Model-Checking II

(WS 2008/09)

Stefan Schwoon

Institut für Informatik (I7)
Technische Universität München

Organisatorisches

Vorlesung: Montag, 14:15–15:45, 03.11.058

Dozent: Stefan Schwoon, `schwoon@in.tum.de`

Sprechstunde nach Vereinbarung, MI 03.011.053

Stunden: 2V bzw. 3 ECTS-Credits

Prüfung am Ende des Semesters (mündlich nach Vereinbarung)

Ankündigungen

Webseite:

`http://www7.in.tum.de/um/courses/mc2/ws0809/`

dort auch Folien etc.

E-Mails:

für kurzfristige Ankündigungen (Terminänderungen etc.)

Sonstiges

Voraussetzungen:

Grundkenntnisse in Logik, Diskrete Strukturen, Graphen, . . .

Automaten- und Komplexitätstheorie hilfreich

Kenntnisse aus Vorlesung Model-Checking I ebenfalls hilfreich

Literatur:

verschiedene Papiere, werden abschnittsweise angegeben

Teil 1: Einführung

Einige Fakten

Softwareentwickler verbringen zwischen 50% und 70% ihrer Arbeitszeit mit Testen und Validierung von Code.

Trotzdem werden mangelnde Zuverlässigkeit und Sicherheit heute oft als das grösste Problem der Softwareindustrie betrachtet.

Eine Studie von NIST hat die Kosten der durch unzuverlässige Software verursachten Schäden auf 60 Milliarden Dollar pro Jahr beziffert.

Einige Fakten

Softwareentwickler verbringen zwischen 50% und 70% ihrer Arbeitszeit mit Testen und Validierung von Code.

Trotzdem werden mangelnde Zuverlässigkeit und Sicherheit heute oft als das grösste Problem der Softwareindustrie betrachtet.

Eine Studie von NIST hat die Kosten der durch unzuverlässige Software verursachten Schäden auf 60 Milliarden Dollar pro Jahr beziffert.

> Wenn das Automobil dieselbe Entwicklung wie der Computer genommen hätte, würde ein Rolls-Royce heute 100 Dollar kosten, eine Million Meilen pro Liter fahren und einmal im Jahr explodieren, wobei alle Insassen ums Leben kommen.
>
> *Robert Cringely*

Model-Checking

Eine Technik zur Verifikation von (Hardware- oder Software-) Systemen

Gegeben: System, Spezifikation
Gefragt: Erfüllt das System die Spezifikation?

Abgrenzung von MC gegenüber anderen Techniken:

- Erforschung des Zustandsraums des Systems

- automatisch / algorithmische Lösungen

Rückblick auf Model Checking I

(was bisher geschah. . .)

System: Gerichteter, gefärbter Graph (Knoten = Zustände)

Lineare Sichtweise: Zähle alle Sequenzen von Zuständen auf

Baumartige Sichtweise: Betrachte die azyklische Entfaltung

Spezifikation: Formel in einer Logik von Sequenzen oder Bäumen

Linear-Zeit-Logik, z.B. “Enthält jede Sequenz einen roten Zustand?”

Baum-Zeit-Logik, z.B., “Gibt es an jedem Punkt eine Verzweigung, mit der man zum Anfangszustand zurückkommt?”

Rücklick

In MC1: Graphen mit endlich vielen Zuständen

Entscheidungsalgorithmen für lineare und baumartige Logiken:
in linearer Zeit in der Größe des Graphen

Optimierungsalgorithmen zur Behandlung von Zustandsexplosion

Ausblick: (was geschehen wird...)
Analyse von Systemen mit unendlich vielen Zuständen

Gründe für unendlich viele Zustände

Daten: ganze Zahlen, Listen, Bäume, Zeigerstrukturen, ...

Kontrolle: Prozeduren, dynamische Prozesserzeugung, ...

Asynchrone Kommunikation: unbeschränkte FIFO-Kanäle

Unbekannte Parameter: Zahl von Protokoll-Teilnehmern, ...

Echtzeit: diskrete oder stetige Zeit

Viele dieser Probleme treten eher bei Software als bei Hardware auf!

Hardware- und Software-Verifikation

Hardware-Verifikation:

Erfolgreiche Anwendung in der Industrie, Verifikation sehr großer Designs, z.B. arithmetisch-logische Einheit des Pentium-Prozessors.

Die meisten großen Hardware-Firmen beschäftigen Forschungsgruppen zum Thema Verifikation (IBM, Intel, Motorola, Siemens).

Software-Verifikation:

noch in den “Kinderschuhen”

Qualitätssicherung in der Praxis durch Testen, Software-Engineering-Ansätze

Warum ist Software-Verifikation nicht so weit entwickelt wie Hardware-Verifikation?

Wirtschaftliche Gründe:

Hoher Aufwand: nur für große Firmen wirtschaftlich lohnend.

Software lässt sich auch nach der Auslieferung noch ändern; Kosten eines Fehler (in der Regel) geringer.

Technische Gründe:

Software-Verifikation ist inhärent schwieriger: unendliche Zustandsräume

Problem 1: Entscheidbarkeit

Beispiel für System mit unendlichem Zustandsraum: Turing-Maschine

Jede (nicht-triviale) Spezifikation unentscheidbar.

Ansatz: Finde Automatenmodelle mit folgenden Eigenschaften:

- nicht Turing-mächtig, entscheidbare Model-Checking-Probleme
- mächtig genug, um interessante Features abzubilden

  Beispiele: endliche Automaten (endlich viele Zustände), Kellerautomaten, "exotischere" Modelle, ...

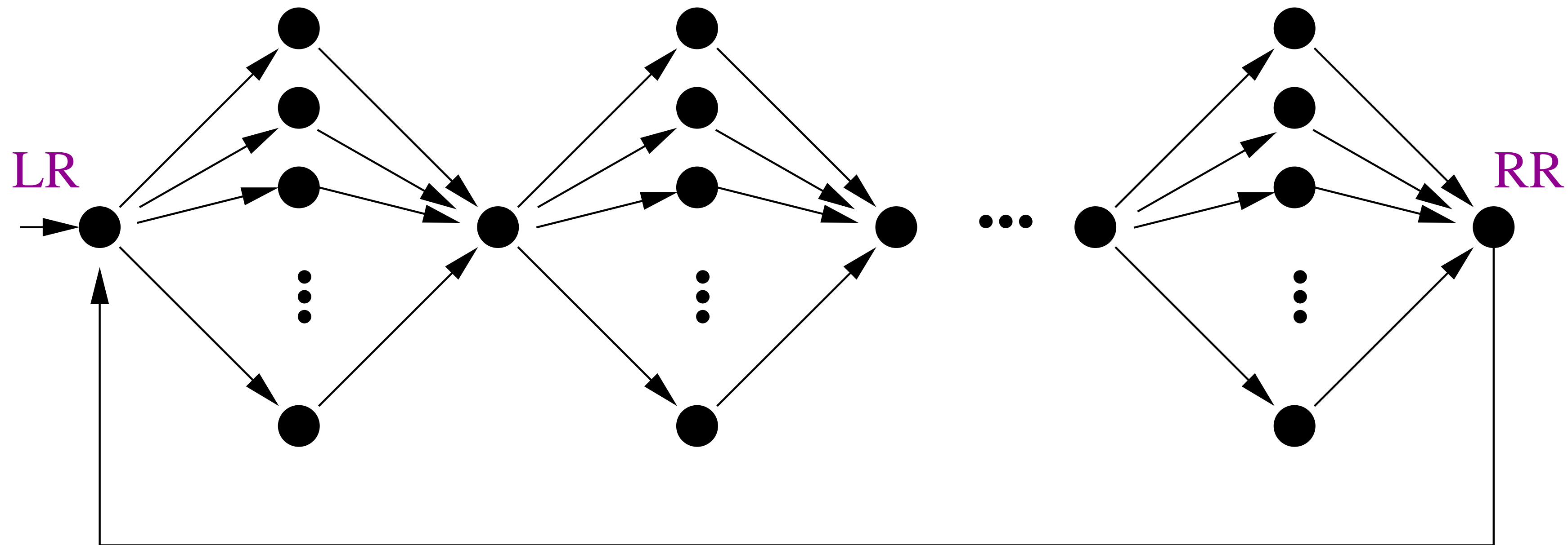
$\longrightarrow$ Automatentheorie

“Bescheidenerer” Ansatz als endlichen Systemen

- Optimierungen sind kein so großes Thema
- stärkerer Fokus auf Entscheidbarkeit
- Einordnung in Komplexitätsklassen (NP, PSPACE, EXPTIME, . . .)
- Betrachtung einfacherer Analyseprobleme (Erreichbarkeit)

Entscheidbarkeitsgrenzen bei unendlichen Systemen sind diffiziler als bei endlichen

Beispiel: “NFA + LTL = LBM”



Endliches System, aber lineare Logik kann korrekte Abläufe einer Turing-Maschine simulieren (siehe MC1, Abschnitt 7)

⇒ Analyse platz-beschränkter Turing-Maschinen

Unendliche Systeme + Logik: womöglich Analyse allgemeiner Turing-Maschinen

Problem 2: Darstellung von Zustandsmengen

Aufgabe: Darstellung einer (unendlichen) Menge von Zuständen,
z.B. die erreichbaren Zustände

Nicht möglich durch Aufzählung oder andere explizite Techniken.

Symbolische Darstellungen erforderlich:

Ungleichungen, endliche Automaten, Baumautomaten, ...

Beispiel 1: Echtzeit, Ungleichungen

Annahme: Es gibt mehrere Stoppuhren. Erlaubte Operationen auf Uhren: zurücksetzen, abfragen, ob bestimmte Zeit verstrichen ist.

Eine vereinfachte Version von Fischers Mutex-Protokoll:

```
              var v:{0,1,2} init 0;

a0: if v = 0 then reset c1;         b0: if v = 0 then reset c2;
     delay < D;                          delay < D;
a1: reset c1; v = 1;                b1: reset c2; v = 2;
     delay > D;                          delay > D;
a2: if v = 1 then goto cs1          b2: if v = 2 then g oto cs2
```

Regionen: endliche Partitionierung des Zustandsraums

Zwei Zeitvektoren $\vec{t} = (t_1, \ldots, t_n)$ und $\vec{u} = (u_1, \ldots, u_n)$ eines Zeitautomaten sind zeitäquivalent ($\vec{t} \approx \vec{u}$), wenn sie dieselben Bedingungen der Form

$$c_i \leq m \qquad \text{und} \qquad c_i - c_j \leq m$$

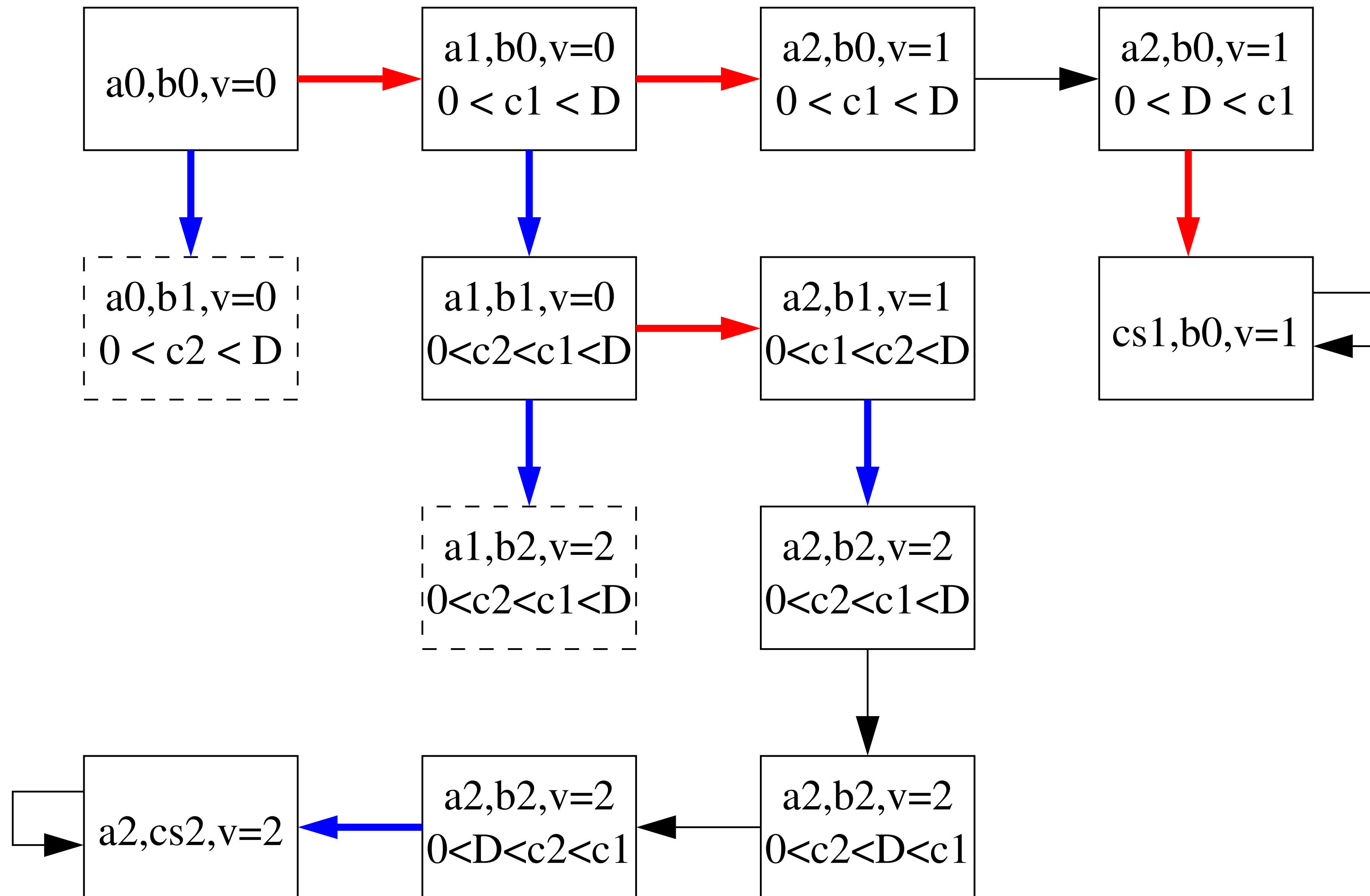
erfüllen für jedes m, das kleiner oder gleich der maximalen Verzögerung in der syntaktischen Beschreibung des Automaten ist (wir nehmen an, dass Verzögerungen ganze Zahlen sind).

Zwei Zustände $\langle q, \vec{t} \rangle$ und $\langle q', \vec{u} \rangle$ sind zeitäquivalent, wenn $q = q'$ und $\vec{t} \approx \vec{u}$

Äquivalenzklassen zeitäquivalenter Zustände werden Regionen genannt

Satz [Alur, Dill 90]: Die Anzahl dieser Äquivalenzklassen ist endlich.

Regionengraph von Fischers Protokoll (eine Hälfte)



Beispiel 2: Rekursive Programme

```
void m() {
    if (?) {
        s(); right();
        if (?) m();
    } else {
        up(); m(); down();
    }
}
```

```
void s() {
    if (?) return;
    up(); m(); down();
}

main() {
  s();
}
```

Modell: Kellerautomat

```
void s() {
```

Regeln im Kellerautomaten

s_0: `if (?) return;`	$s_0 \rightarrow \varepsilon$ `bzw.` $s_0 \rightarrow s_2$
s_2: `up();`	$s_2 \rightarrow up_0\ s_3$
s_3: `m();`	$s_3 \rightarrow m_0\ s_4$
s_4: `down();`	$s_4 \rightarrow down_0\ s_5$
s_5: `return;`	$s_5 \rightarrow \varepsilon$

```
}
```

Symbolische Erreichbakeit für Kellerautomaten

Zustände sind Wörter w, d.h. Folgen von Sprungmarken (Stack)

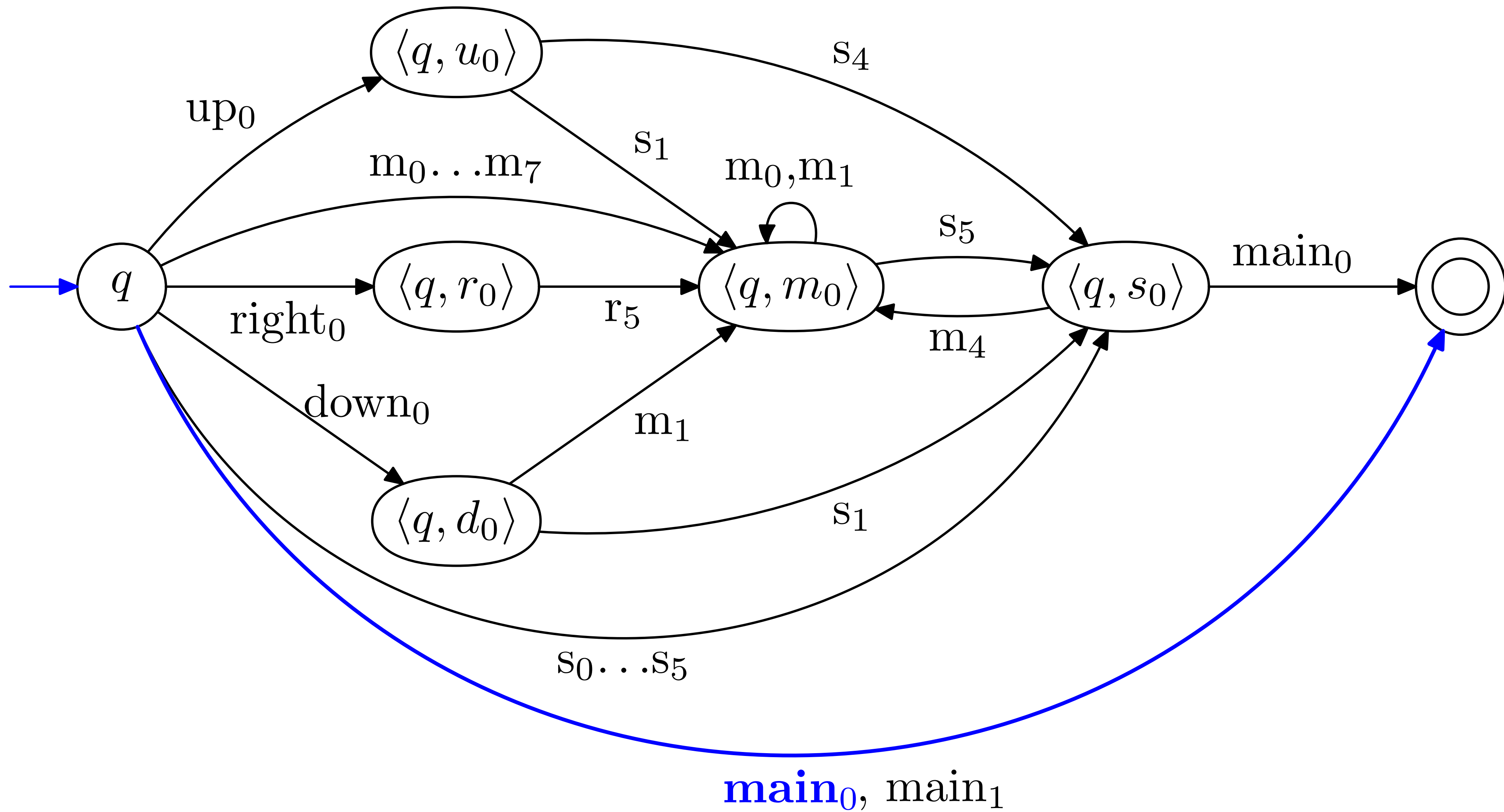
Idee: (unendliche) reguläre Mengen von Wörtern können durch endliche Automaten beschrieben werden

Nützliche Eigenschaft:

Ist C eine reguläre Menge, so auch $post^*(C)$. [Büchi 64]

$post^*(C)$ kann durch automatentheoretische Operationen berechnet werden.

Erreichbare Zustände des Skyline-Beispiels



Problem 3: Ausdruckskraft

Problemstellung: Gegeben zwei Automatenmodelle, haben sie die gleiche (oder unterschiedliche) Ausdruckskraft?

Beispiele:

- Ist ein gegebenes Modell Turing-mächtig?
- Ist ein gegebenes Modell so mächtig wie ein bekanntes Modell, für das Entscheidungsalgorithmen bekannt sind?

Analogie aus den Formalen Sprachen:

- NFA und DFA sind gleichmächtig, d.h. akzeptieren die gleichen Sprachen
- NFA sind weniger mächtig als PDA

Hier: Interesse an Verhaltensweisen, nicht an Sprachen
Hilfsmittel: Bisimulation

Bisimulation (informell)

Gegeben: zwei Systeme mit Zustandsräumen S bzw. T
Zustände können unterschiedlich gefärbt sein

Frage: Existiert eine Relation $R \subset S \times T$, so dass:

jedes Paar $(s, t) \in R$ die gleichen Farben hat

das Paar der Anfangszustände in R ist

wenn $(s, t) \in R$ und $s \to s'$, dann existiert $t \to t'$ mit $(s', t') \in R$,
und umgekehrt

Falls R existiert, heißen die Systeme bisimilar.

Variation: S, T können "unsichtbare" Zustände haben, Übergang $t \to t'$ darf über unsichtbare Zustände gehen.

Bisimulation in der Automatentheorie

Vergleich zweier Automatenklassen $\mathcal{A}$ und $\mathcal{B}$, z.B.:

Gibt es für jeden Automaten aus $\mathcal{A}$ einen bisimilaren Automaten aus $\mathcal{B}$? (und umgekehrt?)

Beispiel: Präfix-Wortersetzungs-Systeme

Gemeinsamer Zustandsraum: Σ^* für irgendein Alphabet Σ

Regeln der Form $u \to v$ mit $u, v \in \Sigma^*$ mit der Bedeutung:

uw kann in vw umgeschrieben werden für jedes $w \in \Sigma^*$

$\mathcal{A}$ sei die Menge aller Präfix-Wortersetzungs-Systeme;
$\mathcal{B}$ ist die Menge derjenigen, wo $|u| \leq 2$ und $|v| \leq 3$ bei allen Regeln.

Jedes System aus $\mathcal{A}$ ist bisimilar zu einem aus $\mathcal{B}$!
(gilt nicht, falls $|u|, |v|$ weiter beschränkt werden!)

Teil 2: Grundlegende Konzepte

Transitionssysteme

Als allgemeinstes Modell benutzen wir den Begriff des Transitionssystems:

$$\mathcal{T} = (S, \rightarrow, r)$$

S	$\widehat{=}$	Zustände ("states"), die das System annehmen kann (endliche oder unendliche Menge)
$\rightarrow \subseteq S \times S$	$\widehat{=}$	Transitionsrelation; beschreibt, welche Aktionen bzw. Zustandsübergänge möglich sind
$r \in S$	$\widehat{=}$	Anfangszustand ("root")

Kripke-Strukturen

Idee: “Farbige” Transitionssysteme $\mathcal{K} = (S, \rightarrow, r, C, \nu)$

$(S, \rightarrow, r)$ $\widehat{=}$ zugrundeliegendes Transitionssystem

C $\widehat{=}$ Menge von “Farben”

$\nu : S \rightarrow C$ $\widehat{=}$ Färbung der Zustände

Intuition:

Zustände mit unterschiedlicher Farbe haben unterschiedliche (Kombinationen von) Eigenschaften
(z.B. normale Zustände schwarz, Fehlerzustände rot)

Wahl der Farben hängt von der zu untersuchenden Eigenschaft ab

In MC1: Farben $\widehat{=}$ Belegung von Grundaussagen

Notation für Transitionssysteme

Wir schreiben $s \longrightarrow t$, falls $(s, t) \in \longrightarrow$ gilt.

Falls $s \longrightarrow t$, ist s direkter Vorgänger von t und t direkter Nachfolger von s.

S^* bezeichnet die *endlichen*, S^ω die *unendlichen* Sequenzen (Wörter) über S.

$w = s_0 \ldots s_n$ ist ein Pfad der Länge n, falls $s_i \longrightarrow s_{i+1}$ für alle $0 \leq i < n$.

$\rho = s_0 s_1 \ldots$ ist ein unendlicher Pfad, falls $s_i \longrightarrow s_{i+1}$ für alle $i \geq 0$.

Notation für Transitionssysteme II

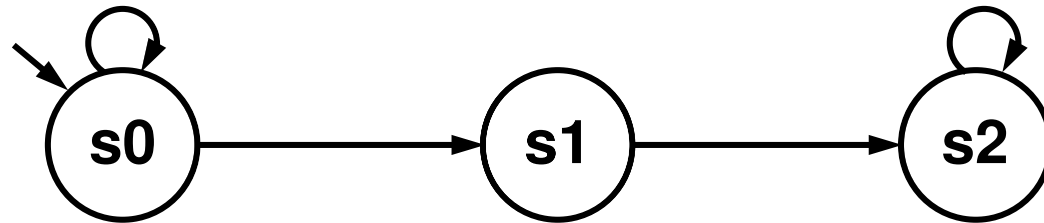
$\rho(i)$ sei das i-te Element von ρ und ρ^i der bei $\rho(i)$ beginnende Suffix.

$s \longrightarrow^* t$, falls ein Pfad von s nach t existiert.

$s \longrightarrow^+ t$, falls ein Pfad von s nach t mit positiver Länge existiert.

Falls $s \longrightarrow^* t$, ist s Vorgänger von t und t Nachfolger von s.

Beispiel



$S = \{s_0, s_1, s_2\}$;   Anfangszustand s_0

$s_0 \longrightarrow s_0$ $s_0 \longrightarrow s_1$ $s_1 \longrightarrow s_2$ $s_2 \longrightarrow s_2$

$s_0 s_1 s_2$ ist ein Pfad der Länge 2, d.h. $s_0 \longrightarrow^* s_2$ und $s_0 \longrightarrow^+ s_2$

$s_1 \longrightarrow^* s_1$, aber $s_1 \not\longrightarrow^+ s_1$

$\rho = s_0 s_0 s_1 s_2 s_2 s_2 \ldots$ ist ein unendlicher Pfad.

$\rho(2) = s_1$ $\rho^1 = s_0 s_1 s_2 s_2 s_2 \ldots$

Lineare und baumartige Sichtweise

Sei $\mathcal{K} = (S, \rightarrow, r, C, \nu)$ eine Kripke-Struktur.

In der linearen Sichtweise interessieren wir uns für die Farbsequenzen in $\mathcal{K}$.

Sei $\rho \in S^\omega$, dann sei $\nu(\rho) \in C^\omega$ definiert durch:

$$\nu(\rho)(i) = \nu(\rho(i)) \text{ für alle } i \geq 0$$

Wir definieren $\mathcal{L}_\mathcal{K} := \{\, \nu(r\rho) \mid r\rho \in S^\omega \text{ ist unendlicher Pfad von } \mathcal{K} \,\}$.

In der baumartigen Sichtweise interessieren wir uns für die azyklische Entfaltung von $\mathcal{K}$.

Mit $\mathcal{T}_{\mathcal{K},s} = (T, \Rightarrow, w, C, \xi)$, für $s \in S$, bezeichnen wir eine Kripke-Struktur mit folgenden Eigenschaften:

Der gerichtete Graph $(T, \Rightarrow)$ ist ein Baum.

$\xi(w) = \nu(s)$

Für jeden Zustand t mit $s \to t$ hat w einen Unterbaum isomorph zu $\mathcal{T}_{\mathcal{K},t}$.

Wir definieren $\mathcal{T}_{\mathcal{K}} := \mathcal{T}_{\mathcal{K},r}$.

Beschriftete Transitionssysteme

Manchmal ist es nützlich, die Aktionen in die Beschreibung eines Transitionssystems/einer Kripke-Struktur einzubeziehen (z.B. wenn Korrektheitseigenschaften über Aktionen formuliert werden sollen).

Definition: Ein Tupel $(S, A, \rightarrow, r, C, \nu)$ heißt beschriftete Kripke-Struktur, wobei

- A eine Menge von Aktionen ist;
- $\rightarrow \subseteq S \times A \times S$ (Schreibweise für ein Element: $s \xrightarrow{a} s'$);
- $\nu: A \rightarrow C$ eine Färbung der *Aktionen* ist.

Lineare Sichtweise: Für eine beschriftete Kripke-Struktur definieren wir

$$\mathcal{L}_{\mathcal{K}} := \{\, \nu(a_1)\nu(a_2)\ldots \mid r \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots \text{ ist unendlicher Pfad von } \mathcal{K} \,\}.$$

Baumartige Sichtweise: Es sei $\mathcal{T}_{\mathcal{K},s} = (T, \Rightarrow, w, C, \xi)$, für $s \in S$, eine (unbeschriftete) Kripke-Struktur, wobei

der gerichtete Graph $(T, \Rightarrow)$ ein Baum ist;

für jeden Zustand t mit $s \xrightarrow{a} t$ hat w einen Unterbaum isomorph zu $\mathcal{T}_{\mathcal{K},t}$, für dessen Wurzel w' gilt $\xi(w') = \nu(a)$.

Wir definieren $\mathcal{T}_{\mathcal{K}} := \mathcal{T}_{\mathcal{K},r}$ (Färbung der Wurzel beliebig).

Drei Logiken

In der Folge stellen wir drei Logiken vor:

LTL ist eine lineare Logik, sie wird auf Farbsequenzen interpretiert.

CTL ist eine Baumlogik, sie wird auf Farbbäumen interpretiert.

Die Ausdrucksmächtigkeit von LTL und CTL ist unvergleichbar, siehe MC1.

Wir betrachten eine dritte Logik CTL*, die mächtiger ist als LTL und CTL zusammen. CTL* ist ebenfalls eine Baumlogik.

Mehr zu diesem Thema: siehe Folien zu MC1!

Minimale Syntax von LTL

Sei C eine Menge von Farben.
Die Menge der LTL-Formeln über C ist wie folgt:

Ist $p \in C$, so ist p eine Formel.

Sind ϕ_1, ϕ_2 Formeln, so auch

$$\neg\phi_1, \qquad \phi_1 \vee \phi_2, \qquad \mathbf{X}\,\phi_1, \qquad \phi_1\,\mathbf{U}\,\phi_2$$

Bedeutung: $\mathbf{X} \mathrel{\widehat{=}}$ “next”, $\mathbf{U} \mathrel{\widehat{=}}$ “until”.

Semantik von LTL

Sei ϕ eine LTL-Formel und $\sigma \in C^\omega$ eine Farbsequenz.
Wir schreiben $\sigma \models \phi$ für "σ erfüllt ϕ."

$$
\begin{array}{lll}
\sigma \models & p & \text{falls } p \in C \text{ und } \sigma(0) = p \\
\sigma \models & \neg\phi & \text{falls } \sigma \not\models \phi \\
\sigma \models & \phi_1 \vee \phi_2 & \text{falls } \sigma \models \phi_1 \text{ oder } \sigma \models \phi_2 \\
\sigma \models & \mathbf{X}\,\phi & \text{falls } \sigma^1 \models \phi \\
\sigma \models & \phi_1 \,\mathbf{U}\, \phi_2 & \text{falls } \exists i\colon \left(\sigma^i \models \phi_2 \;\wedge\; \forall k < i\colon \sigma^k \models \phi_1\right)
\end{array}
$$

Semantik von ϕ: $[\![\phi]\!] = \{\, \sigma \mid \sigma \models \phi \,\}$

Beispiele

Sei $C = \{r, b, g\}$. Überlegen Sie, ob die Sequenz

$$\sigma = pbp^{\omega}$$

folgende Formeln erfüllt (d.h. die $\models$-Beziehung gilt):

$$r$$

$$b$$

$$\mathbf{X}\, b$$

$$\mathbf{X}\, \neg r$$

$$r\, \mathbf{U}\, b$$

$$b\, \mathbf{U}\, r$$

$$(b \vee r)\, \mathbf{U}\, g$$

Erweiterte Syntax

Folgende Abkürzungen werden uns oft nützlich sein:

$$\begin{aligned} \phi_1 \wedge \phi_2 &\equiv \neg(\neg\phi_1 \vee \neg\phi_2) \\ \phi_1 \rightarrow \phi_2 &\equiv \neg\phi_1 \vee \phi_2 \\ \mathbf{true} &\equiv a \vee \neg a \\ \mathbf{false} &\equiv \neg\mathbf{true} \end{aligned}$$

$$\begin{aligned} \mathbf{F}\,\phi &\equiv \mathbf{true}\,\mathbf{U}\,\phi \\ \mathbf{G}\,\phi &\equiv \neg\,\mathbf{F}\,\neg\phi \\ \phi_1\,\mathbf{W}\,\phi_2 &\equiv (\phi_1\,\mathbf{U}\,\phi_2) \vee \mathbf{G}\,\phi_1 \\ \phi_1\,\mathbf{R}\,\phi_2 &\equiv \neg(\neg\phi_1\,\mathbf{U}\,\neg\phi_2) \end{aligned}$$

Bedeutung: $\mathbf{F} \mathrel{\widehat{=}}$ “finally” (irgendwann), $\mathbf{G} \mathrel{\widehat{=}}$ “globally” (immer),
$\mathbf{W} \mathrel{\widehat{=}}$ “weak until”, $\mathbf{R} \mathrel{\widehat{=}}$ “release”.

Einige Beispiel-Formeln

$\mathbf{G}\,\neg(cs_1 \wedge cs_2)$

cs_1 and cs_2 treten niemals gemeinsam auf.

$\mathbf{G}\,\mathbf{F}\,p$

p tritt unendlich oft auf.

