Formale Sprachen

Teil V: Formale Sprachen

1. Sprachen und Grammatiken 
2. Endliche Automaten 

1. Sprachen und Grammatiken

- Formale Sprachen
- Formale Sprachen vs. gesprochene Sprachen
- Grammatiken
- Die Sequenz
- Typen-Einteilung für Grammatiken nach N-Chomsky
- Chomsky-Hierarchie und Programmiersprachen
- Chomsky-Normalform (CNF) für Typ 2 Sprachen
- Reguläre Ausdrücke
- Alternative Darstellung: BNF/EBNF
- Syntaxdiagramme

Formale Sprachen

- **Ziele:**

  Formulierung von Algorithmen in eindeutiger und für Computer verständlicher Weise.

- **Mittel:**

  Formalismen, die gewisse Ähnlichkeiten mit gesprochenen Sprachen haben, sich aber in Bezug auf Zweckmässigkeit und Eindeutigkeit von gesprochenen Sprachen abgrenzen

Formale Sprachen (1)

- Gesprochene Sprache hat u. a.
  - Formalen Aufbau (Grammatik, d.h. Regeln)
  - Bedeutung (Semantik)

→ auch bei formalen Sprachen

- „kleine“ grammatisch korrekte Unterschiede können zu großen Bedeutungsunterschieden führen; auch jenseits von Gegenteiligkeit

Bsp.: Der Weg ist das Ziel.
Weg ist das Ziel.

→ auch in formalen Sprachen möglich

Formale Sprachen (2)

- Dasselbe Wort, d. h. dieselbe Buchstabenfolge kann in verschiedenen gesprochenen Sprachen vorkommen und dann verschiedene Bedeutungen haben

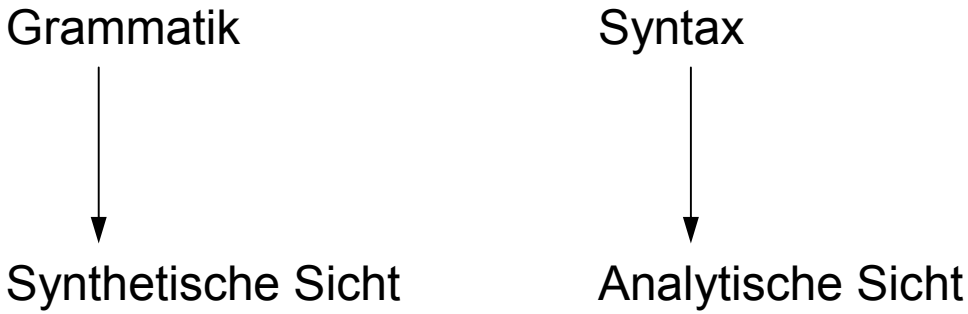
→ Auch in verschiedenen formalen Sprachen zulässig.

Formale Sprachen vs. gesprochene Sprachen

Zeichen aus Alphabet	Buchstaben aus Alphabet
Wörter	Wörter
Ausdrücke, Anweisungen, Wörter, (Sätze)	Sätze

Grammatiken

- Um mit Sprachen, die im Allgemeinen unendliche Objekte sind, algorithmisch umgehen zu können, benötigen wir endliche Beschreibungsmöglichkeiten für Sprachen.
  Dazu dienen sowohl Grammatiken als auch Automaten



Grammatiken (1)

NP = Nominalphrase
VP = Verbalphrase
N = Nomen
A = Artikel
PP = Präpositionalphrase
V = Verb
P = Präposition

Satz	→	NP	VP
NP	→	NP	PP
NP	→	N	
NP	→	A	N
VP	→	V	
VP	→	V	NP
VP	→	VP	PP
PP	→	P	NP

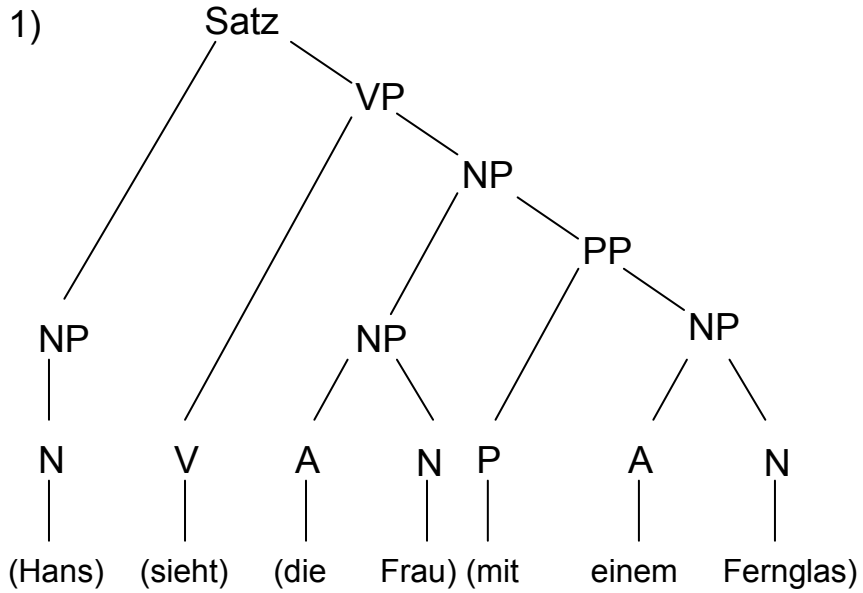
P	→	mit
P	→	in
P	→	auf
N	→	Hans
N	→	Frau
N	→	Fernglas
N	→	Park

V	→	sieht
V	→	geht
A	→	der
A	→	die
A	→	das
A	→	einem

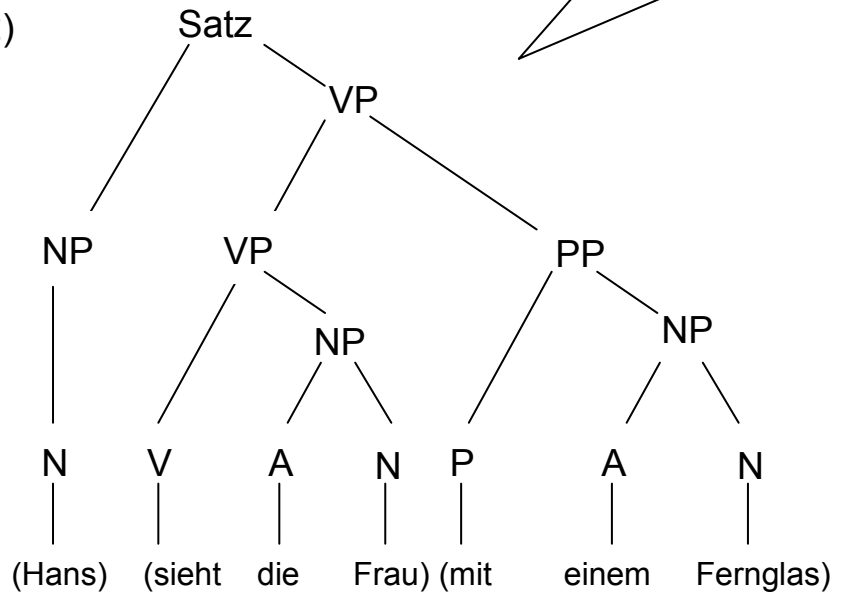
➢ Syntaxbaum zu „Hans sieht die Frau mit einem Fernglas“

Mehrdeutige Grammatik

1)



2)



Grammatiken (2)

- Eine **Grammatik** wird spezifiziert durch 4 Angaben:

$G = (V, \Sigma, P, S)$

V = endliche Menge der **Variablen**
Σ = endliche Menge der **Terminalzeichen**
$S \in V$ = die **Startvariable**
P = endliche Menge der **Regeln** (oder Produktionen)

auch üblich: (V, A, P, S), (N, T, P, S), (S_N, S_T, P, w_S)

- Es gilt: $V \cap \Sigma = \varnothing$
- Regeln „Produktionsregeln“ haben die Form:
  linke Seite → rechte Seite
- linke und rechte Seite können aus **Variablen** (= Nichtterminalzeichen) und **Terminalzeichen** zusammengesetzt sein

Grammatiken (3)

➢ Beispiel:

$V = \{ S \}$
$\Sigma = \{ a, b \}$

lies: „S erzeugt aSb“
oder „aus S folgt aSb“

Regeln 1) $S \rightarrow aSb$
2) $S \rightarrow ab$

$S \Rightarrow ab$
$S \Rightarrow aSb \Rightarrow aabb = a^2b^2$
$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb = a^3b^3$

d. h. Bei dieser Grammatik sind ableitbar
alle Wörter der Form a^nb^n, $n \geq 1$

Grammatiken (4)

- Definition:

  Eine endliche, nicht-leere Menge von Terminalzeichen nennt man auch **Alphabet**.

  Die Elemente eines Alphabets heißen Symbole.

  (Buchstaben, Zeichen, Token)

  Falls Σ ein Alphabet ist, so bezeichnet Σ^* die Menge aller **Worte** bestehend aus Buchstaben $\in \Sigma$.

  (≙ endliche Folgen)

  leeres Wort

  z. B.: $\Sigma^* = \{ \varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, ... \}$

  $\Sigma^+ = \Sigma^* \setminus \{ \varepsilon \}$

Grammatiken (5)

- Beispiel:

  A = { a, b, …, z }; B = { 0, 1 }
  C = { for, end, begin, if, then, else, … }

  (endliche Folgen)

  Durch Hintereinanderschalten entstehen Wörter
  abbcad $\in$ A*; 01101 $\in$ B*; begin if end $\in$ C*;

Grammatiken (6)

Die **Länge** eines Wortes ist die Anzahl seiner Buchstaben.

$abbcda \in \Sigma^*$, mit $|abbcda| = 6$

$|\varepsilon| = 0$

Es gilt:
$|w1\ w2| = |w1| + |w2|$

Für $w \in \Sigma^*$ bezeichnet $|w|$ die Länge von w.

