1. 2 Modelle eingebetteter Systeme

Zur Vorlesung

Embedded Systems

WS 14/15

Reiner Kolla

1.2.1 Einführendes Beispiel zur Modellierung

Wir beginnen den Diskurs mit einem

Beispiel: Steuerung eines Fahrstuhls

Spezifikation in natürlicher Sprache

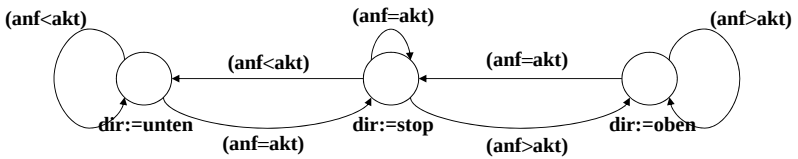
"Basierend auf einer Anfrage einer gewünschten Etage soll der Fahrstuhl bis zur gewünschten Etage fahren und dort so lange stehen bleiben, bis eine neue Anfrage vorliegt"

Sehr ungenau !
Was passiert z.B., wenn sich die Anfrage während der Fahrt des Fahrstuhls ändert?

Spezifikation und Modellierung -- Beispiel

Beispiel: Steuerung eines Fahrstuhls

Spezifikation durch einen Moore-Automaten

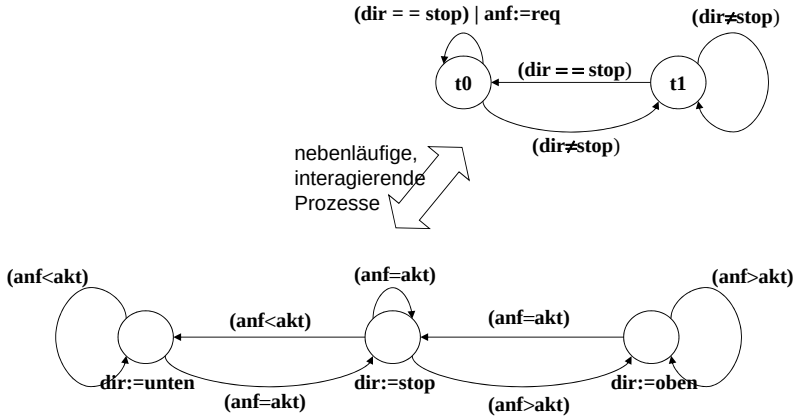


Dies ist eine formale, aber ebenfalls unvollständige Spezifikation !

Was geschieht, wenn sich „anf“ im Zustand s0 oder s2 ändert?

Spezifikation und Modellierung

Erweiterung der Spezifikation
durch einen Mealy-Automaten



… und die Moral der Geschicht' ....

- Formale Methoden zur Spezifikation von Systemen sind notwendig, um Realisierungen mit korrektem Verhalten synthetisieren zu können.
- Spezifizieren mit formalen Methoden ist nicht trivial
  ... aber der einzige gehbare Weg !

Wir wollen zunächst ein sehr allgemeines Modell zur Spezifikation asynchroner, nebenläufiger Syteme entwickeln, aus dem sich viele andere Modelle durch Spezialisierung ableiten lassen.

1.2.2 Petri-Netze

Definition

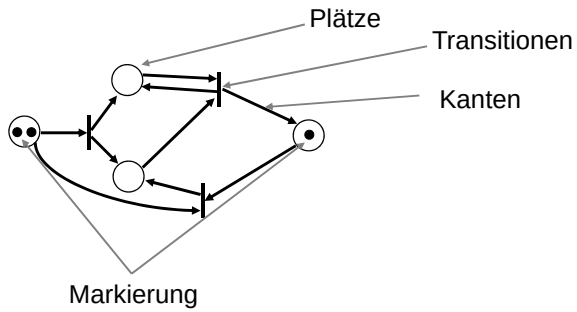
Ein Petri-Netz ist ein 6-Tupel $G=(P,T,F,K,W,M_0)$ mit

- $P = \{ p_1,...,p_m \}$ die Menge der Plätze (oder Stellen)
- $T = \{ t_1,...,t_n \}$ die Menge der Transitionen
- $P \cap T = \emptyset$. Beide Mengen zusammen bilden die Menge der Knoten
- $F \subseteq (P \times T) \cup (T \times P)$, die Menge der Kanten, auch Flussrelation genannt
- $K : P \rightarrow \mathcal{N} \cup \{\infty\}$, die jedem Platz eine Kapazität zuordnet
- $W : F \rightarrow \mathcal{N}$, die jeder Kante ein Kantengewicht zuordnet
- $M_0 : P \rightarrow \mathcal{N}_0$ mit $\forall p \in P: M_0(p) \leq K(p)$, die Anfangsmarkierung

Petri-Netze sind also gerichtete bipartite Graphen, denen man eine Interpretation im Sinne erreichbarer Markierungen gibt. Wir wollen zuvor aber noch am Beispiel eine gängige Notation einführen:

Petri-Netze --Beispiel

Beispiel



Es seien alle Kapazitäten gleich ∞ und Kantengewichte alle gleich 1

Schreibweise	
$\forall t \in T$: $\bullet t = \{ p; (p,t) \in F \}$	-- den **Vorbereich** von t
$\forall p \in P$: $\bullet p = \{ t; (t,p) \in F \}$	-- den **Vorbereich** von p
$\forall t \in T$: $t\bullet = \{ p; (t,p) \in F \}$	-- den **Nachbereich** von t
$\forall p \in P$: $p\bullet = \{ t; (p,t) \in F \}$	-- den **Nachbereich** von p

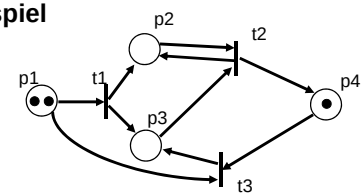
Schaltbereitschaft von Transitionen

Definition

Eine Transition $t \in T$ eines Petri-Netzes $G=(P,T,F,K,W,M_0)$ heißt schaltbereit unter der Markierung $M:P \rightarrow N_0$, geschrieben $M[t\rangle$, wenn

1. $\forall p \in \bullet t \setminus t\bullet$: $M(p) \geq W(p,t)$
2. $\forall p \in t\bullet \setminus \bullet t$: $M(p) \leq K(p) - W(t,p)$
3. $\forall p \in t\bullet \cap \bullet t$: $W(p,t) \leq M(p) \leq K(p) - W(t,p) + W(p,t)$

Beispiel

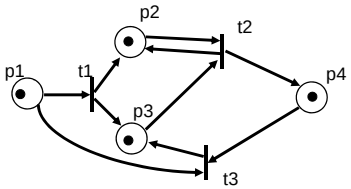


Schaltbereit sind t1 und t3

Kapazitäten alle gleich ∞
Kantengewichte alle gleich 1

Schaltbereitschaft von Transitionen -- ff

Feuert einer der schaltbereiten Transitionen, z.B. t_1, so verändert sich die Markierung !



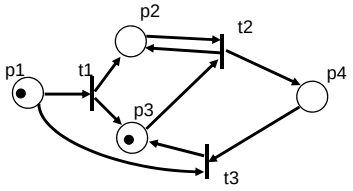
Definition

Eine Transition $t \in T$ eines Petri-Netzes $G=(P,T,F,K,W,M_0)$ schaltet von der Markierung M auf M' geschrieben $M[t\rangle M'$, wenn t unter M schaltbereit ist und M' aus M wie folgt gebildet wird:

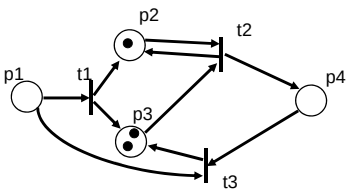
- $M'(p) = M(p)-W(p,t)$, falls $p \in {\bullet t} \setminus t{\bullet}$
- $M'(p) = M(p)+W(t,p)$, falls $p \in t{\bullet} \setminus {\bullet t}$
- $M'(p) = M(p)-W(p,t)+W(t,p)$, falls $p \in t{\bullet} \cap {\bullet t}$
- $M'(p) = M(p)$, sonst

Schaltbereitschaft von Transitionen -- ff

die Transition t_3 hätte auch feuern können.



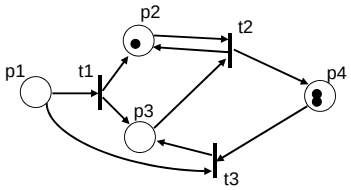
Sogar beide gleichzeitig.



Definition

Zwei Transitionen $t_1,t_2 \in T$ eines Petri-Netzes $G=(P,T,F,K,W,M_0)$ stehen in Konflikt, wenn $M[t_1\rangle$ und $M[t_2\rangle$ gelten, aber nicht $M[\{t_1,t_2\}\rangle$ gilt. Dabei bezeichne $M[\{t_1,...,t_n\}\rangle$ die gleichzeitige nebenläufige Schaltbereitschaft aller Transitionen $t_1,...,t_n$ (ohne Verletzung der Markenbedingungen)

Let's play the game !



