Grundlagen: Wörter

Alphabet: *endliche*, nicht-leere Menge atomarer Symbole (Σ, T)

Wort über Σ: endliche Folge von Symbolen aus Σ

Länge eines Wortes w über Σ (geschrieben $|w|$): Anzahl der Symbole, die w enthält

Leerwort: Wort mit der Länge 0, geschrieben ε, d.h. $|\varepsilon| = 0$

Für ein Wort w über Σ und ein Symbol $a \in \Sigma$ bezeichnen wir die **Anzahl der Symbole** a **in** w mit $|w|_a$.

Konkatenation: Hintereinanderschreiben von Wörtern
Seien x, y Wörter mit $|x| = n$, $|y| = m$, dann ist $x \cdot y = xy$ und $|xy| = n + m$

Potenzbildung: Verkettung eines Wortes w mit sich selbst, wobei $w^0 = \varepsilon$ und $w^n = ww^{n-1}$

Sei $w = a_1 a_2 \dots a_{n-1} a_n$, dann ist $w^r = a_n a_{n-1} \dots a_2 a_1$ das **Spiegelbild** von w.

Ein Wort w heißt **Palindrom**, wenn $w = w^r$ gilt.

Operationen auf Sprachen

Konkatenation: $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1, w_2 \in L_2\}$

Bsp: $\{ab, b\}\{a, bb\} = \{aba, abbb, ba, bbb\}$
(Achtung: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$)

Potenzbildung: $L^n = \{w_1 \dots w_n \mid w_1, \dots, w_n \in L\}$
($L^0 = \{\varepsilon\}$ und $L^{n+1} = LL^n$ für $n \geq 0$)

Bsp: $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$

Kleene-Stern[1]: $L^* = \bigcup_{n \geq 0} L^n$ $\qquad$ $(L^+ = \bigcup_{n \geq 1} L^n)$

Bsp: $\{01\}^* = \{\varepsilon, 01, 0101, 010101, \dots\} \neq \{0, 1\}^*$

$(\mathcal{P}(\Sigma^*), \cdot, \{\varepsilon\})$ bildet ein *Monoid*.

- $\varepsilon \in L^*$ gilt für alle Sprachen L (z.B.: $\{\}^* = \{\varepsilon\}$)
- $\varepsilon \in L^+$ gilt für eine Sprache L genau dann, wenn $\varepsilon \in L$

[1] benannt nach Stephen Cole Kleene (1909 - 1994)

Rechenregeln für Sprachoperatoren

$$
\begin{array}{rcll}
A \cdot (B \cdot C) & = & (A \cdot B) \cdot C & \text{Assoziativität von} \cdot \\
A \cdot (B \cup C) & = & A \cdot B \cup A \cdot C & \text{Distributivität von} \cdot \\
(B \cup C) \cdot A & = & B \cdot A \cup C \cdot A & \text{Distributivität von} \cdot
\end{array}
$$

$$
\begin{array}{rcl}
(A \cup \{\varepsilon\})^* & = & A^* \\
(A^*)^* & = & A^* \\
A \cdot A^* & = & A^+ \\
A^* \cdot A & = & A^+ \\
A^+ \cup \{\varepsilon\} & = & A^*
\end{array}
\qquad
\begin{array}{rcl}
\{\varepsilon\} \cdot A & = & A \\
A \cdot \{\varepsilon\} & = & A \\
\{\} \cdot A & = & \{\} \\
A \cdot \{\} & = & \{\}
\end{array}
$$

$(\mathcal{P}(\Sigma^*), \cup, \cdot, \{\}, \{\varepsilon\})$ bildet einen *nichtkommutativen Semiring*.

Mengenoperationen auf Sprachen

Vereinigung:
$A \cup B = \{x \in \Sigma^* \mid x \in A \text{ oder } x \in B\}$

Durchschnitt:
$A \cap B = \{x \in \Sigma^* \mid x \in A \text{ und } x \in B\}$
(A und B sind **disjunkt** wenn $A \cap B = \{\}$)

Differenz:
$A - B = \{x \in \Sigma^* \mid x \in A \text{ und } x \notin B\}$

Komplement:
$\overline{A} = \Sigma^* - A = \{x \in \Sigma^* \mid x \notin A\}$

$A \subseteq B$: A ist eine **Teilmenge** von B (jedes Element von A ist auch ein Element von B)

$A \subset B$: A ist eine **echte Teilmenge** von B (A ist eine Teilmenge von B, die nicht gleich B ist)

$A = B$ wenn $A \subseteq B$ und $B \subseteq A$

Abzählbare Mengen

Die Anzahl der Elemente einer Menge M bezeichnet man als **Kardinalität**, geschrieben als $card(M)$ oder $|M|$.

$|\mathcal{P}(A)| = 2^{|A|}$

Zwei Mengen A und B heißen **gleichmächtig**, wenn es eine bijektive Abbildung von A auf B gibt.

Jede Menge, die gleichmächtig mit $\mathbb{N}$ (d.h., mit der Menge der Natürlichen Zahlen) ist, heißt **abzählbar (unendlich)**.

Σ^* ist abzählbar (falls Σ endlich).

Beweis (Idee)

$$
\begin{array}{rcccccccccccc}
\{0,1\}^* & = & \{ & \varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000, & \dots & \} \\
\mathbb{N} & = & \{ & 0, & 1, & 2, & 3, & 4, & 5, & 6, & 7, & \dots & \}
\end{array}
$$

□

Überabzählbare Mengen

Jede unendliche, nicht abzählbare Menge heißt **überabzählbar**.

Die Menge aller reellen Zahlen $\mathbb{R}$ ist gleichmächtig mit der Menge der reellen Zahlen in jedem beliebigen Intervall $(a, b) = \{x \mid x \text{ reell und } a < x < b\}$; beide Mengen sind nicht abzählbar, was auf ähnliche Weise wie der folgende Satz gezeigt werden kann.

Formale Beschreibung einer Turingmaschine

Eine **Turingmaschine** ist ein Siebentupel

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

- Q endliche Menge von Zuständen,
- Σ endliche Menge der Eingabesymbole,
- Γ endliche Menge der Bandsymbole, $\Sigma \subset \Gamma$,
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, S\}$ Übergangsfunktion,
- $q_0 \in Q$ Startzustand,
- $B \in \Gamma - \Sigma$ Blank-Symbol,
- $F \subseteq Q$ Menge von Endzuständen.

Übergänge

$$(q, X; p, Y, D) \in \delta$$

M liest im Zustand q auf dem Band Symbol X

- wechselt nun davon abhängig in den Zustand p,
- überschreibt Symbol X durch Symbol Y,
- bewegt den Lese-/Schreibkopf in die durch D beschriebene Richtung ($D \in \{L, R, S\}$).

Deterministische Turingmaschine (DTM)

Eine Turingmaschine M heißt **deterministisch**, wenn für alle $(q, X) \in Q \times \Gamma$ höchstens ein Element $(q, X; p, Y, D) \in \delta$ existiert, $D \in \{L, R, S\}$; wir schreiben dann auch $\delta(q, X) = (p, Y, D)$.

(Deterministische) Turingmaschinen: akzeptierte Sprache

Akzeptierte Sprache

Die von der (deterministischen) Turingmaschine M **akzeptierte Sprache** $L(M)$ besteht aus genau all jenen Wörtern, bei deren Analyse M einen Endzustand erreicht:

$$L(M) = \{w \in \Sigma^* \mid q_0 w \Rightarrow^* \alpha p \beta, \quad p \in F, \quad \alpha, \beta \in \Gamma^*\}$$

Rekursiv aufzählbare Sprachen

Eine formale Sprache heißt **rekursiv aufzählbar**[2] wenn sie von einer (deterministischen) Turingmaschine akzeptiert wird.

[2] engl: recursively enumerable, auch bezeichnet als: Turing-erkennbar (Turing-recognizable), semi-entscheidbar (semi-decidable)

(Deterministische) Turingmaschinen: Akzeptieren durch Anhalten

Wir können stets annehmen, dass eine Turingmaschine anhält, wenn sie akzeptiert.
($\delta(q, X)$ kann für $q \in F$ als undefiniert bezeichet werden.)

Im Folgenden betrachten wir Turingmaschinen, die immer anhalten, wenn sie sich in einem akzeptierenden Zustand befinden.

Entscheidbare (Rekursive) Sprachen

Wird eine Turingmaschine M auf einem Input w gestartet, so gibt es folgende Möglichkeiten:

- M hält in einem Zustand $q \in F$ und akzeptiert somit die Eingabe
- M hält in einem Zustand $q \notin F$ und verwirft somit die Eingabe
- M gerät in eine Endlosschleife und hält nicht

Rekursive (Entscheidbare) Sprachen

Eine formale Sprache heißt **rekursiv** oder **entscheidbar**[3] wenn sie von einer (deterministischen) Turingmaschine akzeptiert wird, die **immer hält**. (Man sagt dann auch: die TM **entscheidet** die Sprache)

[3] engl: recursive, decidable, auch bezeichnet als: Turing-entscheidbar (Turing-decidable)

Nichtdeterministische Turingmaschinen

Eine **nichtdeterministische Turingmaschine** (NTM) unterscheidet sich zur deterministischen Variante in der Übergangsfunktion δ:

$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), ..., (q_k, Y_k, D_k)\},$$

wobei k eine beliebige endliche ganze Zahl ist.

Eine NTM M akzeptiert eine Eingabe, wenn es eine Folge möglicher Bewegungen gibt, die von der Startkonfiguration zu einer Konfiguration mit einem akzeptierenden Zustand führt.[4]

Zu jeder nichtdeterministischen Turingmaschine M_N gibt es eine deterministische Turingmaschine M_D mit $L(M_N) = L(M_D)$.

[4] Die Existenz eventueller weiterer Auswahlmöglichkeiten ist dabei irrelevant.

(Nicht)deterministische Turingmaschinen

Turingmaschinen können auch Funktionen (auf nicht-negativen ganzen Zahlen) berechnen, bzw. Sprachen erzeugen:

Berechnung von Funktionen

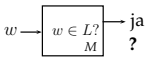
Eine (*deterministische*) Turingmaschine M kann auch zur **Berechnung von Funktionen** verwendet werden, d.h., M beginnt mit dem Input auf dem Band, der Output, also der Funktionswert zum Input, ist das Ergebnis der Berechnung, wenn M hält.

Erzeugung von Sprachen

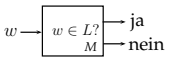
Eine (*nichtdeterministische*) Turingmaschine M kann auch zur **Erzeugung von Wörtern** verwendet werden, d.h., M beginnt mit dem leeren Band, das Ergebnis der Berechnung ist das Wort, das am Ende einer Berechnung, wenn M hält, auf dem Band steht.

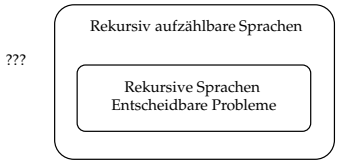
Problemlandschaft

Rekursiv aufzählbare Sprachen L:
Es gibt eine TM M die akzeptiert und hält wenn $w \in L$, sonst aber u.U. unendlich läuft.



