

**Legend:**

- schlecht
- gut
- Nicht-verhorrende Grammatiken
- Satz / Definition / Titel
- Kapitel
- Beispiel
- Determinierung
- Legende:

**1. Sprachen und Grammatiken**

$U = \begin{cases} A \notin N: \text{Unterscheidung } 0 \vee 1 & M^+: \text{Alles außer } \emptyset \\ A \in N: \text{Unterscheidung } 2 \vee 3 & M^*: \text{Alles} \end{cases}$

**Chomsky-Hierarchie  $u \rightarrow v$**

0	Allgemein	• Alles was nicht kontextsensitiv ist.
1	Kontext-sensitiv	• $u = u_1 A w_2; v = w_1 w_2$ ( $A$ wird durch $v$ ersetzt) z.B.: $w_1 A w_2 \rightarrow w_1 a b c w_2$ <span style="color:red">Genau 1 Variable wird ersetzt!</span>
2	Kontextfrei	• Alles was nicht regulär ist $u \in N, v \in (N \cup T)^*$
3	Regulär	• entweder: $v = a$ oder $v = aA$ (rechtslinear) $(a \in T, A \in N)$ • oder: $v = a$ oder $v = Aa$ (linkslinear) • Wenn: $v = aA$ UND $v = Aa \rightarrow \text{NICHT regulär} \rightarrow \text{linear}$

**Grammatiken**  $G := (N, T, R, S)$  Verkürzende Regeln:  $u \rightarrow v: |u| \geq |v| \quad A \rightarrow C$   
 $N = \{S, A, B\}, T = \{0, 1\}$  reguläre, kontextfrei und  $cAc \rightarrow cAb \rightarrow cAb$   
- sensitiv sind nichtverhorrend

**Regeln:**

- $S \rightarrow 0$  regulär
- $S \rightarrow 1$  regulär
- $S \rightarrow OSO$  kontextfrei
- $S \rightarrow 1\$1$  kontextfrei Endl. Automate nur für Reg. Sprachen; Automat hat nur Zustände, kein Speicher.

**Klassifizierung:** Ziel ist es, Sprachen zu klassifizieren.

- Grammatik hat die Klasse ihrer schlechtesten Regel
- Sprache hat die Klasse ihrer besten Grammatik

**2. Endliche Automaten**  $A = (\mathcal{Q}, \Sigma, \delta, q_0, F)$  Darstellung mittels Tabelle oder Graphen

**Graphische Darstellung**

Die Übergangsfunktion stellen wir durch Pfeile zwischen den Knoten dar:

$\delta: Q \times \Sigma \rightarrow Q$  Übergangsfunktion

$q_0$ : Startzustand

$F$ : Menge der akzeptierenden Zustände

**Deterministische Automaten** Vorgehen immer eindeutig → Pfeile immer für alle Variablen da.

**Satz:** Die von einem deterministischen, endlichen Automaten  $A$  akzeptierte Sprache  $L(A)$  ist regulär. Jedoch werden nicht alle regulären Sprachen durch diesen Automaten erkannt.

**Nichtdeterministische Automaten** Vorgehen nicht eindeutig; Pfeile können fehlen oder mehrfach vorkommen

**Satz von Robin-Scott:** Zu jeder nichtdeterministischen, endlichen Automaten kann eine deterministische (NFA  $\rightarrow$  DFA) Version konstruiert werden, der die gleiche Sprache akzeptiert.

**Konstruktion NFA  $\rightarrow$  DFA**

**Konstruktion**

Gegeben:

Akzeptiert, weil q0 akzeptiert.

**Produktion:** Knoten mit dem gleichen „Zusatzpfeil“ werden zusammenfasst. Wenn einer der Ursprungsknoten akzeptiert, akzeptiert auch der neue Knoten.

**Satz:** Zu jeder regulären Sprache  $L$  gibt es einen nichtdeterministischen, endlichen Automaten  $A$  mit  $L(A) = L$ .

**Nichtdeterministische Automaten mit  $\epsilon$ -Übergängen**

**Satz:** Zu jedem NFA mit  $\epsilon$ -Übergängen (NFA/ $\epsilon$ ) gibt es einen NFA, der die gleiche Sprache akzeptiert.

**Konstruktion NFA/ $\epsilon$   $\rightarrow$  NFA**

①  $\epsilon$ -Hölle für alle Zustände erstellen \rightarrow Menge aller Zustände, die vom Zustand aus durch  $\epsilon$ -Übergänge erreichbar sind + den „eigenen“ Zustand selbst.  
 $[q_i]_\epsilon^* := \{q \in Q | q_i \xrightarrow{\epsilon} q\} = \{q_3, q_1\}$

**2. Automatentheorie**

**2.1. Automaten und Sprachen**

**Übergangsautomaten**

**Übergänge erstellen:** Für den Übergangsübersichtstabellen prüfen, ob ein Übergang besteht. Am Ende alle Menschen vereinen \rightarrow Zielzustand"

② Ist in der  $\epsilon$ -Hölle des Zustands ein alter akzeptierender Zustand, so ist der neue Zustand auch akzeptierend.