... Huch, das Spiel geht aber schnell zu Ende

Deadlock !

Aktivierbare und lebendige Transitionen

Definition

Ist $M:P \rightarrow N_0$ eine Markierung eines Petri-Netzes, so heißt eine Markierung M' des Petri-Netzes **erreichbar**, wenn

- $\exists t \in T: M[t\rangle M'$ oder
- $\exists t \in T: M''$ $[t\rangle M'$ und M'' ist von M aus erreichbar

$[M\rangle$ bezeichne die Menge der von M aus erreichbaren Markierungen

Eine Transition t eines Petri-Netzes heißt bzgl. einer Markierung M **tot**, wenn sie unter keiner von M aus erreichbaren Markierung schaltbereit ist, d.h. $\forall M' \in [M\rangle: \neg(M' [t\rangle)$. Ansonsten heißt sie **aktivierbar**.

Eine Transition t eines Petri-Netzes heißt bzgl. einer Markierung M **lebendig**, wenn sie bzgl. jeder von M aus erreichbaren Markierung aktivierbar ist.

Deadlockfreie und lebendige Petri-Netze

Definition

Ein Petri-Netz $G=(P,T,F,K,W,M_0)$ heißt **deadlockfrei** oder **schwach lebendig**, wenn es zu jeder von M_0 aus erreichbaren Markierung M' eine Transition $t \in T$ gibt, die schaltbereit ist.

Definition

Ein Petri-Netz $G=(P,T,F,K,W,M_0)$ heißt **lebendig** oder **stark lebendig**, wenn jede seiner Transitionen bzgl. M_0 lebendig ist.

Beschränkte Petri-Netze

Definition

Sei $G=(P,T,F,K,W,M_0)$ ein Petri-Netz und $B:P \rightarrow \mathbb{N}_0 \cup \{\infty\}$ eine Abbildung, die jeder Stelle eine 'kritische Markenzahl' zuordnet ($B(p) \leq K(p)$). Das Petri-Netz G heißt ***B*-sicher** oder ***B*-beschränkt**, wenn $\forall M \in [M_0\rangle\ \forall p \in P: M(p) \leq B(p)$.

Das Petri-Netz G heißt **sicher** oder **beschränkt**, wenn es eine natürliche Zahl b gibt, für die G b-beschränkt ist.

Satz

Ein Petri-Netz ist genau dann beschränkt, wenn seine Erreichbarkeitsmenge $[M_0\rangle$ endlich ist.

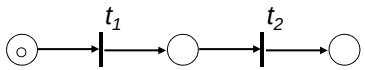
Beweis:

Ist G b-beschränkt für ein $b \in \mathbb{N}_0$, dann gibt es höchstens $(b+1)^{\#P}$ viele verschiedene Markierungen.

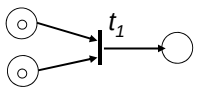
Ist G nicht b-beschränkt, ist die Zahl der Marken schon auf mindestens einem Platz p in $[M_0\rangle$ unendlich und damit auch $[M_0\rangle$

Ausdruckskraft von Petri-Netzen

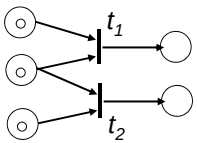
Sequentialität



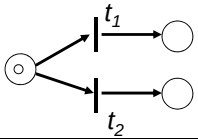
Synchronisation



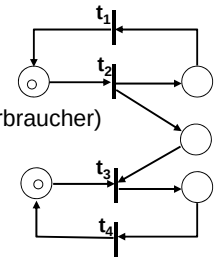
Resourcenkonflikt



Nichtdeterministische Verzweigung



Nebenläufigkeit
(Bsp: Erzeuger/Verbraucher)



Überdeckung von Markierungen

Wir wollen uns nun mit der Frage beschäftigen, ob ein Netz beschränkt ist. Dies macht eigentlich nur Sinn, wenn wir keine Kapazitätsgrenzen auf Plätzen haben. Von nun an sei die Kapazität eines Platzes stets unendlich.

Seien M und M' Markierungen. Dann ist $M+M'$ die Markierung die man durch $M+M'(p) := M(p) + M'(p)$ erhält. Entsprechend sei „-“ auf Markierungen definiert.

Eine Markierung M **überdeckt** M' $(M \geq M')$ genau dann, wenn für alle Plätze p gilt: $M(p) \geq M'(p)$

Beobachtungen:

$M \geq M' \Leftrightarrow M - M' \geq 0$

$M \geq M' \Rightarrow \forall t \in T: (M'[t\rangle \Rightarrow M[t\rangle)$

Für eine Folge $t_1,...,t_k$ von Transitionen, mit

$M_1[t_1\rangle M_2[t_2\rangle \ldots M_k[t_k\rangle M_{k+1} = M_1 + X,\ X \geq 0$

ist jedes $M_1 + nX$, $n \in N$ von M_1 aus erreichbar. Damit ist das Netz nicht beschränkt für keinen Platz p mit $X(p) > 0$.

ω - Erweiterungen

Eine ω-erweiterte Markierung ist gegeben durch
M: $P \to \mathbb{N}_0 \cup \{\omega\}$

Wobei $M \geq M' \Leftrightarrow \forall p$: (M(p) = ω *oder* $M(p) \geq M'(p)$)
für ω-erweiterte Markierungen

ω-erweiterte Markierungen stehen für unendliche Mengen von Markierungen, genauer

$M' \in Set(M) :\Leftrightarrow \forall p: M'(p) = M(p)$ *oder* $M(p) = \omega$

Eine Transition *t* ist unter einer ω- Erweiterung *M* feuerbereit, wenn es ein $M' \in Set(M)$ gibt mit $M'[t>$

Für *M,M'* sei $\Omega(M,M')$ die Markierung mit minimaler Anzahl von ω-markierten Plätzen, so dass

$M \in Set(\Omega(M,M'))$ und $M' \in Set(\Omega(M,M'))$

Der Überdeckungsgraph

Man kann nun zu jedem Petri-Netz wie folgt einen Überdeckungsgraphen von M_0 aus konstruieren:

Knoten: ω-erweiterte Markierungen M: $P \to \mathbb{N}_0 \cup \{\omega\}$

M_0 ist ein Knoten des Graphen

Kanten:

Ist *M* ein Knoten, dann ist für jede feuerbereite Transition $M[t>M'$

M' Nachfolgeknoten, falls kein Vorgänger M'' von *M* existiert mit $M' \geq M''$

$\Omega(M'',M')$ Nachfolgeknoten falls M'' Vorgänger von *M* mit $M' \geq M''$

Iteriere die Konstruktion per depth first search von M_0 aus, bis keine neuen Knoten mehr hinzukommen.

Der Überdeckungsgraph -- Fakten

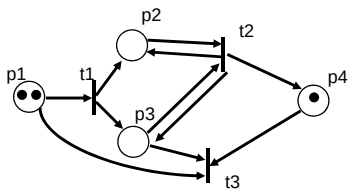
(1) Der Überdeckungsgraph ist stets endlich (ohne Beweis), d.h. die DFS bricht irgendwann ab.

(2) Ein Petri-Netz ist beschränkt dann und nur dann, wenn bei der Konstruktion des Überdeckungsgraphen keine ω-erweiterten Knoten entstehen.

(3) Petri-Netze können den gleichen Überdeckungs-graphen haben und doch ungleiches Verhalten.

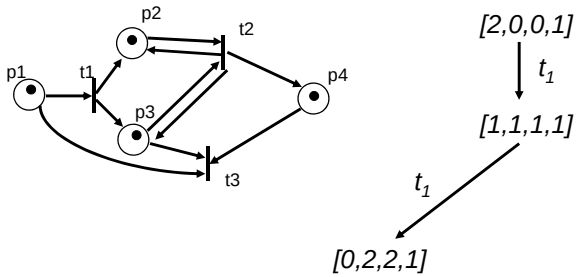
(4) Bei beschränkten Petri-Netzen ist das Verhalten genau dann gleich, wenn die Überdeckungsgraphen isomorph sind.