$\mathbf{F}\,\mathbf{G}\,p$

Ab irgendeinem Zeitpunkt gilt p immer.

$\mathbf{G}(try_1 \rightarrow \mathbf{F}\,cs_1)$

Bei Mutex-Algorithmen: Wenn Prozess 1 versucht, in seinen kritischen Abschnitt einzutreten, wird ihm das auch irgendwann gelingen.

Das LTL-Model-Checking-Problem

Gegeben: $\mathcal{K} = (S, \rightarrow, r, C, \nu)$, Formel ϕ über C.

Gefragt: Erfüllen alle Farbsequenzen in $\mathcal{K}$ die Formel, d.h. gilt $\mathcal{L}_{\mathcal{K}} \subseteq [\![\phi]\!]$?

Falls ja, sagen wir, dass $\mathcal{K}$ die Formel ϕ erfüllt.

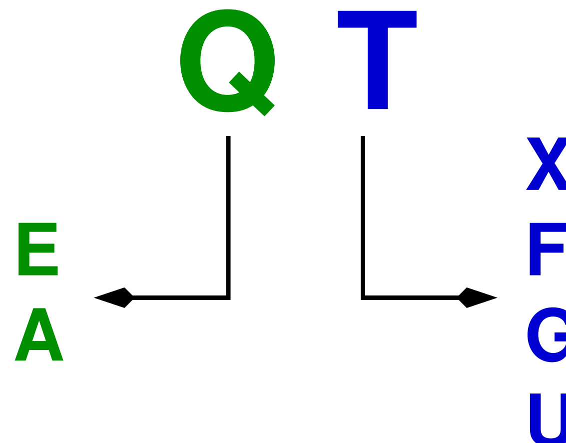
Andernfalls existiert ein Gegenbeispiel, d.h. ein Pfad $r\rho \in [\![\neg\phi]\!]$.

CTL: Überblick

CTL = Computation-Tree Logic

Aussagenlogik mit zusätzlichen, quantifizierten Pfad-Operatoren:

Operatoren haben die folgende Form:



es gibt einen Pfad — E
für alle Pfade — A

X — next
F — finally
G — globally
U — until

CTL: Minimale Syntax

Sei C eine Menge von Farben. Die Menge der CTL-Formeln über C ist wie folgt:

Ist $p \in C$, so ist p eine CTL-Formel.

Sind ϕ_1, ϕ_2 CTL-Formeln, so auch

$$\neg\phi_1, \qquad \phi_1 \vee \phi_2, \qquad \mathbf{EX}\,\phi_1, \qquad \mathbf{EG}\,\phi_1, \qquad \phi_1\,\mathbf{EU}\,\phi_2$$

CTL: Semantik

Sei ϕ eine CTL-Formel über C und $\mathcal{T} = (V, \rightarrow, r, AP, \nu)$ eine Kripke-Struktur, wobei $(V, \rightarrow)$ ein gerichteter Baum ist. Es sei $\lceil v \rceil$ der Unterbaum, der bei $v \in V$ beginnt. Wir schreiben $\mathcal{T} \models \phi$ für "$\mathcal{T}$ erfüllt ϕ". Diese Relation ist wie folgt definiert:

$\mathcal{T} \models p$	gdw. $p \in C$ und $\nu(r) = p$
$\mathcal{T} \models \neg\phi$	gdw. $\mathcal{T} \not\models \phi$
$\mathcal{T} \models \phi_1 \vee \phi_2$	gdw. $\mathcal{T} \models \phi_1$ oder $\mathcal{T} \models \phi_2$
$\mathcal{T} \models \mathbf{EX}\,\phi$	gdw. $\exists v\colon r \rightarrow v$ und $\lceil v \rceil \models \phi$
$\mathcal{T} \models \mathbf{EG}\,\phi$	gdw. $\mathcal{T}$ hat einen unendl. Pfad $r = v_0 \rightarrow v_1 \rightarrow \ldots$, so dass für alle $i \geq 0$ gilt: $\lceil v_i \rceil \models \phi$
$\mathcal{T} \models \phi_1\,\mathbf{EU}\,\phi_2$	gdw. $\mathcal{T}$ hat einen unendl. Pfad $r = v_0 \rightarrow v_1 \rightarrow \ldots$, so dass $\exists i\colon \lceil v_i \rceil \models \phi_2 \wedge \forall k < i\colon \lceil v_k \rceil \models \phi_1$

Das CTL-Model-Checking-Problem

Gegeben: $\mathcal{K} = (S, \rightarrow, r, C, \nu)$, Formel ϕ über C.

Gefragt: Gilt $\mathcal{T}_{\mathcal{K}} \models \phi$?

Falls ja, sagen wir, dass $\mathcal{K}$ die Formel ϕ erfüllt.

Andernfalls... existiert nicht notwendigerweise ein lineares Gegenbeispiel.

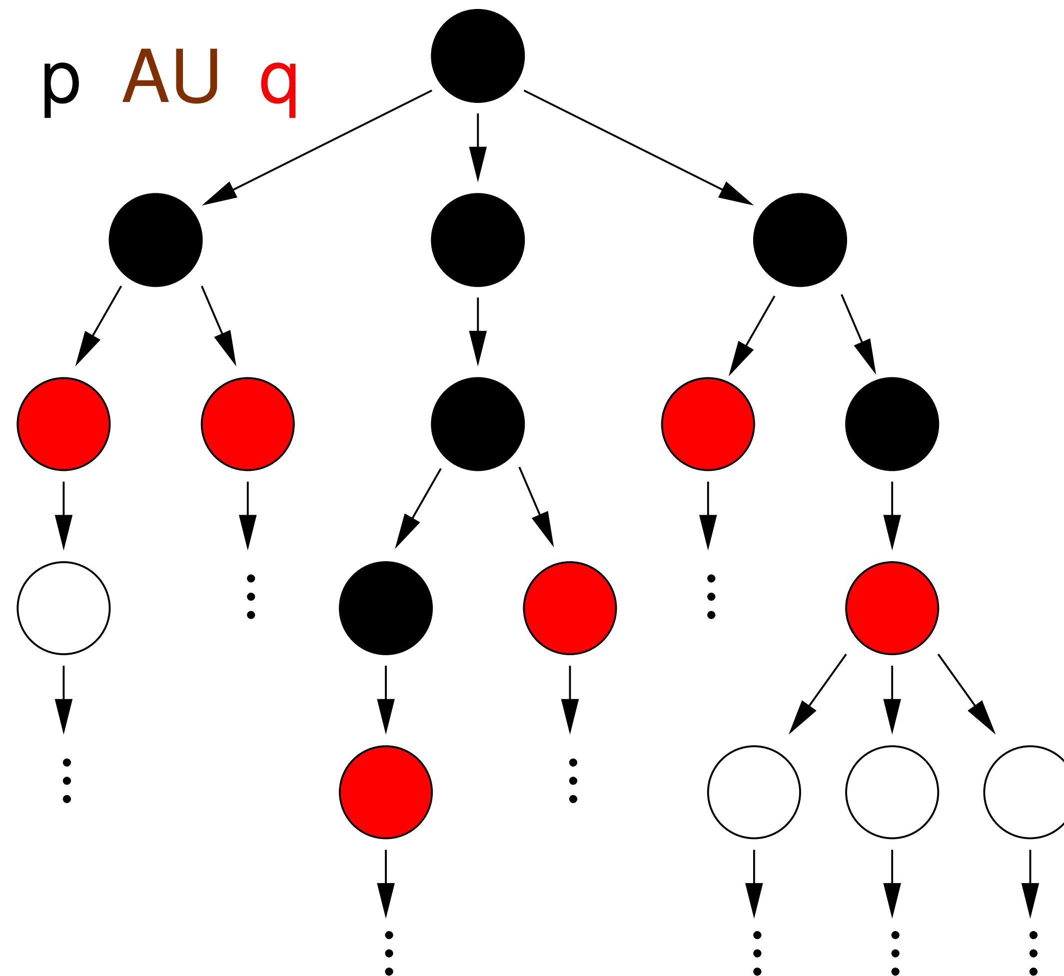
CTL: Erweiterte Syntax

Wir vereinbaren folgende abkürzende Schreibweisen: (weitere logische Operatoren können analog vereinbart werden)

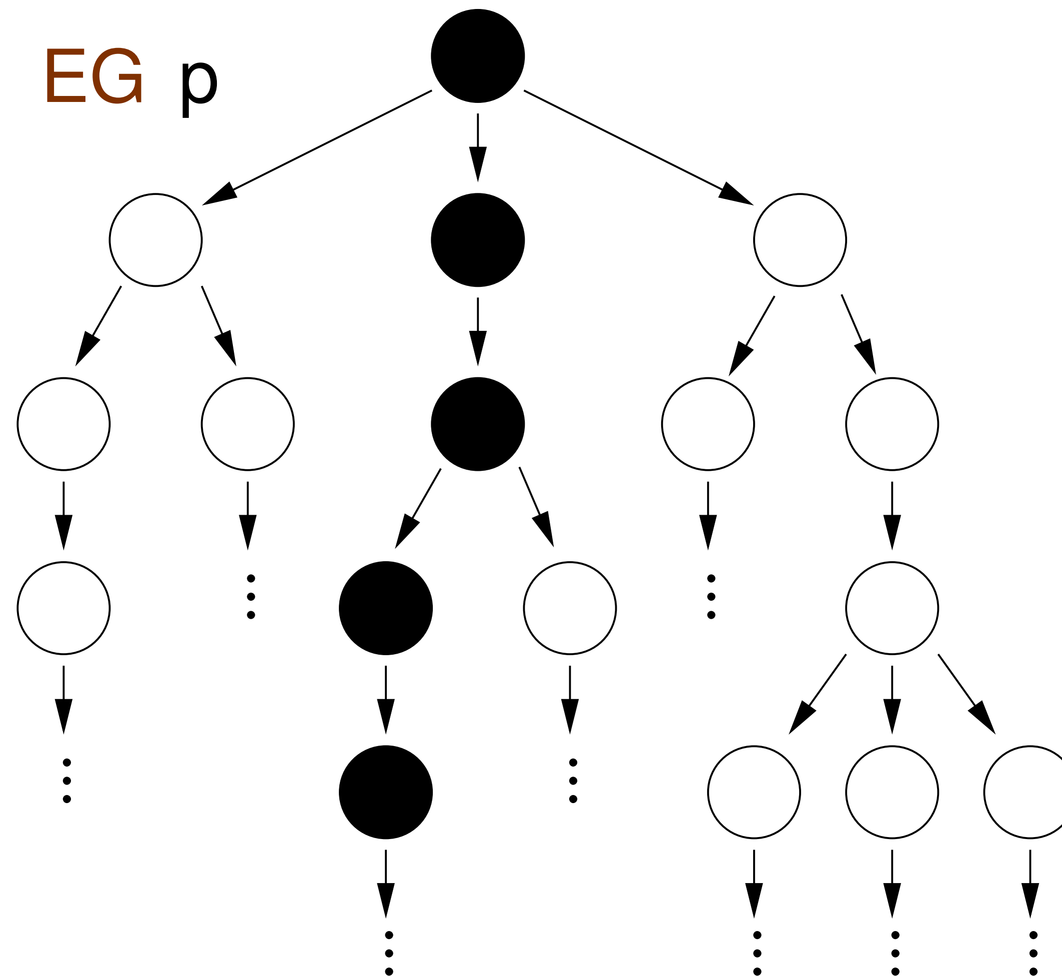
$$
\begin{aligned}
\phi_1 \wedge \phi_2 &\equiv \neg(\neg\phi_1 \vee \neg\phi_2) \\
\mathbf{true} &\equiv a \vee \neg a \\
\mathbf{false} &\equiv \neg\mathbf{true} \\
\phi_1 \,\mathbf{EW}\, \phi_2 &\equiv \mathbf{EG}\,\phi_1 \vee (\phi_1 \,\mathbf{EU}\, \phi_2) \\
\mathbf{EF}\,\phi &\equiv \mathbf{true}\,\mathbf{EU}\,\phi
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{AX}\,\phi &\equiv \neg\,\mathbf{EX}\,\neg\phi \\
\mathbf{AG}\,\phi &\equiv \neg\,\mathbf{EF}\,\neg\phi \\
\mathbf{AF}\,\phi &\equiv \neg\,\mathbf{EG}\,\neg\phi \\
\phi_1 \,\mathbf{AW}\, \phi_2 &\equiv \neg(\neg\phi_2 \,\mathbf{EU}\, \neg(\phi_1 \vee \phi_2)) \\
\phi_1 \,\mathbf{AU}\, \phi_2 &\equiv \mathbf{AF}\,\phi_2 \wedge (\phi_1 \,\mathbf{AW}\, \phi_2)
\end{aligned}
$$

Beispiel für CTL (1/2)



Beispiel für CTL (2/2)



CTL*: Syntax

CTL* ist (wie CTL) eine Baumlogik. Bei den Teilformeln unterscheidet man zwischen *Zustands-* und *Pfad-Formeln*. Jede CTL*-Formel ist eine Zustandsformel.

Zustandsformeln: Wenn $p \in C$ ist, ϕ_1, ϕ_2 Zustandsformeln sind und ρ eine Pfadformel, so sind folgende Formeln ebenfalls Zustandsformeln:

$$p, \qquad \phi_1 \vee \phi_2, \qquad \neg\phi_1, \qquad \mathbf{E}\,\rho$$

Pfadformeln: Wenn ϕ eine Zustandsformel ist und ρ_1, ρ_2 Pfadformeln, so sind folgende Formeln ebenfalls Pfadformeln:

$$\phi, \qquad \rho_1 \vee \rho_2, \qquad \neg\rho_1, \qquad \mathbf{X}\,\rho_1, \qquad \rho_1\,\mathbf{U}\,\rho_2$$

CTL*: Semantik (informell)

Wird auf $\mathcal{T}_{\mathcal{K}}$ interpretiert:

Zustandsformel:

Ein Baum erfüllt $p \in C$, wenn seine Wurzel mit p gefärbt ist.

Ein Baum erfüllt $\mathbf{E}\,\rho$, wenn es beginnend bei der Wurzel einen Pfad gibt, der ρ erfüllt.

Negation, Disjunktion: offensichtlich

Pfadformel:

Ein Pfad erfüllt ϕ (Zustandsformel), wenn der Baum, der beim ersten Zustand des Pfads beginnt, ϕ erfüllt.

Bedeutung von $\mathbf{X}$, $\mathbf{U}$, Disjunktion, Negation: wie bei LTL

CTL*: Erweiterte Syntax

$\mathbf{A}\,\rho :\equiv \neg\,\mathbf{E}\,\neg\rho$, außerdem $\mathbf{F}, \mathbf{G}, \mathbf{W}$ wie bei LTL

Beispiel:

> $\mathbf{E}\,\mathbf{G}\,\mathbf{F}\,p$: es existiert ein Pfad, auf dem unendlich oft p auftritt

Bemerkung:

> Jede CTL-Formel ist eine CTL*-Formel mit gleicher Semantik.
>
> Sei ρ eine LTL-Formel und $\mathcal{K}$ eine Kripke-Struktur.
> Dann gilt $\mathcal{L}_{\mathcal{K}} \subseteq [\![\rho]\!]$ gdw. $\mathcal{T}_{\mathcal{K}} \models \mathbf{A}\,\rho$.
> (bzw. bei beschrifteten Strukturen $\mathcal{L}_{\mathcal{K}} \subseteq [\![\rho]\!]$ gdw. $\mathcal{T}_{\mathcal{K}} \models \mathbf{A}\,\mathbf{X}\,\rho$).

Simulation

Seien $\mathcal{K}_1 = (S, \rightarrow_1, s_0, C, \nu)$ und $\mathcal{K}_2(T, \rightarrow_2, t_0, C, \mu)$ zwei Kripke-Strukturen (mit gleichen Farben) und $H \subseteq S \times T$ eine Relation.

H heißt Simulation von $\mathcal{K}_1$ nach $\mathcal{K}_2$ gdw.

(i) $(s_0, t_0) \in H$;

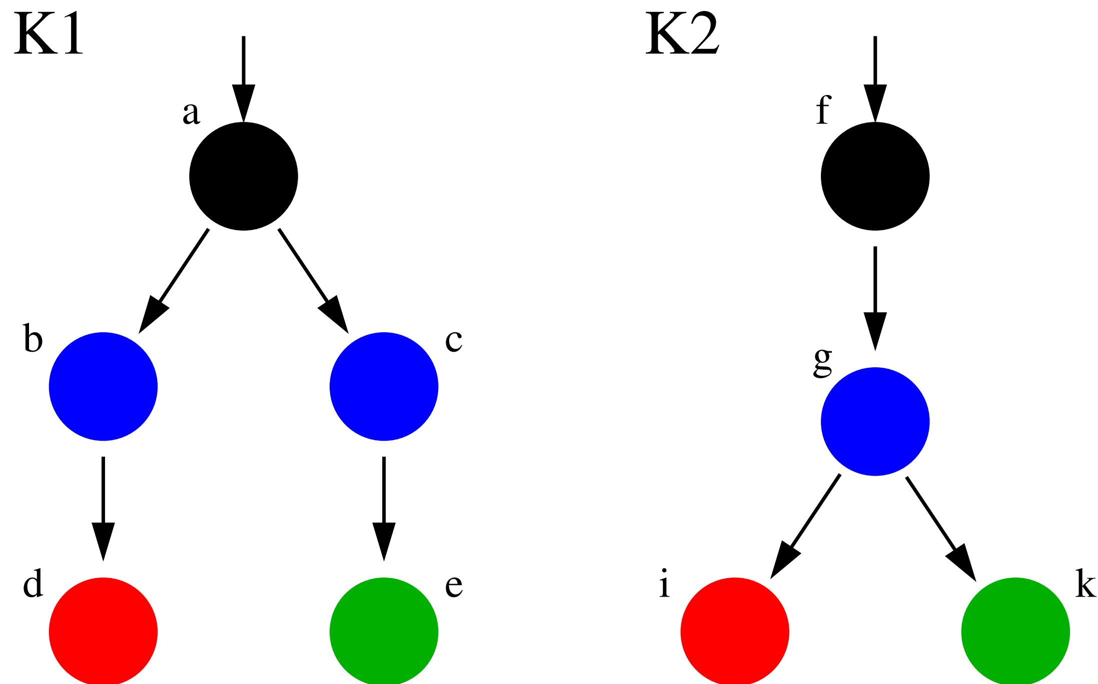
(ii) für alle $(s, t) \in H$ gilt: $\nu(s) = \mu(t)$;

(iii) falls $(s, t) \in H$ und $s \rightarrow_1 s'$, dann existiert t' mit $t \rightarrow_2 t'$ und $(s', t') \in H$.

Wir sagen: $\mathcal{K}_2$ simuliert $\mathcal{K}_1$ (geschrieben $\mathcal{K}_1 \leq \mathcal{K}_2$), falls eine solche Simulation H existiert.

Bemerkung: Definition für beschriftete KS analog, statt (ii) und (iii) verlangen wir: falls $(s, t) \in H$ und $s \xrightarrow{a}_1 s'$, dann gibt es $t \xrightarrow{b}_2 t'$ mit $\nu(a) = \mu(b)$ und $(t, t') \in H$.

Intuition: In $\mathcal{K}_2$ ist alles möglich, was auch in $\mathcal{K}_1$ möglich ist.



$\mathcal{K}_2$ simuliert $\mathcal{K}_1$ (mit $H = \{(a, f), (b, g), (c, g), (d, i), (e, k)\}$).

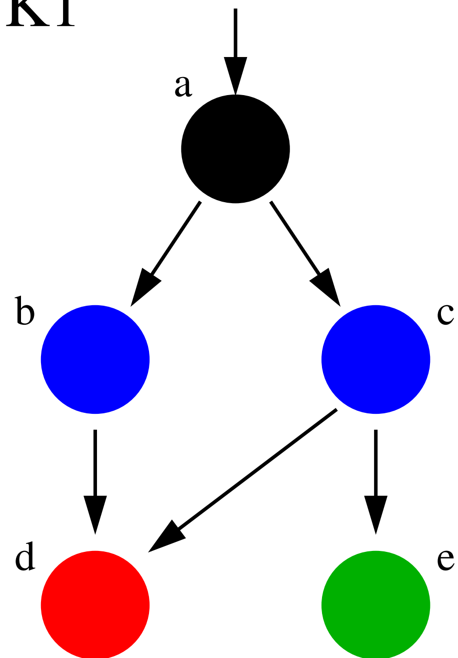
Aber: $\mathcal{K}_1$ simuliert *nicht* $\mathcal{K}_2$!

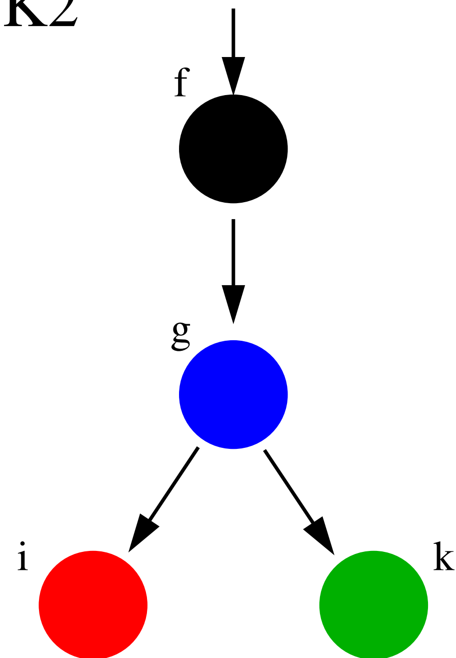
Bisimulation

Eine Relation H heißt Bisimulation zwischen $\mathcal{K}_1$ und $\mathcal{K}_2$ gdw. H eine Simulation von $\mathcal{K}_1$ nach $\mathcal{K}_2$ ist und $\{\,(t, s) \mid (s, t) \in H\,\}$ eine Simulation von $\mathcal{K}_2$ nach $\mathcal{K}_1$.

Wir sagen: $\mathcal{K}_1$ und $\mathcal{K}_2$ sind bisimilar (geschrieben $\mathcal{K}_1 \equiv \mathcal{K}_2$) gdw. eine solche Relation H existiert.

Vorsicht: Aus $\mathcal{K}_1 \leq \mathcal{K}_2$ und $\mathcal{K}_2 \leq \mathcal{K}_1$ folgt *nicht* $\mathcal{K}_1 \equiv \mathcal{K}_2$!





(Bi-)Simulation und Model-Checking

Sei $\mathcal{K}_1 \leq \mathcal{K}_2$ und ϕ eine LTL-Formel.

Dann gilt: $\mathcal{K}_2 \models \phi$ impliziert $\mathcal{K}_1 \models \phi$.

Beweis: Es gilt $\mathcal{L}_{\mathcal{K}_1} \subseteq \mathcal{L}_{\mathcal{K}_2} \subseteq [\![\phi]\!]$.

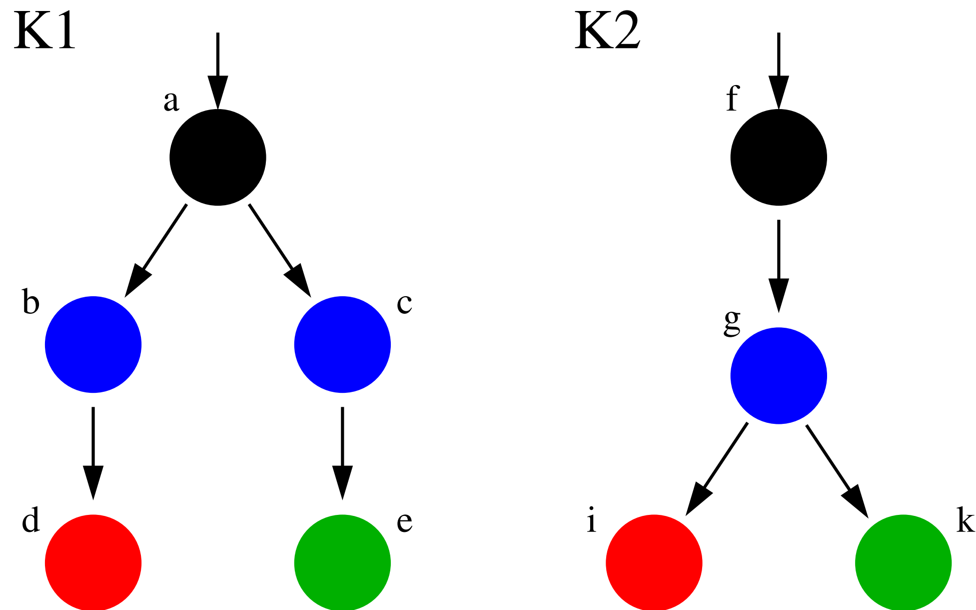
Sei $\mathcal{K}_1 \equiv \mathcal{K}_2$ und ϕ eine CTL*-Formel.

Dann gilt: $\mathcal{K}_1 \models \phi$ gdw. $\mathcal{K}_2 \models \phi$.

Beweis: Strukturelle Induktion über die Formel ϕ.

Seien $\mathcal{K}_1, \mathcal{K}_2$ zwei Strukturen mit $\mathcal{K}_2 \not\leq \mathcal{K}_1$.

Dann gibt es eine CTL-Formel ϕ mit $\mathcal{K}_2 \models \phi$, aber $\mathcal{K}_1 \not\models \phi$.



Beispiel: $\mathit{schwarz} \wedge \mathbf{EX}\Big(\mathit{blau} \wedge (\mathbf{EX}\,\mathit{rot}) \wedge (\mathbf{EX}\ \mathit{grün})\Big)$
(Die Formel beschreibt das Verzweigungs-Verhalten von $\mathcal{K}_2$.)

Die vorangegangenen Bemerkungen unterstreichen, warum Bisimulation interessant ist:

Bisimilare Strukturen werden von Model-Checking nicht unterschieden, nicht-bisimilare hingegen schon.

Anwendung: Vergleich zweier Klassen von Kripke-Strukturen $\mathcal{A}, \mathcal{B}$; Klasse $\mathcal{B}$ gilt als mächtiger ($\mathcal{A} < \mathcal{B}$), falls

> für jede Struktur aus $\mathcal{A}$ eine bisimilare Struktur aus $\mathcal{B}$ existiert;
>
> und es Strukturen aus $\mathcal{B}$ gibt, die nicht bisimilar zu irgendeiner Struktur aus $\mathcal{A}$ sind.

Teil 3: Die PRS-Hierarchie

Literatur

Dissertation von Richard Mayr (TU München, 1998):

Decidability and Complexity of Model Checking Problems for Infinite-State Systems

PRS = Prefix Rewrite Systems

Prozess-Terme

Sei V eine so genannte Prozess-Variablen.

Die Menge der Prozess-Terme aus V, geschrieben $\mathcal{T}_V$ ist induktiv wie folgt definiert:

- ε ist ein Prozess-Term (der leere Term).
- Ist $X \in V$, so ist X ein Prozess-Term.
- Sind t_1, t_2 Prozess-Terme, so auch
  - $t_1 \parallel t_2$ (parallele Komposition)
  - $t_1 \,.\, t_2$ (sequentielle Komposition)

Intuition

Beispiel: $((X\ .\ W) \parallel \varepsilon)\ .\ (Y \parallel Y \parallel Z)$

Ein Prozess-Term stellt den *Zustand eines nebenläufigen Systems* dar:

Jede Variable ist ein “Agent” (oder: Sprungmarke), die in andere Sub-Terme übergehen kann.

ε entspricht einem beendeten Prozess.

Jede Variable, die *nicht* hinter irgendeinem Sequenz-Operator steht, entspricht einem aktiven Prozess (im Beispiel: X).

Jede Variable hinter einem Sequenz-Operator entspricht einem Prozess, der auf das Terminieren eines vor ihm stehenden wartet (z.B. Prozeduraufruf).

Schreibweise / Äquivalenz

Zur Vereinfachung der Schreibweise vereinbaren wir folgende Äquivalenzen zwischen Termen:

- Sequenzielle und parallele Komposition sind assoziativ, d.h. $t_1 \,.\, (t_2 \,.\, t_3) \equiv (t_1 \,.\, t_2) \,.\, t_3$ und $t_1 \parallel (t_2 \parallel t_3) \equiv (t_1 \parallel t_2) \parallel t_3$.
- Parallele Komposition ist kommutativ, d.h. $t_1 \parallel t_2 \equiv t_2 \parallel t_1$.
- Der leere Prozess “zählt nicht”, d.h. $\varepsilon \,.\, t \equiv t \,.\, \varepsilon \equiv t \equiv t \parallel \varepsilon$.

Außerdem bindet $.$ stärker als $\parallel$, d.h. $t_1 \,.\, t_2 \parallel t_3$ bedeutet $(t_1 \,.\, t_2) \parallel t_3$.

Wir unterscheiden in der Folge nicht mehr zwischen äquivalenten Termen, d.h. wenn wir einen Term erwähnen, meinen wir implizit seine Äquivalenzklasse.

Prefix Rewrite Systems

Sei V eine Menge von Prozess-Variablen und A eine Menge von Aktionen.

Sei $\mathcal{P}$ eine *endliche* Menge von Regeln der Form $t \xrightarrow{a} t'$, wobei t, t' Prozess-Terme über V sind und $a \in A$ eine Aktion ist, und sei $X \in V$ irgendeine Variable. $(\mathcal{P}, X)$ heißt Prefix Rewrite System (PRS) über (V, A).

Beispiel: $X \xrightarrow{a} Y \parallel Z, \quad Y \xrightarrow{b} Z \,.\, Z, \quad Z \xrightarrow{c} \varepsilon$

Ein PRS $(\mathcal{P}, X)$ definiert eine beschriftete Kripke-Struktur $\mathcal{K}_{\mathcal{P},X} = (\mathcal{T}_V, \Rightarrow, A, X, A, \mathrm{id}_A)$ wie folgt:

Die Zustände sind die Prozess-Terme, und X ist der Anfangsterm.

Jede Aktion ist eine Farbe für sich.

Falls $t \xrightarrow{a} t' \in \mathcal{P}$, dann $t \overset{a}{\Rightarrow} t'$.

Falls $t \overset{a}{\Rightarrow} t'$, dann auch $t \,.\, t'' \overset{a}{\Rightarrow} t' \,.\, t''$.

Falls $t \overset{a}{\Rightarrow} t'$, dann auch $t \parallel t'' \overset{a}{\Rightarrow} t' \parallel t''$.

Term-Klassen

Wir betrachten im Folgenden diese vier Klassen von Prozess-Termen:

$\mathcal{T}_1 = V$ seien die Terme, die aus genau einer Variablen bestehen.

$\mathcal{T}_S$ seien die Terme, die aus ε, Elementen von V sowie dem Sequenz-Operator gebildet sind.

$\mathcal{T}_P$ seien die Terme, die aus ε, Elementen von V sowie dem Parallel-Operator gebildet sind.

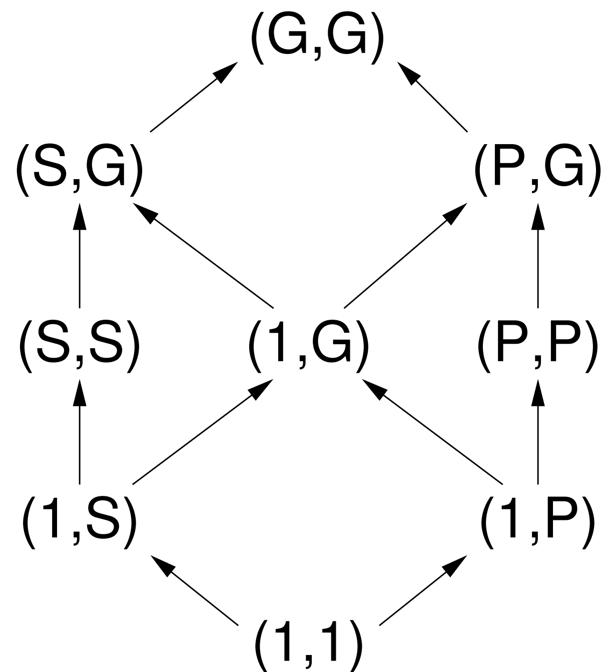
$\mathcal{T}_G = \mathcal{T}_V$ seien alle Prozess-Terme.

Ein PRS hat den Typ (α, β) (mit $\alpha, \beta \in \{1, S, P, G\}$) gdw. alle Regeln $t \xrightarrow{a} t'$ dergestalt sind, dass $t \in \mathcal{T}_\alpha$ und $t' \in \mathcal{T}_\beta$.

Beispiel: Ein (1,1)-PRS hat nur Regeln der Art $X \xrightarrow{a} Y$.
Jedes PRS ist automatisch ein (G,G)-PRS.

Die PRS-Hierarchie

Die PRS-Hierarchie ist wie folgt aufgebaut, wobei ein Eintrag (α, β) die Klasse aller PRS dieses Typs bedeutet.



Klassen weiter oben in der Hierarchie enthalten diejenigen, die weiter unten stehen.

Interessante Eigenschaften der PRS-Hierarchie

Einige Klassen der Hierarchie entsprechen bekannten Automatenklassen:

(1,1)-PRS = endliche Automaten

(S,S)-PRS = Kellerautomaten

(P,P)-PRS = Petri-Netze

Sequentielle und parallele Komposition sind natürliche Operationen in vielen Systemen; die PRS-Hierarchie untersucht ihre Interaktion.

Viele Entscheidbarkeits- und Komplexitätsresultate über die verschiedenen PRS-Klassen sind bekannt!

Die PRS-Hierarchie ist strikt, d.h. jede Klasse weiter oben in der Hierarchie ist mächtiger als die Klassen weiter unten.

