Endliche Automaten

Beispiele:

- Fahrkartenautomat
- Waschmaschine
- Lichtschalter

Eingabeinformation:

- "Geld einwerfen"
- "Waschprogramm eingeben"
- "Warentaste drücken"
- "Automat liest das Zeichen A"
  $\rightarrow$ Wörter über einem Eingabealphabet

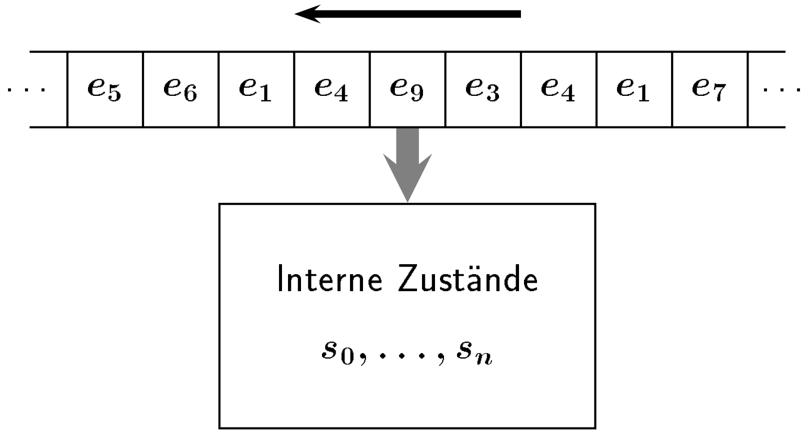
interne Zustände: (Gedächtnis)

- "5 DM eingeworfen"
- "Waschmaschine befindet sich im Vorwaschgang"
- Automat befindet sich zu jedem Zeitpunkt in einem eindeutig bestimmten Zustand
- endlich viele Zustände

(Ausgabeinformation:) Wörter über einem Ausgabealphabet

Endliche Automaten

Funktionsweise: Automat liest pro Zeittakt vom Eingabeband ein Zeichen und verändert seinen internen Zustand in Abhängigkeit von dem gelesenen Zeichen.



Die Bewegungsrichtung des Eingabebandes ist nicht umkehrbar!

Spezifikation der Zustandsänderung:

- Zustandsdiagramm
- Zustandstafel
- Übergangsfunktion

Endliche Automaten

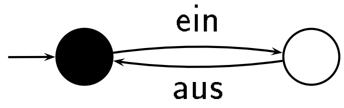
Zustandsdiagramm: gerichteter Graph

Anfangszustand: $\rightarrow (s_0)$

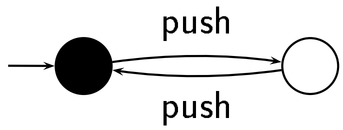
Zustandsübergang: $(s_i) \xrightarrow{e_i} (s_j)$

Beispiel: Lichtschalter

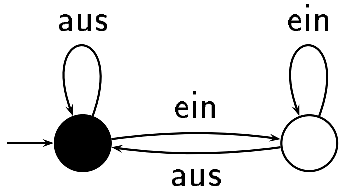
Kippschalter:



Eintastschalter:



Zweitastschalter:



Endzustand: $((s_f))$

λ-Übergang: Zustandübergang, der kein Eingabesymbol konsumiert

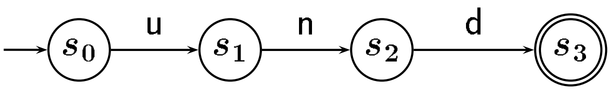
$(s_i) \xrightarrow{\lambda} (s_j)$

Endliche Automaten

Akzeptor: Befindet sich der endliche Automat A nach dem Lesen des letzten Eingabezeichens in einem Endzustand, so hat er das Eingabewort akzeptiert.

Eingabesprache: Die Menge aller von einem Automaten akzeptierten Worte nennt man die von A akzeptierte Sprache $L(A)$.

> Ein endlicher Automat ist ein binärer Klassifikator, der über die Zugehörigkeit eines Eingabewortes zur Sprache des Automaten entscheidet.



Reale Automaten

Jeder technisch realisierbare Automat ist prinzipiell als endlicher Automat darstellbar

- ggf. nur sehr viele Zustände
- Beispiel: Zustandsanzahl eines Computers ist im wesentlichen durch die Größe seines Hauptspeichers bestimmt
  - 1 MByte $= 2^{10} * 2^{10} * 2^3$ Bit Hauptspeicher
  - 128 Zeichen Eingabealphabet (ASCII)

  $\rightarrow$ mehr als $2^{8388608} \approx 10^{2525223}$ Zustände, von denen jeweils 128 Kanten ausgehen

Definition

Endlicher Automat: Ein endlicher Automat ist durch ein Quintupel $A = (E, S, \delta, s_0, F)$ beschrieben, wobei

$E = \{e_1, \ldots, e_r\}$ eine nichtleere Menge, das Eingabealphabet,
$S = \{s_0, \ldots, s_n\}$ eine nichtleere Menge, das Zustandsalphabet,
$\delta = S \times E \to S$, eine totale Funktion, die Überführungsfunktion,
$s_0 \in S$ der Anfangszustand und
$F \subseteq S$ die nichtleere Menge der Endzustände ist.

