C) &= \big\{(u,u,[e_2 \mapsto 1]),(v,v,[e_5 \mapsto 1]), \\
&\quad (u,v,[e_{11} \mapsto 1]),(v,u,[e_8 \mapsto 1])\big\} \\
\Pi_G(C^2) &= \big\{(u,u,[e_2 \mapsto 2]),(u,u,[e_{11} \mapsto 1, e_8 \mapsto 1]), \\
&\quad (v,v,[e_5 \mapsto 2]),(v,v,[e_8 \mapsto 1, e_{11} \mapsto 1]), \\
&\quad (u,v,[e_2 \mapsto 1, e_{11} \mapsto 1]),(u,v,[e_{11} \mapsto 1, e_5 \mapsto 1]), \\
&\quad (v,u,[e_8 \mapsto 1, e_2 \mapsto 1]),(v,u,[e_5 \mapsto 1, e_8 \mapsto 1])\big\} \\
\Pi_G(P_{int}P_{ext}) &= \big\{(u,u,[e_3 \mapsto 1, e_1 \mapsto 1]),(v,v,[e_4 \mapsto 1, e_6 \mapsto 1])\big\} \\
\Pi_G(P_{int}+P_{ext}) &= \{(u,u,[e_1 \mapsto 1]),(u,u,[e_3 \mapsto 1]),(v,v,[e_4 \mapsto 1]),(v,v,[e_6 \mapsto 1])\} \\
\Pi_G(C^*) &= \big\{(u,u,0),(v,v,0), \\
&\quad (u,u,[e_2 \mapsto 1]),(v,v,[e_5 \mapsto 1]), \\
&\quad (u,v,[e_{11} \mapsto 1]),(v,u,[e_8 \mapsto 1]), \\
&\quad (u,u,[e_2 \mapsto 2]),(u,u,[e_{11} \mapsto 1, e_8 \mapsto 1]), \\
&\quad (v,v,[e_5 \mapsto 2]),(v,v,[e_8 \mapsto 1, e_{11} \mapsto 1]), \\
&\quad (u,v,[e_2 \mapsto 1, e_{11} \mapsto 1]),(u,v,[e_{11} \mapsto 1, e_5 \mapsto 1]), \\
&\quad (v,u,[e_8 \mapsto 1, e_2 \mapsto 1]),(v,u,[e_5 \mapsto 1, e_8 \mapsto 1]) \\
&\quad \dots\big\} \\
&= \bigcup_{i=0}^{\infty} \Pi_G(C^i)
\end{aligned}$$

$\Pi_G(C^*)$ ist hier noch unendlich. □

Das nächste Ziel ist eine endliche Darstellung für $\Pi_G(r)$.

2.4 Eine (unendliche) Formel

Definition 13 ((unendliche) Formel) Sei die folgende Formel gegeben:

$$F_{\Pi_G} = \bigvee_{\pi=(u,v,\zeta)\in\Pi_G(r)} \left(\bigwedge_{e\in E} (\#e \geq \zeta(e)) \right)$$

□

Satz 3 *Sei G ein Graph und r ein regulärer Ausdruck r.*
Dann gilt für alle Markierungen m:

$$Reg(G, m, r) \Leftrightarrow m \models F_{\Pi_G}$$

Beweis Der Beweis besteht aus zwei Teilen:

$\boxed{\Rightarrow}$ Sei m eine Markierung über den Graphen G.

Nach den Definitionen von *Reg* (siehe Definition 9) und $P_G(r)$ (siehe Definition 11) gilt:

$$\begin{aligned} Reg(G, m, r) \quad \Leftrightarrow \quad & (\exists p = e_1 e_2 \dots e_n \in P_G(r)) \\ & (\forall i \in \mathbb{N}_n)\ \#_{e_i} p \leq m(e_i) \end{aligned}$$

Man nimmt an, dass $Reg(G, m, r)$ wahr ist.

Dann gibt es einen Pfad $p = e_1 e_2 \dots e_n \in P_G(r)$, wobei für alle $i \in \mathbb{N}_n$ gilt: $\#_{e_i} p \leq m(e_i)$

Sei $\pi = (u, v, \zeta)$ der Wert, der zu p in der Definition von $\Pi_G(r)$ gehört: $s_G(e_i) = u$, $t_G(e_n) = v$ und $(\forall i \in \mathbb{N}_n)\ \zeta(e_i) = \#_{e_i} p$.

Die Markierung m erfüllt zunächst die folgende Formel:

$$m \models \bigwedge_{e \in E_G} \left(\#e \geq \zeta(e)\right)$$

Außerdem gilt:

$$m \models \bigvee_{\pi = (u,v,\zeta) \in \Pi_G(r)} \left(\bigwedge_{e \in E_G} (\#e \geq \zeta(e)) \right)$$

$\boxed{\Leftarrow}$ Sei m eine Markierung, die die Formel F_{Π_G} erfüllt: $m \models F_{\Pi_G}$.

Es gilt dann $\pi = (u, v, \zeta) \in \Pi_G(r)$, wobei $(\forall e \in E_G)\ m(e) \geq \zeta(e)$.

Nach der Definition 12 von $\Pi_G(r)$ gibt es $p = e_1 e_2 \dots e_n \in P_G(r)$, wobei für alle $e \in E_G$ $\zeta(e) = \#_e p$ gilt. Dann ist $(\forall e \in E_G)\ \#_e p \leq m(e)$.

Die Definition 11 von $P_G(r)$ ergibt zunächst: $(\forall i \in \mathbb{N}_{n-1})\ t_G(e_i) = s_G(e_{i+1})$ und $l_G(e_1) l_G(e_2) \dots l_G(e_n) \in L(r)$.

Daraus folgt, dass $Reg(G, m, r)$ wahr ist.

■

Kapitel 3

Lösungsansatz

In diesem Kapitel wird die vorherige unendliche Formel in eine endliche Formel umgewandelt. Die neue Formel soll mit Hilfe eines Algorithmus erzeugt werden.

Dafür behält man nur jene Pfade von der vorherigen Formel, die relevant sind. Diese Pfade bilden dann eine minimale Überdeckung mit allen Pfaden, die dem regulären Ausdruck entsprechen.

3.1 Die minimale Überdeckung

Es lässt sich eine partielle Ordnung definieren:

Definition 14 ($\preceq$) Seien $(\pi_1, \pi_2) \in (V^2 \times E^{\oplus})^2$, wobei:

$$\begin{cases} \pi_1 & = & (u_1, v_1, \zeta_1) \\ \pi_2 & = & (u_2, v_2, \zeta_2) \end{cases}$$

Dann gilt:

$$\pi_1 \preceq \pi_2 \quad \Leftrightarrow \quad \begin{cases} u_1 = u_1 \\ v_1 = v_2 \\ (\forall e \in E)\ \zeta_1(e) \leq \zeta_2(e) \end{cases}$$

□

Nun beweist man, dass $(V^2 \times E^{\oplus}, \preceq)$ eigentlich eine partielle Ordnung ist.

Satz 4 *$\preceq$ ist eine partielle Ordnung auf $V^2 \times E^{\oplus}$.*

Beweis Man beweist, dass $\preceq$ reflexiv, transitiv und antisymmetrisch ist.

reflexiv: Für alle $\pi \in V^2 \times E^{\oplus}$ gilt: $\pi \preceq \pi$

transitiv: Nach Definition 14 erhält man einfach $\pi_1 \preceq \pi_3$, wenn $\pi_1 \preceq \pi_2$ und $\pi_2 \preceq \pi_3$ gilt.

$$\left.\begin{matrix} \pi_1 \preceq \pi_2 \\ \pi_2 \preceq \pi_3 \end{matrix}\right\} \Rightarrow \pi_1 \preceq \pi_3$$

antisymmetrisch: Es liegt Antisymmetrie vor, wenn $\pi_1 \preceq \pi_2$ und $\pi_2 \preceq \pi_1$. Dann gilt $\pi_1 = \pi_2$.

■

Weiterhin gibt es auch eine partielle Ordnung für die Mengen in $V^2 \times E^{\oplus}$.

Definition 15 ($\sqsubseteq$) Seien $(\Pi_1, \Pi_2) \subseteq V^2 \times E^{\oplus}$. $\Pi_1 \sqsubseteq \Pi_2$ stehen in Relation, falls Π_2 eine Obermenge von Π_1 und wenn es für alle Elemente von Π_2 ein Element in Π_1 gibt, das kleiner ist.

$$\Pi_1 \sqsubseteq \Pi_2 \quad \Leftrightarrow \quad \begin{cases} \Pi_1 \subseteq \Pi_2 \\ (\forall \pi_2 \in \Pi_2)(\exists \pi_1 \in \Pi_1)\ \pi_1 \preceq \pi_2 \end{cases}$$

□

Satz 5 *$\sqsubseteq$ ist eine partielle Ordnung auf $\mathcal{P}(V^2 \times E^{\oplus})$.*

Beweis Es reicht, die Reflexivität, die Transitivität und die Antisymmetrie zu beweisen.

reflexiv Sei $\pi \in \Pi$. Da $\pi \preceq \pi$, gilt $\Pi \sqsubseteq \Pi$

transitiv Seien $(\Pi_1, \Pi_2, \Pi_3) \in \mathcal{P}(V^2 \times E^{\oplus})^3$ gegeben, wobei es $\Pi_1 \sqsubseteq \Pi_2$ und $\Pi_2 \sqsubseteq \Pi_3$ gilt.

- $\Pi_1 \subseteq \Pi_2$ und $\Pi_2 \subseteq \Pi_3$ ergibt $\Pi_1 \subseteq \Pi_3$
- Sei $\pi_3 \in \Pi_3$. Es gibt $\pi_2 \in \Pi_2$ mit $\pi_2 \preceq \pi_3$. Dann gilt auch $\pi_1 \in \Pi_1$ mit $\pi_1 \preceq \pi_2$. Daraus folgt: es gilt $\pi_1 \in \Pi_1$ mit $\pi_1 \preceq \pi_3$

Daraus folgt: $\Pi_1 \sqsubseteq \Pi_3$.

antisymmetrisch Seien $(\Pi_1, \Pi_2) \in \mathcal{P}(V^2 \times E^{\oplus})^2$ gegeben, wobei $\Pi_1 \sqsubseteq \Pi_2$ und $\Pi_2 \sqsubseteq \Pi_1$.

$\Pi_1 \subseteq \Pi_2$ und $\Pi_2 \subseteq \Pi_1$ ergibt $\Pi_1 = \Pi_2$

■

Definition 16 (μ) Bei einer Menge in $\mathcal{P}(V^2 \times E^{\oplus})$ gibt μ die minimalen Elemente zurück.

$$\begin{array}{lll} \mu: & \mathcal{P}(V^2 \times E^{\oplus}) & \to \mathcal{P}(V^2 \times E^{\oplus}) \\ & \Pi & \mapsto \left\{\pi \in \Pi \mid (\forall \pi' \in \Pi)\ \pi' \preceq \pi \Rightarrow \pi' = \pi\right\} \\ & & \mapsto \left\{\pi \in \Pi \mid (\nexists \pi' \in \Pi)\ \pi' \prec \pi\right\} \end{array}$$

Die Abbildung 3.1 auf der nächsten Seite beschreibt das Verhalten der μ-Funktion. □

Definition 17 (Minimale Überdeckung) f_G stellt die Minimale Überdeckung dar, die von einem regulären Ausdruck erzeugt wird.

$$\begin{array}{lll} f_G: & reg(\Lambda) & \to \mathcal{P}(V^2 \times E^{\oplus}) \\ & r & \mapsto \mu\big(\Pi_G(r)\big) \end{array}$$

□

Illustrieren wir nun die minimale Überdeckung mit dem Beispiel von privaten-öffentlichen Server (siehe 1.3 auf Seite 6 und 2.1 auf Seite 10).

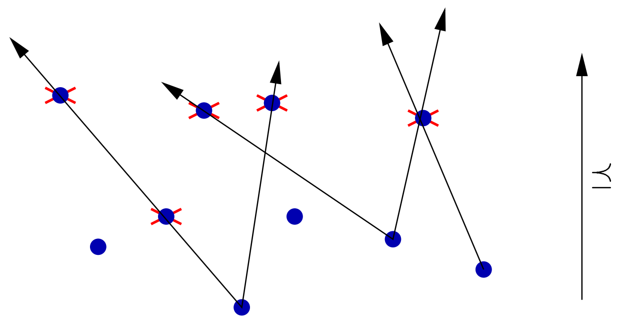


Abbildung 3.1: Die μ-Funktion

Beispiel 4 Nach den Ergebnissen aus dem Beispiel 3 hat man:

$$\begin{aligned}
f_G(\epsilon) &= \mu\big(\Pi_G(\epsilon)\big) \\
&= \big\{(u,u,0),(v,v,0)\big\} \\
f_G(C) &= \mu\big(\Pi_G(C)\big) \\
&= \big\{(u,u,e_2 \mapsto 1),(v,v,e_5 \mapsto 1), \\
&\qquad (u,v,e_{11} \mapsto 1),(v,u,e_8 \mapsto 1)\big\} \\
f_G(C^2) &= \mu\big(\Pi_G(C^2)\big) \\
&= \big\{(u,u,e_2 \mapsto 2),(u,u,[e_{11} \mapsto 1, e_8 \mapsto 1]), \\
&\qquad (v,v,e_5 \mapsto 2),(v,v,[e_8 \mapsto 1, e_{11} \mapsto 1]), \\
&\qquad (u,v,[e_2 \mapsto 1, e_{11} \mapsto 1]),(u,v,[e_{11} \mapsto 1, e_5 \mapsto 1]), \\
&\qquad (v,u,[e_8 \mapsto 1, e_2 \mapsto 1]),(v,u,[e_5 \mapsto 1, e_8 \mapsto 1])\big\} \\
f_G(C^*) &= \mu\big(\Pi_G(C^*)\big) \\
&= \big\{(u,u,0),(v,v,0), \\
&\qquad (u,v,e_{11} \mapsto 1) \\
&\qquad (v,u,e_8 \mapsto 1)\big\}
\end{aligned}$$

Die Abbildung 3.2 auf der nächsten Seite zeigt, wie die Menge $f_G(C^*)$ aufgebaut wird:

1. $\Pi_G(\epsilon)$ wird aufgebaut.
2. $\Pi_G(C)$ wird aufgebaut, aber nur die Elemente, welche nicht größer als die Elemente sind, die man schon hat (hier $\Pi_G(\epsilon)$), werden behalten.
3. Alle Elemente aus $\Pi_G(C^2)$ sind größer als Elemente aus $\Pi_G(\epsilon)$ und $\Pi_G(C)$. Da $\Pi_G(C^2)$ keine neue Elemente ergibt, hört der Prozess hier auf.

