Spezifikation und Verifikation
Kapitel 4
Überpüfen von Eigenschaften bei Petri Netzen

Frank Heitmann
heitmann@informatik.uni-hamburg.de

6. Juni 2014

Motivation

Bisher hatten wir CTL und LTL Model Checking

- CTL Model Checking
  - direkt auf dem Graphen des Modells
- LTL Model Checking
  - mittels Umweg über Büchi-Automaten

Meist modelliert man aber nicht mittels eines Transitionssystems...

Motivation

Man modelliert mit anderen Formalismen und wandelt diese (intern) in Transitionssysteme um.

Heute:

1. (1-sichere) Petri Netze
2. (sichere) Eos
3. dafür: on-the-fly LTL/CTL-Model Checking

Petri Netze

Definition (P/T Netz)

Ein *P/T Netz* ist ein Tupel $N = (P, T, F, W)$ mit

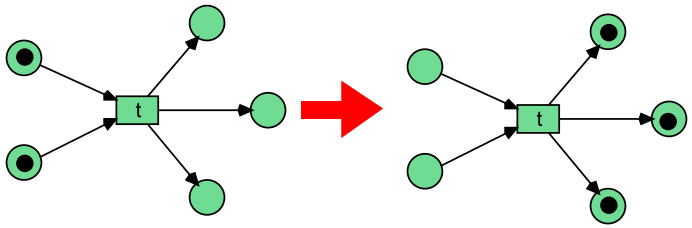
1. Der endlichen Menge der *Plätze* P.
2. Der endlichen Menge der *Transitionen* T.
3. Der *Flussrelation* $F \subseteq (P \times T) \cup (T \times P)$.
4. Der Kantengewichtung $W : F \to \mathbb{N} \setminus \{0\}$.

Ein *P/T Netz System* hat zusätzlich eine Startmarkierung $m_0 : P \to \mathbb{N}$.

Anmerkung

Wir gehen nachfolgende von $W(x, y) = 1 \forall (x, y) \in F$ aus und bezeichnen dann auch das Tupel $N = (P, T, F)$ als P/T Netz bzw. (P, T, F, m_0) als P/T Netz System.

Ein Beispiel



Petri Netze werden zur Modellierung verteilter Systeme und Protokolle erfolgreich eingesetzt.

Wichtige Begriffe

Begriffe (solltet ihr aus FGI2 kennen):

- Markierung
- Aktivierte Transition
- Schalten/Feuern einer Transition/Transitionssequenz
- Erreichbarkeitsgraph
- Beschränktheit, Lebendigkeit, Erreichbarkeit

Bemerkung

Findet ihr alles bei Bedarf in der Standardliteratur oder auch im FGI2-Skript.

1-sichere P/T Netze

Wir beschränken uns nachfolgend auf 1-sichere P/T Netze:

Definition (*n*-sichere P/T Netze)

Ein P/T Netz ist *n*-sicher wenn ein $n \in \mathbb{N}$ existiert, so dass in jeder erreichbaren Markierung höchstens n Marken auf jedem Platz sind:

$$\forall m \in RS(\mathbf{m}_0) : \forall p \in P : m(p) \leq n$$

Satz

Ein P/T Netz $\mathcal{N}$ ist n-sicher für ein n gdw. $|RS(\mathcal{N})| < \infty$, d.h. gdw. der Zustandsraum endlich ist.

Die Verbindung

Die Verbindung zwischen 1-sicheren P/T Netzen und dem Model Checking von Transitionssystemen:

- Im *Erreichbarkeitsgraphen* eines 1-sicheren P/T Netzes tritt in jeder Markierung jeder Platz maximal einmal auf.
- Eine Markierung kann also als Teilmenge der Plätze oder als $p_1 + p_5$ etc. beschrieben werden.
- Man kann nun
  - den Erreichbarkeitsgraphen als Transitionssystem interpretieren und
  - als atomare Formeln die Plätze nehmen.

Das Problem

Wie groß wird aber der Zustandsraum eines 1-sicheren P/T Netzes? Antwort: 2^n.

Wenden wir also unsere bisherigen Algorithmen an, so brauchen wir viel Zeit und Platz.

Das geht besser! Insbesondere mit weniger Platz!

PSpace-hardness

Das quasi alle Probleme zu 1-sicheren P/T Netzen PSPACE-schwierig sind folgt aus

Satz

Die meisten Fragen zu LBAs sind PSPACE-schwierig, z.B. die Frage, ob ein LBA eine gegebene Konfiguration erreicht, anhält, die leere Eingabe akzeptiert usw.

Satz

Ein LBA der Größe n kann von einem 1-sicheren P/T Netz der Größe $O(n^2)$ simuliert werden. Die Konstruktion geht in Polynomialzeit.

Beweis.

