Grundlagen der Programmierung

Studiengang „Informatik", „Angewandte Informatik" SS'05

Prof. Dr. Madlener

Universität Kaiserslautern

Vorlesung:
Di 08.15-09.45 42/115 Fr 08.15-09.45 42/115

- Informationen
  www-madlener.informatik.uni-kl.de/ag-madlener/teaching/ss2005/gdp/gdp.html.
- Grundlage der Vorlesung: Buch
  Sperschneider/Hammer: Theoretische Informatik. Eine problemorientierte Einführung (ZBIB:LINF31), Springer Verlag.
- Bewertungsverfahren:
  Übungen: max. 10 Bonuspunkte (5 bis Aufsichtsarbeit 5 danach)
  Aufsichtsarbeit: max. 30 Punkte,
  Abschlussklausur: max. 70 Punkte.
- Aufsichtsarbeit:Sa .06 (12 - 15) Ort: Mensa
- Erste Abschlussklausur: 1.08.05 Beachte Anmeldeschluss
- Übungstermine: Siehe Vorlesungsseite
- Listen Übungen: Anmeldung auf Vorlesungsseite ab 12:00 heute

Inhaltsverzeichnis

1 Einleitung . . . 1
2 Mengen, Relationen, Funktionen . . . 6
3 Kalküle . . . 10
4 Semantik von Programmiersprachen . . . 24
4.1 Datenstrukturen/Algebren . . . 25
4.2 Sprache zur Beschreibung von Eigenschaften in Algebren 29
4.3 Bewertung und Gültigkeit von Formeln . . . 38
4.4 While - Programme . . . 45
5 Programmverifikation: Prädikatenlogik und Hoare Kalkül . . 54
6 Berechenbarkeit . . . 89
6.1 Primitiv rekursive Funktionen $\mathcal{P}(\mathbb{N})$ . . . 90
6.2 $\boldsymbol{\mu}$-Rekursive Funktionen $\boldsymbol{\mathcal{R}_p}(\mathbb{N})$ (partiell rekursive Funktionen) . . . 127
6.3 Universalität der $\boldsymbol{\mu}$-rekursiven Funktionen . . . 135
6.4 Grundzüge der Rekursionstheorie . . . 150
6.5 Die Churchsche These . . . 173
6.6 Berechenbarkeit auf Zeichenreihen Wortfunktionen . . . 190
7 Die Chomsky-Hierarchie . . . 205

7.1 Grammatiken . . . 206
7.2 Chomsky Hierarchie . . . 210
7.3 Endliche Automaten - reguläre Sprachen - Typ 3-Sprachen . . . 220
7.4 Kontextfreie Sprachen - Typ2-Sprachen . . . 242
7.5 Algorithmus von Cocke-Kasami-Younger . . . 269
7.6 Unentscheidbare Probleme für kontextfreie Grammatiken 272
8 Grundlagen der Programmierung Zusammenfassung . . . 278

1 Einleitung

Methoden zur Lösung von Problemen mit Hilfe von Rechnern

Probleme (technisch mathematisch formal)

Formalisierung ($\equiv$ Festlegung $\equiv$ "Spezifikation")

Beispiele
Optimierungsprobleme

1. Rundreise minimaler Länge (Kosten minimal).
   Formalisierung z. B. Karte, Städte, Verbindungsabstand.
   Graph: Knoten $\longleftrightarrow$ Städte Kanten (Gewicht) $\longleftrightarrow$ Verb.
   Eingabe: Menge von Knoten: V, Gewichtskanten E.
   Ausgabe: Rundreise, die alle Knoten aus V enthält mit minimalen Kosten.
   $\leadsto$ Rundreise Problem (Travelling Salesman) **TSP**
2. Anteile bestimmter Ware mit Wertigkeit bis Maximalgewicht.
   Eingabe: Objekte mit Gewicht, Profit, Maximalgewicht W.
   Ausgabe: (Anteile) Objekte die Maximalgewicht nicht überschreiten und Profit maximal sind. D. h. $\sum x_i w_i \leq W$, $\sum x_i p_i$ max.
   $\leadsto$ Rucksackproblem (Varianten $x_i \in 0/1$ RP, x_i rational RP).
   Knapsack Problem. 0/1-KP, rat-KP.

3. **Weitere Optimierungsprobleme**
   - Optimale Bandspeicherung (Zugriffszeiten minimal)
   - Verschnitt minimieren
   - Bin Packing
4. **Probleme aus der Zahlentheorie oder allgemeinem Theorembeweisen**

   Kryptologie: Große Primzahlen.

   Ist $1\,2\,3\,4\,5\,6\,7\,8\,9\,1\,2\,3\,4\,5\,6\,7\,8\,1\,2\,3\,4\,5\,6\,7$ Primzahl?

   Probleme aus der Zahlentheorie:

   z. B. „Jede natürliche Zahl lässt sich als Summe von vier Quadratzahlen schreiben“.

   Formalisiert: Sprache der Prädikatenlogik (**PL1**):

   $$\forall X \exists A \exists B \exists C \exists D \qquad X = A^2 + B^2 + C^2 + D^2$$

   „Interpretation“: $\mathbb{N}$ natürliche Zahlen, $+,^2$ wie üblich interpretiert.

   oder **Fermat´s Problem**

   $$\exists N \exists X \exists Y \exists Z \qquad (N > 2 \wedge X^N + Y^N = Z^N).$$

   Allgemeiner aus **Hilbert´s Problemliste**:

   Gesucht “algorithmische Lösung” von

   Eingabe: Satz φ aus PL1 über $\mathbb{N}$.

   Ausgabe: Ja, falls φ Theorem.
   Nein, falls φ kein Theorem.

   Angenommen es gibt ein Programm $0_{\mathbb{N}}$ dafür.

   Frage: Hält $0_{\mathbb{N}}$ für jede Eingabe φ?

   $\rightsquigarrow$ Entscheidungsprobleme, Aufzählungsprobleme.

Entscheidungsprobleme

5. Sei P Menge von Programmen in PS (z.B. JAVA).

   Problem: Für $x \in A$ Eingabe für $p \in P$.

   Frage: Hält p bei Eingabe x?

   $\rightsquigarrow$ Halteproblem für p.

   Varianten:

   Problem: Für $p \in P$.

   Frage: Hält p für alle erlaubten Eingaben $x \in A$?

   $\rightsquigarrow$ Uniformes Halteproblem für P.(“Ist p total definiert”).

6. Wortpuzzle: (Gödel, Post, ..., Hofstädter)

   Alphabet Σ, $\Sigma^* = \{$endliche Folgen von Buchstaben aus $\Sigma\}$.

   $w = a_1 a_2 \dots a_n \in \Sigma^*$, $|w| = n$, $n = 0$ erlaubt $\varepsilon \equiv$ leeres Wort.

   Eingabe: Endlich viele Wortpaare
   $(u_1, w_1), (u_2, w_2) \dots (u_n, w_n)$
   $u_i, w_i \in \Sigma^+ = \Sigma^* - \{\varepsilon\}$.

   Frage: $\exists k > 0 \; \exists n_1, \dots, n_k \; n_i \in \{1, 2, \dots, n\}$:
   $u_{n_1} u_{n_2} \dots u_{n_k} = w_{n_1} w_{n_2} \dots w_{n_k}$

   $\rightsquigarrow$ Post’sches Korrespondenzproblem PCP (Emil Post).

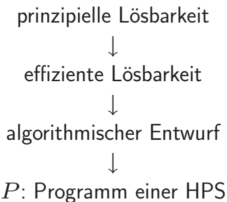
Wichtige Fragestellungen

- Was heißt es diese Probleme sind algorithmisch lösbar?
- Was nutzt es solche Probleme algorithmisch zu lösen?
- Was nutzt es „gute“ Lösungen für TSP zu haben?
- Wie schwierig sind diese Probleme?
- Wie hängen sie zusammen?
- Wie erkennt und beweist man, dass es keine („gute“) algorithmische Lösungen geben kann!

Nötig: Konzepte, Fakten, Methoden, Theoreme, Theorien

$\rightsquigarrow$ **Berechnungsmodelle/Programmiersprachen**.

Algorithmische Unlösbarkeit?



Problem: Syntaktische und semantische Verifikation von P.

Schwerpunkte

- **Syntaxanalyse**

  Formale Sprachen: Chomski-Hierarchie
  Context freie Sprachen
  Grammatiken/Erzeugungsprozess
  Automaten (endliche, keller)/Erkennungsprozess
- **Berechenbarkeitsmodelle**

  Semantik von Programmiersprachen: WHILE-Programme
  Maschinenmodelle: Turing- und Registermaschinen
  Rekursive und Partiel-Rekursive Funktionen
- **Programmverifikation**

  Tut P auch was erwartet wird.
  Beweisverpflichtungen: Zusicherungen, Invarianten, Terminierungsbedingungen
  Beweise: Siehe Logik Vorlesung

2 Mengen, Relationen, Funktionen

- A endliche Menge, $|A|$ Anzahl der Elemente, **Kardinalität**.
- A, B Mengen, $|A| = |B|$ gdw
  es gibt eine Bijektion $h : A \to B$ (injektiv+surjektiv).
  $|\{0, \ldots, n-1\}| := n$ $\quad A$ **endlich** gdw $\exists n : |A| = n$.
- A heißt **abzählbar**, falls A endlich ist oder $|A| = |\mathbb{N}|$, d. h. es gibt eine Abzählungsfunktion f für A.
  f ist Bijektion von
  a) $f : \{0, 1, \ldots, |A| - 1\} \to A$ $\qquad A$ endlich.
  b) $f : \mathbb{N} \to A$ $\qquad A$ unendlich.

2.1 Satz Satz von Cantor: Es gibt nicht abzählbare Mengen.

z. B. $\mathbb{R}$.

Diagonalisierungstechnik:

$A = \{f : \mathbb{N} \to \mathbb{N} \mid dom(f) = \mathbb{N}\}$.

Angenommen A ist abzählbar. Da A nicht endlich ist, gibt es eine Bijektion von $\mathbb{N}$ auf A. Sei diese Abzählung $f_0, f_1, f_2, \ldots$

Definiere $\bar{f}(x) := f_x(x) + 1$ für $x \in \mathbb{N}$.

Offenbar $\bar{f} \notin \{f_i \mid i \in \mathbb{N}\} = A$, da $\bar{f}(x) \neq f_x(x)$, aber $\bar{f}$ ist ganz auf $\mathbb{N}$ definiert, d. h. $\bar{f} \in A$ ↯

Beachte: jede Teilmenge einer abzählbaren Menge ist abzählbar.

Relationen und Funktionen

Kartesisches Produkt von Mengen $A \times B$. Tupelschreibweise: $\vec{a} = (a_1, \ldots, a_n) \in A_1 \times \cdots \times A_n$

- **Relationen**: $R \subseteq A_1 \times \cdots \times A_n$
  $(a_1, \ldots, a_n) \in R$ schreibe $Ra_1 \ldots a_n$ oder $R(a_1, \ldots, a_n)$
  $n = 2$ Infix Notation $a_1 R a_2$ (z.B. $a \leq b$)
- **Funktionen als Relationen**:
  $f : A \to B$: **(A,B, graph(f))**, wobei $graph(f) \subseteq A \times B$, für jedes $a \in A$ gibt es höchstens ein $b \in B$ mit $(a, b) \in graph(f)$.
  Schreibe $f(a) = b$ für $(a, b) \in graph(f)$ und für
  ein b mit $(a, b) \in graph(f)$: $\mathbf{f(a)} \downarrow$ ($f(a)$ ist definiert),
  kein b mit $(a, b) \in graph(f)$: $\mathbf{f(a)} \uparrow$ ($f(a)$ nicht definiert).
- Sind $(A, B, graph(f)), (B, C, graph(g))$ Funktionen.
  $g \circ f$ (**Verkettung oder Komposition**) $g \circ f : A \to C$
  $(A, C, \{(a, g(f(a)))\})$.
- **Definitionsbereich**:
  $f : A \to B : \ \mathbf{dom(f) = \{a \in A : f(a) \downarrow\}}$.
- **Wertebereich**:
  $f : A \to B : \ \mathbf{im(f) = \{f(a) \in B : a \in A, f(a) \downarrow\}}$.
- $f : A \to B$ gilt $dom(f) = A$ so ist $f : A \to B$ **total**.
- $f : A \to A$ Funktion: Die **Iteration** $f^n : A \to A$. $\{(a, b) \in A^2 : f(f(\cdots(f(a)\cdots)) \downarrow, b = f(f \cdots f(a) \cdots))\}$
  Für n = 0 ist $f^n = id$, d.h. die Identitätsfunktion auf A.

Funktionen (Forts.), Alphabete, Sprachen

- **Charakteristische Funktion von** $R \subseteq A$:
  $\chi_R : A \to \{0, 1\}$ totale Funktion mit ($\chi_R(a) = 1$ gdw $a \in R$).
- **Alphabet** ist eine abzählbare Menge $\mathbf{\Sigma}$ von Buchstaben.
- $\mathbf{\Sigma}^*$ Menge der endlichen Folgen von Buchstaben (**Wörter**).
- $a \in \Sigma^*$ $a = a_1 \cdots a_n$ $\quad n \geq 0$ $\quad n$ **Länge** von a,
  $|a| = n$ $\quad n = 0$ $\quad \varepsilon$ leeres Wort.
- $a = a_1 \cdots a_n, b = b_1 \cdots b_m$ **Konkatenation**
  $ab = a_1 \cdots a_n b_1 \cdots b_m$ mit $|ab| = n + m$
- Präfix, Suffix, Teilwort.
- **Wortfunktionen**: $f : \Sigma^* \to \Sigma^*$,
  z. B. Spiegelung: $a = a_1 \cdots a_n \to a^{\mathrm{mi}} = a_n \cdots a_1$.
- **Sprachen** sind Teilmengen von Σ^*.
  $L, M \subseteq \Sigma^* : L \cup M$ Vereinigung.
  $LM = \{uv : u \in L, v \in M\}$ Konkatenation.
  $L^* = \{v_1 \cdots v_n : n \in \mathbb{N}, v_i \in L, i = 1, \ldots, n\}$
  $= \bigcup_{n \geq 0} L^n$ $\qquad$ Iteration.
- **Beachte**: Σ^* ist abzählbar.
  Programme sind Wörter (Zeichenreihen) über Alphabet ⇝ Menge der Programme einer PS ist stets abzählbar.
  Also gibt es stets Funktionen, die nicht von Programmen berechnet werden können.

Versuch einer Definition von algorithmisch lösbar

Informelle Präzisierung für $f : A \to B$ „berechenbar".

2.2 Definition

a) Ein effektiver Prozess zur Lösung eines Problems (einer Klasse von Problemstellungen) hat folgende Eigenschaften:
   1. Der Prozess besitzt eine endliche Beschreibung.
   2. Er besteht aus Einzelschritten, die mechanisch in endlicher Zeit ausführbar sind, d. h. jeder solche Schritt kann z. B. nur von endlich vielen Daten abhängen.
   3. Er ist deterministisch, d. h. der jeweils nächste Schritt ist immer eindeutig bestimmt (falls er überhaupt existiert) kein Raten oder auswählen.
   4. Lässt die Problemstellung eine Antwort zu, so liefert der Prozess die korrekte Antwort und terminiert nach Ausführung endlich viele Einzelschritte.
      Lässt die Problemstellung keine Antwort zu, so liefert der Prozess die Antwort „?" oder er terminiert nicht.

b) Ein **Algorithmus** ist eine endliche Beschreibung eines effektiven Prozesses (Repräsentation).

c) Eine Funktion $f : A \to B$ heißt **effektiv berechenbar**, falls es einen effektiven Prozess gibt, der für $x \in A$ (Instanz der Problemstellung).
   1. Stoppt mit Antwort $f(x)$, falls $x \in dom(f)$.
   2. Stoppt nicht, falls $x \notin dom(f)(f(x) \uparrow)$.

3 Kalküle

Erzeugung (Generierung) von Mengen, Relationen, Funktionen

Kalkül besteht aus **(Axiome, Regeln)**

Erzeugung syntaktischer Objekte:

Sprachen, Formeln, Terme, ..., Bilder, Graphen (Wörter über Alphabet Σ)

3.1 Definition

Regel: Vorschrift um Objekt κ zu erzeugen (Konklusion der Regel) sofern Objekte $\Pi_1, \dots, \Pi_n$ $(n \geq 0)$ (Prämissen) bereits vorhanden.

Schreibweise: $R :: \dfrac{\Pi_1, \dots, \Pi_n}{\kappa}$.

(d. h. z. B.: $R \in ((\Sigma^*)^n \times \Sigma^*)$ für ein $n \in \mathbb{N}$).

