

Theoretische Informatik I / II

Prof. Kupka

WS 1995/96

SS 1996

Wilko Hein

1 Formale Sprachen und Grammatiken

..... Definitionen

Alphabet (nichtleere endliche Menge), Symbole, Wort über Σ , Leeres Wort, Länge. L aus Σ^*
Formale Sprache (Mengenbeschreibungen, Grammatiken, reguläre Ausdrücke, Graphen).

1.1 Mathematische Mengenbeschreibungen

Endliche Auflistung
Unendliche Auflistung
Charakterisierung durch logische Aussagen
Rekursive Mengengleichungen $L = \{\varepsilon\} \cup \{a\}L\{b\}$

1.2 Grammatiken

Produktion: Paar $\langle p, q \rangle \in \Sigma^+ \times \Sigma^*$
Semi-Thue-System: (Σ, P) , P endlich und nicht leer
Terminalzeichen: T
Nichtterminalzeichen: N
Chomsky-Grammatik: $G = (T, N, P, S)$, T und N disjunkte Alphabete, $(T \cup N, P)$
Semi-Thue-System, S aus N Startelement

Definition Typen von Grammatiken

$G = (T, N, P, S)$ hat Typ:
0: $P \subseteq V^+ \times V^*$ (ε nie vorne)
1: $P \subseteq V^* N V^* \times V^*$ [Monoton] (links mindestens ein Nicht-Term) $\{a^n b^n\}$
Entweder $|p| \leq |q|$ (Sprachen, die ε nicht enthalten) oder $p = S$ und $q = \varepsilon$, ε nirgends rechts
2: $P \subseteq N \times V^*$ ($A \rightarrow w$) [Kontextfrei] $\{a^n b^n\}$
3: $P \subseteq N \times \{\{\varepsilon\} \cup T \cup TN\}$ ($A \rightarrow \varepsilon, A \rightarrow t, A \rightarrow tB$ [Rechtslinear], [Regulär], vv.) $\{a^n b^m\}$

Definition Ableitungen

Direkte Ableitung ($\exists a, b, p, q \in (T \cup N)^*, x = apb, y = aqb, (p \rightarrow q) \in P$), Relation
Ableitung (Relation ist reflexive transitive Hülle)
Erzeugte Sprache $L(G)$
Satzform (Aus Startsymbol ableitbares Wort), terminale Satzform

Satz Typ 1,2,3 => Typ 0

Definition Äquivalenz

Grammatiken heißen äquivalent, wenn gleiche Sprache erzeugen

Übersicht Grammatik \leftrightarrow Sprache

0	---	Aufzählbar
1	Monoton	Kontextsensitiv
2	Kontextfrei	Kontextfrei
3	Regulär	Regulär

Definition Kontextsensitiv

Alle $p \rightarrow q$ der Form $p = \alpha A \gamma, q = \alpha \beta \gamma, \beta \neq \varepsilon$ oder ε nur auf r.S. in $S \rightarrow \varepsilon$

Satz Kontextsensitive Grammatik ist monoton

Satz Monotone G. => Kontextsensitive G.

Konstruktiv. Alle Terms x durch Nichtterms X mit $X \rightarrow x$. Alle $A_1 A_2 \dots A_n \rightarrow B_1 B_2 \dots B_m, n \leq m$
(Monotonie): Aufspulen: Von links alle einzeln durch neue Nichtterms. Abspulen: Von rechts alle neuen Nichtterms durch Zielsymbole.

GÜ Definition ... ϵ -Bedingung bei Typ-2

Höchstens $S \rightarrow \epsilon$ in Grammatik, sonst ϵ nirgends rechts.

Definition Nutzlose Symbole und Regeln bei Typ-2

Nutzloses Symbol Z: Gibt keine Ableitung $S \Rightarrow \alpha Z \beta \Rightarrow w$ mit w als terminaler Satzform
Nutzlose Regeln: Regeln mit nutzlosen Symbolen.

GÜ Satz Beautifier für Typ-2

$\epsilon \notin L \Rightarrow$ Äquiv. G. ohne nutzlose Symbole, Regeln, ϵ -Regeln, Kettenprod.
 ϵ -Regeln eliminieren: In V_ϵ sukzessive alle sammeln, die auf ϵ . Auf jeder rechten Seite alle möglichen Kombinationen von NichtTerms aus V_ϵ weglassen, aber nie ϵ alleine rechts.
Nutzlose Symbole und -Regeln eliminieren: V_T (Alle NichtTerms, die auf Terminales) per Induktion: Als neue Nichtterms. Dann per Induktion alle unerreichbaren Zeichen entfernen.

GÜ Definition ... Rekursivität bei Typ-2

Rekursiv: $A \Rightarrow \alpha A \beta$. Linksrekursiv: $\alpha = \epsilon$. Rechtsrekursiv: $\beta = \epsilon$

GÜ Satz Typ-2 \Rightarrow Typ-2 ohne linksrekursive Variablen

$A_i \rightarrow A_i \alpha_1 \mid A_i \alpha_2 \mid \dots \mid \beta_1 \mid \beta_2 \mid \dots$, β beginnen nicht mit A_i .
 $\Rightarrow A_i \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_1 A_i' \mid \beta_2 A_i' \mid \dots$, $A_i' \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_1 A_i' \mid \alpha_2 A_i' \mid \dots$
Danach für dieses i und alle kleineren j : $A_i \rightarrow A_j \alpha$
 $\Rightarrow A_i \rightarrow \gamma_1 \alpha \mid \gamma_2 \alpha \mid \dots$, wobei die γ aus $A_j \rightarrow \gamma_1 \mid \gamma_2 \mid \dots$

Definition Chomsky-Normalform

$G = (T, N, P, S)$, Alle Regeln der Form $A \rightarrow a$, $A \rightarrow BC$
Ableitungsbäume sind Binärbäume, Satzformen in Knoten!

Satz Typ-2 mit $\epsilon \notin L \Rightarrow$ Typ-2 in Chomsky-Normalform

Vorr.: Keine nutzlosen Symbole, ϵ -Regeln, Kettenproduktionen
Terms x durch Nichtterms X mit $X \rightarrow x$. Splitten durch neue Nichtterms

Definition Typ-2: Greibach-Normalform

Alle Regeln der Form $A \rightarrow a \beta$, $A \in N$, $a \in T$, $\beta \in N^*$
 $\Rightarrow \epsilon \notin L$. Zweck: Erkennen, ob Wort in L . Linear abzuarbeiten.

GÜ Satz Typ-2 ohne linksrekursive Variablen \Rightarrow Typ-2 in Greibach-Normalform

3 Stufen. Ordne NichtTerms so, daß **nicht** $A_i \Rightarrow A_j \alpha$ für $j < i$. Sukzessive NichtTerm suchen, das auf kein noch nicht eingeordnetes am Anfang führt und vorne an Liste anfügen.
In Liste von hinten nach vorne jedes $A_i \rightarrow A_j \alpha$ mit $i < j$
 $\Rightarrow A_i \rightarrow \beta_1 \alpha \mid \dots \mid \beta_k \alpha$ mit β aus $A_j \rightarrow \beta_1 \mid \beta_2 \mid \dots$ (beginnen mit const.)
In $A \rightarrow \alpha x_1 \dots x_k \Rightarrow A \rightarrow \alpha X_1 \dots X_k$, $X_j \rightarrow x_j$ ($S \rightarrow \epsilon$) evtl wieder dazu.

1.3 Reguläre Ausdrücke

Idee: Beschreibung formaler Sprachen durch Ausdrücke mit wenigen Operatoren

Definition Mengenprodukt AB , A^0 , A^{i+1} , A^* (Auch: Kleene'sche Hülle)**Definition Reguläre Mengenoperationen**

Vereinigung, Mengenproduktbildung, Hüllenbildung

Definition Reguläre Mengen

\emptyset , $\{\epsilon\}$, $\{a\}$ reguläre Mengen. Dazu alle durch reguläre Mengenoperationen aus regulären Mengen entstehenden Mengen

Definition Reguläre Ausdrücke (Worte über Σ)

\emptyset regulärer Ausdruck mit $L(\emptyset) = \emptyset$
 ϵ regulärer Ausdruck mit $L(\epsilon) = \epsilon$
 a regulärer Ausdruck mit $L(a) = a$

r_1, r_2 reguläre Ausdrücke. Dann sind auch $(r_1 \cup r_2) [= (r_1 \mid r_2) = (r_1 + r_2)]$,
 $(r_1 r_2), (r_1^*)$ reguläre Ausdrücke mit $L(r_1 r_2) = L(r_1)L(r_2), \dots$

Definitionen Mathematische Begriffe

Halbgruppe: (A, o) mit o assoziativ

Monoid: Halbgruppe mit 1-Element

Homomorphismus: $f(a \circ b) = f(a) \circ f(b)$ mit (A, o) und (B, O) Halbgruppen

Satz Mathematische Operationen auf Mengen

(Σ^*, \cdot) ist Monoid ($1 = \varepsilon$)

Längenfunktion ist Monoidhomomorphismus

$(\mathcal{P}(\Sigma^*), \cdot)$ ist Monoid ($1 = \{\varepsilon\}$)

$(R, \cdot), (R, \cup)$ Monoide, R : reguläre Ausdrücke, $1 = \{\varepsilon\}$ bzw. $1 = \emptyset$.

Dito mit Ausdrücken \mathbf{R} . Halbgruppe! *ea und a haben nur gleiche Bedeutung!*

L : reg. Ausdruck \rightarrow reg. Menge ist Homomorphismus

Satz von Arden (Gleichungsauflösung mit Mengengleichungen)

$X = A X \cup B$ besitzt $X = A^* B$ als Lösung. Einzige Lsg., wenn $\varepsilon \notin A$.

$X = X A \cup B$ besitzt $X = B A^*$ als Lösung. Einzige Lsg., wenn $\varepsilon \notin A$.

Algorithmus Regulärer Ausdruck \Rightarrow bel. Wort aus regulärer Menge

Nichtdeterminismus bei Vereinigung und Hülle!