Namen der Klassen

Folgende Namen sind in der Literatur für die einzelnen Prozessklassen gebräuchlich (nach Mayr):

(1,1)-PRS = FS (finite-state systems)

(1,S)-PRS = BPA (basic process algebra)

(1,P)-PRS = BPP (basic parallel processes)

(1,G)-PRS = PA (process algebra)

(S,S)-PRS = PDS (pushdown systems/processes)

(P,P)-PRS = PN (Petri nets)

(S,G)-PRS = PAD (process algebra/pushdown)

(P,G)-PRS = PAN (process algebra/Petri nets)

(G,G)-PRS = PRS (process rewrite systems)

Mächtigkeit der PRS-Klassen

Wir betrachten das (1,S)-PRS $\mathcal{P}$ mit den zwei Regeln

$$A \xrightarrow{a} A \,.\, A \qquad \text{und} \qquad A \xrightarrow{b} \varepsilon$$

und Anfangssymbol A.

Behauptung: $\mathcal{P}$ ist nicht bisimilar zu irgendeinem (1,1)-PRS.

Beweis: Sei $\mathcal{Q}$ ein FS mit n Variablen, das bisimilar zu $\mathcal{P}$ ist wg. Relation H. Sei

$$C_0 \stackrel{a}{\Longrightarrow} \cdots \stackrel{a}{\Longrightarrow} C_n$$

ein Ablauf in Q. Dann ist $C_k = C_\ell$ für irgendwelche $k < \ell \leq n$. Also müssen (C_k, A^{k+1}) und $(C_\ell, A^{\ell+1})$ in H sein. Nun betrachten wir die Fortsetzung $C_k = C_\ell \stackrel{b^{k+1}}{\Longrightarrow}{}^{*} D$. Dann müssen (D, ε) und $(D, A^{\ell-k})$ in H sein. Das aber bedeutet, dass H keine Bisimulation sein kann.

Anmerkung: Mit dem System

$$A \xrightarrow{a} A \parallel A \qquad \text{und} \qquad A \xrightarrow{b} \varepsilon$$

zeigt man analog, dass (1,P)-PRS mächtiger als (1,1)-PRS sind.

BPA und BPP

Auf ähnliche Weise zeigt man, dass (1,S) und (1,P) verschieden mächtig sind:

BPP $\not\subseteq$ BPA: Man betrachte das BPP $\mathcal{P}_1$

$$X \xrightarrow{a} X \parallel B \qquad X \xrightarrow{c} \varepsilon \qquad B \xrightarrow{b} \varepsilon$$

Dieses BPP ist nicht bisimilar zu irgendeinem BPA. (Wegen fehlender globaler Kontrolle kann sich ein BPA nicht “merken”, ob ein c schon gekommen ist.)

BPA $\not\subseteq$ BPP: Man betrachte das BPA $\mathcal{Q}_1$

$$X \xrightarrow{a} X \,.\, B \qquad X \xrightarrow{c} \varepsilon \qquad B \xrightarrow{b} \varepsilon$$

Dieses BPA ist nicht bisimilar zu irgendeinem BPP. (In diesem BPA geht X mit $a^n c b^n$ zu ε über. Ein BPP müsste zwei Stellen gleichzeitig abfragen, um zu schauen, ob schon ein c gekommen ist.)

Bemerkung: Daraus folgt, dass (1,G) mächtiger ist als (1,S) und (1,P).

Beweis zu BPP $\not\subseteq$ BPA

Angenommen, es gäbe ein BPA $\mathcal{P}_2$ (mit Variablen V und Anfangsvariable Y), so dass $\mathcal{P}_1 \equiv \mathcal{P}_2$ (mit Relation H).

Wir betrachten die Sequenz $Y \stackrel{a}{\Longrightarrow} Y_1 . \alpha_1 \stackrel{a}{\Longrightarrow} Y_2 . \alpha_2 \stackrel{a}{\Longrightarrow} \cdots$ in $\mathcal{P}_2$. Davon wiederum betrachten wir die Untersequenz derjenigen $Y_j . \alpha_j$, so dass $|\alpha_j| \leq |\alpha_k|$ für alle $k \geq j$. Man sieht leicht, dass diese Untersequenz existiert und unendlich ist. Da sie unendlich ist, $\mathcal{P}_2$ aber nur endlich viele Variablen hat, muss eine von diesen unendlich oft in der Untersequenz vorkommen, sagen wir Z.

Daher gibt es $m, n > 0$ und $\alpha, \beta \in V^*$ so dass

$$Y \stackrel{a^m}{\Longrightarrow}^* Z . \alpha \stackrel{a^n}{\Longrightarrow}^* Z . \beta . \alpha.$$

Notwendigerweise sind (X, Y), $(X \parallel B^m, Z . \alpha)$, $(X \parallel B^{m+n}, Z . \beta . \alpha)$ in H enthalten.

Wir zeigen jetzt, dass es $k \leq m$ geben muss, so dass $Z \stackrel{b^k}{\Longrightarrow}^* \varepsilon$ gilt. Angenommen, k existiert nicht. Dann sei $W \in V$ eine Variable mit der

Eigenschaft $Z \overset{b^m}{\Longrightarrow}^* W\gamma$ für irgendein $\gamma \in V^*$. Dann gilt

$$Y \overset{a^{m+n}}{\Longrightarrow}^* Z \,.\, \beta \,.\, \alpha \overset{b^m}{\Longrightarrow}^* W \,.\, \gamma \,.\, \beta \,.\, \alpha$$

und $(X \parallel B^n, W \,.\, \gamma \,.\, \beta \,.\, \alpha) \in H$. Wegen $n \geq 0$ muss es eine Regel $W \xrightarrow{b} \delta$ geben. Andererseits gilt auch

$$Y \overset{a^m}{\Longrightarrow}^* Z \,.\, \alpha \overset{b^m}{\Longrightarrow}^* W \,.\, \gamma \,.\, \alpha$$

und $(X, W \,.\, \gamma \,.\, \alpha) \in H$. Aber der Term X in $\mathcal{P}_1$ kann kein b ausführen, der Term $W \,.\, \gamma \,.\, \alpha$ in $\mathcal{P}_2$ hingegen schon. Widerspruch.

Analog zeigt man, dass es $\ell \leq m$ geben muss, so dass $Z \overset{cb^\ell}{\Longrightarrow}^* \varepsilon$ gilt.

Dann aber gilt

$$Y \overset{a^m b^k}{\Longrightarrow}^* \alpha \qquad \text{und} \qquad Y \xrightarrow{a^m c b^\ell}{}^* \alpha.$$

Demzufolge müssen $(X \parallel B^{m-k}, \alpha)$ und $(B^{m-\ell}, \alpha)$ in H enthalten sein. Wegen $X \parallel B^{m-k}$ muss α ein a ausführen können, wegen $B^{m-\ell}$ darf es das nicht. Widerspruch.

Beweis zu BPP $\not\subseteq$ BPA

Angenommen, es gäbe ein BPP $\mathcal{Q}_2$ (mit Variablen V und Anfangsvariable Y), so dass $\mathcal{Q}_1 \equiv \mathcal{Q}_2$ (mit Relation H).

Wir betrachten für alle $i \geq 1$ die Terme α_i, β_i, die man durch $Y \stackrel{a^i}{\Longrightarrow}^* \alpha_i \stackrel{c}{\Longrightarrow}^* \beta_i$ erhält. Es sind jeweils $(X \,.\, B^i, \alpha_i)$ und (B^i, β_i) in H enthalten.

Da $\mathcal{Q}_2$ nur endlich viele Regeln hat, muss irgendeine Regel der Form $Z \stackrel{c}{\longrightarrow} \gamma$ bei unendlich vielen i für den Übergang von α_i zu β_i benutzt werden, d.h. für diese i gilt $\alpha_i = \alpha_i' \parallel Z$ und $\beta_i = \alpha_i' \parallel \gamma$. Seien $k < \ell$ zwei Indizes, bei denen die genannte Regel benutzt wird.

In α_ℓ ist keine mit b beschriftete Aktion möglich, aber aus β_ℓ ist die Sequenz b^ℓ möglich. Diese kann also nur von γ abhängen, d.h. es gibt eine Sequenz der Form $\gamma \stackrel{b^\ell}{\Longrightarrow}^* \eta$. Aber dann ist in $\mathcal{Q}_2$ Folgendes möglich:

$$Y \stackrel{a^k}{\Longrightarrow}^* \alpha_k' \parallel Z \stackrel{c}{\Longrightarrow} \alpha_k' \parallel \gamma \stackrel{b^\ell}{\Longrightarrow}^* \alpha_k' \parallel \eta,$$

Wegen $k < \ell$ geht dies in $\mathcal{Q}_1$ nicht. Widerspruch.

BPA und PDS

Das PDS mit den Regeln

$$X \xrightarrow{a} X.B \qquad X \xrightarrow{c} \varepsilon \qquad B \xrightarrow{b} \varepsilon \qquad X.B \xrightarrow{b} \varepsilon$$

hat dasselbe Verhalten wie $\mathcal{P}_1$. Daher ist (S,S) mächtiger als (1,S).

Hingegen ist das BPP

$$X \xrightarrow{a} X \parallel B \quad X \xrightarrow{c} X \parallel D \qquad X \xrightarrow{e} \varepsilon \qquad B \xrightarrow{b} \varepsilon \qquad D \xrightarrow{d} \varepsilon$$

nicht bisimilar zu irgendeinem PDS. Daher ist (S,S) unvergleichbar mit (1,P).

> Beweis: Gäbe es ein bisimilares PDS, so wäre die Sprache $\{\, \alpha \mid X \overset{\alpha}{\Longrightarrow}^* \varepsilon \}$ kontextfrei, daher auch der Schnitt mit der regulären Sprache $a^*c^*b^*d^*e$. Dieser aber ist $\{\, a^k c^\ell c^k d^\ell e \mid k, \ell \geq 0 \,\}$, was nicht kontextfrei ist.

Durch ähnliche Beispiele zeigt man analog, dass (1,P) in (P,P) enthalten ist, und dass (1,S) nicht in (P,P) enthalten ist.

PAD und PAN

Es gibt ein Pushdown-System (S,S), dass nicht bisimilar zu einem PAN (P,G) ist.

Es gibt ein Petri-Netz (P,P), dass nicht bisimilar zu einem PAD (S,G) ist.

Aus den vorangegangenen Sätzen folgt:

> Von je zwei Klassen, die in der PRS-Hierarchie durch Pfeile verbunden sind, ist die obere mächtiger als die untere.
>
> Jedes Paar von Klassen, die nicht (transitiv) durch Pfeile verbunden sind, sind unvergleichbar.

Beweis zu PDS $\not\subseteq$ PAN

Wir betrachten das PDS mit Anfangsvariable S und folgenden Regeln:

$$
\begin{array}{rcllrcll}
S & \xrightarrow{g} & U.X & & & & & \\
U.x & \xrightarrow{a} & U.A.x & (x \in \{X,A,B\}) & x.A & \xrightarrow{a} & x & (x \in \{V,W\}) \\
U.x & \xrightarrow{b} & U.B.x & (x \in \{X,A,B\}) & x.B & \xrightarrow{b} & x & (x \in \{V,W\}) \\
U.x & \xrightarrow{c} & V.x & (x \in \{X,A,B\}) & V.X & \xrightarrow{e} & V & \\
U.x & \xrightarrow{d} & W.x & (x \in \{X,A,B\}) & W.X & \xrightarrow{f} & W &
\end{array}
$$

Die möglichen Aktionsfolgen beginnen mit einem g, gefolgt von einer Sequenz $w \in \{a,b\}^*$. Anschließend entweder c, gefolgt von w rückwärts, schließlich e, oder d, gefolgt von w rückwärts, schließlich f.

Angenommen, es gäbe ein PAN, das bisimilar zu diesem PDS ist mit Relation H. Dann sei $\Sigma \in \{A, B\}^*$ und t ein Term des PAN, so dass $(U \,.\, \Sigma \,.\, X, t) \in H$.

In t müssen die Aktionen c und d möglich sein (und zu Termen t_c bzw. t_d führen), aber sie müssen sich gegenseitg deaktivieren. Daher muss t einen Unterterm $\alpha \in \mathcal{T}_P$ haben, so dass $\alpha \stackrel{c}{\Longrightarrow} \alpha_c$ und $\alpha \stackrel{d}{\Longrightarrow} \alpha_d$ für irgendwelche $\alpha_c, \alpha_d \in \mathcal{T}_G$. Damit hat t die Form

$$t = ((((\alpha \,.\, t_S^0) \parallel t_P^0) \,.\, t_S^1) \parallel t_P^1) \,.\, \cdots,$$

wobei t_S^i, t_P^i irgendwelche Prozess-Terme sind, d.h. α eingebettet in irgendeinen Kontext.

Wir betrachten jetzt den Term $\beta := t_P^0 \parallel t_P^1 \parallel \cdots$. In β dürfen keine Aktionen c, d, e, f, g möglich sein, sonst wären diese in t, t_c, t_d möglich, was nicht sein darf. Nehmen wir an, Σ beginnt mit einem A. Dann darf in t_c (und damit in β) auch keine Aktion b möglich sein. Aber es ist die Sequenz $t \stackrel{b}{\Longrightarrow} t' \stackrel{c}{\Longrightarrow} t''$ möglich, wegen zuvor erwähnter Beschränkung muss dies wegen $\alpha \stackrel{b}{\Longrightarrow} \alpha'$ und $\alpha' \,.\, t_S^0 \stackrel{c}{\Longrightarrow} \alpha''$ sein, für irgendwelche Terme t', t'' und α', α''. Dann gilt $(V \,.\, B \,.\, \Sigma \,.\, X, t'') \in H$ und

$$t'' = ((((\alpha'') \parallel t_P^0) \,.\, t_S^1) \parallel t_P^1) \,.\, \cdots.$$

In t'' darf keine a-Aktion möglich sein, daher auch nicht in β. Beginnt Σ mit einem B, argumentieren wir analog.

Wir folgern also, dass in β überhaupt gar keine Aktionen möglich sind, β ist quasi “tot”. Wir können also o.b.d.A. annehmen, dass t die Form $\alpha \,.\, t_S$ hat. Nun macht t ein c oder d und am Ende ein e oder f. Da α sowohl c als auch d machen kann, liegt es auf der Hand, dass die Entscheidung über e und f nicht in t_S gespeichert sein kann. Daher können wir o.b.d.A. annehmen, dass α nie nach ε übergeht.

Einschub: Für den nächsten Schritt braucht man Dicksons Lemma.

> Gegeben eine unendliche Sequenz von (endlichen) Vektoren $M_1, M_2, \ldots$, deren Einträge natürliche Zahlen sind, existieren Indizes $k < \ell$, so dass alle Komponenten von M_k kleiner gleich sind wie die von M_ℓ.

Wir betrachten also für alle $i \geq 0$ die Aktions-Sequenz ba^i. Sei t_i ein Term mit $(U \,.\, A^i \,.\, B \,.\, X, t_i) \in H$, und sei $\alpha_i \in \mathcal{T}_P$ der Unterterm von t_i, der wie oben c und d ermöglicht.

Wegen Dicksons Lemma muss es $k < \ell$ geben, so dass $\alpha_\ell = \alpha_k \parallel \gamma$ für irgendein γ. In α_k muss die Sequenz $ca^k be$ möglich sein, daher auch in α_ℓ, was aber nicht sein darf. Damit haben wir einen Widerspruch erreicht.

Beweis zu PN $\not\subseteq$ PAD

Wir betrachten folgendes Petri-Netz mit Anfangsvariable X:

$$X \xrightarrow{g} X \parallel A \parallel B \qquad X \xrightarrow{c} Y \qquad Y \parallel A \xrightarrow{a} Y \qquad Y \parallel B \xrightarrow{b} Y$$
$$X \parallel A \xrightarrow{d} Z \qquad X \parallel B \xrightarrow{d} Z \qquad Y \parallel A \xrightarrow{d} Z \qquad Y \parallel B \xrightarrow{d} Z$$

Das Netz erzeugt zunächst beliebig viele (aber gleich viele) Kopien von A und B, macht dann ein c und baut die As und Bs wieder ab. Außerdem kann es jederzeit mit einer d-Aktion terminieren.

Wir zeigen zunächst eine Hilfsbehauptung: Wenn es ein PAD gibt, dass bisimilar (mit Relation H) zu diesem Netz ist, dann gibt es auch ein PDS mit der gleichen Eigenschaft.

Sei α ein Term des Petri-Netzes und $t_1 \parallel t_2$ ein Unterterm irgendeines Terms t des PAD, so dass $(\alpha, t) \in H$ ist. Wenn in α überhaupt eine Aktion möglich ist, dann auch d und nach d keine weitere. Jede Regel des PAD kann nur t_1 *oder* t_2 modifizieren, aber nicht beide, da die linke Seite aus $\mathcal{T}_S$ ist. Daraus folgt, dass t_1 oder t_2 (oder beide) keine Aktionen ausführen können.

Falls das PAD also eine Regel $u \xrightarrow{a} t$ enthält und t einen Unterterm $t_1 \parallel t_2$ enthält, so ist o.b.d.A. in t_2 keine Aktion möglich. Man kann daher $t_1 \parallel t_2$ durch $t_1 \,.\, t_2$ ersetzen, ohne das Verhalten des Systems zu ändern. So eliminiert man alle Vorkommen des Parallel-Operators im PAD und erhält ein PDS.

Wenn es also ein PAD gibt, das bisimilar zum Petri-Netz ist, so muss die Sprache aller Wörter, die vom Term X zum Term Y führen, kontextfrei sein, damit auch der Schnitt mit der regulären Sprache $g^* ca^* b^*$. Dieser Schnitt ist die Sprache

$$\{ g^n ca^n b^n \mid n \geq 0 \}.$$

Es lässt sich leicht zeigen (Pumping-Lemma), dass diese Sprache aber nicht kontextfrei ist, womit wir einen Widerspruch erreicht haben.

Teil 4: Pushdown-Systeme

Einleitung

Zur Erinnerung: PDS ist die Klasse der (S,S)-PRS, d.h. mit Regeln der Form $\alpha \xrightarrow{a} \beta$, wobei α, β Sequenzen von Variablen sind.

Sei $\mathcal{P}$ ein PDS und ϕ eine (CTL/LTL)-Formel. Es gibt ein PDS $\mathcal{Q}$ und eine Formel ψ mit folgenden Eigenschaften:

(i) für jede Regel $\alpha \xrightarrow{a} \beta$ in $\mathcal{Q}$ gilt $|\alpha| = 2$ und $1 \leq |\beta| \leq 3$.

(ii) $\mathcal{P}$ erfüllt ϕ gdw. $\mathcal{Q}$ erfüllt ψ.

Beweis (Skizze):
(i) leicht, nötigenfalls wird ein Schritt durch mehrere ersetzt;
(ii) der erste Schritt erzeugt evtl. zusätzlichen "Zwischenkonfigurationen", ggfs. müssen $\mathbf{X}$-Unterformeln in ϕ durch geeignete $\mathbf{U}$-Unterformeln in ψ ersetzt werden.

“Normierte” Darstellung eines PDS

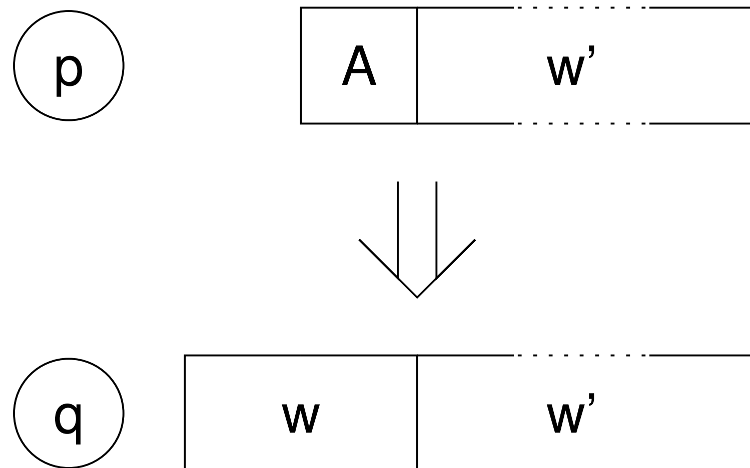
Wir betrachten jetzt nur noch PDS, die die Eigenschaften von $\mathcal{Q}$ haben.

Außerdem unterscheiden wir die Variablen, die am Anfang einer Konfiguration auftauchen, von den übrigen Variablen und bezeichnen sie als Kontrollzustände.

Definition: Ein (normiertes) Pushdown-System ist ein Tripel $(P, \Gamma, A, \Delta, c_0)$, wobei

- P eine endliche Menge von Kontrollzuständen ist;
- Γ ein endliches Stackalphabet ist;
- A eine endliche Menge von Aktionen ist;
- Δ eine endliche Menge von Regeln ist.
- $c_0 \in P \times \Gamma^*$ ist die Anfangskonfiguration.

Regeln haben die Form $pA \xrightarrow{a} qw$ mit $p, q \in P$, $A \in \Gamma$, $a \in A$, $w \in \Gamma^*$.



Dies ist die übliche Darstellung von Kellerautomaten in Formale Sprachen!

Diese Normierung ist nützlich für die Darstellung prozeduraler Programme.

Beispiel 1

Ein kleines Programm (mit Parameter $n \geq 1$):

```
bool g=true;
void main() {
    level_1();
    level_1();
    assume(g);
}
void level_n() {
    g:=not g;
}
```

```
void level_i() {
    for j:=1 to 8 do skip;
    level_{i+1}();
    level_{i+1}();
}
```

Frage: Ist `g` am Ende des Programms true?

Das vorige Beispiel hat *endlich* viele Zustände.
(Der Stack der Prozedur-Aufrufe hat Länge $\mathcal{O}(n)$.

Behandelbar durch “Inlining” (Prozeduraufruf ersetzen durch Kopie der Prozedur).

Inlining sorgt für exponenzielle Vergrößerung des Programms.

Inlining ist ineffizient: Jede Kopie eine Prozedur wird gesondert untersucht.

Inlining nicht anwendbar bei rekursiven Aufrufen.

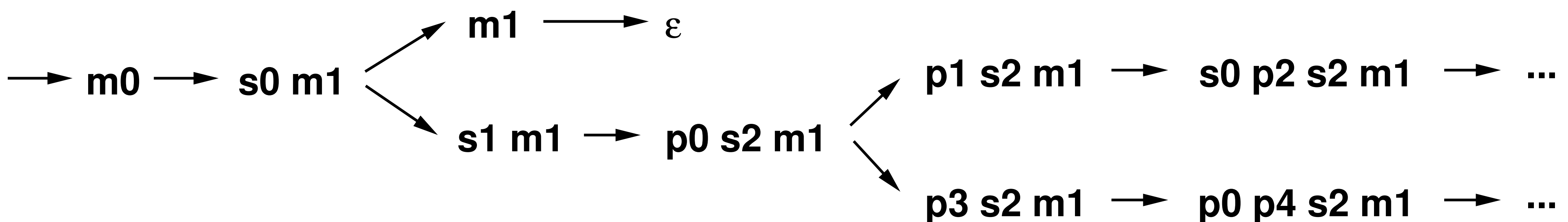
Beispiel 2: Rekursives Programm (Plotter)

procedure p;
p_0: **if** ? **then**
p_1: **call** s;
p_2: **if** ? **then call** p; **end if**;
else
p_3: **call** p;
end if
p_4: **return**

procedure s;
s_0: **if** ? **then return**; **end if**;
s_1: **call** p;
s_2: **return**;

procedure $main$;
m_0: **call** s;
m_1: **return**;

$S = \{p_0, \ldots, p_4, s_0, \ldots, s_2, m_0, m_1\}^*$, Anfangszustand m_0



Beispiel 2 hat unendlich viele Zustände.

Nicht durch Inlining behandelbar!

Nicht durch naives Auflisten der Zustände behandelbar.

Endliche Darstellung von unendlichen Zustandsmengen erforderlich.

Beispiel 3: Quicksort

```
void quicksort (int left, int right) {
   int lo,hi,piv;
   if (left >= right) return;
   piv = a[right]; lo = left; hi = right;
   while (lo <= hi) {
     if (a[hi]>piv) {
       hi = hi - 1;
     } else {
       swap a[lo],a[hi];
       lo = lo + 1;
     }
   }
   quicksort(left,hi);
   quicksort(lo,right);
}
```

Frage: Sortiert Beispiel 3 korrekt? Terminiert es immer?

Beispiel 3 kann man nicht ansehen, wie viele Zustände es hat:

- *endlich* viele, falls es korrekt ist
- womöglich *unendlich* viele, falls es einen Fehler hat

Terminierung nur durch direkte Behandlung unendlicher Mengen feststellbar.

Ein Modell für prozedurale Programme

Kontrollfluss:

- sequentielles Programm (keine parallelelen Threads)
- Prozeduren
- gegenseitige Aufrufe (womöglich rekursiv)

Daten:

- globale Variablen (Einschränkung: nur endlich viel Speicher)
- lokale Variablen in jeder Prozedur (eine Kopie pro Aufruf)

Korrespondenz Programme / PDS

P seien die Belegungen der globalen Variablen

Γ enthalte Tupel der Form $(\textit{Programmzeiger}, \textit{lokale Belegung})$

Interpretation einer Konfiguration pAw:

globale Werte in p, aktuelle Prozedur mit lokalen Variablen in A

“wartende” Prozeduren in w

Regeln:

$pA \hookrightarrow qB \mathrel{\widehat{=}}$ Anweisung innerhalb einer Prozedur

$pA \hookrightarrow qBC \mathrel{\widehat{=}}$ Prozeduraufruf

$pA \hookrightarrow q\varepsilon \mathrel{\widehat{=}}$ Rückkehr aus einer Prozedur

Weitere Vorgehensweise

Wir gehen in drei Stufen vor, die aufeinander aufbauen:

Erst kümmern wir uns um das Erreichbarkeitsproblem für PDS.

Daraus entwickeln wir eine Methode für LTL-Model-Checking.

Daraus wiederum ergibt sich die Entscheidbarkeit von CTL*.

4.1: Erreichbarkeit in PDS

Erreichbarkeit

Sei $\mathcal{P}$ ein PDS und seien c, c' zwei Konfigurationen.

Anmerkung: Im Folgenden ignorieren wir zunächst die Aktionen auf den Regeln.

Frage: Gilt $c \Rightarrow^* c'$ in $\mathcal{T}_\mathcal{P}$?

Methode: Generalisierung des CYK-Algorithmus

Verallgemeinerung: Sei C eine Menge von Konfigurationen. Wir berechnen die Menge $pre^*(C) = \{\, c' \mid \exists c \in C \colon c' \Rightarrow^* c \,\}$ der Vorgänger von C.

Endliche Automaten

Um (unendliche) Mengen von Konfigurationen darzustellen, benutzen wir endliche Automaten.

Sei $\mathcal{P} = (P, \Gamma, A, \Delta, c_0)$ ein PDS. Wir nennen $\mathcal{A} = (\Gamma, Q, P, \delta, F)$ einen $\mathcal{P}$-Automaten.

Eingabealphabet von $\mathcal{A}$ ist das Stackalphabet Γ.

“Anfangszustände” von $\mathcal{A}$ sind die Kontrollzustände P.

$\mathcal{A}$ akzeptiert die Konfiguration pw, falls man bei p beginnend die Eingabe w lesen kann und an einem Endzustand anlangt.

Sei $\mathcal{L}(\mathcal{A})$ die Menge der von $\mathcal{A}$ akzeptierten Konfigurationen.

Eine Menge C (von Konfigurationen) heißt regulär gdw. sie von einem $\mathcal{P}$-Automaten akzeptiert wird.

Ein Automat heißt normal, falls keine Kanten in Zustände aus P führen.

Bemerkung: Im Folgenden unterscheiden wir die transitiven Erreichbarkeitsrelationen durch unterschiedliche Pfeile:

$pw \Rightarrow^* p'w'$ (im PDS $\mathcal{P}$) bzw. $p \xrightarrow{w} q$ (in $\mathcal{P}$-Automaten)

Erreichbarkeit in PDS

Bekanntes Resultat (Büchi 1964):

Sei C eine reguläre Menge und $\mathcal{A}$ ein (normaler) $\mathcal{P}$-Automat, der C akzeptiert.

Ist C regulär, so auch $pre^*(C)$.

Ist C regulär, so kann $\mathcal{A}$ in einen Automaten transformiert werden, der $pre^*(C)$ akzeptiert.

Vorgehensweise

Sättigungsregel: Erweitere $\mathcal{A}$ wie folgt um Transitionen:

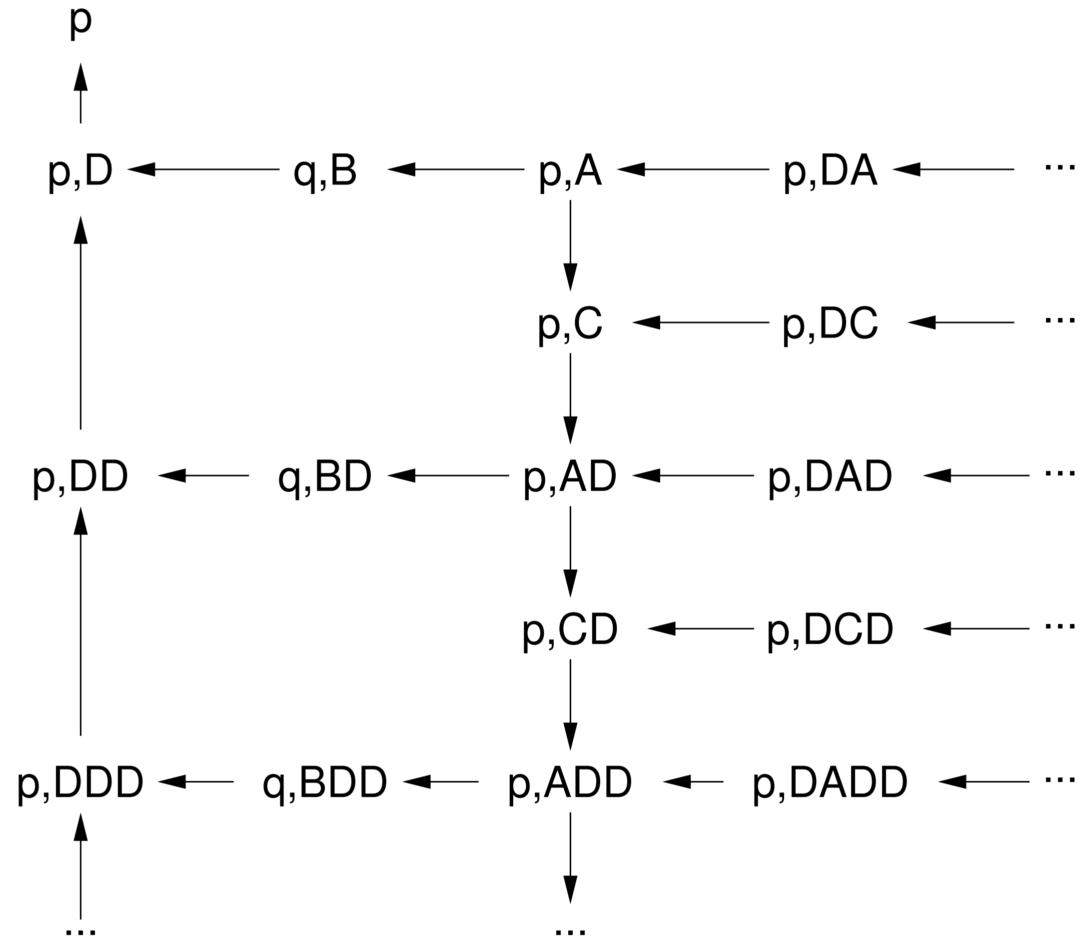
Falls $q \xrightarrow{w} r$ im Automaten $\mathcal{A}$ geht und $pA \hookrightarrow qw$ eine Regel ist, füge die Transition (p, A, r) hinzu.

Wiederhole dies, bis keine Transition mehr hinzugefügt werden kann.

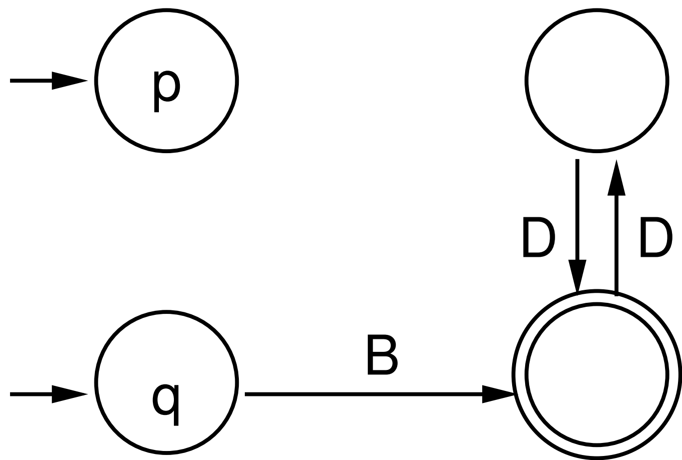
Am Ende akzeptiert der Automat $pre^*(C)$.