Zustandsalphabet: Menge von *Namen*

- ☐ Wahl der Namen ist beliebig
- ☐ Konsistente Umbenennung verändert das Resultat nicht

Überführungsfunktion: verschiedene Darstellungsmöglichkeiten: Zustandsdiagramm, Zustandstafel, ...

	e_1	e_2	$\ldots$	e_r
s_0				
s_1				
$\ldots$				
s_n				

Beispiele

Kippschalter:

	ein	aus
Licht aus	Licht an	
Licht an		Licht aus

Eintastschalter:

	push
Licht aus	Licht an
Licht an	Licht aus

Zweitastschalter:

	ein	aus
Licht aus	Licht an	Licht aus
Licht an	Licht an	Licht aus

"und"-Akzeptor:

	u	n	d
s_0	s_1		
s_1		s_2	
s_2			$\boxed{s_3}$

	u	n	d
s_0	s_1	s_0	s_0
s_1	s_0	s_2	s_0
s_2	s_0	s_0	$\boxed{s_3}$

Reguläre Ausdrücke – Endliche Automaten

> Für jeden regulären Ausdruck R existiert ein äquivalenter endlicher Automat A, der die gleiche Sprache L beschreibt, d.h. $L(A) = L(R)$.

Einzeichenausdruck: Einem *Einzeichenausdruck* in R entspricht ein *Zustandübergang* in A.

EA	$s_i \longrightarrow s_j$

Zeichen : Einem *Zeichen* z in R entspricht ein Zustandsübergang in A, der z konsumiert.

a	$s_i \xrightarrow{\text{a}} s_j$

Universelles Zeichen: Einem *universellen Zeichen* in R entspricht eine Schar paralleler Zustandsübergänge in A, wobei jeder Übergang jeweils ein Zeichen aus dem Eingabealphabet konsumiert.

.	$s_i \xrightarrow{\text{a}/\ldots/\text{Z}} s_j$

Reguläre Ausdrücke – Endliche Automaten

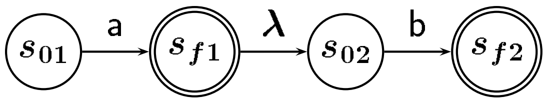
Zeichenauswahl: ...

negative Zeichenauswahl: ...

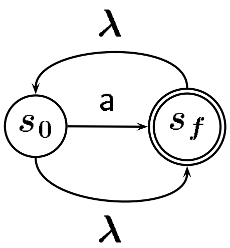
Bereichsauswahl: ...

Klammerung: ...

Verkettung: Eine *Verkettung* zweier Regulärer Ausdrücke $\boldsymbol{R_1 R_2}$ entspricht einer Verkopplung der beiden äquivalenten Endlichen Automaten $\boldsymbol{A_1}$ und $\boldsymbol{A_2}$, so daß jeder Endzustand von $\boldsymbol{A_1}$ mit dem Anfangszustand von $\boldsymbol{A_2}$ durch einen $\boldsymbol{\lambda}$-Übergang verbunden wird.

ab	

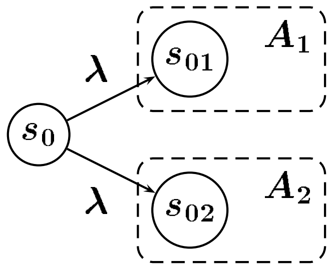
Wiederholung: Eine *Wiederholung* eines Regulären Ausdrucks $\boldsymbol{R}$ entspricht einem äquivalenten Endlichen Automaten, dessen Anfangsszustand mit allen Endzuständen durch $\boldsymbol{\lambda}$-Übergänge (in beiden Richtungen) miteinander verbunden wurden.

a*	

Eingeschränkte Wiederholung: ...

Reguläre Ausdrücke – Endliche Automaten

Alternative: Eine *Alternative* zweier regulärer Ausdrücke $R_1|R_2$ entspricht der parallelen Verkopplung zweier äquivalenter endlicher Automaten A_1 und A_2, so daß der gemeinsame Startzustand mit den Startzuständen von A_1 und A_2 durch λ-Übergänge verbunden ist.

<table>
<tr>
<td>$R_1|R_2$</td>
<td>


</td>
</tr>
</table>

Zusammenfassung:

> Für jeden endlichen Automaten A existiert ein äquivalenter regulärer Ausdruck R, der die gleiche Sprache beschreibt, d.h. $L(R) = L(A)$.

> Reguläre Ausdrücke und endliche Automaten
>
> sind erzeugungsadäquat
>
> besitzen die gleiche generative Kapazität
>
> sind gleich mächtig.

Reguläre Sprache: Sprache, die von einem endlichen Automaten akzeptiert wird.