1.4 Graphen

Praxis der grammatischen Darstellung. Früher: Backus-Naur-Form (Algol 60) \Rightarrow kontextfreie Grammatik. Heute: Syntax-Diagramme (Pascal). Unterscheide "mit" und "ohne" Referenzen.

..... Syntaxdiagramme mit Referenzen \Leftarrow gleichmächtig \Rightarrow Typ-2-G.

..... Syntaxdiagramme ohne Referenzen \Leftarrow gleichmächtig \Rightarrow Typ-3-G.

Algorithmus Syntaxdiagramm \Rightarrow Graphenform rechtslinearer Grammatik

Satz Sprache aus Syntaxdiagramm ohne Referenzen \Rightarrow rechtslineare Grammatik

Satz Rechtslineare Grammatik \Rightarrow Syntaxdiagramm ohne Referenzen

2 Endliche Automaten

2.1 Grundkonzepte für Automaten

Aus Syntaxdiagramm "Zustände". Zustände besitzen Semantik.

2.2 DEA

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

Q : Nichtleere, endliche Menge, sog. Zustandsmenge

Σ : Alphabet

δ : totale Abbildung: $\delta: Q \times \Sigma \rightarrow Q$

$q_0 \in Q$: Anfangszustand

$F \subseteq Q$: Endzustandsmenge

Definition Arbeit eines DEA

Arbeitsschritt: Tripel (p, a, q) , wobei $\delta(p, a) = q$

Arbeitsvorgang: Durch Wort $w = a_1 \dots a_n$ ausgelöste Folge $p_0, p_1, \dots, p_n \in Q^*$ mit

$p_0 = q_0, (p_i, a_{i+1}, p_{i+1})$ ist Arbeitsschritt

Definition Leistung eines DEA

A erkennt Wort w dann, wenn für den durch w ausgelösten Arbeitsvorgang gilt: $p_n \in F$

2.3 NEA

Definition NEA $A=(Q, \Sigma, \delta, S, F)$

Q : Nichtleere, endliche Menge, sog. Zustandsmenge

Σ : Alphabet

δ : totale Abbildung: $\delta: Q \times \Sigma \rightarrow P(Q) := \{ P \mid P \subseteq Q \}$

$S \subseteq Q$: Anfangszustandsmenge

$F \subseteq Q$: Endzustandsmenge

Definition Arbeit eines NEA

Arbeitsschritt: Tripel $(p, a, q) \in Q \times \Sigma \times Q$ mit $q \in \delta(p, a)$ (Wahlmöglichkeit)

Arbeitsvorgang: s.o., $s_0 \in S$

Definition Leistung eines NEA

Wort akzeptiert \Leftrightarrow Ex. von w ausgelöster Arbeitsvorgang mit $p_n \in F$

angelischer Nichtdeterminismus: akzeptierbar \rightarrow akzeptiert

dämonischer Nichtdeterminismus: rückweisbar \rightarrow weist zurück

Definition Erweiterte Transitionsfunktion beim DEA

$\tilde{\delta}: Q \times \Sigma^* \rightarrow Q$. $\tilde{\delta}(q, \epsilon) = q$, $\tilde{\delta}(q, aw) = \tilde{\delta}(\delta(q, a), w)$

\Rightarrow Für $|w|=1$ $\tilde{\delta}(q, w) = \delta(q, w)$. $\tilde{\delta}$ wohldefiniert. $\tilde{\delta}(q, wb) = \delta(\tilde{\delta}(q, w), b)$

DEA A : $L(A) = \{ w \mid \tilde{\delta}(q_0, w) \in F \}$

Definition Mächtigkeit

1) Automat X äquivalent zu $Y \Leftrightarrow L(X) = L(Y)$

2) Klasse K_1 von Automaten äquivalent zu Klasse $K_2 \Leftrightarrow$ Zu jedem $X \in K_1 \exists \text{ äq. } Y \in K_2$ und umgekehrt.

[Klasse der DEA \cong Klasse der NEA \cong Klasse der NEAe \cong Klasse der EA \cong Klasse NRSA]

Definition Erweiterte Transitionsfunktion beim NEA

$\tilde{\delta}: P(Q) \times \Sigma^* \rightarrow P(Q)$

$\tilde{\delta}(P, \epsilon) = P$ ($P \subseteq Q$), $\tilde{\delta}(P, wa) = \delta_{\cup}(\tilde{\delta}(P, w), a)$ mit $\delta_{\cup}(R, a) = \cup_{p \in R} \delta(p, a)$

NEA A : $L(A) = \{ w \mid \tilde{\delta}(S, w) \cap F \neq \emptyset \}$

Satz DEA \Leftrightarrow NEA

\Rightarrow : $Q' := Q$, $S = \{q_0\}$, $F' = F$, $\delta'(q, a) = \{ \delta(q, a) \}$

\Leftarrow : Stundenübung! !!!

Algorithmus NEA \Rightarrow DEA

1. $Q' := \{S\}$

2. $D := \{ \cup_{p \in S} \delta(p, a) \mid a \in \Sigma \}$ (Die Zustände, die mit einem Zeichen erreichbar)

3. $\forall X \in D$ ($X \subseteq Q$!!), $X \notin Q'$, bilde Update $Q' = Q' \cup \{X\}$.

$D = D \cup \{ \cup_{p \in X} \delta(p, a) \mid a \in \Sigma \}$. GOTO 3

Praxis: Tabelle Zustand - Symbol. Sukzessive erweitern. Neue Zustände: Mehrfachindizes.

Dabei „Leere Menge“ beachten!

Definition EA

$A = (Q, \Sigma, P, S, F)$

Q, Σ, S, F wie NEA

$(Q \cup \Sigma, P)$ ist Semi-Thue-System, $P \subseteq Q \Sigma \times Q$, d.h. $qa \rightarrow q'$

Arbeitsschritt: (q, a, q') mit $(qa \rightarrow q') \in P$

Arbeitsvorgang: s.o.

Erkennen: s.o.

Satz **EA \leftrightarrow NEA**

$$\Rightarrow: \delta(q,a) = \{ p \mid p \in Q, (qa \rightarrow p) \in P \}$$

$$\Leftarrow: P = \{ (qa \rightarrow p) \mid p \in \delta(q,a) \}$$

Bemerkung **EA \leftrightarrow DEA (?)**

EA entspricht DEA, wenn $|S|=1$ und $|\{ p \mid (qa \rightarrow p) \in P \}|=1$

Bemerkung **Darstellung des Nichtdeterminismus**

- Funktion, die in die Menge der Alternativen abbildet

- Relation als Verallgemeinerung von Funktion: $R \subseteq A \times B$

Definition **NEA ϵ**

$$A = (Q, \Sigma, \delta, S, F)$$

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$$

Arbeit, Arbeitsschritt, Leistung

Definition **Erweiterte Transitionsfunktion**

$$\tilde{\delta}^*: \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$$

$\tilde{\delta}^*(P, \epsilon) = S_\epsilon(P) =$ Menge aller von P durch ϵ -Übergänge oder Nichtstun erreichbare Zst.

$$S_\epsilon(P) = P \cup \delta_{\{\epsilon\}}(S_\epsilon(P), \epsilon), \quad \delta_{\{\epsilon\}} \text{ wie oben}$$

$$\tilde{\delta}^*(P, wa) = S_\epsilon(\delta_{\{a\}}(\tilde{\delta}^*(P, w), a))$$

Satz **NEA $\epsilon \Leftrightarrow$ NEA**

$$\Rightarrow: \delta^*(q,a) = \tilde{\delta}^*(\{q\}, a), \quad F' = F \cup \{ q \in S \mid S_\epsilon(q) \cap F \neq \emptyset \} \text{ (Die, die } \epsilon \text{ direkt sammelt)}$$

$$\Leftarrow: \text{Triv.}$$

Beweis: Fallunterscheidungen!!!

Definition **NRSA (Nicht-deterministischer Rabin-Scott-Automat)**

Wie NEA ϵ , kann aber in einem Schritt ganzes Wort (oder auch leeres Wort) lesen.

Folgerung **Klasse der von allen DEA's, NEA's, NEA ϵ 's und NRSA's erkannten Sprachen gleich****2.3** **Endliche Automaten mit Ausgabe**

Mealy-Automat: Ausgabe in Abhängigkeit von Zustand & Eingabe.

Moore-Automat: Ausgabe in Abhängigkeit vom Zustand.

Definition **Mealy-Automat**

$$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

Q: Endliche, nichtleere Menge, Zustandsmenge

Σ, Δ : Zwei Alphabete: Ein- und Ausgabealphabet

$\delta: Q \times \Sigma \rightarrow Q$ totale Fkt.

$\lambda: Q \times \Sigma \rightarrow \Delta$ totale Fkt.

$q_0 \in Q$: Anfangszustand

Definition **Ausgabefolge**

Eingabefolge $a_1 a_2 \dots a_n \in \Sigma^*$

Ausgabefolge $b_1 b_2 \dots b_n \in \Delta^*$ mit $b_i = \lambda(q_{i-1}, a_i), q_i = \delta(q_{i-1}, a_i)$

Beispiel **Serienaddierer****Satz** **Es gibt keinen endlichen Mealy-Automaten, der Binärzahlen multipliziert**

Beweis: Übung!!!

Definition **Moore-Automat**

$$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

Differenz zum Mealy-Automaten: $\lambda: Q \rightarrow \Delta$ total

Definition **Ausgabefolge**

Ausgabefolge $b_0b_1b_2\dots b_n \in \Delta^*$ mit $b_i = \lambda(q_i)$, $q_i = \delta(q_{i-1}, a_i)$. Beachte b_0 ! Länge!

Definition **Äquivalenz von Automaten mit Ausgabe**

Moore- und Mealy äquivalent, wenn sich die zu gleicher Eingabefolge produzierten Ausgaben nur um $b_0 = \lambda(q_0)$ unterscheiden.

2.4 **Charakterisierung der Klasse regulärer Sprachen**

Definition **Äquivalenz**

$L(A_1) = L(A_2)$. Sprache erkannt (EA) oder erzeugt (Grammatik) oder beschrieben (Reg. Ausdr.), beide Seiten gleich!