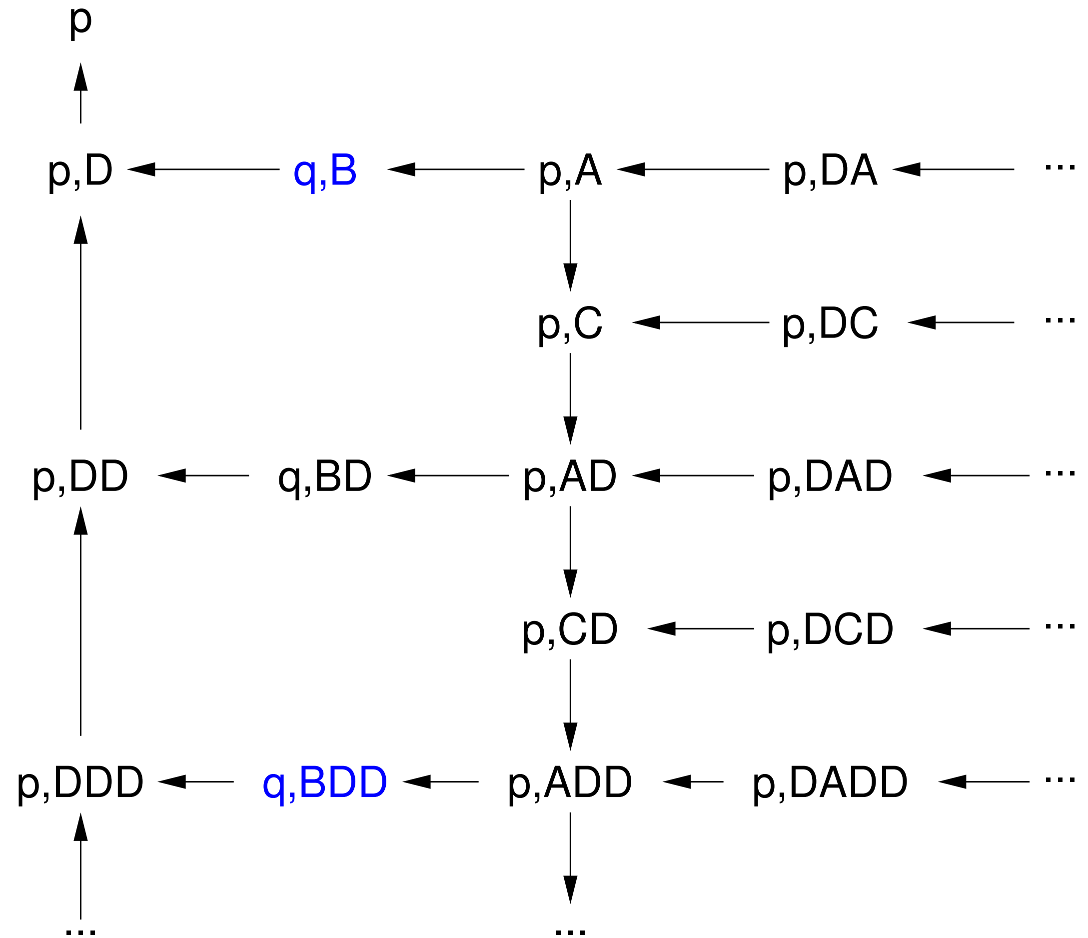
Beispiel: Ein PDS und seine Kripke-Struktur

$pA \longrightarrow qB$
$pA \longrightarrow pC$
$qB \longrightarrow pD$
$pC \longrightarrow pAD$
$pD \longrightarrow p\varepsilon$

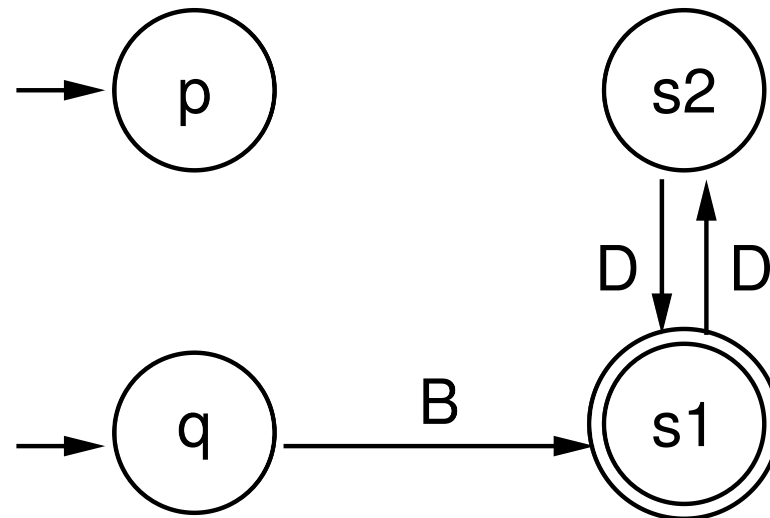


Automat $\mathcal{A}$ für C



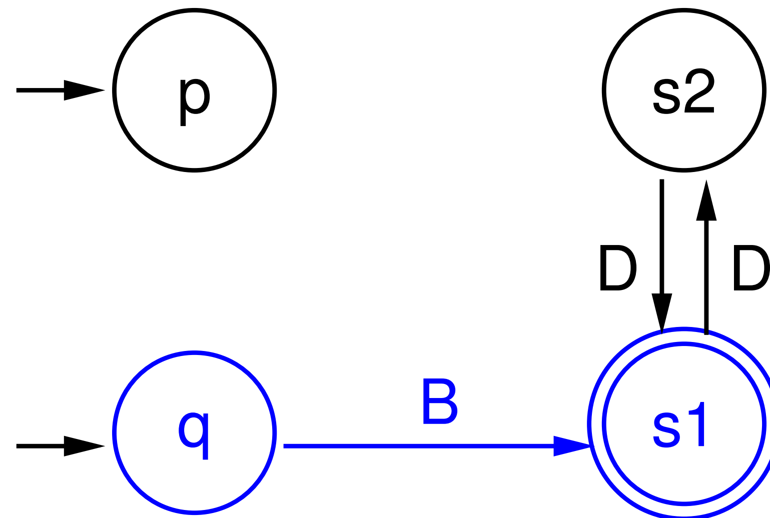


Erweitere $\mathcal{A}$



Erweitere $\mathcal{A}$

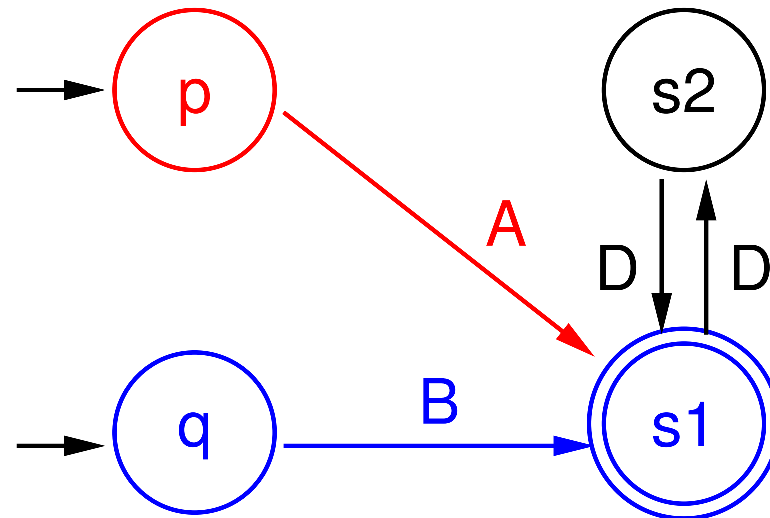
Wenn die rechte Seite einer Regel gelesen werden kann,



Regel: $pA \longrightarrow qB$ Pfad: $q \overset{B}{\longrightarrow} s_1$

Erweitere $\mathcal{A}$

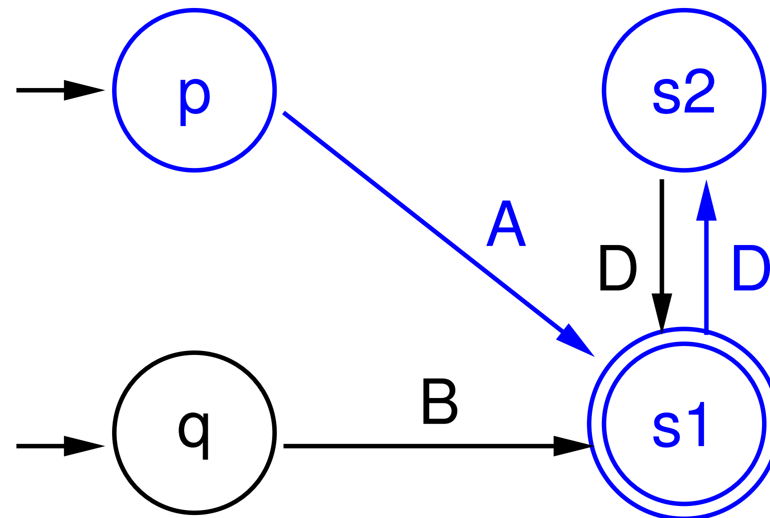
Wenn die rechte Seite einer Regel gelesen werden kann, ergänze die linke Seite.



Regel: $pA \longrightarrow qB$ Pfad: $q \xrightarrow{B} s_1$ Neuer Pfad: $p \xrightarrow{A} s_1$

Erweitere $\mathcal{A}$

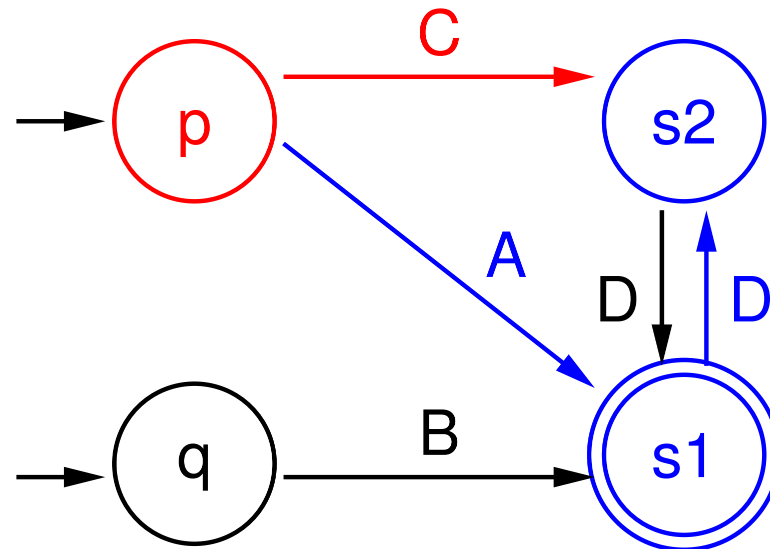
Wenn die rechte Seite einer Regel gelesen werden kann,



Regel: $pC \longrightarrow pAD$ Pfad: $p \xrightarrow{A} s_1 \xrightarrow{D} s_2$

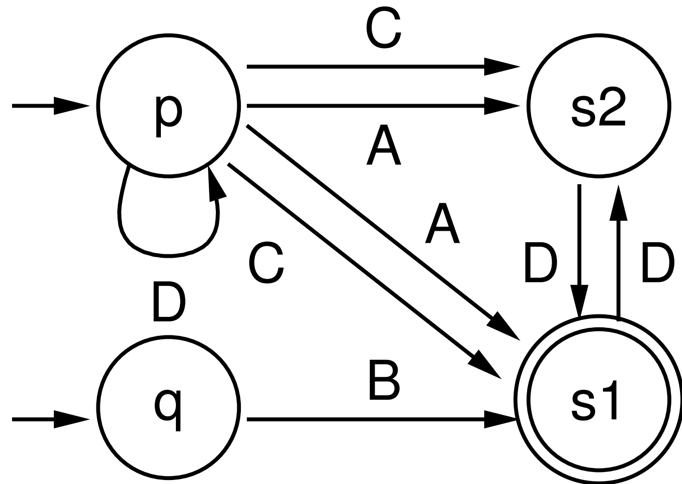
Erweitere $\mathcal{A}$

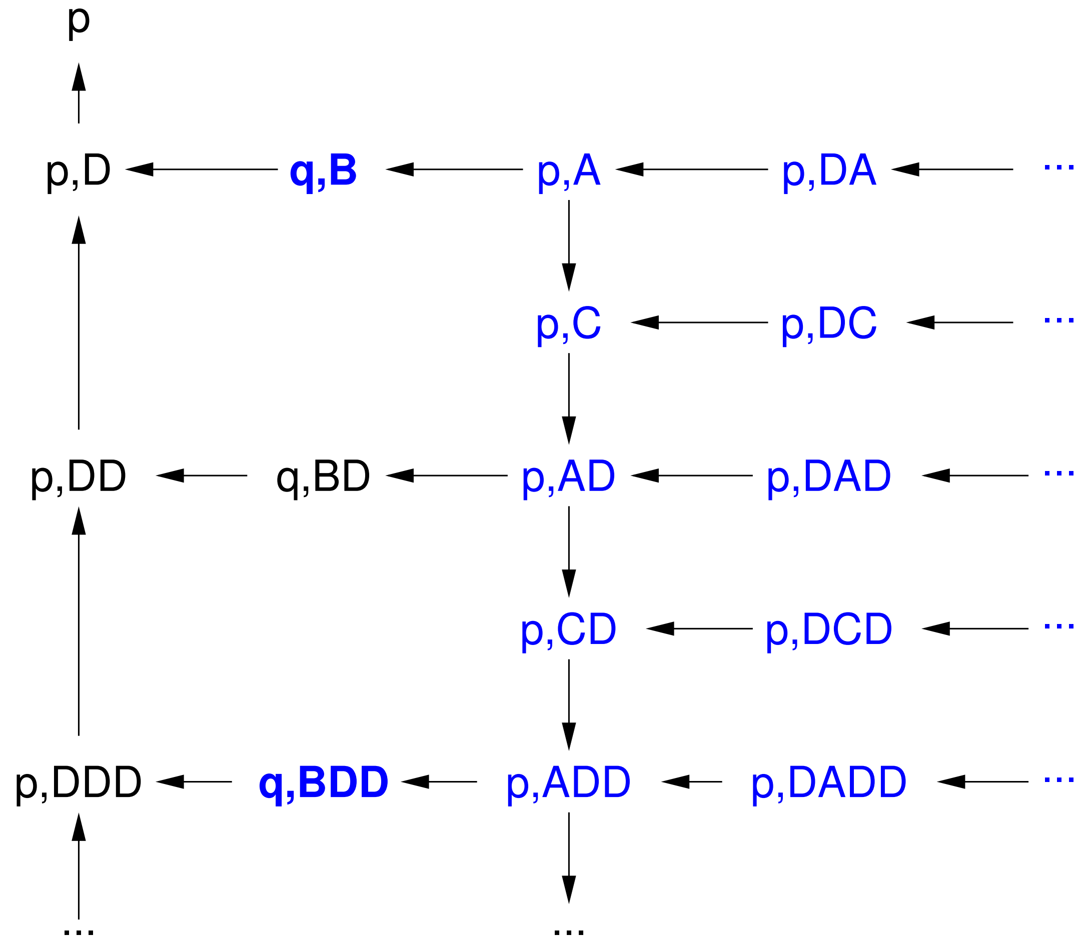
Wenn die rechte Seite einer Regel gelesen werden kann, ergänze die linke Seite.



Regel: $pC \longrightarrow pAD$ Pfad: $p \xrightarrow{A} s_1 \xrightarrow{D} s_2$ Neuer Pfad: $p \xrightarrow{C} s_2$

Endergebnis





Aufwand:

$\mathcal{O}(|Q|^2 \cdot |\Delta|)$ Zeit.

Beweis der Korrektheit

Wir zeigen (für pre^*):

Sei $\mathcal{B}$ der Automat, der aus $\mathcal{A}$ durch die Sättigungsregel entsteht. Dann ist $\mathcal{L}(\mathcal{B}) = pre^*(C)$.

1. Teil: Terminierung

Die Anwendung der Sättigungsregel terminiert, da man nur endlich viele Transitionen hinzufügen kann.

2. Teil: $pre^*(C) \subseteq \mathcal{L}(\mathcal{B})$

Sei $c \in pre^*(C)$ und $c' \in C$, so dass c' von c in k Schritten erreichbar ist. Beweis durch Induktion über k (einfach).

3. Teil: $\mathcal{L}(\mathcal{B}) \subseteq pre^*(C)$

Sei $\xrightarrow[i]{}$ die Transitionsrelation des Automaten nach i-facher Anwendung der Sättigungsregel.

Wir zeigen allgemeiner: Wenn $p \xrightarrow[i]{w} q$ gilt, dann gibt es $p'w'$ mit $p' \xrightarrow[0]{w'} q$ und $pw \Rightarrow^* p'w'$; falls $q \in P$, so gilt zusätzlich $w' = \varepsilon$.

Beweis durch Induktion über i: (IA mit $i = 0$ trivial)

Induktionsschritt: Sei $t = (p_1, A, q')$ die i-te hinzugefügte Transition und j die Anzahl der Male, die t im Pfad $p \xrightarrow[i]{w} q$ vorkommt.

Induktion über j: Für $j = 0$ trivial. Sei also $j > 0$.

Es gibt p_2, p', u, v, w', w_2 mit folgenden Eigenschaften:

(1) $p \xrightarrow[i-1]{u} p_1 \xrightarrow[i]{A} q' \xrightarrow[i]{v} q$ (Aufteilung des Pfads $p \xrightarrow[i]{w} q$)

(2) $p_1 A \longrightarrow p_2 w_2 \in \Delta$ (Vorraussetzung für Sättigungsregel)

(3) $p_2 \xrightarrow[i-1]{w_2} q'$ (Vorraussetzung für Sättigungsregel)

(4) $p_1 u \Rightarrow^* p_1 \varepsilon$ (Ind.-Annahme auf i)

(5) $p_2 w_2 v \Rightarrow^* p' w'$ (Ind.-Annahme auf j)

(6) $p' \xrightarrow[0]{w'} q$ (Ind.-Annahme auf j)

Die gewünschte Aussage folgt aus (1), (4), (2) und (5).
Falls $q \in P$ ist, so folgt die Aussage aus (6) und der Tatsache, dass $\mathcal{A}$ normal ist.

4.2: LTL-Model-Checking mit PDS

LTL und PDS

Sei $\mathcal{P}$ ein PDS und ϕ eine LTL-Formel.

> Frage: Erfüllt $\mathcal{P}$ ϕ, d.h. gilt $\mathcal{L}(\mathcal{K}) \subseteq [\![\phi]\!]$ für die zu $\mathcal{P}$ gehörige Kripke-Struktur $\mathcal{K}$?

Wir benutzen folgende Tatsache:

> Jede LTL-Formel lässt sich in einen Büchi-Automaten übersetzen.
> (ausführlich behandelt in Model-Checking I)

Büchi-Automaten

Ein Büchi-Automat (BA) ist ein Tupel

$$\mathcal{B} = (C, S, s_0, \Delta, F)$$

mit:

C	endliches Alphabet (Farben),
S	endliche Menge von Zuständen,
$s_0 \in S$	Anfangszustand,
$\Delta \subseteq S \times \Sigma \times S$	Übergangsrelation,
$F \subseteq S$	Akzeptanzzustände.

Bemerkungen:

Definition und graphische Darstellung genau wie bei endlichen Automaten.

Büchi-Automaten arbeiten aber mit *unendlichen Wörtern*, andere Akzeptanzbedingung.

Sprache eines Büchi-Automaten

Sei $\mathcal{B} = (\Sigma, S, s_0, \Delta, F)$ ein Büchi-Automat.

Ein Lauf von $\mathcal{B}$ über einem unendlichen Wort $\sigma \in \Sigma^\omega$ ist eine unendliche Folge von Zuständen $\rho \in S^\omega$ mit $\rho(0) = s_0$ und $(\rho(i), \sigma(i), \rho(i+1)) \in \Delta$ für $i \geq 0$.

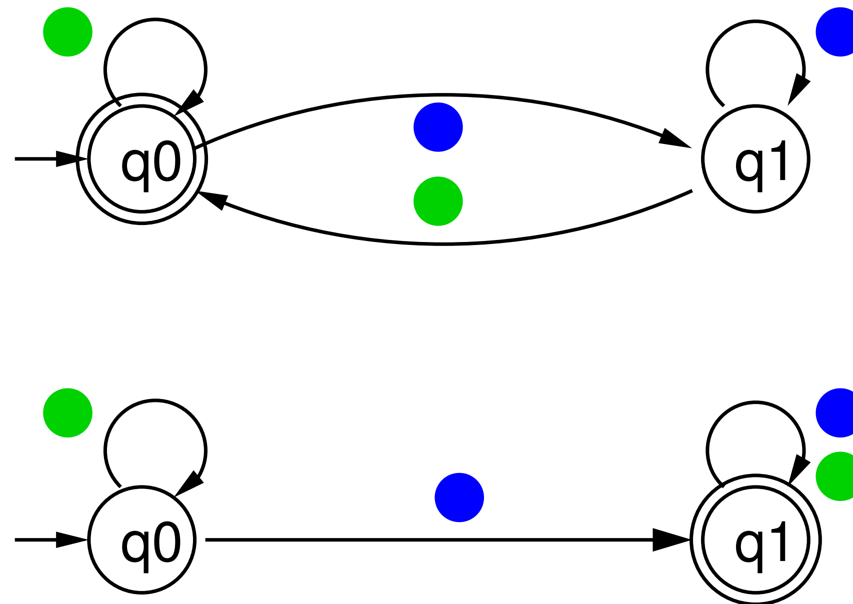
Ein Lauf ρ ist akzeptierend gdw. für unendlich viele i gilt: $\rho(i) \in F$.

$\sigma \in \Sigma^\omega$ wird von $\mathcal{B}$ akzeptiert gdw. ein akzeptierender Lauf über σ in $\mathcal{B}$ existiert.

Die Sprache von $\mathcal{B}$, geschrieben $\mathcal{L}(\mathcal{B})$, ist die Menge der von $\mathcal{B}$ akzeptierten Wörter.

Zwei Beispiel für BA

Graphische Darstellung zweier Büchi-Automaten (mit Alphabet blau, grün):



Der obere Automat akzeptiert die Sprache $[\![\mathbf{G}\,\mathbf{F}\,\textit{grün}]\!]$, der untere die Sprache $[\![\mathbf{F}\,\textit{blau}]\!]$.

Vorgehensweise

Übersetze $\neg\phi$ in einen Büchi-Automaten $\mathcal{B}$.

“Kreuze” $\mathcal{P}$ und $\mathcal{B}$.

Teste das Kreuzprodukt auf “Leerheit”.

Kreuzung von $\mathcal{P}$ und $\mathcal{B}$

Das Kreuzprodukt ist ein PDS $\mathcal{Q}$, das wie folgt gebildet wird:

Sei $\mathcal{P} = (P, \Gamma, A, \Delta, p_0 w_0)$ ein PDS.

Sei $\mathcal{B} = (A, Q, q_0, \delta, F)$ ein Büchi-Automat für $\neg\phi$.

$\mathcal{Q}$ wird wie folgt gebildet:

$\mathcal{Q} = (P \times Q, \Gamma, A, \Delta', (p_0, q_0) w_0)$, wobei

$(p, q)A \xrightarrow{a} (p', q')w \in \Delta'$ gdw.

- $pA \xrightarrow{a} p'w \in \Delta$ und
- $(q, a, q') \in \delta$

Akzeptierende Abläufe

Sei $\mathcal{Q}$ ein Ablauf von ρ, so dass $\rho(i) = (p_i, q_i)w_i$ für $i \geq 0$.

Wir nennen ρ akzeptierend, falls $q_i \in F$ für unendlich viele i.

$\mathcal{Q}$ hat folgende wichtige Eigenschaft:

> $\mathcal{P}$ erfüllt ϕ *nicht* gdw. es in $\mathcal{Q}$ einen akzeptierenden Ablauf ρ gibt.

Beweis: (einfach)

> Mit $\rho_\mathcal{P}$ bezeichnen wir die “Projektion” von ρ auf $\mathcal{P}$, d.h. $\rho_\mathcal{P}(i) := p_i w_i$, und mit $\rho_\mathcal{B}$ die Projektion auf $\mathcal{B}$, d.h. $\rho_\mathcal{B}(i) = q_i$.
>
> $\rho_\mathcal{P}$ ist ein Ablauf von $\mathcal{P}$, und $\rho_\mathcal{B}$ ist ein akzeptierender Lauf in $\mathcal{B}$ für dieselbe Farbfolge, der ϕ nicht erfüllt.

Charakterisierung akzeptierender Abläufe

Frage: Gibt es einen solchen akzeptierenden Ablauf beginnend bei $(p_0, q_0)w_0$?

Im Folgenden betrachten wir diese verallgemeinerte Fragestellung, das globale Model-Checking-Problem:

> Berechne *alle* Konfigurationen c (von $\mathcal{Q}$), so dass es einen akzeptierenden Ablauf beginnend bei c gibt.

Lemma: Für c gibt es einen solchen Ablauf gdw. es $p \in P$, $q \in Q$, $A \in \Gamma$ gibt mit folgendenden Eigenschaften:

(1) $c \Rightarrow^* (p, q)Aw$ für irgendein $w \in \Gamma^*$

(2) $(p, q)A \Rightarrow^* (p, q)Aw'$ für irgendein $w' \in \Gamma^*$, wobei

(a) der Pfad von $(p, q)A$ nach $(p, q)Aw'$ mindestens einen Schritt enthält;

(b) auf diesem Pfad mindestens ein akzeptierender Kontrollzustand vorkommt.

Schleifenköpfe

Wir nennen $(p,q)A$ einen Schleifenkopf, wenn $(p,q)A$ die Eigenschaft (2) besitzt.

Strategie:

1. Berechne die Menge der Schleifenköpfe.
(naiv: prüfe für jedes $(p,q)A$, ob $(p,q)A \in pre^*(\{\,(p,q)Aw \mid w \in \Gamma^*\,\})$)

2. Berechne die Menge
$pre^*(\{\,(p,q)Aw \mid (p,q)A \text{ ist Schleifenkopf}, w \in \Gamma^*\,\})$

3. Teste, ob $(p_0,q_0)w_0$ in der zuvor berechneten Menge enthalten ist.

Model-Checking-Tool für PDS: Moped

Moped: Modelchecker für PDS (Erreichbarkeit und LTL)

Eingabe: PDS oder boolsche Programme (mit Prozeduren)

jMoped: Erweiterung für Java

`http://www7.in.tum.de/tools/jmoped/`

4.3: CTL*-Model-Checking mit PDS

CTL* und PDS

Zur Erinnerung: CTL*-Formeln waren aus Zustandsformeln, in die Pfadformeln eingebettet sein können.

Die Pfadformeln entsprechen den LTL-Formeln!

Die Zustandsformeln sind boolsche Kombinationen von Formeln der Form $\mathbf{E}\,\phi$, wobei ϕ eine Pfadformel ist.

Merke: Wir haben bereits einen Algorithmus, der alle die Konfigurationen findet, die $\mathbf{E}\,\phi$ erfüllen!

CTL*-Modelchecking für endliche Systeme

Sei $\mathcal{K}$ eine Kripke-Struktur mit *endlicher* Zustandsmenge S, Anfangszustand r, Farben C, und ϕ sei eine CTL*-Formel.

Gefragt ist, ob $\mathcal{T}_{\mathcal{K},r} \models \phi$.

Strategie:

> ϕ besteht aus Teilformeln, die entweder Zustandsformeln oder Pfadformeln sind. Zu jeder Pfadformel ψ berechnen wir $[\![\psi]\!] = \{\, s \in S \mid \mathcal{T}_{\mathcal{K},s} \models \psi \,\}$. Am Ende schauen wir, ob $r \in [\![\phi]\!]$.
>
> Für Zustandsformeln der Art p, Disjunktion, Konjunktion: einfach
>
> Für Zustandsformeln der Art $\mathbf{E}\,\psi \Rightarrow$ LTL!

Nehmen wir an, es gäbe einen globalen MC-Algorithmus, der zu einer LTL-Formel die Menge *aller* Zustände liefert, von denen aus *irgendein* Ablauf die Formel erfüllt.

Bottom-Up-Strategie:

Suche eine Teilformel $\mathbf{E}\,\psi$, so dass ψ kein $\mathbf{E}$ enthält und daher eine LTL-Formel ist. Anwendung des globalen MC-Algorithmus auf ψ liefert eine Menge $M \subseteq S$.

Ersetze $\mathcal{K}$ durch $\mathcal{K}'$. Dabei hat $\mathcal{K}'$ die Farben $C \times \{0, 1\}$. Ein Zustand mit Farbe c in $\mathcal{K}$ hat Farbe $(c, 1)$, wenn er in M ist, und $(c, 0)$ sonst.

Ersetze ϕ durch ϕ', indem

alle Vorkommen von $\mathbf{E}\,\psi$ in ϕ durch $\bigvee_{c \in C}(c, 1)$ ersetzt werden;

alle Farben c in ϕ durch $(c, 0) \vee (c, 1)$ ersetzt werden.

Setze das Verfahren mit $\mathcal{K}', \phi'$ fort.

CTL*-Modelchecking für PDS

Gleiches Prinzip wie bei endlichen Systemen.

Problem: (bei Schritt 2) Unser globaler MC-Algorithmus für PDS liefert eine *reguläre* Menge von Konfigurationen, die $\mathbf{E}\,\psi$ erfüllen.

D.h. die Gültigkeit der ‘frischen’ Grundaussage c hängt nicht nur vom Kontrollzustand und dem obersten Stackzeichen ab, sondern auch vom Stack, deshalb kann sie bei späterer Kreuzproduktbildung nicht berücksichtigt werden.

Lösung: Kodiere einen endlichen Automaten ins PDS hinein.

Vorgehensweise

Sei $\mathcal{A}$ der Automat, der beim globalen MC von $\mathbf{E}\,\psi$ entsteht.

Wandle $\mathcal{A}$ per Potenzmengenkonstruktion in einen *deterministischen*, vollständigen Automaten um, der den Stack rückwärts liest.

Sei Γ das Stackalphabet des PDS und Q die Zustände von $\mathcal{A}$. Wir modifizieren das PDS, indem wir das Stackalphabet zu $\Gamma \times Q$ erweitern.

Ziel: Im modifizierten PDS soll $p(A_0, q_0)(A_1, q_1) \dots (A_n, q_n)$ erreichbar sein gdw. $pA_1 \dots A_n$ im ursprünglichen PDS erreichar war, q_n der Anfangszustand von $\mathcal{A}$ ist und $A_n \dots A_1$ von q_0 nach q_n führt.

Anfangskonfiguration geeignet anpassen

Änderung der PDS-Regeln:

Falls $pA \hookrightarrow p'B$ im PDS, dann $p(A,q) \hookrightarrow p'(B,q)$ für alle $q \in Q$.

Falls $pA \hookrightarrow p'\varepsilon$ im PDS, dann $p(A,q) \hookrightarrow p'\varepsilon$ für alle $q \in Q$.

Falls $pA \hookrightarrow p'BC$ im PDS, dann $p(A,q) \hookrightarrow p'(B,q')(C,q)$ so dass q in $\mathcal{A}$ mit C nach q' übergeht.

Jetzt kann man festlegen: $p(A,q)w$ erfülle c gdw. q mit A in einen Zustand s übergeht, so dass $p \in s$.

Teil 5: Die Klasse PA

Einleitung

In diesem Abschnitt zeigen wir, dass folgendes Problem unentscheidbar ist:

Gegeben ein (1,G)-PRS ϕ und eine LTL-Formel ϕ, gilt $\mathcal{K}_{\mathcal{P}} \models \phi$?

Daraus folgt, dass Model-Checking für die Logiken LTL und CTL* in all den Klassen (1,G), (S,G), (P,G) und (G,G) unentscheidbar ist.

Beweisidee:

Reduktion auf das Halteproblem von Turing-Maschinen auf leerem Band

Gegeben eine solche Maschine $\mathcal{M}$ bauen wir $\mathcal{P}$ und ϕ, so dass $\mathcal{K}_{\mathcal{P}} \models \phi$ gdw. $\mathcal{M}$ hält.

Turing-Maschine

Sei $(\Sigma, Q, q_0, q_f, \#, \delta)$ eine Turing-Maschine mit

Bandalphabet Σ, Kontrollzuständen Q, Anfangszustand q_0,

Akzeptanzzustand q_f, Leerbandzeichen $\#$, Transitionen δ.

O.b.d.A. nehmen wir an, dass $\mathcal{M}$ deterministisch ist.

Außerdem sei $\# \notin \Sigma$. Wir definieren $\Sigma' := \Sigma \cup \{\#\}$.

Die Übergänge in δ sind von der Form (p, a, X, q, b) mit $p, q \in Q$, $a, b \in \Sigma$, $X \in \{L, N, R\}$.

Konstruktion des PA

Unser PA habe folgende Variablen:

$$\{S\} \cup \{\, (Q, q) \mid q \in Q \,\} \cup \{\, (X, a) \mid X \in \{L, R\},\ a \in \Sigma \,\}$$

Anfangsvariable ist S mit einer (1,P)-Regel $S \xrightarrow{init} (Q, q_0) \parallel (L, \#) \parallel (R, \#)$, die drei parallele Komponenten erzeugt (im Folgenden Q/L/R-Komponenten genannt).

