7

Modellierung von Abläufen

In diesem Kapitel führen wir zwei grundlegende Kalküle ein, mit denen Abläufe modelliert werden können: *endliche Automaten* und *Petri-Netze*. Sie werden eingesetzt, um das dynamische Verhalten von Systemen zu beschreiben. Typische Beispiele dafür sind

- die Wirkung von Bedienoperationen auf reale Automaten oder auf Benutzungsoberflächen von Softwaresystemen;
- Schaltfolgen von Ampelanlagen;
- Abläufe von Geschäftsprozessen in Firmen;
- Steuerung von Produktionsanlagen.

Solche Abläufe werden modelliert, indem man die Zustände angibt, die das System einnehmen kann, und beschreibt, unter welchen Bedingungen es aus einem Zustand in einen anderen übergeht. Beide Kalküle sind mit recht einfachen Regeln definiert und haben sehr anschauliche grafische Repräsentationen.

In Abb. 7.1 ist ein endlicher Automat angegeben, der den Geldeinwurf für einen einfachen Getränkeautomaten modelliert. Er hat drei Zustände, die jeweils den bisher eingeworfenen Geldbetrag repräsentieren. Zustandsübergänge werden durch Einwurf eines 1- oder 2-Euro-Stückes bzw. durch Drücken des Geldrückgabeknopfes bewirkt. Das Modell beschreibt, welche Folgen von Bedienoperationen korrekt sind. Endliche Automaten eignen sich zur Modellierung sequentieller Abläufe. Demgegenüber kann man mit Petri-Netzen nebenläufige Vorgänge beschreiben, bei denen Ereignisse im Prinzip gleichzeitig an mehreren Stellen des Systems Zustandsänderungen bewirken können.

Abb. 7.2 zeigt ein Petri-Netz, das zwei zyklische Prozesse modelliert. Schaltregeln der Petri-Netze lassen die beiden Prozesse mit ihren Marken jeweils zwischen den Zuständen *1* und *2* bzw. *4* und *5* wechseln. Die markierte mittlere Stelle *3* synchronisiert die beiden Prozesse, sodass sie sich nie gleichzeitig in den Zuständen *2* und *4* befinden können. Auf diese Weise könnte man z. B. beschreiben, wie Autos eine einspurige Brücke von zwei Seiten überqueren, sodass sich immer nur ein Auto auf der Brücke befindet.

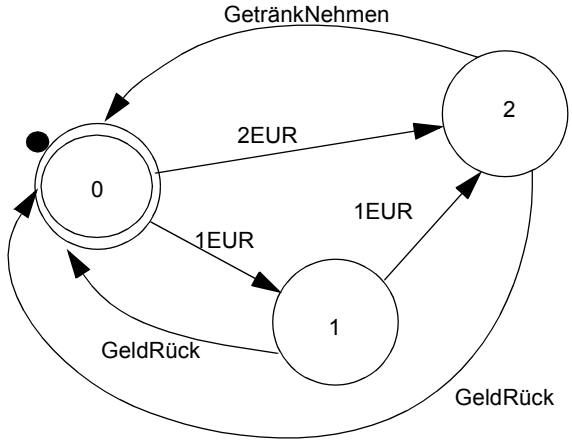


Abbildung 7.1: Endlicher Automat modelliert Geldeinwurf

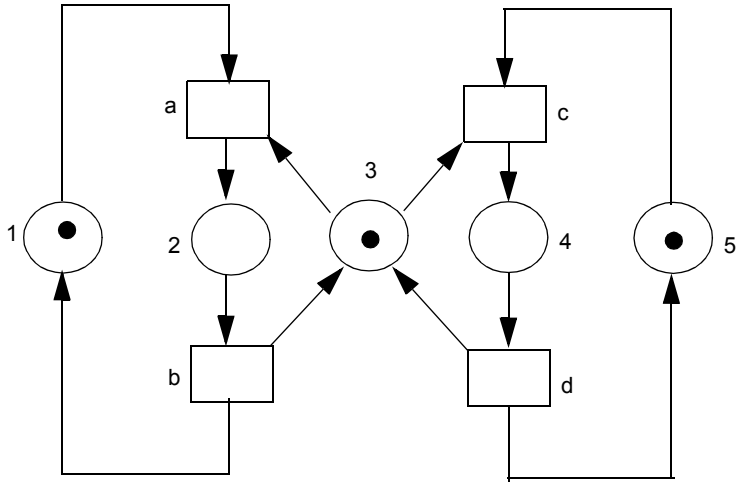


Abbildung 7.2: Petri-Netz modelliert Synchronisation zyklischer Prozesse

7.1 Endliche Automaten

Ein *endlicher Automat* wird mit den Begriffen eines formalen Kalküls definiert, um eine reale oder abstrakte Maschine zu modellieren. Dabei wird insbesondere beschrieben, wie die Maschine

- auf äußere Ereignisse reagiert,

- ihren inneren Zustand ändert und
- gegebenenfalls Ausgabe produziert.

Endliche Automaten werden typisch eingesetzt, um

- das Verhalten realer Maschinen zu spezifizieren, z. B. Getränkeautomaten;
- das Verhalten von Software-Komponenten zu spezifizieren, z. B. wie Benutzungsoberflächen auf Bedienereignisse reagieren;
- Sprachen zu spezifizieren als Menge der Ereignis- oder Symbolfolgen, die der endliche Automat akzeptiert, z. B. die Schreibweise von Bezeichnern und Zahlwerten in Programmen.

Wir beschreiben zunächst nur, wie endliche Automaten die Ereignisse oder Eingaben verarbeiten; die Ausgabe fügen wir später hinzu.

Abb. 7.1 zeigt einen endlichen Automaten in grafischer Darstellung, der die Bedienung eines einfachen Getränkeautomaten spezifiziert. Er hat drei Zustände – die mit *0*, *1* und *2* beschrifteten Kreise. Bei Beginn eines Bedienungszyklus befindet sich der Automat im Anfangszustand (der mit dem schwarzen Punkt gekennzeichnete Zustand *0*). Von jedem Zustand gehen beschriftete Pfeile aus. Sie geben an, welche Ereignisse in dem Zustand akzeptiert werden. So kann im Zustand *0* ein 2-Euro- oder ein 1-Euro-Stück eingeworfen werden. Der Automat wechselt dann in den Zustand, auf den der Pfeil zeigt. Andere Ereignisse sind im Zustand *0* nicht akzeptabel: Versuche, die Geldrückgabetaste zu betätigen oder ein Getränk zu entnehmen, bleiben wirkungslos. Ein Bedienzyklus endet in einem Endzustand, der mit einem Doppelkreis markiert ist; das ist hier auch der Zustand *0*.

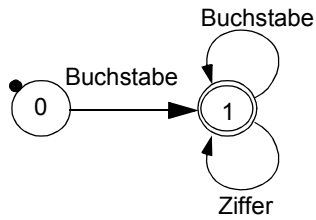


Abbildung 7.3: Endlicher Automat für die Notation von Bezeichnern in Pascal

Der endliche Automat in Abb. 7.3 beschreibt eine abstrakte Maschine, die Folgen von Buchstaben und Ziffern akzeptiert. Hier ist das Ziel, eine Sprache zu definieren, in diesem Fall die Schreibweise der Bezeichner in Pascal-Programmen: Eine beliebig lange Folge von Buchstaben und Ziffern überführt den Automaten vom Anfangszustand *0* in den Endzustand *1*. Solch eine Folge akzeptiert der Automat. Wir nennen sie *ein Wort der Sprache des Automaten*. Beispiele für Worte aus der Sprache dieses Bezeichnerautomaten sind `ggt`, `i`, `max1`, `nt42`, während die Zeichenfolgen `3fach`, `08/15`, `4711` den Automaten nicht in den Endzustand überführen und deshalb nicht zu seiner Sprache gehören.

7.1.1 Zeichenfolgen über Alphabete

Ein endlicher Automat definiert auch eine Sprache. Sie umfasst die Folgen von Symbolen oder Ereignissen, die von dem Automaten als legale Eingabe akzeptiert werden. Deshalb führen wir zunächst die Begriffe *Alphabet* und *reguläre Ausdrücke* ein, um solche Sprachen zu beschreiben.

Definition 7.1: Alphabet

*Ein **Alphabet** ist eine Menge von Zeichen. Sie dienen zur Bildung von Zeichenfolgen. Alphabete werden häufig mit Σ bezeichnet. Wir betrachten hier nur endliche Alphabete.* ■

Beispiele für Alphabete sind:

- die kleinen und großen Buchstaben *{a, ..., z, A, ..., Z}*
- die Dezimalziffern *{0, 1, 8, 9}* und die Binärziffern *{0, 1}*
- der ASCII-Zeichensatz mit 2^7 verschiedenen Zeichen
- der Unicode-Zeichensatz mit 2^{15} verschiedenen Zeichen
- die Menge der Ereignisse, die bei der Bedienung des Getränkeautomaten in Abb. 7.1 ausgelöst werden. Sie werden durch frei erfundene Symbole repräsentiert, wie *1EUR*.

Definition 7.2: Wort

*Ein **Wort w über einem Alphabet** Σ ist eine beliebige Zeichenfolge $w \in \Sigma^*$. Statt der Notation von Folgen aus Kapitel 2 $(a_1, a_2, ..., a_n) \in \Sigma^*$ schreiben wir die Zeichen eines Wortes unmittelbar hintereinander: $a_1a_2 ... a_n$. Für die leere Folge schreiben wir ε.* ■

Jedes Wort dieses Textes ist ein Beispiel für ein Wort über dem Alphabet der deutschen Buchstaben. Zahlen wie *127* und *42* sind Worte über dem Alphabet der Dezimalziffern. Häufig soll die Menge von beliebigen Worten über einem Alphabet auf eine Teilmenge eingeschränkt werden, deren Worte nach bestimmten Regeln aufgebaut sind. Zum Beispiel werden Bezeichner in Pascal-Programmen als Worte über dem Alphabet der ASCII-Zeichen gebildet. Sie dürfen aber nur Buchstaben und Ziffern enthalten und müssen mit einem Buchstaben beginnen. Solche Regeln zum Aufbau von Worten aus Teilworten und Zeichen formuliert man mit regulären Ausdrücken.

Definition 7.3: Reguläre Ausdrücke

*Ein **regulärer Ausdruck F** über einem Alphabet Σ definiert eine Menge von Worten L (F) $\subseteq \Sigma^*$. Häufig schreiben wir auch einfach F statt L (F) für die durch F definierte Wortmenge. Die Konstrukte zur Bildung regulärer Ausdrücke werden rekursiv definiert. Die folgende Liste gibt jeweils die Schreibweise eines regulären Ausdruckes an und die Wortmenge, die er definiert. Dabei stehen **F** und **G** für beliebige reguläre Ausdrücke:*

1. *∅ definiert die leere Menge, also die Sprache, die keine Worte enthält.*
2. *a definiert {a}, die Menge, die nur das Zeichen a als Wort enthält.*
3. *ε definiert {ε}, die Menge, die nur das leere Wort enthält.*
4. *F | G definiert {f | f ∈ F} ∪ {g | g ∈ G} mit Worten, die aus F oder G stammen.*
5. *F G definiert {f g | f ∈ F, g ∈ G} mit Worten, die aus jeweils einem Wort aus F und G zusammengesetzt sind.*
6. *F^n definiert $\{f_1 f_2 ... f_n \mid \forall\ i \in \{1, ..., n\}: f_i \in F\}$ mit Worten, die jeweils aus n Worten aus F bestehen.*
7. *F* definiert $\{f_1 f_2 ... f_n \mid n \geq 0$ und $\forall\ i \in \{1, ..., n\}\ f_i \in F\}$ mit Worten, die beliebig lange Folgen von Worten aus F sind.*
8. *F^+ definiert $\{f_1 f_2 ... f_n \mid n \geq 1$ und $\forall\ i \in \{1, ..., n\}: f_i \in F\}$ mit Worten, die nicht-leere Folgen von Worten aus F sind.*
9. *(F) definiert F, dient zur Strukturierung von regulären Ausdrücken.* ■

Die Schreibweise von Pascal-Bezeichnern kann man mit dem folgenden regulären Ausdruck beschreiben:

(a | b | ... | z) ((a | b | ...| z) | (0 | 1 | ... | 9))*

Wir haben hier darauf verzichtet, alle Buchstaben und Ziffern aufzuzählen. Häufig gibt man regulären Ausdrücken Namen, um sie wieder zu verwenden, z. B.