$n = 0$: Regel ohne Prämissen ist **Axiom**.

(Axiome erlauben die Erzeugung ihrer Konklusion ohne Voraussetzungen. Initialisierung des Generierungsprozesses).

$n > 0$: **Echte Regel**.

Kalkül ist Menge von Regeln

$$K \subseteq \bigcup_{n \geq 0} (A^n \times A)$$

A Objektmenge (z.B. Σ^*, Menge von Bildern etc.)

Ableitungen

3.2 Beispiel

1. $\Sigma = \{a_1, \dots, a_n\}$ Regelmenge: $\overline{\varepsilon}$, $\dfrac{u}{ua_1}, \dots, \dfrac{u}{ua_n}$ $u \in \Sigma^*$ (unendlich viele Regeln - Regelschema -, u als Wortvariable aufgefasst).
2. mu-Kalkül: für alle Wörter $X, Y \in \{i, u, m\}^*$
   Regeln:
   $$\left\{ \frac{Xi}{Xiu}, \frac{mY}{mYY}, \frac{XiiiY}{XuY}, \frac{XuuY}{XY} \right\}$$

Frage: kann man aus mi das Wort mu ableiten?

3.3 Definition

a) Sei K ein Kalkül. Eine **Ableitung** in K ist eine Folge $(\varphi_1, \varphi_2, \dots, \varphi_n)$ von Objekten, so dass für alle $i = 1, \dots, n$ φ_i die Konklusion einer Regel von K ist, deren Prämissen alle in $\{\varphi_1, \dots, \varphi_{i-1}\}$ enthalten sind.

b) Ein Objekt φ ist in K **ableitbar**, falls es eine Ableitung in K mit letztem Objekt φ gibt.
Schreibweise: $\underset{K}{\vdash} \varphi$.

c) Ein Objekt φ ist in K **aus einer Menge** M von Objekten ableitbar, falls es eine Ableitung in $K(M)$ mit letztem Objekt φ gibt, wobei $K(M)$ die Erweiterung von K um die Axiome $\overline{\kappa}$ $(\kappa \in M)$ ist. Schreibweise: $M \underset{K}{\vdash} \varphi$.

Beispiel 3.2 (Fort.)

a) Ableitbar sind $\varepsilon, a_1, a_2, \dots, a_n, \dots a_i a_j, \dots$, d. h. alle Wörter aus Σ^*.

b) Ableitbar aus mi sind z. B.

mi	mi	mi
Regelt. 1	2	2
miu	mii	mii
Regelt. 2	1	2
$miuiu$	$miiu$	$miiii$
Regelt. 2	2	3
$miuiuiuiu$	$miiuiiu$	mui
⋮	⋮	1
		$muiu$

...

$\{mi, miu, miuiu, \dots, mii, mui, \dots\}$

Frage: Liegt mu in dieser Menge?

Beachte: Es können stets mehrere Regeln anwendbar sein.

Darstellung von Ableitungen

Ableitungen können als Bäume dargestellt werden.
Blätter: Prämissen, Wurzel: Konklusion.

$\Pi_1 \quad \cdots \quad \Pi_n$

Regel:

K

Ableitung $(\varphi_1, \dots, \varphi_n)$: Blätter: Konklusionen von Axiomen (Annahmen).
Innere Knoten: Regel.
Wurzel: φ_n.

Ableitungsbäume $\rightsquigarrow$ Fragen: Tiefe, Eindeutigkeit usw.

3.4 Lemma Kompaktheitssatz für Kalküle

Sei $M \subset A$. Dann gilt:
$M \underset{K}{\vdash} \varphi$ genau dann wenn es eine endliche Teilmenge $F \subset M$ gibt mit $F \underset{K}{\vdash} \varphi$.

Diese Tatsache ist die Grundlage für die vielfältige Verwendung von Kalkülen für die Fundierung vieler Begriffe und Methoden der Informatik.

Kalküle definieren Hüllenoperatoren

Allgemeiner Hüllenoperator für Teilmengen einer Menge bildet Teilmengen in Teilmengen ab, z.B. Transitive Hülle, Folgerungshülle usw.

$\Gamma_K : \mathcal{P}(A) \to \mathcal{P}(A)$ (Teilmengen von A werden in Teilmengen abgebildet).

Mit $\Gamma_K(M) := \{\varphi \in A : M \underset{K}{\vdash} \varphi\}$ für $M \subset A$.

Wichtige Eigenschaften für Hüllenoperatoren:
Einbettung, Monotonie, Abgeschlossenheit.

Es gilt für Γ_K:

Einbettung: für alle $M : M \subseteq \Gamma_K(M)$.

Monotonie: $M \subseteq M'$ so $\Gamma_K(M) \subseteq \Gamma_K(M')$
($M \underset{K}{\vdash} \varphi$ so auch $M' \underset{K}{\vdash} \varphi$).

Abgeschlossenheit: $\Gamma_K(\Gamma_K(M)) = \Gamma_K(M)$.

Der Ableitbarkeitsbegriff ist **transitiv**: aus $M \underset{K}{\vdash} \varphi$ und $M \cup \{\varphi\} \underset{K}{\vdash} \psi$ folgt $M \underset{K}{\vdash} \psi$.

(Die Verwendung eines ableitbaren Objekts als Voraussetzung (Blatt) in einer Ableitung kann stets eliminiert werden, d.h. Blatt wird ersetzt durch Ableitungsbaum mit entsprechender Wurzel).

Schrittweise Konstruktion von $\Gamma_K(.)$

„Konstruktive“ Sicht der Menge der ableitbaren Objekte in K:
$K \subseteq A^* \times A$ $(:= \bigcup_{n\geq 0} A^n \times A)$.

$\Gamma_K(B) := \bigcup_{i\geq 0} B_i$ mit $B_0 = B$, $B_{i+1} := B_i \cup \Gamma_K^1(B_i)$, wobei

$\Gamma_K^1(B_i) := \{\varphi \in A \mid$ es gibt $n \geq 0, \Pi_1, \ldots, \Pi_n \in B_i,$
$((\Pi_1, \ldots, \Pi_n), \varphi) \in K\}$

„Einschritt-Ableitungen aus B_i“

i ist Maß für die „Tiefe“ des Ableitungsbaums für $\varphi \in B_i$.

Spezialfall: $A = \Sigma^*$. Zeichenreihen.

Wortersetzungssysteme (Semi-Thue-Systeme 1914).

3.5 Definition

Ein **Wortersetzungssystem** ist ein Paar (Σ, Π) mit einem endlichen Alphabet Σ und einer endlichen Menge Π von Produktionen über Σ. Eine Produktion über Σ ist eine Zeichenreihe der Form

$$l ::= r$$

(oft auch $l \to r$ „Regel“) mit $l \neq \varepsilon, l, r \in \Sigma^*$.

Der durch (Σ, Π) definierte Kalkül $K(\Sigma, \Pi)$ auf Σ^* besteht aus allen Regeln

$$\frac{ulv}{urv} \quad l ::= r \in \Pi, u, v \in \Sigma^*$$

Wortersetzungssysteme (Fort.)

Beachte ∞-viele Regeln, endlich viele Regelschemata.

Ableitbarkeit im Wortersetzungssystem (Σ, Π):

$$x \underset{\Pi}{\vdash} y \;\; \text{gdw} \;\; \{x\} \underset{K(\Sigma,\Pi)}{\vdash} y$$

Äquivalente Darstellung: Ableitbarkeit in Schritten $\overset{n}{\underset{\Pi}{\vdash}}$.

$x \overset{1}{\underset{\Pi}{\vdash}} y$ gdw es gibt $l ::= r \in \Pi, u, v \in \Sigma^*$ mit
$x = ulv$ und $y = urv$

$x \overset{n}{\underset{\Pi}{\vdash}} y$ gdw es gibt $z_0, \ldots, z_n \in \Sigma^*$ mit

$$x = z_0,\ z_i \overset{1}{\underset{\Pi}{\vdash}} z_{i+1}\ (i < n),\ z_n = y.$$

$n = 0$ liefert $x \overset{0}{\underset{\Pi}{\vdash}} y$ gdw $x = y$.

3.6 Lemma

$x \underset{\Pi}{\vdash} y$ gdw es gibt $n \in \mathbb{N}$ mit $x \overset{n}{\underset{\Pi}{\vdash}} y$

Beispiele

3.7 Beispiel

1. Wortersetzungssysteme $\quad \Sigma = \{a, b\}$
   $$aba ::= baab$$
   Kalkül-Regel: $\dfrac{uabav}{ubaabv}$

   Dann ist
   $aba \overset{1}{\underset{\Pi}{\vdash}} baab$ nur endlich viele Wörter ableitbar aus aba und

   $a\underline{aba} \overset{1}{\underset{\Pi}{\vdash}} \underline{aba}ab \overset{1}{\underset{\Pi}{\vdash}} ba\underline{aba}b \overset{1}{\underset{\Pi}{\vdash}} b\underline{aba}abb$
   $\overset{1}{\underset{\Pi}{\vdash}} bba\underline{aba}bb \overset{1}{\underset{\Pi}{\vdash}} \cdots$
   unendlich viele Wörter ableitbar.

   $ababa \overset{1}{\underset{\Pi}{\vdash}} baabba$
   $ababa \overset{1}{\underset{\Pi}{\vdash}} abbaab$

2. Grammatiken als Wortersetzungssysteme

leftside::=rightside		(EBNF) Schreibweise für Produktionen
Statement Expression:		$A ::= B$
Assignment	$\rightsquigarrow$	$A ::= C$
MethodInvocation		$A ::= D$
ClassInstanceCreationExpression		$\vdots$
$\vdots$		aus A ableitbar

Beispiele (Forts.)

3. $\Pi :: S \to \varepsilon,\ S \to aSbb$
   Ableitbare „Sprache“ $L = \{w \in \{a,b\}^* \mid S \underset{\Pi}{\vdash} w\}$

$$\begin{array}{ccccccc} S & \overset{1}{\underset{\Pi}{\vdash}} & aSbb & \overset{1}{\underset{\Pi}{\vdash}} & aaSbbbb \vdash \cdots & \overset{1}{\underset{\Pi}{\vdash}} & a^nSb^{2n} \\ \top & & \top & & \top & & \top \\ \varepsilon & & abb & & a^2b^4 & & a^nb^{2n} \end{array}$$

   Offenbar gilt: $L = \{a^nb^{2n} : n \in \mathbb{N}\}$.

   Wie zeigt man dies? **Induktionsbeweise**.

Nachweis von Eigenschaften ableitbarer Objekte

Induktionsprinzipien

Erinnerung: Induktionsprinzip in den natürlichen Zahlen:
Sei A eine Aussage über natürliche Zahlen: $\mathbb{N}$

A soll für alle $n \in \mathbb{N}$ richtig sein:
Methode:

Zeige: A trifft für 0 zu: **Induktionsanfang**.

Unter der Annahme, dass A für n gilt,

Zeige: A trifft auch für $n+1$ zu: **Induktionsschritt**.

Analog für Σ^*: Anfang: ε
Schritt: $|u| = n \rightsquigarrow |w| = n+1$

3.8 Satz Strukturelle Induktion (Induktion über Aufbau)

Sei A Menge, $K \subseteq \bigcup_{n>0}(A^n \times A)$ Kalkül, $B \subseteq A$.

Um eine Eigenschaft P für alle aus B in K ableitbaren Elemente (d.h. aus $\Gamma_K(B)$) zu beweisen genügt es folgendes nachzuweisen:

a) Alle Elemente in B haben die Eigenschaft P.
b) Haben $\Pi_1, \ldots, \Pi_n \in A$ die Eigenschaft P und ist $((\Pi_1, \ldots, \Pi_n), \varphi) \in K$, dann hat auch φ die Eigenschaft P.

Nachweis von Eigenschaften ableitbarer Objekte (Forts.)

Wichtige Spezialfälle:

- $B = \varnothing$, dann entfällt a).
- B endlich a) muss für endliche viele geprüft werden.

Beweismethode entspricht auch der Induktion nach Ableitungslänge.

3.9 Beispiel mu-Kalkül.

Behauptung: Angenommen $mi \underset{\Pi}{\vdash} w \rightsquigarrow 3 \nmid |w|_i$

(3 teilt nicht die i-Länge von w (Anzahl der i-s in w)).

a) Überprüfe die Behauptung für Wort mi : $|mi|_i = 1$.
b) Regel:

$$\frac{Xi}{Xiu}, \frac{mY}{mYY}, \frac{XiiiY}{XuY}, \frac{XuuY}{XY}$$

   $|Xi|_i = |Xiu|_i, 2|mY|_i = |mYY|_i,$

   $|XuY|_i = |XiiiY|_i - 3, |XuuY|_i = |XY|_i$

   Ist für die Prämisse die Behauptung richtig, so auch für die Konklusion.
   Wegen $3 \mid |mu|_i = 0$ kann mu nicht aus mi abgeleitet werden.

Erzeugung von Funktionen: Rekursion

3.10 Beispiel **1**: $S : \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ $\quad (\mathbb{N}^+ = \mathbb{N} - \{0\})$.

$$S(x,y) = \begin{cases} x & \text{falls } x = y \\ S(x-y,y) & \text{falls } x > y \\ S(x,y-x) & \text{falls } y > x \end{cases}$$

Welche „Funktion“ soll von einer solchen „Gleichung“ definiert werden!
Semantik

Offenbar sollte z. B. $S(5,5) = 5$ und wohl $S(10,5) = S(5,5) = 5 \ldots$

Frage ist S total? Ist S “effektiv berechenbar”?

Prinzip: Initiale Werte Axiome.

Definierte Funktionswerte führen zu neu definierten Werten.

Beispiel 2: $t : \mathbb{N} \to \mathbb{N}$ mit

$$t(n) = \begin{cases} 1 & \text{falls } n = 1 \\ t(n/2) & \text{falls } n \text{ gerade} \\ t(3n+1) & \text{falls } n \text{ ungerade} \end{cases}$$

$$\begin{aligned} t(15) &= t(46) = t(23) = t(70) = t(35) = t(106) \\ &= t(53) = t(160) = t(80) = t(20) = t(10) \\ &= t(5) = t(16) = t(8) = t(4) = t(2) = t(1) = 1 \end{aligned}$$

Gibt es totale Funktionen, die die Gleichung erfüllen?

Erzeugung von Funktionen: Rekursion (Forts.)

Wie beim Hüllenoperator ist die von einer rekursiven Gleichung definierten Funktion als die „kleinste“ Funktion, die die Gleichung erfüllt gemeint. Dabei ist für Funktionen $f, g : A \to B \qquad f \sqsubseteq g$

„f kleiner als g“ durch
$dom(f) \subseteq dom(g)$ und für alle $x \in dom(f)$ gilt $f(x) = g(x)$.

D. h. sucht man die Lösung, muss man unter den Lösungen der Gleichungen die „kleinste“, d.h. die am wenigsten definierte, bestimmen.

Beispiel 3: Gleichung für f sei $f(z) = \begin{cases} 0 & z = 0 \\ f(z-2) + \frac{1}{2}z & z \text{ gerade} \end{cases}$

Wobei $f : \mathbb{Z} \to \mathbb{Z}$.

Fange mit undefinierten Funktionen an $f_0 = \varnothing \subset \mathbb{Z} \times \mathbb{Z}$.

Setze: $f_{i+1}(z) = \begin{cases} 0 & z = 0 \\ f_i(z-2) + \frac{1}{2}z & z \text{ gerade} \neq 0 \end{cases}$

$$f_1(z) = \begin{cases} 0 & z = 0 \\ \uparrow & \text{sonst} \end{cases}$$

$$f_2(z) = \begin{cases} 0 & z = 0 \\ 1 & z = 2 \\ \uparrow & \text{sonst} \end{cases}$$

Erzeugung von Funktionen: Rekursion (Forts.)

$$f_3(z) = \begin{cases} 0 & z = 0 \\ 1 & z = 2 \\ 3 & z = 4 \\ \uparrow & \text{sonst} \end{cases} \qquad f_4(z) = \begin{cases} 0 & z = 0 \\ 1 & z = 2 \\ 3 & z = 4 \\ 6 & z = 6 \\ \uparrow & \text{sonst} \end{cases}$$

Dann ist $f_i \sqsubseteq f_{i+1}$ und $f = \bigcup_{i \geq 0} f_i$ die gesuchte Funktion.

$f_i \sqsubseteq f_{i+1}$: Induktion nach i: $i = 0$ einfach.

Induktionsschritt: Sei $i > 0$ und $z \in dom(f_i)$. Ist $z = 0$ so Beh. klar. Also ist $0 \neq z$ gerade und $f_i(z) = f_{i-1}(z-2) + \frac{1}{2}z = f_i(z-2) + \frac{1}{2}z = f_{i+1}(z)$ nach Def. von f_i, Ind. Annahme und Def. von f_{i+1}.