Deklarativität

Dualität:

Konsumtion	–	Produktion
Akzeptor	–	Generator
Analyse	–	Generierung

Abstraktion von Verarbeitungsdetails: Verarbeitungsrichtung

> Einen Formalismus, der einen Verarbeitungszusammenhang unabhängig von der Verarbeitungsrichtung beschreibt, nennen wir *deklarativ*.

- Viele Formalismen sind nichtdeklarativ (imperative Programmiersprachen, Funktionen, ...).
- Viele Probleme eignen sich nicht für eine deklarative Beschreibung (numerische Näherungsverfahren, ...).

Deklarativität

Vorteile deklarativer Formalismen:

- Deklarative Beschreibungen sind abstrakter und spiegeln das zugrundeliegende Problem oftmals besser wieder.
- Deklarative Formalismen erlauben die Definition und Implementation (monotoner) algebraischer Operationen zur kompositionellen Beschreibung komplexer Modelle.

  Verkettung: $AB \Leftrightarrow L(A) \times L(B)$

  Vereinigung: $A \cup B \Leftrightarrow L(A) \cup L(B)$

  Durchschnitt: $A \cap B \Leftrightarrow L(A) \cup L(B)$

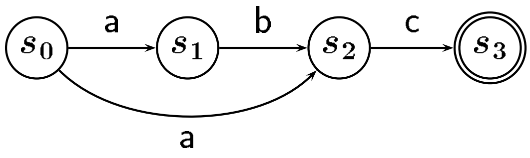
  Differenz: $A \setminus B \Leftrightarrow L(A) \setminus L(B)$

  Jeder algebraische Verknüpfung zweier endlicher Automaten ergibt wieder einen endlichen Automaten.

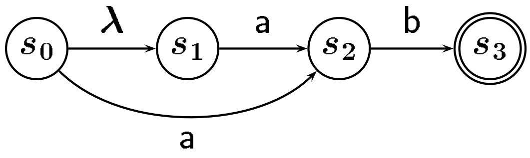
Nichtdeterministische endliche Automaten

Deterministischer endlicher Automat: Folgezustand ist eindeutig festgelegt.

Nichtdeterministischer endlicher Automat: Folgezustand kann aus einer Menge von Alternativen ausgewählt werden.
$\rightarrow$ lokal ist nicht mehr eindeutig bestimmbar, welcher Folgezustand als nächster zu wählen ist.



> Jeder endliche Automat mit $\boldsymbol{\lambda}$-Übergängen ist ein nichtdeterministischer endlicher Automat



> Zu jedem nichtdeterministischen endlichen Automaten $\boldsymbol{A_n}$ existiert ein deterministischer endlicher Automat $\boldsymbol{A_d}$ und umgekehrt, mit $\boldsymbol{L(A_n) = L(A_d)}$.

Reguläre Grammatiken

Terminalsymbole: T (Eingabealphabet)

Nichtterminalsymbole: N (Zustände)

$$N \cap T = \emptyset$$

Ersetzungsregeln/Produktionen: Menge P von Regeln der Art

lhs $\rightarrow$ rhs mit lhs, rhs $\in (T \cup N)^*$

Startsymbol: S $\in N$

CHOMSKY-**Grammatik:** G = (N, T, P, S)

(Rechts-)Reguläre Grammatik: Regeln der Art

lrs $\rightarrow \boldsymbol{\lambda}$
lrs $\rightarrow$ anfang rest mit lrs, rest $\in N$, anfang $\in T$
lrs $\rightarrow$ anfang

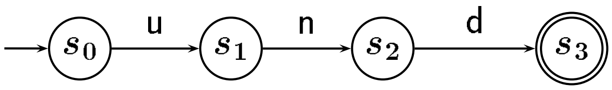
Reguläre Grammatiken

Generatives Modell:

1. Beginne mit einer Symbolfolge, die nur das Startsymbol der Grammatik enthält.
2. Ersetze ein beliebiges Nichtterminalsymbol in der Symbolfolge durch die rechte Seite einer beliebigen Regel, in der das Symbol als linke Regelseite auftritt.
3. Wiederhole 2. solange bis die Symbolfolge keine Nichtterminalsymbole mehr enthält.