B = (a | b | ... | z)
D = (0 | 1 |... | 9)
Bezeichner = B (B | D)*

Die Wortmenge dieses regulären Ausdruckes enthält z. B. die Worte

max, min1, i ∈ Bezeichner

zwar ist *3d* eine Zeichenfolge über demselben Alphabet wie das des regulären Ausdruckes *Bezeichner*; sie gehört jedoch nicht zu seiner Wortmenge. Wir geben weitere Beispiele für reguläre Ausdrücke an:

e (+ | – | ε) D^+	mit Worten wie	e+10	e-8	e42
D^* . D^+	mit Worten wie	2.35	0.4	.4
1^2 (1 | 0)* 0^2	mit Worten wie	1100	11100	

Im Zusammenhang mit regulären Ausdrücken muss man die Bedeutungen folgender Notationen sorgfältig unterscheiden:

- ε: das leere Wort
- ε: der reguläre Ausdruck, der die Wortmenge $\{\varepsilon\}$ definiert
- $\{\varepsilon\}$: die Menge, die das leere Wort als einziges Element enthält
- $\varnothing$: die leere Menge, also die Sprache, die keine Worte enthält.

Wir können auch einen regulären Ausdruck mit den Symbolen für die Ereignisse bei der Bedienung des Getränkeautomaten in Abb. 7.1 angeben:

((1EUR GeldRück)* (1EUR 1EUR | 2EUR) GetränkNehmen | GeldRück))*

Seine Wortmenge enthält alle akzeptierten Ereignisfolgen.

7.1.2 Deterministische endliche Automaten

Wir führen nun endliche Automaten ein, mit denen Abläufe modelliert und Wortmengen definiert werden können.

Definition 7.4: Deterministischer endlicher Automat

*Ein **deterministischer endlicher Automat** (engl: deterministic finite automaton oder deterministic finite state machine) ist definiert als 5-Tupel*

$A = (\Sigma, Q, \delta, q_0, F)$ mit

a) Σ ist ein endliches Eingabealphabet,
b) Q ist eine endliche Menge von Zuständen,
c) δ ist eine Übergangsfunktion aus $Q \times \Sigma \rightarrow Q$,
d) q_0 ist der Anfangszustand, $q_0 \in Q$,
e) F ist die Menge der Endzustände, $F \subseteq Q$

$r = \delta(q, a)$ heißt Nachfolgezustand von q unter a. ■

Solche Automaten heißen deterministisch, weil es zu jedem Paar *(q, a)* mit $q \in Q$ und $a \in \Sigma$ höchstens einen Nachfolgezustand δ *(q, a)* gibt, d. h. δ ist eine Funktion in Q. Der Bezeichner-Automat aus Abb. 7.3 wird durch folgendes 5-Tupel formal beschrieben:

Σ = {B, D}
Q = {0, 1}
δ = {((0, B), 1), ((1, B), 1), ((1, D), 1)}
q_0 = 0
F = {1}

Für das Modell des Getränkeautomaten in Abb. 7.1 lautet die Übergangsfunktion:

d = { (0, 1EUR), 1), (0, 2EUR), 2), (1, 1EUR), 2), (1, GeldRück), 0),
(2, GeldRück), 0), (2, GetränkNehmen), 0)}

	1 EUR	2 EUR	GeldRück	GetränkNehmen
0	1	2		
1	2		0	
2			0	0

Abbildung 7.4: Übergangsfunktion des Getränkeautomaten als Tabelle

Die Übergangsfunktion ist einfacher zu lesen, wenn wir sie, wie in Abb. 7.4, als Tabelle angeben.

Betrachtet man die Übergangsfunktionen der Automaten, dann erkennt man, dass es Zustände gibt, in denen die Verarbeitung bestimmter Eingaben nicht definiert ist, z. B. das Einwerfen einer 2-Euro-Münze im Zustand *1*.

Definition 7.5: Vollständiger Automat

*Ein endlicher Automat heißt **vollständig**, wenn die Übergangsfunktion δ eine totale Funktion ist.* ■

Für Zwecke der Realisierung von Automaten sowie für formale Argumentationen kann es notwendig sein, einen Automaten so zu erweitern, dass er vollständig ist: Man ergänzt einen weiteren Zustand *f* und Übergänge *((q, a), f)* für alle Paare *(q, a)*, die in δ nicht definiert sind. Im Falle des Bezeichnerautomaten wären das die Übergänge

((0, D), f), ((f, B), f), ((f, D), f).

f repräsentiert einen Fehlerzustand. Er darf nicht Endzustand sein, damit die vom Automaten akzeptierte Wortmenge durch die Transformation nicht geändert wird.

Endliche Automaten kann man sehr anschaulich als Graphen darstellen, wie in Abb. 7.1 und Abb. 7.3. Die Zustände des Automaten werden durch Knoten repräsentiert, die mit dem Namen des Zustandes markiert sind. Anfangs- und Endzustände werden speziell gekennzeichnet. Die Übergänge werden durch gerichtete Kanten repräsentiert, die mit dem Eingabezeichen markiert sind. Häufig fasst man Zeichen zu einer Zeichenklasse zusammen, wie a, b, ... z, zu Buchstabe, wenn alle Zeichen aus der Klasse in jedem Zustand dieselben Übergänge bewirken. Damit wird die Anzahl gleicher Übergänge reduziert. Wenn wir mit endlichen Automaten modellieren, interessieren uns meist die Zustandsübergänge, die die Zeichen einer Zeichenfolge insgesamt bewirken. Deshalb erweitern wir die Übergangsfunktion, sodass sie auch für Zeichenfolgen definiert ist.

Definition 7.6: Übergangsfunktion für Zeichenfolgen

*Sei δ: $Q \times \Sigma \to Q$ eine Übergangsfunktion eines deterministischen endlichen Automaten. Dann wird die **Übergangsfunktion für Zeichenfolgen** $\hat{\delta}$: $Q \times \Sigma^* \to Q$, rekursiv wie folgt definiert:*

a) $\hat{\delta}(q, \varepsilon) = q$ *für alle* $q \in Q$ *(Übergang mit dem leeren Wort)*
b) $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$ *für alle* $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$.

Statt $\hat{\delta}$ schreiben wir auch δ. ■

Mit diesem Begriff können wir nun sehr einfach die Menge der Worte definieren, die ein endlicher Automat akzeptiert.

Definition 7.7: Akzeptierte Sprache

*Sei $A = (\Sigma, Q, \delta, q_0, F)$ ein deterministischer endlicher Automat und sei $w \in \Sigma^*$. **A akzeptiert das Wort w** genau dann, wenn $\hat{\delta}(q_0, w) \in F$. Die Menge $L(A) = \{w \mid \hat{\delta}(q_0, w) \in F\}$ heißt die **von A akzeptierte Sprache**.* ■

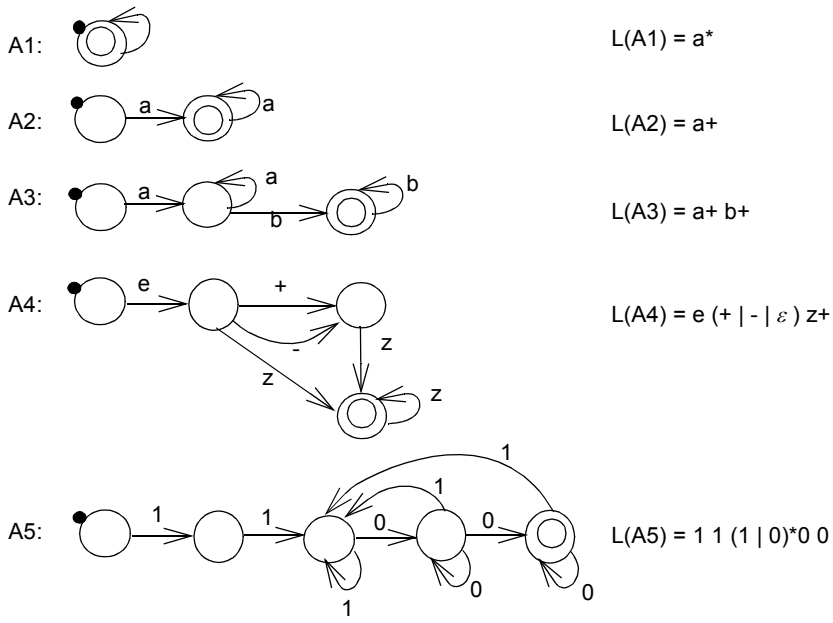


Abbildung 7.5: Einige einfache Automaten und ihre Sprachen

Für die Automaten aus Abb. 7.1 und Abb. 7.3 haben wir oben schon die jeweils akzeptierten Sprachen durch reguläre Ausdrücke beschrieben. In Abb. 7.5 haben wir einige einfache Automaten und ihre Sprache angegeben. Man beachte die typischen Muster für Zustandsübergänge, die Zeichenfolgen $a*$ bzw. a^+ akzeptieren. Sie kommen auch innerhalb größerer Automaten wie A_3 und A_5 vor. Der Automat A_4 zeigt ein Beispiel dafür, wie Alternativen der Sprachmenge modelliert werden. Am Automaten A_5 erkennt man, dass die Erkennung der Wortanfänge einfacher zu modellieren ist als die Entscheidung, ob die gerade akzeptierte *0* eine der beiden letzten ist.

Mit den Mustern aus Abb. 7.5 kann man auf einfache Weise Automaten für die Sprachen $a\ b$, $a^2\ b^2$, $a^3\ b^3$, $a^+\ b^+$ konstruieren. Aber einen endlichen Automaten mit der Sprache $a^n\ b^n$ für beliebige n gibt es nicht. Ein solcher Automat bräuchte $n \geq 0$ Zustände, um bis n zu zählen und weitere Zustände, um gleich viele b zu akzeptieren. Da die Zustandsmenge endlich ist, kann sie nicht für diese Sprache mit beliebig großen n ausreichen.

7.1.3 Nicht-deterministische endliche Automaten

Für manche Modellierungen ist die Forderung, dass δ eine Funktion sein muss, zu restriktiv: Man möchte in manchen Zuständen für den Übergang mit einem Zeichen mehrere Möglichkeiten angeben, ohne festzulegen, welche davon gewählt wird. Solche Entscheidungsfreiheiten in der Modellierung von Abläufen nennt man nicht-deterministisch.

Nicht-deterministische Modelle sind häufig einfacher aufzustellen und leichter zu verstehen als deterministische. So könnte man die Sprache $L(A_5) = 1\ 1\ (1\ /\ 0)^*\ 0\ 0$ aus Abb. 7.5 einfacher durch den nicht-deterministischen Automaten in Abb. 7.6 beschreiben.

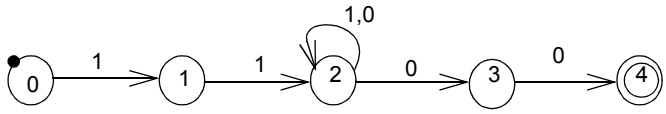


Abbildung 7.6: Nicht-deterministischer Automat

Definition 7.8: Nicht-deterministischer Automat

*Ein **nicht-deterministischer endlicher Automat** ist definiert als 5-Tupel $A = (\Sigma, Q, \delta, q_0, F)$. Die Übergangsfunktion δ bildet einen Zustand q und ein Eingabezeichen a auf eine Menge von Nachfolgezuständen ab, d. h. $\delta: Q \times \Sigma \rightarrow Pow(Q)$. Der Automat kann in jeden Zustand aus $\delta(q, a)$ übergehen. Σ, Q, q_0 und F sind wie für deterministische endliche Automaten definiert.*

■

Die Übergangsfunktion für den Automaten aus Abb. 7.6 ist in Abb. 7.7 angegeben.

	0	1
0	∅	{1}
1	∅	{2}
2	{2, 3}	{2}
3	{4}	∅
4	∅	{∅}

Abbildung 7.7: Übergangsfunktion des nicht-deterministischen Automaten aus Abb. 7.6

Um die Sprachmenge zu definieren, erweitern wir auch hier die Übergangsfunktion auf Zeichenfolgen.