Rekursive Sprachen (**Entscheidbare Probleme**) L:
Es gibt eine TM M mit $L(M) = L$ die immer hält. (d.h., es gibt einen **Algorithmus**, der bei Eingabe von w "ja" antwortet, wenn $w \in L$ und "nein" antwortet, wenn $w \notin L$)



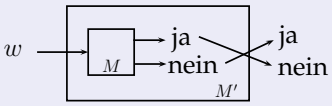


Rekursive (entscheidbare) Sprachen und ihr Komplement

Das Komplement einer rekursiven Sprache ist rekursiv.

Beweis

Sei L rekursiv und M eine TM mit $L(M) = L$, die immer hält. Wir konstruieren M' aus M so, dass M' stoppt ohne zu akzeptieren, wenn M auf einer Eingabe w in einen Endzustand übergeht. Hält M ohne zu akzeptieren, so geht M' in einen Endzustand über. Da eines dieser beiden Ereignisse eintritt, ist $\overline{L} = L(M')$ (das Komplement von L) eine rekursive Sprache.



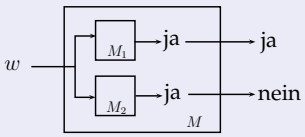
□

Rekursive (entscheidbare) Sprachen und ihr Komplement

Ist eine Sprache L und auch ihr Komplement $\overline{L}$ rekursiv aufzählbar, so ist L (wie auch $\overline{L}$) rekursiv.

Beweis

M_1 und M_2 akzeptieren L bzw. $\overline{L}$. Daraus konstruieren wir M so, dass sie w akzeptiert, wenn w von M_1 akzeptiert wird, und verwirft, wenn w von M_2 akzeptiert wird. Da w entweder in L oder in $\overline{L}$ ist, wird genau eine der beiden Turingmaschinen M_1, M_2 akzeptieren. Also wird M immer entweder "ja" oder "nein" ausgeben, aber nie beides. Somit ist L rekursiv.



□

Rekursive und rekursiv aufzählbare Sprachen

Konsequenz: Sei L eine Sprache und $\overline{L}$ ihr Komplement.
Dann gilt eine der folgenden Aussagen:

1. sowohl L als auch $\overline{L}$ ist rekursiv.
2. sowohl L als auch $\overline{L}$ ist nicht rekursiv aufzählbar.
3. entweder L oder $\overline{L}$ ist rekursiv aufzählbar, aber nicht rekursiv, und die jeweils andere Sprache ist nicht rekursiv aufzählbar.

Nicht rekursiv aufzählbare Sprachen

Es gibt Sprachen über $\Sigma = \{0, 1\}$, die nicht rekursiv aufzählbar sind.

Beweis

- Zu jeder rekursiv aufzählbaren Sprache L über $\Sigma = \{0, 1\}$ (d.h. $L \in \mathcal{P}(\Sigma^*)$) gibt es eine Turingmaschine, die sie akzeptiert.
- Jede Turingmaschine M kann als String über $\Sigma = \{0, 1\}$ (d.h. $M \in \Sigma^*$) dargestellt werden.
- Die Menge aller Sprachen $\mathcal{P}(\Sigma^*)$ ist überabzählbar (unendlich), die Menge aller Wörter Σ^* hingegen abzählbar (unendlich).
- Es gibt also überabzählbar viele Sprachen, von denen jedoch nur abzählbar viele von einer Turingmaschine akzeptiert werden.

□

Das Halteproblem ist rekursiv aufzählbar

$L_u = \{(M, w) \mid M$ ist eine TM die w akzeptiert $\}$

L_u ist **rekursiv aufzählbar**, aber **nicht entscheidbar**.

Beweis Teil 1: L_u ist rekursiv aufzählbar

Wir konstruieren eine universelle Turingmaschine U wie folgt:
Mit Eingabe (M, w) simuliert U die TM M auf w. Wenn M die Eingabe akzeptiert, so akzeptiert auch U, akzeptiert M nicht, so akzeptiert auch U nicht:

$$U \text{ akzeptiert } (M, w) \iff M \text{ akzeptiert } w \iff (M, w) \in L_u$$

Folglich akzeptiert U die Sprache L_u. □

Das Halteproblem ist unentscheidbar

Beweis Teil 2: L_u ist nicht entscheidbar

Beweis indirekt. Angenommen, es gibt eine Maschine H, die L_u *entscheidet*:

$$H((M, w)) = \begin{cases} \textbf{ja} & \text{wenn } w \text{ von } M \text{ akzeptiert wird} \\ \textbf{nein} & \text{wenn } w \text{ von } M \text{ nicht akzeptiert wird} \end{cases}$$

Dann gibt es aber auch eine Maschine D, die genau das Gegenteil von H macht.
D simuliert zunächst H mit der Eingabe $(M, (M))$ und gibt dann **ja** aus, wenn M nicht akzeptiert, bzw. **nein**, wenn M akzeptiert:

$$D((M)) = \begin{cases} \textbf{ja} & \text{wenn } (M) \text{ von } M \text{ nicht akzeptiert wird} \\ \textbf{nein} & \text{wenn } (M) \text{ von } M \text{ akzeptiert wird} \end{cases}$$

Offensichtlich gilt: wenn H existiert, so existiert auch D.

Das Halteproblem ist unentscheidbar - ctd.

Setzt man nun D auf ihren eigenen Code an, so ergibt sich folgendes Verhalten:

$$D((D)) = \begin{cases} \textbf{ja} & \text{wenn } (D) \text{ von } D \text{ nicht akzeptiert wird} \\ \textbf{nein} & \text{wenn } (D) \text{ von } D \text{ akzeptiert wird} \end{cases}$$

Was auch immer D macht, sie muss genau das Gegenteil machen. Das ist offensichtlich ein Widerspruch! Es kann also weder D noch H geben, demnach ist L_u nicht entscheidbar. □

Reduktionen und (Nicht-)Entscheidbarkeit

Seien A und B Sprachen mit $A \leq B$. Dann gilt:

- Ist A nicht entscheidbar, so ist auch B nicht entscheidbar.
- Ist A nicht rekursiv aufzählbar, so ist auch B nicht rekursiv aufzählbar.
- Ist B entscheidbar, so ist auch A entscheidbar.
- Ist B rekursiv aufzählbar, so ist auch A rekursiv aufzählbar.

Eigenschaften von Sprachen

Eine **Eigenschaft** von Sprachen ist eine Teilmenge P von rekursiv aufzählbaren Sprachen (über einem geeigneten Alphabet).

Beispiele für Eigenschaften

$P_1 = \{L \mid L$ ist entscheidbar $\}$
$P_2 = \{L \mid L$ ist endlich $\}$
$P_3 = \{L \mid L = L^*\}$
$P_4 = \{\{\varepsilon\}\}$
$P_5 = \{\}$
$P_6 = \{L \mid L$ ist rekursiv aufzählbar $\}$

$\{a, b\}^*$ hat die Eigenschaften P_1, P_3, P_6, aber nicht P_2, P_4, P_5

Der Satz von Rice[6]

Eine Eigenschaft ist **trivial**, wenn sie entweder leer ist (d.h., sie kommt keiner Sprache zu), oder aus allen rekursiv aufzählbaren Sprachen besteht. Andernfalls ist sie nicht trivial.

Triviale Eigenschaften

$P_5 = \{\}$
$P_6 = \{L \mid L$ ist rekursiv aufzählbar $\}$
sind triviale Eigenschaften.

Satz von Rice

Jede nicht triviale Eigenschaft der rekursiv aufzählbaren Sprachen ist unentscheidbar.

[6] Henry Gordon Rice (* 1920)

Der Satz von Rice: Anwendungen

Einige Beispiele von Eigenschaften, für die der Satz von Rice unmittelbar die Unentscheidbarkeit impliziert.

- Es ist nicht entscheidbar, ob eine Turingmaschine mehr als sieben Wörter akzeptiert.
  (Die Eigenschaft P ist also definiert durch $P = \{L \mid |L| \geq 7\}$. Um den Satz von Rice anwenden zu können, müssen wir noch überprüfen, ob die Eigenschaft P nicht-trivial ist. Wir müssen also mindestens eine rekursiv aufzählbare Sprache finden, die die Eigenschaft erfüllt, und eine, die die Eigenschaft nicht erfüllt: z.B. $L_1 = \{a, a^2, a^3, \ldots, a^7\} \in P$ und $L_2 = \{\} \notin P$.)
- Es ist nicht entscheidbar, ob eine Turingmaschine nur endlich viele Wörter akzeptiert (d. h. ob die von einer Turingmaschine akzeptierte Sprache endlich ist).
- Es ist nicht entscheidbar, ob die von einer Turingmaschine akzeptierte Sprache regulär ist.

Nicht Anwendbarkeit des Satzes von Rice

Es gibt aber durchaus Eigenschaften für Turingmaschinen, die entscheidbar sind. Für diese Eigenschaften ist dann aber eine der Voraussetzungen des Satzes von Rice nicht erfüllt. (Entweder handelt es sich nicht um eine Eigenschaft der akzeptierten Sprache der Turingmaschine oder die Eigenschaft ist trivial)

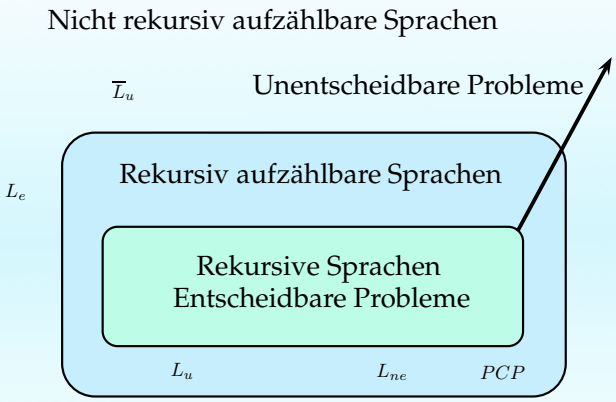
- Es ist entscheidbar, ob eine Turingmaschine mindestens 7 Zustände besitzt.
  (Diese Eigenschaft ist keine Eigenschaft der akzeptierten Sprache, sondern eine Eigenschaft der Turingmaschine selbst.)
- Es ist entscheidbar, ob die von einer Turingmaschine akzeptierte Sprache rekursiv aufzählbar ist.
  (Diese Eigenschaft ist trivial, sie ist für jede rekursiv aufzählbare Sprache per Definition erfüllt.)

Beispiele für unentscheidbare Probleme in der Praxis

Folgende Probleme sind nur in Einzelfällen lösbar, nicht jedoch für beliebige Programme und Spezifikationen:

- Feststellen, ob eine bestimmte Codezeile in einem Programm jemals ausgeführt wird
- Feststellen, ob es eine Endlosschleife in einem Programm gibt
- Feststellen, ob 2 Programme die gleiche Wirkung (Semantik) haben
- Feststellen, ob ein Programm eine Spezifikation erfüllt
- Zu einer Spezifikation automatisch ein Programm generieren
- Feststellen, ob 2 Spezifikationen die gleiche Klasse von Programmen festlegen

Problemlandschaft



Normalform

Eine (deterministische) Turingmaschine ist in **Normalform**, wenn

- sie nur einen Endzustand besitzt,
- das Arbeitsband am Ende eines akzeptierenden Laufes der Turingmaschine leer ist und
- der letzte Übergang von der Gestalt $\delta(s, Z_2, Z_0) = (f, Z_0, S, R)$ ist, wobei f dieser einzige Endzustand und s ein beliebiger anderer Zustand ist.