Konstruktion: Tafel ... Zur Nachbereitung siehe [Esparza]. □

Die Idee

Die Idee zum LTL Model Checking ist ähnlich wie bisher. Nur

1. können wir nicht den Automaten ganz konstruieren, sondern dürfen das nur teilweise machen und
2. wir nutzen Nichtdeterminismus aus (da ja PSPACE = NPSPACE gilt!)

Formeln und Automaten

Satz

Zu einer LTL Formel ϕ können ein DFA A_ϕ und ein Büchi Automat B_ϕ konstruiert werden, so dass $L(A_\phi) \cup L_\omega(B_\phi)$ gerade die Menge jener Wörter ist, die ϕ erfüllen.

Bemerkung

A wird hier für *endliche* Rechnungen benötigt. (Praktisch für Deadlocks etc.; kriegt man aber auch ohne hin.)

Noch mehr Automaten

Es werden zwei weitere Automaten

1. $A_N = (2^P, \Sigma_N, \delta_N, z_{0,N}, F_N^A)$ und
2. $B_N = (2^P, \Sigma_N, \delta_N, z_{0,N}, F_N^B)$

benötigt mit

1. $\Sigma_N = RS(N)$ (die erreichbaren Markierungen)
2. $z_{0,N} = m_0$ (Startmarkierung)
3. δ_N enthält (m_1, m_1, m_2) wenn $m_1 \xrightarrow{t} m_2$ für ein $t \in T$.
4. F_N^A ist die Menge der Deadlocks $m \in RS(N)$.
5. $F_N^B = RS(N)$

- Im Grunde genommen sind dies gerade die Erreichbarkeitsgraphen mit einer Besonderheit in der Beschriftung der Kanten (was hier dann den Produktautomaten vereinfacht).
- $L(A_N)$ ist die Menge aller endlichen Schaltfolgen von N.
- $L_\omega(B_N)$ ist die Menge aller unendlichen Schaltfolgen von N.

Die Produktautomaten

Mit der üblichen Produktautomatenkonstruktion:

- $L(A) = L(A_{\neg\phi}) \cap L(A_N)$ und
- $L_\omega(B) = L_\omega(B_{\neg\phi}) \cap L_\omega(B_N)$

$\Rightarrow L(A) \cup L_\omega(B)$ enthält die Folgen von N, die ϕ *nicht* erfüllen.

$\Rightarrow N$ erfüllt ϕ gdw. $L(A) = \emptyset$ und $L_\omega(B) = \emptyset$.

Die Idee

Wir nutzen nun zwei Dinge aus:

- Wegen $\mathrm{PSPACE} = \mathrm{NPSPACE}$ können wir auch einen *nichtdeterministischen* Algorithmus angeben.
- Da wir den dann determinieren können (und $\mathrm{PSPACE} = co\mathrm{PSPACE}$ ist), genügt es auch das Nicht-Leerheitsproblem zu entscheiden!

Der Algorithmus

Algorithmus 1 Überprüfe A auf Nicht-Leerheit

Ensure: z vom Typ Zustand von $A_{\neg\phi}$
Ensure: m vom Typ Zustand von A_N
1: $(z, m) \leftarrow (z_0, m_0)$;
2: **while** (z, m) kein Endzustand von A ist **do**
3: rate einen Zustand z' von $A_{\neg\phi}$ mit $z \xrightarrow[A_{\neg\phi}]{m} z'$
4: rate eine Markierung m' und eine Transition t mit $m \xrightarrow[N]{t} m'$
5: $(z, m) \leftarrow (z', m')$
6: **end while**
7: **return true**

Warum PSPACE?

Im Algorithmus wurde

- eine konstante Anzahl an Zuständen gespeichert,
- überprüft ob $(z_1, m, z_2) \in \delta_{\neg\phi}$ gilt,
- überprüft, ob eine Transition schalten kann,
- überprüft, ob ein Zustand ein Endzustand ist.

Das geht alles in polynomiellen Platz in $|\phi|$ und $|N|$.

Der Fall B

Der Fall für B ist nur etwas schwieriger:

- Wenn ein Endzustand erreicht wird, rate ob dieser wieder besucht wird.
- Falls nein geraten, mache weiter wie eben.
- Falls ja geraten, speichere dies und mache wieder weiter wie eben, suche nun aber den gespeicherten Endzustand.
  - Wird dieser wieder gefunden, sind wir fertig!

Geht wieder in NPSPACE und damit in PSPACE.

Das Ergebnis

Theorem

Gegeben ein 1-sicheres P/T Netz N und eine LTL Formel ϕ kann in Platz polynomiell in der Größe von N und ϕ überprüft werden, ob ϕ von N erfüllt wird.

Wichtige Anmerkung

Zur Betonung: Dabei wird der Zustandsraum nie ganz aufgebaut!