Definition 7.9: Übergangsfunktion für Zeichenfolgen

*Sei $\delta: Q \times S \rightarrow Pow(Q)$ eine Übergangsfunktion eines nicht-deterministischen endlichen Automaten. Dann ist die **Übergangsfunktion für Zeichenfolgen** $\hat{\delta}: Q \times \Sigma^* \rightarrow Pow(Q)$ rekursiv wie folgt definiert:*

a) $\hat{\delta}(q, \varepsilon) = \{q\}$ für alle $q \in Q$ (Übergang mit dem leeren Wort)

b) $\hat{\delta}(q, wa) = \{q' \mid \exists\ p \in Q : p \in \hat{\delta}(q, w)$ und $q' \in \delta(p, a)\}$, für alle $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$; d. h. die Menge aller Zustände, die man von q mit wa erreichen kann.

Statt $\hat{\delta}$ schreiben wir meist auch δ.

■

Definition 7.10: Akzeptierte Sprache

Sei $A = (\Sigma, Q, \delta, q_0, F)$ ein nicht-deterministischer Automat und $w \in \Sigma^*$. A akzeptiert w genau dann, wenn $\hat{\delta}(q_0, w) \cap F \neq \emptyset$. Die Menge $L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$ heißt die ***von A akzeptierte Sprache****.* ■

Es kann aus unterschiedlichen Gründen sinnvoll sein, Abläufe oder Sprachmengen zunächst durch einen nicht-deterministischen Automaten zu modellieren, und nur, wenn es nötig ist, diesen in einen deterministischen Automaten zu transformieren. Wir geben Beispiele für die drei wichtigsten Gründe:

Einfacher Entwurf: Für manche Sprachmenge lässt sich ein regulärer Ausdruck einfacher in einen nicht-deterministischen als in einen deterministischen Automaten übertragen. Der Vergleich der Automaten in Abb. 7.5 und 7.6 belegt dies.

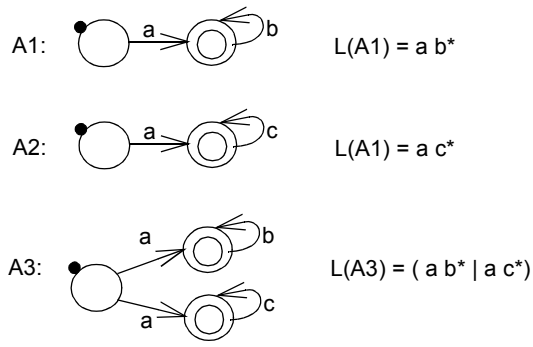


Abbildung 7.8: Zusammensetzen von Automaten

Zusammensetzen von Automaten: Abb. 7.8 zeigt zwei deterministische Automaten für die Sprachmengen $L_1 = a\ b^*$ und $L_2 = a\ c^*$. Man kann leicht einen Automaten für die Vereinigung der beiden Mengen konstruieren $L_1 \cup L_2 = (a\ b^* \mid a\ c^*)$. Dazu vereinigt man die Anfangszustände der beiden Automaten zum Anfangszustand des neuen Automaten, wie in Abb. 7.8. Der neue Automat kann dann, wie in diesem Beispiel, nicht-deterministisch sein.

Nicht-modellierte Entscheidungen: Es soll bei dem Entwurf des Automaten absichtlich offen gelassen werden, wie in manchen Zuständen auf Ereignisse reagiert wird. Die Entscheidung fällt dann außerhalb des Modellierten. Ein Automat zum Geldwechseln kann für ein 1-Euro-Stück Wechselgeld in unterschiedlicher Stückelung ausgeben. Jede Stükkelung sei durch einen Zustand modelliert, in den jeweils unter derselben Eingabe übergegangen wird. Die Entscheidung ist dann nicht modelliert und der Automat nicht-deterministisch.

Wir brauchen keine Nachteile zu befürchten, wenn wir zunächst mit nicht-deterministischen Automaten modellieren. Denn es gilt folgender Satz:

Satz 7.1: Existenz deterministischer Automaten

Sei L(A) die Sprache eines nicht-deterministischen endlichen Automaten. Dann gibt es einen deterministischen Automaten, dessen Sprache auch L (A) ist. ■

Wenn die Modellierung oder die Implementierung es erfordern, kann man aus einem nicht-deterministischen Automaten $A = (\Sigma, Q, \delta, q_0, F)$ einen deterministischen $A' = (\Sigma, Q', \delta', q_0', F')$ mit $L(A) = L(A')$ systematisch konstruieren.

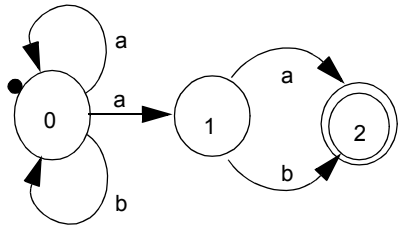


Abbildung 7.9: Nicht-deterministischer Automat für Worte mit a als vorletztem Zeichen

Abb. 7.9 zeigt einen nicht-deterministischen Automaten A mit $L(A) = (a \mid b)^* a\ (a \mid b)$; das sind alle Worte über $\Sigma = \{a, b\}$, deren vorletztes Zeichen ein a ist. Die Korrespondenz zwischen dem regulären Ausdruck und dem Automaten ist leicht erkennbar. Abb. 7.10 zeigt einen deterministischen Automaten A' mit derselben Sprachmenge. Wir geben nun ein allgemeines Konstruktionsverfahren dafür an:

Jeder Zustand aus Q' repräsentiert eine Menge von Zuständen aus Q. Folgende Konstruktionsschritte sind auszuführen:

1. Der Anfangszustand $q'_0 = \{q_0\}$ wird in Q' eingefügt.
2. Wähle einen schon konstruierten Zustand $q' \in Q'$, wähle ein Zeichen $a \in \Sigma$ und berechne $r' = \delta'(q', a) = \bigcup_{q \in Q} \delta'(q, a)$, d. h. r' repräsentiert die Vereinigung aller Zustände, die in A von q unter a erreicht werden können. r' wird als Zustand in Q' aufgenommen und $\delta(q', a) = r'$ wird ein Übergang in δ'.
3. Wiederhole Schritt 2, bis es kein Paar (q', a) mehr gibt, zu dem ein neuer Zustand oder Übergang konstruiert werden kann.
4. Die Endzustände sind $F' = \{q' \in Q \mid q' \cap F \neq \varnothing\}$, d. h. q' ist Endzustand, genau dann, wenn seine Zustandsmenge einen Endzustand von A enthält.

Der Automat in Abb. 7.10 ist nach diesem Verfahren entstanden. Das Konstruktionsverfahren garantiert, dass immer ein deterministischer Automat entsteht und dass seine Sprachmenge mit der des ursprünglichen Automaten übereinstimmt. Die zentrale Konstruktionsidee ist: nicht-deterministische Entscheidungen im Automaten A werden in „spätere“ Zustände in B verschoben, durch Zusammenfassen von Zuständen zu Mengen.

Es ist allerdings möglich, dass sich im schlimmsten Fall die Zahl der Zustände durch die Konstruktion exponentiell vergrößert.

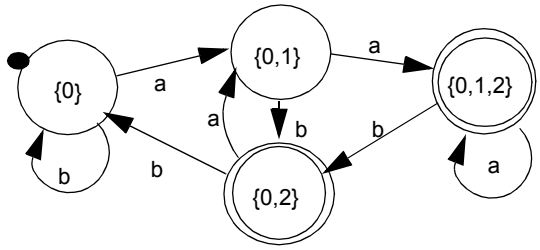


Abbildung 7.10: Deterministischer Automat zum Automaten aus Abb. 7.9

7.1.4 Endliche Automaten mit Ausgabe

Die Zustandsübergänge eines endlichen Automaten werden durch Eingabe von Symbolen des Eingabealphabetes bestimmt. Sie können auch Ereignisse modellieren, die auf das System einwirken und sein Verhalten bestimmen. Will man außerdem auch Reaktionen des Systems modellieren, so erweitert man den endlichen Automaten um eine *Ausgabefunktion*, die Worte über einem Ausgabealphabet erzeugt. Das kann man in zwei Varianten tun:

Definition 7.11: Mealy-Automat

Sei A ein deterministischer oder nicht-deterministischer endlicher Automat, T ein ***endliches Ausgabealphabet*** *und* $\lambda: Q \times \Sigma \rightarrow T^*$ *eine* ***Ausgabefunktion****, die den* ***Zustandsübergängen*** *von A' jeweils ein Wort über T zuordnet. Dann ist A erweitert um T und* λ *ein* ***Mealy-Automat****.* ■

Definition 7.12: Moore-Automat

Sei A ein deterministischer oder nicht-deterministischer endlicher Automat, T ein ***endliches Ausgabealphabet*** *und* $\mu: Q \rightarrow T^*$ *eine* ***Ausgabefunktion****, die den* ***Zuständen*** *von A jeweils ein Wort über T zuordnet. Dann ist A erweitert um T und* μ *ein* ***Moore-Automat****.* ■

Wir haben in Abb. 7.11 einen sehr einfachen, abstrakten Mealy-Automaten und in Abb. 7.12 einen Moore-Automaten als Beispiele angegeben. Beide akzeptieren die beiden Worte *a* und *ba* und erzeugen dabei als Ausgabe *x* bzw. *yx*. Der Mealy-Automat kann bei jedem Übergang unterschiedliche Ausgabeworte erzeugen und deshalb die Ausgabe feiner differenzieren, als es der Moore-Automat vermag. Er erzeugt Ausgabe beim Erreichen eines Zustandes, unabhängig vom Übergang dorthin. In unserem Beispiel sind die Ergebnisse gleich, weil in diesem Mealy-Automaten alle Übergänge, die in einen Zustand führen, dieselbe Ausgabe erzeugen, die im Moore Automaten in *q* erzeugt wird. Es ist

deshalb auch sehr einfach, einen Moore-Automaten in einen Mealy-Automaten zu transformieren. In umgekehrter Richtung muss man ggf. neue Zustände einführen, um die Ausgabe zu differenzieren.

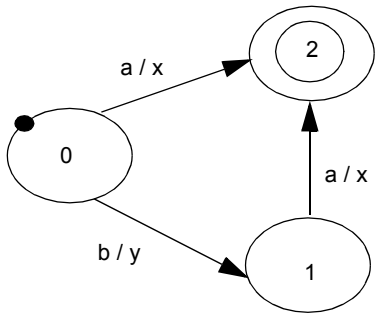


Abbildung 7.11: Mealy-Automat

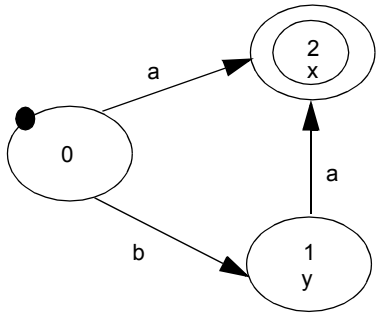


Abbildung 7.12: Moore-Automat

Hier haben wir als Wertebereich der Ausgabefunktion **Worte** über dem Ausgabealphabet *T* definiert. Dadurch ist es möglich, auch einigen Übergängen bzw. Zuständen das leere Wort als Ausgabe zuzuordnen. Das ist nützlich, wenn das modellierte System nicht auf jede Eingabe mit einer Ausgabe reagiert. Die Möglichkeit, eine Zeichenfolge statt einzelner Zeichen zu erzeugen, erlaubt es, einen Automaten um Ausgabe zu erweitern, ohne zusätzliche Zustände nur zum Zwecke der Ausgabe von Zeichenfolgen einführen zu müssen.

In den Abb. 7.13 und 7.14 haben wir den Getränkeautomaten aus Abb. 7.1 um Ausgabe zu einem Mealy- bzw. zu einem Moore-Automaten erweitert. Im Moore-Automaten der Abb. 7.14 mussten wir die Zustände *3*, *4*, *5* zusätzlich einführen, um darin die unterschiedliche Ausgabe von einem Getränk, 1-Euro- und 2-Euro-Stück zuzuordnen. Bei dem Mealy-Automaten in Abb. 7.13 konnten wir einfach die Ausgabe an den Übergängen in denselben Endzustand differenzieren.