$dom(f) = 2\mathbb{N}$ und f erfüllt die Rekursionsgleichung und ist die kleinste (bzgl. $\sqsubseteq$) Funktion die diese Gleichung erfüllt. **Beweis!**.

Die Funktion $h(z) = \begin{cases} \sum_{i=o}^{z/2} i & z \geq 0 \text{ gerade} \\ \uparrow & \text{sonst} \end{cases}$
erfüllt die Gleichung mit selben Definitionsbereich, d.h. $f = h$.

4 Semantik von Programmiersprachen

Spezifizieren - Implementieren - Verifizieren

- Datenstrukturen sind Algebren: Signatur + Interpretation.
- Programmierbarkeit über einer Algebra: A.
- Programme $\in$ Programmiersprache: α.
- Semantik: Denotational, Operational.
- Partielle Korrektheit: $A \models \{\varphi\}\alpha\{\psi\}$.
- Kalkül von Hoare.

Benötigt wird:

- Sprachen für **Signaturen** (Funktionen, Prädikate, Stelligkeit).
- **Algebren** zur Signatur (Interpretationen).
- Sprache zur Beschreibung von **Eigenschaften**.
  Prädikatenlogik: $\neg, \wedge, \vee, \to, \leftrightarrow, \forall, \exists, =$
- Sprache zur Beschreibung (Bezeichnung) der **Programme**.
  while , **do** , **end**, **if** , **then** , **else** ...
- **Zustände** zur Beschreibung der Wirkung der Programme.

4.1 Datenstrukturen/Algebren

4.1 Definition

Eine **Signatur** ist ein Paar (S, Σ) mit einer endlichen Menge S von **Sortensymbolen** (Typbezeichnern) und einer endlichen Menge Σ von **(Operationssymbol -) Deklarationen** der Form

$c :\to s \qquad f : s_1 \times \cdots \times s_n \to s \qquad p : s_1 \times \cdots \times s_n$

Mit $n > 0$, $s, s_1, \ldots, s_n \in S$.

- c heißt **Konstantensymbol**.
- f heißt **Funktionssymbol** der **Stelligkeit** n.
- p heißt **Relationssymbol** (Prädikatssymbol) der **Stelligkeit** n.

Keines der Zeichen c, f, p darf in verschiedenen Deklarationen vorkommen (kein „overloading“).

Eine **Variablenmenge** V über der Signatur (S, Σ) besteht aus **Variablendeklarationen** $X : s$ mit einem **Variablenbezeichner** X und einer Sorte $s \in S$. Kein Variablenbezeichner darf in zwei verschiedenen Variablendeklarationen vorkommen.

$X : s \in V \qquad X$ Variable vom Typ s.

Interpretationen von Signaturen (S, Σ): Algebren

4.2 Definition Algebren (Relationalstrukturen)
Eine (S, Σ)-**Algebra** A ist eine Abbildung, die jeder Sorte $s \in S$ eine nichtleere Menge s_A, jedem Konstantensymbol $c :\rightarrow s$ in Σ eine Konstante $c_A \in s_A$, jedem Funktionssymbol $f : s_1 \times \cdots \times s_n \rightarrow s$ in Σ eine totale Funktion $f_A : s_{1A} \times \cdots \times s_{nA} \rightarrow s_A$ und jedem Relationssymbol $p : s_1 \times \cdots \times s_n$ in Σ eine Relation $p_A \subseteq s_{1A} \times \cdots \times s_{nA}$ zuordnet.

s_A heißt **Grundbereich** der Sorte s der Algebra A $(s \in S)$.

Mehrsortige Algebren. Schreibe

$$A = (s_A(s \in S), c_A(c \in \Sigma), f_A(f \in \Sigma), p_A(p \in \Sigma))$$

4.3 Definition Zustände - Belegung von Variablen
Sei V eine endliche Variablenmenge über (S, Σ) und A eine (S, Σ)-Algebra. Ein **Zustand** z über A und V ist eine Funktion z, die jeder Variablen X mit $X : s$ in V einen Wert $z(X)$ in der Menge s_A zuordnet. $\mathcal{Z}(\mathbf{A}, \mathbf{V})$ sei die Menge der Zustände über A und V.

Bezeichnungen: $z : V \rightarrow A$ (genauer in $\bigcup_{s \in S} s_A$).

Für $X : s$ in V, $a \in s_A$ sei **z(X/a)** der Zustand über A und V der X den Wert a und allen $Y \neq X$ den Wert $z(Y)$ zuordnet.

Entsprechend ist $z(X_1/a_1, \ldots, X_n/a_n)$ (kurz $z(\vec{X}/\vec{a})$) auf paarweise verschiedenen Variablen $X_1, \ldots, X_n$ definiert.

Beispiele: Algebren

4.4 Beispiel

- N::
  Signatur $(\{nat\}, \{0 :\rightarrow nat, succ : nat \rightarrow nat\})$
  Interpretation:
  $nat \rightarrow nat_N = \mathbb{N}$
  $0 \quad 0_N = 0 \in \mathbb{N}$
  $succ \quad succ_N(x) = x + 1$
- Nat::
  Signaturerweiterung der Signatur von N um
  $+ : nat \times nat \rightarrow nat$
  $* : nat \times nat \rightarrow nat$
  $< : nat \times nat$
  $+_{Nat}(x, y) = x + y \; (+ \; in \; \mathbb{N})$
  $*_{Nat}(x, y) = x * y \; (* \; in \; \mathbb{N})$
  $<_{Nat}(x, y)$ gdw $x < y \; (< \; in \; \mathbb{N})$
- $Boolean$::
  Signatur
  $(\{b\}, \{$true, false $:\rightarrow b$, not $: b \rightarrow b$, and, or $: b \times b \rightarrow b\})$
  Standard-Interpretation: z. B.
  $b_{Boolean} = \{W, F\}$
  true$_{Boolean} = W$ und false$_{Boolean} = F$
  or$_{Boolean}(F, F) = F$
  or$_{Boolean}(x, y) = W$ für $(x, y) \neq (F, F)$
  ...

Beispiele: Algebren (Forts.)

- $set_d(A)$::
  Für A eine (S, Σ)-Algebra $d \in S$.
  Intention: Sorte, die endliche Teilmengen der Menge d_A beschreibt:
  Signaturerweiterung von (S, Σ) um Sorte set_d und Funktionssymbole $\emptyset :\rightarrow set_d$ und $insert : set_d \times d \rightarrow set_d$
  Interpretation A erweitert um
  $(set_d)_{set_d(A)} :=$ alle endlichen Teilmengen von d_A
  $\emptyset_{set_d(A)} :=$ leere Menge
  $insert_{set_d(A)}(M, a) := M \cup \{a\}$ für M endlich und $a \in d_A$.
- $set_{nat}(Nat)$::
  Variablenmenge V: $X, Y : nat$ und $X_1, X_2 : set_{nat}$ dann sind z_1 und z_2 mit
  $z_1(X) = 0, \quad z_1(Y) = 3, \quad z_1(X_1) = \emptyset,$
  $z_1(X_2) = \{0, 1, 3\}$ und
  $z_2(X) = 1, \quad z_2(Y) = 0, \quad z_2(X_1) = \{0, 1, 3\},$
  $z_2(X_2) = \emptyset$ Zustände.

  $z_1(X/1, Y/0, X_1/\{0, 1, 3\}, X_2/\emptyset) = z_2$

4.2 Sprache zur Beschreibung von Eigenschaften in Algebren

Prädikatenlogik (erster Stufe) als Spezifikationssprache.

Terme als Bezeichner für Objekte einer (S, Σ)-Algebra.

4.5 Definition $\mathbf{Term(S, \Sigma, V)}$

Terme über einer Signatur und Variablenmenge V mit jeweiligem Typ sind induktiv wie folgt definiert:

- Ist $X : s$ eine Variable in V, so ist X ein Term vom Typ s.
- Ist $c :\rightarrow s$ in Σ, so ist c ein Term vom Typ s.
- Ist $f : s_1 \times \cdots \times s_n \rightarrow s$ in Σ, t_i ein Term über (S, Σ), V vom Typ s_i für $i = 1, \ldots, n$, so ist $f(t_1, \ldots, t_n)$ ein Term über (S, Σ) und V vom Typ s.

Termkalkül für $Term(S, \Sigma, V)$ Menge der Terme über (S, Σ) und V und Definition von $Typ : Term(S, \Sigma, V) \rightarrow S$

$\overline{X} \; (X : s \in V) \qquad Typ(X) = s$

$\overline{c} \; (c :\rightarrow s \in \Sigma) \qquad Typ(c) = s$

$\dfrac{t_1, \ldots, t_n}{f(t_1, \ldots, t_n)} \qquad (Typ(t_i) = s_i, f : s_1 \times \cdots \times s_n \rightarrow s \in \Sigma).$

$Typ(f(t_1, \ldots, t_n)) = s.$

Beispiele

4.6 Beispiel Beachte: Der Termkalkül ist eindeutig, d.h. jeder Term wird eindeutig aus den Teiltermen aufgebaut. Somit ist auch Typ eine wohldefinierte Funktion auf $Term$.

a) Signatur von N Variablenmenge:: V $X : nat$

$$0 \quad X \quad succ^n(0) \quad succ^n(X) \quad (n \in \mathbb{N})$$

b) Signatur von Nat $X, Y : nat$

Terme sind:

$(X + 0), (X * Y), (X + succ(Y)), succ((X + Y))$

eigentlich

$+(X, 0), *(X, Y), +(X, succ(Y)), succ(+(X, Y))$

Beachte Infixnotation und Klammerungsregelung. Wird für die Operationssymbole Infix - Notation gewählt, so sind äußere Klammern zu verwenden um die Eindeutigkeit der Zerlegung eines Terms in Teiltermen sicherzustellen. Zur besseren Lesbarkeit werden oft äußere Klammern unterdrückt und Prioritäten (Bindungsstärken) zwischen den Operationen vereinbart.

$succX + 0 * Y$ steht für $+(succ(X), *(0, Y))$.

Priorität $succ, *, +$.

Keine Terme sind:

$X+, *succ(0), \ldots$

Die Sprache der Prädikatenlogik - Formeln

4.7 Definition $\mathbf{Form(S, \Sigma, V)}$

(S, Σ) Signatur, V Variablenmenge über (S, Σ).

Boolesche Formeln über $(\mathbf{S}, \mathbf{\Sigma}), \mathbf{V}$ sind definiert durch:

- $p(t_1, \ldots, t_n)$ ist **atomare Boolesche-Formel** für $p : s_1 \times \cdots \times s_n \in \Sigma$, t_i Term vom Typ s_i $(i = 1, \ldots, n)$.
- $t_1 = t_2$ ist **Gleichung** für t_1, t_2 Terme über (S, Σ) und V vom gleichen Typ.
- Sind φ und ψ Boolesche-Formeln über (S, Σ), V, so auch die Folgenden:
  $\neg\varphi$ -nicht φ-, $(\varphi \to \psi)$ -φ impliziert ψ-,
  $(\varphi \wedge \psi)$ -φ und ψ-, $(\varphi \vee \psi)$ -φ oder ψ-
  $(\varphi \leftrightarrow \psi)$ -φ äquivalent ψ-

Prädikatenlogische Formeln über $(\mathbf{S}, \mathbf{\Sigma}), \mathbf{V}$

- Eine Boolesche-Formel ist eine PL-Formel.
- Ist φ Formel, so auch $\neg\varphi$.
  Sind φ, ψ Formeln, so auch $(\varphi * \psi)$ $* \in \{\wedge, \vee, \to, \leftrightarrow\}$.
- φ Formel, $X : s \in V$, so auch $\forall X \varphi$, $\exists X \varphi$.
  φ ist der Wirkungsbereich vom Quantor $\forall X$ bzw. $\exists X$.

Ein Vorkommen einer Variablen X in einer Formel heißt **frei**, sofern es nicht im Wirkungsbereich eines Quantors $\forall X$ oder $\exists X$ auftritt. Andernfalls heißt das Vorkommen **gebunden**.

Eindeutiger Kalkül zur Erzeugung der Formeln $\mathbf{Form(S, \Sigma, V)}$

- $\frac{}{p(t_1, \ldots, t_n)}$ $\quad (Typ(t_i) = s_i, p : s_1 \times \cdots \times s_n \in \Sigma)$
- $\frac{}{t_1 = t_2}$ $\quad (Typ(t_1) = Typ(t_2))$
- $\frac{\varphi}{\neg\varphi}$, $\frac{\varphi, \psi}{(\varphi * \psi)}$ $\quad (* \in \{\wedge, \vee, \to, \leftrightarrow\})$
- $\frac{\varphi}{\forall X \varphi}$ -für alle-, $\frac{\varphi}{\exists X \varphi}$ -es gibt- für X Variablenbezeichner.

4.8 Beispiel Nat

1. $\exists Y succ(Y) = X$
2. $(X + succ(Y)) = succ(X + Y)$
3. $\forall X \forall Y (X + succ(Y)) = succ(X + Y)$ („Gleichungsaxiom")
4. $(\exists Y succ(Y) = X \wedge (X < succ(X) \wedge Y = 0))$

In

1. Y kommt nur gebunden vor, X nur frei.
2. Alle Vorkommen von X und Y sind frei.
3. Alle Vorkommen von X und Y sind gebunden.(„Abgeschlossen")
4. Alle Vorkommen von X sind frei.
   Erstes Vorkommen von Y ist gebunden.
   Zweites Vorkommen von Y ist frei.

Abgeschlossene Formeln - Substitution

4.9 Definition Eine Formel heißt **abgeschlossen**, falls sie keine freien Vorkommen von Variablen enthält.

Im Beispiel 4.8: 3. ist abgeschlossen (auch **Satz** oder **Sentence**).

4.10 Definition Substitution

Eine **Substitution** σ über $(S, \Sigma), V$ ist eine typtreue Abbildung der Menge der Variablenbezeichner in $Term(S, \Sigma, V)$, die nur an endlich vielen Stellen von der Identität verschieden ist.

Sie kann somit durch die Menge $\{X_1/s_1, \ldots, X_m/s_m\}$ ($\{\vec{X}/\vec{s}\}$ als Vektor) beschrieben werden: Hierbei sind

- $X_1, \ldots, X_m$ Variablenbezeichner, paarweise verschieden.
- $s_1, \ldots, s_m$ sind Σ-Terme über V.
- X_i und s_i sind verschieden und vom selben Typ.

Fortsetzung von σ auf $Term(S, \Sigma, V)$:

$t\sigma$ für $t \in$ Term(S, Σ, V) ist definiert als:

- $X_i\sigma = s_i \quad 1 \leq i \leq m$
- $Y\sigma = Y \quad Y \in V \setminus \{X_1, \ldots, X_m\}$
- $c\sigma = c$
- $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$

Substitution (Forts.)

4.11 Lemma
σ ist wohldefiniert und total auf $Term(S, \Sigma, V)$.

Beweis: einfache strukturelle Induktion.

4.12 Beispiel

and$(X, Y)\{X/Y, Y/$true$\}$ = and$(Y,$ true$)$ $\quad X \neq Y$

$(X + succ(Y))\{X/succ(0), Y/succ(X)\} = (succ(0) + succ(succ(X)))$

$succ(X)\{X/succ(X + Y)\} = succ(succ(X + Y))$

Beachte:

1. $t\sigma$ hängt nur von den Werten der in t vorkommenden Variablen ab.
2. σ ist Homomorphismus der "Termalgebra" $Term(S, \Sigma, V)$ in sich selbst.
3. Enthält t keine Variablen (d.h. t ist Grundterm), so $t\sigma = t$ für jede Substitution σ.

Bewertung von Termen

Frage: Welche Werte haben Terme im Zustand z?

4.13 Definition Werte von Terme im Zustand z.
Sei (S, Σ) eine Signatur, V eine Variablenmenge über (S, Σ), A eine (S, Σ)-Algebra, z Zustand über A und V.

Für $t \in Term(S, \Sigma, V)$ sei der Wert von t in Algebra A und Zustand z, kurz $val_{A,z}(t)$, induktiv wie folgt definiert:

- $val_{A,z}(X) = z(X)$ für $X \in V$
- $val_{A,z}(c) = c_A$ für $c \in \Sigma$
- $val_{A,z}(f(t_1, \ldots, t_n)) = f_A(val_{A,z}(t_1), \ldots, val_{A,z}(t_n))$

4.14 Beispiel

$z = (X/5, Y/3)$, d. h. $z(X) = 5$, $z(Y) = 3$

$$\begin{aligned} val_{Nat,z}(X + succ(Y)) &= 5 +_{Nat} val_{Nat,z}(succ(Y)) \\ &= 5 +_{Nat} (3 +_{Nat} 1) = 9 \end{aligned}$$

$$\begin{aligned} val_{N,z}(succ^5(0)) &= val_{N,z}(succ^4(0)) + 1 \\ &= val_{N,z}(succ^3(0)) + 1 + 1 = \cdots \\ &= 5 \end{aligned}$$

Bewertung von Termen - Zustände und Substitutionen

4.15 Lemma

a) $val_{A,z}$ ist wohldefiniert und $val_{A,z}(t) \in Typ(t)_A$.
b) Ist z' ein weiterer Zustand über A und V mit $z(X) = z'(X)$ für alle in t vorkommenden Variablen X, so ist $val_{A,z}(t) = val_{A,z'}(t)$.