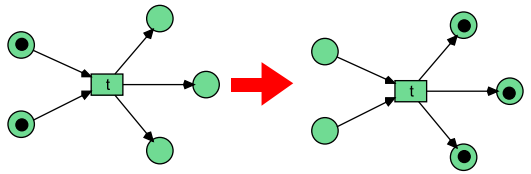
Die Idee

CTL Model Checking 1-sichere P/T Netze geht auch! Die Idee:

- Der Erreichbarkeitsgraph (bzw. das Transitionssystem) hat "nur" 2^n Markierungen (bzw. Zustände)
- Das kann wieder nicht aufgebaut werden, wenn man in PSPACE bleiben will!
- ABER: Man kann die zu überprüfenden Eigenschaften auf Pfaden der Länge maximal 2^n überprüfen.
- Diese kann man auf Eigenschaften überprüfen, indem man sie immer weiter halbiert und dann die linke und die rechte Hälfte einzeln betrachtet.
- Dabei wird Platz wiederholt benutzt. Man braucht aber nur $\log 2^n = n$ Aufrufe auf dem Stack speichern!

Von Petri Netzen zu Objektnetzen

Petri Netze sind ein weit verbreiteter Formalismus, *um verteilte Systeme zu modellieren*



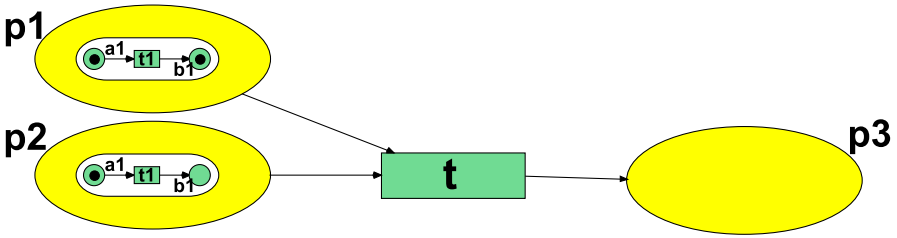
Schwierig oder unintuitiv zu modellieren:

- Mobilität
- Interaktion
- Verschachtelung

$\Rightarrow$ Objektnetze, Netze als Marken (Valk 1991)

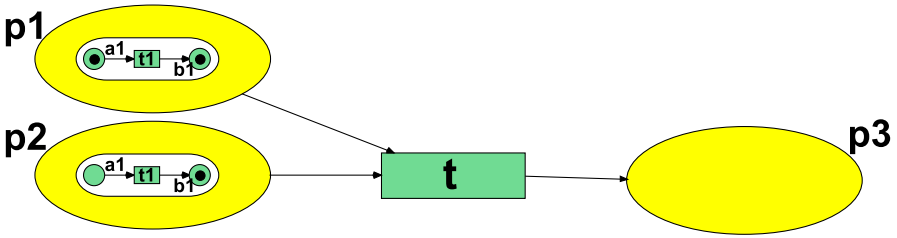
Schaltregel am Beispiel (unabhängig)

Unabhängiges Schalten...



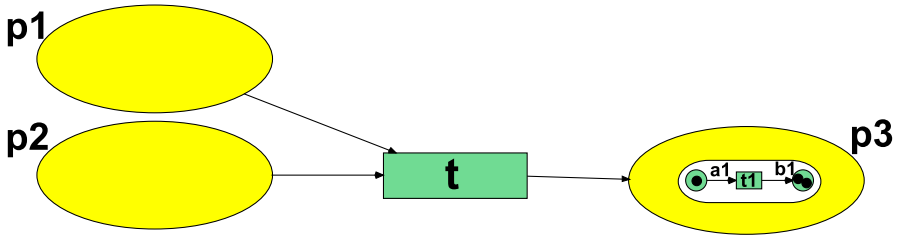
Schaltregel am Beispiel (unabhängig)

Unabhängiges Schalten...



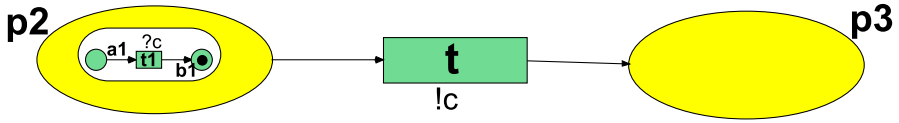
Schaltregel am Beispiel (unabhängig)

Unabhängiges Schalten...



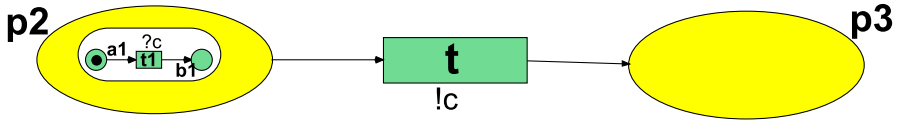
Schaltregel am Beispiel (synchron)

Synchrones Schalten...