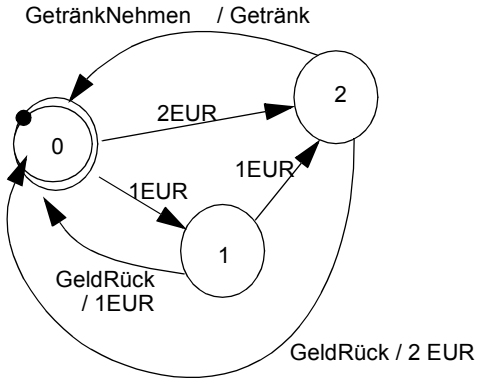


Abbildung 7.13: Getränkeautomat als Mealy-Automat

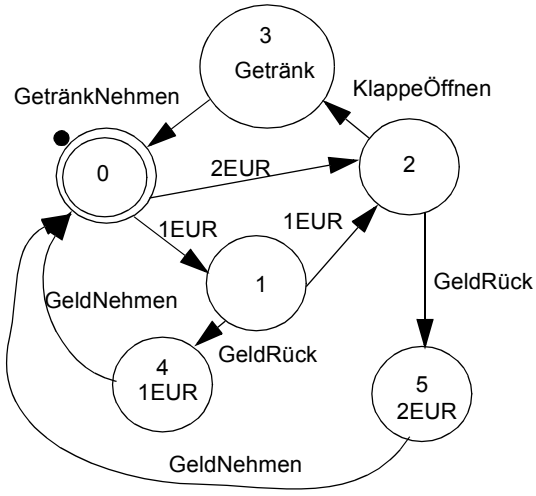


Abbildung 7.14: Getränkeautomat als Moore-Automat

Abb. 7.15 zeigt als abschließendes größeres Beispiel die Modellierung der Abläufe bei der Bedienung eines Telefons. Die Eingaben sind Ereignisse, die durch den Bediener (z. B. digi t(n)) oder durch Signale auf der Leitung (z. B. number busy) ausgelöst werden. Die Ausgabe ist durch die Kombination beider Arten modelliert: im Zustand wie beim Moore-Automaten (z. B. do: sound dial tone) oder am Übergang wie beim Mealy-Automaten (z. B. disconnect line).

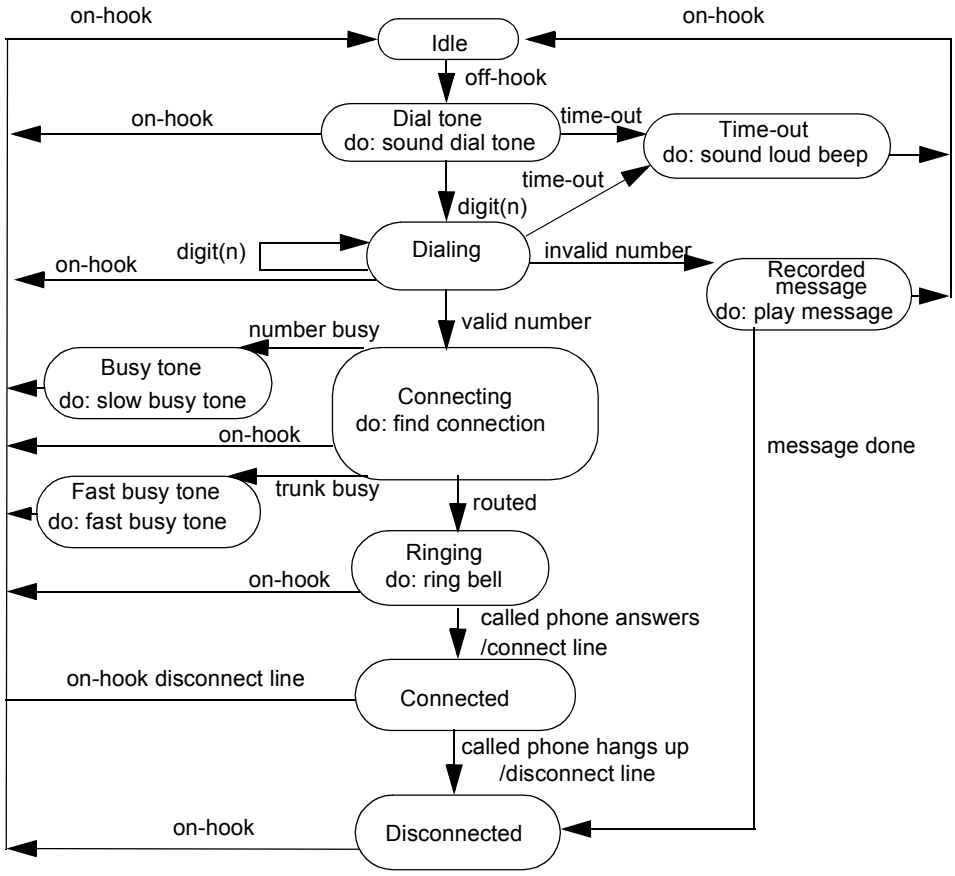


Abbildung 7.15: Automat modelliert die Bedienung eines Telefons, aus [29]

7.1.5 Endliche Automaten in UML

Die Modellierungssprache UML enthält eine Teilsprache für eine spezielle Form endlicher Automaten in grafischer Notation. Sie wird eingesetzt, um das Verhalten von Systemkomponenten, z. B. die Ausführung von Operationen, zu beschreiben. (Abschnitt 6.4 enthält eine kurze Einführung zu UML.)

Die in UML verwendete Variante endlicher Automaten basiert auf den sogenannten Statecharts, die 1987 von Harel eingeführt wurden. Sie zeichnen sich besonders dadurch aus, dass Zustände des Automaten hierarchisch verfeinert werden können, um Automaten in Teilautomaten zu zerlegen. Auf diese Weise lassen sich komplexe Automaten übersichtlicher darstellen. Außerdem können mehrere Teilautomaten gleichzeitig Übergänge ausführen, wodurch parallele Ausführungen einfach modelliert werden können. Zu die-

sen beiden Aspekten zeigen wir im Folgenden je ein Beispiel. Die Statecharts und UML bieten noch eine Reihe notationeller Erweiterungen zu endlichen Automaten, welche die Modellierung komplexer Abläufe vereinfachen, z. B. Ausgabe sowohl nach dem *Mealy*- wie nach dem *Moore-Prinzip*; auf diese Besonderheiten gehen wir hier nicht weiter ein.

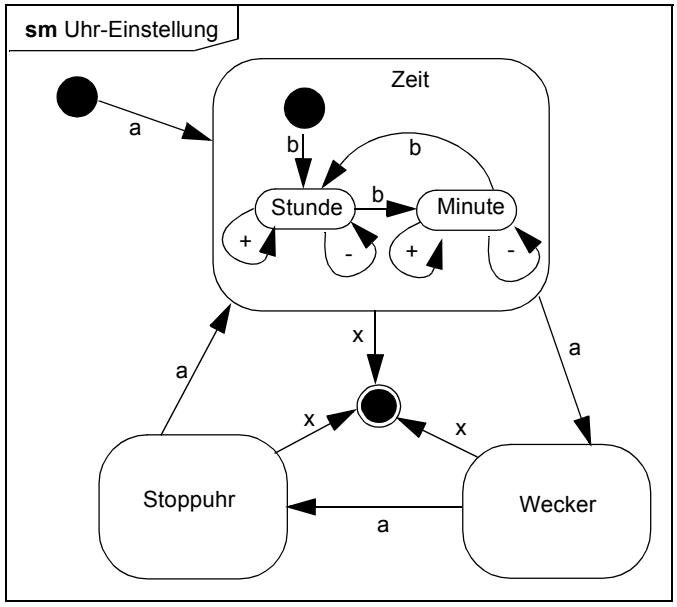


Abbildung 7.16: Hierarchisches Modell der Einstellung einer Uhr

In Abb. 7.16 zeigen wir die hierarchische Modellierung der Bedienung einer Uhr. Sie ist gegliedert in das Einstellen der Zeit, des Weckers und der Stoppuhr, die auf oberster Hierarchieebene durch drei zusammengesetzte Zustände modelliert werden. Der Anfangszustand des gesamten Automaten ist der ausgefüllte Kreis in der linken oberen Ecke, der Endzustand der eingekreiste ausgefüllte Kreis in der Mitte. Durch wiederholtes Betätigen der Taste *a* schaltet man zwischen der Bedienung der drei Funktionen der Uhr um; betätigt man die Taste *x*, wird das Einstellen beendet.

Der Teilautomat für die Zeiteinstellung ist in Abb. 7.16 ausgefüllt. Er enthält einen Anfangszustand, der aktiviert wird, wenn man diesen zusammengesetzten Zustand von außen erreicht. Mit der Taste *b* kann man zwischen dem Einstellen von Stunden und Minuten hin- und herschalten. Der Teilautomat hat keinen Endzustand. Er wird verlassen, wenn man in irgendeinem seiner Zustände durch Betätigen der Taste *a* zum nächsten Teilautomaten wechselt oder mit der Taste *x* das Einstellen beendet.

Jedes solche hierarchische Modell lässt sich in einen normalen endlichen Automaten transformieren. Dazu muss man alle Übergänge, die auf einen zusammengesetzten Zustand führen, auf dessen Anfangszustand zielen. Jeder Übergang, der von einem zusam-

mengesetzten Zustand wegführt, muss ersetzt werden durch entsprechende Übergänge, die von jedem inneren Zustand des Teilautomaten ausgehen. Es ist klar, dass solch ein flaches Modell dasselbe beschreibt, aber wesentlich unübersichtlicher ist.

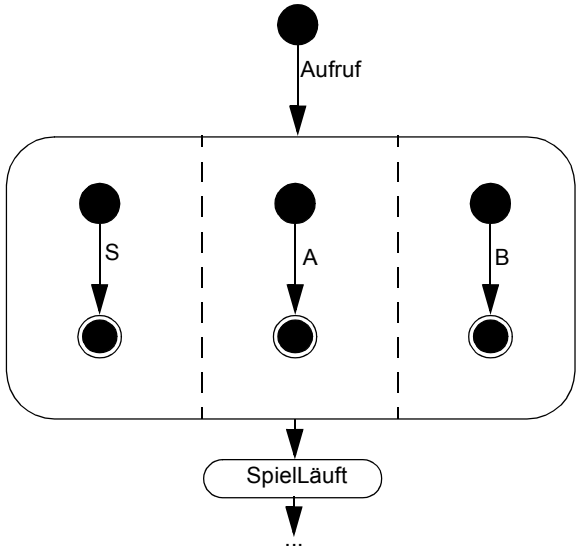


Abbildung 7.17: Statechart modelliert die Ankunft der an einem Tennisspiel Beteiligten

Die Modellierung nebenläufiger Abläufe zeigen wir an der Beschreibung des Beginns eines Tennisspieles. Das Statechart in Abb. 7.17 zeigt im Zentrum einen Zustand, der drei Teilautomaten enthält, die gleichzeitig aktiviert werden. Jeder besteht aus nur einem Übergang, der ausgeführt wird, wenn der Schiedsrichter, der Spieler *A* bzw. der Spieler *B* angekommen sind. Alle drei Anfangszustände der Teilautomaten werden aktiviert, wenn man mit dem Aufruf des Spiels den Übergang zum zusammengesetzten Zustand ausführt. Erst wenn jeder der Teilautomaten seinen Endzustand erreicht hat, wird der Übergang (ohne Eingabe) in den Zustand *SpielLäuft* ausgeführt.

Das Konstrukt der gleichzeitig aktivierten Teilautomaten erlaubt es zu ignorieren, in welcher Reihenfolge die drei Personen auf dem Tennisplatz eintreffen. Dieses Statechart ist aber gleichwertig einem normalen Endlichen Automaten (Abb. 7.18), der alle Reihenfolgen explizit macht, in der die drei Personen eintreffen können.