Sei $w \in \Sigma^*$ ein Wort, $n \in \mathbb{N}$

Dann ist $w^n = \underbrace{ww\ldots w}_{\text{n-mal}}$ ein Wort der Länge $|w^n| = n \cdot |w|$

Sei Σ ein Alphabet, dann heißt $L \subseteq \Sigma^*$ eine (formale) **Sprache**

Grammatiken (7)

- Problem:
  Sprachen enthalten i. a. unendlich viele Wörter

- Ziel:
  Endlichen Formalismus angeben, der in der Lage ist, unendlich viele Sprachen zu bezeichnen.

- Beispiel:
  Grammatik aus vorangegangenem Beispiel war **kontextfrei**, d. h. auf der linken Seite der Regeln steht nur eine Variable.

Grammatiken (8)

kontextsensitive

- Beispiel für eine nicht-kontextfreie Grammatik

$V = \{ S, B \}$ S: Startvariable

$\Sigma = \{ a, b, c \}$

Regeln:

1) S → aSBc
2) S → abc
3) cB → Bc
4) bB → bb

Eine mögliche Ableitung eines Wortes $\in \Sigma^*$

$$S \underset{1)}{\Rightarrow} aSBc \underset{2)}{\Rightarrow} aabcBc \underset{3)}{\Rightarrow} aabBcc \underset{4)}{\Rightarrow} \underbrace{aabbcc}_{\in \Sigma^*} = a^2b^2c^2$$

- Bei dieser Grammatik sind ableitbar:

  alle Wörter der Form $a^n b^n c^n$, $n \geq 1$

Grammatiken (9)

- Beispiel

  Für n = 3

$$
\begin{aligned}
S &\Rightarrow a\,S\,B\,c \\
&\Rightarrow aa\,S\,B\,c\,B\,c \\
&\Rightarrow aaabc\,B\,c\,B\,c \\
&\Rightarrow aaab\,B\,cc\,B\,c \\
&\Rightarrow aaab\,B\,c\,B\,cc \\
&\Rightarrow aaab\,B\,B\,ccc \\
&\Rightarrow aaabb\,B\,ccc \\
&\Rightarrow aaabbbccc
\end{aligned}
$$

Grammatiken (10)

- Definition:

  Die von einer Grammatik G = (V, Σ, P, S) erzeugte (definierte) **Sprache** ist $L(G) := \{ w \in \Sigma^* \mid S \Rightarrow \ldots. \Rightarrow w \}$

vgl. Bsp. von oben:
$L(G) = \{ a^n b^n c^n \mid n \geq 1 \}$

$w_1 \Rightarrow w_2$ bedeutet eine Ableitung:

Das Wort w_1 enthält die linke Seite einer Grammatik-Regel. Diese linke Seite wurde in w_2 durch die rechte Seite ersetzt. Eine Ableitung endet, wenn w nur noch Terminalsymbole enthält.

Die Sequenz

- $S \Rightarrow \ldots \Rightarrow w$

  bedeutet, dass es eine Ableitung, d. h. eine endliche Folge von Regelanwendungen gibt, die von S auf w führt.

- Diese Folge ist nicht zwingend und es kann passieren, dass bestimmte ($\rightarrow$ schlechte) Folgen zu gar keiner Sequenz aus Σ^* führen.

- **Beispiel:** $V = \{ S, B \}$, $\Sigma = \{ a, b, c \}$

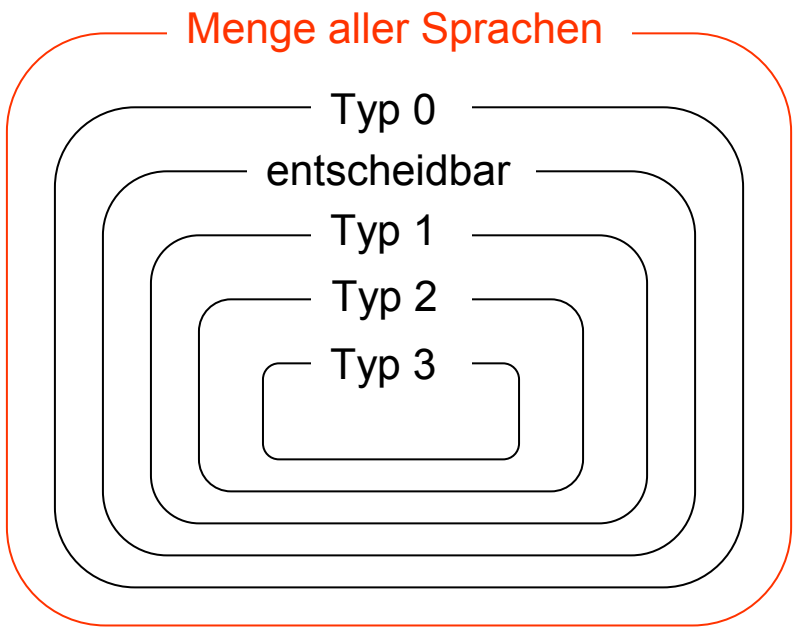
  $S \rightarrow aSBc$
  $S \rightarrow abc$
  $cB \rightarrow Bc$
  $bB \rightarrow bb$
  $aS \rightarrow aB$

  $S \Rightarrow aSBc \Rightarrow aBBc$ [stop]

  d. i. keine Sequenz nur aus Terminalsymbolen

Typen – Einteilung für Grammatiken nach N-Chomsky

- Typ 0: keine Einschränkungen
- Typ 1: (oder **kontextsensitiv**)
  Für alle Regeln $w_1 \rightarrow w_2$
  der Grammatik muss gelten:
  $| w_1 | \leq | w_2 |$
- Typ 2: (oder **kontextfrei**)
  für alle Regeln $w_1 \rightarrow w_2$
  der Grammatik gilt: $w_1 \in V$
- Typ 3: (oder **regulär**)
  wie Typ 2, zusätzlich muss w_2 eine der beiden Bauarten haben $w_2 \in \Sigma$ oder w_2 besteht aus Terminal gefolgt von Variabler



Hinweis:
Eine Sprache $L \subseteq \Sigma^*$
ist Typ $i \in \{ 0, 1, 2, 3 \}$,
falls es Grammatik G
vom Typ i gibt mit
$L = L(G)$

Typen – Einteilung für Grammatiken nach N-Chomsky (1)

- Eine Sprache L ist vom Typ 0, 1, 2, oder 3,
  wenn es eine Grammatik G von entsprechendem Typ gibt,
  die die Sprache festlegt, also L = L(G).
- Also: Sprachtypen sind gleich Grammatiktypen.

Typen – Einteilung für Grammatiken nach N-Chomsky (2)

- Beispiel für reguläre Sprache (Typ 3)

$V = \{ S, A, B \}$

$\Sigma = \{ a, b \}$

$P = \{$ 1. $S \rightarrow bS$,
2. $S \rightarrow aA$,
3. $A \rightarrow bS$,
4. $A \rightarrow aB$,
5. $A \rightarrow a$,
6. $B \rightarrow bB$,
7. $B \rightarrow aB$,
8. $B \rightarrow b$,
9. $B \rightarrow a$ $\}$

Diese Grammatik ist vom Typ 3, denn

1. auf der linken Seite steht bei jeder Regel nur eine Variable.
2. auf der rechten Seite steht nur ein einziges Terminalsymbol (Regeln 5, 8, 9) oder ein Terminal-symbol gefolgt von einer Variablen (Regeln 1, 2, 3, 4, 6, 7).

Typen – Einteilung für Grammatiken nach N-Chomsky (3)

- Welche Sprache wird von vorangegangener Grammatik erzeugt?

z. B.: $S \Rightarrow aA \Rightarrow aaB \Rightarrow aaa$

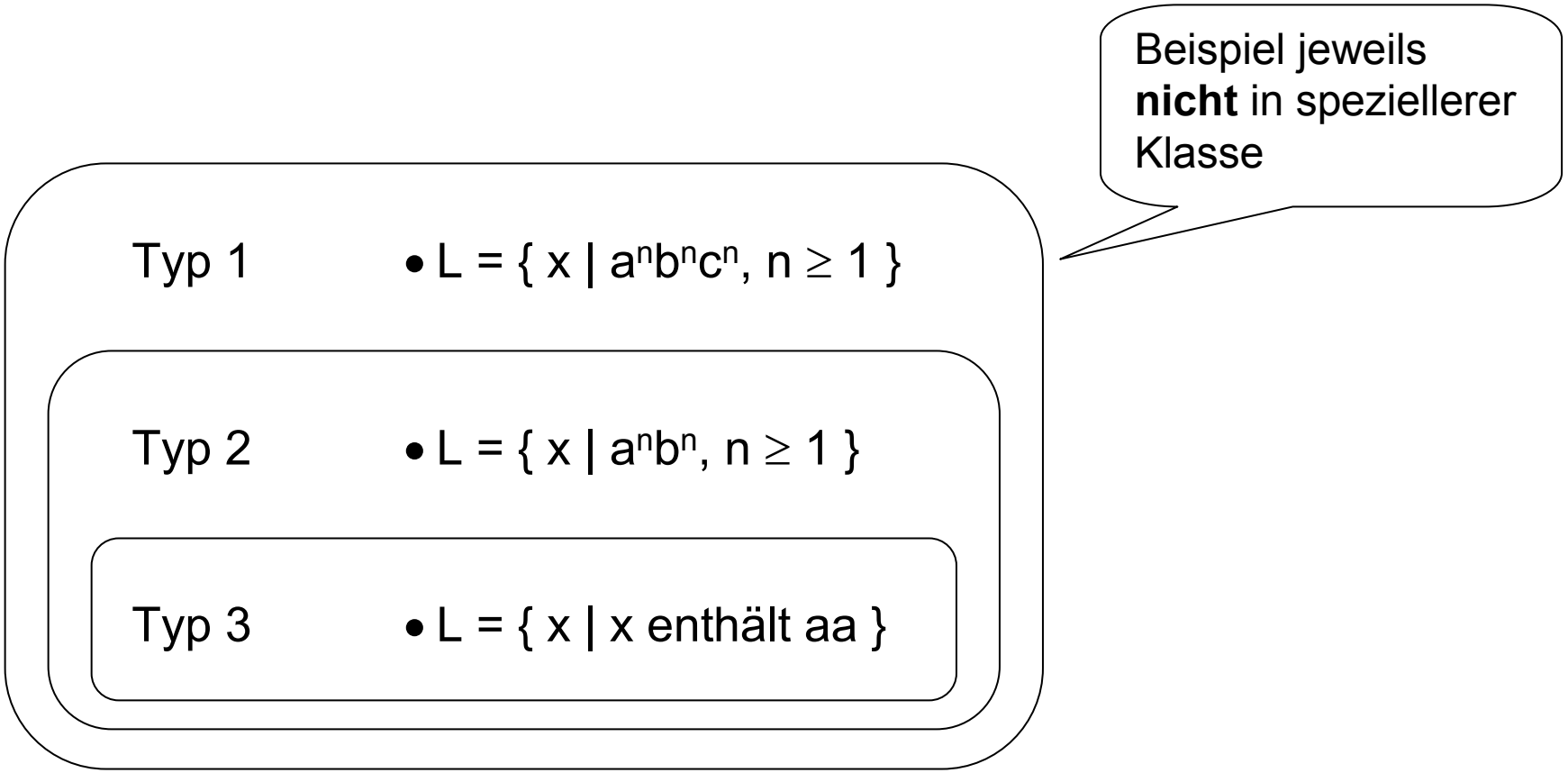
$S \Rightarrow bS \Rightarrow bbS \Rightarrow bbaA \Rightarrow bbaa$