Definition **Korrespondenz (!!!)**

s.o., linke und rechte Seiten verschiedene Darstellungen.

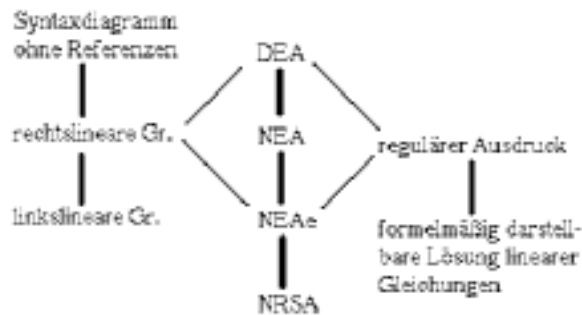
Beispiel **Korrespondenz (<->)**

Endlicher Automat <-> regulärer Ausdruck <-> Typ-3-Grammatik <-> geschlossene Lsg. linearer Gleichungen

Bemerkung **Charakterisierung von regulären Sprachen**

Direkter Nachweis: Finde reg. Ausdruck, reg. Grammatik oder endlichen Automaten
 Pumping-Lemma (Destruktiv)
 Abschlußeigenschaften (L_1, L_2 regulär $\Rightarrow L_1 \cup L_2$ regulär)

Übersicht **über Korrespondenzen und Äquivalenzen der Chomsky-Hierarchie**



Satz **Endliche Automaten <--> regulärer Ausdruck**

\Leftarrow : Regulärer Ausdruck \rightarrow NEA ϵ . Induktion über Aufbau von regulären Ausdrücken.
 $r = \emptyset \Rightarrow Ar$: erkennt nichts. ..., $r = r_1r_2$: Automaten konkatenieren
 \Rightarrow : DEA \rightarrow Regulärer Ausdruck. MANNA: Gegeben: DEA als Transitionsdiagramm. Neuer Startzustand S und Endzustand f. Streiche einen Zustand, setze direkte Pfeile für alle Durchläufe und beschrifte korrekt. Fortsetzen, bis nur ein Pfeil: regulärer Ausdruck
 2. Beweis: Menge R_{ij}^k = Menge aller Worte durch Übergang q_i nach q_j mit höchstens q_1, \dots, q_k als Zwischenzustände.
 $R_{ij}^k = \{ w \in \Sigma^* \mid \tilde{\delta}(q_i, w) = q_j, \tilde{\delta}(q_i, \text{Linker Teil von } w) \in \{q_1, \dots, q_k\} \}$.
 $R_{ij}^0 = \{ a \in \Sigma \mid \delta(q_i, a) = q_j \} \cup \{ \epsilon \mid i=j \}$ ist regulär. Rekursion: $R_{ij}^k = R_{ij}^{k-1} \cup R_{ik}^k (R_{kk}^k)^* R_{kj}^k$.
 Induktion \Rightarrow auch regulär. $L = \cup R_{ij}^n$ für alle $q_j \in F$ regulär.

Satz **Pumping-Lemma, UVW-Theorem**

Sei L reg. Sprache. Dann $\exists n \in \mathbb{N}$: Jedes $z \in L$ mit $|z| \geq n$ zerlegbar in $z = uvw$ mit
 $|uv| \leq n, |v| \geq 1$ ($v \neq \epsilon$), $uv^i w \in L \forall i \geq 0$
 Beweis: Anschaulich. Pfad, Zustand doppelt,...

Definition **Abschlußeigenschaft**

$\mathcal{L}_1 = \{ L \mid L \text{ ist Typ-i-Sprache} \}$. Speziell: $\mathcal{L}_3 = \{ L \mid L \text{ reguläre Sprache} \}$

$f: \mathcal{L} \rightarrow \mathcal{L}$ (z.B. Permutation), $g: \mathcal{L} \times \mathcal{L} \rightarrow \mathcal{L}$ (z.B. Konkatenation)

Gilt $f(\mathcal{L}_1) \subseteq \mathcal{L}_1$, so f erfüllte Abschlußeigenschaft. Analog Negat. Analog $g(\mathcal{L}_1 \times \mathcal{L}_1) \subseteq \mathcal{L}_1$

Satz **Reguläre Operationen (5 Stück)**

Sind L_1, L_2 regulär, so auch:

$L_1 L_2, L_1 \cup L_2, L_1^*, L_1 \cap L_1, \Sigma^* - L_1$

Beweis: \cap durch Automaten mit \times , siehe Übung. Komplement durch Automaten, $F := Q - F$

Redeweise: \mathcal{L}_3 abgeschlossen unter $\cdot, \cup, \cap, *, C$

GÜ Satz **Abschluß unter speziellen Funktionen**

\mathcal{L}_3 abgeschlossen unter Substitution $\sigma: \Sigma \rightarrow \mathbf{P}(\Delta^*)$, Homomorphismus $h: \Sigma \rightarrow \Delta^*$,

inversen Homomorphismus $h^{-1}: \Delta^* \rightarrow \Sigma$

Beweis: Substitution: Induktion über Aufbau regulärer Mengen oder anschaulich: Jede Überführung durch einen NEA ϵ ersetzen, der mit ϵ -Übergängen angeschlossen wird.

Homomorphismus: A' liest $x \Leftrightarrow$ Arbeitsweise emuliert, als ob A $h(x)$ liest. Induktion

Definition **Entscheidbarkeit zu Typ-3 der Chomsky-Hierarchie**

\Leftrightarrow Es gibt zu Problem einen Algorithmus, der immer hält, und zu jeder Instanz des Problems korrekte Antwort liefert. Beispiel: Totschleife eines Programms

Satz **über Kardinalität von $L \in \mathcal{L}_3$**

$L(A) \neq \emptyset \Leftrightarrow A$ akzeptiert Wort w mit $|w| < n$

$L(A)$ unendlich $\Leftrightarrow A$ akzeptiert Wort mit $n \leq |w| < 2n$

Satz **Entscheidbare Aussagen für Typ-3-Grammatiken**

1) $w \in L$, 2) $L(A) = \emptyset$, 3) $L = \Sigma^*$

Beweis: 1) DEA aufbauen, Wort lesen. Endzustand?

2) Alle Worte w mit $|w| < n$ durchprobieren. Oder:

$X := F$. $X := X \cup \{q\}$ für $q \notin X$ und $\exists a$ mit $\delta(q, a) \in X$. Solange noch Veränderungen. $q_0 \in X$?

3) Komplementären Automaten betrachten, dann 2)

3 **Kellerautomaten****3.1** **Definition von Kellerautomaten**

EA reichen nicht zur Erkennung *kontextfreier* Sprachen!

Definition **Kellerautomat**

$KA = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

Q : Endliche Menge, Zustandsmenge

Σ : Eingabealphabet

Γ : Kelleralphabet

δ : Abbildung (Transitionsfunktion). $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \mathbf{P}(Q \times \Gamma^*)$

mit $\delta(q, a, Z)$ endlich für jedes Tripel (q, a, Z)

$q_0 \in Q$: Anfangszustand

$Z_0 \in \Gamma$: Startsymbol des Kellers (Keller-Grundsymbol)

$F \subseteq Q$: Endzustandsmenge

Definition **"Erkennen"**

Variante 1: Restwort & Keller leer

Variante 2: Restwort leer, KA in einem Endzustand

Definition Konfiguration

$(q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$. q ist aktueller Zustand, w das Restwort und γ der Kellerinhalt

Ausgangskonfiguration: (q_0, w, Z_0) .

Folgekonfigurationen durch Relation \vdash definiert. $(q, aw, z\alpha) \vdash (p, w, \gamma\alpha)$ für $(p, \gamma) \in \delta(q, a, z)$

Definition KA erkannte Sprache

Leerer Keller: $N(A) = \{ w \in \Sigma^* \mid (q_0, w, z_0) \vdash^* (q, \varepsilon, \varepsilon), q \in Q \}$

Endzustand: $L(A) = \{ w \in \Sigma^* \mid (q_0, w, z_0) \vdash^* (q, \varepsilon, \gamma), q \in F, \gamma \in \Gamma^* \}$

Definition Transitionsdiagramm für KA

Syntax: $a, A/BCD$. a gelesen, A auf Keller. a weglesen, BCD auf Keller schreiben

Definition Darstellung KA als Semi-Thue-System

$\delta(q, a, X) \ni (p, \gamma) \Rightarrow X q a \rightarrow \gamma p$ mit invertiertem Keller!. X Keller, q Zustand, a Symbol
Nichtdeterministisch. Teilweisen Determinismus durch Wort-Ende-Symbol #.

3.2 Parsing-Methoden & Kellerautomaten

Parsing: Syntaktische Zergliederung.

Parser: Aufstellen eines Ableitungsbaumes.

Gilt alles für kontextfreie Grammatiken

Definition Ableitungsbaum (Mit -->)

1. Geordneter Baum

2. Jeder Knoten beschriftet

3. Wurzel mit S beschriftet

4. Knoten kann nur Nachfolger haben, wenn mit Nichtterm beschriftet. Nachfolger sind die X_1, \dots, X_k mit $X \rightarrow X_1 X_2 \dots X_k$ bzw. ε , falls $X \rightarrow \varepsilon$

\Rightarrow Blätter v.l.n.r. bilden Satzform

GÜ Definition ... Entscheidungsbaum

Zu jeder Satzform alle möglichen Ersetzungen in den Nachfolgeknoten

Definition Kanonische Ableitung (Nach fester Regel)

Kanonische Linksableitung, wenn in jedem Schritt das am weitesten links stehende Nichtterm ersetzt wird. Analog Kanonische Rechtsableitung

Bemerkung Allgemeine Parsing-Methoden

Top-Down-Parsing: S vorgeben, versuchen, w zu erreichen

Bottom-Up-Parsing: w vorgeben, Regeln rückwärts anwenden, versuchen, S zu erreichen

Konstruktion KA für Top-Down-Parsing

Semi-Thue-Darstellung:

$X \rightarrow X_1 X_2 \dots X_k \Rightarrow X q \rightarrow X_k X_{k-1} \dots X_1 q$ (Inversion!)