Der Überdeckungsgraph -- Beispiel

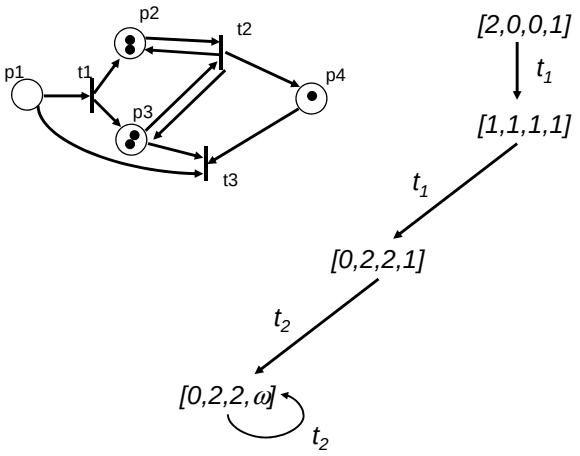


[2,0,0,1]
t_1
[1,1,1,1]

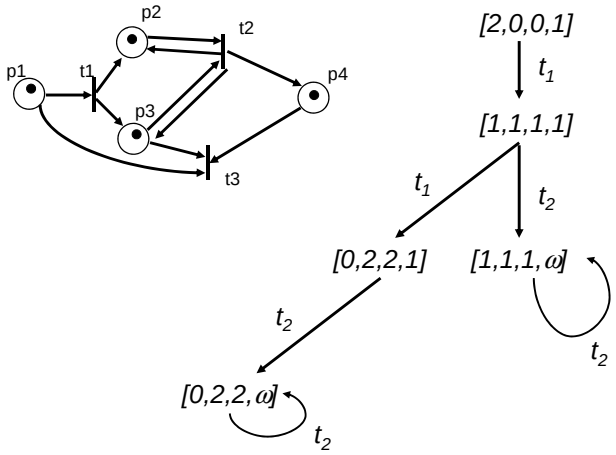
Der Überdeckungsgraph -- Beispiel



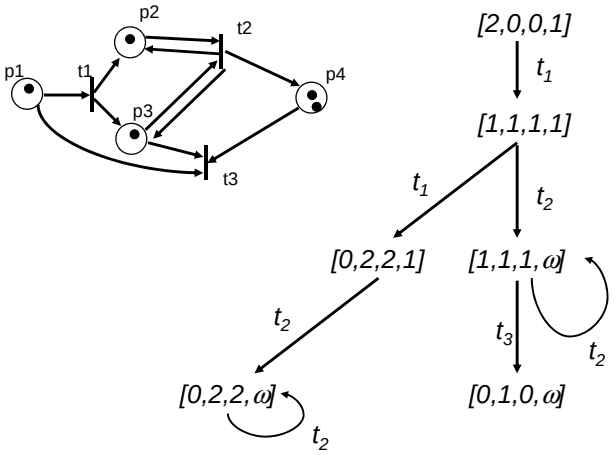
Der Überdeckungsgraph -- Beispiel



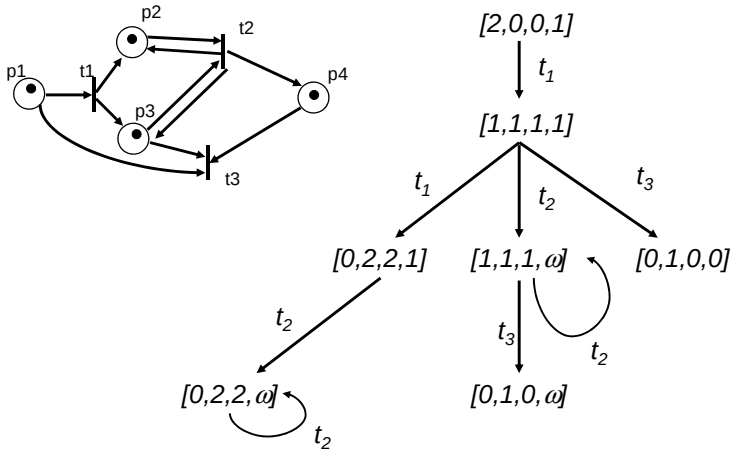
Der Überdeckungsgraph -- Beispiel



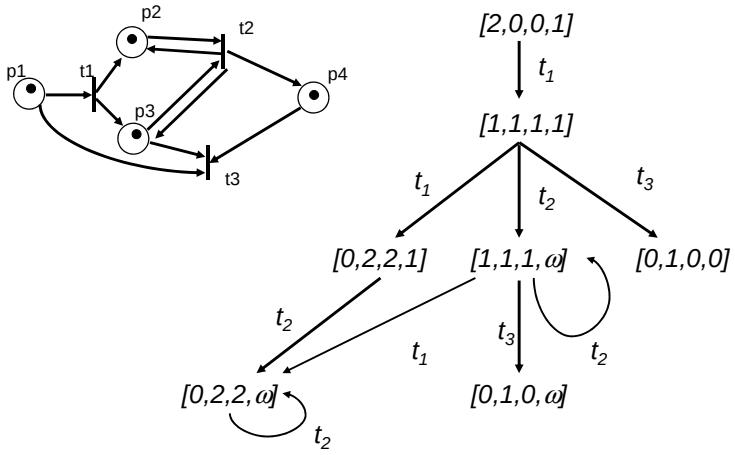
Der Überdeckungsgraph -- Beispiel



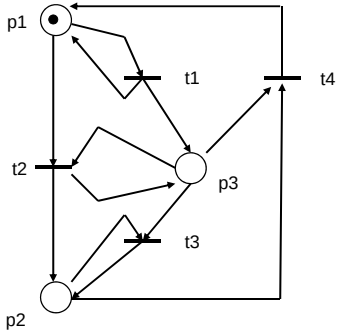
Der Überdeckungsgraph -- Beispiel

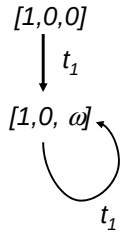


Der Überdeckungsgraph -- Beispiel

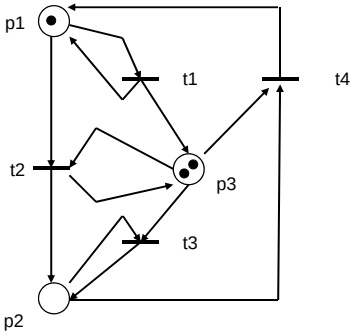


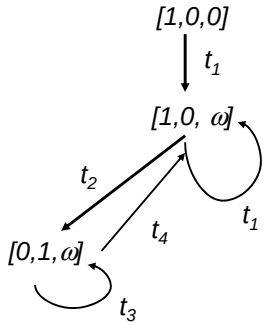
Der Überdeckungsgraph – Beispiel (2)



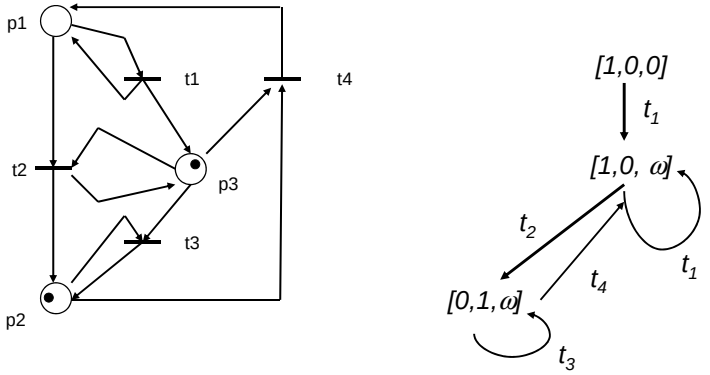


Der Überdeckungsgraph – Beispiel (2)



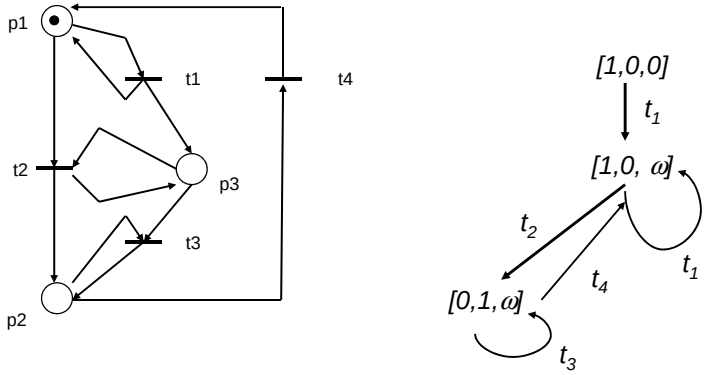


Der Überdeckungsgraph – Beispiel (2)



Deadlock bei t1,t2,t3 nicht bei t1,t1,t2,t3

Der Überdeckungsgraph – Beispiel (2)



Gleicher Überdeckungsgraph, deadlock frei

1.2.3 Zustandsorientierte Modelle

Definition

Ein **endlicher Automat** ist gegeben durch ein 6-Tupel
$M:=(I,O,S,R,\delta,\lambda)$
Dabei sind S, I, O endliche Mengen.

S heißt **Zustandsmenge**, $R \subseteq S$ die Menge der **Startzustände**

I **Eingabemenge**,

O **Ausgabemenge**,

$\delta: S \times I \rightarrow S$ heißt **Übergangsfunktion** und

$\lambda: S \times I \rightarrow O$ **Ausgabefunktion**.

Sind δ, λ („echte“) Relationen, nennt man den Automaten auch **nichtdeterministisch**.