Akzeptierte Sprachen

Eine formale Sprache heißt

0. **rekursiv aufzählbar**, wenn Sie von einer *Turingmaschine* akzeptiert wird
1. **kontextsensitiv**, wenn sie von einem *linear beschränkten Automaten (LBA)* akzeptiert wird
2. **kontextfrei**, wenn sie von einem *Kellerautomaten* akzeptiert wird
3. **regulär**, wenn Sie von einem *endlichen Automaten* akzeptiert wird.

Reguläre Sprachen

Reguläre Mengen (Sprachen)

Die Menge $\mathcal{L}_{\text{reg}}(\Sigma)$ der regulären Mengen (Sprachen) über Σ ist die kleinste Menge, sodass

- $\{\}, \{a\} \in \mathcal{L}_{\text{reg}}(\Sigma)$ für alle $a \in \Sigma$
- $A, B \in \mathcal{L}_{\text{reg}}(\Sigma) \Rightarrow A \cup B, A \cdot B, A^* \in \mathcal{L}_{\text{reg}}(\Sigma)$

Reguläre Ausdrücke (Algebraische Notation)

s	statt	$\{s\}$ für $s \in \Sigma$	$L_1 + L_2$	statt	$L_1 \cup L_2$
ε	statt	$\{\varepsilon\}$	$L_1 L_2$	statt	$L_1 \cdot L_2$
$\emptyset$	statt	$\{\}$	L^*	bleibt	L^*

* hat die höchste Priorität, $+$ die niedrigste.

Anmerkung: Genau genommen, ist ein regulärer Ausdruck E keine Sprache. Will man auf die Sprache Bezug nehmen, die von E beschrieben wird, sollte man eigentlich $L(E)$ verwenden.

Deterministischer endlicher Automat

Um das Verhalten eines DEA auf einer Zeichenkette formal zu beschreiben, erweitern wir die Übergangsfunktion δ auf beliebige Wörter aus Σ^*:

Erweiterte Übergangsfunktion

$\delta^*\colon Q \times \Sigma^* \to Q$

$$\delta^*(q, \varepsilon) = q, \qquad \delta^*(q, aw) = \delta^*(\delta(q, a), w)$$

für alle $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$.

Akzeptierte Sprache

$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$

Deterministische endliche Automaten (DEA)

DEA

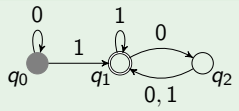
Ein deterministischer endlicher Automat (DEA) ist ein 5-Tupel

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F), \text{wobei}$$

Q ... endliche Menge von Zuständen
Σ ... Eingabealphabet
$\delta\colon Q \times \Sigma \to Q$ ... Übergangsfunktion (total)
$q_0 \in Q$ ... Anfangszustand
$F \subseteq Q$ ... Menge von Endzuständen

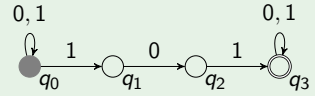
Nichtdeterministische endliche Automaten (NEA)

DEA



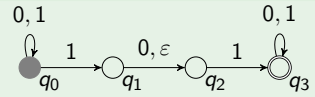
Von einem Zustand aus gibt es mit ein und demselben Eingabesymbol genau einen Folgezustand.

NEA



Von einem Zustand aus kann es mit ein und demselben Eingabesymbol mehrere Folgezustände geben.

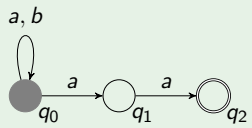
ε-NEA



Von einem Zustand aus kann es mit ein und demselben Eingabesymbol mehrere Folgezustände geben, Übergänge sind auch mit ε möglich (ε-Übergänge).

Determinisierung: Beispiel

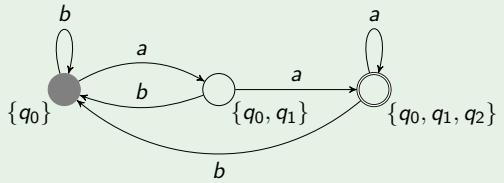
NEA:



δ	a	b
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{\}$
q_2	$\{\}$	$\{\}$

$\hat{\delta}$	a	b
SZ $\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$
EZ $\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$

DEA:



Abschlusseigenschaften regulärer Sprachen

$\mathcal{L}_{\text{reg}}(\Sigma)$ ist **abgeschlossen** gegenüber:

- Vereinigung, Verkettung, Stern-Operator: per Definition.
- Plus-Operator: $A^+ = A^* \cdot A$.
- Komplement bzgl. Σ^*: konstruiere DEA und vertausche End- und Nichtendzustände. (Falle nicht vergessen!)
- Durchschnitt: $A \cap B = \overline{\overline{A} \cup \overline{B}}$.
- Differenz: $A - B = A \cap \overline{B}$.
- Spiegelung: Konstruiere EA, vertausche Start- mit Endzustand, kehre alle Zustandsübergänge um.
- Homomorphismen: repräsentiere L durch regulären Ausdruck und ersetze alle Vorkommnisse von a durch Ausdruck für $h(a)$.

Homomorphismus

Seien Σ und Γ zwei Alphabete. **Homomorphismus**: $h : \Sigma \longrightarrow \Gamma^*$
Induktive Erweiterung auf Wörter über Σ:

1. $h(\varepsilon) = \varepsilon$
2. $h(wa) = h(w)h(a)$ für alle $w \in \Sigma^*$ und $a \in \Sigma$.

Anwendung von h auf jedes Wort der Sprache L:

$$h(L) = \{h(w) \mid w \in L\}$$

Beispiel

Sei $h : \{0,1\}^* \longrightarrow \{a,b\}^*$ ein Homomorphismus mit $h(0) = ab$ und $h(1) = \varepsilon$

Dann ist $h(0011) = abab$ und $h(L(10^*1)) = L((ab)^*)$

Entscheidungsprobleme

Folgende Probleme sind für reguläre Sprachen L, L' **entscheidbar**:

- Gehört ein Wort w der Sprache L an? (**Wortproblem**)
  Konstruiere einen DEA für L und prüfe, ob $\delta^*(q_0, w) \in F$.
- Ist L leer?
  Konstruiere einen DEA für L und prüfe, ob von q_0 aus ein Endzustand erreichbar ist.
- Ist L endlich oder unendlich?
  L ist unendlich gdw. der minimale DEA für L (abgesehen von der Falle!) einen Zyklus enthält.
- Gilt $L = L'$?
  Überprüfe, ob $L - L'$ und $L' - L$ leer sind.

Pumping Lemma für reguläre Sprachen

(Unendliche) Reguläre Sprachen haben eine spezielle Eigenschaft: Jedes Wort ab einer bestimmten Länge kann "aufgepumpt" werden, d.h. jedes solche Wort enthält ein Teilstück, das beliebig oft wiederholt werden kann, wobei die resultierenden Wörter noch immer in derselben Sprache liegen.

Pumping Lemma für reguläre Sprachen

Sei L eine unendliche reguläre Sprache. Dann gibt es eine (nur von L abhängige) Schranke $m > 0$ so, dass für jedes Wort $w \in L$ mit $|w| \geq m$ Wörter x, y, z so existieren, dass

$$w = xyz \qquad \text{mit } |xy| \leq m \text{ und } |y| > 0$$

sowie

$$w_i = xy^iz \in L \qquad \text{für alle } i \geq 0.$$

Reguläres Pumping Lemma - Beispiel

Zu zeigen: $L = \{a^n b^n \mid n \geq 0\}$ ist nicht regulär.

Beweis durch Widerspruch:

Angenommen L sei regulär. Für **beliebiges** m (Konstante aus dem Pumping Lemma) wähle **ein** $w \in L$ mit $|w| \geq m$, z.B.

$$w = a^m b^m$$

Betrachte **beliebige** Zerlegungen von w in xyz, wobei $|xy| \leq m$ und $|y| > 0$: xy kann nur aus Symbolen a bestehen.

Wähle **ein** i so, dass xy^iz nicht von der Gestalt $a^n b^n$ ist:

z.B. $i = 2$, dann müsste – wäre L regulär – auch $xy^2z = a^{m+|y|}b^m$ aus L sein, was aber nicht der Fall ist! Widerspruch! L kann nicht regulär sein.

Grammatik: Formale Definition

Grammatik

Grammatik $G = \langle N, T, P, S \rangle$, wobei

N ... endliche Menge von Nonterminalen (Variablen)
T ... endliche Menge von Terminalsymbolen ($N \cap T = \{\}$)
$P \subseteq (N \cup T)^+ \times (N \cup T)^*$ ... Produktionen
$S \in N$ ... Startsymbol

Wir schreiben
$\alpha \to \beta$ statt $(\alpha, \beta) \in P$
$\alpha \to \beta_1 \mid \cdots \mid \beta_n$ statt $\alpha \to \beta_1, \ldots, \alpha \to \beta_n$

Grammatiken: Beispiele

$\{a^n b^n c^n | n \geq 1\}$

$$G_3 = (\{S, A, C\}, \{a, b, c\}, P_3, S) \text{wobei}$$

$$P_3 = \{S \to abc, S \to aAbc, A \to aAbC, A \to abC, Cb \to bC, Cc \to cc\}$$

$$\mathcal{L}(G_3) = \{a^n b^n c^n | n \geq 1\}$$

Alle in G_3 möglichen Ableitungen sind von der Gestalt $S \Rightarrow abc$ bzw. für $n \geq 2$:

$$S \Rightarrow aAbc \Rightarrow^{n-2} a^{n-1}A(bC)^{n-2}bc \Rightarrow a^n(bC)^{n-1}bc \Rightarrow^* a^n b^n c^n$$

Grammatik-Typen: Typ-i-Grammatiken

Typ-i-Grammatiken

Sei $G = \langle N, T, P, S \rangle$ eine Grammatik. Dann heißt G auch **unbeschränkte Grammatik** (*Typ-0*).
Gilt für alle Produktionen $\alpha \to \beta \in P$

- $|\alpha| \leq |\beta|$ so heißt G **monoton**;
- $\alpha = uAv$ und $\beta = uwv$ für ein $A \in N$, $w \in (N \cup T)^+$ und $u, v \in (N \cup T)^*$ so heißt G **kontextsensitiv** (**Typ-1**);
- $A \to \beta$ für ein $A \in N$, so heißt G **kontextfrei** (**Typ-2**);
- $A \to aB$ oder $A \to \varepsilon$ für $A, B \in N$ und $a \in T$, so heißt G **regulär** (**Typ-3**).

Für monotone und kontextsensitive Grammatiken gilt: Kommt S nicht auf der rechten Seite einer Produktion vor, so ist auch die Produktion $S \to \varepsilon$ erlaubt.

Erzeugte Sprachen

Erzeugte Sprachen

Eine formale Sprache heißt **rekursiv aufzählbar**, **monoton**, **kontextsensitv**, **kontextfrei** bzw. **regulär**, wenn sie von einer Typ-0-, monotonen, Typ-1-, Typ-2-, bzw. Typ-3-Grammatik erzeugt wird.

Aufgrund der Definition können wir nun die einzelnen Sprachen aus den vorigen Beispielen klassifizieren:

Es ergibt sich, dass
$\mathcal{L}(G_1) = \{a^n | n \geq 0\}$ regulär
$\mathcal{L}(G_2) = \{a^n b^n | n \geq 0\}$ kontextfrei und
$\mathcal{L}(G_3) = \{a^n b^n c^n | n \geq 1\}$ monoton ist.