Für alle $a \in \Sigma \Rightarrow a q a \rightarrow q$ (Auffressen!)

Konstruktion KA für Bottom-Up-Parsing

$X \rightarrow X_1 X_2 \dots X_k \Rightarrow X_1 X_2 \dots X_k q \rightarrow X q$ (Inversion!). δ hier verallgemeinert!

Für alle $a \in \Sigma \Rightarrow q a \rightarrow a q$ (Richtige auffressen!)

Übersicht Chomsky-Hierarchie

	Typ-III	Typ-II	Typ-I	Typ-0
Grammatik	Reguläre G. · rechtslineare G. · linkslineare G.	Kontextfreie G. (Normalform: Chomsky, Greibach)	Monotone G. (Normalform: Kontextsens. G.)	Phrasenstruktur-G.
Sprache	Reguläre Sp.	Kontextfreie Sp.	Kontextsensitive Sp.	Aufzählbare Sp.
Weitere Charakterisierung:	· Reguläre Ausdrücke · Lösung rek. Mengengl.	· Rekursive Gl.	-	Wird von TM aufgezählt
Erkennender Automat	EA	(N-det.) KA	Einschränkung der TM	TM
Beispiel		Pascal ohne „Variablen vor Nutzung deklariert“	Deutsche Sprache	

Übersicht Schachtelung der Sprachklassen

Endliche Sprachen \subset
 Typ-III \subset
 Det. kf. Sprachen (haben det. KA) \subset
 Typ-II \subset
 Typ-I \subset
 Entscheidbare Sprachen \subset
 Typ-0 (Aufzählbar) \subset
 Alle Sprachen ($UP(\Sigma^*)$)

Beispiele Sprachen

L_d = Diagonalsprachen: Nicht aufzählbar
 $a^n b^n c^n$: Typ-I
 $a^n b^n$: Det. kf. Sprache \subset Typ-II

Satz Pumping-Lemma für kontextfreie Sprachen

Zu $L \in \mathcal{L}_2, \exists n \in \mathbb{N}, \forall z \in L, |z| \geq n \exists$ Zerlegung $z=uvwxy$ mit
 1) $|vwx| \leq n, \quad 2) |vx| \geq 1, \quad 3) uv^iwx^iy \in L$
 Spezialfall für reguläre Sprachen: Rechtslinear: $u=v=\epsilon$. Linkslinear: $x=y=\epsilon$
 Beweisidee: Ableitungsbäume

4 **Compilerbau**

4.1 **Einführung**

Definition Compiler

Erkennender Automat, der Struktur auswertet, falls das „Wort“ (Programm) zur (Typ-II-) Sprache gehört.

Bemerkungen ... **Kontextfreie Sprachen**

L vom Typ-II $\Leftrightarrow \exists$ Kellerautomat (i.a. nicht-det.), der L erkennt.
 \exists kontextfreie Sprachen, die von keinem det. KA erkannt. Für Praxis nicht relevant
 Relevant: Unterklasse der deterministischen kf. Sprachen.

Definition **Look-Ahead**

Ansatz: $Q=\Gamma$, d.h. Zustände und Symbole im Keller identisch. $\Rightarrow q_n \dots q_1 a \rightarrow q'$

Mit Look-Ahead: $q_n \dots q_1 a \beta \rightarrow q' \beta$, $\beta \in \Sigma^*$, $a \in \Sigma \cup \{\epsilon\}$

Mit Look-Ahead von genau k Symbolen: $q_n \dots q_1 a \beta \rightarrow q' \beta$, $\beta \in \Sigma^k$, $a \in \Sigma \cup \{\epsilon\}$

Definition LL(k), LR(k)

LL(k): Von Links lesen, Linkskanonische Abl., dabei k Sym. Look Ahead. Top-Down-Parsing.

LR(k): Von Links lesen, Rechtskanonische Abl., dabei k Look Ahead. Bottom-Up-Parsing.

Definition Kontextfreies Item einer kf. Sprache

Zustand eines erkennenden KA, $[X \rightarrow \alpha . \beta]$ bei $(X \rightarrow \alpha \beta) \in P$

Formal: Tripel $(X, \alpha, \beta) \in N \times (N \cup T)^* \times (N \cup T)^*$. $X \rightarrow \alpha \beta$ soll erledigt werden, bis α fertig.

Für Erkennung einer kf. Grammatik, d.h. ohne Look-Ahead.

Definition Item-Kellerautomat einer kf. Grammatik

$G=(T,N,P,S)$. $G':=(T, N \cup \{S'\}, P \cup \{S' \rightarrow S\}, S')$. KA $K=(Q, \Sigma, \Gamma, R, q_0, Z_0, F)$

$Q=\Gamma$ =Menge aller kontextfreien Items. $\Sigma=T$, $q_0=[S' \rightarrow .S]$, $Z_0=\epsilon$, $F=\{[S' \rightarrow S.]\}$.

$R=R_{Expansion} \cup R_{TerminalerShift} \cup R_{VariablenShift}$

$R_{Expansion} = \{ [X \rightarrow \beta . Y \gamma] \rightarrow [X \rightarrow \beta . Y \gamma] [Y \rightarrow . \alpha] \mid [X \rightarrow \beta . Y \gamma], [Y \rightarrow . \alpha] \in Q \}$

$R_{TerminalerShift} = \{ [X \rightarrow \beta . a \gamma] a \rightarrow [X \rightarrow \beta a . \gamma] \mid [X \rightarrow \beta . a \gamma] \in Q, a \in \Sigma \}$

$R_{VariablenShift} = \{ [X \rightarrow \beta . Y \gamma] [Y \rightarrow \alpha .] \rightarrow [X \rightarrow \beta Y . \gamma] \mid [X \rightarrow \beta . Y \gamma], [Y \rightarrow \alpha .] \in Q \}$

Definition k-Kopf eines terminalen Wortes $t \in T^*$

$k \geq 0$, $\# \notin T$, $t=t_1 t_2 \dots t_n$. $k : t = \{ t_1 t_2 \dots t_k, \text{ falls } n \geq k. t_1 t_2 \dots t_n \#, \text{ falls } n < k \}$

Definition Menge $First_k(w)$

$w \in (N \cup T)^*$. $First_k(w) = \{ k : u \mid w \Rightarrow u, u \in T^* \}$

Beispiel: $S \rightarrow ab \mid aSb$. $First_1(S)=\{a\}$, $First_2(S)=\{aa, ab\}$, $First_3(S)=\{aaa, aab, ab\# \}$

Definition Verallgemeinerung $First_k(w\Omega)$

$First_k(w\Omega) = \{ k : u x \mid w \Rightarrow u, (x \in \Omega \cap T^*) \vee (x \# \in \Omega) \}$, $\Omega \subseteq \{ k : t \mid t \in T^* \}$

Definition Kontextsensitives Item einer kf. Grammatik

$A \rightarrow \alpha \beta$, $k \geq 0$, Ω Menge von k-Köpfen. $\Rightarrow [A \rightarrow \alpha . \beta; \Omega]$ kontextsensitives Item

Ableitungsprozeß soweit fortgeschritten, daß α abgeleitet ist; auf das aus $\alpha \beta$ abgeleitete terminale Wort folgen Symbol-Sequenzen mit Länge $\leq k$ aus Ω .

Definition Eigenschaft LL(k)

\Leftrightarrow Für alle Ableitungssituationen gilt:

Aus $S \Rightarrow^L \mu A \chi \Rightarrow^{A \rightarrow v} \mu v \chi \Rightarrow^* \mu \gamma$ und $S \Rightarrow^L \mu A \chi \Rightarrow^{A \rightarrow w} \mu w \chi \Rightarrow^* \mu \gamma'$, $\mu, \gamma, \gamma' \in T^*$, $\chi, v, w \in (T \cup N)^*$, $A \in N$, und $k : \gamma = k : \gamma'$ folgt: $v=w$

Satz Algorithmus LL(k)

Eingabe: Bel. kf. Grammatik, $k \geq 0$. Ausgabe: KA $A=(Q, \Sigma, \Gamma, R, q_0, z_0, F)$,

$L(G)=N(A)$ („leerer Keller“). $\Gamma=Q$, $\Sigma=T$, $q_0=[S' \rightarrow .S; \{\#\}]$, $F=bel.$ $z_0=q_0$.

1. $Q:=\{q_0\}$, $R:=\emptyset$.

2. Wähle noch nicht betrachtetes $q \in Q$, $q=[X \rightarrow m.n; \Omega]$

3. Falls $n=\epsilon$, dann $R:=R \cup \{q \rightarrow \epsilon\}$ („Pop-Regel“)

4. Falls $n=tg$, $t \in T$, $g \in (N \cup T)^*$, dann $q'=[X \rightarrow mt.g; \Omega]$, $Q=Q \cup \{q'\}$, $R=R \cup \{qt \rightarrow q'\}$ („Terminaler Shift“)

5. Falls $n=Bg$, $B \in N$, $g \in (N \cup T)^*$, dann $q'=[X \rightarrow mB.g; \Omega]$, und für jede Regel $B \rightarrow b_i$:

$h_i := [B \rightarrow .b_i; First_k(g\Omega)]$

B ableiten, danach kommt Kontext Ω

$Q:=Q \cup \{q'\} \cup U\{h_i\}$

$R:=R \cup \{q t_i \rightarrow q' h_i t_i \mid t_i \in First_k(b_i g \Omega)\}$ („Variablenshift“)