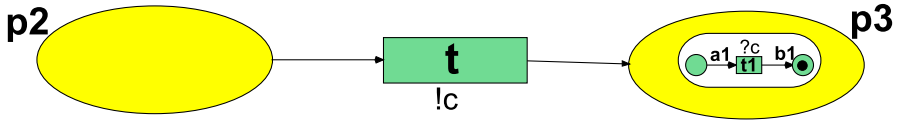
Schaltregel am Beispiel (synchron)

Synchrones Schalten...

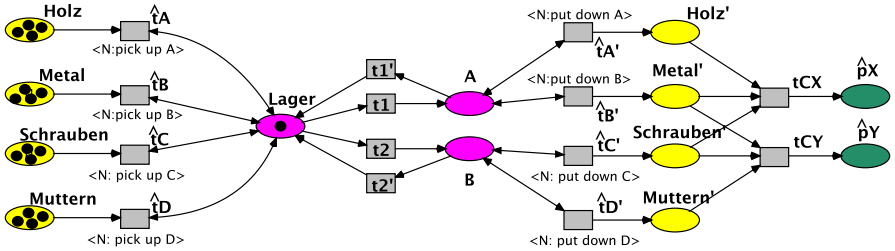


Schaltregel am Beispiel (synchron)

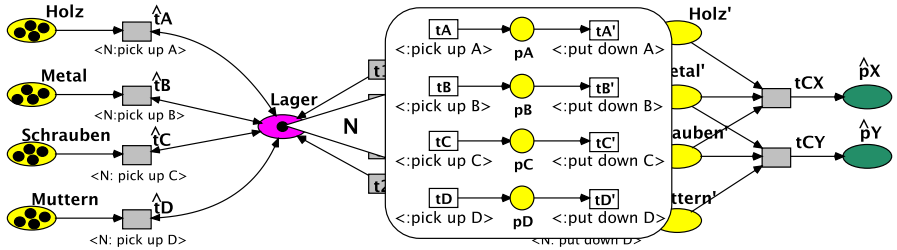
Synchrones Schalten...