**Eigenschaften regulärer Sprachen**

**Abgeschlossenheit regulärer Sprachen:** Für zwei reguläre Sprachen  $L_1$  und  $L_2$  gilt. Die Sprachen  $\bar{L}_1, L_1 \cup L_2, L_1 \cap L_2, L_1 \circ L_2, L_1^*$  sind alle wieder regulär.

**Operationen auf Sprachen**

Potenzen:  $L^0 := \{\epsilon\}, L^1 := L, L^2 := L \circ L, L^{n+1} = L^n \circ L$

Verknüpfung:  $L_1 \circ L_2 := \{w_1 w_2 | w_1 \in L_1, w_2 \in L_2\}$

Akzeptierende Zustände von  $L_1$  werden invertiert und mit  $\epsilon$ -Übergängen an Startzustand von  $L_2$  geknüpft.

**Kleene-Stern:**  $L^* := L^0 \cup L^1 \cup L^2 \cup L^3$

Endzustand wird invertiert und mit Start verknüpft. Start wird akzeptierend.

4)  $L \in \text{Reg}_{\Sigma}$   
 $L = \{01^*\}$

**2.2. Minimierung von DFA**

**DFA:**  $q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_4 \xrightarrow{0} q_3 \xrightarrow{1} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_4 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_4$

**Invertieren der Zustandseigenschaften eines DFA**

1)  $L \in \text{Reg}_{\Sigma}$

$L_1 = \{01^*\}$   
 $\bar{C} = (Q, \Sigma, \delta, q_0, \bar{F})$

2)  $L_1 \cup L_2 \in \text{Reg}_{\Sigma}$

**Vereinigung  $L_1 \cup L_2$  DFA**

Neuer Startzustand mit  $\epsilon$ -Übergängen zu allen Starts

3)  $L_1 \cap L_2 \in \text{Reg}_{\Sigma}$

**Produkt:  $L_1 \cap L_2$**

$A \cap B = (\overline{A \cap B})$

$L(A \cap B) = \{1^0 0^1 1^0 1^0 \dots\}$

**Produkt der Startzustände ist neuer Startpunkt.**

**Produkt der akzeptierenden Zustände ergeben die neuen.**

**Pumping-Lemma und Myhill-Nerode**

**Pumping-Lemma:** Beweismethode, um die Regularität von Sprachen zu widerlegen

**Satz:** Jede reguläre Sprache  $L$  über  $\Sigma$  hat folgende Eigenschaft:  
 $\exists n \in \mathbb{N}: \forall w \in L, |w| \geq n \rightarrow \exists x, y, z \in \Sigma^* \text{ mit:}$

- $w = xyz$
- $|y| > 0$
- $|xy| \leq n$
- $\forall i \in \mathbb{N}: xy^i z \in L$ 

**Myhill-Nerode**

**Satz:** Es sei  $L$  eine beliebige Sprache über einem Alphabet  $\Sigma$ , dann gilt:  
 $L$  ist regulär  $\Leftrightarrow \text{ind}(R_L) < \infty$

**Konstruktion Nerode-Automat von  $L = \{0, 1, 00, 11\}$**

① Äquivalenzklassen der Nerode-Relation aufstellen.

② Repräsentanten der Äquivalenzklassen sind neue Zustände

③ Neue Antwortfunktion: Repräsentant nehmen und Buchstabe anhängen.  
 $[0] = f(0, 0, 1, 00, 11)$   
 $[1] = f(1, 0, 1, 00, 11)$   
 $[00] = f(00, 0, 1, 00, 11)$   
 $[11] = f(11, 0, 1, 00, 11)$

**Minimale Automaten**

**Satz:** In jeder Äquivalenzklasse der Relation  $\equiv$  auf DFA gibt es bis auf Isomorphie genau einen minimalen Automaten, d.h. einen mit einem minimalen Anzahl von Zuständen.

**Minimalisierungs-Algorithmus:** Zeigt auf, welche Zustände nicht in der gleichen Klasse sind.

① Akzeptierender Zustand unterscheidet sich zu nicht akzeptierendem \*

② Zwei Zustände wählen, die auf einen bereits bekannten Zustand zurückzuführen sind. Daraufhin wählen um auf zwei Zustände zu kommen, die nicht in der gleichen Klasse sind.  
 $q_1 \xrightarrow{0} q_4, q_0 \xrightarrow{0} q_1 \Rightarrow *$

**Schriftweise vorsehen**

Zustände trennen

  - Ein „\*“ aus akzeptierend und nicht akzeptierend = \*
  - Wenn ich aus zwei Zuständen mit dem gleichen Wort an die gleiche Ort komme → kein \*
  - Wenn ich mit zwei Zuständen z.B. mit null weitergehe und mit einem Zustand mit einem anderen Wort komme und mit dem anderen nicht, dann ist er nicht im selben Block = \*
  - ODER den Automaten minimieren und die neu kreierten Zustände leer lassen und den Rest mit Stern versehen.