Reguläre Grammatiken

Reguläre Grammatiken [1]

Eine Grammatik heißt *regulär*, wenn alle Produktionen von der Form

$$A \to aB \qquad \text{oder} \qquad A \to \varepsilon$$

sind ($A, B \in N$, $a \in T$).

Alternativ:

Reguläre Grammatiken [2]

Eine Grammatik heißt *regulär*, wenn alle Produktionen von der Form

$$A \to aB \qquad \text{oder} \qquad A \to a$$

sind ($A, B \in N$, $a \in T$). Um auch das Leerwort erzeugen zu können, ist $S \to \varepsilon$ erlaubt, sofern S nicht auf der rechten Seite einer Produktion vorkommt.

Kontextfreie Grammatiken

Kontextfreie Grammatik

Eine Grammatik heißt *kontextfrei*, wenn alle Produktionen von der Form $A \to \beta$ sind, wobei $A \in N$ und $\beta \in (N \cup T)^*$.
Eine Sprache L heißt kontextfrei, wenn es eine kontextfreie Grammatik G gibt, sodass $L = \mathcal{L}(G)$.

Linksableitung:

$xAy \Rightarrow_L x\beta y$ falls $A \to \beta \in P$ und $x \in T^*$

Rechtsableitung:

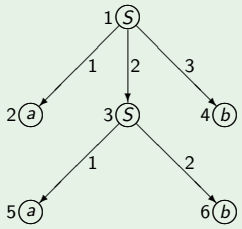
$xAy \Rightarrow_R x\beta y$ falls $A \to \beta \in P$ und $y \in T^*$

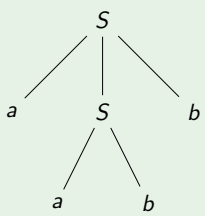
Parallelableitung:

$x_0 A_1 x_1 \cdots A_n x_n \Rightarrow_P x_0 \beta_1 x_1 \cdots \beta_n x_n$ falls
$A_1 \to \beta_1, \ldots, A_n \to \beta_n \in P$ und $x_0, \ldots, x_n \in T^*$

Ableitungsbäume: Beispiel

$G = (\{S\}, \{a, b\}, \{S \to aSb, S \to ab\}, S)$
Ableitungsbaum für G:





Front des Ableitungsbaums:
ergibt sich als Folge der Labels der Knoten 2, 5, 6, 4 zu *aabb*

Eindeutigkeit, (inhärente) Mehrdeutigkeit

Sei $G = \langle N, T, P, S \rangle$ eine kontextfreie Grammatik. G heißt **eindeutig**, wenn es zu jedem in G ableitbaren Terminalwort genau eine Linksableitung in G gibt. Ansonsten heißt G **mehrdeutig**.

Eine kontextfreie *Sprache* L heißt **inhärent mehrdeutig**, wenn jede Grammatik, die L erzeugt, mehrdeutig ist.

Vorbereitende Schritte zur Chomsky NF: Beispiel

Schritt 1

$G = \langle \{S, A, B, C, D\}, \{a\}, P, S \rangle$ mit
$P = \{S \to aA \mid B \mid D, A \to aB, B \to A, B \to \varepsilon, C \to a\}$

Wir untersuchen, ob aus jeder Variablen in G ein Terminalwort ableitbar ist:

- $N_1^{(1)} = \{B, C\}$
- $N_1^{(2)} = \{S, A\} \cup \{B, C\} = N_1^{(3)}$.

Aus D ist kein Terminalwort ableitbar, wir erhalten also

$$\begin{aligned} G_1 &= \langle N_1, \{a\}, P_1, S \rangle \text{ mit} \\ N_1 &= \{S, A, B, C\} \\ P_1 &= \{S \to aA \mid B, A \to aB, B \to A, B \to \varepsilon, C \to a\} \ . \end{aligned}$$

Nun ist in G_1 aus jeder Variablen ein Terminalwort ableitbar und es gilt $\mathcal{L}(G_1) = \mathcal{L}(G)$.

Vorbereitende Schritte zur Chomsky NF: Beispiel ctd.

Schritt 2

Ausgehend von G_1 überprüfen wir, ob alle Symbole vom Startsymbol S aus erreichbar sind:

- $V_2^{(1)} = \{S\}$
- $V_2^{(2)} = \{A, B, a\} \cup \{S\} = V_2^{(3)}$

C nicht vom Startsymbol erreichbar. Daher erhalten wir

$$\begin{aligned} G_2 &= \langle N_2, \{a\}, P_2, S \rangle \text{ mit} \\ N_2 &= \{S, A, B\} \\ P_2 &= \{S \to aA \mid B, A \to aB, B \to A, B \to \varepsilon\} \ . \end{aligned}$$

Mit G_2 haben wir nun also eine zu G äquivalente Grammatik ohne nutzlose Symbole und es gilt $\mathcal{L}(G_2) = \mathcal{L}(G)$.

Vorbereitende Schritte zur Chomsky NF: Beispiel ctd.

Schritt 3

Nun sehen wir uns die ε-Produktionen in G_2 an:

- $N_3^{(1)} = \{B\}$
- $N_3^{(2)} = \{S\} \cup \{B\}$ Also: $\varepsilon \in \mathcal{L}(G)$!

Wir erhalten somit

$$\begin{aligned} G_3 &= \langle N_3, \{a\}, P_3, S \rangle \text{ mit} \\ N_3 &= N_2 = \{S, A, B\} \\ P_3 &= \{S \to aA \mid B, A \to aB \mid a, B \to A\} \ . \end{aligned}$$

G_3 enthält nun weder nutzlose Symbole noch ε-Produktionen, und es gilt $\mathcal{L}(G_3) = \mathcal{L}(G) - \varepsilon$.

Vorbereitende Schritte zur Chomsky NF: Beispiel ctd.

Schritt 4

In G_3 gibt es noch die Einheitsproduktion $B \to A$, die wir nun eliminieren. Nachdem $B \Rightarrow A \Rightarrow aB$ bzw. $B \Rightarrow A \Rightarrow a$, ersetzen wir nun $B \to A$ durch $B \to aB \mid a$ sowie $S \to B$ durch $S \to aB \mid a$ und erhalten damit die reduzierte Grammatik

$$\begin{aligned} G_4 &= \langle N_4, \{a\}, P_4, S \rangle \text{ mit} \\ N_4 &= N_3 = N_2 = \{S, A, B\} \\ P_4 &= \{S \to aA \mid aB \mid a, A \to aB \mid a, B \to aB \mid a\} \end{aligned}$$

Wiederholt man die Schritte 1 und 2, so stellt man fest, dass in diesem Fall keine neuen nutzlosen Symbole erzeugt wurden.

Chomsky[4] Normalform

reduzierte Grammatik: Als Ergebnis der Schritte 1-4 erhalten wir eine sogenannte *reduzierte* Grammatik, also eine kontextfreie Grammatik, die weder nutzlose Symbole, noch ε- oder Einheitsproduktionen enthält.

Chomsky Normalform

Alle Produktionen besitzen die Form

$$A \to B\,C \qquad \text{oder} \qquad A \to a \qquad (a \in T,\ A, B, C \in N) \ .$$

Um das Leerwort erzeugen zu können, ist $S \to \varepsilon$ erlaubt, sofern S nicht auf der rechten Seite einer Produktion vorkommt.

Zu jeder kontextfreien Grammatik G kann man effektiv eine äquivalente Grammatik G' konstruieren, sodass $G' = \langle N', T', P', S' \rangle$ in Chomsky NF ist.

[4] Noam Chomsky, *1928

Chomsky NF - Beispiel ctd.

Schritt 5 - Chomsky NF

Wir konstruieren zu G_4 (die $\mathcal{L}(G) - \varepsilon$ erzeugt) eine reduzierte Grammatik G' in Chomsky-Normalform mit $\mathcal{L}(G') = \mathcal{L}(G)$:
Ersetze a durch X_a in allen Produktionen, in denen a nicht alleine auf der rechten Seite vorkommt:

$$\{S \to X_aA \mid X_aB \mid a, A \to X_aB \mid a, B \to X_aB \mid a, X_a \to a\}.$$

Nachdem $\varepsilon \in \mathcal{L}(G)$, muss noch $S \to \varepsilon$ hinzugefügt werden, also:

$$\begin{aligned} G' &= \langle \{S, A, B, X_a, Y_1\}, \{a\}, P', S \rangle \text{ mit} \\ P' &= \{S \to X_aA \mid X_aB \mid a \mid \varepsilon, A \to X_aB, \\ &\qquad A \to a, B \to X_aB \mid a, X_a \to a\} \end{aligned}$$

Greibach[5] Normalform für kf Grammatiken

Greibach Normalform:

Alle Produktionen haben die Form

$$A \to a\,w \qquad (a \in T,\ A \in N,\ w \in N^*)\ .$$

Erweiterte Greibach Normalform:

Alle Produktionen haben die Form

$$A \to a\,w \qquad (a \in T,\ A \in N,\ w \in (N \cup T)^*)\ .$$

Um das Leerwort erzeugen zu können, ist $S \to \varepsilon$ erlaubt, sofern S nicht auf der rechten Seite einer Produktion vorkommt.

Zu jeder kontextfreien Grammatik G kann man effektiv eine äquivalente Grammatik G' konstruieren, sodass G' in (erweiterter) Greibach NF ist.

[5] Sheila Greibach, *1939

Erweiterte Greibach Normalform - Beispiele

$L_1 = \left\{0^{2n}1^{4k}0^{2n} \mid n, k \geq 1\right\}$

Grammatik für L_1 in erweiterter Greibach Normalform:
$G_1 = \langle\{S, X\}, \{0, 1\}, P_1, S\rangle$ mit
$P_1 = \{S \to 0^2S0^2, S \to 0^2X0^2, X \to 1^4X, X \to 1^4\}$

Linksableitungen:
$S \Rightarrow^{n-1} 0^{2(n-1)}S0^{2(n-1)} \Rightarrow 0^{2n}X0^{2n} \Rightarrow^{k-1} 0^{2n}1^{4(k-1)}X0^{2n} \Rightarrow 0^{2n}1^{4k}0^{2n} \qquad \forall n, k \geq 1$

$L_2 = \{a^{n-4}b^ka^{5n} \mid n > 4, k \geq 3\}$

Grammatik für L_2 in erweiterter Greibach Normalform:
$G_2 = \langle\{S, A, B\}, \{a, b\}, P_2, S\rangle$ mit
$P_2 = \{S \to aAa^{25}, A \to aAa^5 \mid b^2B, B \to bB \mid b\}$

Abschlusseigenschaften von Sprachfamilien

	$\mathcal{L}_3$	$\mathcal{L}_2$	$\mathcal{L}_1$	$\mathcal{L}_0$
Vereinigung	ja	ja	ja	ja
Konkatenation	ja	ja	ja	ja
Kleenescher Stern	ja	ja	ja	ja
Komplement	ja	nein	ja	nein
Durchschnitt	ja	nein	ja	ja
Durchschnitt mit reg. Mengen	ja	ja	ja	ja
Homomorphismen	ja	ja	nein	ja
ε-freie Homomorphismen	ja	ja	ja	ja
inverse Homomorphismen	ja	ja	ja	ja
gsm-Abbildungen	ja	ja	nein	ja
ε-freie *gsm*-Abbildungen	ja	ja	ja	ja