□

Lemma 1 *Alle Mengen von unvergleichbaren Elementen aus $V^2 \times E^{\oplus}$ sind endlich.*

Beweis Der Beweis ist in [Rei85], Seite 70. ■

Satz 6 *Für jeden regulären Ausdruck r ist $f_G(r)$ endlich.*

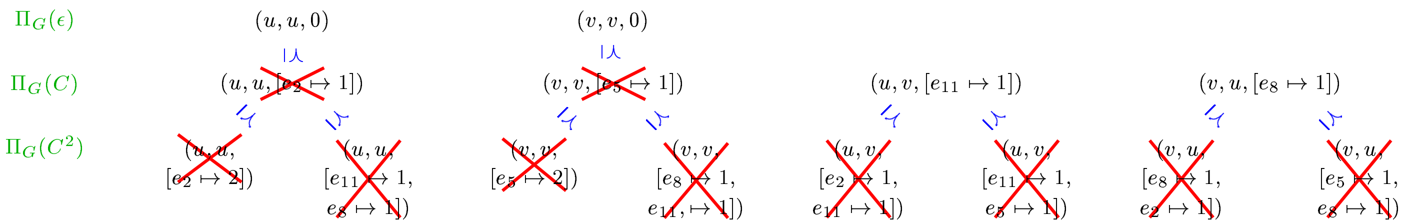


Abbildung 3.2: Aufbau von $f_G(C^*)$

BEWEIS Sei r ein regulärer Ausdruck über Λ. Nach der Definition 16 auf Seite 22 von μ ist $f_G(r) = \mu\big(\Pi_G(r)\big)$ die Menge aller minimalen Elemente von $\Pi_G(r)$. Die minimalen Elemente sind unvergleichbar, dann aus Lemma 1 folgt, dass $f_G(r)$ endlich ist. ■

3.2 Zur Formel

3.2.1 Eine endliche Darstellung der Formel

Die Vereinfachung nur die minimale Überdeckungen zu betrachten ergibt eine endliche Formel im Gegensatz zu der Formel in 2.2.1 auf Seite 13.

Die Formel wird zuerst definiert, anschließend beweist man, dass die zwei Formeln die gleiche Semantik haben.

Definition 18 Sei die folgende Formel:

$$F_{f_G} = \bigvee_{\pi=(u,v,\zeta)\in f_G(r)} \left(\bigwedge_{e\in E_G} (\#e \geq \zeta(e)) \right)$$

□

Satz 7 *Die Formeln 1 und 2, die in Definition 13 und Definition 18 definiert wurden, sind gleich:*

$$(\forall m \in {E_G}^{\oplus})\ m \models F_{\Pi_G} \Leftrightarrow m \models F_{f_G}$$

BEWEIS Sei G ein Graph und r ein regulärer Ausdruck.

Der Beweis besteht aus zwei Teilen:

$\boxed{\Rightarrow}$ Sei m eine Markierung über dem Graphen G, wobei F_{Pi_G} wahr ist.

Dann gibt es ein $\pi = (u, v, \zeta) \in \Pi_G(r)$, so dass $\bigwedge_{e\in E_G}\big(\#e \geq \zeta(e)\big)$ wahr ist.

Nun ist $\pi' = (u, v, \zeta') \in f_G(r)$ kleiner als π, denn für alle $e \in E_G$ gilt $\zeta'(e) \leq \zeta(e)$. Es folgt, dass $\bigwedge_{e\in E_G}\big(\#e \geq \zeta'(e)\big)$ wahr ist.

Dann ist auch F_{f_G} wahr.

$\boxed{\Leftarrow}$ Da $f_G(r) \subseteq \Pi_G(r)$ gilt, sieht man sofort, dass $F_{f_G} \Rightarrow F_{Pi_G}$.

■

Korollar 1 *Aus den Sätzen 7 und 3 auf Seite 20 folgt:*

$$Reg(G, m, r) \Leftrightarrow (m \models F_{f_G})$$

□

3.2.2 Warum ist die endliche Formel relevant

Einige Fälle, die durch einen regulären Ausdruck beschreiben werden, sollen nie eintreten. Es reicht dann, die Konfiguration r in F_{f_G} zu testen. Wenn F_{f_G} falsch für alle mögliche Markierungen ist, ist auch $Reg(G, m, r)$ für alle Markierungen falsch:

$$(m \nvDash F_{f_G}) \Rightarrow \neg Reg(G, m, r)$$

Kapitel 4

Implementierung der Lösung

4.1 Algorithmus

Um zu beschreiben, was durch den Algorithmus berechnet wird, braucht man zuerst ein paar Definitionen:

Definition 19 ($A_G(r)$) Sei $A_G : reg(\Lambda) \mapsto \mathcal{P}(V^2 \times E^{\oplus})$ die Funktion, die rekursiv mit den folgenden Regeln definiert wird:

$$\begin{array}{rcl}
A_G(\epsilon) & = & \big\{(v, v, 0) \mid v \in V_G\big\} \\
A_G(a \in \Lambda) & = & \big\{(u, v, \zeta) \mid (\exists e \in E_G)\ s_G(e) = u,\, t_G(e) = v,\, l_G(e) = a,\, \zeta(e) = 1 \\
 & & \qquad\qquad \textit{und} \quad (\forall e' \in E_G)\ e' \neq e \Rightarrow \zeta(e') = 0\big\} \\
A_G(r_1 + r_2) & = & \mu\big(A_G(r_1) \cup A_G(r_2)\big) \\
A_G(r_1 r_2) & = & \mu\Big(\big\{(u, w, \zeta_3) \mid (\exists v \in V_G)\ (u, v, \zeta_1) \in A_G(r_1),\, (v, w, \zeta_2) \in A_G(r_2) \\
 & & \qquad\qquad \textit{und} \quad \zeta_3 = \zeta_1 + \zeta_2 : e \mapsto \zeta_1(e) + \zeta_2(e)\big\}\Big) \\
A_G(r^*) & = & A_n \quad \text{wobei:} \\
 & & \quad \star \quad n = \min\{p \in \mathbb{N} \mid A_p = A_{p+1}\} \text{ gilt} \\
 & & \quad \star \quad \text{die Folge } (A_n)_{n \in \mathbb{N}} \text{ wie unten in Definition 20 definiert wird}
\end{array}$$

□

Definition 20 (A_n) Sei $(A_n)_{n\in\mathbb{N}}$ die Folge, die durch:

$$\begin{cases} A_0 = A_G(\epsilon) = A_G(r^0) \\ A_{n+1} = \mu\big[A_n \cup A_G(r^{n+1})\big] \end{cases}$$

definiert wird. □

Diese Definitionen führen zu dem Algorithmus 1 auf der nächsten Seite, der $A_G(r)$ berechnet. Die Variable X berechnet einfach die Werte der Folge $(A_n)_{n\in\mathbb{N}}$.

Daher reicht es zu zeigen, dass für jeden regulären Ausdruck r über Λ $A_G(r) = f_G(r)$ gilt. Dies wird später durch Induktion über r bewiesen.

Algorithm 1 Berechnung von $A_G(r)$

Require: $r \in reg(\Lambda)$
Ensure: $X = A_G(r)$

```
if r = ε then                                              /* leeres Wort */
    X ← {(v, v, 0) | v ∈ V_G}
else if r ∈ Λ then                                         /* Buchstabe */
    X ← {(u, v, ζ) | (∃e ∈ E_G) s_G(e) = u, t_G(e) = v, l_G(e) = r, ζ(e) = 1 and (∀e' ∈ E_G) e' ≠
        e ⇒ ζ(e') = 0}
else if r = r_1 + r_2 then                                 /* Summe */
    X ← μ(A_G(r_1) ∪ A_G(r_2))
else if r = r_1 r_2 then                                   /* Konkatenation */
    Y ← {(u, w, ζ_3) | (∃v ∈ V_G) (u, v, ζ_1) ∈ A_G(r_1), (v, w, ζ_2) ∈ A_G(r_2) and ζ_3 = ζ_1 + ζ_2 : e ↦
        ζ_1(e) + ζ_2(e)}
    X ← μ(Y)
else if r = r_1^* then                                     /* Kleene'sche Hülle */
    X ← A_G(ε)                                             /* X ← A_0 */
    Y ← ∅
    i ← 1
    while X ≠ Y do                                         /* solange, dass A_p ≠ A_{p+1} gilt, fortsetzen */
        Y ← X
        X ← μ(X ∪ A_G(r^i))                                /* X berechnet (A_n)_{n∈ℕ} */
        i ← i + 1
    end while                                              /* X = Y, denn n = p und X = A_n = A_G(r^*) */
else                                                       /* Unmöglich */
                                                           /* nichts */
end if
```

4.2 Korrektheit des Algorithmus

Die Aufgabe ist nun zu beweisen, dass der Algorithmus 1 auf der vorherigen Seite das richtige Ergebnis liefert und terminiert.

Am Anfang werden die zwei einfachsten Fälle $r = \epsilon$ und $r = l \in \Lambda$ behandelt. Dann wird ein regulärer Ausdruck r als eine Kombination von zwei regulären Ausdrücken $(r_1, r_2) \in reg(\Lambda) \times reg(\Lambda)$ mit einer der folgenden Operatoren: Vereinigung, Konkatenation oder Kleene'sche Hülle behandelt.

Man beweist zuerst die Richtigkeit der Zeilen 1 und 2 ($r = \epsilon$).

Satz 8 ($\boldsymbol{f_G(\epsilon)}$) *Es gilt:*

$$f_G(\epsilon) = \big\{(v, v, 0) \mid v \in V_G\big\}$$

Beweis Da $\Pi_G(\epsilon)$ gleich $\big\{(v, v, 0) \mid v \in V_G\big\}$ ist und da μ die Menge der minimale Elementen charakterisiert, gilt $\Pi_G(\epsilon) = \mu\big(\Pi_G(\epsilon)\big) = f_G(\epsilon)$. Es gilt nämlich: $\big(\forall(\pi, \pi') \in \Pi_G(\epsilon)^2\big)\ \pi' \preceq \pi \Rightarrow v' = v \Rightarrow \pi' = \pi$. ■

Korollar 2 *Nach dem vorherigen Satz stellt man sofort fest, dass $f_G(\epsilon) = A_G(\epsilon)$.*

Damit ist der Algorithmus korrekt, falls $r = \epsilon$. □

Der Fall $r = l$ wird in den Zeilen 3 und 4 behandelt.

Satz 9 ($\boldsymbol{f_G(l)}$) *Sei $l \in \Lambda$. Dann:*

$$\begin{aligned} f_G(l) = \big\{(u, v, \zeta) \mid (\exists e \in E_G) \quad & s_G(e) = u,\ t_G(e) = v, \\ & l_G(e) = l,\ \zeta(e) = 1 \\ \textit{und} \quad & (\forall e' \in E_G)\ e' \neq e \Rightarrow \zeta(e') = 0\big\} \end{aligned}$$

Beweis Nach den Definitionen 12 auf Seite 19 und 14 auf Seite 21 gilt:

$$\begin{aligned} \Pi_G(l) = \big\{(u, v, \zeta) \mid (\exists e \in E_G) \quad & s_G(e) = u,\ t_G(e) = v, \\ & l_G(e) = l,\ \zeta(e) = 1 \\ \textit{und} \quad & (\forall e' \in E_G)\ e' \neq e \Rightarrow \zeta(e') = 0\big\} \end{aligned}$$

und:

$$\big(\forall(\pi, \pi') \in \Pi_G(l)^2\big)\ \pi' \preceq \pi \Rightarrow \begin{cases} u = u' \\ v = v' \\ (\forall e \in E_G)\ \zeta'(e) \leq \zeta(e) \end{cases}$$

Sei $(\pi, \pi') \in \Pi_G(l)^2$, wobei $\begin{cases} \pi = (u, v, \zeta) \\ \pi' = (u', v', \zeta') \end{cases}$ und $\pi' \preceq \pi$.

Sei nun $e \in E_G$ mit $\zeta(e) = 1$ und $(\forall e'' \in E_G)\ e'' \neq e \Rightarrow \zeta(e'') = 0$.

Dann: $(\forall e'' \in E_G)\ e'' \neq e \Rightarrow \zeta'(e'') \leq \zeta(e'') = 0$.

Hieraus folgt, dass ζ gleich ζ' ist, und dass π' auch gleich π ist.

Denn $\Pi_G(l) = \mu\big(\Pi_G(l)\big) = f_G(l)$ ■

Korollar 3 *Nach dem vorherigen Satz gilt:*

$$(\forall l \in \Lambda)\ A_G(l) = f_G(l)$$

Wenn $r = l \in \Lambda$, dann terminiert der Algorithmus und gibt einen richtigen Ausgabewert zurück. □

Der Fall der Vereinigung wird in den Zeilen 5 und 6 behandelt.

Der Algorithmus ist rekursiv. Das bedeutet, dass $A_G(r_1)$ und $A_G(r_2)$ zuerst berechnet werden müssen, und dann $A_G(r = r_1 + r_2)$.

Lemma 2

$$\left(\forall (A_0, A_1) \subseteq \left(V^2 \times E^{\oplus}\right)^2\right)\ \mu(A_0 \cup A_1) = \mu\big(\mu(A_0) \cup \mu(A_1)\big)$$

Beweis Man zeigt die zwei Inklusionen ($\subseteq$ und $\supseteq$):

$\boxed{\subseteq}$ Sei $x \in \mu(A_0 \cup A_1)$.

Um zu zeigen, dass $x \in \mu\big(\mu(A_0) \cup \mu(A_1)\big)$ gilt, beweist man zuerst, dass $x \in \mu(A_i) \subseteq \mu(A_0) \cup \mu(A_1)$ und dass:
$\big(\forall y \in \mu(A_0) \cup \mu(A_1)\big)\ y \preceq x \Rightarrow y = x$.

- $x \in \mu(A_0 \cup A_1) \subseteq A_0 \cup A_1$ folgt aus der Voraussetzung.
  Sei $i \in \{0, 1\}$ mit $x \in A_i$.
  Sei nun ein beliebiges $y \in A_i \subseteq A_i \cup A_{1-i}$ mit $y \preceq x$.
  Da $x \in \mu(A_i \cup A_{i-1})$, $y = x$ und $x \in \mu(A_i) \subseteq \mu(A_0) \cup \mu(A_1)$.
- Sei ein beliebiges $y \in \mu(A_0) \cup \mu(A_1)$ mit $y \preceq x$.
  Sei $i \in \{0, 1\}$, wobei $x \in \mu(A_i)$.
  Es gibt jetzt zwei Fälle: $y \in \mu(A_i)$ oder $y \in \mu(A_{1-i})$.
  - $y \in \mu(A_i) \subseteq A_i$. Daraus folgt $y = x$
  - Wenn $y \in \mu(A_{1-i})$:
    Einerseits gilt $y \in \mu(A_{1-i}) \subseteq A_{1-i} \subseteq A_0 \cup A_1$, anderseits gilt $x \in \mu(A_0 \cup A_1)$.
    Da $y \preceq x$, ist y gleich x.

  Schließlich muss man beweisen, dass aus $x \in \mu\big(\mu(A_0) \cup \mu(A_1)\big)$ folgt: $x \in \mu(A_0 \cup A_1)$.

$\boxed{\supseteq}$ Sei $x \in \mu\big(\mu(A_0) \cup \mu(A_1)\big) \subseteq \mu(A_0) \cup \mu(A_1) \subseteq A_0 \cup A_1$.

Seien nun $y \in A_0 \cup A_1$ mit $y \preceq x$, und $i \in \{0, 1\}$ mit $y \in A_i$.

Dann gibt es ein z in $\mu(A_i)$, wobei $z \preceq y$.

Einerseits hat man $z \preceq y \preceq x$.