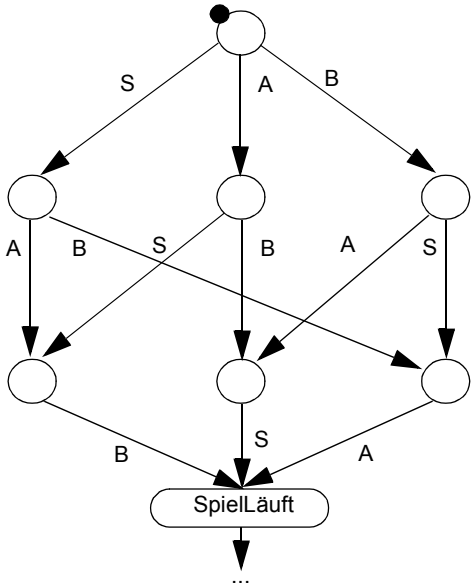


Abbildung 7.18: DEA modelliert die Ankunft der an einem Tennisspiel Beteiligten

7.2 Petri-Netze

Der formale Kalkül der *Petri-Netze* dient zur Modellierung von nebenläufigen Vorgängen, an denen mehrere Prozesse beteiligt sind. Die Modelle beschreiben insbesondere die Interaktionen zwischen den Prozessen und die Effekte, die sich daraus ergeben, dass Operationen prinzipiell gleichzeitig ausgeführt werden können. Typische Beispiele für die Anwendung von Petri-Netzen sind Modellierungen von

- realen oder abstrakten Automaten und Maschinen;
- kommunizierenden Prozessen in der Realität oder in Rechnern;
- Verhalten von Software- oder Hardware-Komponenten;
- Geschäftsabläufen;
- Spielen nach bestimmten Regeln.

Der Kalkül wurde 1962 von C. A. Petri eingeführt. Es sind zahlreiche Varianten und Verfeinerungen daraus abgeleitet worden. Wir befassen uns hier nur mit der Grundform.

Definition 7.13: Petri-Netz

*Ein **Petri-Netz** ist ein Tripel $P = (S, T, F)$ mit einer Menge von **Stellen** S, einer endlichen Menge von **Transitionen** T und einer **Relation** F mit $F \subseteq S \times T \cup T \times S$. P bildet einen bipartiten Graphen mit den Knoten $S \cup T$ und den gerichteten Kanten F.* ■

Die Stellen S repräsentieren *Bedingungen* oder *Zustände* des modellierten Systems. Sie werden in der Grafik durch Kreise dargestellt. Die Transitionen repräsentieren *Zustandsübergänge* oder *Aktivitäten* und werden durch Rechtecke dargestellt.

Definition 7.14: Markierung

*Der Zustand eines Petri-Netzes P wird durch eine **Markierung** angegeben. Das ist eine **Funktion** $M_P: S \rightarrow \mathbb{N}_0$, **die jeder Stelle eine Anzahl von Marken zuordnet.*** ■

Die Marken werden in der Grafik als Punkte in den Stellen dargestellt. Sind die Stellen von *1* an durchnummeriert, so kann man die Markierung als eine Folge von nicht-negativen Zahlen angeben. Wir erläutern die Definition an dem Beispiel in Abb. 7.2 in der Einleitung zu diesem Kapitel. Das Petri-Netz modelliert zwei zyklisch ablaufende Prozesse. Sie werden so synchronisiert, dass sie nicht gleichzeitig die Zustände *2* und *4* einnehmen können. Der links dargestellte Prozess führt immer wieder nacheinander die Operationen aus, die durch die Transitionen *a* und *b* modelliert sind. Vor der Ausführung von *a* befindet er sich in einem Zustand, der durch die Stelle *1* modelliert ist, danach im Zustand *2*. Der zweite Prozess ist mit den Transitionen *c* und *d* und den Stellen *4* und *5* ebenso modelliert. Die Stelle *3* dient der Synchronisation der Prozesse. In Abb. 7.2 sind die Stellen *1*, *3* und *5* markiert. Die Markierung können wir auch als *Markierungsfolge (1, 0, 1, 0, 1)* angeben.

Mit den folgenden Begriffen wird die Dynamik der Petri-Netze eingeführt.

Definition 7.15: Vor- und Nachbereich

Zu einer Transition t in einem Petri-Netz P sind zwei Mengen von Stellen definiert:

Vorbereich (t) $:=$ $\{s \mid (s, t) \in F\}$

Nachbereich (t) $:=$ $\{s \mid (t, s) \in F\}$ ■

Abb. 7.19 zeigt als Beispiel eine Transition t mit ihrem Vorbereich $\{1, 2, 3\}$ und ihrem Nachbereich $\{4, 5\}$. Die Transition s hat den Vorbereich $\{6, 7\}$ und den Nachbereich $\{7, 8\}$.

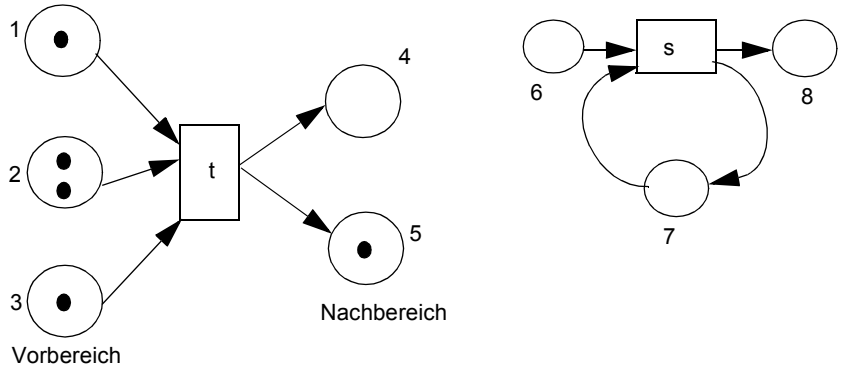


Abbildung 7.19: Vor- und Nachbereich von Transitionen

Petri-Netze verändern ihren Zustand, indem Transitionen schalten und dabei die Markierung verändern.

Definition 7.16: Schaltregel

*Eine **Transition t** eines Petri-Netzes P **kann schalten**, wenn für alle Stellen $s \in$ **Vorbereich(t)** gilt $M(s) \geq 1$. Wenn zu einem Zeitpunkt mehrere Transitionen schalten können, wird eine davon **nicht-deterministisch ausgewählt**; sie schaltet als nächste.*

*Wenn bei einer Markierung M eine Transition t **schaltet**, gilt für die **Nachfolgemarkierung M'**:*

$M'(v) = M(v) - 1$ *für alle* $v \in$ *Vorbereich(t) - Nachbereich(t)*

$M'(n) = M(n) + 1$ *für alle* $n \in$ *Nachbereich(t) - Vorbereich(t)*

$M'(s) = M(s)$ *für alle übrigen Stellen* ■

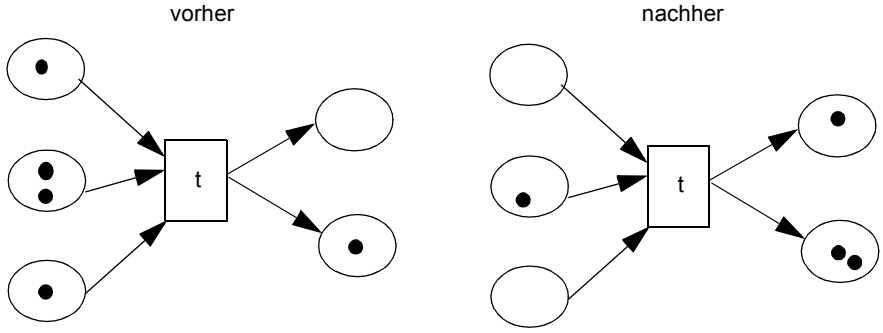


Abbildung 7.20: Schalten einer Transition

Eine Transition entnimmt also beim Schalten je eine Marke von den Stellen ihres Vorbereiches und fügt je eine Marke den Stellen ihres Nachbereiches zu. Abb. 7.20 zeigt Markierungen vor und nach dem Schalten der Stelle *t*. Wenn die Zahl der Stellen im Vorbereich verschieden von der im Nachbereich ist, verändert sich natürlich beim Schalten die Gesamtzahl der Marken.

In einem Petri-Netz schaltet gemäß Definition 7.16 in jedem Schritt genau eine Transition; d. h. das Modell wird streng sequentiell ausgeführt. Durch die nicht-deterministische Auswahl aus der Menge der Transitionen, die schalten können, sind aber ebenso viele unterschiedliche Abläufe möglich wie in dem modellierten parallelen System. Dieser Vergleich gilt natürlich nur für die modellierte Granularität: Transitionen modellieren Operationen als atomar.

Die Schaltregel können wir nun auf unser einführendes Beispiel aus Abb. 7.2 anwenden. Bei der angegebenen Markierung *(1, 0, 1, 0, 1)* können die Transitionen *a* und *c* schalten. Wenn wir *a* schalten lassen, ergibt sich als Nachfolgemarkierung *(0, 1, 0, 0, 1)*. In diesem Zustand ist die Stelle *2* markiert. Die Transition *c* kann nicht schalten, da die Stelle *3* nicht markiert ist. Erst nachdem die Transition b geschaltet hat, erreichen wir wieder die Markierung *(1, 0, 1, 0, 1)*, in der *a* und *c* schalten können.

Die Stelle *3* dient in diesem Netz zur Synchronisation der Prozesse: Sie gibt höchstens einem der beiden Prozesse das Recht, den durch die Stellen *2* bzw. *4* modellierten Zustand einzunehmen. Solche Zustände von Prozessen werden auch *kritische Abschnitte* genannt, die unter *gegenseitigem Ausschluss* ausgeführt werden müssen. Das Petri-Netz aus Abb. 7.2 ist ein typisches Muster für solch eine Synchronisation.

In dem Petri-Netz aus Abb. 7.2 konkurrieren die Transitionen *a* und *c* um die Marke auf der Stelle *3*. Solch eine Situation charakterisieren wir in folgender Definition:

Definition 7.17: Konflikt

Zwei Transitionen s und t ***stehen im Konflikt****, wenn*
Vorbereich (s) $\cap$ *Vorbereich (t)* $\neq \emptyset$*.* ■

Wenn alle Stellen in den Vorbereichen von *s* und *t* markiert sind, aber die Stellen im Vorbereich beider Transitionen nur eine Marke haben, dann können zwar *s* und *t* schalten; aber nachdem eine geschaltet hat, kann die andere im Allgemeinen nicht mehr schalten. In Abb. 7.2 stehen *a* und *c* in solch einem Konflikt. Er bewirkt, dass bei der Markierung *(1, 0, 1, 0, 1)* nicht beide Transitionen *a* und *c* unmittelbar nacheinander schalten.

Das Schaltverhalten und die durchlaufenen Markierungen eines Petri-Netzes kann man durch einen Graphen beschreiben:

Definition 7.18: Markierungsgraph

Zu einem Petri-Netz P und einer Anfangsmarkierung M_0 *kann man einen gerichteten* ***Markierungsgraphen*** *angeben: Seine Knoten repräsentieren jeweils eine Markierung. Der Graph hat eine Kante* $x \xrightarrow{t} y$*, wenn x von der Anfangs-*

markierung erreicht werden, t in x schalten kann und damit x in y überführt wird. ■

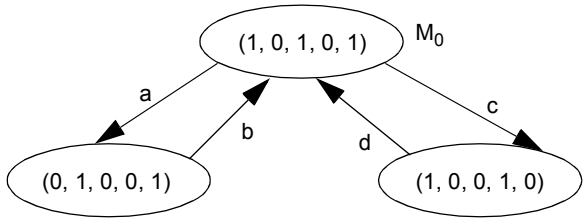


Abbildung 7.21: Markierungsgraphen für das Petri-Netz in Abb. 7.2

Abb. 7.21 gibt den Markierungsgraphen für das Petri-Netz in Abb. 7.2 an. Da das Petri-Netz nur endlich viele Markierungen hat, ist der Markierungsgraph endlich.

In einem Petri-Netz kann man eine Folge durchgeführter Schaltschritte entweder durch die Folge der dabei durchlaufenen Markierungen beschreiben oder durch die Folge der Transitionen, die geschaltet haben; im Petri-Netz von Abb. 7.2 zum Beispiel

(1, 0, 1, 0, 1)	a
(0, 1, 0, 0, 1)	b
(1, 0, 1, 0, 1)	c
(1, 0, 0, 1, 0)	d
(1, 0, 1, 0, 1)	

Damit bietet sich auch an, die *Menge der Schaltfolgen zu einem Petri-Netz* als eine Sprache aufzufassen. Im Petri-Netz von Abb. 7.2 führen nur Schaltfolgen der Form $(a\ b \mid c\ d)^*$ von der Anfangsmarkierung wieder in die Anfangsmarkierung.

