Prozesse und Nebenläufigkeit (PNL)

© Prof. Dr. Rüdiger Valk

Prüfungsunterlagen zur Grundlagenveranstaltung PNL
- gehalten im Wintersemester 2002/2003 -

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Vorlesung baut auf dem Grundstudiumszyklus *Formale Grundlagen der Informatik*, insbesondere *F4: Parallelität und Nebenläufigkeit* auf. Die dort behandelten Themen hatten eher das Ziel, Grundphänomene verständlich zu machen. Die Grundlagenveranstaltung *PNL: Prozesse und Nebenläufigkeit* soll dagegen verstärkt den Aspekt der formalen Methoden berücksichtigen.

Im Vordergrund stehen dabei die Behandlung verschiedener Modelle zur Modellierung verteilter und paralleler Systeme, wobei die Betonung auf *verschiedene* liegt, sowie Methoden der Analyse und Verifikation. In den ersten drei Kapitel werden dementsprechend drei verschiedene solche Modelle behandelt: Prozessalgebra, Harel-Graphen (Statecharts) und Petrinetze. In der Vorlesung F4 wurde eine CSP-artige Sprache zur Einführung behandelt, die aber schon einige Eigenschaften von Prozessalgebra aufwies, aber eben eine ausführbare (abstrakte) Sprache war. Prozessalgebra dient zur Spezifikation, Beschreibung und Verifikation und ist weiter entfernt von realen Sprachkonstrukten. Der algebraische Ansatz erlaubt dafür eine besonders klare und elegante mathematische Fassung semantischer Aspekte. So werden sowohl eine operationale Semantik, wie auch ein Kalkül zur Verhaltensähnlichkeit (Bisimulation) eingeführt. Die wichtigsten theoretischen Ergebnisse beziehen sich auf die Korrektheit und Vollständigkeit dieser Kalküle. Das Thema wird so weit ausgeführt, dass ein einigermaßen komplexes System (das Alternierbit-Protokoll) voll verifiziert werden kann (was auch vorgeführt wird).

Obwohl wie die Prozessalgebra aus der Theorie endlicher Automaten entstanden, wird mit Harel-Graphen (Statecharts) ein deutlich anderer Weg zur Spezifikation verteilter Systeme begangen. Harel geht von einen wesentlichen Unterschied zwischen zwei Arten von Systemen aus: der Unterscheidung von *transformierenden* und den *reaktiven* Systemen.

Unter einem transformierenden System verstehen Harel und Pnueli ein System, welches Eingaben akzeptiert, daraufhin Transformationen durchführt und schließlich Ausgaben produziert. Diese Definition schließt auch solche Systeme ein, die während ihres Einsatzes nach weiteren Eingaben verlangen oder zusätzliche Ausgaben produzieren; entscheidend ist der Umstand, daß ein transformierendes System eine Ein-/Ausgabe-Operation durchführt. Ein reaktives System wird hingegen wiederholt durch seine Umgebung zu Aktivitäten veranlaßt und reagiert ständig auf externe Ereignisse, ist also ereignisgesteuert. Es berechnet i. A. keine Funktion

und führt auch keine solche aus, sondern erhält in gewisser Hinsicht eine bestimmte Beziehung zu seiner Umgebung aufrecht. Beispiele für reaktive Systeme lassen sich leicht und zahlreich finden; so sind neben Flugelektronik-Systemen usw. auch Armbanduhren, Mikrowellengeräte, viele moderne medizinische Geräte, Telekommunikationsanlagen, Betriebssysteme sowie die Mensch-Maschine-Schnittstellen zahlreicher Softwareprodukte reaktive Systeme. Statecharts haben auch Bedeutung für die Softwaretechnik im Rahmen der objektorientierten Analyse gewonnen.

Petrinetze wurden schon in der Vorlesung F4 eingeführt. Für einfache Netze werden Vergröberungen und Verfeinerungen definiert und mit dem Begriff des Netzmorphismus in einen algebraischen Kontext gestellt. Zur Modellierung komplexer Systeme eignen sich besonders höhere Petrinetze, wie z.B. gefärbte Netze, die in F4 nicht über ihre Definition hinaus behandelt wurden. An Beispielen wie dem Datenbank-Manager-System wird ihre Semantik und Analyse erläutert. Dabei werden die in F4 behandelten Platz-Invarianten entsprechend verallgemeinert.

Näher an Programmiersprachen und objektorientierte Konzepte rücken Petrinetze durch Referenznetze und Objekt-Petrinetze. Petrinetze treten hier als eigenständige Objekte in größeren Umgebungen auf, die von Klassendefinitionen beliebig oft instanziiert und über synchrone Kanäle miteinander kommunizieren können. Die Strukturierung komplexer Systeme wird besonders durch eine Hierarchie-Bildung gefördert, was dadurch erreicht wird, dass Netz-Objekte als Marken in Umgebungsnetzen auftreten können. Dies wird an einem Beipiel aus dem Gebiet Workflow und dem Garbage Can System erläutert. Letzteres zeigt die Modellierung einer großen Anzahl von dynamisch erzeugten und verschwindenden Objekten, die interagierend ein Grundsystem durchlaufen.

Die Informatik kennt eine unüberschaubare Menge von Prozessbegriffen, von denen hier einige Grundformen vorgestellt werden. Für kommunizierende sequentielle Prozesse werden Verfahren zur Einführung einer globalen Sicht vorgestellt, wie *logische* und *vektorielle Zeitstempel*. Prozesse von Petrinetzen sind allgemeiner. Sie reflektieren die kausale Struktur von Handlungen und erhalten deren Nebenläufigkeit. Dies ist im Gegensatz zu Schaltfolgen zu sehen, die häufig als Prozesse in "Interleaving-Semantik" bezeichnet werden.

Die Analyse komplexer Systeme hat heute größere Bedeutung denn je, da immer mehr sicherheitssensible Anwendungen entwickelt werden. Generell bedeutet Verifikation die Überprüfung, ob das Verhalten eines Systems seiner Spezifikation entspricht. Dabei können jedoch häufig das Verhalten und/oder die Spezifikation nicht genau dargestellt werden. Mit Hilfe der Prozessalgebra wurde ein formaler Ansatz einer solchen Verifikation behandelt. Andere Ansätze wählen das *Model Checking*, das die Untersuchung aller möglichen Zustände des Systems beinhaltet. Dabei sind die zu untersuchenden Zustandsmenge häufig so groß, dass spezielle Methoden der Reduktion, Datenhaltung oder Suche gefunden werden müssen. Zu spezifizierende Eigenschaften werden häufig in temporaler Logik formuliert, für die spezielle Model-Check-Werkzeuge entwickelt wurden. Ein anderer Ansatz ist die *Strukturelle Analyse*. Hier wird das im Verhältnis zu seinem Zustandsraum viel kleinere System selbst in seiner Struktur untersucht. Dieser Ansatz wurde für Petrinetze (P/T-Netze) am erfolgreichsten untersucht. Die Methoden sind zum Teil auch auf höhere Netze und andere Modelle übertragbar.

Zuverlässige Systeme erforden Redundanz zur Verdeckung von Fehlern und Ausfällen. Am Beispiel des *byzantinischen Konsens* werden für synchrone Netzwerke mit vollständigen Netzwerkgraphen verschiedene solche Verfahren vorgestellt. Die Übertragbarkeit der Ergebnisse auf beliebige Netzwerktopologien und auf asynchrone Netzwerke wird diskutiert.

Probabilistische Algorithmen benutzen Zufallswerte. Es wird gezeigt, wie unlösbare Probleme lösbar werden oder lösbare Probleme effizienter behandelt werden können. Dabei ist die "total sichere" Korrektheit durch eine Korrektheit mit hoher Wahrscheinlichkeit zu ersetzen. Solche Verfahren werden für Konsens, Auswahl und wechselseiten Ausschluss behandelt.

Implementierte Basissysteme enthalten meist nur elementare Kommunikationsprimitive. Durch einfache Verfahren kann zum Beispiel "broadcast" durch Punkt-zu-Punkt-Kommunikation simuliert werden. Anspruchsvoller ist die Simulation von Broadcast, der eine totale kausale Ordnung der Nachrichten in allen Knoten des Netzwerkes erzeugt. Solche Verfahren können dann zur konsistenten Datenbehandlung in verteilten Systemen eingesetzt werden.

Kapitel 2

Prozessalgebra

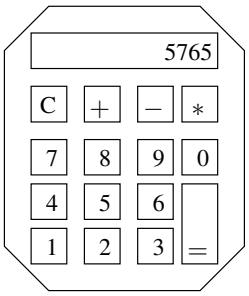
2.1 Einleitung

Die Prozessalgebra wurde aus der Automatentheorie entwickelt, um nebenläufige und reaktive Prozesse und Systeme beschreiben, modellieren und verifizieren zu können. Dabei wurden wie bei endlichen Automaten Zustände festgelegt und für Aktionen oder Folgen von Aktionen spezifiziert, welches der Nachfolgezustand ist. Die *elementare Prozessalgebra* entspricht der Modellierung eines einzigen Automaten. Im Unterschied zur traditionellen Automatentheorie wird aber eine algebraische Behandlung und der Aspekt der Beobachtbarkeit und Verhaltensäqivalenz reaktiver Systeme betont. Nebenläufige und kommunizierende Systeme werden durch mehrere Automaten beschrieben, die mittels Rendezvous-Synchronisation gekoppelt werden.

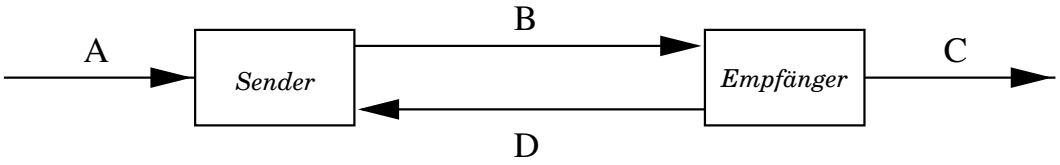
Die Darstellung in diesem Kapitel stützt sich in erster Linie auf [**?**] und teilweise auf [**?**]. Weitere einschlägige Literaturquellen sind: [**?**], [**?**], [**?**] und [**?**].

Beispiele

Taschenrechner:



Das Alternierbitprotokoll:



Beschreibung: siehe Skript F4.

2.2 Prozess-Graphen und elementare Prozessterme

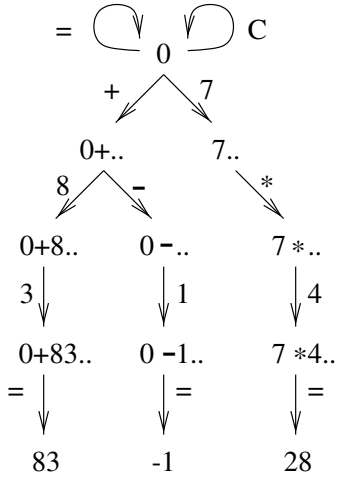
Ein *Transitionssystem* ist eine Menge von Transitionen, zusammen mit einem ausgezeichneten Zustand (*Wurzelzustand, Anfangszustand*) .

Definition 2.1 *Sei S eine Menge von Zuständen mit $\surd \in S$ und A eine Menge von (atomaren) Aktionen. $TS = (S, A, tr, s_0)$ heißt* Transitionssystem*, wobei $tr \subseteq S \times A \times S$* Transitionsrelation *und $s_0 \in S$* Anfangszustand *heißt.*

- Eine *Transition* $(s, a, s') \in tr$ (geschrieben : $s \xrightarrow{a} s'$) drückt aus, dass der Zustand s durch die Aktion a in den Zustand s' wechseln kann.
- Eine *Transition* $(s, a, \surd) \in tr$ (geschrieben : $s \xrightarrow{a} \surd$) drückt aus, dass der Zustand s durch die Aktion a ordnungsgemäß terminieren kann.

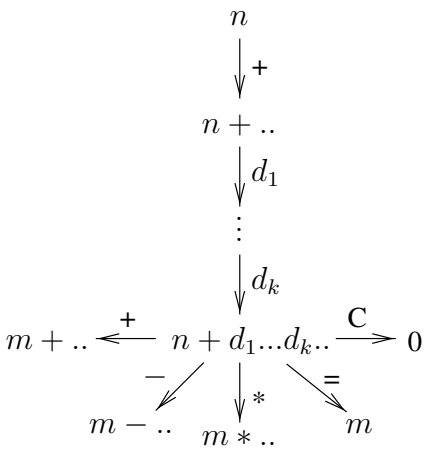
Beispiel:

Ausschnitt des Transitionssystems des Taschenrechner(beispiel)s:



Beispiel – (Fortsetzung)

Verhalten einer Komponente: Plus-Knopf



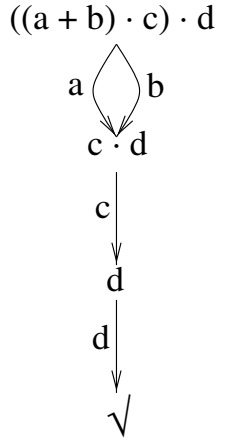
Definition 2.2 *(elementare Prozess-Terme, basic process terms)*

Die Menge der elementaren Prozess-Terme *bzw.* Prozess-Ausdrücke BPA *wird aus* atomaren Aktionen $a \in A$ *sowie der und* Hintereinanderausführung *gebildet:*

- *Jedes* $a \in A$ *ist ein BPA (*atomare Aktion*).*
- *Für alle* $t_1, t_2 \in BPA$ *ist* $t_1 + t_2 \in BPA$ *(*Auswahl*: Es wird entweder* t_1 *oder* t_2 *ausgeführt).*
- *Für alle* $t_1, t_2 \in BPA$ *ist* $t_1 \cdot t_2 \in BPA$ *(*Hintereinanderausführung, Sequenz*: Nach der ordnungsgemäßen Termination von* t_1 *wird* t_2 *ausgeführt).*
- *Nur nach diesen Regeln gebildete Terme liegen in* BPA*.*

Beispiel:

$((a + b) \cdot c) \cdot d \in BPA$ repräsentiert das Verhalten des folgenden Transitionssystems:



Als *Prozessgraphen* bezeichnen wir ein Transitionssystem, dessen Zustände Prozessterme sind. Ein solcher Prozessterm repräsentiert das Verhalten, das das Transitionssystem hätte, falls dieser Zustand (Prozessterm) der Anfangszustand wäre. Die Zustände eines Prozessgraphen beschreiben also das noch bevorstehende (Rest-)Verhalten, nachdem diesen Zustand erreicht wurde. Ein Prozessgraph wird formal über folgendes Kalkül definiert, wobei (A) das Axiom ist. Danach werden die Schlussregel angegeben. Durch das Transitionssystem wird eine operationale Semantik der Prozessalgebra definiert.

Definition 2.3 *(Prozessgraph)*

Für Prozessterme wird durch folgendes Axiom A_0 und nachfolgende Regeln ein Transitionssystem definiert, dessen Zustände Prozessterme sind. Anfangszustand ist der Prozessterm ohne Vorgänger. Dieses Transitionssystem heißt Prozessgraph *des Prozessterms, der den Anfangszustand darstellt.*

Für $v \in A, x, y, x'y' \in BPA$ sei:

$$\frac{}{v \xrightarrow{v} \surd} \qquad (A_0)$$

$$\frac{x \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} x'}{x + y \xrightarrow{v} x'}$$

$$\frac{y \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd} \qquad \frac{y \xrightarrow{v} y'}{x + y \xrightarrow{v} y'}$$

$$\frac{x \xrightarrow{v} \surd}{x \cdot y \xrightarrow{v} y} \qquad \frac{x \xrightarrow{v} x'}{x \cdot y \xrightarrow{v} x' \cdot y}$$

Beispiel: Beweis von $((a + b) \cdot c) \cdot d \xrightarrow{b} c \cdot d$ aus den Transitionsregeln:

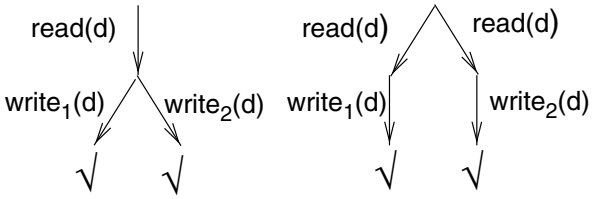
$$\dfrac{\dfrac{\dfrac{b \xrightarrow{b} \surd}{a + b \xrightarrow{b} \surd}}{(a + b) \cdot c \xrightarrow{b} c}}{((a + b) \cdot c) \cdot d \xrightarrow{b} c \cdot d} \qquad \begin{array}{l} \dfrac{}{v \xrightarrow{v} \surd} \\[2ex] \dfrac{y \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd} \\[2ex] \dfrac{x \xrightarrow{v} \surd}{x \cdot y \xrightarrow{v} y} \\[2ex] \dfrac{x \xrightarrow{v} x'}{x \cdot y \xrightarrow{v} x' \cdot y} \end{array}$$

(links: die Ableitung im Kalkül, rechts: die benutzten Regeln)

2.3 Bisimulation und Äquivalenz

Um das Verhalten eines Systems mit dem Verhalten eines anderen Systems oder einer Spezifikation zu vergleichen, wurde der Begriff der wechselseitigen Simulation oder Bisimulation eingeführt.

Beispiel:



Der linke Prozeß liest d. Dann wird entschieden, ob d auf Platte 1 oder Platte 2 geschrieben wird. In dem anderen Prozeß wird die Entscheidung vor dem Lesen getroffen.

Beide Prozesse haben

$$read(d)write_1(d) \text{ und } read(d)write_2(d)$$

als Schaltfolgen und sind daher "schaltfolgenäquivalent" (trace equivalent).

Diese Art der Äquivalenz ist jedoch häufig nicht angemessen, z.b. wenn die Platte 1 ausfällt. Dann würde der erste Prozeß d bei jedem Ablauf auf Platte 2 schreiben - im Gegensatz zum anderen Prozeß, der in eine Verklemmung geraten kann.

Dies ist die Motivation, für eine auf "Bisimulation" beruhende Äquivalenz.

Äquivalenz durch Bisimulation

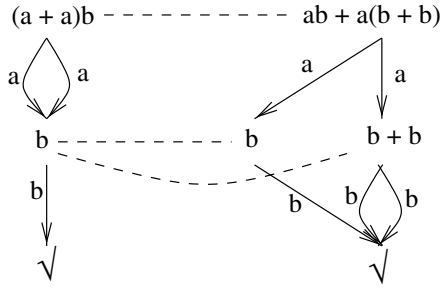
Eine *Bisimulation* ist eine binäre Relation $\mathcal{B}$ auf BPA (d.h. $\mathcal{B} \subseteq BPA \times BPA$) mit folgenden Eigenschaften:

1. falls $p \mathcal{B} q$ und $p \xrightarrow{a} p'$, dann $q \xrightarrow{a} q'$ mit $p' \mathcal{B} q'$
2. falls $p \mathcal{B} q$ und $q \xrightarrow{a} q'$, dann $p \xrightarrow{a} p'$ mit $p' \mathcal{B} q'$
3. falls $p \mathcal{B} q$ und $p \xrightarrow{a} \surd$, dann $q \xrightarrow{a} \surd$
4. falls $p \mathcal{B} q$ und $q \xrightarrow{a} \surd$, dann $p \xrightarrow{a} \surd$

Zwei Prozesse p und q heißen *bisimilar* (Bezeichnung: $p \underline{\leftrightarrow} q$), falls es eine Bisimulation $\mathcal{B}$ mit $p \mathcal{B} q$ gibt.

Lemma 2.4 *Die Bisimulation ist eine Äqivalenzrelation.*

Beispiel: $(a+a)b \leftrightarrow ab + a(b+b)$



Diese Bisimulation $\mathcal{B}$ wird definiert durch:

$(a+a)b \; \mathcal{B} \; ab + a(b+b)$

$b \; \mathcal{B} \; b$

$b \; \mathcal{B} \; b + b.$

Kongruenzäquivalenz

Theorem 2.5 *Die Äquivalenz der Bisimulation ist eine Kongruenz bezgl.* $\{+, \cdot\}$ *auf BPA, d.h.: falls* $s \leftrightarrow s'$ *und* $t \leftrightarrow t'$*, dann auch* $s + t \leftrightarrow s' + t'$ *und* $s \cdot t \leftrightarrow s' \cdot t'$*.*

Die Kongruenzeigenschaft folgt daraus, dass die Transitionsregeln in einer bestimmten Normalform (Panth-Form) sind.

Mit obenstehender Definition ist es aufwendig zu überprüfen, ob zwei elementare Prozessterme bisimilar sind: es müssen zunächst die Prozessgraphen und dann zwischen ihnen die Bisimulationsrelation konstruiert werden. Es wird daher jetzt ein Kalkül für eine Gleichheitsrelation zwischen elementaren Prozesstermen eingeführt, der diese Aufgabe durch die Konstruktion von Ableitungen lösen soll.

Axiome des BPA-Kalküls:

A1	$x + y$	$=$	$y + x$
A2	$(x + y) + z$	$=$	$x + (y + z)$
A3	$x + x$	$=$	x
A4	$(x + y) \cdot z$	$=$	$x \cdot z + y \cdot z$
A5	$(x \cdot y) \cdot z$	$=$	$x \cdot (y \cdot z)$

Schlussregeln des BPA-Kalküls:

- (Substitution)

  Sei σ eine Substitution, die alle Variablen durch Terme aus BPA ersetzt. Ist dann $s = t$ ein Axiom, dann gilt $\sigma(s) = \sigma(t)$.

- (Äquivalenz)

  - $t = t$ für alle $t \in BPA$
  - falls $s = t$, dann $t = s$
  - falls $s = t$ und $t = u$, dann $s = u$

- (Kontext)

  Falls $s = s'$ und $t = t'$, dann $s + t = s' + t'$ und $s \cdot t = s' \cdot t'$.

Um die Relation *bisimilar* mittels des BPA-Kalküls zu berechen, muß natürlich bewiesen werden, dass zwei Terme genau dann bisimilar sind, wenn sie äquivalent sind. Traditionell wird diese Eigenschaft in zwei Teilen als *Korrektheit* und *Vollständigkeit* des BPA-Kalküls formuliert und bewiesen.

Korrektheit (soundness) Das BPA-Kalkül heißt *korrekt (sound)*, wenn für alle Terme $s, t \in BPA$ aus $s = t$ auch $s \underline{\leftrightarrow} t$ folgt.

Vollständigkeit (completeness) Das BPA-Kalkül heißt *vollständig (complete)*, wenn für alle Terme $s, t \in BPA$ aus $s \underline{\leftrightarrow} t$ auch $s = t$ folgt.

Satz 2.6 *Der BPA-Kalkül ist korrekt.*

Beweis:
Es wird hier nur die Beweisstruktur dargestellt. Wie fast immer bei Beweisen für die Korrektheit eines Kalküls ist zu zeigen:

1.) Die Axiome sind korrekt.

2.) Die Regel überführen korrekte Terme in korrekte Terme.

zu 1.): Erfolgt mit 2 a): Substitution

zu 2.):

a) Substitution: Es ist $\sigma(s) \underline{\leftrightarrow} \sigma(t)$ für jedes Axiom $s = t$ zu beweisen, wobei σ eine Substitution ist, die alle Variablen in s und t auf elementare Prozessterme abbildet. Dabei ist auf die Definition der Bisimulation zurückzugreifen. Dies wird hier nicht ausgeführt. Informell steht dahinter in Bezug auf die Axiome A1 ... A5 folgende Argumentation:

   – A1: Die Terme $s + t$ und $t + s$ stellen beide eine Auswahl zwischen s und t dar.

- A2: Die Terme $(s + t) + u$ und $s + (t + u)$ stellen beide eine Auswahl zwischen s, t und u dar.
- A3: Eine Auswahl zwischen t und t ist eine Wahl für t.
- A4: Die Terme $(s + t) \cdot u$ und $s \cdot u + t \cdot u$ stellen beide eine Auswahl zwischen s und t dar, worauf u ausgeführt wird.
- A5: Die Terme $(s \cdot t) \cdot u$ und $s \cdot (t \cdot u)$ stellen beide die Aktion s dar, gefolgt von t und u.

b) Gilt, da die Bisimulations-Relation eine Äquivalenz ist.

c) Gilt, da die Bisimulations-Relation eine Kongruenz ist.

□

Aufgabe 2.7

Motivieren Sie, dass folgendes Distributivgesetz nicht gilt: $x \cdot (y + z) = x \cdot y + x \cdot z$.
Hinweis: Setzen Sie $x = read(d)$ usw. in obigem Beispiel.

Satz 2.8 *Der BPA-Kalkül ist vollständig.*

Beweis:
Zu beweisen ist also, dass aus $s \underline{\leftrightarrow} t$ auch $s = t$ folgt. Zunächst wird der Beweis für die einfachere Relation $=_{\mathrm{AC}}$ bewiesen, d.h. $s \underline{\leftrightarrow} t \Rightarrow s =_{\mathrm{AC}} t$. Daraus wird dann dann die Behauptung des Satzes abgeleitet.

Per definitionem gelte $s =_{\mathrm{AC}} t$, wenn der Ausdruck nur mittels der Axiome $A1$ (Kummutativität) und $A2$ (Assoziativität) abgeleitet werden kann. Jede Äquivalenzklasse bezüglich dieser Relation wird durch einen Ausdruck aus "Summanden" der Form $s_1 + \ldots + s_k$ dargestellt, wobei s_i entweder eine atomare Aktion a ist oder die Form $t_1 \cdot t_2$ hat.

Die übrigen Axime werden in Ersetzungsregeln mit der Richtung "links nach rechts" umgeformt:

$$\begin{array}{rcl} x + y & =_{\mathrm{AC}} & y + x \\ (x + y) + z & =_{\mathrm{AC}} & x + (y + z) \\ x + x & \rightarrow & x \\ (x + y) \cdot z & \rightarrow & x \cdot z + y \cdot z \\ (x \cdot y) \cdot z & \rightarrow & x \cdot (y \cdot z) \end{array}$$

Auf diese Weise erhalten wir ein Ersetzungskalkül, bei dem zwischen den Regelanwendungen Umformungen bzgl. der Relation $=_{\mathrm{AC}}$ angewendet werden können (siehe Beispiel 2.9).

Wendet man die Regeln dieses Ersetzungskalküls immer wieder auf einen elementaren Prozessterm $s \in BPA$ an, so gelangt man nach einer endlichen Zahl von Schritten zu einem elementaren Prozessterm $t \in BPA$, der nicht weiter reduziert werden kann. Allgemein heißt ein

solches t in Ersetzungskalkülen *Normalform*. Das Ersetzungskalkül selbst heißt *terminierend*. Dies wird üblicherweise mit einer *Gewichtsfuktion*

$$gew : BPA \mapsto \mathbb{N}$$

bewiesen, die im vorliegenden Fall folgendermaßen definiert werden kann ($v \in A, s, t \in BPA$).

$$gew(v) := 2$$

$$gew(s + t) := gew(s) + gew(t)$$

$$gew(s \cdot t) := gew(s)^2 \cdot gew(t)$$

Da $gew(s_1) > gew(s_2) > \ldots > gew(s_q) > \ldots$für jede Ableitung $s_1, s_2, \ldots, s_q, \ldots$ gilt, kann es keine unendlichen Ableitungen geben.

Terme in Normalform haben die Struktur $t_1, \ldots, t_k$, wobei jedes t_i eine atomare Aktion $a \in A$ ist oder die Form $a \cdot s (a \in A, s$ in Normalform) hat. Durch Induktion über ihre Länge beweist man für Normalformen n und n' :

$$n \underline{\leftrightarrow} n' \Rightarrow n =_{\mathrm{AC}} n'$$

- Falls n einen Summanden der Form a enthält, dann gilt $n \xrightarrow{a} \surd$ und wegen $n \underline{\leftrightarrow} n'$ auch $n' \xrightarrow{a} \surd$. Also ist a auch in n' als Summand enthalten.
- Falls n einen Summanden der Form $a \cdot s$ enthält, dann gilt $n \xrightarrow{a} s$ und wegen $n \underline{\leftrightarrow} n'$ auch $n' \xrightarrow{a} t$ mit $s \underline{\leftrightarrow} t$. Also ist $a \cdot t$ in n' als Summand enthalten. Da s und t in Normalform, aber kleiner als n und n' sind, folgt durch Induktion $s =_{\mathrm{AC}} t$.

Da somit n und n' dieselben Summanden haben, gilt $n =_{\mathrm{AC}} n'$.

Um nun den Beweis von $s \underline{\leftrightarrow} t \Rightarrow s = t$ zu führen, sei $s \underline{\leftrightarrow} t$ angenommen. s und t können durch das Ersetzungssystem zu Normalformen n und n' reduziert werden. Dies könnte auch durch den Kalkül geschehen, d.h. es gilt: $s = n$ und $t = n'$. Aus der Korrektheit des Kalküls folgt $s \underline{\leftrightarrow} n$ und $t \underline{\leftrightarrow} n'$, also insgesamt $n \underline{\leftrightarrow} n'$. Für solche Normalformen wurde aber oben gezeigt: $n =_{\mathrm{AC}} n'$. Damit ergibt sich insgesamt: $s = n =_{\mathrm{AC}} n' = t$, d.h. $s = t$. □

Anmerkung: Aus dem Beweis ergibt sich ein Verfahren, mit dem man durch ein Kalkül $s \underline{\leftrightarrow} t$ bzw. $s = t$ entscheiden kann. (siehe das folgende Beispiel).

Beispiel 2.9

Zu entscheiden ist: $(a + a)(cd) + (bc)(d + d) = ((b + a)(c + c))d$

$$\begin{array}{ll} & (\underline{a + a})(cd) + (bc)(d + d) \\ \overset{\mathrm{A3}}{\rightarrow} & a(cd) + (bc)(\underline{d + d}) \\ \overset{\mathrm{A3}}{\rightarrow} & a(cd) + \underline{(bc)d} \\ \overset{\mathrm{A5}}{\rightarrow} & a(cd) + b(cd) \end{array}$$

$$\begin{array}{ll} & ((b+a)(\underline{c+c}))d \\ \stackrel{\text{A3}}{\rightarrow} & \underline{((b+a)c)d} \\ \stackrel{\text{A5}}{\rightarrow} & \underline{(b+a)(cd)} \\ \stackrel{\text{A4}}{\rightarrow} & b(cd)+a(cd) \end{array}$$

Die beiden berechneten Normalformen sind äquivalent (modulo $=_{\text{AC}}$) und daher auch die Ausgangsterme.

Aufgabe 2.10 Leiten Sie die folgenden drei Äquivalenzen mit dem Kalkül ab.

a) $((a+a)(b+b))(c+c) = a(bc)$

b) $(a+a)(bc)+(ab)(c+c) = (a(b+b))(c+c)$

c) $((a+b)c+ac)d = (b+a))(cd)$

2.4 Parallele und kommunizierende Prozesse

Durch den *Parallelоperator (merge)* $\|$ wird die parallele (besser: nebenläufige) Ausführung der beiden Prozesse dargestellt, die er als Argument hat. In den folgenden Regeln sei $\{v, w\} \subseteq A$ und x, x', y, y' seien Prozessterme.

$$\frac{x \stackrel{v}{\rightarrow} \surd}{x\|y \stackrel{v}{\rightarrow} y} \qquad \frac{x \stackrel{v}{\rightarrow} x'}{x\|y \stackrel{v}{\rightarrow} x'\|y}$$

$$\frac{y \stackrel{v}{\rightarrow} \surd}{x\|y \stackrel{v}{\rightarrow} x} \qquad \frac{y \stackrel{v}{\rightarrow} y'}{x\|y \stackrel{v}{\rightarrow} x\|y'}$$

Zwei parallel ablaufende Prozesse kommunizieren mittels einer Kommunikationsfunktion.

Eine *Kommunikationsfunktion* $\gamma : A \times A \rightarrow A$ erzeugt für jedes Paar atomarer Aktionen a und b ihre Kommunikations-Aktion $\gamma(a, b)$.

Die Kommunikationsfunktion γ ist kommutativ und assoziativ:

$$\begin{array}{rcl} \gamma(a,b) & \equiv & \gamma(b,a) \\ \gamma(\gamma(a,b),c) & \equiv & \gamma(a,\gamma(b,c)) \end{array}$$

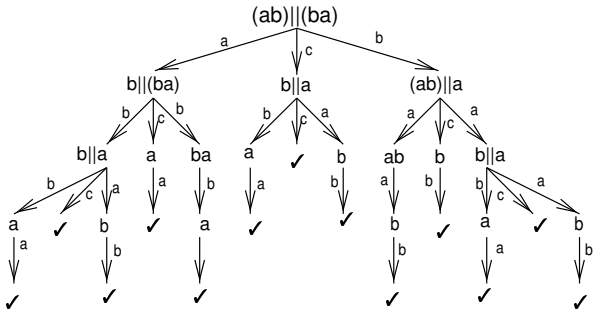
Der Parallelоperator kann eine solche Kommunikation enthalten. Sie ist eine unteilbare Aktion beider Prozesse.

$$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} \surd}{x \| y \xrightarrow{\gamma(v,w)} \surd} \qquad \frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} y'}{x \| y \xrightarrow{\gamma(v,w)} y'}$$

$$\frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} \surd}{x \| y \xrightarrow{\gamma(v,w)} x'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} y'}{x \| y \xrightarrow{\gamma(v,w)} x' \| y'}$$

Beispiel 2.11

Der Prizessgraph von $(ab)\|(ba)$:



Links-Merge und Kommunikations-Merge

Um eine Axiomatisierung mit dem Paralleloperator zu erhalten, werden zwei Hilfsoperatoren benötigt:

Der *Links-Merge-Operator (left merge)* $\between$ erlaubt die Ausführung der ersten Transition des ersten (linken) Argumentes:

$$\frac{x \xrightarrow{v} \surd}{x \between y \xrightarrow{v} y} \qquad \frac{x \xrightarrow{v} x'}{x \between y \xrightarrow{v} x' \| y}$$

Der *Kommunikations-Merge-Operator (communication merge)* $|$ stellt die Kommunikation der beiden ersten Transitionen der beiden Argumente dar:

$$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} \surd}{x | y \xrightarrow{\gamma(v,w)} \surd} \qquad \frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} y'}{x | y \xrightarrow{\gamma(v,w)} y'}$$

$$\frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} \surd}{x | y \xrightarrow{\gamma(v,w)} x'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} y'}{x | y \xrightarrow{\gamma(v,w)} x' \| y'}$$

Die Erweiterung der elementaren Prozessalgebra um die Operatoren *Paralleloperator (merge)* $\|$, *Links-Merge-Operator (left merge)* $\between$ und *Kommunikations-Merge-Operator (communication*

merge) | heißt PAP (*Prozessalgebra mit Parallelismus*). In ihr sollen die neuen Paralleloperatoren stärker binden als $+$, d.h.: $a \between b + a \| b$ steht für $(a \between b) + (a \| b)$. Der Paralleloperator $\|$ kann durch $\between$ und | ausgedrückt werden: $s \| t \leftrightarrow (s \between t + t \between s) + s|t$.

Anmerkung: PAP ist eine konservative Erweiterung von BPA, d.h. die neuen Transitionsregeln verändern nicht die alten. Anders ausgedrückt bedeutet dies, dass der auf BPA eingeschränkte Prozessgraph unverändert bleibt.

Satz 2.12 *Die Äquivalenzrelation Bisimulation ist eine Kongruenzrelation, d.h.: wenn $s \leftrightarrow s'$ und $t \leftrightarrow t'$, dann*

- $s + t \leftrightarrow s' + t'$,
- $s \cdot t \leftrightarrow s' \cdot t'$,
- $s \| t \leftrightarrow s' \| t'$,
- $s \between t \leftrightarrow s' \between t'$ *und*
- $s|t \leftrightarrow s'|t'$.

Axiome des PAP-Kalküls: Axiome A1, ..., A5 (Seite 10) und

M1	$x \| y$	$=$	$(x \between y + y \between x) + x|y$
LM2	$v \between y$	$=$	$v \cdot y$
LM3	$(v \cdot x) \between y$	$=$	$v \cdot (x \| y)$
LM4	$(x + y) \between z$	$=$	$x \between z + y \between z$
CM5	$v|w$	$=$	$\gamma(v, w)$
CM6	$v|(w \cdot y)$	$=$	$\gamma(v, w) \cdot y$
CM7	$(v \cdot x)|w$	$=$	$\gamma(v, w) \cdot x$
CM8	$(v \cdot x)|(w \cdot y)$	$=$	$\gamma(v, w) \cdot (x \| y)$
CM9	$(x + y)|z$	$=$	$x|z + y|z$
CM10	$x|(y + z)$	$=$	$x|y + x|z$

Satz 2.13 *Der PAP-Kalkül ist korrekt, d.h.: $s = t \Rightarrow s \leftrightarrow t$.*

Beweis:
Beweisskizze: Da die Bisimulationsäquivalenz eine Kongruenz ist, genügt es für jedes Axiom $s = t$ die Relation $\sigma(s) \leftrightarrow \sigma(t)$ für alle Substitutionen von den Variablen aus s und t in Processterme zu beweisen. □

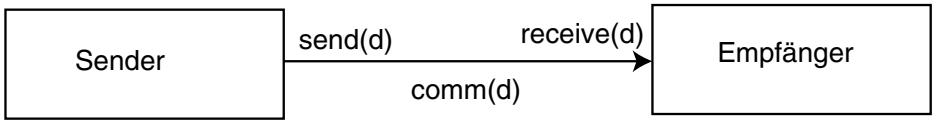


Abbildung 2.1: Kommunikation mit Sender und Empfänger

Satz 2.14 *Der PAP-Kalkül ist vollständig, d.h.:* $s \leftrightarrow t \;\Rightarrow\; s = t$.

Beweis:
Beweisskizze: Dies kann man beweisen, indem die Axiome für PAP in ein (Term-) Ersetzungskalkül modulo + verwandelt. Jeder Prozessterm über PAP ist in Normalform reduzierbar. Wenn $s \leftrightarrow t$ ist, wobei s und t Normalformen s' und t' haben, dann gilt $s' =_{\mathrm{AC}} t'$, also auch $s = s' =_{\mathrm{AC}} t' = t$. □

Abbruch (deadlock) und **Verdeckung** (encapsulation) dienen dazu, Teile einer Kommunikation (wie $send(d)$ und das zugehörige $receive(d)$) zu einer Operation (z.B. $comm(d)$, vergl. Abb. 2.1) zu verschmelzen. Darüberhinaus können diese Operationen als Einzelaktionen unterbunden werden.

Der Operator *Abbruch* δ zeigt kein sichtbares Verhalten. Es gibt daher auch dazu keine Transitionsregel.

Der Operator *Verdecken* ∂_H, mit $H \subseteq A$, benennt alle Aktionen aus H, die bei ihm als Argument auftreten, in δ um:

$$\frac{x \xrightarrow{v} \surd \quad (v \notin H)}{\partial_H(x) \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} x' \quad (v \notin H)}{\partial_H(x) \xrightarrow{v} \partial_H(x')}$$

Der Bildbereich der Kommunikationsfunktion γ wird um δ erweitert:

$$\gamma : A \times A \to A \cup \{\delta\}.$$

Das soll bedeuten: wenn a und b nicht kommunizieren, dann soll $\gamma(a, b) \equiv \delta$ gelten.

Der Verdeckungsoperator erzwingt Kommunikation. Beispielsweise kann $\partial_{\{a,b\}}(a \| b)$ nur als $\gamma(a, b)$ ausgeführt werden (falls $\gamma(a, b) \neq \delta$).

Definition 2.15 *Die Erweiterung des Kalküls PAP durch die nachstehenden Axiome für Abbruch und Verdeckung wird mit ACP bezeichnet (algebra of communicating processes).*

Axiome des Kalküls ACP : die Axiome von PAP und folgende für Abbruch und Verdeckung:

$$
\begin{array}{lrcll}
\text{A6} & x+\delta & = & x & \\
\text{A7} & \delta\cdot x & = & \delta & \\
 & & & & \\
\text{D1} & \partial_H(v) & = & v & (v\notin H)\\
\text{D2} & \partial_H(v) & = & \delta & (v\in H)\\
\text{D3} & \partial_H(\delta) & = & \delta & \\
\text{D4} & \partial_H(x+y) & = & \partial_H(x)+\partial_H(y) & \\
\text{D5} & \partial_H(x\cdot y) & = & \partial_H(x)\cdot\partial_H(y) & \\
 & & & & \\
\text{LM11} & \delta\between x & = & \delta & \\
\text{CM12} & \delta|x & = & \delta & \\
\text{CM13} & x|\delta & = & \delta &
\end{array}
$$

Beispiel 2.16 Der Prozessterm zum einführenden Beispiel zu Abb. 2.1 lautet:

$\partial_{\{send(0),send(1),read(0),read(1)\}}((send(0)+send(1))\|(read(0)+read(1)))$
mit $\gamma(send(d),read(d))=comm(d)$ für $d\in\{0,1\}$.

Theorem 2.17 *a) Bisimulation ist eine Kongruence für ACP: wenn $s \underline{\leftrightarrow} s'$ und $t \underline{\leftrightarrow} t'$, dann $s+t \underline{\leftrightarrow} s'+t'$, $s\cdot t \underline{\leftrightarrow} s'\cdot t'$, $s\|t \underline{\leftrightarrow} s'\|t'$, $s\between t \underline{\leftrightarrow} s'\between t'$, $s|t \underline{\leftrightarrow} s'|t'$ und $\partial_H(s) \underline{\leftrightarrow} \partial_H(t)$.*

b) Der Kalkül ACP ist korrekt: $s=t \;\Rightarrow\; s \underline{\leftrightarrow} t$.

c) Der Kalkül ACP ist vollständig: $s \underline{\leftrightarrow} t \;\Rightarrow\; s=t$.

Beispiel 2.18

Seinen $\gamma(a,b)\equiv c$ und $\gamma(a',b')\equiv c'$ zunächst die einzigen Kommunikationsaktionen zwischen Aktionen.

$$
\begin{array}{l}
(a+a')\|(b+b')\\
\overset{\text{M1}}{=}\\
(a+a')\between(b+b')+(b+b')\between(a+a')\\
+(a+a')|(b+b')\\
\overset{\text{LM4, CM9,10}}{=}\\
a\between(b+b')+a'\between(b+b')+b\between(a+a')+b'\between(a+a')\\
+a|b+a|b'+a'|b+a'|b'\\
\overset{\text{LM2, CM5}}{=}\\
a\cdot(b+b')+a'\cdot(b+b')+b\cdot(a+a')+b'\cdot(a+a')\\
+c+\delta+\delta+c'\\
\overset{\text{A6}}{=}\\
a\cdot(b+b')+a'\cdot(b+b')+b\cdot(a+a')+b'\cdot(a+a')\\
+c+c'
\end{array}
$$

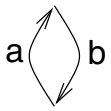


Abbildung 2.2: Transitionssystem für abab...

Beispiel 2.19 (Fortsetzung)

Sei nun $H = \{a, a', b, b'\}$.

$$
\begin{aligned}
&\partial_H((a + a')\|(b + b')) \\
&= \\
&\partial_H(a \cdot (b + b') + a' \cdot (b + b') + b \cdot (a + a') \\
&+ b' \cdot (a + a') + c + c') \\
&\overset{\text{D1,2,4,5}}{=} \\
&\delta \cdot \partial_H(b + b') + \delta \cdot \partial_H(b + b') + \delta \cdot \partial_H(a + a') \\
&+ \delta \cdot \partial_H(a + a') + c + c' \\
&\overset{\text{A6,7}}{=} \\
&c + c'
\end{aligned}
$$

∂_H erzwingt also die Kommunication zwischen a und b einerseits und zwischen a' und b' andererseits.

Aufgabe 2.20 Konstruieren Sie die Prozessgraphen zu folgenden Prozesstermen:

a) $\partial_{\{a\}}(ac)$

b) $\partial_{\{a\}}((a + b)c)$

c) $\partial_{\{c\}}((a + b)c)$

d) $\partial_{\{a,b\}}((ab)\|(ba))$ mit $\gamma(a, b) = c$

2.5 Rekursion und Abstraktion

Bislang wurden nur Prozesse endlicher Länge spezifiziert. Unendliche Prozesse, wie z.B. der Prozess *ababab*... mit dem Transitionssystem von Abb. 2.2 werden durch Rekursion definiert.

Konkret geschiet dies z.B. durch das folgende *rekursive Gleichungssystem*:

$$\begin{aligned} X &= aY \\ Y &= bX \end{aligned}$$

Dabei sind X und Y *Rekursionsvariable*, die zwei Zustände des Prozesses representieren.

Definition 2.21 *Eine* rekursive Spezifikation *ist eine Menge von Gleichungen der Form:*

$$\begin{aligned} X_1 &= t_1(X_1, \ldots, X_n) \\ &\vdots \\ X_n &= t_n(X_1, \ldots, X_n) \end{aligned}$$

wobei $t_i(X_1, \ldots, X_n)$ Prozessterme des Kalküls ACP mit (möglichen) freien Variablen $X_1, \ldots, X_n$ sind.

Definition 2.22 *Prozesse $p_1, \ldots, p_n$ werden als eine* Lösung *einer rekursiven Spezifikation*

$$\{X_i = t_i(X_1, \ldots, X_n) \mid i \in \{1, \ldots, n\}\}$$

modulo Bisimulations-Äquivalenz bezeichnet, falls $p_i \leftrightarrow t_i(p_1, \ldots, p_n)$ for $i \in \{1, \ldots, n\}$ gilt.

Lösungen sollen eindeutig bezüglich Bisimulations-Äquivalenz sein, d.h. falls $p_1, \ldots, p_n$ und $q_1, \ldots, q_n$ zwei solche Lösungen sind, dann erwarten wir $p_i \leftrightarrow q_i$ for $i \in \{1, \ldots, n\}$.

Beispiel 2.23

a) $X = X$ hat alle Prozesse als Spezifikation.

b) Alle Prozesse p mit $p \xrightarrow{a} \surd$ sind Lösungen von der rekursiven Spezifikation $\{X = a + X\}$.

c) Alle nichtterminierende Prozesse sind Lösungen von der rekursiven Spezifikation $\{X = Xa\}$.

d) $\{X = aY, Y = bX\}$ hat als einzige Lösung $\{X \leftrightarrow abab\ldots\}$ und $\{Y \leftrightarrow baba\ldots\}$.

e) $\{X = Y, Y = aX\}$ hat als einzige Lösung $\{X \leftrightarrow aaaa\ldots\}$ und $\{Y \leftrightarrow aaaa\ldots\}$.

f) $\{X = a \| X\}$ hat zwei verschiedene Lösungen $\{X \leftrightarrow aaaa\ldots\}$ und $\{X \leftrightarrow (a+b)(a+b)(a+b)(a+b)\ldots\}$ wobei $\gamma(a,a) := \delta$ und $\gamma(a,b) := \delta$.

g) $\{X = (a+b) \between X\}$ hat als einzige Lösung $X \leftrightarrow (a+b)(a+b)(a+b)\ldots\}$.

"Einzig" bzw. "zwei verschiedene" ist hier jeweils modulo Bisimulation zu verstehen.

Definition 2.24 *Eine rekursive Spezifikation*

$$\{X_i = t_i(X_1, \ldots, X_n) \mid i \in \{1, \ldots, n\}\}$$

heißt geschützt (guarded) *falls die rechten Seiten ihrer Gleichungen in folgende Form*

$$a_1 \cdot s_1(X_1, \ldots, X_n) + \cdots + a_k \cdot s_k(X_1, \ldots, X_n) + b_1 + \cdots + b_\ell$$

überführt werden können, indem die Axiome des Kalküls ACP benutzt werden. Außerdem dürfen bei dieser Umformung Variable einer Rekursionsgleichung durch die entsprechende rechte Seite ersetzt werden.

Eine rekursive Spezifikation ist genau dann eindeutig (modulo Bisimulation), wenn sie geschützt ist. Z.B. sind einige der rekursiven Spezifikationen von Beispiel 2.23 nicht geschützt.

Beispiel 2.25 Sei $\gamma(a, b) \equiv c$ und $\gamma(b, b) \equiv c$. $\{X{=}Y \| Z,\ Y{=}Z + a,\ Z{=}bZ\}$ ist geschützt, denn:

- $Z{=}bZ$ hat schon die gewünschte Form.
- $Y{=}Z + a$ wird transformiert, indem Z durch bZ ersetzt wird.
- $X{=}Y \| Z$ wird transformiert, indem Y durch $bZ + a$ und Z durch bZ ersetzt wird und der so erhaltene Term $(bZ + a) \| bZ$ mittels der Axiome transformiert wird:

$$\begin{aligned} & (bZ + a) \| bZ \\ = \ & (bZ + a) \between bZ + bZ \between (bZ + a) + (bZ + a) | bZ \\ = \ & bZ \between bZ + a \between bZ + bZ \between (bZ + a) + bZ | bZ + a | bZ \\ = \ & b(Z \| bZ) + abZ + b(Z \| (bZ + a)) + c(Z \| Z) + cZ \end{aligned}$$

Definition 2.26 *Für eine geschützte rekursive Spezifikation E:*

$$\{X_i = t_i(X_1, \ldots, X_n) \mid i \in \{1, \ldots, n\}\}$$

soll nun

$$\langle X_i | E \rangle \qquad (i \in \{1, \ldots, n\})$$

die Lösung X_i in E bedeuten.

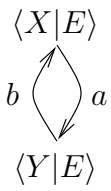
Transitionsregeln für Rekursion

$$\frac{t_i(\langle X_1 | E \rangle, \ldots, \langle X_n | E \rangle) \xrightarrow{v} \surd}{\langle X_i | E \rangle \xrightarrow{v} \surd}$$

$$\frac{t_i(\langle X_1|E\rangle, \ldots, \langle X_n|E\rangle) \xrightarrow{v} y}{\langle X_i|E\rangle \xrightarrow{v} y}$$

d.h.: $\langle X_i|E\rangle$ übernimmt das Verhalten von $t_i(\langle X_1|E\rangle, \ldots, \langle X_n|E\rangle)$:

Beispiel 2.27 Sei E die geschützte rekursive Spezifikation $\{X{=}aY,\ Y{=}bX\}$. Der Prozessgraph von $\langle X|E\rangle$ ist:



Wir leiten her: $\langle X|E\rangle \xrightarrow{a} \langle Y|E\rangle$:

$$\frac{a \xrightarrow{a} \surd \qquad \dfrac{}{v \xrightarrow{v} \surd} \qquad v := a}{a \cdot \langle Y|E\rangle \xrightarrow{a} \langle Y|E\rangle \qquad \dfrac{x \xrightarrow{v} \surd}{x \cdot y \xrightarrow{v} y} \qquad v := a,\quad x := a,\quad y := \langle Y|E\rangle}$$

$$\frac{a \cdot \langle Y|E\rangle \xrightarrow{a} \langle Y|E\rangle \qquad \dfrac{a \cdot \langle Y|E\rangle \xrightarrow{v} y}{\langle X|E\rangle \xrightarrow{v} y} \qquad v := a,\quad y := \langle Y|E\rangle}{\langle X|E\rangle \xrightarrow{a} \langle Y|E\rangle}$$

Aufgabe 2.28 Leiten Sie für die folgenden drei Prozessterme den Prozessgraphen ab:

a) $\langle X|X = ab\rangle$

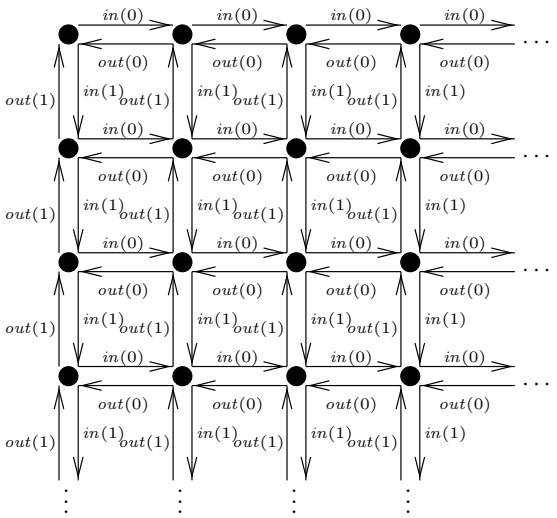
b) $\langle X|X = aXb\rangle$

c) $\langle X|X = aXb + c\rangle$

Anmerkung: ACP mit geschützter Rekursion ist eine konservative Erweiterung von ACP und Bisimulation ist eine Kongruenzrelation.

Beispiel 2.29 Multimenge (bag) über $\{0, 1\}$.

Die Elemente 0 and 1 werden durch $in(0)$ und $in(1)$ in die Multimenge eingefügt und können durch $out(0)$ und $out(1)$ in beliebiger Reihenfolge entfernt werden.



Eine rekursive Spezifikation ist:

$$X \;=\; in(0)(X \| out(0)) + in(1)(X \| out(1))$$

Aufgabe 2.30 Geben Sie eine Bisimulations-Relation für Beispiel 2.29 an, bei der der Prozessterm $\langle X|E\rangle$ dem Anfangszustand (links-oben) zugeordnet wird.

Axiome für Rekursion:

Für eine rekursive Spezifikation E gilt:

$$\{X_i = t_i(X_1, \ldots, X_n) \mid i \in \{1, \ldots, n\}\}$$

$$\text{RDP} \quad \langle X_i|E\rangle \;=\; t_i(\langle X_1|E\rangle, \ldots, \langle X_n|E\rangle) \qquad (i \in \{1, \ldots, n\})$$

RSP Falls $y_i = t_i(y_1, \ldots, y_n)$ für $i \in \{1, \ldots, n\}$, dann

$$y_i \;=\; \langle X_i|E\rangle \qquad (i \in \{1, \ldots, n\})$$

Korrektheit

Der Kalkül ACP mit geschützter Rekursion ist korrekt bezüglich Bisimulsation:

$$s = t \;\Rightarrow\; s \underline{\leftrightarrow} t$$

RSP ist **nicht** korrekt für ungeschützte Rekursion. Beispielsweise ergibt RSP wegen $t = t$ die Gleichung

$$t \;=\; \langle X \,|\, X{=}X\rangle$$

für alle Prozessterme t.

Beispiel 2.31

$$\begin{aligned}\langle Z \mid Z{=}aZ\rangle &\overset{\text{RDP}}{=} a\langle Z \mid Z{=}aZ\rangle\\ &\overset{\text{RDP}}{=} a(a\langle Z \mid Z{=}aZ\rangle)\\ &\overset{\text{A5}}{=} (aa)\langle Z \mid Z{=}aZ\rangle\end{aligned}$$

Also gilt mit RSP,

$$\langle Z \mid Z{=}aZ\rangle = \langle X \mid X{=}(aa)X\rangle$$

Weiter gilt:

$$\begin{aligned}\langle Z \mid Z{=}aZ\rangle &\overset{\text{RDP}}{=} a\langle Z \mid Z{=}aZ\rangle\\ &\overset{\text{RDP}}{=} a(a\langle Z \mid Z{=}aZ\rangle)\\ &\overset{\text{RDP}}{=} a(a(a\langle Z \mid Z{=}aZ\rangle))\\ &\overset{\text{A5}}{=} ((aa)a)\langle Z \mid Z{=}aZ\rangle\end{aligned}$$

und mit RSP:

$$\langle Z \mid Z{=}aZ\rangle = \langle Y \mid Y{=}((aa)a)Y\rangle$$

also:

$$\begin{aligned}\langle X \mid X{=}(aa)X\rangle &= \langle Z \mid Z{=}aZ\rangle\\ &= \langle Y \mid Y{=}((aa)a)Y\rangle\end{aligned}$$

Aufgabe 2.32 Prüfen Sie für $\gamma(a,b) \equiv c$ die folgende Ableitung von

$$\partial_{\{a,b\}}(\langle X \mid X{=}aX\rangle \| \langle Y \mid Y{=}bY\rangle) = \langle Z \mid Z{=}cZ\rangle$$

$$\begin{aligned}&\langle X \mid X{=}aX\rangle \| \langle Y \mid Y{=}bY\rangle\\ =\ &\langle X \mid X{=}aX\rangle \between \langle Y \mid Y{=}bY\rangle\\ +\ &\langle Y \mid Y{=}bY\rangle \between \langle X \mid X{=}aX\rangle\\ +\ &\langle X \mid X{=}aX\rangle | \langle Y \mid Y{=}bY\rangle\\ =\ &a(\langle X \mid X{=}aX\rangle \| \langle Y \mid Y{=}bY\rangle)\\ +\ &b(\langle Y \mid Y{=}bY\rangle \| \langle X \mid X{=}aX\rangle)\\ +\ &c(\langle X \mid X{=}aX\rangle \| \langle Y \mid Y{=}bY\rangle)\end{aligned}$$

Aus

$$\begin{aligned}&\partial_{\{a,b\}}(\langle X \mid X{=}aX\rangle \| \langle Y \mid Y{=}bY\rangle)\\ =\ &c \cdot \partial_{\{a,b\}}(\langle X \mid X{=}aX\rangle \| \langle Y \mid Y{=}bY\rangle)\end{aligned}$$

folgt mit RSP das Ergebnis.

Sei E, E' eine geschützte rekursive Spezifikation, wobei E' aus E entsteht, indem die rechten Seiten ihrer rekursiven Gleichungen modifiziert werden, indem

- die Axiome für ACP mit geschützter Rekursion benutzt werden und

- Rekursionsvariable durch rechte Seiten entsprechender Rekursionsgleichungen ersetzt werden.

Dann kann $\langle X|E\rangle = \langle X|E'\rangle$ im Kalkül ACP mitgeschützter Rekursion für alle Rekursionsvariablen abgeleitet werden. Zum Beispiel kann

$$\langle X \mid X{=}aX{+}aX\rangle = \langle X \mid X{=}(aa)X\rangle$$

abgeleitet werden, indem

- erst A3 angewandt wird: $(x + x = x)$,
- dann X durch die rechte Seite aX ersetzt wird,
- und dann zuletzt A5 angewandt wird: $((xy)z = x(yz))$.

Abstraktion

Wird ein spezifiziertes System implementiert, so führt es i.A. mehr Aktionen aus, als in der Spezifikation vorgegeben sind. Diese Aktionen sind zwar für den internen Ablauf wichtig, für den Benutzer oder Auftraggeber jedoch nicht relevant. Um dies zu beschreiben, werden *stille* (silent), *spontane* oder *interne* Aktionen eingeführt. Ihre Bezeichnung ist meist τ (Tau).

Der stille Systemschritt τ kann ohne Vorbedingung ausgeführt werden und terminiert dann erfolgreich:

$$\frac{}{\tau \xrightarrow{\tau} \surd}$$

Die Transitionsregeln werden nun so erweitert, dass sie τ enthalten:

neue Menge von Aktionen :

$$A' := A \cup \{\tau\}$$

neue Kommunikationsfunktion :

$$\gamma : A' \times A' \to A \cup \{\delta\}.$$

Der *Abstraktions-Operator* τ_I, mit $I \subseteq A$, benennt alle Aktionen aus I, die er als Argument führt, in τ um:

$$\frac{x \xrightarrow{v} \surd \quad (v \notin I)}{\tau_I(x) \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} x' \quad (v \notin I)}{\tau_I(x) \xrightarrow{v} \tau_I(x')}$$

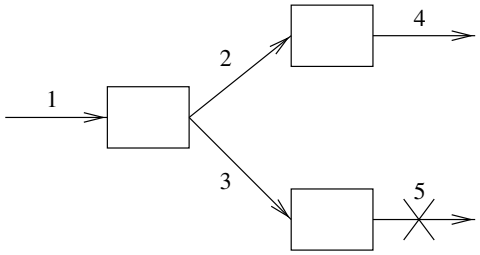
$$\frac{x \xrightarrow{v} \surd \quad (v \in I)}{\tau_I(x) \xrightarrow{\tau} \surd} \qquad \frac{x \xrightarrow{v} x' \quad (v \in I)}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}$$

Der Kalkül ACP über A' mit τ-Transition und Abstraktionsoperator wird mit ACP_τ bezeichnet.

Wenn in einer Folge abc von Aktionen b intern ist, dann ist $a\tau b$ (für die externe Spezifikation) äquivalent zu ac, d.h. stille Aktionen sind (extern) wirkungslos. Wie das folgende Beispiel zeigt, kann nicht generell still gleich wirkungslos gesetzt werden.

Beispiel 2.33

Daten d werden über Kanal 1 empfangen ($r_1(d)$) und über Kanal 2 oder 3 weitergeleitet ($c_2(d), c_3(d)$). Im ersten Fall wird d über Kanal 4 weitergeleitet ($s_4(d)$), im zweiten Fall ist dies jedoch nicht möglich, da der Kanal 5 blockiert ist, d.h. $s_5(d)$ ist blockiert.



Dieses Verhalten wird durch folgenden Prozessausdruck beschrieben:

$$\partial_{\{s_5(d)\}}(r_1(d) \cdot (c_2(d) \cdot s_4(d) + c_3(d) \cdot s_3(d)))$$

$$= r_1(d) \cdot (c_2(d) \cdot s_4(d) + c_3(d) \cdot \delta)$$

(Die Umformung erfolgt mit den Regeln D1,D2,D4,D5 von Seite 18.)

Spezifiziert man $c_2(d)$ und $c_3(d)$ als interne Aktionen, so erhält man

$$r_1(d) \cdot (\tau \cdot s_4(d) + \tau \cdot \delta).$$

Werden beide stille Aktionen τ gestrichen, so erhält man mit $r_1(d) \cdot (s_4(d) + \delta)$ einen Prozess ohne Verklemmung. Er wird daher als nicht (Verhaltens-)aquivalent betrachtet.

Beispiel 2.34 Weitere nichtäquivalente Prozessausdrücke:

- $a + \tau\delta$ und a
- $\partial_{\{b\}}(a + \tau b)$ und $\partial_{\{b\}}(a + b)$
- $a + \tau b$ und $a + b$

Es stellt sich also die Frage: Welche τ-Transitionen sind wirkungslos?

Antwort: diejenigen, deren Streichung nicht das Verhalten ändern, wie zum Beispiel: $a+\tau(a+b)$ und $a+b$. Nach der Ausführung von τ ist a immer noch ausführbar! Dies wird durch die folgende Definition der *Verzweigungs-Bisimulation* (branching bisimulation) formalisiert. Dazu wird ein spezielles Terminations-Prädikat $\downarrow$ benutzt.

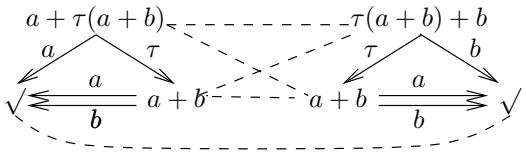
Definition 2.35 *Sei $\downarrow$ ein spezielles Terminations-Prädikat und $\surd$ ein Zustand mit $\surd \downarrow$* [1]*. Eine* Verzweigungs-Bisimulation (branching bisimulation) *ist eine binäre Relation $\mathcal{B}$ auf Prozessen, für die gilt:*

1. *Wenn $p \mathcal{B} q$ und $p \xrightarrow{a} p'$, dann*
   - *entweder $a \equiv \tau$* [2] *und $p' \mathcal{B} q$*
   - *oder $q \xrightarrow{\tau} \cdots \xrightarrow{\tau} q_0$ mit $p \mathcal{B} q_0$ und $q_0 \xrightarrow{a} q'$ mit $p' \mathcal{B} q'$*
2. *Wenn $p \mathcal{B} q$ und $q \xrightarrow{a} q'$, dann*
   - *entweder $a \equiv \tau$ und $p \mathcal{B} q'$*
   - *oder $p \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_0$ mit $p_0 \mathcal{B} q$ und $p_0 \xrightarrow{a} p'$ mit $p' \mathcal{B} q'$*
3. *Wenn $p \mathcal{B} q$ und $p \downarrow$, dann $q \xrightarrow{\tau} \cdots \xrightarrow{\tau} q_0$ wobei $q_0 \downarrow$*
4. *Wenn $p \mathcal{B} q$ und $q \downarrow$, dann $p \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_0$ wobei $p_0 \downarrow$*

Zwei Prozesse p und q heißen verzweigungsbisimilar (branching bisimilar)*, in Zeichen $p \underline{\leftrightarrow}_b q$, wenn es eine Verzweigungs-Bisimulations-Relation $\mathcal{B}$ gibt mit $p \mathcal{B} q$.*

Beispiel 2.36

$a + \tau(a+b) \underline{\leftrightarrow}_b \tau(a+b) + b$



Die Relation $\mathcal{B}$ ist definiert durch:

$a + \tau(a+b) \mathcal{B} \tau(a+b) + b$

$a + b \mathcal{B} \tau(a+b) + b$

$a + \tau(a+b) \mathcal{B} a + b$

$a + b \mathcal{B} a + b$

$\surd \mathcal{B} \surd$

[1] Ist s ein Zustand und P ein Prädikat für s, dann wird $P(s)$ als sP geschrieben. $\surd \downarrow$ bedeutet also, dass das Prädikat $\downarrow$ für den Zustand $\surd$ gilt.

[2] $\equiv$ bezeichnet die syntaktische Gleichheit.

Aufgabe 2.37

1. Geben Sie eine Verzweigungs-Bisimulations-Relation an, die zeigt dass $\tau(\tau(a+b)+b)+a$ und $a+b$ verzweigungsbisimilar sind.
2. Begründen Sie, warum $\tau a + \tau b$ und $a+b$ nicht verzweigungsbisimilar sind.

Die Verzweigungs-Bisimulations-Relation ist zwar eine Äquivalenzrelation, aber keine Kongruenz bezüglich BPA. Beispielsweise sind τa und a verzweigungsbisimilar, aber nicht $\tau a + b$ und $a + b$. Milner [**?**] hat gezeigt, dass man dieses Problem dadurch lösen kann, dass eine Initialisierungsbedingung gefordert wird: initiale τ-Transitionen werden nie eliminiert, das heisst in solchen Fällen wird keine Äquivalenz definiert:

- $a + \tau\delta$ und a
- $\partial_{\{b\}}(a + \tau b)$ und $\partial_{\{b\}}(a + b)$
- $a + \tau b$ und $a + b$
- b und τb

Definition 2.38 *Sei $\downarrow$ ein spezielles Terminations-Predikat und $\surd$ ein Zustand mit $\surd\downarrow$. Eine* initiale Verzweigungs-Bisimulation (rooted branching bisimulation) *ist eine binäre Relation $\mathcal{B}$ auf Prozessen, für die gilt:*

1. *Wenn $p\,\mathcal{B}\,q$ und $p \xrightarrow{a} p'$, dann $q \xrightarrow{a} q'$ mit $p' \underline{\leftrightarrow}_b q'$.*
2. *Wenn $p\,\mathcal{B}\,q$ und $q \xrightarrow{a} q'$, dann $p \xrightarrow{a} p'$ mit $p' \underline{\leftrightarrow}_b q'$.*
3. *Wenn $p\,\mathcal{B}\,q$ und $p\downarrow$, dann $q\downarrow$.*
4. *Wenn $p\,\mathcal{B}\,q$ und $q\downarrow$, dann $p\downarrow$*

Zwei Prozesse p und q heißen initial verzweigungsbisimilar (rooted branching bisimilar)*, in Zeichen $p \underline{\leftrightarrow}_{rb} q$, wenn es eine initiale Verzweigungs-Bisimulations-Relation $\mathcal{B}$ gibt mit $p\,\mathcal{B}\,q$.*

Initiale Verzweigungs-Bisimulation ist wie Verzweigungs-Bisimulation eine Äquivalenzrelation. Es gilt $\underline{\leftrightarrow} \subseteq \underline{\leftrightarrow}_{rb} \subseteq \underline{\leftrightarrow}_b$ und $\underline{\leftrightarrow} = \underline{\leftrightarrow}_b$ falls τ nicht vorkommt (z.B. in ACP).