Anderseits:

$$\left.\begin{array}{l} x \in \mu\big(\mu(A_0) \cup \mu(A_1)\big) \\ z \in \mu(A_i) \subseteq \mu(A_0) \cup \mu(A_1) \end{array}\right\} \Rightarrow x = z$$

Daraus folgt, dass x gleich y ist. Dann ist $x \in \mu(A_0 \cup A_1)$.

■

Dieses Lemma führt zu folgendenem Satz:

Satz 10 ($f_G(r_1 + r_2)$) $f_G(r_1 + r_2) = \mu\big(f_G(r_1) \cup f_G(r_2)\big)$

Beweis Da $L(r_1 + r_2) = L(r_1) \cup L(r_2)$, hat man $P_G(r_1 + r_2) = P_G(r_1) \cup P_G(r_2)$ und $\Pi_G(r_1 + r_2) = \Pi_G(r_1) \cup \Pi_G(r_2)$.

Und mit Lemma 2 gilt $f_G(r_1 + r_2) = \mu\big(f_G(r_1) \cup f_G(r_2)\big)$. ■

Korollar 4 *Wenn $A_G(r_1)$ gleich $f_G(r_1)$ und $A_G(r_2)$ gleich $f_G(r_2)$ sind, gilt nach obigen Satz:*

$$A_G(r = r_1 + r_2) = f_G(r)$$

Dadurch wird der Fall $r = r_1 + r_2$ behandelt. □

Nun fährt man mit der Konkatenation fort: Zeile 7 bis 9.

Wie für die Vereinigung ist der Algorithmus rekursiv. $f_G(r_1)$ und $f_G(r_2)$ werden zuerst berechnet, und anschließend $f_G(r_1 r_2)$.

Um $f_G(r_1 r_2) = \mu\Big[\big\{(u, w, \zeta_3) \mid (\exists v \in V_G)(u, v, \zeta_1) \in f_G(r_1), (v, w, \zeta_2) \in f_G(r_2) \text{ und } \zeta_3 = \zeta_1 + \zeta_2\big\}\Big]$ zu beweisen, wird zuerst das folgende Lemma gezeigt:

Lemma 3 *Man hat das folgende Ergebnis:*

$$\mu\Big[\big\{(u, w, \zeta_3) \;\mid\; (\exists v \in V_G)(u, v, \zeta_1) \in f_G(r_1), (v, w, \zeta_2) \in f_G(r_2) \\ \text{und } \zeta_3 = \zeta_1 + \zeta_2\big\}\Big] \subseteq \Pi_G(r_1 r_2)$$

Beweis Wegen $L(r_1 r_2) = L(r_1)L(r_2)$ gilt $P_G(r_1 r_2) = \big\{p_1 p_2 \mid p_1 = e_{11} \dots e_{1m} \in P_G(r_1), p_2 = e_{21} \dots e_{2n}$ und $v = t_G(e_{1m}) = s_G(e_{21})\big\}$.

Dies ergibt dann nach Definition 12:

$$\begin{aligned}
\Pi_G(r_1 r_2) &= \big\{(u, w, \zeta_3) \mid && p_1 = e_{11} \dots e_{1m} \in P_G(r_1), \\
& && p_2 = e_{21} \dots e_{2n} \in P_G(r_2), \\
& && v = t_G(e_{1m}) = s_G(e_{21}), \\
& && s_G(e_{11}) = u \quad \text{und} \quad t_G(e_{2n}) = v, \\
& && (\forall e \in E_G)\ \zeta_3(e) = \#_e p_1 + \#_e p_2 = \zeta_1(e) + \zeta_2(e)\big\} \\
&= \big\{(u, w, \zeta_3) \mid && (\exists v \in V_G)\ (u, v, \zeta_1) \in \Pi_G(r_1), \\
& && (v, w, \zeta_2) \in \Pi_G(r_2) \\
& && \text{und} \quad \zeta_3 = \zeta_1 + \zeta_2\big\}
\end{aligned}$$

Da $f_G(r_1) \subseteq \Pi_G(r_1)$ und $f_G(r_2) \subseteq \Pi_G(r_2)$, gilt $\big\{(u, w, \zeta_3) \mid (\exists v \in V_G)(u, v, \zeta_1) \in f_G(r_1), (v, w, \zeta_2) \in f_G(r_2) \text{ und } \zeta_3 = \zeta_1 + \zeta_2\big\} \subseteq \Pi_G(r_1 r_2)$.

Dann gilt mit $\mu(A) \subseteq A$ das Lemma 3. ■

Jetzt kann man den folgenden Satz zeigen:

Satz 11 ($f_G(r_1r_2)$) *Für (r_1, r_2): zwei reguläre Ausdrücke über Λ gilt*

$$f_G(r_1r_2) = \mu\Big[\big\{(u,w,\zeta_3) \;\mid\; (\exists v \in V_G)\ (u,v,\zeta_1) \in f_G(r_1), (v,w,\zeta_2) \in f_G(r_2) \textit{ und } \zeta_3 = \zeta_1 + \zeta_2\big\}\Big]$$

Beweis Man beweist die zwei Inklusionen separat:

$\boxed{\supseteq}$ Seien $x \in \mu\Big[\big\{(u,w,\zeta_3) \mid (\exists v \in V_G)\ (u,v,\zeta_1) \in f_G(r_1), (v,w,\zeta_2) \in f_G(r_2) \text{ und } \zeta_3 = \zeta_1 + \zeta_2\big\}\Big] \subseteq \Pi_G(r_1r_2)$ und $y \in \Pi_G(r_1r_2)$, wobei $y \preceq x$.

Dann kann man $y = (u_y, w_y, \zeta_{3y})$ schreiben, wobei $v_y \in V_G$, $(u_y, v_y, \zeta_{1y}) \in \Pi_G(r_1)$, $(v_y, w_y, \zeta_{2y}) \in \Pi_G(r_2)$ und $\zeta_{3y} = \zeta_{1y} + \zeta_{2y}$.

Sei nun $z = (u_y, w_y, \zeta_{3z})$ mit $(u_y, v_y, \zeta_{1z}) \in f_G(r_1)$, $(v_y, w_y, \zeta_{2z}) \in f_G(r_2)$ und $\zeta_{3z} = \zeta_{1z} + \zeta_{2z}$.

Da $(u_y, v_y, \zeta_{1z}) \preceq (u_y, v_y, \zeta_{1y})$, gilt $(\forall e \in E_G)\ \zeta_{1z}(e) \leq \zeta_{1y}(e)$

Man hat ähnliche Ergebnisse mit ζ_{2y} und ζ_{2z}.

Es folgt: $(\forall e \in E_G)\ \zeta_{3z}(e) \leq \zeta_{3y}(e)$ und $z \preceq y$

Einerseits hat man: $x \preceq y \preceq x$. Anderseits ist x in $\big\{(u,w,\zeta_3) \mid (\exists v \in V_G)(u,v,\zeta_1) \in f_G(r_1), (v,w,\zeta_2) \in f_G(r_2) \text{ und } \zeta_3 = \zeta_1 + \zeta_2\big\}$.

Daher ist z gleich x.

Außerdem $x \preceq y \preceq x$, denn $y = x$ und x ist in $\mu\big[\Pi_G(r_1r_2)\big] = f_G(r_1r_2)$.

Dies ergibt die erste Inklusion: $\mu\Big[\big\{(u,w,\zeta_3) \mid (\exists v \in V_G)\ (u,v,\zeta_1) \in f_G(r_1), (v,w,\zeta_2) \in f_G(r_2) \text{ und } \zeta_3 = \zeta_1 + \zeta_2\big\}\Big] \subseteq f_G(r_1r_2)$.

$\boxed{\subseteq}$ Um die zweite Inklusion zu beweisen, zeigt man zuerst, dass $f_G(r_1r_2) \subseteq \big\{(u,w,\zeta_3) \mid (\exists v \in V_G)\ (u,v,\zeta_1) \in f_G(r_1), (v,w,\zeta_2) \in f_G(r_2) \text{ und } \zeta_3 = \zeta_1+\zeta_2\big\}$ gilt, und dann dass alle Elemente von $f_G(r_1r_2)$ minimale Elemente von dieser Menge sind:

- Sei $x \in \mu\big[\mu(G, r_1r_2)\big] = f_G(r_1r_2)$.
  Da x in $\Pi_G(r_1r_2)$ ist, gilt $x = (u,w,\zeta_3)$, wobei $v \in V_G$, $(u,v,\zeta_1) \in \Pi_G(r_1)$, $(v,w,\zeta_2) \in \Pi_G(r_2)$ und $\zeta_3 = \zeta_1 + \zeta_2$.
  Nun sei $z = (u,w,\zeta_3')$ mit: $(u,v,\zeta_1') \in \Pi_G(r_1)$, $(v,w,\zeta_2') \in \Pi_G(r_2)$ und $\zeta_3' = \zeta_1' + \zeta_2'$.
  Man weiß bereits: $z \preceq x$.
  $z \in \Pi_G(r_1r_2)$ und $x \in \mu\big[\Pi_G(r_1r_2)\big]$ geben zunächst, dass x gleich z ist: $x = z$.
  Denn es gilt: $x \in \big\{(u,w,\zeta_3) \mid (\exists v \in V_G)(u,v,\zeta_1) \in f_G(r_1), (v,w,\zeta_2) \in f_G(r_2) \text{ und } \zeta_3 = \zeta_1 + \zeta_2\big\}$.
- Sei $y \in \big\{(u,w,\zeta_3) \mid (\exists v \in V_G)\ (u,v,\zeta_1) \in f_G(r_1), (v,w,\zeta_2) \in f_G(r_2) \text{ und } \zeta_3 = \zeta_1 + \zeta_2\big\}$, wobei $y \preceq x$.
  Da $y \in \Pi_G(r_1r_2)$, $x \in \mu\big[\Pi_G(r_1r_2)\big]$ und $y \preceq x$, ist y gleich x $(y = x)$.
  Dann ist x in $\mu\Big[\big\{(u,w,\zeta_3) \mid (\exists v \in V_G)(u,v,\zeta_1) \in f_G(r_1), (v,w,\zeta_2) \in f_G(r_2) \text{ und } \zeta_3 = \zeta_1 + \zeta_2\big\}\Big]$.

Nun gilt $f_G(r_1r_2) \subseteq \mu\Big[\big\{(u,w,\zeta_3) \mid (\exists v \in V_G)(u,v,\zeta_1) \in f_G(r_1),\ (v,w,\zeta_2) \in f_G(r_2) \text{ und } \zeta_3 = \zeta_1 + \zeta_2\big\}\Big]$ und der Satz 11 ist bewiesen.

■

Korollar 5 *Es folgt, dass $A_G(r = r_1r_2) = f_G(r)$ gilt, wenn $A_G(r_1)$ gleich $f_G(r_1)$ und $A_G(r_2)$ gleich $f_G(r_2)$ ist.* □

Da der Fall $r = r_1r_2$ behandelt wurde, kann man nun die Berechnung der *Kleene'sche Hülle* überprüfen.

Dieser Fall (r^*) wird in den Zeilen 10 bis 18 im Algorithmus 1 behandelt.

Die Aufgabe ist nun zu beweisen, dass:

- der Algorithmus terminiert.

  Er genügt, zu zeigen, dass $\{p \in \mathbb{N} \mid A_p = A_{p+1}\}$ keine leere Menge ist.

- der Algorithmus korrekt ist.

  Zuerst wird bewiesen, dass $A_G(r^*) = A_n = f_G(r^*)$ gilt, wenn $(\forall i \in \mathbb{N})\ f_G(r^i) = A_G(r^i)$ gilt.

  Dann kann man durch Induktion über r zeigen, dass $(\forall r \in reg(\Lambda))\ A_G(r) = f_G(r)$ gilt.

Sei $r \in reg(\Lambda)$ ein regulärer Ausdruck über Λ.

In den nächsten Sätze wird angenommen, dass:

$$(\forall i \in \mathbb{N})\ A_G(r^i) = f_G(r^i)$$

Dementsprechend kann man $(A_n)_{n\in\mathbb{N}}$ folgendermaßen schreiben:

$$\begin{cases} A_0 = f_G(\epsilon) = f_G(r^0) \\ A_{n+1} = \mu\big[A_n \cup f_G(r^{n+1})\big] \end{cases}$$

Diese Definition von $(A_n)_{n\in\mathbb{N}}$, wird in den nächsten Beweisen benutzt.

Satz 12 *Für alle $n \in \mathbb{N}$ gilt mit der Hypothese $(\forall i \in \mathbb{N})\ A_G(r^i) = f_G(r^i)$:*

$$A_n = \mu\left[\bigcup_{i=0}^{n} f_G(r^i)\right]$$

Beweis Dieser Satz wird mit Induktion gezeigt.

$\boxed{n = 0}$ Für $n = 0$ gilt:

$$A_0 = f_G(r^0) = \mu\left[f_G(r^0)\right]$$

$\boxed{n \to n+1}$ Sei $A_n = \mu\left[\bigcup_{i=0}^{n} f_G(r^i)\right]$ die Induktionsvoraussetzung.

Dann gilt:

$$A_{n+1} = \mu\left(\mu\left[\bigcup_{i=0}^{n} f_G(r^i)\right] \cup f_G(r^{n+1})\right)$$

Da Lemma 2 das folgende Ergebnis: $\mu\left[\mu(B) \cup \mu(C)\right] = \mu(B \cup C)$ ergab, folgt daraus:

$$A_{n+1} = \mu\left(\mu\left[\bigcup_{i=0}^{n} f_G(r^i) \cup f_G(r^{n+1})\right]\right) = \mu\left[\bigcup_{i=0}^{n+1} f_G(r^i)\right]$$

Die Induktionsvoraussetzung ist damit für $n+1$ geprüft.

■

Satz 13 *Wenn* $(\forall i \in \mathbb{N})\ A_G(r^i) = f_G(r^i)$ *gilt:*

$$A_n \xrightarrow[n\to\infty]{} f_G(r^*)$$

Beweis Da $r^* = \sum_{i=0}^{\infty} r^i$, gilt nach dem Lemma 2: $f_G(r^*) = \mu\left[\bigcup_{i=0}^{\infty} f_G(r^i)\right]$. ■

Beweisen wir nun, dass es ein kleinstes $n \in \mathbb{N}$ gibt, wobei $A_{n+1} = A_n$ gilt, für das auch das folgende Ergebnis gilt: $A_n = A_\infty = f_G(r^*)$.

Dafür beginnt man mit dem folgenden Lemma:

Lemma 4 *Für alle* $n \in \mathbb{N}$ *gilt mit der Annahme* $(\forall i \in \mathbb{N})\ A_G(r^i) = f_G(r^i)$*:*

$$A_{n+1} = A_n \Leftrightarrow \big(\forall x \in f_G(r^{n+1})\big)(\exists y \in A_n)\ y \preceq x \tag{4.1}$$

Beweis Rufen wir uns in Erinnerung, dass $A_{n+1} = \mu\big\{A_n \cup f_G(r^{n+1})\big\}$ gilt, wenn $(\forall i \in \mathbb{N})\ A_G(r^i) = f_G(r^i)$.