Chomsky-Hierarchie

Sprachfamilie	Grammatiktyp	Maschinenmodell
$\mathcal{L}_0$ (rekursiv aufzählb.)	unbeschränkt	Turingmaschine (= EA + RAM)
$\mathcal{L}_1$ (kontextsensitiv)	kontextsensitiv monoton	Linear beschränkter Aut. (= EA + beschr.RAM)
$\mathcal{L}_2$ (kontextfrei)	kontextfrei	Kellerautomat (= EA + Stack)
$\mathcal{L}_3$ (regulär)	regulär	endlicher Automat (EA)

Chomsky Hierarchie

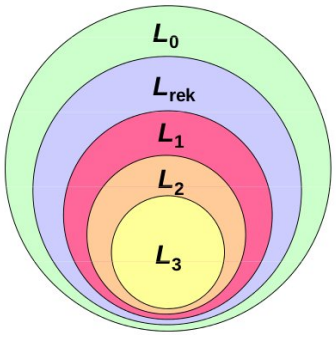
$$\mathcal{L}_3 \subsetneqq \mathcal{L}_2 \subsetneqq \mathcal{L}_1 \subsetneqq \mathcal{L}_{rec} \subsetneqq \mathcal{L}_0.$$

Chomsky-Hierarchie

Chomsky Hierarchie

$$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_{rec} \subset \mathcal{L}_0.$$

nicht rekursiv aufzählbare Sprachen



Satz von Chomsky-Schützenberger[7]

Jede kontextfreie Sprache ist homomorphes Bild des Durchschnitts einer regulären Sprache mit einer Dyck-Sprache:

Satz von Chomsky-Schützenberger

Eine Sprache L über Σ ist genau dann kontextfrei, wenn zu einem $n \geq 0$ ein Homomorphismus $h : \Gamma_n^* \longrightarrow \Sigma^*$ so existiert, dass

$$L = h(D_n \cap R),$$

wobei R eine reguläre Sprache über Γ_n bezeichnet.

$L = \{a^nb^n \mid n \geq 0\}$ ist kontextfrei, da

$$L = h(D_1 \cap \{(\}^*\{)\}^*)$$

wobei $h : \{(,)\}^* \longrightarrow \{a, b\}^*$ mit $h(() = a, \quad h()) = b$

[7] Marcel-Paul Schützenberger, 1920 - 1996

Entscheidungsprobleme für kontextfreie Sprachen

Entscheidbare Probleme:

- Ist die k.f. Sprache L leer/endlich/unendlich?
- Liegt ein Wort w in der k.f. Sprache L? (Wort-Problem)

Unentscheidbare Probleme:

- Gilt $L = \Sigma^*$?
- $L_1 = L_2$ (Äquivalenzproblem)?
- $L_1 \subseteq L_2$?
- $L_1 \cap L_2 = \{\}$?
- Ist L regulär?
- Ist $L_1 \cap L_2$ bzw. $\Sigma^* - L$ kontextfrei?

Monotone Grammatiken

Monotone Grammatik

Eine Grammatik heißt *monoton*, wenn für alle Produktionen $\alpha \to \beta$ die Länge von α kleiner gleich der Länge von β ist ($\alpha \neq \varepsilon$).
Kommt das Startsymbol S auf keiner rechten Seite vor, ist auch $S \to \varepsilon$ zugelassen.

$L = \{a^n b^n c^n \mid n \geq 0\}$

$G = \langle \{S, T, C\}, \{a, b, c\}, P, S \rangle$

wobei P folgende Produktionen enthält:

$$\begin{array}{lcl} S & \to & \varepsilon \mid abc \mid a\,T\,bc \\ T & \to & a\,T\,b\,C \mid ab\,C \end{array} \qquad \begin{array}{lcl} C\,b & \to & b\,C \\ C\,c & \to & cc \end{array}$$

Es gilt: $\mathcal{L}(G) = L$.

Kontextsensitive Grammatiken

Kontextsensitive Grammatik

Eine Grammatik heißt *kontextsensitiv*, wenn alle Produktionen die Form $uAv \to u\beta v$ besitzen, wobei $u, v \in (N \cup T)^*$, $A \in N$ und $\beta \in (N \cup T)^+$.
Kommt Startvariable S auf keiner rechten Seite vor, ist auch $S \to \varepsilon$ zugelassen.

Für jede Sprache, die von einer monotonen Grammatik erzeugt wird, gibt es auch eine kontextsensitive Grammatik, und umgekehrt.

Rekursive Sprachen

Rekursive Sprachen

Eine Sprache $L \in \mathcal{L}_0(\Sigma)$ heißt *rekursiv*, wenn auch das Komplement der Sprache $\overline{L} = \Sigma^* - L$, eine rekursiv aufzählbare Sprache ist, d.h., $\overline{L} \in \mathcal{L}_0(\Sigma)$. Die Menge der rekursiven Sprachen aus $\mathcal{L}_0(\Sigma)$ wird mit $\mathcal{L}_{rec}(\Sigma)$ bezeichnet, die Familie der rekursiven Sprachen mit $\mathcal{L}_{rec}$.

Die Komplexitätsklasse **P**

Die Klasse P

$\mathbf{P} = \{L \mid L$ ist in polynomiell beschränkter Zeit von einer DTM entscheidbar$\}$

Lässt man polynomielle Laufzeit-Unterschiede ausser Acht ("vernachlässigbar") so ist **P** invariant für alle Berechnungsmodelle, die polynomiell äquivalent zu DTM mit einem Band sind.

Die Klasse **P** umfasst genau jene Probleme, für die effiziente Algorithmen existieren.

Die Komplexitätsklasse **NP**

DTM ... Determinisitische Turingmaschine
NTM ... Nichtdetermininistische Turingmaschine

Die Klasse NP

$\mathbf{NP} = \{L \mid L$ ist in polynomiell beschränkter Zeit von einer NTM entscheidbar$\}$

Zu jeder Sprache $L \in \mathbf{NP}$ gibt es eine DTM, die L in höchstens exponentieller Zeit ($\to$ Klasse **EXPTIME**) entscheidet.

$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME}$

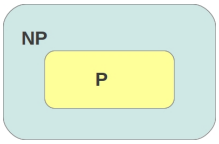
P versus NP

P ... Klasse von Sprachen, die in polynomieller Zeit entschieden werden können.
NP... Klasse von Sprachen, die in polynomieller Zeit verifiziert werden können.

Bisher konnte von keiner Sprache gezeigt werden, dass sie in **NP**, aber nicht in **P** liegt.

Dementsprechend ist die Frage, ob die Inklusion $\mathbf{P} \subseteq \mathbf{NP}$ echt ist (d.h., ob $\mathbf{P} = \mathbf{NP}$ gilt oder nicht) eines der bekanntesten offenen Probleme der Theoretischen Informatik und Mathematik.

Mutmaßliche Beziehung zwischen **P** und **NP**:



Polynomielle Reduktion

Gilt $A \leq_p B$ und $B \in \mathbf{NP}$, so ist auch $A \in \mathbf{NP}$.

Gilt $A \leq_p B$ und $B \in \mathbf{P}$, so ist auch $A \in \mathbf{P}$.

Beweis

Sei $A \leq_p B$ und $B \in \mathbf{P}$. Dann existieren Polynome $p(n)$ und $q(n)$ sowie Turingmaschinen M und N mit folgenden Eigenschaften:

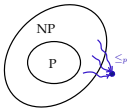
- M berechnet aus einer Eingabe $w \in \Sigma^*$ in Zeit $p(|w|)$ ein Wort $f(w)$ mit $w \in A \iff f(w) \in B$
  (Beachte: Da M in $p(|w|)$ Schritten nur eine Ausgabe der Länge höchstens $p(|w|)$ erzeugen kann, gilt $|f(w)| \leq p(|w|)$)
- N akzeptiert die Sprache B in Zeit $q(n)$

NP-harte und NP-vollständige Probleme

Die Komplexität einiger Probleme in **NP** spiegelt jene der gesamten Klasse wieder. Würde ein Polynomialzeit-Algorithmus für eines dieser Probleme existieren, wären alle Probleme in **NP** in polynomieller Zeit lösbar. Solche Probleme heißen **NP**-vollständig.

NP-hart

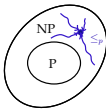
Ein Problem A heißt **NP**-hart genau dann, wenn $A' \leq_p A$ für alle Mengen $A' \in \mathbf{NP}$ gilt.



NP-vollständig

Ein Problem A heißt **NP**-vollständig genau dann, wenn

- A **NP**-hart ist und
- $A \in \mathbf{NP}$ gilt.



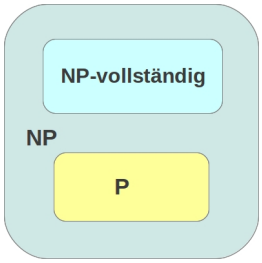
sofern $\mathbf{P} \neq \mathbf{NP}$

Informell: Wenn A **NP**-hart ist, dann ist es mindestens so schwer zu lösen wie irgendein Problem in **NP**. Ist A **NP**-vollständig, dann ist es eines der schwierigsten Probleme in **NP**.

P, NP, NP-vollständig

Sei A **NP**-vollständig. Dann gilt: $A \in \mathbf{P}$ genau dann wenn $\mathbf{P} = \mathbf{NP}$

Mutmaßliche Beziehung zwischen **P**, **NP** und **NP**-vollständig:



Beweis der NP-vollständigkeit

Seien A und B Sprachen aus **NP** mit $A \leq_p B$, und sei A **NP**-vollständig. Dann ist auch B **NP**-vollständig.

Damit erhalten wir folgendes Verfahren, um die **NP**-Vollständigkeit eines Problems B nachzuweisen:

1. Zeige, dass B in **NP** ist.
2. Zeige, dass irgendein Problem, dessen **NP**-Vollständigkeit bekannt ist, auf B polynomiell reduzierbar ist.

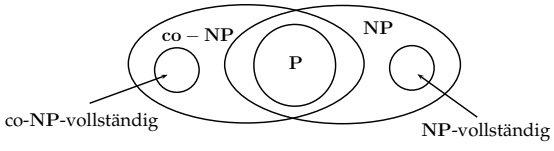
Co-NP

Co-NP

Co-**NP** enthält genau die Sprachen, deren Komplement in **NP** ist.

- **P** ist abgeschlossen unter Komplement. (Akzeptieren/Verwerfen einer DTM ist (bei rekursiven Sprachen) vertauschbar)
- Also: $\mathbf{P} \subseteq$ Co-**NP**
- Wenn $\mathbf{P} = \mathbf{NP}$, dann $\mathbf{P} = \mathbf{NP} =$ Co-**NP**

Vermutete Beziehung:



Die Klassen **PSPACE** und **NPSPACE**

Platzkomplexität

Sei M eine deterministische Turingmaschine (DTM), die eine Sprache **entscheidet**. Die Platzkomplexität von M ist die Abbildung $f : \mathbb{N} \longrightarrow \mathbb{N}$, wobei $f(n)$ die maximale Anzahl von Bandzellen ist, die M benötigt, um auf einem Input der Länge n zu halten.