**Reguläre Ausdrücke  $A: \text{Reg}_{\Sigma} \rightarrow \text{NFA}_{\Sigma}$**

**Satz  $E(\text{Reg}_{\Sigma}) = \text{Reg}_{\Sigma}$ :** Abbildung  $E$  surjektiv, d.h. jede reg. Sprache kann durch einen reg. Ausdruck beschrieben werden.

Zudem: jede reg. Sprache kann aus den endlichen Sprachen durch die Vereinigung, Produkt und Stern erzeugt werden.

**Beispiel:**  $A(a) \cup A(b)$

und für reguläre Ausdrücke  $\alpha$  und  $\beta$  gelten:

  - $A(\alpha \# \beta) := A(\alpha) \cup A(\beta)$ ,
  - $A(\alpha \beta) := A(\alpha) \circ A(\beta)$ ,
  - $A(\alpha^*) := A(\alpha)^*$ .

**Kontextfreie Sprachen**  $G := (N, T, R, S)$  mit  $N = \{S\} \cup N_1 \cup N_2$  und alte Regeln  $v$  den neuen unten.

**Satz:** Jede kontextfreie Grammatik kann in die Chomsky-Normalform gebracht werden.

① Einführung eines neuen Startsymbols  $S_0$  und der Regel  $S_0 \rightarrow S$  (bzw.  $S_0$  auf alle alten Startsymbole, welche von z.B.  $S_A, S_B$  etc. genannt und als normale Variablen behandelt werden.)

**Chomsky-Normalform**

**Definition:** Grammatik mit nur Regeln der Form  $A \rightarrow BC, A \rightarrow a$  oder  $S \rightarrow \epsilon$  (Wenn  $\epsilon \in L$ ) mit  $A \in N$  und  $B, C \in N - \{S\}$  und  $a \in T$

② Elimination von Regeln der Form  $A \rightarrow \epsilon$ , falls  $A \neq S_0$  ( $S_0 \rightarrow \epsilon$  ist also o.k.)

  - Regel der Form  $A \rightarrow \epsilon$  streichen
  - Auf der rechten Seite aller Regeln (z.B.  $S_0 \rightarrow A$ ) das  $A$  in einer neuen Regel durch  $\epsilon$  ersetzen: z.B.  $S \rightarrow ASA$  wird zu  $S \rightarrow SA, S \rightarrow AS$  und  $(S \rightarrow S)$
  - Falls bereits bearbeitete  $\epsilon$ -Regeln wieder auftauchen (z.B.  $S \rightarrow \epsilon$ ): Streichen!



## Loop - Programme

### Sprache LOOP<sub>n</sub>

$$\text{LOOP} := \bigcup_{i=1}^{\infty} \text{LOOP}_i$$

$$L(G) = \{ w \in T^* \mid \text{<programm>} \Rightarrow^* w \} \quad (n = \text{Anzahl Variablenzeichen})$$

### LOOP-Semantik

Ein Wort/Programm aus LOOP<sub>n</sub> operiert auf einem Register von n-Speichern.

#### Grundregeln:

- $x_i := 0$  Speicher  $x_i$  wird auf Null gesetzt.
- $x_i := x_j$  Speicher  $x_i$  wird auf den Wert des Speichers  $x_j$  gesetzt.
- $x_i := S(x_i)$  Speicher  $x_i$  wird auf den Wert  $x_i + 1$  gesetzt.
- $x_i := P(x_i)$  Speicher  $x_i$  wird auf den Wert  $x_i - 1$  gesetzt.

#### Register:

$$x_1 \ x_2 \ x_3 \ \dots \ x_n$$

Zu Beginn sind die Speicher belegt oder automatisch 0:

$$1 \ 3 \ 0 \ \dots \ 0$$

#### Ausführungsregel:

loop  $x$ ; do <programm> ad:

<programm> wird so oft ausgeführt, wie der Inhalt von Speicher  $x$  beim ersten Mal lesen angibt.

Output: Werte der Speicher nach Terminierung des Programms wird mit folgender Funktion bezeichnet:

$$\Phi_i(x_1, \dots, x_n), i=1, \dots, n \leftarrow \text{Speicher } x_i \text{ enthält den Wert } \Phi_i(x_1, \dots, x_n)$$

(Im Allgemeinen ist die erste Zelle des Registers nach Ausführung der Output.