Der Beweis besteht aus zwei Teilen:

$\boxed{\Rightarrow}$ Beweisen wir hier den Satz durch Widerlegen der Gegenhypothese:

$$\big(\exists x \in f_G(r^{n+1})\big)(\forall y \in A_n)\ y \npreceq x \Rightarrow A_{n+1} \neq A_n$$

Sei $x_0 \in f_G(r^{n+1})$ mit:

$$(\forall y \in A_n)\ y \npreceq x_0 \tag{4.2}$$

Ergo: $x_0 \notin A_n$.

Da $x_0 \in f_G(r^{n+1})$, ist x_0 nach der Definition 17 in $\Big\{\pi \in \Pi_G(r^{n+1}) \mid \big(\forall \pi' \in \Pi_G(r^{n+1})\big)\ \pi' \preceq \pi \Rightarrow \pi' = \pi\Big\}$.

Wegen $f_G(r^{n+1}) \subseteq \Pi_G(r^{n+1})$ gilt: $\Big\{\pi \in \Pi_G(r^{n+1}) \mid \big(\forall \pi' \in \Pi_G(r^{n+1})\big)\ \pi' \preceq \pi \Rightarrow \pi' = \pi\Big\} \subseteq \Big\{\pi \in \Pi_G(r^{n+1}) \mid \big(\forall \pi' \in f_G(r^{n+1})\big)\ \pi' \preceq \pi \Rightarrow \pi' = \pi\Big\}$.

x_0 ist dann in $\left\{\pi \in \Pi_G(r^{n+1}) \mid \big(\forall \pi' \in f_G(r^{n+1})\big)\ \pi' \preceq \pi \Rightarrow \pi' = \pi\right\}$.

Denn:

$$\big(\forall \pi' \in f_G(r^{n+1})\big)\ \pi' \preceq x_0 \Rightarrow \pi' = x_0 \tag{4.3}$$

Die Gleichungen 4.2 und 4.3 führen zunächst zu dem folgenden Ausdruck:

$$\big(\forall y' \in A_n \cup f_G(r^{n+1})\big)\ y' \preceq x_0 \Rightarrow y' = x_0$$

Es folgt: $x_0 \in \mu\big[f_G(r^{n+1}) \cup A_n\big] = A_{n+1}$ und $A_n \neq A_{n+1}$.

$\boxed{\Leftarrow}$ Hier werden die beiden Inklusionen bewiesen:

$\boxed{\subseteq}$ Sei $x_0 \in A_{n+1}$.
Dann gilt:

$$\big(\forall y \in f_G(r^{n+1}) \cup A_n\big)\ y \preceq x_0 \Rightarrow y = x_0 \tag{4.4}$$

Und angenommen dass:

$$x_0 \in f_G(r^{n+1}) \setminus A_n \tag{4.5}$$

Es gibt also ein $y_0 \in A_n$, wobei $y_0 \preceq x_0$.
Daraus folgt, dass nach 4.4 y_0 gleich x_0 ist und dass x_0 in A_n ist: $x_0 \in A_n$.
Da dies mit 4.5 in Widerspruch steht, gehört x nicht zu $f_G(r^{n+1}) \setminus A_n$.
Denn x_0 ist in A_n und $A_{n+1} \subseteq A_n$.

$\boxed{\supseteq}$ Sei $x_0 \in A_n$.
Nun ist zu beweisen, dass es keine Elemente aus $A_n \cup f_G(r^{n+1})$ gibt, die kleiner als x_0 sind.
Zwei Fällen sollen überprüft werden:

- Zuerst sei $y_0 \in A_n$.
  Da $A_n = \mu(A_n)$, gilt: $y_0 \preceq x_0 \Rightarrow y_0 = x_0$.
- Dann sei $y_1 \in f_G(r^{n+1})$.
  Es gibt diesmal ein z_1 in A_n, wobei $z_1 \preceq y_1$.
  Damit folgt:

$$\begin{aligned} y_1 \preceq x_0 \quad &\Rightarrow \quad z_1 \preceq y_1 \preceq x_0 \\ &\Rightarrow \quad z_1 = x_0 \text{ (weil } z_1 \in A_n) \\ &\Rightarrow \quad x_0 = y_1 \end{aligned}$$

Schlussletztlich gilt: $x_0 \in A_{n+1}$

■

Das letzte Lemma führt zu dem folgenden Satz:

Satz 14 (Stationär) *Wenn $A_{n+1} = A_n$ und $(\forall i \in \mathbb{N})$ $A_G(r^i) = f_G(r^i)$ gilt, ist die Folge $(A_p)_{p \geq n}$ stationär:*

$$A_{n+1} = A_n \Rightarrow A_{n+2} = A_{n+1}$$

Beweis Sei zuerst der folgende Operator:

$$\begin{array}{rlcl} \boxplus: & (V^2 \times E^{\oplus})^2 & \to & V^2 \times E^{\oplus} \cup \textsc{Error} \\ & \big((u,v,\zeta_1),(v,w,\zeta_2)\big) & \mapsto & (u,w,\zeta_3 = \zeta_1 + \zeta_2) \\ & \big((u,v,\zeta_1),(v',w,\zeta_2)\big) & \mapsto & \textsc{Error}, \textit{wenn } v \neq v' \end{array}$$

Sei nun $x_0 \in f_G(r^{n+2})$.

Man kann nach Satz 11 x_0 wie folgt schreiben: $x_0 = y_0 \boxplus z_0$, wobei $y_0 \in f_G(r^{n+1})$ und $z_0 \in f_G(r)$.

Die Gleichung 4.1 von Lemma 4 ergibt dann die Existenz von einem y_1 in A_n mit $y_1 \preceq y_0$.

Da $A_n = \mu\left[\bigcup_{i=0}^{n} f_G(r^i)\right]$, ist y_1 in $f_G(r^k)$ mit $k \leq n$.

Daraus folgt, dass es ein x_1 in $f_G(r^{k+1})$ gibt, wobei: $x_1 \preceq y_1 \boxplus z_0 \preceq y_0 \boxplus z_0 = x_0$ ($\boxplus$ ist nämlich monoton), und dass es ein x_2 in A_n mit $x_2 \preceq x_1$ gibt.

Schließlich gibt es ein x_2 in $A_n = A_{n+1}$ mit $x_2 \preceq x_0$ und $A_{n+2} = A_{n+1} = A_n$. ■

Dieses Ergebnis ergibt zunächst die folgenden Korollare:

Korollar 6 *Für alle $n \in \mathbb{N}$ gilt mit der Hypothese $(\forall i \in \mathbb{N})\ A_G(r^i) = f_G(r^i)$:*

$$A_{n+1} = A_n \Rightarrow (\forall k \geq 0)\ A_{n+k} = A_n$$

Beweis Nach dem Satz 14 mit Induktion. ■

Korollar 7 *Für $A_{n+1} = A_n$ und $(\forall i \in \mathbb{N})\ A_G(r^i) = f_G(r^i)$, wird die Folge ab n stationär.*

Beweis Aus dem Korollar 6. ■

Korollar 8 *Für $A_{n+1} = A_n$ und $(\forall i \in \mathbb{N})\ A_G(r^i) = f_G(r^i)$, ist der Grenzwert der Folge $(A_n)_{n\in\mathbb{N}}$ $A_\infty = A_n$.*

Beweis Das ist die Begrenzung einer stationären Folge. ■

Korollar 9 *Mit der Voraussetzung, dass $(\forall i \in \mathbb{N})\ A_G(r^i) = f_G(r^i)$ und dass der Algorithmus terminiert, gilt:*

$$A_G(r^*) = f_G(r^*)$$

Beweis Im Fall $r' = r^*$ terminiert der Algorithmus, wenn in der Schleife $Y = X$ erfüllt wird (siehe Zeile 14 im Algorithmus 1 auf Seite 28). Das bedeutet, dass ein $n \in \mathbb{N}$ existiert, mit $A_n = A_{n+1}$. Hierbei ist X gleich A_{n+1} und Y gleich A_n.

Wenn $(\forall i \in \mathbb{N})\ A_G(r^i) = f_G(r^i)$ gilt, ist nach dem Korollar 8 und dem Satz 13 X gleich $A_\infty = f_G(r^*)$. Daraus folgt, dass $A_G(r^*) = f_G(r^*)$ gilt, wenn $(\forall i \in \mathbb{N})\ A_G(r^i) = f_G(r^i)$ gilt und wenn der Algorithmus terminiert. ■

Korollar 10 *Wenn der Algorithmus terminiert, ist er korrekt.*

Beweis Nach den Korollaren 2 auf Seite 29, 3 auf Seite 30, 4 auf Seite 31, 5 auf Seite 33 und 9 auf der vorherigen Seite durch Induktion über r. ■

Die Aufgabe besteht nun darin, festzustellen, ob es immer ein n mit $A_n = A_{n+1}$ gibt.

Dafür suchen wir eine Ordnung, bei der $(A_n)_{n\in\mathbb{N}}$ steigend ist und für die es eine obere Schranke gibt.

Desweiteren ist zu beweisen, dass die Menge von Elementen, die kleiner gleich als diese obere Schranke sind, endlich ist.

Dies führt zu den folgenden zwei Definitionen:

Definition 21 Sei E die Menge, die wie folgt definiert wird:

$$E = \left\{ A \subseteq \mathcal{P}(V^2 \times E^{\oplus}) \mid \left(\forall (x,y) \in A^2\right) x \preceq y \Rightarrow x = y \right\}$$

In E sind alle Mengen, die nur kleinste Elemente haben. □

Daraus folgen diese zwei Bemerkungen:

Bemerkung 1 Da $A_i = \mu \left[\bigcup_{i=0}^{n} f_G(r^i)\right]$, ist A_i in E für alle $i \in \mathbb{N}$. □

Bemerkung 2 Zwei verschiedene Elemente einer Menge von E sind unvergleichbar. □

Definieren wir zunächst eine partielle Ordnung dieser Menge E:

Definition 22 ($\trianglelefteq$) Für alle $(A_i, A_j) \in E^2$, gilt:

$$A_i \trianglelefteq A_j \Leftrightarrow (\forall x \in A_i)(\exists y \in A_j)\ y \preceq x$$

□

Die Definition soll mit dem folgenden Satz begleitet werden:

Satz 15 *Die Ordnung $\trianglelefteq$, die oben definiert wird, ist eine* partielle Ordnung *in E.*

Beweis Es reicht hier noch, die Reflexivität, die Transitivität und die Antisymmetrie zu zeigen:

Reflexivität: Da $(\forall x \in A)\ x \preceq x$, reicht es für alle $x \in A$, x zu wählen.

Es gilt dann $A \trianglelefteq A$.

Transitivität: Angenommen, dass $A \trianglelefteq B$ und $B \trianglelefteq C$ gilt.

Sei nun $x \in A$.

Nach $A \trianglelefteq B$ gibt es ein $y \in B$ mit $y \preceq x$.

$B \trianglelefteq C$ ergibt zunächst die Existenz von einem $z \in C$ mit $z \preceq y$.

Daraus folgt, dass $z \preceq y \preceq x$ und $A \trianglelefteq C$ gilt.

Antisymmetrie: Angenommen sei, dass $A \trianglelefteq B$ und $B \trianglelefteq A$ gilt.

Sei $x \in A$.

Wie oben gibt es aufgrund von $A \trianglelefteq B$ ein $y \in B$, wobei $y \preceq x$, und wegen $B \trianglelefteq A$ ein $x' \in A$ mit $x' \preceq y$.

Zusammenfassend gilt $x' \preceq y \preceq x$.

Da $A \in E$ und $x \preceq x'$ gilt, ist x gleich x'.

Daraus folgt, dass x gleich $y \in B$ ist und damit auch $A \subseteq B$.

Man wiederholt das Gleiche mit B und es gilt $A = B$.

■

Daraus folgt:

Satz 16 *Die Folge $(A_n)_{n\in\mathbb{N}}$ ist monoton steigend mit der Ordnung $(E, \trianglelefteq)$.*

BEWEIS Sei $x \in A_n$.

Um zu zeigen dass $A_n \trianglelefteq A_{n+1}$ gilt (d.h. $(\forall x \in A_n)(\exists y \in A_{n+1})\ y \preceq x$), finden wir ein $y \in A_{n+1}$, für welches $y \preceq x$ gilt.

- Entweder $x \in A_{n+1}$ und es reicht, $y = x \preceq x$ zu wählen.
- Oder $x \notin A_{n+1}$.

  In diesem Fall bauen wir rekursiv eine Folge $(x_i)_{i\in\mathbb{N}}$:

  - $x_0 = x$
  - Wenn $x_i \in A_{n+1}$, $x_{i+1} = x_i$
  - Wenn $x_i \notin A_{n+1}$, gibt es nach der Definition von $A_{n+1} = \mu\big[A_n \cup f_G(r^{n+1})\big]$ ein $x_{i+1} \in A_n \cup f_G(r^{n+1})$, bei dem $x_{i+1} \preceq x_i$ und $x_{i+1} \neq x_i$ gilt.

  Zusammenfassung:

  $$(x_i)_{i\in\mathbb{N}} \begin{cases} x_0 = x & \\ \text{wenn } x_i \in A_{n+1}, & x_{i+1} = x_i \\ \text{wenn } x_i \notin A_{n+1}, & x_{i+1} \in A_n \cup f_G(r^{n+1}) \\ & \text{mit } x_{i+1} \preceq x_i \text{ und } x_{i+1} \neq x_i \end{cases}$$

  Diese Folge $(x_i)_{i\in\mathbb{N}}$ ist offensichtlich abfallend.

  Da $A_n \cup f_G(r^{n+1})$ endlich ist, $x_i \xrightarrow[i\to\infty]{} x_\infty \in A_{n+1}$ und $x_\infty \preceq x_0 = x$.

  Denn $A_n \trianglelefteq A_{n+1}$ und die Folge $(A_n)_{n\in\mathbb{N}}$ steigt mit $(E, \trianglelefteq)$.

■

Die Abbildung 4.1 auf der nächsten Seite beschreibt das Verhalten der Folge $(A_n)_{n\in\mathbb{N}}$.

Satz 17 *Die Menge $\{A_i \mid i \in \mathbb{N}\}$ ist endlich.*

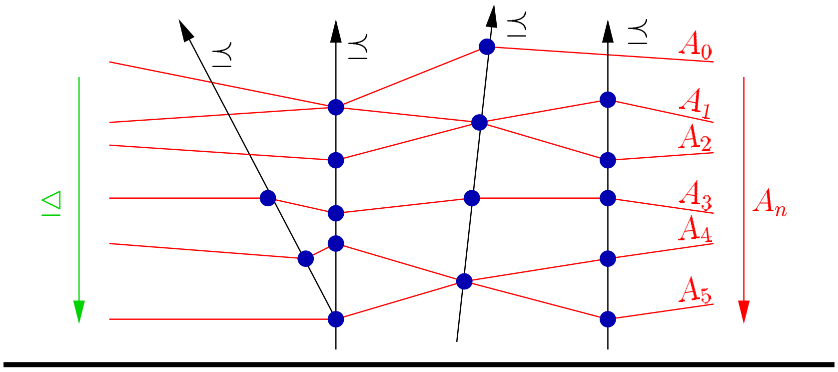


Abbildung 4.1: Die Ordnung $(E, \trianglelefteq)$ und die Folge $(A_n)_{n\in\mathbb{N}}$

BEWEIS Man beweist diesen Satz in zwei Stufen.