Aufgabe 2.39 Bestimmen Sie für jedes der folgenden Paare von Prozesstermen, ob es bisimilar, initial verzweigungsbisimilar oder verzweigungsbisimilar ist bzw. nicht ist - möglichst mit stichhaltiger Begründung:

a) $(a+b)(c+d)$ und $ac + ad + bc + bd$,

b) $(a+b)(c+d)$ und $(b+a)(d+c)+a(c+d)$,

c) $\tau(b+a)+\tau(a+b)$ und $a+b$,

d) $c(\tau(b+a)+\tau(a+b))$ und $c(a+b)$ sowie

e) $a(\tau b+c)$ und $a(b+\tau c)$.

geschützte lineare Rekursion

Alle Prozessterme τs stellen eine Lösung für die Spezifikation $X = \tau X$ dar, da $\tau s \leftrightarrow_{rb} \tau\tau s$. Also ist $X = \tau X$ *ungeschützt.*

Definition 2.40 *1. Eine rekursive Spezifikation ist* linear*, wenn ihre rekursiven Gleichungen die folgende Form haben:*

$$X \;=\; a_1X_1 + \cdots + a_kX_k + b_1 + \cdots + b_\ell \qquad\qquad (a_i, b_j \in A \cup \{\tau\})$$

2. Eine lineare rekursive Specification E ist geschützt*, wenn es keine unendliche Folge von τ-Transitionen der folgenden Form gibt:*

$$\langle X|E\rangle \xrightarrow{\tau} \langle X'|E\rangle \xrightarrow{\tau} \langle X''|E\rangle \xrightarrow{\tau} \cdots$$

Die geschützten linearen rekursiven Spezifikationen sind genau diejenigen rekursiven Spezifikationen, die eine eindeutige Lösung modulo initialer Verzweigungs-Bisimulation haben.

Satz 2.41 *Initiale Verzweigungs-Bisimulation ist eine Kongrunzrelation für ACP_τ und geschützter linearer Rekursion.*

Wieder geben wie eine Axiomatisierung des erweiterten Kalküls ACP_τ an. ACP_τ bezeichne jetzt den Kalkül ACP mit τ-Aktion, Abstraktionsoperator, geschützter linearer Rekursion und den folgenden Axiomen:

Axiome für Abstraktion

B1	$v \cdot \tau$	$=$	v	
B2	$v \cdot (\tau \cdot (x+y) + x)$	$=$	$v \cdot (x+y)$	
TI1	$\tau_I(v)$	$=$	v	$(v \notin I)$
TI2	$\tau_I(v)$	$=$	τ	$(v \in I)$
TI3	$\tau_I(\delta)$	$=$	δ	
TI4	$\tau_I(x+y)$	$=$	$\tau_I(x) + \tau_I(y)$	
TI5	$\tau_I(x \cdot y)$	$=$	$\tau_I(x) \cdot \tau_I(y)$	

Theorem 2.42 *ACP_τ ist korrekt in Bezug auf initiale Verzweigungsbisimulation.*

Aufgabe 2.43 Leiten sie $\tau_{\{b\}}(\langle X|X = aY, Y = bX\rangle) = \langle Z|Z = aZ\rangle$ in ACP_τ ab.

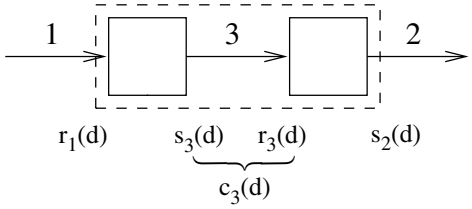
Beispiel 2.44 : Tandempuffer

Betrachtet werden zwei in Serie angeordnete Puffer der Kapazität 1 die mit Kanälen 1, 2 und 3 verbunden sind. $r_i(d)$ bzw. $s_i(d)$ bezeichne das Schreiben bzw. Lesen vom Kanal i, wobei im Folgenden der Datenparameter $d \in \Delta$ weggelassen wird. Δ ist eine endliche Menge von Daten. Das Verhalten ist:

$$\begin{aligned} Q_1 &= \Sigma_{d\in\Delta} r_1\, s_3\, Q_1 \\ Q_2 &= \Sigma_{d\in\Delta} r_3\, s_2\, Q_2 \end{aligned}$$

wobei $d \in \Delta$ weggelassen wird, d.h. wir tun so, als ob Δ nur ein Element enthalten würde:

$$\begin{aligned} Q_1 &= r_1\, s_3\, Q_1 \\ Q_2 &= r_3\, s_2\, Q_2 \end{aligned}$$



Die Puffer Q_1 und Q_2 der Kapazität 1 arbeiten parallel und sind durch die Aktion $\gamma(s_3, r_3) = c_3$ synchronisiert:

$$\tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_2 \| Q_1))$$

Das gemeinsame Verhalten ist dasjenige eines Puffers der Kapazität 2:

$$\begin{aligned} X &= r_1\, Y \\ Y &= r_1\, Z + s_2\, X \\ Z &= s_2\, Y \end{aligned}$$

Was ist die Lösung dieses rekursiven Gleichungssystems E? Im Folgenden berechnen wir im Kalkül A_τ die Lösung:

$$\tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_2 \| Q_1)) = \langle X|E\rangle$$

mit

$$\begin{aligned} X &:= \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_2 \| Q_1)) \\ Y &:= \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_1 \| (s_2 Q_2))) \\ Z &:= \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}((s_3 Q_1) \| (s_2 Q_2))) \end{aligned}$$

Dazu berechnen wir zunächst:

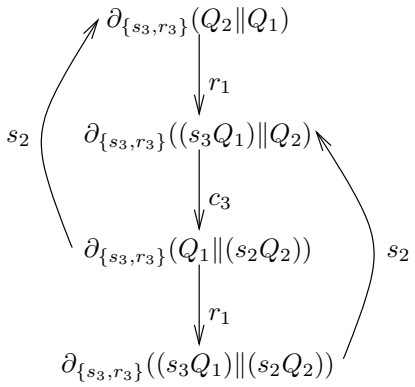
$$
\begin{array}{rl}
 & Q_2 \| Q_1 \\
\stackrel{\text{M1}}{=} & Q_2 \between Q_1 + Q_1 \between Q_2 + Q_2 | Q_1 \\
\stackrel{\text{RDP}}{=} & (r_3 s_2 Q_2) \between Q_1 + (r_1 s_3 Q_1) \between Q_2 \\
 & + (r_3 s_2 Q_2) | (r_1 s_3 Q_1) \\
\stackrel{\text{LM3,CM8}}{=} & r_3 \cdot ((s_2 Q_2) \| Q_1) + r_1 \cdot ((s_3 Q_1) \| Q_2) \\
 & + \delta \cdot ((s_2 Q_2) \| (s_3 Q_1)) \\
\stackrel{\text{A7,A6}}{=} & r_3 \cdot ((s_2 Q_2) \| Q_1) + r_1 \cdot ((s_3 Q_1) \| Q_2)
\end{array}
$$

$$
\begin{array}{rl}
 & \partial_{\{s_3,r_3\}}(Q_2 \| Q_1) \\
= & \partial_{\{s_3,r_3\}}(r_3 \cdot ((s_2 Q_2) \| Q_1) + r_1 \cdot ((s_3 Q_1) \| Q_2)) \\
= & \partial_{\{s_3,r_3\}}(r_3 \cdot ((s_2 Q_2) \| Q_1)) \\
 & + \partial_{\{s_3,r_3\}}(r_1 \cdot ((s_3 Q_1) \| Q_2)) \\
= & \partial_{\{s_3,r_3\}}(r_3) \cdot \partial_{\{s_3,r_3\}}((s_2 Q_2) \| Q_1) \\
 & + \partial_{\{s_3,r_3\}}(r_1) \cdot \partial_{\{s_3,r_3\}}((s_3 Q_1) \| Q_2) \\
= & \delta \cdot \partial_{\{s_3,r_3\}}((s_2 Q_2) \| Q_1) \\
 & + r_1 \cdot \partial_{\{s_3,r_3\}}((s_3 Q_1) \| Q_2) \\
= & r_1 \cdot \partial_{\{s_3,r_3\}}((s_3 Q_1) \| Q_2)
\end{array}
$$

Weiter rechnen wir:

$$
\begin{array}{l}
\partial_{\{s_3,r_3\}}((s_3 Q_1) \| Q_2) = c_3 \cdot \partial_{\{s_3,r_3\}}(Q_1 \| (s_2 Q_2)) \\
\partial_{\{s_3,r_3\}}(Q_1 \| (s_2 Q_2)) = r_1 \cdot \partial_{\{s_3,r_3\}}((s_3 Q_1) \| (s_2 Q_2)) \\
\qquad\qquad + s_2 \cdot \partial_{\{s_3,r_3\}}(Q_2 \| Q_1) \\
\partial_{\{s_3,r_3\}}((s_3 Q_1) \| (s_2 Q_2)) = s_2 \cdot \partial_{\{s_3,r_3\}}((s_3 Q_1) \| Q_2)
\end{array}
$$

Wir fassen die gerechneten Transitionsübergänge zusammen:



Die Axiome für die τ-Aktion und Abstraktion ergeben:

$$\begin{aligned}
& \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_2 \| Q_1)) \\
= \;& \tau_{\{c_3\}}(r_1 \cdot \partial_{\{s_3,r_3\}}((s_3 Q_1) \| Q_2)) \\
= \;& r_1 \cdot \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}((s_3 Q_1) \| Q_2)) \\
= \;& r_1 \cdot \tau_{\{c_3\}}(c_3 \cdot \partial_{\{s_3,r_3\}}(Q_1 \| (s_2 Q_2))) \\
= \;& r_1 \cdot \tau \cdot \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_1 \| (s_2 Q_2))) \\
= \;& r_1 \cdot \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_1 \| (s_2 Q_2)))
\end{aligned}$$

Weiter rechnen wir:

$$\begin{aligned}
\tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_1 \| (s_2 Q_2))) \;=\; & \\
& r_1 \cdot \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}((s_3 Q_1) \| (s_2 Q_2))) \\
& + s_2 \cdot \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_2 \| Q_1)) \\
\tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}((s_3 Q_1) \| (s_2 Q_2))) \;=\; & \\
& s_2 \cdot \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_1 \| (s_2 Q_2)))
\end{aligned}$$

Damit erhalten wir als Lösung für die lineare rekursive Spezifikation E für einen Puffer der Kapazität 2:

$$\begin{aligned}
X &:= \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_2 \| Q_1)) \\
Y &:= \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}(Q_1 \| (s_2 Q_2))) \\
Z &:= \tau_{\{c_3\}}(\partial_{\{s_3,r_3\}}((s_3 Q_1) \| (s_2 Q_2)))
\end{aligned}$$

Die Fairness-Regel

Es ist möglich, durch Abstraktion τ-Schleifen zu konstruieren: $\tau_{\{a\}}(\langle X \,|\, X{=}aX\rangle)$ führt unendlich lange die τ-Aktion aus.

Definition 2.45 *Sei E eine geschützte lineare rekursive Spezifikation.*

- *Rekursionsvariablen X und Y in E sind in der selben* (Fairness-)Gruppe *(cluster) für I, falls $\langle X|E\rangle \xrightarrow{b_1} \cdots \xrightarrow{b_m} \langle Y|E\rangle$ und $\langle Y|E\rangle \xrightarrow{c_1} \cdots \xrightarrow{c_n} \langle X|E\rangle$ für Aktionen b_i und c_i gilt.*
- *a und aX heißen* Ausgang *(exit) der Gruppe C falls:*
  1. *a oder aX ein Summand auf der rechten Seite der Rekursionsgleichung für eine Rekursionsvariable in C ist, und*
  2. *im Fall aX zusätzlich $a \notin I \cup \{\tau\}$ oder $X \notin C$ gilt.*

Die Fairness-Regel CFAR:

Falls X in einer Gruppe I mit Ausgängen

$\{v_1 Y_1, \ldots, v_m Y_m, w_1, \ldots, w_n\}$ ist, dann gilt

$$\begin{aligned}
& v \cdot \tau_I(\langle X|E\rangle) \;= \\
& v \cdot \tau_I(v_1\langle Y_1|E\rangle + \cdots + v_m\langle Y_m|E\rangle + w_1 + \cdots + w_n)
\end{aligned}$$

Zur Erläuterung von CFAR sei E das Gleichungssystem:

$$\begin{array}{lcl} X_1 & = & aX_2 + b_1 \\ & \vdots & \\ X_{n-1} & = & aX_n + b_{n-1} \\ X_n & = & aX_1 + b_n \end{array}$$

$\tau_{\{a\}}(\langle X_1|E\rangle)$ führt τ-Transitionen aus, bis eine Aktion b_i für $i \in \{1, \ldots, n\}$ ausgeführt wird.

Faire Abstraction bedeutet, dass $\tau_{\{a\}}(\langle X_1|E\rangle)$ nicht für immer in einer τ-Schleife bleibt, d.h. irgend wann wird einmal ein b_i ausgeführt:

$$\tau_{\{a\}}(\langle X_1|E\rangle) \underline{\leftrightarrow}_{rb} b_1 + \tau(b_1 + \cdots + b_n)$$

Anfangs führt $\tau_{\{a\}}(\langle X_1|E\rangle)$ die Aktionen b_1 oder τ aus. Im letzteren Fall wird nach einer Reihe von möglichen τ-Aktionen ein b_i ausgeführt.

Beispiel 2.46

$$X \;=\; heads \cdot X + tails$$

$\langle X|E\rangle$ stellt das Werfen einer idealen Münze dar, das mit *tails* endet. Von den Ergebnissen "Kopf" wird abstrahiert: (head) $\tau_{\{heads\}}(\langle X|E\rangle)$.

$\{X\}$ ist die einzige Gruppe für $\{heads\}$ mit dem einzigen Ausgang *tails*. Daher erhält man mit der Regel CFAR:

$$\begin{array}{rcl} \tau \cdot \tau_{\{heads\}}(\langle X|E\rangle) & = & \tau \cdot \tau_{\{heads\}}(tails) \\ & = & \tau \cdot tails \end{array}$$

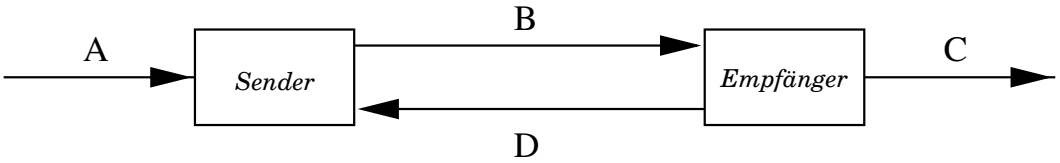
und

$$\begin{array}{rcl} \tau_{\{heads\}}(\langle X|E\rangle) & = & \tau_{\{heads\}}(heads \cdot \langle X|E\rangle + tails) \\ & = & \tau \cdot \tau_{\{heads\}}(\langle X|E\rangle) + tails \\ & = & \tau \cdot tails + tails \end{array}$$

Anmerkung: Der Kalkül ACP_τ mit geschützter linearer Rekursion ist korrekt und vollständig in Bezug auf initiale Verzweigungsbisimulation:

$$s = t \;\Leftrightarrow\; s \underline{\leftrightarrow}_{rb} t$$

2.6 Verifikation des Alternierbitprotokolls



Das Alternierbitprotokoll wurde in der Vorlesung F4 eingeführt und als Petrinetz-Modell ausführlicher erläutert. Dies wird hier durch einen Korrektheitsbeweis mit den Mitteln der Prozessalgebra fortgeführt.

Datenelemente d werden von einem Sender über einen störanfälligen Kanal B zu einem Empfänger gesandt. Aufgabe des Protokolls ist es, durch wiederholtes Senden die Störung zu kompensieren. Dazu fügt der Sender den Datenelementen alternativ ein Bit 0 oder 1 hinzu. Wenn der Empfänger das Datenelement korrekt erhalten hat, sendet er das Bit über den (ebenfalls störanfälligen) Kanal D an den Sender als Quittung zurück. Falls die Nachricht gestört war, sendet er jedoch das vorangehende Bit zurück.

Der Sender wiederholt das Senden eines Datenelementes mit Bit b solange bis er eine Quittung b erhält. Dann sendet er das nächste Datenelement mit Bit $1-b$ bis er $1-b$ als Quittung erhält.

Spezifikation des Senders für das Senden mit Bit b:

$$\begin{aligned} S_b &= \sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot T_{db} \\ T_{db} &= (s_{\mathrm{B}}(d,b)+s_{\mathrm{B}}(\bot))\cdot U_{db} \\ U_{db} &= r_{\mathrm{D}}(b)\cdot S_{1-b} \;+\; (r_{\mathrm{D}}(1-b)+r_{\mathrm{D}}(\bot))\cdot T_{db} \end{aligned}$$

Für ein Argument u bedeutet $r_{\mathrm{X}}(u)$ bzw. $s_{\mathrm{X}}(u)$ wieder das Lesen von dem bzw. das Schreiben in den Kanal X.

Spezifikation des Empfängers für das Empfangen mit Bit b:

$$\begin{aligned} R_b &= \sum_{d\in\Delta}\{r_{\mathrm{B}}(d,b)\cdot s_{\mathrm{C}}(d)\cdot Q_b \\ &\quad + r_{\mathrm{B}}(d,1-b)\cdot Q_{1-b}\} + r_{\mathrm{B}}(\bot)\cdot Q_{1-b} \\ Q_b &= (s_{\mathrm{D}}(b)+s_{\mathrm{D}}(\bot))\cdot R_{1-b} \end{aligned}$$

Als externes Verhalten des Alternierbitprotokolls erhalten wir also:

$$\tau_I(\partial_H(R_0\|S_0))$$

Dabei werden durch ∂_H falsche Kommunikationspaare ausgeschlossen, d.h. wir definieren

- $\gamma(s_B(d,b),r_B(d,b)) := c_B(d,b)$
- $\gamma(s_B(\bot),r_B(\bot)) := c_B(\bot)$

- $\gamma(s_D(b), r_D(b)) := c_D(b)$
- $\gamma(s_D(\bot), r_D(\bot)) := c_D(\bot)$

für $d \in \Delta, b \in \{0,1\}$. Dabei ist $\bot$ die gestörte Nachricht. H besteht also aus allen Aktionen, die auf der linken Seite dieser Definition vorkommen. τ_I abstrahiert von den internen Aktionen in $I := \{c_B(d,b), c_D(b) | d \in \Delta, b \in \{0,1\}\} \cup \{c_B(\bot), c_D(\bot)\}$.

Als Korrektheitsbeweis werden wir ableiten:

$$\tau_I(\partial_H(R_0 \| S_0)) = \sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(R_0 \| S_0))$$

Das spezifizierte Protokoll hat damit also das gewünschte Verhalten:

$$r_{\mathrm{A}}(d_0), s_{\mathrm{C}}(d_0), r_{\mathrm{A}}(d_1), s_{\mathrm{C}}(d_1), r_{\mathrm{A}}(d_2), s_{\mathrm{C}}(d_2), \ldots$$

d.h. alle Datenelemente $d_0, d_1, d_2, \ldots$ werden in der richtigen Reihenfolge (und ohne Verlust oder Verdoppelung) übertragen.

Als erster Schritt leitet man unter Benutzung der Axiome M1, RDP, LM4, CM9, CM10, LM3, CM8, A6, A7 ab:

$$\begin{aligned} R_0 \| S_0 \;=\; & \textstyle\sum_{d' \in \Delta} \{ r_{\mathrm{B}}(d', 0) \cdot ((s_{\mathrm{C}}(d') Q_0) \| S_0) \\ & + r_{\mathrm{B}}(d', 1) \cdot (Q_1 \| S_0) \} + r_{\mathrm{B}}(\bot) \cdot (Q_1 \| S_0) \\ & + \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot (T_{d0} \| R_0) \end{aligned}$$

Weiter erhält man mit D4, D1, D2, D5, A6, A7:

$$\partial_H(R_0 \| S_0) \;=\; \textstyle\sum_{d \in \Delta} r_{\mathrm{A}}(d) \cdot \partial_H(T_{d0} \| R_0)$$

Diese Äquivalenz entspricht dem Übergang vom Zustand 1 in den Zustand 2 im Transitionsgraph von $\partial_H(R_0 \| S_0)$ in Abbildung 2.3.

$$\begin{aligned} T_{d0} \| R_0 \;=\; & (s_{\mathrm{B}}(d, 0) + s_{\mathrm{B}}(\bot)) \cdot (U_{d0} \| R_0) \\ & + \textstyle\sum_{d' \in \Delta} \{ r_{\mathrm{B}}(d', 0) \cdot ((s_{\mathrm{C}}(d') Q_0) \| T_{d0}) \\ & + r_{\mathrm{B}}(d', 1) \cdot (Q_1 \| T_{d0}) \} + r_{\mathrm{B}}(\bot) \cdot (Q_1 \| T_{d0}) \\ & + c_{\mathrm{B}}(d, 0) \cdot (U_{d0} \| (s_{\mathrm{C}}(d) Q_0)) + c_{\mathrm{B}}(\bot) \cdot (U_{d0} \| Q_1) \\ \partial_H(T_{d0} \| R_0) \;=\; & c_{\mathrm{B}}(d, 0) \cdot \partial_H(U_{d0} \| (s_{\mathrm{C}}(d) Q_0)) \\ & + c_{\mathrm{B}}(\bot) \cdot \partial_H(U_{d0} \| Q_1) \end{aligned}$$

Diese Äquivalenz entspricht dem Übergang vom Zustand 2 in die Zustände 3 und 4 im Transitionsgraph von Abbildung 2.3.

Entsprechend erhält man für die Übergänge bis zum Zustand 7:

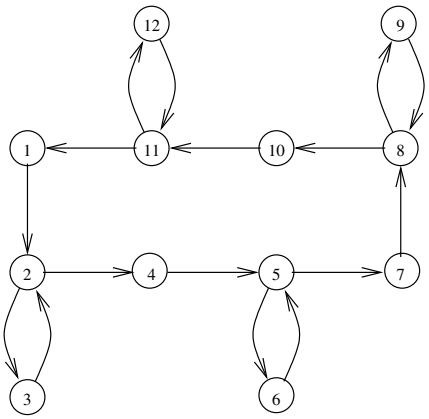


Abbildung 2.3: Transitionsgraph von $\partial_H(R_0 \| S_0)$

$$
\begin{aligned}
U_{d0}\|Q_1 &= r_{\mathrm{D}}(0)\cdot(S_1\|Q_1) \\
&\quad + (r_{\mathrm{D}}(1)+r_{\mathrm{D}}(\bot))\cdot(T_{d0}\|Q_1) \\
&\quad + (s_{\mathrm{D}}(1)+s_{\mathrm{D}}(\bot))\cdot(R_0\|U_{d0}) \\
&\quad + (c_{\mathrm{D}}(1)+c_{\mathrm{D}}(\bot))\cdot(T_{d0}\|R_0) \\
\partial_H(U_{d0}\|Q_1) &= (c_{\mathrm{D}}(1)+c_{\mathrm{D}}(\bot))\cdot\partial_H(T_{d0}\|R_0)
\end{aligned}
$$

$$
\begin{aligned}
U_{d0}\|(s_{\mathrm{C}}(d)Q_0) &= r_{\mathrm{D}}(0)\cdot(S_1\|(s_{\mathrm{C}}(d)Q_0)) \\
&\quad + (r_{\mathrm{D}}(1)+r_{\mathrm{D}}(\bot))\cdot(T_{d0}\|(s_{\mathrm{C}}(d)Q_0)) \\
&\quad + s_{\mathrm{C}}(d)\cdot(Q_0\|U_{d0}) \\
\partial_H(U_{d0}\|(s_{\mathrm{C}}(d)Q_0)) &= s_{\mathrm{C}}(d)\cdot\partial_H(Q_0\|U_{d0})
\end{aligned}
$$

$$
\begin{aligned}
Q_0\|U_{d0} &= (s_{\mathrm{D}}(0)+s_{\mathrm{D}}(\bot))\cdot(R_1\|U_{d0}) \\
&\quad + r_{\mathrm{D}}(0)\cdot(S_1\|Q_0)+(r_{\mathrm{D}}(1)+r_{\mathrm{D}}(\bot))\cdot(T_{d0}\|Q_0) \\
&\quad + c_{\mathrm{D}}(0)\cdot(R_1\|S_1)+c_{\mathrm{D}}(\bot)\cdot(R_1\|T_{d0}) \\
\partial_H(Q_0\|U_{d0}) &= c_{\mathrm{D}}(0)\cdot\partial_H(R_1\|S_1) \\
&\quad + c_{\mathrm{D}}(\bot)\cdot\partial_H(R_1\|T_{d0})
\end{aligned}
$$

$$
\begin{aligned}
R_1\|T_{d0} &= \sum_{d'\in\Delta}\{r_{\mathrm{B}}(d',1)\cdot((s_{\mathrm{C}}(d')Q_1)\|T_{d0}) \\
&\quad + r_{\mathrm{B}}(d',0)\cdot(Q_0\|T_{d0})\}+r_{\mathrm{B}}(\bot)\cdot(Q_0\|T_{d0}) \\
&\quad + (s_{\mathrm{B}}(d,0)+s_{\mathrm{B}}(\bot))\cdot(U_{d0}\|R_1) \\
&\quad + (c_{\mathrm{B}}(d,0)+c_{\mathrm{B}}(\bot))\cdot(Q_0\|U_{d0}) \\
\partial_H(R_1\|T_{d0}) &= (c_{\mathrm{B}}(d,0)+c_{\mathrm{B}}(\bot))\cdot\partial_H(Q_0\|U_{d0})
\end{aligned}
$$

Dann erhält man für die Übergänge von Zustand 7 bis zum Zustand 1:

$$
\begin{aligned}
\partial_H(R_1\|S_1) &= \textstyle\sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot\partial_H(T_{d1}\|R_1)\\
\partial_H(T_{d1}\|R_1) &= c_{\mathrm{B}}(d,1)\cdot\partial_H(U_{d1}\|(s_{\mathrm{C}}(d)\cdot Q_1))\\
&\quad + c_{\mathrm{B}}(\bot)\cdot\partial_H(U_{d1}\|Q_0)\\
\partial_H(U_{d1}\|Q_0) &= (c_{\mathrm{D}}(0)+c_{\mathrm{D}}(\bot))\cdot\partial_H(T_{d1}\|R_1)\\
\partial_H(U_{d1}\|(s_{\mathrm{C}}(d)Q_1)) &= s_{\mathrm{C}}(d)\cdot\partial_H(Q_1\|U_{d1})\\
\partial_H(Q_1\|U_{d1}) &= c_{\mathrm{D}}(1)\cdot\partial_H(R_0\|S_0)\\
&\quad + c_{\mathrm{D}}(\bot)\cdot\partial_H(R_0\|T_{d1})\\
\partial_H(R_0\|T_{d1}) &= (c_{\mathrm{B}}(d,1)+c_{\mathrm{B}}(\bot))\cdot\partial_H(Q_1\|U_{d1})
\end{aligned}
$$

Insgesamt ergeben sich 12 Gleichungen (die den 12 Zuständen entsprechen) und mit RSP:

$$\partial_H(R_0\|S_0) \;=\; \langle X_1|E\rangle$$

wobei E die folgende lineare rekursive Specification ist.

$$
\begin{aligned}
\{\quad X_1 &= \textstyle\sum_{d'\in\Delta} r_{\mathrm{A}}(d')\cdot X_{2d'}\\
X_{2d} &= c_{\mathrm{B}}(d,0)\cdot X_{4d} + c_{\mathrm{B}}(\bot)\cdot X_{3d}\\
X_{3d} &= (c_{\mathrm{D}}(1)+c_{\mathrm{D}}(\bot))\cdot X_{2d}\\
X_{4d} &= s_{\mathrm{C}}(d)\cdot X_{5d}\\
X_{5d} &= c_{\mathrm{D}}(0)\cdot Y_1 + c_{\mathrm{D}}(\bot)\cdot X_{6d}\\
X_{6d} &= (c_{\mathrm{B}}(d,0)+c_{\mathrm{B}}(\bot))\cdot X_{5d}\\
Y_1 &= \textstyle\sum_{d'\in\Delta} r_{\mathrm{A}}(d')\cdot Y_{2d'}\\
Y_{2d} &= c_{\mathrm{B}}(d,1)\cdot Y_{4d} + c_{\mathrm{B}}(\bot)\cdot Y_{3d}\\
Y_{3d} &= (c_{\mathrm{D}}(0)+c_{\mathrm{D}}(\bot))\cdot Y_{2d}\\
Y_{4d} &= s_{\mathrm{C}}(d)\cdot Y_{5d}\\
Y_{5d} &= c_{\mathrm{D}}(1)\cdot X_1 + c_{\mathrm{D}}(\bot)\cdot Y_{6d}\\
Y_{6d} &= (c_{\mathrm{B}}(d,1)+c_{\mathrm{B}}(\bot))\cdot Y_{5d}\\
|\quad d\in\Delta\quad\}
\end{aligned}
$$

Durch die Anwendung von τ_I auf $\langle X_1|E\rangle$ werden die Kommunikationsschleifen zu τ-Schleifen, die durch CFAR eliminiert werden.

Beispielsweise bilden X_{2d} und X_{3d} eine τ-Gruppe I mit Ausgang $c_{\mathrm{B}}(d,0)\cdot X_{4d}$, also:

$$
\begin{aligned}
& r_{\mathrm{A}}(d)\cdot\tau_I(\langle X_{2d}|E\rangle)\\
=\;& r_{\mathrm{A}}(d)\cdot\tau_I(c_{\mathrm{B}}(d,0)\cdot\langle X_{4d}|E\rangle)\\
=\;& r_{\mathrm{A}}(d)\cdot\tau\cdot\tau_I(\langle X_{4d}|E\rangle)\\
=\;& r_{\mathrm{A}}(d)\cdot\tau_I(\langle X_{4d}|E\rangle)
\end{aligned}
$$

Entsprechend:

$$\begin{aligned} s_{\mathrm{C}}(d) \cdot \tau_I(\langle X_{5d}|E\rangle) &= s_{\mathrm{C}}(d) \cdot \tau_I(\langle Y_1|E\rangle) \\ r_{\mathrm{A}}(d) \cdot \tau_I(\langle Y_{2d}|E\rangle) &= r_{\mathrm{A}}(d) \cdot \tau_I(\langle Y_{4d}|E\rangle) \\ s_{\mathrm{C}}(d) \cdot \tau_I(\langle Y_{5d}|E\rangle) &= s_{\mathrm{C}}(d) \cdot \tau_I(\langle X_1|E\rangle) \end{aligned}$$

$$\begin{aligned} \tau_I(\langle X_1|E\rangle) &= \textstyle\sum_{d\in\Delta} r_{\mathrm{A}}(d) \cdot \tau_I(\langle X_{2d}|E\rangle) \\ &= \textstyle\sum_{d\in\Delta} r_{\mathrm{A}}(d) \cdot \tau_I(\langle X_{4d}|E\rangle) \\ &= \textstyle\sum_{d\in\Delta} r_{\mathrm{A}}(d) \cdot s_{\mathrm{C}}(d) \cdot \tau_I(\langle X_{5d}|E\rangle) \\ &= \textstyle\sum_{d\in\Delta} r_{\mathrm{A}}(d) \cdot s_{\mathrm{C}}(d) \cdot \tau_I(\langle Y_1|E\rangle) \end{aligned}$$

$$\tau_I(\langle Y_1|E\rangle) = \textstyle\sum_{d\in\Delta} r_{\mathrm{A}}(d) \cdot s_{\mathrm{C}}(d) \cdot \tau_I(\langle X_1|E\rangle)$$

Mit RSP folgt $\tau_I(\langle X_1|E\rangle) = \langle Z \mid Z = r_{\mathrm{A}}(d) \cdot s_{\mathrm{C}}(d) \cdot Z\rangle$.

Damit ist das oben angesprochene Ziel des Beweises erreicht:

$$\tau_I(\partial_H(R_0 \| S_0)) = \sum_{d\in\Delta} r_{\mathrm{A}}(d) \cdot s_{\mathrm{C}}(d) \cdot \tau_I(\partial_H(R_0 \| S_0))$$

Aufgabe 2.47

a) Ersetzen Sie im Beweis des Alternierbitprotokolls die Spezifikation von U_{db} durch

$$U_{db} = (r_D(b) + r_D(\bot)) \cdot S_{1-b} + r_D(1-b) \cdot T_{db}.$$

Interpretieren Sie dies inhaltlich und formal für den Beweis.

b) Modellieren Sie im Modell des Alternierbitprotokolls die (gestörten) Kanäle als eigene Funktionseinheiten K und L, an die - im Gegensatz zur behandelten Form - die Daten ungestört übergeben werden. Formulieren Sie die Spezifikation des geänderten Modells.

Kapitel 3

Harel-Graphen (statecharts)

3.1 Einleitung

Im vorigen Jahrzehnt entwickelte David Harel, Professor für Angewandte Mathematik und Informatik am Weizmann Institute of Science in Israel, im Rahmen einer Beratertätigkeit für die Forschungs- und Entwicklungsabteilung der Israel Aircraft Industries (IAI) eine grafische Notation für die Modellierung des Verhaltens komplexer Systeme mit Nebenläufigkeit, den Formalismus der *Statecharts*. Harel hatte damals die Aufgabe, gemeinsam mit einer Gruppe von Ingenieuren die Spezifikation für ein Flugelektronik-System eines israelischen Kampfflugzeuges zu erarbeiten.

Obwohl es bereits zahlreiche Bücher und Artikel über Methoden für die Spezifikation und den Entwurf komplexer Systeme gab, erwies sich die gestellte Aufgabe als schwierig. Den Grund für diese Schwierigkeit sah Harel darin, daß es zwar für viele Arten von Systemen eine grundlegende Entwurfsphilosophie gab, sehr komplexe Systeme, die aus vielen nebenläufigen Hard- und Software-Komponenten bestehen, den bekannten Methoden jedoch kaum zugänglich waren. Gemeinsam mit Amir Pnueli, einem Experten für Systemspezifikation und -verifikation, der ebenfalls am Weizmann Institute of Science tätig ist, führt Harel diese Erkenntnis auf einen wesentlichen Unterschied zwischen zwei Arten von Systemen zurück, den *transformierenden* und den *reaktiven* Systemen.

Unter einem transformierenden System verstehen Harel und Pnueli ein System, welches Eingaben akzeptiert, daraufhin Transformationen durchführt und schließlich Ausgaben produziert (siehe Abb. 3.1(a)). Diese Definition schließt auch solche Systeme ein, die während ihres Einsatzes nach weiteren Eingaben verlangen oder zusätzliche Ausgaben produzieren; entscheidend ist der Umstand, daß ein transformierendes System eine Ein-/Ausgabe-Operation durchführt. Ein reaktives System wird hingegen wiederholt durch seine Umgebung zu Aktivitäten veranlaßt und reagiert ständig auf externe Ereignisse, ist also ereignisgesteuert (siehe Abb. 3.1(b)). Es berechnet i. a. keine Funktion und führt auch keine solche aus, sondern erhält in gewisser Hinsicht eine bestimmte Beziehung zu seiner Umgebung aufrecht. Beispiele für reaktive Systeme

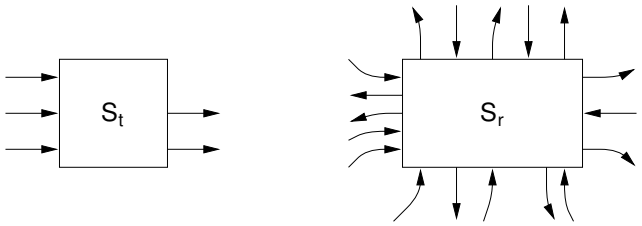


Abb. 3.1: Ein transformierendes System in (a)
sowie ein reaktives System in (b)

lassen sich leicht und zahlreich finden; so sind neben Flugelektronik-Systemen u. a. auch Armbanduhren, Mikrowellengeräte, viele moderne medizinische Geräte, Telekommunikationsanlagen, Betriebssysteme sowie die Mensch-Maschine-Schnittstellen zahlreicher Softwareprodukte reaktive Systeme.

Harel und Pnueli machen deutlich, daß sich ihre Sicht des Begriffs „System“ nicht auf softwarebasierte, hardwarebasierte oder eingebettete Systeme beschränkt. Die von ihnen verwendete Terminologie ist sehr allgemein, und die Form, in der ein System schließlich implementiert wird, ist zunächst nicht von Bedeutung.

Es stellt sich die Frage, warum Harel und Pnueli die Einteilung in transformierende und reaktive Systeme als die grundlegendste aller Dichotomien ansehen. In [**?**] werden hierfür im wesentlichen zwei Argumente genannt. Zum einen ist diese Unterscheidung orthogonal zu anderen Eigenschaften: sowohl transformierende als auch reaktive Systeme können deterministisch oder nichtdeterministisch, terminierend oder nicht terminierend, synchron oder asynchron sowie sequentiell oder nebenläufig sein. Zum anderen liegt in der Natur der reaktiven Systeme eine besondere Problematik, die durch die Notwendigkeit einer Beschreibung des Systemverhaltens zutage tritt. Dieser zweite Punkt soll im folgenden näher betrachtet werden.

Nach Ansicht Harels und Pnuelis können der Entwurf und die Konstruktion eines Systems nicht ohne ein klares Verständnis des intendierten Systemverhaltens durchgeführt werden. Dieses Verständnis ist für jedes Entwicklungsstadium eines Systems wichtig. Ein Entwicklungsstadium ist, so Harel und Pnueli, durch einen bestimmten Stand der Implementation und einen bestimmten Detaillierungsgrad der zum implementierten System gehörigen Verhaltensbeschreibung gekennzeichnet. Durch Verfeinerungen der Implementation und der Verhaltensbeschreibung gelangt man zu weiteren Stadien der Entwicklung. Für jedes Stadium muß eine Schnittstellenbeschreibung erstellt werden, die die Interaktion des Systems mit seiner Umgebung über Ein- und Ausgabekanäle darstellt. Diese Beschreibung kann für unsere Zwecke als eine Liste von Ereignissen bestimmter Granularität betrachtet werden. Die zugehörige Verhaltensbeschreibung spiegelt dann das Verhalten des Systems unter Verwendung der in dieser Liste aufgeführten Ereignisse wider.

Die Probleme, vor die reaktive Systeme ihre Entwickler stellen, treten im Verlauf des Entwurfs-

und Konstruktionsprozesses immer deutlicher zutage: Es muß nicht nur die Konsistenz zweier Verhaltensbeschreibungen eines reaktiven Systems mit unterschiedlichen Detaillierungsgraden überprüft, sondern es muß vor allem eine Möglichkeit gefunden werden, das eventuell äußerst komplexe Verhalten eines reaktiven Systems in kleinere Einheiten zu gliedern. Während zum Erscheinungszeitpunkt von [**?**] für transformierende Systeme geeignete Methoden zur Dekomposition aus dem Bereich der Strukturierten Analyse weit verbreitet waren, gab es für reaktive Systeme nahezu keine entsprechenden Vorgehensweisen. Dennoch war der Bedarf an derartigen Methoden unverkennbar, denn jedes komplexe reaktive System besteht aus reaktiven Subsystemen, deren Verhaltenscharakteristika das Verhalten anderer Subsysteme und des gesamten Systems beeinflussen.

Man kann sich an diesem Punkt die Frage stellen, ob es damals sinnvoll gewesen wäre, reaktive als transformierende Systeme aufzufassen, indem man die Sequenz aller das System von außen beeinflussenden Ereignisse als Eingabe betrachtet, die vom System in die zugehörige Sequenz der ausgesandten Ereignisse überführt wird. Das Argument, das gegen diese Idee spricht, besteht darin, daß die auf ein reaktives System Einfluß nehmenden Ereignisse von zu früheren Zeitpunkten erfolgten Reaktionen des Systems abhängen können; somit ist die Beschreibung einer Relation zwischen Ein- und Ausgabesequenzen von Ereignissen nicht ausreichend (siehe [**?**]).

Die bisherige Diskussion macht deutlich, daß eine Methode wünschenswert ist, die es ermöglicht, das Verhalten eines reaktiven Systems in kleinere Einheiten zu gliedern. Harel und Pnueli formulieren in [**?**] folgende Anforderungen, die eine derartige Methode erfüllen sollte:

- Sie sollte Beschreibungen zur Verfügung stellen, die wohlstrukturiert, prägnant, unzweideutig, lesbar und leicht zu verstehen sind.
- Sie sollte ausschließlich beschreibend sein und keine Abhängigkeiten von Implementationsaspekten aufweisen oder diese zumindest minimieren.

Der von Harel und Pnueli vorgeschlagenen Methode liegt der grafische Formalismus der *Statecharts* zugrunde, welcher diesen Forderungen genügt und im folgenden vorgestellt werden soll.

3.2 Der graphische Formalismus der Statecharts

Statecharts stellen eine Erweiterung herkömmlicher endlicher Automaten sowie ihrer Zustandsdiagramme dar und ermöglichen die Beschreibung des Verhaltens komplexer reaktiver Systeme in kompakter, ausdrucksmächtiger Form. Endliche Automaten werden in nahezu allen Teilgebieten der Informatik eingesetzt und sind daher von besonderem Interesse. Sie finden nicht nur bei der Anwendungsentwicklung, sondern z. B. auch in den Bereichen Betriebssysteme (siehe z. B. [**?**]), Kommunikationsprotokolle (siehe z. B. [**?**]) und Rechnerarchitekturen (siehe z. B. [**?**]) Verwendung.

Schon seit langem werden endliche Automaten für Verhaltensbeschreibungen eingesetzt, um den Entwurf von Systemen zu erleichtern und die getroffenen Entscheidungen zu dokumentieren (siehe z. B. [**?**]). Trotz ihrer breiten Akzeptanz weisen endliche Automaten jedoch erhebliche Nachteile auf. Martin und McClure schreiben den Zustandsdiagrammen endlicher Automaten in [**?**] zwar z. B. zu, eher problem- als programmbezogen zu sein, nennen jedoch auch eine Reihe von negativen Eigenschaften. So sind die Zustandsdiagramme endlicher Automaten u. a. oftmals

- nicht leicht zu lesen,
- schwer zu zeichnen und zu ändern,
- nur schwer zugänglich,
- nicht für schrittweise Verfeinerungen geeignet sowie
- als Darstellungsmittel für komplexe Spezifikationen, die einen hohen Grad an Fehlerfreiheit erfordern, nicht zu verwenden.

Harel, Pnueli sowie zwei Koautoren vertreten in [**?**] die Auffassung, daß viele, die sich mit dem Entwurf komplexer Anwendungen beschäftigen, es nahezu aufgegeben hätten, herkömmliche endliche Automaten sowie ihre Zustandsdiagramme einzusetzen, und nennen hierfür vier konkrete Gründe:

1. Zustandsdiagramme sind „flach“. Sie sehen keine Konzepte für Tiefe, Hierarchie oder Modularität vor und unterstützen daher keine schrittweisen Top-Down- oder Bottom-Up-Entwicklungen.
2. Ein Ereignis, das einen identischen Übergang von einer großen Anzahl von Zuständen bewirkt, wie z. B. ein Interrupt hoher Abstraktion, muß jedem dieser Zustände separat zugeordnet werden, so daß sich im zugehörigen Zustandsdiagramm unnötig viele Pfeile ergeben.
3. Zustandsdiagramme sind im Hinblick auf die Anzahl der Zustände unökonomisch und daher häufig nahezu unmöglich zu erstellen. Wächst das zu beschreibende System in seiner Größe linear, so wächst die Anzahl der Zustände exponentiell, da der Formalismus herkömmlicher endlicher Automaten die Entwickler dazu zwingt, alle Systemzustände explizit darzustellen.
4. Zustandsdiagramme sind ihrer Natur nach sequentiell und bieten keine Möglichkeit zur Darstellung von Nebenläufigkeit.

Der von Harel entwickelte Statechart-Formalismus überwindet diese Unzulänglichkeiten, wobei er das visuelle Erscheinungsbild der Zustandsdiagramme konventioneller endlicher Automaten in vielfacher Hinsicht verbessert.

Die folgenden Unterabschnitte bilden eine in sich geschlossene Darstellung der wesentlichen Charakteristika von Statecharts sowie einiger denkbarer Erweiterungen. Dabei orientiert sich der Text weitgehend an der Ausarbeitung [**?**], die sich ihrerseits auf die für den Bereich als grundlegend geltende Arbeit [**?**] stützt.

Die Bildung von Zustandshierarchien

Eine wichtige Eigenschaft von Statecharts ist die Möglichkeit, Zustandshierarchien bilden zu können. Zustandshierarchien erlauben verschiedene Grade der Detaillierung und lassen in bestimmter Hinsicht zusammengehörige Zustände leicht als solche erkennen. Mit ihrer Hilfe kann somit eine Strukturierung des Zustandsraumes erreicht werden, die für ein klares Verständnis komplexer Verhaltensbeschreibungen unerläßlich ist.

Auf jeder Hierarchieebene werden Zustände durch abgerundete Rechtecke dargestellt, welche mit einem Bezeichner versehen sind (siehe Abb. 3.2(a)). Die Hierarchie-Relation auf der Menge der Zustände wird durch grafischen Einschluß ausgedrückt. Somit besagt Abb. 3.2(b), daß Zustand A ein Subzustand des Zustands E ist.

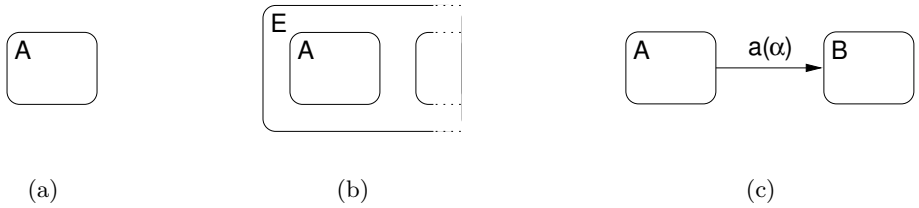
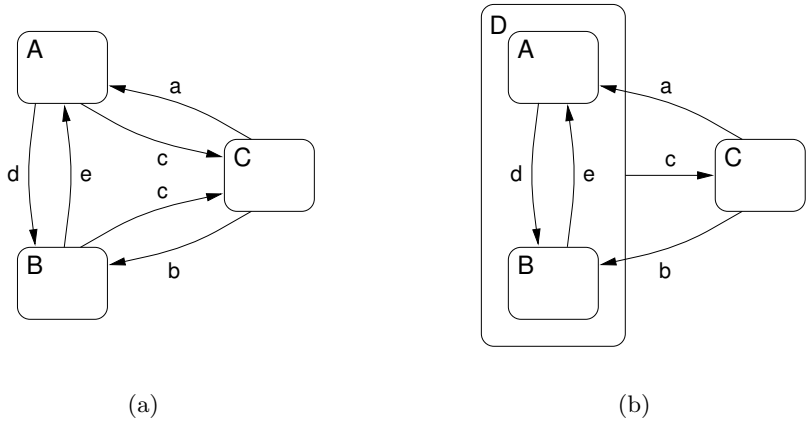


Abb. 3.2: Ein Zustand in (a), die Darstellung eines Teils einer Hierarchie-Relation in (b) und ein Zustandsübergang in (c)

Mögliche Zustandsübergänge werden durch beschriftete Pfeile dargestellt, die Zustände beliebiger Hierarchieebenen miteinander verbinden können. Die Beschriftung eines Pfeils besteht aus einem Bezeichner für das Ereignis, das den tatsächlichen Übergang bewirkt. Zustandsübergänge können von einer Bedingung abhängen; in einem solchen Fall wird der entsprechende Pfeil zusätzlich mit einem Bezeichner für diese Bedingung versehen, welcher in runden Klammern hinter dem Ereignisbezeichner aufgeführt wird. Abb. 3.2(c) zeigt ein Beispiel für einen Zustandsübergang. In der einführenden Darstellung folgt die Beschriftung der Zustände sowie der die Zustandsübergänge repräsentierenden Pfeile soweit wie möglich einem einheitlichen Schema; fast immer bezeichnen wir Zustände mit lateinischen Großbuchstaben, Ereignisse mit lateinischen Kleinbuchstaben und Bedingungen mit griechischen Kleinbuchstaben.

Betrachtet man Abb. 3.3(a), so erkennt man, daß ein Zustandsübergang von A nach C oder von B nach C stattfinden kann, wenn das Ereignis c eintritt. Die Zustände A und B weisen also eine gemeinsame Eigenschaft auf, und es ist möglich, sie im Sinne einer Bottom-Up-Entwicklung in einem neuen Zustand D zusammenzufassen, wie es in Abb. 3.3(b) gezeigt wird.

Der Zustand D hat dann die Bedeutung eines *Exklusiv-Oder* (XOR): Befindet sich das System im Zustand D, so befindet es sich in einem der Zustände A und B, aber nicht in beiden. Somit



(a) (b)

Abb. 3.3: Ein einfacher Statechart in (a)
sowie eine Abstraktion in (b)

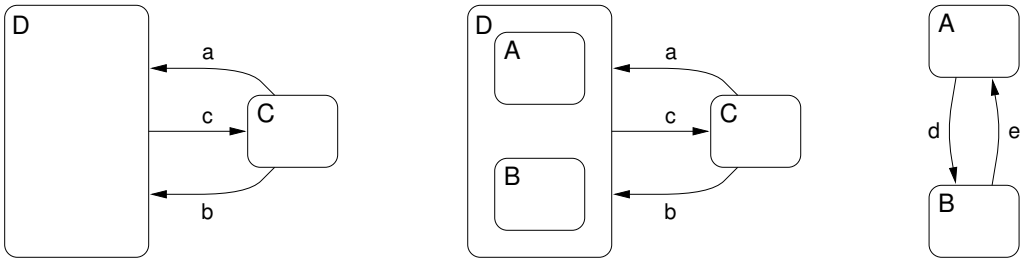
läßt sich D als *Abstraktion* von A und B auffassen. Man beachte, daß durch die Einführung des Zustands D nicht nur eine Zustandshierarchie gebildet, sondern darüber hinaus die Anzahl der Pfeile im Diagramm reduziert wird. Eine in großem Umfang erfolgende Anwendung des obengenannten Prinzips hat daher zur Folge, daß die Nachteile endlicher Automaten, die in der Aufzählung auf Seite 42 unter den ersten beiden Punkten genannt sind, überwunden werden.

Geht man von Abb. 3.4(a) aus, so kann man sich D als aus A und B bestehend vorstellen und im Sinne eines Top-Down-Entwurfs zu Abb. 3.4(b) gelangen; in diesem Fall kann man von einer *Verfeinerung* sprechen. Hierbei ist zu bedenken, daß die Pfeile mit den Beschriftungen a und b nicht ausreichend spezifiziert sind. Verlängert man diese Pfeile bis zu den Zuständen A und B und verfeinert man D mit Hilfe von Abb. 3.4(c) weiter, so entsteht auf diese Weise der Statechart aus Abb. 3.3(b), welcher zuvor das Ergebnis eines Abstraktionsvorganges war. Der Folgezustand, der sich in Abb. 3.4(b) ergibt, wenn Zustand C verlassen wird, kann auch durch einen weiter unten beschriebenen Mechanismen (Default-Zustand, bedingter Eintritt, selektiver Eintritt, History-Funktion) erfolgen.

Orthogonalität

Neben der oben vorgestellten XOR-Dekomposition von Zuständen stellt der Formalismus der Statecharts auch die Möglichkeit einer AND-Dekomposition zur Verfügung. Durch sie wird festgelegt, daß alle Subzustände eines Zustandes *gleichzeitig* eingenommen werden, wenn dieser betreten wird. Die Subzustände eines mittels der AND-Dekomposition verfeinerten Zustandes nennt man gemäß den Ausführungen in [**?**] *orthogonal*; dementsprechend bezeichnet man die der AND-Dekomposition zugrundeliegende Eigenschaft des Statechart-Formalismus als *Orthogonalität*. Im Falle einer XOR-Dekomposition spricht man von (*einander*) *ausschließenden* Subzuständen. Orthogonalität kann – wie auch die XOR-Dekomposition – auf jeder Hierarchieebene eingesetzt werden, um Zustände genauer zu beschreiben.

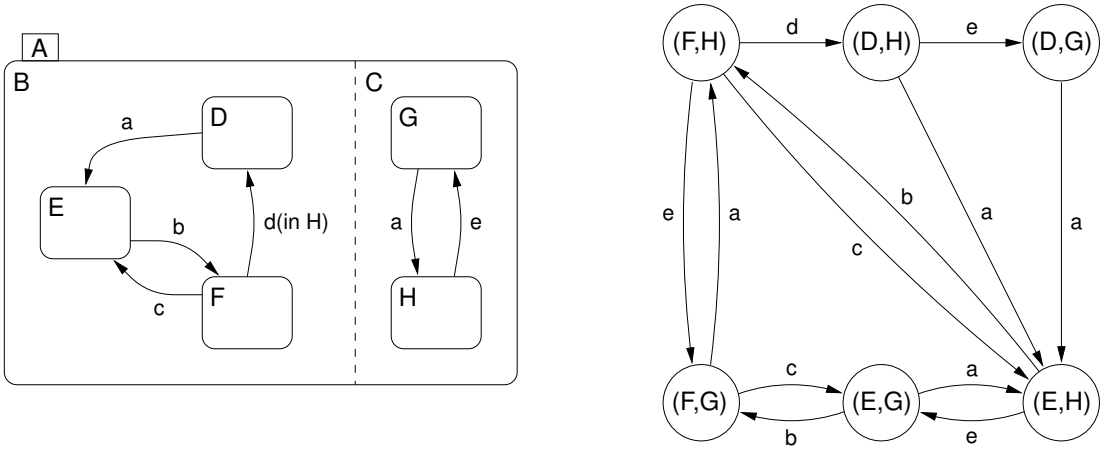
Die grafische Repräsentation von Orthogonalität erfolgt mit Hilfe gestrichelter Linien, die die



(a) (b) (c)

Abb. 3.4: Ein einfacher Statechart in (a) sowie eine denkbare Verfeinerung des Zustands D in (b); (c) zeigt eine weitere Verfeinerung von D

einzelnen AND-Komponenten voneinander trennen. Abb. 3.5(a) zeigt das Beispiel eines Zustandes A, der aus zwei orthogonalen Komponenten B und C besteht. Geht das System in den Zustand A über, so bedeutet dies, daß die Zustände B und C gleichzeitig betreten werden. Da B und C mittels der XOR-Dekomposition verfeinerte Zustände sind, ist dies gleichbedeutend damit, daß genau eines der Zustandspaare (D, G), (D, H), (E, G), (E, H), (F, G) und (F, H) eingenommen wird.



(a) (b)

Abb. 3.5: Ein Beispiel für Orthogonalität in (a); (b) zeigt ein zum Statechart in (a) äquivalentes Zustandsdiagramm eines herkömmlichen endlichen Automaten

Im vorliegenden Fall ist das bei einem Zustandsübergang nach A einzunehmende Zustandspaar nicht festgelegt, und somit ist die durch den Statechart in Abb. 3.5(a) gegebene Spezifikation eines denkbaren Verhaltens nicht ausreichend. Die Möglichkeiten, das initiale Zustandspaar

zu bestimmen, sollen an dieser Stelle jedoch nicht näher betrachtet werden, da sie Teil der allgemeinen Darstellung des Abschnittes „Einen Zustand einnehmen und verlassen“ sind.

Ist im Beispiel von Abb. 3.5(a) (D, G) der aktuelle Zustand des Systems, so überführt das Ereignis a das System in den Zustand (E, H), wobei die beiden Zustandswechsel zu E und H gleichzeitig erfolgen. Auf diese Weise ermöglicht Orthogonalität eine bestimmte Form der *Synchronisation*. Tritt hingegen im Systemzustand (D, H) das Ereignis e auf, so erfolgt in C ein Zustandswechsel von H nach G, während die Komponente B keiner Veränderung unterliegt; es besteht aufgrund der Orthogonalität eine gewisse *Unabhängigkeit*, denn der Zustandswechsel innerhalb von C wird nicht durch den aktuellen Zustand in B beeinflußt.

Somit verdeutlicht das Beispiel in Abb. 3.5(a), daß mit Hilfe orthogonaler Zustände die den herkömmlichen Zustandsdiagrammen innewohnende Beschränkung auf Sequentialität (siehe Punkt 4 in der Aufzählung der Nachteile endlicher Automaten auf S. 42) aufgehoben und Nebenläufigkeit dargestellt werden kann. Beachtenswert ist in diesem Zusammenhang die in Abb. 3.5(a) neben dem Ereignis d aufgeführte Bedingung „in H“, welche aufzeigt, daß orthogonale Zustände aufeinander Bezug nehmen können.

Abb. 3.5(b) zeigt ein zum Statechart in Abb. 3.5(a) äquivalentes Zustandsdiagramm eines herkömmlichen endlichen Automaten und deutet einen besonders bedeutsamen Aspekt an: Orthogonalität kann zu einer eventuell drastischen Verkleinerung der Zustandsmenge führen und somit die unter Punkt 3 auf Seite 42 genannten Nachteile der Zustandsdiagramme endlicher Automaten überwinden. Man denke hier z. B. an zwei orthogonale Komponenten mit jeweils eintausend Subzuständen, welche in einem konventionellen Zustandsdiagramm mit bis zu einer Million Zuständen resultieren würden.

Einen Zustand einnehmen und verlassen

In den vorangegangenen Abschnitten wurden mit der XOR- und der AND-Dekomposition zwei wesentliche Eigenschaften von Statecharts vorgestellt. Weitgehend offen geblieben ist bisher die Frage, nach welchen Regeln die einzelnen Zustände eines Statecharts, in dem diese beiden Prinzipien Anwendung finden, eingenommen und verlassen werden sollen.

So ist z. B. im Statechart in Abb. 3.6(a) nicht festgelegt, welcher der beiden Zustände B und C bei einem durch das Ereignis a ausgelösten Übergang von A nach D eingenommen werden soll. Eine naheliegende Lösung dieses Problems besteht darin, den entsprechenden Pfeil zu verlängern; auf diese Weise wird in Abb. 3.6(b) der Zustand B als nächster Zustand gekennzeichnet.

Harel beschreibt in [**?**] weitere Möglichkeiten, den Folgezustand beim Übergang in einen mittels der XOR-Dekomposition verfeinerten Zustand festzulegen:

- Es kann ein *Default-Zustand* unter den einander ausschließenden Subzuständen bestimmt werden, der eingenommen wird, sofern kein anderer Zustand durch einen Pfeil explizit als Folgezustand gekennzeichnet wird.

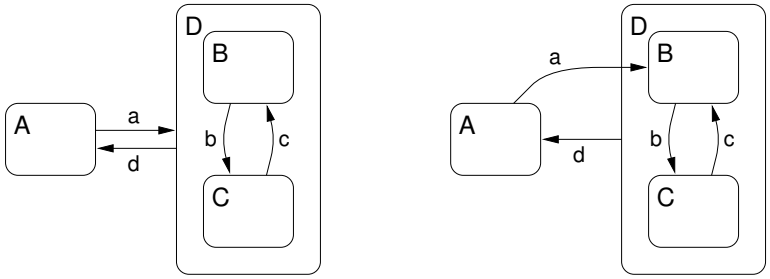


Abb. 3.6: Ein unterspezifizierter Übergang in einen verfeinerten Zustand in (a) sowie eine denkbare Lösung in (b)

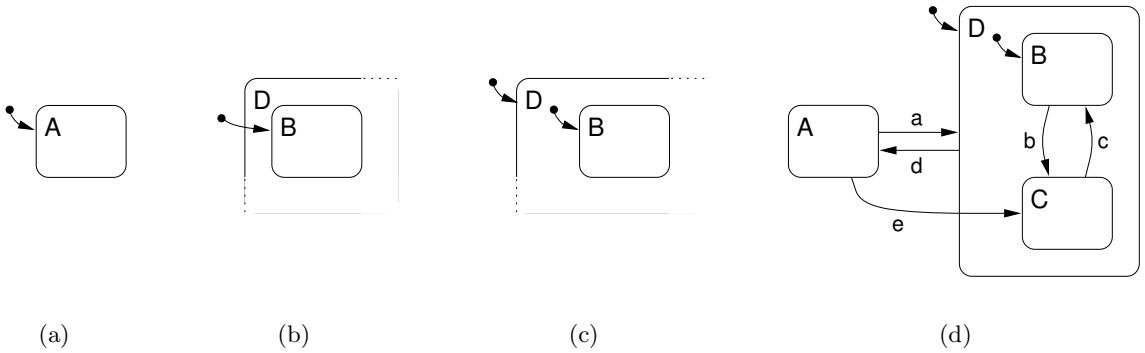
- Mit Hilfe einer *History-Funktion* kann derjenige Zustand zum Folgezustand erklärt werden, in dem sich das System befand, als der unmittelbar übergeordnete Zustand zuletzt verlassen wurde.
- *Selektive* und *bedingte Eintritte* können verwendet werden, wenn es besondere Beziehungen zwischen Zuständen und Ereignissen oder zwischen Zuständen und Bedingungen gibt (s. u.).

Erklärt man einen Zustand A aus einer Menge von Subzuständen, die alle der gleichen Hierarchieebene zuzuordnen sind, zum Default-Zustand, so bedeutet dies, daß A eingenommen wird, wenn der übergeordnete Zustand betreten und nicht explizit ein anderer Folgezustand festgelegt wird. Die grafische Kennzeichnung eines Zustandes als Default-Zustand erfolgt mit Hilfe eines kleinen unbeschrifteten Pfeiles; Abb. 3.7(a) zeigt ein Beispiel. Eine besondere Situation ergibt sich auf der obersten Hierarchieebene: Dort stellt ein Default-Zustand den initialen Zustand eines Systems dar; die Verwandtschaft der Default-Zustände mit den Startzuständen herkömmlicher endlicher Automaten wird hier besonders deutlich.

Da jeder Zustand unabhängig von der Hierarchieebene, zu der er gehört, ein Startzustand des durch den Statechart beschriebenen Systems sein kann, kann in dem Beispiel aus Abb. 3.6(a) auch der Zustand B der initiale Zustand sein. Für diesen Fall sowie allgemein für den Übergang in einen mittels der XOR-Dekomposition verfeinerten Zustand sind zwei Notationen denkbar, die in Abb. 3.7(b) und Abb. 3.7(c) dargestellt sind. Wir schlagen vor, stets die in Abb. 3.7(c) gezeigte Darstellung zu verwenden, da diese es einfacher macht, zusätzliche Verfeinerungen ein- und auszublenden.

Abb. 3.7(d) zeigt eine Ergänzung des Beispiels aus Abb. 3.6(a). Zusätzlich wurde eine mit dem Ereignis e beschriftete Kante eingeführt, die verdeutlicht, wie die mittels eines Default-Zustandes getroffene Entscheidung für einen Folgezustand aufgehoben werden kann.

Eine weitere Möglichkeit, aus einer Gruppe von Zuständen den Folgezustand zu bestimmen,

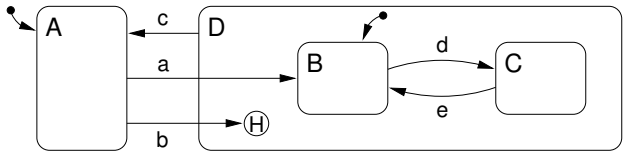


(a) (b) (c) (d)

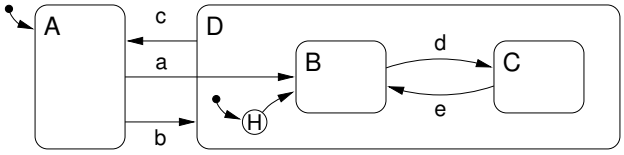
Abb. 3.7: Default-Zustand in (a); Alternativen für die Kennzeichnung eines Default-Zustandes einer niedrigeren Hierarchieebene in (b) und (c); Ergänzung des Beispiels aus Abb. 3.6 in (d)

stellt die History-Funktion dar. Der Einsatz der History-Funktion führt dazu, daß derjenige Zustand eingenommen wird, welcher innerhalb der Zustandsgruppe der zuletzt besuchte ist.

Diese Form des Zustandsübergangs wird im Statechart durch das Symbol „Ⓗ“ dargestellt. Abb. 3.8(a) zeigt ein Beispiel: Befindet sich das zugehörige System im Zustand A, so überführt das Ereignis a das System in den Zustand B, während im Falle des Ereignisses b der Folgezustand anhand der „System-Geschichte“ bestimmt wird. Abb. 3.8(b) stellt eine alternative Darstellung vor.



(a)



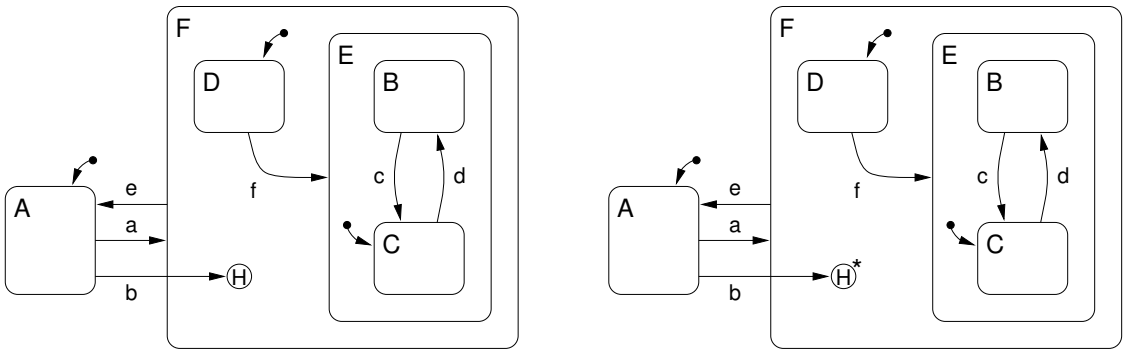
(b)

Abb. 3.8: Zwei Darstellungsformen für die History-Funktion

Zu beachten ist, daß durch das Symbol „Ⓗ“ nur eine Anwendung der History-Funktion auf derjenigen Zustandsebene ausgedrückt wird, auf der es auftritt. Es ist jedoch ebenfalls möglich, die

History-Funktion auf alle Verfeinerungsebenen eines Zustands anzuwenden; zu diesem Zweck wird die Notation „Ⓗ* “ verwendet, die an die Darstellung der reflexiven und transitiven Hülle von Relationen erinnert.

Abb. 3.9 verdeutlicht den Unterschied zwischen diesen beiden Arten der History-Funktion. Für den mit dem Ereignis *a* beschrifteten Übergang gilt in beiden dargestellten Fällen, daß der Zustand D eingenommen wird – unabhängig von dem zuletzt innerhalb von F besuchten Zustand. Tritt jedoch das Ereignis *b* ein, so können beim Übergang von A nach F unterschiedliche Folgezustände resultieren: Nimmt man an, daß B der zuletzt eingenommene Zustand war, bevor F verlassen wurde, so erfolgt im Statechart aus Abb. 3.9(a) ein Übergang nach C, da allein der Umstand, daß das System in E verweilte, als Information genutzt wird. Demgegenüber wird im Statechart aus Abb. 3.9(b) erneut der Zustand B eingenommen, weil in diesem Fall die entsprechende Information für alle Hierarchieebenen gespeichert wurde.



(a) (b)

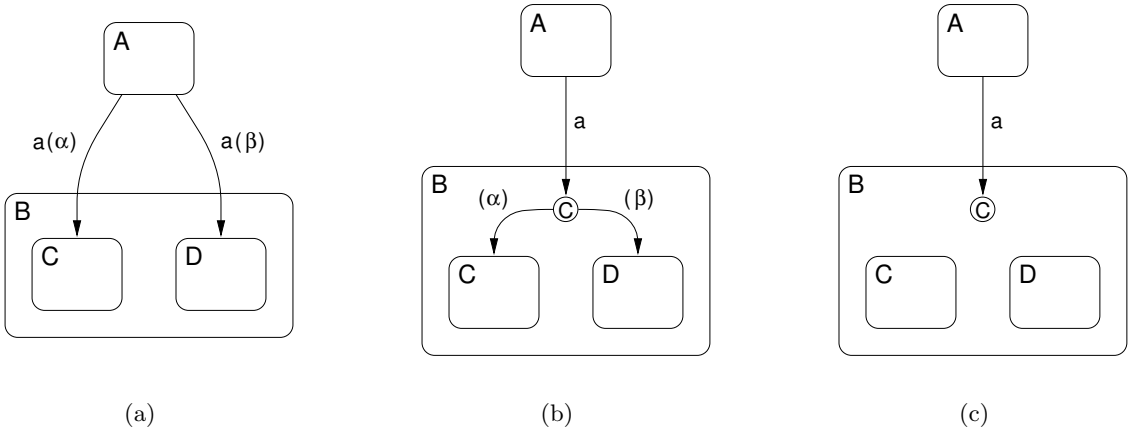
Abb. 3.9: Vergleich der beiden Varianten der History-Funktion

Das Beispiel aus Abb. 3.9 läßt auch das Zusammenwirken von Default-Zuständen und Anwendungen der History-Funktion erkennen: Die History-Funktion kommt nur dann wirklich zum Einsatz, wenn der Zustand F bereits betreten worden ist; beim ersten Übergang nach F wird der Default-Zustand D eingenommen.