### LOOP-berechenbare Funktionen

Definition: Die Funktion  $f: \mathbb{N}^n \rightarrow \mathbb{N}$  ist LOOP-berechenbar, genau dann, wenn gilt:

$$\exists m \geq n, \Pi \in \text{LOOP}_m : f(x_1, \dots, x_n) = \Phi_\Pi(x_1, \dots, x_n, 0, \dots, 0)$$

#### Beispiel:

$\text{add}: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  Wir geben ein Programm  $\Pi_{\text{add}} \in \text{LOOP}_2$  an:

$$(a, b) \mapsto a+b$$

loop  $x_2$  do  $\Pi_{\text{add}}$  operiert auf  $a \ b$

$$x_1 := S(x_1)$$

od

#### Länge einer Funktion ( $\Pi$ ):

Wird definiert durch die Anzahl Programme.

z.B. mult:  $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

$\Pi_{\text{mult}} \in \text{LOOP}_3$  und operiert auf  $x_1 \ x_2 \ x_3 := a \ b \ 0$

$$(a, b) \mapsto ab$$

loop  $x_1$  do  $\Pi_{\text{mult}}$

od

$$(x_1 := x_3)$$

### Rekursive Funktionen

Sind es komplexere Funktionen aus einfachen, zweifelsfrei berechenbaren Funktionen aufzubauen.

#### Grundfunktionen

Die folgenden Funktionen sind sicher berechenbar.

① Die nullstellige Nullfunktion:  $Z: 0 \in \mathbb{N}$

③ Die Vorgängerfunktion:  $P: \mathbb{N} \rightarrow \mathbb{N}$

② Die Nachfolgerfunktion:  $S: \mathbb{N} \rightarrow \mathbb{N}$

$x \mapsto x+1$

④ Die Projektionsfunktionen:  $U_i^n: \mathbb{N}^n \rightarrow \mathbb{N}$

$$(x_1, x_2, \dots, x_n) \mapsto x_i$$

#### Kompositionsschema

$h$ : eine  $n$ -stellige Funktion

Durch Komposition entsteht die  $n+1$ -stellige Funktion

$g_1, \dots, g_m$ ,  $m \times n$ -stellige Funktionen

$$f(x_1, \dots, x_n) = h(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n))$$

#### Primitives Rekursionsschema

$g$ :  $n$ -stellige Funktion

Durch (primitive) Rekursion ergibt sich die  $n+1$ -stellige Funktion  $f$ :

$h$ :  $n+2$ -stellige Funktion

$$f(x_1, \dots, x_n, 0) = g(x_1, \dots, x_n)$$

$$f(x_1, \dots, x_n, y+1) = h(x_1, \dots, x_n, y, f(x_1, \dots, x_n, y))$$

#### Beispiel: Konstruktion einer 2-stelligen Funktion

Aus der 1-stelligen Funktion  $g(x)$ : ist die 2-stellige Funktion  $f(x, y)$  wie folgt def:

$$f(x, 0) = g(x)$$

$$f(x, y+1) = h(x, y, f(x, y))$$

### Evaluation des Schemas: (Mit vorherigem Beispiel)

$$\begin{aligned} \text{Vert von } f(5, 3) \text{ mit } f(x, 0) &= g(x) \\ f(x, y+1) &= h(x, y, f(x, y)) \end{aligned} \quad \left. \begin{array}{l} f(5, 0) = g(5) \\ f(5, 1) = h(5, 0, f(5, 0)) \\ f(5, 2) = h(5, 0, f(5, 1)) \\ f(5, 3) = h(5, 0, f(5, 2)) \end{array} \right\}$$

### Die primitiv-rekursiven Funktionen PR

kleinste Menge von Funktionen  $f: \mathbb{N}^n \rightarrow \mathbb{N}$  mit beliebiger Stelligkeit, welche die Grundfunktionen enthält und bzgl. Komposition und (primitive) Rekursion abgeschlossen ist, heisst man primitiv-rek. funk PR

#### Beispiel:

Zeigen, dass Addition PR ist: Mittels primitiver Rekursion, mit  $g(x)$  und  $h(x, y, z)$ :

$$\text{add}: \mathbb{N}^2 \rightarrow \mathbb{N} \quad \text{add}(x, 0) = x = U_1(x)$$

$$(x, y) \mapsto x+y = \text{add}(x, y) = S(\text{add}(x, y)) = S(U_3(x, y, \text{add}(x, y)))$$

$$\text{ergebt: } g(x) = U_1(x) \text{ und } h(x, y, z) = S(U_3(x, y, z))$$

### LOOP-Programme und RP

Eine Funktion  $f: \mathbb{N}^n \rightarrow \mathbb{N}$ , die mit Hilfe eines LOOP-Programms berechnet werden kann, ist loop-berechenbar.

• Eine Funktion ist genau dann loop-berechenbar, wenn sie primitiv-rekursiv ist.

Berechenbare Funktionen, die nicht in RR sind

② PR umfasst nicht alle berechenbaren Funktionen (partielle Funktionen sind nicht in PR)

#### Beweis

$f: \mathbb{N} \rightarrow \mathbb{N}$ , wobei  $f(n)$  die grösste Zahl ist. → Funktion ist berechenbar und streng monoton wachsend.