Zuerst zeigt man, dass es ein n_{max} gibt, so dass alle Elemente aus A_n für $n \geq n_{max}$ mit mindestens einem Element aus einem A_k – $k \leq n_{max}$ – vergleichbar sind.

Zunächst kann man einfacher beweisen, dass die Menge $\{A_n \mid n \geq n_{max}\}$ endlich ist.

- Zu Anfang beweisen wir die Existenz von n_{max}.

  Sei der folgende Operator:

$$\begin{array}{rlcl} \phi: & \mathcal{D} \subseteq \mathbb{N} & \to & \mathbb{N} \\ & 0 & \mapsto & 0 \\ & p \geq 1 & \mapsto & \phi_p = \min\big\{n \in \mathbb{N} \mid n > \phi_{p-1} \text{ und} \\ & & & \qquad (\exists y_p \in A_n)(\forall x \in \bigcup_{i=0}^{\phi_{p-1}} A_i)\ y_p \not\preceq x\big\} \end{array}$$

  $\mathcal{D}$ ist die Menge, auf der ϕ definiert ist. Das heißt, dass $\mathcal{D}$ die Menge von $p \in \mathbb{N}$ ist, wobei $p-1$ schon in $\mathcal{D}$ ist und wobei $\big\{n \in \mathbb{N} \mid n > \phi_{p-1}$ und $(\exists y_p \in A_n)(\forall x \in \bigcup_{i=0}^{\phi_{p-1}} A_i)\ y_p \not\preceq x\big\}$ nicht leer ist.

  Die vorherige Definition zeigt auch die Existenz einer Folge $(y_p)_{p\in\mathcal{D}^*}$.

  Beispiel 5 Nehmen wir wieder das vorherige Beispiel. In diesem Fall gilt: $\phi_0 = 0$, $\phi_1 = 1$, $\phi_2 = 3$ und $\mathcal{D} = \{0, 1, 2\}$.

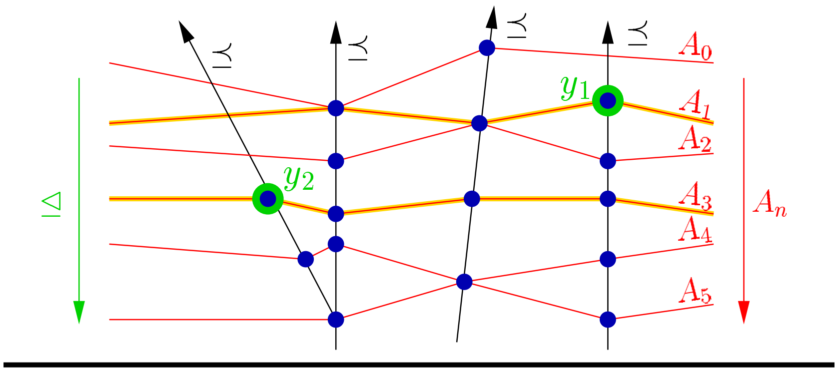


Abbildung 4.2: Der Aufbau von ϕ und $(y_p)_{p\in\mathcal{D}^*}$

  Der Aufbau der Folge $(y_p)_{p\in\mathcal{D}^*}$ wird in Abbildung 4.2 gezeigt. □

Zeigen wir zunächst in zwei Stufen, dass zwei verschiedene Elemente von $(y_p)_{p\in\mathcal{D}^*}$ unvergleichbar sind: $\left(\forall(i,j)\in\mathcal{D}^{*2}\right)\ y_i \npreceq y_j\ und\, y_j \npreceq y_i$.

- $(y_i, y_j) \in A_{\phi_i} \times A_{\phi_j}$, wobei $y_i \in A_{\phi_i} \subseteq \bigcup_{k=0}^{\phi_j-1} A_k$, gibt: $(\forall i < j)\ y_j \npreceq y_i$.
- Angenommen sei, dass ein Tupel $(i,j) \in \mathbb{N}^2$ existiert, wobei $i < j$ und $y_i \prec y_j$ gilt. Da $y_i \in A_{\phi_i} \subseteq \bigcup_{k=0}^{\phi_j-1} A_k$ und $y_j \in A_{\phi_j} = \mu\left[\left(\bigcup_{k=0}^{\phi_j-1} A_k\right) \cup f_G(r^{\phi_j})\right]$ gilt, ist y_i nach der Definition 16 auf Seite 22 von μ gleich y_j.
  Dies ruft einen Widerspruch hervor, da $(\forall i < j)\ y_i \nprec y_j$. Außerdem hat man schon bewiesen, dass $y_i \neq y_j$. Ergo gilt $(\forall i < j)\ y_i \npreceq y_j$.

Daraus folgt nach Lemma 1 auf Seite 23, dass $(y_p)_{p\in\mathcal{D}^*}$ endlich ist.

Da alle y_p ungleich sind, ist $\mathcal{D}$ endlich und man kann $\mathcal{D}$ wie folgt darstellen: $\mathcal{D} = [|0, p_{max}|]$, wobei $[|i,j|]$ gleich $\{i, i+1, i+2, \ldots, j\}$ ist.

Dann kann man $\phi_{p_{max}}$ für n_{max} wählen.

- Der vorherige Abschnitt ergibt die folgende Eigenschaft:

$$(\forall n \geq \phi_{p_{max}})(\forall y \in A_n)\left(\exists x \in \bigcup_{i=0}^{\phi_{p_{max}}} A_i\right)\ y \preceq x \tag{4.6}$$

Die Aufgabe ist nun, zu beweisen, dass $\{A_n \mid n \geq \phi_{p_{max}}\}$ endlich ist.

Da $\{x \in V^2 \times E^{\oplus} \mid x \preceq \pi\}$ für alle $\pi \in V^2 \times E^{\oplus}$ und A_i für alle $i \in \mathbb{N}$ endlich ist, ist $H = \left\{y \in V^2 \times E^{\oplus} \mid \left(\exists x \in \bigcup_{i=0}^{\phi_{p_{max}}} A_i\right)\ y \preceq x\right\}$ auch endlich.

Nach der Gleichung 4.6 gilt zunächst: $(\forall n \geq \phi_{p_{max}})\ A_n \subseteq H$.

Denn $\{A_n | n \geq \phi_{p_{max}}\}$ ist endlich.

Die Menge H wird in Abbildung 4.3 dargestellt. Diese veranschaulicht auch, warum $\{A_n \mid n \geq \phi_{p_{max}}\}$ endlich ist.

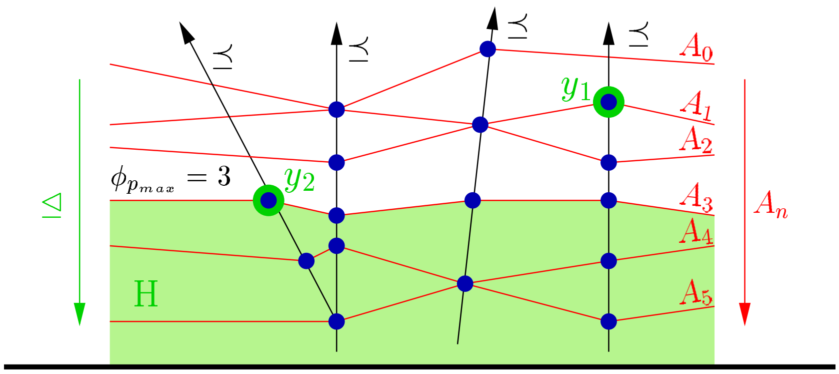


Abbildung 4.3: Die Menge H und $(A_n)_{n\in\mathbb{N}}$

Schließlich ist $\{A_i \mid i \in \mathbb{N}\}$ endlich. ■

Die zwei letzten Sätze führen zu den zwei letzten Ergebnissen:

Korollar 11 *Es existiert ein* $n \in \mathbb{N}$, *wobei* $A_n = A_{n+1}$.

Beweis Folgt direkt aus den Sätze 16 auf Seite 38 und 17 auf Seite 38. ■

Korollar 12 *Der Algorithmus 1 auf Seite 28 ist korrekt und terminiert.*

Beweis Nach Korollar 10 und Korollar 11. ■

Kapitel 5

Anwendung an einem Beispiel

5.1 Öffentliche-Private Server – Version 1

5.1.1 Ziel

Nehmen wir wieder das Beispiel des Abschnitts 1.3 auf Seite 5.

Man will nun automatisch prüfen, ob die Konfigurationen $S_{prv}C^*S_{pub}$ und $S_{prv}P_{ext}$ in dem dynamischen System nie stattfinden (siehe Abschnitt 1.4 auf Seite 5)

Dazu benutzt man die Werkzeuge, die in 1.1 auf Seite 1 und auf Abbildung 1.2 auf Seite 4 beschrieben wurden.

5.1.2 `aunfold`

Der erste Schritt ist, das Graphtransformations-System in einen Petri-Graphen mit Hilfe von `aunfold` zu verwandeln.

Die Eingangsdatei soll mit einer Erweiterung der GXL (Graph eXchange Language, http://www.gupro.de/GX
die GTXL (Graph Transformation eXchange Language) geschrieben werden.

Leider gibt es eine Begrenzung in der Implementierung. In der Regel sollte man eine Hyperkante nicht zweimal mit dem gleichen Label auf der linken Seite finden.

Da dies der Fall in 1.3 auf Seite 6 ist, müssen die Regeln umformuliert werden. Die Änderung sieht man in Abbildung 5.1 auf der nächsten Seite.

Dies ergibt die Eingangsdatei `pub_priv_server.xml`.

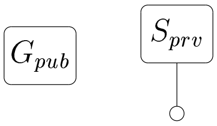
Einen Auszug der Eingangsdatei findet man im Anhang (siehe A.1.1 auf Seite 61).

```
% aunfold -nc pub_priv_server.xml pub_priv_server.gxl
```

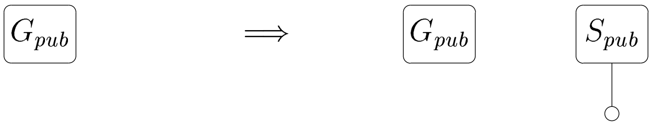
wandelt dann den Petri-Graphen in GXL Format um.

Mit `pg2neato` kann man die Ergebnisse graphisch visualisieren: der Hypergraph in Abbildung 5.2 auf Seite 45 und das Petri-Netz in Abbildung 5.3 auf Seite 46.

Anfangsgraph:

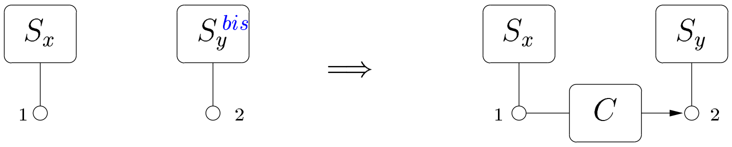


Regeln:

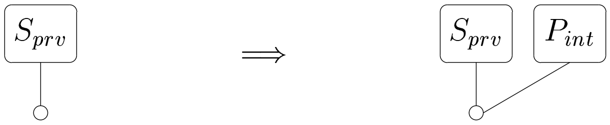


einen öffentlichen Server herstellen

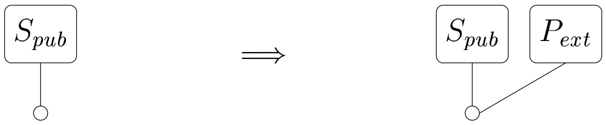
einen öffentlichen Generator löschen



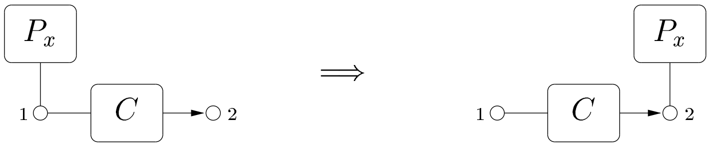
eine Verbindung herstellen
$(x \neq pub \vee y \neq prv)$
$x, y \in \{pub, prv\}$