D. h. der Wert von t hängt nur von den Werten der in t vorkommenden Variablen ab. Enthält der Term t keine Variablen (Grundterm), so hängt der Wert nicht vom Zustand z ab.
Beweis: Eindeutigkeit des Termkalküls und rekursive Definition von val mithilfe der Werte der Teilterme.

4.16 Lemma Substitutionslemma für Terme
A eine (S, Σ)-Algebra, V Variablenmenge, z Zustand über A, V.
$X \in V$, $r, t \in Term(S, \Sigma, V)$.
Sei $a = val_{A,z}(r)$. Dann gilt

$$val_{A,z}(t\{X/r\}) = val_{A,z(X/a)}(t)$$

Eine Substitution kann also bei der Termauswertung durch eine Zustandsmodifikation simuliert werden.

Substitutionslemma für Terme (fort.)

Das Lemma lässt sich auf Simultansubstitution mehrerer Var $\vec{X}$ durch Terme $\vec{r}$ verallgemeinern.

Beweis:

Strukturelle Induktion über Aufbau der Terme (Kalkül).

$t \equiv$:

1. X:
$$\begin{aligned} val_{A,z}(t\{X/r\}) &= val_{A,z}(r) = a (= val_{A,z}(r)) \\ &= val_{A,z(X/a)}(X) = val_{A,z(X/a)}(t) \end{aligned}$$
2. Y von X verschieden:
$$\begin{aligned} val_{A,z}(t\{X/r\}) &= val_{A,z}(Y) \\ &= z(Y) = z(X/a)(Y) \\ &= val_{A,z(X/a)}(Y) = val_{A,z(X/a)}(t) \end{aligned}$$
3. c:
$$val_{A,z}(t\{X/r\}) = c_A = val_{A,z(X/a)}(t)$$
4. $f(t_1, \ldots, t_n)$:
$$\begin{aligned} val_{A,z}(t\{X/r\}) &= val_{A,z}(f(t_1\{X/r\}, \ldots, t_n\{X/r\})) \\ &= f_A(val_{A,z}(t_1\{X/r\}), \ldots, \\ &\qquad val_{A,z}(t_n\{X/r\})) \\ &= f_A(val_{A,z(X/a)}(t_1), \ldots, val_{A,z(X/a)}(t_n)) \\ &= val_{A,z(X/a)}(f(t_1, \ldots, t_n)) \\ &= val_{A,z(X/a)}(t) \end{aligned}$$

4.3 Bewertung und Gültigkeit von Formeln

4.17 Definition Gültigkeit im Zustand

A (S, Σ)-Algebra, V Variablenmenge, z Zustand über A, V.

Sei $\xi \in Form(S, \Sigma, V)$ Formel und X mit $Typ(X) = s$.

ξ gilt in der Algebra A im Zustand z: $A \models_z \xi$:
(Schreibe $A \not\models_z \xi$ für $A \models_z \xi$ gilt nicht).

Wird induktiv definiert durch

$A \models_z p(t_1, \ldots, t_n)$ gdw $(val_{A,z}(t_1), \ldots, val_{A,z}(t_n)) \in p_A$
$(A \not\models_z p(t_1, \ldots, t_n)$ gdw $(val_{A,z}(t_1), \ldots, val_{A,z}(t_n)) \notin p_A)$
$A \models_z t_1 = t_2$ gdw $val_{A,z}(t_1) = val_{A,z}(t_2)$
$A \models_z \neg\varphi$ gdw $A \not\models_z \varphi$
$A \models_z (\varphi \wedge \psi)$ gdw $A \models_z \varphi$ und $A \models_z \psi$
$A \models_z (\varphi \vee \psi)$ gdw $A \models_z \varphi$ oder $A \models_z \psi$
$A \models_z (\varphi \rightarrow \psi)$ gdw $A \not\models_z \varphi$ oder $A \models_z \psi$
$A \models_z (\varphi \leftrightarrow \psi)$ gdw $(A \models_z \varphi$ und $A \models_z \psi)$ oder $(A \not\models_z \varphi$ und $A \not\models_z \psi)$
$A \models_z \exists X\varphi$ gdw $A \models_{z(X/a)} \varphi$ für ein $a \in s_A$
$A \models_z \forall X\varphi$ gdw $A \models_{z(X/a)} \varphi$ für alle $a \in s_A$

Beachte: Für jede Formel ξ gilt entweder $A \models_z \xi$ oder $A \not\models_z \xi$.

Insbesondere entweder $A \models_z \xi$ oder $A \models_z \neg\xi$.

4.18 Definition Gültigkeit
$A \models \varphi$ (φ gilt in A oder φ ist gültig in A) gdw $A \models_z \varphi$ für alle Zustände z über A, V.

Beispiele

4.19 Beispiel In **Boolean:**

$A \models \text{and}(X, Y) = \text{and}(Y, X)$
$\models \text{or}(X, \text{not}(X)) = \text{true}$
$\models \text{and}(X, \text{not}(X)) = \text{false}$
$\models \text{not}(\text{and}(\text{not}(X), \text{not}(Y))) = \text{or}(X, Y)$

In jeder Struktur $\mathbf{A}$, beliebige Formeln φ, ψ über Signatur von A.

$A \models (\varphi \vee \neg\varphi)$
$\models \varphi \leftrightarrow \neg\neg\varphi$
$\models (\varphi \vee \psi) \leftrightarrow (\neg\varphi \rightarrow \psi)$
$\models (\varphi \wedge \psi) \leftrightarrow \neg(\varphi \rightarrow \neg\psi)$

Gilt für Formeln φ, ψ: $A \models \varphi \leftrightarrow \psi$, so heißen sie **logisch äquivalent in** A.

Eigenschaft: Jede Formel φ lässt sich effektiv in eine logisch äquivalente Formel ψ, die nur die Operationen $\neg$, $\rightarrow$ enthält, transformieren.

In $\mathbf{N}$:

$$N \models \forall Y \exists X \quad X = succ(Y)$$

Frage: Gilt auch

$$N \overset{?}{\models} \forall X \exists Y \quad X = succ(Y) \text{ oder}$$
$$N \overset{?}{\models} \exists X \forall Y \quad X = succ(Y)?$$

Beispiele (Forts.)

Behauptung: Nein, dafür
finde Zustand z mit

$$N \not\models_z \exists Y \quad X = succ(Y) \quad z.B.\ z(X) = 0$$

bzw.
für alle Zustände z gilt $N \not\models_z \forall Y\ X = succ(Y)$.

Sei $z(X)$ beliebig aber fest, dann liefert $z(Y) := z(X)$ ein „Gegenbeispiel".

Beachte:

$A \models (\forall X\varphi \leftrightarrow \neg\exists X\neg\varphi)$
$A \models (\exists X\varphi \leftrightarrow \neg\forall X\neg\varphi)$
$N \models (\exists X\ X = succ(Y) \leftrightarrow \exists Z\ Z = succ(Y))$

Anwendung von Substitutionen auf Formeln

$N \models \forall Y \exists X \ \ X = succ(Y)$

Die Ersetzung von Y durch einen beliebigen Term sollte eine Formel liefern, die in N gilt.

Vorsicht:

$\{Y/0\} :: \ N \models_z \exists X \ \ X = succ(0)$ (bel. z)

$\{Y/X\} :: N \not\models_z \exists X \ \ X = succ(X)$ (bel. z)

Anwendung von Substitutionen auf Formeln

Problem: Im Wirkungsbereich des Quantors $\exists X$ wird ein Term substituiert der X enthält, d. h. freies Vorkommen von Y wird gebundenes Vorkommen von X.

Lösung: Umbenennung gebundener Variablen.

4.20 Definition

Sei $\sigma = \{X_1/s_1, \ldots, X_m/s_m\}$ eine Substitution.
Induktiv über die Struktur der Formel φ sei $[\varphi]\sigma$ wie folgt definiert:

$[p(t_1, \ldots, t_n)]\sigma = p(t_1\sigma, \ldots, t_n\sigma)$
$[t_1 = t_2]\sigma = t_1\sigma = t_2\sigma$
$[\neg\varphi]\sigma = \neg[\varphi]\sigma$
$[(\varphi * \psi)]\sigma = ([\varphi]\sigma * [\psi]\sigma),\ * \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$
$[QX\varphi]\sigma = QY[[\varphi]\{X/Y\}]\sigma, \quad Q \in \{\forall, \exists\}$

Wobei Y eine „frische" Variable ist d. h. Y kommt **nicht** in QX, φ, σ vor.

Dabei kommt eine Variable in Substitution σ vor, falls sie in $\{X_1, \ldots, X_m\}$ oder $\{s_1, \ldots, s_m\}$ vorkommt.

4.21 Beispiel

In N: $[\exists X\ X = succ(Y)]\{Y/X\}$ Z „neu"
$= \exists Z[Z = succ(Y)]\{Y/X\}$
$= \exists Z\ Z = succ(X)$

Anwendung von Substitutionen auf Formeln (Forts.)

In Nat: Sei $\sigma : \{X/(X+Y), Y/(Y+Z), Z/0\}$

$\varphi ::\quad \forall X \forall Y (X + succ(Y)) = succ(X+Y)$

$[\varphi]\sigma =$

$\varphi' ::\quad \forall Y (X + succ(Y)) = succ(X+Y)$

$[\varphi']\sigma =$

$\varphi'' ::\quad (X + succ(Y)) = succ(X+Y)$

$[\varphi'']\sigma =$

Substitutionslemma für Formeln

4.22 Lemma

Sei A eine (S,Σ)-Algebra, φ eine Formel, $\vec{X}$ Variablen, $\vec{t}$ Terme vom selben Typ und z ein Zustand, der auf allen freien Variablen von $[\varphi]\{\vec{X}/\vec{t}\}$ definiert ist. Es sei $\vec{a} = val_{A,z}(\vec{t})$.

Dann ist $z(\vec{X}/\vec{a})$ auf allen freien Variablen von φ definiert und es gilt

$$A \models_z [\varphi]\{\vec{X}/\vec{t}\} \text{ gdw } A \models_{z(\vec{X}/\vec{a})} \varphi$$

Beweis: Induktion über Aufbau von φ

(Beachte z ist o.B.d.A. auf allen Variablen der t_i definiert und somit ist $z(\vec{X}/\vec{a})$ auf allen Variablen der t_i und der Variablen in $\vec{X}$ definiert.)

Fall $\varphi = \forall Z \psi$, Y sei eine Variable, die in ψ, Z und $\{\vec{X}/\vec{t}\}$ nicht vorkommt. Dann

$$A \models_z [\forall Z \psi]\{\vec{X}/\vec{t}\} \text{ gdw } A \models_z \forall Y[[\psi]\{Z/Y\}]\{\vec{X}/\vec{t}\}$$

$\text{gdw } A \models_{z(Y/b)} [[\psi]\{Z/Y\}]\{\vec{X}/\vec{t}\}$ für alle b in Typ$(Y)_A$

$\overset{(IV)}{\text{gdw}}\ A \models_{z(Y/b)(\vec{X}/\vec{a})} [\psi]\{Z/Y\}$ für alle b

$\overset{(IV)}{\text{gdw}}\ A \models_{z(Y/b)(\vec{X}/\vec{a})(Z/b)} \psi$ für alle b

$\text{gdw } A \models_{z(\vec{X}/\vec{a})(Z/b)} \psi$ für alle b

$\text{gdw } A \models_{z(\vec{X}/\vec{a})} \forall Z \psi$

Substitutionslemma für Formeln (Forts.)

4.23 Folgerung

Für alle Algebren A, Zustände z, Formeln φ, Variablen X und Terme t vom selben Typ gilt:

$$A \models_z (\forall X \varphi \rightarrow [\varphi]\{X/t\})$$

Also ist $(\forall X \varphi \rightarrow [\varphi]\{X/t\})$ „universell“ gültig, „allgemein gültig“.

Beachte Literatur:
Andere Definitionen und Schreibweisen üblich,
z. B. - „erlaubte Substitutionen“, - φ_t^X für $[\varphi]\{X/t\}$ oder $\varphi_{\vec{t}}^{\vec{x}}$

4.24 Lemma Koinzidenzlemma für Formeln

Seien A, φ, V gegeben. Sind z und z' Zustände über V mit $z(X) = z'(X)$ für alle freien Variablen X von φ, dann gilt

$$A \models_z \varphi \text{ gdw } A \models_{z'} \varphi$$

Die Bewertung einer Formel (ob φ im Zustand z gilt oder nicht gilt) hängt nur von den Werten der in φ **frei** vorkommenden Variablen ab.

Insbesondere gilt dies für abgeschlossene Formeln, für solche gilt entweder $A \models \varphi$ oder $A \models \neg\varphi$.

Beachte dies muss nicht für Formeln mit freien Variablen gelten.

4.4 While - Programme

Zuweisung, Verzweigung, Iteration sind wesentliche Konstrukte jeder „universellen“ Programmiersprache.

Programm:: Mittel zur Beschreibung eines effektiven Prozesses.

4.25 Definition $\mathbf{Prog(S,\Sigma,V)}$

Sei (S,Σ) eine Signatur, V endliche Variablenmenge. Die Menge der **While-Programme** über $(S,\Sigma), V$ sei durch folgenden Kalkül definiert.

Regel	Bedingung	
$\overline{\varepsilon}$	ε	**leeres Programm**
$\overline{X := t;}$	$X : s \in V, t \in$ Term(S,Σ,V), Typ$(t) = s$	**Zuweisung**
$\dfrac{\beta, \gamma}{\textbf{if } B \textbf{ then } \beta \textbf{ else } \gamma \textbf{ end};}$	B Boolesche-Formel über $(S,\Sigma), V$	**Test**
$\dfrac{\beta}{\textbf{while } B \textbf{ do } \beta \textbf{ end};}$	B Boolesche-Formel über $(S,\Sigma), V$	**Schleife**
$\dfrac{\alpha, \beta}{\alpha\beta}$	(als Konkatenation von Zeichenreihen)	**Komposition**

Eine **Anweisung** ist entweder eine Zuweisung, ein Test oder eine Schleife.

Beachte:

Jedes Programm ist entweder ε oder fängt mit einer Anweisung an.

Beispiele

Im Beispiel enthalten die Variablenmengen stets die Programmvariablen.

4.26 Beispiel While-Programm α über Signatur von Nat.

$\alpha :: Y := 0;\ Z := 0;$
  while $\neg Y = X$ **do**
    $Z := succ(Z + (Y + Y));\ Y := succ(Y);$
  end;

While-Programme β und γ über Signatur von N.

$\beta :: Z := X;\ Z' := 0;$
  while $\neg Y = Z'$ **do**
    $Z := succ(Z);\ Z' := succ(Z');$
  end;

$\gamma :: Z := 0;\ Z' := 0;$
  while $\neg Y = Z'$ **do**
    $\beta\{X/Z, Y/X, Z'/Z''\};\ Z' := succ(Z');$
  end;

Makros: Hierbei steht $\beta\{X/Z, Y/X, Z'/Z''\}$ für Programm welches durch Substitution der entsprechenden Variablen entsteht, d. h.

$Z := Z;\ Z'' := 0;$
while $\neg X = Z''$ **do**
  $Z := succ(Z);\ Z'' := succ(Z'');$
end;

Denotationale Programmsemantik

4.27 Definition

Sei A eine (S, Σ)-Algebra, V Variablenmenge, $z, z' \in \mathcal{Z}(A, V)$ (Zustände über A, V) und α ein Programm über (S, Σ) und V.

Induktiv über den Aufbau eines Programms wird definiert, was es heißt, dass der Zustand z durch Abarbeitung von α in den Zustand z' überführt wird, notiert als $z[[\alpha]]_A z'$, d. h. α bezeichnet eine Relation $[[\alpha]]_A$ auf $\mathcal{Z}(A, V)$.