$\vdots$

$L = \{ x \in \{ a, b \}^* \mid \text{in } x \text{ kommt } aa \text{ vor} \}.$

Typen – Einteilung für Grammatiken nach N-Chomsky (4)

➢ Chomsky Hierarchie ist strikt



Sprach“erkennung“

- Es gibt effiziente Algorithmen, die bei einem vorgelegten Wort w und einer Typ 2 Grammatik G feststellen, ob dieses Wort aus G ableitbar ist, d. h. zur Sprache L(G) gehört.
- Für Typ 1 Grammatiken gibt es solche effiziente Algorithmen nicht!

Chomsky Hierarchie und Programmiersprachen

- Heutige Programmiersprachen liegen zwischen Typ 2 (kontextfrei) und Typ 1 (kontextsensitiv).

  D. h. die „meisten“ Anweisungen sind formale Wörter einer kontextfreien Sprache, aber „einige“ Anweisungen sind formale Wörter einer darüber hinaus gehenden Sprache.

Chomsky Hierarchie und Programmiersprachen (1)

- Die Bedeutung von Typ 2 Grammatiken (kontextfrei) motiviert die Analyse ihrer Regeln:

  Während die linke Seite einfach ist (nur eine Variable), darf die rechte Seite beliebig kompliziert sein.
  Dies kann jedoch eingeschränkt werden.

  Die Chomsky-Normalform CNF erlaubt nur die folgenden Typen:

  - $w_1 \rightarrow w_2\, w_3$ $\qquad$ $w_1\, w_2\, w_3 \in V$
  - $w_1 \rightarrow x$ $\qquad$ $x \in \Sigma$

- Jede Typ 2 Grammatik kann (mittels Einführung neuer Variablen) in CNF geschrieben werden.

Chomsky Hierarchie und Programmiersprachen (2)

- Mittels CNF sind Typ 2 Grammatiken nur „wenig“ komplizierter als Typ 3 Grammatiken.

Typ 2		Typ 3	
$w_1 \rightarrow w_2\, w_3$	$w_1, w_2, w_3 \in V$	$w_1 \rightarrow y\, w_3$	$w_1, w_3 \in V, \quad y \in \Sigma$
$w_1 \rightarrow x$	$x \in \Sigma$	$w_1 \rightarrow x$	$x \in \Sigma$

Chomsky-Normalform (CNF) (für Typ 2 Grammatiken)

Grammatik Regeln dürfen nur diese zwei Formen haben:

Variable → Variable Variable

Variable → Terminalzeichen

Jede Typ 2 Grammatik lässt sich so äquivalent umformen, dass eine CNF entsteht

➢Beispiel:

A → a B b D c (nicht CNF)

1. Schritt: neue Variablen X, Y, Z einführen

A → X B Y D Z

X → a

Y → b

Z → c

} in CNF

2. Schritt: neue Variablen U, V, W einführen

A → UV U → XW

V → DZ W → BY

Chomsky-Normalform (CNF) (für Typ 2 Grammatiken) (1)

A → a B b D c

X B Y D Z

A → X W V

A → U V

1. Schritt: „Anheben“

2. Schritt: Bildung von Paaren bis nur zwei Variablen übrig bleiben

Umformung in CNF ist nicht eindeutig!

A → a B b D c

X B Y D Z

A → R S Z

A → R T

X → a
Y → b
Z → c
R → XB
S → YD
T → SZ
A → RT

Chomsky-Normalform (CNF) (für Typ 2 Grammatiken) (2)

- Beispiel: $L = \{ x \in \{ a, b \}^* \mid x = a^n b^n, n \geq 1 \}$ von Typ 2 mit

$S \rightarrow aSb$ $\quad V = \{ S \}$
$S \rightarrow ab$ $\quad \Sigma = \{ a, b \}$

Umformung in CNF:

$S \rightarrow AB$ $\quad V = \{ S, A, B, X \}$
$S \rightarrow AX$ $\quad \Sigma = \{ a, b \}$
$X \rightarrow SB$
$A \rightarrow a$
$B \rightarrow b$

Beispielsweise lässt sich a^3b^3 erzeugen durch:

$$S \Rightarrow AX \Rightarrow ASB \Rightarrow AAXB \Rightarrow AASBB \Rightarrow AAABBB \Rightarrow \ldots \Rightarrow aaabbb.$$

Reguläre Ausdrücke

- Reguläre Sprachen (Typ 3) lassen sich durch reguläre Ausdrücke beschreiben.

- Der reguläre Ausdruck (0|1)*00(0|1)* erzeugt z. B. die Wörter 00, 100, 0100111.

  | : „oder“

  * : keine oder eine oder beliebig viele Wiederholungen

  Hilfszeichen zusätzlich zu “(“ und “)“.

Reguläre Ausdrücke (1)

- b* (ab*ab*ab*)* beschreibt Wörter mit durch 3 teilbarer Anzahl von a‘s, z. B. bbabbaa, ababab, abababababab, …

- 0|1|2|3|4|5|6|7|8|9 (0|1|2|3|4|5|6|7|8|9)*
  beschreibt die natürlichen Zahlen einschließlich 0, 00, 000, …

- 0|1 (0|1)* (+|–|·) 0|1 (0|1)*
  beschreibt Addition oder Substraktion oder Multiplikation von Binärzahlen.

- 0|1 (0|1)* / 1 (0|1)* beschreibt Division von Binärzahlen.

Reguläre Ausdrücke (2)

- (A|T|C|G) (A|T|C|G) (A|T|C|G)

  beschreibt die $4^3 = 64$ möglichen Triplets über dem Alphabet { A, T, C, G }:
  Wörter des genetischen Codes.

Reguläre Ausdrücke (3)

- weitere Beispiele: $\Sigma = \{ 0, 1 \}$

(1) (0|1)* beschreibt { 0, 1 }*

(2) (0|1)* 0 (0|1)* } Menge aller Wörter die mindestens eine 0 enthalten

(3) 1* 0 (0|1)* }

(4) (0|1)* 0 1*

(5) 0* 1 0* 1 (0|1)* Menge aller Wörter die mindestens zwei 1‘en enthalten

(6) (0* 1 0* 1 0*)* Menge aller Wörter, so dass # der 1‘en gerade ist

Reguläre Ausdrücke (4)

- Verschiedene reguläre Ausdrücke können dieselbe Typ 3 Sprache beschreiben:

- (0|1)* (0|1)* äquivalent zu (0|1)* äquivalent zu (0*|1*)*

- Sprache = { ε, 0, 1, 00, 01, … }
  (beliebig viele 0, 1 in beliebiger Reihenfolge)

Reguläre Ausdrücke (5)

- Einige Umformungsregeln:

$\alpha\ (\beta \mid \gamma) = (\alpha\ \beta \mid \alpha\ \gamma)$

$(\alpha^*)^* = \alpha^*$

$(\alpha \mid \beta)^* = (\alpha^*\ \beta^*)^*$

Es gibt keine „Normalform“ (Minimalform) für reguläre Ausdrücke.

- Satz:

  Die Menge der Sprachen, die mittels regulärer Ausdrücke beschrieben werden können, sind genau die Typ3-Sprachen.

- Bemerkung:

  Das Äquivalenzproblem für reguläre Ausdrücke ist (zwar) entscheidbar, aber nur mittels ineffizienter (= exponentieller) Algorithmen.

Reguläre Ausdrücke (6)

- Die von regulären Ausdrücken erzeugten Sprachen sind Typ 3 Sprachen

- Zur Veranschaulichung dieses Zusammenhangs wird die Sprache des regulären Ausdrucks (0|1)* über eine reguläre Grammatik erzeugt:

$S \rightarrow \varepsilon$ $\quad V = \{ S \}$
$S \rightarrow 0$ $\quad \Sigma = \{ 0, 1 \}$
$S \rightarrow 1$
$S \rightarrow 0S$
$S \rightarrow 1S$

Verkürzte Schreibweise (1)

- Die Notation regulärer Ausdrücke kann auf die Produktionsregeln übertragen werden:

$S \rightarrow \varepsilon$
$S \rightarrow 0$
$S \rightarrow 1$
$S \rightarrow 0S$
$S \rightarrow 1S$

(Typ 3)

wird äquivalent geschrieben als

$S \rightarrow \varepsilon \mid 0 \mid 1 \mid 0S \mid 1S$

Verkürzte Schreibweise (2)

- Diese Notation wird auf die Regeln einer Typ 2 Sprache übertragen

z.B. $S \rightarrow A$
$S \rightarrow Abb$ wird äquivalent geschrieben als
$S \rightarrow aA$ $S \rightarrow A \mid Abb \mid aA$

Die Regeln brauchen nicht in CNF vorliegen.