Sind δ, λ partiell, d.h. nicht für alle Paare (s,x) definiert, nennt man den Automaten auch **unvollständig**. Es muss dann aber gelten, dass $\lambda(s,x)$ definiert $\Rightarrow$ $\delta(s,x)$ definiert.

Endlicher Automat als Petri-Netz

Endliche Automaten können als Spezialfall interpretierter Petri-Netze aufgefasst werden:

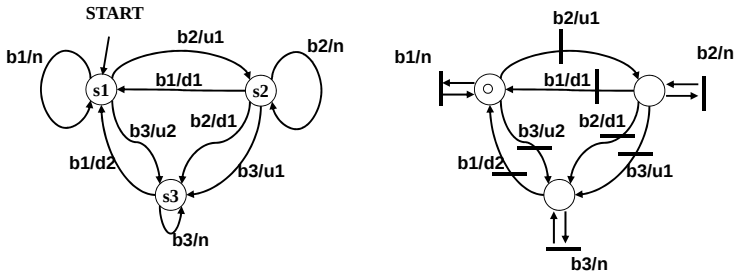
Ein nichtdeterministischer endlicher Automat ist ein Petri-Netz $G=(P,T,F,K,W,M_0)$ mit

- $\forall t \in T$: $| \bullet t | = | t \bullet | = 1$
- $\exists p_0 \in P$: $M_0(p_0) = 1$ und $\forall p \neq p_0$: $M_0(p) = 0$
- $K = W = \underline{1}$
- Jeder Transition ist ein Prädikat (Eingabeereignis) als zusätzliche Schaltbedingung und ein Ausgabeereignis als Aktion zugewiesen

Beobachtung:

Aus $\forall t \in T$: $| \bullet t | = | t \bullet | = 1$ und $\Sigma_{p \in P} M_0(p) = 1$ folgt, dass stets genau eine Marke in einer erreichbaren Markierung vorliegen kann. D.h. es gibt höchsten $\#P$ viele erreichbare Markierungen, die man mit dem Platz, auf dem die Marke liegt, identifizieren kann.

Beispiel: endlicher Automat



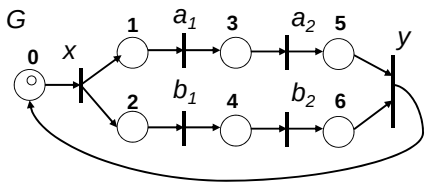
Beschränktes Petri-Netz als endlicher Automat

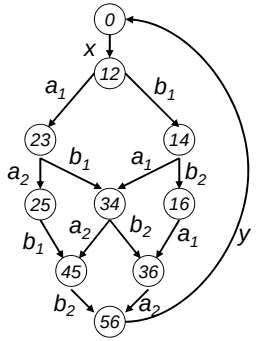
Endliche Automaten sind ihrerseits ebenfalls ein so allgemeines Modell, dass man beschränkte Petri-Netze darauf zurückführen kann:

Das Zustandsübergangsdiagramm (state transition diagram STD) zu einem beschränkten Petri-Netz $G=(P,T,F,K,W,M_0)$ ist gegeben durch $STD(G) = (T, T, [M_0>, M_0, \delta, \lambda)$, wobei δ, λ wie folgt definiert sind:

- $\forall M \in [M_0>: \delta(M,t) := M' \Leftrightarrow M[t> = M'$
- $\lambda(M,t) := t$, falls $\delta(M,t)$ definiert ist.

Beispiel





Beschränktes Petri-Netz als endlicher Automat - ff

Man erkennt aus der Konstruktion des STD's:

- Das STD zu einem *b*-beschränkten Petri-Netz kann $(b+1)^{\#P}$ viele Zustände haben.
- Petri-Netze können Systeme mit sehr großer Zustandszahl kompakt darstellen.

Gibt es auch Möglichkeiten, Systeme mit großer Zustandszahl kompakt durch Automaten zu beschreiben?

Erweiterung zustandsorientierter Modelle

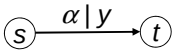
Endliche Automaten können durch folgende Maßnahmen ergonomischer notiert werden:

- Hierarchie
- Nebenläufige, kooperierende Automaten
- ➔ Kommunizierende hierarchische nebenläufige Automaten (Statecharts [Harel])

Wir wollen im folgenden eine kompakte graphische Notation für nebenläufige, kooperierende Automaten entwickeln, und jedem Konstrukt eine Bedeutung im Sinne eines endlichen Automaten zuordnen. Dieser Weg wird etwa in Statecharts genommen:

Statecharts

Die Eingabesymbole endlicher Automaten sind in der Regel Sensorsignale oder Zustandssignale kooperierender Automaten.
➢ Notiere die Übergänge als Kanten mit Prädikaten (Ereignisse), die auf genau den Eingaben erfüllt sind, unter denen der Übergang stattfindet.
Die Ausgabesymbole sind in der Regel Steuersignale für Aktoren
➢ Notiere neben die Übergangsprädikate Aktionen

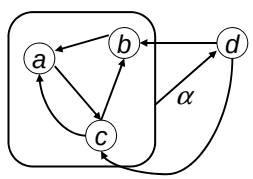


Unter allen Eingaben *x* mit $\alpha(x)$ gehe mit Ausgabe *y* nach *t*

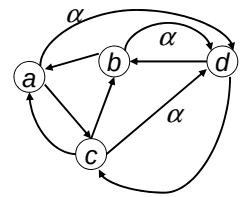
Wir nehmen ferner an, dass die Maschine vollständig ist, d.h. wann immer für die von *s* ausgehenden Kanten $\alpha_1,\ldots,\alpha_k$ gilt $\neg(\alpha_1 \vee \ldots \vee \alpha_k)$ verbleiben wir im Zustand *s*.
Die Benutzung von Prädikaten statt Eingabesymbolen erspart die Notation mit Mehrfachkanten. Sehr oft hat man jedoch für ganze Teilmengen von Zuständen stets den gleichen Übergang zu einem festen Zielzustand (Bsp.: Verlassen einer Kontrollschleife).

Statecharts -- Gruppierung

Gruppiere Zustände durch Einkreisen, und erlaube Übergänge aus Zustandsgruppen: Ein Übergang aus einer Zustandsgruppe steht für Übergänge von jedem Zustand der Gruppe aus zu dem gegebenen Zielzustand.
Beispiel:



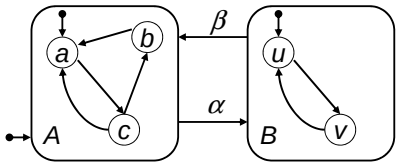
Steht für



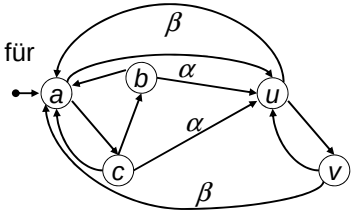
Sehr oft liegt der Zielzustand eines solchen Gruppenübergangs ebenfalls in einer Gruppe und wird dort stets als Startzustand angenommen. Es bietet sich also an, ganze Zustandsgruppen zu Zuständen eines übergeordneten Diagramms zu machen.
➢ **Hierarchischer Entwurf**

Statecharts -- Hierarchie

Gruppiere Zustände durch Einkreisen, benenne sie und markiere einen Startzustand.
Übergänge in einem Übergangsdiagramm mit Gruppen stehen für Übergänge von jedem Zustand der Quellgruppe in den ausgezeichneten Startzustand der Zielgruppe:
Beispiel:



Steht für



Bei mehreren Gruppen ist stets eine Gruppe als Startgruppe auszuzeichnen (im Beispiel A und damit a).
Bisher ist unsere Notation nur ergonomischer geworden, wir müssen aber noch alle Zustände hinzeichnen. Die eigentliche Ausdrucksstärke kommt nun durch die Einführung nebenläufiger, kooperierender Automaten ins Spiel:

Statecharts -- Nebenläufigkeit

Gruppiere zwei oder mehrere Automaten mit ausgezeichnetem Startzustand als nebenläufig.

Um formal die Bedeutung zweier nebenläufig gruppierter Automaten festlegen zu können, müssen wir zunächst ein paar Annahmen treffen:
• die Eingabemenge der Automaten sei $I_1 = I \times S_2$ *bzw.* $I_2 = I \times S_1$
• die Ausgabemenge beider Automaten sei $O=2^A$, wobei *A* eine Menge von zueinander kompatiblen Aktionen ist.