Beispiel:

Endlicher Automat:



Reguläre Grammatik: $\boldsymbol{N} = \{\mathsf{s}_0,\mathsf{s}_1,\mathsf{s}_2,\mathsf{s}_3\}$, $\boldsymbol{T} = \{\mathsf{d},\mathsf{n},\mathsf{u}\}$, $\mathsf{S} = \mathsf{s}_0$,

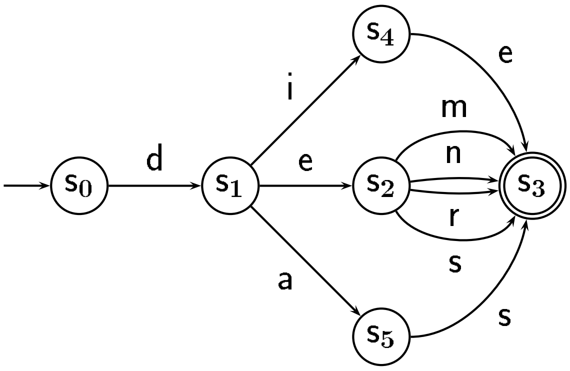
$$\begin{aligned} \boldsymbol{P} = \{\ & \mathsf{s}_0 \rightarrow \mathsf{u}\ \mathsf{s}_1, \\ & \mathsf{s}_1 \rightarrow \mathsf{n}\ \mathsf{s}_2, \\ & \mathsf{s}_2 \rightarrow \mathsf{d}\ \} \end{aligned}$$

Beispiel

Regulärer Ausdruck:

d(e[mnrs]|as|ie) (egrep-Syntax)

Endlicher Automat:



	a	d	e	i	m	n	r	s
s_0	s_0	s_1	s_0	s_0	s_0	s_0	s_0	s_0
s_1	s_5	s_0	s_2	s_4	s_0	s_0	s_0	s_0
s_2	s_0	s_0	s_0	s_0	$\boxed{s_3}$	$\boxed{s_3}$	$\boxed{s_3}$	$\boxed{s_3}$
s_4	s_0	s_0	$\boxed{s_3}$	s_0	s_0	s_0	s_0	s_0
s_5	s_0	s_0	s_0	s_0	s_0	s_0	s_0	$\boxed{s_3}$

Reguläre Grammatik:

$\boldsymbol{N} = \{s_0,s_1,s_2,s_3,s_4,s_5\}$, $\boldsymbol{T} = \{a,d,e,i,m,n,r,s\}$, $S = s_0$,

$$\boldsymbol{P} = \{\ s_0 \rightarrow d\ s_1,$$
$$s_1 \rightarrow e\ s_2,$$
$$s_1 \rightarrow i\ s_4,$$
$$s_1 \rightarrow a\ s_5,$$
$$s_2 \rightarrow m,$$
$$s_2 \rightarrow n,$$
$$s_2 \rightarrow r,$$
$$s_2 \rightarrow s,$$
$$s_4 \rightarrow e,$$
$$s_5 \rightarrow s\ \}$$

Konsistente Umbenennung der Nichtterminale verändert die Grammatik nicht (Namen):

$\boldsymbol{N} = \{A,B,C,D,E,F\}$, $\boldsymbol{T} = \{a,d,e,i,m,n,r,s\}$, $S = A$,

$$\boldsymbol{P} = \{\ A \rightarrow d\ s_1,$$
$$B \rightarrow e\ C,$$
$$B \rightarrow i\ D,$$
$$B \rightarrow a\ E,$$
$$C \rightarrow m,$$
$$C \rightarrow n,$$
$$C \rightarrow r,$$
$$C \rightarrow s,$$
$$D \rightarrow e,$$
$$E \rightarrow s\ \}$$

Grenzen endlicher Automaten

- Beschränkung auf reguläre Sprachen

  nichtreguläre Sprachen: Jede Sprache, die

  nicht durch einen regulären Ausdruck beschrieben werden kann.
  nicht durch einen endlichen Automaten akzeptiert werden kann.
  nicht durch eine reguläre Grammatik erzeugt werden kann.

  z.B. Unbeschränkte Klammersprachen: $a^n b^n$

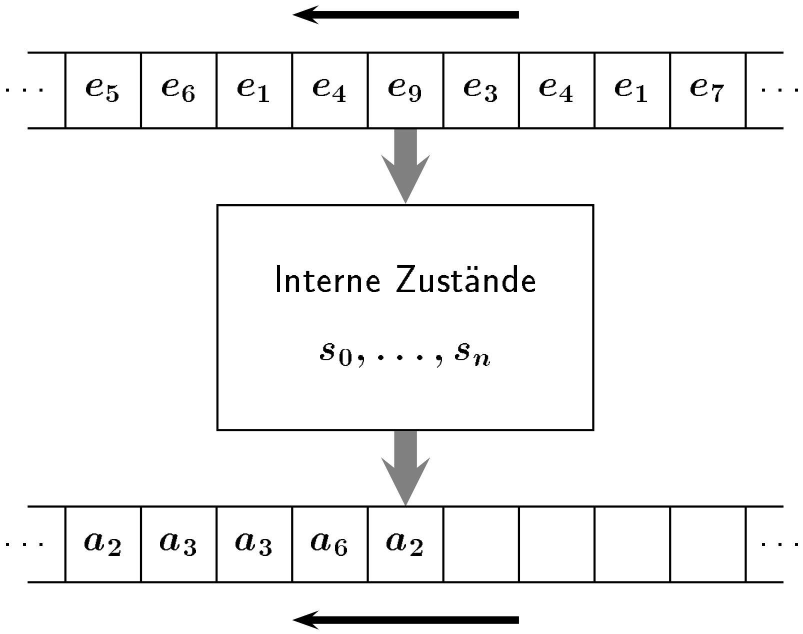
  ab
  aabb
  aaabbb
  aaaabbbb
  ...