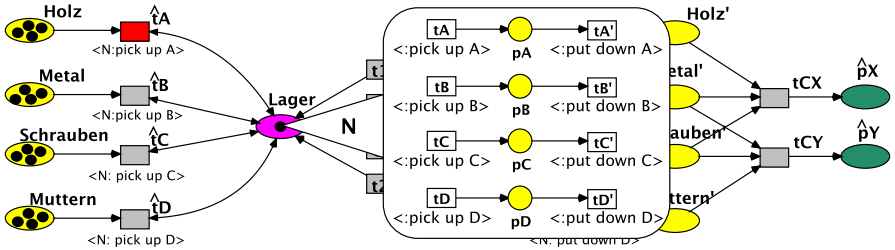
Ein größeres Beispiel



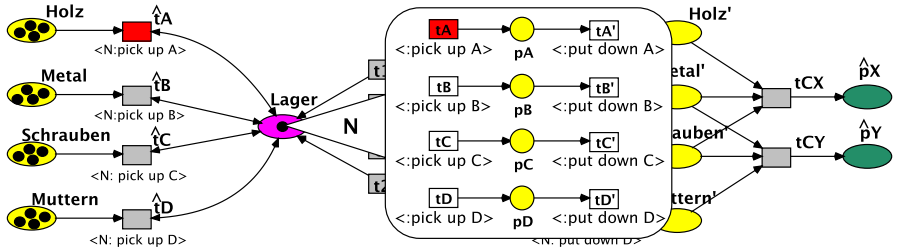
Ein größeres Beispiel



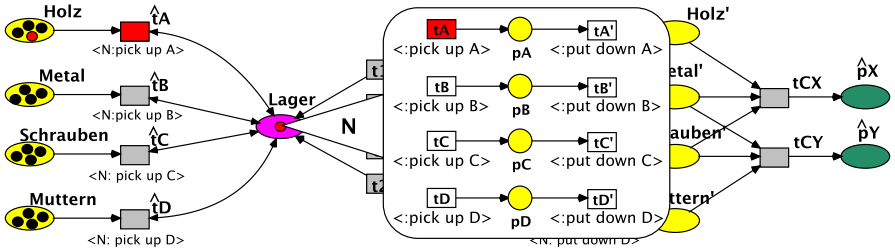
Ein größeres Beispiel



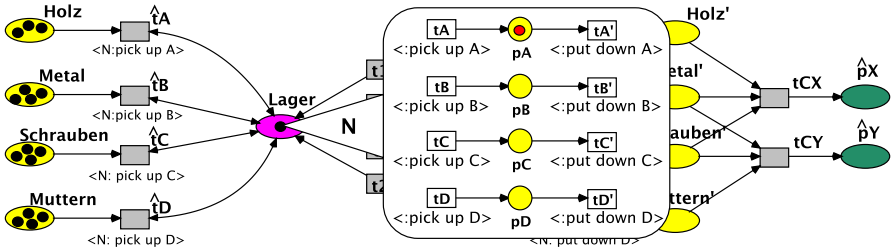
Ein größeres Beispiel



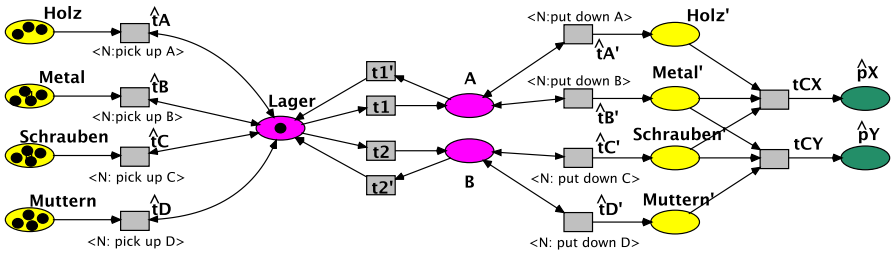
Ein größeres Beispiel



Ein größeres Beispiel



Ein größeres Beispiel



Elementare Objektsysteme

Definition (Elementares Objektsystem)

Ein elementares Objektsystem (Eos) ist ein Tupel $OS = (\widehat{N}, \mathcal{N}, d, l, \mu_0)$, wobei

1. $\widehat{N}$ ist ein P/T Netz (das *Systemnetz*).
2. $\mathcal{N}$ ist eine endliche Menge disjunkter P/T Netze (die *Objektnetze*).
3. $d : \widehat{P} \to \mathcal{N}$ ist eine *Typisierung* von $\widehat{P}$.
4. $l = (\widehat{l}, (l_N)_{N \in \mathcal{N}})$ ist die *Synchronisationsbeschriftung*.
5. μ_0 ist die *Startmarkierung*.

Anmerkung

- Größe in $O(\mathcal{N} \cdot (P_{max} + T_{max} + F_{max}) + \mu_0)$

Die Beschriftung

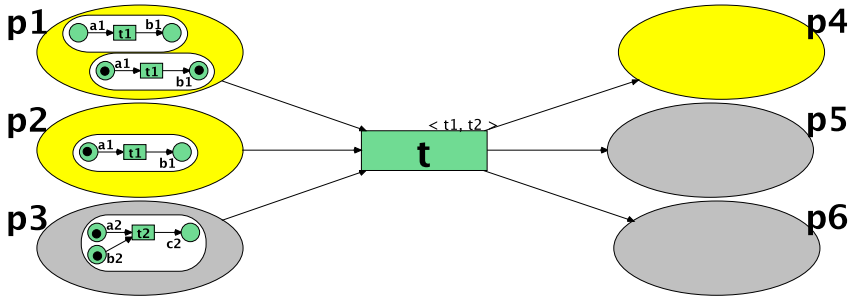
Die Synchronisationsbeschriftung ist ein Tupel $I = (\widehat{l}, (l_N)_{N \in \mathcal{N}})$

- $\widehat{l} : \widehat{T} \to (\mathcal{N} \to (C \cup \{\tau\}))$,
- $l_N : T_N \to (C \cup \{\tau\})$ für alle $N \in \mathcal{N}$.

C ist dabei eine Menge von Kanälen, $\tau \notin C$. Obige Funktionen sind alle total. Bedeutung:

- τ wird benutzt, um (System- oder Objekt-)autonomes Schalten zu modellieren.
- $l_N(t) = c$ bzw. $\widehat{l}(\widehat{t})(N) = c$ mit $c \neq \tau$ modelliert synchrones Schalten.
- Bei einem synchronen Ereignis müssen die Transitionen des Systemnetzes und des Objektnetzes mit dem gleichen Kanal beschriftet sein.

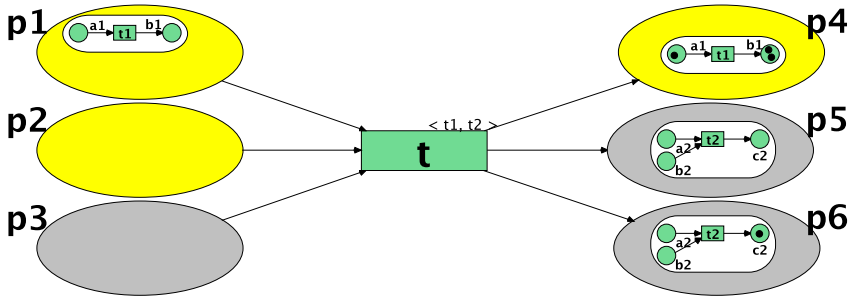
Ein Eos



Markierung:

$$\mu = p_1[\mathbf{0}] + p_1[a_1 + b_1] + p_2[a_1] + p_3[a_2 + b_2]$$

Ein Eos



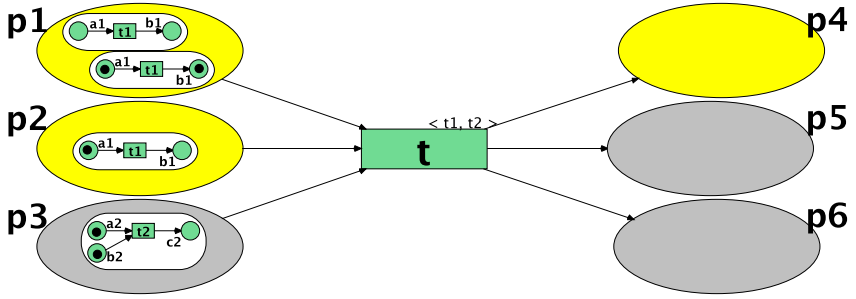
Nachfolgemarkierung:

$$\mu' = p_1[\mathbf{0}] + p_4[a_1 + 2 \cdot b_1] + p_5[\mathbf{0}] + p_6[c_2]$$

Ein Eos

Markierung:

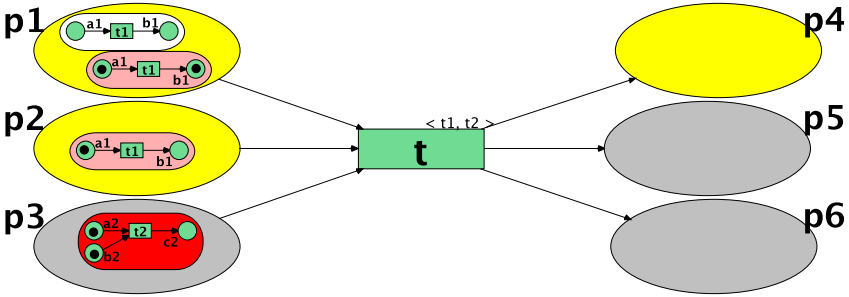
$$\mu = p_1[\mathbf{0}] + p_1[a_1 + b_1] + p_2[a_1] + p_3[a_2 + b_2]$$



Ein Eos

Teilmarkierung λ zum Schalten auswählen:

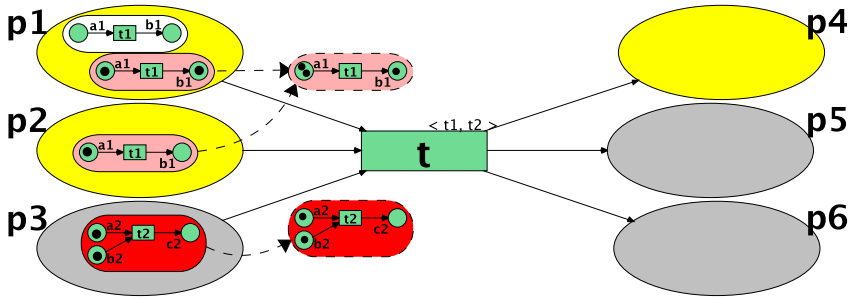
$$\lambda = p_1[a_1 + b_1] + p_2[a_1] + p_3[a_2 + b_2]$$



Ein Eos

Markierung von Netzmarken gleichen Typs aufaddieren:

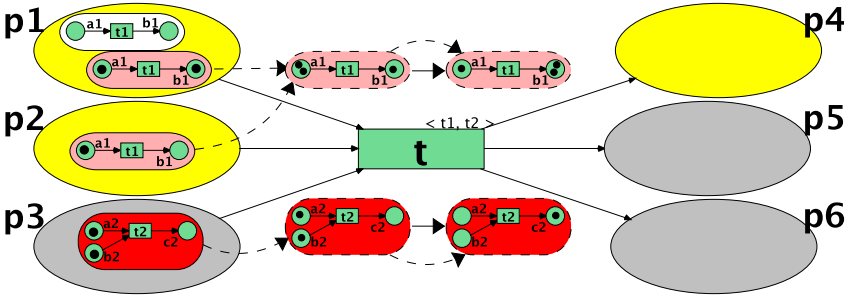
$$\Pi^2_{N_1}(\lambda) = 2 \cdot a_1 + b_1 \quad \Pi^2_{N_2}(\lambda) = a_2 + b_2$$



Ein Eos

Synchrones Schalten des Events

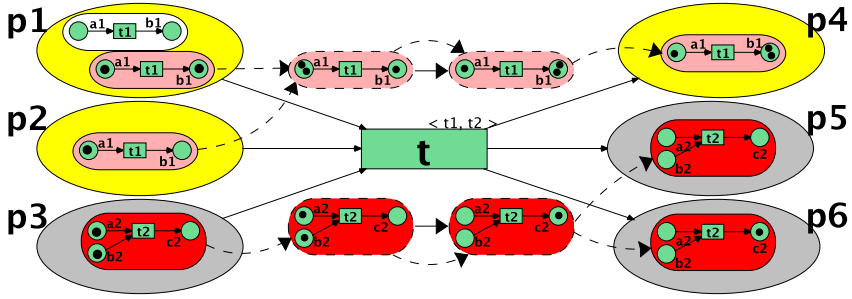
$$\theta = t[t_1, t_2]$$



Ein Eos

Markierung der Netzmarken in Teilmarkierung ρ verteilen.