- $z[[\varepsilon]]_A z'$ gdw $z = z'$.
- $z[[X := t]]_A z'$ gdw $z' = z(X/val_{A,z}(t))$
- $z[[$**if** B **then** β **else** γ **end**; $]]_A z'$ gdw
  ($A \models_z B$ und $z[[\beta]]_A z'$) oder ($A \not\models_z B$ und $z[[\gamma]]_A z'$).
- $z[[$**while** B **do** β **end**; $]]_A z'$ gdw
  es gibt eine Zahl $n \in \mathbb{N}$ und Zustände $z_0, \ldots, z_n$ mit
  - $z = z_0$,
  - $A \models_{z_i} B$ und $z_i[[\beta]]_A z_{i+1}$ für $0 \leq i < n$
  - $A \not\models_{z_n} B$
  - $z_n = z'$
- $z[[\alpha\beta]]_A z'$ gdw es gibt einen Zustand z_1 mit $z[[\alpha]]_A z_1$ und $z_1[[\beta]]_A z'$.

Beachte: Die zweistellige Relation $[[\alpha]]_A$ ist rechtseindeutig aber nicht immer rechtsvollständig, d. h. aus $z[[\alpha]]_A z'$ und $z[[\alpha]]_A z''$ folgt $z' = z''$, aber nicht zu jedem α und z muss es ein z' geben mit $z[[\alpha]]_A z'$.

Beispiele

Bei einer While-Schleife beschreibt $[[-]]_A$ den Ablauf.

$z = z_0$	$[[\beta]]_A$	z_1	$[[\beta]]_A$	$\cdots$	z_{n-1}	$[[\beta]]_A$	$z_n = z'$
B gilt	$\cdots$	gilt		$\cdots$	gilt		B gilt nicht mehr

4.28 Beispiel Im Beispiel 4.26: $z[[\alpha]]_{Nat} z'$ und sei $z(X) = 2$:

$\alpha :: z(X) = 2, z(Y) = 0, z(Z) = 0$

- $Nat \models_z \neg Y = X$
  $z_1(X) = 2, z_1(Y) = 1, z_1(Z) = 1$
- $Nat \models_{z_1} \neg Y = X$
  $z_2(X) = 2, z_2(Y) = 2, z_2(Z) = 4$
- $Nat \not\models_{z_2} \neg Y = X$
  $z' = z_2$

Allgemein gilt:

$z'(Z) = z(X)^2$
$z'(X) = z(X)$
$z'(Y) = z(X)$

α „berechnet" mit Eingabe X in Z die Funktion $f(x) = x^2$.

Analog β berechnet in Z die Funktion $f(x, y) = x + y$.

Analog γ berechnet in Z die Funktion $f(x, y) = x * y$.

D. h. $+_{nat}, *_{nat}$ sind über N „berechenbar" durch While-Programme.

Interpretersemantik

4.29 Definition Sei A eine (S, Σ)-Algebra und V eine Variablenmenge. Die Interpreterfunktion I_A ist eine zweistellige totale Funktion, die einem Programm α und Zustand z, das Programm α' und den Zustand z' zuordnet, $(d.h. I_A : (Prog, \mathcal{Z}) \to (Prog, \mathcal{Z}))$, die sich nach Abarbeitung der ersten Anweisung von α im Zustand z als Restprogramm und neuer Zustand ergeben. Sie wird induktiv wie folgt definiert:

- $I_A(\varepsilon, z) = (\varepsilon, z)$.
- $I_A(X := t;\ \beta, z) = (\beta, z(X/val_{A,z}(t)))$.
- $I_A($**if** B **then** γ **else** δ **end**; $\beta, z) = \begin{cases} (\gamma\beta, z) & \text{falls } A \models_z B \\ (\delta\beta, z) & \text{sonst} \end{cases}$
- $I_A($**while** B **do** γ **end**; $\beta, z) =$
  $\begin{cases} (\gamma\,\textbf{while}\ B\ \textbf{do}\ \gamma\ \textbf{end};\ \beta, z) & \text{falls } A \models_z B \\ (\beta, z) & \text{sonst} \end{cases}$

I_A ist wohldefiniert und total auf der Menge $(Prog, \mathcal{Z})$.

Beachte in $I_A($**while** B **do** γ **end**; $\beta, z)$ ist „Restprogramm" strukturell komplexer als Ausgangsprogramm (im Fall $A \models_z B$).

Beispiel (Fort.)

4.30 Beispiel Im Beispiel 4.26: While-Programm $\alpha = S_1S_2S_3$ über Signatur von Nat.

$z(X) = 2, z(Y) = 0, z(Z) = 3$:
Iteration von I_A.

$$\begin{aligned} I_A^9(\alpha, z) &= I_A^9(S_1S_2S_3, z) = I_A^7(S_3, z(Y/0, Z/0)) \\ &= I_A^6(Z := succ(Z + (Y + Y)); \\ &\qquad Y := succ(Y);\ S_3, z(Y/0, Z/0)) \\ &= I_A^5(Y := succ(Y);\ S_3, z(Y/0, Z/1)) \\ &= I_A^4(S_3, z(Y/1, Z/1)) \\ &= I_A^3(Z := succ(Z + (Y + Y)); \\ &\qquad Y := succ(Y);\ S_3, z(Y/1, Z/1)) \\ &\dots \\ &= I_A(S_3, z(Y/2, Z/4)) \\ &= (\varepsilon, z(Y/2, Z/4)) \end{aligned}$$

D. h. $I_A^9(\alpha, z) = (\varepsilon, z')$ mit z' wie gehabt.
Offenbar gilt $z[[\alpha]]_A z'$ für dieses Beispiel.

Äquivalenz der Semantikbegriffe

4.31 Lemma Sei A eine (S, Σ)-Algebra, V eine Variablenmenge, z, z' Zustände über A und V und α ein Programm über (S, Σ) und V. Dann gilt

$$z[[\alpha]]_A z' \text{ gdw } \exists n \in \mathbb{N}^+ : I_A^n(\alpha, z) = (\varepsilon, z')$$

Beweis: Induktion über Aufbau von α.

- α Zuweisung $X := t;$, t Term mit Typ(X) = Typ(t)
  $z[[\alpha]]_A z'$ gdw $z' = z(X/val_{A,z}(t))$
  gdw $I_A(\alpha, z) = (\varepsilon, z')$ (d. h. $n = 1$ gewählt)
- α Test **if** B **then** β **else** γ **end**; analog.
- α Schleife **while** B **do** β **end**; und die Behauptung gelte für β.
  Es gelte $z[[\alpha]]z'$, dann
  $z[[$**while** B **do** β **end**; $]]_A z'$ gdw
  es gibt eine Zahl $m \in \mathbb{N}$ und Zustände $z_0, \dots, z_m$ mit
  - $z = z_0$,
  - $A \models_{z_i} B$ und $z_i[[\beta]]_A z_{i+1}$ für $0 \leq i < m$
  - $A \not\models_{z_m} B$
  - $z_m = z'$

  Wähle minimale Zahlen $n_i (i = 0, \dots, m-1)$ mit $I_A^{n_i}(\beta, z_i) = (\varepsilon, z_{i+1})$. Mit $n := n_0 + \dots + n_{m-1}$ folgt die Behauptung.
  Umkehrung?.

Äquivalenz der Semantikbegriffe (Forts.)

- $\alpha = \beta\gamma$ o.B.d.A. $\beta, \gamma \neq \varepsilon$ Angenommen $z[[\alpha]]_A z'$, dann gibt es einen Zustand z_1 mit $z[[\beta]]z_1$ und $z_1[[\gamma]]z'$. Nach Induktionsvoraussetzung gibt es Zahlen n_1, n_2, so dass
  $I_A^{n_1}(\beta, z) = (\varepsilon, z_1)$ und $I_A^{n_2}(\gamma, z_1) = (\varepsilon, z')$.

Wähle n_1, n_2 minimal mit dieser Eigenschaft. Dann gilt mit $n = n_1 + n_2$:

$$\begin{aligned} I_A^n(\alpha, z) &= I_A^{n_1+n_2}(\beta\gamma, z) = I_A^{n_2}(I_A^{n_1}(\beta\gamma, z)) \\ &= I_A^{n_2}(\gamma, z_1) = (\varepsilon, z') \end{aligned}$$

Wo benötigt man die Minimalität von n_1?

Sei umgekehrt $z[[\alpha]]_A z'$ nicht gültig. Entweder gilt dann $z[[\alpha]]_A z''$ für einen anderen Zustand, insbesondere auch $I_A^n(\alpha, z) = (\varepsilon, z'')$ für ein n wie eben gezeigt, und also für kein n' $I_A^{n'}(\alpha, z) = (\varepsilon, z')$; oder es gibt keinen Zustand z', so dass $z[[\alpha]]_A z'$ gilt. Dann gibt es entweder z'' mit $z[[\beta]]z''$ aber keinen Zustand z' mit $z[[\gamma]]z'$ oder keinen Zustand z'' mit $z[[\beta]]z''$.

Die induktive Voraussetzung liefert im ersten Fall: $I_A^n(\beta, z) = (\varepsilon, z'')$ für ein n und für kein n' kann I_A γ in n' auf den Zustand z'' zum leeren Programm abarbeiten. Dann kann aber auch kein n'' existieren, so dass I_A $\beta\gamma$ auf den Zustand z zum leeren Programm abgearbeitet wird.

Analog im zweiten Fall.

While-Berechenbare Funktionen

4.32 Definition
Sei A eine (S, Σ)-Algebra. Eine Funktion $f : s_A^1 \times \dots \times s_A^n \to s_A$ heißt (while-) **programmierbar** (while-berechenbar) in A, falls es eine Variablenmenge V über (S, Σ), die die paarweise verschiedenen Variablen X_i vom Typ s^i (Eingabevariablen) und Y vom Typ s (Ausgabevariable) enthält, und ein Programm α über (S, Σ) und V gibt, so dass für alle Zustände $z \in \mathcal{Z}(A, V)$ gilt:

$f(z(X_1), \dots, z(X_n)) \downarrow$
$\rightsquigarrow$ es gibt ein $z' \in \mathcal{Z}(A, V)$ mit
$z[[\alpha]]_A z'$ und $z'(Y) = f(z(X_1), \dots, z(X_n))$.

$f(z(X_1), \dots, z(X_n)) \uparrow$
$\rightsquigarrow$ es gibt kein $z' \in \mathcal{Z}(A, V)$ mit $z[[\alpha]]_A z'$.

4.33 Beispiel Beispiel 4.26 (Fort.)

Programm α berechnet die Funktion $f(x) = x^2$ in Nat mit
Eingabevariable X Ausgabevariable Z

Programm β berechnet die Funktion $f(x, y) = x +_{Nat} y$ in N mit
Eingabevariable X, Y Ausgabevariable Z

Programm γ berechnet die Funktion $f(x, y) = x *_{Nat} y$ in N mit
Eingabevariable X, Y Ausgabevariable Z

5 Programmverifikation: Prädikatenlogik und Hoare Kalkül

Spezifizieren - Implementieren - Verifizieren

Prädikatenlogik - While-Programme - Hoare-Kalkül

5.1 Definition Sei (S, Σ) Signatur, V Variablenmenge. Eine **partielle Korrektheitsaussage** über (S, Σ) und V ist eine Zeichenreihe der Form $\{\varphi\}\alpha\{\psi\}$ mit α Programm und φ (**Vorbedingung**), ψ (**Nachbedingung**) Formeln über (S, Σ) und V.

Eine partielle Korrektheitsaussage heißt in einer (S, Σ) Algebra A **gültig**, falls für alle Zustände $z, z' \in \mathcal{Z}(A, V)$ gilt:
$(A \models_z \varphi$ und $z[[\alpha]]_A z') \rightsquigarrow A \models_{z'} \psi$

Schreibweise: $A \models \{\varphi\}\alpha\{\psi\}$

5.2 Bemerkung

Beachte: Es wird somit zugesichert, dass das Programm α, wenn es in einem Zustand, in dem φ gilt, gestartet wird, und wenn es terminiert, einen Zustand in dem ψ gilt, berechnet. Gilt φ im Zustand z, aber terminiert α vom Startzustand z aus nicht (d. h. es gibt gar kein z' mit $z[[\alpha]]_A z'$), so wird *nichts* ausgesagt.

Terminierung kann durch **totalen Korrektheitsaussagen** erfasst werden : $[\boldsymbol{\varphi}]\boldsymbol{\alpha}[\boldsymbol{\psi}]$. Wir behandeln diese hier nicht.
Partielle Korrektheit ist eine logische Eigenschaft, hingegen hängt die

Terminierung von Wohlordnungen ab. Siehe Loeckx, Sieber oder Sperschneider, Antoniou. Terminierungsbedingungen werden hauptsächlich in Verbindung mit Schleifen verwendet.

Beispiele (Fort.)

5.3 Beispiel Programm α über Nat von 4.26

$\alpha :: Y := 0;\ Z := 0;$
$\{Z = Y * Y\}$
while $\neg Y = X$ **do**
$Z := succ(Z + (Y + Y));\ Y := succ(Y);$
end;
$\{Y = X \wedge Z = X * X\}$

α berechnet die Funktion $f(x) = x^2$ in der Variablen Z.

Gültigkeit der partiellen Korrektheitsaussage

$$\{X = X\}\alpha\{Z = X * X\} \text{ in } Nat$$

Abkürzung für Formel, die für jeden Zustand gültig ist: **true**, analog **false** Formel, die für keinen Zustand gültig ist.

Zeige: $Nat \models \{true\}\alpha\{Z = X * X\}$

Beweis: Sei z Zustand, der Variablen in α belegt (genügt $z(X)$!) Es gelte $z[[\alpha]]z'$. Zu zeigen ist $z'(Z) = z'(X)^2$ in Nat.

- $z[[Y := 0;\ Z := 0]]z_1$, dann ist $z_1(Y) = 0, z_1(Z) = 0$ und $z_1(Z) = z_1(Y)^2$.
- Ist $z(X) = 0$, so fertig.

Beispiele (Fort.)

- Sonst
  $z_1[[Z := succ(Z + (Y + Y));\ Y := succ(Y)]]z_2$
  mit
  $z_2(Y) = z_1(Y) + 1$
  $z_2(Z) = z_1(Z) + 2z_1(Y) + 1 = z_1(Y)^2 + 2z_1(Y) + 1$
  $= (z_1(Y) + 1)^2 = z_2(Y)^2$
- Ist $z(X) = 1$, so fertig.
- Induktion nach $z(X)$ liefert Behauptung, da $z_i(X) = z(X)$ und beim Austreten aus der While-Schleife $z_n(Y) = z'(Y) = z(X)$.

Für die Programme β und γ über N gilt entsprechend:

$$Nat \models \{true\}\beta\{Z = X + Y\}$$

bzw.

$$Nat \models \{true\}\gamma\{Z = X * Y\}$$

Sind dies auch partielle Korrektheitsaussagen über der Signatur von N und sind sie gültig in N?

Gibt es eine Möglichkeit den Nachweis von Korrektheitsaussagen systematisch zu führen?

Kalkül zur Ableitung partieller Korrektheitsaussagen

5.4 Definition Der Kalkül von Hoare

φ, ψ, ξ seien Formeln über einer Signatur (S, Σ) und Variablenmenge V, X Variablenbezeichner in V, t ein Term vom selben Typ, B eine boolesche Formel und α, β Programme über $(S, \Sigma), V$.

Regeln des hoareschen Kalküls (HC)

Regeln für das leere Programm:

$$\frac{(\varphi \to \psi)}{\{\varphi\}\varepsilon\{\psi\}}$$

Regeln für Zuweisungen:

$$\frac{\varphi \to [\psi]\{X/t\}}{\{\varphi\}X := t;\ \{\psi\}}$$

Regeln für Testanweisungen:

$$\frac{\{(\varphi \land B)\}\alpha\{\psi\}, \{(\varphi \land \neg B)\}\beta\{\psi\}}{\{\varphi\}\underline{\textbf{if}}\ B\ \underline{\textbf{then}}\ \alpha\ \underline{\textbf{else}}\ \beta\ \underline{\textbf{end}};\ \{\psi\}}$$

Kalkül zur Ableitung partieller Korrektheitsaussagen (2)

Regeln für Schleifen:

$$\frac{(\varphi \to \xi), \{(\xi \land B)\}\alpha\{\xi\}, ((\xi \land \neg B) \to \psi)}{\{\varphi\}\underline{\textbf{while}}\ B\ \underline{\textbf{do}}\ \alpha\ \underline{\textbf{end}};\ \{\psi\}}$$

ξ wird **Schleifeninvariante** genannt.

Regeln für Anweisungsfolgen:

$$\frac{\{\varphi\}\alpha\{\xi\}, \{\xi\}\beta\{\psi\}}{\{\varphi\}\alpha\beta\{\psi\}}$$

Für eine Algebra A ist **HC**(A) die Erweiterung von HC um die Menge aller in A gültigen prädikatenlogischen Formeln über die Signatur von A als Axiome. Schreibweise $\underset{HC(A)}{\vdash} \{\varphi\}\alpha\{\psi\}$.