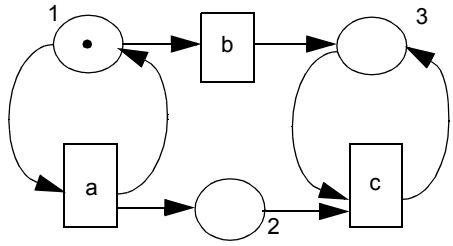


Abbildung 7.22: Petri-Netz definiert die Sprache $a^n\ b\ c^n$

Betrachten wir nun das Petri-Netz in Abb. 7.22 mit der dort angegebenen Anfangsmarkierung, dann erkennen wir, dass anfangs die Transition *a* beliebig oft schalten kann, z. B. *n* mal, und dabei *n* Marken auf der Stelle *2* platziert. *a* steht im Konflikt mit *b*. Wenn *b* geschaltet hat, kann nur noch *c* schalten, und zwar auch *n* mal. Alle Schaltfolgen dieses Petri-Netzes, die keine Nachfolgemarkierung haben, haben also die Form $a^n\ b\ c^u$. An diesem Beispiel erkennt man, dass Petri-Netze im Gegensatz zu endlichen Automaten unbe-

grenzt weit zählen, Zahlenwerte speichern und damit wieder eine entsprechende Anzahl von Schaltvorgängen initiieren können. Durch wiederholtes Schalten können beliebig viele Marken auf Stellen gesammelt und wieder abgezogen werden.

Unser nächstes Beispiel in Abb. 7.23 zeigt, wie man logische Aussagen über den Zustand eines dynamischen Systems modelliert. Das System ist eine sehr einfache Ampelkreuzung mit je einer Ampel für die beiden Fahrtrichtungen Nord-Süd und Ost-West. Jede Ampel wird durch einen zyklischen Prozess gesteuert (wie im Beispiel von Abb. 7.2). Er kann einen von zwei Zuständen annehmen: Die Ampel ist *rot* oder die Ampel ist *grün*. (Die gelbe Lampe realer Ampeln haben wir zur Vereinfachung des Modells eingespart.) Die Aussage *Die Ampel ist rot* gilt genau dann, wenn eine Marke auf der Stelle *rot* liegt; entsprechend für die Stelle *grün*.

Die Transitionen zwischen den Stellen ändern durch Schalten den Zustand von *rot* auf *grün* und umgekehrt. Wir benötigen eine Synchronisation zwischen den beiden Prozessen, damit nicht für beide Fahrtrichtungen gleichzeitig grün oder gleichzeitig rot angezeigt wird. Das leisten die beiden Stellen zwischen den Prozessen. Sie agieren als Wechselschalter: Ist die rechte markiert, so kann der rechte Prozess auf *grün* schalten. Wenn er wieder auf *rot* schaltet, wird eine Marke auf die linke Stelle des Wechselschalters platziert.

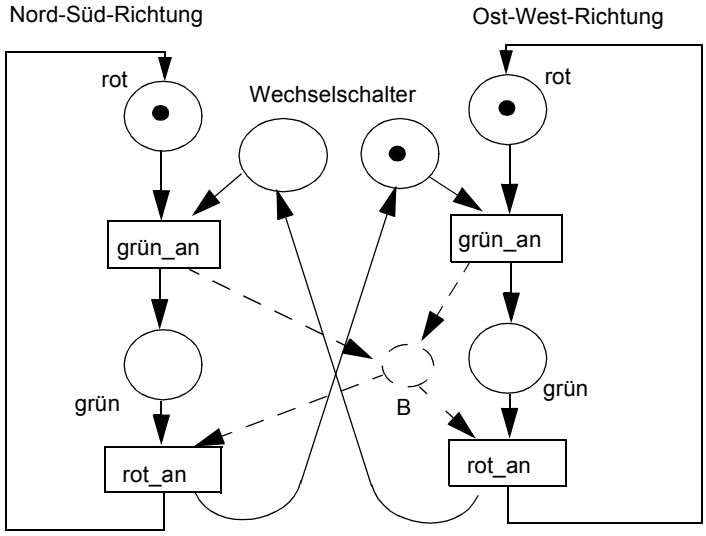


Abbildung 7.23: Modell der Schaltvorgänge einer Kreuzungsampel

Auf diese Weise wird ein strikt sequentieller Ablauf der Prozesse modelliert. Als Gedankenexperiment können wir eine *Beobachtungsstelle B* einfügen. Sie ist so verbunden, dass sie eine Marke erhält, wenn die Ampel einer Fahrtrichtung auf *grün* schaltet, und eine Marke verliert, wenn die Ampel auf *rot* schaltet. Man kann leicht zeigen, dass *B* nie mehr als eine Marke haben kann. Deshalb funktioniert die Ampelschaltung wunschge-

mäß. Wenn wir in einem Petri-Netz Bedingungen durch Stellen modellieren wollen, wie in der Ampelschaltung von Abb. 7.23, dann darf die Zahl der Marken auf einer Stelle *1* nicht überschreiten. Deshalb definiert man:

Definition 7.19: Binär, sicher

*Ein Petri-Netz mit einer Anfangsmarkierung M_0 heißt **binär** bzw. **sicher**, wenn für alle aus M_0 erreichbaren Markierungen M und für alle Stellen s gilt $M(s) \leq 1$.* ■

Als Beispiel bauen wir unsere Ampelschaltung weiter aus. Diesmal modellieren wir in Abb. 7.24 Baustellenampeln an einer wenig befahrenen Straße. Wieder dürfen nicht beide Richtungen gleichzeitig Grün haben. Da ein striktes Wechseln zwischen den Fahrtrichtungen unnötige Wartezeiten verursachen würde, führen wir je einen Sensor für jede Richtung ein, der ein Schalten auf Grün anfordert. Im Petri-Netz der Abb. 7.24 haben wir wieder zwei zyklische Prozesse, die die Ampeln für die beiden Richtungen schalten. Die Anfangsmarkierung repräsentiert die unbefahrene Baustelle: beide Ampeln stehen auf Rot und beide Zufahrten zu den Sensoren sind frei.

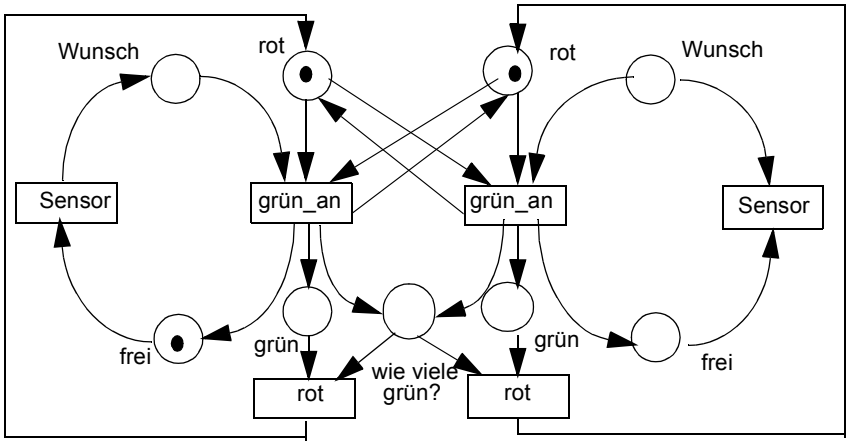


Abbildung 7.24: Ampelschaltung mit Grünanforderung

Betrachten wir nun die Transition grün_an des linken Prozesses. Sie hat drei Stellen im Vorbereich: eigener Wunsch, eigenes rot und anderes rot. Sie müssen markiert sein, d. h. diese Vorbedingungen müssen erfüllt sein, damit die Transition grün_an schalten kann. Sie markiert dann die Stellen im Nachbereich, d. h. erfüllt die Nachbedingungen: eigenes grün, frei, anderes rot und wie viele grün?. Man beachte, dass die Stelle anderes rot sowohl im Vorbereich als auch im Nachbereich der Transition grün_an liegt. Eine Marke auf dieser Stelle ist zwar Vorbedingung damit die Transition schalten kann; sie wird durch das Schalten aber nicht entfernt. Dies ist ein typisches Muster, mit dem die Schaltbarkeit einer Stelle durch Systemzustände kontrolliert wird.

Die mittlere Stelle ist, wie in Abb. 7.23, eine Beobachtungsstelle. Man kann zeigen, dass sie und keine andere Stelle mehr als eine Marke tragen kann. Daher ist das Netz mit der angegebenen Anfangsmarkierung binär bzw. sicher.

Häufig werden mit Petri-Netzen Systeme modelliert, die nicht anhalten sollen, so wie die Ampelschaltungen der Abb. 7.23 und 7.24 und die zyklischen Prozesse aus Abb. 7.2. Die folgenden Definitionen charakterisieren diese Eigenschaft formal.

Definition 7.20: Schwach lebendig

*Ein **Petri-Netz P** mit einer Anfangsmarkierung M_0 heißt **schwach lebendig**, wenn es zu jeder von M_0 erreichbaren Markierung eine Nachfolgemarkierung gibt.* ■

In einem schwach lebendigen Netz gibt es immer eine Transition, die schalten kann. Es ist aber möglich, dass einige Transitionen nie wieder schalten können.

Definition 7.21: Lebendige Transition

*In einem Petri-Netz P mit einer Anfangsmarkierung M_0 heißt eine **Transition t lebendig**, wenn es zu jeder von M_0 erreichbaren Markierung M' eine Markierung M'' gibt, die von M' erreichbar ist und in der t schalten kann.* ■

Definition 7.22: Lebendiges Petri-Netz

*Ein **Petri-Netz** P mit einer Anfangsmarkierung M_0 heißt **lebendig**, wenn alle seine Transitionen lebendig sind.* ■

In einem lebendigen Petri-Netz könnte jede Transition immer wieder mal schalten. Man kann leicht erkennen, dass die Petri-Netze in den Abb. 7.2, 7.23 und 7.24 lebendig und deshalb auch schwach lebendig sind.

Ein Entwurfsfehler im System oder im Modell kann dazu führen, dass ein Petri-Netz anhalten kann, obwohl das nicht vorgesehen ist. Das System hält dann an, weil das Schalten einiger Transitionen zyklisch voneinander abhängt. Solch einen Zustand nennen wir *Verklemmung*.

Definition 7.23: Vor-, Nachbereich von Stellen

Sei $\sigma \subseteq S$ eine Teilmenge der Stellen eines Petri-Netzes. Dann sind

***Vorbereich** $(\sigma) := \{t \mid \exists\, s \in \sigma: (t, s) \in F\}$*

***Nachbereich** $(\sigma) := \{t \mid \exists\, s \in \sigma: (s, t) \in F\}$*

Mengen von Transitionen, die auf Stellen in σ wirken bzw. die Stellen in σ als Vorbedingung haben. ■

Abb. 7.25 zeigt ein Beispiel für den Vor- und Nachbereich einer Menge von Stellen.

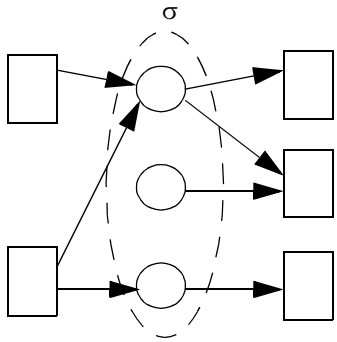


Abbildung 7.25: Vor- und Nachbereich einer Menge von Stellen

Definition 7.24: Verklemmung

Eine nicht-leere Menge σ von Stellen eines Petri-Netzes heißt ***Verklemmung****, wenn gilt Vorbereich (σ) $\subseteq$ Nachbereich (σ).* ■

Wenn in einer Markierung *M* alle Stellen *s* einer Verklemmung nicht markiert sind, kann keine Transition des Nachbereiches schalten, also auch keine des Vorbereiches von σ; also können Stellen aus σ nie wieder markiert werden. Abb. 7.26 zeigt ein Beispiel für solch eine Verklemmung.