  Ursache: Endliche Automaten können nicht beliebig weit zählen.
  Anzahl der Zustände ist immer beschränkt.

- Kann man mit endlichen Automaten rechnen?
  - Was ist Rechnen? (1): Input in Output umwandeln

Endliche Automaten mit Ausgabe

Endlicher Automat mit Ausgabe: Endliche Maschine



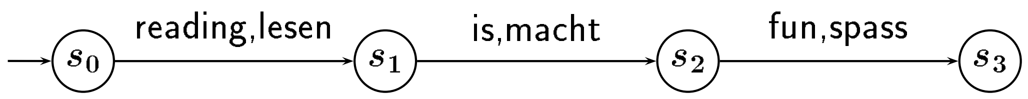
Mealey-**Maschine:** $M = (E, A, S, \delta, \gamma, s_0)$

- **Eingabealphabet:** $E = \{e_1, \ldots, e_r\}$
- **Ausgabealphabet:** $A = \{a_1, \ldots, a_m\}$
- **Zustandsmenge:** $S = \{s_0, \ldots, s_n\}$
- **Überführungsfunktion:** $\delta : S \times E \rightarrow S$
- **Ausgabefunktion:** $\gamma : S \times E \rightarrow A$
- **Anfangszustand:** $s_0 \in S$

- ☐ Ausgabe ist abhängig vom Zustandsübergang
- ☐ Keine explizite Spezifikation von Endezuständen!

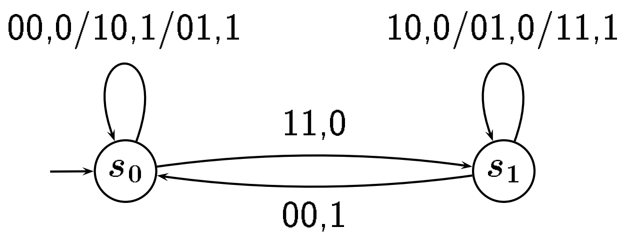
Endliche Automaten mit Ausgabe

Übersetzung zwischen zwei Kodierungen:



Vollständig deklarative Spezifikation der Übersetzungsvorschrift!

Binäraddierer:

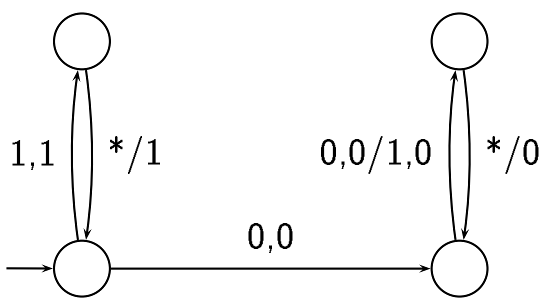


Auswertung arithmetischer Ausdrücke (z.B. Multiplikation)