5.5 Bemerkung Objekte sind hier PL-Formeln und partielle Korrektheitsaussagen.

Ziel ist es gültige Korrektheitsaussagen in einer Algebra A abzuleiten. Dies wird durch Ableitungen in HC(A) realisiert. Alles was in HC(A) abgeleitet werden kann, sollte in A gültig sein.

Zuweisungsregel: Andere Formen z. B. als Axiom

$$\frac{}{\{[\varphi]\{X/t\}\}X := t;\ \{\varphi\}}$$

klar aus Substitutionslemma $A \models_z [\varphi]\{X/t\}$ gdw $A \models_{z(X/a)} \varphi$

Kalkül zur Ableitung partieller Korrektheitsaussagen (3)

Abschwächung der Vor- und Nachbedingungen

$$\frac{\varphi \to \psi, \{\psi\}\alpha\{\xi\}, \xi \to \eta}{\{\varphi\}\alpha\{\eta\}}$$

Dies kann simuliert werden mit HC(A) (über Regel für das leere Programm und Regel für Anweisungsfolgen mit $\alpha\varepsilon = \varepsilon\alpha = \alpha$ für jedes Programm α).

Bei der Anwendung der Schleifenregel ist eine geeignete Invariante zu finden. Gibt es stets eine Formel, die als Invariante verwendet werden kann ?

Bei der Anwendung der Anweisungsfolgenregel ist eine geeignete „Zwischenformel“ ξ zu finden.

Schnittstelle zur Datenstruktur (Theorie von A): Über Zuweisungsregel und Regel für das leere Programm.
Schwierigkeiten bei der Programmverifikation: Der Nachweis von Eigenschaften der Datenstruktur ist für viele Datenstrukturen nicht effektiv durchzuführen (z.B. für Nat).

Notation für Ableitungen im hoareschen Kalkül

5.6 Definition Programme mit Kommentaren

- Kommentar: { PL-Formel }
  Erweiterung der Syntax von Programmen:
- $\{\varphi\}X := t;\ \{\psi\}$
- $\{\varphi\}\underline{\textbf{if}}\ B\ \underline{\textbf{then}}\ \{(\varphi \land B)\}\alpha\{\psi\}\ \underline{\textbf{else}}\ \{(\varphi \land \neg B)\}\beta\{\psi\}\ \underline{\textbf{end}};\ \{\psi\}$
- $\{\varphi\}\{\xi\}\underline{\textbf{while}}\ B\ \underline{\textbf{do}}\ \{(\xi \land B)\}\alpha\{\xi\}\ \underline{\textbf{end}};\ \{(\xi \land \neg B)\}\{\psi\}$
- $\{\varphi\}\alpha\{\xi\}\beta\{\psi\}$

Ein Programm ist **syntaktisch korrekt kommentiert**, wenn die Kommentare im Programm diese Regeln erfüllen. Für die Herleitbarkeit im Kalkül $HC(A)$ benötigt man nur noch die **Beweisverpflichtungen** als Theoreme in A nachzuweisen.
Gezeigt werden müssen:
$A \models (\varphi \to \psi)$,
wenn $\{\varphi\}\{\psi\}$ oder $\{\varphi\}\varepsilon\{\psi\}$ im kommentierten Programmtext vorkommt, bzw.
$A \models (\varphi \to [\psi]\{X/t\})$,
wenn $\{\varphi\}X := t;\ \{\psi\}$ im kommentierten Programmtext vorkommt.

Notation für Ableitungen im hoareschen Kalkül (Forts.)

Vollständig kommentierte Programme:

Zwischen zwei Anweisungen stets Kommentar.

$\{\varphi\}$
A_1
$\{\ldots\}$
$\{\ldots\}$
A_2
$\{\ldots\}$
$\{\ldots\}$
A_3
$\{\ldots\}$
$\{\ldots\}$
$\ldots$
$\{\ldots\}$
$\{\ldots\}$
A_n
$\{\psi\}$

Hilfsmittel beim Nachweis von $\underset{HC(A)}{\vdash} \{\varphi\}A_1 \ldots A_n\{\psi\}$

Werden bei einem vollständig kommentierten Programm alle Beweisverpflichtungen als gültig in A nachgewiesen, so gilt $\underset{HC(A)}{\vdash} \{\varphi\}A_1 \ldots A_n\{\psi\}$.

Beweis !

Beispiel

5.7 Beispiel

Fort. Beispiel 4.26 kommentiertes Programm α für $f(x) = x^2$ über Nat. Spezifikation Pre::{true} Post::$\{Z = (X * X)\}$.

	{true}
	$Y := 0;$
	$\{Y = 0\}$
	$Z := 0;$
φ ::	$\{Y = 0 \land Z = 0\}$
ξ ::	$\{(Y < X \lor Y = X) \land Z = (Y * Y)\}$
	while $\neg Y = X$ **do**
$\xi \land B$ ::	$\{(Y < X \lor Y = X) \land Z = Y * Y \land \neg Y = X\}$
	$Z := succ(Z + (Y + Y));$
	$\{(Y < X \land Z = (succ(Y) * succ(Y))\}$
	$Y := succ(Y);$
ξ ::	$\{(Y < X \lor Y = X) \land Z = (Y * Y)\}$
	end;
$\xi \land \neg B$ ::	$\{((Y < X \lor Y = X) \land Z = (Y * Y)) \land$
	$\neg\neg Y = X\}$
	$\{Z = (X * X)\}$

Syntaktisch korrekt kommentiert, vollständig.

Beispiel (Forts.)

Die Ableitbarkeit in HC(Nat) folgt nun aus den Nachweis der Beweisverpflichtungen:

$(1) Nat \models \text{true} \rightarrow 0 = 0$

$(2) Nat \models Y = 0 \rightarrow (Y = 0 \land 0 = 0)$

$(3) Nat \models (Y = 0 \land Z = 0) \rightarrow ((Y < X \lor Y = X) \land Z = (Y * Y))$

$(4) Nat \models (((Y < X \lor Y = X) \land Z = Y * Y) \land \neg Y = X) \rightarrow (Y < X \land succ(Z + (Y + Y)) = succ(Y) * succ(Y))$

$(5) Nat \models (Y < X \land Z = succ(Y) * succ(Y)) \rightarrow (succ(Y) < X \lor succ(Y) = X) \land Z = succ(Y) * succ(Y)$

$(6) Nat \models (((Y < X \lor Y = X) \land Z = Y * Y) \land \neg\neg Y = X) \rightarrow Z = X * X$

Korrektheit des hoareschen Kalküls

5.8 Satz

Ist die partielle Korrektheitsaussage $\{\varphi\}\alpha\{\psi\}$ in HC(A) ableitbar, so ist $\{\varphi\}\alpha\{\psi\}$ in A gültig.

D. h.

$$\underset{HC(A)}{\vdash} \{\varphi\}\alpha\{\psi\} \rightsquigarrow A \models \{\varphi\}\alpha\{\psi\}$$

Beweis: Strukturelle Induktion (oder Induktion über Länge der Ableitung in HC(A)).

Regel für das leere Programm:
Vor: $A \models (\varphi \rightarrow \psi)$
z. Z. $A \models \{\varphi\}\varepsilon\{\psi\}$.
Seien z, z' Zustände mit $A \models_z \varphi$ und $z[[\varepsilon]]_A z'$. Nach Definition der Semantik gilt $z = z'$, also wegen $A \models_{z'} \varphi$ und $A \models_{z'} (\varphi \rightarrow \psi)$ auch $A \models_{z'} \psi$.

Regel für Zuweisung:
Vor: $A \models \varphi \rightarrow [\psi]\{X/t\}$
z. Z. $A \models \{\varphi\}X := t;\ \{\psi\}$.
Seien z, z' Zustände mit $A \models_z \varphi$ und $z[[X := t]]_A z'$. Nach Definition der Semantik gilt $z' = z(X/a)$ mit $a = val_{A,z}(t)$. Wegen $A \models_z [\psi]\{X/t\}$ folgt $A \models_{z(X/a)} \psi$ aus Substitutionslemma.

Korrektheit des hoareschen Kalküls (2)

Regel für Testanweisung:
Vor: $A \models \{(\varphi \wedge B)\}\beta\{\psi\}$, $A \models \{(\varphi \wedge \neg B)\}\gamma\{\psi\}$

z. Z. $A \models \{\varphi\}$**if** B **then** β **else** γ **end**; $\{\psi\}$.

Seien z, z' Zustände mit $A \models_z \varphi$ und $z[[$**if** B **then** β **else** γ **end**; $]]_A z'$.

Im Fall $A \models_z B$ folgt $A \models_z (\varphi \wedge B)$ und $z[[\beta]]_A z'$. Aus $A \models \{(\varphi \wedge B)\}\beta\{\psi\}$ folgt $A \models_{z'} \psi$.
Analog Fall $A \models_z \neg B$.

Regel für Schleifen:
Vor: $A \models (\varphi \rightarrow \xi)$, $A \models \{(\xi \wedge B)\}\alpha\{\xi\}$ und
$A \models ((\xi \wedge \neg B) \rightarrow \psi)$

z. Z. $A \models \{\varphi\}$**while** B **do** β **end**; $\{\psi\}$

Sei $A \models_z \varphi$ und $z[[$**while** B **do** β **end**$]]_A z'$. Nach Definition von $[[\cdot]]_A$ gibt es $t \in \mathbb{N}$ und Zustände $z_0, \ldots, z_t$ mit $z = z_0$, $A \models_{z_i} B$ und $z_i[[\beta]]_A z_{i+1}$, $0 \leq i < t$, $A \models_{z_t} \neg B$ und $z_t = z'$.

Korrektheit des hoareschen Kalküls (3)

Daraus ergibt sich:
$A \models_{z_0} \varphi \quad (z = z_0)$, $A \models_{z_0} \xi \quad$ (da $A \models \varphi \rightarrow \xi$)
$A \models_{z_0} (\xi \wedge B)$ (B gilt in z_0), $A \models_{z_1} \xi$ (Invarianz von ξ)
$A \models_{z_1} (\xi \wedge B) \cdots$
$\cdots$
$A \models_{z_{t-1}} (\xi \wedge B)$ (B gilt in z_{t-1}) $A \models_{z_t} \xi$ (Invarianz von ξ)
$A \models_{z_t} (\xi \wedge \neg B)$ B gilt nicht mehr
$A \models_{z_t} \psi \quad$ (da $A \models (\xi \wedge \neg B) \rightarrow \psi$)
$A \models_{z'} \psi$ Beh.

Regel für Anweisungsfolgen:
Vor: $A \models \{\varphi\}\alpha\{\xi\}$, $A \models \{\xi\}\beta\{\psi\}$

z. Z. $A \models \{\varphi\}\alpha\beta\{\psi\}$

Sei $A \models_z \varphi$ und $z[[\alpha\beta]]_A z'$ für Zustände z, z'.

Nach Definition von $[[\cdot]]_A$ gibt es Zustand z'' mit $z[[\alpha]]_A z''$ und $z''[[\beta]]_A z'$. Nach Vor. $A \models_{z''} \xi$ und auch $A \models_{z'} \psi$.

Abgeleitete Regeln - Vereinfachungen

5.9 Bemerkung Der Hoarsche Kalkül bleibt korrekt wenn er um die Regel für Zuweisungsfolgen erweitert wird:

$$\frac{(\varphi \rightarrow [\cdots[\psi]\{X_n/t_n\}\cdots]\{X_1/t_1\})}{\{\varphi\}X_1 := t_1;\ \cdots\ ;\ X_n := t_n;\ \{\psi\}} \quad \text{Typ}(X_i) = \text{Typ}(t_i)$$

Dies folgt, da die Regel durch Anwenden der Regeln

$$\frac{(\varphi \rightarrow [\cdots[\psi]\{X_n/t_n\}\cdots]\{X_1/t_1\})}{\{\varphi\}X_1 := t_1;\ \{[\cdots[\psi]\{X_n/t_n\}\cdots]\{X_2/t_2\}\}}$$

$$\frac{([\cdots[\psi]\{X_n/t_n\}\cdots]\{X_i/t_i\} \rightarrow [\cdots[\psi]\{X_n/t_n\}\cdots]}{\{[\cdots[\psi]\{X_n/t_n\}\cdots]\{X_i/t_i\}\}X_i := t_i;\ \dfrac{\{X_i/t_i\}}{\{[\cdots[\psi]\{X_n/t_n\}\cdots]\{X_{i+1}/t_i\}}}$$

für $(i = 2, \ldots, n)$ und iteriertes Anwenden der Anweisungsfolgen-regel simulieren lässt.

Als Kommentar in den Programmtext übertrage die Regel als

$$\begin{array}{l} \{\varphi\} \\ X_1 := t_1; \\ \cdots \\ X_n := t_n; \\ \{\psi\} \end{array}$$

Als Beweisverpflichtung muss gezeigt werden
$A \models (\varphi \rightarrow [\cdots[\psi]\{X_n/t_n\}\cdots]\{X_1/t_1\})$

Beachte dabei die Reihenfolge der Substitutionen.

Beispiele

5.10 Beispiel Programm, das den Wert der Variablen X, Y tauscht.

X, Y, X', Y', Z seien Variablen vom gleichen Typ.

$$\begin{array}{l} \{(X = X' \wedge Y = Y')\} \\ Z := X;\ X := Y;\ Y := Z; \\ \{(Y = X' \wedge X = Y')\} \end{array}$$

z. Z.
$A \models (X = X' \wedge Y = Y') \rightarrow$
$[[[(Y = X' \wedge X = Y')]\{Y/Z\}]\{X/Y\}]\{Z/X\} =$
$[[(Z = X' \wedge X = Y')]\{X/Y\}]\{Z/X\} =$
$[(Z = X' \wedge Y = Y')]\{Z/X\} =$
$(X = X' \wedge Y = Y')$

d. h. z. Z.:

$$A \models (X = X' \wedge Y = Y') \rightarrow (X = X' \wedge Y = Y')$$

was richtig ist.

Problemspezifikation:

- Festlegung der Signatur (S, Σ).
- Festlegung der Algebra A.
- Festlegung der Vor- und Nachbedingung φ, ψ.
- Festlegung weiterer Hilfsinformation.

Finde Programm α mit $A \models \{\varphi\}\alpha\{\psi\}$.

Beispiel: GGT-Berechnung

5.11 Beispiel Algebra Nat mit Signatur-Erweiterung

$- : nat \times nat \rightarrow nat$ mit $n -_{Nat} m = \begin{cases} 0 & m > n \\ n - m & \text{sonst} \end{cases}$

(Übung: Schreibe while-Programm über Signatur von Nat dafür).

$X \mid Y \equiv \exists V\, V * X = Y, \quad X \leq Y \equiv (X = Y \vee X < Y)$

als Abkürzungen, dann gilt in Nat

$\text{GGT}(X, Y) = Z \equiv$
$Z \mid X \wedge Z \mid Y \wedge \forall V((V \mid X \wedge V \mid Y) \rightarrow V \mid Z)$

```
φ:: {X > 0, Y > 0}
    A := X; B := Y;
    {ξ :: {X > 0, Y > 0, GGT(X,Y) = GGT(A,B)}}
    while A ≠ B do            {ξ, A ≠ B}
        if B < A
          then               {ξ, A ≠ B, B < A}
               A := A − B;           {ξ}
          else               {ξ, A ≠ B, ¬B < A}
               B := B − A;           {ξ}
        end;            {ξ}
    end;            {ξ, A = B}
ψ:: {A = GGT(X,Y)}
```


Beispiel GGT-Berechnung (Forts.)

Die nachzuweisenden Beweisverpflichtungen in Nat sind:

- $(X > 0 \wedge Y > 0) \rightarrow [[\xi]\{B/Y\}]\{A/X\}$
  d. h.
  $(X > 0 \wedge Y > 0) \rightarrow (X > 0 \wedge Y > 0 \wedge \text{GGT}(X, Y) = \text{GGT}(X, Y))$
- $(\xi \wedge A \neq B \wedge B < A) \rightarrow [\xi]\{A/A - B\}$
  d. h.
  $(X > 0 \wedge Y > 0 \wedge \text{GGT}(X, Y) = \text{GGT}(A, B) \wedge B < A) \rightarrow (X > 0 \wedge Y > 0 \wedge \text{GGT}(X, Y) = \text{GGT}(A - B, B))$
- $(\xi \wedge A \neq B \wedge \neg B < A) \rightarrow [\xi]\{B/B - A\}$
  d. h.
  $(X > 0 \wedge Y > 0 \wedge \text{GGT}(X, Y) = \text{GGT}(A, B) \wedge A < B) \rightarrow (X > 0 \wedge Y > 0 \wedge \text{GGT}(X, Y) = \text{GGT}(A, B - A))$
- $(\xi \wedge A = B) \rightarrow A = \text{GGT}(X, Y)$
  d. h.
  $X > 0 \wedge Y > 0 \wedge \text{GGT}(A, B) = \text{GGT}(X, Y) \wedge A = B) \rightarrow A = \text{GGT}(X, Y)$

Diese sind „leicht“ über Nat als gültig nachzuweisen.