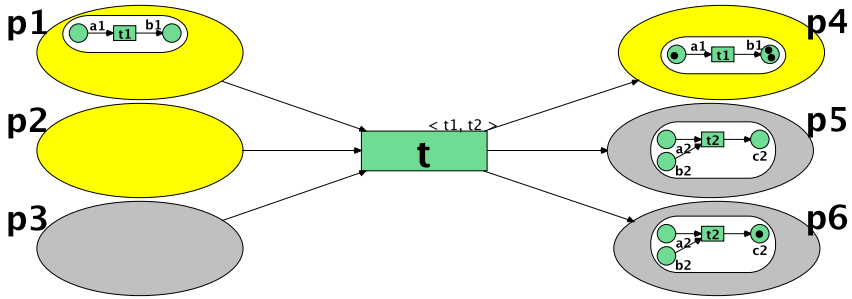
$$\rho = p_4[a_1 + 2 \cdot b_1] + p_5[\mathbf{0}] + p_6[c_2]$$



Ein Eos

Nachfolgemarkierung:

$$\mu' = \mu - \lambda + \rho = p_1[\mathbf{0}] + p_4[a_1 + 2 \cdot b_1] + p_5[\mathbf{0}] + p_6[c_2]$$



Die Schaltregel erlaubt einen Null-Test:

Theorem (Köhler 2007)

Eos können 2-Zähler-Automaten simulieren. Wichtige Probleme wie Erreichbarkeit und Lebendigkeit sind damit unentscheidbar.

Das (übliche) Ziel

Da Eos Turing-vollständig sind, ist es nun unser Ziel den Formalismus so einzuschränken, dass

- wir immer noch viele praktische Anwendungen modellieren können (so viele wie möglich)
- wichtige Verifikationsprobleme entscheidbar werden (so viele wie möglich, so schnell wie möglich)

Anmerkungen

Dies ist ein ganz typisches Vorgehen, wenn man einen Formalismus für etwas einführt.

Sichere Eos - Definitionen

Definition

Ein Petri Netz ist n-sicher mit $n \in \mathbb{N}$, wenn in jeder erreichbaren Markierung höchsten n Token auf jedem Platz liegen:

$$\forall m \in RS(\mathbf{m}_0) : \forall p \in P : m(p) \leq n$$

Für Eos existieren vier Varianten mit

$$\text{sicher}(4) \Rightarrow \text{sicher}(3) \Rightarrow \text{sicher}(2) \Rightarrow \text{sicher}(1)$$

Sichere Eos - Definitionen

Definition

OS ist *sicher(3)* (oder *sicher*) gdw. in allen erreichbaren Markierungen höchstens ein Token auf jedem Systemnetzplatz liegt *und* jede Netzmarke 1-sicher ist:

$$\forall \mu \in RS(OS): \quad \forall \widehat{p} \in \widehat{P}: \Pi^1(\mu)(\widehat{p}) \leq 1 \wedge \forall N \in \mathcal{N}: \forall p \in P_N: \forall \widehat{p}[M] \leq \mu: M(p) \leq 1$$

Sichere Eos - Ergebnisse

Theorem (5.14)

Das Erreichbarkeitsproblem ist für sichere(1) oder sichere(2) Eos unentscheidbar - sogar für str. det. und konservative Eos.

Theorem (5.11)

Der Zustandsraum eines sicheren(3) oder sicheren(4) Eos ist endlich.

Beweisidee

Mit $n := |\widehat{P}|$ und $m := max\{|P_N| \mid N \in \mathcal{N}\}$ gibt es höchstens $(1+2^m)^n$ verschiedene Markierungen.

Theorem (5.15)

Zu entscheiden, ob ein Eos sicher(3) ist, ist PSpace-vollständig.

Das Erreichbarkeitsproblem

- Bei 1-sicheren Netzen sind die meisten Probleme PSpace-schwierig.
- Ein 1-sicheres Petri Netz ist eine spezielle sicher(4) GSM.

$\Rightarrow$ Ergebnisse übertragbar.

- Bei sicheren(3) und sicheren(4) Eos maximal $(1+2^m)^n$ verschiedene Markierungen.

$\Rightarrow$ Erreichbarkeit etc. entscheidbar.