Operatoren: $\{*\}$

Operanden: $\{0,1\}$

Resultate: $\{0,1\}$

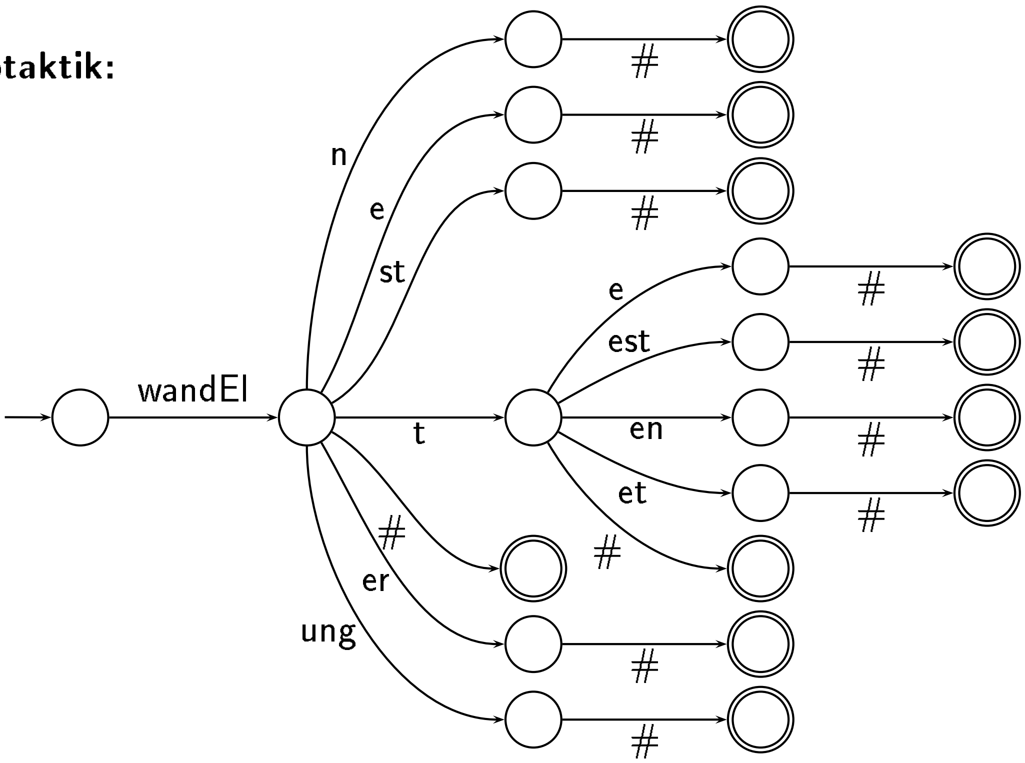


Modellierung morphologischer Prozesse

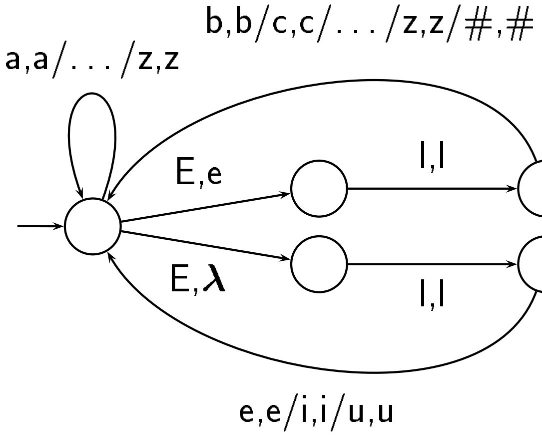
Formenbestand:

wandeln wandle wandelst wandelt
wandelte wandeltest wandelten
Wandel Wandlung Wandler

Morphotaktik:



Morphophonologie:



Erweiterungen

- Umfangreichere Sprachklassen

  → kontextfreie Grammatiken

- Stochastische Modelle

  → MARKOV-Modelle

- Nebenläufigkeit

  → PETRI-Netze

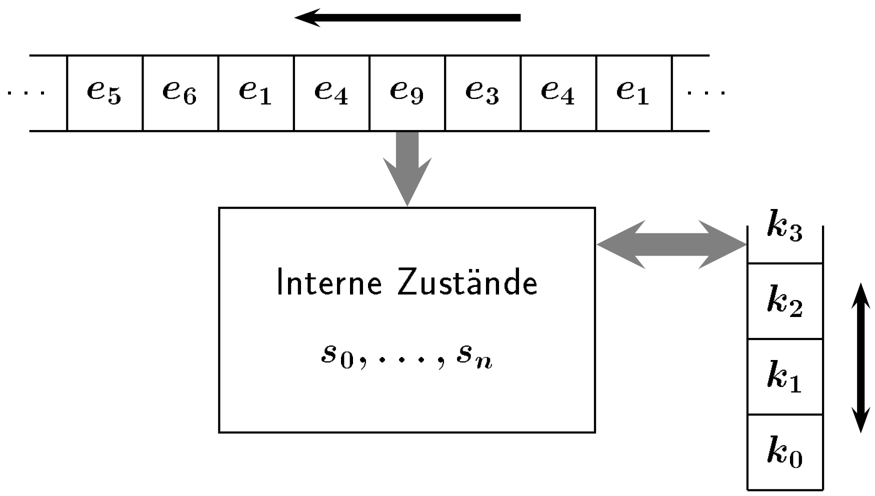
  → Neuronale Netze

Modellierung umfangreicherer Sprachklassen

Reguläre Sprachen → Klammersprachen (kontextfreie Sprachen)

erforderlich: Fähigkeit zum unbeschränkten Zählen

Maschinenmodell: Kellerautomat



Kellerspeicher (Stack)

- ☐ einseitig beschränkter Speicher
- ☐ Zugriff nur auf das oberste Speicherelement (push, pop)

Grammatiktyp: kontextfreie Grammatik

Kontextfreie Grammatiken

Regeln des Typs:

$$\mathsf{a} \rightarrow \mathsf{b}_1 \ldots \mathsf{b}_n \qquad\qquad \mathsf{a} \in \boldsymbol{N},\ \mathsf{b}_i \in \boldsymbol{N} \cup \boldsymbol{T}$$

Zur Namensgebung: Anwendung der Regel kann immer dann erfolgen, wenn das Symbol auf der linken Seite der Regel (a) in der Symbolsequenz vorkommt, d.h. *unabhängig* vom Kontext, in dem a auftritt.