Alle Ableitungen im korrekten Kontext

6. Solange 2) bis alle Zustände betrachtet.

- Satz** **LL(k)**
 Sprache LL(k) \Leftrightarrow LL(k)-Algorithmus erzeugt det. KA (sonst n.det. KA)
- Satz** **Kf. Sprachen genau die durch (n.-det.) KA definierten Sprachen**
 Beweis: \Rightarrow : Konstruiere n.-det. KA (top-down-/ bottom-up-Parsing)
 \Leftarrow : Ohne Beweis
- Definition** **Klasse der det. kf. Sprachen**
 Klasse der Sprachen, zu denen ein det. KA existiert.
- Definition** **LL(k)-Klassen**
 Klassen von Grammatiken, welche garantieren, daß die zugehörigen Sprachen det. kf. sind.
 Nur beispielsweise: $LL(0) \subseteq LL(1) \subseteq \dots \subseteq LL(k) \subseteq LL(k+1) \subsetneq$ Det. kf. Sprachen
- Bemerkung:**
 Grammatik nicht in einer dieser Klassen \nRightarrow äquivalente Grammatik nicht in einer der Klassen
 L det. kf. \nRightarrow Es ex. zugehörige Grammatik aus einer der obigen Klassen.
- Satz** **LL(k)**
 \exists kf. Grammatiken zu det. Sprachen, die **nicht** LL(k) $\forall k \geq 0$
- Beispiel** **LL(k)**
 $Z \rightarrow X, X \rightarrow Yc, X \rightarrow Yd, Y \rightarrow a, Y \rightarrow bY$ nicht LL(k).
 $Z \rightarrow X, X \rightarrow YX', X' \rightarrow c, X' \rightarrow d, Y \rightarrow a, Y \rightarrow bY$ dazu äquivalent und LL(1).
- Satz** **Linksrekursion**
 Grammatik besitzt Linksrekursion \Rightarrow Bei Top-Down-Parsing Nicht-Determinismus
 Linksrekursion durch Rechtsrekursion ersetzbar.
- Satz** **LL(k)**
 Grammatik LL(k) \Rightarrow Grammatik besitzt keine Linksrekursion
- Satz** **LL(k) mit ϵ -Produktionen**
 G LL(k) mit ϵ -Produktionen $\Rightarrow \exists G'$ LL(k+1) ohne ϵ -Produktionen, die $L(G) \setminus \{\epsilon\}$ produziert
 G LL(k+1) ohne ϵ -Produktionen $\Rightarrow \exists$ äquivalentes G' LL(k) mit ϵ -Produktionen
- Übersicht** **Pumping-Lemma**

Für reguläre Sprachen

Für kontextfreie Sprachen

Für jede Sprache L \exists ein $n \in \mathbb{N}$, so daß für $z \in L, |z| \geq n$ gilt:

$z=uvw$		$z=uvwxy$
$ uv \leq n$		$ vwx \leq n$
$ v \geq 1$		$ vx \geq 1$
$u v^i w \in L$	$\forall i \geq 0$	$u v^i w x^i y \in L$

Beweisidee: Mindestens ein doppelter Zustand in Zustandsfolge

Beweisidee: Chomsky-Normalform. Ableitungsbaum Tiefe i \Rightarrow Worte $|w| \leq 2^{i-1}$ per Induktion. $n := 2^k$ mit $k :=$ Anz. der Non-Terms. \Rightarrow Tiefe $> k \Rightarrow$ Im Pfad ein Non-Term. doppelt. Von unten her bis zu diesen beiden Non-Terms.

Bemerkung **Anwendung des Pumping-Lemma**
 Zeigen, daß L **nicht** kf. ist

$G(M) := \{ w \in \Sigma^* \mid \# w \# \text{ wird irgendwann auf Ausgabeband geschrieben} \}$
 ist die von M erzeugte Sprache

Definition **Sprache rekursiv aufzählbar**

L rekursiv aufzählbar \Leftrightarrow Es gibt GTM M mit $L=G(M)$

Satz **L rekursiv aufzählbar $\Leftrightarrow L=L(M)$ für eine TM M**

Wichtig: Die TM M muß *nicht* für jede Eingabe halten!

Beweis: \Rightarrow : Konstruktion TM aus GTM: Wort suchen. *Hält nicht unbedingt!*

\Leftarrow : GTM zählt Worte aus Σ^* auf, simuliert darauf die TM und schreibt evtl. auf Ausgabeband. *Simulierte TM könnte nicht halten!*

Bemerkung **Probleme dieses Satzes**

- Wie erzeugt man alle Worte? (Nutze lexikographische Ordnung.)

- TM M kann bei gewissen Eingaben niemals halten. (Erzeuge nach CAUCHYSchem Diagonal-Verfahren Paare (i, j) und versuche, das i -te Wort in j Schritten zu erkennen)

Definition **Lexikographische Ordnung**

1. Vergleichskriterium: Wortlänge. Kürzere Worte zuerst.

2. Vergleichskriterium: 1. verschiedenes Symbol durch Ordnung im Alphabet.

Definition **Sprache rekursiv**

L rekursiv $\Leftrightarrow L=G(M)$ Generator-TM, zählt in lexikographischer Ordnung auf

Satz **L rekursiv $\Leftrightarrow L = L(M)$, M TM, die für jede Eingabe hält**

Beweis: \Rightarrow : GTM simulieren, Wort in Ausgabe suchen. Wortlänge überschritten \Rightarrow Abbruch

\Leftarrow : Σ^* lexikographisch aufzählen, M simulieren.

Satz **„ L rekursiv“ bzgl. Komplement abgeschlossen**

$L \subseteq \Sigma^*$ rekursiv $\Rightarrow \Sigma^* \setminus L$ rekursiv

Beweis: Σ^* aufzählen, in Ausgabe der GTM suchen

Satz **Sprache und Komplement rekursiv aufzählbar \Rightarrow Beide rekursiv**

L und $CL = \Sigma^* \setminus L$ rekursiv aufzählbar $\Rightarrow L$ und CL rekursiv

Beweis: Eingegebenes Wort **parallel** in Ausgabe der beiden GTM suchen, findet es in endlich vielen Schritten in der einen oder der anderen

Korollar **L Sprache, so gilt genau eine der folgenden Aussagen:**

1. L, CL rekursiv

2. Weder L noch CL rekursiv aufzählbar

3. Entweder L oder CL rekursiv aufzählbar und die jeweils andere nicht rekursiv aufzählbar

5.2 Entscheidbarkeit, Unentscheidbarkeit und eine universelle TM

Definition **Problem, Instanz, Lösung**

Problem: Liste von Parametern und Fragestellung, auf die in Abhängigkeit von Parametern eine Antwort zu geben ist.

Instanz eines Problems: Problem mit festgelegten Werten für die Parameter.

Lösung eines Problems: Algorithmus, der für alle Instanzen die Fragestellung beantwortet.

Parameter und Fragestellung müssen syntaktisch korrekt sein!

Definition **Sprache des Problems**

$L_{\Pi} := \{ w \mid w \text{ ist Instanz von } \Pi \}$, Π ist ein Problem

Definition **Entscheidungsproblem**

Problem, bei dem die Fragestellung eine Ja/Nein-Antwort verlangt

Besteht aus L_{Π} , Menge $Y_{\Pi} \subseteq L_{\Pi}$ und der Fragestellung „Ist $w \in L_{\Pi}$ in Y_{Π} ?“

Definition **Entscheidbar, unentscheidbar, semi-entscheidbar**
 Entscheidungsproblem Π **entscheidbar**, wenn die Sprache Y_Π rekursiv, sonst **unentscheidbar**. **Semi-entscheidbar**, wenn Y_Π rekursiv aufzählbar.

Definition **Halteproblem**
 M eine TM, w ein Wort. „Hält M bei der Eingabe von w?“

Definition **GÖDEL-Nummer**
 Codierung einer TM $M=(Q, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$. $x_1:=0$ $x_2:=1$ $x_3:=B$ $R_1:=1$ $R_2:=r$
 Codiere $\delta(q_i, x_j) = (q_k, x_p, R_n)$ eindeutig durch das Wort $\alpha_i = 1 0^i 1 0^j 1 0^k 1 0^l 1 0^n$
 Codiere M als $\langle M \rangle := 111 \alpha_1 11 \alpha_2 11 \dots \alpha_r 111$, α_i alle Überführungen. GÖDEL-Nummer
 $L_{TM} := \{ \langle M \rangle \mid M \text{ ist TM} \} \subseteq \{0, 1\}^*$

Lemma **L_{TM} ist rekursiv**
 Beweis: GM zählt Worte über $\{0, 1\}$ lexikographisch auf und schreibt nur die syntaktisch korrekten auf Ausgabeband

Korollar **Problem „Ist $w \in L_{TM}$?“ entscheidbar**

Korollar **$L \subseteq \{0, 1\}^*$ rekursiv aufzählbar $\Leftrightarrow L=L(M)$ mit $\langle M \rangle \in L_{TM}$**

Beispiel **Nicht-rekursiv-aufzählbare Sprache**
 Mengen $\{0, 1\}^*$ und LTM aufzählen. $A(i, j)=1 \Leftrightarrow$ Maschine M_j erkennt Wort w_i . $A(i, j)=0 \Leftrightarrow$ Maschine M_j erkennt Wort w_i nicht. Diagonalsprache L_d definieren durch: $w_i \in L_d \Leftrightarrow A(i, i)=0$. Annahme: L_d rekursiv aufzählbar $\Rightarrow \exists$ TM M, die L_d über $\{0, 1, B\}$ akzeptiert. $\Rightarrow \exists i$ mit $\langle M \rangle = \langle M_i \rangle$. $\Rightarrow w_i \in L_d \Leftrightarrow w_i \in L(M) \Leftrightarrow w_i \in L(M_i) \Leftrightarrow A(i, i)=1$. Widerspruch!

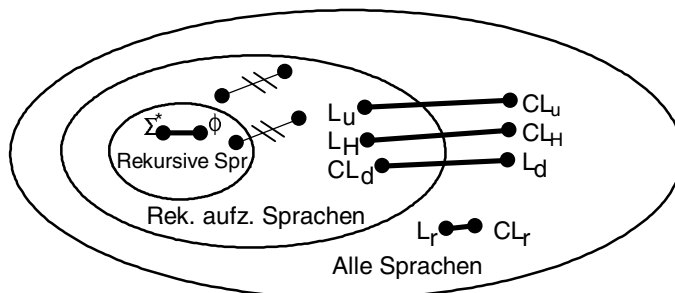
Definition **Universelle Sprache L_u**
 w Wort, $\langle M \rangle$ Codierung einer TM M. $\langle M \rangle w$ Konkatenation der Wörter $\langle M \rangle$ und w.
 Universelle Sprache $L_u := \{ \langle M \rangle w \mid M \text{ akzeptiert } w \}$

Satz **L_u rekursiv aufzählbar**
 Beweisidee: 3 Bänder. 1: Eingabeband $\langle M \rangle w$, 2: Kopie von w, 3: Zustand der simulierten Maschine. Diese TM M_u : **universelle TM**.