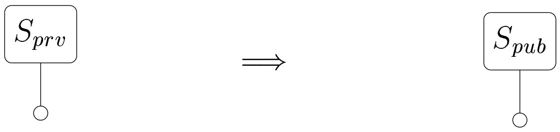
einen internen Prozess herstellen



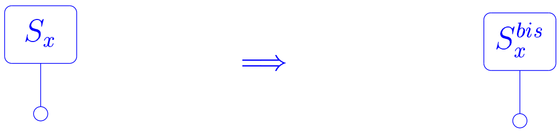
einen externen Prozess herstellen



ein Prozess überquert eine Verbindung



privater Server wird zum öffentlichen Server



den Zustand eines Servers verwandeln
$x \in \{pub, prv\}$

Abbildung 5.1: Neue Grammatik

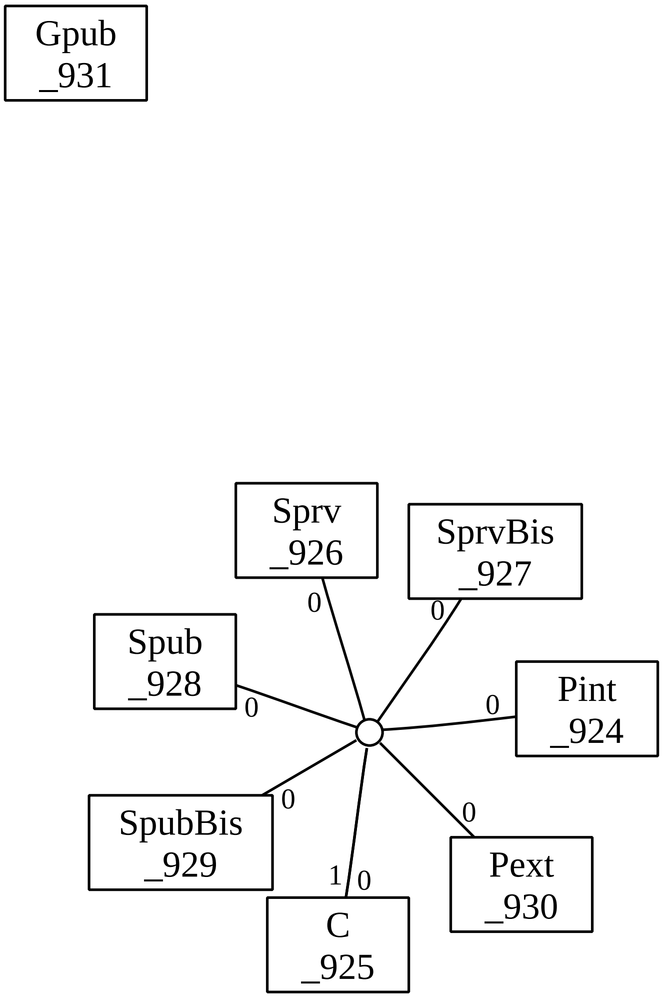


Abbildung 5.2: Öffentliche-Private Server – Version 1 – Graph

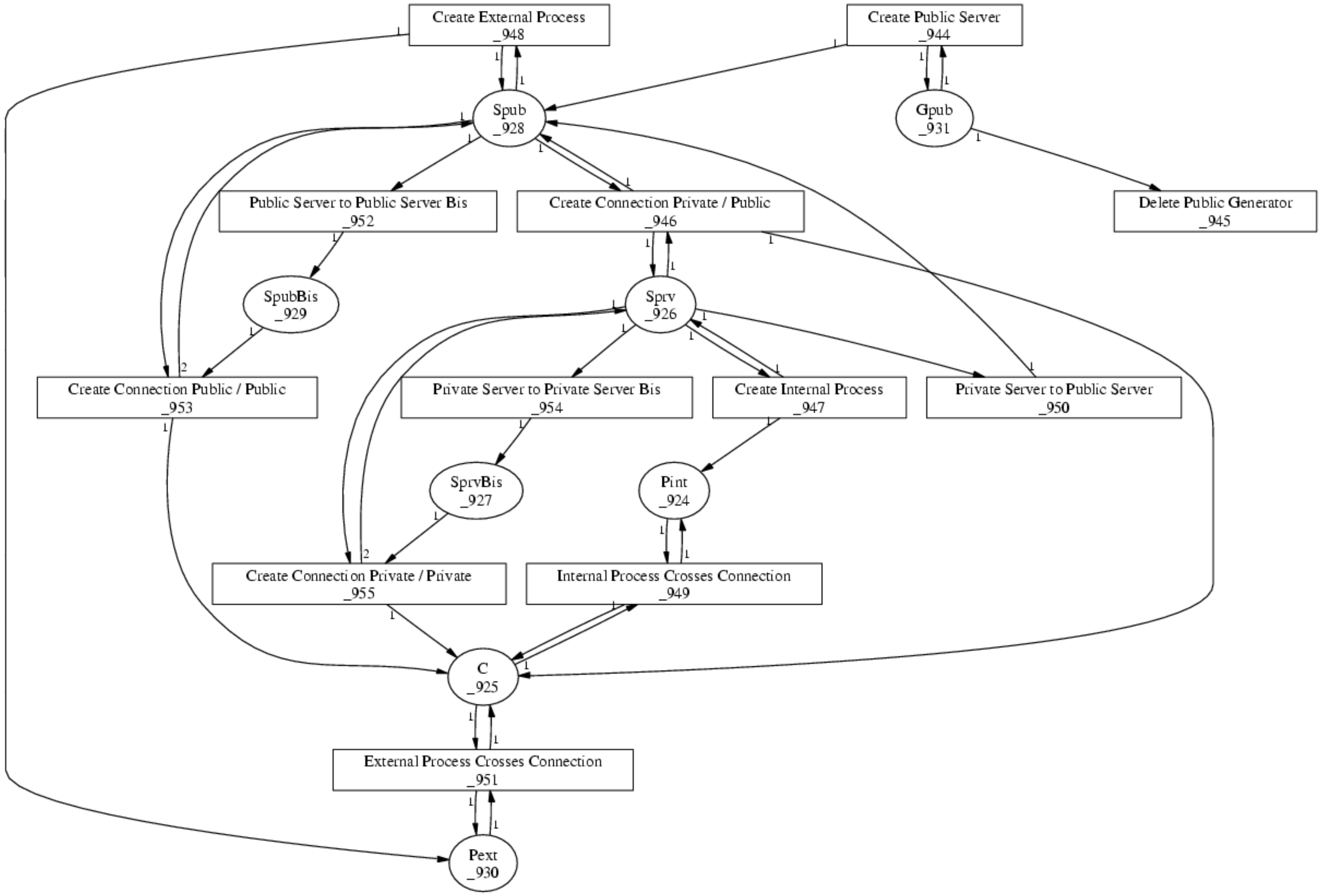


Abbildung 5.3: Öffentliche-Private Server – Version 1 – Petri-Netz

5.1.3 sponge

Gegeben ist ein regulärer Ausdruck und der Petri-Graph aus `aunfold`. So erlaubt `sponge` die Formel zu erzeugen, die in Abschnitt 3.2 auf Seite 25 beschrieben wird.

`sponge` implementiert den Algorithmus 1 auf Seite 28.

Um die Formel zu erzeugen:

```
% sponge -q -f pub_priv_server.gxl
'Spub'C*'Sprv'
FORMULA ( Sprv_926 >= 1 AND Spub_928 >= 1 )

% sponge -q -f pub_priv_server.gxl
'Sprv''Pext'
FORMULA ( Sprv_926 >= 1 AND Pext_930 >= 1 )
```

Man kann diese Ergebnisse auch in der Syntax von Definition 7 schreiben:

$$\begin{aligned} F_{S_{pub}C^*S_{prv}} &= (\#Sprv\_926 \geq 1) \wedge (\#Spub\_928 \geq 1) \\ F_{S_{prv}P_{ext}} &= (\#Sprv\_926 \geq 1) \wedge (\#Pext\_930 \geq 1) \end{aligned}$$

Das Programm `bra`, das die Überdeckbarkeit in Petri-Netz überprüft, versteht eine andere Syntax. Die gesuchte Datei wird dann mit den folgenden Kommandos gegeben:

```
% sponge -q pub_priv_server.gxl
'Spub'C*'Sprv'
MARKING Sprv_926: 1, Spub_928: 1;

% sponge -q pub_priv_server.gxl
'Sprv''Pext'
MARKING Sprv_926: 1, Pext_930: 1;
```

Dies stellt die Menge minimalen Markierungen M dar:

$$m' \models F \Leftrightarrow (\exists m \in M)\ m' \geq m$$

5.1.4 bra

Mit `bra` (für *Backward Reachability*) kann man überprüfen, ob es eine erreichbare Markierung gibt, die die Formel erfüllt. Dieses Werkzeug stammt aus dem Studienarbeit von Sinan Turan [Tur04] und ist eine Implementierung des Algorithmus, der in [AJKP98] beschrieben wird.

Die Eingangsdateien sind:

- das Petri-Netz in LoLA Format (siehe http://www.informatik.hu-berlin.de/~kschmidt/lola.html)
- Eine Liste von Markierungen (siehe oben)

Um das Petri-Netz in LoLA Format zu erhalten:

```
% pg2lola pub_priv_server.gxl > pub_priv_server.lola
```

Nun kann man `bra` ausführen:

```
% bra -q pub_priv_server.lola pub_priv_server-markierung
true

% bra -q pub_priv_server.lola pub_priv_server-markierung2
true
```

5.1.5 Schlussbemerkung

Nach dem vorherigen Abschnitt ist es möglich, eine erreichbare Markierung zu finden, die die Formel erfüllt. Aber das bedeutet noch nicht, dass es möglich ist, eine Konfiguration mit einem Wort aus dem regulären Ausdruck in dem dynamischen System zu finden.

`aunfold` approximiert nämlich manchmal das ursprüngliche System zu stark. Dabei gehen Eigenschaften verloren. Wenn man die Abbildungen 2.1 auf Seite 10 und 5.2 auf Seite 45 vergleicht, bemerkt man, dass zwei Knoten verschmolzen sind, obwohl dies nicht wünschenswert ist.

Man muss dann das System etwas genauer approximieren. Dafür gibt es zwei Möglichkeiten:

- ein äquivalentes System finden, bei dem die zwei Knoten nicht verschmelzen
- Optionen von `aunfold` verwenden, so dass die zwei Knoten getrennt bleiben

Im nächsten Abschnitt versucht man die zweite Lösung.

5.2 Öffentliche-Private Server – Version 2

5.2.1 Überblick von `-depth` Option in `aunfold`

In dieser Version benutzt man die `-depth=1` Option von `aunfold`, so dass die zwei Knoten getrennt bleiben. `-depth=k` erreicht, dass Knoten bis zur Tiefe `k` (Tiefe bezüglich der Entfaltung) nicht miteinander verschmolzen werden.

5.2.2 `aunfold`

Man berechnet zuerst den Petri-Graphen:

```
% aunfold -nc -depth=1 pub_priv_server.xml pub_priv_server2.gxl
```

Das Ergebnis ist in den Abbildungen 5.4 auf der nächsten Seite und 5.5 auf Seite 50 dargestellt. Einen Auszug der Ausgangsdatei findet man im Anhang (siehe A.1.2 auf Seite 62).

Man stellt fest, dass das System jetzt viel komplizierter ist, die zwei Knoten getrennt geblieben sind. Es ist jedoch genauer approximiert.

In den nächsten zwei Stufen überprüft man, ob das System mit ausreichender Genauigkeit approximiert wurde.

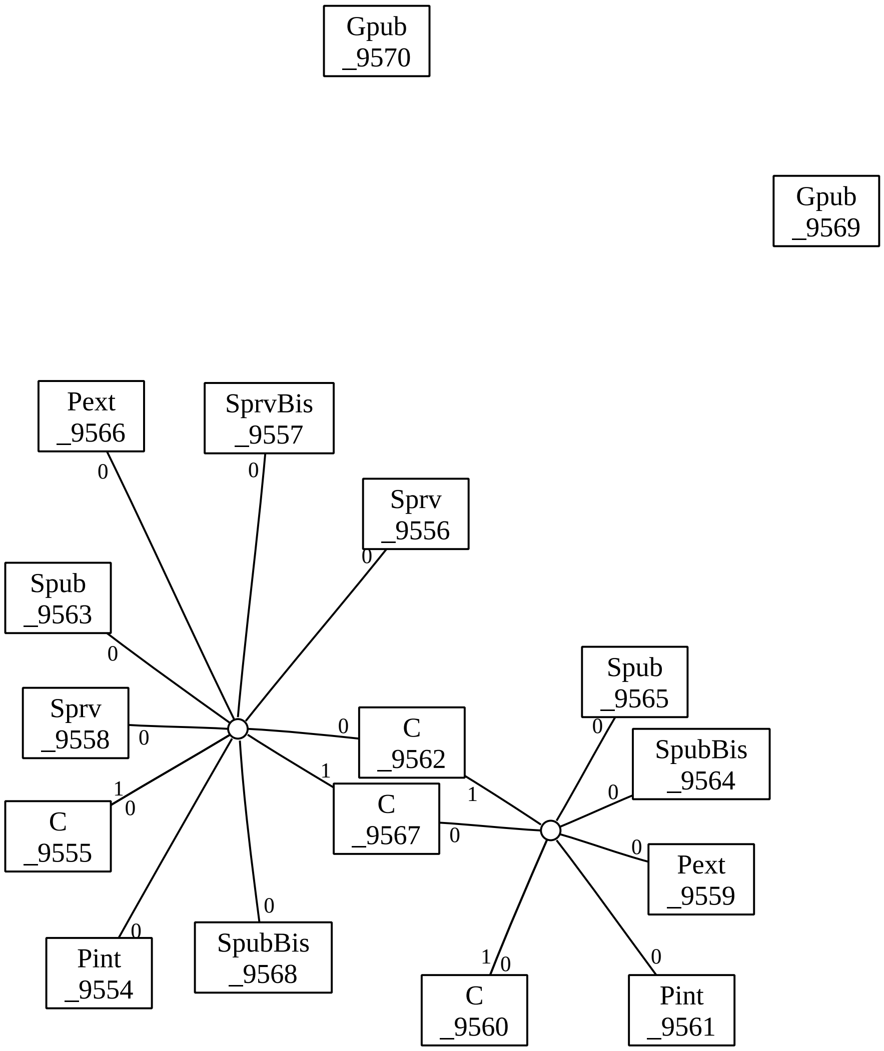


Abbildung 5.4: Öffentliche-Private Server – Version 2 – Hypergraph

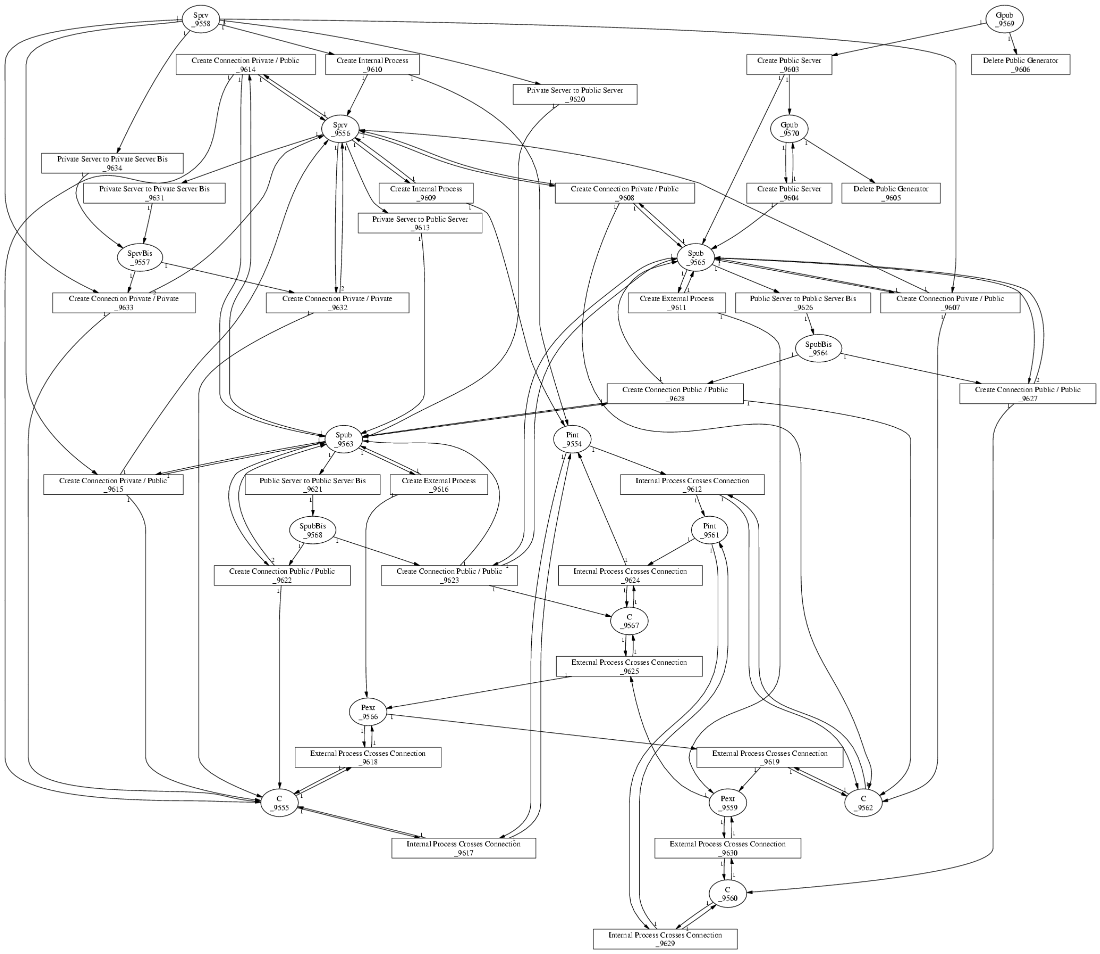


Abbildung 5.5: Öffentliche-Private Server – Version 2 – Petri-Netz

5.2.3 sponge

Die Ausführung von sponge gibt:

```
% sponge -q pub_priv_server2.gxl
'Spub'C*'Sprv'
MARKING Sprv_9556: 1, Spub_9563: 1;
MARKING Sprv_9558: 1, Spub_9563: 1;
MARKING Sprv_9556: 1, Spub_9565: 1, C_9567: 1;
MARKING Sprv_9558: 1, Spub_9565: 1, C_9567: 1;

% sponge -q pub_priv_server2.gxl
'Sprv''Pext'
MARKING Sprv_9556: 1, Pext_9566: 1;
MARKING Sprv_9558: 1, Pext_9566: 1;
```

Daraus erhält man folgende Formeln:

$$
\begin{aligned}
F_{S_{pub}C^*S_{prv}} &= (\#Sprv\_9556 \geq 1 \ \wedge \ \#Spub\_9563 \geq 1) \ \vee \\
& (\#Sprv\_9558 \geq 1 \ \wedge \ \#Spub\_9563 \geq 1) \ \vee \\
& (\#Sprv\_9556 \geq 1 \ \wedge \ \#Spub\_9565 \geq 1 \ \wedge \ \#C\_9567 \geq 1) \ \vee \\
& (\#Sprv\_9556 \geq 1 \ \wedge \ \#Spub\_9565 \geq 1 \ \wedge \ \#C\_9567 \geq 1)
\end{aligned}
$$

$$
\begin{aligned}
F_{S_{prv}P_{ext}} &= (\#Sprv\_9556 \geq 1 \ \wedge \ \#Pext\_9566 \geq 1) \ \vee \\
& (\#Sprv\_9558 \geq 1 \ \wedge \ \#Pext\_9566 \geq 1)
\end{aligned}
$$

5.2.4 bra

```
% pg2lola pub_priv_server2.gxl > pub_priv_server2.lola

% bra -q pub_priv_server2.lola pub_priv_server-markierung
false

% bra -q pub_priv_server2.lola pub_priv_server-markierung2
false
```

Es gibt dann keine erreichbare Markierung, die die Formeln erfüllt.