Beispiel: $\boldsymbol{a^n b^n}$

S → ab
S → a S b

Ableitung:

S
a S b
a a S b b
...
a a a a a b b b b b

Kontextfreie Grammatiken

Beispiel: Arithmetische Ausdrücke

Ausdruck → atomarer_Ausdruck
Ausdruck → einfacher_Ausdruck
Ausdruck → geklammerter_Ausdruck

atomarer_Ausdruck → Zahl
atomarer_Ausdruck → Variable

Zahl → Vorzeichen positive_Zahl
Zahl → positive_Zahl
positive_Zahl → Ziffer
positive_Zahl → Ziffer positive_Zahl

Variable → Alphazeichen
Variable → Alphazeichen V_Rest
V_Rest → Zeichen V_Rest

einfacher_Ausdruck → Ausdruck + Ausdruck
einfacher_Ausdruck → Ausdruck – Ausdruck
einfacher_Ausdruck → Ausdruck * Ausdruck
einfacher_Ausdruck → Ausdruck / Ausdruck

geklammerter_Ausdruck → (einfacher_Ausdruck)

Erweiterte Backus-Naur-Form

EBNF: Zusammenfassung kontextfreier Regeln mit identischer linker Seite

	$A \rightarrow B\ C$	A ::= B C.
Alternative	$A \rightarrow B\ C$ $A \rightarrow D$	A ::= B C | D.
Optionalität	$A \rightarrow B\ C$ $A \rightarrow B$	A ::= B [C].
Wiederholung	$A \rightarrow B$ $A \rightarrow B\ C$ $A \rightarrow B\ C\ C$ $\ldots$	A ::= B {C}.
Klammerung	$A \rightarrow B\ B$ $A \rightarrow B\ C$ $A \rightarrow C\ C$ $A \rightarrow C\ B$	A ::= (B | C) (B | C).

Erweiterte Backus-Naur-Form

Beispiel: Arithmetische Ausdrücke, ... die Zweite

Ausdruck	::=	atomarer_Ausdruck | einfacher_Ausdruck
		| geklammerter_Ausdruck.
atomarer_Ausdruck	::=	Zahl | Variable.
Zahl	::=	[Vorzeichen] Ziffer {Ziffer}.
Variable	::=	Alphazeichen {Zeichen}.
einfacher_Ausdruck	::=	Ausdruck (+ | – | * | /) Ausdruck.
geklammerter_Ausdruck	::=	'(' einfacher_Ausdruck ')'.

Beispiel: EBNF in EBNF

Grammatik	::=	{Regel}.
Regel	::=	Nichtterminalsymbol '::=' Körper '.' .
Körper	::=	Ausdruck { '|' Ausdruck}.
Ausdruck	::=	Term {Term}.
Term	::=	Bezeichner | Terminalsymbol | '{' Ausdruck '}'
		| '[' Ausdruck ']' | '(' Ausdruck ')'.

Sprachhierarchie

Sprachklasse	abstraktes Maschinenmodell	Grammatiktyp
allgemeine Sprachen	TURING-Maschine	unbeschränkte Grammatik
kontextsensitive Sprachen	linear beschränkter Automat	kontextsensitive Grammatik
kontextfreie Sprachen	Kellerautomat	kontextfreie Grammatik
reguläre Sprachen	endlicher Automat	reguläre Grammatik

Stochastische Modelle

Stochastische Modellierung: zusätzliche Berücksichtigung der Wahrscheinlichkeit für die Zugehörigkeit eines Wortes zu einer Sprache

- stochastische Automaten
- stochastische Grammatiken
- MARKOV-Ketten

MARKOV-**Modell** (erster Ordnung): Endlicher Automat mit Transitionswahrscheinlichkeiten $p(s_i|s_j)$

MARKOV-**Modelle höherer Ordnung:** $p(s_i|s_j, \ldots, s_k)$

Stochastische Modelle

Beispiele für MARKOV-Ketten unterschiedlicher Ordnung (Deutsch)

- Unigramme (0. Ordnung) $p(s_i)$
  aiobnin*tarsfneonlpiitdregedcoa*ds*e*dbieastnreleeucdkeaitb*
  dnurlarsls*omn*keu**svdleeoieei* ...
- Bigramme (1. Ordnung) $p(s_i|s_j)$
  er*agepteprteiningeit*gerelen*re*unk*ves*mterone*hin*d*an*
  nzerurbom* ...
- Trigramme (2. Ordnung) $p(s_i|s_j s_k)$
  billunten*zugen*die*hin*se*sch*wel*war*gen*man*nicheleblant*
  diertunderstim* ...
- 4-gramme (3. Ordnung) $p(s_i|s_j s_k s_l)$
  eist*des*nich*in*den*plassen*kann*tragen*was*wiese*zufahr* ...