Die beiden bisher behandelten Varianten der History-Funktion stellen zwei Extreme dar: Entweder wird nur die „System-Geschichte“ der jeweils obersten Hierarchieebene oder auch diejenige aller feineren Ebenen betrachtet. Verwendet man die History-Funktion auf verschiedenen Ebenen, so kann man beliebige Ergebnisse zwischen diesen beiden Polen erzielen. Als eine ausdrucksmächtige Ergänzung der History-Funktion schlägt Harel den Einsatz temporaler Logik vor; in [**?**] findet man eine kurze Betrachtung dieser Idee.

Damit die bis zu einem bestimmten Zeitpunkt vorgenommenen History-Einträge gelöscht werden können, sieht Harel die beiden Aktionen *clear-history*(*state*) und *clear-history*($state^*$) vor, von denen die erste nur die für die Ebene des Zustandes *state* gültige History-Information, die zweite hingegen auch alle entsprechenden Vermerke der unterhalb von *state* liegenden Hier-

archieebenen löscht. Eine Anwendung dieser beiden Aktionen findet sich in dem Beispiel, das Gegenstand von Abschnitt 3.3 ist.



(a) (b) (c)

Abb. 3.10: Ein Statechart in (a); Möglichkeiten der vereinfachten Darstellung mit Hilfe des bedingten Eintritts in (b) und (c)

Zwei weitere Möglichkeiten, einen Zustand einzunehmen, stellen der *bedingte* und der *selektive Eintritt* zur Verfügung, die es erlauben, komplizierte Zustandsübergänge in einer einfachen grafischen Form darzustellen. Der bedingte Eintritt ist für Fälle vorgesehen, die der in Abb. 3.10(a) dargestellten Situation ähneln: Ein Ereignis a löst einen Übergang von einem Zustand A in einen Zustand B aus, wobei der einzunehmende Subzustand von B jedoch davon abhängt, ob bestimmte Bedingungen erfüllt sind. Mit Hilfe des Zeichens „Ⓒ“, das einen bedingten Eintritt kennzeichnet (der Buchstabe „C“ soll an das Wort „conditional“ erinnern), kann man das Diagramm vereinfachen und zu Abb. 3.10(b) gelangen.

Ist man zunächst an einer groben Darstellung interessiert, so kann man sich auf eine Vereinfachung gemäß Abb. 3.10(c) beschränken; die erforderlichen Einzelheiten sollten dann separat festgehalten werden, damit sie bei Bedarf zur Verfügung stehen.

Der selektive Eintritt stellt eine weitere Art dar, einen verfeinerten Zustand einzunehmen. Er kann verwendet werden, wenn es eine 1:1-Beziehung zwischen der Menge der denkbaren Ereignisse, die einen Übergang auslösen können, und der Menge der Subzustände, die mögliche Folgezustände sind, gibt. In einem solchen Fall kann man jedes der Ereignisse als Auswahl aus einer Menge von Optionen ansehen, die durch die einzelnen Zustände repräsentiert werden, wobei die Zugehörigkeit eines Ereignisses zu einem bestimmten Zustand anhand der gewählten Bezeichnungen erkennbar ist.

Abb. 3.11(a) zeigt ein an [**?**] orientiertes Beispiel, das einige Einstellungsmöglichkeiten einer Digitaluhr wiedergibt. Der Benutzer der Uhr kann sich zunächst für eine einzustellende Komponente entscheiden, die er durch einen Knopfdruck auswählt. Die gewählte Komponente kann er daraufhin mit Hilfe eines anderen Knopfes aktualisieren. Abb. 3.11(b) zeigt, wie die Situation mittels des selektiven Eintritts, dargestellt durch das Symbol „Ⓢ“, modelliert werden kann.

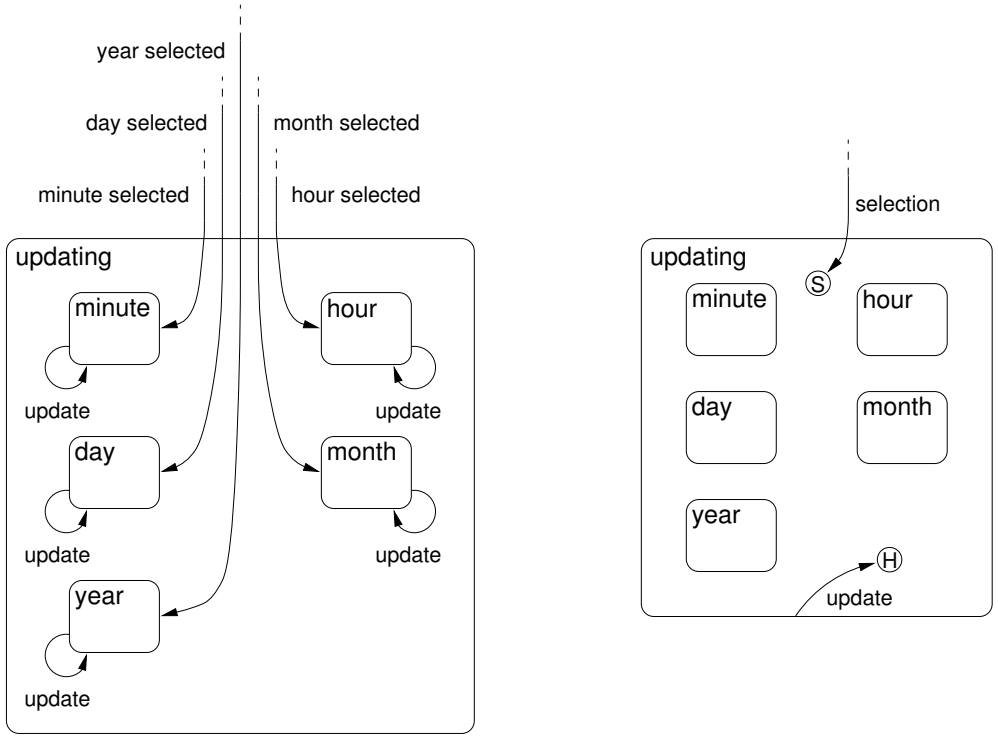


Abb. 3.11: Ein Statechart in (a) und eine mögliche Vereinfachung durch einen selektiven Eintritt in (b)

Die History-Funktion wird hier in besonders eleganter Form benutzt, um die Darstellung weiter zu vereinfachen.

Alle bisher in diesem Abschnitt beschriebenen Teile des grafischen Formalismus der Statecharts stellen Möglichkeiten dar, denjenigen Zustand zu charakterisieren, der anfangs oder bei einem Zustandsübergang eingenommen werden soll. Im folgenden wird auch betrachtet, wie Zustände verlassen werden.

Von besonderem Interesse sind Zustände, die aus orthogonalen Komponenten bestehen, da sie auf vielfältige Weise eingenommen und verlassen werden können. Abb. 3.12(a) zeigt ein Beispiel, das einige Möglichkeiten verdeutlicht. Einen einfachen Fall stellen diejenigen Zustandsübergänge dar, die durch die Ereignisse a und b ausgelöst werden. Der eine Übergang resultiert im Zustandspaar (F, H), das durch das Charakteristikum „Default-Zustand“ bestimmt wird, während der andere unabhängig vom gegenwärtigen Zustandspaar dazu führt, daß der Zustand B verlassen wird.

Im Falle des durch c ausgelösten Übergangs wird der Zustand G explizit bestimmt, während die andere Komponente des einzunehmenden Zustandspaares wiederum durch den Default-

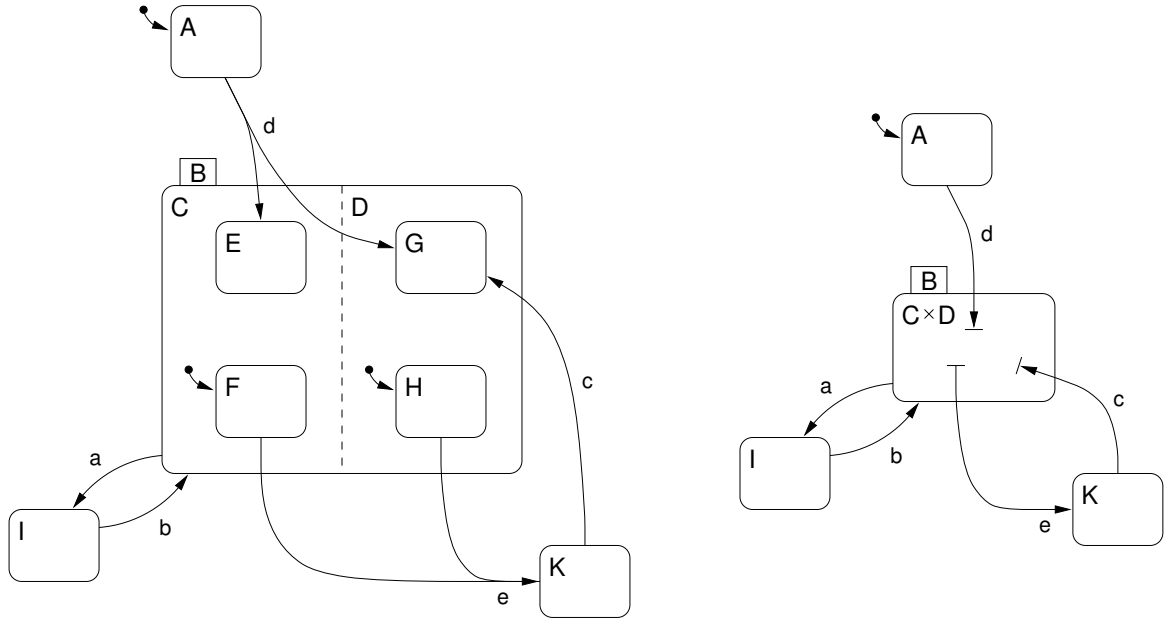


Abb. 3.12: Einen Zustand mit orthogonalen Subzuständen einnehmen und verlassen (in (a)); möglicher Ausgangspunkt beim Entwurf in (b)

Zustand festgelegt ist. Eine bisher nicht behandelte Darstellungsmöglichkeit zeigen die beiden verbleibenden Übergänge auf. Durch den sich aufspaltenden Pfeil, der mit dem Ereignis d beschriftet ist, werden beide Komponenten des Folgezustandspaares bestimmt; der zum Zustand K führende Pfeil legt fest, daß im Falle des Ereignisses e das Zustandspaar (F, H) der gegenwärtige Zustand sein muß, damit ein Übergang nach K stattfinden kann. Neben den in Abb. 3.12(a) dargestellten Möglichkeiten, einen mit Hilfe der AND-Dekomposition verfeinerten Zustand einzunehmen und zu verlassen, sind zahlreiche weitere vorstellbar; man denke z. B. daran, daß auch die History-Funktion, der bedingte und der selektive Eintritt verwendet werden können, um eine Komponente eines Zustandspaares festzulegen.

Entwirft man einen Statechart, so kann die genaue Struktur eines Zustands anfangs unbekannt sein, während die Art und die Anzahl der Übergänge zwischen diesem Zustand und anderen durchaus bereits feststehen kann. In solchen Fällen ist es nützlich, die Übergänge bereits einzuzeichnen, ihre Start- oder Zielzustände aber durch kleine Striche als „noch nicht festgelegt“ zu kennzeichnen. Abb. 3.12(b) zeigt das Ergebnis eines solchen Vorgehens für den Statechart aus Abb. 3.12(a).

Abschließend betrachten wir einen Aspekt, der für Realzeitsysteme von Bedeutung ist: Häufig werden Zustände mit zeitlichen Einschränkungen der Art „10 Sekunden warten, bevor die Schranke geschlossen wird“ oder „Nach 5 Minuten ohne Eingabe den Bildschirmschoner akti-

vieren“ verknüpft. Für solche Fälle sieht der Formalismus der Statecharts eine spezielle Notation vor. Die Grundlage bildet stets ein Ereignis der Form *timeout*(*event, number*), das nach einem Ereignis *event* generiert wird, sobald eine bestimmte Anzahl von Zeiteinheiten, die durch *number* festgelegt wird, verstrichen ist. Mit Hilfe eines solchen Ereignisses kann Einfluß darauf genommen werden, wie lange ein System in einem bestimmten Zustand verweilt.

Abb. 3.13 zeigt ein einfaches Beispiel. Durch die gezackte Linie am oberen Rand des Zustandes wird verdeutlicht, daß der Zustand einer zeitlichen Beschränkung unterliegt, die direkt unter ihr angegeben ist. Im vorliegenden Fall handelt es sich um eine obere Schranke; es ist ebenfalls möglich, ein Zeitintervall in der Form $t_1 < t_2$ oder eine untere Schranke anzugeben, was zur Folge hätte, daß Ereignisse im gegenwärtigen Zustand bis zu einem bestimmten Zeitpunkt nicht berücksichtigt würden, so daß eine Mindestverweildauer gewährleistet wäre. Das Ereignis *timeout* steht für *timeout(„Zustand betreten“, Obergrenze)*, durch das garantiert wird, daß sich das System höchstens für eine bestimmte Zeit, die durch *Obergrenze* vorgegeben wird, im betrachteten Zustand befindet.

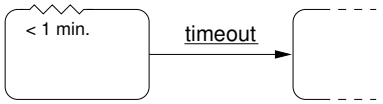


Abb. 3.13: Ein Zustand mit maximaler Verweildauer

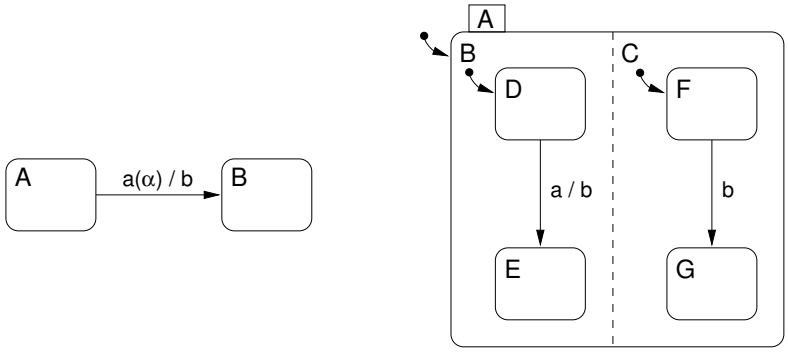
Kommunikation: Ereignisse, Aktionen und Aktivitäten

Reaktive Systeme arbeiten weitgehend ereignisgesteuert. Sie reagieren auf externe Stimuli, und ihr Verhalten kann nach [**?**] als „Menge der erlaubten Folgen von Ein- und Ausgabe-Ereignissen, Bedingungen und Aktionen“ betrachtet werden. Harel definiert in seinen grundlegenden Artikeln über Statecharts den Begriff „Ereignis“ nicht explizit, jedoch kann man aufgrund seiner Erläuterungen und der von ihm vorgestellten Beispiele davon ausgehen, daß er sich auf die allgemeine Bedeutung dieses Wortes bezieht.

In den vorangegangenen Abschnitten wurden einige Statecharts behandelt, die einfache Systeme beschreiben. Die Reaktionsmöglichkeiten der zugehörigen Systeme sind in diesen Beispielen auf Zustandswechsel beschränkt. Dabei wird keine Verbindung zwischen den einzelnen Zuständen und den Übergängen zwischen ihnen einerseits sowie den tatsächlichen Systemreaktionen andererseits, die Einflüsse auf die reale Welt bedeuten, hergestellt. Die Statecharts fungierten somit bisher ausschließlich als Kontrolleinrichtungen, die in Abhängigkeit von auftretenden Ereignissen sowie geltenden Bedingungen unter Berücksichtigung zeitlicher Gegebenheiten das Verhalten des gesamten Systems lediglich bedingen.

Die geschilderten Unzulänglichkeiten können durch eine Erweiterung des bisherigen Formalismus überwunden werden, die die Möglichkeit beinhaltet, Ereignisse innerhalb von Statecharts zu generieren. Kommt es zu einem Zustandsübergang, so kann dann nicht nur ein neuer Zustand eingenommen, sondern auch ein Ereignis erzeugt werden; ein derartiges Ereignis wird von Ha-

rel als *Aktion* bezeichnet.[1] In der grafischen Darstellung können Aktionen durch die Notation „. . . /*aktion*“ ausgedrückt werden, die der Beschriftung des entsprechenden Pfeils hinzuzufügen ist. Abb. 3.14(a) zeigt ein Beispiel, in dem bei einem Zustandsübergang ein Ereignis b generiert wird.



(a) (b)

Abb. 3.14: Aktion bei einem Zustandsübergang in (a); Aktion, die einen Übergang in einer orthogonalen Komponente bewirkt, in (b)

Aktionen sind Geschehnisse von extrem kurzer Dauer, und in Übereinstimmung mit der Brockhaus-Definition für „Ereignis“ können sie als momentane Vorkommnisse, d. h. als augenblicklich, betrachtet werden. Unter den Aktionen lassen sich zwei Arten unterscheiden:

- Aktionen, die unmittelbar auf die Umgebung des reaktiven Systems wirken (z. B. „Fahrkarte und Wechselgeld ausgeben“), sowie
- Aktionen, die Zustandsübergänge in orthogonalen Komponenten auslösen und somit nur einen indirekten Einfluß auf die Umgebung ausüben können.

Die Aktionen der ersten Kategorie sind Ausgaben des Systems, die in derjenigen Weise erfolgen, wie sie für Mealy-Automaten charakteristisch ist (siehe z. B. [**?**]). Aktionen, die der zweiten Gruppe angehören, sind vom Statechart selbst generierte Ereignisse, die in allen orthogonalen Komponenten registriert werden; Harel spricht in diesem Zusammenhang von *broadcast-communication*. Der Bereich, in dem auf Aktionen reagiert werden kann, ist somit stets auf den entsprechenden Statechart begrenzt (siehe [**?**]).[2]

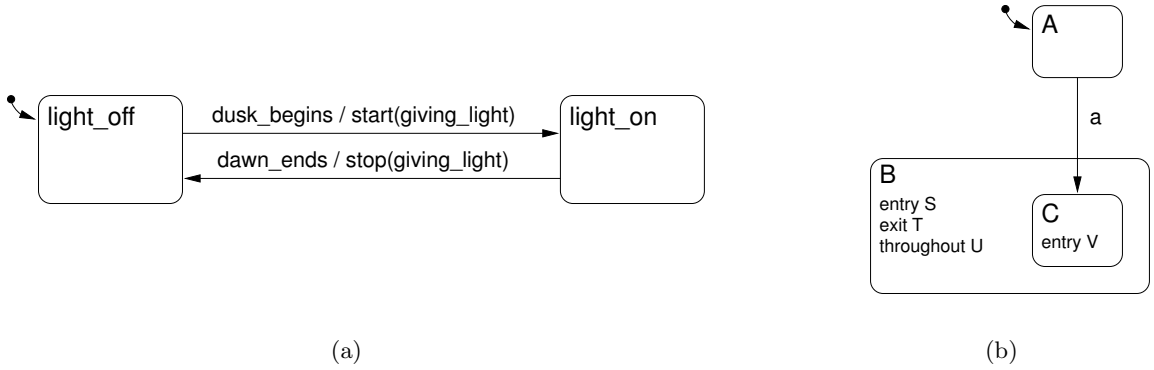
[1] Bei einem Zustandsübergang kann eine *Aktion* ausgeführt werden, die ein *Ereignis* generiert. Da Harel eine Aktion und das durch sie generierte Ereignis ohnehin mit demselben Bezeichner versieht, kann man die beiden Begriffe in diesem Zusammenhang als Synonyme ansehen.

[2] Der Broadcast-Mechanismus stellt neben der XOR- und der AND-Dekomposition das dritte wesentliche Charakteristikum von Statecharts dar; Harel beschreibt Statecharts in einer Kurzform folgendermaßen: „*statecharts = state-diagrams + depth + orthogonality + broadcast-communication*“.

Abb. 3.14(b) verdeutlicht die Möglichkeit, Transitionen einer Komponente durch Transitionen einer orthogonalen Komponente zu steuern: Befindet sich das System im Zustand (D, F) und tritt das Ereignis a ein, so kommt es in der Komponente B zum Übergang in den Zustand E. Hierbei wird das Ereignis b generiert, das daraufhin in der Komponente C registriert wird und dort dazu führt, daß der Zustand G eingenommen wird. Dieses Beispiel ist sehr einfach, und die Abb. 3.14(b) läßt nur eine Interpretation zu. Es sind allerdings wesentlich kompliziertere Fälle vorstellbar, in denen es zu Zyklen kommen kann, wenn Ereignisse in einer bestimmten Reihenfolge generiert werden. Statecharts, die derartige Systeme beschreiben, sind zunächst mehrdeutig, so daß ihnen eine präzise Semantik zugeordnet werden muß. Von grundlegender Bedeutung ist in diesem Zusammenhang die Frage, ob in einem Zeitschritt das registrierte externe Ereignis sowie alle sich daraus ergebenden Aktionen abgearbeitet oder generierte Ereignisse erst in späteren Zeitschritten betrachtet werden.

Durch die Möglichkeit, Ereignisse selbst zu generieren, können Statecharts Einfluß auf ihre Umgebung ausüben. Da die beschriebenen Aktionen allerdings als augenblicklich betrachtet werden, während reale Systeme stets durch Abläufe gekennzeichnet sind, die eine bestimmte Zeitdauer beanspruchen, ist eine zusätzliche Erweiterung des Formalismus erforderlich. Harel führt zu diesem Zweck den Begriff der *Aktivität* ein, mit dem Vorgänge bezeichnet werden, die Zeit benötigen.

Damit Statecharts Aktivitäten steuern können, sind zwei Aktionen vonnöten, die den Start und das Ende einer Aktivität zur Folge haben; aus diesem Grund ordnet Harel jeder Aktivität X die beiden Aktionen $start(X)$ und $stop(X)$ zu, die die entsprechenden Bedeutungen besitzen. Darüber hinaus kann mit der Bedingung $active(X)$ geprüft werden, ob die Aktivität X zum betrachteten Zeitpunkt ausgeführt wird. Abb. 3.15(a) verdeutlicht, wie zwei Aktionen eine Aktivität steuern können: Setzt die Abenddämmerung ein, so wird automatisch eine Außenbeleuchtung eingeschaltet, die während der gesamten Nacht leuchtet. Das Ende der Morgendämmerung führt schließlich dazu, daß das Licht ausgeschaltet wird.



(a) (b)

Abb. 3.15: Aktionen steuern in (a) nach dem Vorbild von Mealy-Automaten eine Aktivität; von Moore-Automaten stammende Eigenschaften in (b)

Es ist zu beachten, daß die Aktivitäten selbst zunächst nicht durch Statecharts spezifiziert

werden. Harel schlägt vor, für die Spezifikation sequentieller Aktivitäten herkömmliche Programmiersprachen zu verwenden. Sind die Aktivitäten hingegen selbst reaktiver Natur, könnten auch sie ihrerseits durch einzelne Statecharts beschrieben werden, so daß Hierarchien von Statecharts und Aktivitäten denkbar sind. Das Programmsystem STATEMATE, das die Entwicklung reaktiver Systeme mit Hilfe von Statecharts ermöglicht und von der Firma i-Logix vertrieben wird, verwendet einen weiteren grafischen Formalismus (sogenannte *activity-charts*), um Aktivitäten zu spezifizieren (siehe [**?**]).

Aktionen und Aktivitäten können nicht nur in der bisher beschriebenen Form, sondern darüber hinaus auch auf der Grundlage des Mechanismus, der für Moore-Automaten charakteristisch ist (siehe wiederum [**?**]), ausgeführt werden. In diesem Fall werden Aktionen nicht mit Zustandsübergängen, sondern vielmehr mit einzelnen Zuständen des Statechart verknüpft.

Indem man einen Zustand zusätzlich mit einem der Ausdrücke *entry* S oder *exit* S versieht, kann eine Aktion S ausgeführt werden, wenn er betreten oder verlassen wird. Ferner kann während der gesamten Zeit, in der ein System in einem Zustand verweilt, eine Aktivität X ablaufen; dieses kann man durch die Notation *throughout* X kennzeichnen, die äquivalent zur gleichzeitigen Beschriftung des Zustands mit *entry start*(X) und *exit stop*(X) ist.

Abb. 3.15(b) zeigt für eine Aktivität U sowie für Aktionen S, T und V, in welcher Weise man Statecharts um die zusätzlichen Ausdrücke ergänzen kann, und macht außerdem deutlich, daß bereits Zustandshierarchien zu Nebenläufigkeit führen können: Wird nach dem Ereignis a der Zustand C betreten, so werden die Aktionen S und V gleichzeitig ausgeführt.

3.3 Ein Beispiel

In den vorangegangenen Abschnitten wurden die syntaktischen Konstrukte von Statecharts weitgehend isoliert voneinander betrachtet. Um ihren Nutzen zu verdeutlichen, sollen nun anhand eines Anwendungsbeispiels einige Möglichkeiten aufgezeigt werden, wie sie zusammenwirken können.

Als Beispiel dient ein bereits existierendes Produkt: ein Heimtrainer.[3] Somit wird der Statechart-Formalismus im weiteren nicht für einen Entwurf, sondern zu Dokumentationszwecken verwendet. Der Heimtrainer ist ein in sich verständliches Beispiel. Er ist einerseits überschaubar genug, um hier behandelt zu werden, andererseits trotz seiner vermeintlichen Einfachheit bereits recht komplex.

Die Funktionen des Heimtrainers

Der Heimtrainer ist ein Trimmrad, das der Beinbewegung des Trainierenden einen Tretwiderstand entgegensetzt. Er ist vor allem für das Training der Beinmuskulatur und des Kreislaufs geeignet. An einer Bedien- und Anzeigeeinheit (siehe Abb. 3.16) kann man mit vier Tasten Einstellungen vornehmen und Daten ablesen.

[3] Es handelt sich hierbei um das Gerät „WIMFIT 110“ der Siegmann & Schröder GmbH, Hamburg.

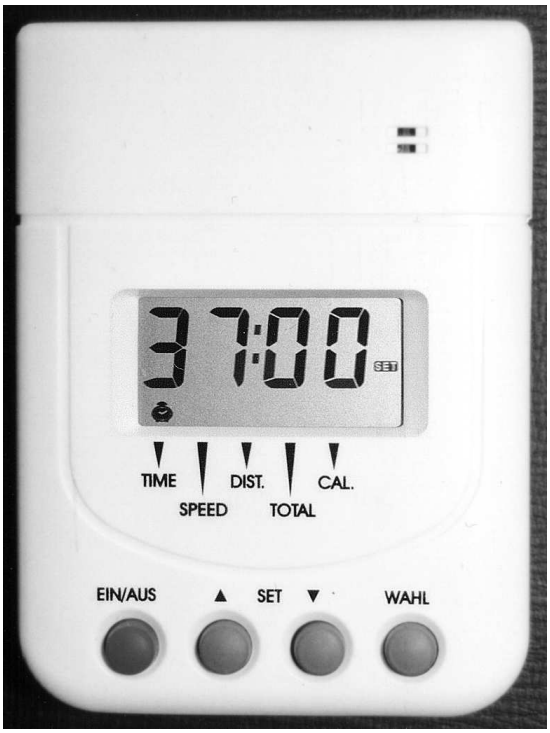


Abb. 3.16: Die Bedien- und Anzeigeeinheit des Heimtrainers

Über die Taste „EIN/AUS“ kann der Heimtrainer ein- und ausgeschaltet werden. Ist er eingeschaltet, so können mit Hilfe der Taste „WAHL“ die bisher benötigte Zeit, die momentane Geschwindigkeit, die zurückgelegte Entfernung, die Gesamtdistanz aller Trainingseinheiten und der ungefähre Kalorienverbrauch angezeigt werden. Ferner ist es möglich, über die „WAHL“-Taste einen „Scan“-Modus zu aktivieren, in dem diese Anzeigen bei einer jeweiligen Verweildauer von ca. 5 Sekunden automatisch nacheinander durchlaufen werden.

Befindet sich die Anzeige im Zeit- oder Entfernungsmodus, so kann mit den beiden „SET“-Tasten eine Zeit oder Entfernung, die man zu absolvieren wünscht, vorgegeben werden. Das Gerät zählt in beiden Fällen bis Null herunter, läßt für ca. 8 Sekunden ein Alarmsignal ertönen und zählt dann im positiven Bereich weiter.

Wird der Heimtrainer für ungefähr 4 Minuten nicht benutzt, so erlischt die Digitalanzeige.[4]

Eine Verhaltensbeschreibung mit Hilfe von Statecharts

Betrachtet man die Bedien- und Anzeigeeinheit, so lassen sich zunächst einmal zwei Zustände unterscheiden: Der Heimtrainer ist entweder ein- oder ausgeschaltet (siehe Komponente *main* des Statecharts in Abb. 3.17). Betätigt man im Zustand *off* die Taste „EIN/AUS“, so wird der Heimtrainer eingeschaltet; der Tastendruck ist durch das Ereignis *ein_aus* gekennzeichnet. Ein Benutzer hat darüber hinaus die Möglichkeit, den Heimtrainer dadurch zu aktivieren, daß er

[4]Dies gilt allerdings nicht uneingeschränkt (siehe Abschnitt 3.3).

einfach zu treten beginnt. Wir gehen davon aus, daß die kontinuierliche Tretbewegung in eine Folge elektrischer Impulse umgesetzt wird. Durch das Ereignis *cycle* verdeutlichen wir, daß ein solcher Impuls gesendet wird.[5]

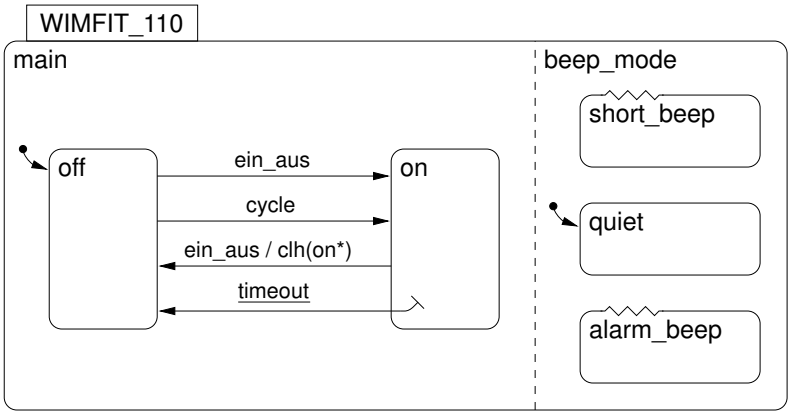


Abb. 3.17: Der Heimtrainer als Statechart

Ist der Heimtrainer eingeschaltet, so kann er mit Hilfe der Taste „EIN/AUS“ ausgeschaltet werden. Wenn dieses Ereignis eintritt, wird die Aktion *clear-history*, abgekürzt durch *clh*, ausgelöst, die die Information über die bisher im Zustand *on* durchlaufene Folge von Subzuständen vollständig löscht (vergleiche Seite 49). Das Gerät wechselt selbständig in den Zustand *off*, wenn es für eine bestimmte Zeitspanne nicht benutzt worden ist. Da dies jedoch nicht uneingeschränkt gilt, beginnt der zum Ereignis *timeout* gehörige Pfeil innerhalb der Begrenzung des Zustandes *on*.

Wird die Taste „EIN/AUS“ gedrückt, so hat dies nicht nur einen Zustandswechsel zur Folge, sondern es ertönt darüber hinaus stets ein kurzer Piepton. Soll das Ereignis *ein_aus* verwendet werden, um in einen zugehörigen Zustand zu wechseln, so kann ein entsprechender Übergang nicht innerhalb eines der Zustände *on* und *off* liegen, sofern man sich an der in [**?**] beschriebenen Semantik orientiert. Dies ergibt sich aus der Tatsache, daß *on* und *off* im Falle des Ereignisses *ein_aus* verlassen werden; in solchen zunächst nichtdeterministischen Situationen hat stets derjenige Übergang, der von einem Zustand einer höheren Abstraktionsebene ausgeht, Priorität, so daß im vorliegenden Fall der Zustand, der mit dem Piepton verknüpft ist, überhaupt nicht eingenommen würde.[6]

Wir haben uns dafür entschieden, die Zustände *on* und *off* in einer Komponente *main* zu kapseln und eine zu dieser orthogonale Komponente *beep_mode* einzuführen, die u. a. den Zustand *short_beep* beinhaltet, der für den Piepton bei einem Tastendruck steht. Durch den Zustand

[5] Da wir Ereignisse, die Tastendrücken entsprechen, gemäß den auf der Bedien- und Anzeigeeinheit aufgeführten Begriffen benannt haben, treten englische Bezeichnungen neben deutschen auf; alle übrigen Bezeichner entstammen dem Englischen.

[6] Auch eine Aktion, ausgelöst bei einem Übergang, der durch *ein_aus* hervorgerufen wird, hilft hier nicht weiter, weil intern generierte Ereignisse nach [**?**] erst im folgenden Zeitschritt abgearbeitet werden. Da der Ausgangszustand dann bereits verlassen worden ist, kann die ausgelöste Aktion bei zumindest einem Zustandswechsel zwischen *on* und *off* nicht den Übergang in den Zustand, der für den Piepton steht, bewirken.

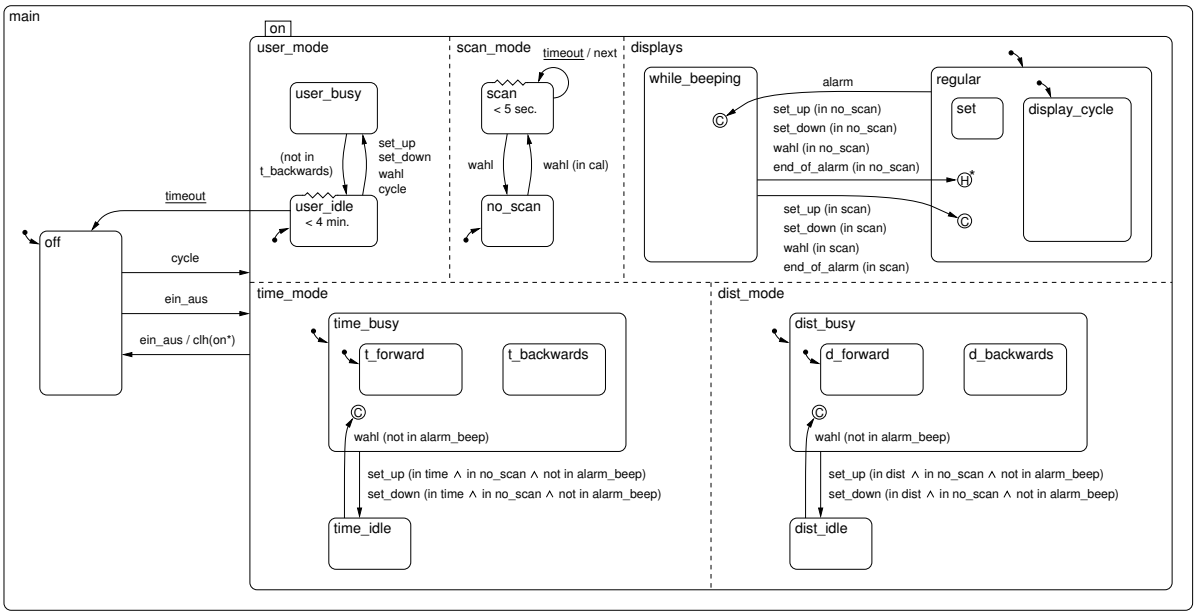


Abbildung 3.18: Der Zustand *on* innerhalb von *main*

alarm_beep wird erfaßt, daß der eingangs beschriebene Alarmton erklingt. Da dieses Alarmsignal durch einen Tastendruck, der einen kurzen Piepton zur Folge hat, beendet werden kann, ist auch *alarm_beep* Bestandteil von *beep_mode*. Im Anfangszustand des Heimtrainers erklingt kein Ton (Zustand *quiet*). Die Übergänge zwischen den einzelnen Subzuständen von *beep_mode* werden hier noch nicht betrachtet.

Abb. 3.18 zeigt, wie der Zustand *on* aufgebaut ist. Durch die Komponente *user_mode* wird gesteuert, unter welchen Umständen sich der Heimtrainer automatisch ausschaltet. Zunächst befindet sich der Heimtrainer im Zustand *user_idle*, der nach einer Zeit von 4 Minuten automatisch verlassen wird. Drückt der Benutzer eine der beiden „SET“-Tasten oder den „WAHL“-Knopf oder macht eine Tretbewegung, so erfolgt ein Übergang nach *user_busy*. Dieser Zustand wird im Regelfall sofort wieder verlassen. Hierbei ist zu beachten, daß der entsprechende Übergang nicht durch ein Ereignis ausgelöst wird; er ist lediglich von der Bedingung „*not in t_backwards*“ abhängig, die ausdrückt, daß die Zeit nicht rückwärts gezählt wird (siehe Komponente *time_mode* in Abb. 3.18 und Abb. 3.20(a)). Genau dies ist der bereits erwähnte Fall, in dem der Heimtrainer auch nach Ablauf von 4 Minuten weiterhin eingeschaltet bleibt.

Mit Hilfe der zu *user_mode* orthogonalen Komponente *scan_mode* wird festgehalten, ob sich der Heimtrainer im „Scan“-Modus befindet. Dieser ist in die Folge der Anzeigen für Zeit, Geschwindigkeit, Strecke, Gesamtdistanz und Kalorienverbrauch integriert und daher ebenfalls über die „WAHL“-Taste zu erreichen: Wird der bisherige Kalorienverbrauch angezeigt (Bedingung „*in cal*“), so kann man mittels der Taste „WAHL“ den „Scan“-Modus aktivieren, der wieder verlassen werden kann, indem erneut „WAHL“ gedrückt wird. Alle 5 Sekunden wird *scan* durch *timeout* automatisch verlassen und erneut eingenommen. Hierbei wird stets das interne Ereignis *next* generiert, das in der Komponente *displays* verwendet wird, um die Anzeige

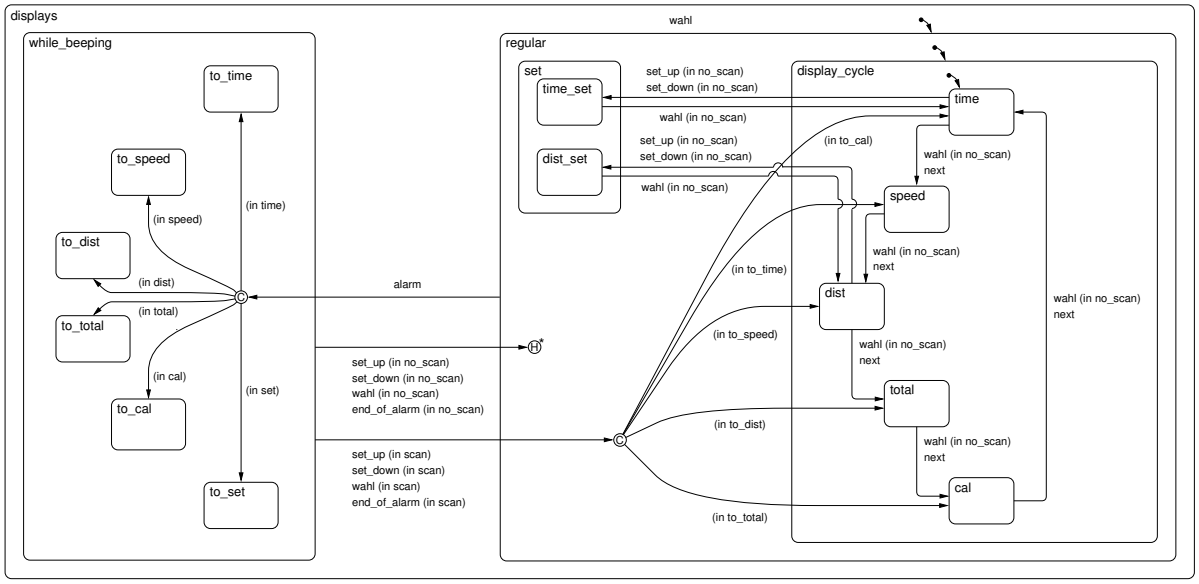


Abbildung 3.19: Die Komponente *displays*

fortzuschalten.

Über die Komponente *displays* wird erfaßt,

- welche Information angezeigt werden soll (z. B. die bisher benötigte Zeit),
- ob der Benutzer im Augenblick eine Zeitdauer oder Streckenlänge vorgibt und
- ob z. Zt. das Alarmsignal, welches anzeigt, daß eine zuvor eingestellte Zeitdauer abgelaufen oder eine festgelegte Distanz absolviert worden ist, ertönt.

Betätigt der Benutzer die „WAHL“-Taste oder eine der beiden „SET“-Tasten, so hängt die Wirkung davon ab, welche dieser Umstände vorliegen, d. h. welcher Subzustand innerhalb von *displays* zuletzt eingenommen wurde. Mit Hilfe des Zustands *regular* wird festgelegt, welche Information anzuzeigen ist: Entweder ist dies eine der Angaben über Zeit, Geschwindigkeit, Strecke, Gesamtdistanz oder Kalorienverbrauch (Zustand *display_cycle*), oder der Benutzer ist im Begriff, eine Zeit oder eine Streckenlänge vorzugeben (Zustand *set*). Das Ereignis *alarm*, welches signalisiert, daß die Zeit oder die Strecke auf Null heruntergezählt worden ist, und in den beiden Komponenten *time_mode* und *dist_mode* ausgelöst werden kann (siehe Abb. 3.20), führt zu einem Übergang in den Zustand *while_beeping*, durch den festgehalten wird, welche Information nach einem Tastendruck während des Alarmtons oder nach Ende dieses Signals angezeigt werden soll. Der Zustand *while_beeping* kann verlassen werden, indem man die „WAHL“- oder eine der beiden „SET“-Tasten drückt; durch das Ereignis *end_of_alarm*, das in der Komponente *beep_mode* nach dem Ende des Alarmtons erzeugt wird (siehe Abb. 3.22), wird *while_beeping* automatisch verlassen.

Abb. 3.19 zeigt den Zustand *displays* im Detail. Mit Hilfe der Subzustände von *display_cycle* wird festgehalten, welche der fünf bereits vorgestellten Angaben anzuzeigen ist. Befindet sich der Heimtrainer nicht im „Scan“-Modus, so kann man zur jeweils nächsten Anzeige wechseln, indem man die „WAHL“-Taste drückt. Durch das interne Ereignis *next*, das in der Komponente *scan_mode* generiert wird (siehe Abb. 3.18), kann der Anzeige-Zyklus automatisch durchlaufen werden. Wird die Zeit oder die Streckenlänge angezeigt, so kann mittels der beiden „SET“-Knöpfe ein Übergang nach *time_set* oder *dist_set* erfolgen. Diese beiden Zustände ermöglichen es dem Benutzer, die zu absolvierende Trainingszeit oder eine Distanz vorzugeben, und können über die „WAHL“-Taste wieder verlassen werden.

Erfolgt durch das Ereignis *alarm* ein Übergang nach *while_beeping*, so wird die Information über den innerhalb von *regular* zuletzt besuchten Zustand gespeichert, indem ein entsprechender Subzustand eingenommen wird. Befindet sich der Heimtrainer im „Scan“-Modus, so wird diese Information beim Verlassen von *while_beeping* dazu benutzt, über einen bedingten Eintritt zur nächsten Anzeige innerhalb des Zyklus zu schalten. Ist hingegen *no_scan* der aktuelle Zustand innerhalb von *scan_mode* (siehe Abb. 3.18), so bewirken die Ereignisse, durch die *while_beeping* verlassen wird, eine Rückkehr zur bisherigen Anzeige; in diesem Fall ist die mittels eines Subzustandes von *while_beeping* gespeicherte Information redundant, weil die History-Funktion verwendet werden kann.[7]

Die beiden bisher noch nicht erläuterten Komponenten des Zustands *on* sind *time_mode* und *dist_mode* (siehe Abb. 3.18), über die festgehalten wird, ob die Zeit oder die Strecke im Augenblick nicht fortzuschreiben ist, da der Benutzer gerade einen Wert vorgibt, oder ob vor- oder rückwärts zu zählen ist. Da *time_mode* und *dist_mode* die gleiche Struktur besitzen (siehe Abb. 3.20), beschränkt sich die folgende Darstellung auf den Zustand *time_mode*.

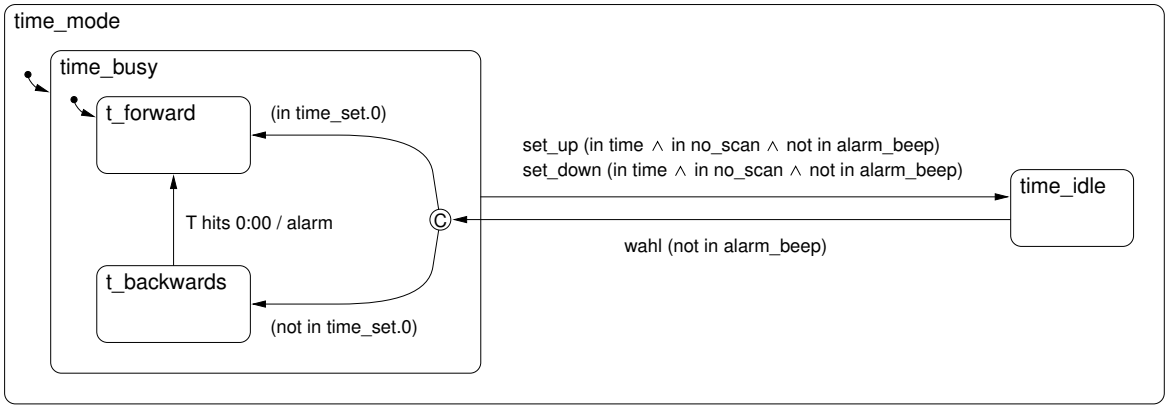
Wird der Heimtrainer eingeschaltet, so wird die Zeit fortgeschrieben (Zustand *time_busy*), und es wird vorwärts gezählt (Zustand *t_forward*). Während der Zeitanzeige führt jede der beiden „SET“-Tasten in den Zustand *time_idle*, sofern sich der Heimtrainer nicht im „Scan“-Modus befindet und kein Alarmsignal ertönt. Innerhalb von *time_idle* werden keine Zeiteinheiten gezählt. Mittels der „WAHL“-Taste wird die eingestellte Zeit bestätigt (siehe Zustand *regular* in Abb. 3.19); in *time_mode* führt dies dazu, daß im Regelfall rückwärts gezählt wird (Übergang nach *t_backwards*). Die Bedingung „*not in time_set.0*“ drückt hierbei aus, daß der Benutzer einen größeren Wert als Null vorgegeben hat (siehe Zustand *time_set* in Abb. 3.21).[8] Hat der Benutzer hingegen den Wert Null eingestellt, so wird vorwärts gezählt. Erreicht die rückwärts gezählte Zeit, die in der Variablen T gespeichert ist, die Nullmarke, so wird das Ereignis *alarm* generiert, das das Alarmsignal auslöst.[9]

Abb. 3.21 zeigt den Aufbau der beiden Subzustände von *set*, der Zustände *time_set* und *dist_set*. Der Zustand *time_set* konnte in eleganter Form als parametrisierter Zustand (siehe

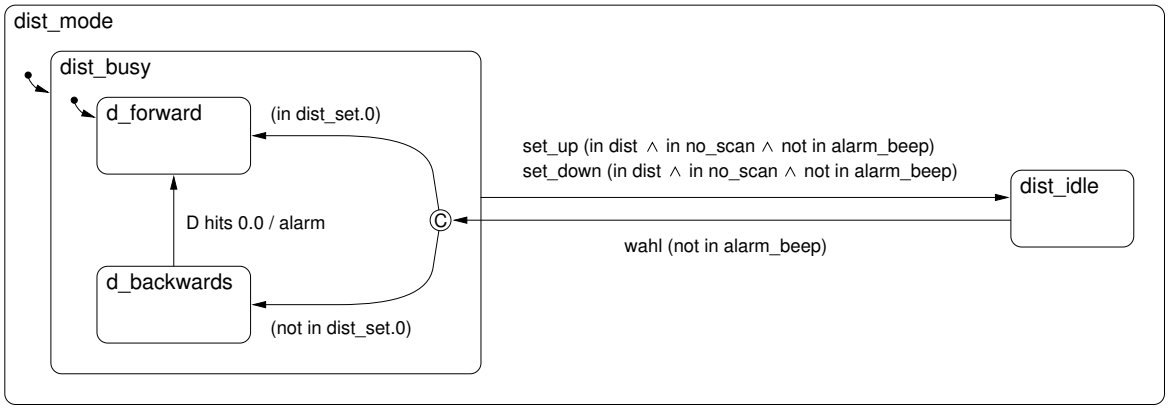
[7] Da sich der Heimtrainer nicht gleichzeitig im „Scan“- und im „Set“-Modus befinden kann, ist der Zustand *to_set* innerhalb von *while_beeping* ohne Nutzen; er ist dennoch eingezeichnet worden, um alle grundsätzlichen Möglichkeiten aufzuführen.

[8] Wie die Bedingung „*not in time_set.0*“ zeigt, können qualifizierte Bezeichner verwendet werden, um einen Bezug auf Subzustände zu ermöglichen.

[9] In *dist_mode* wird die Variable D verwendet, um die noch zu absolvierende Strecke zu speichern.



(a)



(b)

Abb. 3.20: Die Komponenten *time_mode* und *dist_mode*

Abschnitt **??**) realisiert werden; die grafische Darstellung drückt den Sachverhalt in kompakter und präziser Form aus:

- Es gibt 100 Subzustände, bezeichnet mit *0* bis *99*, die jeweils einer einstellbaren Zeit in Minuten entsprechen.
- Die Zustände sind zyklisch angeordnet; mit *set_up* wechselt der Heimtrainer in den Zustand mit der nächsthöheren, mit *set_down* in denjenigen mit der nächstniedrigeren Zahl.

Demgegenüber wirkt die für *dist_set* gewählte Struktur ungeeignet; die Subzustände sind einzeln eingezeichnet, und anstelle eines Verhaltensmusters wird jeder Übergang explizit aufgeführt. Den Grund für die Wahl dieser Struktur stellt eine beobachtete Anomalie dar: Über

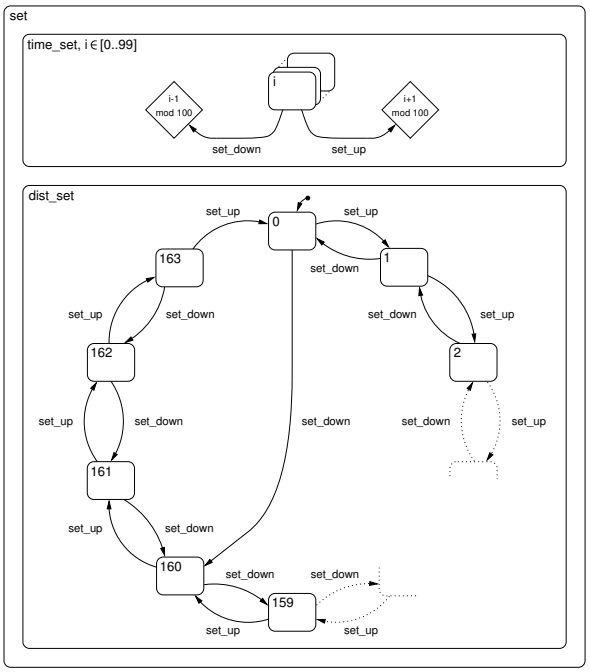


Abbildung 3.21: Die Zustände *time_set* und *dist_set*

set_up sind Entfernungen bis zu 163 km einstellbar, während *set_down* im Falle des Zustands *0* direkt in den Zustand *160* führt. Dieser Umstand macht deutlich, daß eine Erweiterung des Statechart-Formalismus durch eine (grafisch orientierte) Sprache, die solche und wesentlich kompliziertere Strukturen in überschaubarer, auf das Wesentliche reduzierter Form beschreibt, wünschenswert ist.

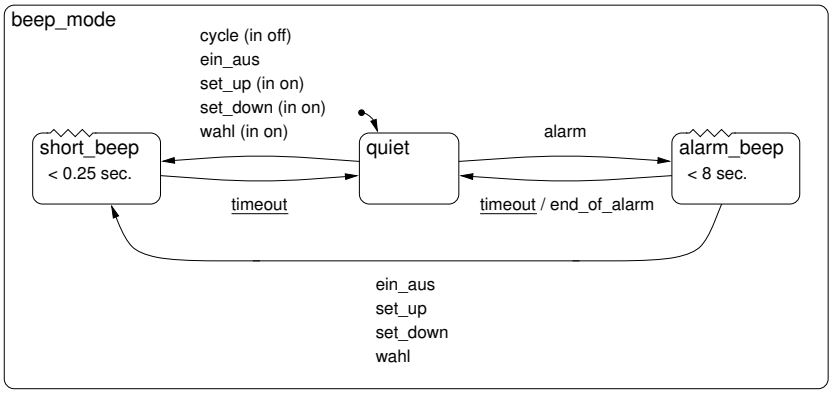


Abb. 3.22: Die Komponente *beep_mode*

Die Komponente *main* ist mit dieser abschließenden Betrachtung des Zustands *set* vollständig behandelt worden. Der zu *main* orthogonale Zustand *beep_mode* ist sehr einfach aufgebaut (siehe Abb. 3.22). Im Zustand *quiet* führt jeder Tastendruck dazu, daß *short_beep* eingenommen wird; dies gilt für den „EIN/AUS“-Knopf uneingeschränkt, während im Falle einer der drei anderen Tasten nur dann ein kurzes Piepsignal ertönt, wenn der Heimtrainer eingeschaltet ist. Beginnt der Benutzer im Zustand *off* mit einer Tretbewegung, so wird über das Ereignis *cycle* ebenfalls ein Piepton erzeugt.

Wenn in einer der Komponenten *time_mode* und *dist_mode* (siehe Abb. 3.20) das interne Ereignis *alarm* generiert wird, so kommt es in *beep_mode* zu einem Übergang nach *alarm_beep* und dazu, daß das bereits mehrfach erwähnte Alarmsignal ertönt. Dieses Signal kann der Benutzer beenden, indem er eine beliebige Taste drückt; auch hierbei ist ein kurzer Piepton zu hören. Die Zustände *short_beep* und *alarm_beep* werden jeweils nach einer festgelegten Zeitspanne verlassen, wobei der Übergang von *alarm_beep* nach *quiet* auch bedeutet, daß das Ereignis *end_of_alarm*, das in der Komponente *displays* die Rückkehr in den Zustand *regular* ermöglicht (siehe Abb. 3.19), generiert wird.

Einige abschließende Bemerkungen

Das Beispiel des Heimtrainers verdeutlicht nicht nur, wie die einzelnen Konstrukte des Statechart-Formalismus verwendet werden können, sondern zeigt auch dessen Grenzen auf: Detaillierte Beschreibungen sind für kleine Systeme recht schnell erstellbar, wenn man erst einmal einen Modellierungsansatz entwickelt hat; um komplexe Systeme mit Hilfe von Statecharts beschreiben zu können, ist eine grundlegende Vorgehensweise, in die die Verhaltensbeschreibung durch Statecharts eingebunden werden kann (z. B. ein objektorientierter Ansatz), unverzichtbar.

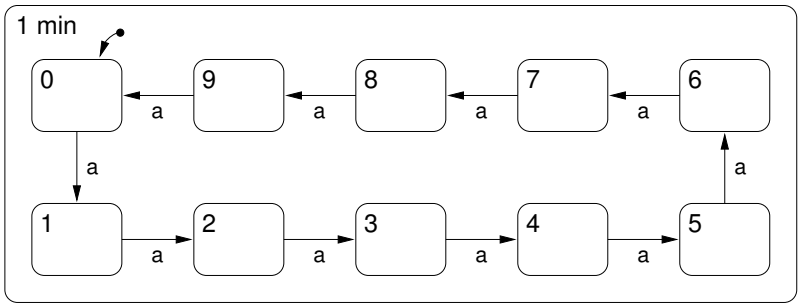
Obwohl der Heimtrainer nur wenige Funktionen aufweist, ist die Beschreibung in Abschnitt 3.3 nicht vollständig. Dies liegt zum einen daran, daß bestimmte Ereignisse nicht berücksichtigt worden sind (hierzu zählt z. B. das Ausfallen der Stromversorgung infolge einer schwachen Batterie), rührt aber vor allem daher, daß grundlegende Aktivitäten nicht mit den Statecharts durch Aktionen verknüpft worden sind. So kann durch den Zustand *time* innerhalb der Komponente *displays* (siehe Abb. 3.19) zwar festgehalten werden, daß die anzuzeigende Information die Zeit ist; die Zeit wird jedoch nicht tatsächlich fortgeschrieben.[10] Die Initialisierung der benötigten Variablen (z. B. von T für die Zeit) erfaßt die Darstellung in Abschnitt 3.3 ebenfalls nicht; sie könnte durch Aktionen erfolgen, die ausgeführt werden, wenn der Zustand *on* betreten wird.

3.4 Anhang: Erweiterungen

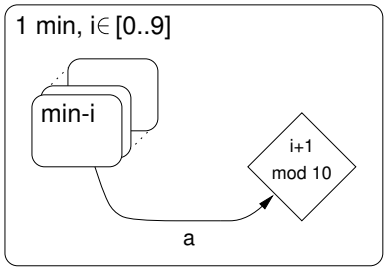
Nachdem in den vorangegangenen Abschnitten die wesentlichen Konzepte des Statechart-Formalismus vorgestellt worden sind, sollen nun einige denkbare Erweiterungen behandelt werden, die Harel ebenfalls in [**?**] beschreibt. Zum Erscheinungszeitpunkt des Artikels war den

[10] Dies könnte durch eine orthogonale Komponente modelliert werden.

meisten dieser zusätzlichen Eigenschaften keine Syntax oder formale Semantik zugeordnet; es handelt sich überwiegend um Ideen, die durch die Anwendung des Formalismus im Rahmen realer Projekte entstanden sind und zu einer weiteren Vereinfachung der Darstellung beitragen können.



(a)



(b)

Abb. 3.23: Zustand mit mehreren Subzuständen gleicher Struktur in (a) sowie eine übersichtlichere Darstellung mit Hilfe eines parametrisierten Zustands in (b)

Häufig besitzen verschiedene Zustände eines Statechart dieselbe innere Struktur und weisen ähnliche Möglichkeiten für einen Zustandswechsel auf. Harel schlägt vor, derartige Zustände in einem sogenannten *parametrisierten Zustand* zusammenzufassen, wobei die ursprünglichen Zustände durch einen Parameter identifiziert werden. In Abb. 3.23(a) ist ein Beispiel dargestellt, das die möglichen Zustände einer minutengenauen einstelligen Anzeige beschreibt. Allen Zuständen ist gemein, daß sie nach einem Zeitsignal a verlassen werden, damit die Folgeziffer angezeigt werden kann. Abb. 3.23(b) zeigt, wie die Darstellung durch einen parametrisierten Zustand vereinfacht wird.

Neben der Möglichkeit, für Zustände, die gemäß der XOR-Dekomposition verfeinert sind, parametrisierte Zustände zu verwenden, ist eine Anwendung dieser Idee auch im Falle von Zuständen mit orthogonalen Komponenten vorstellbar. Harel nennt als ein Beispiel eine Telefonanlage, die 1000 Telefone umfaßt; das Verhalten einer solchen Anlage könnte durch einen Statechart gemäß

Abb. 3.24 beschrieben werden. Sowohl für die XOR-Dekomposition als auch für die AND-Dekomposition ist jedoch, wie Harel betont, häufig die letztendliche Programmiersprache am besten geeignet, um komplizierte Fälle von Parametrisierung zu spezifizieren.

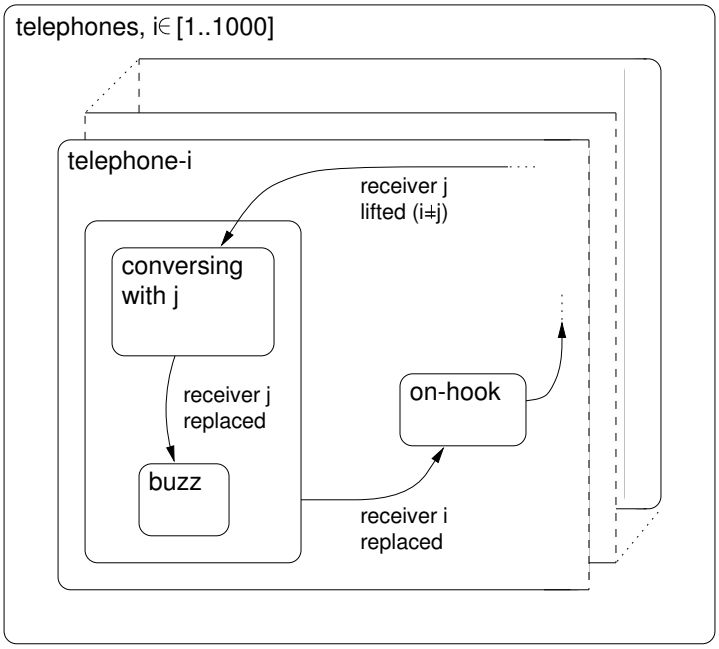


Abb. 3.24: Eine Telefonanlage mit 1000 Telefonen, beschrieben durch einen parametrisierten Zustand

Eine andere Erweiterung des Statechart-Formalismus betrifft die Organisation des Zustandsraumes. Bisher waren die Zustände eines Statecharts stets in einer baumartigen Struktur angeordnet, so daß jeder Zustand – mit Ausnahme desjenigen der höchsten Hierarchieebene – genau einen ihm direkt übergeordneten Zustand aufwies; diese Anordnung ist dem menschlichen Verständnis leicht zugänglich und bildet somit eine gute Arbeitsgrundlage.

Es ist jedoch auch vorstellbar, daß ein Zustand mehrere direkt übergeordnete Zustände hat, wodurch auf der Basis einer *Oder*-Beziehung (OR) *überlappende Zustände* entstehen. Dieser Fall kann z. B. dann eintreten, wenn zwei Subzustände zweier sich ausschließender Zustände in anwendungsfachlicher Hinsicht große Ähnlichkeiten aufweisen oder gar identisch sind, kann aber auch einfach herbeigeführt werden, um weniger Transitionen einzeichnen zu müssen und somit die Komplexität des Diagramms zu verringern. In Abb. 3.25 ist ein Beispiel dargestellt: Dem Zustand E sind die beiden Zustände A und B unmittelbar übergeordnet; er kann nach einem der Ereignisse a, b, c oder d verlassen werden.

Auch Fälle ganz anderer Art, in denen überlappende Zustände nützlich sein können, sind denkbar. So kann es z. B. sein, daß ein Zustand A unter gewissen Umständen als orthogonale Komponente eines Zustandes B auftritt, in anderen hingegen als eigenständiger Zustand. Die zunächst naheliegende Lösung, im Sinne der Abb. 3.26(a) zu modellieren, weist den Nachteil

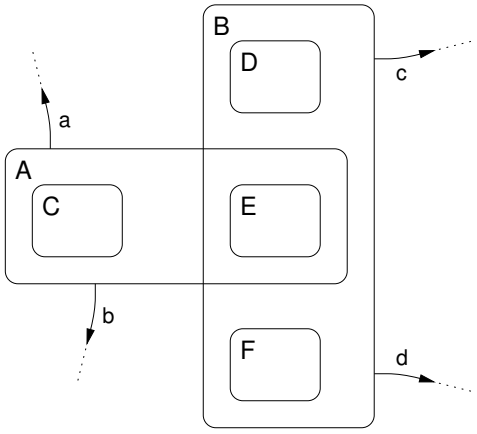


Abb. 3.25: Überlappende Zustände auf der Grundlage einer Oder-Beziehung

der Redundanz auf, die sich besonders bei einer komplexen Struktur von A negativ bemerkbar macht.

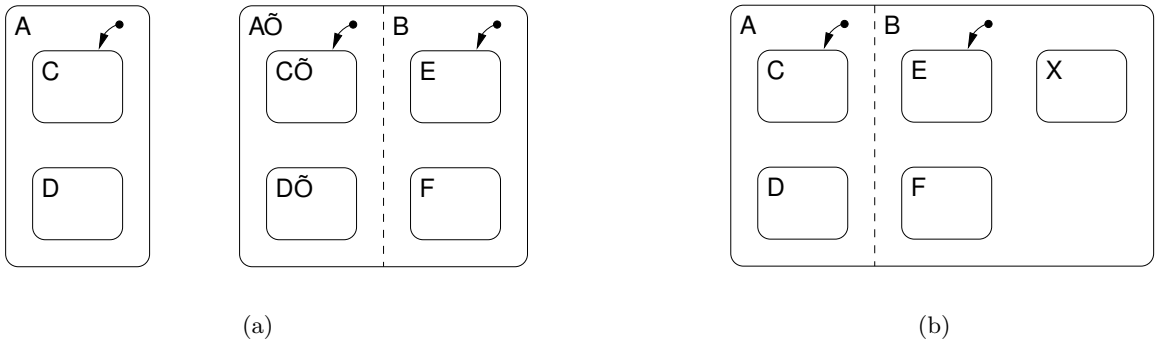


Abb. 3.26: Zustand mit und ohne zu ihm orthogonalen Zustand (Redundanz in (a) und „künstliche“ Lösung in (b))

Auch eine Lösung gemäß Abb. 3.26(b) ist nicht zufriedenstellend: Ein zusätzlich eingeführter Zustand X soll eingenommen werden, wenn A als eigenständiger Zustand und B als „inaktiv“ aufgefaßt werden soll. Harel schlägt für Fälle der vorliegenden Art wiederum die Verwendung überlappender Zustände vor, so daß sich eine Darstellung wie in Abb. 3.27 ergibt: A und A' sind die Zustände C und D gemein.

Durch das Ereignis a kann ein Übergang nach (C, E) ausgelöst werden, während das Ereignis b dazu führt, daß lediglich A' eingenommen und B nicht aktiv wird. Das Ereignis c gestattet es, den Zustand C unabhängig davon, ob sich das System in A oder A' befindet, zu verlassen; hingegen führt d nur dann dazu, daß C verlassen wird, wenn B aktiv ist. Mittels e kann der Zustand D betreten werden, wobei der Bogen im zum Ereignis e gehörigen Pfeil anzeigt, daß

die Begrenzung von A' nicht überschritten, der Produktzustand aus A und B betreten und somit der Folgezustand (D, E) eingenommen wird. Bei dem zum Ereignis f gehörigen Pfeil liegt ein anderer Fall vor: Mit D wird gleichzeitig A' eingenommen, und B ist inaktiv.