5.2.5 Schlussbemerkung

Nach der Bemerkung im Abschnitt 3.2.2 auf Seite 26 sind $Reg(G, m, S_{pub}C^*S_{prv})$ und $Reg(G, m, S_{prv}P_{ext})$ für alle mögliche Markierungen falsch.

Nun weiß man, dass eine Verbindung von einem öffentlichen Server zu einem privaten Server im Graphtransformationssytem nie stattfindet. Damit ist es auch nicht möglich, einen externen Prozess in einem privaten Server zu finden.

5.3 Rot-Schwarz-Bäume

5.3.1 Definition

Ein Rot-Schwarz-Baum ist ein binärer Baum mit folgenden Eigenschaften:

- jeder Knoten hat eine Farbe (rot oder schwarz)
- der Wurzelknoten ist schwarz
- alle Blattknoten sind schwarz
- rote Knoten haben nur schwarze Kinderknoten
- alle Pfade von einem Knoten zu seinem Blatt enthalten die gleiche Anzahl von schwarzen Knoten

Abbildung 5.6 zeigt ein Beispiel eines Rot-Schwarz-Baums.

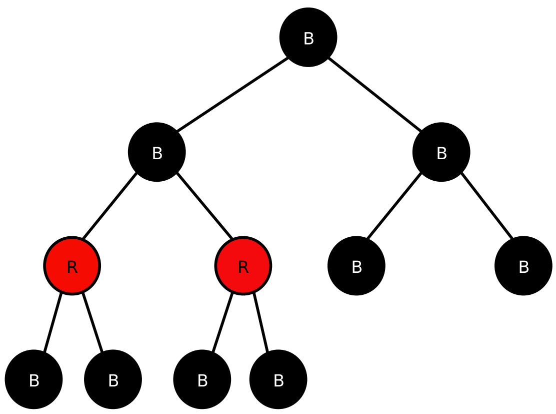


Abbildung 5.6: Ein Rot-Schwarz-Baum

5.3.2 Rot-Schwarz-Bäume als Graphtransformations-System

[Meh84] beschreibt jene Algorithmen, die erlauben die Rot-Schwarz-Baum-Eigenschaften zu erhalten, wenn zum Beispiel ein Knoten hinzugefügt oder gelöscht wird.[1]

Es ist möglich, das dynamische System aller Rot-Schwarz-Bäume mit Hilfe dieser Algorithmen als Graphtransformation-System zu beschreiben.

Einen Auszug von der XML-Datei dieses Systems findet sich im Anhang (siehe A.2 auf Seite 66).

[1] In [Meh84] sind die Kanten anstelle der Knoten farbig: eine schwarze bzw. rote Kante hat einen schwarzen bzw. roten Kinderknoten.

5.3.3 Verifikation des Systems

Man will überprüfen, dass ein roter Knoten nie einen roten Kinderknoten hat. Diese Eigenschaft lässt sich mit Hilfe des folgenden regulären Ausdruck beschreiben:

$$(R + R_x)(R + R_x)$$

wobei R und R_x zwei verschiedenen Typen von roten Knoten sind.

5.3.4 Berechnung

Ein Petri-Graph, der das System approximiert, zuerst wird berechnet:

```
% aunfold -nc rbnew.xml rbnew.gxl
```

Die Abbildungen 5.7 und 5.8 auf der nächsten Seite zeigen die Komplexität des Systems.

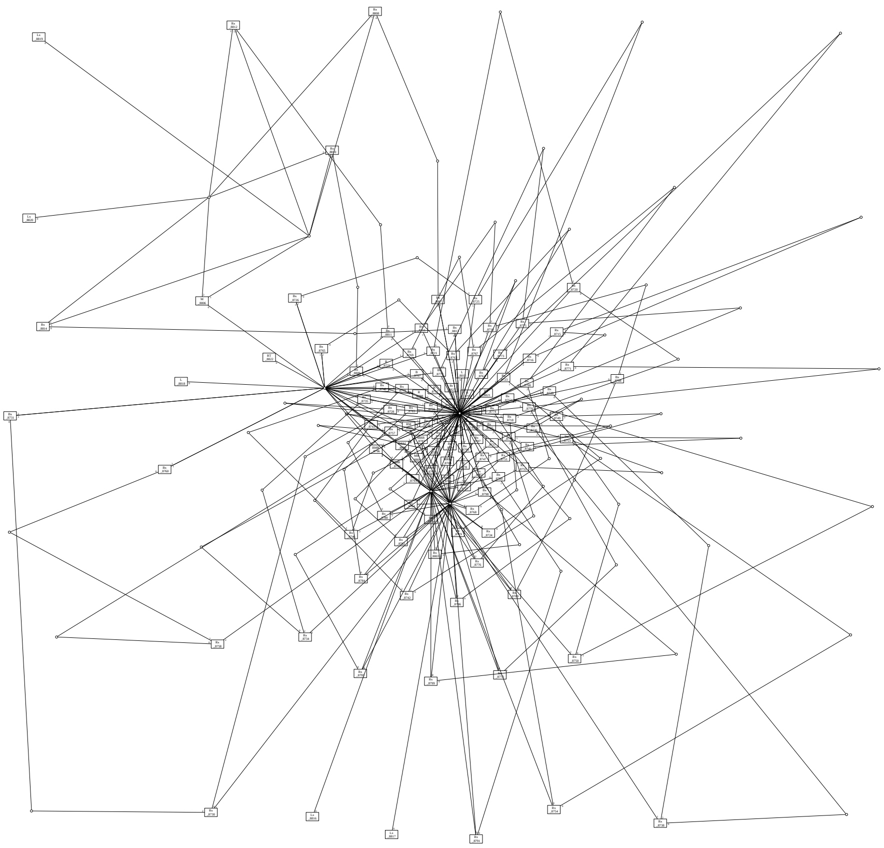


Abbildung 5.7: Rot-Schwarz-Baum – Hypergraph

Die Ausführung von `sponge` ergibt dann:

```
% sponge -q -f rbnew.gxl
(R+'Rx')(R+'Rx')
FORMULA FALSE
```

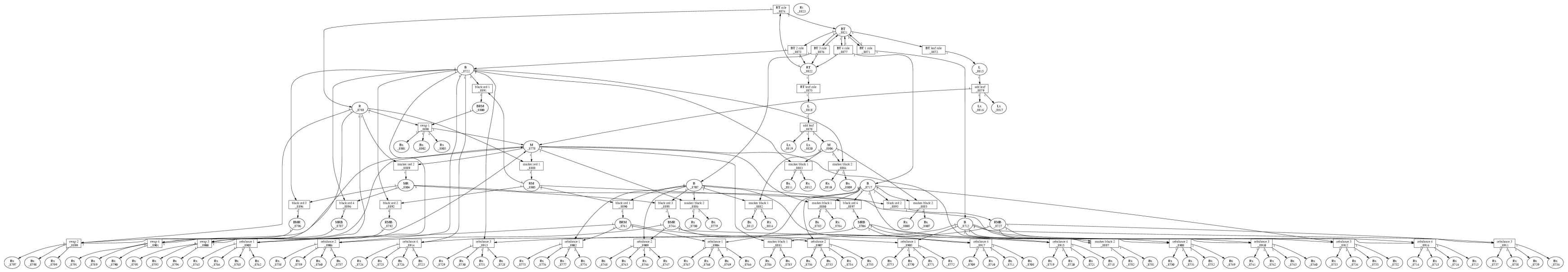


Abbildung 5.8: Rot-Schwarz-Baum – Petri-Netz

5.3.5 Schlussbemerkung

Die verschiedene Algorithmen der Rot-Schwarz-Bäumen behalten die Eigenschaft, dass ein roter Knoten nie einen roten Kinderknoten hat.

Trotz der Komplexität des Systems erlaubt `sponge` hier, diese Eigenschaft schnell zu verifizieren.

Man bemerkt auch, dass der Werkzeug `bra` hier nutzlos war, da bereits die Graphkomponente des Petri-Graphen keinen entsprechenden Subgraphen enthält.

Schlussbemerkungen

Obwohl die Theorie, die dieser Arbeit zugrunde liegt, nicht immer einfach ist, ist der Algorithmus 1 auf Seite 28 nicht kompliziert. Das erlaubt eine effiziente Implementierung. In der Praxis ist die Approximation des Graphtransformations-System durch den Petri-Graphen der kritische Schritt.

Die zwei Beispiele zeigen bereits zwei interessante Anwendungen des Algorithmus. In der Zukunft werden wahrscheinlich noch viele neue Nutzungsmöglichkeiten der Software gefunden.

Die regulären Ausdrücke könnten auch anders behandelt werden. Man könnte überprüfen wollen, dass im dynamischen System der reguläre Ausdruck nie alle Hyperkanten des Graphen beinhaltet. Diese Interpretation der regulären Ausdrücken kann sehr nützlich sein. Sie erlaubt zum Beispiel, zu bestimmen, ob es ein Deadlock im *Dining-Philosophers-Problem* gibt, das in [BCK01] beschrieben wird. In diesem Fall könnten semi-lineare Mengen nützlich sein [Reu90]. Sie behalten nämlich mehr Informationen als die Elemente aus $V^2 \times E^{\oplus}$. Die Pfade, die mehrmals durchgelaufen werden dürfen, könnten mit Hilfe eines Vektors dargestellt werden. Auch der Einsatz des Konzepts der Produktautomation könnte sich in diesem Zusammenhang als nützlich erweisen.

Man kann sich auch den Zusammenhang zu monadischer Logik zweite Stufe überlegen. Ist es möglich, die regulären Ausdrücke in diese Logik überzusetzen? Man könnte dann einen Übersetzungsalgorithmus schreiben, der eine Brücke zwischen dieser Arbeit und [BKK03] schlägt.

Danksagung

Vielen Dank zuerst an meine Betreuerin *Barbara König*, die mir viele gute Ratschlage gegeben hat.

Ich danke auch *Caroline Schmid* und *Sophie Rued*, die meinen Bericht durchgelesen haben.

Sinan Turan, *Vitali Kozioura*, *Ingo Walther* und *Lars Heinemann* haben Werkzeuge entwickelt, ohne die meine Arbeit nutzlos gewesen wäre.

Anhang A

XML-Dateien (Auszüge)

A.1 Öffentliche-Private Server

A.1.1 pub_priv_server.xml (Auszug)

Diese Dateien beschreiben das Graphtransformations-System.

Im ersten Teil wird der Anfangsgraph beschrieben:

```
<?xml version="1.0"?>
<!DOCTYPE gtxl SYSTEM "gtxl_new.dtd">
 <GTS id="Public / Private Servers in mobile systems and processes">
  <Initial>
   <Graph id="Private Server and public generator" edgeids="true" hypergraph="true"
       edgemode="undirected">
    <node id="n" />
    <rel id="Gpub">
     <attr name="label">
      <string>Gpub</string>
     </attr>
    </rel>
    <rel id="Sprv">
     <attr name="label">
      <string>Sprv</string>
     </attr>
     <relend id="spriv" target="n" startorder="0" />
    </rel>
   </Graph>
  </Initial>
```

Dann werden die Regeln definiert. Hier zum Beispiel die Regel, wo ein öffentlicher Server herstellt wird:

```
  <Rule id="Create Public Server">
   <LHS>
    <RuleGraph id="cps_lrg">
     <Graph id="cps_lg" edgeids="true" hypergraph="true" edgemode="undirected">
      <rel id="cps_lh_gpub" >
       <attr name="label">
        <string>Gpub</string>
       </attr>
      </rel>
     </Graph>
    </RuleGraph>
   </LHS>
```

```
<RHS>
  <RuleGraph id="cps_rrg">
    <Graph id="cps_rg" edgeids="true" hypergraph="true" edgemode="undirected">
      <node id="n" />
      <rel id="cps_rh_Gpub">
        <attr name="label">
          <string>Gpub</string>
        </attr>
      </rel>
      <rel id="cps_rh_Spub">
        <attr name="label">
          <string>Spub</string>
        </attr>
        <relend id="cps_rh_spub" target="n" startorder="0" />
      </rel>
    </Graph>
  </RuleGraph>
</RHS>
<Mapping id="cps_mapping">
</Mapping>
</Rule>
```

A.1.2 `pub_priv_server2.gxl` (Auszug)

Diese Datei stellt die Entfaltung dar, die durch `aunfold` mit der Option `-depth=1` geschaffen wurde. Hier ist der Anfang der Datei, mit der der Knoten eines internen Prozesses beschrieben wird:

```
<?xml version="1.0"?>
<gxl>
  <graph id="result" hypergraph="true" edgemode="undirected">
    <node id="_9552">
      <attr name="vertex"/>
    </node>
    <node id="_9553">
      <attr name="vertex"/>
    </node>
    <rel id="_9554">
      <attr name="label">
        <string>Pint</string>
      </attr>
      <attr name="initial_marking">
        <int>0</int>
      </attr>
      <relend target="_9552" role="vertex" startorder="0"/>
      <relend target="_9609" role="postset">
        <attr name="weight">
          <int>1</int>
        </attr>
      </relend>
      <relend target="_9610" role="postset">
        <attr name="weight">
          <int>1</int>
        </attr>
      </relend>
      <relend target="_9612" role="preset">
        <attr name="weight">
          <int>1</int>
        </attr>
      </relend>
      <relend target="_9617" role="preset">
        <attr name="weight">
          <int>1</int>
        </attr>
      </relend>
      <relend target="_9617" role="postset">
```

```
      <attr name="weight">
        <int>1</int>
      </attr>
    </relend>
    <relend target="_9624" role="postset">
      <attr name="weight">
        <int>1</int>
      </attr>
    </relend>
  </rel>
```