Theorem (5.18)

Erreichbarkeit ist für sichere(3) und sichere(4) Eos in ExpSpace entscheidbar und PSpace-schwierig.

Model Checking sicherer Eos

Theorem (5.22)

Gegeben ein sicheres Eos OS und eine LTL Formel ϕ. In polynomiellen Platz kann entschieden werden, ob $OS \models \phi$ gilt.

Corollary (5.23)

Erreichbarkeit für sichere Eos ist PSpace-vollständig.

Theorem (5.31)

Gegeben ein sicheres Eos OS und eine CTL Formel ϕ. In polynomiellen Platz kann entschieden werden, ob $OS \models \phi$ gilt.

Corollary (5.32)

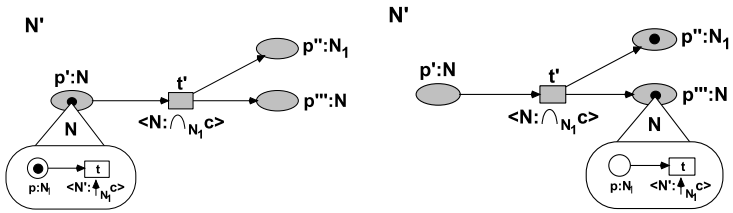
Lebendigkeit für sichere Eos ist PSpace-vollständig.

Anmerkungen

Anmerkungen

- Die Ideen sind so wie bei 1-sicheren P/T Netzen diskutiert. Hier ist noch viel weitere Arbeit nötig (Erreichbarkeitsgraph definieren, überprüfen, ob man alles wieder in PSPACE machen kann usw.)
- Wer es nachlesen möchte findet es in *Algorithms and Hardness Results for Object Nets*, Frank Heitmann, Universität Hamburg, 2013. Darauf beziehen sich auch die Nummern in den Theoremen oben.
- Dort steht auch noch mehr, z.B. kann man den Formalismus so erweitern, dass Objektnetze in der Hierarchie/Verschachtelung nach oben und unten wandern können.

Objektnetzsysteme - Schaltregel



$$\phi_{\Uparrow_{N_1},\cap_{N_1}}(t,t',\lambda,\lambda',\rho,\rho') \iff$$
$$\Pi^1(\lambda) = \mathbf{pre}(t') \wedge \Pi^1(\rho) = \mathbf{post}(t') \wedge$$
$$\Pi^1(\lambda') = \mathbf{pre}(t) \wedge \Pi^1(\rho') = \mathbf{post}(t) \wedge$$
$$\forall N' \in \mathcal{N} \setminus \{N, N_1\} : \Pi^2_{N'}(\rho) = \Pi^2_{N'}(\lambda) \wedge$$
$$\forall N' \in \mathcal{N} \setminus \{N_1\} : \Pi^2_{N'}(\rho') = \Pi^2_{N'}(\lambda') \wedge$$
$$\Pi^2_N(\rho) = \Pi^2_N(\lambda) - \lambda' + \rho' \wedge$$
$$\Pi^2_{N_1}(\rho) = \Pi^2_{N_1}(\lambda') \wedge$$
$$\Pi^2_{N_1}(\rho') = \mathbf{0}$$

Ein Ergebnis

Anmerkung

Auch hierfür kann ein Sicherheitsbegriff eingeführt werden und das CTL- und LTL-Model-Checking gelingt wieder in PSPACE!
Das führt hier aber zu weit ...

Zusammenfassung

Was wir bisher gemacht haben:

- CTL Model Checking
  - direkt auf dem Graphen des Modells
- LTL Model Checking
  - mittels Umweg über Büchi-Automaten

Zusammefassung - heute

Selten ist das Modell ein Transitionssystem. Es werden andere Formalismen benutzt, die dann in Transitionssysteme umgewandelt werden. Heute:

- 1-sichere Petri Netze
- sichere Eos

Dabei hätte man direkt auf obige Ansätze zurückgreifen können. Um Speicherplatz zu sparen kann man aber auf eine

- on-the-fly Methode

zurückgreifen. Diese haben wir heute kennengelernt.

Zusammenfassung - heute

Heute also:

- LTL-MC 1-sicherer P/T Netze
- CTL-MC 1-sicherer P/T Netze [kommt noch genauer]
- LTL-MC sicherer Eos
- CTL-MC sicherer Eos

Literatur

Literaturhinweis

Die heutige Vorlesung basierte vor allem auf:

1. Javier Esparza. *Decidability and complexity of Petri net problems an introduction*. In Wolfgang Reisig and Grzegorz Rozenberg, editors, Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, volume 1491 of Lecture Notes in Computer Science, pages 374–428. Springer-Verlag, 1998. (Sehr schönes Paper...)
2. Frank Heitmann. *Algorithms and Hardness Results for Object Nets*. PhD thesis, University of Hamburg, 2013.