Alternative Darstellungen für kontextfreie Grammatiken

➢ BNF (Backus-Naur-Form)

- **verkürzte Darstellung** für **kontextfreie** Grammatiken (Typ 2).

  Für mehrere Regeln, die alle dieselbe linke Seite haben

$$A \to \beta_1$$
$$A \to \beta_2$$
$$\vdots$$
$$A \to \beta_n$$

  kann verkürzend eine einzige „Metaregel“ angegeben werden (unter Verwendung des „Metasymbols“ |):

$$A \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n \qquad \textbf{(Alternative)}$$

  (Backus und Naur verwendeten statt $\to$ allerdings ::=).

Alternative Darstellungen für kontextfreie Grammatiken (1)

➢ EBNF (erweiterte Backus-Naur-Form)

- Weitere „Abkürzungen“ werden eingeführt:

$A \to \alpha\gamma$
$A \to \alpha\beta\gamma$

wird zu

$A \to \alpha[\beta]\gamma$

Bedeutung: Der Ausdruck β kann – muss aber nicht – zwischen α und γ eingefügt werden. **(einmal oder keinmal)**

$A \to \alpha\gamma$
$A \to \alpha B\gamma$
$B \to \beta$
$B \to \beta B$

wird zu

$A \to \alpha\{\beta\}\gamma$

Bedeutung: Der Ausdruck β kann zwischen α und γ beliebig oft (auch null-mal) wiederholt werden.
(keinmal, einmal oder n-mal)

Beispiel:

G = (V, Σ, P, S)

G = ({ S, A }, { 0, 1 }, { S → 0A11, A → ε, A → 0A, A → 1A }, S)

Aus den Produktionen:

S → 0A11, A → ε, A → 0A, A → 1A

wird in EBNF:

S → 0A11, **A → { 0 | 1 }**

EBNF genauso mächtig, aber kürzer

Beispiel: Syntax für ganze Dezimalzahlen

Herkömmliche Notation:

$$\langle GanzeZahl \rangle \to \langle Zahl \rangle$$
$$\langle GanzeZahl \rangle \to \langle Vorzeichen \rangle\langle Zahl \rangle$$
$$\langle Zahl \rangle \to \langle Ziffer \rangle$$
$$\langle Zahl \rangle \to \langle Ziffer \rangle\langle Zahl \rangle$$
$$\langle Vorzeichen \rangle \to +$$
$$\langle Vorzeichen \rangle \to -$$
$$\langle Ziffer \rangle \to 0$$
$$\langle Ziffer \rangle \to 1$$
$$\langle Ziffer \rangle \to 2$$
$$\langle Ziffer \rangle \to 3$$
$$\langle Ziffer \rangle \to 4$$
$$\langle Ziffer \rangle \to 5$$
$$\langle Ziffer \rangle \to 6$$
$$\langle Ziffer \rangle \to 7$$
$$\langle Ziffer \rangle \to 8$$
$$\langle Ziffer \rangle \to 9$$

BNF:

$$\langle GanzeZahl \rangle \to \langle Zahl \rangle \mid \langle Vorzeichen \rangle\langle Zahl \rangle$$
$$\langle Zahl \rangle \to \langle Ziffer \rangle \mid \langle Ziffer \rangle\langle Zahl \rangle$$
$$\langle Vorzeichen \rangle \to + \mid -$$
$$\langle Ziffer \rangle \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

EBNF:

$$\langle GanzeZahl \rangle \to [+ \mid -]\langle Ziffer \rangle\{\langle Ziffer \rangle\}$$
$$\langle Ziffer \rangle \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Beachte:

In diesem Beispiel sind Nichtterminalsymbole mittels < …> geklammert dargestellt, Terminal-symbole nicht geklammert.

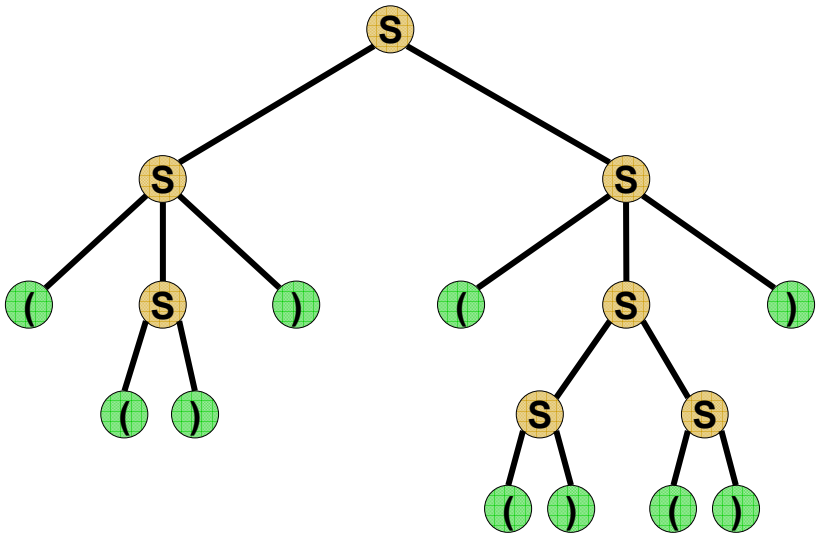
Syntaxbäume (Ableitungsbäume)

G = (V, Σ, P, S)

G = ({ S }, { (,) }, { S → (), S → (S) , S → SS }, S)

Frage : **Gehört (( )) (( ) ( )) zur Sprache ?**

- Mehrere Ableitungen für das selbe Wort, die sich nur in der Reihenfolge der Anwendungen der Produktionen unterscheiden, lassen sich in **einem** Syntaxbaum darstellen.
  (Es kann auch mehrere Syntaxbäume für eine Ableitung geben)



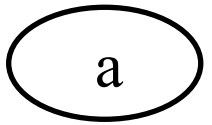
Syntaxdiagramme

Graphische Darstellung für **kontextfreie** Grammatiken (Typ 2)

Grundbausteine der Diagramme:

- nichtterminale Symbole (Variablen) A
  (→ Platzhalter für ein weiteres Diagramm)

  Terminalsymbole a
  (→ Symbole der formalen Sprache)



- Konkatenationen (Wortkonstruktionen)
  (→ Übergang von einem Knoten
  () zum Folgenden)

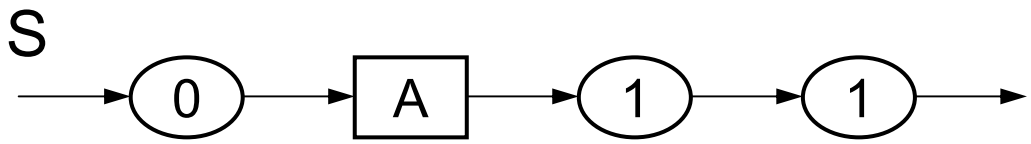
Syntaxdiagramme (1)

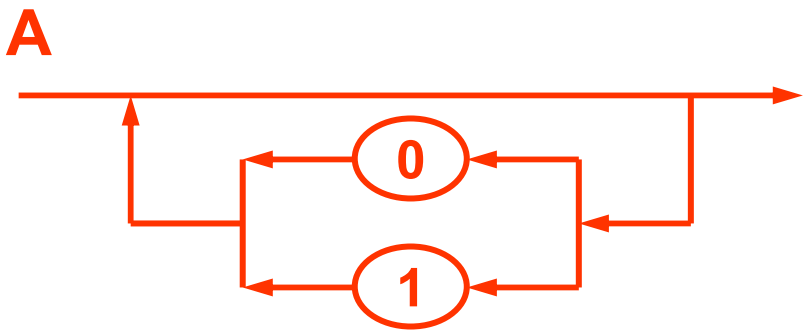
Regeln im Umgang mit Syntaxdiagrammen:

- Jedes Diagramm besitzt einen Namen, genau einen eingehenden Pfeil und genau einen ausgehenden Pfeil.
- Jeder Knoten hat genau einen eingehenden Pfeil und genau einen ausgehenden Pfeil.
- Jedes Rechteck (Variable, Nichtterminalsymbol) verweist auf ein weiteres Syntaxdiagramm, welches an der Rechteck-Stelle hinein kopiert zu denken ist.
- Um ein syntaktisch korrektes Wort zu erhalten, durchläuft man das Syntaxdiagramm beim Eingangspfeil beginnend auf einem der möglichen Wege bis zum Ausgangspfeil. Dabei notiert man der Reihe nach die Terminalsymbole (in den Ovalen), an denen man vorbei kommt.

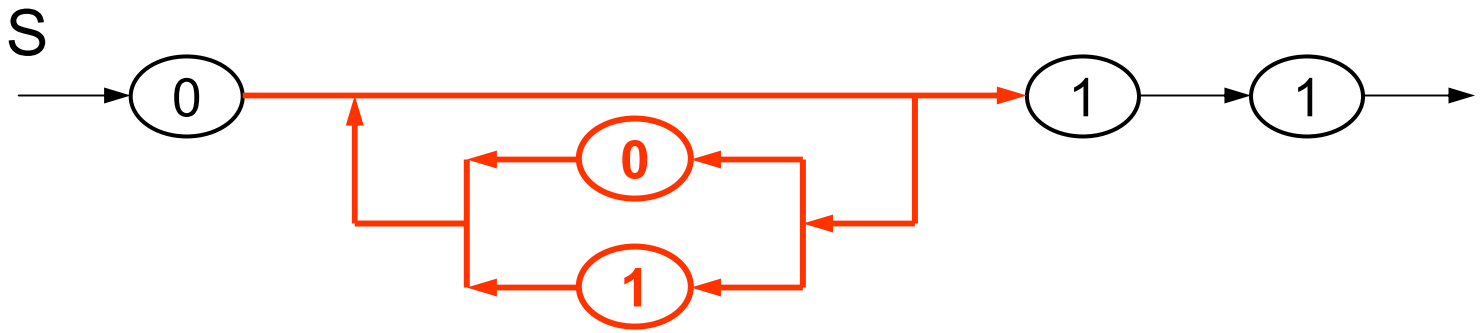
Syntaxdiagramme (2)

➢ Beispiel: Grammatik von Folie 43





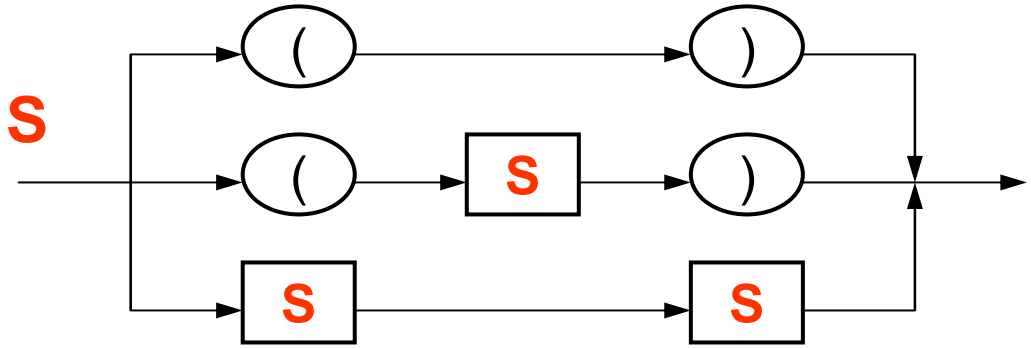
A in S hineinkopiert:



Syntaxdiagramme (3)

Beispiel: „Ausgeglichene Klammern“ von Folie 45

Rekursion: Syntaxdiagramm kann **in sich selbst** eingesetzt/kopiert werden.