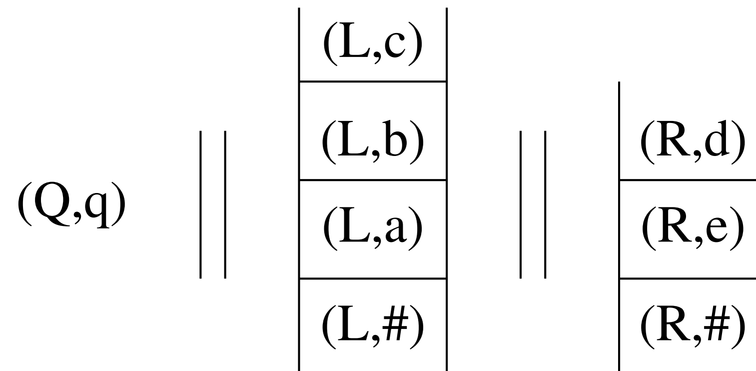
Alle weiteren Regeln sind

(1,1)-Regeln, die die Q-Komponente modifizieren

(1,S)-Regeln, die die L/R-Komponenten modifizieren

Erreichbare Terme des PA-Systems

Jeder nach S erreichbare Term besteht also aus einer einzelnen Variablen und zwei Stacks:



Jede solchen Konfiguration können wir eindeutig auf eine Konfiguration von $\mathcal{M}$ abbilden (im Beispiel der Bandinhalt $\#abcde\#$ mit Zustand q, wobei sich der Lesekopf auf dem c befindet).

Regeln des PA

Die übrigen Regeln simulieren die üblichen Operationen auf den Stacks, die Aktionen auf den Kanten "protokollieren", was geschieht.

Operationen auf der Q-Komponente:

$$(Q,q) \xrightarrow{(Q,q,q')} (Q,q') \text{ für alle } q,q' \in Q \text{ und } q \neq q_f \text{ sowie } (Q,q_f) \xrightarrow{\mathit{Halt}} \varepsilon$$

Operationen auf den L/R-Komponenten:

$$(X,a) \xrightarrow{(X,a,b)} (L,b) \quad \text{für alle } X \in \{L,R\},\ a,b \in \Sigma'$$

$$(X,a) \xrightarrow{(X,a,\varepsilon)} \varepsilon \quad \text{für alle } X \in \{L,R\},\ a \in \Sigma$$

$$(X,\#) \xrightarrow{(X,\#,\varepsilon)} (X,\#) \quad \text{für alle } X \in \{L,R\}$$

$$(X,a) \xrightarrow{(X,a,bc)} (X,c)\,.\,(X,b) \text{ für alle } X \in \{L,R\},\ a,b,c \in \Sigma'$$

Verhalten des PA

Merke: $\mathcal{P}$ kann das (einzig richtige) Verhalten von $\mathcal{M}$ nachahmen, es kann aber auch irgendetwas völlig anderes machen.

Unsere LTL-Formel ϕ wird die Bedeutung haben: Wenn sich $\mathcal{P}$ wie $\mathcal{M}$ verhält, dann erreicht sie den Akzeptanzzustand, d.h. ϕ hat die Form

$$\phi_{\mathcal{M}} \rightarrow \textit{Halt}$$

$\phi_{\mathcal{M}}$ spezifiziert, wie sich $\mathcal{P}$ verhalten sollte, um $\mathcal{M}$ zu simulieren.

Konstruktion von $\phi_{\mathcal{M}}$

Jeder Schritt von $\mathcal{M}$ modifiziert die drei Komponenten; wir verlangen, dass $\mathcal{P}$ abwechselnd auf diesen arbeitet:

$$\phi_1 := \mathit{Init} \;\wedge\; \mathbf{X}\, Q \;\wedge\; \mathbf{G}(Q \rightarrow (\mathbf{X}\, L \;\wedge\; \mathbf{X}\,\mathbf{X}\, R \;\wedge\; \mathbf{X}\,\mathbf{X}\,\mathbf{X}(Q \vee \mathit{Halt})))$$

Dabei stehen Q, L, R abkürzend für die Disjunktion aller Aktionen, die mit Q, L, R beginnen.

Für jeden Übergang $t = (p, a, X, q, b) \in \delta$ stellen wir eine Formel auf, die genau dann wahr ist, wenn eine Sequenz von drei Aktionen der Ausführung von t entspricht.

falls $X = N$: $\phi_t := (Q, p, q) \wedge \mathbf{X}(L, a, b) \wedge \mathbf{X}\,\mathbf{X} \bigvee_{c \in \Sigma'} (R, c, c)$

falls $X = L$: $\phi_t := (Q, p, q) \wedge \mathbf{X}(L, a, \varepsilon) \wedge \mathbf{X}\,\mathbf{X} \bigvee_{c \in \Sigma'} (R, c, cb)$

falls $X = R$: $\phi_t := (Q, p, q) \wedge \bigvee_{c \in \Sigma'} \left(\mathbf{X}(L, a, bc) \wedge \mathbf{X}\,\mathbf{X}(R, c, \varepsilon) \right)$

Schluss

Wir setzen jetzt

$$\phi_{\mathcal{M}} := \phi_1 \quad \wedge \quad \mathbf{G}(Q \to \bigvee_{t \in \delta} \phi_t)$$

Somit besagt ϕ, dass jeder korrekte Ablauf (es gibt nur einen) den Akzeptanzzustand erreicht. Dies ist genau dann der Fall, wenn $\mathcal{M}$, angesetzt auf leeres Band, hält, was unentscheidbar ist.

Teil 6: Petri-Netze

Literatur über Petri-Netze

Reisig, *Petrinetze: Eine Einführung*, Springer, 1986

Reisig, *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets*, Springer, 1998

Tools: PEP

`http://theoretica.informatik.uni-oldenburg.de/~pep/`

im Internet: Petri Nets World

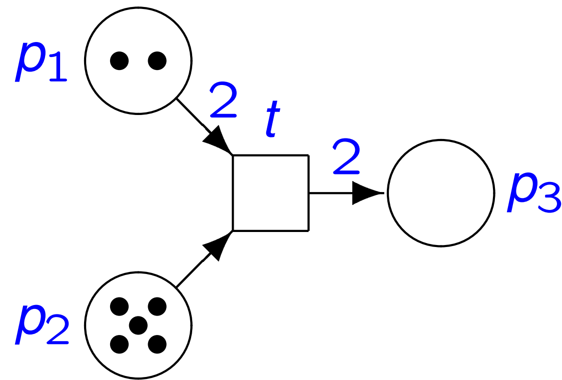
`http://www.informatik.uni-hamburg.de/TGI/PetriNets/`

Petri-Netze

Ein Petri-Netz ist ein Tupel $N = \langle P, T, F, W, M_0 \rangle$, wobei

- P eine endliche Menge von Stellen ist;
- T eine endliche Menge von Transitionen ist;
- Stellen und Transitionen disjunkt sind; ($P \cap T = \emptyset$),
- $F \subseteq (P \times T) \cup (T \times P)$ die Flussrelation ist;
- $W\colon ((P \times T) \cup (T \times P)) \to \mathbb{N}$ die Kantengewichte sind (wir nehmen $W(f) = 0$ an für alle $f \notin F$ und $W(f) > 0$ sonst;
- $M_0\colon P \to \mathbb{N}$ eine Anfangsmarkierung ist.

Petri-Netz: Beispiel



Obige graphische Darstellung entspricht dem Netz $\langle P, T, F, W, M_0 \rangle$ mit

- $P = \{p_1, p_2, p_3\}$,
- $T = \{t\}$,
- $F = \{\langle p_1, t \rangle, \langle p_2, t \rangle, \langle t, p_3 \rangle\}$,
- $W = \{\langle p_1, t \rangle \mapsto 2, \langle p_2, t \rangle \mapsto 1, \langle t, p_3 \rangle \mapsto 2\}$,
- $M_0 = \{p_1 \mapsto 2, p_2 \mapsto 5, p_3 \mapsto 0\}$.

Notation für Markierungen

Eine Markierung ist eine Abbildung $P \longrightarrow \mathbb{N}$. Markierungen sind die “Zustände” eines Petri-Netzes.

$M(p)$ bedeutet die Anzahl der Marken (“Tokens”) auf Stelle p in Markierung M.

Meistens legen wir eine Ordnung auf den Stellen fest und schreiben Markierungen als Vektoren, z.B. $M = \langle 2, 5, 0 \rangle$.

Falls keine Stelle mehr als eine Marke enthält, ist die Markierung auch als Menge darstellbar, z.B. $M = \{p_5, p_7, p_8\}$.

Falls doch, kann man eine Markierung auch als Multimenge ansehen: $M = \{p_1, p_1, p_2, p_2, p_2, p_2, p_2\}$.

Bemerkungen

Falls $\langle p, t\rangle \in F \cap (P \times T)$, heißt p Eingangsstelle von t.

Falls $\langle t, p\rangle \in F \cap (T \times P)$, heißt p Ausgangsstelle von t.

Sei $a \in P \cup T$. Die Menge ${}^\bullet a = \{a' \mid \langle a', a\rangle \in F\}$ ist der Preset von a, und die Menge $a^\bullet = \{a' \mid \langle a, a'\rangle \in F\}$ sein Postset.

Bei der graphischen Darstellung von Petri-Netzen lassen wir Kantengewichte von 1 üblicherweise weg. Markierungen werden durch schwarze Kreise oder durch Zahlen dargestellt.

Dynamik eines Petri-Netzes

Sei $\langle P, T, F, W, M_0 \rangle$ ein Petri-Netz und $M : P \to \mathbb{N}$ eine Markierung.

Schaltbedingung:

Transition $t \in T$ ist M-aktiviert (or: aktiviert in M), geschrieben $M \xrightarrow{t}$, gdw.

$$\forall p \in {}^\bullet t : M(p) \geq W(p, t).$$

Schaltregel:

Eine M-aktivierte Transition t kann schalten und in eine Nachfolge-Markierung M' übergehen ($M \xrightarrow{t} M'$), wobei

$$\forall p \in P : M'(p) = M(p) - W(p, t) + W(t, p).$$

Petri-Netze und (P,P)-PRS

Man sieht leicht, dass Petri-Netze genau die Klasse der (P,P)-PRS sind:

Jede Variable entspricht einer Stelle.

Jeder Term aus $\mathcal{T}_{\mathcal{P}}$ entspricht einer Markierung.

Jede Regel entspricht einer Transition;

jede Variable auf der linken Seite einer Marke auf einer Eingangsstelle

jede Variable auf der rechten Seite einer Marke auf eine Ausgangsstelle.

Endliche und unendliche Petri-Netze

Gegeben sei ein Petri-Netz N mit Anfangsmarkierung M_0.

$R(N)$ bezeichne die Menge der erreichbaren Markierungen, d.h.
$\{ M \mid M_0 \Longrightarrow^* M \}$.

Merke: $R(N)$ ist endlich gdw. es eine Schranke k gibt, so dass keine Stelle in irgendeiner Markierung von $R(N)$ mehr als k Marken trägt.

Falls es ein solches k gibt, heißt N auch k-beschränkt (oder einfach *beschränkt*).

Im Folgenden schauen wir einige Analysemethoden für beschränkte und unbeschränkte Petri-Netze an.

6.1: Erreichbarkeits- und Überdeckbarkeitsgraph

Erreichbarkeitsgraph

Der Erreichbarkeitsgraph eines Netzes $N = \langle P, T, F, W, M_0 \rangle$ ist ein gerichteter Graph $G = \langle V, E, v_0 \rangle$, wobei

- $V = \mathcal{R}(N)$ die Menge der Knoten ist;
- $v_0 = M_0$ der "Anfangsknoten" ist;
- $E = \left\{ \langle M, t, M' \rangle \;\middle|\; M \in V \text{ und } M \xrightarrow{t} M' \right\}$ die Kanten sind.

G ist das zu N gehörige Transitionssystem;
falls G endlich ist, so ist es bisimilar zu einem (1,1)-PRS.

Falls G endlich ist, kann man ihn durch "chaotische Suche" konstruieren.

Überdeckbarkeitsgraph

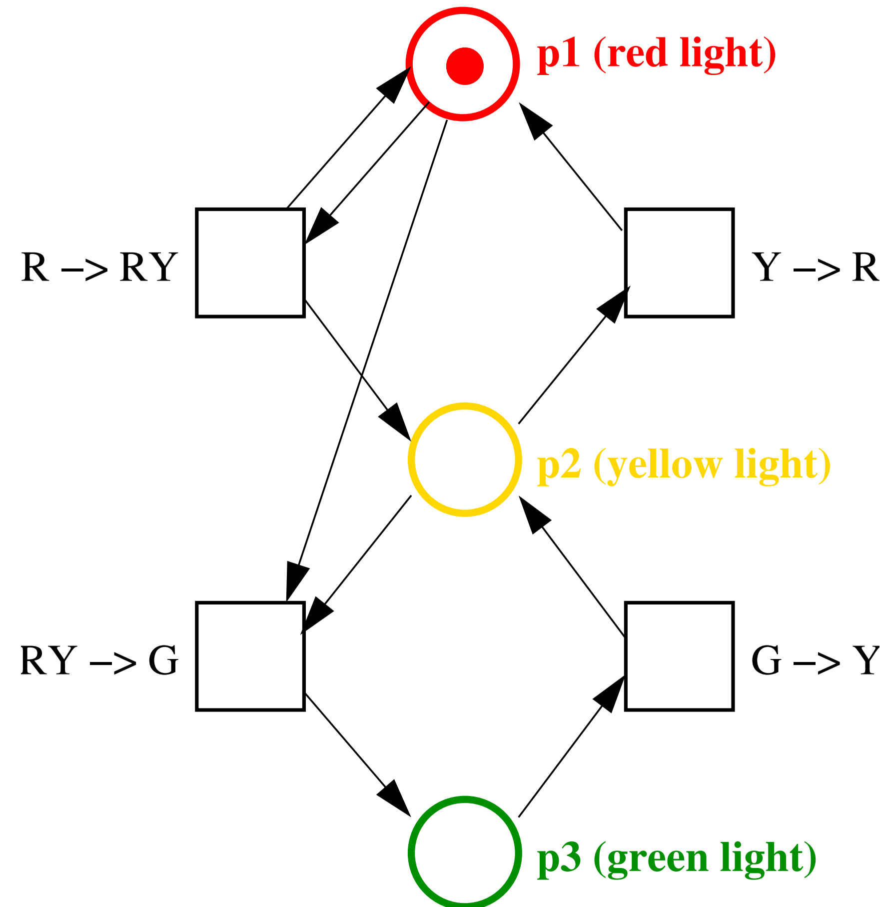
Problem: Wenn man nicht weiß, ob G endlich ist, weiß man auch nicht, ob die Konstruktion von G terminiert.

Wir stellen eine Methode vor, die immer terminiert und G liefert, falls G endlich ist, und ansonsten immer noch nützliche Information über die erreichbaren Markierungen liefert.

Diese Methode konstruiert einen so genannten Überdeckbarkeitsgraphen (engl.: coverability graph).

Beispiel

Folgendes Netz hat unendlich viele erreichbare Markierungen:



ω-Markierungen

Überdeckbarkeitsgraphen sind ähnlich wie Erreichbarkeitsgraphen; die Knoten repräsentieren Markierungen. Wir führen aber das Symbol ω für “beliebig viele” Marken ein.

Für ω gelten folgende Rechenregeln (für alle $n \in \mathbb{N}$):

$n + \omega = \omega + n = \omega$,

$\omega + \omega = \omega$,

$\omega - n = \omega$,

$0 \cdot \omega = 0$, $\omega \cdot \omega = \omega$,

$n \geq 1 \Rightarrow n \cdot \omega = \omega \cdot n = \omega$,

$n \leq \omega$, and $\omega \leq \omega$.

Bemerkung: $\omega - \omega$ ist undefiniert, wird aber nicht gebraucht.

ω-Markierungen

Definition: Eine ω-Markierung ist eine Abbildung $P \longrightarrow \mathbb{N} \cup \{\omega\}$, d.h. jede Stelle hat $n \in \mathbb{N}$ Marken oder ω, d.h. beliebig viele.

Bemerkung: Daraus folgt nicht etwa, dass irgendeine erreichbare Markierung unendlich viele Marken besitzt! Eine ω-Markierung steht vielmehr für eine unendliche *Menge* endlicher Markierungen.

Beispiel: Eine ω-Markierung $(1, \omega, 0)$ steht für die Menge der Markierungen mit einer Marke auf der ersten Stelle, keiner auf der dritten und beliebig vielen auf der zweiten.

Schaltbedingung und -regel

Die Schaltbedingung und Schaltregel von Petri-Netzen überträgt sich nahtlos auf ω-Markierungen.

Wenn eine Transition eine Eingangsstelle mit ω Marken hat, ist die Schaltregel bzgl. dieser Stelle stets erfüllt, unabhängig vom Kantengewicht ($\omega - n = \omega$).

Wenn eine Transition eine Ausganggstelle mit ω Marken hat, so ändert das Schalten der Transition nichts an der Stelle ($\omega + n = \omega$).

Überdeckung

Eine ω-Markierung M' überdeckt eine ω-Markierung M, geschrieben $M \leq M'$, gdw.

$$\forall p \in P \colon M(p) \leq M'(p).$$

Eine ω-Markierung M' überdeckt strikt eine ω-Markierung M, geschrieben $M < M'$, gdw.

$$M \leq M' \quad \text{und} \quad M' \neq M.$$

Überdeckung und Schaltfolgen (1/2)

Beobachtung: Seien M und M' zwei Markierungen mit $M \leq M'$.
Dann gilt für alle Transitionen t:

$$\text{Wenn } M \xrightarrow{t} \text{ gilt, dann auch } M' \xrightarrow{t}.$$

Mit anderen Worten, wenn M' mindestens so viele Marken wie M hat (auf jeder Stelle), dann ist in M' alles möglich, was auch in M möglich ist (Monotonizität).

Diese Beobachtung überträgt sich auf Schaltfolgen:
Schreiben wir $M \xrightarrow{t_1 t_2 \ldots t_n} M'$ für

$$\exists M_1, M_2, \ldots, M_n : M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \cdots \xrightarrow{t_n} M_n = M'.$$

Dann gilt, falls $M \xrightarrow{t_1 t_2 \ldots t_n}$ und $M \leq M'$, auch $M' \xrightarrow{t_1 t_2 \ldots t_n}$.

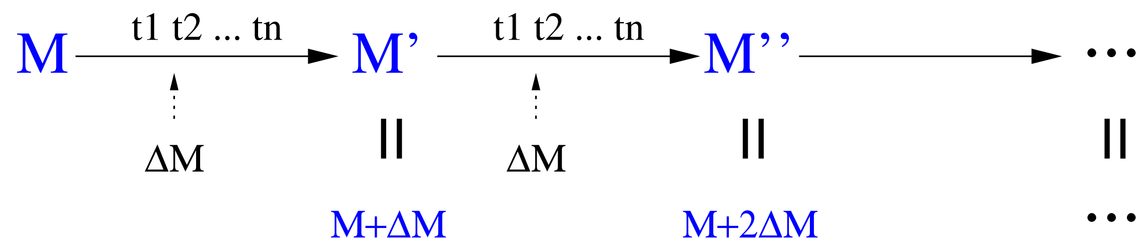
Überdeckung und Schaltfolgen (2/2)

Seien M, M' Markierungen mit $M < M'$, und es gebe eine Schaltfolge, die M in M' überführt: $M \xrightarrow{t_1 t_2 \ldots t_n} M'$

Dann gibt es M'' mit $M' \xrightarrow{t_1 t_2 \ldots t_n} M''$.

Sei $\Delta M := M' - M$ (stellenweise Differenz). Wegen $M < M'$ sind die Werte von ΔM nicht-negativ, und mindestens ein Eintrag ist positiv.

Es gilt $M'' = M' + \Delta M = M + 2\Delta M$.



Indem wir $t_1 t_2 \ldots t_n$ wiederholt schalten, können wir beliebig viele Marken auf jede Stelle legen, die in ΔM positive Werte haben.

Die Idee für Überdeckbarkeitsgraphen ist, M' durch eine ω-Markierung zu ersetzen, in der alle Stellen mit positiven Werten in ΔM durch ω ersetzt werden.

Algorithmus für Überdeckbarkeitsgraphen (1/2)

```
CovGraph(⟨P, T, F, W, M0⟩)
 1  ⟨V, E, v0⟩ := ⟨{M0}, ∅, M0⟩;
 2  Work : set := {M0};
 3  while Work ≠ ∅
 4  do select M from Work;
 5       Work := Work \ {M};
 6       for t ∈ aktiviert(M)
 7       do M′ := schalte(M, t);
 8            M′ := AddOmegas(M, t, M′, V, E);
 9            if M′ ∉ V
10              then V := V ∪ {M′}
11                     Work := Work ∪ {M′};
12            E := E ∪ {⟨M, t, M′⟩};
13  return ⟨V, E, v0⟩;
```

Der Algorithmus konstruiert die erreichbaren Markierungen durch (chaotische) Iteration, wobei in jeder Runde die Nachfolgemarkierungen irgendeiner ω-Markierung berechnet werden. aktiviert(M) seien die in M aktivierten Transitionen; schalte(M, t) die durch Schalten von t in M erzeugte Markierung.

Algorithmus für Überdeckbarkeitsgraphen (2/2)

Für die Unterroutine AddOmegas benutzen wir folgende Notation:

- $M'' \longrightarrow^* M$ gdw. der Überdeckbarkeitsgraph augenblicklich einen (womöglich leeren) Pfad von M'' nach M enthält.

ADDOMEGAS(M, t, M', V, E)

1 **repeat** *saved* $:= M'$;
2 **for** all $M'' \in V$ s.t. $M'' \longrightarrow^* M$
3 **do if** $M'' < M'$
4 **then** $M' := M' + ((M' - M'') \cdot \omega)$;
5 **until** *saved* $= M'$;
6 **return** M';

Die Routine prüft alle Vorgängermarkierungen im Graphen darauf, ob die M' strikt überdecken und daher eine Schleife gefunden wurde. Falls ja, werden die ω-Stellen entsprechend angepasst.

Eigenschaften von Überdeckungsgraphen

Die Konstruktion terminiert immer. (Beweis: Dickson's Lemma...)

Der konstruierte Graph ist identisch zum Erreichbarkeitsgraphen gdw. $R(N)$ endlich ist.

Der konstruierte Überdeckbarkeitsgraph hat folgende wichtige Eigenschafen:

> Ist M ein Knoten des Überdeckbarkeitsgraphen und $M' \leq M$, so gibt es eine erreichbare Markierung $M'' \in R(N)$, so dass $M' \leq M''$.

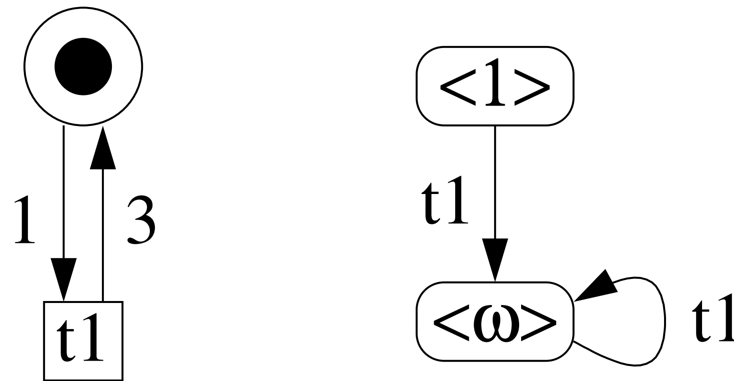
Beweis (Skizze): Diese Eigenschaft ist eine Invariante des Algorithmus, der den Überdeckbarkeitsgraphen konstruiert, d.h. gilt zu Beginn und wird bei jedem hinzugefügten Knoten aufrecht erhalten.

> Zu jeder Markierung $M \in R(N)$ gibt es einen Knoten M' des Überdeckbarkeitsgraphen mit $M \leq M'$.

Beweis (Skizze): Induktion über die Länge der Schaltfolge von M_0 zu M.

Bemerkung: Die letztgenannte Eigenschaft ist eine einseitige Implikation, die umgekehrte Richtung gilt *nicht*:

Eine überdeckte Markierung ist nicht notwendigerweise erreichbar:

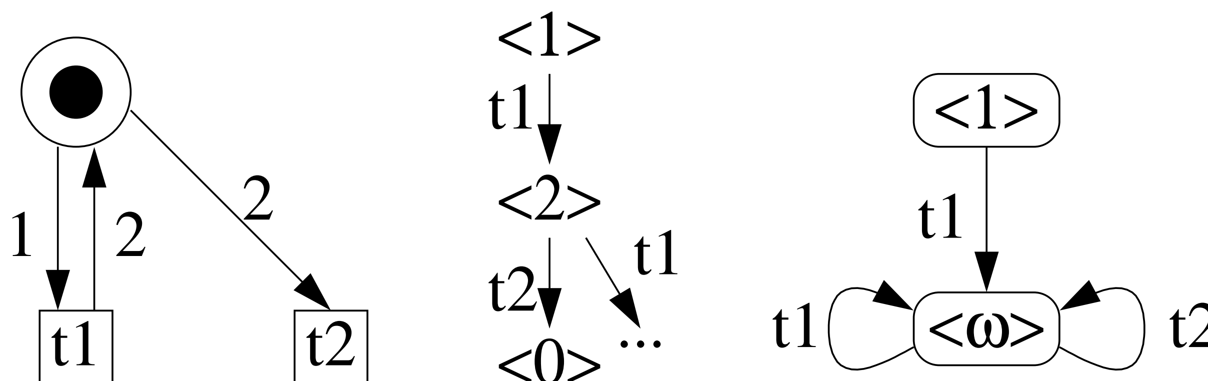


In diesem Netz sind nur Markierungen mit *ungeraden* Marken erreichbar, aber *gerade* Anzahlen sind ebenfalls überdeckt.

Überdeckbarkeitsgraphen sind nicht eindeutig.

Der vorgestellte Algorithmus ist nicht-deterministisch und kann unterschiedliche (nicht isomorphe) Graphen hervorbringen. Alle haben jedoch die erwähnte “wichtige” Eigenschaft.

Eine Verklemmung (Deadlock) heißt eine Markierung, in der keine Transition mehr geschaltet werden kann. Die Existenz von Deadlocks lässt sich mithilfe von Überdeckbarkeitsgraphen nicht feststellen:



6.2: Erweiterte Petri-Netz-Modelle

Motivation

Wir betrachten zwei Erweiterungen von Petri-Netzen:

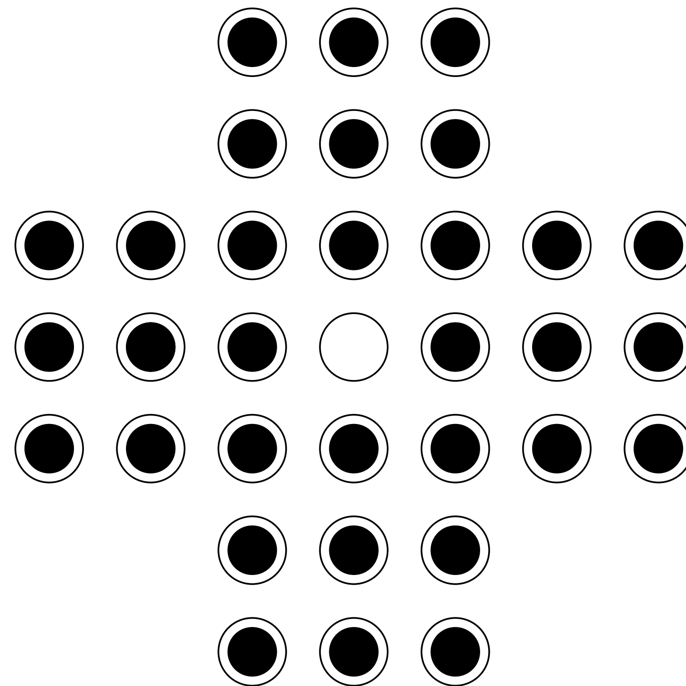
- Netze mit Kapazitäten
- High-Level-Netze

Diese Erweiterungen sind genauso ausdrucksstark wie die bisher betrachteten “einfachen” Petri-Netze, d.h. Netze mit diesen Erweiterungen kann man in bisimilare einfache Netze übersetzen.

Die Erweiterungen ermöglichen daher in erster Linie bequemere Modellierung.

Beispiel: Solitär

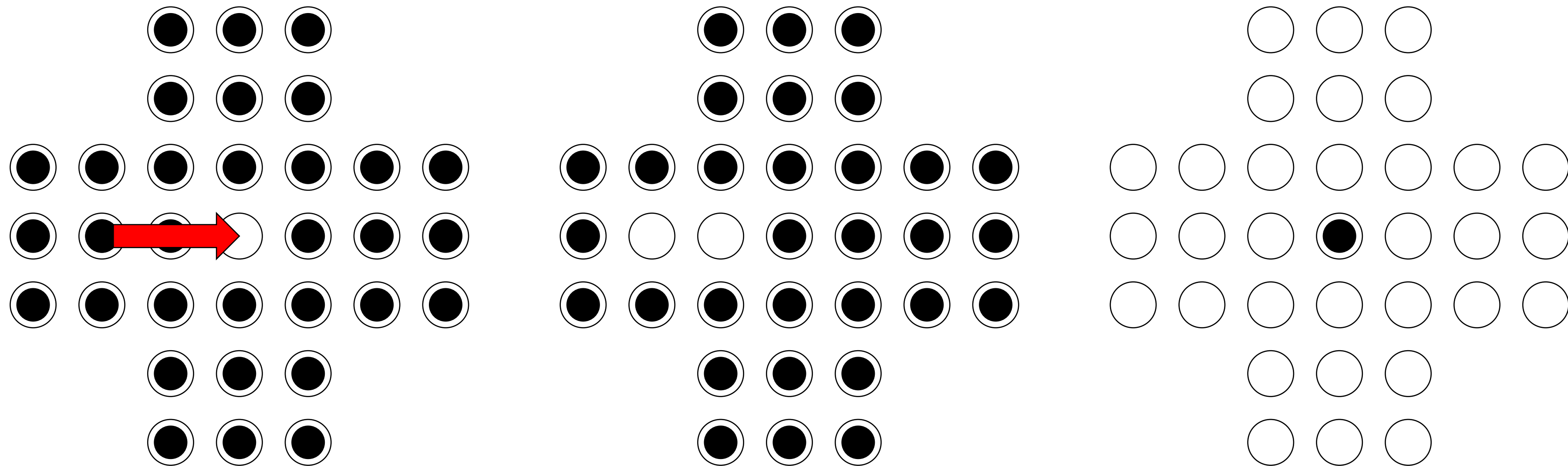
Solitär ist ein Brettspiel auf einem kreuzförmigen Spielfeld mit 33 Löchern.



Ein Loch kann leer sein oder mit einem Stein belegt sein (aber nicht mit mehreren Steinen). Zu Beginn sind alle Löcher bis auf das in der Mitte belegt.

Solitär: Regeln

Ein Zug besteht daraus, mit einem Stein einen anderen zu überspringen, dabei wird der übersprungene Stein entfernt (unten links/Mitte). Ziel ist, die unten rechts dargestellte Konfiguration zu erreichen.



Solitär: Versuch einer Modellierung

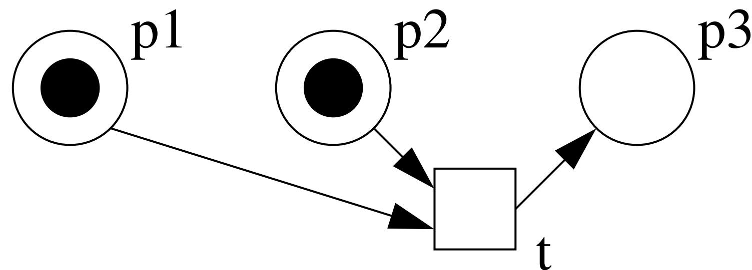
Wir versuchen, Solitär wie folgt zu modellieren:

Löcher $\widehat{=}$ Stellen

Steine $\widehat{=}$ Marken

Züge $\widehat{=}$ Transitionen

(Zu) einfache Idee:
Für je drei benachbarte Stellen eine Transition der Form:



Problem: t beachtet nicht die Bedingung, dass jedes Loch nur einen Stein fassen kann, dass also auf p_3 keine Marken liegen dürfen.

Zur Erinnerung: Die Schaltbedingung für Petri-Netze war: Transition t ist aktiviert in M genau dann, wenn

$$\forall p \in {}^{\bullet}t : M(p) \geq W(p, t)$$

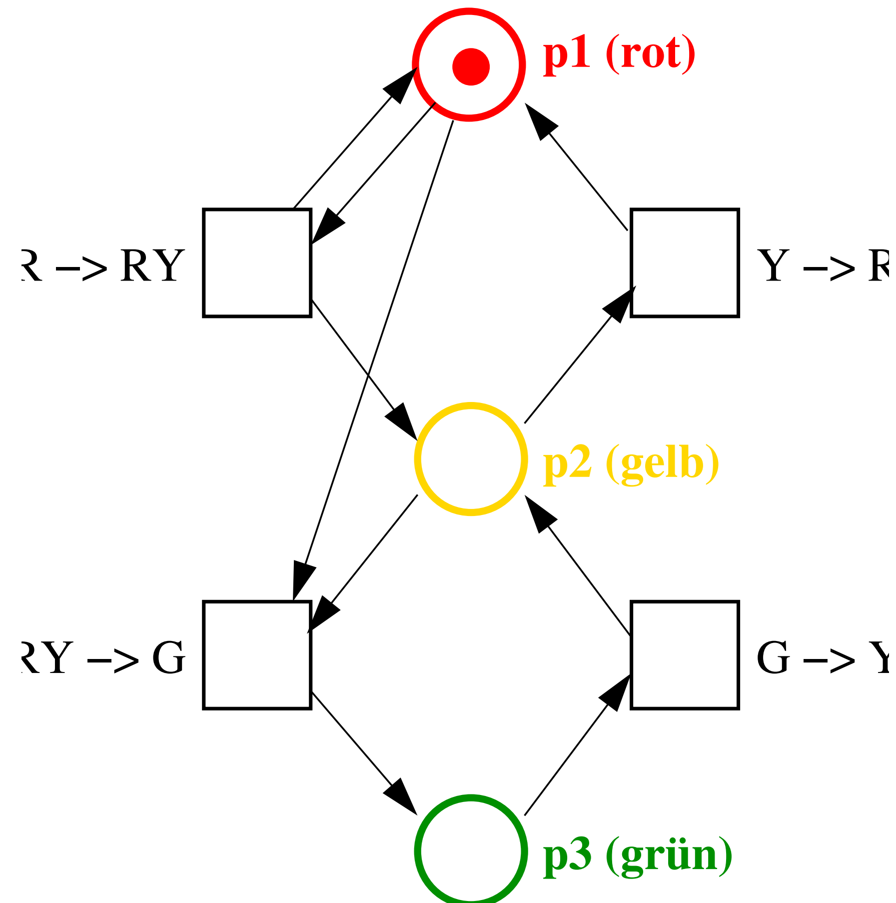
gilt.