Die Klasse **PSPACE**

PSPACE $= \{L \mid L$ ist von einer polynomiell platzbeschränkten DTM entscheidbar$\}$

Die Klasse **NPSPACE**

NPSPACE $= \{L \mid L$ ist von einer polynomiell platzbeschränkten NTM entscheidbar$\}$

Satz von Savitch

Satz von Savitch[4] (1970)

PSPACE = **NPSPACE**

Insgesamt erhalten wir:

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE} \subseteq \mathbf{EXPTIME}$$

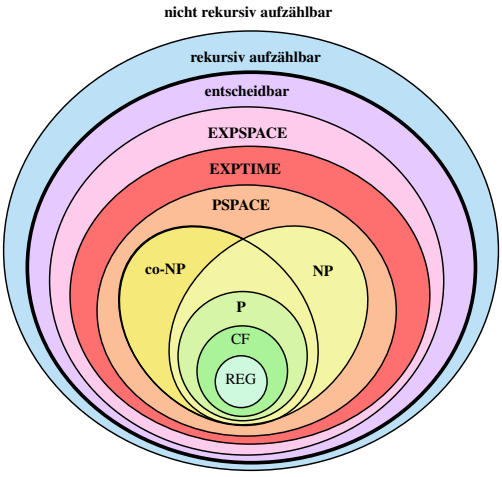
Es ist nur bekannt, dass mindestens eine dieser Inklusionen echt ist, da:

$$\mathbf{P} \neq \mathbf{EXPTIME}$$

Es wird jedoch vermutet, dass alle obigen Inklusionen echt sind.

[4] Walter J. Savitch (* 1943)

Zusammenfassung



Beispiele zu Variablenvorkommen

Variablen in unseren Beispielen wohlgeformter Formeln

- $F_1 = [\neg P(f(x), a) \wedge P(g(a, y), b)]$
  $FV(F_1) = \{x, y\}$ — zur Erinnerung: a, b sind Konstantensymbole
- $F_2 = \forall x[\neg P(f(x), a) \wedge P(g(a, x), b)]$
  $FV(F_2) = \{\}$, F_2 ist also geschlossen
- $F_3 = [\forall x \neg P(f(x), a) \wedge P(g(a, x), b)]$
  $FV(F_3) = \{x\}$, x kommt frei und gebunden vor
  Klammereinsparungsregeln: '[' und ']' könnten weggelassen werden (auch in F_1, nicht aber in F_2)
- $F_4 = \forall x \exists y \exists z(f(y) < a \vee x + 1 = b \vee z > f(1))$
  $FV(F_4) = \{\}$, F_4 ist also geschlossen
- $F_5 = \forall x\, P(y, z) \quad FV(F) = \{y, z\}$
- $F_6 = \forall x \exists x\, P(x, x) \quad FV(F) = \{\}$

Variablensubstitution

Notation:

$F(x/t)$ bezeichnet die Formel, die aus F entsteht, indem alle freien (und nur die freien) Vorkommen von x in F durch den Term t ersetzt werden.

Beispiele zur Variablensubstitution

Sei $F = \forall x[P(x, y) \supset Q(x, y)] \wedge R(x)$ und $t = f(y)$, dann gilt:

- $F(x/t) = \forall x[P(x, y) \supset Q(x, y)] \wedge R(f(y))$
- $F(y/t) = \forall x[P(x, f(y)) \supset Q(x, f(y))] \wedge R(x)$
- $F(x/t)(y/t) = \forall x[P(x, f(y)) \supset Q(x, f(y))] \wedge R(f(f(y)))$
- $F(y/t)(x/t) = \forall x[P(x, f(y)) \supset Q(x, f(y))] \wedge R(f(y))$

Beispiele von Modellstrukturen

Beachte: Für Zahlenbegriffe identifiziert man oft Menge und Struktur. Im Folgenden ist aber größere Genauigkeit gefordert!
Daher: Wir verwenden ab nun konsequent die Bezeichnungen $Z = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ und $\omega = \{0, 1, 2, \ldots\}$ für die Mengen und bezeichnen mit $\mathbb{Z}$ bzw. $\mathbb{N}$ die jeweiligen Modellstrukturen.

Ganze Zahlen:

$\mathbb{Z} = \langle Z;\ \{<\}, Z,\ \{+, -, *\}\rangle$
$<$ ... „kleiner"-Relation
$+, -, *$ ... übliche arithmetische Grundfunktionen
Beachte: jede ganze Zahl ist auch als Konstante verfügbar

Natürliche Zahlen:

$\mathbb{N} = \langle \omega;\ \{<\}, \{0, 1\},\ \{+, \dot{-}, *\}\rangle$
$+, *, <$ wie für $\mathbb{Z}$, aber $x \dot{-} y = \begin{cases} x - y & \text{für } x \geq y \\ 0 & \text{für } x < y \end{cases}$

Beachte: Nur 0 und 1 sind als Konstanten verfügbar!

Semantik – PL-Interpretationen

Definition

Eine prädikatenlogische Interpretation $\mathcal{I}$ über der Signatur $\Sigma = \langle PS_\Sigma, KS_\Sigma, FS_\Sigma \rangle$ ist ein Tupel $\langle D, \Phi, \xi \rangle$, wobei

- D: Domäne (Gegenstandsbereich): eine beliebige nicht-leere Menge
- Φ ist eine Signaturinterpretation:
  - $P \in PS_\Sigma$ n-stellig $\Longrightarrow \Phi(P)$: $D^n \to \{\mathbf{f}, \mathbf{t}\}$
  - $c \in KS_\Sigma \Longrightarrow \Phi(c) \in D$
  - $f \in FS_\Sigma$ n-stellig $\Longrightarrow \Phi(f)$: $D^n \to D$
- $\xi : IVS \to D$ ist eine Variablenbelegung

$PINT_\Sigma$ ... Menge aller PL-Interpretationen über der Signatur Σ

Beachte: Jede PL-Interpretation $\mathcal{I}$ bestimmt eine Modellstruktur $\mathcal{D}_\mathcal{I} = \langle D; P_D, K_D, F_D \rangle$, wobei
$P_D = \{\Phi(P) \mid P \in PS_\Sigma\}$, $K_D = \{\Phi(c) \mid c \in KS_\Sigma\}$, $F_D = \{\Phi(f) \mid f \in FS_\Sigma\}$.
D.h.: Jedes $\mathcal{I}$ entspricht einer Modellstruktur + Variablenbelegung
Man spricht vom Interpetieren bzw. Ausdrücken über $\mathcal{D}_\mathcal{I}$ (z.B. $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{S}$). 385

Semantik — Ein einfaches Beispiel

Auswertung der PL-Formel $F = \forall x \exists y \neg P(x, f(y))$
in der Interpretation $\mathcal{I} = \langle \{3, 4\}, \Phi, \xi \rangle$, wobei:
$\Phi(P) =$"$<$" (P bedeudet "kleiner als"), $\Phi(f) = id$ (Identitätsfunktion)
$\xi(v) = 3$ für alle $v \in IVS$ – hier aber irrelevant, da $FV(F) = \{\}$!

$$
\begin{aligned}
\mathrm{val}_\mathcal{I}(F) = \mathbf{t} &\overset{MP8}{\iff} \mathrm{val}_{\mathcal{I}'}(\exists y \neg P(x, f(y))) = \mathbf{t} \text{ für alle } \mathcal{I}' \overset{x}{\sim} \mathcal{I} \\
&\overset{MP9}{\iff} \mathrm{val}_{\mathcal{I}''}(\neg P(x, f(y))) = \mathbf{t} \text{ für ein } \mathcal{I}'' \overset{y}{\sim} \mathcal{I}' \\
&\overset{MP4}{\iff} \mathrm{val}_{\mathcal{I}''}(P(x, f(y))) = \mathbf{f} \\
&\overset{MP2}{\iff} \mathcal{M}_\mathcal{T}(\xi'', x) \geq \mathcal{M}_\mathcal{T}(\xi'', f(y)) \quad \text{für } \mathcal{I}'' = \langle D, \Phi, \xi'' \rangle \\
&\overset{MT1,MT3}{\iff} \xi''(x) \geq id(\mathcal{M}_\mathcal{T}(\xi'', y)) \iff \xi''(x) \geq \xi''(y)
\end{aligned}
$$

$\mathrm{val}_\mathcal{I}(F) = \mathbf{t}$, da zu jedem $\mathcal{I}' \overset{x}{\sim} \mathcal{I}$ ein $\mathcal{I}'' \overset{y}{\sim} \mathcal{I}'$ mit $\xi''(x) \geq \xi''(y)$ existiert 387

Semantische Grundbegriffe der PL (1)

Eine Formel $F \in \mathcal{PF}_\Sigma$ heißt

- erfüllbar, wenn $\mathrm{val}_\mathcal{I}(F) = \mathbf{t}$ für ein $\mathcal{I} \in PINT_\Sigma$, $\mathcal{I}$ heißt Modell von F, man sagt auch: $\mathcal{I}$ erfüllt F;
- widerlegbar, wenn $\mathrm{val}_\mathcal{I}(F) = \mathbf{f}$ für ein $\mathcal{I} \in PINT_\Sigma$, $\mathcal{I}$ heißt Gegenbeispiel zu F;
- unerfüllbar, wenn $\mathrm{val}_\mathcal{I}(F) = \mathbf{f}$ für alle $\mathcal{I} \in PINT_\Sigma$ (also: alle Interpretationen sind Gegenbeispiele);
- (logisch bzw. allgemein) gültig, wenn $\mathrm{val}_\mathcal{I}(F) = \mathbf{t}$ für alle $\mathcal{I} \in PINT_\Sigma$ (also: alle Interpretationen sind Modelle).

Eine Menge $\mathcal{F} \subseteq \mathcal{PF}_\Sigma$ heißt erfüllbar, wenn es ein $\mathcal{I} \in PINT_\Sigma$ gibt, sodass $\mathrm{val}_\mathcal{I}(F) = \mathbf{t}$ für alle $F \in \mathcal{F}$. Wenn es hingegen zu allen $\mathcal{I} \in PINT_\Sigma$ ein $F \in \mathcal{F}$ gibt, sodass $\mathrm{val}_\mathcal{I}(F) = \mathbf{f}$, dann ist $\mathcal{F}$ unerfüllbar.

Beachte: $\mathcal{F}$ kann unerfüllbar sein, obwohl alle $F \in \mathcal{F}$ erfüllbar sind.
Erfüllbarkeit von $\mathcal{F}$ erfordert ein gemeinsames Modell seiner Elemente. 388

Semantische Grundbegriffe der PL (2)

Bezogen auf eine gegebene Modellstruktur $\mathcal{D}$ und eine entsprechende Signaturinterpretation Φ heißt $F \in \mathcal{PF}_\Sigma$:

- erfüllbar in $\mathcal{D}$ (bezüglich Φ), wenn $\mathrm{val}_\mathcal{I}(F) = \mathbf{t}$ für ein $\mathcal{I}$ mit $\mathcal{D}_\mathcal{I} = \mathcal{D}$ (mit Φ als Signaturinterpretation), $\mathcal{I}$ heißt Modell in $\mathcal{D}$ von F;
- widerlegbar in $\mathcal{D}$, wenn $\mathrm{val}_\mathcal{I}(F) = \mathbf{f}$ für ein $\mathcal{I}$ mit $\mathcal{D}_\mathcal{I} = \mathcal{D}$, $\mathcal{I}$ heißt Gegenbeispiel in $\mathcal{D}$ zu F;
- unerfüllbar in $\mathcal{D}$ (bzgl. Φ), wenn $\mathrm{val}_\mathcal{I}(F) = \mathbf{f}$ für alle $\mathcal{I}$ mit $\mathcal{D}_\mathcal{I} = \mathcal{D}$;
- gültig in $\mathcal{D}$, wenn $\mathrm{val}_\mathcal{I}(F) = \mathbf{t}$ für alle $\mathcal{I}$ mit $\mathcal{D}_\mathcal{I} = \mathcal{D}$.