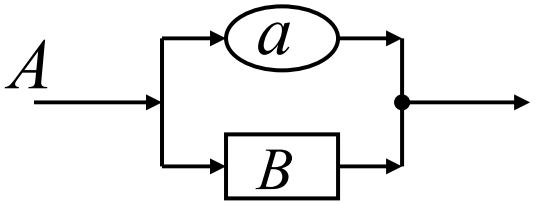


Was wird hier kopiert/eingesetzt?

Allgemeiner Zusammenhang: EBNF – Syntaxdiagramm

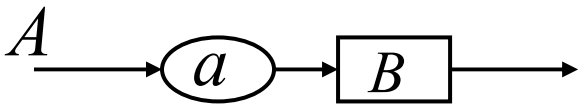
1. **Alternative**

$$A \rightarrow a \mid B\,. \quad \Leftrightarrow$$



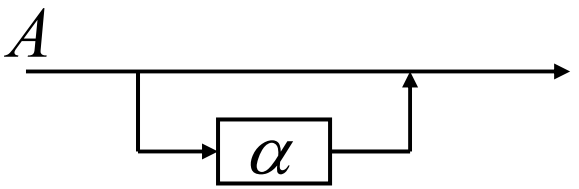
2. **Verkettung**

$$A \rightarrow aB\,. \quad \Leftrightarrow$$

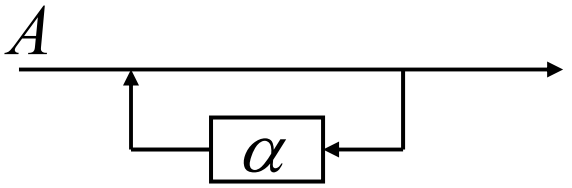


3. **Wiederholungen**

kein- oder einmal α $\quad A \rightarrow [\alpha\,]. \quad \Leftrightarrow$



kein-, ein- oder n-mal α $\quad A \rightarrow \{\alpha\,\}. \quad \Leftrightarrow$



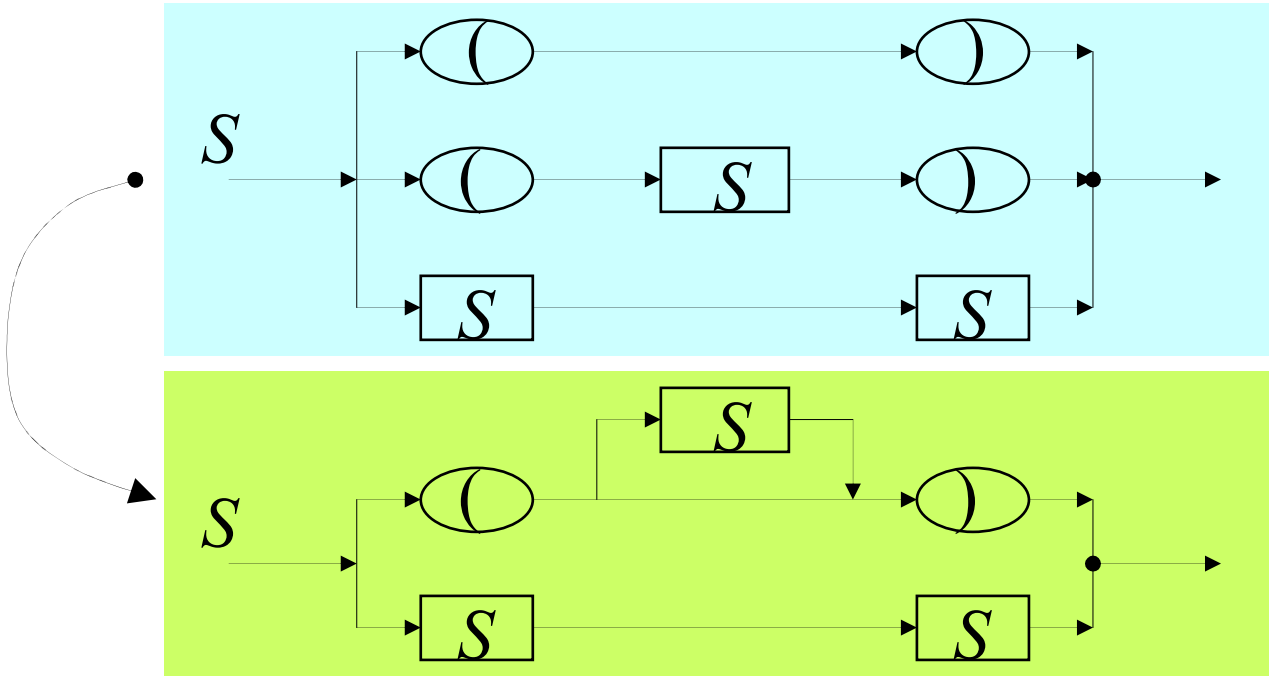
Umsetzung: Produktionen $\rightarrow$ Syntaxdiagramm

Beispiel: „Ausgeglichene Klammern“

1. **Produktionen und EBNF**

$$P = \{S \rightarrow (), S \rightarrow (S), S \rightarrow SS\} \quad \Longleftrightarrow \quad P : S \rightarrow ([S]) \mid SS$$

2. **Diagramme**



2. Endliche Automaten

- Endliche Automaten
- Konstruktionen mit endlichen Automaten
- Abgeschlossenheit
- Minimalautomat
- Äquivalenzproblem für Endliche Automaten
- Leerheitsproblem und Wortproblem
- Kellerautomaten
- Turingmaschine

Endliche Automaten

Eine andere Methode, um Typ 3 Sprachen zu definieren

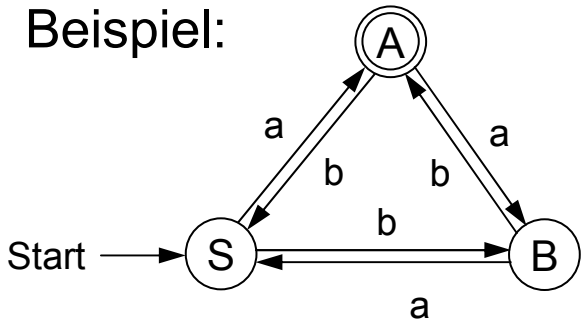
- Bestandteile:

  Zustände: 

  Zustandsübergänge: →

  (gerichtete Kanten, beschriftet mit a $\in \Sigma$, Σ Alphabet)

- Beispiel:



Die vom Automaten M akzeptierte Sprache T(M) ist die Menge aller Wörter w $\in \Sigma^*$, die vom Startzustand zu einem Endzustand führen

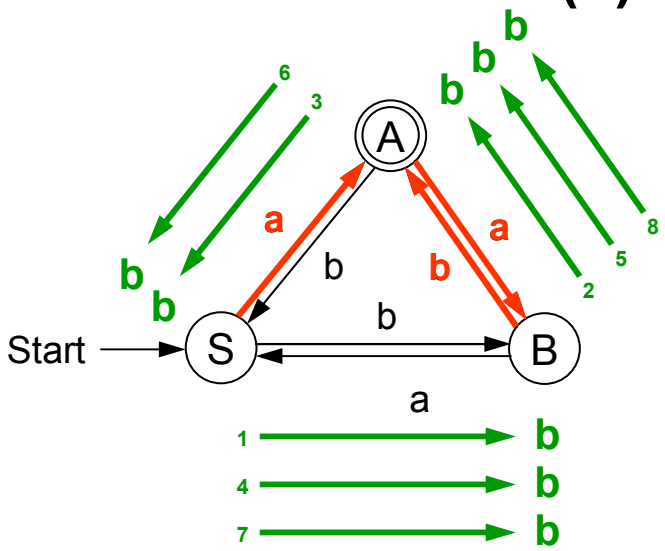
a $\in$ T(M)
b $\notin$ T(M)
aab $\in$ T(M)
aba $\in$ T(M)
baa $\in$ T(M)
usw.

$\in$ { …, -5, -2, 1, 4, 7,… }

Es gilt:
T(M) = { w $\in$ { a, b }* | ((# a‘s in w) – (# b‘s in w)) ≡ 1(mod 3) }

Endliche Automaten (1)

M_1:



a a b

Endzustände dürfen „zwischen-durch“ verlassen werden!