Satz **L_u nicht rekursiv**
 Beweis: Indirekt, Annahme $\Rightarrow CL_d$ rekursiv \Rightarrow Wid.

Übersicht **Rek. (aufz). Sprachen und ihre Komplemente**

- $L_r := \{ \langle M \rangle \mid L(M) \text{ rekursiv} \}$
- $L_u := \{ \langle M \rangle w \mid M \text{ akzeptiert } w \}$ Universelle Sprache
- $L_H := \{ \langle M \rangle w \mid M \text{ hält bei Eingabewort } w \}$



Satz **L_H rekursiv aufzählbar, aber nicht rekursiv**
 Beweis: Rekursiv aufzählbar: TM simulieren.
 Nicht rekursiv: Indirekt. Annahme: L_H rekursiv \Rightarrow TM hält für jedes Wort $\langle M \rangle w$. ???
 „Um herauszufinden, ob ein Algorithmus Eingabe akzeptiert bzw. hält, bleibt nichts anderes übrig, als ihn laufen zu lassen und zu warten.“

Satz Typ-0-Sprachen genau die rekursiv aufzählbare Sprachen

$L=L(G)$ Typ-0 $\Leftrightarrow L=L(M)$ für TM M

Beweis: \Leftarrow : n-det. Worte $[a_1, a_1] \dots [a_n, a_n]$ erzeugen. Regeln wie z.B. $q [x, a_i] \rightarrow [y, a_i] q'$. Bei Erreichen eines Endzustandes $q_f [x, a_i] \rightarrow a_i$ 1. Komponenten vernichten.
 \Rightarrow : TM als „universeller Rechner“ kann Ableitung von Typ-0 simulieren

Definition Linear beschränkter Automat (LBA)

Nicht-det. TM mit Einschränkung: Symbole $\$$ und $\#$ als linke und rechte Endmarkierung, die nie verändert und nie überschritten werden.

M LBA, $L(M) := \{ w \mid w \in (\Sigma \setminus \{ \#, \$ \})^*, \# w \$ \text{ wird von } M \text{ akzeptiert} \}$

Satz Typ-1-Sprachen genau die von LBA erkannten Sprachen

$L=L(G)$ Typ-1 $\Leftrightarrow L=L(M)$ für LBA M

Beweisidee: \Leftarrow : n-det. Worte $\# [z, a_1, a_1] \dots [z, a_n, a_n] \$$ erzeugen, $[z, a_i, a_i]$ ein Symbol. z gibt für ein Symbol Zustand des Kopfes an, $z \neq \#$ für restliche Symbole (Werden aktuell nicht vom Kopf gelesen). Regeln wie z.B. $[q, x, a_i] [\#, z, a_{i+1}] \rightarrow [\#, y, a_i] [q', z, a_{i+1}]$. Bei Erreichen eines Endzustandes $[z, x, a_i] \rightarrow a_i$ 1. Komponenten vernichten. Grammatik ist monoton
 \Rightarrow : Erzeuge n-det. Worte $\# w \$$ und simuliere durch LBA die Grammatik. Endmarkierungen nicht zu überschreiten, da Grammatik monoton ???

Satz Hierarchiesatz (CHOMSKY)

$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_{rek} \subset \mathcal{L}_{rek.aufz.} = \mathcal{L}_0$ mit echter Enthaltensbeziehung
 Ohne Beweis oder klar.


Bemerkung Hierarchiesatz

Keine echte Typ-0-Grammatik bekannt, die nicht Typ-1

Satz Reguläre Sprachen genau die von linkslinearer Grammatik erzeugte Sprachen

Beweis: \Rightarrow Induktiv über Aufbau regulärer Mengen
 \Leftarrow : Reguläre Sprachen gegenüber Spiegelung abgeschlossen \Rightarrow Aus rechtslinearer Grammatik folgt linkslineare Grammatik

Übersicht Korrespondenzen und Äquivalenzen der CHOMSKY-Hierarchie

Ansage! 

Sprachklasse	\mathcal{L}_3	\mathcal{L}_2	\mathcal{L}_1	\mathcal{L}_{rek}	$\mathcal{L}_{rek.aufz.} = \mathcal{L}_0$
Grammatik	rechtslinear, linkslinear, Syntaxdiagramm ohne Referenzen	Kontextfreie Gr., (CNF, GNF) Syntaxdiagramme mit Referenzen	Monotone Gr., Kontextsensitive Gr.		Unbeschränkte Gr., Uneingeschr. Gr., Phrasenstruktur-Gr.
Erkennender Automat	DEA, NEA, NEA ϵ , NRSA	KA	LBA	TM, die für alle Eingaben hält	TM
Sonstige Sprachdefinitione n	Regulärer Ausdr., reguläre Mengen	„lineares“ Gleichungssystem		Lexikographisch aufzählbar	(Rekursiv) aufzählbar
$\cup, \cap, *, ^R$	Abg.	Abg.	Abg.	Abg.	Abg.
Komplement	Abg.	---	Abg.	Abg.	---
Durchschnitt	Abg.	---	Abg.	Abg.	Abg.
Substitution	Abg.	Abg.	Abg.	---	---
„Ist $w \in L$?“	E	E	E	E	U
„Ist $L = \emptyset$?“	E	E	U	U	U
„Ist $L = \Sigma^*$?“	E	U	U	U	U
„Ist $L_1 = L_2$?“	E	U	U	U	U

„Ist Kompl. in \mathcal{L} “	Triv.	U	Triv.	Triv.	U
-----------------------------------	-------	---	-------	-------	---

6 Komplexitätstheorie

6.1 Grundzüge

Definition Off-line-TM

Ein schreibgeschütztes Eingabeband und mehrere einseitig unendliche Arbeitsbänder

Definition Bandkomplexität $S(n)$

M Off-Line-TM und $S: \mathbb{N} \rightarrow \mathbb{N}$ Funktion derart, daß M für Eingabe der Länge n nicht mehr als $S(n)$ Felder auf jedem Speicherband bearbeitet

Definition Zeitkomplexität $T(n)$

M mehrbändige TM mit beidseitig unendlichen Bändern, $T: \mathbb{N} \rightarrow \mathbb{N}$ Funktion derart, daß M für jede Eingabe der Länge n nicht mehr als $T(n)$ Kopfbewegungen durchführt

Vereinbarung ... Bandkomplexität

Jede TM benötigt wenigstens ein Feld für jede Eingabe, benötigt daher $\max(1, S(n))$ Felder.

Vereinbarung ... Zeitkomplexität

$T(n) \geq n+1$, da die Eingabe komplett gelesen werden soll. $\max(n+1, T(n))$

6.2 Komplexitätsklassen

Definition Komplexitätsklassen DSPACE, DTIME, NSPACE, NTIME

$DSPACE(S(n)) := DBAND(S(n)) := \{ L \mid \text{Es gibt TM, die } L \text{ erkennt und für jede Eingabe der Länge } n \text{ höchstens } \max\{1, S(n)\} \text{ Felder auf jedem Band (zusätzlich zur Eingabe) benötigt} \}$

Sprachfamilie von Bandkomplexität $S(n)$.

$DTIME(T(n)) := DZEIT(T(n))$ analog **Zeitkomplexität $T(n)$.**

$NSPACE(S(n)) := NBAND(S(n)) := \{ L \mid \text{Es gibt NTM, die } L \text{ erkennt und für jede Eingabe der Länge } n \text{ höchstens } \max\{1, S(n)\} \text{ Felder auf jedem Band (zusätzlich zur Eingabe) benötigt} \}$

Sprachfamilie von Bandkomplexität $S(n)$.

$NTIME(T(n)) := NZEIT(T(n))$ analog **Zeitkomplexität $T(n)$.**

Satz Bandkompression

$\forall c > 0: DSPACE(S(n)) = DSPACE(c \cdot S(n)), NSPACE(S(n)) = NSPACE(c \cdot S(n))$

Beweis: Zusammenfassen von Symbolen unter Hinzufügen neuer Regeln

Satz Lineare Beschleunigung

1) $\liminf T(n)/n = \infty \Rightarrow \forall c > 0: DTIME(S(n)) = DTIME(c \cdot S(n)).$ NTIME analog.

2) $T(n) = r \cdot n, r > 1 \Rightarrow \forall \varepsilon > 0 DTIME(S(n)) = DTIME((1+\varepsilon) \cdot n)$

(Beliebig nahe an 1-Linearität, aber nie vollständig $T(n)=n$)

Beweis: Zusammenfassen von Regeln

Satz Anzahl der Bänder hat keinen Einfluß auf Bandkomplexität

Beweis: Bei Reduktion von k -bändiger auf 1-bändige TM keine Änderung des Platzbedarfes

Satz $L \in DTIME(T(n)) \Rightarrow L = L(M)$ für einbändige TM, die $T(n)^2$ -zeitbeschränkt

Beweis: Erste TM nach n Schritten max. n Felder beschrieben. Auf einbändiger TM max. für jeden Schritt Köpfe zusammensuchen, also alle n Felder durchsuchen. $\Rightarrow T(n)^2$

Bemerkung NTM

Diese Sätze gelten analog auch für NTM

Bemerkung Es gibt beliebig „schwere“ Probleme

Zu $f(n)$ „berechenbare“ Fkt. gibt es rekursive Sprachen, die nicht von $f(n)$ -Zeit- bzw.

Bandbeschränkter TM erkannt werden können.
Beweis über Diagonalisierungsargument.