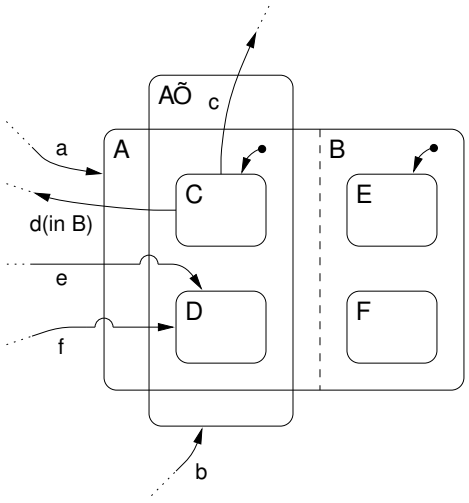


Abb. 3.27: Verbesserung der Darstellungen aus Abb. 3.26
mit Hilfe überlappender Zustände

Obwohl die grafische Umsetzung der Idee überlappender Zustände sehr anschaulich ist, bestehen einige Probleme, die die Verwendungsmöglichkeiten dieses Konzeptes erheblich einschränken können. So muß z. B. berücksichtigt werden, daß ein übergeordneter Zustand, dessen Begrenzungslinie übersprungen werden soll, dennoch vom System einzunehmen ist, wenn der zugehörige Pfeil keine Begrenzung eines anderen nicht atomaren Zustands, der ebenfalls dem letztlichen Zielzustand übergeordnet ist, kreuzt. Eine ausführliche Betrachtung der Möglichkeiten, die überlappende Zustände bieten, sowie der durch sie entstehenden Schwierigkeiten findet man in [**?**]. Die dortige Darstellung zeigt auch, daß eine geeignete formale Syntax und vor allem eine passende formale Semantik für überlappende Zustände ausgesprochen umfangreich sind, so daß die resultierende Komplexität einer entsprechenden Beschreibung den Nutzen der eleganten Darstellung zunichte machen kann.

Als eine weitere Ergänzung des Statechart-Formalismus zieht Harel den Gebrauch *temporaler Logik* in Betracht. Es wäre dann möglich, die Spezifikation eines Systemverhaltens um allgemeine Aussagen zu erweitern, die z. B. Verklemmungsfreiheit oder die Einhaltung bestimmter globaler Zeitbeschränkungen garantieren könnten. Harel hält folgende Arten der Integration temporaler Logik für denkbar:

- Für eine gegebene Spezifikation mit Hilfe von Statecharts kann untersucht werden, ob sie einer Menge von Sätzen der temporalen Logik genügt.
- Menschen denken häufig in einer linearen, auf Szenarien basierenden Weise. Da sich Szenarien mit Hilfe temporaler Logik in geeigneter Form ausdrücken lassen, kann versucht werden, Statecharts aus Sätzen der temporalen Logik abzuleiten.

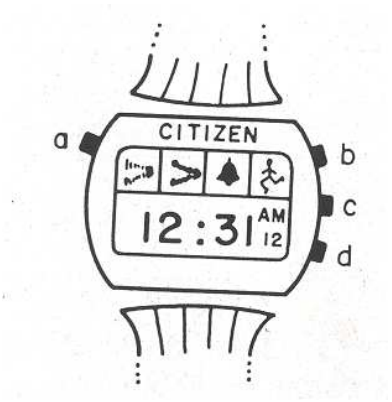


Abbildung 3.28: Citizen Quartz Multi-Alarm III

- Temporale Logik kann in Statecharts selbst verwendet werden, um komplexe Bedingungen zu formulieren. So ist es bisher nur mit Hilfe der History-Funktion möglich, das bisherige Systemverhalten zur Steuerung weiterer Abläufe einzusetzen. Durch die Verwendung temporaler Logik werden Bedingungen der Form

  $$(\text{in } B) \wedge \neg(\text{in } C) \textbf{ seit } (\text{in } A)$$

  möglich.

Auch das Konzept der *Rekursion* kann nach Ansicht Harels in den Statechart-Formalismus integriert werden; bei Zustandsübergängen könnte dann über einen Bezeichner in einen anderen Statechart verzweigt werden, welcher aus einem terminalen Zustand heraus wieder verlassen werden könnte.[11] In Anlehnung an Markov-Ketten, die durch eine besondere Form endlicher Automaten dargestellt werden können, sind darüber hinaus *probabilistische Statecharts* denkbar, die sich dadurch auszeichnen, daß anhand festgelegter Wahrscheinlichkeiten nichtdeterministisch Übergänge wählbar sind.

3.5 Aufgabe

Aufgabe 3.1

Die folgende Beschreibung der Steuerung einer elektonischen Arbanduhr ist der Arbeit: D. Harel: Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming 8 (1987) 231-274 entnommen. Vergleichen Sie die verbale Beschreibung mit der Spezifikation durch den Harel-Graphen. Klären Sie jede der dort dargestellten Funktionen.

[11] Ein ähnlicher Ansatz wird im dynamischen Modell von OMT verfolgt ([**?**] und [**?**]).

Let us know introduce our running example. The Citizen Quartz Multi-Alarm III Watch has a main display area and four smaller ones, a two-tone beeper, and four control buttons denoted here a, b, c and d. See Fig. 3.28.

It can display the time (with am/pm or 24 hour time modes) or the date (day of month, month, day of week), it has a chime (beeps on the four if enabled), two independent alarms, a stopwatch (with lap and regular display modes, and a 1/100 s display), a light for illumination, a weak battery blinking indication, and a beeper test. We shall assume throughout that the main functions of these are known, and will use liberal terminology, such as 'power weakens' to denote certain events of obvious meaning, though , of course, to make things comlete one would have to tie these events up with actual happenings in the physical parts of the system, or to specify them as output events produced in other, separately specified, components. The main external events will be the depressing and releasing of buttons (e.g., event a denotes button a being depressed, and $\hat{a}$ denotes it being released), and there will be certain internal ones too. (...) We remark here that while the description of the watch presented herein is intended to be as faithful to its actual workings as possible, there are some very minor differences that are not worth dwelling upon here, and that most likely will not be detected in normal use of the watch. The point is, however, that the statechart of the watch (cf. Fig. 3.29, 3.30) was obtained by the author using the obviously inappropriare method of observation from the final product; had it been the basis for the initial specification and design of that final product, in the spirit of the gradual development presented below, the undescribed anomalies might have been avoided. A refinement of the displays state yields Fig. 9 in which it has been decided that there will be a cycle of displays linked by repeated depressings of a; that the time and date displays are linked by d's but that the time display will resume after 2 minutes in date. Also, the time display is the default, meaning, among other things, that the entrances from alarms-beep in Fig. 3.29, 3.30 will actually be entrances to time.

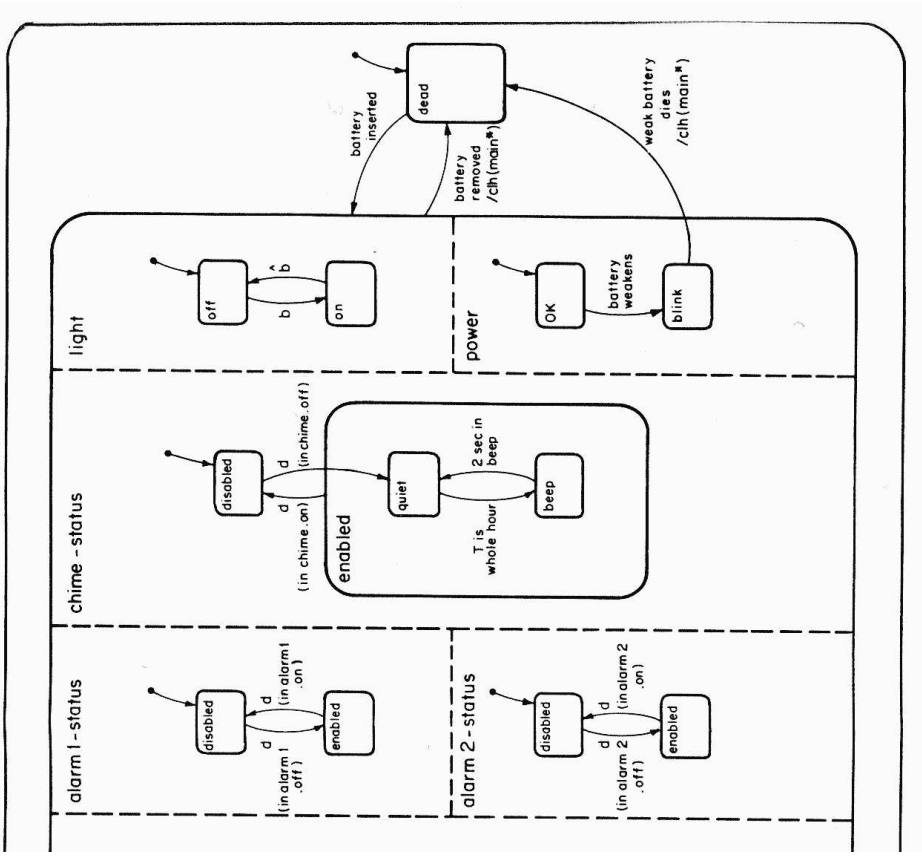


Abbildung 3.29: Harel-Graph für die Citizen Uhr (Teil 1)

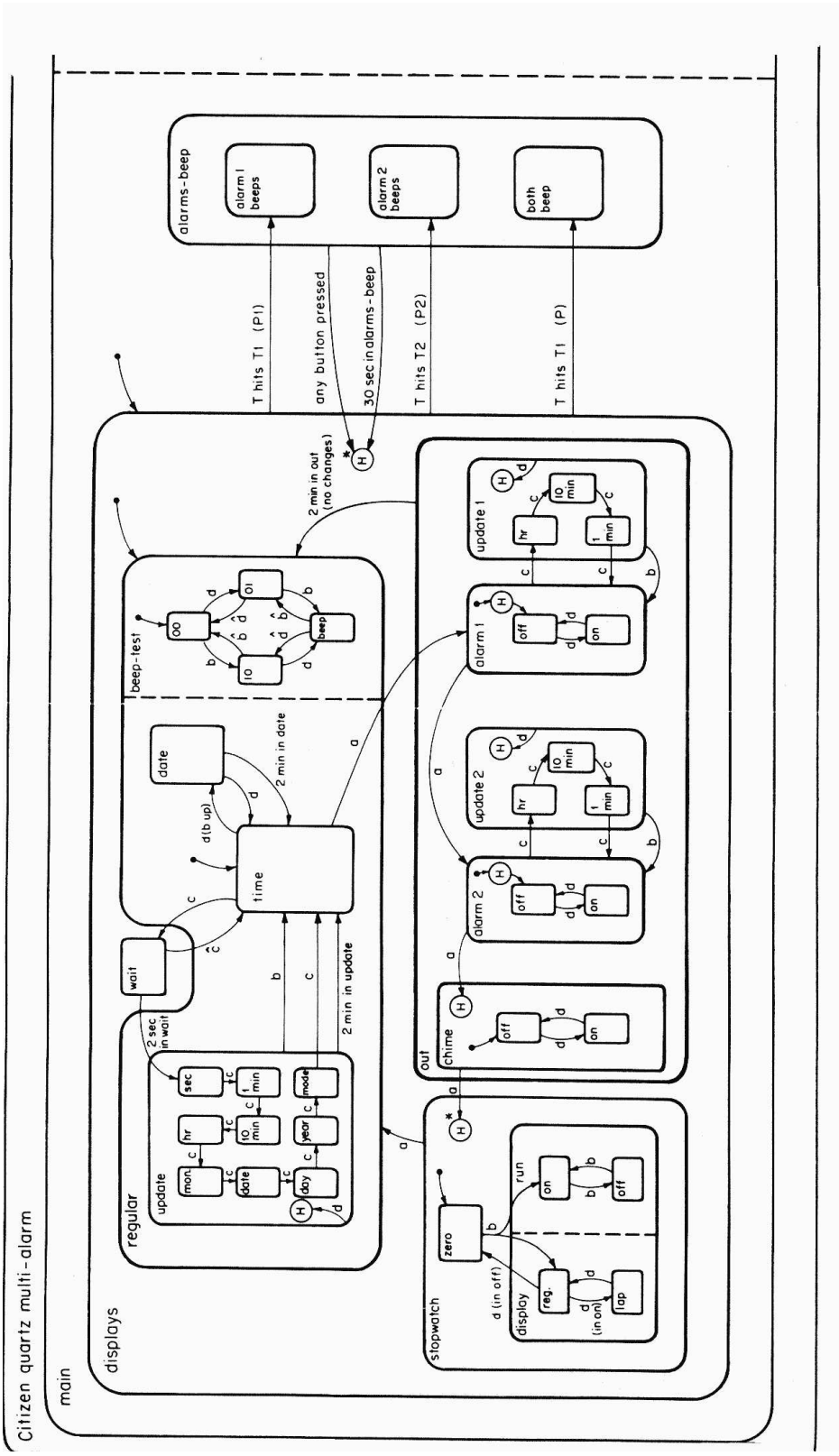


Abbildung 3.30: Harel-Graph für die Citizen Uhr (Teil 2)

Kapitel 4

Referenznetze

4.1 Einleitung

Dieses Kapitel führt die Einführung von Petrinetzen aus der Vorlesung F4 fort. Dort wurden bereits einige der im Folgenden dargestellten Vorteile deutlich.

1.) graphische und äquivalente algebraische/textuelle Darstellung

2.) (formal abgesicherte) Algorithmen für die Analyse

3.) Abstraktion und hierarchische Strukturen

4.) hoch entwickelte Theorie der Nebenläufigkeit (concurrency)

5.) Rechnerwerkzeuge für Editieren, Simulation und Analyse

6.) Universalität in Anwendbarkeit (Anwendungen in fast allen Gebieten)

7.) Varianten des gleichen Modellierungskonzeptes (Zeit-Netze, stochastische Netze, high-level, objektorientiert, ...)

Dies bewirkt unter anderem, dass sie "einfach" zu vermitteln sind, dass Werkzeuge "einfach" miteinander verknüpft werden können sowie die Verträglichkeit von verschiedenen Abstraktionsebenen.

Hierarchiebildung bei der Systemmodellierung war ein wesentliches Konzept der Harel-Graphen im vorangehenden Kapitel. Die entsprechenden Begriffe der Vergröberung und Verfeinerung werden hier zunächst bei einfachen Netzen vorgestellt. Danach werden verstärkt gefärbte Netze und darüber hinaus Referenznetze behandelt. Letztere erlauben die Kommunikation zwischen verschiedenen Netzen über Kanäle, die Instanziierung von Netzen anhand von Klassenmustern und die Modellierung von Netzen in Netzen, d.h. die Marken sind Referenzen auf andere Netze.

4.2 Einfache Netze

Zunächst erinnern wir an die Definition eines Netzes. Ein Beispiel ist in Abbildung 4.10 gegeben. Sein Verhalten wird im Abschnitt 4.3.1 erklärt, wobei natürlich die angegebene Anfangsmarkierung eine wichtige Rolle spielt. Die folgende Definition eines Netzes enthält jedoch noch nicht den Begriff der Markierung, da diese für die darauf aufbauenden Definitionen von P/T-Netzen oder gefärbten Netzen individuell erfolgt.

Definition 4.1 *Ein* Netz *ist ein Tripel* $\mathcal{N} = (P, T, F)$*, wobei*

- P *eine Menge von* Plätzen *(oder* Stellen*) (places),*
- T *eine mit* P *disjunkte Menge von* Transitionen *(transitions) und*
- F *die* Flussrelation *(flow relation)* $F \subseteq (P \times T) \cup (T \times P)$ *darstellt.*

Falls P *und* T *endlich sind, dann heißt auch das Netz* $\mathcal{N}$ *endlich.*

4.2.1 Verfeinerung und Komposition

Die Konstruktion von Systemhierarchien durch Vergröberung (Abstraktion) und Verfeinerung ist ein wichte Methode des Systementwurfs. Petrinetze unterstützen dies durch besondere mit ihrer Struktur kompatible Konzepte. Diese werden unhabhängig von Markierungen und speziellen Netzmodellen gebildet und daher für einfache Netze definiert. Wir beginnen mit dem Begriff des *Randes* einer Menge von Plätzen und Transitionen, der die Schnittstelle des zu vergröbernden Teiles bilden wird.

Sei $\mathcal{N} = (P, T, F)$ ein Netz, $X := P \cup T$ und $Y \subseteq X$ eine Menge von Elementen. Dann heißt $\partial(Y) := \{y \in Y | \exists x \notin Y \,.\, x \in loc(y)\}$ der *Rand* (border) (der Menge Y). Y heißt *Platz-berandet* (place-bordered) oder *offen*[1], wenn $\partial(Y) \subseteq P$, und *Transitions-berandet* (transition-bordered) oder *abgeschlossen*[1], falls $\partial(Y) \subseteq T$. Um eine Vergröberung mit der Netzstruktur verträglich zu gestalten, sollten im Normalfall Platz- bzw. Transitions-berandete Mengen durch einen Platz bzw. eine Transition ersetzt werden.

Anmerkung: Eine Menge Y kann gleichzeitig offen *und* abgeschlossen sein, wie z.B.: $Y := P \cup T$. In diesem Fall hängt es von der Interpretation bzw. Anwendung ab, ob Y durch einen Platz oder eine Transition ersetzt wird.

Die Menge $Y = \{p_3, p_4, t_2, t_3, t_4\}$ des Netzes in Abb. 4.1 ist Transitions-berandet und wird daher zu einer Transition t_Y vergröbert. Auf diese Weise erhält man wieder ein Netz, das in Abb. 4.2 dargestellt ist. Diese Operation wird nun formalisiert.

[1] Offene und abgeschlossene Mengen definieren eine Topologie, die eine Formalisierung von Nachbarschaft auf der graphischen Struktur von Netzen darstellt.

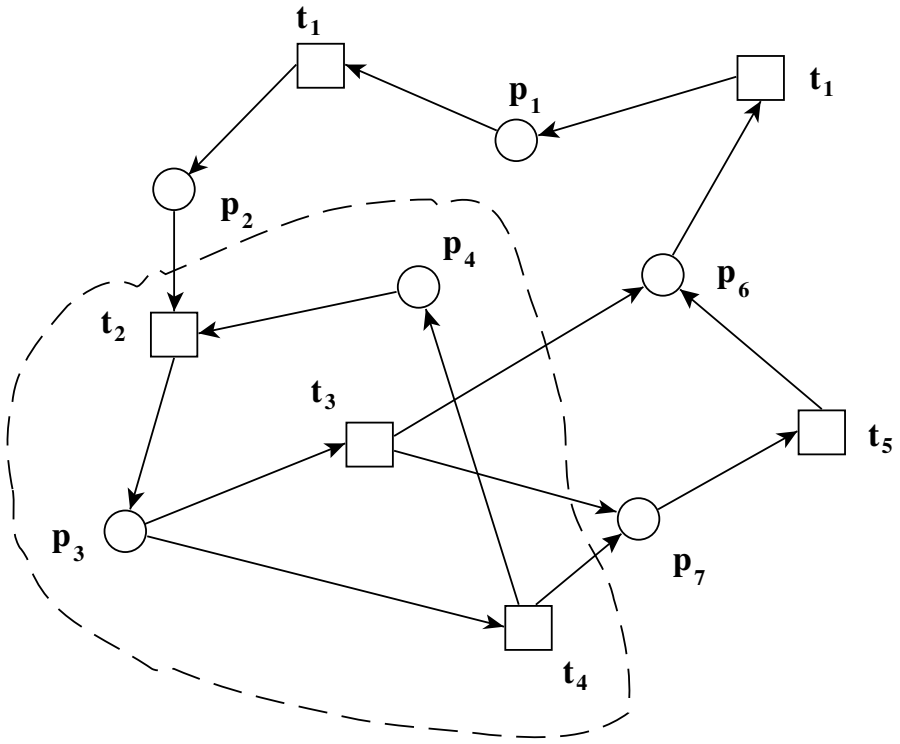


Abbildung 4.1: Eine Transitions-berandete Menge

Sei $\mathcal{N} = (P, T, F)$ ein Netz und Y eine nicht leere Transitions-berandete Menge von Elementen. Dann heißt $\mathcal{N}[Y] = (P[Y], T[Y], F[Y])$ *einfache Vergröberung* (simple abstraction) von $\mathcal{N}$ in Bezug auf Y falls gilt: $P[Y] = P \backslash Y$, $T[Y] = (T \backslash Y) \cup \{t_Y\}$, wobei t_Y ein neues Element ist, $F[Y] = \{(x,y) | x \notin Y \wedge y \notin Y \wedge (x,y) \in F\} \cup \{(x,t_Y) | x \notin Y \wedge \exists y \in Y \,.\, (x,y) \in F\} \cup \{(t_Y,x) | x \notin Y \wedge \exists y \in Y \,.\, (y,x) \in F\}$. $P[Y]$ enthält alle Plätze mit Ausnahme derjenigen aus Y. $T[Y]$ enthält alle Transitionen mit Ausnahme derjenigen aus Y und ein neues Element t_Y. $F[Y]$ ist die Vereinigung von 3 Kantenmengen, nämlich (1) derjenigen, die kein Ende in Y haben, (2) derjenigen die von außerhalb von Y zu t_Y führen und (3) derjenigen, die von t_Y nach Außerhalb führen.

Entsprechend, wenn Y eine Platz-berandete Menge ist, dann erhält man $\mathcal{N}[Y] = (P[Y], T[Y], F[Y])$ durch $P[Y] = (P \backslash Y) \cup \{p_Y\}$, wobei p_Y ein neues Element ist, $T[Y] = T \backslash Y$, $F[Y] = \{(x,y) | x \notin Y \wedge y \notin Y \wedge (x,y) \in F\} \cup \{(x,p_Y) | x \notin Y \wedge \exists y \in Y \,.\, (x,y) \in F\} \cup \{(p_Y,x) | x \notin Y \wedge \exists y \in Y \,.\, (y,x) \in F\}$.

Anmerkung: Die Definition von $\mathcal{N}[Y]$ ist mehrdeutig, falls Y gleichzeitig Platz- und Transitions-berandet ist. Dann schreiben wir $\mathcal{N}[Y^{(p)}]$ falls Y als Platz-berandete Menge aufgefasst wird und $\mathcal{N}[Y^{(t)}]$ im anderen Fall.

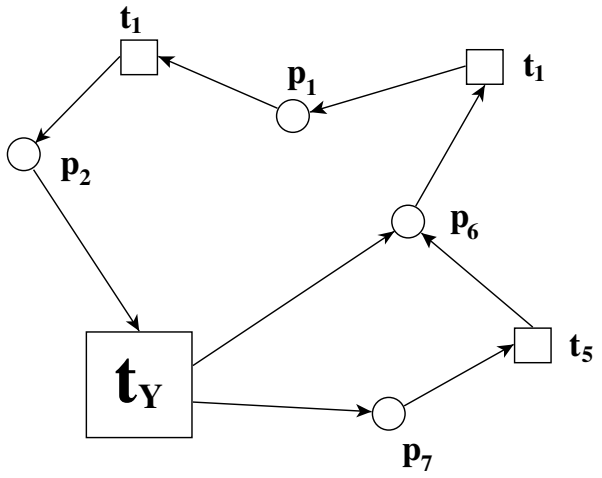


Abbildung 4.2: Vergröberung des Netzes aus Abbildung 4.1

Definition 4.2 *a) Wenn $\mathcal{N}_2 = \mathcal{N}_1[Y]$ eine einfache Vergröberung von $\mathcal{N}_1$ für eine Platz- oder Transitions-berandete Menge Y ist, dann heißt $\mathcal{N}_1$* einfache Verfeinerung *(simple refinement) von $\mathcal{N}_2$. Für eine Menge $\{Y_1, Y_2, \ldots, Y_n\}$ von paarweise disjunkten, Platz- oder Transitions-berandeten Teilmengen von $P_1 \cup T_1$ wird $\mathcal{N}_2 = (\ldots((\mathcal{N}_1[Y_1])[Y_2])\ldots[Y_n])$* Vergröberung *(abstraction) von $\mathcal{N}_1$ genannt und $\mathcal{N}_1$ ist eine* Verfeinerung *(refinement) von $\mathcal{N}_2$. $\mathcal{N}_2$ wird durch $\mathcal{N}_2 = \mathcal{N}_1[Y_1, Y_2, \ldots, Y_n]$ bezeichnet.*

b) Eine Vergröberung $\mathcal{N}_2 = \mathcal{N}_1[Y_1, Y_2, \ldots, Y_n]$ von $\mathcal{N}_1$ wird als strikt *(strict) bezeichnet, wenn Y_i entweder eine Menge von Plätzen, d.h. $Y_i \subseteq P_1$, oder eine Menge von Transitionen, d.h. $Y_i \subseteq T_1$, ist. In der Definition einer strikten Abstraktion wird Y_i im ersten Fall durch einen Platz p_{Y_i} und im zweiten Fall durch eine Transition t_{Y_i} ersetzt. $\mathcal{N}_1$ wird* strikte Verfeinerung *von $\mathcal{N}_2$ genannt.*

Durch folgende Konvention können Mehrdeutigkeiten vermieden werden: falls in a) oder b) eine Menge Y_i $(1 \leq i \leq n)$ sowohl Platz- als auch Transitions-berandet ist, kann die Vergröberung durch $\mathcal{N}_2 = \mathcal{N}_1[Y_1, \ldots, Y_i^{(d)}, \ldots, Y_n]$ bezeichnet werden, wobei $d = p$ bzw. $d = t$ ist und Y_i als a Platz- bzw. Transitions-berandete Menge betrachtet wird.

Abbildung 4.3 zeigt eine nicht einfache Vergröberung. Das obere Netz stellt das aus der Vorlesung F1 bekannte Beispielnetz zum Start eines Autorennens dar. Dabei sind die vergröberten Mengen $Y = \{t_1, t_2, t_3, t_4, t_5, p_2, p_4, p_5, p_8, p_9, p_{11}\}$, $Y_1 = \{t_6, p_{13}, t_7\}$ und $Y_2 = \{t_8, p_{15}, t_9\}$, in der oberen Abbildung durch eine gestrichelte Linien dargestellt. Es handelt sich um drei Transitions-berandete Mengen, die zu den Transitionen t_Y, t_{Y_1} und t_{Y_2} im Netz $\mathcal{N}[Y, Y_1, Y_2]$ der unteren Abbildung verwandelt werden.

Anmerkung: In der Literatur und auch witer unten in diesem Skript werden strikte Vergröberungen als *Faltungen* (foldings) bezeichnet. Eine Faltung wird jedoch als ein spezieller Netzmorphismus eingeführt. Im Abschnitt 4.2.2 wird die Äquivalenz von (Epi-)Faltungen und

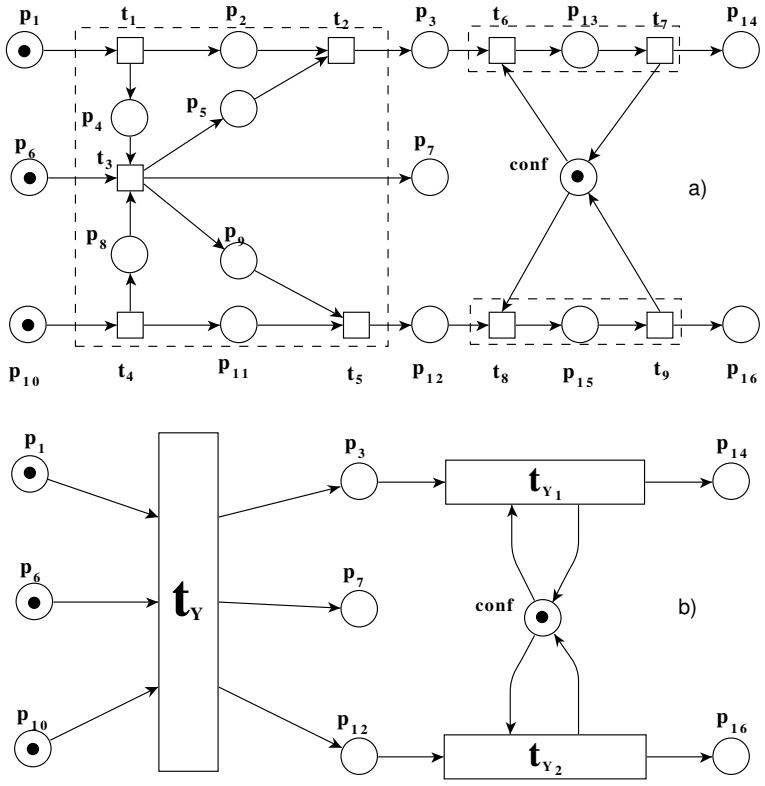


Abbildung 4.3: Eine (nicht einfache) Vergröberung

strikten Vergröberungen formal gezeigt werden. Dadurch wird dann diese Bezeichungsweise gerechtfertigt.

Es ist einfach zu zeigen, dass die Vergröberung eines Netzes wieder ein Netz ist, d.h. der Definition 4.1 genügt. Die Vergröberung im unteren Teil von Abbildung 4.3 hat sinngemäß das entsprechende Verhalten des Netzes darüber. Eine Vergröberung muss jedoch nicht eine sinnvolle Interpretation haben. Insbesondere überträgt sich der Begriff nicht zwangsläufig auf seine Semantik (d.h. die Menge seiner Prozesse). Als Beispiel betrachte man das Netzfragment in Abbildung 4.4 a). Intuitiv hat die Vergröberung in Abbildung 4.4 d) das entsprechende Verhalten: Marken können von links nach rechts "durchlaufen". Allerdings ist dieses Netz auch Vergröberung von Abbildung 4.4 d). Das Verhalten ist hier jedoch völlig verschieden.

Die Vergröberung in Abbildung 4.4d lässt sich in diesem Fall sinnvoll interpretieren, und zwar als Verschmelzung zweier Plätze als Schnittstelle zweier Komponenten. Die Operation wird als *Verschmelzung* oder *Fusion* bezeichnet und zuweilen wie in 4.4c dargestellt. Die Abbilsungen 4.4e-h. zeigen die entsprechenden Fälle für Transitions-berandete Mengen. Es handelt sich um die *Verschmelzung* oder *Fusion* von Transitionen.

Mit Abbildung 4.5 sind zwei Netze gegeben, deren Komposition durch Platzverschmelzung das Netz von Abbildung 4.3 a) ergibt.

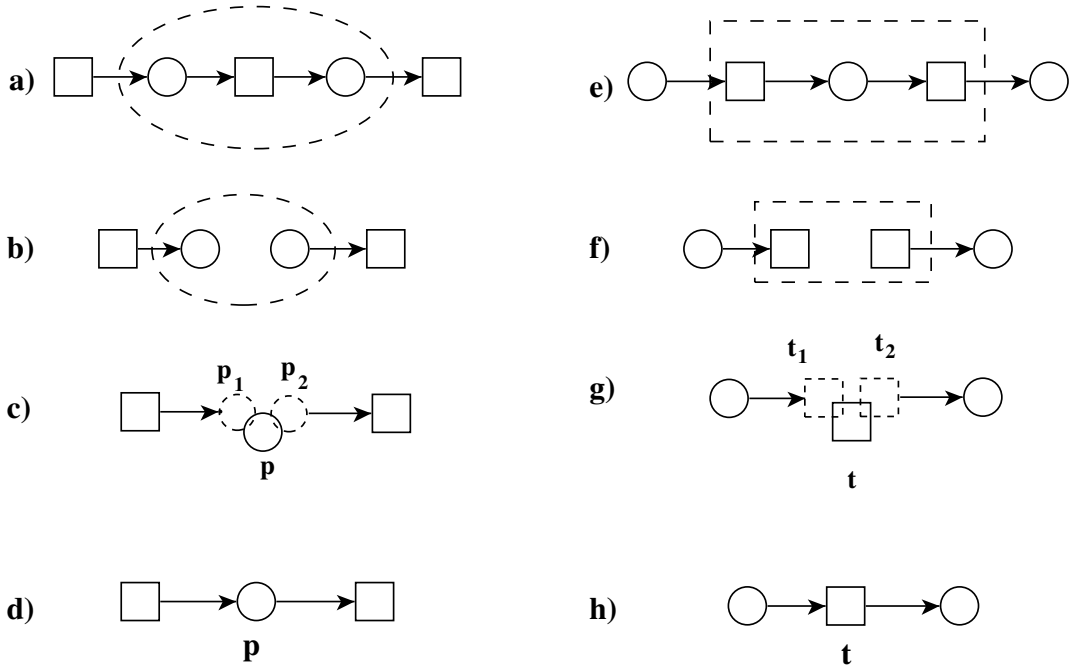


Abbildung 4.4: Vergröberung und Verschmelzung (Fusion)

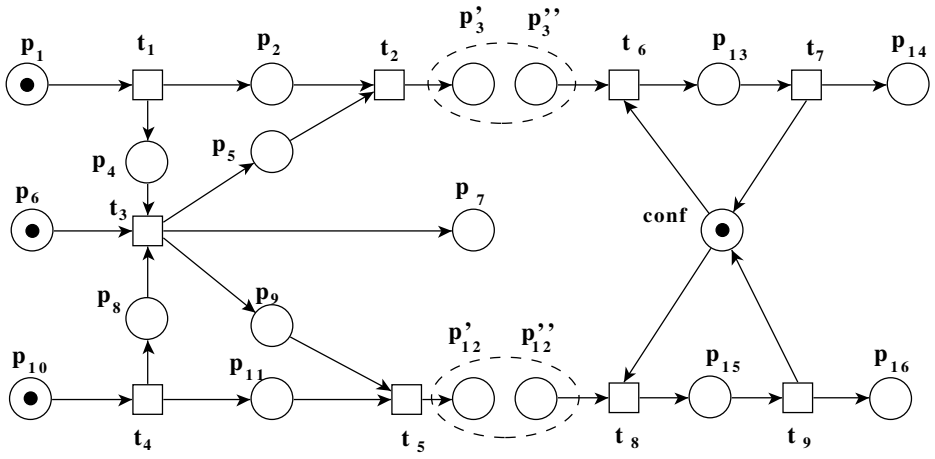


Abbildung 4.5: Komposition durch Verschmelzung (Fusion)

4.2.2 Netzmorphismen

Wie in algebraischen Theorien allgemein üblich, werden strukturerhaltende Abbildungen *Homomorphismen* oder kürzer *Morphismen* genannt. Um strukturverträgliche Transformationon, wie sie beispielsweise in Werkzeugen benötigt werden, zu definieren, wurde der Begriff auch für Netze eingeführt. Er stellt die Basis für viele Operationen, wie Vergröberung, Verfeinerung und Fusion, dar.

Folgender Ansatz ist naheliegend. Für gegebene Netze $\mathcal{N}_1 = (P_1, T_1, F_1)$ und $\mathcal{N}_2 = (P_2, T_2, F_2)$, könnte man in einem ersten Schritt einen *Netzmorphismus* als eine Abbildung $\varphi : P_1 \cup T_1 \to P_2 \cup T_2$ definieren, die die durch die Netzkanten in F gegebene Struktur erhält:

$$\forall x, y \in P_1 \cup T_1 \,.\, (x, y) \in F_1 \Rightarrow (\varphi(x), \varphi(y)) \in F_2 \vee \varphi(x) = \varphi(y)$$

Sie wird daher als *F-erhaltend* bezeichnet. In Abbidung 4.6 sind zwei F-erhaltende Abbildungen eines Netzes $\mathcal{N}_1$ in ein Netz $\mathcal{N}_2$ durch gestichelte Pfeile dargestellt. Währemd im Fall b) damit tatsächlich eine Vergröberung (im Sinne von Definition 4.2) beschrieben wird, gilt dies nicht für den Fall a).

Woran liegt dies?

Als einen Grund dafür erkennt man die Tatsache, dass Platz-berandete Mengen (z.B. die Menge {p'}) Bild von Mengen sind, die diese Eigenschaft nicht haben. Entsprechendes gilt für Transitions-berandete Mengen (z.B. {t'}). In diesem Abschnitt werden Platz-berandete (bzw. Transitions-berandete) Mengen als offen (bzw. abgeschlossen) bezeichnet (siehe Seite 74 und dortige Fußnote). In dieser Termonologie entspricht der gerade beschriebene Fall der Forderung, dass offene (bzw. abgeschlossene) Mengen ebensolche Urbilder haben.[2]

Mit einer solchen Bedingung kann eine Kante $(p, t) \in F$ nicht einer Kante $(f(p), f(t)) \in F'$ zugeordnet werden, bei der $f(p)$ kein Platz oder $f(t)$ keine Transition ist. In der nun folgenden vollständigen Definition eines Netzmorphismus wird dazu die von Petri eingeführte *at-Relation* A und die zugehörige Eigenschaft der *A-Erhaltung* [**?**] definiert.

Definition 4.3 *Seien $\mathcal{N}_1 = (P_1, T_1, F_1)$ und $\mathcal{N}_2 = (P_2, T_2, F_2)$ zwei Netze und $\varphi : X_1 \to X_2$ mit $X_i = P_i \cup T_i$. Die at-Relation wird jeweils für $i \in \{1, 2\}$ durch $A_i := (F_i \cup F_i^{-1}) \cap (P_i \times T_i)$, where $i \in \{1, 2\}$ definiert.*

a) φ heißt Netzmorphismus *falls:*

1. $\forall x, y \in P_1 \cup T_1 \,.\, (x, y) \in F_1 \Rightarrow (\varphi(x), \varphi(y)) \in F_2 \vee \varphi(x) = \varphi(y)$
   ("F-preservation"),
2. $\forall x, y \in P_1 \cup T_1 \,.\, (x, y) \in A_1 \Rightarrow (\varphi(x), \varphi(y)) \in A_2 \vee \varphi(x) = \varphi(y)$
   ("A-Erhaltung", "Steigkeit").

b) Ein Netzmorphismus φ heißt Faltung *, falls $\varphi(P_1) \subseteq P_2$ und $\varphi(T_1) \subseteq T_2$.*

[2] Dies ist genau die Definition einer stetigen Abbildung im Sinne der mathematischen Topologie.

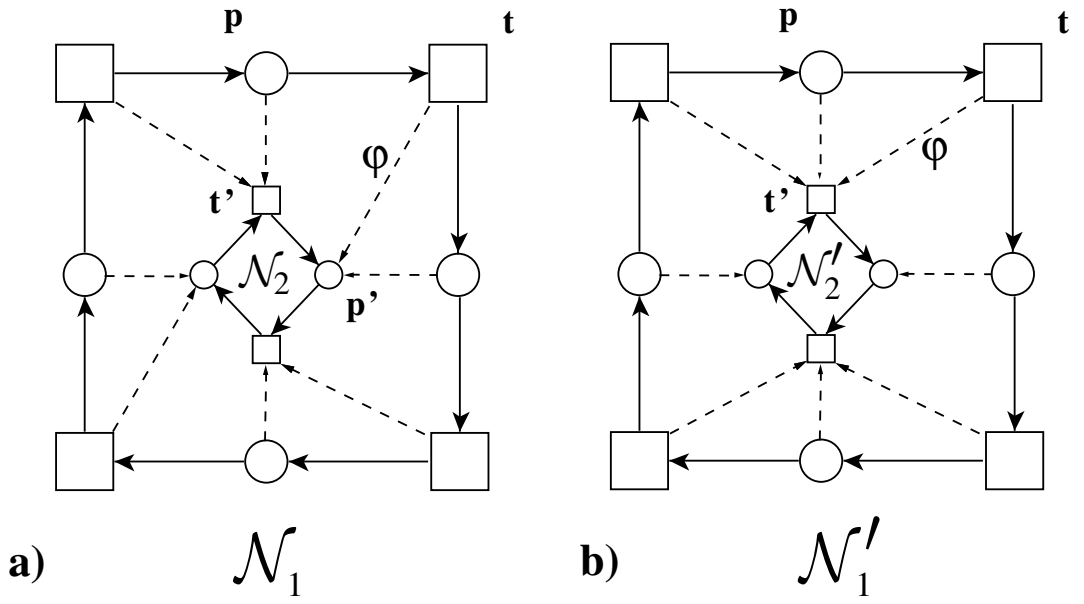


Abbildung 4.6: Zwei F-erhaltende Abbildungen

c) Ein Netzmorphismus φ ist ein Epimorphismus*, falls φ surjective ist und für jedes $(x_2, y_2) \in F_2$ eine Kante $(x_1, y_1) \in F_1$ mit $(x_2, y_2) = (\varphi(x_1), \varphi(y_1))$ existiert. Eine Faltung mit dieser Eigenschaft wird* Epifaltung *genannt.*

d) Ein Netzmorphismus φ ist ein Netzisomorphismus*, falls φ eine Bijektion ist and φ^{-1} ebenfalls ein Netzmorphismus ist.*

Die at-Relation beschreibt die "Nachbarschaft" der Elemente eines Netzes. Die Abbildung φ in Bild 4.6a) ist F-erhaltend aber nicht A-erhaltend. Eine äquivalente Definition eines Netzmorphismus aus [**?**] kommt ohne die explizite Einführung der at-Relation aus:

Eine Abbildung $\varphi : X_1 \to X_2$ heißt *Netzmorphismus*, wenn folgendes gilt:

1. $(x, y) \in F_1 \cap (P_1 \times T_1) \Rightarrow (\varphi(x), \varphi(y)) \in F_2 \cap (P_2 \times T_2) \vee \varphi(x) = \varphi(y)$ und
2. $(x, y) \in F_1 \cap (T_1 \times P_1) \Rightarrow (\varphi(x), \varphi(y)) \in F_2 \cap (T_2 \times P_2) \vee \varphi(x) = \varphi(y)$.

Die folgende Definition und das folgende Lemma werden im Beweis des Hauptergebnisses (Satz 4.7) dieses Abschnittes benutzt. Die Eigenschaften wurden in [**?**] unter der Bezeichnung *vicinity respecting property* behandelt.

Definition 4.4 *Sei φ eine Abbildung wie in Definition 4.3. Dann heißt φ*

a) lokal abgeschlossen, *falls* $p \in P_1 \wedge \varphi(p) = t' \in T_2 \Rightarrow \varphi(loc(p)) = \{t'\}$ *und*

b) lokal offen, *falls* $t \in T_1 \wedge \varphi(t) = p' \in P_2 \Rightarrow \varphi(loc(t)) = \{p'\}$.

Dabei ist $loc(t) := \{t\} \cup {}^\bullet t \cup t^\bullet$. *die* Lokalität *von t (siehe Definition 3.2 im F4-Skript).*

Lemma 4.5 *Wenn φ is A-erhaltend ist, dann ist φ sowahl lokal abgeschlossen als auch lokal offen. Der umgekehrte Schluss gilt, wenn φ als F-erhaltend vorausgetzt wird.*

Beweis:
Um φ als lokal abgeschlossen zu beweisen, nehmen wir an: $p \in P_1$ und $\varphi(p) = t' \in T_2$. Falls $t \in loc(p)$, dann ist zu zeigen: $\varphi(t) = t'$. In der Tat, wenn $t \in {}^\bullet p$ und $(t,p) \in F$, dann folgt $(p,t) \in F^{-1}$ und $(p,t) \in A_1$. Falls $t \in p^\bullet$ und $(p,t) \in F$, dann auch $(p,t) \in A_1$. Also gilt wegen der angenommenen A-Erhaltung auch $(\varphi(p), \varphi(t)) \in A_2$ bzw. $\varphi(p) = \varphi(t)$. Weiter kann $(\varphi(p), \varphi(t)) \in A_2$ nicht gelten, da $\varphi(p) \in T_2$, womit $\varphi(p) = \varphi(t)$ bewiesen ist. In entsprechender Weise ist zu zeigen, dass φ lokal offen ist.

Um den zweiten Teil der Behauptung zu zeigen, nehme man an. dass $(x,y) \in A_1$ und $(\varphi(x), \varphi(y)) \notin A_2$. Wir müssen nun $\varphi(x) = \varphi(y)$ beweisen.

Es sind drei Fälle zu behandeln: a) $\varphi(x) \notin P_2$, b) $\varphi(y) \notin T_2$ und c) $(\varphi(x), \varphi(y)) \notin F_2 \cup F_2^{-1}$. Wegen $(x,y) \in A_1$ gilt $x \in P_1$, $y \in T_1$ und $y \in loc(x)$. Da φ lokal abgeschlossen ist, folgt für Fall a) die Gleichheit $\varphi(x) = \varphi(y)$. Da φ lokal offen ist, impliziert Fall b) auch $\varphi(x) = \varphi(y)$. Es bleibt also Fall c). Mit der Annahme, dass φ F-erhaltend ist, folgt aus $y \in loc(x)$ die Beziehung $(\varphi(x), \varphi(y)) \in F_2 \cup F_2^{-1}$, womit Fall c) zu einem Widerspruch führt und daher nicht gelten kann.. □

Aufgabe 4.6

a) Erläutern Sie das Lemma 4.5 anhand der Bilder 4.6a und 4.6b.

b) Zeigen Sie mit Hilfe der Abbildung 4.7, dass die in Lemma 4.5 für die Umkehrung geforderte Bedingung unverzichtbar ist.

Wir kommen nun zu dem Hauptergebnis dieses Abschnittes.

Satz 4.7 *Seien $\mathcal{N}_1 = (P_1, T_1, F_1)$ und $\mathcal{N}_2 = (P_2, T_2, F_2)$ zwei endliche Netze.*

a) $\mathcal{N}_2$ ist genau dann eine Vergröberung von $\mathcal{N}_1$, wenn es einen Epimorphismus von $\mathcal{N}_1$ nach $\mathcal{N}_2$ gibt.

b) $\mathcal{N}_2$ ist genau dann eine strikte Vergröberung von $\mathcal{N}_1$, wenn es eine Epifaltung von $\mathcal{N}_1$ nach $\mathcal{N}_2$ gibt.

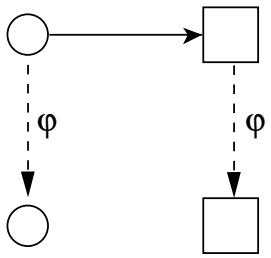


Abbildung 4.7: Eine Abbildung zwischen Netzen zu Aufgabe 4.6

Beweis:
Um den Beweis des Satzes zu vereinfachen, schließen wir auch den trivialen Fall mit ein, dass in einer Vergröberung $\mathcal{N}_1[Y]$ die Menge Y nur aus einem einzigen Element besteht. d.h. $Y = \{x\}$ mit $x \in P_1 \cup T_1$. In diesem Fall würde bei der Konstruktion einer Vergröberung $x \in Y$ einfach durch x_Y ersetzt. Dadurch wird es möglich jede Vergröberung als von der Form $\mathcal{N}_1[Y_1, \ldots, Y_k]$ aufzufassen, wobei $Y = \{Y_1, \ldots, Y_k\}$ eine endliche Partition von $P_1 \cup T_1$ ist (d.h. wie üblich eine Vereinigung von disjunkten nichtleeren Mengen).

Um a) zu beweisen, sei φ ein Epimorphismus von $\mathcal{N}_1$ in $\mathcal{N}_2$. Dann definieren wir $Y = \{Y_1, \ldots, Y_k\} := \{\varphi^{-1}(x) | x \in P_2 \cup T_2\}$.

Da φ eine Abbildung und $P_1 \cup T_1$ endlich ist, muss Y eine endliche Partition sein. Aus der Annahme, dass φ sowohl F- als auch A-erhaltend ist, folgt mit Lemma 4.5 dass $Y_i = \varphi^{-1}(x)$ offen oder abgeschlossen ist.

Wir müssen nun beweisen, dass $\mathcal{N}_1[Y_1, \ldots, Y_k]$ "isomorph" zu $\mathcal{N}_2$ ist, d.h. formal gesehen, dass es einen Isomorphismus ψ von $\mathcal{N}_1[Y_1, \ldots, Y_k]$ auf $\mathcal{N}_2$ gibt. Diese Abbildung wird durch $\psi(x_{Y_i}) := \varphi(y)$ definiert, wobei y beliebig aus Y_i gewählt wird. Diese Definition ergibt tatsächlich eine bijektive Abbildung, was hier nicht ausgeführt wird.

Darüberhinaus ist die Abbildung ein Netzmorphismus. Dies beweisen wir wie folgt. Falls (x_{Y_i}, x_{Y_j}) eine F-Kante in $\mathcal{N}_1[Y_1, \ldots, Y_k]$ ist, dann gibt es nach der Konstruktion von $\mathcal{N}_1[Y_1, \ldots, Y_k]$ Elemente $x' \in Y_i$ und $y' \in Y_j$ derart, dass $(x', y') \in F_1$. Da $Y_i \neq Y_j$ folgt auch $\varphi(x') \neq \varphi(y')$ und $(\varphi(x'), \varphi(y')) = (\psi(x_{Y_i}), \psi(x_{Y_j})) \in F_2$. Damit ist gezeigt, dass ψ is F-erhaltend ist. Aueßrdem gilt: $\{x_{Y_i}\}$ offen $\Leftrightarrow Y_i$ offen $\Leftrightarrow \psi(x_{Y_i})$ offen sowie $\{x_{Y_i}\}$ abgeschlossen $\Leftrightarrow Y_i$ abgeschlossen $\Leftrightarrow \psi(x_{Y_i})$ abgeschlossen. Mit dem Lemma ist ψ also A-erhaltend.

Wir beweisen nun die Umkehrung der Behauptung. Für $(x_2, y_2) \in F_2$ folgt aus der Tatsache, dass φ ein Epimorphismus ist, dass es Elemente $(x', y') \in F_1$ gibt, die $(\varphi(x'), \varphi(y')) = (x_2, y_2)$ erfüllen. Für $Y_i = \varphi^{-1}(x_2)$ und $Y_j = \varphi^{-1}(y_2)$ gilt $x' \in Y_i$, $y' \in Y_j$, $x_{Y_i} = \psi^{-1}(x_2)$ und $x_{Y_j} = \psi^{-1}(y_2)$. Ferner gibt es eine F-Kante (x_{Y_i}, x_{Y_j}) in $\mathcal{N}_1[Y_1, \ldots, Y_k]$. Damit ist ψ^{-1} ein Netzmorphismus und ψ ein Isomorphismus.

Um den Beweis von a) zu Ende zu führen, nehmen wir an, dass $\mathcal{N}_2$ eine Vergröberung $\mathcal{N}_1[Y_1, \ldots, Y_k]$ von $\mathcal{N}_1$ ist (wobei $\{Y_1, \ldots, Y_k\}$ eine Partition von $P_1 \cup T_1$ ist). Dann ist es einfach zu zeigen, dass die durch $\varphi_1(x_1) = x_Y :\Leftrightarrow x_1 \in Y$ definierte Abbildung φ_1 ein Epi-

morphismus ist. Beachte, dass aus der Definition einer Vergröberung folgt, dass die Mengen Y_i nicht leer sind.

Der Beweis von Teil b) beruht auf der Tatsache, dass jede offene bzw. abgeschlossene Menge Y_i nur aus Plätzen bzw. nur aus Transitionen besteht. Das Entsprechende gilt für jede Menge $\varphi^{-1}(x)$ mit $x \in P_2 \cup T_2$. □

Die Bedingung über die Endlichkeit der Netze kann fallen gelassen werden, wenn Vergröberungen durch allgemeine Partitionen definiert werden. Als Beispiel für den Satz 4.7 betrachte man die Vergröberung von Abbildung 4.3b für das Netz aus Abbildung 4.3a. Der Epimorphismus φ ist dann gegeben durch die Abbildung $p_1 \mapsto p_1$, $p_2 \mapsto t_Y$, $p_3 \mapsto p_3$, ..., $p_{13} \mapsto t_{Y1}$, $p_{12} \mapsto p_{12}$, ..., $p_{15} \mapsto t_{Y2}$,..., $t_1 \mapsto t_Y$, $t_2 \mapsto t_Y$,..., $t_6 \mapsto t_{Y1}$, $t_7 \mapsto t_{Y1}$, $t_8 \mapsto t_{Y2}$, $t_9 \mapsto t_{Y2}$. Die Abbildung φ aus Bild 4.6b ist ein weiteres Beispiel.

Aufgabe 4.8

a) Betrachten Sie die Netze in Bild 4.8. Um welche Art von Netzen handelt es sich? Sind die Abbildungen φ_1 und φ_2 Morphismen und wenn ja von welcher Art? Wie kann man sie interpretieren?

b) Betrachten Sie die Netze in Bild 4.9. Um welche Art von Netzen handelt es sich? Sind die Abbildungen φ_3 und φ_4 Morphismen und wenn ja von welcher Art? Wie kann man sie interpretieren?

4.3 Gefärbte Netze

4.3.1 Definition von gefärbten Netzen

Die Definition eines gefärbten Netzes beruht auf derjenigen eines Netzes (Def. 4.1), welches in der graphischen Darstellung den Plätzen (Stellen), Transitionen und Kanten entspricht. In Abbildung 4.10 ist ein solches Netz (nach [**?**]) dargestellt und zusätzlich eine Anzahl von Marken in den Plätzen (Anfangsmarkierung). Durch die Anfangsmarkierung wird es zu einen P/T-Netz, dessen Definition und Semantik ebenfalls in F4 behandelt wurde. Dieses Netz kann folgendermaßen interpretiert werden: zwei Geschäftsleute A und B können jeweils von zu Hause (`at home`) zum Arbeitsplatz (`at work`) wechseln. Die Fahrt kostet Geld. Dieses (und mehr) kann jedoch beim gemeisamen Handel (`talk business`) wieder verdient werden. Abstrakt gesehen, handelt sich hier also um zwei Betriebsmittel verbrauchende und erzeugende Funktionseinheiten.

Wie in F4 führen wir an diesem Beispiel gefärbte Netze durch *Faltung* ein: ähnliche Strukturen werden zusammengelegt, d.h. hier die Plätze und Transitionen von A und B. Letztere werden dafür durch *individuelle Marken* A und B unterschieden, die in Abbildung 4.11 als "*A*ünd

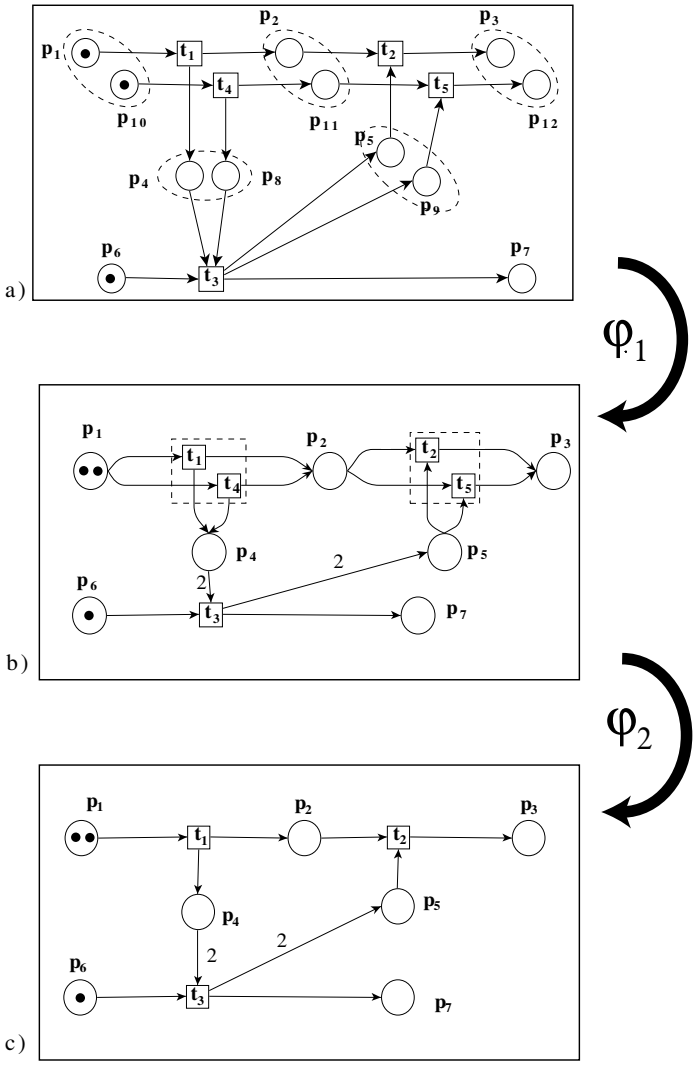


Abbildung 4.8: Zwei Abbildungen φ_1 und φ_2 zu Aufgabe 4.8

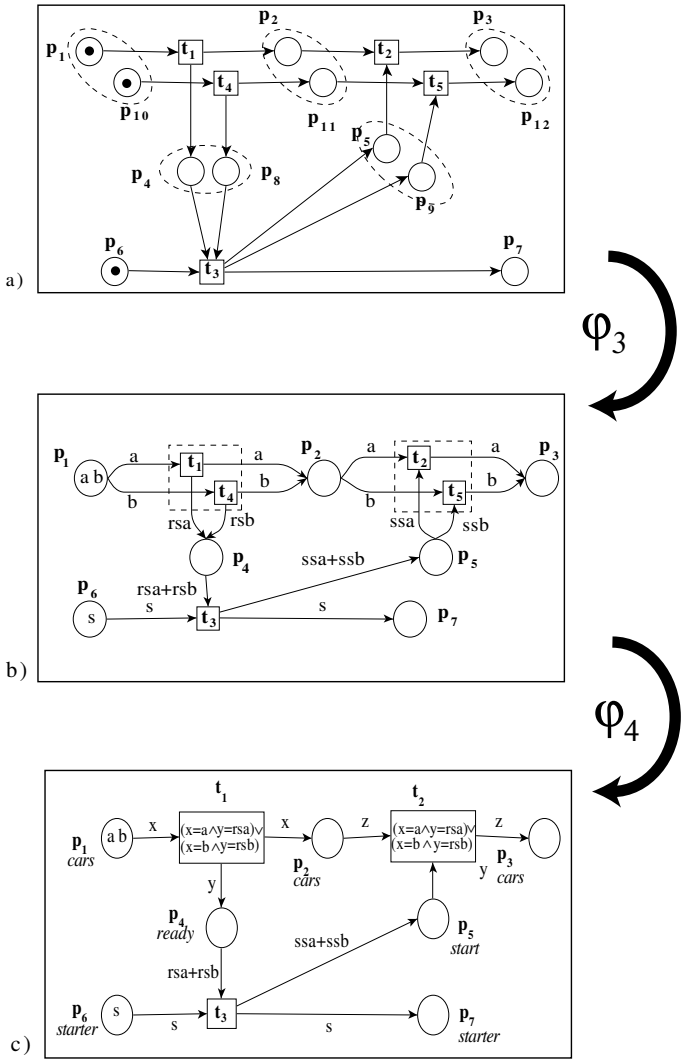


Abbildung 4.9: Zwei Abbildungen φ_3 und φ_4 zu Aufgabe 4.8

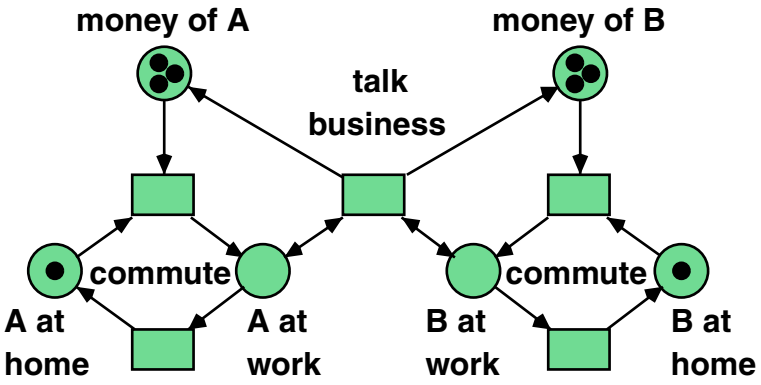


Abbildung 4.10: Das Leben zweier Geschäftsleute

"B"dargestellt sind. Diese Abbildung zeigt das Schalten der oberen Transition. Die Zusammenfaltung des Netzes 4.10 ist in Abbildung 4.12 zu sehen. Hier wird eine Geldeinheit von A bzw. B ebenfalls durch "A"bzw. "Brepräsentiert. Wird eine Darstellung mit Bezeichner und Wert gewünscht, so kann man wie in Abbildung 4.13 vorgehen.

Aufgabe 4.9 Konstruieren Sie die im Text genannte Faltung zwischen den Netzen von Abbildung 4.10 und Abbildung 4.12.

Der Markentyp eines Platzes heißt *Farbe*, im Beispiel also immer $cd(p) = \{A, B\}$ für alle Plätze p. Zum Schalten einer Transition müssen ihre Variablen mit Werten belegt werden, und zwar aus der Farbe des angrenzenden Platzes. Für den Schaltvorgang in Abbildung 4.11 ist die Belegung $\beta = [x = A]$, für die Markierung in Abbildung 4.13 und das Schalten von `deposit` ist sie $\beta = [n = A, a = 300]$.

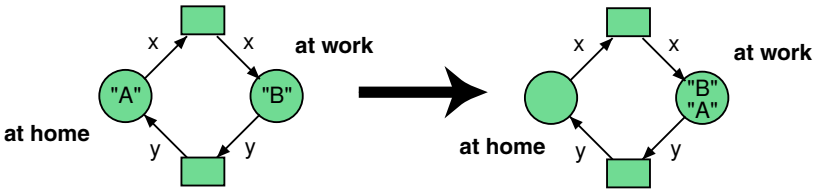


Abbildung 4.11: Schalten eines gefärbten Netzes

Definition 4.10 *Sei* $Var := \{x_1, x_2, x_3, \ldots\}$ *eine Menge von Variablen mit Wertebereichen* $dom(x)$ *für jedes* $x \in Var$. *Eine* Belegung *ist eine Zuordnung (Abbildung)* $\beta = [x_1 = u_1, x_2 = u_2, x_3 = u_3, \ldots]$ *von Werten* $u_i \in dom(x_i)$ *zu den Variablen.*

Definition 4.11 *Ein* gefärbtes Petrinetz *(coloured Petri net, CPN) wird als Tupel* $\mathcal{N} = \langle P, T, F, \mathcal{C}, cd, Var, Guards, \widehat{W}, \mathbf{m}_0 \rangle$ *definiert, wobei gilt:*

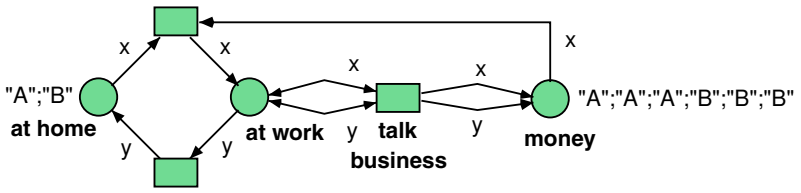


Abbildung 4.12: Faltung von Netz 4.10 zu einem gefärbten Netz

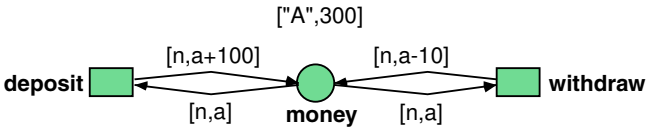


Abbildung 4.13: Ein nicht ganz so einfaches Bankkonto

- (P, T, F) *ist ein endliches Netz (Def. 4.1),*
- $\mathcal{C}$ *ist eine Menge von* Farbenmengen*,*
- $cd\colon P \to \mathcal{C}$ *ist die* Farbzuweisungsabbildung *(colour domain mapping). Sie wird durch* $cd\colon F \to \mathcal{C}$, $cd(x,y) := \textbf{if } x \in P \textbf{ then } cd(x) \textbf{ else } cd(y) \textbf{ fi}$ *auf* F *erweitert.*
- Var *ist eine Menge von Variablen mit Wertebereichen* $dom(x)$ *für jedes* $x \in Var$*.*
- $Guards = \{guard_t \mid t \in T\}$ *ordnet jeder Transition* $t \in T$ *ein Prädikat* $guard_t$ *mit Variablen aus Var zu.*
- $\widehat{W} = \{W_\beta \mid \beta$ *ist Belegung von* $Var\}$ *ist eine Menge von Kantengewichtungen der Form* $W_\beta : F \to Bag(\bigcup \mathcal{C})$*, wobei* $W_\beta(x,y) \in Bag(cd(x,y))$ *für alle* $(x,y) \in F$ *gilt.*
- $\mathbf{m}_0 : P \to Bag(\bigcup \mathcal{C})$ *mit* $\mathbf{m}_0(p) \in Bag(cd(p))$ *für alle* $p \in P$ *ist die Anfangsmarkierung.*

Definition 4.12 *a) Die Markierung eines gefärbten Netzes (CPN)*
$\mathcal{N} = \langle P, T, F, \mathcal{C}, cd, Var, Guards, \widehat{W}, \mathbf{m}_0 \rangle$ *ist ein Vektor* $\mathbf{m}$ *mit* $\mathbf{m}(p) \in Bag(cd(p))$ *für jedes* $p \in P$ *(auch als Abbildung* $\mathbf{m} : P \to Bag(\bigcup \mathcal{C})$ *mit* $\mathbf{m}(p) \in Bag(cd(p))$ *für jedes* $p \in P$ *aufzufassen).*

b) Sei β *eine Belegung für* Var*. Die Transition* $t \in T$ *heißt* $\beta-$*aktiviert in einer Markierung* $\mathbf{m}$ *falls* $guard_t(\beta) = true$ *und* $\forall p \in {}^\bullet t.\ \mathbf{m}(p) \geq W_\beta(p,t)$ *(als Relation:* $\mathbf{m} \xrightarrow{t,\beta}$*).*

c) Es sei

$$\widetilde{W}_\beta(p,t) := \begin{cases} W_\beta(p,t) & \text{falls } (p,t) \in F \\ \emptyset & \text{sonst} \end{cases} \quad \text{und entsprechend}$$

$$\widetilde{W}_\beta(t,p) := \begin{cases} W(t,p) & \text{falls } (t,p) \in F \\ \emptyset & \text{sonst} \end{cases} .$$

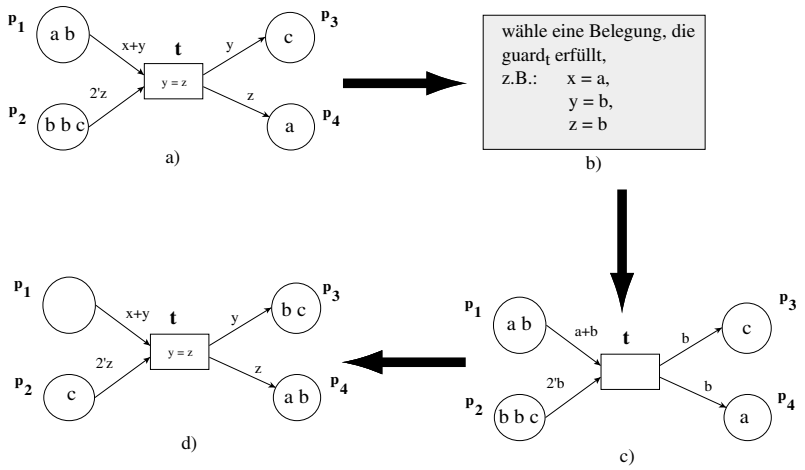


Abbildung 4.14: Schaltregel für gefärbte Netze

Ist t in $\mathbf{m}$ $\beta-$*aktiviert, dann ist die* Nachfolgemarkierung *definiert durch:*
$\mathbf{m} \xrightarrow{t,\beta} \mathbf{m}' \Leftrightarrow \forall p \in P.\ (\mathbf{m}(p) \geq \widetilde{W}_\beta(p,t) \wedge \mathbf{m}'(p) = \mathbf{m}(p) - \widetilde{W}_\beta(p,t) + \widetilde{W}_\beta(t,p))$.
(Beachte, dass es sich um Multimengenoperationen handelt!).

d) Definiert man $W_\beta(\bullet,t) := \widetilde{W}_\beta(p_1,t),\ldots,\widetilde{W}_\beta(p_{|P|},t))$ *als Vektor der Länge* $|P|$ *und entsprechend* $W_\beta(t,\bullet) := (\widetilde{W}_\beta(t,p_1),\ldots,\widetilde{W}_\beta(t,p_{|P|}))$*, dann kann die Nachfolgemarkierung einfacher durch Vektoren definiert werden:*
$\mathbf{m} \xrightarrow{t,\beta} \mathbf{m}' \ \Leftrightarrow\ \mathbf{m} \geq W_\beta(\bullet,t)\ \wedge\ \mathbf{m}' = \mathbf{m} - W_\beta(\bullet,t) + W_\beta(t,\bullet))$.
Dabei sind die Multimengenoperatoren komponentenweise auf Vektoren zu erweitern.

e) $\mathbf{m} \xrightarrow{t} \mathbf{m}' :\Leftrightarrow \exists\beta.\ \mathbf{m} \xrightarrow{t,\beta} \mathbf{m}'$

Beispiel: Das Netz aus Abbildung 4.15 ist eine Faltung des Bankiers-P/T-Netzes aus der Vorlesung F4. Seine Anfangsmarkierung (als Vektor dargestellt[3]) ist
$\mathbf{m}_0 = (10'\bullet, \emptyset, 8'a+3'b+9'c)$. Für $\beta = [x = a]$ und $W_\beta(\bullet, GRANT) = (1'\bullet, \emptyset, 1'a))$ ist die Transition GRANT in $\mathbf{m}_0$ aktiviert und die Nachfolgemarkierung ist $\mathbf{m}' = \mathbf{m}_0 + W_\beta[GRANT,\bullet] - W_\beta(\bullet, GRANT) = (10'\bullet, \emptyset, 8'a+3'b+9'c) + (\emptyset, 1'a, \emptyset) - (1'\bullet, \emptyset, 1'a) = (9'\bullet, 1'a, 7'a+3'b+9'c)$. Um die Wirkung der Transition RETURN unter der Belegung $\beta = [x = b]$ zu berechnen, gehen wir von der Markierung $\mathbf{m}_1 = (3'\bullet, 3'a+3'b, 5'a++9'c)$ aus. Die entsprechende Rechnung lautet dann $\mathbf{m}' = \mathbf{m}_1 + W_\beta[RETURN,\bullet] - W_\beta(\bullet, RETURN) = (3'\bullet, 3'a+3'b, 5'a+9'c) + (3'\bullet, \emptyset, 3'b) - (\emptyset, 3'b, \emptyset) = (6'\bullet, 3'a, 5'a+3'b+9'c)$.