Einfache Beispiele:

- $x + y = y + x$ ist gültig in $\mathbb{Z}$ und $\mathbb{N}$ (bezogen auf die Standard-Signaturinterpretation), aber nicht allgemein gültig.
- $1 + x = x$ ist erfüllbar, aber nicht in $\mathbb{Z}$ oder in $\mathbb{N}$.
- $\forall x \exists y\, x + y = z$ ist erfüllbar in $\mathbb{Z}$, aber nicht in $\mathbb{N}$. 390

Regeln des (aussagenlogischen) Tableau-Kalküls

Wir arbeiten in einer eingeschränkten Syntax: nur $\vee, \wedge, \supset, \neg$

$$
\frac{\mathbf{f} : A \vee B}{\begin{array}{l}\mathbf{f} : A \\ \mathbf{f} : B\end{array}} \qquad
\frac{\mathbf{t} : A \vee B}{\mathbf{t} : A \mid \mathbf{t} : B} \qquad
\frac{\mathbf{f} : A \wedge B}{\mathbf{f} : A \mid \mathbf{f} : B} \qquad
\frac{\mathbf{t} : A \wedge B}{\begin{array}{l}\mathbf{t} : A \\ \mathbf{t} : B\end{array}}
$$

$$
\frac{\mathbf{f} : A \supset B}{\begin{array}{l}\mathbf{t} : A \\ \mathbf{f} : B\end{array}} \qquad
\frac{\mathbf{t} : A \supset B}{\mathbf{f} : A \mid \mathbf{t} : B} \qquad
\frac{\mathbf{f} : \neg A}{\mathbf{t} : A} \qquad
\frac{\mathbf{t} : \neg A}{\mathbf{f} : A}
$$

Beachte:

- Je zwei Regeln: eine für $\mathbf{f} : A * B$, eine für $\mathbf{t} : A * B$ ($\mathbf{f} : \neg A$ / $\mathbf{t} : \neg A$).
- Regeln aus den Wahrheitsfunktionen (*or*, *and*, *implies*, *not*) ablesbar.
- Beweise sind auf den Kopf gestellte Bäume (*downward trees*).
- Jede dieser Expansionsregeln muss auf jedes Vorkommen einer bewerteten Formel nur einmal angewendet werden.
  $\Longrightarrow$ Jeder Beweisversuch terminiert. 433

Korrektheit und Vollständigkeit des Tableau-Kalküls (1)

- Ein Tableau-Ast heißt geschlossen[a], wenn auf ihm für eine Formel F sowohl $\mathbf{t}$:F als auch $\mathbf{f}$:F vorkommt (Abschlussregel).
- Ein Tableau heißt geschlossen, wenn alle Äste geschlossen sind.
- Ein nicht geschlossener/s Ast/Tableau heißt offen.
- Ein Tableau-Beweis einer Konsequenzbehauptung $F_1, \ldots, F_n \models G$ ist ein geschlossenes Tableau, das mit den Annahmen $\mathbf{t}$:F_1, ..., $\mathbf{t}$:F_n, $\mathbf{f}$:G beginnt. Spezialfall: Ein Tableau-Beweis [der Gültigkeit] von F ist ein geschlossenes Tableau mit Wurzel $\mathbf{f}$:F.

[a] "geschlossen" hat hier nichts mit geschlossenen Formeln zu tun!

Satz (Korrektheit):

Wenn ein entsprechender Tableau-Beweis existiert, dann $F_1, \ldots, F_n \models G$.

Satz (Vollständigkeit):

Wenn $F_1, \ldots, F_n \models G$, dann existiert ein entsprechender Tableau-Beweis. 434

Korrektheit und Vollständigkeit des Tableau-Kalküls (3)

Korrektheitsbeweis (Skizze):
Die Korrektheit des Tableau-Kalküls ist äquivalent zu:

Wenn ein Tableau mit den Annahmen $\mathbf{t}{:}F_1,\ldots$, $\mathbf{t}{:}F_n, \mathbf{f}{:}G$ geschlossen ist, dann gibt es kein Gegenbeispiel I zu $F_1, \ldots, F_n \models G$.

Wir gehen wieder indirekt vor und nehmen an, dass es doch so ein I gibt.
Es gelte also $\mathrm{val}_I(F_1) = \mathbf{t}, \ldots, \mathrm{val}_I(F_n) = \mathbf{t}$ und $\mathrm{val}_I(G) = \mathbf{f}$.
Wegen des Lemmas gilt dann aber für jeden Ast Γ des Tableau:
$\mathrm{val}_I(F) = \mathbf{t}\,[\mathbf{f}]$ wenn $\mathbf{t}{:}F\,[\mathbf{f}{:}F]$ auf Γ liegt.
Da das Tableau geschlossen, ist auch jeder Ast Γ geschlossen.
Es gibt also zu jedem Γ ein F, sodass $\mathbf{t}{:}F$ und gleichzeitig $\mathbf{f}{:}F$ auf Γ liegen.
Das bedeutet, dass das angenommene Gegenbeispiel I nicht existiert. QED.

Folgerungen:

- Entscheidungsverfahren:
  Vollständige Expansion liefert einen Beweis oder ein Gegenbeispiel.
- *Don't care*-Indeterminismus:
  Ergebnis ist unabhängig von der Reihenfolge der Regelanwendungen.

Quantoren-Regeln des Tableau-Kalküls

Σ^{par} ... Signatur Σ erweitert um Parameter $c, d, e, \ldots$
$\mathbf{t} : \forall x F$ und $\mathbf{f} : \exists x F$ heißen γ-Formeln
$\mathbf{f} : \forall x F$ und $\mathbf{t} : \exists x F$ heißen δ-Formeln

γ-Regeln: $\dfrac{\mathbf{t} : \forall x F}{\mathbf{t} : F(x/t)}$ $\qquad$ $\dfrac{\mathbf{f} : \exists x F}{\mathbf{f} : F(x/t)}$

für beliebige geschlossene (=variablenfreie) Terme t über Σ^{par}

δ-Regeln: $\dfrac{\mathbf{f} : \forall x F}{\mathbf{f} : F(x/c)}$ $\qquad$ $\dfrac{\mathbf{t} : \exists x F}{\mathbf{t} : F(x/c)}$

für einen neuen Parameter c in Σ^{par}

Die α- und β-Regeln (AL) bleiben unverändert!

Achtung: γ-Regeln müssen i.A. auf die selbe γ-Formel öfters, mit verschiedenen t angewendet werden. (Im Gegensatz zur δ-Regel.)
Überlege: Warum ist das so?

PL-Tableaux – Beispiele (4)

Gültigkeit von $\forall x[P(x) \vee Q(x)] \supset [\exists x P(x) \vee \forall x Q(x)]$

(1)	$\mathbf{f} : \forall x[P(x) \vee Q(x)] \supset [\exists x P(x) \vee \forall x Q(x)]$				Annahme
(2)	$\mathbf{t} : \forall x[P(x) \vee Q(x)]$				von 1 (γ-Formel)
(3)	$\mathbf{f} : \exists x P(x) \vee \forall x Q(x)$				von 1
(4)	$\mathbf{f} : \exists x P(x)$				von 3 (γ-Formel)
(5)	$\mathbf{f} : \forall x Q(x)$				von 3 (δ-Formel)
(6)	$\mathbf{f} : Q(c)$				von 5
(7)	$\mathbf{t} : P(c) \vee Q(c)$				von 2
(8)	$\mathbf{t} : P(c)$	von 7	(9)	$\mathbf{t} : Q(c)$	von 7
(10)	$\mathbf{f} : P(c)$	von 4		$\times$	Wid.: 6/9
	$\times$	Wid.: 8/10			

Eine sehr nützliche Heuristik:

"δ vor γ": immer wenn möglich zunächst die δ-Regel anwenden, da damit die Wahl geeigneter "Zeugen-Terme" für γ-Formeln unterstützt wird.

PL-Tableaux – Beispiele (7)

Beweis von $\forall x[P(x) \vee Q(f(x))], \neg\exists x P(x) \models \exists x Q(x)$

(1)	$\mathbf{t} : \forall x[P(x) \vee Q(f(x))]$				Ann. (γ-Formel)
(2)	$\mathbf{t} : \neg\exists x P(x)$				Ann.
(3)	$\mathbf{f} : \exists x Q(x)$				Ann. (γ-Formel)
(4)	$\mathbf{f} : \exists x P(x)$				von 2 (γ-Formel)
(5)	$\mathbf{t} : P(c) \vee Q(f(c))$				von 1
(6)	$\mathbf{t} : P(c)$	von 5	(7)	$\mathbf{t} : Q(f(c))$	von 5
(8)	$\mathbf{f} : P(c)$	von 4	(9)	$\mathbf{f} : Q(f(c))$	von 3
	$\times$	Wid.: 6/8		$\times$	Wid.: 7/9

Beachte: Da für die γ-Formel (1) in (5) ein geschlossener Term gewählt werden muss, muss hier eine neue Konstante (c) eingeführt werden.

PL-Tableaux – Beispiele (8)

Ein etwas interessantere Aufgabe:

Zeigen Sie mit dem Tableau-Kalkül, dass jede transitive, symmetrische und serielle Relation auch reflexiv ist.
Also:

$$\mathit{trans}, \mathit{sym}, \mathit{ser} \models \mathit{refl}$$

wobei

$$\begin{aligned} \mathit{trans} &= \forall x \forall y \forall z[(R(x,y) \wedge R(y,z)) \supset R(x,z)] \\ \mathit{sym} &= \forall x \forall y[R(x,y) \supset R(y,x)] \\ \mathit{ser} &= \forall x \exists y R(x,y) \\ \mathit{refl} &= \forall x R(x,x) \end{aligned}$$

PL-Tableaux – Beispiele (8) [Forts.]