Dann ist der durch nebenläufige Gruppierung von $M_1:=(I_1,O,S_1,\{r_1\},\delta_1,\lambda_1)$, $M_2:=(I_2,O,S_2,\{r_2\},\delta_2,\lambda_2)$, definierte Automat gegeben durch folgenden **Produktautomaten**

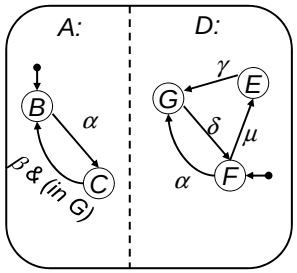
$$M:=(I, O, S_1 \times S_2,\{(r_1,r_2)\},\delta,\lambda),$$

mit $\delta((a,b),x) := (u,v)$
$: \Leftrightarrow \delta_1(a,(x,b)) = u$ und $\delta_2(b,(x,a)) = v$
sowie $\lambda((a,b),x) := \lambda_1(a,(x,b)) \cup \lambda_2(b,(x,a))$

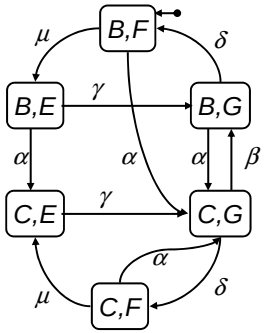
Da wir die Übergänge in M_1, M_2 durch Prädikate beschreiben, können wir so optional in M_1 (M_2) Zustände von M_2 (M_1) als Bedingung fordern, oder machen den Übergang unter jedem Zustand in M_2 (M_1) .

Statecharts -- Nebenläufigkeit

Beispiel:



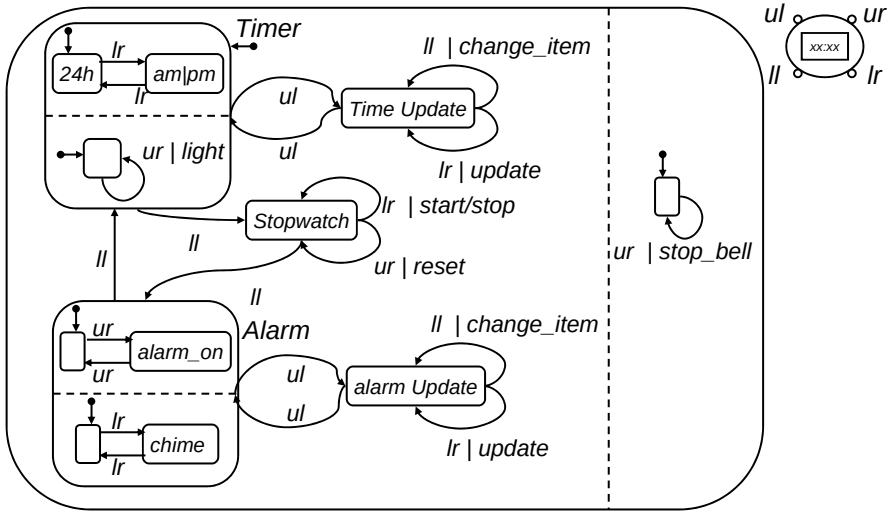
steht für



Man erkennt:
- die Größen der Automaten multiplizieren sich
- die Synchronisation von A mit D im Zustand C:
  Ein Übergang unter β findet erst statt, wenn D nach G gelaufen ist.

Statecharts -- Beispiel

Steuerung einer Armbanduhr mit Alarm und Stoppuhr



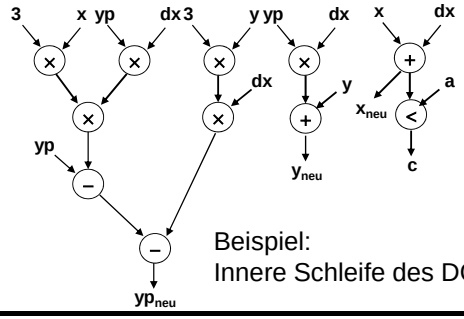
1.2.4 Aktivitätsorientierte Modelle

Datenflussgraphen (DFG)

sind gerichtete Graphen, die die durchzuführenden Berechnungen allein über die Verfügbarkeit der dazu notwendigen Daten definieren.

Knoten entsprechen Operationen (Aktoren),

Kanten entsprechen Datenabhängigkeiten



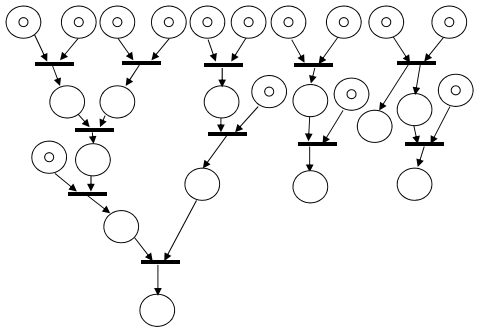
Beispiel:
Innere Schleife des DGL Lösers als DFG

Datenflussgraphen als Petri-Netze

DFGs sind ebenfalls ein Sonderfall von Petri-Netzen:

Ersetze die Operationsknoten durch Transitionen und die Kanten durch Plätze

→ Petri-Netz:



Jeder Platz hat genau eine Nachfolge- und eine Vorgängertransition, denn jede Datenabhängigkeit hat genau einen **Produzenten** und genau einen **Konsumenten → Markierte Graphen**

Markierte Graphen

Formale Definition kann über Petri-Netze erfolgen.

Definition -- MG als Petri-Netz

Ein **markierter Graph (MG)** ist ein Petri-Netz $G=(P,T,F,K,W,M_0)$ mit folgenden Eigenschaften:

- $\forall p \in P: |{\bullet}p| = |p{\bullet}| = 1$
- $W = 1$
- $\forall p \in P: K(p)=\infty$,

Wir nehmen ferner an, dass die Marken (Daten) jeder Stelle p in der Reihenfolge entnommen werden, in der sie dort entstehen.
(FIFO first in first out)

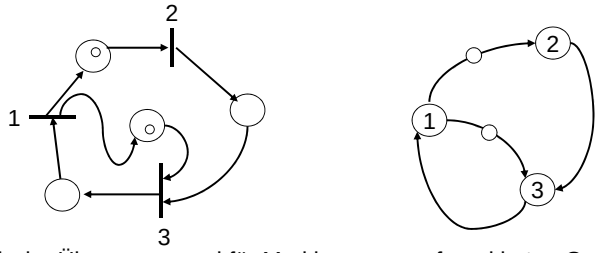
Die Einschränkungen führen dazu, dass man keine Konflikte mehr auf den Transitionen hat, und damit keinen Nichtdeterminismus, bzw. Verzweigungen bei wechselseitigem Ausschluss, beschreiben kann.

Andererseits sind dafür Sicherheits- und Lebendigkeitseigenschaften algorithmisch effizient lösbar:

Markierte Graphen -- ff

Da in markierten Graphen die Plätze stets genau einen Vorgänger und einen Nachfolger haben, kann man die Plätze auch als Knoten im Netz weglassen und direkt die Kanten zwischen Transitionen betrachten und markieren. Daher stammt auch der Begriff markierte Graphen. (Knoten ↔ Transitionen, Kanten ↔ Plätze)

Beispiel: MG als Petri-Netz als markierter, gerichteter Graph



Einfache Übergangsregel für Markierungen auf markierten Graphen:
Ein Knoten kann feuern, wenn alle einlaufenden Kanten markiert sind.
Er nimmt von jeder eingehenden Kante eine Marke, und legt auf jede ausgehende Kante eine Marke.

Markierte Graphen -- Definition ff

Definition -- MG als Graph

Ein **markierter Graph** ist ein gerichteter Graph $G=(V,E,M_0)$ wobei

$M_0: E \rightarrow N_0$

die Anfangsmarkierung der Kanten ist.

Sei $M: E \rightarrow N_0$ eine Markierung der Kanten. Ein Knoten v heißt **feuerbereit** unter M, $M[v> :\Leftrightarrow \forall e \in z^{-1}(v): M(e)>0$.

Eine Markierung M' ensteht aus M durch feuern von v, $M[v>M'$, $:\Leftrightarrow$

- $\forall e \in q^{-1}(v): M'(e)= M(e) + 1$
- $\forall e \in z^{-1}(v): M'(e)= M(e) - 1$, *und*
- $M'(e)= M(e)$ *sonst*

Demnach sind in einem MG Quellen v ($z^{-1}(v)= \{\}$) stets feuerbereit.

Anm.: Ist G streng zusammenhängend, dann hat G weder Quellen noch Senken.

Markierte Graphen -- Eigenschaften

Wir hatten schon erwähnt, dass man zwar Abstriche bei der Modellierungsstärke machen muss, dafür aber Lebendigkeit und Sicherheit effizient entscheiden kann:

Lemma -- MG1

Für einen markierten Graphen $G=(V,E,M_0)$ gilt für jeden gerichteten Zyklus $C = e_1,\ldots,e_k$ stets folgende Invariante:

$$\forall M \in [M_0\rangle : \sum_{e \in C} M(e) = \sum_{e \in C} M_0(e)$$

Beweis:
Wir nehmen an, dass für ein $M \in [M_0>$ die Gleichung gilt, und ein Knoten v feuert, d.h. $M[v >M'$
Betrachte einen beliebigen gerichteten Zyklus C.
- $v \notin C$, so bleibt die obige Gleichung ohnehin für M' erhalten
- $v \in C$, dann ist $q(e_i) = v = z(e_{i-1})$ für ein i, und es gilt, da v feuert:
  $M'(e_i)+M'(e_{i-1}) = M(e_i)+1+M(e_{i-1})-1 = M(e_i)+M(e_{i-1})$

da sich die anderen Kantenmarkierungen auf C nicht ändern, bleibt also die Gleichung erhalten.