Die Schaltregel wird in Abb. 4.14 an einem abstrakten Beispiel erläutert:

1. Wähle (wie in b)) eine Belegung β für Var(t) mit $guard_t(\beta) = true$.
2. Werte mit dieser Belegung die Ausdrücke an den Kanten zu Multimengen aus (wie in c).

[3]Dabei ist die Sortierung (BANK,CREDIT,CLAIM) der Plätze zugrundegelegt.

3. Wende die Schaltregel für kantenkonstsante Netze (siehe Vorl. F4) an (wie in d)).

Definition 4.13 *Die Nachfolgemarkierungsrelation von Definition 4.12 wird wie üblich auf Wörter über T erweitert:*

- $\mathbf{m} \xrightarrow{w} \mathbf{m}'$ *falls w das leere Wort λ ist und* $\mathbf{m} = \mathbf{m}'$,
- $\mathbf{m} \xrightarrow{wt} \mathbf{m}'$ *falls* $\exists \mathbf{m}'' : \mathbf{m} \xrightarrow{w} \mathbf{m}'' \wedge \mathbf{m}'' \xrightarrow{t} \mathbf{m}'$ *für $w \in T^*$ und $t \in T$.*

Die Menge $\mathbf{R}(\mathcal{N}) := \{\mathbf{m} | \exists w \in T^* : \mathbf{m_0} \xrightarrow{w} \mathbf{m}\}$ *ist die Menge der* erreichbaren *Markierungen oder auch* Erreichbarkeitsmenge. $FS(\mathcal{N}) := \{w \in T^* \mid \exists \mathbf{m} : \mathbf{m_0} \xrightarrow{w} \mathbf{m}\}$ *ist die Menge der* Schaltfolgen *(firing sequence set) von $\mathcal{N}$.*

Aufgabe 4.14

a) Berechnen Sie die Wirkung der Transition RETURN unter der Belegung $\beta = [x = b]$.

b) Zeichnen Sie ansatzweise den Erreichbarkeitsgraphen des Netzes von Abbildung 4.15 (mindestens 5 Schritte von der Anfangsmarkierung aus).

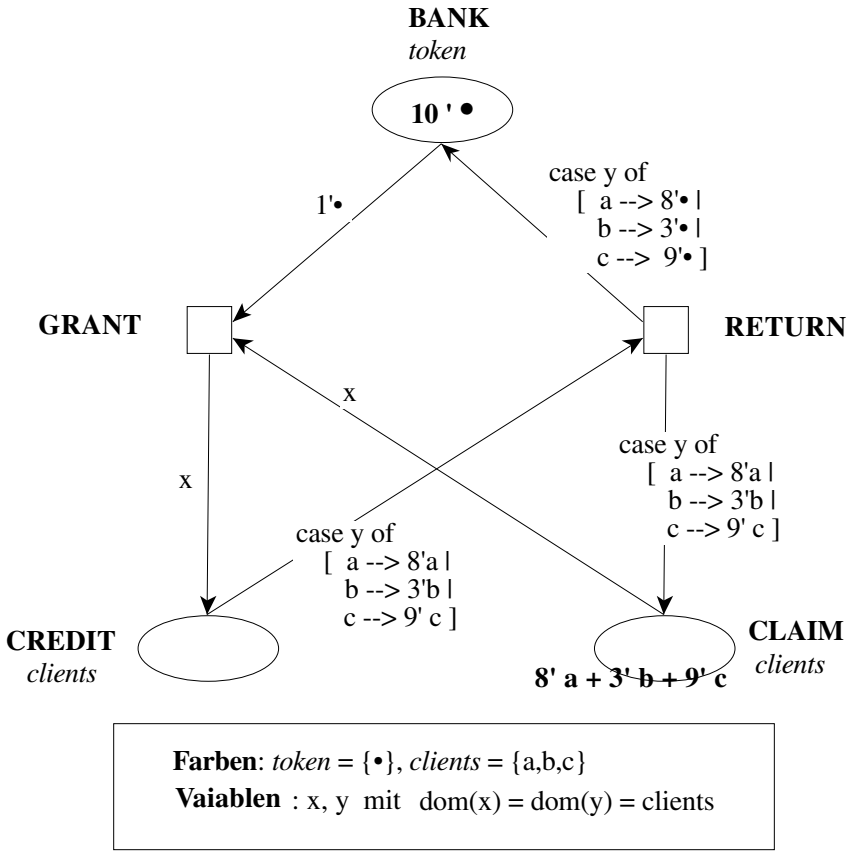


Abbildung 4.15: Faltung des Bankiersnetz zu einem gefärbtes Netz

4.3.2 Beispiele: Ampel, Datenbankmanger

Das Ampelbeispiel[4]

Zunächst wird die Ampelsteuerung für ein Paar synchroner Ampeln entwickelt, z.B. für diejenigen der Nord-Süd-Richtung der Kreuzung (Abb. 4.16).

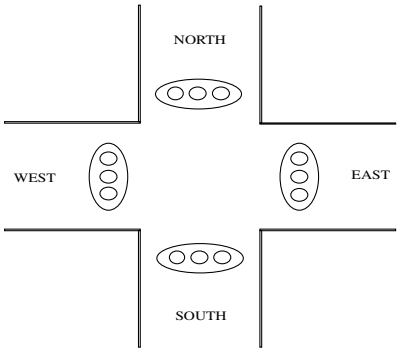


Abbildung 4.16: Kreuzung

Die Lösung von Abbildung 4.17 hat zwar das gewünschte zyklische Verhalten (siehe Markierungsgraph von Abb. 4.18), ist aber selbst kein einfacher zyklischer Graph. Vielmehr sind die Lampen für grünes, gelbes und rotes Licht direkt dargestellt. Gleichzeitiges Leuchten (des roten und gelben Lichtes) wird durch gleichzeitiges Markieren der entsprechenden Plätze realisiert. Dadurch wird gezeigt, wie parallele Ereignisse auch bei völlig synchronen Abläufen durch nichtsequentielle Strukturen ausgedrückt werden.

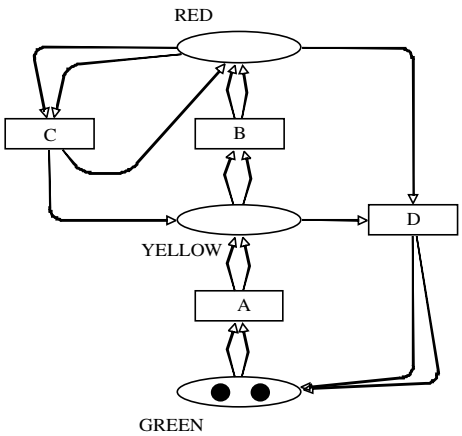


Abbildung 4.17: P/T-Netz für eine Richtung

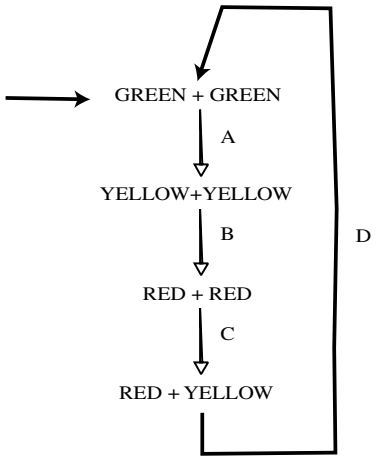


Abbildung 4.18: Markierungsgraph

[4] nach [?]

Dies wird noch deutlicher bei der Erweiterweiterung der Steuerung für beide Richtungen “Nord-Süd (NS)” und Ost-West (EW). Die entsprechenden P/T-Netze für die Einzelrichtungen werden zum gefärbten Netz in Abbildung 4.19 zusammengefaltet. Ein oder zwei Marken NS im Platz GREEN bedeuten, dass für die Nord-Süd-Richtung freie Fahrt gilt. Der zugehörige Markierungsgraph ist in Abbildung 4.20 angegeben.

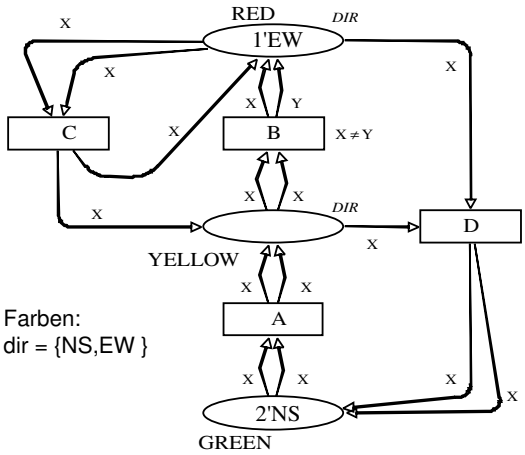


Abbildung 4.19: CPN für zwei Richtungen

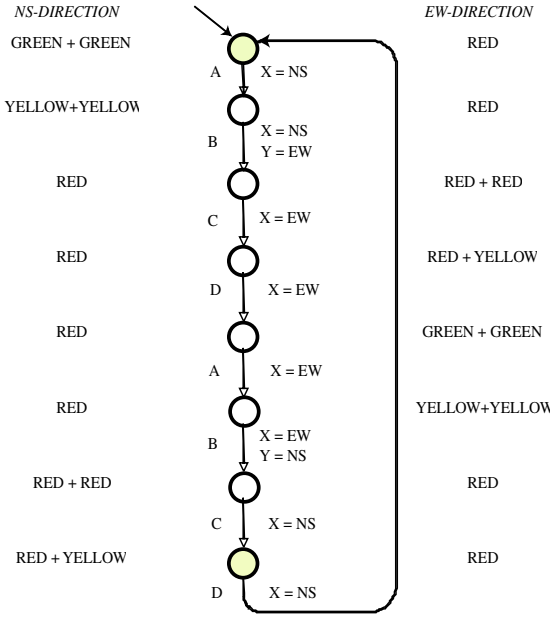


Abbildung 4.20: Markierungsgraph

Aufgabe 4.15 Modellieren Sie das System der 5 Philosophen (siehe Vorlesung F4) als gefärbtes Netz. Durch die Faltung genügen statt der 10 Transitionen und 15 Plätze des P/T-Netzes im gefärbten Netz nur 2 Transitionen und 3 Plätze.

Das Beispiel der Datenbank-Manager[5]

Eine Menge von $n > 0$ Datenbanken soll von n Prozessen, genannt *Datenbank-Manager*, $DBM = \{d_1, d_2, \ldots, d_n\}$ so verwaltet werden, dass sie immer den gleichen Inhalt haben. Um dies zu erreichen, sollen die Manager miteinander kommunizieren. Jeder Manager kann seine eigene Datenbank aktualisieren. Dabei muß er an jeden anderen Manager eine Nachricht senden, die diesen über die Aktualisierung informiert. Der Manager muß warten, bis alle anderen diese Nachricht erhalten, die Aktualisierung durchgeführt und eine entsprechende Rückmeldung zurückgesandt haben. Erst dann kehrt der Manager in den Zustand *inactive* zurück.

Dabei sollen weder die Datenbanken noch ihre Aktualisierung dargestellt werden, sondern nur der Nachrichtenaustausch.

Zustände der Manager : *inactive*, *waiting*, $performing$

Nachrichten : $MB = \{(s,r) | s,r \in DBM \wedge s \neq r\}$

Zustände der Nachrichtenpuffer : *unused*, *sent*, *received*, *acknowledged*

wechselseitiger Ausschluß : *exclusion*

Spezifikation für das gefärbte Netz von Abbildung 4.21:

Farben : $DBM = \{d_1, d_2, \ldots, d_n\}$

$MB = \{(s,r) | s,r \in DBM \wedge s \neq r\}$

$E = \{e\}$

Funktionen :

$MINE : DBM \to Bag(MB)$ $\quad MINE(s) := \sum_{r \neq s} (s,r)$

$REC : MB \to DBM$ $\quad REC((s,r)) := r$

$ABS : DBM \to E$ $\quad ABS(s) := e$

Anfangsmarkierung: $\mathbf{m}_0(p) := \begin{cases} DBM & \text{falls } p = inactive \\ MB & \text{falls } p = unused \\ \{e\} & \text{falls } p = exclusion \\ \emptyset & \text{sonst} \end{cases}$

Nicht alle Funktionen dieser Spezifikation werden in der graphischen Darstellung von Abbildung 4.21 benutzt. Sie sind jedoch nützlich um die Funktion $W_\beta(x,y) \in Bag(cd(x,y))$ (wobei $(x,y) \in F$) aus der Definition 4.11 von gefärbten Netzen zu definieren. Beispielweise ergibt sich für $(x,y) = (T2, unused)$ und die Belegung $\beta = [s = d_1]$ der Wert $W_\beta(T2, unused) = MINE(d_1)$. Diese Funktion ist in der Matrix von Abbildung 4.22 im Eintrag $(unused, T2)$ angegeben. Für $(x,y) = (T3, performing)$ und $\beta = [s = d_1, r = d_2]$ ergibt dies entsprechend $W_\beta(T3, performing)) = REC(d_1, d_2) = d_2$. ID bezeichnet die indentische Abbildung auf DBM bzw, MB. Ein Minuszeichen vor einer Funktion bedeutet nur, dass es sich

[5] nach [**?**]

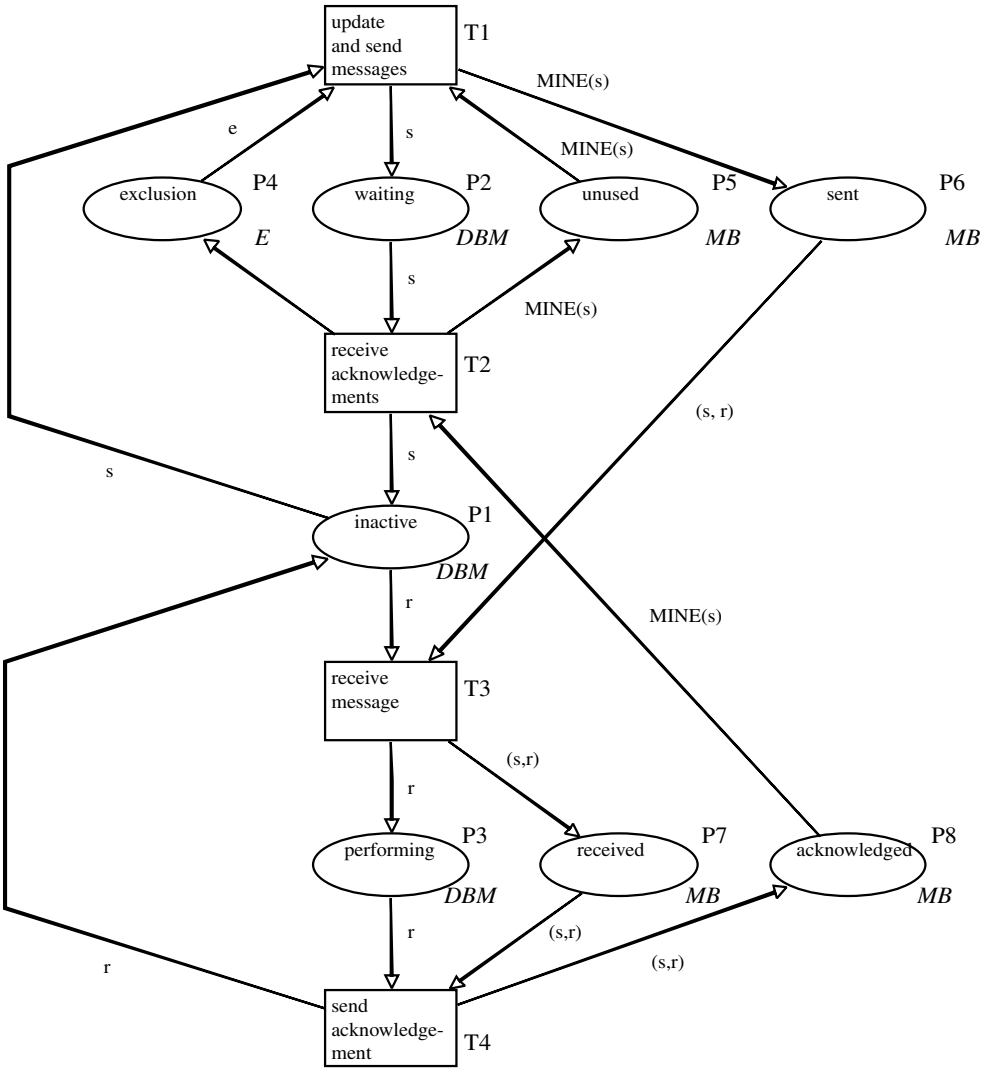


Abbildung 4.21: Datenbank-Manger

um eine Kante $(x, y) \in P \times T$ handelt (kein Minuszeichen: $(x, y) \in T \times P$). Die Matrix (im linken Teil) heißt *Wirkungs-* oder *Inzidenzmatrix*. Neben den Namen der Plätze sind deren Farben angegeben. Unter den Namen der Transitionen sind die Wertebereiche der möglichen Belegungen zu finden (beispielsweise sind für $T3$ Belegungen $\beta = [s = d_1, r = d_2]$ mit $(d_1, d_2) \in MB$ zu wählen.

In der mittleren Spalte von Abbildung 4.22 ist die Anfangsmarkierung aufgeführt. Der rechte Teil (unter w1,..,w5) enthält Invariantenvektoren, die im Anschluß behandelt werden.

Um das Datenbankmanger-System besser zu verstehen und um Eigenschaften zu beweisen, können Invariantenbeziehungen aufgestellt werden, die für alle erreichbaren Markierungen gelten:

1. Jeder DBM ist in genau einem der drei Zustände *inactive*, *waiting* oder *performing*.
2. Jede Nachricht aus MB ist in genau einem der vier Zustände *unused*, *sent*, *received* oder *acknowledged*.
3. Höchstens ein DBM wartet.
4. Ein DBM ist genau dann im Zustand *performing*, wenn eine Nachricht an ihn im Zustand *received* ist.
5. Wenn ein DBM im Zustand *waiting* ist, dann sind alle seine Nachrichten in den Zuständen *sent*, *received* oder *acknowledged*.

Formal werden diese Beziehungen als Gleichungen dargestellt, wobei die oben eingeführten Funktionen ID, MINE, ABS und REC benutzt werden, die in kanonischer Weise auf Multimengen zu erweitern sind.

Satz 4.16 *Für alle erreichbaren Markierungen* $\mathbf{m} \in \mathbf{R}(\mathcal{N})$ *des DBM-Netzes* $\mathcal{N}$ *von Abbildung 4.21 gilt:*

1. $\mathbf{m}(inactive) + \mathbf{m}(waiting) + \mathbf{m}(performing) = DBM$
2. $\mathbf{m}(unused) + \mathbf{m}(sent) + \mathbf{m}(received) + \mathbf{m}(acknowledged) = MB$
3. $ABS(\mathbf{m}(waiting)) + \mathbf{m}(exclusion) = \{e\}$
4. $\mathbf{m}(performing) = REC(\mathbf{m}(received))$
5. $MINE(\mathbf{m}(waiting)) = \mathbf{m}(sent) + \mathbf{m}(received) + \mathbf{m}(acknowledged)$

Beweis:
Man prüft leicht nach, dass alle fünf Gleichungen in der Anfangsmarkierung gelten. Als Induktionsschritt ist nun für jede der Gleichungen, jede Markierung $\mathbf{m}$, jede Transition $t \in T$ und jede passende Belegung β zu beweisen:

Gilt die Gleichung in $\mathbf{m}$ und ist $\mathbf{m} \xrightarrow{t,\beta} \mathbf{m}'$, dann gilt die Gleichung auch in $\mathbf{m}'$.

Wir zeigen dies am Beispiel der 2. Gleichung für $\beta[s = d_1]$ und die Transition $T1$. Diese Transition verändert in der 2. Gleichung nur die Werte von $\mathbf{m}(unused)$ und $\mathbf{m}(sent)$. Da die Menge $MINE/d_1)$ von $\mathbf{m}(unused)$ abgezogen, aber zu $\mathbf{m}(sent)$ hinzugefügt wird, bleibt die Gleichung erhalten und gilt auch für $\mathbf{m}'$. □

Wie für P/T-Netze können *Invariantenvektoren* aus der Wirkungsmatrix berechnet werden. Diese sind unter w1,..,w5 in Abb. 4.22 angeben. Sie ergeben die linken Seiten der Invariantengleichungen. Die rechten Seiten berechnen sich als deren Wert für die Abfangsmarkierung von Satz 4.16. Zu beachten ist jedoch, dass anstatt von Gleichungssystemen über den ganzen Zahlen bei P/T-Netzen bei gefärbten Netzen Gleichungssysteme über Funktionen zu lösen sind, was mit Werkzeugen der Computeralgebra zum Teil möglich ist. Um dies zu umgehen werden oft die gefärbten Netze zu P/T-Netzen aufgefaltet und die traditionellen Methoden benutzt.

Aufgabe 4.17

a) (Vervollständigung einer Markierung)

Sei $\mathbf{m}$ eine erreichbare Markierung mit $\mathbf{m}(performing) = u_1 + u_2 + u_3 \quad (u_i \in DBM)$. Beweisen Sie mit Hilfe der Invariantengleichungen: $\mathbf{m}(received) = (q, u_1)+(q, u_2)+(q, u_3)$ für ein $q \in DBM$ mit $q \neq u_i$ $(1 \leq i \leq 3)$.

b) (Verklemmungsfreiheit)

Beweisen Sie mit Hilfe der Invariantengleichungen, dass das DBM-System verklemmungsfrei ist, d.h. dass in jeder erreichbaren Markierung mindestens eine Transition schalten kann.

Aufgabe 4.18 Geben Sie Invariantengleichungen für das gefärbte Netz der 5 Philosophen von Aufgabe 4.15 an und konstruieren Sie die Inzidenzmatrix.

DATABASE SYSTEM		T1	T2	T3	T4	$\mathbf{m}_0$	*Invariants* w1	w2	w3	w4	w5
		DBM	DBM	MB	MB		DBM	MB	E	DBM	MB
inactive	DBM	-ID	ID	-REC	REC	Σ DBM	ID				
waiting	DBM	ID	-ID				ID		ABS		MINE
performing	DBM			REC	-REC		ID			ID	
exclusion	E	-ABS	ABS			e			ID		
unused	MB	-MINE	MINE			ΣMB		ID			
sent	MB	MINE		-ID				ID			-ID
received	MB			ID	-ID			ID		-REC	-ID
acknowledged	MB		-MINE		ID			ID			-ID

Abbildung 4.22: Inzidenz-Matrix des DBM-Netzes (Abb.4.21)

4.4 Minimale objektbasierte Netze[6]

Im nächsten Abschnitt werden Petrinetze mit für die objektorientierte Modellierung relevanten Konzepten behandelt, wozu auch die dynamische Erzeugung neuer Objekte nach vorgebenen Mustern ("Klassen") gehört. Bekanntlich werden in der theoretischen Informatik prinzipielle Eigenschaften gerne mit *minimalen* Modellen untersucht, d.h. mit Formalismen die keine zur Darstellung des Problems unnötigen Komponenten enthalten. Ein für Informatiksysteme relevante Eigenschaft ist das *Erreichbarkeitsproblem*, bei dem zu entscheiden ist, ob eine vorgebene Konfiguration (hier: Markierung) zu den möglichen Konfigurationen des Systems gehört. Dieses Problem ist für Platz/Transitions-Netze entscheidbar([**?**][**?**]) und unentscheidbar für gefärbte Netze, falls (wie in der Definition dieses Kapitels) unendliche Farbmengen zugelassen sind. Letzteres kann man relativ einfach dadurch beweisen, dass man zeigt, wie ein Zählerautomat durch ein gefärbtes Netz simuliert werden kann. Das Erreichbarkeutsproblem ist nämlich schon für Zählerautomaten mit nur zwei Zählern unentscheidbar, da diese wiederum Turing-Maschinen simulieren können.

In diesem Abschnitt wird nun gezeigt werden, dass die Möglichkeit der Erzeugung von eindeutigen Objekten zu unentscheidbaren Problemen führt. Dazu werden *minimale objektbasierte Netze* eingeführt, die viel einfacher als Platz/Transitions-Netze sind. Sie besitzen aber die Möglichkeit eindeutige Bezeichner zu generieren, die als Bezeichner von Objekten aufgefasst werden können. Allein durch diese zusätzliche Eigenschaft wird das Erreichbarkeitsproblem unentscheidbar. Dies zeigt die prinzipielle Mächtigkeit dieses Mechanismus. Er lässt sich auf jedes andere Modell oder jede Programmiersprache übertragen, die ihn auch enthält. Selbstverständlich schließt das Ergebnis, wie praktisch alle Unentscheidbarkeitsresultate, nicht die Möglichkeit aus, wenigstens für einige praktisch relevante Fälle einen Beweis führen zu können. Aber es schließt zumindest aus, dass ein entsprechendes universell verwendbares Analyseverfahren existiert.

4.4.1 Objekterzeugung und Zählen

Zählerautomaten und Platz/Transitions-Netze sind relativ ähnlich. Deshalb wurden sie auch häufig dazu verwandt, Unentscheidbarkeitsergebnisse für Erweiterungen von Platz/Transitions-Netzen abzuleiten (z.B. in [**?**]). Dabei wird der Zustand der Zählerautomaten durch die Markierung des Petrinetzes simuliert. Beispielsweise können wir einen Zählerstand durch die Anzahl der Marken in einer Stelle darstellen.

Abb. 4.23 stellt im linken Teil einen Test auf einen Zählerstand größer Null des Zählers `x` dar. In der Stelle `x` ist der Zählerstand 2 durch zwei Marken dargestellt. Die Transition `not null` schaltet genau dann, wenn der Zählerstand nicht Null ist. Die Transition `null` kann zwar schalten, wenn der Zählertstand null ist, aber auch sonst. In einem Platz/Transitions-Netz kann man den korrekten Test auf Null durch eine zu `x` komplementäre Stelle darstellen. Dies setzt aber voraus, dass der Zähler nur endlich viele Werte annehmen kann.

[6] nach [**?**] und [**?**]

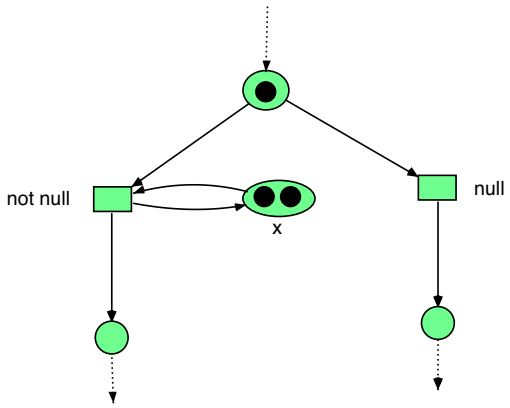


Abbildung 4.23: Test auf Null

Unser Ansatz ist es nun, Nulltests, die während der Berechnung durchgeführt werden, auf den letztmöglichen Zeitpunkt zu verschieben. Dies können wir insbesondere dadurch erreichen, dass wir während eines Nulltests sicherstellen, dass die getestete Stelle, wenn sie denn nicht leer ist, auch nie wieder leer werden kann. Der Endzustand, der leere Zählerstellen erfordert, könnte dann nie erreicht werden, so dass der fehlgeschlagene Nulltest erkannt wird.

Wir definieren nun eine sehr einfache Netzklasse, die die oben skizzierte Simulation von Zählerautomaten zuläßt und die in vielen höheren Petrinetzformalismen nachgebildet werden kann.

Zunächst nehmen wir an, dass ein objektorientierter Formalismus neue Objekte in *unbegrenzter* Anzahl neu erzeugen kann, wie es in allen objektorientierten Programmiersprachen der Fall ist. Dies ist Grundvoraussetzung dafür, daß in einem objektorientierten System die Objekte der Wirklichkeit, von denen potentiell unbeschränkt viele für das System relevant werden könnten, durch Objekte im Modell abgebildet werden können. Ein Formalismus, der keine unbeschränkte Erzeugung erlaubt, könnte auch gängige Programmiertechniken nicht unterstützen. Hier seien nur die klassischen Implementationen von Listen und Bäumen genannt, die pro Element der Liste oder pro Knoten des Baumes ein Objekt einsetzen.

Weiterhin nehmen wir an, dass jedes erzeugte Objekt eine von allen momentan im System befindlichen Objekten *verschiedene* Identität hat. Wäre dies nicht der Fall, so könnte bei Zugriffen nicht geklärt werden, ob auf das neue oder auf das alte Objekt zugegriffen werden soll.

Zuletzt verlangen wir, dass im Formalismus geprüft werden kann, ob zwei gegebene Objekte identisch sind. Der Test auf Identität ist grundlegend für viele Datenstrukturen und wird in allen Programmiersprachen speziell unterstützt. Identität ist eine grundlegende Eigenschaft eines Objekts [**?**]. Ohne einen Test auf Identität wäre es kaum möglich, auch nur eine Menge von Objekten zu verwalten, da die Präsenz eines Objektes nicht entschieden werden könnte.

Sobald diese Voraussetzungen in einem Petrinetzformalismus gegeben sind, sind Unentscheidbarkeiten die notwendige Folge. Es ist zu beachten, dass wir noch nicht einmal einen änderbaren Zustand der Objekte oder gar eine Kommunikation zwischen den Objekten dazu annehmen

müssen.

Definition 4.19 *Ein* minimales objektbasiertes Netz *oder kurz* MOB-Netz *ist ein Tupel* (S, T, F, V, z, e)*, wobei* S *und* T *disjunkte Mengen sind,* $F \subseteq S \times T \cup T \times S$*,* V *eine Menge von Variablen,* $z : F \to V$ *eine Variablenzuordnungsfunktion und* $e \in V$ *eine ausgezeichnete Variable.*

Eine Markierung m *ist eine Funktion von* S *in Multimengen.*

Wir verwenden in dieser Definition Variablen. Auch Variablen könnte man als nicht zwingend notwendig für einen objektorientierten Formalismus bemängeln. Dies ist in der Tat so, doch werden Variablen lediglich benutzt, um elegant den Test auf Gleichheit durch die Bindung an dieselbe Variable darstellen zu können. Es wären auch andere Konzepte hierfür denkbar, etwa durch Äquivalenzrelationen auf den Kanten.

Die ausgezeichnete Variable e wird so gedeutet, dass sie garantiert an ein *neu* erzeugtes Objekt gebunden wird. Dies ist ein sehr eingeschränktes Schema der Objekterzeugung, da insbesondere pro Schalten einer Transition nur ein neues Objekt erzeugt werden kann. Es reicht aber aus, um daraus die Unentscheidbarkeitsergebnisse abzuleiten.

Wir lassen als Markierung einer Stelle beliebige Multimengen zu, insbesondere also beliebig komplizierte Objekte. Wiederum ist dies nicht notwendig, sondern nur eine besonders einfache Darstellung. Da die innere Struktur der Objekte für das Schaltverhalten keine Rolle spielen wird, ist in der Tat die Reduktion auf reine Gleichheit bzw. Ungleichheit gegeben, die wir für den Minimalformalismus anstreben.

Definition 4.20 *Sei* m *eine Markierung und* t *eine Transition. Sei* f *eine Variablenbindungsfunktion mit* ${}^\bullet(f) = V$*. Wenn* $\forall s \in S : (s,t) \in F \Rightarrow f(z(s,t)) \in m(s)$ *und* $\forall s \in S : f(e) \notin m(s)$ *gilt, dann ist* t *aktiviert.*

Nun sei $m'(s) = m(s) - f(z(s,t))$*, wenn* $(s,t) \in F$ *und* $m'(s) = m(s)$ *sonst. Weiterhin* $m''(s) = m'(s) + f(z(t,s))$*, wenn* $(t,s) \in F$ *und* $m''(s) = m'(s)$ *sonst. Dann ist* m'' *die Markierung nach dem Schalten von* t*. Wir schreiben* $m[t> m''$*.*

Die Aktiviertheitsbedingungen stellen sicher, dass alle benötigten Marken auf den Eingangsstellen liegen und daß die ausgezeichnete Variable wirklich an einen frischen Wert gebunden wird.

In Abb. 4.24 ist ein einfaches MOB dargestellt. Beim Schalten der Transitionen werden die Variablen $v1$ und $v2$ an eine der beiden Marken a gebunden, während durch die mit e beschriftete Kante immer eine neue Marke erzeugt wird.

4.4.2 Zählerautomaten

Hier werden die genannten Zählerautomaten formal eingeführt.

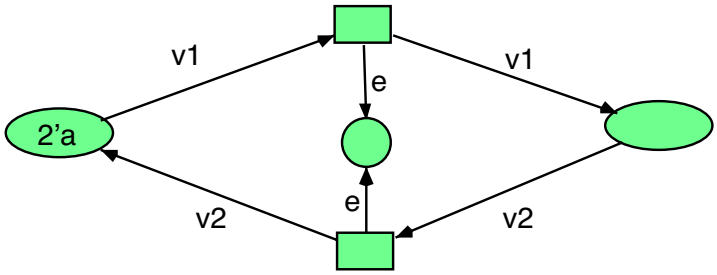


Abbildung 4.24: Ein einfaches minimales objektbasiertes Netz

Definition 4.21 *Ein 2-Zählerautomat wird repräsentiert durch ein 5-Tupel* $(Q, \delta_{0,0}, \delta_{1,0}, \delta_{0,1}, \delta_{1,1})$*, wobei* $\delta_{0,0} : Q \to Q \times \{0,1\} \times \{0,1\}$ *die Übergangsfunktion für den Fall ist, daß beide Zähler null sind,* $\delta_{1,0} : Q \to Q \times \{-1,0,1\} \times \{0,1\}$ *die Übergangsfunktion für den Fall ist, daß nur Zähler 1 ungleich null ist,* $\delta_{0,1} : Q \to Q \times \{0,1\} \times \{-1,0,1\}$ *die Übergangsfunktion für den Fall ist, daß nur Zähler 2 ungleich null ist, und* $\delta_{1,1} : Q \to Q \times \{-1,0,1\} \times \{-1,0,1\}$ *die Übergangsfunktion für den Fall ist, daß beide Zähler ungleich null sind.*

Eine Konfiguration ist ein Tripel $k = (q,i,j)$ *mit* $q \in Q$ *und* $i,j \in \mathcal{N}$*. Um die Nachfolgekonfiguration zu ermitteln, setzen wir zunächst* $(q',x,y) = \delta_{\min(i,1),\min(j,1)}(q)$*. Nun ist* $k' = (q', i+x, j+y)$ *die Nachfolgekonfiguration. Wir beobachten, daß stets* $i+x$ *und* $j+y$ *natürliche Zahlen sind.*

In der vorigen Definition haben wir, anders als bei den meisten anderen Definitionen, keinen Startzustand und kein Eingabealphabet berücksichtigt. Diese Vereinfachung konnten wir deshalb zulassen, weil wir nur an einer Analyse der internen Übergänge des Zählerautomaten und nicht an einer Wechselwirkung der Maschine mit ihrer Umgebung oder an einer akzeptierten Sprache interessiert sind.

Die vier Funktionen $\delta_{i,j}$ wurden getrennt formuliert, da sie jeweils verschiedene Definitionsbereiche aufweisen. Es wäre möglich gewesen, die Funktionen in einer gemeinsamen Funktion $\delta^* : Q \times \{0,1\} \times \{0,1\} \to Q \times \{0,1\} \times \{-1,0,1\}$ zusammenzufassen und die Einschränkungen des Definitionsbereichs durch zusätzliche Bedingungen anzugeben, was aber insgesamt keine Vereinfachung bewirkt hätte.

Trotz der geringfügig veränderten Definition läßt sich das klassische Unentscheidbarkeitsresultat zu Zählerautomaten leicht auf das hier definierte Modell übertragen. Wir geben eine Beweisskizze angelehnt an [**?**].

Satz 4.22 *Sei ein 2-Zählerautomat A mit einem Endzustand q und eine Konfiguration k gegeben, dann ist es unentscheidbar, ob A von k aus die Konfiguration* $(q,0,0)$ *erreicht. Beweis:*

*Sei B eine universelle Online-Turingmaschine mit einem Band. Nun können wir analog zum Beweis in [**?**] A so konstruieren, daß B simuliert wird. Es ist möglich, A so zu wählen, daß, sobald der Übergang in einen Endzustand feststeht, in einer Schleife die Zähler auf null gesetzt werden und danach ein einziger Endzustand q eingenommen wird.*

Damit ist bei A die Endkonfiguration von k aus genau dann erreichbar, wenn B das Wort w akzeptiert. Diese Frage ist jedoch für eine universelle Turingmaschine B unentscheidbar.

Nun könnte man argumentieren, die Darstellbarkeit der natürlichen Zahlen sei nicht grundsätzlich für einen objektorientierten Formalismus notwendig und daher sei eine Entscheidbarkeit des Erreichbarkeitsproblems für objektorientierte Petrinetze nicht grundsätzlich ausgeschlossen. Wir werden jedoch sehen, daß bereits unter deutlich schwächeren Annahmen Unentscheidbarkeitsresultate in objektorientierten Petrinetzen auftreten.

4.4.3 Der Reduktionsbeweis

Jetzt werden wir das Null-Erreichbarkeitsproblem für 2-Zählerautomaten auf das Null-Erreichbarkeitsproblem für MOB-Netze reduzieren.

Wir beobachten allerdings, dass wir keine direkte Möglichkeit haben, einen Nulltest zu implementieren. Wir greifen daher auf eine Technik zurück, die sich bereits in [**?**] als hilfreich für eine Simulation von Nulltests erwiesen hat: Während der eigentlichen Simulation werden keine Nulltests durchgeführt, sondern es werden alle Tests auf das Ende der Simulation verschoben. Dann kann nämlich durch die Forderung nach dem Erreichen der leeren Markierung nachträglich sichergestellt werden, dass alle Tests während der Simulation auch gelungen wären. Dazu sind natürlich bei jedem Nulltest geeignete Änderungen an der Markierung durchzuführen, die später überprüft werden können.

Konstruktion 4.23 () Sei $(Q, \delta_{0,0}, \delta_{1,0}, \delta_{0,1}, \delta_{1,1})$ ein 2-Zählerautomat und $k_0 = (q_0, i_0, j_0)$ eine Anfangskonfiguration. Sei q_E der Endzustand.

Wir konstruieren das zugehörige MOB-Netz. Seien s_1, s_2, c_1, c_2, h_1 und h_2 verschiedene Elemente, die nicht in Q enthalten sind. Die Stellenmenge sei $S = Q \cup \{s_1, s_2, c_1, c_2, h_1, h_2\}$. Die Stellen aus Q zeigen, wenn sie markiert sind, an, daß sich der Automat im entsprechenden Zustand befindet. s_1 und s_2 enthalten exakt eine Marke, mit der auf den Stellen c_1 und c_2 die Zähler simuliert werden. Die Stellen h_1 und h_2 enthalten kurzfristig Marken, die nach c_1 und c_2 verschoben werden sollen.

Der Zählerstand entspricht der Summe der Anzahl der Marken in c_1 und h_1 bzw. c_2 und h_2, die mit der Marke in s_1 bzw. s_2 übereinstimmen.

Sei x ein beliebiges Objekt, es dient als Marke in der Anfangsmarkierung. Die Anfangsmarkierung m_0 sei $m_0(s_1) = 1 \cdot x$, $m_0(s_2) = 1 \cdot x$, $m_0(c_1) = i_0 \cdot x$, $m_0(c_2) = j_0 \cdot x$, $m_0(q_0) = 1 \cdot x$ und die leere Markierung für alle anderen Stellen. Die Zählerstände und der Zustand entsprechen damit der Anfangskonfiguration.

Sei $T' = \{(q, i, j) \mid q \in Q \wedge i, j \in \{0, 1\}\}$. Seien $t_E, b_1, b_2 \notin T' \cup S$ und paarweise verschieden. Sei $T = T' \cup \{t_E, b_1, b_2\}$ die Menge der Transitionen. T' enthält die Transitionen, die die Zustandsübergänge simulieren. t_E ist die Transition, die das Ende der Berechnung rät, b_1 bewegt Marken von h_1 nach c_1, b_2 bewegt Marken von h_2 nach c_2.

Seien v, v', v'', e verschiedene Variablen und $V = \{v, v', v'', e\}$.

F setzt sich zusammen aus Kanten für jede einzelne Transition. Sei $(q, p_1, p_2) = t \in T'$. Sei $\delta_{p_1,p_2}(q) = (q', \Delta_1, \Delta_2)$. Wir fügen zu F die Kanten (q, t) und (t, q') für den Zustandsübergang hinzu. Wir setzen die Variablen $z(q, t) = v$ und $z(t, q') = v$.

Weiterhin fügen wir (s_1, t) und (s_2, t) hinzu mit $z(s_1, t) = v'$ und $z(s_2, t) = v''$.

Für $k \in \{1, 2\}$ fügen wir (t, s_1) hinzu mit $z(t, s_1) = v'$, wenn $p_k = 1$, und $z(t, s_1) = e$, wenn $p_k = 0$.

Für $k \in \{1, 2\}$ fügen wir (c_k, t) hinzu, wenn $p_k = 1$. In diesem Fall ist $z(c_k, t) = z(s_k, t)$. Wir fügen (t, c_k) hinzu, wenn $\Delta_k + p_k \geq 1$. In diesem Fall ist $z(t, c_k) = z(t, s_k)$. Wir fügen (t, h_k) hinzu, wenn $\Delta_k + p_k = 2$. In diesem Fall ist $z(t, h_k) = z(t, s_k)$.

Für $k \in \{1, 2\}$ fügen wir (h_k, b_k) mit $z(h_k, b_k) = v$ und (b_k, c_k) mit $z(b_k, c_k) = v$ hinzu.

Zuletzt fügen wir (q_E, t_E) mit $z(q_E, t_E) = v$, (s_1, t_E) mit $z(s_1, t_E) = v'$ und (s_2, t_E) mit $z(s_2, t_E) = v''$ zur Menge der Kanten hinzu.

Nun ist (S, T, F, V, z, e) ein MOB-Netz.

Satz 4.24 *Wenn der Zählerautomat mit leeren Zählern in den Endzustand gelangen kann, kann das MOB-Netz die leere Markierung erreichen.*

Beweis:
Die Schrittzahl bis zum Erreichen des Endzustands sei n. Damit ergibt sich eine Konfigurationsfolge $k_0 \to k_1 \to \ldots \to k_n$, wobei $k_n = (q_E, 0, 0)$. Für $a \in \{0, \ldots, n\}$ sei $k_a = (q_a, i_a, j_a)$.

Wir werden zeigen: Für jedes $a \in \{0, \ldots, n\}$ gibt es eine erreichbare Markierung m_a und Elemente x, y und z, so daß $m_a(q_a) = 1 \cdot x$, $m_a(s_1) = 1 \cdot y$, $m_a(s_2) = 1 \cdot z$, $m_a(c_1) = i_a \cdot y$, $m_a(c_2) = j_a \cdot z$. Alle anderen Stellen sind unmarkiert.

Für $a = 0$ ist dies durch die Definition der Anfangsmarkierung sofort klar.

Sei nun für einen Induktionsbeweis die Aussage für ein $a \in \{0, \ldots, n-1\}$ erfüllt. Sei $t = (q_a, \min(i_a, 1), \min(j_a, 1))$. Wir zeigen, daß t in m_a aktiviert ist. Die Variablenbindungsfunktion sei wie folgt definiert: $f(v) = x$, $f(v') = y$, $f(v'') = z$ und $f(e)$ beliebig, aber ungleich x, y und z.

Sei $p_1 = \min(i_a, 1)$ und $p_2 = \min(j_a, 1)$. Sei $\delta_{p_1,p_2}(q_a) = (q', \Delta_1, \Delta_2)$.

Die Stellen q_a, s_1 und s_2 sind passend markiert. Wenn t von c_1 eine Marke abzieht, dann per Konstruktion $\min(i_a, 1) = 1$, also $i_a \geq 1$ und auch hier ist eine passende Marke vorhanden. Ebenso für c_2. Von anderen Stellen existiert keine Kante zu t. Also ist t aktiviert.

Sei $x' = x$. Sei $y' = y$, wenn $\min(i_a, 1) = 1$, und $y' = e$ sonst. Sei $z' = z$, wenn $\min(j_a, 1) = 1$, und $z' = e$ sonst. Damit ist die Nachfolgemarkierung $m'(q') = 1 \cdot x'$, $m'(s_1) = 1 \cdot y'$, $m'(s_2) = 1 \cdot z'$, $m'(c_1) = (i_a + \Delta_1 - \max(\Delta_1 + p_1 - 1, 0)) \cdot y'$, $m'(h_1) = \max(\Delta_1 + p_1 - 1, 0) \cdot y'$,

$m'(c_2) = (j_a + \Delta_2 - \max(\Delta_2 + p_2 - 1, 0)) \cdot y''$, $m'(h_2) = \max(\Delta_2 + p_2 - 1, 0) \cdot y''$. Die anderen Stellen sind unmarkiert.

Wenn $\max(\Delta_1 + p_1 - 1, 0) = 0$, dann sei $m'' = m'$. Ansonsten ist $\max(\Delta_1 + p_1 - 1, 0) = 1$ und wir beobachten, daß durch einmaliges Schalten von b_1 die Markierung $m''(q') = 1 \cdot x'$, $m''(s_1) = 1 \cdot y'$, $m''(s_2) = 1 \cdot z'$, $m''(c_1) = m'(c_1) + 1 \cdot y' = (i_a + \Delta_1 - \max(\Delta_1 + p_1 - 1, 0) + 1) \cdot y' = (i_a + \Delta_1) \cdot y'$, $m''(h_1) = m'(h_1) - 1 \cdot y' = \emptyset$, $m''(c_2) = (j_a + \Delta_2 - \max(\Delta_2 + p_2 - 1, 0)) \cdot y''$, $m''(h_2) = \max(\Delta_2 + p_2 - 1, 0) \cdot y''$ erreichbar ist.

Wenn $\max(\Delta_2 + p_2 - 1, 0) = 0$, dann sei $m''' = m''$. Ansonsten ist $\max(\Delta_2 + p_2 - 1, 0) = 1$ und wir beobachten, daß durch einmaliges Schalten von b_2 die Markierung $m'''(q') = 1 \cdot x'$, $m'''(s_1) = 1 \cdot y'$, $m'''(s_2) = 1 \cdot z'$, $m'''(c_1) = (i_a + \Delta_1) \cdot y'$, $m'''(h_1) = \emptyset$, $m'''(c_2) = (j_a + \Delta_2) \cdot y''$, $m'''(h_2) = \emptyset$ erreichbar ist.

Dies ist jedoch gerade $m_{a+1}(q_{a+1}) = 1 \cdot x'$, $m_{a+1}(s_1) = 1 \cdot y'$, $m_{a+1}(s_2) = 1 \cdot z'$, $m_{a+1}(c_1) = i_{a+1} \cdot y'$, $m_{a+1}(c_2) = j_{a+1} \cdot z'$. Alle anderen Stellen sind unmarkiert, damit ist der Induktionsbeweis abgeschlossen.

Wir haben nun insbesondere ein m_n mit $m_n(q_E) = 1 \cdot x$, $m_n(s_1) = 1 \cdot y$ und $m_n(s_2) = 1 \cdot z$ für bestimmte x, y und z. Die anderen Stellen, insbesondere c_1 und c_2, sind unmarkiert. In dieser Markierung kann einmal t_E schalten und wir erreichen die leere Markierung. □

Nun untersuchen wir die Rückrichtung, die etwas umständlicher ist, weil im Petrinetz kein Determinismus vorliegt. Hierbei ist es insbesondere aufwendig, die Verteilung von Marken zwischen c_1 und h_1 sowie c_2 und h_2 zu berücksichtigen.

Satz 4.25 *Wenn die leere Markierung im MOB-Netz erreichbar ist, kann der Zählerautomat mit leeren Zählern in den Endzustand gelangen.*

Beweis:
Sei nach Voraussetzung des Satzes $m_0[t_1> m_1[t_2> \ldots [t_{n-1}> m_{n-1}[t_n> m_n$ eine Schaltfolge, so daß m_n die leere Markierung ist.

Wir beobachten zunächst, daß die Transitionen aus T' die Anzahl der Marken in den Stellen aus Q konstant lassen, da sie genau eine Marke anziehen und eine Marke ablegen. Ebenso die beiden Transitionen b_1 und b_2. In m_0 liegt eine Marke auf q_0. Also muß einmal t_E geschaltet haben. Nach t_E sind höchstens noch b_1 und b_2 aktiviert, da keine Marke mehr auf den Stellen aus Q liegt. Die Transitionen b_1 und b_2 können jedoch nicht in eine leere Markierung schalten, da sie Ausgangskanten haben. Also ist $\forall a \in \{1, \ldots, n-1\} : t_a \in T - \{t_E\}$ und $t_n = t_E$.

Nun lassen die Transitionen aus $T - \{t_E\}$ auch die Anzahl der Marken auf s_1 konstant. Da initial je eine Marke auf dieser Stelle liegt, gilt $\forall a \in \{0, \ldots, n-1\} : |m_a(s_1)| = 1$. Wir bezeichnen die dort liegenden Marken mit l_a für jedes $a \in \{0, \ldots, n-1\}$, so daß $\forall a \in \{0, \ldots, n-1\} : m_a(s_1) = 1 \cdot l_a$.

Die Transition b_1 hat lediglich eine technische Funktion. Wir vermeiden durch sie, mehrere Kanten zwischen einer Transition und einer Stelle zeichnen zu müssen. Eigentlich ist nur die Summe der Marken von c_1 und h_1 interessant. Wir definieren daher $g_a = m_a(c_1) + m_a(h_1)$.

Wir nehmen nun an $\forall x \in g_a : x = l_a$ für ein gewisses $a \in \{1, \ldots, n-1\}$. Wir wollen zeigen $\forall x \in g_{a-1} : x = h_{a-1}$.

Wir wissen $t_{a-1} \in T - \{t_E\}$. Wenn $t_{a-1} = b_1$ oder $t_{a-1} = b_2$, dann $\forall x \in g_{a-1} : x = l_{a-1}$, da g_a weder durch das Schalten von b_1 noch durch das Schalten von b_2 verändert wird. Also $\forall x \in g_{a-1} : x = h_{a-1}$.

Sei daher $t_{a-1} \in T'$ und $t_{a-1} = (q, p_1, p_2)$.

Wenn $p_1 = 1$, dann ist wegen der Konstruktion der Kantenvariablen $l_{a-1} = l_a$. Daher zieht t_{a-1}, wenn überhaupt, nur eine Marke des Typs l_a von c_1 ab, also gilt auch hier $\forall x \in g_{a-1} : x = h_{a-1}$.

Wenn andererseits $p_1 = 0$, dann zieht t_{a-1} nach Konstruktion des Netzes gar keine Marke von c_1 ab. Weiterhin ist $z(t_{a-1}, s_1) = e$, also ist l_a eine vorher unbenutzte Marke. Insbesondere gilt $\forall x \in g_{a-1} : x \neq l_a$. Da aber keine Marken abgezogen werden, bleibt nur noch $g_{a-1} = \emptyset$ als Möglichkeit. Trivialerweise $\forall x \in g_{a-1} : x = h_{a-1}$.

Dies war der Induktionsschritt. Für den Induktionsanfang beachten wir $a = n - 1$, dann gilt $g_a = \emptyset$, weil $t_n = t_E$ und t_E die Markierung von c_1 und s_1 nicht verändert. Also trivialerweise $\forall x \in g_a : x = l_a$ für $a = n - 1$ und per Induktion für alle $a \in \{0, \ldots, n-1\}$.

Nun gebe $i_a = |g_a|$ die Anzahl der Marken in c_1 und h_1 gemeinsam zu jedem Zeitpunkt an.

Betrachten wir nun ein beliebiges $a \in \{0, n-1\}$, so daß $t_a \in T'$ und daher $t_a = (q, p_1, p_2)$. Für den Fall $p_1 = 0$ wurde implizit schon während der Induktion gezeigt, daß $g_a = \emptyset$. Im Falle $p_1 = 1$ zieht t_a nach Konstruktion eine Marke von c_1 ab, also $g_a \neq \emptyset$. Damit ist $p_1 = 0$ genau dann, wenn $i_a = 0$.

Analog definieren wir j_a als die Anzahl der Marken in c_2 und h_2 gemeinsam zu jedem Zeitpunkt. Wir können analog zeigen, daß $p_2 = 0$ genau dann, wenn $j_a = 0$.

Da die Transitionen aus $T - \{t_E\}$ die Anzahl der Marken in den Stellen aus Q konstant lassen, ist für $a \in \{0, \ldots, n-1\}$ stets genau eine Stelle aus Q mit einer Marke markiert, die restlichen Stellen sind unmarkiert. Sei q_a dasjenige Element von q mit $|m_a(q_a)| = 1$.

Wir definieren $k_a = (q_a, i_a, j_a)$ für $a \in \{0, \ldots, n-1\}$ und zeigen, daß jedes k_a als Konfiguration im 2-Zählerautomaten erreichbar ist.

Für $a = 0$ ist dies klar, denn die Anfangsmarkierung gibt gerade den Anfangszustand wieder.

Sei nun k_a erreichbar für $a \in \{0, \ldots, n-2\}$, wir wollen zeigen, daß auch auch k_{a+1} erreichbar ist. Wenn $t_a \in \{b_1, b_2\}$, dann ist $k_{a+1} = k_a$ und die Aussage ist trivialerweise erfüllt.

Betrachten wir also $t_a \in T'$. Wir haben $t_a = (q, p_1, p_2)$. t_a entfernt eine Marke aus q, also muß gelten $q_a = q$. Wir haben bereits gezeigt, daß $p_1 = 0 \Leftrightarrow i_a = 0$ und $p_2 = 0 \Leftrightarrow j_a = 0$, also ist der Zustandsübergang auch in dem 2-Zählerautomaten möglich. Weiterhin entspricht die Änderung von i_a und j_a genau der Änderung der Zähler des Automaten. Also ist auch k_{a+1} eine erreichbare Konfiguration.

Durch Induktion wurde damit gezeigt, daß jede Konfiguration k_a erreichbar ist. Dies gilt insbesondere für k_{n-1}. Weil $t_{n-1} = t_E$ und m_n die leere Markierung ist, schließen wir

$k_{n-1} = (q_E, 0, 0)$, was der Endkonfiguration entspricht. Also ist die Endkonfiguration erreichbar. □

Satz 4.26 *Das Null-Erreichbarkeitsproblem für MOB-Netze ist unentscheidbar.*

Beweis:
Das in 4.23 konstruierte MOB-Netz kann die leere Markierung genau dann erreichen, wenn der Zählerautomat den Endzustand erreichen kann, wie in 4.24 und 4.25 gezeigt. Damit lässt sich ein gemäß 4.22 unentscheidbares Problem auf das Null-Erreichbarkeitsproblem reduzieren. Dieses Problem muß also selbst unentscheidbar sein. □

Dieses Theorem erlaubt sehr einfache Beweise für Unentscheidbarkeitseigenschaften in den verschiedensten Netzformalismen. Für gefärbte Petrinetze ist selbstverständlich die Unentscheidbarkeit des Erreichbarkeitsproblems Folklore, aber sie lässt sich nun leicht neu beweisen. Variablen an Ein- und Ausgangskanten sind ohnehin ausdrückbar, die spezielle Variable e, die stets an einen frischen Wert gebunden wird, lässt sich durch eine weitere Stelle realisieren, auf der ein Zähler kontinuierlich hochgezählt wird oder auf der eine andere geeignete Operation auf einem Datenwert iteriert wird.

Für objektorientierte Formalismen entspricht das Binden von e einfach dem Erzeugen eines neuen Objekts. Hier wird stets garantiert, dass das neu erzeugte Objekt von allen existierenden verschieden ist. Damit wird für alle objektorientierten Petrinetzformalismen eine Übertragung des Unentscheidbarkeitsresultats leicht gelingen.

4.5 Referenznetze[7]

4.5.1 Netz-Kanäle

Ein ernstzunehmender Einwand gegen das gefärbte Netz aus Abb. 4.12 ist, dass die Bewegung von Geld und die Bewegung der Personen zu stark miteinander verwoben sind. Auf den ersten Blick ist es nicht klar, welche Teile des Netzes welchen Aspekt beschreiben.

In Abb. 4.25 bereiten wir eine Teilung des Systems aus Abb. 4.12 vor. Zwei der Transitionen werden doppelt in verschiedenen Teilen des Netzdiagramms aufgezeichnet. Um die Schaltsemantik des Netzes korrekt beizubehalten, müssten die Transitionen wieder verschmolzen werden.

Wir gehen einen Schritt weiter und deuten die beabsichtigte Bedeutung des Netzes durch eine textuelle Anschrift an. In Abb. 4.26 bedeutet die Anschrift `this:withdraw(x)`, dass in diesem Netz (Englisch *this net*) eine andere Transition vorhanden sein sollte, die die Auszahlung von Geld vom Konto von `x` übernehmen kann. Wir geben dabei die Variable am Ende der Anschrift an, um klarzumachen, welche Information umhergereicht werden muß.

[7] aus [**?**]

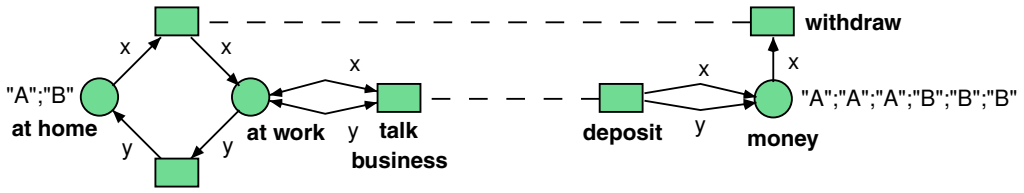


Abbildung 4.25: Personen und Konten werden geteilt

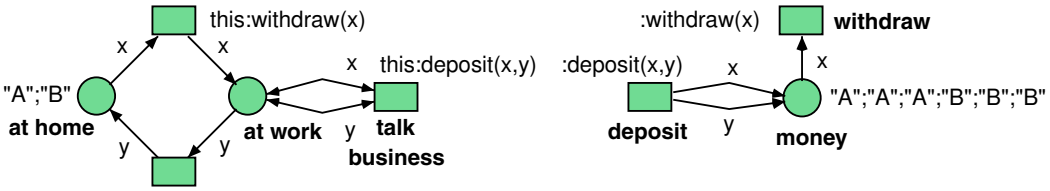


Abbildung 4.26: Textuelle Anschriften kennzeichnen Synchronisation

Solche Anschriften wollen wir als synchrone Kanäle bezeichnen, weil sie die Transitionen zwingen, synchron zu schalten und weil sie den Fluß von Informationen kanalisieren. Die Autoren von [**?**] haben diese Konzept für höhere Petrinetze eingeführt.

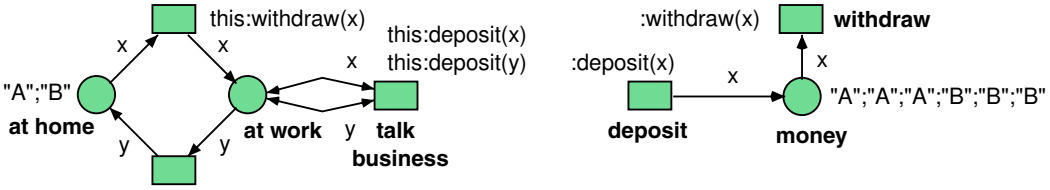


Abbildung 4.27: Wiederverwendung eines Kanals

Weil wir textuelle Anschriften für die Kanäle verwendet haben, können wir mehrere Synchronisationen gleichzeitig anfordern, ohne dass das Diagramm seine Klarheit verliert. In Abb. 4.27 wurde das Netz aus Abb. 4.26 so umstrukturiert, dass von der Transition `talk business` zwei Aufrufe des Kanals `deposit` getätigt werden. Dies veranlasst die Transition `deposit`, zweimal zu schalten, was das gewünschte Verhalten in unserem Beispiel ist.

Jetzt können wir die Lösung aus Abb. 4.27 mit dem gefärbten Netz zur Modellierung eines Bankkontos aus Abb. 4.13 kombinieren und jedes Konto als Wertepaar repräsentieren. Abb. 4.28 zeigt das Resultat. Beachtenswert ist, wie die Anzahl der Geldeinheiten, die ein- oder auszuzahlen ist, als zweiter Parameter über den Kanal übergeben wird.

Gegenüber den gewöhnlichen Petrinetzen fügen synchrone Kanäle die Fähigkeit hinzu, über gleichzeitige, nicht nur über aufeinanderfolgende oder über nebenläufige Handlungen zu sprechen.

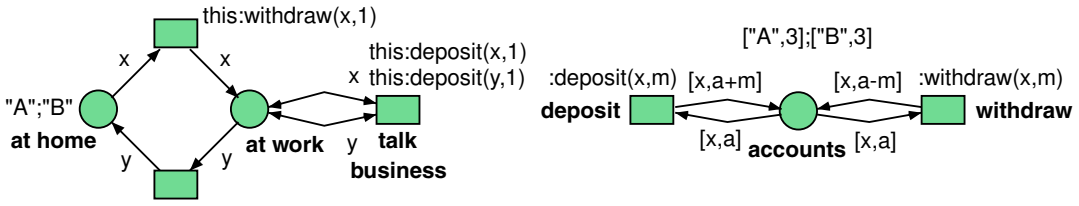


Abbildung 4.28: Buchführung mit Algebra

4.5.2 Netzinstanzen

Bei klassischen Petrinetzen existiert jede gezeichnete Stelle und Transition während der Ausführung eines Netzes genau einmal. Wir können die Beschränkung aufheben und Exemplare von Netzen zulassen. Um verschiedene Exemplare unterscheiden zu können, erlauben wir Referenzen auf Netzexemplare als Marken in anderen Netzexemplaren. Dies führt uns von gefärbten Netzen zu Referenznetzen.

Im folgenden werden wir von Netzexemplaren sprechen, wenn betont werden soll, daß jetzt exakt ein Netzexemplar von möglicherweise vielen gesondert betrachtet werden soll. Wenn wir uns explizit auf die statische Struktur der Stellen, Transitionen und Kanten beziehen, werden wir von einem Netzmuster sprechen. Entsprechend werden auch die Bezeichnungen Stellenmuster, Transitionsmuster, Stellenexemplar und Transitionsexemplar verwendet.

Sofern aus dem Kontext klar ist, ob von Mustern oder Exemplaren die Rede ist, werden wir auch den Zusatz weglassen und weiterhin einfach von Netzen sprechen. Wenn es zu einem Netzmuster nur ein Exemplar geben soll, wäre die Unterscheidung wenig hilfreich, denn Muster und Exemplar können als Einheit gesehen werden.

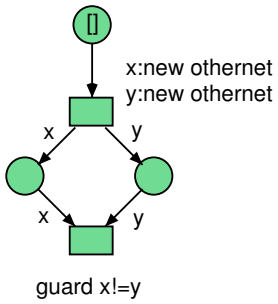


Abbildung 4.29: Das Hauptnetz `creator`

Abbildung 4.30: Das Netz `othernet`

In Abb. 4.29 und 4.30 sehen wir ein einfaches Beispiel von Netzreferenzen. Das Hauptnetz aus Abb. 4.29 erzeugt zwei Netzexemplare des Netzmusters mit dem Namen `othernet` und speichert die Referenzen in verschiedenen Stellen. Letztlich prüft die untere Transition, dass die die beiden erzeugten Netzexemplare wirklich voneinander unterschieden werden können. Da Netzexemplare, die in verschiedenen Anschriften oder zu verschiedenen Zeitpunkten oder

von verschiedenen Transitionen erzeugt wurden, immer unterschiedlich sind, gelingt der Test immer.

Der Test wird in einem *guard* (einer Schutzanweisung) formuliert. Ein Guard gibt eine boolsche Bedingung an, die wahr sein muss, damit eine Transition schalten kann. Nebenbei sehen wir im Netz, dass ungefärbte Marken in Referenznetzen als `[]` repräsentiert werden, das heißt als leeres Tupel.

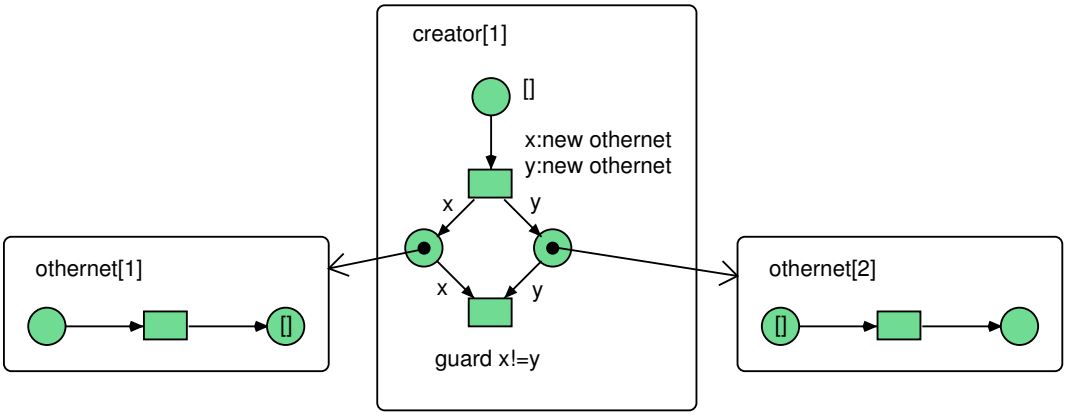


Abbildung 4.31: Netzexemplare mit Netzreferenzen

In Abb. 4.31 sehen wir einen Schnappschuß (Markierung) während der Ausführung dieses Beispiels. Das Hauptnetz hat bereits die beiden anderen Netzexemplare erzeugt und in einem der beiden Netzexemplare hat bereits das Transitionsexemplar geschaltet, so dass eine Marke bewegt wurde. Wenn jetzt die untere Transition des Hauptnetzes schaltet, dann existieren keine Referenzen auf die Exemplare des Netzmusters `othernet` mehr. Die Netzexemplare existieren aber sehr wohl noch, so dass insbesondere auch Schaltungen stattfinden können. Die Variante in Abb. 4.32 zeigt, dass es sich um Referenzen (und nicht Kopien von `othernet`) handelt!

Wir kehren zu unseren Standardbeispiel zurück und bemerken, dass Personen und Bankkonten durch die Verwendung von synchronen Kanälen bereits recht unabhängig geworden sind. Es wäre daher sinnvoll, das Netzmuster in zwei Netzmuster zu zerlegen. Damit die Personen auf ihre Bankkonten zugreifen können, muß das Netzexemplar der Personen die Kontoverwaltung kennen und referenzieren.