Beispiel: Problemspezifikation

5.12 Beispiel Finde Programm über den natürlichen Zahlen mit Division und Modulofunktion, das zu zwei Zahlen X, Y die Zahl X^Y berechnet.

$\text{div} : nat \times nat \rightarrow nat$ ganzzahlige Division
$\text{mod} : nat \times nat \rightarrow nat$ Rest modulo ...

(Hilfsfunktionen: Übung: Schreibe Programme über Nat für diese Funktionen)

Vorbedingung: true $(0 = 0)$ **Eingabevariable**: $A, B : nat$
Nachbedingung: $Z = A^B$ **Ausgabevariable**: $Z : nat$

(Hierbei steht $Z = A^B$ als Abkürzung für PL-Formel über Signatur von Nat, die A^B beschreibt: A^B steht für $A * \cdots * A$ B-mal $(B = 0$, so 1, d.h. auch $0^0 = 1))$

Existenz einer solchen Formel wird vorausgesetzt!

Formeln um Eigenschaften zu beschreiben, insbesondere Folgen mit bestimmten Eigenschaften ⇝ Logik.

Verwendet werden im Programm nur

$Y \bmod 2$ und $Y \operatorname{div} 2$

Programm

Programm α mit $nat \models \{0 = 0\}\alpha\{Z = A^B\}$:

Verwendete Idee: $X^{2k} = (X^2)^k$
$X^{2k+1} = X^{2k} * X$

```
{0 = 0}
X := A; Y := B; Z := 1;
{ξ :: {X^Y * Z = A^B}
while ¬Y = 0 do
    {                                }
    if Y mod 2 = 0
      then
          {                              }
          Y := Y div 2; X := X * X;
          {                              }
      else
          {                              }
          Y := Y − 1; Z := Z * X;
          {                              }
    end;
    {                                }
end; {X^Y * Z = A^B, Y = 0}
{Z = A^B}
```

Beweisverpflichtungen

Verbleibende Beweisverpflichtungen: in Nat.

- $\models 0 = 0 \rightarrow A^B * 1 = A^B$
- $\models (\neg Y = 0 \wedge Y \bmod 2 = 0 \wedge X^Y * Z = A^B) \rightarrow (X * X)^{Y \operatorname{div} 2} * Z = A^B$
- $\models (\neg Y = 0 \wedge \neg Y \bmod 2 = 0 \wedge X^Y * Z = A^B) \rightarrow X^{Y-1} * Z * X = A^B$
- $\models (X^Y * Z = A^B \wedge Y = 0) \rightarrow Z = A^B$

Die Gültigkeit dieser Formeln folgt aus „Idee“ und aus Bedeutung von A^B.

Frage: Kalkül von Hoare ist korrekt. Ist er auch für jede Signatur und Algebra A vollständig? D. h. gilt

aus $\models_A \{\varphi\}\alpha\{\psi\}$ folgt stets $\underset{HC(A)}{\vdash} \{\varphi\}\alpha\{\psi\}$ für φ, ψ PL-Formeln und α While-Programme.

Nein nur falls A **ausdrucksstark** (oder expressiv).

Frage:

1. Gegeben α, ψ. Gibt es stets eine Formel φ mit $A \models \{\varphi\}\alpha\{\psi\}$?
   $\rightsquigarrow$ „Schwächste Vorbedingung“
2. Gegeben φ, α. Gibt es stets eine Formel ψ mit $A \models \{\varphi\}\alpha\{\psi\}$?
   $\rightsquigarrow$ „Stärkste Nachbedingung“

Schwächste Vorbedingung Stärkste Nachbedingung

5.13 Definition

Sei A eine (S, Σ)-Algebra, α ein Programm und φ eine Formel über (S, Σ) und V die Menge der in α und φ auftretenden Variablen.

Die **schwächste Vorbedingung** $wlp_A(\alpha, \varphi)$ von α und φ über der Algebra A ist folgende Menge von Zuständen über A, V:

$$wlp_A(\alpha, \varphi) := \{z : \forall z'(z[[\alpha]]_A z' \Rightarrow A \models_{z'} \varphi)\}$$

Die **stärkste Nachbedingung** $spc_A(\varphi, \alpha)$ von φ und α über der Algebra A ist folgende Menge von Zuständen über A, V:

$$spc_A(\varphi, \alpha) := \{z : \exists z'(A \models_{z'} \varphi \wedge z'[[\alpha]]_A z)\}$$

Sei $Z \subseteq \mathcal{Z}(A, V)$ eine Menge von Zuständen über einer Algebra A und Variablenmenge V. Z heißt in A **definierbar**, falls es eine Formel φ mit freien Variablen in V und $Z = \{z : A \models_z \varphi\}$ gibt.

Literatur: $wlp_A(\alpha, \varphi)$ „Schwächste liberale Vorbedingung“

Eigenschaften von wlp_A und spc_A

5.14 Lemma

A sei eine Algebra, α ein Programm, φ, ψ Formeln und z, z' Zustände über der sich ergebenden Variablenmenge. Es gilt

- aus $z \in wlp_A(\alpha, \psi)$ und $z[[\alpha]]_A z'$ folgt $A \models_{z'} \psi$,
- aus $A \models \{\varphi\}\alpha\{\psi\}$ und $A \models_z \varphi$ folgt $z \in wlp_A(\alpha, \psi)$,
- aus $A \models_z \varphi$ und $z[[\alpha]]_A z'$ folgt $z' \in spc_A(\varphi, \alpha)$,
- aus $A \models \{\varphi\}\alpha\{\psi\}$ und $z' \in spc_A(\varphi, \alpha)$ folgt $A \models_{z'} \psi$,
- $wlp_A(\alpha, \psi)$ sei in A durch eine Formel $W_{\alpha,\psi}$ definierbar. Dann gilt
  $A \models \{W_{\alpha,\psi}\}\alpha\{\psi\}$ und für alle ξ folgt aus $A \models \{\xi\}\alpha\{\psi\}$ die Aussage $A \models (\xi \rightarrow W_{\alpha,\psi})$
  (d. h. sie wird von jeder anderen Vorbedingung impliziert).
- $spc_A(\varphi, \alpha)$ sei in A durch eine Formel $S_{\varphi,\alpha}$ definierbar. Dann gilt
  $A \models \{\varphi\}\alpha\{S_{\varphi,\alpha}\}$ und für alle ξ folgt aus $A \models \{\varphi\}\alpha\{\xi\}$ die Aussage $A \models (S_{\varphi,\alpha} \rightarrow \xi)$.
  (d. h. sie impliziert jede andere Nachbedingung).
- $wlp_A(X := t, \psi)$ definierbar durch $W_{X:=t,\psi} = [\psi]\{X/t\}$.
- $S_{\varphi,X:=t} = \exists Y([\varphi]\{X/Y\} \wedge X = t\{X/Y\})$.
  $(Y \notin \mathrm{Var}(X, t, \varphi))$ (Definiert $spc_A(\varphi, X := t)$)

Beispiele

5.15 Beispiel Algebra Nat mit $- : nat \times nat \rightarrow nat$.

a) Sei $\varphi :: X > Y \qquad X := t; \quad t :: X - Y$
   $[\varphi]\{X/t\} :: X - Y > Y$ offenbar

   $$Nat \models \{X - Y > Y\}X := X - Y;\ \{X > Y\}$$

b) $\exists Z([\varphi]\{X/Z\} \wedge X = t\{X/Z\})$

   $$\exists Z(Z > Y \wedge X = Z - Y)\ (\underset{Nat}{\leftrightarrow} X > 0)$$

   Also ist
   $$Nat \models \{X > Y\}X := X - Y;\ \{X > 0\}$$

c) $\models \{\text{true}\}X := X - Y; \quad \{\exists Z(\text{true} \wedge X = Z - Y)\}$
   $\updownarrow$
   $\{\exists Z X = Z - Y\}$

Beispiele (Forts.)

d) $\models \{X = Y\} X := X + Y;\ Y := X + Y;\ \{3X = 2Y\}$
semantisch klar!
Unter Anwendung von wlp
$\models \{3X = 2(X + Y)\} Y := X + Y;\ \{3X = 2Y\}$
$\models \{3(X + Y) = 2(X + Y + Y)\} X := X + Y;$
$\{3X = 2(X + Y)\}$
Dann ist
$\models \{3(X + Y) = 2(X + Y + Y)\} X := X + Y;$
$Y := X + Y;\ \{3X = 2Y\}$.
Es gilt:
$Nat \models X = Y \rightarrow 3(X + Y) = 2(X + Y + Y)$.
Also gilt auch
$Nat \models \{X = Y\} X := X + Y;\ Y := X + Y;$
$\{3X = 2Y\}$.
Unter Anwendung von spc
$\models \{X = Y\} X := X+Y;\ \{\exists Z([X = Y]\{X/Z\} \wedge X = (X + Y)\{X/Z\})\}$ d.h.
$\models \{X = Y\} X := X+Y;\ \{\exists Z(Z = Y \wedge X = Z+Y)\}$
$\models \{X = Y\} X := X + Y;\ \{X = Y + Y)\}$
$\models \{X = Y + Y\} Y := X + Y;\ \{\exists V([X = Y + Y]\{Y/V\} \wedge Y = (X + Y)\{Y/V\}\}$ d.h. $\models \{X = Y + Y)\} Y := X + Y;\ \{\exists V(X = V + V \wedge Y = X + V)\}$
$\models_{Nat} \exists V(X = V + V \wedge Y = X + V) \rightarrow 3X = 2Y$

Beispiele (Forts.)

e) Sei N über $(nat, 0 \rightarrow nat, succ : nat \rightarrow nat)$.
$\mathbb{N}$ als Grundbereich, 0_N, $succ_N$ die üblichen.
Welche Teilmengen von $\mathbb{N}$ lassen sich durch Formeln über der Signatur von N darstellen? (Formeln mit einer freien Variablen X).
$X = succ^n(0)$ stellt $\{n\}$ dar.
$X = succ^n(X)$ stellt $\varnothing$ oder $\mathbb{N}$ $(n = 0)$ dar.
$succ^n(0) = succ^m(X)$ stellt $\underset{(m>n)}{\varnothing}$ oder $\{n - m\}$ dar.
Endliche und co-endliche (Komplement endlich)Teilmengen von $\mathbb{N}$.
$\exists Y : X = succ^n(Y)$:

f) Sei A eine Algebra. z Zustand über A und V. Wann ist z definierbar?
Hinreichende Bedingung: $V = \{X_1, \ldots, X_n : s\}$

$$z(X_1) = a_1 \ldots \quad z(X_n) = a_n \quad a_1, \ldots, a_n \in s_A$$

Angenommen es gibt Grundterme (Terme ohne Variablen) über Signatur von A mit $val_{A,\cdot}(t_i) = a_i$.
Dann definiert $(X_1 = t_1 \wedge \cdots \wedge X_n = t_n)$ den Zustand z.
Offenbar lässt sich dann jede endliche Menge von Zuständen definieren und falls die Grundbereiche der Algebra endlich sind, lässt sich jede Zustandsmenge durch eine Formel φ definieren.

Ausdrucksstarke Algebren

5.16 Definition

Sei (S, Σ) eine Signatur. Eine (S, Σ)-Algebra A heißt **ausdrucksstark** (expressiv), falls für jedes Programm α und jede Formel φ über (S, Σ) die Zustandsmenge $wlp_A(\alpha, \varphi)$ in A definierbar ist.

Bemerkung: Man kann diesen Begriff äquivalent über die Menge $spc(\varphi, \alpha)$ definieren.(Beweis: Übung macht den Meister).

5.17 Satz

Über einer ausdrucksstarken Algebra A ist $HC(A)$ vollständig.

Beweis: Es gelte $A \models \{\varphi\}\alpha\{\psi\}$, d. h. für z, z' Zustände folgt aus $A \models_z \varphi$ und $z[[\alpha]]_A z'$ stets $A \models_{z'} \psi$.

Zeige $\underset{HC(A)}{\vdash} \{\varphi\}\alpha\{\psi\}$.

Induktion über Aufbau von α: nur while Fall.

Sei $A \models \{\varphi\}$**while** B **do** β **end**; $\{\psi\}$.

Sei ξ Formel, die $wlp_A($**while** B **do** β **end**; $, \psi)$ in A definiert. Dann gilt

- $A \models (\varphi \rightarrow \xi)$ aus Definition der schwächsten Vorbedingung und $A \models \{\varphi\}$**while** B **do** β **end**; $\{\psi\}$.
  Also gilt $\underset{HC(A)}{\vdash} (\varphi \rightarrow \xi)$.

Ausdrucksstarke Algebren (Fort.)

- $A \models \{\xi \wedge B\}\beta\{\xi\}$ da:
  Sei $A \models_z (\xi \wedge B)$ und $z[[\beta]]_A z'$ für Zustände z, z'.
  Es ist $A \models_{z'} \xi$ zu zeigen.
  Zeige dazu $z' \in wlp_A($**while** B **do** β **end**; $, \psi)$.
  Sei also $z'[[$**while** B **do** β **end**; $]]_A z''$.
  Wegen $A \models_z B$ und $z[[\beta]]_A z'$, gilt auch $z[[$**while** B **do** β **end**; $]]_A z''$.
  Wegen $A \models_z \xi$ gilt auch $z \in wlp_A($**while** B **do** β **end**; $, \psi)$
  also $A \models_{z''} \psi$. Also $A \models_{z'} \xi$.
- $A \models ((\xi \wedge \neg B) \rightarrow \psi)$ ergibt sich wie folgt:
  Sei $A \models_z (\xi \wedge \neg B)$. Da die Schleife sofort abgebrochen wird, gilt $z[[$**while** B **do** β **end**; $]]_A z$. Wegen $A \models_z \xi$ kann man $z \in wlp_A($**while** B **do** β **end**; $, \psi)$ folgern.
  Mit der Definition von wlp folgt $A \models_z \psi$.

Mit der Induktionsvoraussetzung und der Schleifenregel erhält man die Ableitbarkeit in $HC(A)$.

Ohne Beweis.

- Die Algebra Nat ist ausdrucksstark.
- Die Algebra N ist nicht ausdrucksstark.
- Jede Algebra A mit endlichen Grundbereichen ist ausdrucksstark.

Beweis für Interessierte: Seite 36-39 Sp/Ha oder Loeckx/Sieber.

Erweiterungen der Programmiersprache

5.18 Bemerkung Die meisten Programmiersprachen bieten eine Vielzahl weiterer Programmkonstrukte an. Will man solche Erweiterungen der Sprache der While-Programme zulassen, so muss eine geeignete Syntax für diese Konstrukte gewählt werden und entsprechende Syntax-Regeln angegeben werden. Die denotationale Semantik bzw. die Interpretersemantik muss für diese Konstrukte erweitert werden und schließlich müssen die Regeln im Hoare-Kalkül angegeben werden um die Verifikationsaufgabe zu systematisieren.
Programmiersprachen bieten Möglichkeiten für Makros,(rekursive) Prozeduren, rekursive Programme, Sprünge usw.. Sie ermöglichen und unterstützen damit die strukturierte Programmentwicklung. Es stellt sich die Frage, ob diese Konstrukte die Berechnungsmächtigkeit erweitern oder ob die Klasse der While-berechenbaren Funktionen sich dadurch nicht verändert.

Konstrukte wie etwa „**repeat** α **until** B **end;**“ oder eine zählergesteuerte Schleife lassen sich leicht mit Hilfe der while-Schleife simulieren, d.h. sie sind eigentlich nur „syntaktischer Zucker“. Makros und nicht-rekursive Prozeduren fallen auch in diese Kategorie.

An dieser Stelle sollen nur noch die rekursiven Prozeduren anhand von Beispielen erläutert werden. Für eine genauere Untersuchung siehe Sperschneider/Hammer. Prozeduren müssen vor ihrem Aufruf mit **call** P(...) deklariert werden und ihre Wirkung spezifiziert werden.
D.h man benötigt **Sonderzeichen: proc call in out**

Beispiele für Prozeduren

N mit Signatur $0 :\to nat, succ : nat \to nat$

β:: $Z := X;$
$Z' := 0;$
while $\neg Y = Z'$ **do**
$Z := succ(Z);$
$Z' := succ(Z');$
end;

Kopf: **proc** ADD_1(**in** $X, Y : nat,$ **out** $Z : nat)$
Komment: {true} **call** $ADD_1(X, Y, Z);\ \{Z = X +_{nat} Y\}$
begin
Rumpf: β
end.