Annehmen, dass  $f$  loop-berechenbar. Daraus folgt:  $g(n) = f(2n) \quad \exists \Pi_g$ , dass  $g(n)$  auf Input  $x_1 = n$  berechnet

Sei  $I := \{\Pi_g\}$  und  $I_m$  das folgende Programm:

$$\begin{cases} x_1 := S(x_1) \\ x_1 := S(x_1) \\ \dots \\ x_1 := S(x_1) \end{cases} \quad | \quad n$$

$\Pi_m$  berechnet  $g(n)$  auf Input  $x_1 = 0$

Wir erhalten  $I(I_m) = n+1$

Für  $n > 1$  gilt aber:  $f(2n) = g(n) = \Phi(I_m) \leq f(n+1)$

Widerspruch zur Monotonie von  $f(x)$

### Der $\nu$ -Operator

Existiert, um auch nicht totale Funktionen berechnen zu können.  $h(y) := \nu x [f(x, y) = 0]: \mathbb{N} \rightarrow \mathbb{N}$

$\rightarrow x$  iterieren (beginnend mit  $x=0$ ), bis  $f(x_0, y) = 0$  oder nicht definiert ( $= \perp$ )

Beispiel: Sei  $f(x, y) = x+y$ . Wir binden mit dem  $\nu$ -Operator die Variable  $x$ :  $h(y) := [\nu x [f(x, y) = 0]]: \mathbb{N} \rightarrow \mathbb{N}$

$\Rightarrow h(y)$  wird höchstens den Wert  $y=0$  definiert sein (Bedingung nicht mehr erfüllt)

$$\nu x [f(x, y) = 0] = \begin{cases} 0, \text{ falls } y=0 \text{ ist} \\ \perp, \text{ sonst} \end{cases}$$

WHILE-Programme:  $\text{WHILE}[f(x, y) = 0]$  wird durch das folgende Pseudoprogramm berechnet.

Input:  $y$

$$x := 0$$

while  $f(x, y) \neq 0$  do

$$x := x+1$$

od

• Eine Funktion ist genau dann WHILE-berechenbar, wenn sie rekursiv ist.

$$\text{R} = \text{WHILE-berechenbare F.} \quad \text{T} = \text{T-berechenbare F.}$$

$$\text{PR} = \text{LOOP-berechenbare F.}$$

Einige Eigenschaften / Sätze: reg, kontextfrei u. kontextsensitiv  $\in L_T$ ; allgemein  $\in \text{kre}$

◦ Das Komplement einer rekursiven Sprache ist rekursiv. ( $L$  ist rekursiv und hat TM  $\Rightarrow$  A/F ebenfalls)

◦ Eine Sprache  $L$ , die zusammen mit ihrem Komplement  $\bar{L}$  rekursiv aufzählbar ist, ist rekursiv. ( $L_1$  und  $L_2$  je ein Schnitt)

◦ Eine Sprache  $L$ , die eine nicht verkürzende Grammatik besitzt, ist rekursiv. (Suche im Ableitungsbäum der Gram.)

Aussagen aus Prüfungen: ( $L \subseteq \Sigma^*$  und  $\bar{L}$ )

① Wenn  $L$  kontextfrei ist, dann ist  $\bar{L}$  immer rekursiv (Kontextfreie Sprachen sind rekursiv)

② Wenn  $L$  und  $\bar{L}$  semi-entscheidbar sind, dann ist  $L$  rekursiv (Siehe oben)

③ Auch wenn  $L$  und  $\bar{L}$  nicht rekursiv sind, ist die Vereinigung der beiden Sprachen rekursiv

④  $L$  kann rekursiv aufzählbar sein und  $\bar{L}$  nicht

⑤ Es gibt Sprachen, für die es keine Grammatik gilt, ist rekursiv

⑥ Der Schnitt zweier rekursiver Sprachen ist immer rekursiv

⑦ Jede kontextsensitive Sprache ist rekursiv, ihr Komplement jedoch nicht unbedingt

⑧ Wenn  $L$  rekursiv ist, dann ist  $\bar{L}$  rekursiv aufzählbar, aber nicht natürlicherweise rekursiv

⑨ Jede kontextfreie Sprache ist rekursiv, aber nicht jede kontextsensitive Sprache ist rekursiv.

⑩  $L_{univ}$  ist nicht rekursiv, aber es ist eine offene Frage, ob  $L_{univ}$  rekursiv aufzählbar ist.

### Universelle Turing-Maschine

Gedächtnis: Kodierung von TMs über einem Alphabet

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

$$Q = \{q_0, q_1, \dots, q_n\}$$

$$\Sigma := \{0, 1, \dots\}$$

$$\Gamma := \Sigma \cup \{ \}$$

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, H, R\}$$

$$wobei \quad x_1 = 0, \quad x_2 = 1, \quad x_3 = b \quad \text{und}$$

$$D_1 = L, \quad D_2 = H, \quad D_3 = R$$

bedeuteten sollen.

$$\text{code}(x_1) := 0^{1+1}0^10^{1+1}0^10^{10^m}.$$

Gödelnummer:  $\langle M \rangle := \text{code}(1) \ M \ \text{code}(2) \ M \ \dots \ \text{code}(s) \ M$

Nicht rekursive Sprachen

Sprache Lding:  $L_{ding} := \{w \in \Sigma^* \mid M_i(w) = 0 \text{, wenn } w = w_i\}$

◦  $L_{ding}$  ist damit auch nicht rekursiv

◦  $L_{ding}$  und  $\bar{L}_{ding}$  können beide nicht rekursiv aufzählbar sein

Sprache Luniv:  $L_{univ} := \{ \langle M \rangle w \mid M \text{ akzeptiert } w\} \leftarrow$  rekursiv aufzählbar, jedoch nicht rekursiv

$L_{univ}$  ist die Menge aller Wörter, die aus einer Gödelnummer  $\langle M \rangle$  und einem Wort  $w$  bestehen, so dass die Turing-Maschine  $M$  das Wort  $w$  akzeptiert.