Am Ende der Datei gibt es alle Transitionen des Petri-Netzes:

```
  <node id="_9603">
    <attr name="transition"/>
    <attr name="rule">
      <string>Create Public Server</string>
    </attr>
  </node>
  <node id="_9604">
    <attr name="transition"/>
    <attr name="rule">
      <string>Create Public Server</string>
    </attr>
  </node>
  <node id="_9605">
    <attr name="transition"/>
    <attr name="rule">
      <string>Delete Public Generator</string>
    </attr>
  </node>
  <node id="_9606">
    <attr name="transition"/>
    <attr name="rule">
      <string>Delete Public Generator</string>
    </attr>
  </node>
  <node id="_9607">
    <attr name="transition"/>
    <attr name="rule">
      <string>Create Connection Private / Public</string>
    </attr>
  </node>
  <node id="_9608">
    <attr name="transition"/>
    <attr name="rule">
      <string>Create Connection Private / Public</string>
    </attr>
  </node>
  <node id="_9609">
    <attr name="transition"/>
    <attr name="rule">
      <string>Create Internal Process</string>
    </attr>
  </node>
  <node id="_9610">
    <attr name="transition"/>
    <attr name="rule">
      <string>Create Internal Process</string>
    </attr>
  </node>
  <node id="_9611">
    <attr name="transition"/>
    <attr name="rule">
      <string>Create External Process</string>
    </attr>
  </node>
  <node id="_9612">
```

```
  <attr name="transition"/>
  <attr name="rule">
    <string>Internal Process Crosses Connection</string>
  </attr>
</node>
<node id="_9613">
  <attr name="transition"/>
  <attr name="rule">
    <string>Private Server to Public Server</string>
  </attr>
</node>
<node id="_9614">
  <attr name="transition"/>
  <attr name="rule">
    <string>Create Connection Private / Public</string>
  </attr>
</node>
<node id="_9615">
  <attr name="transition"/>
  <attr name="rule">
    <string>Create Connection Private / Public</string>
  </attr>
</node>
<node id="_9616">
  <attr name="transition"/>
  <attr name="rule">
    <string>Create External Process</string>
  </attr>
</node>
<node id="_9617">
  <attr name="transition"/>
  <attr name="rule">
    <string>Internal Process Crosses Connection</string>
  </attr>
</node>
<node id="_9618">
  <attr name="transition"/>
  <attr name="rule">
    <string>External Process Crosses Connection</string>
  </attr>
</node>
<node id="_9619">
  <attr name="transition"/>
  <attr name="rule">
    <string>External Process Crosses Connection</string>
  </attr>
</node>
<node id="_9620">
  <attr name="transition"/>
  <attr name="rule">
    <string>Private Server to Public Server</string>
  </attr>
</node>
<node id="_9621">
  <attr name="transition"/>
  <attr name="rule">
    <string>Public Server to Public Server Bis</string>
  </attr>
</node>
<node id="_9622">
  <attr name="transition"/>
  <attr name="rule">
    <string>Create Connection Public / Public</string>
  </attr>
</node>
<node id="_9623">
  <attr name="transition"/>
  <attr name="rule">
```

```
      <string>Create Connection Public / Public</string>
    </attr>
  </node>
  <node id="_9624">
    <attr name="transition"/>
    <attr name="rule">
      <string>Internal Process Crosses Connection</string>
    </attr>
  </node>
  <node id="_9625">
    <attr name="transition"/>
    <attr name="rule">
      <string>External Process Crosses Connection</string>
    </attr>
  </node>
  <node id="_9626">
    <attr name="transition"/>
    <attr name="rule">
      <string>Public Server to Public Server Bis</string>
    </attr>
  </node>
  <node id="_9627">
    <attr name="transition"/>
    <attr name="rule">
      <string>Create Connection Public / Public</string>
    </attr>
  </node>
  <node id="_9628">
    <attr name="transition"/>
    <attr name="rule">
      <string>Create Connection Public / Public</string>
    </attr>
  </node>
  <node id="_9629">
    <attr name="transition"/>
    <attr name="rule">
      <string>Internal Process Crosses Connection</string>
    </attr>
  </node>
  <node id="_9630">
    <attr name="transition"/>
    <attr name="rule">
      <string>External Process Crosses Connection</string>
    </attr>
  </node>
  <node id="_9631">
    <attr name="transition"/>
    <attr name="rule">
      <string>Private Server to Private Server Bis</string>
    </attr>
  </node>
  <node id="_9632">
    <attr name="transition"/>
    <attr name="rule">
      <string>Create Connection Private / Private</string>
    </attr>
  </node>
  <node id="_9633">
    <attr name="transition"/>
    <attr name="rule">
      <string>Create Connection Private / Private</string>
    </attr>
  </node>
  <node id="_9634">
    <attr name="transition"/>
    <attr name="rule">
      <string>Private Server to Private Server Bis</string>
    </attr>
```

```
    </node>
  </graph>
</gxl>
```

A.2 Rot-Schwarz-Bäume

Hier der Anfangsgraph und die erste Regel des Graphtransformations-Systems, das Rot-Schwarz-Bäume erzeugt:

```
<?xml version="1.0"?>
<!DOCTYPE gtxl SYSTEM "gtxl_new.dtd">

  <!-- Specification of red-black trees consisting of a context-free
       grammar generating all possible (unbalanced red-black trees) and
       a context-sensitive grammar describing the addition of a new leaf -->

  <GTS id="Context-free grammar generating red-black trees">

    <!-- Start graph: the root node is marked with a unary hyperedge labelled Rt -->

    <Initial>
        <Graph id="Start graph" edgeids="true" hypergraph="true" edgemode="undirected">
           <node id="v1" />
           <rel id="BT">
              <attr name="label">
                 <string>BT</string>
              </attr>
            <relend id="BT_relend_1" target="v1" startorder = "0" />
           </rel>
           <rel id="Rt">
              <attr name="label">
                 <string>Rt</string>
              </attr>
            <relend id="BT_relend_1" target="v1" startorder = "0" />
           </rel>
        </Graph>
    </Initial>

   <!-- CFG generating the set of all red-black trees including all
        unbalanced ones.
        Condition: there must not be two subsequent red nodes. -->

    <!-- First rule of CFG: non-terminal BT is expanded into a black node
         with two black trees as subtrees -->

    <Rule id="BT 1 rule">
       <LHS>
           <RuleGraph id="BT 1 rule Left Hand RuleGraph">
             <Graph id="BT 1 rule lefthandGraph" edgeids="true" hypergraph="true" edgemode="
                 undirected">
               <node id="r1_lhs_1" />
               <rel id="r1_BT">
                  <attr name="label">
                     <string>BT</string>
                  </attr>
                  <relend id="BT_relend_1" target="r1_lhs_1" startorder = "0" />
               </rel>
             </Graph>
           </RuleGraph>
       </LHS>
       <RHS>
           <RuleGraph id="BT 1 rule Right Hand RuleGraph">
             <Graph id="BT 1 rule righthandGraph" edgeids="true" hypergraph="true" edgemode=
                 "undirected">
```

```
        <node id="r1_rhs_1" />
        <node id="r1_rhs_2" />
        <node id="r1_rhs_3" />
        <rel id="r1_B">
          <attr name="label">
            <string>B</string>
          </attr>
          <relend id="B_relend_1" target="r1_rhs_1" startorder = "0" />
          <relend id="B_relend_2" target="r1_rhs_2" startorder = "1" />
          <relend id="B_relend_3" target="r1_rhs_3" startorder = "2" />
        </rel>
        <rel id="r1_BT1">
          <attr name="label">
            <string>BT</string>
          </attr>
          <relend id="BT1_relend_1" target="r1_rhs_2" startorder = "0" />
        </rel>
        <rel id="r1_BT2">
          <attr name="label">
            <string>BT</string>
          </attr>
          <relend id="BT2_relend_1" target="r1_rhs_3" startorder = "0" />
        </rel>
      </Graph>
    </RuleGraph>
  </RHS>
  <Mapping id="r1_mapping">
    <MapElem from="r1_lhs_1" to="r1_rhs_1" />
  </Mapping>
</Rule>
```

Anhang B

man page of sponge

NAME

sponge – return markings from a hypergraph and a regular expression

SYNTAX

sponge [–q | –d] [–m] [–f] *hypergraph_file.xml*
sponge —-help
sponge —-version

DESCRIPTION

Given an Hypergraph and a regular expression, this program finds some conditions on the markings of the hypergraph, which ensure that a path in the hypergraph matches the regular expression.

The **regular expression** is given in *stdin*. The regular expression must be given according the following grammar:

reg_exp ::= reg_exp + term /* union */
term ::= term factor /* concatenation */
factor ::= **(** reg_exp **)**
| factor ***** /* kleene–closure */
| epsilon
| motif
epsilon ::= **~** | /* epsilon */
motif ::= char
| **'** word **'**
| digit word digit
| **'** number word number **'**
word ::= char word
| char
char ::= [**A**–**Za**–**z** **_**]
number ::= digit number
| digit
digit ::= [**0**–**9**]

Note: the following delimiters: space, tabulation, newline, are not taken in account.

The **numbers** and the **modes** (see the –m option below) allow to specify how to read the labels on hyperedges.
If numbers are specified around the label, the first number stands for the incoming vertex, the second number for the outgoing vertex. For example, *0a2* means you only take in consideration the paths that go from the vertex 0 to the vertex 2 in the hyperedges that are labelled *a*.
If no number is specified, the meaning of the hyperedge depends on the number of vertices:
0 vertex: the hyperedge behaves like an isolated binary edge, that means that *a* and *0a0* return the same result.
1 vertex: the hyperedge is like a binary edge that connects the same vertex. Thus *a* is the

same as *0a0*.
2 vertices: this is a binary edge and *a* means *0a1*.
> **2 vertices:** the behaviour of the hyperedge depends on the used mode. In **mode 0**, the default mode, the hyperedge connects the first vertex 0 to all the vertices, whose number i is strictly positive (i > 0). In **mode 1**, which is switched on with the –**m** option, the hyperedge connects the vertices i with all vertices j > i.

The hypergraph file *hypergraph_file.xml* is written in GXL (Graph eXchange Language), an XML format to describe graphs (see http://www.gupro.de/GXL/).
The output of the **aunfold** program is in the right format (a mix of hypergraph and Petri net).

The result of the program, a condition on markings, is then given in *stdout*.
Two different output formats are possible:

Default output: A list of minimal markings so that the regular expression could match the hypergraph. This is the output that is appropriate for the program **bra**.

Formula output (–**f** option): This output, that can be switched on with the –**f** option. The output is then in the formula format as described in the documentation of LoLA (see http://www.informatik.hu–berlin.de/˜kschmidt/lola.html).

OPTIONS

–**q** quiet mode, disable verbose output

–**d** debug mode, show debugging information

–**m** mode 1 for hypergraph: from all i to j>i (default: mode 0, from 0 to j>0)

–**f** switch to formula output (default: markings output)

—**-help** Output help information and exit.

—**-version** Output version information and exit.

EXAMPLES

To run this program the standard way type:

sponge < regular_expression_file > output_file hypergraph_file.xml

If you do not want any verbose messages, use the –**q** option (here is the output displayed on the screen instead of being written into a file):

sponge –q < regular_expression_file hypergraph_file.xml

Alternatively it is also possible to enter the regular expression in an interactive way:

sponge –q hypergraph_file.xml

AUTHORS

Nicolas Relange <relangns@studi.informatik.uni–stuttgart.de>
Ingo Walther
Lars Heinemann
Barbara König <koenigba@fmi.uni–stuttgart.de>

SEE ALSO

aunfold(1) bra(1)

Literaturverzeichnis

[AJKP98] Parosh Aziz Abdulla, Bengt Jonsson, Mats Kindahl, and Doron Peled. A general approach to partial order reductions in symbolic verification. In *Proc. of CAV '98*, pages 379–390. Springer, 1998. LNCS 1427.

[BCK01] Paolo Baldan, Andrea Corradini, and Barbara König. A static analysis technique for graph transformation systems. In *Proc. of CONCUR '01*, pages 381–395. Springer-Verlag, 2001. LNCS 2154.

[BK02] Paolo Baldan and Barbara König. Approximating the behaviour of graph transformation systems. In *Proc. of ICGT '02 (International Conference on Graph Transformation)*, volume 2505 of *LNCS*, pages 14–29. Springer-Verlag, 2002. LNCS 2505.

[BKK03] Paolo Baldan, Barbara König, and Bernhard König. A logic for analyzing abstractions of graph transformation systems. In *Proc. of SAS '03 (International Static Analysis Symposium)*, volume 2694 of *LNCS*, pages 255–272. Springer-Verlag, 2003.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL '77 (Los Angeles, California)*, pages 238–252. ACM, 1977.

[CGL99] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 1999.

[Cou96] Patrick Cousot. Abstract interpretation. *ACM Computing Surveys*, 28(2), 1996.

[Meh84] Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching.* EATCS Monographs on Theoretical Computer Science. Springer, 1984.

[NNH99] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis.* Springer-Verlag, 1999.

[Rei85] Reisig. *Petri Nets, EATCS Monographs on Theoretical Computer Science.* Springer, 1985.

[Reu90] Christophe Reutenauer. *The Mathematics of Petri Nets.* Prentice-Hall, 1990.

[Tur04] Sinan Turan. Effiziente Berechnung der Überdeckbarkeit bei Petri-Netzen, Juni 2004. Studienarbeit, Universität Stuttgart.

Index

$(A_n)_{n\in\mathbb{N}}$, 27
$A_G(r)$, 27
A_∞, 36
E, 37
$E^\oplus$, 9
$P_G(r)$, 18
$Reg'(G, r)$, 13
$Reg(G, m, r)$, 14
$\mathbb{N}_n$, 13
$\Pi_G(r)$, 19
$\boxplus$, 36
μ, 22
$\preceq$, 21
$\sqsubseteq$, 22
$\trianglelefteq$, 37
f_G, 22
$graph_G(m)$, 11
`aunfold`, 2, 9, 43, 48, 53
`bra`, 2, 47
`sponge`, 2, 47, 51, 53

abstrakte Interpretation, 1
Algorithmus, 27, 28
Approximation, 9, 11, 48

dining philosophers, 57
dynamisches System, 1

erreichbarer Graph, 11, 14

Formel, 13, 19, 25, 47, 51

Graph, 2
Graphmorphismus, 5, 11, 14
Graphtransformations-System, 2, 5, 43
GTXL, 43
GXL, 43

Hypergraph, 5, 15, 45, 49, 53
Hyperkante, 15

Kante, 2
kanten-bijektiv, 5
Kleene'sche Hülle, 33
Knoten, 2
Korrektheit, 29, 36

Label, 2
LoLA, 47

Markierung, 9, 11
minimale Elemente, 22
minimale Überdeckung, 21, 22
Mode, 17
Model-Checking, 1
Modellierung, 1
monadische Logik 2. Stufe, 57

Öffentliche-Private Server, 5, 22, 43
Ordnung, 21, 22, 37

Petri-Graph, 9
Petri-Netz, 9, 46, 50, 54
Programmanalyse, 1

Regel, 2, 5
regulärer Ausdruck, 5, 13, 15
Rot-Schwarz-Baum, 52

Stationär, 35

Terminierung, 41

verbotener Pfad, 7
Verifikation, 1

Zustand des Systems, 13

Declaration

I hereby declare that this thesis was composed by myself, and that I have used no sources other than those cited in the thesis.

(Nicolas Relange)