Tableau für $\mathit{trans}, \mathit{sym}, \mathit{ser} \models \mathit{refl}$

(1)	$\mathbf{t} : \forall x \forall y \forall z[(R(x,y) \wedge R(y,z)) \supset R(x,z)]$					Ann.(γ)
(2)	$\mathbf{t} : \forall x \forall y[R(x,y) \supset R(y,x)]$					Ann.(γ)
(3)	$\mathbf{t} : \forall x \exists y R(x,y)$					Ann.(γ)
(4)	$\mathbf{f} : \forall x R(x,x)$					Ann.(δ)
(5)	$\mathbf{f} : R(c,c)$					von 4
(6)	$\mathbf{t} : \exists y R(c,y)$					von 3(δ)
(7)	$\mathbf{t} : R(c,d)$					von 6
(8)	$\mathbf{t} : \forall y[R(c,y) \supset R(y,c)]$					von 2(γ)
(9)	$\mathbf{t} : R(c,d) \supset R(d,c)$					von 8
(10)	$\mathbf{t} : \forall y \forall z[(R(c,y) \wedge R(y,z)) \supset R(c,z)]$					von 1(γ)
(11)	$\mathbf{t} : \forall z[(R(c,d) \wedge R(d,z)) \supset R(c,z)]$					von 10(γ)
(12)	$\mathbf{t} : (R(c,d) \wedge R(d,c)) \supset R(c,c)$					von 11
(13)	$\mathbf{f}{:}R(c,d)$ v. 9	(14)	$\mathbf{t} : R(d,c)$			von 9
	$\times$ (7/13)	(15)	$\mathbf{f} : R(c,d) \wedge R(d,c)$ v. 12		(16) $\mathbf{t}{:}R(c,c)$	von 12
		(17)	$\mathbf{f}{:}R(c,d)$	(18) $\mathbf{f}{:}R(d,c)$	$\times$ (5/16)	
			$\times$ (7/17)	$\times$ (14/18)		

PL-Tableaux mit Gleichheit – Beispiele (1)

Reflexivität, Symmetrie und Transitivität der Gleichheit

(1)	$\mathbf{f} : \forall x\, x = x$	Ann.(δ)
(2)	$\mathbf{f} : a = a$	von 1
	$\times$	$AB^{=}$

(1)	$\mathbf{t} : a = b$	Ann.
(2)	$\mathbf{f} : b = a$	Ann.
(3)	$\mathbf{f} : a = a$	$S^{=}$: 1→2
	$\times$	$AB^{=}$

(1)	$\mathbf{t} : a = b$	Ann.
(2)	$\mathbf{t} : b = c$	Ann.
(3)	$\mathbf{f} : a = c$	Ann.
(4)	$\mathbf{f} : b = c$	$S^{=}$: 1→3
	$\times$	Wid. 2/4

Beachte: In den Tableaux für Symmetrie und Transitivität könnten statt den Konstanten a, b, c beliebige Terme stehen.
Alternativ könnte man mit $\mathbf{f} : \forall x \forall y \forall z[(x = y \wedge y = z) \supset x = z]$ beginnen.

PL-Tableaux – Zusammenfassung

- PL-Tableaux sind Bäume von bewerteten geschlossenen Formeln.
- Ein (indirekter) Beweis liegt vor, wenn alle Äste Widersprüche enthalten, d.h. wenn all Äste geschlossen sind.
- γ-Regeln sind im Allgemeinen mehrmals – auch auf dem selben Ast – auf die selbe Formel anzuwenden.
- Im Gegensatz zu AL-Tableaux, sind offene Äste i.A. unendlich lang.
- PL ist unentscheidbar; daher kann es auch keine berechenbare Grenze für die maximale Größe eines Tableau-Beweises (abhängig von der Größe der initialen Annahmen) geben.
- Der PL-Tableau-Kalkül ist vollständig und korrekt:
  Ein Tableau kann genau dann geschlossen werden wenn die entsprechende Konsequenzbehauptung stimmt, also die initialen Annahmen unerfüllbar sind.
- Beweissuche mit Tableau kann (viel besser als für ND) automatisiert werden. In der Praxis verwendet man dazu die Variante der "free variable tableaux", die auf dem Unifikationsprinzip beruhen.

Die Unentscheidbarkeit der Prädikatenlogik

Zur Erinnerung:

Entscheidbare (= rekursive) Sprachen

Eine formale Sprache heißt **rekursiv** oder **entscheidbar** wenn sie von einer (deterministischen) Turingmaschine akzeptiert wird, die **immer hält**.
Man sagt dann auch: die TM **entscheidet** die Sprache.

Satz (Unentscheidbarkeit prädikatenlogischer Gültigkeit)

Die Menge der gültigen PL-Formeln ist keine entscheidbare Sprache.

Wegen des Deduktionstheorems und der Church-Turing-These gilt auch:

Satz (Unentscheidbarkeit prädikatenlogischer Konsequenz)

Es gibt kein Verfahren, das für beliebige PL-Formeln $F_1, \ldots F_n, G$ feststellen kann, ob $F_1, \ldots F_n \models G$ oder nicht.

Überlege: Warum und wie ergibt sich der zweite Satz aus dem ersten?

Zuweisung (4)

$$\{x + 5 = 7\} \qquad x \leftarrow x + 5 \qquad \{x = 7\}$$

Wenn $I(x) + 5 = 7$ und $I'(x) = I(x) + 5$ dann $I'(x) = 7$.

Verallgemeinerung der Beobachtung:
$\{G\left[\begin{smallmatrix} e \\ v \end{smallmatrix}\right]\}\, v \leftarrow e\, \{G\}$ ... korrektes Axiom

$G\left[\begin{smallmatrix} e \\ v \end{smallmatrix}\right]$ ... Formel, die aus G in zwei Schritten entsteht:

1. Benenne alle gebundenen Variablen in G so um, dass weder v noch die Variablen in e gebunden in G vorkommen.
2. Ersetze in G alle v durch e.

$\{y = 2(2x)\}\, x \leftarrow 2x\, \{y = 2x\}$ ✓

$\{y = 8(x/2)\}\, x \leftarrow x/2\, \{y = 8x\}$ ✓

$$\frac{x = 2 \supset 3 = 3 \quad \{3 = 3\}\, x \leftarrow 3\, \{x = 3\}}{\{x = 2\}\, x \leftarrow 3\, \{x = 3\}}\ (\text{imp}) \qquad \checkmark$$

Hintereinanderausführung (begin-end)

Angenommen $\{F\}\, \alpha\, \{G\}$ und $\{G\}\, \beta\, \{H\}$ sind beide wahr.

Dann gilt für jede Wahrheitsbelegung I:

- Aus $\mathcal{M}(I, F) = \mathbf{t}$ folgt $\mathcal{M}(I', G) = \mathbf{t}$, wobei $I' = \mathcal{M}(I, \alpha)$.
- Aus $\mathcal{M}(I', G) = \mathbf{t}$ folgt $\mathcal{M}(I'', H) = \mathbf{t}$, wobei $I'' = \mathcal{M}(I', \beta)$.

Zusammengenommen ergibt sich:

- Aus $\mathcal{M}(I, F) = \mathbf{t}$ folgt $\mathcal{M}(I'', H) = \mathbf{t}$,
  wobei $I'' = \mathcal{M}(I', \beta) = \mathcal{M}(\mathcal{M}(I, \alpha), \beta) = \mathcal{M}(I, \texttt{begin}\ \alpha\,;\beta\ \texttt{end})$.

Somit ist die Korrektheitsaussage $\{F\}\, \texttt{begin}\ \alpha\,;\beta\ \texttt{end}\, \{H\}$ wahr.

Regel für begin-end-Blöcke

$$\frac{\{F\}\, \alpha\, \{G\} \qquad \{G\}\, \beta\, \{H\}}{\{F\}\, \texttt{begin}\ \alpha\,;\beta\ \texttt{end}\, \{H\}}\ (\text{be})$$

Problem: Wie lässt sich die Interpolante G bestimmen?

Beispiel: Wertetausch (Beweis mit Annotationen)

$\{\, P\colon x = x_0 \wedge y = y_0 \,\}$
$\{\, F_3\colon Q\left[\begin{smallmatrix} t \\ y \end{smallmatrix}\right]\left[\begin{smallmatrix} y \\ x \end{smallmatrix}\right]\left[\begin{smallmatrix} x \\ t \end{smallmatrix}\right] \,\}$ zw ↑
`begin begin`
$t \leftarrow x$;
$\{\, F_2\colon Q\left[\begin{smallmatrix} t \\ y \end{smallmatrix}\right]\left[\begin{smallmatrix} y \\ x \end{smallmatrix}\right] \,\}$ zw ↑
$x \leftarrow y$
`end`;
$\{\, F_1\colon Q\left[\begin{smallmatrix} t \\ y \end{smallmatrix}\right] \,\}$ zw ↑
$y \leftarrow t$
`end`
$\{\, Q\colon x = y_0 \wedge y = x_0 \,\}$

Implikationsregel: Beweise die Implikation

$$P \supset F_3\colon \quad (x = x_0 \wedge y = y_0) \supset Q\left[\begin{smallmatrix} t \\ y \end{smallmatrix}\right]\left[\begin{smallmatrix} y \\ x \end{smallmatrix}\right]\left[\begin{smallmatrix} x \\ t \end{smallmatrix}\right]$$

Beispiel: Maximum (1)

$\{\top\}$
if $x \leq y$ **then**
  $\{\top \wedge x \leq y\}$ if$\downarrow$
  $\{y = \max(x,y)\}$ zw$\uparrow$
  $z \leftarrow y$
  $\{z = \max(x,y)\}$ fi$\uparrow$
else
  $\{\top \wedge x \not\leq y\}$ if$\downarrow$
  $\{x = \max(x,y)\}$ zw$\uparrow$
  $z \leftarrow x$
  $\{z = \max(x,y)\}$ fi$\uparrow$
// Ende **if**
$\{z = \max(x,y)\}$

Es bleibt zu zeigen:

$\top \wedge x \leq y$
$\quad \supset y = \max(x,y)$ imp

und

$\top \wedge x \not\leq y$
$\quad \supset x = \max(x,y)$ imp

Beispiel: Maximum (2)

$\{\top\}$
$\{(x \leq y \supset y = \max(x,y)) \wedge (x \not\leq y \supset x = \max(x,y))\}$ if$\uparrow$
if $x \leq y$ **then**
  $\{y = \max(x,y)\}$ zw$\uparrow$
  $z \leftarrow y$
  $\{z = \max(x,y)\}$ fi$\uparrow$
else
  $\{x = \max(x,y)\}$ zw$\uparrow$
  $z \leftarrow x$
  $\{z = \max(x,y)\}$ fi$\uparrow$
// Ende **if**
$\{z = \max(x,y)\}$

Es bleibt zu zeigen:

$$\top \supset ((x \leq y \supset y = \max(x,y)) \wedge (x \not\leq y \supset x = \max(x,y)))$$

(Implikationsregel)

Partielle Korrektheit von Schleifen

$$\frac{\{Inv \wedge e\}\,\alpha\,\{Inv\}}{\{Inv\}\ \texttt{while}\ e\ \texttt{do}\ \alpha\ \{Inv \wedge \neg e\}}\ (\text{wh})$$

Annotierung:

$$\begin{array}{lll} & \texttt{while}\ e\ \texttt{do} & \cdots \\ \mapsto \{Inv\} & \texttt{while}\ e\ \texttt{do}\,\{Inv \wedge e\} & \cdots\ \{Inv\}\ \ \{Inv \wedge \neg e\} \end{array} \quad (\text{wh})$$

Inv: "Schleifeninvariante"; gilt vor, während und nach der Schleife.

The Schleifenaussage ist korrekt, wenn eine Ableitung mit irgendeiner Formel *Inv* möglich ist.
Das Finden einer geeigneten Invarianten kann schwierig sein.

Termination

while $y \neq 0$ **do**
  begin
  $z \leftarrow z + x$;
  $y \leftarrow y - 1$
  end

Warum terminiert diese Schleife?

- Schleifenbedingung ist falsch, falls $I(y) = 0$.
- Schleifenanweisungen verringern den Wert von y um 1.
- Daher terminiert die Schleife, wenn y zu Beginn nicht negativ ist.

Allgemein:

- Finde Ausdruck für die „Größe" der aktuellen Wertebelegung $|I| = I(y)$
- Zeige, dass sich die Größe in jedem Durchlauf ganzzahlig verringert.
- Zeige, dass die Größe nach unten beschränkt ist. $|I| \geq 0$

Dann terminiert die Schleife sicher.