Abb. 4.33 beschreibt das neue Netz für die beiden Personen. Es enthält eine explizite Initialisierung der Konten. Die Initialisierungstransition auf der rechten Seite legt Referenzen auf die erzeugten Netzexemplare in die Stelle `bank`. Zwar können Netzexemplare nach ihrer Erzeugung wie andere Marken verwendet werden, aber da sie aktive Objekte darstellen, muss ihre Erzeugung gesondert behandelt werden. Daher ist auch eine explizite Anschrift `acc:new account` für die Erzeugung notwendig.

Die Aufrufe des synchronen Kanals rufen nicht länger Transitionsexemplare im lokalen Netzexemplar auf (`this`), sondern müssen auf das Kontonetzexemplar zielen, das für die betreffende

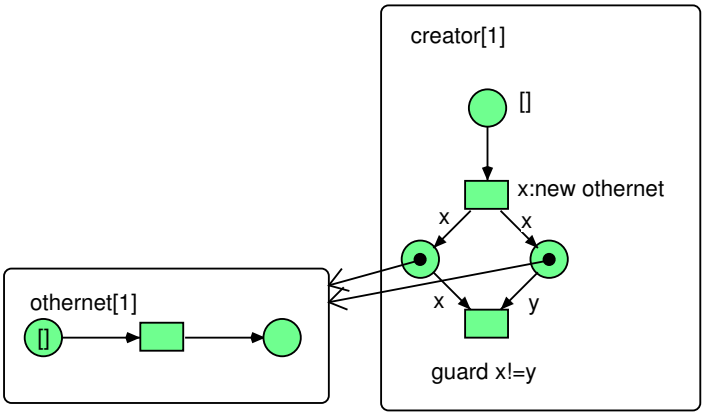


Abbildung 4.32: Netzexemplare mit Netzreferenzen: Variante

Person zuständig ist. Um eine Referenz auf das richtige Netzexemplar zu erhalten, ist ein Zugriff auf die Stelle `accounts` erforderlich.

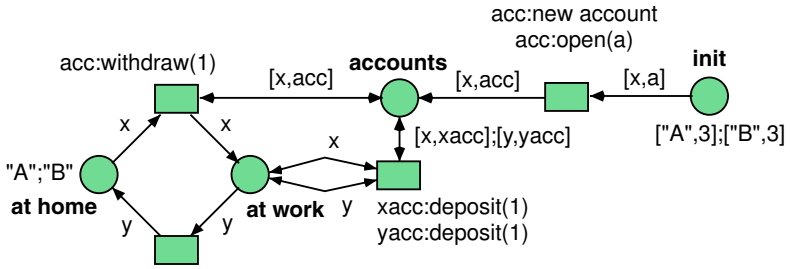


Abbildung 4.33: Das Netz `person`

Wir bemerken, dass wir durch schrittweise Erweiterung des Beispiels in einem Zustand gekommen sind, wo wir viele Details sowohl für die Personen als auch für die Konten ohne große Probleme hinzufügen könnten. Die Möglichkeit, Netzmuster und -exemplare durch Einführung von Netzreferenzen zu trennen und dennoch gemeinsam agieren zu lassen, hält die individuellen Netzmuster einfach und übersichtlich.

Wir könnten fortfahren und ein Netzexemplar pro beteiligter Person einführen. Dazu könnten zyklische Referenzen erforderlich sein, wenn sowohl die Person ihr Konto, als auch das Konto die Person kennen soll. Dies ist zulässig, Netzreferenzen sind nicht auf azyklische Strukturen beschränkt und Netzexemplar müssen keine Hierarchie bilden.

Netzexemplare sind ein wichtiges Konzept, denn sie erlauben es uns, über die Identität von Objekten zu sprechen und nicht nur über die Gleichheit von Werten, wie bei normalen gefärbten Petrinetzen. Sie führen eine neue Ebene der Dynamik in die Petrinetztheorie ein, indem nicht nur die Markierung eines Netzes, sondern das Beziehungsgeflecht von verschiedenen Netzexem-

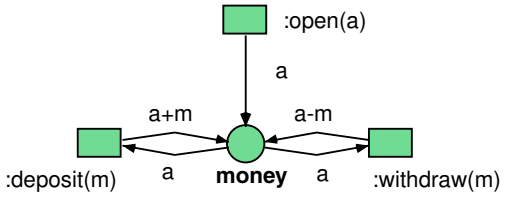


Abbildung 4.34: Das Netz `account`

plaren sich im Laufe der Zeit ändern kann, wenn Netzreferenzen erzeugt, verschoben oder gelöscht werden.

Die Netze dieses Abschnitts wurden mit dem Petrinetzwerkzeug *Renew* [**?**] erzeugt, dessen Werkzeugleiste in der Abbildung Figure 4.35 dargestellt ist. Renew [**?**] kann auch für die Ausführung der erzeugten Netze benutzt werden. Als Beschriftungssprache wurden Elemente der Programmsprache Java [**?**] gewählt, die in vielen ihrer Eigenschaften Petrinetze sinnvoll ergänzt. Das Werkzeug ist selbst in Java geschrieben und durch Transitionen können Java-Klassen ausgeführt werden.

Abbildung 4.35: Werkzeugleiste des Renew-Werkzeugs

Ausführung wird in der Petrinetzliteratur auch als *Simulation* bezeichnet. Dies bedeutet die Simulation des formalen Modells und nicht eines realen Weltausschnittes. Letzteres gilt natürlich auch, wenn das Petrinetz den realen Weltausschnitt hinreichend genau darstellt. Dazu können auch Zeitschranken für das Schalten benutzt werden. Um etwas über andere Petrinetz-Tools zu erfahren, siehe die Internetseite *The Petri Nets World* mit der URL [**?**]. Über sie ist auch Information zu Literatur zu Petrinetzen, Forschungsgruppen und -projekte zugänglich.

4.5.3 Beispiele: Workflow und Garbage Can

Das Workflow-Beispiel

Zunächst behandeln wir die Modellierung eines Geschäftsprozesses (workflow), der in der originalen Formulierung so lautet:

A workflow of the Dutch Justice Department (Wil van der Aalst):
When a criminal offence happened and the police has a suspect a `record` is made by an

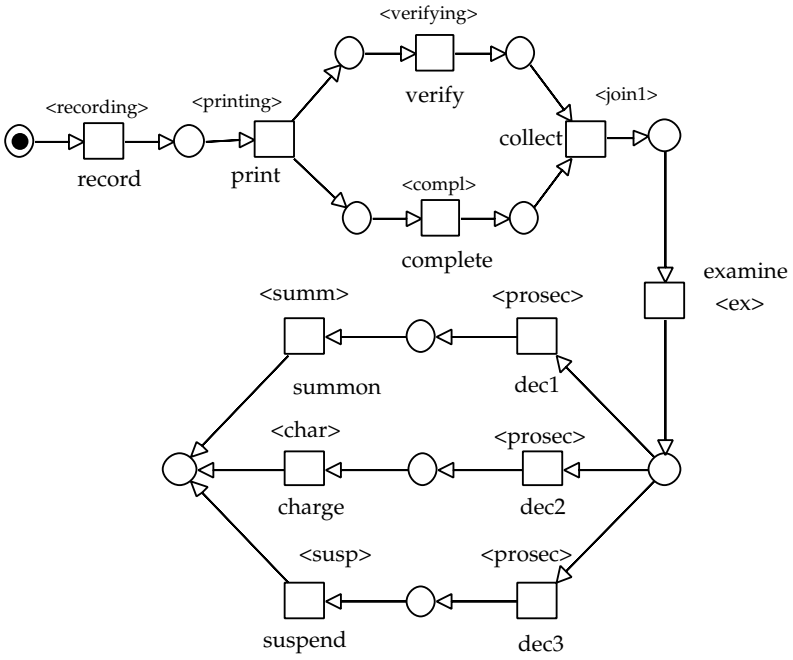


Abbildung 4.36: Workflow: Objektnetz

official. This is `printed` and sent to the secretary of the Justice Department. Extra information about the history of the suspect and some data from the local government are supplied and `completed` by a second official. Meanwhile the information on the official record is `verified` by a secretary. When these activities are completed, the first official `examinates` the case and a prosecutor determines (`dec`ides) whether the suspect is `summoned`[8], `charged`[9] or that the case is `suspended`[10].

In dieser Darstellung sind die zu modellierenden Aktionen teilweise schon hervorgehoben, andere wurden ergänzt: `record`, `print`, `verify`, `complete`, `collect`, `examine`, `dec1`, `dec2`, `dec3`, `summon`, `charge`, `suspend`. Mit Abblidung 4.36 wird eine Modellierung als P/T-Netz gegeben. Darüberhinaus sollen nun aber auch die jeweils ausführenden Funktionseinheiten modelliert werden:

[8] to summon: vorladen
[9] to charge: anklagen
[10] to suspend: aussetzen

Aktion	*Funktionseinheit*	*Interaktionsrelation*
`record`	`official1`[11]	`<recording>`
`print`	`printer`	`<printing>`
`verify`	`secretary`	`<verifying>`
`complete`	`official2`	`<completing>`
`collect`	`put_together`	`<join1>`
`examine`	`official1`	`<ex>`
`dec1`	`prosecutor` [12]	`<prosec>`
`dec2`	`prosecutor`	`<prosec>`
`dec3`	`prosecutor`	`<prosec>`
`summon`	`tribunal`[13]	`<summ>`
`charge`	`tribunal`	`<char>`
`suspend`	`official3`	`<susp>`

Die funktionale Verküpfung dieser Funktionseinheiten sei auch vorgegeben. Sie ist schon in das entsprechende P/T-Netz von Abbildung 4.37 eingebaut. Das Paradigma der "Netze in Netzen" erlaubt es nun, den Workflow als Marke diese Netzes zu modellierung, so wie die Bearbeitungsakte durch die Behörde läuft. Dabei wird die Zuordnung der Aktionen zu den Funktionseinheiten durch eine Relation (*Interaktionsrelation*) angegeben, die dadurch dargestellt wird, dass die entsprechenden Transitionen das gleiche Attribut in spitzen Klammern haben. So ist zum Beispiel die Aktion `record` durch das Attribut `<recording>` der Funktionseinheit `official1` zugeordnet. Wegen `<ex>` kann diese Funktionseinheit aber auch die Aktion `examnine` ausführen.

Diese Art der Netze heißen *Objektnetzsysteme* [**?**]. Das Workflownetz heißt dabei allgemein *Objektnetz* und das zugrundeliegende Netz der Funktionseinheiten *Systemnetz*. Die Schaltregel für Objektnetzsysteme lautet wie folgt. Steht eine Transition des Objektnetzes oder des Systemnetzes nicht in der Interaktionsrelation, dann schaltet sie wie gewöhnlich und alleine. Falls zwei Transionen des Objektnetzes und des Systemnetzes in der Interaktionsrelation stehen, dann schalten sie nur, wenn sie beide aktiviert sind und dann gemeinsam in einem Schritt. Die Abläufe des vorliegenden Beispiels können gut in der "platten" Form von Abbildung 4.38 verfolgt werden. Für das vorliegende Beispiel mag diese Form einfacher erscheinen. Im Allgemeinen wird diese Form bei großen Systemen jedoch sehr unübersichtlich. Vor allem wird sie nicht der Tatsache gerecht, dass das Objektnetz (als Formular, Akte, ...) *in* den Plätzen (Kanälen, Eingangsstapeln, to-do-Listen, ..) des Systemnetzes liegt und nicht sonstwo.

Durch Referenznetze (und mit dem Werkzeug `Renew`) lassen sich diese Netze implementieren, was in Abbildungen 4.39 und 4.40 dargestellt ist. Wegen der Definition der synchronen Kanäle der Referenznetze muss allerdings die doppelte Verküpfung der Transition `official1` mit `record` und `examine` durch eine Kopie von `official1` dargestellt werden. Außerdem ist am Ende eine kleine Ausgabe implementiert, die anzeigt, welche der drei Alternativen aufgrund der Entscheidung von `prosecutor` ausgeführt wurde.

[11] Beamtin1/Beamter1
[12] Staatsanwältin/Staatsanwalt
[13] Gericht

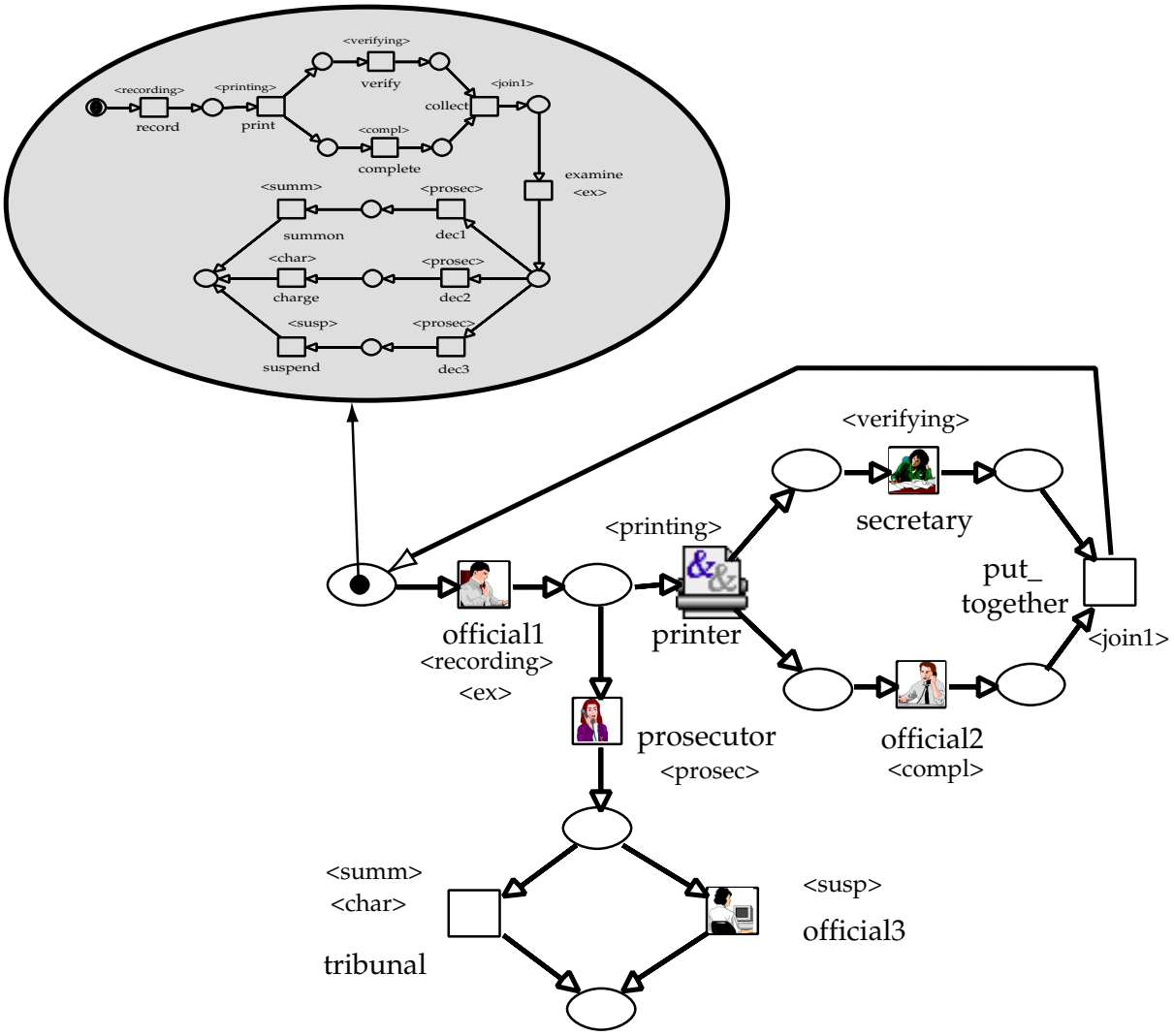


Abbildung 4.37: Workflow: System- mit Objektnet

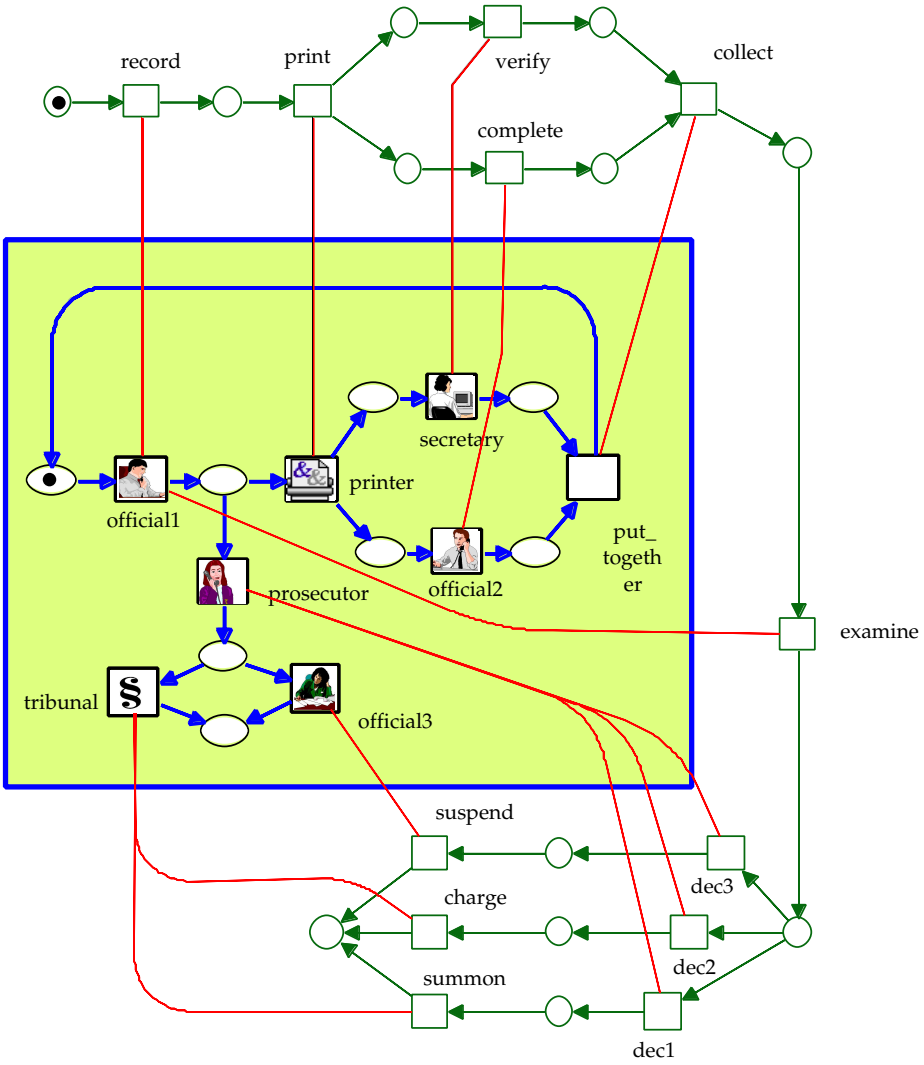


Abbildung 4.38: Workflow: nichthierarchische Form

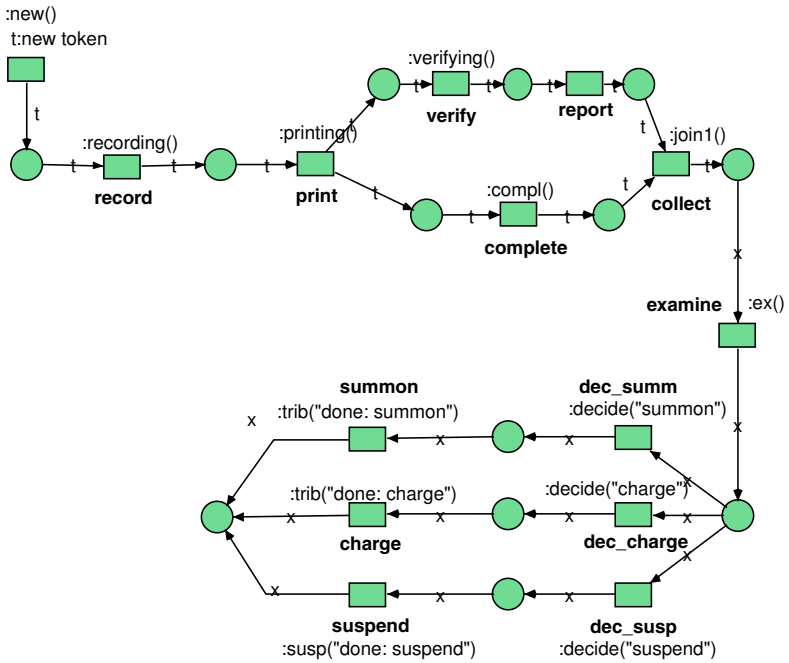


Abbildung 4.39: Workflow: Renew-Modell `Task`

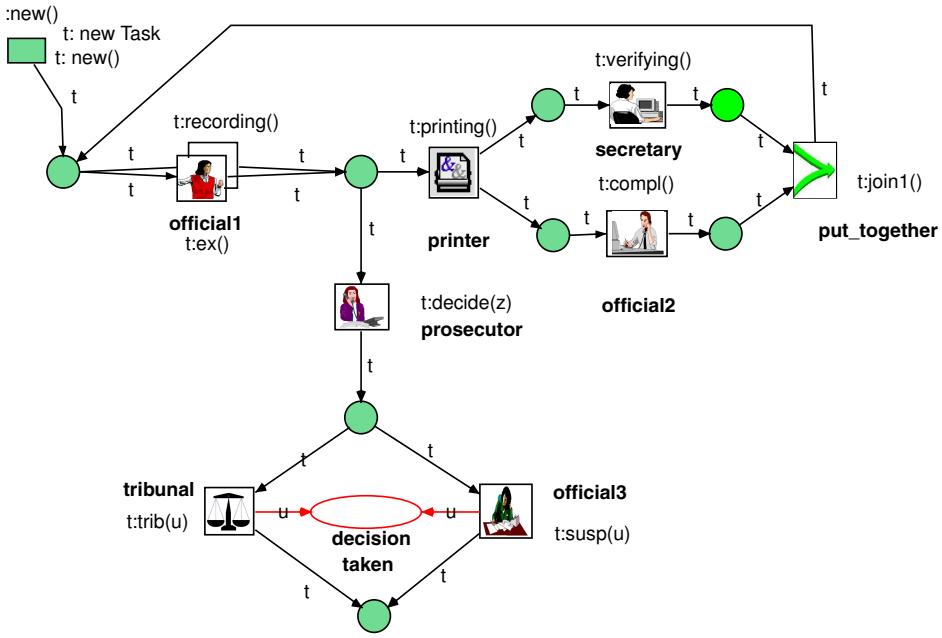


Abbildung 4.40: Workflow: Renew-Modell `FESystem`

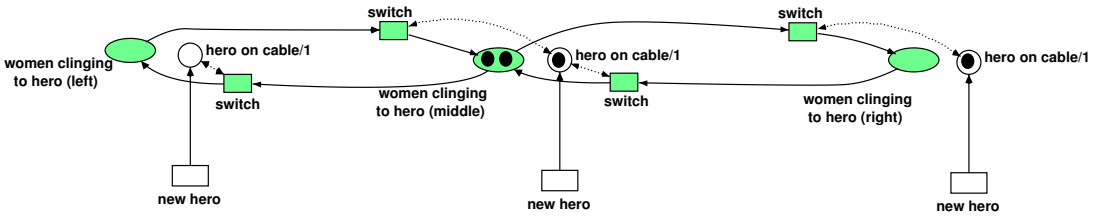


Abbildung 4.41: Das Garbage Can P/T-Netz: Plätze für heros und women auf der Brücke

Das Garbage Can Beispiel

Anhand des folgenden Beispiels soll gezeigt werden, wie eine große Anzahl von Objekten modelliert werden kann, die in einem System erscheinen und verschwinden und zudem nichttriviales Interaktionsverhalten aufweisen. Solche Verhältnisse treten typischerweise in großen Systemen und Rechnernetzen auf und sind meist so komplex, dass sie nur in speziellem Kontext erläutert werden können. Das folgende Beispiel wurde gewählt, da es klein und anschaulich ist. Außerdem zeigt es die Anwendung der Modellierung mittels Petrinetzen in einem nichttechnischen Bereich, nämlich von Entscheidungsvorgängen in Organisationen. Zur Erläuterung des Szenarios wird zunächst eine Modellierung durch ein P/T-Netz gezeigt. Das Hauptziel dieses Abschnittes ist jedoch das darauf folgende Modell eines Referenznetzes.

Verhaltenswissenschaftliche Entscheidungstheorie ist ein Gebiet der Soziologie, das überwiegend einer allgemeinen Organisationstheorie zuzuordnen ist. Während diese ihren Schwerpunkt mehr oder weniger explizit auf die Organisationsform "Unternehmen" legt, handelt es sich bei einer Weiterentwicklung des Ansatzes, dem sogenannten *Garbage Can-Modell (GC-Modell)* von Cohen, March und Olsen (1972) [**?**], um eine Theorie, mit der sich besonders treffend die Entscheidungsprozesse in öffentlich-rechtlichen Institutionen beschreiben lassen. Eine Arbeit von Masuch/LaPotin (1989) [**?**] enthält eine Einkleidung des Problems in eine fiktive Szene, die sich aber gut für eine konkrete Modellierung und Simulation eignet.

Vorzustellen ist sich das Finale des James Bond Films *A view to a kill*. Agent 007 (*hero*) balanciert auf dem Hauptseil der Golden Gate Bridge, eine Frau in Not (*woman in distress*) hält sich an seinem Arm fest, ein Luftschiff (*blimp*) taucht zur Rettung auf. Im happy end des Films wird der Held (007) schließlich mit der Frau am Arm gerettet. In der genannten Arbeit wird das Szenario erweitert, wie es in einem Computerspiel verwendet werden könnte. Nun sind mehrere Helden, Frauen und Luftschiffe vorhanden. Die Personalisierung der Akteure fördert natürlich die Anschaulichkeit des Szenarios, produziert jedoch zuweilen unpassende Assoziationen. Wir wählen daher in partieller Verfremdung für die Akteure die Bezeichnungen `hero`, `women` und `blimp` (mit ebenfalls englischen Pluralbildungen `heros`, `women` und `blimps`).

Akteure aller drei Arten können in zufälligen Abständen aus dem Nichts auftauchen. Alle `heros` befinden sich auf dem Seil mit keiner, einer oder mehreren `women`, die sich an seinem Arm festhalten. Die `blimps` schweben über der Szenerie. Die `heros` sind zwar stark und können mehrere `women` am Arm tragen, doch ein einzelnes Luftschiff kann nur eine begrenzte Last tragen, d.h. durch die Last "zu schwere" `heros` können nicht gerettet werden. Die `women` in Not wissen dies

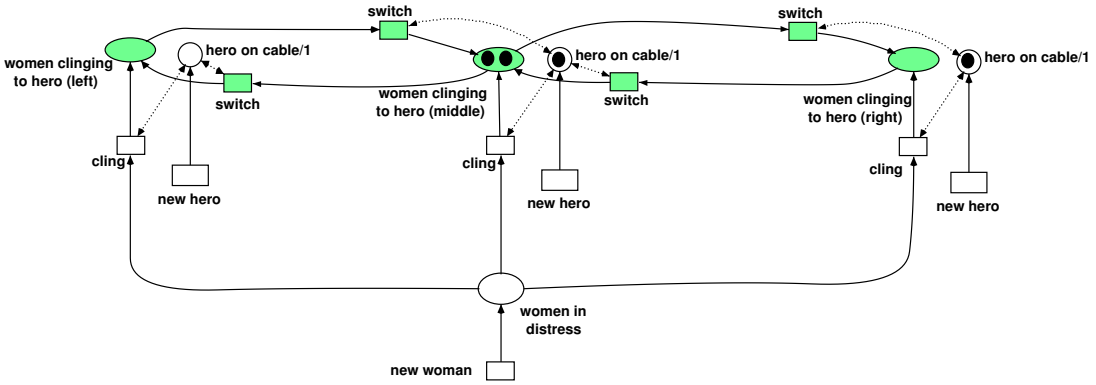


Abbildung 4.42: Das Garbage Can P/T-Netz: Generierung der women

und verhalten sich opportunistisch, indem sie zu jeweils dem `hero` wechseln, der einer Rettung am nächsten ist. Weil `women` wie `blimps` ihre Entscheidungen zwar simultan, aber unabhängig voneinander treffen, kann es passieren, dass ein "leichter" `hero` kurz vor der Rettung plötzlich von zu vielen `women` überlastet wird. "Schwere" `heros` hingegen werden plötzlich wieder rettungsfähig, weil `women` sie verlassen. Dieser Mechanismus, im GC-Modell *fluid participation* genannt, erzeugt die Möglichkeit von sinnlosen Lösungen oder Nicht-Lösungen.

Ist der `hero` mit zu vielen `women` am Arm überlastet, kann er nicht gerettet bzw. das Problem nicht gelöst werden. Wenn ein `hero` gerettet wird, nachdem ihn gerade alle `women` "verlassen" haben, ist eine Entscheidung durch Flucht (*decision by flight*) getroffen worden. Wird der `hero` gerettet, bevor die `women` ihn überhaupt als Auswahlalternative wahrnehmen und nutzen konnten, wird die Entscheidung versehentlich getroffen (*decision by oversight*). Es kommt aber auch vor, dass ein `hero` mit einer ihn nicht überlastenden Anzahl an `women` am Arm gerettet wird. Dann wurde das Problem gelöst (*decision by resolution*).

Modellierung durch Platz/Transitions-Netze

Wir entwickeln ein einfaches Petrinetzmodell für das 007-Beispiel. Das P/T-Netz in Abb. 4.41 geht von einer Situation aus, in der `heros` und `women` an drei verschieden, nebeneinander liegenden Pfeilern der Brücke positioniert sein können. Für die `women` sind dies die Plätze mit den Namen `women clinging to hero (left)`, `women clinging to hero (middle)` und `women clinging to hero (right)`. Wie bei diesem Beispiel haben verschiedene Plätze auch verschiedene Bezeichner. In den Netzen von Abbildung 4.41 bis 4.44 ist dies in den anderen Fällen jedoch nicht ausgeführt. GGf. ist `left`, `middle` bzw. `right` zu ergänzen.

In P/T-Netzen sind alle Marken ununterscheidbar „schwarz “, d.h. sie können nur dadurch unterschieden werden, dass sie in verschiedenen Plätzen liegen. Eine solche Marke kann jederzeit durch eine Transition wie `new hero` in Abb. 4.41 erzeugt werden, da diese Transitionen keinen Eingangsplatz (d.h. keinen auf sie von einem Platz zeigenden Pfeil haben). In dem Netz sind bereits vier Marken vorhanden und zwar zwei in dem Platz `women clinging to hero (middle)` für zwei `women` auf dem mittleren Brückenpfeiler und je eine auf den Plätzen `hero`

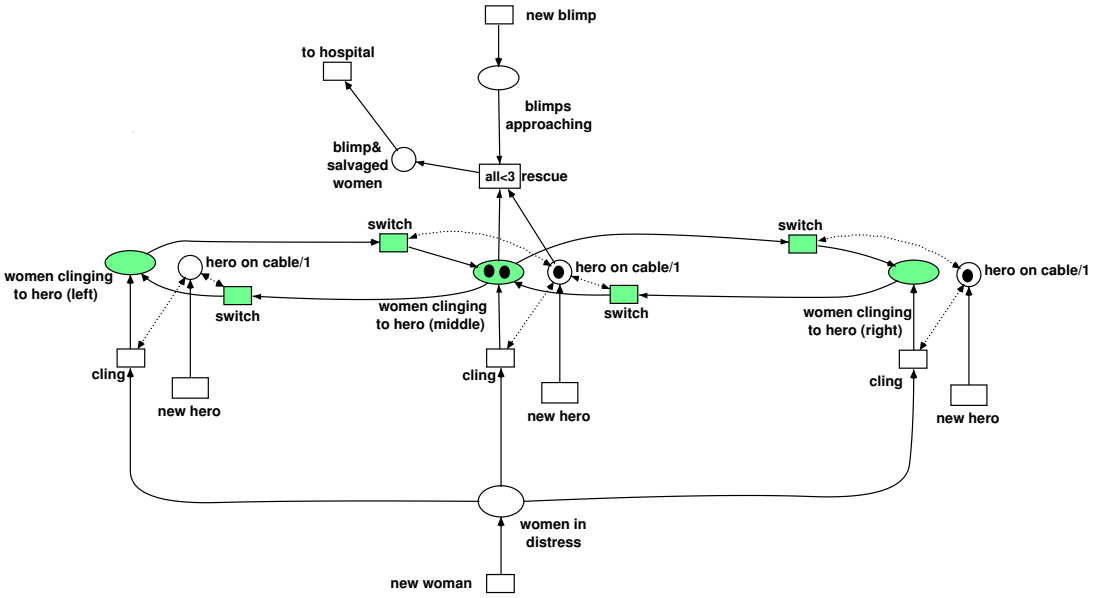


Abbildung 4.43: Das Garbage Can P/T-Netz: Blimp zur Rettung der mittleren Position

`on cable (middle)` und `hero on cable (right)` für zwei `heros` auf dem mittleren und rechten Pfeiler. Die Erweiterung `/1` bedeutet eine Anzahlbeschränkung des Platzes auf eine Marke, d.h. der Platz kann eine oder keine Marke enthalten [14]. Im Netz von Abb. 4.41 können also `heros` spontan auf der Szene erscheinen (durch die Tansitionen `new hero`, aber nur höchstens einer in jedem Platz `hero on cable` (durch die Kapazitätsbeschränkung). Weiterhin können die `women` durch Schalten der Transitionen `switch` von einem zum benachbarten `hero` wechseln, aber natürlich nur, falls ein solcher anwesend ist. Dies ist durch den punktierten Doppelpfeil (der für zwei Pfeile in entgegengesetzter Richtung steht) sicher gestellt. In Abb 4.41 kann also ein `women` vom mittleren `hero` nicht nach rechts wechseln.

In Abb. 4.42 ist eine Transition `new woman` für das spontane Erscheinen einer `woman` ergänzt. Über eine Transition `cling` kann sie sich einem `hero` zuwenden, aber natürlich wieder nur, falls ein solcher präsent. `blimps` erscheinen in dem Netz von Abb. 4.43 durch die Transition `new blimp` (zunächst nur für die mittlere Position. Sie begeben sich zu dem (anwesenden) mittleren `hero` durch eine Transition mit der Beschriftung "all < 3". Beim Schalten dieser Transition wird die Marke aus `hero on cable` und alle Marken aus `women clinging to hero` entfernt und es wird eine Marke auf den Platz `blimp & salvaged women` gelegt. Dabei kann der Platz `women clinging to hero` keine, eine oder zwei Marken enthalten. Bei einer Markenzahl ab 3 kann die Transition nicht schalten. In dem Beispielnetz ist die Kapazität der `blimps` also dahingehend beschränkt, dass ein `hero` mit bis zu zwei `women` gerettet werden kann, größere Gewichte jedoch nicht. Jede andere Kapazität hätte ebenso gewählt werden können.[15] Durch

[14] Dies gehört i.A. nicht zur Standardkonstruktion von P/T-Netzen, kann aber häufig - wie hier - durch eine Hilfskonstruktion erreicht werden.

[15] Auch Kapazitäten gehören nicht immer zur Standardkonstruktion von P/T-Netzen, können aber durch

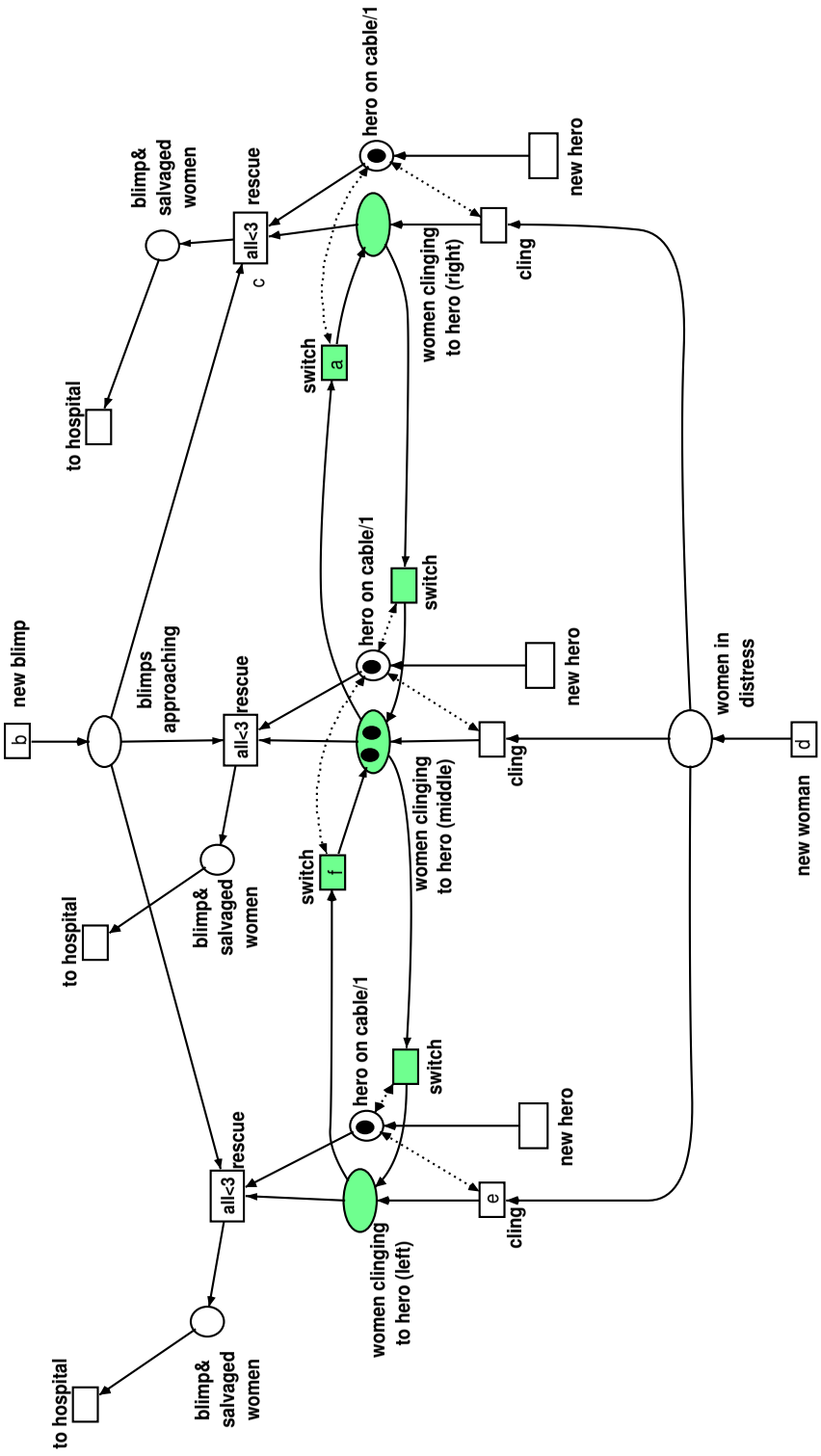


Abbildung 4.44: Das Garbage Can P/T-Netz: vollständige Darstellung

die Transition `to hospital` verläßt ein Luftschiff mit den Geretteten das Szenario.

Indem diese Ergänzung auch für die anderen beiden Pfeiler hinzugefügt wird, erhalten wir das vollständige Netz von Abb. 4.44 (die zusätzliche Marke und die Buchstabenbezeichner werden in dem nachfolgenden Beeispiel verwendet). Das Netz stellt also Abläufe dar, in denen

- `heros` ohne `women` gerettet werden,
- `heros` mit bis zu zwei `women` gerettet werden,
- eine Rettung wegen Übergewicht nicht möglich ist oder
- die `women` durch Wechsel zu einem benachbarten `hero` eine Rettung ermöglichen oder verhindern.

Beispielsweise könnte

- ein `blimp` über die rechte Transition `rescue` den rechten `hero` ohne `woman` oder
- über die mittlere Transition `rescue` den mittleren `hero` mit zwei `woman` retten.
- Wenn jedoch vorher ein weiteres `woman` zum mittleren `hero` wechselt, dann wäre eine Rettung wegen Übergewicht nicht möglich.
- Umgekehrt kann ein solcher Wechsel auch eine Rettung ermöglichen.

Als konkreten Ablauf betrachte man den Fall, dass sich in der Markierung von Abb. 4.44 ein `blimp` in Richtung des rechten `hero` nähert und eine der `women` in mittlerer Position nach rechts wechselt, eine weitere `woman` auf der Szene erscheint, sich zunächst beim linken `hero` einhängt und dann zum mittleren `hero` wechselt. An diesem Ablauf sind also die mit a bis f bezeichneten Transitionen beteiligt, von dene beispielsweise a und b unabhängig (d.h. parallel oder nebenläufig) schalten. Ein Beobachter könnte etwa die Folge $a\ b\ c\ d\ e\ f$ beobachten und daraus den Schluss ziehen, dass das `blimp` erst einfliegt (Transition b) nachdem die `woman` nach rechts gewechselt ist (a). Erst nach deren Rettung (c) finden die Ereignisse auf der linken Seite ($d\ e\ f$) statt. Wegen der Nebenläufigkeit von a und b ist natürlich auch die Folge $b\ a\ c\ d\ e\ f$ beobachtbar, woraus ein Betrachter den Schluss ziehen kann, die `woman` würde von der Mitte nach rechts wechseln, da sie sich wegen der Annäherung des `blimp` bessere Rettungschancen verspricht. Zu betonen ist hier, dass das Petrinetzmodell wie das Garbage can Modell von solchen Interpretationen abstrahiert und nur das Zustandekommen oder Nichtzustandekommen von Lösungen modelliert. Zwischen den Transitionen a bis f bestehen weitere Nebenläufigkeitsbeziehungen. So ist zum Beispiel auch die Folge $d\ e\ a\ f\ b\ c$ beobachtbar, was die Interpretation zulässt, dass die mittlere `woman` nach rechts ausweicht, weil sich die neue `woman` von links nähert. Durch die (beobachtbare) Folge $b\ d\ e\ a\ f\ c$ könnte man weiterhin meinen, die Bewegungen der `women` seien erst durch das Erscheinen (b) des `blimp` ausgelöst worden. Wir heben nochmal hervor, dass *alle* genannten vier Folgen, ähnlich wie in Abb. 3 (der Word-Datei) dem selben Ablauf des

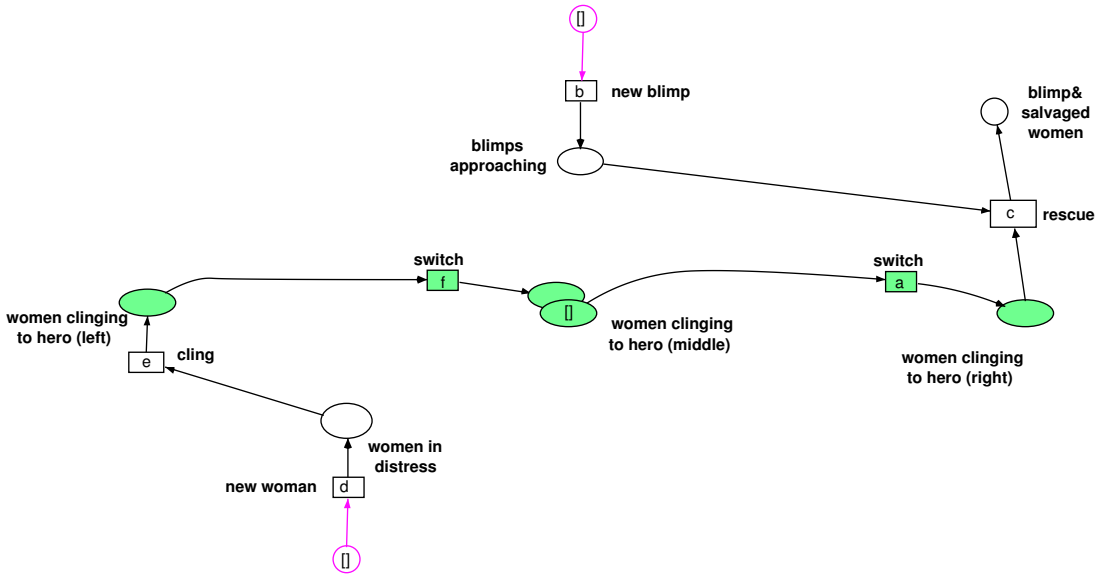


Abbildung 4.45: Ein Prozess des Garbage Can Beispiels

Petrinetzes entsprechen und nur durch unterschiedliche Reihung der Folgensemantik zustande gekommen sind!

Soll der Ablauf unabhängig spezieller Beobachtungen dargestellt werden, so ist auf das Konzept des *Prozesses* zurückzugreifen, wie es in Kapitel 6 vorgestellt werden wird. Ein Petrinetz-Prozess ist ein Ausschnitt des Petrinetzes, der alle ausgeführten Transitionen enthält, bei dem aber wiederholt markierte Plätze und wiederholt ausgeführte Transitionen entsprechend oft wiederholt gezeichnet werden. In Abb. 4.45 ist der Prozess zum vorstehenden Beispiel dargestellt. Dabei ist der Platz `women clinging to hero (middle)` zweimal enthalten, da er zweimal unabhängig markiert wurde, einmal für die am Anfang benutzte Marke und einmal am Ende. Erkennbar sind auch parallele (nebenläufige) Transitionen, wie zum Beispiel a und b oder a und e. Sie sind dadurch charkterisiert, dass sie nicht durch eine gerichtete Floge von Pfeilen verbunden sind, wie etwa

Das Netz ist leicht in ein geschlossenes System zu transformieren, in dem die ausscheidenden Marken (auch getrennt für `heros`, `women` und `blimps`) zur erzeugenden Transition zurückgeführt werden. Dies kann bei Simulationen und Leistungsbewertungen erforderlich sein.

Die hier vorgestellte Petrinetz-Modellierung zeigt die typischen Eigenschaften des Garbage Can Modells: problematische Präferenzen, unklare Technologie und flukturierende Partizipation. Masuch und LaPotin (1989) Ihre symbolische Präsentation des Garbage Can Modells besticht in ihrer Klarheit bei der Erfassung und Darstellung der Ursachen und Folgewirkungen von bounded rationality in einer Weise, die über das von March/Cohen/Olsen [**?**] vorgestellte Modell hinausgehen. Die dort vorgestellten Simulationen durch numerische Daten verfügen nicht

komplementäre Plätze dargestellt werden.

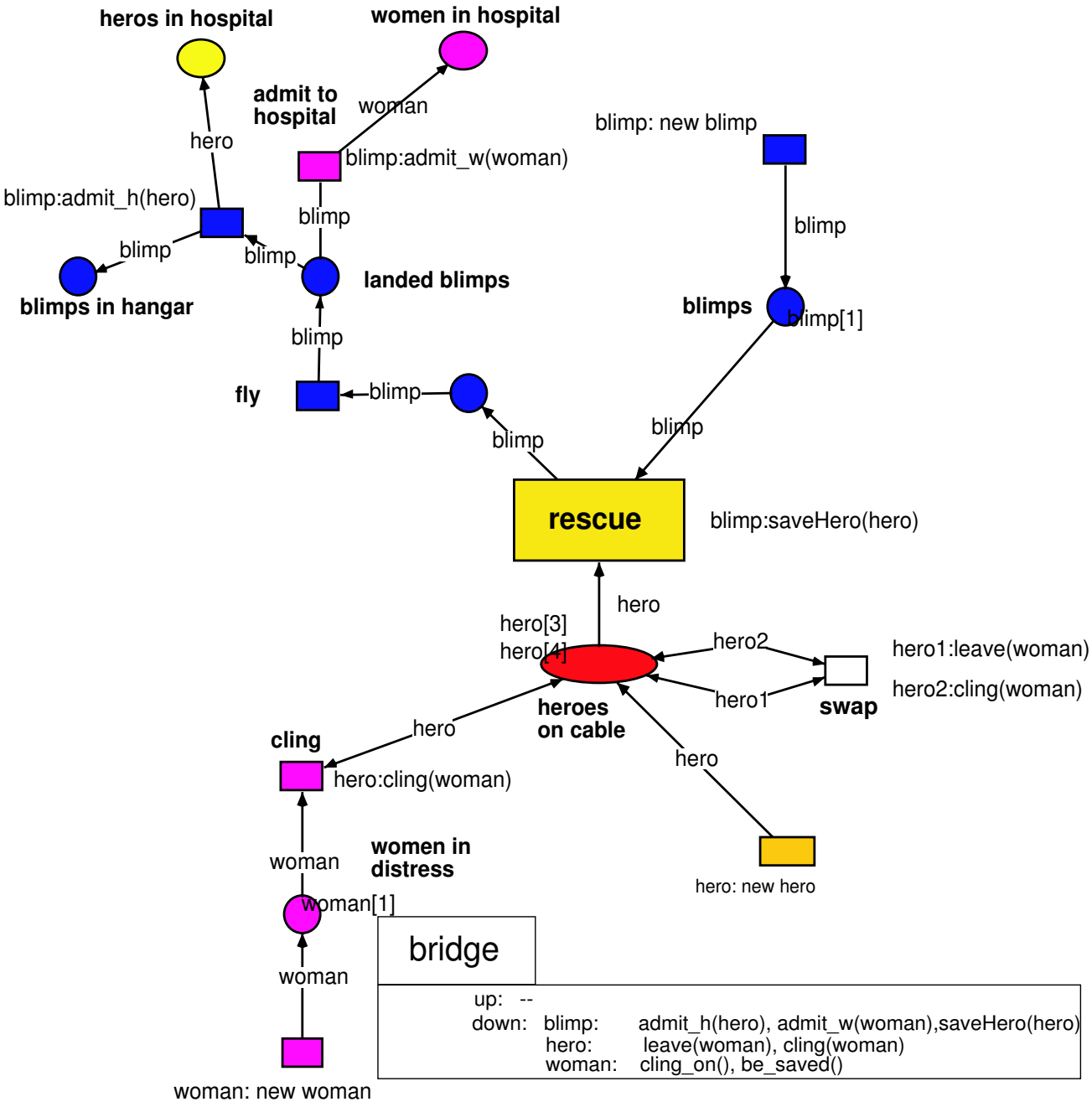
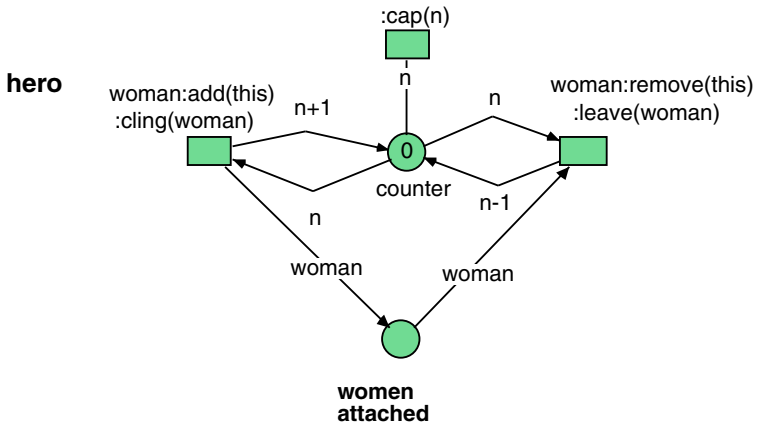


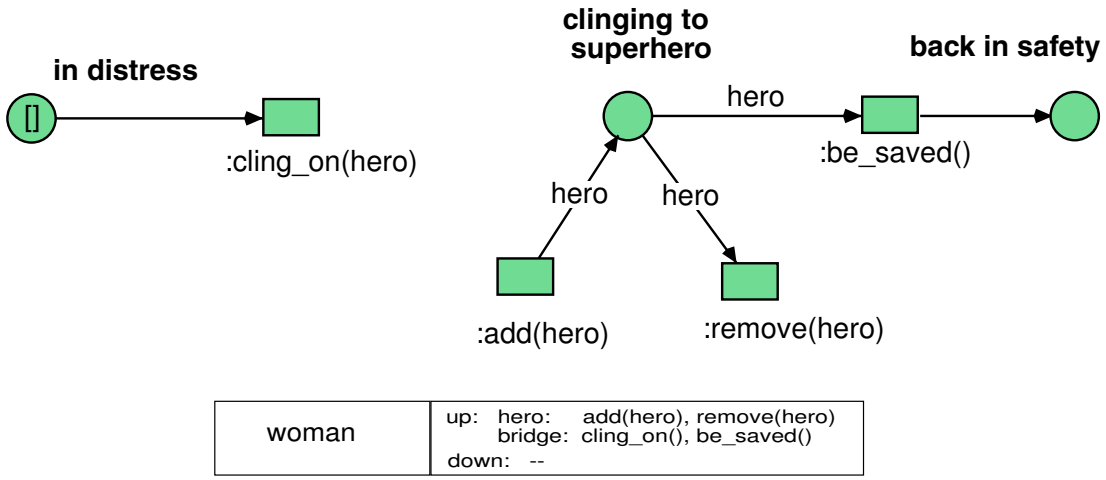
Abbildung 4.46: Garbage can: bridge



hero	up: bridge: cling(woman), leave(woman) blimp: cap(integer) down: woman: add(hero), remove(hero)

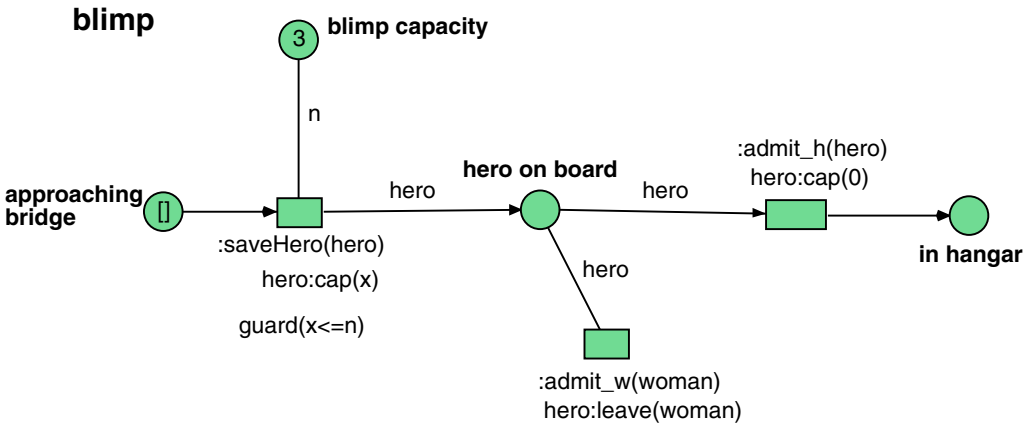
Abbildung 4.47: Garbage can: hero

über das erforderliche Potenzial, die vielfältigen und teils chaotischen Aspekte der menschlichen Entscheidungsfindung ausreichend detailliert aufzunehmen und abzubilden. Das Problem liegt in der Definition (und Formalisierung) der zumeist nur lose gekoppelten Entscheidungselemente (Probleme, Lösungen, Teilnehmer, Auswahlmöglichkeiten). Hinzu kommt, daß Entscheidungssituationen häufig durch Unsicherheit, Zufälligkeiten und den Entscheidungsspielraum beschränkende Auflagen getroffen werden (müssen).



woman	up: hero: add(hero), remove(hero) bridge: cling_on(), be_saved() down: --

Abbildung 4.48: Garbage can: woman



blimp	up: bridge: admit_h(hero), admit_w(woman), saveHero(hero) down: hero: cap(integer), leave(woman)

Abbildung 4.49: Garbage can: blimp

Das 007-Beispiel als Referenz-Netz.

Bei der Modellierung des Garbage-Can-Beispiels durch ein PT-Netz wurden die Akteure `woman`, `hero` und `blimp` durch jeweils eine Marke dargestellt, die keine interne Struktur hat. Dies war für den dort angstrebten Abstraktionsgrad passend. Zur Darstellung bestimmter Strukturen oder Verhaltensweisen kann es jedoch wünschenswert sein, die interne Struktur explixit zu programmieren. Beispieleweise soll unter bestimmten Umständen ein `hero` zählen können, wieviele Objekte `woman` ihm zugeordnet sind oder er soll sogar die Namen dieser Objekte „kennen “, d.h. intern abspeichern können.

Referenznetze eignen sich in besonderem Maße zu einer solchen detaillierten Modellierung. Bei ihnen ist es nicht nur möglich verschiedene Marken in einem Platz zu unterscheiden, sondern diese Marken können selbst wieder die Struktur eines Referenznetzes haben (siehe auch Anhang A). In Abb. 4.46 ist zunächst das Grundnetz („Systemnetz“) dargestellt, das die Erzeugung der Akteure `woman` (durch die Transition mit Beschriftung `woman: new woman`) sowie entsprechend der Akteure `hero` und `blimp` enthält. Eine Differenzierung unterschiedlicher Brückenpfeiler als Plätze für die `heros` wurde zur Vereinfachung weggelassen, könnte aber hinzugefügt werden. Die `heros` bewegen sich stattdessen auf einen gemeinsamen Platz `heros on cable` und können ein oder mehrere `women` aufnehmen. Durch die Transition `rescue` können sie von einem `blimp` aufgenommen werden und durch die Transition `fly` in Richtung Hangar fliegen. In Abb. 4.46 sind als Beispiele die Akteure `women[1]`, `hero[3]`, `hero[4]` und `blimp[1]` eingezeichnet. `blimp[1]` wird bei der Initialisierung durch die mit `blimp: new blimp` bezeichnete Transition nach dem in Abb. 4.49 angegebenen Muster erzeugt. Die `[1]` zeigt an, dass es sich um eine solche Instanz handelt, von denen es natürlich mehrere geben kann. Im Platz `heros on cable` liegen z.B. zwei Instanzen `hero[3]` und `hero[4]` des Musters `hero` (Abb. 4.47). Die Abbildungen 4.47 und 4.48 zeigen die Muster von `hero` und `woman`. In der Abbildung 4.50 ist die Instanz `hero[3]` dargestellt, in der `woman[2]` und `woman[4]` aufgenommen wurde. Entsprechend ist in `woman[2]` der `hero[3]` vermerkt (genauer gesagt: es besteht ein Verweis auf `hero[3]`).

Ein `hero` kann zählen, wieviele Objekte `woman` ihm zugeordnet sind (2 für `hero[3]` in Abb. 4.50). Dies erfolgt durch den Platz `counter`. Das Hinzufügen bzw. Entfernen bewirken Transitionen, die die up-links `cling(woman)` bzw. `leave(woman)` haben. Die zugehörigen down-links sind im Netz `bridge` aus Abbildung 4.46 zu finden. `cap(n)` ist ein up-link zu `blimp` (Netz 4.49), wodurch die momentane Last durch `blimp` abgefragt werden kann. `woman:add(this)` und `woman:remove(this)` sind down-links, wodurch `woman` mitgeteilt wird, welchem `hero` sie zugeordnet ist.

Die jeweiligen up- und down-links sind in den Netzen gesondert als Deklarationen angegeben, um die Übersichtlichkeit zu verbessern[16]. Generell wird eine Zugriffshierarchie eingehalten, die der (dynamischen) Relation “*ist enthalten in*” entspricht. In einem potentiellen Ablauf existiere zunächst nur `bridge[1]`. Dann könnten beispielsweise unabhängig von einander `woman[2]`, `woman[4]`, `hero[3]`, `hero[4]` und `blimp[1]` generiert werden (Abb. 4.51 a)). Später seien `woman[2]` und `woman[4]` dem `hero[3]` dem zugeordnet (Abb. 4.51 b)). Nun könnte `woman[4]` zu `hero[4]` wechseln (Abb. 4.51 c)). Dann nehme `blimp[1]` den `hero[4]` auf und damit implizit

[16] Von dem ausführenden Simulationswerkzeug `Renew` werden sie nicht verarbeitet.

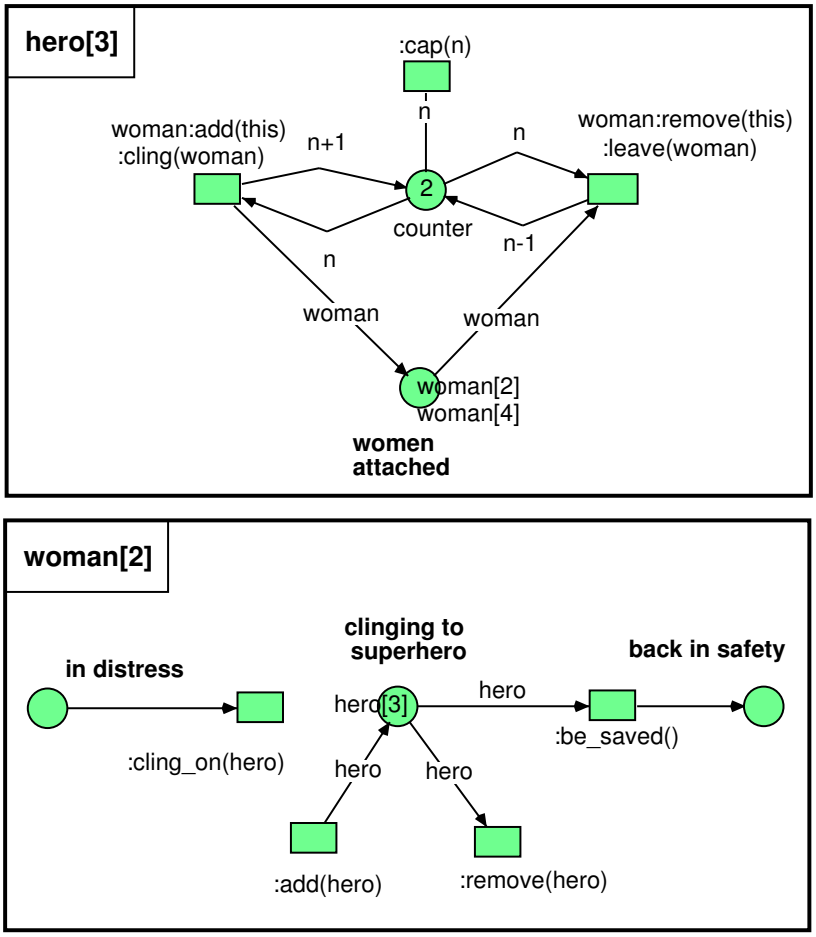


Abbildung 4.50: Die Instanzen `hero[3]` und `woman[2]`

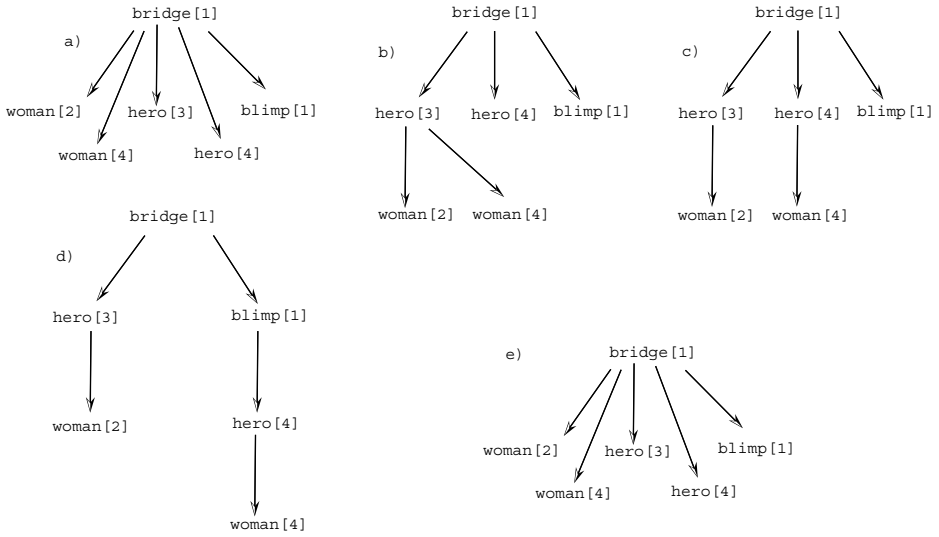


Abbildung 4.51: Dynamische Objekt-Hierarchie

auch `woman[4]` (Abb. 4.51 d)). Am Ende sind alle Objekte wieder getrennt wie in Abbildung 4.51 e) und am Anfang.

Interessant ist in diesem Zusammenhang die Modellierung des Wechsels vom `woman[4]` von `hero[3]` zu `hero[4]`. `woman[4]` kann dies nur „über“ihren `hero[3]`, der (in diesem Modell) die Kommunikation weiterleiten muss. Dieser hat jedoch auch noch keinen Zugriff auf `hero[4]`. Vielmehr erfolgt dies mit Rückgriff auf die übergeordnete `bridge[1]` in der Transition `swap`. Kommunikation von zwei Objekten kiann also nur über ein gemeinsames übergeordnetes Medium erfolgen, was natürlich der Realität entspricht. Sollen die `women` direkt per Sichtkontakt oder über Mobiltelefon kommunizieren, dann müsste dieses Medium modelliert werden.

Die Instanziierungen der down-links

```
hero1: leave(woman)      hero2: cling(woman)
```

an der Transition `swap` sind dementsprechend

```
hero[3]: leave(woman[4])      hero[4]: cling(woman[4]).
```

Dabei wird `woman[4]` in `hero[3]` entfernt und zu `hero[4]` hinzugefügt. Eine “Ebene” tiefer wird in `woman[4]` durch die Instanziierungen `:add(hero[4])` und `:remove(hero[3])` die Zugehörigkeit von `woman[4]` zu `hero[3]` in `hero[4]` umgeändert.

`bridge` hätte natürlich wie im P/T-Netz von Abbildung 4.44 modelliert werden können, d.h. mit drei Plätzen für `hero`s in Reihe und mit Übergängen zu benachbarten Pfeilern. Die vorliegende Lösung stellt dagegen die Plätze der `hero`es nicht direkt dar, erlaubt dagegen beliebig viele solche ohne Einschränkungen der Übergänge.

Folgende Vorteile der Modellierung mit Petrinetzen sind an diesem Beispiel deutlich geworden:

- Die Modellierung ist direkt, d.h. die Objekte sind unmittelbar indentifizierbar und nicht wie in [**?**] (FORTRAN) und [**?**] (VKI-Agenten) in der Programmkodierung verborgen.
- Das Modell ist anschaulich und kann auch Laien erklärt werden.
- Verschiedene Abstraktionsebenen werden sowohl bei der Modellierung als auch bei der Simulation angeboten.
- Die Modellierung ist hierarchisch, d.h. eine „gehört zu“-Relation ist direkt darstellbar.
- Das Modell ist leicht zu ändern, z.B. wurde in Abb. 4.46 oben links über das ursprüngliche Szenario hinaus die Verteilung der Akteure in `blimps in hangar`, `heros in hospital` und `woman in hospital` dargestellt.
- Das Modell ist so abstrakt, das es leicht auf andere Fälle übertragen werden kann.
- Das Modell erlaubt dynamische Änderungen, wie z.B. das Generieren von Akteuren.
- Nebenläufigkeit und Nichtdeterminismus sind ebenso Bestandteil wie bei P/T-Netzen.
- Die Modelierung von zeitlichem Verhalten, von Ressourcen wie „Ladungen“und „Energiepotentiale“wird unterstützt (hier aber nicht ausgeführt).

Aufgabe 4.27 Wie kann das Garbage Can System geändert werden, wenn

a) ein festes Netz von Plätzen für die heros vorgegeben ist,

b) ein baumartiges Netz von Plätzen für die heros dynamisch wachsen soll und

c) ein beliebiges Netz von Plätzen für die heros dynamisch wachsen kann?

Sobald ein Platz generiert wurde, kann er wiederholt von heros besucht werden, die dort ihre Heldentat vollbringen.

Kapitel 5

Verifikation und Model Checking

5.1 Einleitung

Validierung von Hardware- und Softwaresystemen wird zunehmend von Bedeutung.