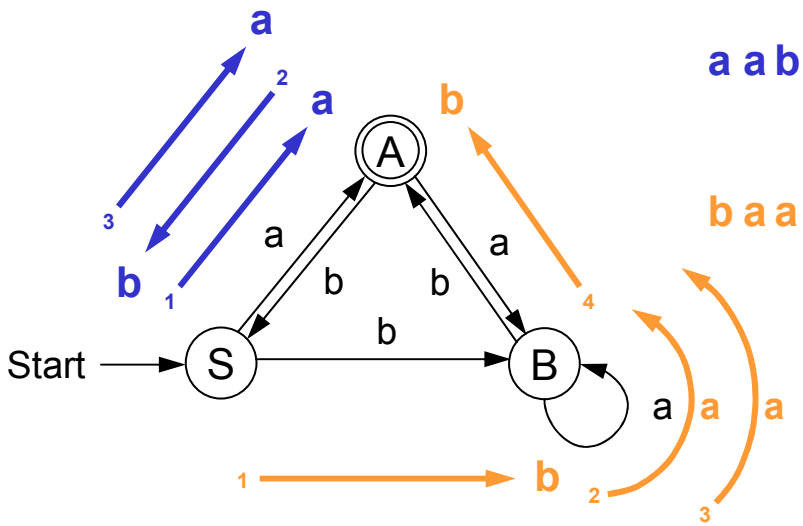
Endzustand = Startzustand erlaubt!

b b b b b b b b

T(M) = { w | # a‘s in w – # b‘s in w = 1 mod 3 }

(beim Teilen durch 3 entsteht eín Rest von 1)
…, –2, 1, 4, 7, 11, …

M_2:



a a b

(geht in beiden Endlichen Automaten)

b a a b

(geht nur im unteren Endl. Autom.)

Endliche Automaten (2)

- Vom Automaten zur Typ 3-Grammatik:

M_1: $S \rightarrow aA \mid bB \mid a$
$A \rightarrow aB \mid bS$
$B \rightarrow aS \mid bA \mid b$

kompaktere Schreibweise für:
$S \rightarrow aA$
$S \rightarrow bB$
$S \rightarrow a$

- Beispiel einer Ableitung:

$S \Rightarrow aA \Rightarrow aaB \Rightarrow aab$

Für jeden Automaten M gibt es eine Typ 3-Grammatik mit L(G) = T(M)

- Es gibt auch die Umkehrung:

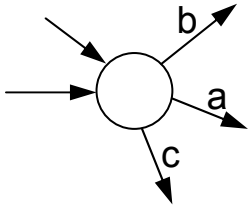
Zu jeder Typ 3-Grammatik G gibt es einen Automaten M mit T(M) = L(G)

Endliche Automaten (3)

- Bemerkung:

  Ein endlicher Automat muss folgende Bedingungen erfüllen:

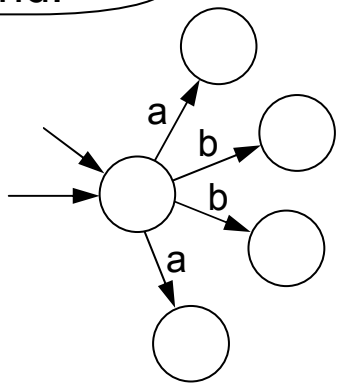
Jeder Zustand besitzt genau $|\Sigma|$ viele hinausgehende Kanten, die mit $a \in \Sigma$ beschriftet sind.



$\Sigma = \{ a, b, c \}$

- Unzulässig:

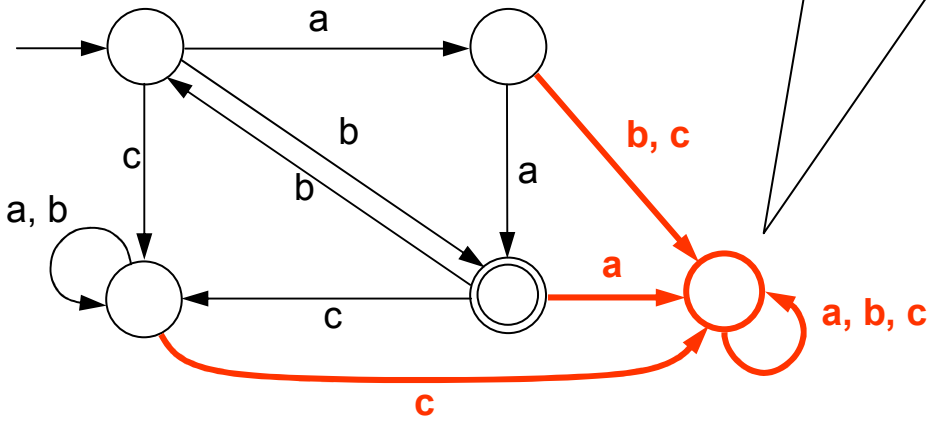
sog. nichtdeterministischer Automat



- Unvollständiger Automat:

(Kanten fehlen)

Vervollständigung

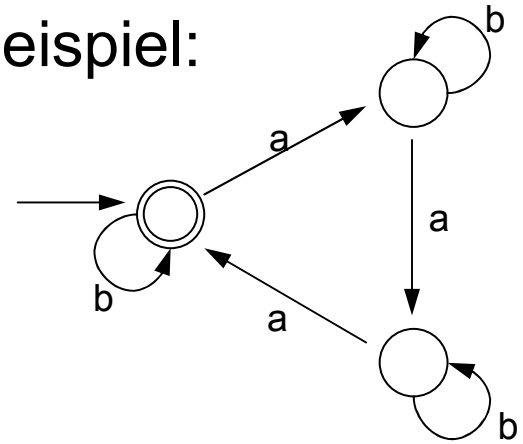


Konstruktionen mit endlichen Automaten

- **Komplement**

  Gegeben sei ein vollständiger Automat M mit L = T(M)

- Beispiel:

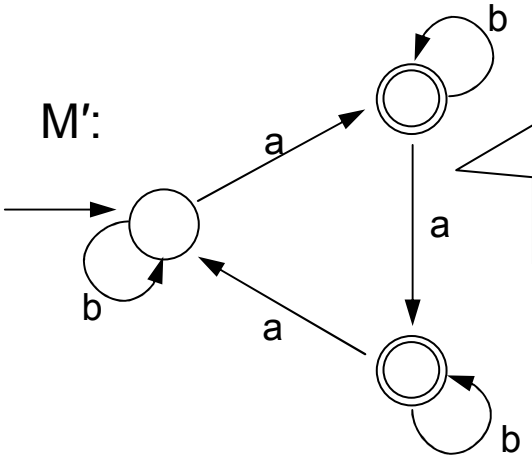


T(M) = { w ∈ { a, b }* | # der a´s in w ist durch 3 teilbar }

Gesucht ist ein M′,

so dass T(M′) = $\overline{L}$:= Σ*\L

M′:



⇒ Man vertauscht Endzustände mit Nicht-Endzuständen

Konstruktionen mit endlichen Automaten (1)

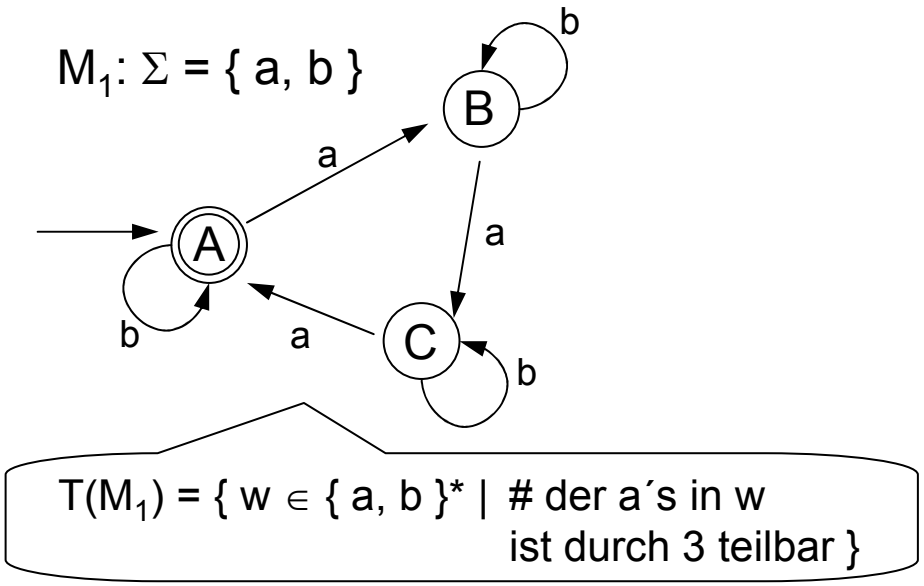
- Die Typ 3-Sprachen sind unter Komplementbildung abgeschlossen.

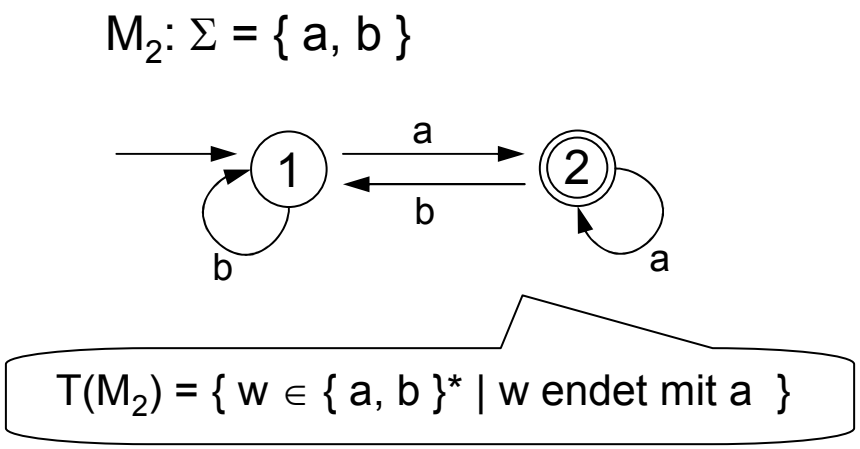
  Falls L vom Typ 3, also L = T(M) für einen Automaten M, so ist auch $\overline{L}$ vom Typ 3, weil für M′ gilt $\overline{L}$ = T(M′).

Konstruktionen mit endlichen Automaten (2)

- Kann man 2 Automaten M_1, M_2 so „vereinigen“ zu einem Automaten M, so daß dieser den Ablauf von M_1, M_2 simultan nachvollziehen kann?