Markierte Graphen -- Eigenschaft MG2

Lemma -- MG2

Ein markierter Graph $G=(V,E,M_0)$ ist genau dann lebendig, wenn für die Markensumme auf jedem gerichteten Zyklus C gilt:

$$\forall C \text{ gerichteter Zyklus}: \sum_{e \in C} M_0(e) > 0$$

Beweis:
$\Rightarrow$ Gäbe es einen Zyklus C, auf dem die Markensumme unter M_0 *0* ist, so könnte niemals ein Knoten v auf C schaltbereit werden, weil mit MG1 C in jeder von M_0 erreichbaren Markierung Markensumme *0* hat.

Markierte Graphen -- Eigenschaft MG2 ff

Beweis -- ff:
$\Leftarrow$ Sei M eine beliebige, von M_0 erreichbare Markierung.

Breche alle Kanten e heraus, mit $M(e) > 0$.
Dann ist der Restgraph G' azyklisch, da für jeden Zyklus unter M_0 die Markensumme > *0* war, und sich die Markensumme mit MG1 nicht ändert.

Alle Quellen von G' sind Quellen in G oder Ziele von herausgenommenen Kanten e.
Damit sind also alle Quellen von G' schaltbereit in G.
Sei nun v ein beliebiger Knoten.
Da G' azyklisch ist kann man nun alle Knoten in topologischer Reihenfolge feuern lassen, bis irgendwann auch v feuerbereit wird.

Also ist G lebendig.

Markierte Graphen -- Eigenschaft MG3

Lemma -- MG3

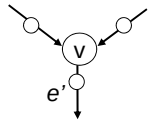
Ein lebendiger markierter Graph $G=(V,E,M_0)$ ist 1-beschränkt (sicher) genau dann, wenn es zu jeder Kante einen gerichteten Zyklus mit Markensumme 1 gibt, oder formal

$$\forall e \in E \; \exists C \text{ gerichteter Zyklus}, e \in C: \sum_{e' \in C} M_0(e') = 1$$

Beweis:
$\Rightarrow$ Gäbe es eine Kante e, für die die Markensumme unter M_0 auf jedem Zyklus auf dem e liegt > *1* ist, so gilt dies auch für eine Folgemarkierung M unter der e eine Marke trägt.
Breche alle Kanten e' mit $M(e') > 0$ heraus, mit Ausnahme von e selbst.
Dann ist der Restgraph G' wieder azyklisch, da G lebendig ist, und alle Quellen von G' sind schaltbereit in G.
Schalte die Knoten in topologischer Reihenfolge, bis $q(e)$ feuert. Danach liegen auf e **2 Marken**. ↯

Markierte Graphen -- Eigenschaft MG3

Beweis -- ff:
$\Leftarrow$
Annahme: in einer von M_0 aus erreichbaren Markierung M wird ein Knoten v, auf dessen ausgehender Kante e' eine Marke liegt, feuerbereit.



Dann ist für jeden Zyklus, auf dem e' liegt offenbar

$$\forall C \text{ gerichteter Zyklus}, e' \in C: \sum_{e \in C} M(e) > 1$$

Da mit MG1 aber die Markensumme auf Zyklen invariant unter dem Feuern von Knoten ist, gilt, da M von M_0 erreichbar ist

$$\exists e' \in E: \forall C \text{ gerichteter Zyklus}, e' \in C: \sum_{e \in C} M_0(e) > 1$$

Widerspruch. ↯

Markierte Graphen -- Eigenschaft MG4

Lemma -- MG4

Für jeden streng zusammenhängenden, markierten Graphen $G=(V,E)$ gibt es eine Anfangsmarkierung M_0 unter der er lebendig und sicher ist.

Beweis: Wir konstruieren M_0:

1. Markiere jede Kante.

   Dann hat jeder Zyklus eine Markensumme > 0, und damit ist *G* lebendig. (wg MG2)

2. Wir benutzen nun iterativ das Argument aus dem letzten Beweis von MG3, um *G* sicher zu machen:

   Solange es eine Kante *e* gibt, für die jeder Zyklus eine Markenzahl >1 hat:

   - Lasse *G* feuern bis *e* mindestens 2 Marken hat.
   - Nehme alle Marken bis auf eine von *e* weg.
     Damit reduziert sich die Markenzahl auf allen Zyklen, auf denen *e* liegt, auf allen anderen Zyklen nicht. *G* bleibt lebendig.

   Iteriere dieses Spiel so lange, bis *G* sicher ist.

"Synchrone" Datenflussgraphen

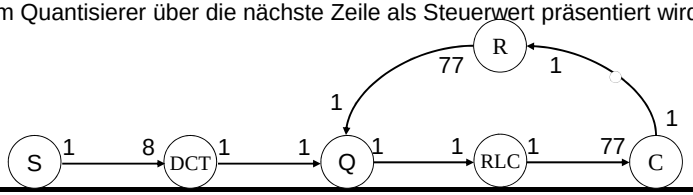
... bei Datenströmen, insbesondere solchen, wie sie in der digitalen Signalverarbeitung vorkommen, hat man häufig Operatoren, die mit unterschiedlichen Raten ausgeführt werden.

Beispiel: Bildkompression

Ein von einer Quelle *S* kommender Strom von Bildpixeln wird zunächst blockweise einer diskreten Cosinustransformation (*DCT*) pro 8 Pixel unterzogen, die transformierten Werte laufen dann in einen Quantisierer (*Q*), der für jede neue Bildzeile je 616 Pixel = 616/8 Blöcke = 77 Blöcke einen neuen Steuerwert erhält.

Die Ausgabe des Quantisierers läuft in einen Kodierer (*RLC*), dessen Ausgabe von einem Controller (*C*) über eine Bildzeile beobachtet wird.

Der Controller erzeugt daraus einen neuen Steuerwert, der über einen Replikator (*R*) dem Quantisierer über die nächste Zeile als Steuerwert präsentiert wird.



Synchrone Datenflussgraphen

Wir können auch hier eine formale Definition zunächst aus Petri-Netzen ableiten

Definition

Ein **synchroner Datenflussgraph (SDFG)** ist ein Petri-Netz $G=(P,T,F,K,W,M_0)$ mit folgenden Eigenschaften:

- $\forall p \in P: |\bullet p| = |p\bullet| = 1$
- $\forall p \in P: K(p)=\infty$,

Wir nehmen ferner an, dass die Marken (Daten) jeder Stelle *p* in der Reihenfolge entnommen werden, in der sie dort entstehen.
(FIFO first in first out)

Wir haben lediglich gegenüber markierten Graphen die Bedingung

$W(t,p) = 1 = W(p,t)$ für alle $p \in P, t \in T$

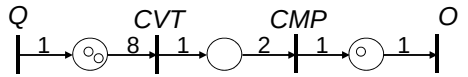
weggelassen, d.h. die Kanten erhalten echte Gewichte.

Synchrone Datenflussgraphen -- Beispiel

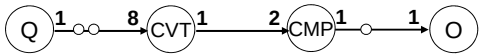
Da wieder stets $|\bullet p| = |p\bullet| = 1$ können wir SDFG's ebenfalls wieder als Graphen auffassen, bei denen die Knoten Transitionen, und die Kanten den Plätzen entsprechen. Allerdings müssen die Quellen und Ziele der Kanten nun mit den Gewichten markiert werden:

Beispiel:
Verarbeite einen Bitstrom einer Quelle Q wie folgt: Fasse je 8 Bit zu einem Character zusammen (CVT) schicke diese Character in einen Vergleicher (CMP) und gebe den Vergleich aus (O).

Als Petri-Netz



In SDFG Notation



SDFG -- Definition

Definition -- SDFG als Graph

Ein **SDFG** G ist ein 5 Tupel $G=(V,E,cons,prod,d)$ mit

- V ist die Menge der Knoten. -- Knoten stehen für Operationen.
- $E \subseteq V \times V$ ist die Menge der Kanten. -- Auf Kanten können mehrere Daten liegen. Es wird die FIFO-Semantik auf den Kanten angenommen.
- $cons : E \rightarrow \mathrm{N}$ gibt die Anzahl der beim Feuern des Zielknotens konsumierten Marken an. -- Sie stehen am Pfeilende.
- $prod : E \rightarrow \mathrm{N}$ gibt die Anzahl der beim Feuern des Quellknotens produzierten Marken an. -- Sie stehen am Pfeilanfang.
- $d : E \rightarrow \mathrm{N}_0$ gibt die Anfangsmarkierung der Kanten an, d.h. die Anzahl der anfangs auf jeder Kante verfügbaren Daten. Markierungen d werden im folgenden als Vektoren aus $N_0^{|E|}$ aufgefasst.