Falls t in M aktiviert ist, ist t auch in jeder Markierung M' mit $M' \geq M$ aktiviert.

Die Schaltbedingung kann also nur die Anwesenheit von Marken prüfen, nicht aber ihre Abwesenheit.

Weiteres Beispiel: Ampel

R $\longrightarrow$ RY sollte nur schalten, wenn das gelbe Licht aus ist.



Netze mit Kapazitäten

Wir betrachten daher eine Erweiterung von Petri-Netzen, bei der die Anzahl der Marken, die maximal auf einer Stelle liegen dürfen, explizit vorgegeben werden.

Die Anzahl der Marken, die auf Stelle p liegen dürfen, nennen wir Kapazität von p.

In den Beispielen setzen wir dann die Kapazität jeder Stelle auf 1, wodurch eine klare und übersichtliche Modellierung entsteht.

Wir werden außerdem sehen, dass es eine systematische Übersetzung von Netzen mit Kapazitäten in “einfache” Petri-Netze gibt.

Ein Tupel $N = \langle P, T, F, W, K, M_0 \rangle$ heißt Kapazitäts-Petri-Netz (KPN), wobei

- P, T, F, W, M_0 wie bei normalen Petri-Netzen (PN) sind;
- $K\colon P \to (\mathbb{N} \cup \{\infty\})$ die Kapazität jeder Stelle angibt.

Die Kapazität $K(p)$ gibt an, wie viele Marken auf der Stelle p liegen dürfen. Wir modifizieren die Schaltregel, um dies sicherzustellen.

Modifizierte Schaltregel für Kapazitäten

Sei $\langle P, T, F, W, K, M_0 \rangle$ ein KPN und M eine Markierung.

Schaltbedigung: (verändert)
Transition $t \in T$ ist M-aktiviert, gdw. $\forall p \in {}^\bullet t : M(p) \geq W(p,t)$
und $\forall p \in t^\bullet : M(p) - W(p,t) + W(t,p) \leq K(p)$.

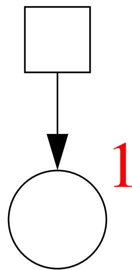
Schaltregel: (unverändert)
Das Schalten einer M-aktivierten Transition führt M über in die Nachfolger-Markierung M', wobei
$\forall p \in P : M'(p) = M(p) - W(p,t) + W(t,p)$.

Ein KPN, bei dem alle Stellen die Kapazität ∞ haben, verhält sich offensichtlich so wie ein normales PN.

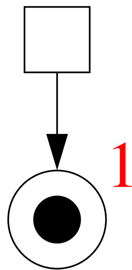
Beispiele

Im Folgenden sind Kapazitäten durch rote Zahlen neben der Stelle angedeutet. (Kapazitäten ∞ sind meistens nicht dargestellt.)

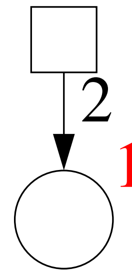
Unten gezeigt sind einige Beispiele von KPN-Transitionen, die in der jeweiligen Markierung aktiviert bzw. nicht aktiviert sind:



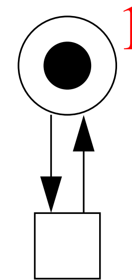
aktiviert



nicht aktiviert



nicht aktiviert



aktiviert

Komplementärstellen

Sei $\langle P, T, F, W, M_0 \rangle$ ein Petri-Netz (ohne Kapazitäten).

Zwei Stellen p und p' sind Komplementärstellen, falls es eine Zahl k gibt, so dass in allen Markierungen $M \in R(N)$ gilt: $M(p) + M(p') = k$.

Intuition: In jeder erreichbaren Markierung enthält p diejenigen der k Marken, die "nicht auf p' liegen".

Komplement-Konstruktion

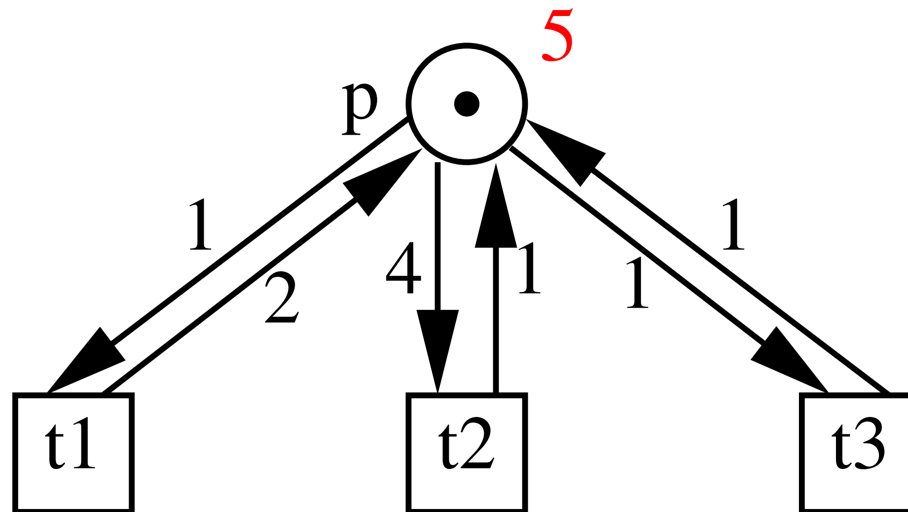
Sei $N = \langle P, T, F, W, K, M_0 \rangle$ ein KPN.

Wir konstruieren ein Netz $N' = \langle P', T', F', W', M_0' \rangle$ (*ohne* Kapazitäten) wie folgt:

- $P' = P \cup \{ p' \mid \exists p \in P \colon K(p) \neq \infty \}$
  (d.h. eine zusätzliche Stelle für jede Stelle mit endlicher Kapazität);
- $T' = T$ (unverändert);
- F' und W' sind gegenüber F und W wie folgt erweitert: Für alle $t \in T$ und $p \in P$ mit $K(p) \neq \infty$ sei $\Delta_{p,t} := W(t,p) - W(p,t)$. Dann:
  - $(p', t) \in F'$ und $W'(p', t) = \Delta_{p,t}$, falls $\Delta_{p,t} > 0$;
  - $(t, p') \in F'$ und $W'(t, p') = -\Delta_{p,t}$, falls $\Delta_{p,t} < 0$.
- $M_0'(p) = M_0(p)$ für alle $p \in P$;
  $M_0'(p') = K(p) - M_0(p)$ für alle $p \in P$ mit $K(p) \neq \infty$.

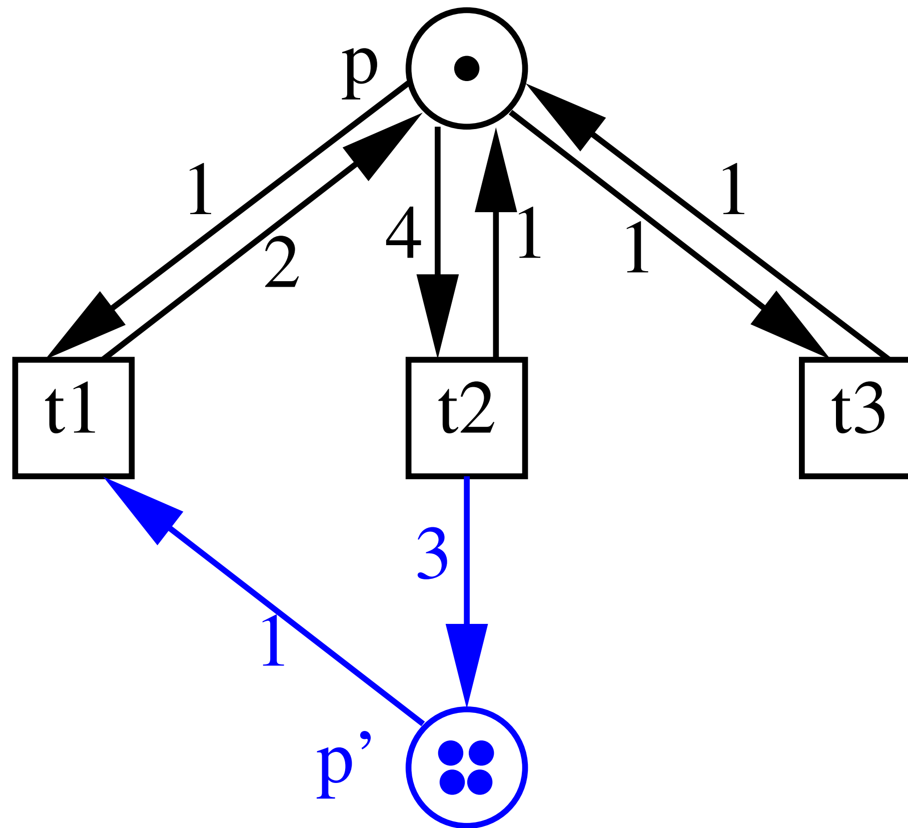
Beispiel (1/2)

Netz *N* mit Kapazitäten:



Beispiel (2/2)

Zugehöriges Netz N' ohne Kapazitäten:



Eigenschaften der Komplementkonstruktion

Definition: Ist M' eine Markierung von N', so schreiben wir $M'|_P$ für die Restriktion von M auf die Stellen in P. $M'|_P$ ist also eine Markierung von N.

N' hat folgende Eigenschaften:

- Die Paare p und p' (wobei p' die wegen p hinzugefügte Stelle ist) sind Komplementärstellen.
- Eine Markierung M' ist in N' erreichbar gdw. $M'|_P$ in N erreichbar ist.

Beweis: Man zeigt, dass dies für die Anfangsmarkierung gilt und unter Schalten einer Transition invariant bleibt.

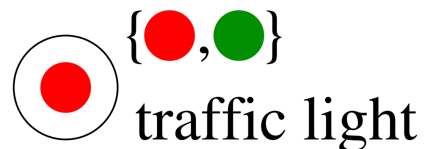
High-Level-Netze

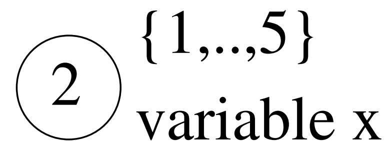
Wir betrachten jetzt eine zweite Erweiterung. Bis jetzt haben wir Marken stets als als “schwarze Kreise” betrachtet, die keine Information mit sich tragen (nur ihre Anzahl und Verteilung ist wichtig).

Wenn wir *farbige* Marken zulassen, könnten wir z.B. eine Fußgänger-Ampel mit nur einer Stelle und unterschiedlich gefärbten Marken darstellen:

Noch allgemeiner können wir uns beliebige Werte als (Beschriftung von) Marken denken, z.B. um Integer-Variablen zu modellieren:

Eine allgemeine Lösung ist, jeder Stelle einen Typ zuzuordnen, der angibt, welche Marken auf der Stelle erlaubt sind:



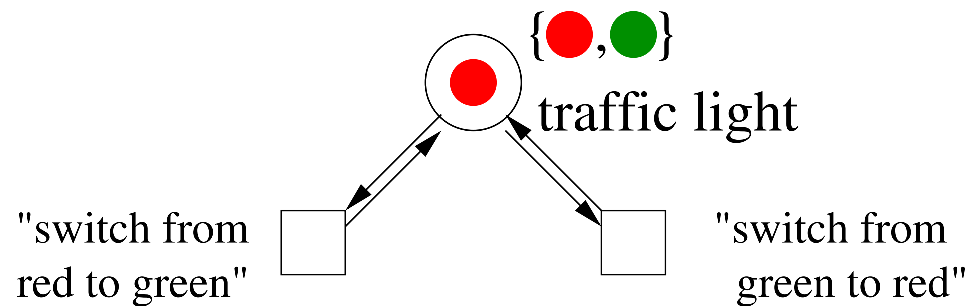


Im allgemeinen kann eine Stelle dann eine Multimenge ihres Typs enthalten.
(Multimenge = Menge, in der Werte mehrfach vorkommen können)

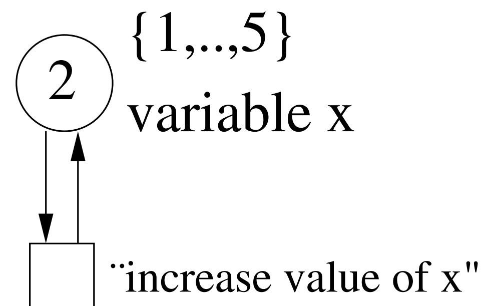
Transitionen von High-level-Netzen

Wir lassen zu, dass Transitionen über die Werte der Marken reden können:

Ampel umschalten:



Variable um eins erhöhen:



High-Level-Netze

(Das sieht komplizierter aus, als es eigentlich ist. . .)

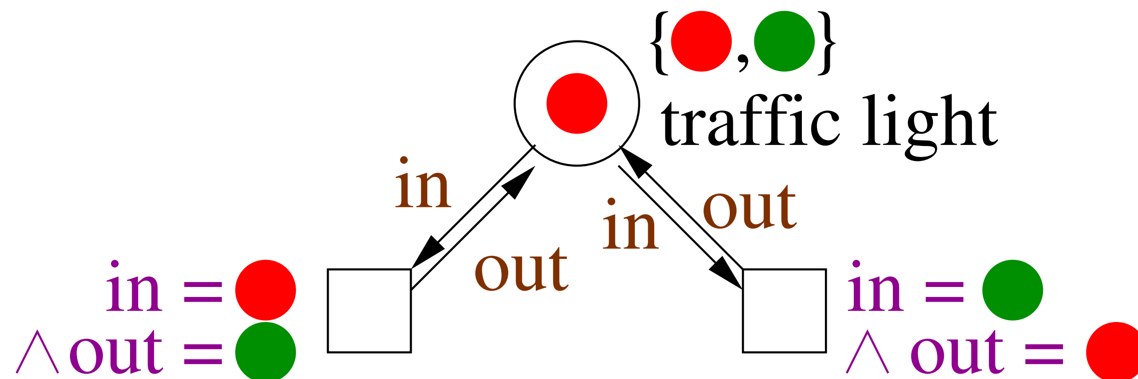
Ein High-Level-Netz (HL-Netz) ist ein Tupel $N = \langle P, T, F, W, V, S, C, M_0 \rangle$, wobei

- P, T, F, W wie üblich sind;
- V eine *endliche* Menge erlaubter Werte ist;
- $S\colon P \to 2^V$ die Typzuweisung der Stellen vornimmt;
- $(C_t)_{t \in T}$ die Schaltbedingungen der Transitionen sind (siehe nächste Folie);
- $M_0\colon P \times V \to \mathbb{N}$ die Anfangsmarkierung ist.
  Dabei gilt $M_0(p, v) = 0$ für alle $v \notin S(p)$.

High-Level-Transitionen

Die Schaltbedingungen entscheiden, welche Marken aus den Eingangsstellen abgezogen und welche auf die Ausgangsstellen geschoben werden.

Die Kanten zwischen Stellen und Transitionen werden mit Namen von Variablen versehen, die für die Werte der ein- bzw. ausfließenden Marken stehen. Die Schaltbedingung ist dann eine Bedingung, die die zulässigen Werte der Variablen angibt, z.B.:



Bemerkungen

Wenn eine Schaltbedingung unter einer bestimmten Belegung der Variablen wahr wird, kann die Transition *unter dieser Belegung* schalten.

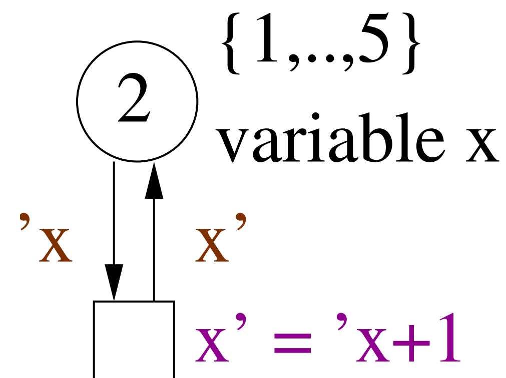
Die Belegung muss die Typen beachten, d.h. einer Variablen, die auf einer Kante von/nach p steht, können nur Werte aus $S(p)$ zugewiesen werden.

Das Schalten under einer bestimmten Belegung ist möglich, wenn jede Eingangsstelle eine Marke mit dem Wert enthält, die die Belegung der Variablen auf der Kante zwischen Stelle und Transition zuweist.

Das Schalten entfernt diese Marken und fügt den Ausgangsstellen in entsprechender Weise Marken hinzu.

Beispiele

Im untenstehenden Beispiel kann die Transition unter den Belegungen
${}'x = 1, x' = 2$, ${}'x = 2, x' = 3$, ${}'x = 3, x' = 4$, ${}'x = 4, x' = 5$ schalten.



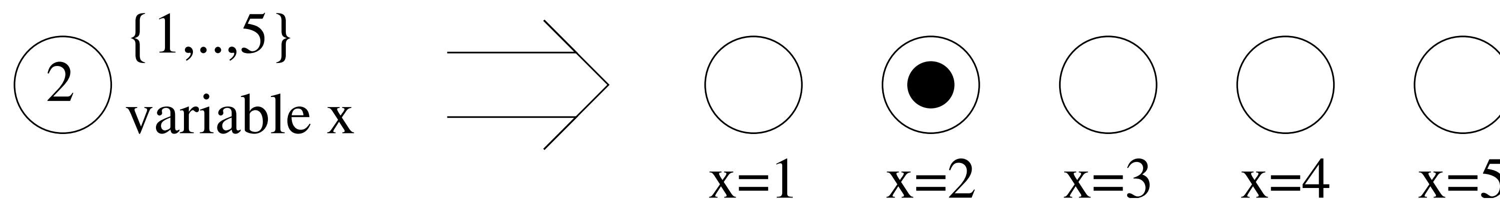
In der gezeigten Markierung kann die 2 entfernt und durch eine 3 ersetzt werden.

Übersetzung von HL-Netzen in Petri-Netze

Wir zeigen nun, dass jedes HL-Netz durch ein einfaches Petri-Netz simuliert werden kann.

Für jede HL-Stelle p und jeden Wert $v \in S(p)$ erzeugen wir eine Stelle p_v.

Falls $M_0(p, v) = k$, dann enthält p_v anfangs k Marken.

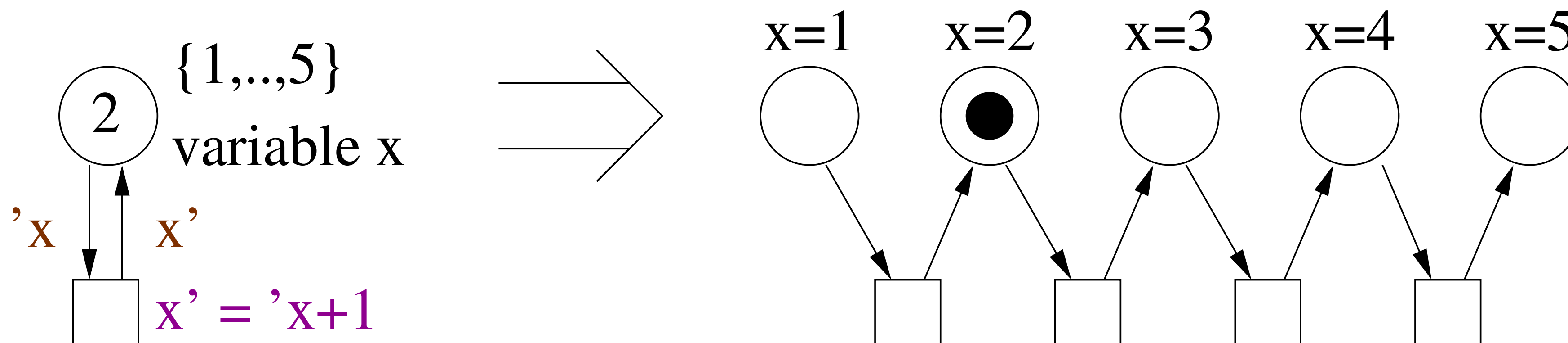


Das einfache Petri-Netz wird folgende Eigenschaft haben: Eine Markierung M (des einfachen Netzes) ist erreichbar, gdw. im HL-Netz eine Markierung M' erreichbar ist, so dass $M(p_v) = M'(p, v)$ für alle p, v.

Übersetzung von HL-Transitionen

Für jede HL-Transition t und jede Belegung b, unter der sie schalten kann, erzeugen wir eine Transition t_b.

Falls eine Kante (p, t) bzw. (t, p) mit x beschriftet ist, t unter b schalten kann und $b(x) = v$, so enthält das einfache Petri-Netz eine Kante (p_v, t_b) bzw. (t_b, p_v).



6.3: Strukturelle Analyse von Petri-Netzen

Strukturelle Analyse: Motivation

Als Analysemethoden für Petri-Netze haben wir bisher die Konstruktion von Erreichbarkeits- und Überdeckbarkeitsgraph kennengelernt.

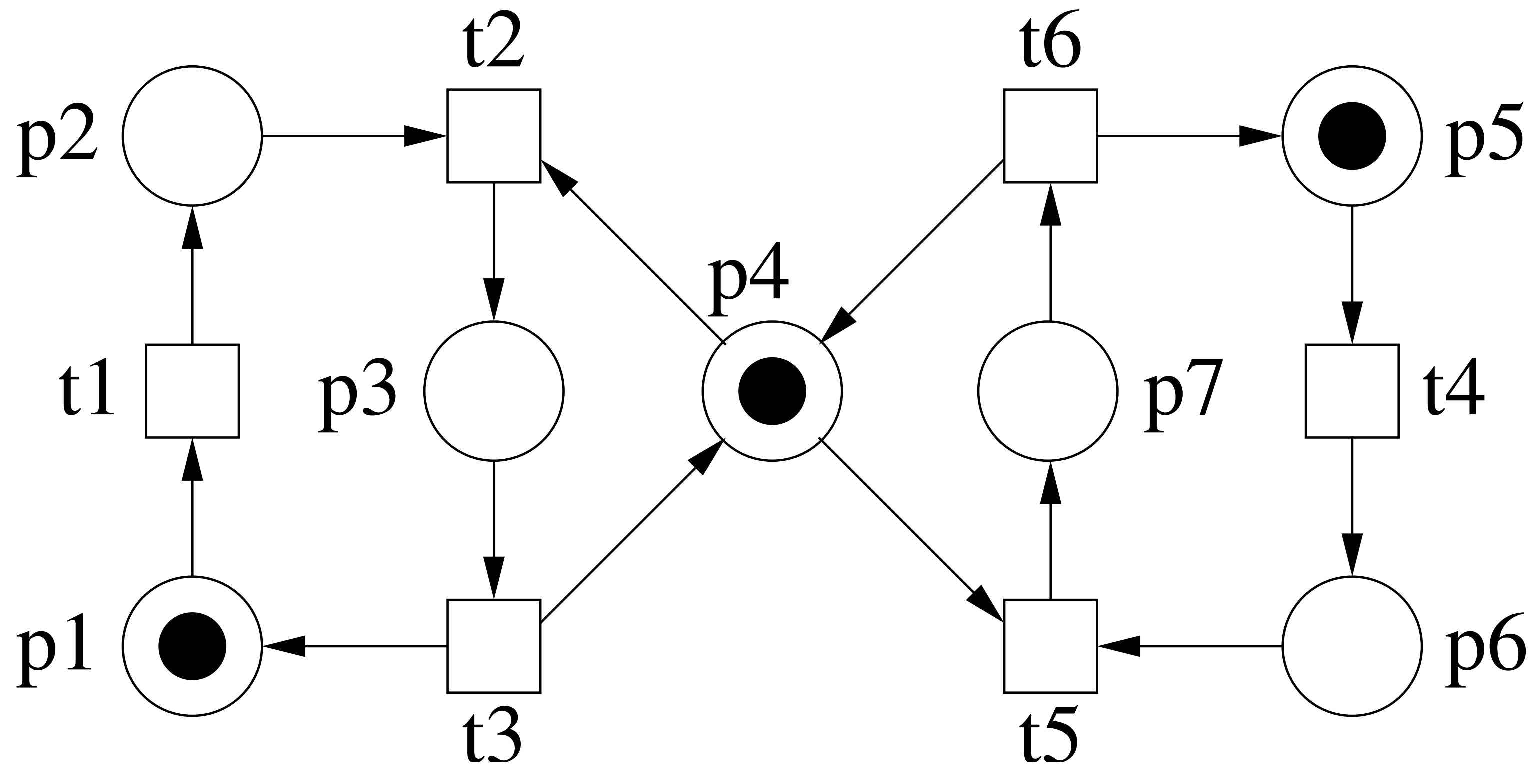
Beide erforschen den gesamten Zustandsraum des Petri-Netzes und können ggfs. sehr groß werden.

Strukturelle Analyse versucht, Aussagen über Petri-Netze zu machen, *ohne* den Zustandsraum explizit zu erforschen. Teilweise muss nur ein Teil des Netzes betrachtet werden, um nützliche Aussagen zu gewinnen. Die Methoden funktionieren i.A. auch für unbeschränkte Netze.

Wir betrachten folgende Techniken:

- (Stellen-)Invarianten
- Fallen

Beispiel 1



Inzidenz-Matrix

Sei $N = \langle P, T, F, W, M_0 \rangle$ ein Petri-Netz. $C\colon P \times T \to \mathbb{Z}$ heißt die Inzidenz-Matrix von N; ihre Zeilen entsprechen den Stellen von N und ihre Spalten den Transitionen. Eine zu $t \in T$ gehörige Spalte gibt an, welchen Effekt das Schalten von t auf die Markierung hat: $C(t, p) = W(t, p) - W(p, t)$.

Die Inzidenz-Matrix von Beispiel 1:

$$
\begin{array}{c}
\begin{pmatrix}
t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \\
-1 & 0 & 1 & 0 & 0 & 0 \\
1 & -1 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & -1 & 1 \\
0 & 0 & 0 & -1 & 0 & 1 \\
0 & 0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 1 & -1
\end{pmatrix}
\begin{array}{l}
\\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7
\end{array}
\end{array}
$$

Markierungs-Vektoren

Im Folgenden beschreiben wir Markierungen durch Spaltenvektoren. Die Anfangsmarkierung von Beispiel 1 entspricht dem Vektor $M_0 = (1\ 0\ 0\ 1\ 1\ 0\ 0)^T$.

Auf ähnliche Weise können wir einer Schaltfolge einen Vektor zuordnen, indem wir zu jeder Transition aufschreiben, wie oft sie in der Schaltfolge auftritt. Die Schaltfolge $\sigma = t_1 t_2 t_4$ in Beispiel 1 entspräche dem Vektor $u = (1\ 1\ 0\ 1\ 0\ 0)^T$.

Die durch σ entstehende Markierung ist dann $M_0 + C \cdot u$.

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Merke: Unterschiedliche Schaltfolgen können demselben Vektor zugeordnet sein, führen dann aber zurselben Markierung.

Allgemein gilt: Sei N ein Petri-Netz mit Inzidenzmatrix C, und seien M, M' zwei Markierungen von N. Dann gilt:

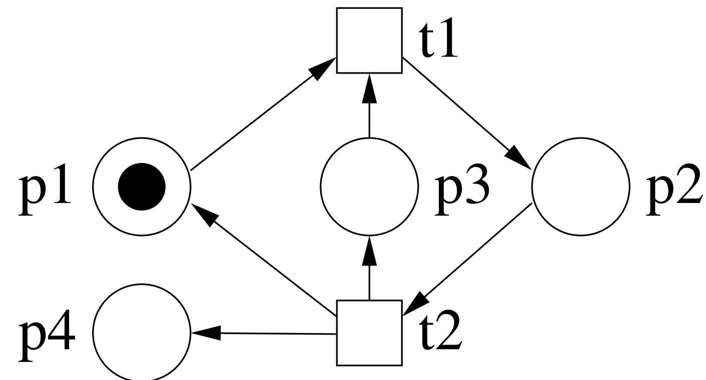
> Falls eine Schaltfolge existiert, die M in M' überführt, dann existiert ein Vektor u mit $M' = M + C \cdot u$, so dass alle Einträge von u natürliche Zahlen sind.

Im Allgemeinen gilt die umgekehrte Implikation nicht!

Das liegt am Informationsverlust der Inzidenz-Matrix; zwei Kanten (p, t) und (t, p) heben sich in der Matrix gegenseitig auf. Falls es etwa in Beispiel 1 eine Kante von p_1 nach t_3 und zurück gäbe, so wäre die Inzidenz-Matrix unverändert, aber die Markierung $\{p_3, p_6\}$ (von der vorigen Folie) wäre unerreichbar.

Beispiel 2

Ein weiteres Beispiel ohne “Hin-und-Zurück-Kanten”:



Obwohl

$$\begin{pmatrix}1\\0\\0\\0\end{pmatrix} + \begin{pmatrix}-1 & 1\\1 & -1\\-1 & 1\\0 & 1\end{pmatrix} \cdot \begin{pmatrix}1\\1\end{pmatrix} = \begin{pmatrix}1\\0\\0\\1\end{pmatrix},$$

gilt, ist keine der zu $(1\ 1)^T$ passenden Sequenzen, $t_1 t_2$ oder $t_2 t_1$, möglich.

Inzidenzmatrizen und Erreichbarkeit

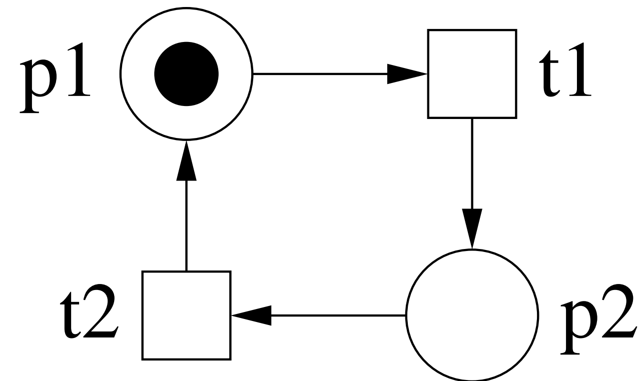
Zusammenfassung: Die Markierungen, die man durch Berechnungen der Form $M_0 + cu$ erhalten kann, sind eine Überabschätzung der tatsächlich erreichbaren Markierungen.

Wir können Inzidenzmatrizen also nicht benutzen, um *Erreichbarkeit* einer Markierung zu beweisen, manchmal jedoch, um zu zeigen, dass eine Markierung *unerreichbar* ist (wie bei Überdeckbarkeitsgraphen...).

D.h., falls $M' = M + Cu$ keine Lösung in den natürlichen Zahlen für u hat, so ist M' nicht von M aus erreichbar.

Beispiel 3

Wir betrachten das unten gezeigte Netz und die Markierung $M = (1\ 1)^T$.



$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

hat keine Lösung, also ist M nicht erreichbar.

Transitions-Invarianten

Sei N ein Netz und C seine Inzidenzmatrix. Eine (nicht-triviale) Lösung der Gleichung $Cu = \mathbf{0}$, bei der alle Einträge von u natürliche Zahlen sind, heißt Transitions-Invariante (oder: T-Invariante) von N.

Merke: Eine T-Invariante besitzt einen Eintrag für jede Transition.

In Beispiel 3 ist $u = (1\ 1)^T$ eine T-Invariante.

Eine T-Invariant weist auf eine mögliche Schleife im Netz hin, d.h. eine Schaltfolge, deren Netto-Effekt Null ist und wieder zurück zur Ausgangsmarkierung führt.

Stellen-Invarianten

Sei N ein Netz und C seine Inzidenzmatrix. Eine (nicht-triviale) Lösung der Gleichung $C^T x = \mathbf{0}$, bei der alle Einträge von x natürliche Zahlen sind, heißt Stellen-Invariante (oder: P-Invariante) von N.

Merke: Eine P-Invariante besitzt einen Eintrag für jede Stelle.

In Beispiel 1 sind $x_1 = (1\ 1\ 1\ 0\ 0\ 0\ 0)^T$, $x_2 = (0\ 0\ 1\ 1\ 0\ 0\ 1)^T$ und $x_3 = (0\ 0\ 0\ 0\ 1\ 1\ 1)^T$ P-Invarianten.

Eine P-Invariante besagt, dass die *Anzahl* der Marken in allen erreichbaren Markierungen in bestimmter Weise “invariant” bleibt (siehe nächste Folie).