Sprache Lhalt:  $L_{halt} := \{ \langle M \rangle w \mid M \text{ hält auf } w\} \leftarrow$  rekursiv aufzählbar, jedoch nicht rekursiv

Reduktion many-to-one total T-berechenbar

Seien  $L_1$  und  $L_2$  zwei Sprachen über  $\Sigma$

Def:  $L_1$  ist auf  $L_2$  reduzierbar:  $\Leftrightarrow \exists f: \Sigma^* \rightarrow \Sigma^*$ ,

total und Turing-berechenbar mit

$\forall w \in \Sigma^*: w \in L_1 \Leftrightarrow f(w) \in L_2$

Notation:  $L_1 \leq_m L_2$  (Relation ist reflex. und trans.)

Bedeutungserklärung für rek. Sprachen: Wenn wir eine Sprache  $L_2$  haben, die sich auf eine nicht-rekursive Sprache (z.B. Lding) reduzieren lässt, ist bewiesen, dass auch  $L_2$  nicht rekursiv ist. Wäre  $L_2$  rekursiv, hätten wir damit auch bewiesen, dass die andere Sprache rekursiv ist.

Lemma: Es seien  $L_1$  und  $L_2$  zwei Sprachen über  $\Sigma \rightarrow$  Voraussetzung:  $L_1 \leq L_2$

Behauptung: 1. Wenn  $L_2$  rekursiv ist, dann ist auch  $L_1$  rekursiv.

2. Wenn  $L_1$  nicht rekursiv ist, dann ist auch  $L_2$  nicht rekursiv.

Beispiel:

$L_1 = \{0^k 1^k \mid k \in \mathbb{N}\}$  kontextfrei

$L_2 = \{0, 1, 000\}$  regulär (dk endlich)

$f: \Sigma^* \rightarrow \Sigma^* \quad \forall x \in \Sigma^*: x \in L_1 \Leftrightarrow f(x) \in L_2$  (Definition)

Behauptung:  $L_1 \leq_m L_2$  f:  $\Sigma^* \rightarrow \Sigma^*$

$x \mapsto f(x) := \begin{cases} 000, \text{ falls } x \in L_1 \\ 1, \text{ falls } x \notin L_1 \end{cases}$

Wir benutzen, dass  $L_1$  kontextfrei  $\Rightarrow L_1$  rekursiv

## Kommt partiell an Prf $\rightarrow$ Satz von Rice!

Satz von Rice

Voraussetzung:

1.  $R := \{f: \Sigma^* \rightarrow \Sigma^* \mid \text{partiell, } T\text{-berechenbar}\}$
2.  $S \subseteq R$  mit  $S = \emptyset$  und  $S \neq R$
3.  $L(S) := \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } S\}$

Behauptung:  $L(S)$  ist nicht rekursiv

Bedeutung: Der Satz besagt, dass es keine Turing-Maschine gibt, die auf Input  $\langle M \rangle$  entscheiden kann, ob die Turing-Maschine  $M$  eine Funktion aus  $S$  berechnet.

Einige unentscheidbare Probleme

Gegeben seien zwei kontextfreie Grammatiken  $G_1$  und  $G_2$ . Die folgenden Probleme sind alle unentscheidbar.

Die folgenden Probleme sind unentscheidbar:

- Das postn. Korrespondenzproblem
- 1.  $L(G_1) \cap L(G_2) = \emptyset$
- 2.  $|L(G_1)| / |L(G_2)| = \infty$
- 3. Ist  $L(G_1) \cap L(G_2)$  kontextfrei?
- 4.  $L(G_1) \subset L(G_2)$
- 5.  $L(G_1) \supseteq L(G_2)$
- 6. Ist  $L(G_1)$  kontextfrei?
- 7. Ist  $L(G_1)$  regulär?
- Das 10. hilbertsche Problem
- Viele Fragen über endlich präsentierbare Gruppen sind unentscheidbar: Satz von Adian-Rabin.
- Es führt dazu, das auch wichtige geometrisch-topologische Fragen unentscheidbar sind.

Komplexitätstheorie | Siehe Kostenmasse bei Turing-Maschine)

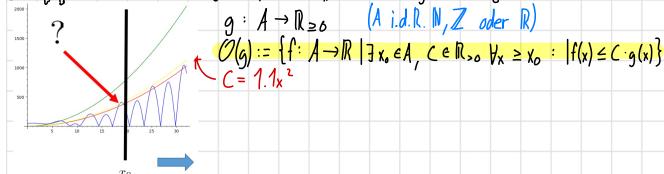
Wir wollen Aussagen über das asymptotische Wachstum des Aufwands in Abhängigkeit von der Größe einer Instanz des Problems im schlimmsten Fall.