System-Validierung:

a) traditionell: **Testen und Simulation**
   - am Anfang bei einfachen Fehlern sehr wirksam,
   - aber immer weniger effektiv, wenn komplexe und verborgene Fehler auftreten,
   - insbesondere bei bei Systemen mit Asynchronität, Parallelität, Nebenläufigkeit,
   - Fehler oft von speziellen Zeitparametern/Nachrichtenlaufzeiten abhängig.

b) alternativ: **formale Verifikation**
   - umfassende Prüfung,
   - alle Zweige des Verhaltens werden geprüft.

wichtige Vorgehensweisen der formale Verifikation:

a) Axiomatische Semantik (Hoare-Systeme, Zusicherungen, Invarianten)

b) temporale Logik

c) Analyse des Zustandsraumes (model checking)

a) wird in der Grundlagenvorlesung *Logik und Semantik (LOS)* behandelt. Eine Einführung zu b) und c) wird in diesem Kapitel gegeben.

Zustandsraum-Analyse/Model Checking

Vorteil:

- ohne besondere Kenntnisse anwendbar,
- bei nicht korrektem System: Generierung von Abläufen, die zu den Fehlern führen.

Nachteil:

- Größe des Zustandsraumes

Abhilfe:

- *symbolisches* Model Checking
- *Faltung*
- Ausnutzung von Symmetrien
- ....

Model Checking heißt *Analyse des Zustandsraumes*. Zustandsräume sind für praktisch alle Systemmodelle in natürlicher Weise gegeben.

Beispiele:

System	*Zustandsraum*
Programm	*Zustandsgraph*
Schaltkreis	*Zustandsgraph*
Statechart	*Transitionssystem*
Prozessausdruck	*Transitionssystem*
Petrinetz	*Markierungsgraph* = *Erreichbarkeitsgraph*

Literatur:

E.M. Clarke et al.: *Model Checking*, The MIT Press, Cambridge, 1999

B. Bèrard et al.: *Systems and Software Verification*, Springer, Berlin, 1999

C. Girault, R. Valk: *Petri Nets for Systems Engineering*, Part III: Verification Springer, Berlin, 2002,

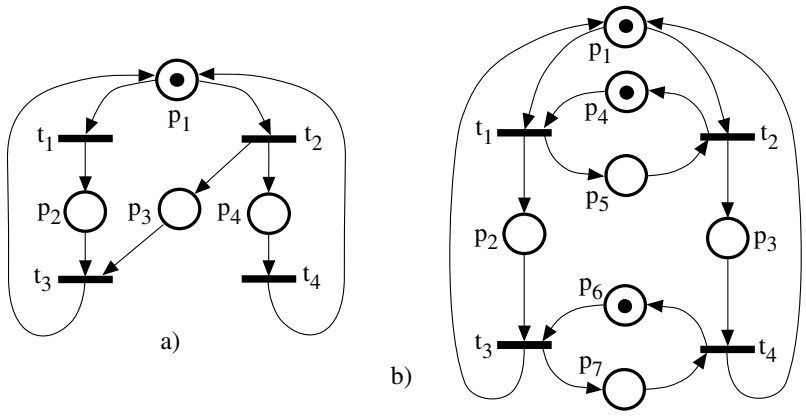


Abbildung 5.1: Beispiele: (a) ein unbeschränktes, nicht-lebendiges und nicht-reversibles Netz (b) ein lebendiges Netz, das aber bei vergrößerter Anfangsmarkierung (z.B. $\mathbf{m_0}[p_5] = 1$) nicht lebendig ist.

5.2 Elementare Systemeigenschaften

In diesem Abschnitt werden einige elementare Systemeigenschaften eingeführt. Diese stellen natürlich nur eine kleine Auswahl von solchen Eigenschaften dar. Obwohl sie für P/T-Netze formuliert werden, sind sie überwiegend auch für andere Systemmodelle (darunter gefärbte Netze) formulierbar. Mit P/T-Netze lassen sie sich jedoch besonders knapp und präzise definieren.

Die wichtigsten dieser Eigenschaften sind:

1) *Beschränktheit (boundedness)*, was die Endlichkeit des Zustandraumes bedeutet,

2) *Lebendigkeit (liveness)*, was die potenzielle Ausführbarkeit bedeutet,

3) *Reversibilität (reversibility)*, was diejenigen Systeme charakterisiert, die immer in den Anfangszustand zurückgesetzt werden können,

4) *wechselseitiger Ausschluss (mutual exclusion)*, was die Unmöglichkeit von simultanen Teilmarkierungen (p-mutex) oder Transitionsausführungen (t-mutex) bedeutet.

Das Netz in Abb. 5.1.a. ist unbeschränkt, was einem „Überlauf“ in realen Systemen entspricht. Darüberhinaus ist es nicht lebendig. Die Vermehrung von Betriebsmitteln (ressources) muss jedoch nicht zu einem Lebendigen Netz führen, sondern kann sogar diese Eigenschaft zerstören, wie Abb. 5.1.b. zeigt.

Für *jede* Anfangsmarkierung gilt für das Netz in Abb. 5.2a die folgenden Markenerhaltungsgesetze (P-Invarintengleichungen)

$$\mathbf{m}[p_1] + \mathbf{m}[p_2] + \mathbf{m}[p_3] = \mathbf{m_0}[p_1] + \mathbf{m_0}[p_2] + \mathbf{m_0}[p_3] = k_1(\mathbf{m_0})$$

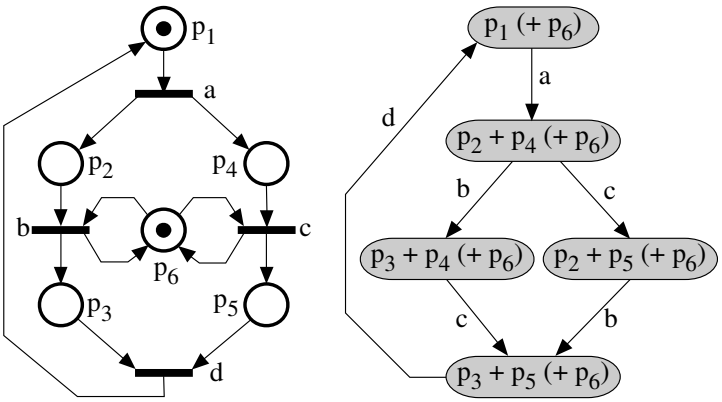


Abbildung 5.2: Beschränktes, lebendiges und reversibles Netz mit Erreichbarkeitsgraph

$$\mathbf{m}[p_1] + \mathbf{m}[p_4] + \mathbf{m}[p_5] = \mathbf{m_0}[p_1] + \mathbf{m_0}[p_4] + \mathbf{m_0}[p_5] = k_2(\mathbf{m_0})$$
$$\mathbf{m}[p_6] = \mathbf{m_0}[p_6] = k_3(\mathbf{m_0})$$

wobei $\mathbf{m_0}$ die Anfangsmarkierung und $\mathbf{m}$ eine beliebige erreichbare Markierung ist. $k_i(\mathbf{m_0}) \in \mathbb{Z}$ ist jeweils eine von $\mathbf{m_0}$ abhängige Konstante. Daher gelten die Ungleichungen:

$$\mathbf{m}[p_1] \leq \min(k_1(\mathbf{m_0}), k_2(\mathbf{m_0}))$$
$$\mathbf{m}[p_i] \leq k_1(\mathbf{m_0}); i = 2, 3$$
$$\mathbf{m}[p_j] \leq k_2(\mathbf{m_0}); j = 4, 5$$
$$\mathbf{m}[p_6] = k_3(\mathbf{m_0})$$

Sie beweisen, dass das Netz für *jede* Anfangsmarkierung beschränkt ist. Dies ist natürlich eine stärkere Eigenschaft als Beschränktheit. Da sie nur von der Struktur des Netzes und nicht von der jeweils gewählten Anfangsmarkierung abhängt, heißt sie *strukturelle Beschränktheit (structural boundedness).*

Wenn im Netz von Abb.5.1a die Transition t_1 schaltet, wird eine *Verklemmung (deadlock)* erreicht, d.h. eine Markierung, in der keine Transition aktiviert ist. Ein Netz heißt *verklemmungsfrei (deadlock-free)* wenn die Erreichbarkeitsmenge keine Verklemmung enthält. *Lebendigkeit (liveness)* ist eine stärkere Eigenschaft. Eine Transition t heißt *potenziell aktivierbar (potentially fireable)* in einer gegebenen Markierung $\mathbf{m}$, wenn eine aktivierte Schaltfolge $\sigma \in T^*$ existiert, die zu einer Markierung $\mathbf{m}'$ führt, in der t aktiviert ist. Eine Transition heißt *lebendig (live)*, wenn sie in jeder erreichbaren Markierung potenziell aktivierbar ist. Ein Netz heißt *lebendig*, wenn alle Transitionen in der Anfangsmarkierung lrbendig sind.

Egal wie man die Anfangsmarkierung des Netzes in Abb. 5.1a wählt, es ist *nicht* lebendig. Dies ist also wieder eine strukturelle Eigenschaft, die daher *strukturelle Nichtlebendigkeit (structural*

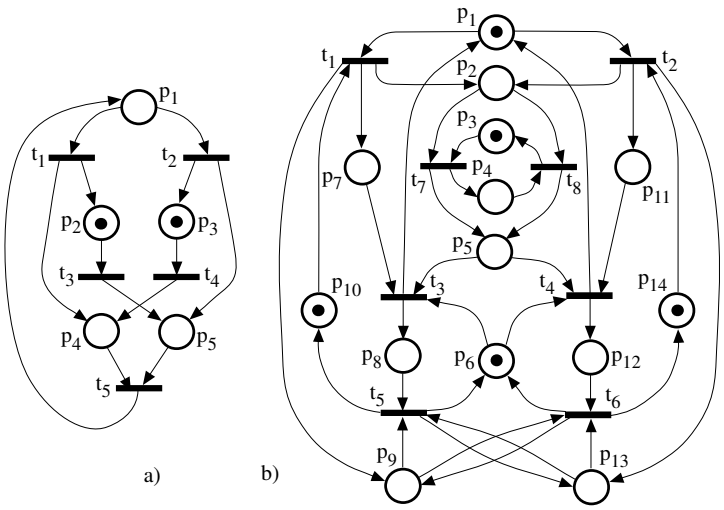


Abbildung 5.3: (a) Die Anfangsmarkierung ist kein Rücksetzzustand, jedoch alle übrigen; (b) Die Anfangsmarkierung ist kein Rücksetzzustand, der übrige Erreichbarkeitsgraph zerfällt in zwei streng zusannenhängende Zusammenhangskomponenten.

non-liveness) heißt. Umgekehrt heißt ein Netz *strukturell lebendig (structural live)*, wenn für es eine lebendige Eigenschaft existiert.

Eine Markierung heißt *Rücksetzzustand (home state)*, wenn sie von jeder erreichbaren Markierung erreicht werden kann. Die Anfangsmarkierung des Netzes in Abb. 5.3a ist kein Rücksetzzustand. Das gilt auch für das Netz in Abb 5.3.b. Es ist ebenfalls lebendig und beschränkt, besitzt aber keinen Rücksetzzustand. Sein Erreichbarkeitsgraph enthält nämlich zwei verschiedene starke Zusammenhangskomponenten, die jeweils alle Transitionen enthalten. Die Menge aller Rücksetzzustände eines Netzes heißt *Rücksetzmenge (home space)*. Die Existenz von Rücksetzzuständen ist natürlich generell für Systeme wünschenswert.

Lebendigkeit, Beschränktheit und Reversibilität sind prominente und „gute“ Systemeigenschaften und man kann die Frage stellen, ob sie voneinander unabhängig sind, d.h. jede Kombination möglich ist. Durch Abbildung 5.4 wird dies tatsächlich bewiesen, da alle 2^3 Kombinationen vertreten sind.

Die letzte der hier betrachteten elementaren Systemeigenschaften ist der *wechselseitige Ausschluss (mutual exclusion)*. Diese Eigenschaft umfasst Bedingungen, wie die Unmöglichkeit des gleichzeitigen Zugriffs zweier Roboter auf ein Objekt.

Zwei Plätze (bzw. Transitionen) befinden sich im wechselseitigen Ausschluss, wenn sie niemals gleichzeitig markiert sind (bzw. schalten). Beispielsweise gilt für das Netz in Abb. 5.2 für jede erreichbare Markierung $\mathbf{m}$: $\mathbf{m}[p_1]+\mathbf{m}[p_2]+\mathbf{m}[p_3] = 1$. Also sind p_1, p_2, und p_3 im wechselseitigen Ausschluss.

In der Tabelle 5.1 sind die formalen Definitionen der diskutierten Eigenschaften zusammengefasst. Darin sind Notationen enthalten, die in den folgenden Definitionen wiederholt bzw.

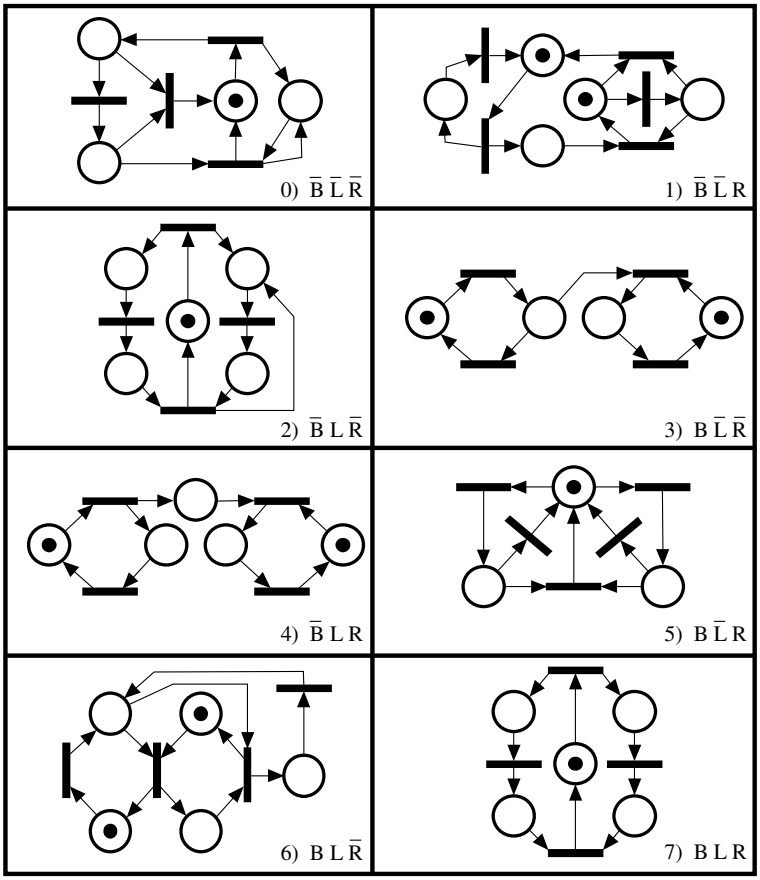


Abbildung 5.4: Beschränktheit (B), Lebendigkeit (L) und Reversibilität (R) sind unabhängige Eigenschaften

eingeführt werden.

Definition 5.1 *Ein Platz/Transitions-Netz (P/T-Netz) wird als Tupel* $\mathcal{N} = \langle P, T, F, W, \mathbf{m}_0 \rangle$ *definiert, wobei*

- (P, T, F) *ein endliches Netz (Def. 4.1) ist ,*
- $W : F \to \mathbb{N}$ *Kantengewichtung heißt und*
- $\mathbf{m}_0 : P \to \mathbb{N}$ *die Anfangsmarkierung ist.*

Um die Anfangsmarkierung $\mathbf{m_0}$ *hervorzuheben, definiert man ein* Netzsystem *als* $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ *mit* $\mathcal{N} = \langle P, T, F, W \rangle$.

Definition 5.2 *a) Die Markierung eines* $P/T-Netzes$ $\mathcal{N} = \langle P, T, F, W, \mathbf{m}_0 \rangle$ *ist ein Vektor* $\mathbf{m}$ *mit* $\mathbf{m}(p) \in \mathbb{N}$ *für jedes* $p \in P$ *(auch als Abbildung* $\mathbf{m} : P \to \mathbb{N}$ *aufzufassen). Die Menge aller Markierungen über* P *(bzw.* S*) wird mit* M_P *(bzw.* M_S*) bezeichnet.*

b) Eine Transition $t \in T$ *heißt* aktiviert *in einer Markierung* $\mathbf{m}$ *falls*

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

(als Relation: $\mathbf{m} \xrightarrow{t}$*).*

c) Es sei $\widetilde{W}(p,t) := \begin{cases} W(p,t) & \text{falls } (p,t) \in F \\ 0 & \text{sonst} \end{cases}$

und entsprechend $\widetilde{W}(t,p) := \begin{cases} W(t,p) & \text{falls } (t,p) \in F \\ 0 & \text{sonst} \end{cases}$

Ist t *in* $\mathbf{m}$ *aktiviert, dann ist die* Nachfolgemarkierung *definiert durch:*

$$\mathbf{m} \xrightarrow{t} \mathbf{m}' \Leftrightarrow \forall p \in P.\ (\mathbf{m}(p) \geq \widetilde{W}(p,t) \wedge \mathbf{m}'(p) = \mathbf{m}(p) - \widetilde{W}(p,t) + \widetilde{W}(t,p))$$

.

d) Definiert man $W(\bullet, t) := (\widetilde{W}(p_1,t), \ldots, \widetilde{W}(p_{|P|},t))$ *als Vektor der Länge* $|P|$ *und entsprechend* $W(t,\bullet) := (\widetilde{W}(t,p_1), \ldots, \widetilde{W}(t,p_{|P|}))$ *, dann kann die Nachfolgemarkierung einfacher durch Vektoren definiert werden:*

$$\mathbf{m} \xrightarrow{t} \mathbf{m}' \ \Leftrightarrow\ \mathbf{m} \geq W(\bullet,t) \ \wedge\ \mathbf{m}' = \mathbf{m} - W(\bullet,t) + W(t,\bullet))$$

Dabei sind die Operatoren auf $\mathbb{N}$ *komponentenweise auf Vektoren zu erweitern.*

Definition 5.3 *Die Nachfolgemarkierungsrelation von Definition 5.2 wird wie üblich auf Wörter über* T *erweitert:*

- $\mathbf{m} \xrightarrow{w} \mathbf{m}'$ *falls* w *das leere Wort* λ *ist und* $\mathbf{m} = \mathbf{m}'$*,*
- $\mathbf{m} \xrightarrow{wt} \mathbf{m}'$ *falls* $\exists \mathbf{m}'' : \mathbf{m} \xrightarrow{w} \mathbf{m}'' \wedge \mathbf{m}'' \xrightarrow{t} \mathbf{m}'$ *für* $w \in T^*$ *und* $t \in T$*.*

Die Menge $\mathbf{R}(\mathcal{N}) := \{\mathbf{m} | \exists w \in T^* : \mathbf{m_0} \xrightarrow{w} \mathbf{m}\}$ *ist die Menge der* erreichbaren *Markierungen oder auch* Erreichbarkeitsmenge*. Für ein Netzsystem* $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ *wird die Erreichbarkeitsmenge durch* $\mathbf{R}(\mathcal{S}) = \mathbf{R}(\langle \mathcal{N}, \mathbf{m_0} \rangle)$ *bezeichnet.*

(1)	Schranke (bound) des Platzes p in $\langle \mathcal{N}, \mathbf{m_0} \rangle$: $\mathbf{b}(p) = \sup\{\mathbf{m}[p] \vert \mathbf{m} \in \mathbf{R}(\mathcal{N}, \mathbf{m_0})\}$
(2)	p heißt beschränkt (bounded) in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ falls $\mathbf{b}(p) < \infty$
(3)	$\langle \mathcal{N}, \mathbf{m_0} \rangle$ heißt beschränkt, wenn alle Plätze beschränkt sind.
(4)	$\langle \mathcal{N}, \mathbf{m_0} \rangle$ heißt verklemmungsfrei (deadlock-free) falls $\forall \mathbf{m} \in \mathbf{R}(\mathcal{N}, \mathbf{m_0}).\ \exists t \in T : \mathbf{m} \xrightarrow{t}$
(5)	t heißt lebendig (live) in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ falls $\forall \mathbf{m} \in \mathbf{R}(\mathcal{N}, \mathbf{m_0}).\ \exists \sigma \in T^* : \mathbf{m} \xrightarrow{\sigma t} \mathbf{m'}$
(6)	$\langle \mathcal{N}, \mathbf{m_0} \rangle$ heißt lebendig falls alle Transitionen lebendig sind.
(7)	$\mathbf{m} \in \mathbf{R}(\mathcal{N}, \mathbf{m_0})$ heißt a Rücksetzzustand (home state) falls $\forall \mathbf{m'} \in \mathbf{R}(\mathcal{N}, \mathbf{m_0}).\ \exists \sigma \in T^*$: $\mathbf{m'} \xrightarrow{\sigma} \mathbf{m}$
(8)	$\langle \mathcal{N}, \mathbf{m_0} \rangle$ heißt reversibel (reversible) falls $\forall \mathbf{m} \in \mathbf{R}(\mathcal{N}, \mathbf{m_0}).\ \exists \sigma \in T^* : \mathbf{m} \xrightarrow{\sigma} \mathbf{m_0}$
(9)	wechselseitiger Ausschluss (mutual exclusion) in $\langle \mathcal{N}, \mathbf{m_0} \rangle$: p_i und p_j sind in Markierungs-Ausschluss (marking mutual exclusion) falls $\nexists \mathbf{m} \in \mathbf{R}(\mathcal{N}, \mathbf{m_0})$: $(\mathbf{m}[p_i] > 0) \wedge (\mathbf{m}[p_j] > 0)$ t_i und t_j sind in Schalt-Ausschluss (firing mutual exclusion) falls $\nexists \mathbf{m} \in \mathbf{R}(\mathcal{N}, \mathbf{m_0}) : \mathbf{m} \geq W(\bullet, t_i) + W(\bullet, t_j)]$
(10)	Strukturelle Eigenschaften : $\mathcal{N}$ heißt strukturell beschränkt (structurally bounded) falls $\forall \mathbf{m_0}$: $\langle \mathcal{N}, \mathbf{m_0} \rangle$ ist beschränkt $\mathcal{N}$ heißt strukturell lebendig (structurally live) falls $\exists \mathbf{m_0} : \langle \mathcal{N}, \mathbf{m_0} \rangle$ ist lebendig

Tabelle 5.1: Definition der elementaren Systemeigenschaften

5.3 Verifikation durch den Erreichbarkeitsgraphen

Verifikationsmethoden für Petrinetze:

a) enumerative Methoden:
   Analyse des Erreichbarkeitsgraphen

b) Transformationen
   Reduktion

c) strukturelle Analyse
   lineare Algebra-Methoden (P-Invarianten, T-Invarianten)
   graphenbasierte Methoden

Definition 5.4 *Der* Erreichbarkeitsgraph *eines Netzsystems* $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ *ist ein gerichteter Graph* $\mathrm{RG}(\mathcal{S}) = (V, E)$*, mit Knotenmenge* $V = \mathbf{R}(\mathcal{S})$ *und Kantenmenge* $E = \{\langle \mathbf{m}, t, \mathbf{m}' \rangle | \mathbf{m}, \mathbf{m}' \in \mathbf{R}(\mathcal{S})$ *und* $\mathbf{m} \xrightarrow{t} \mathbf{m}'\}$.

If the net system $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ is bounded, the $\mathrm{RG}(\mathcal{S})$ is finite and it can be constructed, for example, by the algorithm 5.1. It finishes when all the possible firings from the reachable markings have been explored. The tagging scheme in step 2.1 ensures that no marking is visited more than once. Each marking visited is tagged (step 2.1), and step 2.2.3 ensures that the only markings added to V are ones that have not previously added. When a marking is visited, only those edges representing the firing of an enabled transition are added to the set E in 2.2.4.

The construction of the reachability graph is a very hard problem from the computational point of view. This is because the size of the state space may grow more than exponentially with respect to the size of the Petri net model (measured, for example, by the number of places). In [**?**] the reader can find a discussion on the size of the reachability graph obtained from a Petri net, the role of the concurrency in the state space explosion problem and some methods to obtain reduced representations of the state space.

Let us consider, for example, the net system in Figure 5.2 removing the place p_6 and its reachability graph, obtained by applying the algorithm 5.1,. The net system has five markings, thus it is bounded. It is also easy to conclude that all places are 1-bounded. A closer look allows to state that p_1, p_2 and p_3 (p_1, p_4 and p_5) are in mutual exclusion. Moreover, considering RS and the net structure (the pre-function), firing concurrency between transitions b and c can be decided. Observe at this point that introducing p_6 in our net system, the graph structure of the reachability graph does not change, but transitions b and c become in firing mutual exclusion. This example shows that the obtained RG is a *sequentialised observation* of the net system behaviour, and therefore it is not possible to distinguish on it if two fired transitions from a same marking can be fired concurrently or if they are in conflict relation. To avoid this problem,

Algorithmus 5.1 (Computation of the Reachability Graph)

Input - The net system $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$
Output - The directed graph $\mathrm{RG}(\mathcal{S}) = (V, E)$ for bounded net systems

1. Initialise $\mathrm{RG}(\mathcal{S}) = (\{\mathbf{m_0}\}, \emptyset)$; $\mathbf{m_0}$ is untagged;
2. **while** there are untagged nodes in V **do**
   - 2.1 Select an untagged node $\mathbf{m} \in V$ and tag it
   - 2.2 **for** each enabled transition, t, at $\mathbf{m}$ **do**
     - 2.2.1 Compute $\mathbf{m}'$ such that $\mathbf{m} \xrightarrow{t} \mathbf{m}'$;
     - 2.2.2 **if** there exists $\mathbf{m}'' \in V$ such that $\mathbf{m}'' \xrightarrow{\sigma} \mathbf{m}'$ and $\mathbf{m}'' \lneq \mathbf{m}'$
       then the algorithm fails and exits;
       (the unboundedness condition of $\mathcal{S}$ has been detected)
     - 2.2.3 **if** there is no $\mathbf{m}'' \in V$ such that $\mathbf{m}'' = \mathbf{m}'$
       then $V := V \cup \{\mathbf{m}'\}$; ($\mathbf{m}'$ is an untagged node)
     - 2.2.4 $E := E \cup \{\langle \mathbf{m}, t, \mathbf{m}' \rangle\}$
3. The algorithm succeeds and $\mathrm{RG}(\mathcal{S})$ is the reachability graph

other reachability graphs capturing the true concurrency can be constructed. The basic idea is to increase the number of edges of the conventional RG representing the concurrent firing of transitions from each marking.

For unbounded net systems, $\mathcal{S}$, $\mathrm{RS}(\mathcal{S})$ is not a finite set and therefore the construction of $\mathrm{RG}(\mathcal{S})$ never ends. Karp and Miller [**?**] showed how to detect unboundedness of a net system by means of the following condition (incorporated in step 2.2.2 of algorithm 5.1 as a break condition): the system $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ is unbounded iff there exists $\mathbf{m}'$ reachable from $\mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$, such that $\mathbf{m} \lneq \mathbf{m}'$ (the repetition of σ allows to conclude on unboundedness because the occurrence of σ strictly increases the content of tokens of the starting marking $\mathbf{m}$).

Coverability graphs allow to obtain finite representations of the RG of unbounded net systems [**?**, **?**, **?**, **?**]. Roughly speaking, in a coverability graph the set of nodes is a finite set of marking vectors (called the coverability set) that covers all the markings of the reachability set. There is an edge, representing the firing of a transition t, between two nodes, $\mathbf{m}$ and $\mathbf{m}'$ if and only if t is fireable from $\mathbf{m}$ and a marking covered by $\mathbf{m}'$ is reached. The loss of information in the computation of a coverability graph makes that many important properties (e.g. marking reachability or deadlock freeness) cannot be decided on it.

In order to analyse a given property in a bounded net system, the reachability graph is used as the basis for the corresponding *decision procedure*. It allows to decide whether the net system satisfies a given property. All procedures are, in general, of exponential complexity in the size of the net (measured, for example, by the number of places) but they are of polynomial complexity on the size of the reachability graph (measured, for example, by the number of nodes and arcs). The focus of the rest of this section is in two general decision procedures.

In the sequel we will call *marking predicate* a propositional formula whose atoms are inequalities of the form: $\sum_{p \in A} k_p \mathbf{m}[p] \leq k$, where k_p and k are rational constants and A is a subset of places. Let us consider a net system $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$.

Algorithmus 5.2 (Decision procedure for marking invariances)

Input - The reachability graph $\mathrm{RG}(\mathcal{N}, \mathbf{m_0})$. The property Π.
Output - TRUE if the property is verified.

1. Initialise all elements of $\mathrm{RS}(\mathcal{S})$ as untagged.
2. **while** there is an untagged node $\mathbf{m} \in \mathrm{RS}(\mathcal{S})$ **do**
   2.1 Select an untagged node $\mathbf{m} \in \mathrm{RS}(\mathcal{S})$ and tag it
   2.2 **if** $\mathbf{m}$ does not satisfy Π
   then return FALSE (the property is not verified).
3. Return TRUE

The first group of properties are the so called *marking invariance properties*. A given marking predicate, Π, must be satisfied for all reachable markings (hence the name of marking invariance property): $\forall \mathbf{m} \in \mathrm{RS}(\langle \mathcal{N}, \mathbf{m_0} \rangle)$, $\mathbf{m}$ satisfies Π. Examples of this are:

1) *k-boundedness of place p*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S}) \,.\, \mathbf{m}[p] \leq k$.

2) *Marking mutual exclusion between p and p'*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S}) \,.\, ((\mathbf{m}[p] = 0) \vee (\mathbf{m}[p'] = 0))$.

3) *Deadlock freeness*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S}) \,.\, \exists t \in T \,.\, \mathbf{m} \geq \mathbf{Pre}[P, t]$.

Marking invariance properties can be decided through Algorithm 5.2, which is linear in the size of $\mathrm{RG}(\mathcal{S})$: each node is visited no more than once. If the algorithm succeeds, then all reachable markings from $\mathbf{m_0}$ satisfy Π. If the algorithm fails at step 2.2, there is a path in the $\mathrm{RG}(\mathcal{S})$ from $\mathbf{m_0}$, containing at least a marking that does not satisfy Π.

Beispiel 5.5 Analysis of marking invariance properties Let us consider the net system in Figure 5.2 for which $\mathrm{RS}(\mathcal{S}) = \{p_1+p_6, p_2+p_4+p_6, p_3+p_4+p_6, p_2+p_5+p_6, p_3+p_5+p_6\}$. The execution of Algorithm 5.2 to verify the mutual exclusion property between places p_5 and p_6 ($\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S}) \,.\, (\mathbf{m}[p_5] = 0) \vee (\mathbf{m}[p_6] = 0)$) starts initialising all elements of $\mathrm{RS}(\mathcal{S})$ as untagged (step 1). Then the markings are visited one by one (e.g. in the previous order) until $p_2+p_5+p_6$ is visited, where the predicate Π is false, hence the algorithm stops and return FALSE.

The second group of properties are the so called *liveness invariance properties*. For each reachable marking of a net system, $\mathbf{m}$, there exists at least a reachable marking from it satisfying the property Π: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S}) \,.\, \exists \mathbf{m'} \in \mathrm{RS}(\mathcal{N}, \mathbf{m}) \,.\, (\mathbf{m'}$ satisfies $\Pi)$. Examples of this are:

1) *Liveness of t*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S}) \,.\, \exists \mathbf{m'} \in \mathrm{RS}(\mathcal{N}, \mathbf{m}) \,.\, \mathbf{m'} \geq \mathbf{Pre}[P, t]$.

2) $\mathbf{m}_H$ *is home state*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S}) \,.\, \exists \mathbf{m'} \in \mathrm{RS}(\mathcal{N}, \mathbf{m}) \,.\, \mathbf{m'} = \mathbf{m}_H$.

3) *Reversibility*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S}) \,.\, \exists \mathbf{m'} \in \mathrm{RS}(\mathcal{N}, \mathbf{m}) \,.\, \mathbf{m'} = \mathbf{m_0}$.

Algorithmus 5.3 (Decision procedure for liveness invariances)

Input - The reachability graph $\mathrm{RG}(\mathcal{N}, \mathbf{m_0})$. The property Π
Output - TRUE if the property is verified.

1. Decompose $\mathrm{RG}(\mathcal{N}, \mathbf{m_0})$ into its strongly connected components $C_1, \ldots, C_r$
2. Obtain the graph $\mathrm{RG}^c(\mathcal{S}) = (V_c, E_c)$ by shrinking $C_1, \ldots, C_r$ to a single node, i.e. $V_c = \{C_1, \ldots, C_r\}$. $\langle C_i, t, C_j \rangle \in E_c$ iff there exists $\langle \mathbf{m}, t, \mathbf{m}' \rangle \in E$, such that $\mathbf{m}$ is in the SCC C_i, $\mathbf{m}'$ is in the SCC C_j, and $i \neq j$.
3. Compute the set F of terminal strongly connected components from $\mathrm{RG}^c(\mathcal{S})$
4. **while** there is a $C_i \in F$ **do**
   4.1 **if** C_i does not contain a $\mathbf{m}'$ satisfying Π
   then return FALSE
   4.2 Remove C_i from F
5. Return TRUE

These properties cannot be verified by an exclusive linear inspection of the reachability set (as in algorithm 5.2). The verification requires to find a reachable marking, satisfying Π, from each one of the markings in $\mathrm{RS}(\mathcal{S})$. In order to verify the property we will classify the markings of $\mathrm{RS}(\mathcal{S})$ into subsets of mutually reachable markings through the concept of strongly connected component of a directed graph. Therefore, the property will be easily verified checking that each terminal strongly connected component contains at least a marking satisfying Π. We recall now some basic concepts.

A *path* in a reachability graph $\mathrm{RG}(\mathcal{S})$ is any sequence $\mathbf{m}_1 \ldots \mathbf{m}_i \mathbf{m}_{i+1} \ldots \mathbf{m}_k$ of nodes of $\mathrm{RG}(\mathcal{S}) = (V, E)$ where all successive nodes $\mathbf{m}_i$ and $\mathbf{m}_{i+1}$ in the path satisfy that $\langle \mathbf{m}_i, t, \mathbf{m}_{i+1} \rangle \in E$ for some t. The reachability graph, $\mathrm{RG}(\mathcal{S})$, is *strongly connected (streng zusammenhängend)* iff there is a path from each node in V to any other node in V. A *strongly connected component (strenge Zusammenhangskomponente)* (SCC) of a reachability graph is a maximal strongly connected subgraph. A strongly connected component of a graph will be called *terminal* if no node in the component has an edge leaving the component. The strongly connected components of a digraph (V, E) can be found in order $(|V| + |E|)$ steps (e.g. [**?**]). Abb. 5.6 zeigt die strengen Zusammenhangskomponenten des Graphs von Abb. 5.5, von denen zwei terminal sind.

When computing the SCCs $C_1, \ldots, C_r$ of a reachability graph $\mathrm{RG}(\mathcal{S}) = (V, E)$, a new graph $\mathrm{RG}^c(\mathcal{S}) = (V_c, E_c)$ (genannt *reduzierter Graph*) is induced by shrinking the strongly connected components to a single node, i.e. $V_c = \{C_1, \ldots, C_r\}$. For each edge $\langle \mathbf{m}, t, \mathbf{m}' \rangle \in E$, such that $\mathbf{m}$ is in a SCC C_i, and $\mathbf{m}'$ is in a different SCC C_j, there is an induced edge $\langle C_i, t, C_j \rangle \in E_c$. In Abb. 5.7 ist der reduzierte Graph von Abb. 5.5 dargestellt. Den beiden terminalen SCCs entsprechend, hat der azyklische Graph zwei Knoten, von denen keine Kanten ausgehen. Als Anfangsknoten wird der Knoten definiert, der aus derjenigen SCC entstanden ist, die den Anfangsknoten enthält (vorausgesetzt, der Ausgangsgraph hatte einen solchen).

The graph $\mathrm{RG}^c(\mathcal{S})$ is an acyclic digraph. Therefore the terminal SCCs of $\mathrm{RG}(\mathcal{S})$ can be computed with linear complexity in the size of $\mathrm{RG}^c(\mathcal{S})$. This fact will be exploited in the algorithm 5.3 for liveness invariance checking. Algorithm 5.3 allows to decide liveness invariance properties. The algorithm is of linear complexity in the size of the $\mathrm{RG}(\mathcal{S})$. If the algorithm succeeds, all

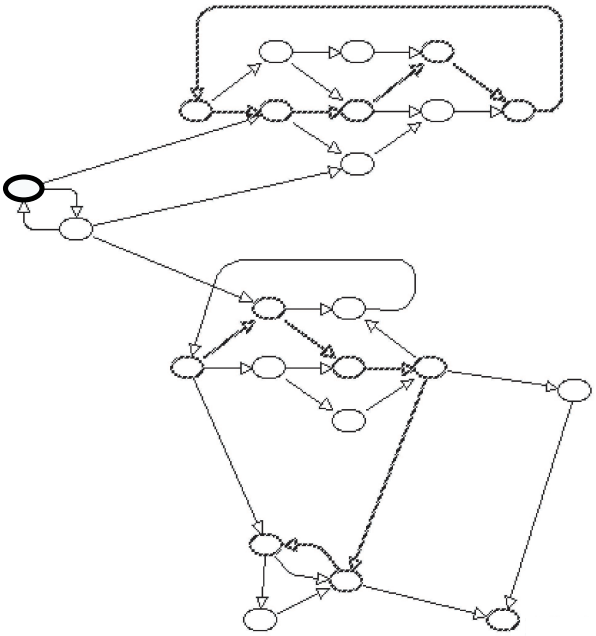


Abbildung 5.5: Ein Graph mit Anfangsknoten (fett)

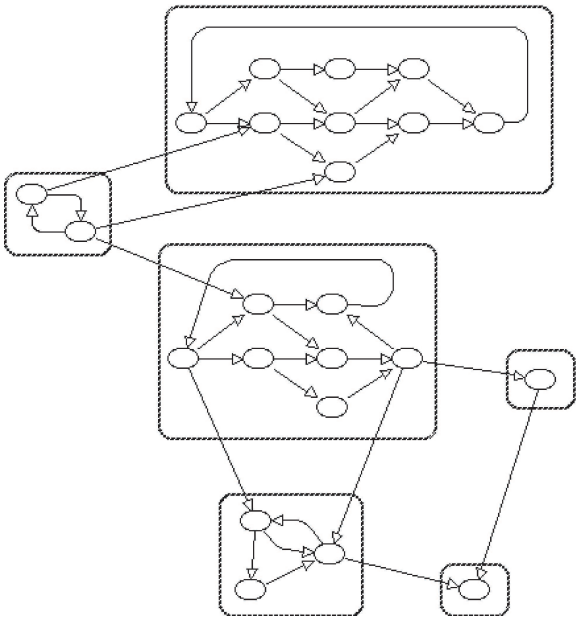


Abbildung 5.6: Der Graph von Abb. 5.5 mit SCCs.

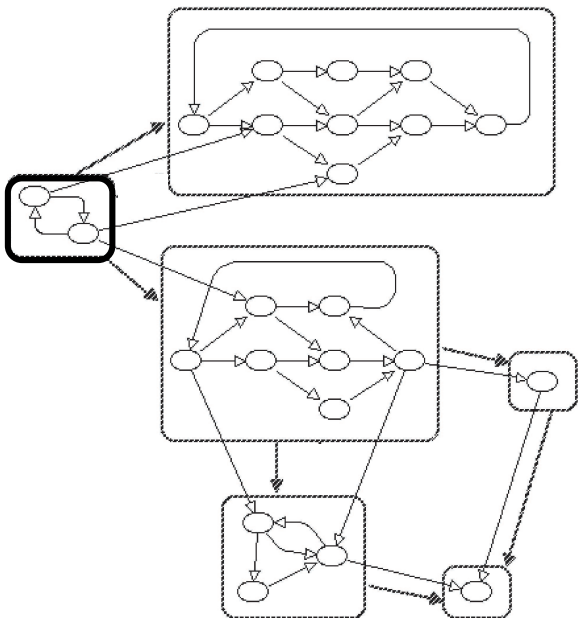


Abbildung 5.7: Der reduzierte Graph des Graphen von Abb. 5.5.

terminal SCCs contain at least one marking satisfying the property Π, and therefore for all reachable marking there exists at least a successor marking satisfying the property Π. If the algorithm fails, there exists at least one terminal SCC that does not contain markings satisfying the property Π, and therefore the reachable markings belonging to this SCC (at least) do not satisfy the liveness invariance property.

Anmerkung: It is possible to design more specific (efficient) decision procedures for the analysis of a property if we know, a priori, some characteristics of the property to be verified or we know some other properties of the net system to be analysed. For the first case we can consider as an example the reversibility property. It is easy to see that if a net system is reversible then all terminal SCCs must contain the initial marking, i.e. the reachability graph must be strongly connected. For the second case, for example, we may know that the net system is reversible; then liveness of a transition t can be decided checking the existence of an edge in the reachability graph labelled t (since the reachability graph is SC and therefore always is possible to reach the marking from which t can be fired).

Beispiel 5.6 Analysis of liveness invariance properties Let us consider the net system in Figure 5.3.b for which the reachability graph is depicted in Figure 8.2. The execution of Algorithm 5.3 to verify the liveness property of this net system ($\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S}) \,.\, \forall t \in T \,.\, \left[\exists \mathbf{m}^t \in \mathrm{RS}(\langle \mathcal{N}, \mathbf{m} \rangle) \,.\, \mathbf{m}^t \geq \mathbf{Pre}[P,t]\right]$) requires the computation of the strongly connected components of the RG($\mathcal{S}$) (step 1). In this case, there are three SCCs depicted in Figure 8.2 and named C_1, C_2 and C_3. The SCCs C_2 and C_3 are the terminal ones. The step 4 of the

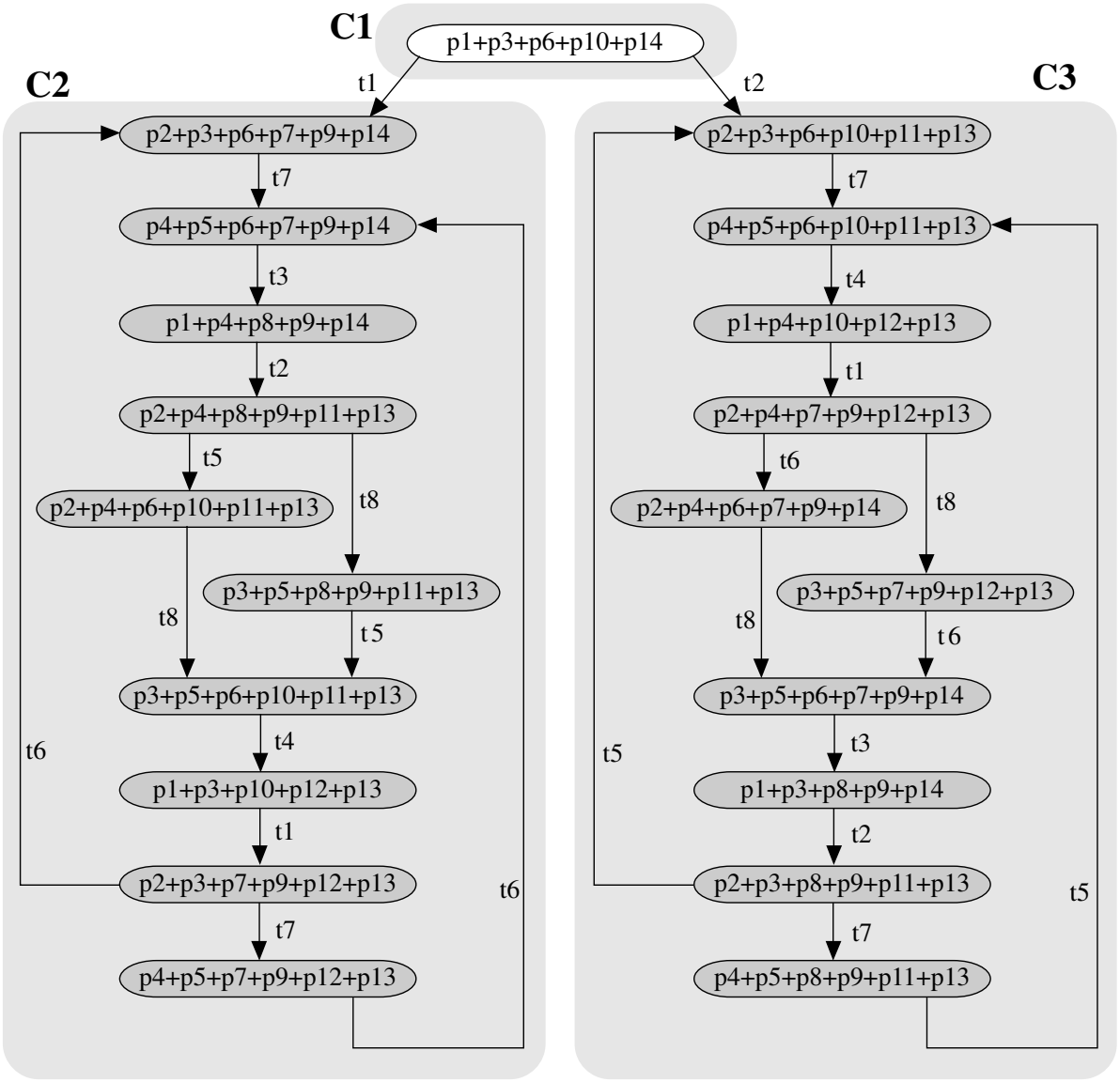


Abbildung 5.8: Reachability graph of the net system in Figure 5.3.b

algorithm will verify that each one of these two SCCs contains for each transition t a marking $\mathbf{m}^t$ satisfying $\mathbf{m}^t \geq \mathbf{Pre}[P,t]$ (equivalently, contains edges labelled with all transitions of the net). The reader can observe by inspection of the figure that all transitions appear in some edge of C_2 and C_3, therefore the answer of the algorithm will be TRUE.

The execution of the algorithm 5.3 to verify that the marking $\mathbf{m}_H = p_2 + p_3 + p_6 + p_7 + p_9 + p_{14}$ is a home state ($\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})\,.\,\exists \mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m})\,.$ such that $\mathbf{m}' = \mathbf{m}_H$) gives as result FALSE, because the terminal SCC C_2 contains the marking $\mathbf{m}_H$, but the terminal SCC C_3 does not. Therefore, step 3.1 returns FALSE.

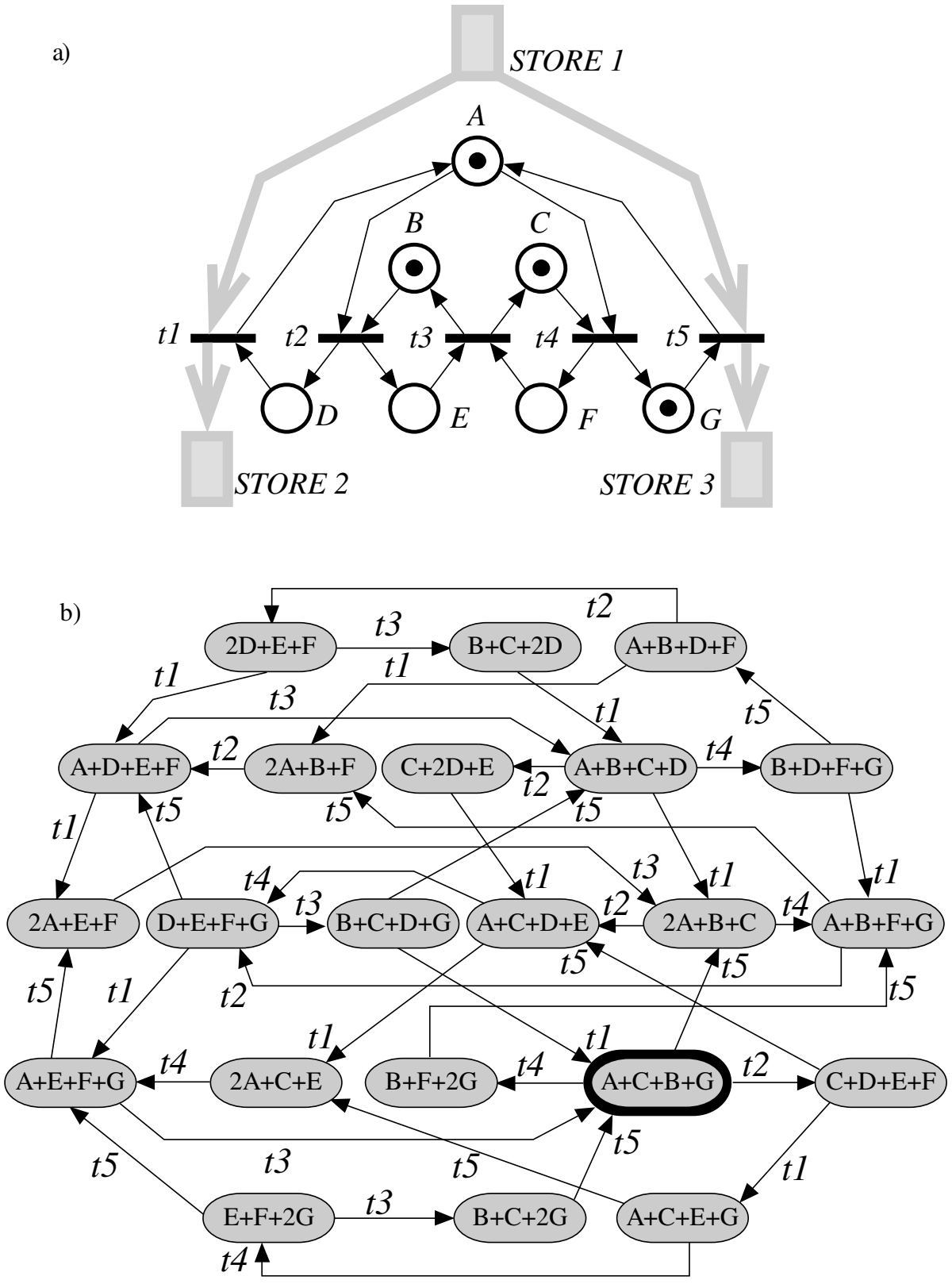
From a practical point of view, it is commonly accepted today that systems are too complex to be verified by hand. As a result, analysis increasingly is becoming synonymous with *computer-aided verification*. Computer-aided verification means using a computer, for increased speed and reliability, to perform the analysis steps. For instance, the following example considers the analysis of a property in the group of the so called *synchronic properties* [**?**], pointing out that an analysis by hand can be very hard.

Beispiel 5.7 Figure 5.9 shows a very simple net system: Parts are sent from *STORE 1* to *STORE 2* and *STORE 3*. The subnet generated by places $\{B, C, E, F\}$ imposes some restrictions on the way parts are distributed to the destination stores (i.e. partially schedule the distribution). The reachability graph is, even if it has been "structured" for more clear presentation, difficult to understand and manage. The reader can try to check on the reachability graph (!) that the imposed distribution strategy is: parts are sent in a 1:1 relation to the destination stores, but allowing sometimes until four consecutive sendings to a given store (i.e. locally adjusting the possible demand, but maintaining the overall fair distribution).

Summarising, analysis techniques based on the reachability graph are only theoretically possible for bounded systems. They are very simple from a conceptual point of view. The problem that makes this approach not practical in many cases is its computational complexity: *the state space explosion problem*.

On the other hand, it must be pointed out that the reachability/coverability graphs are computed for a given initial marking. This means that a parametric analysis of a net system (needed in earlier phases of the system design) where the initial marking of some places (e.g. representing the number of resources in the system) is a parameter, is not possible since for each value of the parameter a (possibly completely different) new reachability graph must be computed. Moreover, the reachability graph presents some difficulties in order to analyse properties where the distinction between conflict and concurrency plays a fundamental role. This is because the reachability graph gives a sequentialised view of the behaviour of the net system.

Although these analysis techniques present the drawbacks above mentioned, for bounded net systems they are the more general ones and, in some cases, the only way to verify a given property.



PNL/WS 2002/03

Abbildung 5.9: Parts of *STORE 1* are sent to *STORE 2* and *STORE 3* according to the strategy defined by the subnet generated by $\{B, C, E, F\}$: (a) the net system; (b) the reachability graph.

5.4 Komplexität nebenläufiger Systeme [1]

5.4.1 Überdeckungsgraph

An einfachen Beispielen läßt sich die Frage nach der Endlichkeit der Erreichbarkeitsmenge oft schnell lösen. Sind die Netze jedoch größer, so ist dies natürlich kein geeignetes Verfahren mehr. Sollen Petrinetze für (Software-) Systementwurf nutzbar gemacht werden, so muß diese Frage mit geeigneten Programmen (tools) lösbar sein. Kurz, es stellt sich die Frage nach der Entscheidbarkeit des folgenden Problems:

Gegeben: Ein endliches S/T-Netz $N := (S, T, W, K, \mathbf{m}_0)$.

Frage: Ist die Erreichbarkeitsmenge $R(N)$ endlich?

Es ist natürlich sofort klar, dass $R(N)$ stets dann endlich ist, wenn für jede Stelle $s \in S$ eine endliche Kapazität $K(s) \in \mathcal{N}$ gefordert ist. Dann nämlich gibt es höchstens $\prod_{s \in S} (K(s) + 1)$ viele verschiedene Markierungen, von denen in der Regel nicht einmal alle von $\mathbf{m}_0$ aus erreichbar sind. Es gibt zwar viele Gründe warum beschränkte Netze in den Anwendungen zu bevorzugen, ja sogar unabdingbar sind, jedoch sind beim Entwurf vielleicht nicht alle Kapazitätsbeschränkungen berücksichtigt worden und daher die algorithmische Lösung für das oben angesprochene *Endlichkeitsproblem* für Erreichbarkeitsmengen von Petrinetzen nötig. Die Lösung gelingt mit der Konstruktion eines sogenannten *Überdeckbarkeitsgraphen*, oder knapper *Überdeckungsgraphen*.

Schon zur Bezeichnung einer nicht vorhandenen endlichen Kapazitätsbeschränkung haben wir das Symbol ω (Omega) benutzt. Mit diesem werden wir nun im Folgenden die Existenz von „beliebig vielen“ Marken auf einer Stelle bezeichnen, und daher die natürlichen Zahlen mit diesem formalen Symbol erweitern, sowie Pseudomarkierungen verwenden, deren Komponenten dieses Omega enthalten:

Definition 5.8 *Es sei* $\mathcal{N}_\omega := \mathcal{N} \cup \{\omega\}$ *zusammen mit folgenden Rechenregeln:*

$$\forall n \in \mathcal{N} : \omega > n;$$

$$\forall n \in \mathcal{N}_\omega : \omega + n = \omega - n := \omega;$$

$$\forall n \in \mathcal{N} \backslash \{0\} : n \cdot \omega = \omega \cdot n := \omega; 0 \cdot \omega = \omega \cdot 0 := 0.$$

Ein Vektor $\mathbf{m} \in \mathcal{N}_\omega^S$ *wird* Pseudomarkierung *genannt, wenn in ihm das Symbol* ω *vorkommt, und für diese wird die Schaltregel formal übernommen. Eine Pseudomarkierung* $\mathbf{m}$ *entspricht einer gewöhnlichen (Teil-) Markierung auf den Stellen* s *mit* $\mathbf{m}(s) \neq \omega$*, wobei die mit* ω *besetzten Komponenten beliebig sind und unberücksichtigt bleiben. Man kann eine Pseudomarkierung auch als Beschreibung derjenigen Hyper-Ebene im* $\mathcal{N}^S$ *verstehen, die senkrecht „auf“ der Hyper-Fläche „steht“, die von den Komponenten ungleich* ω *aufgespannt wird.* .

[1] entnommen dem Skript *Theoretische Grundlagen der Programmierung* von M. Jantzen

Für Vektoren $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^r$ *seien die Operatoren* $+, -, =, \leq$ *jeweils komponentenweise erklärt, d.h.,* $\mathbf{m}_1 \leq \mathbf{m}_2$*, falls* $\forall s \in S : \mathbf{m}_1(s) \leq \mathbf{m}_2(s)$*. Lediglich* $\mathbf{m}_1 < \mathbf{m}_2$ *steht für* $(\mathbf{m}_1 \leq \mathbf{m}_2$ *und* $\mathbf{m}_1 \neq \mathbf{m}_2)$*.*

Konstruktion 5.9 (Überdeckungsgraph) Sei $N := (S, T, W, \mathbf{m}_0)$ ein S/T-Netz ohne Kapazitätsbeschränkungen K.

Wir konstruieren einen gerichteten, kantenbeschrifteten Graphen $G(N) := (V, E)$ mit $V \subseteq \mathcal{N}_\omega^S$ und Kanten $E \subseteq V \times T \times V$ wie folgt:

Wir schreiben dazu $\mathbf{m}_1 \xrightarrow{*}_{w} \mathbf{m}_2$, wenn es in $G(N)$ einen Pfad vom Knoten $\mathbf{m}_1$ zum Knoten $\mathbf{m}_2$ gibt, der die in der Kantenfolge zusammengefügte Beschriftung $w \in T^*$ hat. Wir schreiben $\mathbf{m}_1 \xrightarrow{*} \mathbf{m}_2$, wenn es irgendeinen Pfad gibt, dessen Beschriftung uns nicht wichtig ist.

Algorithmus für G(N):

```
BEGIN
    V := {m_0}; E := ∅;
    LOOP Wähle m ∈ V und t ∈ T so, dass das Paar (m, t) noch nie vorher
        ausgewählt wurde und t in m aktiviert ist, d.h. m[t> gilt.
        IF kein solches Paar vorhanden THEN
            EXIT LOOP G(N) := (V, E); STOP .
        END
        m_1 := m + Δ(t);
        IF m_1 ∈ V THEN
            (1) E := E ∪ {(m, t, m_1)}
        ELSE
            Sei P(m_1) := {m_2 ∈ V | m_2 ≤ m_1 und m_2 →* m} und m⃗_3
            definiert durch
                m_3(s) := { ω,      ∃m_2 ∈ P(m⃗_1) : m_2(s) < m_1(s)
                          { m_1(s), sonst.
            (2) V := V ∪ {m_3}; E := E ∪ {(m, t, m_3)}
        END
    markiere m als behandelt
    END
END
```

Betrachten wir das S/T-Netz N aus Abbildung 5.10, so ergibt sich der Überdeckungsgraph aus Abbildung 5.11:

Die Bedeutung des Überdeckungsgraphen ergibt sich aus folgenden Eigenschaften:

Satz 5.10 *Sei* $N := (S, T, W, \mathbf{m}_0)$ *ein S/T-Netz und* $G(N) = (V, E)$ *ein Überdeckungsgraph zu* N*, dann ist eine Stelle* $s \in S$ *genau dann* beschränkt*, wenn es keinen Knoten* $\mathbf{m} \in V$ *mit* $\mathbf{m}(s) = \omega$ *gibt.*

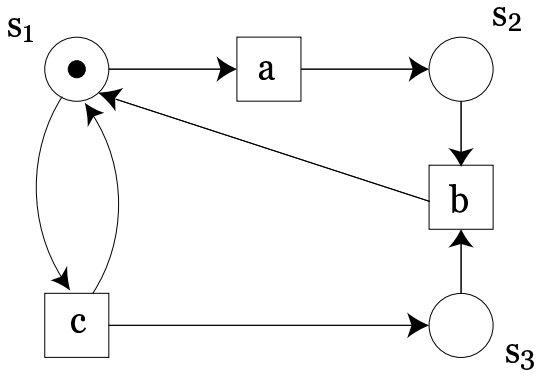


Abbildung 5.10: Netz N

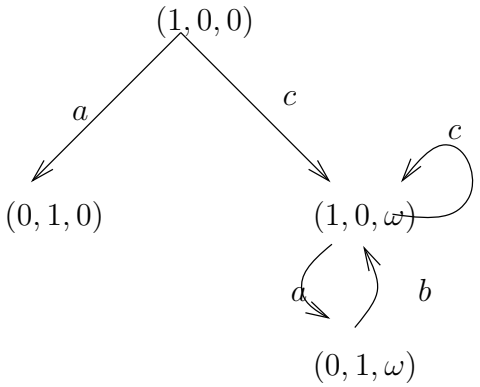


Abbildung 5.11: Überdeckungsgraph zu Netz N

Gilt $\mathbf{m}_0[w> \mathbf{m}$ *im Netz* N *für ein Wort* $w \in T^*$*, so gibt es in* $G(N)$ *einen Knoten* $\mathbf{m}_1$ *mit* $\mathbf{m}_1 \geq \mathbf{m}$ *und* $\mathbf{m}_0 \stackrel{*}{\underset{w}{\rightarrow}} \mathbf{m}_1$.

Aus dieser letzten Eigenschaft leitet sich der Name „Überdeckungsgraph“ für $G(N)$ ab, denn zu jedem in N erreichbaren Knoten $\mathbf{m} \in R(N)$ gibt es einen, i.A. anderen, in $G(N)$, der diesen 'überdeckt'. Um diesen Sachverhalt zu beweisen, zeigen wir zunächst die Termination dieses Algorithmus'.

Satz 5.11 *Der Algorithmus zur Konstruktion von* $G(N)$ *terminiert.*

Beweis:
Angenommen, der Algorithmus terminiere *nicht*, dann ist $|V|$ unendlich.

Begründung: Wäre $|V|$ endlich, so auch die Menge $V \times T$ und alle Paare $(m, t) \in (V \times T)$ wären einmal nach Eintritt in die `LOOP`–Schleife ausgewählt worden und das STOP wird erreicht.

Nun sind nach Konstruktion alle Knoten $\mathbf{m} \in V$ von $\mathbf{m}_0$ aus erreichbar. Betrachten wir den spannenden Baum $B(N)$ von $G(N)$ der dadurch entsteht, dass diejenigen Kanten weggelassen werden, die in den Schritten **(1)** oder **(2)** zu einem schon existierenden Knoten gezeichnet werden. $B(N)$ entsteht also aus $G(N)$ durch Streichen gewisser Kanten. Da $B(N)$ lokal endlich ist (jeden Knoten verlassen maximal $|T|$ viele Kanten) aber selbst unendlich viele Knoten besitzt, gibt es in $B(N)$ nach dem Satz von König (1936) einen unendlichen Pfad $\mathbf{m}_0 \to \mathbf{m}_1 \to \mathbf{m}_2 \to \ldots \quad \ldots \to \mathbf{m}_i \to \mathbf{m}_{i+1} \to \ldots$ in dem kein Knoten $\mathbf{m}_i$ zweimal vor kommt. In jeder unendlichen Folge von paarweise verschiedenen Vektoren $\mathbf{m}_i \in \mathcal{N}^S$ gibt es, nach einem Satz von Dickson (1926), eine unendliche Teilfolge $\mathbf{m}_{i_1} \stackrel{+}{\to} \mathbf{m}_{i_2} \stackrel{+}{\to} \mathbf{m}_{i_3} \stackrel{+}{\to} \ldots \stackrel{+}{\to} \mathbf{m}_{i_j} \stackrel{+}{\to} \mathbf{m}_{i_{j+1}} \stackrel{+}{\to} \ldots$ mit der Eigenschaft $\mathbf{m}_{i_j} < \mathbf{m}_{i_{j+1}}$. Nach Konstruktion muß $\mathbf{m}_{i_{j+1}}$ eine ω-Komponente mehr als $\mathbf{m}_{i_j}$ haben, denn $\mathbf{m}_{i_j} \stackrel{*}{B(N)} \mathbf{m}_{i_{j+1}}$ impliziert $\mathbf{m}_{i_j} \stackrel{*}{G(N)} \mathbf{m}_{i_{j+1}}$ und wenn $\mathbf{m}_{i_j} < \mathbf{m}_{i_{j+1}}$ ist, wird $\mathbf{m}_{i_{j+1}}$ mindestens eine ω-Komponente mehr enthalten als $\mathbf{m}_{i_j}$. Dies führt zu dem gewünschten Widerspruch, denn nur endlich viele ω-Komponenten sind überhaupt möglich. Also terminiert die Konstruktion von $G(N)$ nach dem Verfahren von Konstruktion 5.9. □

Aufgabe 5.12 Konstruiere einen Überdeckungsgraphen $G(N)$ für das S/T-Netz N_6, wenn dabei als Anfangsmarkierung nur die Stelle s_3 eine Marke trägt:

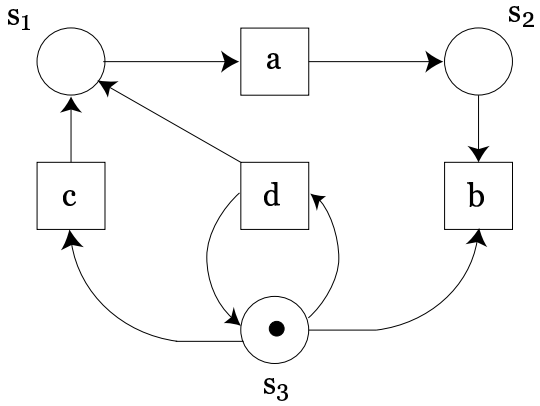


Abbildung 5.12: Netz N_6

Satz 5.13 *Für ein S/T-Netz N ist $R(N)$ genau dann endlich, wenn kein Knoten von $G(N)$ eine ω-Komponente besitzt.*

Beweis:
Kommt in keiner Komponente von Knoten aus V in $G(N) = (V, E)$ die Bezeichnung ω vor, so ist $G(N)$ identisch mit dem Erreichbarkeitsgraph von N, und, wegen der Termination des Verfahrens, ist auch die Menge $V = R(N)$ endlich.

Sei $\mathbf{m}_2 \in V$ mit $\mathbf{m}_2(s) = \omega$ für ein $s \in S$, so gibt es einen Pfad $\mathbf{m}_0 \xrightarrow{*}_{\alpha} \mathbf{m}_{3,s} \xrightarrow{*}_{\beta} \mathbf{m}_s \xrightarrow{}_{t} \mathbf{m}_{1,s} \xrightarrow{*}_{\gamma} \mathbf{m}_{2,s}$ in $G(N)$ und $\mathbf{m}_{1,s} \in \mathcal{N}_\omega^S$ mit $\mathbf{m}_s[t> \mathbf{m}_{1,s}$ und $\mathbf{m}_{3,s} < \mathbf{m}_{1,s}$, genauer: $\mathbf{m}_{3,s}(s) < \mathbf{m}_{1,s}(s)$. Da das Wort w auf dem Pfad von $\mathbf{m}_0$ über $\mathbf{m}_{3,s}$ nach $\mathbf{m}_s$ eine Schaltfolge im Netz N ist, gibt es also Wörter $\alpha, \beta, \gamma \in T^*$ und erreichbare Markierungen $\mathbf{m}_3^\bullet, \mathbf{m}_1^\bullet, \mathbf{m}_4^\bullet, \ldots$ mit $\mathbf{m}_0[\alpha> \mathbf{m}_3^\bullet[\beta t> \mathbf{m}_1^\bullet[\gamma> \mathbf{m}_4^\bullet \ldots$.

Da $\mathbf{m}_3^\bullet(s) < \mathbf{m}_1^\bullet(s) < \mathbf{m}_4^\bullet(s) < \ldots$ gilt, ist die Stelle $s \in S$ also in N nicht beschränkt und $R(N)$ ist keine endliche Menge. □

5.4.2 Komplexität

Für einen Systementwurf mit einem Petri-Netz stellt die Untersuchung der Endlichkeit der Erreichbarkeitsmenge $\mathbf{R}(N)$ schon ein recht nützliches, wenn auch nicht immer ausreichendes Hilfsmittel dar.

Es gibt kleine Petri-Netze mit unangenehm großer Erreichbarkeitsmenge, wie das folgende Ergebnis zeigt, das wir hier nur zitieren.

Satz 5.14 *Es gibt eine unendliche Folge von beschränkten Petri-Netzen $N_1, N_2, N_3, \ldots$, deren Größe* ($|S|+|T|+|F|$) linear *wächst, jedoch die Größe der Erreichbarkeitsmengen $|\mathbf{R}(N_1)|, |\mathbf{R}(N_2)|, |\mathbf{R}(N_3)|, \ldots$ wächst schneller als jede primitiv rekursive Funktion.*

Eine ebenfalls nicht primitiv rekursiv berechenbare Variante der *Ackermann-Funktion* ist folgende:

$$A(0,n) := 2n+1, A(m+1,0) := 1, A(m+1,n+1) := A(m, A(m+1,n))$$

Die Netze N_i berechnen gerade immer Markenzahlen auf einer festgelegten Stelle, die bis an den Wert von $A(i,2)$ heranreichen.