Eigenschaften von P-Invarianten

Sei M eine Markierung, die durch eine Schaltfolge mit Vektor u erreichbar ist, d.h. $M = M_0 + Cu$. Sei x eine P-Invariante. Dann gilt Folgendes:

$$M^T x = (M_0 + Cu)^T x = M_0^T x + (Cu)^T x = M_0^T x + u^T C^T x = M_0^T x$$

Invariante x_2 (von der vorigen Folie) bedeutet z.B., dass alle erreichbaren Markierungen M folgende Gleichung erfüllen:

$$M(p_3) + M(p_4) + M(p_7) = M_0(p_3) + M_0(p_4) + M_0(p_7) = 1 \qquad (1)$$

Ein wichtiger Spezialfall sind P-Invarianten, in denen alle Einträge 0 oder 1 sind. Sie weisen eine Menge von Stellen aus, die zusammen stets die gleiche Anzahl von Marken tragen (in jeder erreichbaren Markierung).

Bemerkung: Jede lineare Kombination von P-Invarianten ist wieder eine P-Invariante.

P-Invarianten kann man z.B. gebrauchen, um gegenseitigen Ausschluss zu beweisen.

Beispiel: Laut Gleichung (refeq:mutex1) trägt in jeder erreichbaren Markierung genau eine der Stellen p_3, p_4 und p_7 eine Marke. Insbesondere können p_3 und p_7 nicht gleichzeitig markiert sein!

Farkas-Algorithmus

Der “Farkas-Algorithmus” (nach *J. Farkas*, 1902) kann benutzt werden, so genannte minimale P-Invarianten zu berechnen. Dies sind Invarianten, von denen aus jede andere Invariante durch lineare Kombinationen erzeugt werden kann.

Es gibt allerdings Netze, bei denen die Anzahl minimaler P-Invarianten exponentiell in der Zahl der Stellen ist. Daher hat der Farkas-Algorithmus (im schlimmsten Fall) exponentielle Laufzeit.

Farkas-Algorithmus

Eingabe: Die Inzidenzmatrix C mit n Zeilen (Stellen) und m Spalten (Transitionen).

Ausgabe: Eine Menge minimaler P-Invarianten.

Notation: $(C \mid E_n)$ beschreibt die Matrix, die entsteht, wenn die Matrix C zur Rechten mit E_n erweitert wird, d.h. der $n \times n$-Einheitsmatrix.

$D_0 := (C \mid E_n)$;

für $i := 1$ **to** m

  für alle Zeilen d_1, d_2 in D_{i-1} mit unterschiedlichen Vorzeichen bei $d_1(i)$ and $d_2(i)$

    $d := |d_2(i)| \cdot d_1 + |d_1(i)| \cdot d_2$; (* $d(i) = 0$ *)

    $d' := d/\text{gcd}(d(1), d(2), \ldots, d(m+n))$;

    erweitere D_{i-1} um d' als unterste Zeile;

  endfor;

  lösche alle Zeilen aus D_{i-1}, deren i-te Spalte

   nicht 0 ist, und nenne das Resultat D_i;

endfor;

Lösche die ersten m Spalten von D_m

Beispiel

Nehmen wir folgende Inzidenzmatrix C an:

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

$$D_0 = (C \mid E_5) = \left(\begin{array}{cccc|ccccc} -1 & 1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

Addition of the Zeilen 1 und 2, 1 und 4, 2 und 5, 4 und 5:

$$D_1 = \left(\begin{array}{cccc|ccccc} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & -2 & 1 & 0 & 0 & 1 & 0 \\ 0 & -1 & -1 & 2 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right)$$

Hinzufügen der Zeilen 3 und 4:

$$D_2 = \left(\begin{array}{cccc|ccccc} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}\right)$$

$$D_3 = D_4 = \left(\begin{array}{cccc|ccccc} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}\right)$$

Minimale P-Invarianten sind $(1, 1, 0, 0, 0)$ und $(0, 0, 0, 1, 1)$.

Beispiel mit vielen minimalen P-Invarianten

Inzidenz-matrix für ein Netz mit $2n$ Stellen:

$$C^T = \begin{pmatrix} -1 & -1 & 1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & -1 & 0 & 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -1 & -1 & 0 & 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix}$$

$(y_1, 1 - y_1, y_2, 1 - y_2, \ldots, y_n, 1 - y_n)$ ist eine Invariante für jede Kombination $y_1, y_2, \ldots, y_n \in \{0, 1\}$, es gibt also 2^n minimale P-Invarianten.

Fallen

Sei $\langle P, T, F, W, M_0 \rangle$ ein Netz. EIne Menge $S \subseteq P$ heißen Falle gdw. $S^\bullet \subseteq {}^\bullet S$.

Mit anderen Worten: Jede Transition, die eine Marke aus der Falle abzieht, tut auch mindestens eine Marke hinein.

Eine Falle S heißt markiert in der Markierung M genau dann, falls es mindestens eine Stelle $p \in S$ gibt mit $M(p) \geq 1$.

Bemerkung: Wenn eine Falle in M_0 markiert ist, so ist sie auch in allen erreichbaren Markierungen markiert.

In Beispiel 4 (siehe nächste Folie), ist $S_1 = \{nc_1, nc_2\}$ eine Falle.

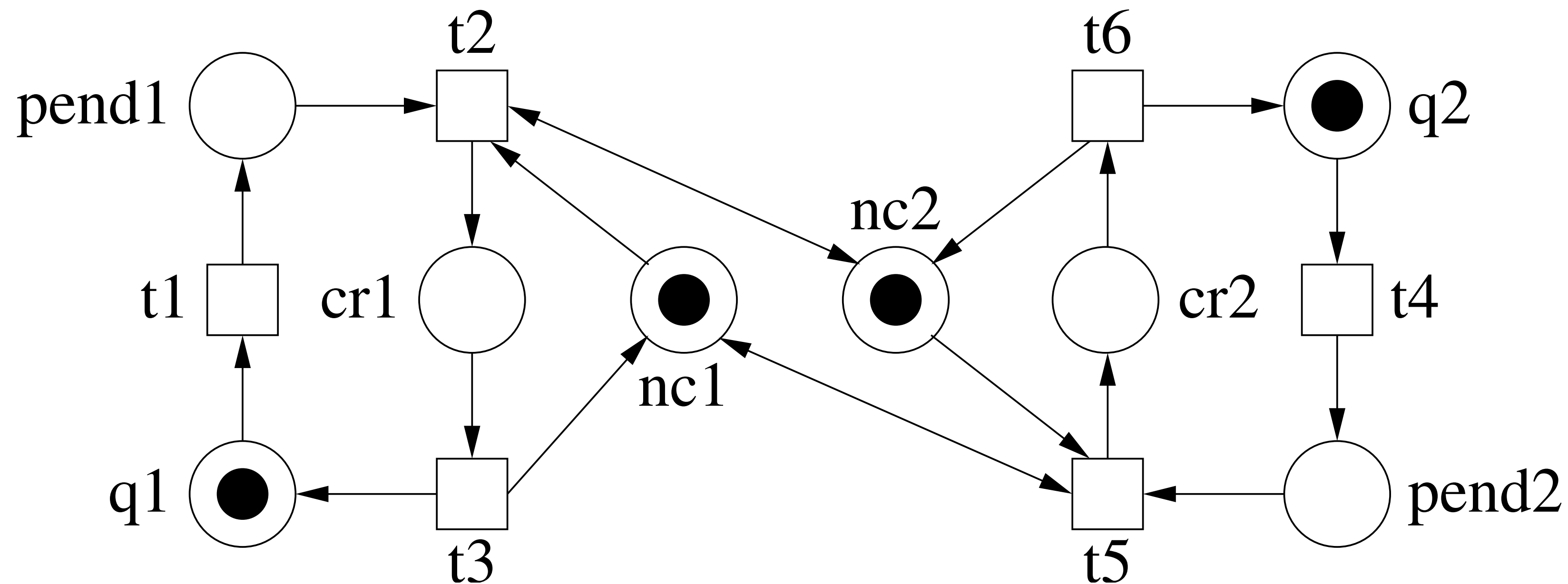
Nur t_2 und t_5 entfernen Marken aus S_1. Beide erzeugen aber auch neue Marken in S_1.

S_1 ist in M_0 markiert, daher gilt für alle erreichbaren Markierungen M Folgendes: $M(nc_1) + M(nc_2) \geq 1$

Fallen können in Zusammenhang mit P-Invarianten nützlich sein, weil sie zusätzliche Information liefern, die in der Inzidenzmatrix nicht enthalten ist.

Beispiel 4

Unten abgebildet ist ein weiteres Protokoll für wechselseitigen Ausschluss zwischen den Stellen cr_1 und cr_2:



In diesem Beispiel wird wechselseitiger Ausschluss erreicht, indem Komplementärstellen hinzugefügt (nc_1 und nc_2) und vom jeweils anderen Prozess abgefragt werden.

Arbeiten mit Fallen

In Beispiel 4 wollen wir zeigen, dass cr_1 und cr_2 nie zugleich markiert sind, dass also für alle erreichbaren Markierungen M gilt:

$$M(cr_1) + M(cr_2) \leq 1$$

Die minimalen P-Invarianten des Netzes sind wie folgt:

$$\begin{aligned}
M(q_1) + M(pend_1) + M(cr_1) &= 1 \quad (2)\\
M(q_2) + M(pend_2) + M(cr_2) &= 1 \quad (3)\\
M(cr_1) + M(nc_1) &= 1 \quad (4)\\
M(cr_2) + M(nc_2) &= 1 \quad (5)
\end{aligned}$$

Allerdings lässt sich die gewünschte Eigenschaft nicht aus (2) bis (5) herleiten.

Wie erwähnt ist $S_1 = \{nc_1, nc_2\}$ eine Falle.

S_1 ist anfänglich markiert, daher auch in allen erreichbaren Markierungen M:

$$M(nc_1) + M(nc_2) \geq 1 \tag{6}$$

Durch Addition (4) und (5) und Subtraktion von (6) erhalten wir die gewünschte Aussage $M(cr_1) + M(cr_2) \leq 1$.

6.4: Petri-Netz-Entfaltungen

Petri-Netz-Entfaltungen

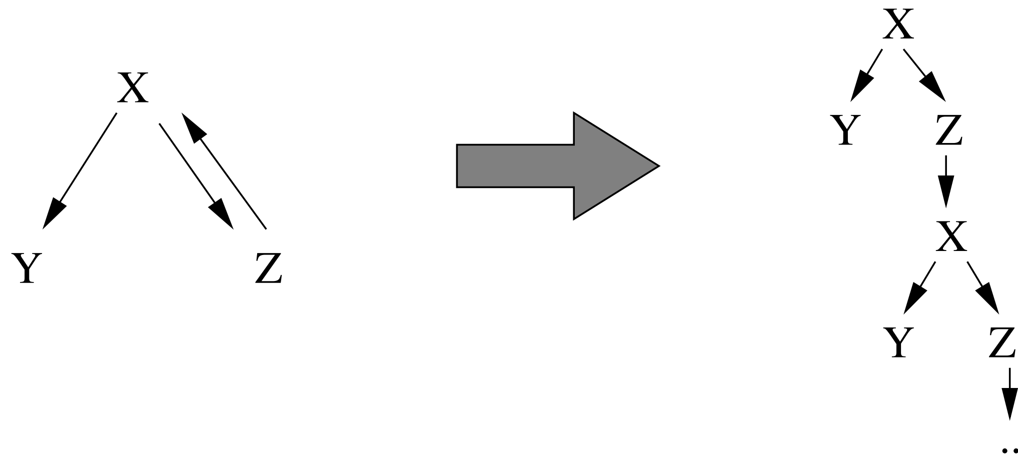
Entfaltungen werden benutzt, um die erreichbaren Markierungen eines Petri-Netzes darzustellen.

Sie eignen sich für *beschränkte* Netze! (Im Folgenden beschäftigen wir uns mit 1-beschränkten Netzen, die Technik funktioniert aber für beliebig beschränkte Netze.)

Die Darstellung der erreichbaren Markierungen erfolgt kompakt, d.h. unter Ausnutzung der inhärenten Nebenläufigkeiten. (vgl. Halbordnungstechniken in MC1)

Entfaltung für endliche Systeme

Sei $\mathcal{P}$ ein (1,1)-PRS mit Anfangsvariable X. Wir haben bereits die azyklische Entfaltung $\mathcal{T}_\mathcal{K}$ der zugehörigen Kripke-Struktur $\mathcal{K}$ definiert:

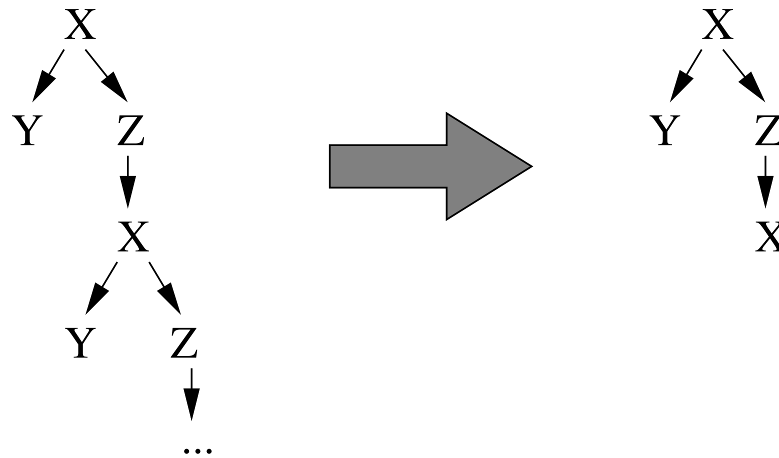


Bemerkung: $\mathcal{T}_\mathcal{K}$ kann man als eine Struktur betrachten, in der jeder Knoten mit einer Variablen von $\mathcal{P}$ beschriftet ist.

$\mathcal{T}_\mathcal{K}$ enthält alle Information über erreichbare Zustände, die auch $\mathcal{P}$ enthält. Außerdem hat $\mathcal{T}_\mathcal{K}$ eine einfachere Struktur (azyklisch, genauer: Baum). Im Allgemeinen ist $\mathcal{T}_\mathcal{K}$ jedoch *unendlich*.

Präfixe

Ein Baum $\mathcal{T}$ heiße Präfix von $\mathcal{T}_\mathcal{K}$, wenn $\mathcal{T}$ durch “Abschneiden” beliebiger Zweige von $\mathcal{T}_\mathcal{K}$ entsteht. Beispiel:



Beobachtung: Man kann immer einen *endlichen* Präfix finden, der dieselbe Information wie $\mathcal{T}_\mathcal{K}$ enthält (indem man Schleifen genau einmal “entrollt”). Einen solchen Präfix nennen wir vollständig.

Konstruktion von vollständigen Präfixen

Wir diskutieren einen Algorithmus, um einen vollständigen Präfix von $\mathcal{T}_{\mathcal{K}}$ zu erstellen, der die gleiche Erreichbarkeitsinformation wie $\mathcal{T}_{\mathcal{K}}$ enthält.

Der Algorithmus verwaltet eine Menge $\mathcal{E}$, die erreichten Konfigurationen im Präfix.

Manche Kanten im Präfix heißen Cutoffs, wir markieren sie rot.

1. Am Anfang enthält der Präfix nur eine Wurzel, die mit X beschriftet ist. Wir setzen $\mathcal{E} = \{X\}$.

2. Wir wählen einen Knoten im Präfix, der nicht Ziel einer Cutoff-Kante ist. Sei der gewählte Knoten mit Y beschriftet, und sei Z eine Variable mit $Y \to Z$, so dass es im Präfix noch keine Kante vom Y-Knoten zu einem mit Z beschrifteten Knoten gibt.

2a. Wenn es kein solches Paar Y, Z gibt, dann Ende.

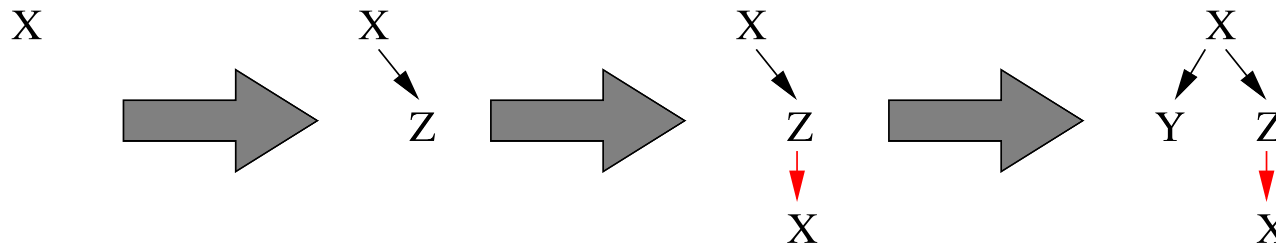
2b. Füge einen neuen, mit Z beschrifteten Knoten zum Präfix hinzu sowie eine Kante vom Y-Knoten dorthin.

2c. Falls $Z \in \mathcal{E}$, so ist die neue Kante ein Cutoff. Sonst $\mathcal{E} := \mathcal{E} \cup \{Z\}$.

3. Weiter bei 1.

Beispiel

Schrittweise Konstruktion des Präfix für voriges Beispiel:



Beobachtung (1): Ein vollständiger Präfix enthält genauso viele Transitionen wie $\mathcal{P}$.

Beobachtung (2): Die Gestalt des Präfix hängt ab von der Reihenfolge, in der Kanten hinzugefügt werden!

Entfaltungen für Petri-Netze

Wir verallgemeinern das Verfahren auf Petri-Netze, wie folgt:

Die Entfaltung eines Petri-Netzes $\mathcal{P}$ (bzw. ein vollständiger Präfix derselben) ist ein azyklisches Petri-Netz $\mathcal{Q}$.

Annahme: $\mathcal{P}$ sei 1-beschränkt (das Verfahren kann auf beliebig k-beschränkte Netze erweitert werden.

Bemerkung: Im Folgenden nennen wir die Stellen von $\mathcal{Q}$ Bedingungen, die Transitionen von $\mathcal{Q}$ Ereignisse. Diese Bezeichnungen dienen lediglich dazu, die Elemente von $\mathcal{P}$ und $\mathcal{Q}$ leichter auseinander zu halten!

Jede Bedingung von $\mathcal{Q}$ ist mit einer Stelle von $\mathcal{P}$ beschriftet,
jedes Ereignis von $\mathcal{Q}$ mit einer Transition von $\mathcal{P}$.

Jedes Ereignis t' ist von der Form (P', t), wobei P' die Eingangsbedingungen von t' sind und t die Beschriftung von t'.

Sei P' eine Menge von Bedingungen. Mit $B(P')$ bezeichnen wir die Menge der Stellen, mit denen die Elemente von P' beschriftet sind.

Jede Bedingung hat genau eine eingehende Kante.

Einige Ereignisse im vollständigen Präfix sind als Cutoffs markiert.

Konstruktion eines Präfixes für Petri-Netze

1. Sei M_0 die Anfangsmarkierung von $\mathcal{P}$. Anfangs enthalte $\mathcal{Q}$ eine Bedingung für jede Stelle in M_0. Die Anfangsmarkierung von $\mathcal{Q}$ besteht aus genau diesen Bedingungen. Wir setzen $\mathcal{E} := \{M_0\}$.

2. Sei t eine Transition von $\mathcal{P}$ und P' eine Menge von Bedingungen, so dass keine Bedingung durch ein Cutoff-Ereignis erzeugt wird. Außerdem sei P' in $\mathcal{Q}$ überdeckbar, es sei $B(P') = {}^\bullet t$, und (P', t) sei noch nicht in $\mathcal{Q}$ enthalten.

2a. Wenn es kein solches Paar (P', t) gibt, dann Ende.

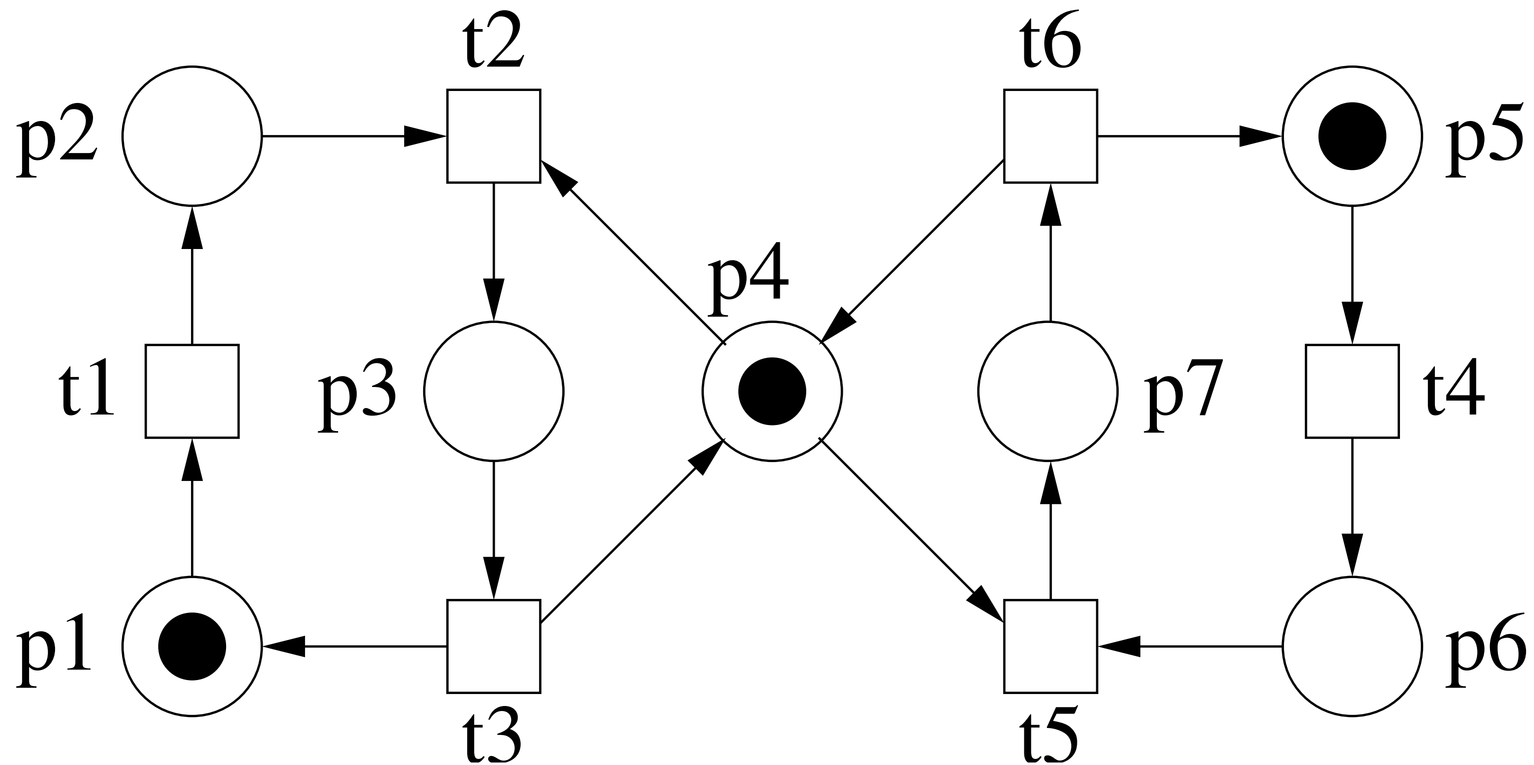
2b. Füge dem Präfix das Ereignis $t' := (P', t)$ hinzu (mit P' als Preset und t als Beschriftung). Erweitere den Präfix um eine Bedingung für jede Ausgangsstelle von t, die zugleich Ausgangstelle von t' wird.

2c. Wir ordnen t' eine (in $\mathcal{P}$ erreichbare) Markierung $M_{t'}$ zu (s.u.). Falls $M_{t'} \in \mathcal{E}$, so ist t' ein Cutoff. Sonst $\mathcal{E} := \mathcal{E} \cup \{M_{t'}\}$.

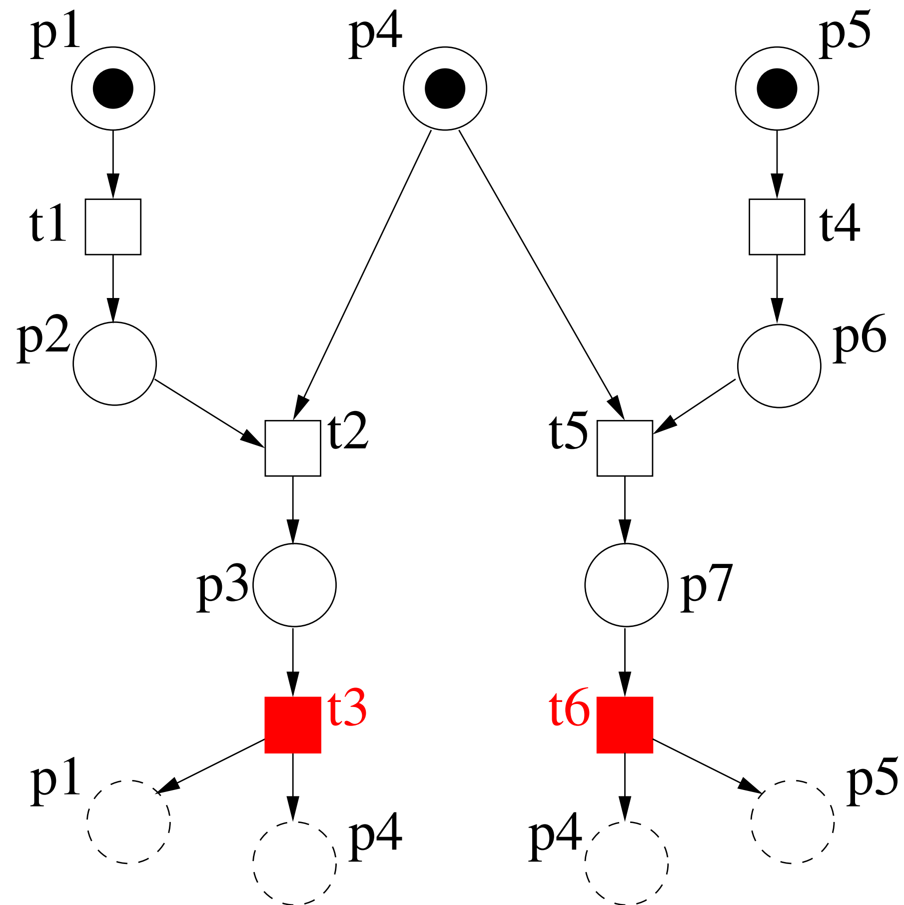
Bemerkung: Lassen wir den Schritt 2c aus (d.h. es gibt keine Cutoffs), so erhalten wir die gesamte (unabgeschnittene) Entfaltung von $\mathcal{P}$.

Die Gestalt von $\mathcal{Q}$ (d.h. welcher Präfix der Entfaltung generiert wird), hängt von der Reihenfolge ab, in der Ereignisse generiert werden. (Wir kommen später hierauf zurück!)

Beispiel 1: Petri-Netz. . .



. . . und möglicher Präfix der Entfaltung



Bestimmung von $M_{t'}$

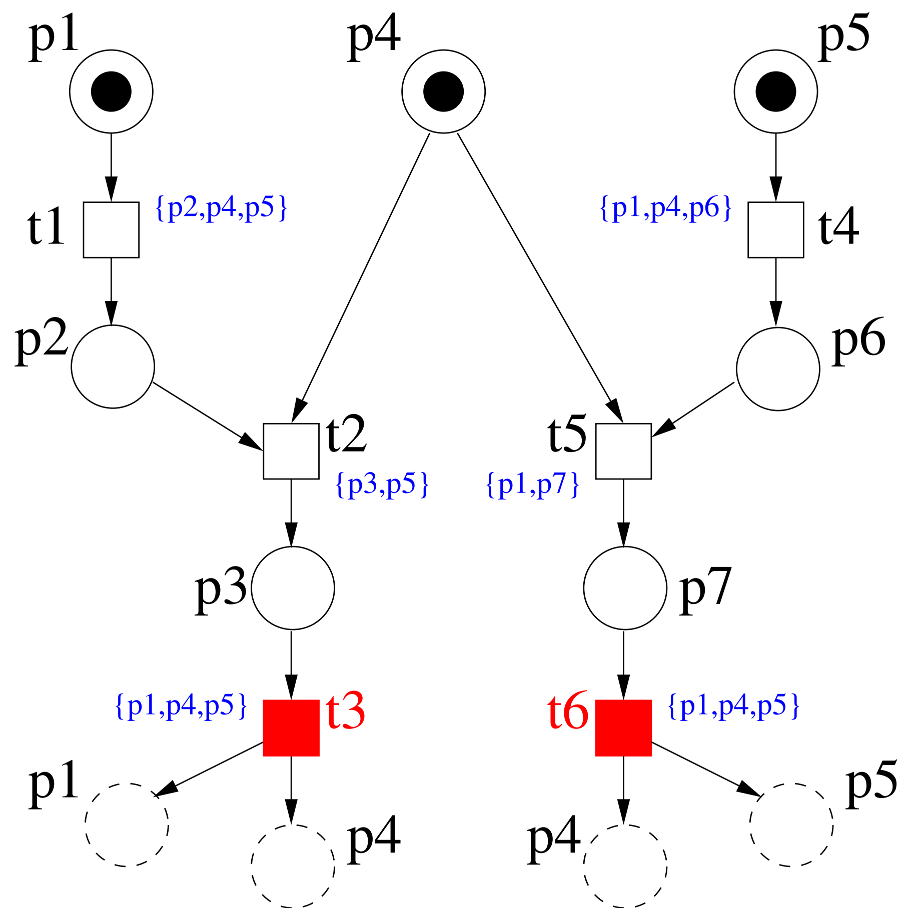
Wenn wir dem Präfix $t' = (P', t)$ hinzufügen, bestimmen wir $M_{t'}$ folgendermaßen:

Idee: $M_{t'}$ ist die “minimale” Markierung, die durch Feuern von t' entsteht.

Seien x, y zwei Knoten (Bedingungen oder Ereignisse) in $\mathcal{Q}$. $<$ sei die kleinste Halbordnung, bei der $x < y$ gilt, wenn es eine Kante von x nach y gilt.

Sei x ein Knoten von $\mathcal{Q}$. Wir definieren $\lfloor x \rfloor := \{\, y \mid y < x \vee y = x \,\}$.

$M_{t'}$ sei die Markierung, die durch Schalten der Ereignisse in $\lfloor t' \rfloor$ entsteht (in irgendeiner Reihenfolge). Merke: Eine solche Schaltfolge gibt es, da P' überdeckbar ist.


p1
p4
p5
t1
{p2,p4,p5}
{p1,p4,p6}
t4
p2
p6
t2
t5
{p3,p5}
{p1,p7}
p3
p7
{p1,p4,p5}
t3
t6
{p1,p4,p5}
p1
p5
p4
p4

Bemerkungen

Die Konstruktion von $\mathcal{Q}$ terminiert, weil jede Markierung $M_{t'}$ in $\mathcal{P}$ erreichbar ist und $\mathcal{P}$ nur endlich viele erreichbare Markierungen hat.

In den meisten Fällen ist ein Präfix

größer als das zugehörige Petri-Netz;

kleiner als der zugehörige Erreichbarkeitsgraph.

Konflikt, Kausalität, Nebenläufigkeit

Aus der Struktur der Entfaltung können wir Aussagen über das Verhältnis von Bedingungen zueinander ableiten.

Seien p, q zwei (unterschiedliche) Bedingungen von $\mathcal{Q}$.

p, q heißen kausal abhängig, falls $p < q$ gilt. (D.h., p muss in jeder Schaltfolge "vor" q markiert werden.)

p, q stehen in Konflikt, falls es Ereignisse t, u gibt mit $t \neq u$, $t \in \lfloor p \rfloor$, $u \in \lfloor q \rfloor$ und ${}^{\bullet}t \cap {}^{\bullet}u \neq \emptyset$. (D.h., p, q können *nicht* gemeinsam markiert werden!)

p, q heißen nebenläufig, falls sie weder kausal abhängig sind noch in Konflikt miteinander. (D.h., p, q *können* zugleich markiert werden!)

Erreichbarkeit in Präfixen

Bemerkung: Eine Menge Bedingungen P' in $\mathcal{Q}$ ist überdeckbar gdw. alle Paare $p, q \in P'$ nebenläufig sind.

Die Nebenläufigkeitsrelation N (mit $p \, N \, q$ gdw. p, q nebenläufig sind) lässt sich während der Generation der Entfaltung effizient generieren.

Falls P' in $\mathcal{Q}$ erreichbar (überdeckbar) ist, so ist $B(P')$ in $\mathcal{P}$ erreichbar (überdeckbar). Gilt auch die umgekehrte Richtung?

Eigenschaften eines vollständigen Präfixes

Für endliche Systeme $\mathcal{P}$ gilt:

Ein Term (d.h. eine Variable) Y ist in $\mathcal{P}$ erreichbar gdw. im vollständigen Präfix ein mit Y beschrifteter Knoten erreichbar ist.

Dies gilt unabhängig von der Reihenfolge, in der die Ereignisse dem Präfixes hinzugefügt werden.