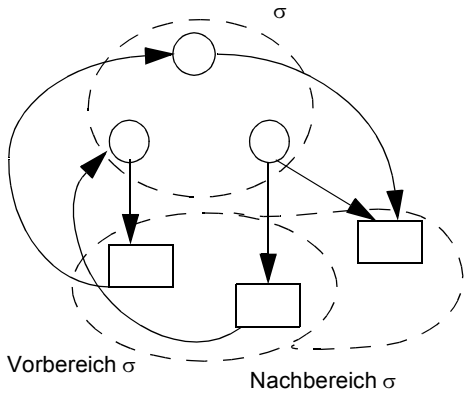


Abbildung 7.26: Beispiel für eine Verklemmung

In Abb. 7.27 zeigen wir ein größeres Beispiel für ein Petri-Netz mit einer Verklemmung. Es modelliert zwei zyklische Prozesse, die beide die gleichen zwei Dateien lesen wollen. Der linke Prozess beginnt mit dem Lesen der Datei *1* beim Schalten der Transition *a*, wodurch die Stelle *3* markiert wird. Schaltet dann die Transition *b* und markiert die Stelle *4*, so liest der Prozess auch die Datei *2*. Das Lesen der Dateien endet mit Schalten der Transition *c*. Der Prozess *2* verhält sich entsprechend; allerdings liest er erst die Datei *2* und

dann auch die Datei *1*. Die Stellen *1* und *2* garantieren, dass jede der beiden Dateien nur jeweils exklusiv von einem der beiden Prozesse gelesen werden. Dafür wird dieselbe Modellierungstechnik wie in dem Petri-Netz aus Abb. 7.2 angewandt.

In diesem Netz können wir folgende Verklemmung identifizieren:

s = {1, 2, 4, 5, 7, 8}
Vorbereich (s) = {b, c, e, f}
Nachbereich (s) = {a, b, e, d, e, f}

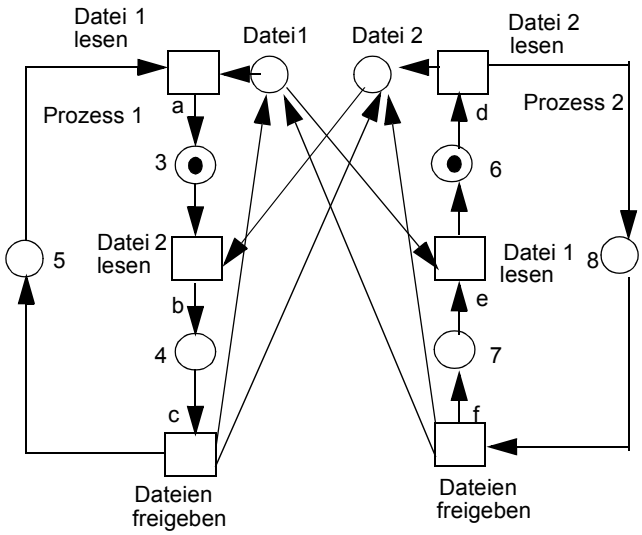


Abbildung 7.27: Verklemmung beim Lesen zweier Dateien

In der Markierung *M* der Abb. 7.27 ist keine Stelle aus σ markiert. Transitionen aus dem Vorbereich (σ) könnten Stellen in σ markieren. Sie liegen jedoch auch alle im Nachbereich (σ) und können deshalb nicht schalten. Diese Verklemmung wird dadurch verursacht, dass die Prozesse die Dateien exklusiv benutzen wollen, sie nacheinander anfordern und in der angegebenen Situation wechselweise aufeinander warten. Wir könnten die Verklemmung beheben, indem wir z. B. beide Prozesse erst die Datei *1* und dann die Datei *2* anfordern lassen.

Zum Schluss wollen wir eine einfache Erweiterung des Petri-Netz-Kalküls einführen: begrenzte *Kapazitäten von Stellen* und *gewichtete Kanten*.

Definition 7.25: Kapazität

In einem Petri-Netz P kann man Stellen begrenzte ***Kapazitäten*** $k \in \mathrm{IN}$ *zuordnen. Wenn eine Stelle s die Kapazität k hat und im Nachbereich einer Transition t liegt, dann kann t nur schalten, falls dadurch die Anzahl der Marken auf s nicht größer als k sein wird.* ■

In Abb. 7.28 hat die Stelle *3* die Kapazität *2*, alle anderen Stellen haben unbegrenzte Kapazität. Bei der angegebenen Markierung kann die Transition *t* nicht schalten, weil dann die Kapazität der Stelle *3* überschritten würde.

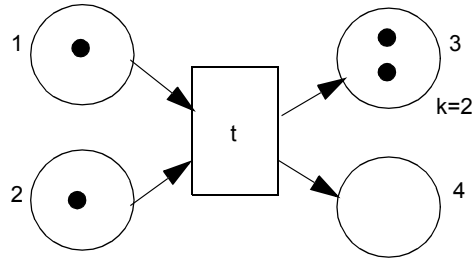


Abbildung 7.28: Stelle mit begrenzter Kapazität

Definition 7.26: Kantengewicht

*In einem Petri-Netz P kann man **Kanten Gewichte** $g \in IN$ **zuordnen**. Wenn eine Kante mit Gewicht g am Schalten einer Transition beteiligt ist, müssen g Marken über diese Kante bewegt werden.* ■

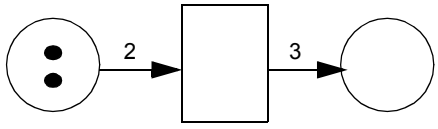


Abbildung 7.29: Kanten mit Gewichten

Die Transition in Abb. 7.29 kann schalten. Wenn sie schaltet, entnimmt sie zwei Marken von der Stelle im Vorbereich und platziert drei Marken auf der Stelle im Nachbereich.

Kapazitäten und Gewichte erlauben es, nicht nur Bedingungen, sondern auch Anzahlen von Aktionen und Elementen übersichtlich zu modellieren. Das Beispiel in Abb. 7.30 modelliert z. B. einen Puffer, dessen Kapazität auf sechs Marken begrenzt ist. Ein zyklischer Produzenten-Prozess legt Elemente im Puffer ab. Die Kante mit dem Gewicht zwei beschreibt, dass der Prozess die Elemente immer paarweise liefert. Er wird blockiert, wenn der Puffer zwei weitere Marken nicht aufnehmen kann. Der Konsumenten-Prozess entnimmt die Marken einzeln aus dem Puffer.

Als abschließendes Beispiel modellieren wir ein *Leser-Schreiber-System*. Das Petri-Netz ist in Abb. 7.31 angegeben. Es hat folgende Eigenschaften: *n* zyklische Leser-Prozesse und *m* zyklische Schreiber-Prozesse operieren auf derselben Datei. Mehrere Leser-Prozesse können die Datei zugleich lesen. Ein Schreiber-Prozess darf nur dann in die Datei schreiben, wenn kein anderer Schreiber- oder Leser-Prozess darauf zugreift.

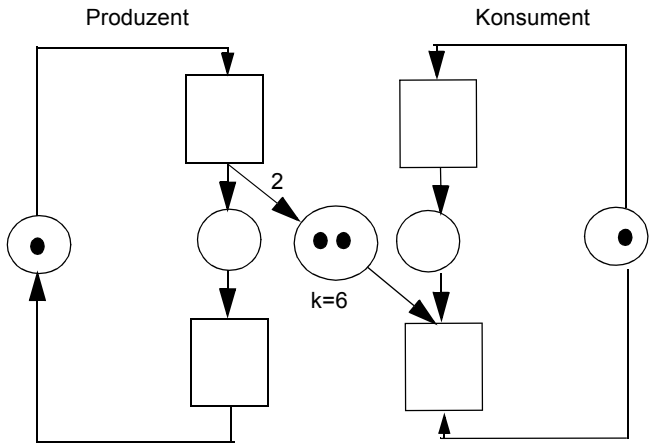


Abbildung 7.30: Begrenzter Puffer zwischen Produzent und Konsument

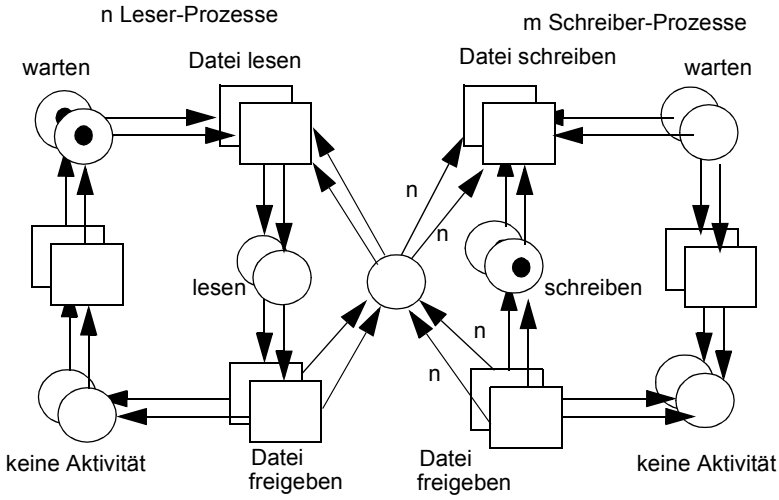


Abbildung 7.31: Leser-Schreiber-System

Im Petri-Netz werden die Prozesse wieder durch eine gemeinsame Stelle synchronisiert. Wenn kein Prozess auf die Datei zugreift, hat sie *n* Marken – soviele, wie Leser im System sind. Ein Leser entnimmt der Stelle eine Marke, wenn er die Datei liest. Es könnten also alle Leser zugleich lesen. Ein Schreiber kann jedoch nur auf die Datei zugreifen, wenn er der Synchronisationsstelle *n* Marken entnehmen kann, d. h. wenn kein anderer Prozess auf die Datei zugreift.

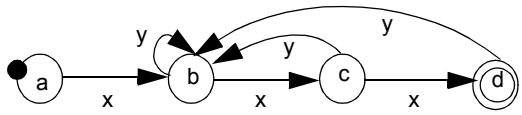
Damit ist das oben beschriebene Systemverhalten wunschgemäß modelliert. Allerdings werden so die Schreiber-Prozesse deutlich benachteiligt: Nur wenn alle Leser gleichzeitig nicht auf die Datei zugreifen, haben sie eine Chance zum Schreiben. Solche Fragen der Fairness kann man an dem Modell diskutieren und durch Modifikation der Synchronisation verändern.

Das Leser-Schreiber-Problem ist ein Synchronisationsmuster, für asymmetrische Zugriffssituationen mit gegenseitigem Ausschluss, das in vielen praktischen Systemen anwendbar ist.

Übungen

7.1 Sprache eines deterministischen endlichen Automaten

Gegeben sei der folgende deterministische endliche Automat A über dem Alphabet $\Sigma = \{x, y\}$.



a) Beschreiben Sie $A = (\Sigma, Q, \delta, q_0, F)$ formal durch die Angabe der *4* Mengen und q_0.

b) Charakterisieren Sie verbal die Zeichenfolgen, die der Automat A akzeptiert.

c) Beschreiben Sie die akzeptierte Sprache $L(A)$ durch einen regulären Ausdruck.

d) Erweitern Sie die Übergangsfunktion δ zu einer totalen Funktion, sodass der Automat vollständig ist, wobei die von dem Automaten akzeptierte Sprache $L(A)$ sich aber nicht ändert.

7.2 Entwurf eines Automaten für eine gegebene Sprache

Von einem Computervirus ist bekannt, dass in befallenen Dateien eine der folgenden Bitfolgen auftreten: 101 oder 111.

a) Modellieren Sie potenziell befallene Dateien durch einen regulären Ausdruck.

b) Geben Sie einen nicht-deterministischen endlichen Automaten an, der potenziell befallene Dateien erkennt (d. h. akzeptiert).

c) Konstruieren Sie einen deterministischen endlichen Automaten, der potenziell befallene Dateien erkennt. Benutzen Sie das Verfahren aus der Vorlesung. (Anmerkung: Der deterministische Automat wird zur Implementierung eines effizienten Virenscanners benötigt.)