Aus diesem Ergebnis folgt, dass das eben angegebene Entscheidungsverfahren (Konstruktion 5.9) für die Endlichkeit der Erreichbarkeitsmenge eines S/T-Netzes mit dem Überdeckungsgraphen leider nicht primitiv-rekursiv ist! Daher entsteht die Frage, ob es vielleicht bessere Verfahren gibt, mit denen dieses Problem entschieden werden kann?

Folgende Ergebnisse sind bekannt:

Satz 5.15 (Rackoff (1978)) *Das Beschränktheitsproblem ist mit $O(2^{c \cdot n \cdot log(n)})$ Platzbedarf entscheidbar.*

Hierbei bezeichnet n die Größe des S/T-Netzes, d.h. im Wesentlichen die Länge der Darstellung der Matrix W. Eine untere Schranke wurde schon zuvor von Lipton gefunden:

Satz 5.16 (Lipton (1976)) *Das Beschränktheitsproblem benötigt für seine Entscheidung mindestens* $O(2^{c\cdot\sqrt{n}})$ *Platzbedarf.*

Ein besseres Ergebnis ist von Rosier und Yen bewiesen worden:

Satz 5.17 (Rosier&Yen (1986)) *Das Beschränktheitsproblem kann für S/T-Netze* $(S,T,W,\mathbf{m}_0)$ *mit* $O\big(2^{c\cdot|S|\cdot log(|S|)}\cdot(log(|T|)+max_{x,y\in S\cup T}(|W(x,y)|)\big)$ *Platzbedarf entschieden werden.*
Für festes $|S|$ *ist das Problem PSPACE-vollständig.*

Erst für spezielle Teilklassen der Petrinetze kann man erträgliche Komplexität für dieses Entscheidungsproblem erwarten.

Satz 5.18 (Rosier et. al. (1987)) *Das Beschränktheitsproblem kann für konflikt-freie Petrinetze mit* $O(n^{1,5})$ *Platzbedarf entschieden werden.*

Dies Beispiel zeigt, dass eine alleinige Beschränkung der Kapazität der Stellen in der Regel auch nicht ausreicht, um zu stets befriedigenden Lösungen zu kommen. Das Netz N_7 hat stets eine endliche Erreichbarkeitsmenge, diese ist jedoch so groß, dass viele Computerprogramme die vorgeben Petrinetze zu analysieren, damit auf ihre Grenzen hin getestet werden können.

Das S/T-Netz N_7 hat 6 Stellen, 6 Transitionen, $30+k$ Kanten und $m+4$ Marken in der eingezeichneten Anfangsmarkierung $\mathbf{m}_0$. Die Größen $k\geq 2$ und $m\geq 0$ sind festlegbar und die maximale Zahl von möglichen Marken bestimmt sich durch die Funktion

$$\max(m,k) := k\cdot f_k(m)+2,$$

wobei f_k wie folgt definiert ist:

$$f_k(m) := \texttt{ IF } m=0 \texttt{ THEN } k \texttt{ ELSE } f_k(m-1)\cdot k^{f_k(m-1)} \texttt{ FI}.$$

Es gilt: $\max(2,2)=4098$, aber schon $\max(3,2)=1048576^{103}+2$ und $\max(2,3)=3^{86}+2$.

Weil beschränkte Petrinetze in aller Allgemeinheit keine Gewähr für praktische Brauchbarkeit geben, ist man bemüht, beschränkte Petrinetze auf andere modulare oder syntaktisch vorgeschriebene Weisen zu definieren. Zum Beispiel werden solche S/T-Netze, bei denen jede Stelle genau eine Eingangs- und eine Ausgangstransition besitzt *markierte Graphen* genannt und oft verwendet, weil sie eine übersichtliche Struktur besitzen und einfach(er) zu analysieren sind.

Definition 5.19 *Ein S/T-Netz* (S,T,W) *heißt* konservativ*, wenn es keine multiplen Kanten besitzt (d.h. stets* $W(x,y)\leq 1$ *gilt) und eine Funktion* $f: S\to\mathcal{N}\setminus\{0\}$ *existiert, für die gilt:* $\forall t\in T: \sum_{s\in{}^\bullet t} f(s)=\sum_{s\in t^\bullet} f(s)$.

Netze, bei denen $f(s)=1$ *für alle* $s\in S$ *ist, heißen* 1-konservativ *und es gilt dann automatisch:* $\forall t\in T: |t^\bullet|=|{}^\bullet t|$.

Ein S/T-Netz (S,T,W) *heißt* markierter Graph*, wenn gilt:*

$$\forall s\in S: |{}^\bullet s|=|s^\bullet|=1.$$

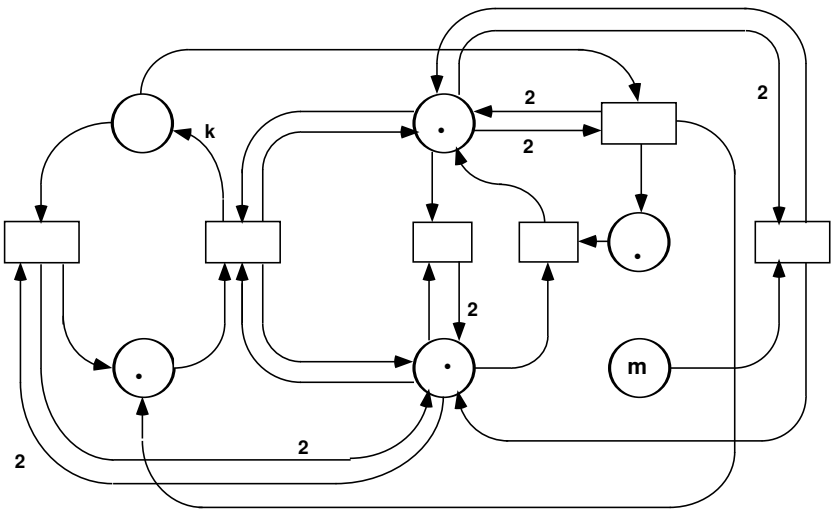


Abbildung 5.13: Netz N_7

Satz 5.20 *Sei $N := (S, T, W, \mathbf{m}_0)$ konservatives S/T-Netz, dann gilt:*
aus $f(\mathbf{m}) := \sum_{s \in S} \mathbf{m}(s) \cdot f(s)$ folgt für jede erreichbare Markierung $\mathbf{m} \in R(N)$ stets $f(\mathbf{m}_0) = f(\mathbf{m})$.

Aufgabe 5.21 Beweise den oben angeführten Satz 5.20 und schließe daraus auf die Endlichkeit der Menge $R(N)$.

Bei konservativen S/T-Netzen ist $R(N)$, wie in Aufgabe 10.5 gezeigt werden sollte, immer eine endliche Menge von Markierungen und daher ist es prinzipiell leicht zu prüfen, ob eine bestimmte Markierung von der Anfangsmarkierung aus erreicht werden kann. Dass dies leider praktisch nicht immer so leicht ist, haben wir an Satz 5.14 gesehen. Andererseits sagte uns auch bisher niemand, dass wir bei endlichen Petrinetzen zur Lösung dieser Frage immer gleich die gesamte Erreichbarkeitsmenge $R(N)$ konstruieren müssen!

Definition 5.22 (Erreichbarkeitsproblem) *Als* Erreichbarkeitsproblem *für Petrinetze bezeichnet man folgendes Problem:*

Gegeben: *Ein S/T-Netz $N := (S, T, W, K, \mathbf{m}_0)$ und eine Markierung $\mathbf{m} \in \mathcal{N}^S$.*

Frage: *Gilt $\mathbf{m} \in R(N)$?*

Leider ist selbst für 1-konservative S/T-Netze das Erreichbarkeitsproblem nicht in vertretbarem Aufwand automatisch zu lösen.

Satz 5.23 *Das Erreichbarkeitsproblem für 1-konservative S/T-Netze ist $\mathcal{PSPACE}$-vollständig.*

Nur in wenigen Teilklassen der allgemeinen S/T-Netze findet man $\mathcal{NP}$-Vollständigkeit und in noch wenigeren sogar deterministische Verfahren, die die Erreichbarkeitsfrage in Polynomzeit lösen. Die in 5.19 definierten *markierten Graphen* gehören zu letzterer Klasse!

Die folgende Bemerkung ist wichtig: Für den allgemeinen Fall ist zwar bekannt, dass das Erreichbarkeitsproblem für S/T- Netze entscheidbar ist, jedoch wird zur Lösung mindestens exponentiell viel Platz (und damit erst recht Zeit) gebraucht.

Die einzigen bekannten Algorithmen für dieses Problem konstruieren eine abgewandelte Form des Überdeckungsgraphen und haben daher eine Laufzeit, die auch bei Fällen endlicher Erreichbarkeitsmengen **keine primitiv rekursive Funktion** der Eingabegröße mehr ist!

Satz 5.24 *Das Erreichbarkeitsproblem für endliche S/T-Netze ist entscheidbar, benötigt jedoch mindestens exponentiell viel Platz.*

Die untere Schranke $NSpace(2^{O(n)})$ wurde von Lipton 1976 bewiesen. Einen guten Überblick über weitere Komplexitätsresultate zu Algorithmen und Problemen bei Petrinetzen finden die Leser(innen) bei Esparza und Nielsen 1994.

In vielen Fällen jedoch sind ganz andere Fragen wichtig, die wir zunächst an dem wohlbekannten Beispiel des Leser/Schreiber-Problems diskutieren wollen.

5.5 Kripke-Strukturen

Kripke-Strukturen sind Zustandsgraphen, ergänzt um logische Fromeln.

Definition 5.25 *Eine* Kripke-Struktur *ist ein Tupel* $M := (S, S_0, R, L)$ *mit:*

1. S *endliche Zustandsmenge,*
2. $S_0 \subseteq S$ *Menge von Anfangszuständen,*
3. $R \subseteq S \times S$ *links totale (Transitions-) Relation,*
4. $L : S \to 2^{AP}$ *Abbildung, die jedem Zustand* s *eine Menge* $L(s) \subseteq AP$ *von aussagenlogischen atomaren Formeln zuordnet (die in diesem Zustand gelten).*

Beispiel: $L(3) = \{\neg Start, Close, \neg Heat, \neg Error\}$ *in der Kripke-Struktur von Abb. 5.17.*

Die Kripke-Struktur kann mittels Prädikaten erster Ordnung representiert werden.

Zustand: $s : V \to D$ ist eine Abbildung von der Menge der Variablen in eine Wertemenge. Z.B. für die Variablen Menge $V = \{v_1, v_2, v_3\}$, wird der Zustand $s =< v_1 \leftarrow 2, v_2 \leftarrow 3, v_3 \leftarrow 5 >$ durch die zugehörige Formel $(v_1 = 2) \wedge (v_2 = 3) \wedge (v_3 = 5)$ representiert. $\mathcal{S}_0$ bezeichnet die Formel für den Anfangszustand.

Transition: Beziehung zwischen Werten der Variablen vor der Transition (Menge V) und Werten der Variablen nach der Transition (Menge V'). Wenn R eine Transitionsrelation ist, so bezeichnet $\mathcal{R}(V, V')$ die entsprechende Formel.

Definition 5.26 *Eine* Kripke-Struktur in logischer Darstellung *ist ein Tupel* $M := (S, S_0, R, L)$ *mit:*

1. S *Menge der Belegungen,*
2. $S_0 \subseteq S$ *Menge von Zuständen, die Anfangsbedingung* $\mathcal{S}_0$ *erfüllen,*
3. $\forall s, s' \in S : R(s, s') \leftrightarrow \mathcal{R}(V, V')$ *mit Belegung* s *für* V *und* s' *für* V'*,*
4. $L(s)$ *Menge der Formel, die bei Belegung* s *gelten.* $(v \in L(s)$ *gdw.* $s(v) = wahr$*,* $v \notin L(s)$ *gdw.* $s(v) = falsch)$

Ein *Pfad* oder *Rechnung* aus $s \in S$ ist eine Folge $\pi = s_0, s_1, s_2, \ldots$ mit $s_0 = s$ und $\forall i \geq$: $R(s_i, s_{i+1})$

Beispiel 5.27 Ein System mit Kripke-Struktur

Variablenmenge: $V = \{x, y\}$,

Wertemenge: $D = \{0, 1\}$,

System: $x := (x + y) \bmod 2$,

Anfangszustand: $x = 1, y = 1$,

Das System kann mit zwei Prädikatenformeln beschrieben werden:

$$\mathcal{S}_0(x, y) \equiv x = 1 \wedge y = 1$$

$$\mathcal{R}(x, y, x', y') \equiv x' = (x + y)\, mod\, 2 \wedge y' = y$$

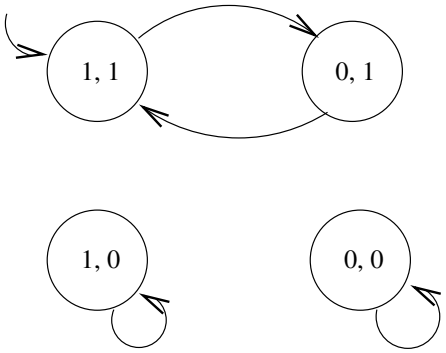
Daraus die Kripke-Struktur: $M = (S, S_0, R, L)$

$S = D \times D$,

$S_0 = \{(1, 1)\}$,

$R = \{((1, 1), (0, 1)), \quad ((0, 1), (1, 1)), \quad ((1, 0), (1, 0)), \quad ((0, 0), (0, 0))\}$,

$L((1, 1)) = \{x = 1, y = 1\}, \ldots$



Pfad vom Anfangszustand (1, 1), (0, 1), (1, 1), (0, 1), ...

Sequentielle Programme als Kripke-Strukturen

Zur Kennzeichnung von Zuständen erhalten die Programme Zeilennummern.

Für Zeilennummern gibt es die Variable pc („Befehlszähler“). Mit $pc = \perp$ ist gemeint, dass das Programm nicht aktiv ist.

Prädikat $same(Y) \equiv \forall y \in Y : (y' = y) \qquad (Y \subseteq V)$

Prädikat $pre(V)$ beschreibt die Anfangsbelegung der Variablen V.

Anfangszustand: $\mathcal{S}_0 \equiv pre(V) \wedge pc = m$, mit m als Startzeilenummer.

Die Prozedur $\mathcal{C}(l, P, l')$ lifert für eine Programmbeschreibung P die Representation $\mathcal{R}$ der Transitionen des Programms als Disjunktion der predikatenlogischen Formeln.

l, l' Zeilennummer jeweils vor und nach der Anweisung

pc, pc' Befehlszähler - Variable

Zuweisung:
$\mathcal{C}(l, v \leftarrow e, l') \equiv pc = l \wedge pc' = l' \wedge v' = e \wedge same(V \backslash \{v\})$

Skip:
$\mathcal{C}(l, skip, l') \equiv pc = l \wedge pc' = l' \wedge same(V)$

Hintereinanderausfürung:
$\mathcal{C}(l,\ P_1;\quad l'' : P_2,\ l') \equiv \mathcal{C}(l, P_1, l'') \vee \mathcal{C}(l'', P_2, l')$

Bedingte Anweisung:
$\mathcal{C}(l,\ \textbf{if}\ b\ \textbf{then}\ l_1 : P_1\ \textbf{else}\ l_2 : P_2\ \textbf{endif},\ l') \equiv$
$(pc = l \wedge pc' = l_1 \wedge b \wedge same(V)) \vee$
$(pc = l \wedge pc' = l_2 \wedge \neg b \wedge same(V)) \vee$
$\mathcal{C}(l_1, P_1, l') \vee \mathcal{C}(l_2, P_2, l')$

Schleifen-Anweisung:
$\mathcal{C}(l,\ \textbf{while}\ b\ \textbf{do}\ l_1 : P_1\ \textbf{endwhile},\ l') \equiv$
$(pc = l \wedge pc' = l_1 \wedge b \wedge same(V)) \vee$
$(pc = l \wedge pc' = l' \wedge \neg b \wedge same(V)) \vee$
$\mathcal{C}(l_1, P_1, l)$

Nebenläufige Programme als Kripke-Strukturen

Programmbeschreibung:

$$P = \textbf{cobegin } P_1 \| P_2 \| \ldots \| P_n \textbf{ coend}$$

Mit Zeilennummern:

$$P^{\mathcal{L}} = \textbf{cobegin } l_1 : P_1^{\mathcal{L}}\ l_1' \| \ldots \| l_n : P_n^{\mathcal{L}}\ l_n'; \textbf{coend}$$

Daraus die Formeln, wobei $PC = \{pc, pc_i | pc_i\text{-Befehlszäler von } P_i\}$):

$$S_0(V, PC) \equiv pre(V) \wedge pc = m \wedge \bigwedge_{i=1}^{n} (pc_i = \bot)$$

P_i sind also am Anfang nicht aktiv. Damit ergibt sich folgende Representaiton:

$\mathcal{C}(l, P^{\mathcal{L}}, l') \equiv$
$(pc = l \wedge pc_1' = l_1 \wedge \ldots \wedge pc_n' = l_n \wedge pc' = \bot) \vee$ Initialisierung
$(pc = \bot \wedge pc_1 = l_1' \wedge \ldots \wedge pc_n = l_n' \wedge pc' = l' \wedge \bigwedge_{i=1}^{n} (pc_i' = \bot)) \vee$ Termination
$(\bigvee_{i=1}^{n} (\mathcal{C}(l_i, P_i, l_i') \wedge same(V \backslash V_i) \wedge same(PC \backslash \{pc_i\})))$ Transition von P_i

V_i ist die Menge der Variablen die von Programm P_i geändert werden.

wait-Anweisung:

$\mathcal{C}(l, \textbf{wait}(b), l') \equiv$
$(pc_i = l \wedge pc_i' \wedge \neg b \wedge same(V_i))$ "busy waiting"
$\vee\ (pc_i = l \wedge pc_i' = l' \wedge b \wedge same(V_i))$

Beispielprogramm für wechselseitigen Ausschluss

$$P = m : \textbf{cobegin } P_0 \| \ P_1 \textbf{ coend}$$

P_0:: l_0: **while** True **do**
$NC_0 : \textbf{wait}(turn = 0);$
$CR_0 : turn := 1;$
endwhile;
l_0'

P_1:: l_1: **while** True **do**
$NC_1 : \textbf{wait}(turn = 1);$
$CR_1 : turn := 0;$
endwhile;
l_1'

$PC = \{pc, pc_0, pc_1\}$, $\quad pc_i$ nimmt die Werte $\{l_i, l_i', NC_i, CR_i\}$ an
$V = V_0 = V_1 = \{turn\}$

Der Anfangszustand (mit $turn := 0$ oder $turn := 1$):

$$S_0(V, PC) \equiv pc = m \ \wedge \ pc_0 = \bot \ \wedge pc_1 = \bot$$

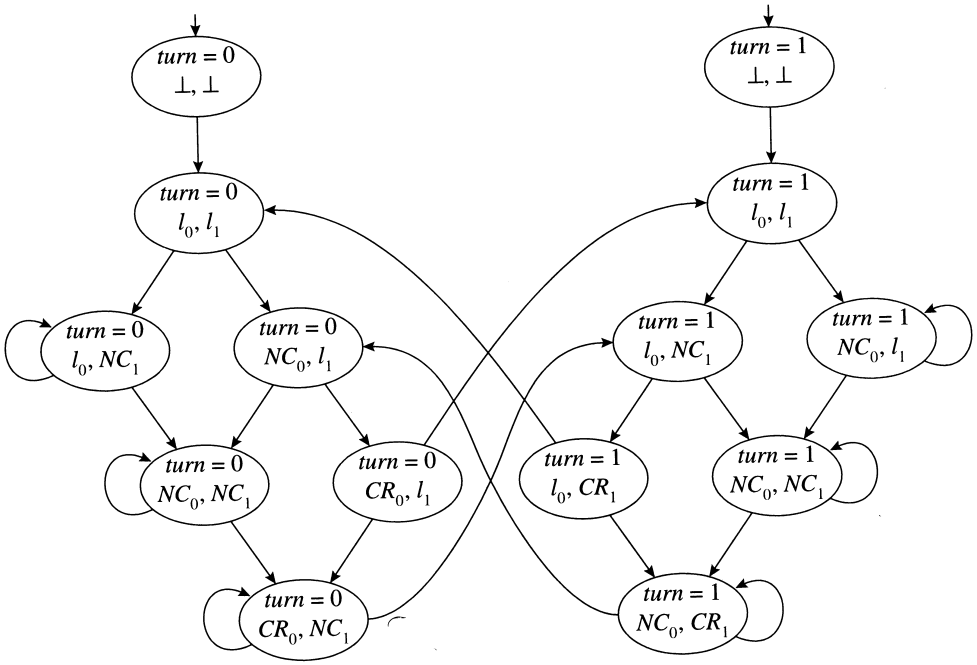
Die Transitionsrelation ist representiert durch $\mathcal{R}(V, PC, V', PC')$ als Disjunktion der folgenden Formeln:

- $pc = m \wedge pc_0' = l_0 \wedge pc_1' = l_1 \wedge pc' = \bot$
- $pc_0 = l_0' \wedge pc_1 = l_1' \wedge pc' = m' \wedge pc_0' = \bot \wedge pc_1' = \bot$
- $\mathcal{C}(l_0, P_0, l_0') \wedge same(V \backslash V_0) \wedge same(PC \backslash \{pc_0\})$ gleich $\mathcal{C}(l_0, P_0, l_0') \wedge same(pc, pc_1)$
- $\mathcal{C}(l_1, P_1, l_1') \wedge same(V \backslash V_1) \wedge same(PC \backslash \{pc_1\})$ gleich $\mathcal{C}(l_1, P_1, l_1') \wedge same(pc, pc_0)$

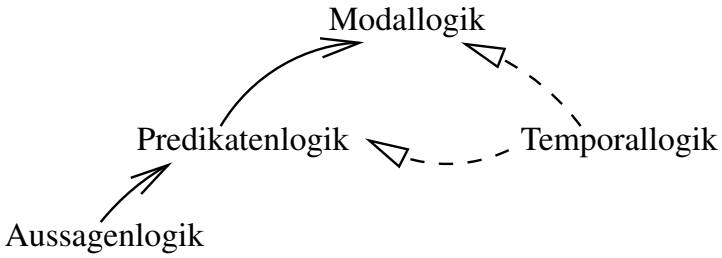
Wobei für $i \in \{0, 1\}$ ist $\mathcal{C}(l_i, P_i, l_i')$ eine Disjunktion von:

- $pc_i = l_i \wedge pc_i' = NC_i \wedge True \wedge same(turn)$
- $pc_i = NC_i \wedge pc_i' = CR_i \wedge turn = i \wedge same(turn)$
- $pc_i = CR_i \wedge pc_i' = l_i \wedge turn' = (i+1) mod 2$
- $pc_i = NC_i \wedge pc_i' = NC_i \wedge turn \neq i \wedge same(turn)$
- $pc_i = l_i \wedge pc_i' = l_i' \wedge False \wedge same(turn)$

- erfüllt: wechselseitiger Ausschluss $\neg(CR_0 \wedge CR_1)$
- nicht erfüllt: Fairness



5.6 Temporale Logik



Beispiel 5.28 Spezifikation eines Aufzuges (Fragment)

I. Jede Anforderung des Aufzugs wird auch erfüllt.

II. Der Aufzug passiert keinen Stockwerk (SW) mit einer nicht erfüllten Anforderung.

Beispiel für physikalisches Bewegungsgesetz: $z(t) = -\frac{1}{2}gt^2$ (freier Fall des Aufzuges)

Die Punkte I und II in Prädikatenlogik:

I. $\forall t, \forall n(app(n,t) \Rightarrow \exists t' > t \,.\, serv(n,t'))$

II.

$$\forall t, \forall t' > t, \forall n \Bigg(\Big(app(n,t) \wedge H(t') \neq n \wedge \exists t_{trav} \,.\, t \leq t_{trav} \leq t' \wedge H(t_{trav}) = n \Big) \\ \Rightarrow \Big(\exists t_{serv} \,.\, t \leq t_{serv} \leq t' \wedge serv(n, t_{serv}) \Big) \Bigg)$$

Dabei bedeuten

$H(t)$ Position des Fahrstuhls zur Zeit t,

$app(n,t)$ offene Anforderung von Stockwerk n zur Zeit t,

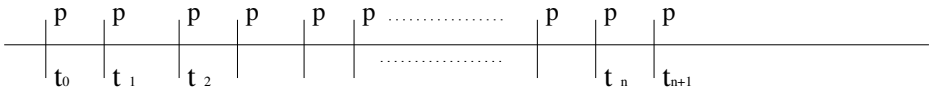
$serv(n,t)$ Fahrstuhl bedient Stockwerk n

Temporale Logik für Infromatik: Pnueli 1977

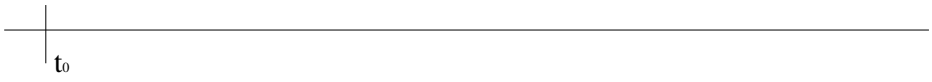
CTL^*:Computation Tree Logic, Emerson und Halpern 1986

$\Diamond p$ irgendwann einmal gilt p

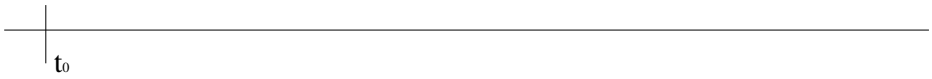
$\Box p$ von jetzt an gilt immer p



$\Diamond \Box p$ bedeutet?



$\Box \Diamond p$ bedeutet?



CTL^*-Formeln

Dienen der Berschreibung von Eigenschaften des Berechnungsbaumes (computation tree). Dieses entsteht z.B. durch "Abwickeln" der Kripke-Struktur. CTL^*-Formel enthalten Pfad-

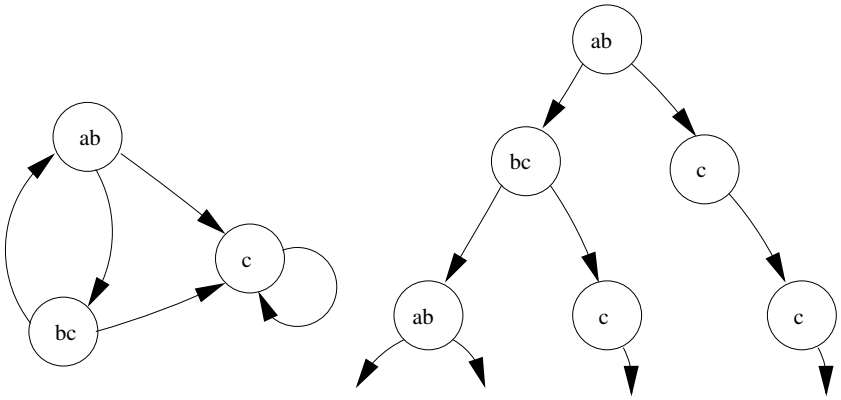


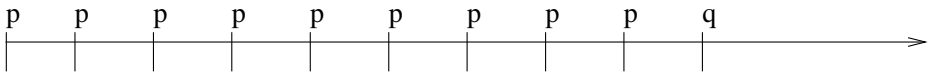
Abbildung 5.14: Abwicklung einer Kripke-Struktur

Quantoren:

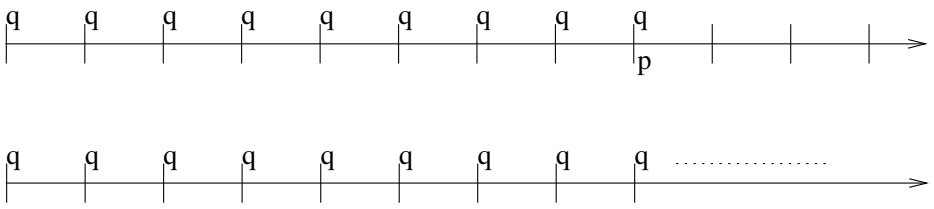
- A "für alle Pfade",
- E "es gibt ein Pfad",

und temporale Quantoren:

- Xp "next time" : p gilt im zweitem Zustand des Pfades (vorher $\bigcirc$),
- Fp "eventually, in the future" : p gilt in einem Zustand des Pfades (vorher $\diamond$) ,
- Gp "always, globally" : p gilt in allen Zuständen des Pfades (vorher $\Box$),
- pUq "until" : es gibt einen Zustand auf dem Pfad, in dem q gilt. Vor diesem Zustand gilt immer p.



pRq "release" : Dual zu pUq. q gilt bis einschliesslich des ersten Zustands, in dem p gilt oder q gilt immer.



Es gibt zwei Arten von CTL^*-Formeln: Zustands-Formeln und Pfad-Formeln. Die Syntax der Formeln ist:

- Eine atomare Aussage $p \in AP$ ist eine Zustands-Formel.
- Sind f und g Zustands-Formeln, dann auch $\neg f$, $f \vee g$ und $f \wedge g$.
- Ist f eine Pfad-Formel, so sind $\mathbf{E}f$ und $\mathbf{A}f$ Zustands-Formeln.
- Ist f eine Zustands-Formel, so ist f auch eine Pfad-Formel.
- Sind f und g Pfad-Formeln, so auch $\neg f$, $f \vee g$, $f \wedge g$, Xf, Ff, Gf, fUg und fRg.
- Diese und nur so gebildete Formeln sind Formeln von CTL^*

CTL^* Semantik in Bezug auf Kripke-Strukturen

Eine Rechnung ist $\pi = s_0 s_1 \cdots$ und der Suffix davon $\pi^i = s_i s_{i+1} \cdots$ für $(i \geq 0)$.

- Ist f eine Zustands-Formel so heißt
  $M, s \models f$: die Formel f gilt im Zustand s der Kripke-Struktur M.
- Ist f eine Pfad-Formel so heißt
  $M, \pi \models f$: die Formel f gilt im Pfad π der Kripke-Struktur M.

Angenommen, daß f_1, f_2 Zustands- und g_1, g_2 Pfad-Formeln sind, so ist $\models$ definiert durch.

1. $M, s \models p \quad \Leftrightarrow \quad p \in L(s)$.
2. $M, s \models \neg f_1 \quad \Leftrightarrow \quad M, s \not\models f_1$.
3. $M, s \models f_1 \vee f_2 \quad \Leftrightarrow \quad M, s \models f_1$ oder $M, s \models f_2$.
4. $M, s \models f_1 \wedge f_2 \quad \Leftrightarrow \quad M, s \models f_1$ und $M, s \models f_2$.
5. $M, s \models Eg_1 \quad \Leftrightarrow \quad \exists \pi = s \cdots . M, \pi \models g_1$.
6. $M, s \models Ag_1 \quad \Leftrightarrow \quad \forall \pi = s \cdots . M, \pi \models g_1$.
7. $M, \pi \models f_1 \quad \Leftrightarrow \quad \pi = s \cdots$ und $M, s \models f_1$.
8. $M, \pi \models \neg g_1 \quad \Leftrightarrow \quad M, \pi \not\models g_1$.
9. $M, \pi \models g_1 \vee g_2 \quad \Leftrightarrow \quad M, \pi \models g_1$ oder $M, \pi \models g_2$.
10. $M, \pi \models g_1 \wedge g_2 \quad \Leftrightarrow \quad M, \pi \models g_1$ und $M, \pi \models g_2$.
11. $M, \pi \models Xg_1 \quad \Leftrightarrow \quad M, \pi^1 \models g_1$.
12. $M, \pi \models Fg_1 \quad \Leftrightarrow \quad \exists k \geq 0 \, . M, \pi^k \models g_1$.
13. $M, \pi \models Gg_1 \quad \Leftrightarrow \quad \forall k \geq 0 \, . M, \pi^k \models g_1$.
14. $M, \pi \models g_1 U g_2 \quad \Leftrightarrow \quad \exists k \geq 0 \, . M, \pi^k \models g_2$ und für alle $0 \leq j < k$ gilt $M, \pi^j \models g_1$.
15. $M, \pi \models g_1 R g_2 \quad \Leftrightarrow \quad \forall j \geq 0$, wenn für jeden $i < j$ $M, \pi^i \not\models g_1$ gilt, dann $M, \pi^j \models g_2$.

Es genügen die Operatoren $\vee, \neg, X, U, E$ um alle Formeln von CTL^* auszudrücken:

- $f \wedge g \equiv \neg(\neg f \vee \neg g)$,
- $f \, \mathbf{R} \, g \equiv \neg(\neg f \, \mathbf{U} \neg g)$,
- $\mathbf{F} \, f \equiv True \, \mathbf{U} \, f$,
- $\mathbf{A}(f) \equiv \neg \mathbf{E}(\neg f)$.

Aufgabe 5.29 Pfad-Formel "Zwischen"
$pZq \Leftrightarrow$ zwischen je zwei Zuständen (Abstand > 2) in denen p gilt, gibt es einen, der q erfüllt. Drücken Sie diese Formel mittels G, X und R aus.

5.6.1 Computation Tree Logic (CTL) und Linear Temporal Logic (LTL)

CTL^* enthält zwei Teillogiken: CTL "computation tree logic" ("branching-time" Logik) und LTL "linear temporal logic".

In CTL müssen vor temporalen Quantoren X, F, G, U, R Pfad-Quantoren A und E stehen. Es gibt also 10 Kombinationen:

- AXf und EXf,
- AFf und EFf,
- AGf und EGf,

- $A[fUg]$ und $E[fUg]$,
- $A[fRg]$ und $E[fRg]$.

diese können mittels EXf, EGf, $E[fUg]$ ausgedrückt werden:

- $AXf = \neg EX(\neg f)$,
- $EFf = E[True\,Uf]$,
- $AGf = \neg EF(\neg f)$,
- $AFf = \neg EG(\neg g)$,
- $A[fUg] \equiv \neg E[\neg gU(\neg f \wedge \neg g)] \wedge \neg EG\neg g$
- $A[fRg] \equiv \neg E[\neg fU\neg g]$
- $E[fRg] \equiv \neg A[\neg fU\neg g]$

Beispiele:

- $EF(Start \wedge \neg Ready)$
  Es ist möglich in einen Zustand zu kommen, in dem „*Start*" aber nicht „*Ready*" gilt.
- $AG(Req \rightarrow AF\,Ack)$
  Immer wenn ein Request *Req* erfolgt, dann wird er später einmal mit *Ack* bestätigt.
- $AG(AF\,DeviceEnabled)$
  Die Aussage „*DeviceEnabled*" gilt unendlich oft auf jedem Pfad.
- $AG(EF\,Restart)$
  Von jedem Zustand aus ist es möglich, einen Zustand mit „*Restart*" zu erreichen.

LTL - Linear Temporal Logic (Pnueli 1981)

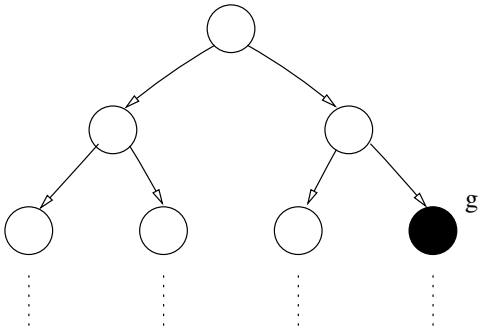
Die Formeln von LTL haben die Form Af wobei f eine Pfad-Formel ist, mit nur trivialen Zustands-Formeln. LTL Pfad-Formeln werden so gebildet:

- Atomare Aussage $p \in AP$ ist eine Pfad-Formel,
- Wenn f und g Pfad-Formeln sind, dann auch $\neg f, f \vee g, f \wedge g, Xf, Ff, Gf, fUg$ und fRg.

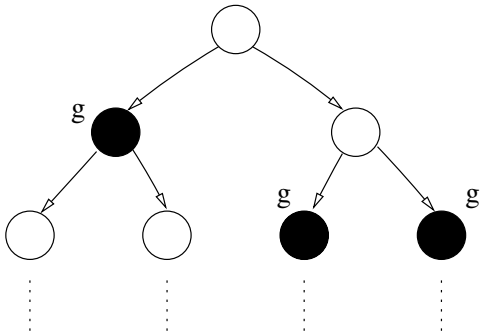
Es gibt keine CTL-Formel, die äquivalent zur LTL-Formel:

$$A(FGp)$$

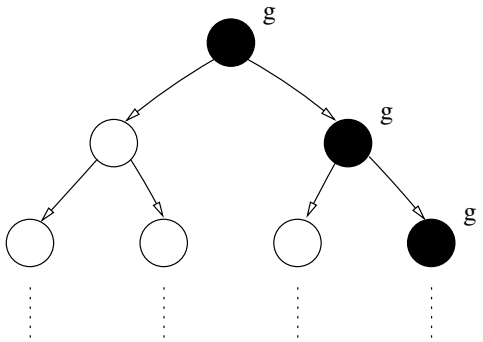
ist! Es heißt: „auf jedem Pfad gibt es einen Zustand, ab dem p immer gilt"



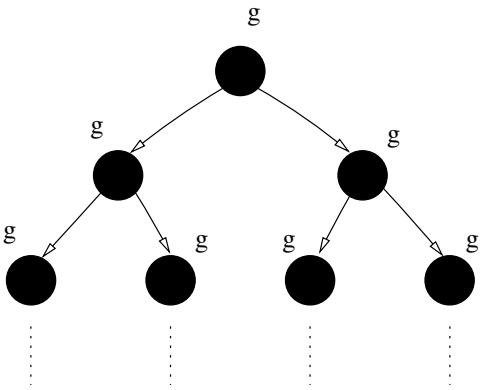
(a) $M, s_0 \models \mathbf{EF}g$



(b) $M, s_0 \models \mathbf{AF}g$

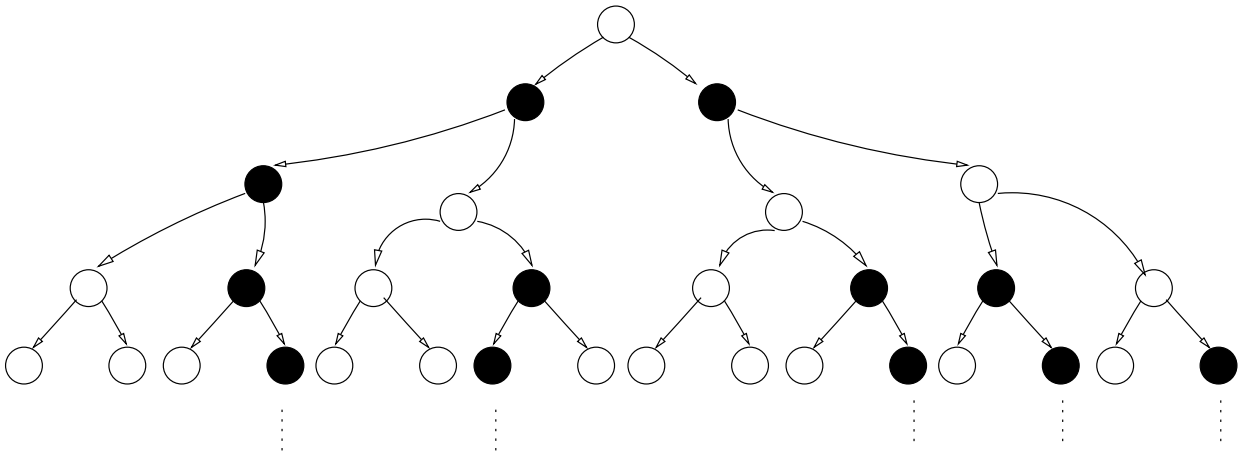


(c) $M, s_0 \models \mathbf{EG}g$



(d) $M, s_0 \models \mathbf{AG}g$

Abbildung 5.15: Einfache CTL Operatoren



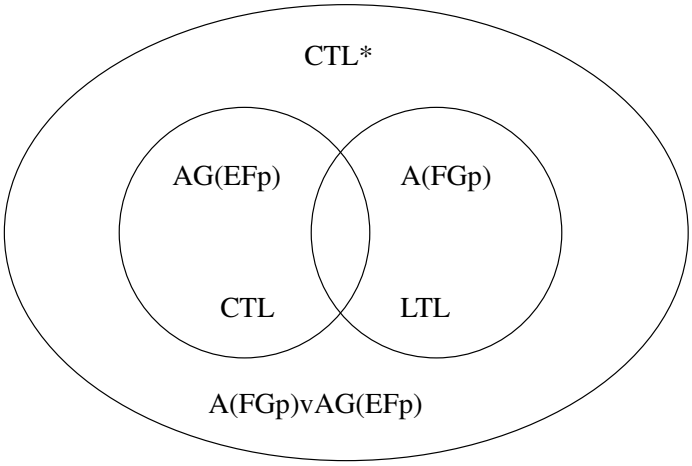
Es gibt keine LTL-Formel, die äquivalent zur CTL-Formel:

$$AG(EFp)$$

ist! Es heißt:
„von jedem Zustand ist ein Zustand erreichbar, in dem p gilt"
$\stackrel{?}{=}$
„alle Pfade enthalten unendlich viele Zustände, in den p gilt"

$$A(GFp)$$

Es gibt eine Formel, z.B. $A(FGp) \vee AG(EFp)$, in CTL^*, die weder in CTL noch in LTL ausdrückbar ist. Also:



Abkürzenden Schreibweisen:

$$\overset{\infty}{\mathbf{G}} := \mathbf{FG}$$

„irgendwann einmal gilt immer"

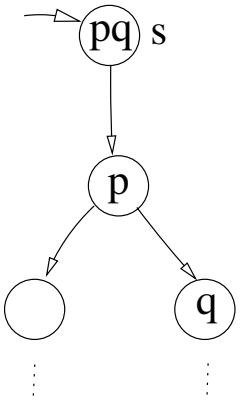
$$\overset{\infty}{\mathbf{F}} := \mathbf{GF}$$

„es gilt unendlich oft"

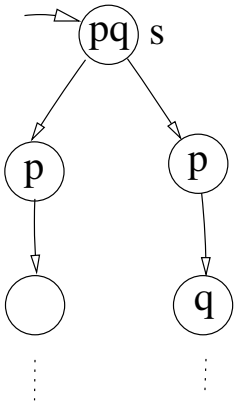
Aufgabe 5.30 Gegeben sind die folgenden Kripke-Stukturen:

a) Gibt es Formeln in LTL, die die Strukturen unterscheiden, d.h. nur in einem Modell gelten?

b) Das gleiche für CTL.



(a) M_1



(b) M_2

5.6.2 Fairness

- „eine Alternative einer sich ständig wiederholender Alternative wird irgendwann einmal auch gewählt" z.B. Hardware Arbiter
- „ein gestörter Kanal übermittelt immer wieder einmal eine Nachricht korrekt" z.B Alternierbitprotokoll

ausdrückbar in CTL^*, aber *nicht* in CTL. CTL ist erwünscht wegen besserer Komplexitäts-Eigenschaften bei der Analyse.
Ausweg: „faire Semantik"
Fairness-Eigenschaften oft durch Zustandsmengen ausdrückbar:

Definition 5.31 (Faire Kripke-Struktur)

$$M = (S, R, L, F)$$

mit $F \subseteq 2^S$.
Für $\pi = s_0 s_1 s_2 \cdots \in S^\omega$ *sei*

$$inf(\pi) = \{s | s = s_i \textit{ für unendlich viele } i \in \mathbb{N}_0\}$$

π heißt *fair* falls für jede Menge $P \in F$ gilt:

$$inf(\pi) \cap P \neq \emptyset$$

Semantik von CTL^* für faire Kripke-Strukturen:

$$\begin{array}{lll} M, s & \models_F & f \qquad \text{(Zustands-Formel f)} \\ M, \pi & \models_F & g \qquad \text{(Pfad-Formel g)} \end{array}$$

wird definiert wie für normale Kripke-Strukturen mit neuen Bedingungen 1, 5 und 6:

1. $M, s \models_F p$ $\Leftrightarrow$ Es gibt einen fairen Pfad der bei s anfängt und $p \in L(s)$.
5. $M, s \models_F E(g_1)$ $\Leftrightarrow$ Es gibt einen fairen Pfad π der bei s anfängt und $\pi \models_F g_1$.
6. $M, s \models_F A(g_1)$ $\Leftrightarrow$ Für alle faire Pfade π, die bei s anfangen, gilt $\pi \models_F g_1$.

Beispiel 5.32

$$F = \{P_1, P_2, \cdots, P_k\}$$

$$P_i = \{s | s \text{ erfüllt } \neg send_i \vee receive_i \text{ für Kanal } i\}$$

fairer Pfad: in jedem Kanal wird unendlich oft empfangen falls gesendet.

5.7 Model Checking

Für eine gegebene Kripke-Struktur $M = (S, R, L)$ und eine gegebene temporal-logische Formel f ist zu berechnen:

$$\{s \in S \mid M, s \models f\}$$

M ist hier zunächst als Graph explicit gegeben.

5.7.1 Algorithmen

Erweitere $L(s)$ für alle $s \in S$ schrittweise zu $label(s)$, die Menge der Teilformeln von f, die in s wahr sind.

in Schritt i: alle Teilformeln mit $i-1$ geschachtelten CTL-Operatoren sind behandelt.

also: Rekursion über Schachtelungstiefe mit 6 Fällen:

1. f atomar:
   für Zustände s mit $f \in L(s)$ setzte $f \in label(s)$.
2. $f = \neg f_1$:
   für Zustände s mit $f_1 \notin label(s)$ setzte $\neg f_1 \in label(s)$.
3. $f = f_1 \vee f_2$:
   für Zustände s mit $f_1 \in label(s)$ oder $f_2 \in label(s)$ setzte $f \in label(s)$.
4. $f = EXf_1$:
   für Zustände s mit $R(s,t)$ und $f_1 \in label(t)$ setzte $f \in label(s)$.
5. $f = E[f_1 U f_2]$:
   für Zustände s mit $f_2 \in label(s)$ setzte $f \in label(s)$;
   für Zustände t mit $R(t,s)$ und $f_1 \in label(s)$ setze $f \in label(t)$;
   fahre schrittweise in gegenrichtung der Transitionen fort und setze $f \in label(s)$, falls es einen Pfad von s zu einem s' mit $f_2 \in label(s')$ gibt, auf dem für alle Zustände t davor $f_1 \in label(t)$ gilt. Siehe Algorithmus 5.4.

Algorithmus 5.4 Auszeichnen mit $E(f_1 U f_2)$

```
PROCEDURE CheckEU(f1, f2)
    T := {s | f2 ∈ label(s)};
    FOR ALL s ∈ T DO label(s) := label(s) ∪ {E[f1 U f2]};
    WHILE T ≠ ∅ DO
        CHOOSE s ∈ T;
        T := T\{s};
        FOR ALL t SUCH THAT R(t, s) DO
            IF E[f1 U f2] ∉ label(t) AND f1 ∈ label(t) THEN
                label(t) := label(t) ∪ {E[f1 U f2]};
                T := T ∪ {t};
            END IF ;
        END FOR ALL ;
    END WHILE ;
END PROCEDURE ;
```

6. $f = EGf_1$: Sei $G = (K, R)$ ein gerichteter Graph mit $R \subseteq K \times K$:

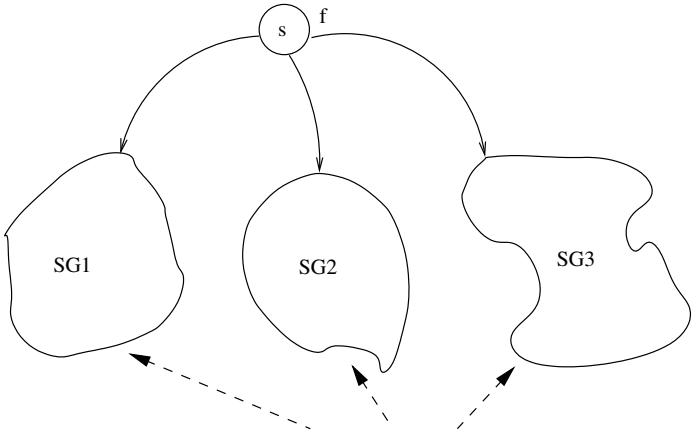


Abbildung 5.16: Stark zusammenhängende Komponenten mit f_1

- $A \subseteq K$ heißt *Zusammenhangskomponente*, falls:

$$\forall a, a' \in A : aR^*a'$$

- sie heißt *stark* oder *streng* (engl. strongly connected components: SCC), falls sie maximal ist:

$$\neg\exists k \in K \backslash A \,.\, \forall a \in A : kR^*a \wedge aR^*k$$

- sie heißt *nichttrivial*, falls:

$$|A| > 1 \text{ oder } \exists a \in A \,.\, aR^+a$$

Algorithmus 5.5 Auszeichnen mit EGf_1

```
PROCEDURE CheckEGf1
    S' := {s | f_a ∈ label(s)};
    SCC := {C | C is a nontrivial SCC of S'};
    T := ⋃_{C∈SCC} {s | s ∈ C};
    FOR ALL s ∈ T DO label(s) := label(s) ∪ {EGf1};
    WHILE T ≠ ∅ DO
        CHOOSE s ∈ T;
        T := T\{s};
        FOR ALL t SUCH THAT t ∈ S' AND R(t,s) DO
            IF EGf1 ∉ label(t) THEN
                label(t) := label(t) ∪ {EGf1};
                T := T ∪ {t};
            END IF ;
        END FOR ALL ;
    END WHILE ;
END PROCEDURE ;
```

betrachte wider $f = EGf_1$:
aus $M = (S, R, L)$ konstruiere $M' = (S', R', L')$ mit:

$$\begin{aligned} S' &= \{s \in S \mid M, s \models f_1\} \\ R' &= R_{|S' \times S'} \\ L' &= L_{|S'} \end{aligned}$$

d.h. die „Einschränkung" von M, ind der f_1 gilt.

Lemma 5.33 *$M, s \models EGf_1$ gdw.*

(a) $s \in S'$

(b) Es gibt einen Pfad in M', der von s zu einer nichttrivialen starken Zusamenhangskomponente in (S', R') führt.

Daraus Algorithmus zur Entscheidung von EGf_1:

(a) Konstruiere $M' = (S', R', L')$

(b) Konstruiere alle SCC von M'. (Algorithmus von Tarjan mit $O(|S'| + |R'|)$ Zeitkomplexität).

(c) Finde Zustände in nichttrivialen SCC.

(d) Suche von diesen rückwärts alle Zustände die dorthin führen.

insgesamt: $O(|S| + |R|)$. Siehe Algorithmus 5.5.

Satz 5.34 *Es gibt einen Algorithmus, der für eine Struktur $M = (S, R, L)$ und eine Formel $f \in CTL$ in $O(|f| \cdot (|S| + |R|))$ Zeitkomplexität entscheidet, ob f für M gilt.*

Beweis:
Wende obiges Verfahren auf die Atome von f an und fahre induktiv fort mit den Teilformeln von f, aufsteigend mit deren Schachtelung. □

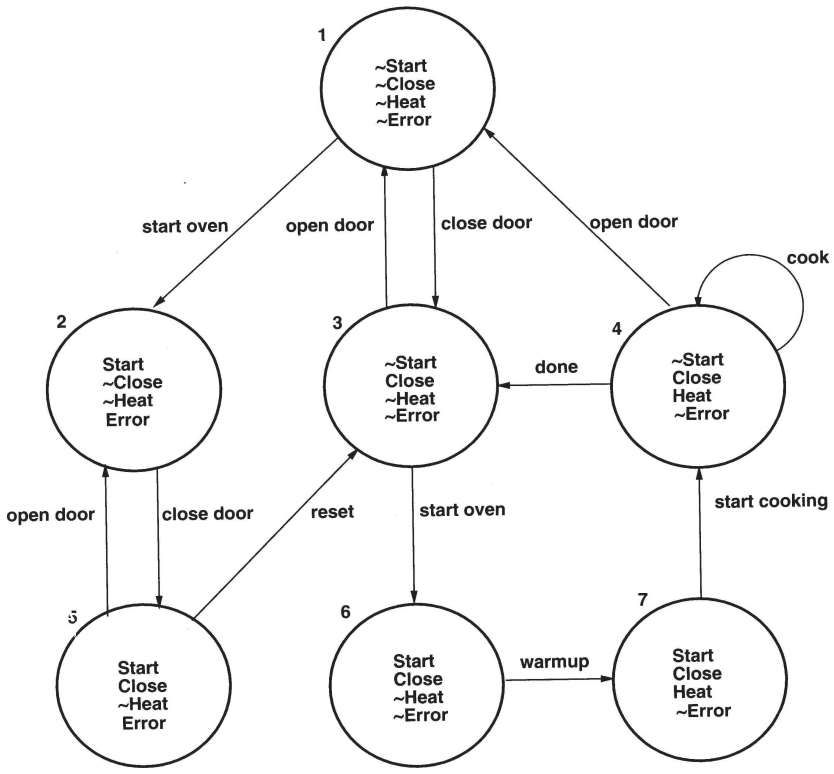


Abbildung 5.17: Kripke-Struktur für das Mikrowellenofenbeispiel

Beispiel 5.35 MW-Ofen Spezifikation zu der Kripke-Struktur in Abb. 5.17:

$$f = AG(Start \rightarrow AF\, Heat)$$

Es gilt immer: nach einem Zustand mit „Start" wird später ein Zustand mit „Heat" erreicht.

Umschreibung von f:

$$\begin{aligned} AGf &= \neg EF(\neg f) \\ &= \neg EF(\neg(\neg Start \vee AF\, Heat)) \\ &= \neg EF(Start \wedge \neg AF\, Heat) && (AFf = \neg EG(\neg f)) \\ &= \neg EF(Start \wedge EG\neg Heat) && (EFf \equiv E[True\, U f]) \end{aligned}$$

$$S(Start) = \{2, 5, 6, 7\}$$

$$S(\neg Heat) = \{1, 2, 3, 5, 6\}$$

Fall $f = EG\neg Heat$:

$$SCC \text{ für } S(\neg Heat) : \{\{1, 2, 3, 5\}\} \quad \text{nicht 6!}$$

Zustände, die mit $EG\neg Heat$ zu markieren sind:

$$T = \{1, 2, 3, 5\} = S(EG\neg Heat)$$

$$S(Start \wedge EG\neg Heat) = \{2, 5, 6, 7\} \cap \{1, 2, 3, 5\} = \{2, 5\}$$

durch Rückwärtspfade:

$$S(EF(Start \wedge EF\neg Heat)) = \{1, 2, 3, 4, 5, 6, 7\}$$

und damit:

$$S(\neg EF(Start \wedge EG\neg Heat)) = \emptyset$$

Also: $M, 1 \not\models AG(Start \rightarrow AF\, Heat)$ d.h. Spezifikation f gilt *nicht* im Anfangszustand 1.

5.7.2 Fairness in CTL

Sei $M = (S, R, L, F)$ eine faire Kripke-Struktur mit $F = \{P_1, \cdots, P_k\} \subseteq 2^S$.
Eine SCC $C \subseteq S$ heißt fair, falls

$$\forall i \in \{1, \cdots, k\} : P_i \cap C \neq \emptyset$$

Ferner sei $M' = (S', R', L', F')$ mit:

$$\begin{array}{lcll} S' & = & \{s \in S \mid M, s \models_F f_1\} & (\models_F \text{ siehe s.169}) \\ R' & = & R_{|S' \times S'} & \\ L' & = & L_{|S'} & \\ F' & = & \{P_i \cap S' \mid P_i \in F\} & \end{array}$$

Lemma 5.36 *Es gilt $M, s \models_F EGf_1$ gdw.*

1. *$s \in S'$,*
2. *Es gibt in S' einen Pfad, der von s zu einem Zustand t in einer fairen SCC C vom Graphen (S', R') führt.*

Daraus die Prozedur $CheckFairEG(f_1)$ um Spezifikation in fairer Semantik zu prüfen:

$$fair := EGTrue \qquad \text{„Es gibt eine unendliche Folge"}$$

z.B.

$$\begin{array}{lll} M, s \models_F p & \text{gdw.} & M, s \models p \wedge fair \\ M, s \models_F EXf_1 & \text{gdw.} & M, s \models EX(f_1 \wedge fair) \\ M, s \models_F E[f_1 U f_2] & \text{gdw.} & M, s \models E[f_1 U (f_2 \wedge fair)] \end{array}$$

Satz 5.37 *Es existiert ein Algorithmus, der für eine Struktur $M = (S, R, L)$ und eine Formel $f \in CTL$ in $0(|f| \cdot (|S| + |R|) \cdot |F|)$ Zeitkomplexität entscheidet, ob f für M* in der fairen Semantik *gilt.*

Beispiel 5.38
Prüfe $f = AG(Start \rightarrow AF\, Heat)$ wobei vorausgesetzt wird, dass die Benutzer den Ofen immer korrekt bedienen.

$$\begin{array}{lcl} \text{„immer”} & \stackrel{\wedge}{=} & \text{„unendlich oft”} \\ \text{„korrekt bedienen”} & \stackrel{\wedge}{=} & \text{„unendlich oft gilt } Start \wedge Close \wedge \neg Error\text{”} \end{array}$$

Also:

$$\begin{array}{rcl} F & = & \{P\} \\ P & = & \{s \mid s \models Start \wedge (Close \wedge \neg Error)\} = \{6, 7\} \end{array}$$

Mit $S(Start)$, $S(\neg Heat)$ wie oben, erhält man:

$$\{1, 2, 3, 5\}$$

nicht fair, da disjunkt zu P. Also:

$$\begin{array}{rcl} S(EG\neg Heat) & = & \emptyset \\ S(EF(Start \wedge EG\neg Heat)) & = & \emptyset \\ S(\neg EF(Start \wedge EG\neg Heat)) & = & \{1, \cdots, 7\} \end{array}$$

Die Spezifikation ist in fairen Semantik erfüllt da die Formel f im Anfangszustand gilt.

5.8 Binäre Entscheidungsdiagramme (BDDs)

BDD - binary decision diagrams
wurden aus binären Entscheidungsbäumen für boole'sche Funktionen entwickelt.

Beispiel 5.39 Für die Funktion:

$$f(a_1, a_2, b_1, b_2) = (a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2)$$

BDD - Boole'sche Funktion

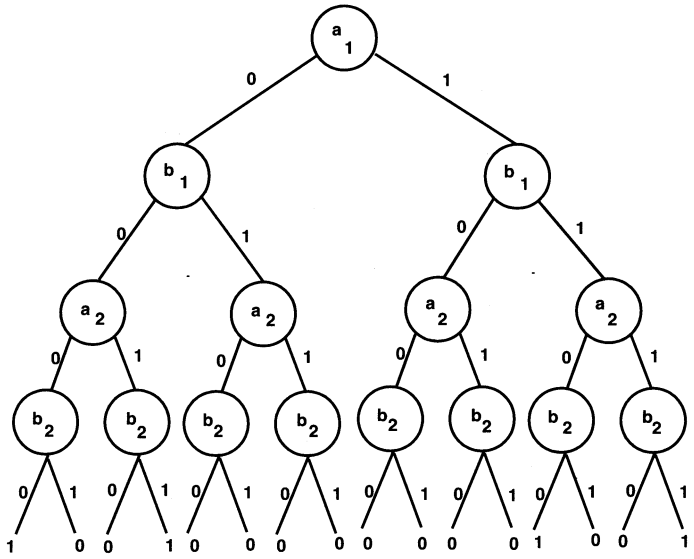


Abbildung 5.18: BDD für 2-bit Komparator

Grösse von BDD's $\stackrel{\wedge}{=}$ Grösse von Entscheidungstabellen

$\longrightarrow$ exponentiell

aber viele redundante Informationen enthalten

$\longrightarrow$ Verschmelzen gleicher Unterbäume

durch Funktion $f_v(x_1, \cdots, x_n)$ für einen Knoten v.

1. Wenn v ein Blatt ist, dann ist $f_v(x_1, \cdots, x_n) = value(v)$
2. Wenn v kein Blatt ist, dann für $x_i = var(v)$ gilt

$$f_v(x_1, \cdots, x_n) = (\neg x_i \wedge f_{low(v)}(x_1, \cdots, x_n)) \vee (x_i \wedge f_{high(v)}(x_1, \cdots, x_n))$$

wobei $value(v)$ die Auszeichnung des Blattes v ist. Für einen inneren Knoten ist $var(v)$ die Variable die ihn auszeichnent. $low(v)$ und $high(v)$ bezeichnen die Nachfolger in Richtung der 0-Kante und der 1-Kante.

Beispiel 5.40 Für den inneren Knoten links unten in Abb. 5.18

$$f_v(a_1, a_2, b_1, b_2) = (\neg b_2 \wedge 1) \vee (b_2 \wedge 0) = \neg b_2$$

5.8.1 Normalformen und logische Operationen auf BDD's

Wünschenswert ist eine Normalform für BDD's. Notwendige Eigenschaft: Zwei BDD's sind equivalent gdw. es Representationen für beide existieren die *isomorph* sind. Dazu zwei Bedingungen von Bryant:

1. Die Variablen mussen geordnet sein. Geordnete BDD's heissen OBDD's.

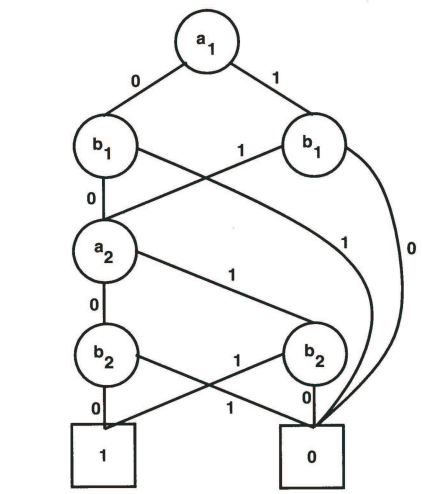
2. keine redundante Unterbäume. Dafür folgende Prozedur *Reduce*:

   - *Entferne doppelte Bläter:* Entferne alle bis auf einen Blat mit gleicher Auszeichnung und biege alle jetzt losen Kanten zu diesem Blat.

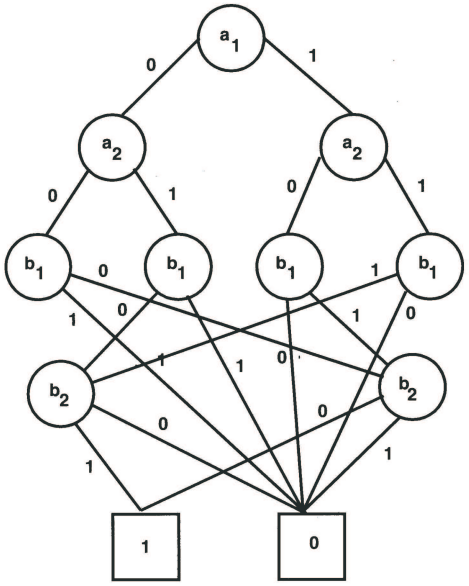
   - *Entferne doppelte innere Knoten:* Für zwei innere Knoten u und v:
     Wenn $var(u) = var(v)$, $low(u) = low(v)$ und $high(u) = high(v)$, dann lösche einen und seine abgehenden Kanten. Die eingehenden Kanten werden zu dem anderen hingebogen.

   - *Entferne redundante Abfragen:* Für einen inneren Knoten v:
     Wenn $low(v) = high(v)$, dann entferne v und seine abgehenden Kanten. Die eingehenden Kanten werden zu $low(v)$ hingebogen.

Ergebnis für das Beispiel 5.39 mit der Ordnung $a_1 < b_1 < a_2 < b_2$. Bei so geordneten Variablen erhält man im algemeinen für einen n-bit Komparator $3n + 2$ Knoten.



Mit der Ordnung $a_1 < a_2 < b_a < b_2$ erhält man in allgemeinem $3 \cdot 2^n - 1$ Knoten.

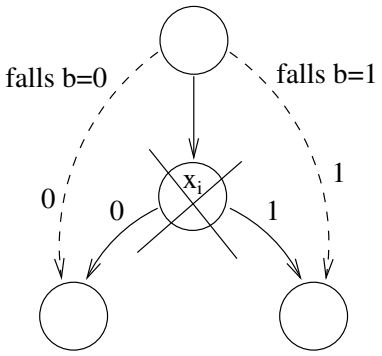


Das Problem, eine optimale Ordnung zu finden ist NP-Vollständig!
Es gibt aber viele heuristische Ansätze. **Logische Operationen auf BDD's**

Einschränken eines Arguments x_i der Funktion f auf einen Wert $b \in \{0, 1\}$

$$f_{|x_i \leftarrow b}(x_1, \cdots, x_n) = f(x_1, \cdots, x_{i-1}, b, x_{i+1}, \cdots, x_n)$$

OBDD von $f_{|x_i \leftarrow b}$ aus OBDD von f:



Danach Anwendung von Reduce.

Shanon-Expansion

$$f = (\neg x \wedge f_{|x \leftarrow 0}) \vee (x \wedge f_{|x \leftarrow 1})$$

Algemeines Verfahren *Apply*:
Sei $\star$ die logische Operation auf die zwei boole'schen Funktionen f und f'

- v und v' sind die Wurzel-Knoten von OBDD's f und f',
- Wenn v und v' Bläter sind, dann $f \star f' = value(v) \star value(v')$, sonst
- $x = var(x)$ $x' = var(x)$
- Wenn $x = x'$, dann „Shanon-Expansion":
$$f \star f' = (\neg x \wedge (f_{|x \leftarrow 0} \star f'_{|x \leftarrow 0})) \vee (x \wedge (f_{|x \leftarrow 1} \star f'_{|x \leftarrow 1}))$$
- Wenn $x < x'$, dann ist f' unabhängig von x. Also:
$$f \star f' = (\neg x \wedge (f_{|x \leftarrow 0} \star f')) \vee (x \wedge (f_{|x \leftarrow 1} \star f'))$$
- Wenn $x' < x$, dann ist f unabhängig von x'. Also:
$$f \star f' = (\neg x' \wedge (f \star f'_{|x' \leftarrow 0})) \vee (x' \wedge (f \star f'_{|x' \leftarrow 0}))$$

Apply hat mit dynamischer Programierung polynomiele Zeitkomplexität.

- OBDD : Bryant 1986
- symbolische Darstellung von Transitionssystemen : McMillan 1987
- Anwendung des CTL-Algorithmuses von Clarke/Emerson 1981 erlaubt drastische Erhöhung der Grösse der Zustandsmenge, z.B. 10^{20} Zustände
- Model Checking System von McMillan : SMV
- Beispiel einer realen Anwendung:
IEEE futurebus standard von 1988, verfikation erst 1992 mit SMV

5.8.2 Darstellung von Kripke-Strukturen durch OBDD's

Für Relation $Q \subseteq \{0,1\}^n$ mit charakteristischer Funktion:

$$f_Q(x_1, \cdots, x_n) = 1 \Leftrightarrow Q(x_1, \cdots, x_n)$$

Falls $Q \subseteq D^n$ mit $|D| = 2^m, \quad m > 1$ wird die Bijektion ϕ so defniert:

$$\phi : \{0,1\}^m \to D$$

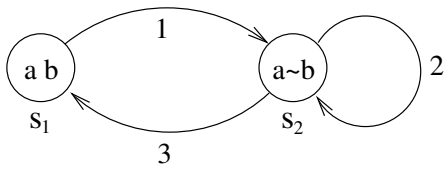
Daraus neue Relation $\hat{Q} \subseteq \{0,1\}^{m \cdot n}$:

$$\hat{Q}(\bar{x}_1, \cdots, \bar{x}_n) = Q(\phi(\bar{x}_1), \cdots, \phi(\bar{x}_n))$$

Die Kripke-Struktur $M = (S, R, L)$ wird folgendermaßend kodiert:

- Zustandsmenge S mit $\phi : \{0,1\}^m \to S$
- Transitionsreleation R mit $\hat{R}(\bar{x}, \bar{x}') = R(\phi(\bar{x}), \phi(\bar{x}'))$
- Auszeichnungsmengen $L : L_p := \{s \,|\, p \in L(s)\}$

Beispiel 5.41



Dazu die Formel:

$$(a \wedge b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge b')$$

wobei a, b die Ausgang- und a', b' Nachfolgerzustandsvariablen sind.