6.3 Die Groß-O-Notation

Definition **O(f), o(f), Θ (f)**

$f: \mathbf{N} \rightarrow \mathbf{R}$.

$O(f) := \{ g: \mathbf{N} \rightarrow \mathbf{R} \mid \exists c > 0 \exists n_0 \in \mathbf{N} \forall n \geq n_0: 0 \leq g(n) \leq c \cdot f(n) \}$ g höchstens so schnell wie f

$o(f) := \{ g: \mathbf{N} \rightarrow \mathbf{R} \mid \forall c > 0 \exists n_0 \in \mathbf{N} \forall n \geq n_0: 0 \leq g(n) < c \cdot f(n) \}$ g langsamer als f

$\Theta(f) := \{ g \mid \exists c_1, c_2 > 0 \exists n_0 \in \mathbf{N} \forall n \geq n_0: 0 \leq c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n) \}$ g und f gleich schnell

Bemerkung **Alternative Definition**

$g \in O(f) \Leftrightarrow \exists c > 0$ mit $\lim |g(x)/f(x)| = c$

$g \in o(f) \Leftrightarrow \lim |g(x)/f(x)| = 0$

Satz **Abgeschlossenheit der O-Klassen**

$\Phi \in \{O, o, \Theta\}$, $h \in \Phi(f)$, $g(n) \geq 0$

$h + g \in \Phi(f+g)$, $h \cdot g \in \Phi(f \cdot g)$, $\forall k > 0: k+h, k \cdot h \in \Phi(f)$

6.4 Hartnäckige Probleme

Bemerkung **Effizienz**

Bisher: Welche Probleme sind lösbar bzw. nicht lösbar?

Jetzt: Welche lösbaren Probleme sind effizient lösbar?

Idee: Kann ein Algorithmus Problem effizient lösen, so auch verwandte Probleme.

Definition **Reduktion, Turing-Berechenbarkeit, Turing-Reduzierbarkeit**

S_1, S_2 Sprachen, $L_1 \subseteq S_1$, $L_2 \subseteq S_2$.

Reduktion von L_2 auf L_1 : $f: S_2 \rightarrow S_1$ mit $w \in L_2 \Leftrightarrow f(w) \in L_1$

Reduktion **Turing-Berechenbar**, wenn DTM zu $w \in S_2$ die Ausgabe $y = f(w)$ liefert.

L_2 (**Turing-**) **Reduzierbar** auf L_1 , wenn es (Turing-) Reduktion von L_2 auf L_1 gibt.

Satz **Zeitbeschränkung bei Turing-Reduzierung**

L_2 turing-reduzierbar auf L_1 durch f , $T_2(n)$ -zeitbeschränkte TM, und L_1 wird durch $T_1(n)$ -zeitbeschränkte TM erkannt, dann kann L_2 durch $(T_2(n) + T_1(T_2(n)))$ -zeitbeschränkte TM erkannt werden.

Beweis: $|w|=n \Rightarrow f(w)$ wird in $T_2(n)$ Schritten berechnet $\Rightarrow |f(w)| \leq T_2(n) \Rightarrow T_1(T_2(n))$

Bemerkungen ... **Effizienz**

Übergang Zweibändige TM \rightarrow Einbändige TM \Rightarrow Quadratischer Zeitvergrößerung

\Rightarrow Effizienz \neq Linearer Zeitaufwand, da Maschinenunabhängigkeit gefordert

\Rightarrow Effizienz $\hat{=}$ Polynomieller Zeitaufwand

Definition **P-Klassen**

$P := \bigcup_{i \geq 1} \text{DTIME}(n^i)$

$NP := \bigcup_{i \geq 1} \text{NTIME}(n^i)$

$PSPACE := \bigcup_{i \geq 1} \text{DSpace}(n^i)$

$NPSpace := \bigcup_{i \geq 1} \text{NSpace}(n^i)$

Satz **$P \subseteq NP \subseteq NPSpace = PSPACE$**

Beweis: $P \subseteq NP$ klar, da DTM auch als NTM aufgefaßt werden kann

$NP \subseteq NPSpace$: TM $T(n)$ Zeitbeschränkt \Rightarrow Kann nicht mehr als $T(n)$ Felder beschreiben

$PSPACE = NPSpace$: Zu zeigen: $NSpace(n^i) = DSpace(n^{2i})$

6.5 NP-Vollständigkeit

Definition L_2 polynomiell auf L_1 reduzierbar

\Leftrightarrow Es gibt Turing-Reduktion durch polynomiell zeitbeschränkte TM, die L_2 auf L_1 reduziert

Schreibweise: $L_2 \leq_p L_1$

Bedeutung: L_2 nicht wesentlich schwieriger als L_1

Satz Polynomielle Reduzierbarkeit

1. $L_1 \in P, L_2 \leq_p L_1 \Rightarrow L_2 \in P$

2. $L_3 \leq_p L_2, L_2 \leq_p L_1 \Rightarrow L_3 \leq_p L_1$

Definition NP-hart, NP-vollständig

Sprache L **NP-hart**, falls für alle $L' \in NP$: $L' \leq_p L$

Sprache L **NP-vollständig**, falls L NP-hart und $L \in NP$

Lemma $P=NP$?

Gibt es NP-vollständige Sprache, die in P liegt, so ist $P=NP$.

Gibt es NP-vollständige Sprache, die in $NP \setminus P$ liegt, ist $P \neq NP$

Satz SAT ist NP-Vollständig

SAT = Erfüllbarkeit boolescher Ausdrücke in KNF

Beweisidee: 1) $SAT \in NP$ (N.-det. Belegung erzeugen, prüfen)

2) $\forall L \in NP \Rightarrow L \leq_p SAT$: Verhalten einer polynomiell-zeitbeschränkter NTM durch logische Formeln beschreibbar; diese in polynomieller Zeit konstruierbar und von polynomieller Länge.

Bemerkung Zeigen, daß $L \in NP$

Einfach, wenn bereits $L' \in NP$ bekannt.

1) $L \in NP$ 2) $L' \leq_p L$

Beispiele NP-vollständige Probleme

TSP (Traveling Salesman Problem)

Partitionierung

Definition Co-NP

$L \in \text{Co-NP} \Leftrightarrow \bar{L} \in NP$

Bemerkung P unter Komplementbildung abgeschlossen

Bemerkung Unbeantwortete Fragestellungen

1) $P=NP$ oder $P \neq NP$

2) $NP=\text{Co-NP}$ oder $NP \neq \text{Co-NP}$

3) $NP=\text{PSPACE}$ oder $NP \neq \text{PSPACE}$

Korollar Problem NP-vollständig \Rightarrow Problem (heute) nicht effizient lösbar

7 Berechenbarkeit

Bemerkungen ... „Was sind alle Leistungen, die ein universeller Computer erbringen kann?“

S. C. Kleene: Jede Programmleistung einer Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$, d.h. Daten werden als natürliche Zahlen codiert. „*partiell-rekursive Funktionen*“

Alan Turing: Auf TM lassen sich Funktionen $f: \mathbb{N}^k \rightarrow \mathbb{N}$ programmieren. **Turing-berechenbar**

Alonzo Church: Definition des λ -Kalküls \Rightarrow λ -berechenbare Funktionen (LISP, SCHEME)

Satz Alle drei Beschreibungen äquivalent

Satz CHURCH'sche These

Alle Berechenbarkeitsbegriffe, die auf universellen Maschinenmodellen beruhen, sind

identisch.

Unbeweisbar, fehlt die präzise Definition von „universelle Maschine“; heute aber allgemein anerkannt.

Definition $F^{(k)}$

$F^{(k)} := \{ f: \mathbb{N}^k \rightarrow \mathbb{N} \text{ partielle Funktionen} \}$

$F := \bigcup_{k \in \mathbb{N}_0} F^{(k)}$

Definition $G := \text{Menge der Grundfunktionen}$

- 1) Nullkonstante $\text{null} \in F^{(0)}$ mit $\text{null}() = 0$, $\text{null} \in G$
- 2) Nullfunktion $\text{zero} \in F^{(1)}$ mit $\text{zero}(x) = 0$, $\text{zero} \in G$
- 3) Zählfunktion $\text{succ} \in F^{(1)}$ mit $\text{succ}(x) = x+1$, $\text{succ} \in G$
- 4) Projektionsfunktion $\text{proj}_j^k \in F^{(k)}$ mit $\text{proj}_j^k(x_1, \dots, x_k) = x_j$, x_1, \dots, x_k , $j, k \in \mathbb{N}$, $1 \leq j \leq k$, $\text{proj}_j^k \in G$

Dies sind alle Funktionen aus G .

Definition **Konstruktionsmechanismen**

- 1) **Komposition:** $f \in F^{(n)}$, $f_1, \dots, f_n \in F^{(m)}$. $\Rightarrow g(x_1, \dots, x_m) = f(f_1(x_1, \dots, x_m), \dots, f_n(x_1, \dots, x_m))$.
Schreibweise: $g := S_n(f, f_1, \dots, f_n)$.
- 2) **Primitive Rekursion:** $g \in F^{(n)}$, $h \in F^{(n+1)}$. $\Rightarrow f \in F^{(n+1)}$: $f(x_1, \dots, x_n, 0) = g(x_1, \dots, x_n)$,
 $f(x_1, \dots, x_n, y+1) = h(x_1, \dots, x_n, y, f(x_1, \dots, x_n, y))$. Symbolisch: $f := R(g, h)$
- 3) **Minimalisierung:** $f \in F^{(n+1)}$. $\mu(f) \in F^{(n)}$ mit $\mu(f)(x_1, \dots, x_n) := \min_{y \in \mathbb{N}} \{ y \mid f(x_1, \dots, x_n, y) = 0 \}$
Wenn f auf totale Fkt. eingeschränkt, so **Minimalisierung im Normalfall**

Definition **Primitiv rekursive Funktion**

- 1) Die Grundfunktionen sind primitiv rekursiv.
- 2) Die durch Komposition aus prim. rek. Fkt. gewonnenen Fkt. sind prim. rek. Fkt.
- 3) Die durch prim. Rek. aus prim. rek. Fkt. gewonnenen Fkt. sind prim. rek. Fkt.
- 4) Ist f prim. rek. Fkt., so aufgrund 1), 2), 3)

Klasse aller prim. rek. Fkt.: F_{prim}

F_{prim} Abschluß von G bzgl. Komposition und primitiver Rekursion.