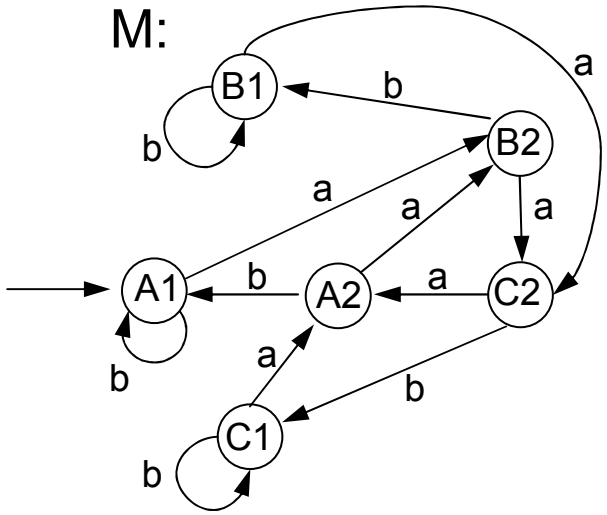
  $\Rightarrow$ Ja → **Kreuzproduktautomat**





Konstruktionen mit endlichen Automaten (3)

Zustände von M = (# Zustände von M_1) · (# Zustände von M_2)



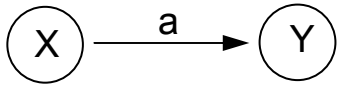
Für
$T(M_1) \cap T(M_2)$ = { w | # der a´s ist durch 3 teilbar **und** w endet mit a }
wähle als Endzustandsmenge E = {A2}

allgemein:
$E = \{ XM \mid X \in E_1$ **und** $M \in E_2 \}$

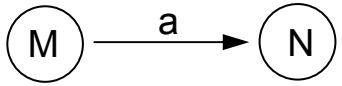
Für
$T(M_1) \cup T(M_2)$ = { w | # der a´s ist durch 3 teilbar **oder** w endet mit a }
wähle als Endzustandsmenge E = { A1, A2, B2, C2 }

allgemein:
$E = \{ XM \mid X \in E_1$ **oder** $M \in E_2 \}$

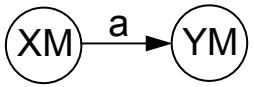
- Methode:

Falls 

in M_1

und 

in M_2

in M: 

Abgeschlossenheit

- Die Typ 3-Sprachen sind auch unter Vereinigung und Schnitt abgeschlossen

	Komplement	Vereinigung	Schnitt
Typ 3	ja	ja	ja
Typ 2	nein	ja	nein
Typ 1	ja	ja	ja
Typ 0	nein	ja	ja

wurde 1985 bewiesen

- Beispiel:

$L_1 = \{ a^n b^n c^m \mid m, n \geq 1 \}$ Typ 2

$L_2 = \{ a^n b^m c^m \mid m, n \geq 1 \}$ Typ 2

$L_1 \cap L_2 = \{ a^n b^n c^n \mid n \geq 1 \}$ Typ 1, nicht Typ 2

$$L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$$

($L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$ (de Morgan))

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

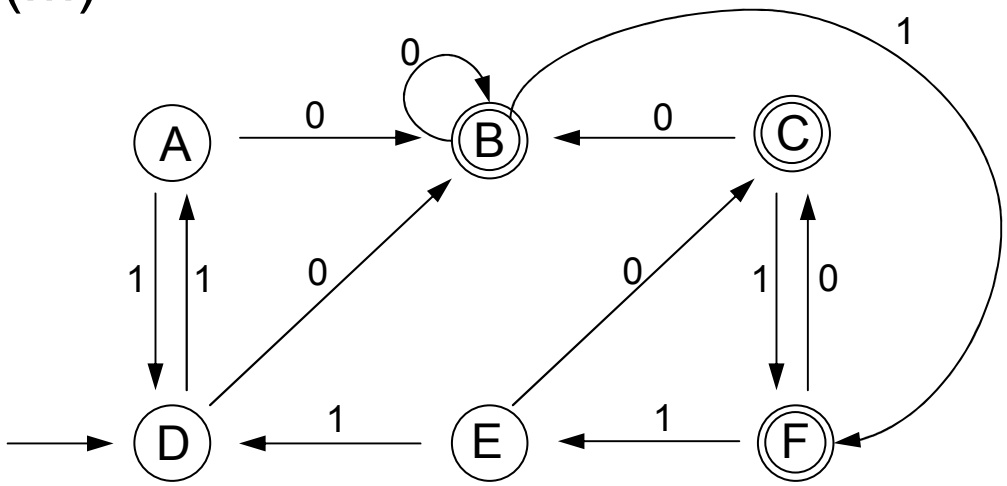
Falls Abschluss vorliegt unter Komplement und Schnitt / Vereinigung, dann auch Abschluss unter Vereinigung / Schnitt

Minimalautomat

- Gegeben sei ein Automat M
  (evtl. mit unnötig vielen Zuständen)

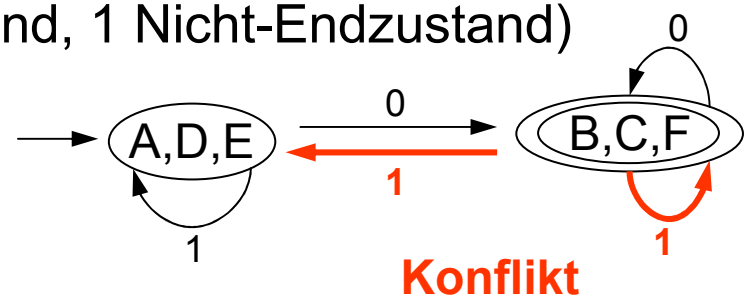
- Gesucht:
  Anzahl M_0 mit minimaler Anzahl Zuständen, so dass gilt
  $T(M_0) = T(M)$.

- Beispiel:

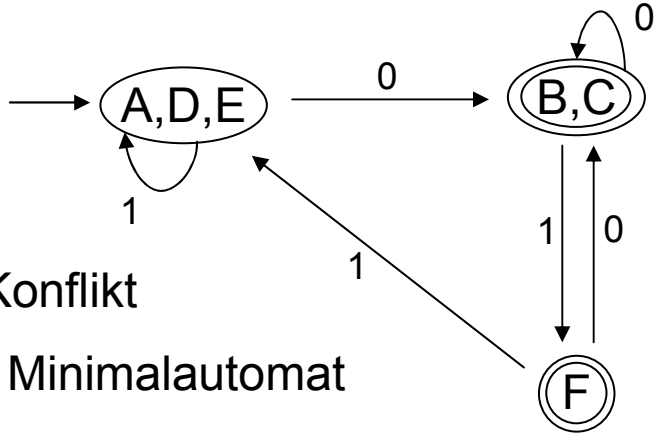


Minimalautomat (1)

- 1. Hypothese:
  Es geht mit 2 Zuständen
  (1 Endzustand, 1 Nicht-Endzustand)



- 2. Hypothese:
  3 Zustände



kein Konflikt

→ Minimalautomat

- Es gilt:
  $T(M_0) = \{ w \in \{ 0,1 \}^* \mid w \text{ endet mit 0 oder mit 01} \}$

Minimalautomat (2)

- Bemerkung:
  Zu jeder Typ 3-Sprache L ist der zugehörige Minimalautomat M_0 mit $L = T(M_0)$ eindeutig (bis auf Bezeichnung der Zustände).

→ Effizienter Äquivalenztest für Typ 3-Sprachen (Automaten):

- Gegeben M_1, M_2, stelle fest, ob $T(M_1) = T(M_2)$.

  1. Konstruiere zu M_1 den Minimalautomat. M_{01}
  2. Konstruiere zu M_2 den Minimalautomat. M_{02}
  3. Vergleiche M_{01} und M_{02} untereinander.
     Falls M_{01} identisch mit M_{02}, so $T(M_1) = T(M_2)$.

Äquivalenzproblem für Endliche Automaten

- Das Äquivalenzproblem für endliche Automaten ist (effizient) entscheidbar.

  Methode: Gegeben sind die Automaten

  M_1, M_2: Konstruiere Minimalautomat zu $M_1 \rightarrow \widehat{M}_1$, und zu $M_2 \rightarrow \widehat{M}_2$

  Es gilt $T(M_1) = T(M_2)$ genau dann, wenn $\widehat{M}_1$ ist isomorph zu $\widehat{M}_2$.

  d. h. M_1 und M_2 sind – bis auf die Benennung der Zustände – identisch.

- Bemerkung:

  Ab Typ 2 (kontextfrei) ist das Äquivalenzproblem unlösbar

  (d. h. es gibt keinen immer stoppenden Algorithmus)

Leerheitsproblem und Wortproblem

	Typ 3	Typ 2	Typ 1	Typ 0
Äquivalenzproblem	E	U	U	U
Leerheitsproblem	E	E	U	U
Wortproblem	E	E	E	U

E = entscheidbar
U = unentscheidbar

- **Äquivalenzproblem**
  gegeben: M_1, M_2, stelle fest, ob $T(M_1) = T(M_2)$ bzw.
  G_1, G_2, stelle fest, ob $L(G_1) = L(G_2)$

- **Leerheitsproblem**
  gegeben: G, stelle fest, ob $L(G) = \emptyset$ bzw.
  M, stelle fest, ob $T(M) = \emptyset$