Beispiel: Entscheidungsproblem der Sprache  $L := \{0^k 1^k \mid k \in \mathbb{N}\} \leftarrow$  kontextfrei

- Mit 1-Band-Turing-Maschine in quadratischer Zeit lösbar:  $T_M(n) = O(n^2)$
- Mit 2-Band-Turing-Maschine in linearer Zeit lösbar:  $T_M(n) = O(n)$

Big-O-Notation

Eine gegebene Funktion oder Folge  $f(x): A \rightarrow \mathbb{R}$  mit einer negativen Vergleichsfunktion



Komplexitätsklassen ist durch vier Parameter spezifiziert: z.B. 1-Band-TM, k-band-TM, RAM

① Berechnungsmodell: Die Art der Maschine, mit welcher die Berechnung ausgeführt wird.

② Berechnungsmodus: Wie die Maschine zu ihrem Schluss kommt, z.B. deterministisch oder nicht.

③ Ressource: z.B. Zeit, Platz etc.

④ Schranke für Ressource: Wie viele Schritte, wie viel Platz darf die Maschine in Abhängigkeit von der Inputlänge verwenden? Schranke ist Funktion  $f: \mathbb{N} \rightarrow \mathbb{N}, n^k, 2^n, n \log n$

These von Cobham und Edmonds polynome Zeit = effizient

Wenn zwei universelle deterministische Berechnungsmodelle mit logarithmischen Kostenmasse gegeben sind, dann existiert ein Polynom  $p(n)$ , so dass  $t$ -Schritte im ersten Modell durch  $p(t)$  Schritte im zweiten Modell simuliert werden kann.

Logarithmisches Kostenmass: Kostenmass, dass die Größe des Inputs, der aus natürlichen (oder davon abgeleitet)

Zahlen besteht, mit Hilfe einer  $m$ -adischen Darstellung. Natürliche Zahl

Wird also z.B. dezimal oder binär geschrieben, jedoch nicht unar.

Wachstum von  $m$ -adischer Darstellung zur unären ist exponentiell.

Klasse P

Zugehörigkeit einer Sprache  $L$  zur Klasse P ist unabhängig vom Berechnungsmodell.

$P := \bigcup_{k>0} \text{DTIME}(n^k)$  P ist die Klasse der in polynomieller Laufzeit, deterministisch entscheidbaren Sprachen.

d.h.:  $L \in P \Leftrightarrow \exists M \in \text{TMs}, k \in \mathbb{N}^*: 1) L = L(M)$

2)  $T_M(n) = O(n^k)$

$NP = P$  nicht bewiesen!

$L = \{0^n 1^{n+1} 0^n \mid n \in \mathbb{N}\} \in \text{DTIME}(n)$ : 2-Band, deterministisch in linearer Zeit.

Beispiele zur Klasse P:

- EP:  $\exists$  Eulerpfad, oder nicht
- EP:  $n$  eine Primzahl
- EP:  $n \in \mathbb{N}$ : Ist  $n$  zusammengesetzt?

Klasse NP

Sind Sprachen, deren Entscheidungsproblem im Allgemeinen nur in polynomieller Zeit lösbar sind. Wenn ein Hinweis (das Zertifikat) vorhanden ist, welches in seiner Länge aber polynomial beschränkt sein muss.

Sprache  $L \subseteq \Sigma^*$  liegt in Klasse NP, wenn gilt:

Es existiert eine deterministische Turing-Maschine  $M$  mit polynomieller Laufzeit und einem Polynom  $p$ , so dass für alle  $x \in \Sigma^*$  gilt:

$$x \in L \Leftrightarrow \exists u \in \Sigma^* \text{ mit } |u| \leq p(|x|) \text{ und } M(x, u) = 1 \quad (1)$$

①  $P \subseteq NP$ . Die Bedingung (1) ist mit  $u = \epsilon$ , dem leeren Wort, erfüllt. ↗ Zertifikat

Beispiele zur Klasse NP:

- Besitzt Graph  $G$  einen Hamiltonpfad?
- Subset-Sum
- 3-GC: Gibt es eine Knotenfärbung mit 3 Farben?
- Factoring

Aufgabensammlung II

Primitiv-rekursive Funktionen

Zeigen Sie, dass die Fakultätsfunktion

$$\begin{aligned} fac: \mathbb{N} &\rightarrow \mathbb{N} \\ x &\mapsto x! \end{aligned}$$

eine primitiv-rekursive Funktion ist.