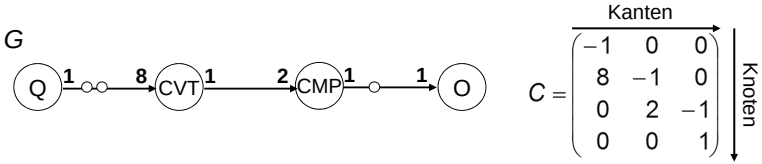
Topologiematrix eines SDF-Graphen

Definition

Sei $G=(V,E,cons,prod,d)$ ein SDFG. Die zu G gehörige **Topologiematrix** ist eine Matrix C aus $Z^{|V|\times|E|}$, deren Zeilen den Knoten und deren Spalten den Kanten von G entsprechen.

Verläuft die Kante e_j vom Knoten v_i zum Knoten v_k, so gilt

$C[i,j] := -prod(v_i,v_k)$, $C[k,j] := cons(v_i,v_k)$ und $C[l,j]=0$ für $l \notin \{i,k\}$.



Anwendung bei der Berechnung von Folgemarkierungen

Ist d_1 die momentane Kantenmarkierung und feuert Knoten v_i jeweils y_i - mal, so erhält man als resultierende Knotenmarkierung

$$d_2=d_1 - C^T \cdot y$$

Konsistenz eines SDFG

Bei SDFGs ist folgende Frage interessant:

- Kann man das Verhalten mit beschränkter Puffergröße realisieren?

Dies kann man über folgende, äquivalente Frage beantworten:

- Gibt es eine endliche Folge von Knotenfeuerungen, mit der man die Startmarkierung wieder erreicht

Falls ja, dann ist ein periodischer Ablauf planbar, bei dem die Pufferbelegung endlich bleibt, da die Folge endlich ist. Interessant ist dann aber noch

- Finde eine Folge mit minimaler Zahl von Knotenfeuerungen.

Definition

Ein zusammenhängender SDF-Graph G mit Topologiematrix C aus $Z^{|V|\times|E|}$ heißt **konsistent**, wenn die Startmarkierung in endlich vielen Schritten wieder angenommen werden kann, d.h. es ein periodisches Verhalten von G gibt.

Konsistenz eines SDFG -- ff

Satz

Ist G zusammenhängend und konsistent, dann gilt

$$Rang(C) = |V| - 1$$

Beweis

Für eine Folge von Feuerungen, die wieder zur Startmarkierung d führt, muss der Vektor y, mit v_i feuert y_i mal, folgende lineare Gleichung erfüllen:

$$d = d - C^T y$$

Da ferner $y \neq 0$ sein muss, muss also $C^T y = 0$ eine nichttriviale Lösung haben.

Also muss $Rang(C) < |V|$ sein.

Andererseits gilt für eine Topologiematrix C, falls G zusammenhängend ist, stets $Rang(C) \geq |V| - 1$. Also gilt " = "

Einschub: Rang der Topologiematrix

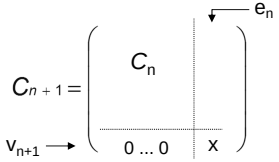
Lemma

Für jeden Baum *G* gilt *Rang(C) =|V|-1*

Beweis -- durch Induktion nach der Anzahl *n* der Knoten

n=2: *G* besteht aus einer Kante, so dass offensichtlich *Rang(C)=1* gilt.

n→*n*+1: C_{n+1} besteht aus *n+1* Zeilen und *n* Spalten



Der Eintrag *x* ist verschieden 0, da die Kante e_n inzident zu dem neuen Knoten v_{n+1} ist.

→ Rang(C_{n+1})=Rang(C_n)+1=n-1+1=n

Da bei einem zusammenhängenden Graphen *G* ein aufspannender Baum stets |*V*| Knoten hat, hat schon die Untermatrix des aufspannenden Baumes Rang |*V*|-*1.*

Minimaler Repetitionsvektor

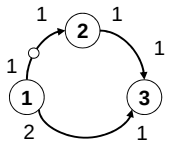
Definition

Sei *G* ein zusammenhängender, konsistenter SDFG. Dann heißt der kleinste positive Vektor $y \neq 0$ im Nullraum von C^T, d.h.

$$y \in \{ x \in N^{|V|};\ C^T x = 0 \}\ \ \Sigma_i y_i \text{ minimal}$$

minimaler Repetitionsvektor (von G).

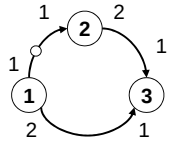
Beispiele:



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

→ *Rang(C) = 3*, also inkonsistent

-- damit 3 feuern kann, muss 1,2 feuern danach liegt aber eine Marke mehr auf (1,3)



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

→ *Rang(C) = 2*, also konsistent

$y = (1,1,2)^T$ ist minimaler Repetitionsvektor

Verklemmungen in SDF-Graphen

Problem: Mit der Existenz eines minimalen Repetitionsvektors *y* ist noch nicht sichergestellt, dass es zu diesem Vektor auch eine konkrete Folge schaltbereiter Knoten gibt, so dass jedes v_i genau y_i -mal feuert. Das Netz könnte verklemmen.

Einfache Überlegung

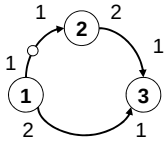
Erreicht man es in irgendeiner Reihenfolge jeden Knoten v_i genau y_i -mal zu feuern, so ist man wieder bei der Anfangsbelegung *d* und hat einen Ablaufplan konstruiert.

Algorithmus -- Konstruktion durch Backtracking

1. Beginne mit der leeren Folge, lege alle feuerbereiten Knoten, die noch nicht y_i-mal gefeuert haben auf einen Stack.
2. Nimm einen beliebigen Knoten v_i vom Stack herunter und lasse ihn feuern. Füge ihn zur Folge hinzu. Berechne wieder alle feuerbereiten Knoten, die noch nicht y_i-mal gefeuert haben, und lege sie auf den Stack.
3. Gibt es keine solchen Knoten mehr, so überprüfe, ob jeder Knoten v_i y_i-mal gefeuert hat.
   - Falls ja: so ergibt die aktuelle Folge einen periodischen Ablaufplan
   - Falls nein: lösche den zuletzt gefeuerten Knoten aus der Folge, mache sein Feuern rückgängig und gehe nach 2.

Beispiel

Wir betrachten unser Beispiel von zuletzt:



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

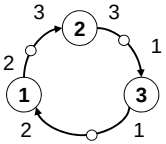
→ *Rang(C) = 2*, minimaler Repetitionsvektor $y = (1,1,2)^T$

1. $d = (1,0,0)^T$ Status *(1,1,2)* feuerbereit: 1,2 Folge: ()
2. $d = (2,2,0)^T$ Status *(0,1,2)* feuerbereit: 2 Folge: (1)
3. $d = (1,2,2)^T$ Status *(0,0,2)* feuerbereit: 3 Folge: (1 2)
4. $d = (1,1,1)^T$ Status *(0,0,1)* feuerbereit: 3 Folge: (1 2 3)
5. $d = (1,0,0)^T$ Status *(0,0,0)* feuerbereit: Folge: (1 2 3 3)

In diesem einfachen Fall erhalten wir direkt einen gültigen periodischen Ablaufplan 1233. Hätten wir im 1. Schritt zunächst 2 gewählt, würde dies ebenfalls zu einem korrekten Ablauf 2133 führen, allerdings hätten wir auf der Kante (1,2) nur Platz 1 statt 2 benötigt(!). Im allgemeinen aber hat das Verfahren keine polynomielle Laufzeit.

noch‘n Beispiel

Wir betrachten ein harmlos aussehendes Beispiel:



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

$\rightarrow$ *Rang(C) = 2*, minimaler Repetitionsvektor $y = (3,2,6)^T$

1. $d = (1,1,1)^T$ Status *(3,2,6)* feuerbereit: 3 Folge: ()
2. $d = (1,0,2)^T$ Status *(3,2,5)* feuerbereit: 1 Folge: (3)
3. $d = (3,0,0)^T$ Status *(2,2,5)* feuerbereit: 2 Folge: (3 1)
4. $d = (0,3,0)^T$ Status *(2,1,5)* feuerbereit: 3 Folge: (3 1 2)
5. $d = (0,2,1)^T$ Status *(2,1,4)* feuerbereit: 3 Folge: (3 1 2 3)
6. $d = (0,1,2)^T$ Status *(2,1,3)* feuerbereit: 1,3 Folge: (3 1 2 3 3)
7. $d = (2,1,0)^T$ Status *(1,1,3)* feuerbereit: 3 Folge: (3 1 2 3 3 1)
8. $d = (2,0,1)^T$ Status *(1,1,2)* feuerbereit: Folge: (3 1 2 3 3 1 3)

Verklemmung!
Die Verklemmung lässt sich auch nicht vermeiden, wenn man in 6. 3 vor 1 feuert. Dann kann nur noch 1 feuern und das System steht.