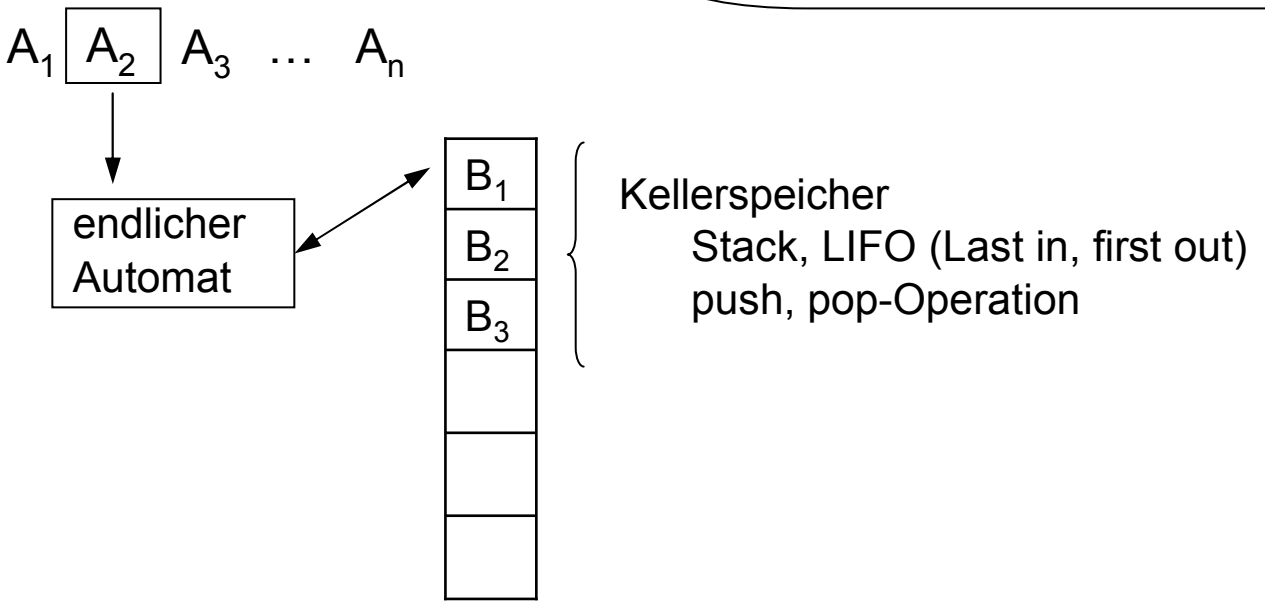
- **Wortproblem**
  gegeben: G und x, stelle fest, ob $x \in L(G)$ bzw.
  M und x, stelle fest, ob $x \in T(M)$

Kellerautomaten

- Typ2 wird durch Kellerautomaten beschrieben

also: L ist vom Typ2 $\Leftrightarrow$ L = T(M)
für einen Kellerautomaten M

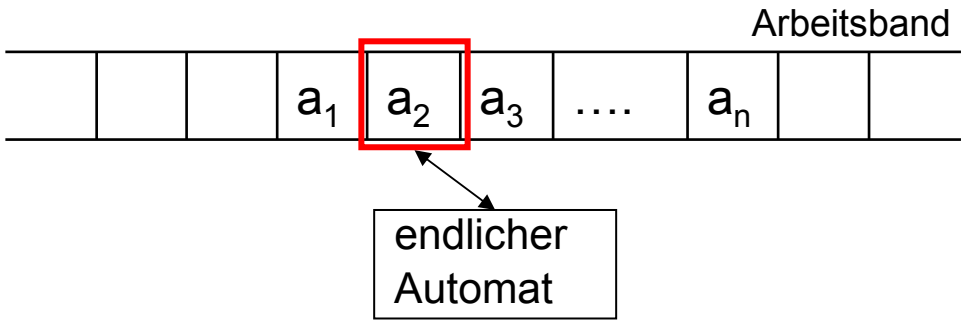
- Skizze:



Turing-Maschine

- Typ0 wird durch die Turing-Maschine beschrieben
  (Alan Turing, 1912-1954)

- Skizze:



Mögliche Aktionen:

- Lesen des aktuellen Zeichens
- Überschreiben des Zeichens mit einem anderen Zeichen
- Kopfbewegung nach links und rechts
- Übergang in einen neuen Zustand

Berechenbarkeit

L ist vom Typ0 $\Leftrightarrow$ L = T(M) für eine Turingmaschine M

T(M) = { $a_1, \ldots, a_n$ | M, bei Eingabe $a_1, \ldots, a_n$ stoppt nach endlich vielen Schritten im Endzustand }

(Die Automaten heißen auch LBAs)

- Typ1 wird beschrieben durch linear beschränkte Turingmaschinen.
  d. h. M darf die Felder links und rechts der Eingabe nicht besuchten / verändern. Die Eingabefelder dürfen beliebig verändert werden.

- Turing wollte mit seiner Maschine das Konzept der „Berechenbarkeit“ formal beschreiben.

- „Church`sche These“:
  Jede Art von Berechung / Algorithmus / Programm kann auch mit Hilfe einer Turingmaschine formuliert werden.

Also bedeutet „f ist berechenbar“ (mittels beliebiger Art von Berechnungsformalismus) dasselbe wie „f ist mittels Turingmaschine berechenbar“

Entscheidbarkeit / Semi-Entscheidbarkeit

- Definition:

  Eine Sprache $L \subseteq \Sigma^*$ heißt entscheidbar (oder auch rekursiv), falls es einen Algorithmus (z.B. Turingmaschine) gibt mit:

  1) $x \in L \Rightarrow$ Algorithmus bei Eingabe x,
     stoppt in endliche vielen Schritten und gibt 1 aus.

  2) $x \notin L \Rightarrow$ Algorithmus bei Eingabe x,
     stoppt in endliche vielen Schritten und gibt 0 aus.

- L heißt semi-entscheidbar, falls nur (1) gilt:

  Im Fall von $x \notin L$ kann M evtl. ∞ so viele Schritte machen.

andere Bezeichnung:
rekursiv aufzählbar

Semi-Entscheidbarkeit

- Beispiel:

$L_1 = \{ a_1 a_2 \ldots a_n \in \{0, 1, \ldots, 9\}^* \mid a_1 a_2 \ldots a_n$ ist Anfangsabschnitt der Dezimalentwicklung von $\pi \}$

$31 \in L_1, \quad 3141 \in L_1$

$56 \notin L_1, \quad 315 \notin L_1,$

L_1 ist entscheidbar:
Starte Approximationsverfahren für π, bis genügend korrekte Ziffern feststehen, vergleiche diese mit $a_1 a_2 \ldots a_n$

$L_2 = \{ a_1 a_2 \ldots a_n \mid a_1 a_2 \ldots a_n$ kommt irgendwo in der Dezimalentwicklung von π vor $\}$

$141 \in L_2, \quad 4159 \in L_2$

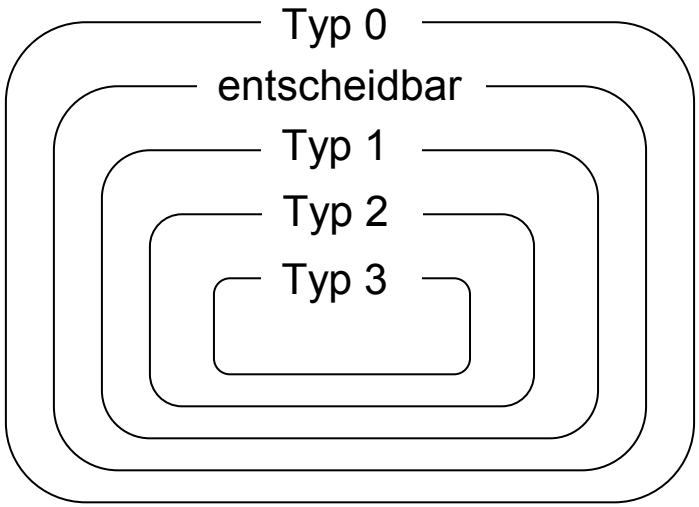
$5645678 \in L_2$

L_2 ist semi-entscheidbar:
Approximationsverfahren liefert immer mehr exakte Ziffern von π, vergleiche diese mit $a_1 \ldots a_n$.
Falls $a_2 \ldots a_n$ vorkommt: stopp und gib 1 aus.
Möglicherweise ist L_2 nicht entscheidbar.

Halteproblem

- Es gilt:

  L ist Typ0 $\Leftrightarrow$ L ist semi-entscheidbar



- Halteproblem (Selbstanwendbarkeitsproblem):
- Bemerkung:

  Programme (Turingmaschinen) sind Wörter über einem Alphabet Σ

  Die Eingabe $a_1 \ldots a_n$ ist ebenfalls Wort über Alphabet.

  $H = \{ x \in \Sigma^* \mid$ x ist Text eines Programms welches gestartet mit x als Eingabe nach endlich vielen Schritten stoppt $\}$

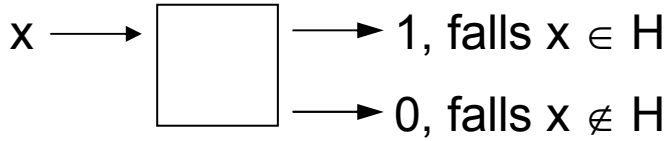
Halteproblem (1)

- Satz

  H ist nicht entscheidbar

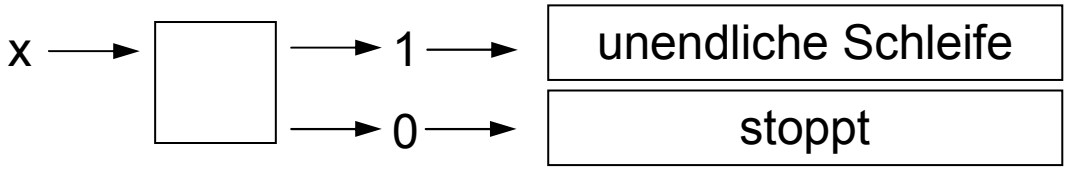
- Beweis: (indirekt)

  Angenommen, H wäre entscheidbar, d.h. es gibt Algorithmus, der H entscheidet



Halteproblem (2)

- Konstruiere weiteres Programm:



Dieses Programm sei z ϵ A*

Entweder:
z stoppt, gestartet mit z als Eingabe

$\Rightarrow$ fiktives Programm für H gibt 0 aus, bei Eingabe z
$\Rightarrow$ z $\notin$ H
$\Rightarrow$ z, gestartet mit z, als Eingabe, stoppt nicht.

oder:
z, gestartet mit z als Eingabe, stoppt nicht

$\Rightarrow$ z bei Eingabe z, gerät in ∞ Schleife
$\Rightarrow$ fiktives Programm für H gibt bei Eingabe z, 1 aus
$\Rightarrow$ z $\in$ H
$\Rightarrow$ z, bei Eingabe z, stoppt.