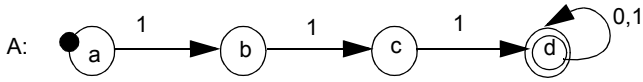
7.3 Fernbedienung durch endlichen Automaten modellieren

Mit einer Multifunktionsfernbedienung kann man verschiedene Geräte steuern. Als Geräte sollen bei dieser Aufgabe nur ein Fernseher und eine Sat-Anlage bedient werden. Es gibt auf der Fernbedienung eine Umschalt-Taste, mit der man in verschiedene Modi ab-

wechselnd umschalten kann, hier jeweils für das Bedienen von TV oder Sat. Je nachdem, in welchem Modus man sich befindet, funktionieren die ON-OFF-Taste und Lautstärke-Taste für das entsprechende Gerät. Das heißt, sie sendet das entsprechende Signal zum passenden Gerät. Modellieren Sie das obige Verhalten der Fernbedienung mit einem Mealy-Automaten. Geben Sie dabei die formale Beschreibung des Automaten, also A = (Σ, *Q*, δ, q_0, F) und das Ausgabealphabet T an.

7.4 Endlichen Automaten erweitern

Gegeben sei ein endlicher Automat *A* über dem Alphabet Σ = {0, 1}:



a) Geben Sie die durch A akzeptierte Sprache *L(A)* formal in Mengenschreibweise an.

b) Erweitern Sie den Automaten *A* zu *B,* sodass *B* nicht-deterministisch alle Zeichenfolgen über Σ akzeptiert, die ein Teilwort 111, also drei aufeinander folgende Buchstaben 1 enthalten.
Beispiele: 010101110010, 000100011110101011.
Hinweis: keine formale Angabe, die Zeichnung reicht aus.

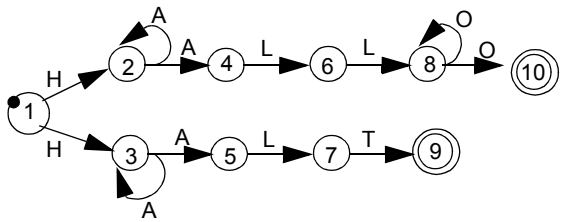
c) Konstruieren Sie zu *B* einen entsprechenden deterministischen, endlichen Automaten *B'* nach dem Ihnen bekannten Verfahren. (Der Automat braucht nur die Übergänge zu enthalten, die für die Wörter der Sprache nötig sind.)

7.5 Geldeinwurf durch endlichen Automaten modellieren

Eine Fahrkarte kostet 3 Euro. Es werden nur 1 Euro und 2 Euro vom Fahrkartenautomaten als Geldeinwurf akzeptiert. Das Geld muss passend eingeworfen werden. Modellieren Sie alle passenden Einwürfe der Geldstücke durch einen deterministischen, endlichen Automaten.

7.6 Endlicher Automat

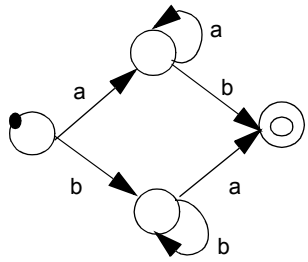
Gegeben sei der folgende nicht-deterministische, endliche Automat *A* über dem Alphabet Σ = *{A, H, L, O, T}*



Beschreiben Sie *A* formal durch die Angaben *A = (*Σ, *Q*, δ, q_0, *F)*

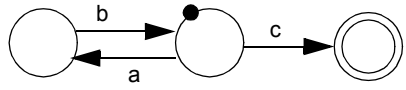
7.7 Sprache eines endlichen Automaten

Welche Sprache akzeptiert folgender Automat *A*? Geben Sie die Sprache durch eine Formel oder einen präzisen Satz an.



7.8 Sprache eines endlichen Automaten

Geben Sie die Sprache des folgenden Automaten als regulären Ausdruck an.



7.9 Endlicher Automat zu gegebener Sprache

Geben Sie einen deterministischen endlichen Automaten an, der die Sprache $(a^+ \mid b^+)\, c$ akzeptiert.

7.10 Statechart mit hierarchischen Zuständen

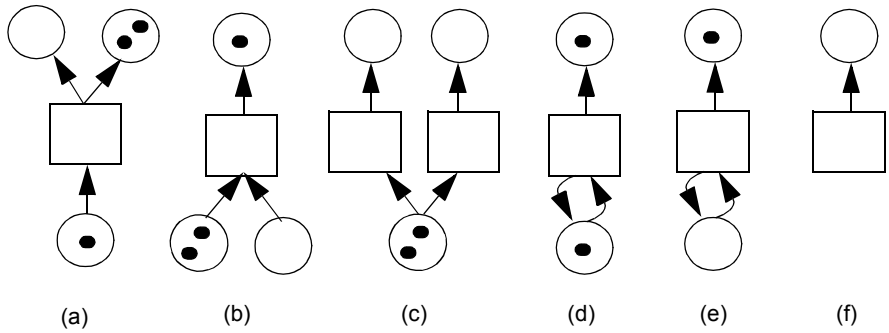
Vervollständigen Sie das Statechart aus Abb. 7.16 durch die Einstellung der Stoppuhr und des Weckers.

7.11 Statechart mit nebenläufigen Teilautomaten

Modellieren Sie folgende Abläufe als Statechart mit nebenläufigen Teilautomaten: Eine Person führt nacheinander die Operationen a, b, c aus; eine andere Person die Operationen A, B, C. Die beiden Operationesfolgen können beliebig verzahnt ablaufen. Modellieren Sie dieselben Abläufe durch einen deterministischen endlichen Automaten.

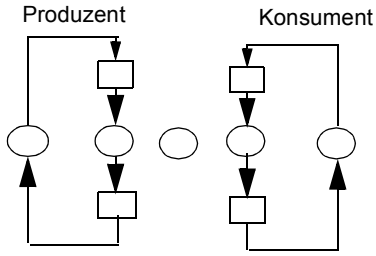
7.12 Schaltregel für Petri-Netze

Geben Sie je eine Nachfolgemarkierung der folgenden Petri-Netze an, falls möglich. Charakterisieren Sie das Verhalten jedes Netzes knapp, aber treffend durch einen Satz.



7.13 Petri-Netz vervollständigen

Sie sollen einen beschränkten Puffer mit folgenden Eigenschaften modellieren: die Puffergröße ist auf *5* beschränkt und der Produzent liefert jeweils immer *2* Einheiten. Ergänzen Sie dazu im folgenden Petri-Netz notwendige Kanten und Beschriftungen und geben Sie eine geeignete Markierung an.



7.14 Petri-Netz

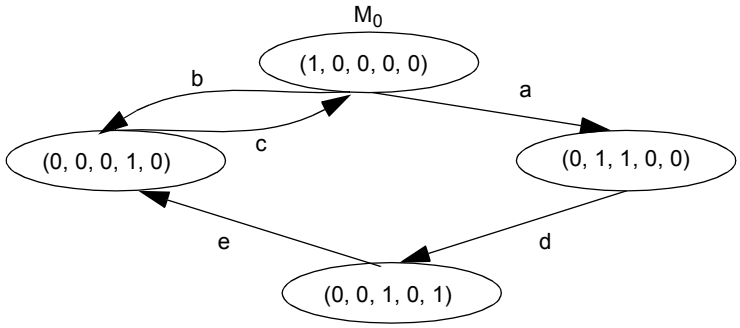
Geben Sie ein kleines Petri-Netz mit einer Markierung an, das folgende Eigenschaft hat: Auf einer seiner Stellen kann die Anzahl der Marken beliebig groß werden.

7.15 Petri-Netz

Skizzieren Sie einen einfachen Ausschnitt aus einem Petri-Netz, in dem zwei Transitionen schalten können, aber nur eine davon schalten wird.

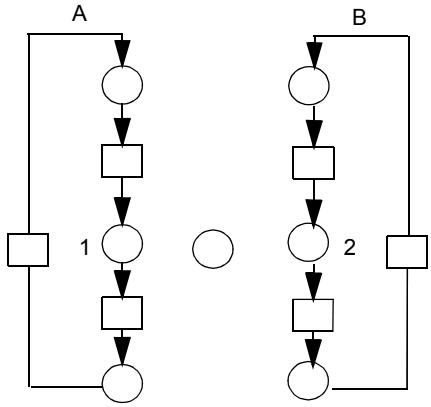
7.16 Markierungsgraph eines Petri-Netzes

Gegeben sei der folgende Markierungsgraph mit der Anfangsmarkierung M_0. Konstruieren Sie daraus ein dazu passendes Petri-Netz.



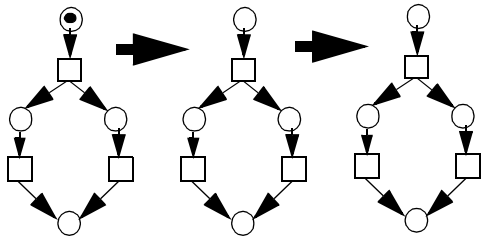
7.17 Petri-Netze vervollständigen

Ergänzen Sie im folgenden Petri-Netz notwendige Kanten, und geben Sie eine Markierung an, sodass die Teilnetze *A* und *B* zyklische Prozesse modellieren und die Stellen *1* und *2* nie gleichzeitig markiert sein können.



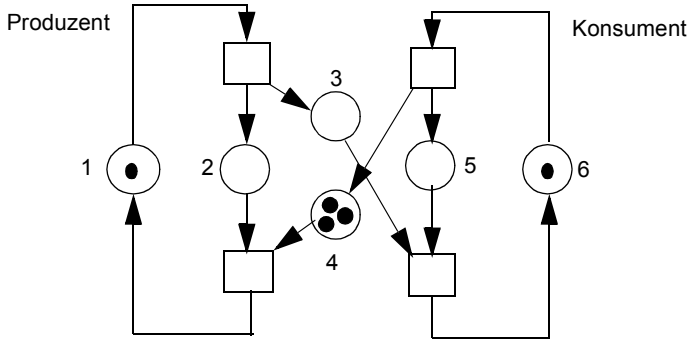
7.18 Folgemarkierungen im Petri-Netz

Geben Sie zwei aufeinander folgende Folgemarkierungen in folgendem Petri-Netz an:



7.19 Petri-Netz für Produzenten und Konsumenten

Das folgende Petri-Netz modelliert das Zusammenwirken eines Produzenten-Prozesses und eines Konsumenten-Prozesses über einen Puffer mit beschränkter Kapazität.



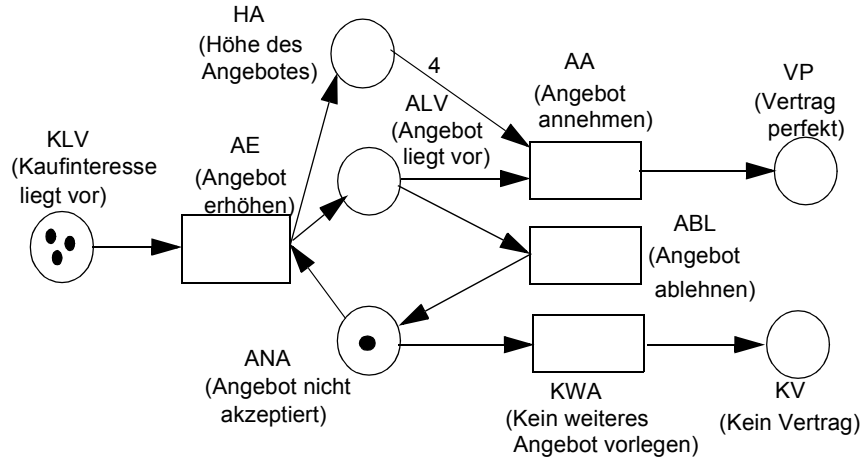
a) Beschreiben Sie die Rollen der Stellen *3* und *4* und der Markierung.

b) Geben Sie eine erreichbare Markierung an, die einen vollen Puffer repräsentiert.

c) Erweitern Sie das Modell um einen weiteren Konsumenten und lassen Sie den Produzenten immer 2 Einheiten zugleich produzieren.

7.20 Petri-Netz modelliert Vertragsverhandlungen

Der Ablauf von Vertragsverhandlungen lässt sich mit Hilfe eines Petri-Netzes darstellen:



a) Geben Sie Vor- und Nachbereich der Transition Angebot erhöhen an.

b) Kann es mit der gegebenen Startmarkierung zu einem Vertrag kommen? Begründen Sie!

7.21 Konflikt im Petri-Netz

Geben Sie einen Ausschnitt eines Petri-Netzes an, in dem zwei Transitionen miteinander in Konflikt stehen.