Lösung:

$$\begin{aligned} fac(0) &= 1 &= S(Z) \\ fac(y+1) &= (y+1) * fac(y) &= mult(S(y), fac(y)) \end{aligned}$$

Zeigen Sie, dass das Prädikat auf der Menge  $\mathbb{N} \times \mathbb{N}$

$$G(x, y) = \begin{cases} 1, & \text{falls } x = y, \\ 0, & \text{sonst.} \end{cases}$$

primitiv rekursiv ist.

Lösung: Wir führen die 1-stellige Hilfsfunktion (Signum)

$$sg(x) = \begin{cases} 1, & \text{falls } x = 0, \\ 0, & \text{sonst.} \end{cases}$$

ein, die auch sonst in diesem Zusammenhang gute Dienste leistet. Wir weisen  $sg$  wie folgt als primitiv rekursiv nach:

$$\begin{aligned} sg(0) &= S(Z) \\ sg(y+1) &= h(y, sg(y)) = Z^2(y, sg(y)) \end{aligned}$$

wobei  $Z^2$  die 2-stellige Nullfunktion ist.

Nun können wir mit Hilfe des Kompositionsschemas die primitive Rekursivität zeigen:

$$G(x, y) = sg(|x - y|).$$

Wenn Sie  $|x - y|$  nicht direkt verwenden wollen, können Sie

$$|x - y| = (x - y) + (y - x)$$

verwenden oder auch

$$sg(|x - y|) = sg(x - y) + sg(y - x) - 1.$$

Zeigen Sie, dass die Multiplikation

$$\begin{aligned} mult: \mathbb{N}^2 &\rightarrow \mathbb{N} \\ (x, y) &\mapsto xy \end{aligned}$$

eine primitiv-rekursive Funktion ist.

Lösung:

$$\begin{aligned} mult(x, 0) &= 0 &= Z^1(x) \\ mult(x, y+1) &= mult(x, y) + x &= add(U_1^3(x, y, mult(x, y)), U_3^3(x, y, mult(x, y))) \end{aligned}$$

mit

$$\begin{aligned} Z^1(0) &= 0 = Z \\ Z^1(y+1) &= 0 = U_2^2(y, Z) \end{aligned}$$

und

$$h(x, y, z) = add(U_3^3(x, y, z), U_1^3(x, y, z)).$$

LOOP und WHILE

Zeigen Sie, dass die Subtraktion

$$\begin{aligned} sub: \mathbb{N}^2 &\rightarrow \mathbb{N} \\ (x, y) &\mapsto x - y \end{aligned}$$

loop-berechenbar ist.

Lösung:

Input:  $(x_1, x_2) = (a, b)$

Output:  $x_1$

loop x2 do  
   $x_1 := P(x_1)$   
od

Zeigen Sie, dass die Funktion if-then-else loop-berechenbar ist:  
if-then-else:  $\begin{array}{c} \mathbb{N}^3 \rightarrow \mathbb{N} \\ (x, y, z) \mapsto \begin{cases} z, & \text{falls } x = 0, \\ y, & \text{sonst.} \end{cases} \end{array}$

if-then-else:  $\begin{array}{c} \mathbb{N}^3 \rightarrow \mathbb{N} \\ (x, y, z) \mapsto \begin{cases} z, & \text{falls } x = 0, \\ y, & \text{sonst.} \end{cases} \end{array}$

Lösung:

Input:  $(x_1, x_2, x_3, x_4) = (a, b, 0, 0)$

Output:  $x_1$

loop x2 do  
  loop x4 do  
     $x_1 := x_3$   
  od  
od

div:  $\begin{array}{c} \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\ (a, b) \mapsto a \text{ div } (b+1) \end{array}$

Programmieren Sie in der Sprache LOOP die folgenden Funktionen:

a)  $sq: \begin{array}{c} \mathbb{N} \rightarrow \mathbb{N} \\ a \mapsto a^2 \end{array}$

wobei mit div die ganzzahlige Division gemeint ist.

Lösung

Input:  $(x_1, x_2) = (a, 0)$

Output:  $x_1$

loop x1 do  
  loop x2 do  $x_2 := S(x_2)$  od  
od  
 $x_1 := x_2$

Ist die folgende Funktion Loop-berechenbar?

mit

$$f(n) = \begin{cases} 0, & \text{falls } n \text{ gerade,} \\ 1, & \text{sonst.} \end{cases}$$

Versuchen Sie ein LOOP-Programm der Länge höchstens 5 zu schreiben, das auf Input  $(0, \dots, 0)$  eine möglichst grosse Zahl als Output ( $x_1$ ) produziert.

Lösung

Input:  $x_1$

Output:  $x_1$

Lösung

Input:  $x_1$

Output:  $x_1$

Lösung

Input:  $x_1$

Output:  $x_1$

Die Länge des Programms ist 5, der Output bei Terminierung  $x_1 = 8$ .

Zeigen Sie, dass die Potenzfunktion

$$pot: \begin{array}{c} \mathbb{N}^2 \rightarrow \mathbb{N} \\ (x, y) \mapsto x^y \end{array}$$

eine primitiv-rekursive Funktion ist.

Lösung:

Input:  $x_1$

Output:  $x_1$

Lösung

Input:  $x_1$