SDFGs und markierte Graphen

SDFGs sind nicht mächtiger als markierte Graphen, sie sind lediglich eine kompaktere Darstellung für MG‘s. Man kann einen konsistenten SDFG mit minimalem Repetitionsvektor *y*, wie folgt in einen MG, entfalten:

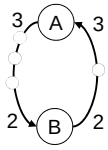
- Zu jedem Knoten *v* betrachte y_v Knoten v^δ, $0 \le \delta < y_v$
- Zu jeder Kante $e = (u,v)$ betrachte $y_u \cdot prod(u,v)$ Kanten
  e^ε, $0 \le \varepsilon < y_u \cdot prod(u,v)$

  mit $q(e^\varepsilon) := u^\delta$ wo $\delta := \varepsilon \, div \, (prod(u,v))$

  $z(e^\varepsilon) := v^\gamma$ wo $\gamma := ((d_e + \varepsilon) \bmod (y_v \cdot cons(u,v))) \, div \, cons(u,v)$

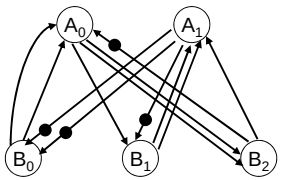
  -- die Kanten laufen in prod(u,v) breiten Paketen aus den Quellexemplaren in cons(u,v) breiten Paketen in die Zielexemplare, wobei die d(u,v) letzten Kanten in die ersten Zielexemplare laufen, da diese zuerst feuerbereit sind.
- Die d_e Marken liegen auf den Kanten e^ω mit
  $y_u \cdot prod(u,v) - 1 \ge \omega \ge y_u \cdot prod(u,v) - d_e$

SDFGs und markierte Graphen -- Beispiel



$$C = \begin{pmatrix} -3 & 3 \\ 2 & -2 \end{pmatrix}$$

Rang(C)=1 und *y=(2,3)*

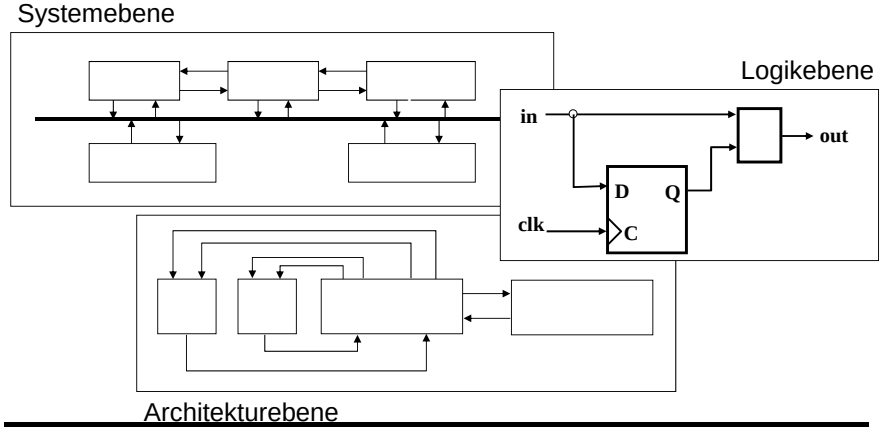


v_{i-1} ist genau dann feuerbereit, wenn *v* zum *i*.-mal feuerbereit ist

Die Entfaltung ist nicht polynomiell beschränkt!

1.2.5 Strukturorientierte Modelle

Neben den gerade beschriebenen verhaltensorientierten Modellen gibt es die strukturorientierten Modelle, die ein System als eine Menge von Komponenten und deren Verbindungen beschreiben.



Strukturorientierte Modelle -- ff

Man kann solch strukturorientierten Modellen in vielerlei Hinsicht Verhalten zuordnen. Die Grundlage bilden

Komponenten mit ihren **Anschlüssen** (Ports)

Netze, als Mengen von Ports

Interpretation

Ports *p* → Belegungen *v(p)*

- Spannungsverläufe $v(p) \in \{f: \mathbf{R} \rightarrow \mathbf{R}\}$
- Zeitdiskrete Waveforms $v(p) \in \{f: \mathbf{Z} \rightarrow M\}$, *M* Bitvektoren, Zahlen …

Komponenten → Menge aller Portbelegungen

- explizit, durch Spezifikation
- implizit, durch Strukurbeschreibung → Hierarchie

Netze → Resolution der beteiligten Ports

- Angabe einer Resolutionsfunktion
- einfache Propagation (orientierte Netze)

1.2.6 Heterogene Modelle

Im allgemeinen werden heterogene Modelle benutzt, da die reinen struktur- bzw. verhaltensorientierten Modelle zwar gut analysierbar, aber in ihrer Modellierungskraft recht eingeschränkt sind.

Will man auf die Ausdruckskraft von Programmiersprachen hinaus, so benutzt man Strukturen, die direkt aus einfachem Programmcode ableitbar sind.

Kontroll-Datenflussgraphen (**CDFG**s)

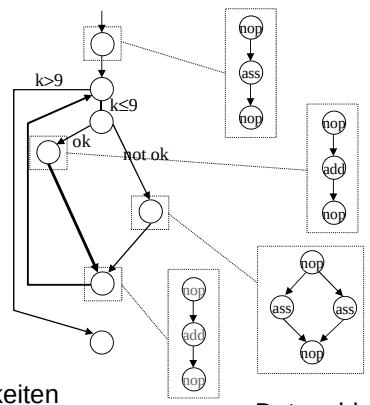
Idee: kombiniere Datenfluss mit dem Kontrollfluss von Programmen

Kontroll-Datenflussgraphen: ein Beispiel

Kontrollflussgraph + Datenflussgraphen

VHDL-Programm

```
(1)  s:=k;
(2)  LOOP
(3)    EXIT WHEN k>9;
(4)    IF (ok=TRUE)
(5)      j:=j+1;
(6)    ELSE
(7)      j:=0;
(8)      ok:=TRUE;
(9)    END IF;
(10)  k:=k+1;
(11) END LOOP;
(12)
```



Kontrollabhängigkeiten

Datenabhängigkeiten

Sequenzgraphen

Um hierarchische, heterogene Beschreibungen zu erlauben, führt man den Begriff des Sequenzgraphen ein:

Definition

Ein Sequenzgraph ist ein azyklischer gerichteter Graph *G=(V,E)* mit zwei ausgezeichneten Knoten, einem Start- und einem Endknoten. Zusätzlich besitzt *G* die folgenden Eigenschaften

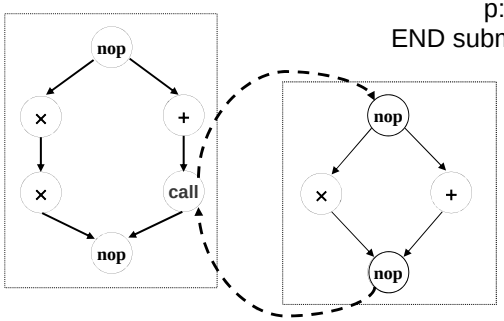
- Die vom Start- und Endknoten verschiedenen Knoten lassen sich in die Klassen
  - Operationsknoten (sog. Tasks)
  - Hierarchieknoten

  aufteilen
- Hierarchieknoten sind Zeiger auf andere Sequenzgraphen. Man unterscheidet hierbei zwischen dem Modulaufruf (CALL), der Verzweigung (BR) und der Iteration (LOOP)

Sequenzgraph: Beispiel eines CALL

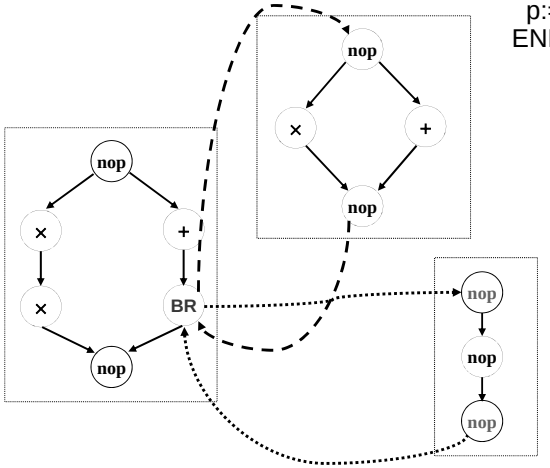
```
x:=a*b; y:=x*c;
z:=a+b; submodule(a,z);

PROCEDURE submodule (m,n)
IS
        p:=m+n; q:=m*n;
END submodul
```



Sequenzgraph: Beispiel eines BRANCH

```
x:=a*b; y:=x*c; z:=a+b;
IF z>0 THEN
  p:=m+n; q:=m*n;
END IF;
```



Sequenzgraph: Beispiel eines LOOP