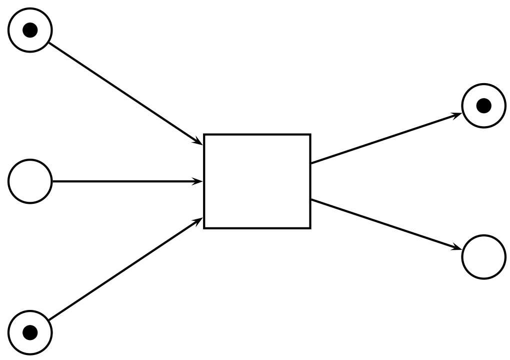
Anwendungen:

- Simulation von Lastverteilungen (Telekommunikation, Hardwaredesign)
- Sprachsignalerkennung
- Korpuslinguistik (Tagging)
- Genomanalyse

PETRI-Netze

Nebenläufigkeit: mehrere Verarbeitungseinheiten arbeiten (zeitweise) parallel

Erweiterung: Bipartiter gerichteter Graph mit Ausbreitungsmechanismus für Marker



Zustände: Können Marker aufnehmen

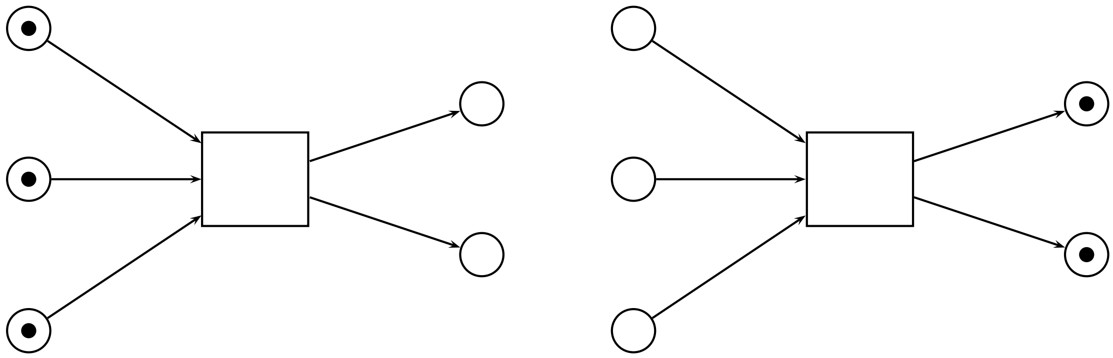
Transitionen: Transportieren Marker durch das Netz ("feuern")

PETRI-**Netze**

Feuern einer Transition

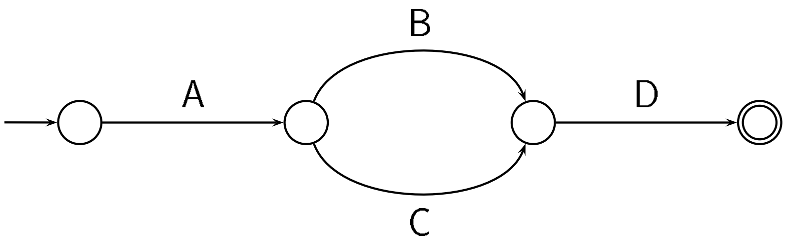
Wenn eine Transition feuert, entfernt sie alle Marker aus ihren Eingabezuständen und belegt alle ihre Ausgabezustände mit Markern.
Eine Transition kann nur dann feuern, wenn

1. alle Eingabezustände mit Markern belegt sind und
2. die Aufnahmekapazität ihrer Ausgabezustände noch nicht überschritten ist.

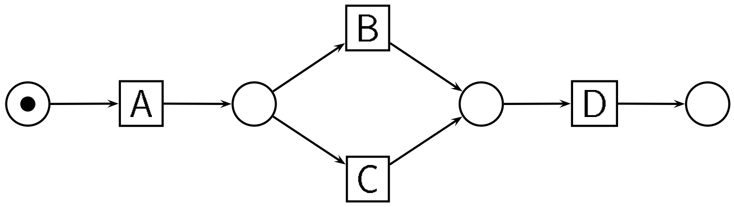


PETRI-Netze

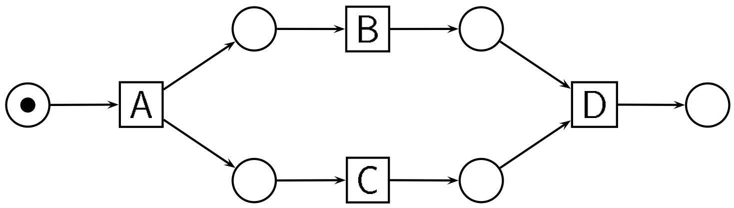
Beispiel: Endlicher Automat mit Alternative



Umformung in ein äquivalentes PETRI-Netz



Modifikation: PETRI-Netz mit Nebenläufigkeit



Konjunktive Verknüpfung von B und C aber keine Festlegung über die zeitliche Abfolge!

Petri-Netze

Anwendungen:

- ☐ Simulation nebenläufiger Rechnerarchitekturen

  Ein-/Ausgabe — Speicherzugriff — Rechenwerk

- ☐ Modellierung menschlicher Handlungsmuster

  Bewegen — Greifen — Loslassen

- ☐ Robotersteuerungen

- ☐ Multimodale Benutzungsschnittstellen

  Zeigen — Sprechen

- ☐ Modellierung zwischenmenschlicher Interaktion

- ☐ Modellierung von Stoffströmen im Umweltmanagement