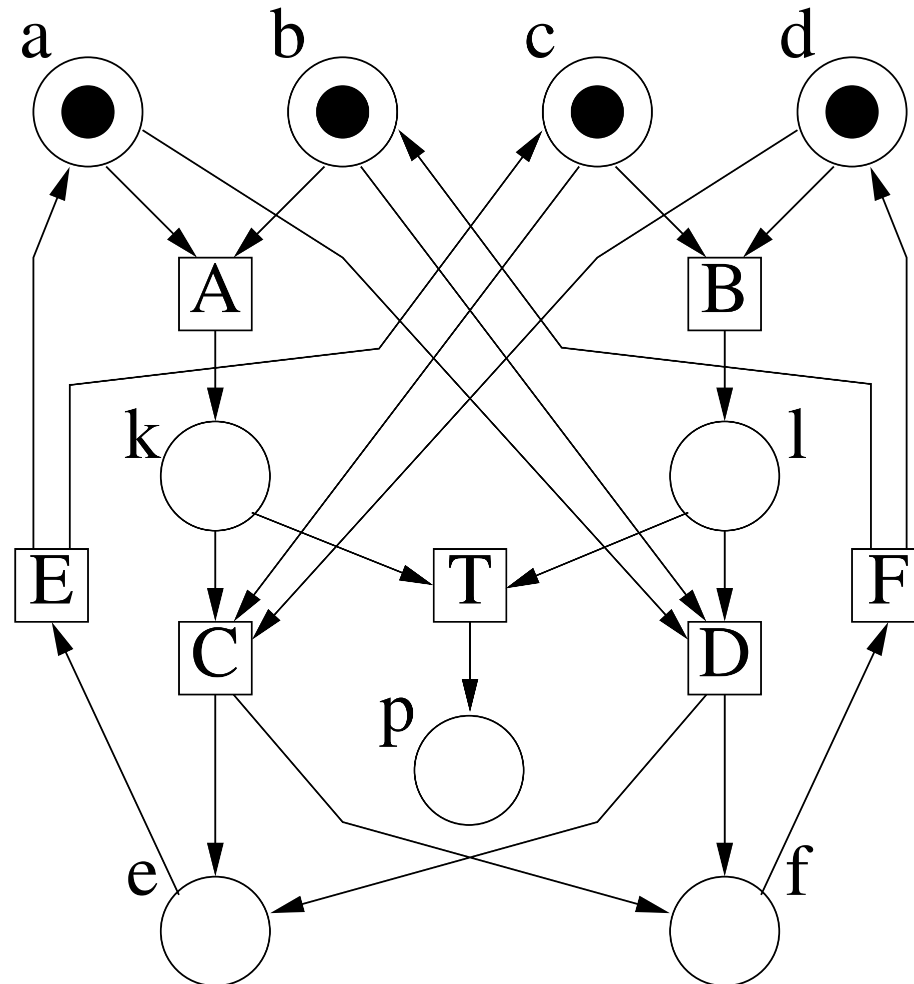
Für Petri-Netze $\mathcal{P}$ (und einen Entfaltungs-Präfix $\mathcal{Q}$ hätten wir gerne folgende Vollständigkeits-Eigenschaft:

Eine Markierung M ist in $\mathcal{P}$ erreichbar *gdw.* in $\mathcal{Q}$ eine Markierung M' mit $B(M') = M$ erreichbar ist.

Leider gilt dies nicht für alle Präfixe $\mathcal{Q}$. Ob $\mathcal{Q}$ die Vollständigkeits-Eigenschaft hat, ist abhängig von der Reihenfolge, in der die Ereignisse von $\mathcal{Q}$ generiert werden!

Beispiel 2

Wir betrachten folgendes Petri-Netz:

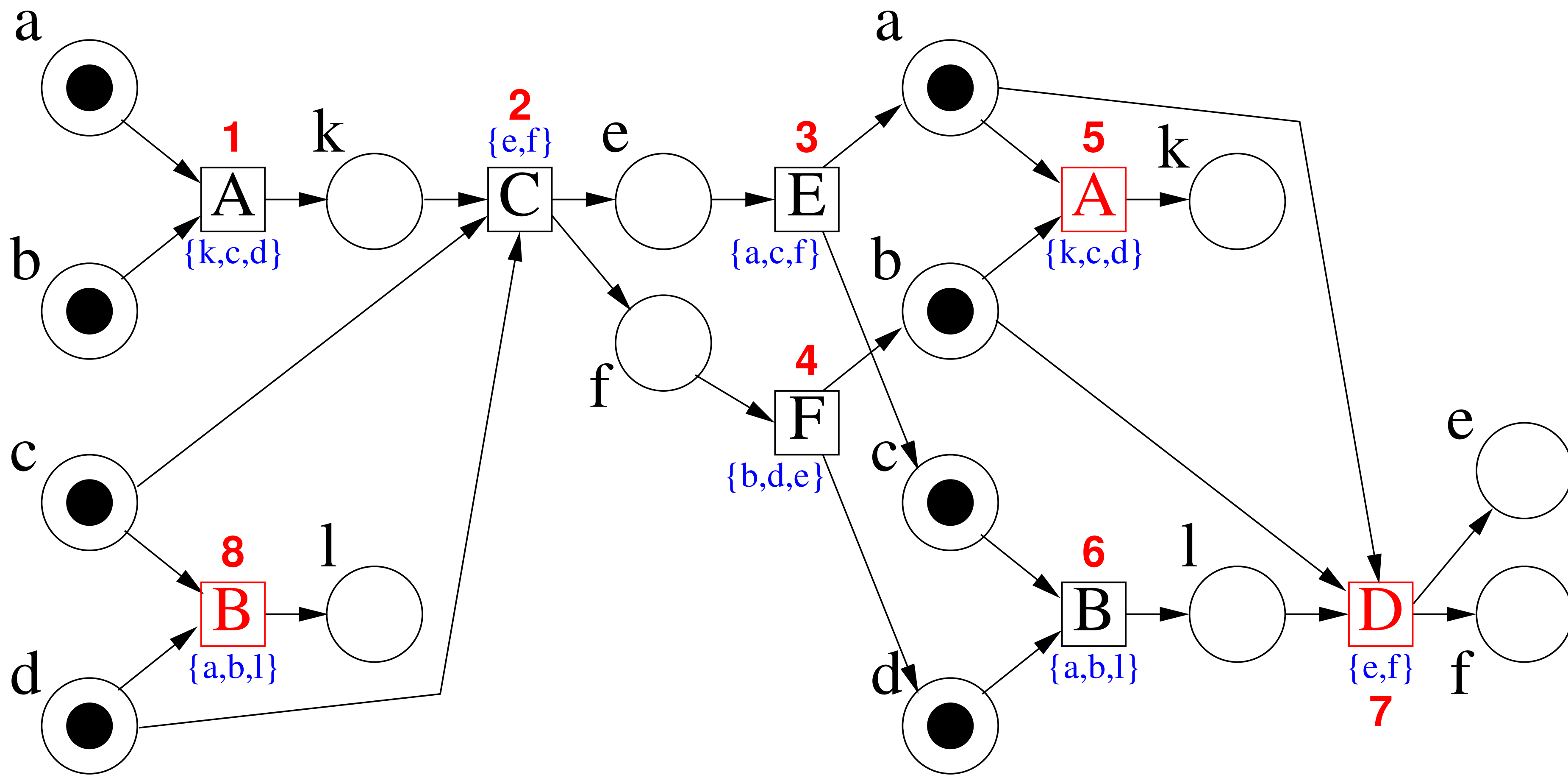


In Beispiel 2 ist die Markierung $\{p\}$ erreichbar, z.B. durch die Schaltfolge $A\,B\,T$.

Das Netz kann auch die Markierung $\{e, f\}$ erreichen, indem entweder $A\,C$ oder $B\,D$ geschaltet wird, und dann durch $E\,F$ wieder zur Anfangsmarkierung zurückkehren.

Wir werden sehen, dass ein Präfix, der nach dem Prinzip der *Tiefensuche* generiert wird, in diesem Beispiel die Transition T übersieht.

Durch Tiefensuche wird untenstehender Präfix erzeugt (Reihenfolge der Ereignisse und Cutoffs in rot):



Adäquate Ordnung

Sei $\mathcal{Q}$ die (womöglich unendliche) Entfaltung von $\mathcal{P}$, die durch Konstruktion des Präfix “ohne Cutoff-Bedingung” entsteht.

Sei $\prec$ eine totale Ordnung auf den Ereignissen von $\mathcal{Q}$, die $<$ verfeinert (d.h. $t < t'$ impliziert $t \prec t'$).

Intuition: $\prec$ ist eine mögliche Reihenfolge, in der die Ereignisse von $\mathcal{Q}$ generiert werden können.

Sei $\mathcal{Q}_\prec$ derjenige (eindeutige!) Präfix von $\mathcal{Q}$, bei dem die Ereignisse in der von $\prec$ vorgegebenen Reihenfolge hinzugefügt wurden.

Wir nennen $\prec$ adäquat, falls $\mathcal{Q}_\prec$ die Vollständigkeits-Eigenschaft hat.

Konfigurationen

Sei M eine erreichbare Markierung in $\mathcal{Q}$. Dann nennen wir $C_M := \bigcup_{p \in M} \lfloor p \rfloor$ eine Konfiguration. Die maximalen (bzgl. $<$) Bedingungen einer Konfiguration C bilden eine erreichbare Markierung M_C.

Bemerkung: Für jedes Ereignis t sind $\lfloor t \rfloor \setminus \{t\}$ und $\lfloor t \rfloor \cup t^\bullet =: C_t$ Konfigurationen. Bemerkung: $M_{C_t} = M_t$.

Wir nennen E eine Erweiterung von C, falls $C \cap E = \emptyset$ und $C \cup E$ ist erneut eine Konfiguration. In diesem Fall schreiben wir $C \oplus E$ für die Konfiguration $C \cup E$.

Seien M, M' zwei Markierungen von $\mathcal{Q}$, so dass $B(M) = B(M')$. Wenn E eine Erweiterung von C_M ist, so gibt es eine zu E isomorphe Erweiterung E' von $C_{M'}$.

Hinreichende Bedingungen für adäquate Ordnungen

Folgende Bedingung garantiert, dass $\prec$ adäquat ist:

> Seien t, t' zwei Bedingungen mit $t \prec t'$ und $M_t = M_{t'}$, und E eine Erweiterung von C_t sowie E' die zu E isomorphe Erweiterung von $C_{t'}$. Dann gilt $u \prec u'$, falls $C_u = C_t \cup E$ und $C_{u'} = C_{t'} \cup E'$.

Bemerkung: Dies wird z.B. garantiert durch $t \prec t'$, falls $|C_t| < |C_{t'}|$.

Beweis: Sei $\mathcal{Q}$ die Entfaltung von $\mathcal{P}$ und $\prec$ eine Ordnung, die obige Bedingung erfüllt, Wir zeigen, dass $\mathcal{Q}_\prec$ vollständig ist. Sei also M eine in $\mathcal{P}$ erreichbare Markierung. Dann gibt es eine Markierung M' in $\mathcal{Q}$ mit $B(M') = M$. Entweder ist $C_{M'}$ in $\mathcal{Q}_\prec$ enthalten, oder $C_{M'} = C_t \oplus E$ für irgendein Cutoff-Ereignis t. Dann aber gibt es ein Ereignis t' mit $C_{t'} = C_t$ und $t' \prec t$ und daher eine Konfiguration $C' := C_{t'} \oplus E'$, wobei E' isomorph zu E ist, und es gilt $B(M_{C'}) = B(M') = M$. Entweder ist C' in $\mathcal{Q}_\prec$ enthalten, oder man wiederholt das Argument, aber nur endlich oft, da $\prec$ wohlfundiert ist.

6.5: Petri-Netze und CTL

Petri-Netze und CTL

Das CTL-Model-Checking-Problem für Petri-Netze ist:

> Gegeben ein Petri-Netz *N* und eine CTL-Formel ϕ, erfüllt *N* (bzw. die zugehörige Kripke-Struktur) ϕ?

Dieses Problem ist unentscheidbar.
Genauer gesagt gilt dies schon für die Klasse der (1,P)-PRS, d.h. Petri-Netze “ohne Synchronisation”.

Zum Beweis zeigen wir, dass *N*, ϕ zusammen den Ablauf einer Registermaschine charakterisieren können.

Zählermaschinen

Eine Zählermaschinen ist ein Tripel $\mathcal{M} = (Q, C, \delta)$.

Bemerkung: Zählermaschinen (engl. counter machines) werden in der Literatur auch Registermaschinen oder Minsky-Maschinen gennant.

$Q = \{q_1, \ldots, q_n\}$ sind Kontrollzustände, auch vorstellbar als Sprungmarken eines Programms.

$C = \{c_1, \ldots, c_m\}$ ist die Menge der Zähler, d.h. Variablen, die natürliche Zahlen enthalten.

Eine *Konfiguration* von $\mathcal{M}$ ist ein Element aus $Q \times \mathbb{N}_0^m$, d.h. Kontrollzustand und Inhalt der Zähler.

Die Anfangskonfiguration ist $(q_1, 0, \ldots, 0)$.
Eine Konfiguration mit Kontrollzustand q_n heißt Endkonfiguration.

$\delta\colon Q \to (C \times Q) \cup (C \times Q \times Q)$ sind die Anweisungen, eine für jeden Kontrollzustand. Es gibt zwei Arten von Anweisungen:

Typ 1: Falls $\delta(q_i) = (c_j, q_k)$, so wird bei Kontrollzustand q_i der Zähler c_j um eins erhöht und zu Kontrollzustand q_k gewechselt.

Typ 2: Falls $\delta(q_i) = (c_j, q_k, q_\ell)$, so wird getestet, ob c_j den Wert 0 hat; falls ja, wird zu q_k gesprungen, sonst c_j um eins erniedrigt und zu q_ℓ gesprungen.

Als Sonderfall ist die Anweisung bei q_n einfach *halt*.

Man kann sich eine Zählermaschine auch wie ein Programm der untenstehenden Art vorstellen:

q_1: inc c_1; goto q_5
q_2: if $c_3 = 0$ then goto q_3 else dec c_3; goto q_4 fi
...
q_{50}: halt

Zählermaschinen sind Turing-mächtig

Es ist bekannt, dass Zählermaschinen Turing-Maschinen simulieren können.

Genauer gesagt gilt dies bereits für Zählermaschinen mit nur zwei Zählern.

Daher ist unentscheidbar, ob eine Zählermaschine aus der Anfangskonfiguration heraus eine Endkonfiguration erreichen kann.

Gegeben eine Zählermaschine $\mathcal{M}$, konstruieren wir ein Netz *N* und eine Formel ϕ, so dass *N* ϕ erfüllt gdw. $\mathcal{M}$ eine Endkonfiguration erreichen.

Wie schon beim Beweis der Unentscheidbarkeit PA/LTL wird *N* Abläufe haben, die in $\mathcal{M}$ nicht möglich sind, die aber von ϕ "ausgefiltert" werden.

Konstruktion von N

Wir geben N in Form eines BPP an, also mit Regeln $X \rightarrow Y_1 \parallel \cdots \parallel Y_k$.

Für jeden Kontrollzustand q_i gibt es drei Variablen (Stellen) $\widehat{Q}_i$, Q_i und R_i sowie die Regel

$$\widehat{Q}_i \stackrel{\cdot}{\longrightarrow} \widehat{Q}_i \parallel Q_i$$

Die Anfangsvariable X hat folgende Regel:

$$X \stackrel{\cdot}{\longrightarrow} \widehat{Q}_1 \parallel \cdots \parallel \widehat{Q}_n \parallel Q_1$$

N hat drei Variablen für jeden Zähler.

Zähler c_j wird durch Variablen C_j, D_j und E_j repräsentiert.

Für jeden Zähler c_j gibt es folgende Regeln:

$$C_j \stackrel{c_j}{\longrightarrow} D_j \qquad D_j \stackrel{d_j}{\longrightarrow} E_j \qquad E_j \stackrel{e_j}{\longrightarrow} \varepsilon$$

Falls $\delta(q_i) = (c_j, q_k)$, so gibt es noch folgende Regel:

$$Q_i \xrightarrow{q_i} C_j \parallel Q_k$$

Falls $\delta(q_i) = (c_j, q_k, q_\ell)$, so gibt es die Regeln

$$Q_i \xrightarrow{q_i} R_i \qquad\qquad R_i \xrightarrow{r_i} \varepsilon$$

Außerdem fügen wir Regeln für q_n hinzu:

$$Q_n \xrightarrow{q_n} R_n \qquad\qquad R_n \xrightarrow{halt} \varepsilon$$

Ein Term (eine Markierung) der Form

$$\widehat{Q}_1 \parallel \cdots \parallel \widehat{Q}_n \parallel Q_i \parallel C_1^{k_1} \parallel \cdots \parallel C_m^{k_m}$$

entspricht einer $\mathcal{M}$-Konfiguration $(q_i, k_1, \ldots, k_m)$. Solche Terme nennen wir *gut*. Die erste aus X erreichbare Konfiguration ist gut und entspricht der Anfangskonfiguration.

Konstruktion von ϕ

Bei Anweisungen des ersten Typs kann ein guter Term direkt zu einem anderen guten Term übergehen in der Weise, dass dies dem zugehörigen Übergang in $\mathcal{M}$ entspricht (Erhöhung eines Zählers).

Um von einem guten Term zu einem anderen guten Term zu kommen, muss man (bei Anweisungen des zweiten Typs) durch einige Zwischenterme gehen. Diese nennen wir im Folgenden *fast gut*.

Wir konstruieren zunächst eine Formel ϕ_g, die alle guten und fast guten Terme charakterisiert. Von jedem guten Term wird es nur eine Sequenz von fast guten Termen geben, die in den nächsten guten Term überleitet, und dies wird der Funktionsweise von $\mathcal{M}$ entsprechen (Test auf 0, ggfs. Zähler erniedrigen und Sprung).

Solange ein Ablauf von N dann in Zuständen bleibt, die ϕ_g erfüllen, entspricht dieser Ablauf dem (einzigen) Ablauf von $\mathcal{M}$.

Hilfsweise definieren wir folgende Formel, wobei
$A := \{q_1, r_1, \ldots, q_n, r_n, d_1, e_1, \ldots, d_m, e_m\}$ ist:

$$Nur(a_1, \ldots, a_k) = \bigwedge_{i=1}^{k} \mathbf{EX}\, a_i \wedge \bigwedge_{i=1}^{k} \neg\, \mathbf{EX}(a_i \wedge \mathbf{EX}\, a_i) \wedge \bigwedge_{a \in A \setminus \{a_1, \ldots, a_k\}} \neg\, \mathbf{EX}\, a$$

Die Formel *Nur* stellt sicher, dass im gegenwärtigen Zustand alle Aktionen $a_1, \ldots, a_k$ ausführbar sind, aber nicht zweimal hintereinander. Außerdem stellt sie sicher, dass die einzigen weiteren Aktionen, die im gegenwärtigen Zustand möglich sind, das Erniedrigen eines Zählers (Aktion c_j) oder Erzeugen eines Kontrollzustands (Aktion $\cdot$) sind.

Bemerkung: Die guten Zustände sind diejenigen, die die Formel

$$\bigvee_{i=1}^{n} Nur(q_i)$$

erfüllen.

Wir setzen $\phi_g := \bigvee_{i=1}^{n} \phi_g^i$, d.h. ϕ_g enthält eine Disjunktion für jeden Kontrollzustand.

Für Anweisungen vom ersten Typ und für q_n setzen wir $\phi_g^i := \mathit{Nur}(q_i)$.

Für Anweisungen vom zweiten Typ setzen wir, falls $\delta(q_i) = (c_j, q_k, q_\ell)$ ist,

$$\begin{aligned}
\phi_g^i \quad := \quad & \Big(\neg\, \mathbf{EX}\, c_j \wedge \big(\mathit{Nur}(q_i) \vee \mathit{Nur}(r_i) \vee \mathit{Nur}(r_i, q_k)\big)\Big) \\
& \vee \Big(\big(\mathit{Nur}(q_i) \wedge \mathbf{EX}\, c_j\big) \vee \mathit{Nur}(q_i, d_j) \vee \mathit{Nur}(r_i, d_j) \\
& \qquad \vee \mathit{Nur}(r_i, e_j) \vee \mathit{Nur}(r_i, e_j, q_\ell) \vee \mathit{Nur}(q_\ell, e_j)\Big)
\end{aligned}$$

Konstruktion von ϕ

Wir setzen jetzt

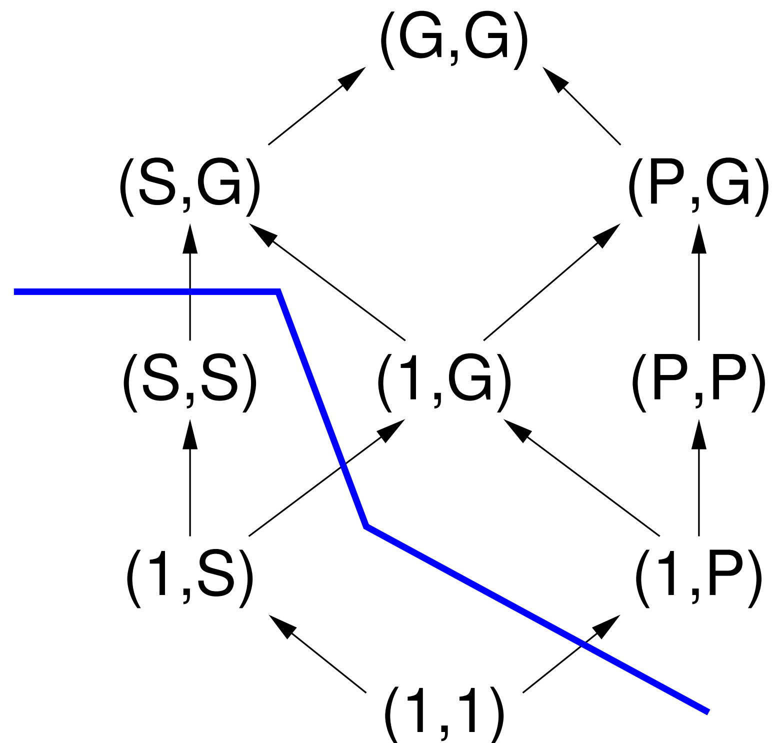
$$\phi = \mathbf{AX}\,\mathbf{AF}(\neg\phi_g \vee \mathbf{EX}\,\textit{halt})$$

Das $\mathbf{AX}$ dient nur dazu, den ersten Initialisierungsschritt zu überspringen. Danach gibt es genau eine Sequenz von Termen, die der Funktionsweise der Zählermaschine entspricht. Alle anderen Sequenzen landen also irgendwann in einem $\neg\phi_g$-Zustand.

Falls die Maschine nicht hält, ist die Formel also nicht erfüllt, denn es gibt einen Sequenz von Termen, in denen immer ϕ_g und nie $\mathbf{EX}\,\textit{halt}$ gilt. Andernfalls ist sie erfüllt.

Entscheidbarkeitsgrenze für CTL und CTL*

Wir haben gesehen, dass CTL* für Klassen (S,S) entscheidbar und CTL für die Klasse (1,P) unentscheidbar ist. Für beide Logiken sind also genau die Klassen unter der blauen Linie entscheidbar.



6.6: Petri-Netze und LTL

Petri-Netze und LTL

Das LTL-Model-Checking-Problem für Petri-Netze ist:

Gegeben ein Petri-Netz N und eine LTL-Formel ϕ, erfüllt N (bzw. die zugehörige Kripke-Struktur) ϕ?

Dieses Problem ist entscheidbar.
(im Gegensatz zum entsprechenden Problem für CTL)

Der Beweis benutzt ähnliche Techniken wie bei Pushdown-Systemen.

Vorgehensweise

Man übersetze $\neg\phi$ in einem Büchi-Automaten $\mathcal{B}$.
(Merke: Das Alphabet von $\mathcal{B}$ sind die Transitionen von N.

Erweitere N zu einem Netz N', wie folgt:

Jeder Zustand von $\mathcal{B}$ wird eine zusätzliche Stelle in N'.

Wenn (q, T, q') eine Transition von $\mathcal{B}$ ist, füge zu N' die Kanten (q, T) und (T, q') hinzu.

Der Anfangszustand von $\mathcal{B}$ enthält eine Marke in der Anfangsmarkierung.

Jede Schaltfolge von N' gibt nun eine Schaltfolge von N mit dem entsprechenden Ablauf in $\mathcal{B}$ wieder.

Reduktion auf zyklische Erreichbarkeit

Eine (unendliche) Schaltfolge in $\mathcal{B}$ nennen wir akzeptierend gdw. sie unendlich oft einen akzeptierenden Zustand von $\mathcal{B}$ markiert.

N erfüllt ϕ *nicht* gdw. N' eine akzeptierende Schaltfolge hat.

Eine Schaltfolge $\sigma = M_0 M_1 \ldots$ ist akzeptierend gdw. es M_i, M_j gibt mit $i < j$, $M_i \leq M_j$ und in M_i und M_j ist ein akzeptierender Zustand markiert.

> Beweis: Die Rückrichtung ist trivial. Für die Hinrichtung betrachte man die (unendliche) Untersequenz aller Markierungen mit akzeptierenden Zuständen. Daraus filtern wir nochmal eine unendliche Untersequenz, in der immer derselbe akzeptierende Zustand markiert ist. Dann existieren M_i und M_j wegen Dicksons Lemma.

Algorithmus

Wir suchen also Markierungen M mit folgenden Eigenschaften:

(1) M markiert einen akzeptierenden Zustand.

(2) $M \Longrightarrow^* M'$ mit $M \leq M'$

(3) $M \in R(N')$

Wenn ein solches M existiert, so erfüllt N ϕ nicht.

Eigenschaft (2) kann mithilfe des Überdeckbarkeitsgraphen getestet werden.

Man wählt M als Anfangsmarkierung und konstruiert den ÜG. Eigenschaft (2) gilt gdw. der ÜG einen Zyklus zu M oder einen Knoten M' mit $M < M'$ enthält.

Für welche Markierungen M sollten wir Eigenschaft (2) testen?

Wir nutzen Dicksons Lemma, um endlich viele solche Tests durchzuführen:

> In jeder unendlichen Schaltsequenz $\sigma = M_0 M_1 \ldots$ gibt es nur endlich viele Indizes i, so dass M_i minimal in σ ist.

Daraus können wir folgende (nicht-deterministische) Vorgehensweise ableiten:

1. Setze $M := M_0$ (die Anfangsmarkierung).

2a. Falls M unvergleichbar ist zu allen bisher gesehenen Markierungen: Wenn M einen akzeptierenden Zustand markiert, führe den Test für (2) durch.

2b. Falls bereits eine Markierung M' mit $M \leq M'$ gesehen wurde: In diesem Fall brechen wir ab.

2c: Falls M strikt eine gesehene Markierung M' überdeckt: Erweitere M zu einer ω-Markierung (analog zum ÜG) und fahre wie bei 2a fort.

3. Suche eine Nachfolge-Markierung M' von M, setze $M := M'$ und fahre bei 2a, 2b oder 2c fort.

Fall 2a kann nach Dicksons Lemma nur endlich oft auftreten. Fall 2c kann ebenfalls nur endlich oft auftreten, weil wir nur endlich viele Stellen auf ω setzen können.

Ein deterministischer Algorithmus kann aus dem obigen durch Backtracking gewonnen werden.

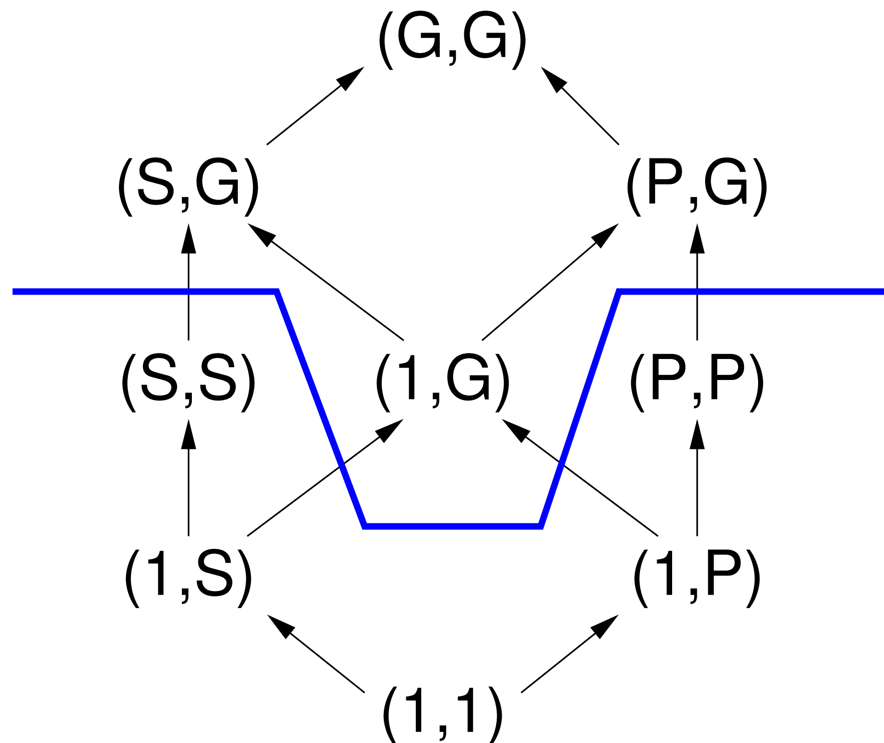
Komplexität:

Man kann zeigen, dass die Anzahl der Schritte exponentiell beschränkt ist, so dass das Problem in *EXPSPACE* liegt.

Kein effizienter Algorithmus bekannt!

Entscheidbarkeitsgrenze für LTL

Wir haben gesehen, dass LTL für die Klassen (S,S) und (P,P) (d.h. Pushdown und Petri-Netze) entscheidbar und für (1,G) unentscheidbar ist.



Teil 7: Die Klasse (G,G)

Die Klasse (G,G)

Die Klasse der (G,G)-PRS ist die allgemeinste Klasse der PRS-Hierarchie.

Wir haben bereits gezeigt, dass das Model-Checking-Problem für LTL, CTL, CTL* unentscheidbar für diese Klasse ist.

Das folgende *Erreichbarkeitsproblem* ist jedoch *entscheidbar*.

> Gegeben ein PRS $\mathcal{P}$ und zwei Terme t, t', gilt $t \Longrightarrow^* t'$?

Erreichbarkeit in Petri-Netzen

Es ist bekannt (Mayr 1984), dass das Erreichbarkeitsproblem für Petri-Netze entscheidbar ist.

> Gegeben ein Petri-Netz N und eine Markierung M, gilt $M \in R(N)$?

Merke: Dieses Erreichbarkeitsproblem lässt sich *nicht* auf das LTL-Problem für Petri-Netze reduzieren.

Der o.g. Entscheidungsalgorithmus hat nicht-primitiv-rekursive Laufzeit.
Es ist bekannt, dass das Erreichbarkeitsproblem mindestens EXPSPACE-hart ist.

Erreichbarkeit in PRS

Die Entscheidbarkeit des Erreichbarkeitsproblems für PRS ist eine (vergleichsweise) einfache Konsequenz aus den Erreichbarkeitsproblemen für PDS und PN.

Man füge zu $\mathcal{P}$ zunächst Regeln der Form $X \to t_1$ und $t_2 \to Y$ hinzu, wobei X, Y “frische” Variablen sind. Dann reduziert sich das Problem auf die Frage $X \Longrightarrow^* Y$.

Dann wandelt man $\mathcal{P}$ in ein PRS $\mathcal{P}'$ um, in dem keine Regel sowohl den Sequenz-Operator als auch den Parallel-Operator enthält (indem man Regeln aus $\mathcal{P}$, die beides enthalten, in mehrere Schritte aufspaltet, mit zusätzlichen frischen Variablen). In $\mathcal{P}'$ gilt $X \Longrightarrow^* Y$ gdw. es in $\mathcal{P}$ gilt.

Mit $\mathcal{P}_P$ bezeichnen wir dann das PRS, das entsteht, wenn man aus $\mathcal{P}'$ alle Regeln mit Sequenz-Operator entfernt, und $\mathcal{P}_S$ entsprechend.

Man zeigt jetzt folgendes Lemma: Falls $X' \Longrightarrow^* Y'$ oder $X' \Longrightarrow^* \varepsilon$ in $\mathcal{P}'$ gilt (für irgendwelche Variablen X', Y'), dann gibt es auch irgendwelche Variablen X'', Y'', so dass $X'' \Longrightarrow^* Y''$ oder $X'' \Longrightarrow^* \varepsilon$ in $\mathcal{P}_P$ oder $\mathcal{P}_S$ gilt.

Die letztere Bedingung ist entscheidbar. Man finde also alle solche Variablen X'', Y'', die die letztere Bedingung erfüllen, und füge zu $\mathcal{P}'$ die Regel $X'' \Longrightarrow^* Y''$ bzw. $X'' \Longrightarrow^* \varepsilon$ hinzu (und damit auch zu $\mathcal{P}_P$ und $\mathcal{P}_S$).

Auf diese Weise "saturiert" man $\mathcal{P}'$, bis es nichts mehr hinzuzufügen gibt, und prüft anschließend, ob man die Regel $X \longrightarrow Y$ hat.

Beweis für das Lemma:

Sei $\sigma(X'', Y'')$ die kürzeste Sequenz von Regeln in $\mathcal{P}'$, die X'' in Y'' überführt (analog $\sigma(X'')$ für X'' und ε).

Wählen wir unter denjenigen X'', Y'' (bzw. X''), für die es noch keine Regel $X'' \longrightarrow Y''$ (bzw. $X'' \longrightarrow \varepsilon$) gibt, diejenigen/dasjenige aus, so dass $\sigma(X'', Y'')$ bzw. $\sigma(X')$ minimal ist.

Dann zeigen wir, dass $\sigma(X'', Y'')$ bzw. $\sigma(X'')$ entweder nur Regeln aus $\mathcal{P}_P$ oder nur Regeln aus $\mathcal{P}_S$ enthält.

(Beweis durch Widerspruch, man zeigt, dass wenn beiderlei Regeln auftreten, entweder eine nicht-minimale Sequenz benutzt wird oder die Wahl von X'', Y'' falsch war.)