Definition **Allgemein rekursive Funktion**

- 1) Die Grundfunktionen sind allgemein rekursiv.
- 2) Die durch Komposition aus allg. rek. Fkt. gewonnenen Fkt. sind allg. rek. Fkt.
- 3) Die durch prim. Rek. aus allg. rek. Fkt. gewonnenen Fkt. sind allg. rek. Fkt.
- 4) Die durch Minimalisierung im Normalfall aus allg. rek. Fkt. gew. Fkt. sind allg. rek.
- 5) Ist f allg. rek. Fkt., so aufgrund 1), 2), 3), 4)

Klasse aller allg. rek. Fkt.: F_{allg}

F_{allg} Abschluß von G bzgl. Komposition, prim. Rek. und Minimalisierung im Normalfall.

Definition **Partiell rekursive Funktion**

- 1) Die Grundfunktionen sind partiell rekursiv.
- 2) Die durch Komposition aus allg. rek. Fkt. gewonnenen Fkt. sind allg. rek. Fkt.
- 3) Die durch prim. Rek. aus allg. rek. Fkt. gewonnenen Fkt. sind allg. rek. Fkt.
- 4) Die durch Minimalisierung aus allg. rek. Fkt. gew. Fkt. sind allg. rek.
- 5) Ist f allg. rek. Fkt., so aufgrund 1), 2), 3), 4)

Klasse aller part. rek. Fkt.: F_{part}

F_{part} Abschluß von G bzgl. Komposition, prim. Rek. und Minimalisierung.

Bemerkungen ... **Rekursion in Bezug auf Programme**

Primitiv rekursiv: Programme, die nach abschätzbarer Zeit halten

Allgemein rekursiv: Programm, welches für alle Eingaben hält

Partiell rekursiv: Beliebige Programm

Satz $F_{\text{prim}} \subset F_{\text{allg}} \subset F_{\text{part}}$ jeweils aber \neq
Beweis: Induktion, direkt aus Def.

Definition $F_{\text{tot}} :=$ Klasse aller totalen Funktionen aus F

Satz $F_{\text{allg}} \subseteq F_{\text{tot}}$

Bemerkung **Berechenbare Funktionen abzählbar**

- Schreibe als Text, ordne lexikographisch
- Numeriere Grundfunktionen und Kompositionen, codiere durch endliche Nummernfolge

Bemerkung **Überabzählbar viele Funktionen $f: \mathbb{N}^k \rightarrow \mathbb{N}$**

Mindestens soviele wie $f: \mathbb{N} \rightarrow \{0, 1, \dots, 9\}$.

Definiere für $r \in (0, 1)$: $f_r(i) = d_i$, wenn r dargestellt als $r = 0.d_1d_2\dots d_i\dots$

$(0, 1)$ überabzählbar, Beweis über Diagonalverfahren. Indirekt: Ann. $r_i \in (0, 1)$ abzählbar.

In Tabelle r_i gegen Stellen. Bilde r mit $r = 0.d_1d_2\dots$, $d_k \neq d_{kk} \Rightarrow$ Wid.

Beispiel **Primitiv rekursive Funktionen**

pred, -, +, ·, exp, !

Beispiel **Allgemein rekursive, nicht primitiv rekursive Funktion**

ACKERMANN-Funktion $A(0, y) = y + 1$, $A(x + 1, 0) = A(x, 1)$, $A(x + 1, y + 1) = A(x, A(x + 1, y))$

Berechnungsschema: Tabelle.

7.1 Der Universal Calculator

Bemerkung **Der UC**

- Universelle Maschine
 - Schrittweise sequentiell
 - Programme in prozeduraler Sprache formulierbar
- Schema: Datenspeicher, Programmspeicher: Zuordnung $x_i \rightarrow n$ auf endlich vielen Plätzen
Befehlsregister, Prozessor

Definition **Instruktionen des UC**

Verwendetes (unendliches) Alphabet

{do, if, then, goto, else, halt, ←, =, (,), 0, S, P, x_1, x_2, \dots , start, l_1, l_2, \dots }

- 1) do $x_i \leftarrow 0$ then goto l_p $x_i := 0, l_p$ nächste Befehlsadr.
- 2) do $x_i \leftarrow x_j$ then goto l_p $x_i := x_j$
- 3) do $x_i \leftarrow S(x_i)$ then goto l_p x_i++
- 4) do $x_i \leftarrow P(x_i)$ then goto l_p x_i-- , $P(0) = 0$ **Modifizierte Differenz**
- 5) goto l_p
- 6) if $x_i = 0$ then goto l_p else goto l_q
- 7) halt

Instr. 1) und 7) immer ausführbar, 3), 4) und 6) nur, falls x_i und 2), falls x_j Wert besitzt

Definition **Programm des UC**

$L = \{\text{start}, l_1, \dots\}$, I die Menge der Instruktionen

$P \subseteq L \times I$ endliche Menge heißt **Programm des UC**, falls

- 1) P enthält genau ein Paar $\langle \text{start}, i \rangle$, $i \in I$
- 2) Enthält $P \langle l, i \rangle$, so daß in i die Folge „goto l_p “ vorkommt, so enthält P genau ein $\langle l_p, i' \rangle$

Definition **Programmvariablen eines Programmes**

Enthält P das Zeichen x_i , so ist x_i Programmvariable von P .

Definition Ablauf eines Programmes mit Programmvariablen x_1, \dots, x_m

Folge s_0, s_1, \dots (endlich oder unendlich) mit

- 1) $s_i = \langle l_i, f_i \rangle, l_i \in L, f_i: \{x_1, \dots, x_m\} \rightarrow N$ partielle Funktion
- 2) $l_0 = \text{start}$
- 3) $\langle \text{start}, \text{goto } l_1 \rangle \in P$ (Nur Konvention!)
- 4) Ist $\langle l_i, h \rangle \in P$, so ist h im Zustand s_i ausführbar
- 5) Ist $\langle l_i, h \rangle \in P, h \neq \text{halt}$, so ist l_{i+1} der Inhalt des Befehlsregisters und f_{i+1} die Variablenbelegung nach Ausführung von h
- 6) Ist $\langle l_k, \text{halt} \rangle \in P$, so ist die Folge endlich und s_k deren letztes Element

Definition UC-Berechenbarkeit

Partielle Funktion $f: N^k \rightarrow N$ UC-berechenbar, wenn es Programm P des UC gibt mit

- 1) P besitzt Variablen $x_1, \dots, x_m, m \geq n$ und eine ausgezeichnete Variable $x_r, r=n$ oder $r=n+1 \leq m$
- 2) Ist $f(a_1, \dots, a_n)$ definiert, so terminiert die Berechnungsfolge zu P mit $f_0 = (x_1 \rightarrow a_1, x_2 \rightarrow a_2, \dots, x_n \rightarrow a_n)$ so, daß deren letztes Element $s_k = \langle l_k, f_k \rangle$ ist und $f_k(x_r) = f(a_1, \dots, a_n)$ gilt.

Satz Alle berechenbaren Funktionen (partiell rek. Fkt.) sind UC-berechenbar

z.z.: Grundfunktionen berechenbar. Bew.: Explizit UC-Programm angeben
 z.z.: UC-berechenbare Funktionen unter Komposition, primitiver Rekursion und Minimalisierung abgeschlossen.
 Bew.: In Textform ansatzweise Konstruktionsvorschriften: Eingabevariablen kopieren, Labels neu nummerieren, Ablaufdiagramm für Minimalisierung.

Definition Universelles UC-Programm ω

- Gibt Codierung durch natürliche Zahlen für alle Programme π des UC, bezeichnet mit π' , für alle Eingaben (a_1, \dots, a_n) , bezeichnet durch $(a_1, \dots, a_n)'$ und alle Resultate a , bezeichnet a' .
- ω hat zwei Eingabevariablen
- Mit Eingabe von π' und $(a_1, \dots, a_n)'$ errechnet ω auf Ausgabevariablen $(f_\pi(a_1, \dots, a_n))'$
- Ist $f_\pi(a_1, \dots, a_n)$ nicht definiert, so hält ω nicht.

Satz Universelles UC-Programm ω

Spezielle Codierung (**Gödelisierung**) der Konstrukte des UC.

- | | |
|--|------------------------------------|
| 5) <u>goto</u> l_p | 2^p |
| 1) <u>do</u> $x_i \leftarrow 0$ <u>then goto</u> l_p | $2^p \cdot 3^{k+1} \cdot 5$ |
| 3) <u>do</u> $x_i \leftarrow S(x_i)$ <u>then goto</u> l_p | $2^p \cdot 3^{k+1} \cdot 7$ |
| 4) <u>do</u> $x_i \leftarrow P(x_i)$ <u>then goto</u> l_p | $2^p \cdot 3^{k+1} \cdot 11$ |
| 2) <u>do</u> $x_i \leftarrow x_j$ <u>then goto</u> l_p | $2^p \cdot 3^{k+1} \cdot 13^{j+1}$ |
| 6) <u>if</u> $x_i = 0$ <u>then goto</u> l_p <u>else goto</u> l_q | $17^{k+1} \cdot 19^p \cdot 23^q$ |
| 7) <u>halt</u> | 29 |

Codierung des Programms π start: goto p ; 1: Instr1; 2: Instr2; ...

$$\pi' = 2^{q_0} \cdot 3^{q_1} \cdot 5^{q_2} \cdot \dots \cdot p_{n+1} q^n$$

Variablen von ω :

x_1 : π' . x_2 : $(a_1, \dots, a_n)'$. x_3 : Aktuelles Label des simulierten Programmes. x_4 : Gödelnummer der aktuellen Instruktion. x_5, \dots Hilfsvariablen
 Programmablauf: klar. Operation auf x_2 !