γ:: $Z := 0;\ Z' := 0;$
while $\neg Y = Z'$ **do**
call $ADD_1(Z, X, X');$
$Z := X';$
$Z' := succ(Z');$
end;

Syntax des Prozeduraufrufs
call $P(t_1, \dots, t_n, U_1, \dots, U_m)$; wobei Typ$(t_i)$ = Typ(X_i)
$U_1, \dots, U_m$ kommen nicht in $t_1, \dots, t_n$ vor: Nebenwirkungsfrei.

Beispiele für Prozeduren

proc G (**in** $X, Y : nat,$ **out** $Z : nat)$
{true} **call** $G(X, Y, Z);\ \{X = succ(Y) \land Z = Y\}$
begin
if $\neg X = succ(Y)$ **then call** $G(X, succ(Y), Z);$
else $Z := Y;$
end;
end.

proc $PRED$ (**in** $X : nat,$ **out** $Z : nat)$
{true} **call** $PRED(X, Z);\ \{(X = 0 \land Z = 0) \lor succ(Z) = X\}$
begin
if $X = 0$ **then** $Z := 0;$ **else call** $G(X, 0, Z);$
end;
end.

proc $PRED'$(**in** $X : nat,$ **out** $Z : nat)$
{true} **call** $PRED'(X, Z);\ \{(X = 0 \land Z = 0) \lor succ(Z) = X\}$
begin $Y := 0;\ Z := 0;$
while $\neg X = Y$ **do** $Z := Y;\ Y := succ(Y);$ **end**;
end.

Prozedurdeklarationen

Eine **Prozedurumgebung** über einer Signatur (S, Σ) besteht aus einer endlichen Menge Ω von Prozedurdeklarationen (Prozeduren) der Form:

Kopf: **proc** P (**in** $\vec{X}$, **out** $\vec{Y}$)
Komment: $\{\varphi(\vec{X})\}$ **call** $P(\vec{X}, \vec{Y});\ \{\psi(\vec{X}, \vec{Y})\}$
begin
Rumpf: β
end.

Mit folgenden Eigenschaften:

- Die verwandten Prozedurnamen sind paarweise verschieden.
- **proc** P(**in** $\vec{X}$, **out** $\vec{Y}$) ist Prozedurkopf über (S, Σ).
- β ist ein rekursives Programm über (S, Σ) und zu jedem Prozeduraufruf **call** $Q(\vec{t}, \vec{U})$; in β gibt es eine Prozedur in Ω mit Namen Q und passendem Prozedurkopf.
- β verändert keine **in**-Parameter d. h. in β keine Anweisungen der Form $X_i := t$: oder **call** $Q(\vec{t} \dots X_i \dots)$, d. h. $X_i \notin \{\vec{U}\}$.
- $var(\varphi(\vec{X})) \subseteq \{\vec{X}\}, var(\psi(\vec{X}, \vec{Y})) \subseteq \{\vec{X}\} \cup \{\vec{Y}\}$.
- Keine „globalen“ Variablen in Prozeduren. Kommmentare werden zur Spezifikation und Verifikation des Programmverhaltens verwendet, sie beeinflussen nicht die Abarbeitung. Sie sind auch Hilfsmittel für die Wiederverwendung (suchen von Programmen mit bestimmten Eigenschaften).
- Beweisverpflichtung: $A \models \{\varphi(\vec{X})\}\beta\{\psi(\vec{X}, \vec{Y})\}$

Prozeduraufrufregel-Beispiel

Der hoarsche Kalkül $HC(\Omega)$ über Ω ergibt sich durch Hinzunahme der folgenden neuen Regel

$$\frac{\pi \to [\varphi]\{\vec{X}/\vec{t}\}, ([\psi]\{\vec{X}/\vec{t}, \vec{Y}/\vec{U}\} \wedge \exists \vec{U}\pi) \to \varrho}{\{\pi\}\ \underline{\textbf{call}}\ P(\vec{t}, \vec{U});\ \{\varrho\}}$$

Zu jeder Prozedur P, die in Ω deklariert ist.

$HC(\Omega, A)$ sei $HC(\Omega)$ erweitert, um die in der Algebra A gültigen PL-Formeln. Eine Prozedurumgebung Ω heißt über einer Algebra A korrekt kommentiert, sofern für jede Prozedurdeklaration in Ω die partielle Korrektheitsaussage $\{\varphi(\vec{X})\}\beta\{\psi(\vec{X}, \vec{Y})\}$ in $HC(\Omega, A)$ ableitbar ist.

Beispiel 91-Funktion von McCarthy über (Nat, -)

```
proc P91( in X : nat, out Y : nat)
  {true} call P91(X, Y); {ψ(X, Y)}
  begin
    if X > 100
      then
          Y := X − 10;
       else call P91(X + 11, U); call P91(U, Y);
    end;
  end.
```


Beispiel 91-Funktion

Mit Spezifikation $\psi(X, Y)$ ::
$((X > 100 \to Y = (X - 10)) \wedge (X \leq 100 \to Y = 91))$
Behauptung:

$\vdash_{HC(\Omega,A)} \{\text{true}\}\beta\{\psi(X, Y)\}$, d. h. korrekt kommentierte Prozedur.

Syntaktisch korrekte Kommentierung von β:

```
{true}
if X > 100
  then
       {X > 100}Y := X − 10; {ψ(X, Y)}
   else {¬X > 100}
        call P91(X + 11, U);
        {¬X > 100 ∧ [ψ]{X/X + 11, Y/U}}
        call P91(U, Y);
        {ψ(X, Y)}
end;
{ψ(X, Y)}
```


Beweisverpflichtungen

- $X > 100 \to ((X > 100 \to (X - 10) = (X - 10)) \wedge (X \leq 100 \to (X - 10) = 91))$ ok.
- $\neg X > 100 \to$ true ok.
- $([\psi]\{X/X + 11, Y/U\} \wedge \exists U \neg X > 100) \to (\neg X > 100 \wedge [\psi]\{X/X + 11, Y/U\})$ ok.
- $(\neg X > 100 \wedge [\psi]\{X/X + 11, Y/U\}) \to$ true ok.
- $[\psi]\{X/U, Y/Y\} \wedge \exists U(\neg X > 100 \wedge [\psi]\{X/X + 11, Y/U\}) \to \psi[X, Y]$

$((U > 100 \to Y = (U - 10)) \wedge (U \leq 100 \to Y = 91))$
$\wedge(\neg X > 100 \wedge (X + 11 > 100 \to U = X + 1) \wedge$
$(X + 11 \leq 100 \to U = 91))$
$\to (X > 100 \to Y = (X - 10)) \wedge (X \leq 100 \to Y = 91)$

Fall: $z(X) =$

$\cdot 100$ $\quad \cdot z(U) = 101$ $\quad z(Y) = z(U - 10) = 91$

$\cdot 90 \leq \cdot < 100$ $\quad \cdot z(U) = z(X + 1) \leq 100$ $\quad z(Y) = 91$

$\cdot \leq 89$ $\quad z(U) = 91$ $\quad z(Y) = 91$

$\rightsquigarrow$ Behauptung

6 Berechenbarkeit

Programmierbare/Berechenbare Funktionen

- Imperative Programmiersprache: while-Programme
- Funktionale Programmiersprache: μ-rekursive Ausdrücke
- Logische Programmiersprachen: Prolog ...
  Deklarativ vs. Prozedural
- Abstrakte Maschinen Modelle
  Turing-Maschine, Register-Maschine;
- Techniken: Simulation von Berechnungen, Übersetzung, Interpretation
- Universelle Modelle: Compiler

Aufbau Kapitel 6:

- Primitiv rekursive Funktionen
- μ-rekursive Funktionen (partiell rekursive Funktionen)
- Universalität
- Rekursionstheorie
- Churchsche These
- Wortfunktionen

6.1 Primitiv rekursive Funktionen $\mathcal{P}(\mathbb{N})$

Funktionen: $f : \mathbb{N}^n \to \mathbb{N} \quad n \geq 1$. **Arithmetische Funktionen.**

Verwende „effektive"- Operatoren auf Funktionen, um aus Ausgangsfunktionen + Operatoren neue Funktionen zu definieren.

Erinnerung: Gleichheit von Funktionen $f : A \to B,\ g : A \to B$

$f \sqsubseteq g$ gdw $dom(f) \subseteq dom(g) \wedge f(x) = g(x)$
für $x \in dom(f)$

$f = g$ gdw $dom(f) = dom(g) \wedge f(x) = g(x)$
für $x \in dom(f)$ gdw $(f \sqsubseteq g \wedge g \sqsubseteq f)$.

6.1 Definition Komposition - Primitive Rekursion

a) Seien $g : \mathbb{N}^n \to \mathbb{N},\ h_1, \ldots, h_n : \mathbb{N}^m \to \mathbb{N}$ Funktionen $n, m \geq 1$
$f : \mathbb{N}^m \to \mathbb{N}$ entsteht aus g und $h_1, \ldots, h_n$ durch **Komposition**, falls gilt
$f(\vec{x}) \downarrow$ gdw $h_1(\vec{x}) \downarrow, \ldots, h_n(\vec{x}) \downarrow$ und $g(h_1(\vec{x}), \ldots, h_n(\vec{x})) \downarrow$
und in diesem Fall ist
$f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_n(\vec{x}))$
Schreibe dafür $\quad f = g \circ (h_1, \ldots, h_n)$
Beachte Stelligkeiten der Funktionen.
Sind $g, h_1, \ldots, h_n$ total, so auch f.
Gilt die Umkehrung?

Operatoren auf Funktionen

b) Seien $g : \mathbb{N}^{n+1} \to \mathbb{N},\ h : \mathbb{N}^{n+2} \to \mathbb{N}$ Funktionen.
$f : \mathbb{N}^{n+1} \to \mathbb{N}$ entsteht aus g und h durch **primitive Rekursion**, falls gilt:
$f(\vec{x}, 0) \downarrow$ gdw $g(\vec{x}, 0) \downarrow$ $(\vec{x} \in \mathbb{N}^n)$ und in diesem Fall ist
$f(\vec{x}, 0) = g(\vec{x}, 0)$ und
$f(\vec{x}, y+1) \downarrow$ gdw $f(\vec{x}, y) \downarrow$ und $h(\vec{x}, f(\vec{x}, y), y) \downarrow$
$(\vec{x} \in \mathbb{N}^n)$ und in diesem Fall ist
$f(\vec{x}, y+1) = h(\vec{x}, f(\vec{x}, y), y)$
Schreibe dafür $\quad f = R(g, h)$.
Beachte Stelligkeiten der Funktionen.

6.2 Bemerkung - Beispiele: Betrachtet man die Gleichung

$$F(\vec{x}, z) = \begin{cases} g(\vec{x}, 0) & z = 0 \\ h(\vec{x}, F(\vec{x}, y), y) & z = y + 1 \end{cases}$$

so ist $f = R(g, h)$ die kleinste (bzgl. $\sqsubseteq$) Funktion, die diese Gleichung erfüllt.

Sind $g(\cdot, 0) : \mathbb{N}^n \to \mathbb{N}$ und $h : \mathbb{N}^{n+2} \to \mathbb{N}$ total, so ist auch f total.

Gilt die Umkehrung?

Die primitive Rekursion folgt einem sehr strengen Schema und entspricht der Berechnung des Funktionswerts $f(\vec{x}, n+1)$ aus dem Funktionswert $f(\vec{x}, n)$, wobei die Verankerung bei $f(\vec{x}, 0)$ erfolgt.

Beispiele

- $+$ Addition auf $\mathbb{N}^2$: Mit $g(u, v) = u$ und $h(u, v, w) = v + 1$ gilt $x + y = R(g, h)$
$$x + y = \begin{cases} x & \text{falls } y = 0 \\ (x + (y-1)) + 1 & \text{falls } y \neq 0 \end{cases}$$
oder
$$x + y = \begin{cases} x & \text{falls } y = 0 \\ (x + z) + 1 & \text{falls } y = z + 1 \end{cases}$$
- $\cdot$ Multiplikation auf $\mathbb{N}^2$: Mit $g(u, v) = 0$ und $h(u, v, w) = v + u$ gilt $x \cdot y = R(g, h)$
$$x \cdot y = \begin{cases} 0 & \text{falls } y = 0 \\ (x \cdot z) + x & \text{falls } y = z + 1 \end{cases}$$
- Fakultät fac auf $\mathbb{N}$: Mit $g(u) = 1$ und $h(u, v) = u \cdot (v + 1)$ gilt $fac(y) = R(g, h)$
$$fac(y) = \begin{cases} 1 & \text{falls } y = 0 \\ fac(z) \cdot (z + 1) & \text{falls } y = z + 1 \end{cases}$$
- Welche Funktion f wird durch folgende Festlegung definiert. Sei
$$h(u, v) = \begin{cases} u & u \text{ gerade} \\ \uparrow & \text{sonst} \end{cases}$$
$$f(y) = \begin{cases} 0 & \text{falls } y = 0 \\ h(f(z), z) & \text{falls } y = z + 1 \end{cases}$$

Primitiv rekursive Ausdrücke

6.3 Definition

Syntax: Die Menge der primitiv rekursiven Ausdrücke sind die Zeichenreihen, die durch den folgenden Kalkül erzeugt werden:

$$\frac{}{NULL} \qquad \frac{}{SUCC} \qquad \frac{}{PROJ(i)} \text{ für } i \geq 1$$

$$\frac{G, H_1, \ldots, H_m}{KOMP(G, H_1, \ldots, H_m)} \text{ für } m \geq 1 \qquad \frac{G, H}{REK(G, H)}$$

Semantik: Jeder primitiv rekursive Ausdruck π repräsentiert für <u>beliebige Stelligkeit</u> $n \geq 1$ eine Funktion $f_\pi^{(n)} : \mathbb{N}^n \to \mathbb{N}$, die induktiv über den Aufbau von π wie folgt definiert ist:

$$\left.\begin{array}{l} f_{NULL}^{(n)}(x_1, \ldots, x_n) = 0 \\ f_{SUCC}^{(n)}(x_1, \ldots, x_n) = x_1 + 1 \\ f_{PROJ(i)}^{(n)}(x_1, \ldots, x_n) = \begin{cases} x_i & \text{falls } 1 \leq i \leq n \\ 0 & \text{sonst} \end{cases} \end{array}\right\} \text{Grundfunktionen}$$

$$f_{KOMP(G, H_1, \ldots, H_m)}^{(n)}(x_1, \ldots, x_n) = f_G^{(m)} \circ (f_{H_1}^{(n)}, \ldots, f_{H_m}^{(n)})(x_1, \ldots, x_n)$$

$f_{REK(G,H)}^{(n)} = R(f_G^{(n)}, f_H^{(n+1)})$, d. h.

$f_{REK(G,H)}^{(n)}(x_1, \ldots, x_{n-1}, 0) = f_G^{(n)}(x_1, \ldots, x_{n-1}, 0)$ und

$$f_{REK(G,H)}^{(n)}(x_1, \ldots, x_{n-1}, y+1) = f_H^{(n+1)}(x_1, \ldots, x_{n-1}, f_{REK(G,H)}^{(n)}(x_1, \ldots, x_{n-1}, y), y)$$

Primitiv rekursive Funktionen

Die Menge aller Funktionen $f : \mathbb{N}^n \to \mathbb{N}$ $(n > 0)$, für die $f = f_\pi^{(n)}$ mit einem primitiv rekursiven Ausdruck π gilt, heißt die Menge der **primitiv rekursiven Funktionen**.

Bezeichnung $\mathcal{P}(\mathbb{N})$

6.4 Beispiel Folgende Funktionen sind primitiv rekursiv.

- Konstante Funktionen beliebiger Stelligkeit:
  $a \in \mathbb{N} \quad c_a^{(n)} : \mathbb{N}^n \to \mathbb{N} \quad c_a^{(n)}(x_1, \ldots, x_n) = a$ (total)

Zeige $c_a^{(n)} = f_\pi^{(n)}$ für geeignetes π:

$a = 0$ so klar, wähle $\pi = NULL$

$a = 1 \quad f_{KOMP(SUCC,NULL)}^{(n)}(\vec{x}) = f_{SUCC}^{(1)}(f_{NULL}^{(n)}(\vec{x})) = 1$

Ind. Schritt:

$a = m$ sei $f_{\pi_a}^{(n)}(\vec{x}) = a$

$a = m+1 \; f_{KOMP(SUCC,\pi_a)}^{(n)}(\vec{x}) = f_{SUCC}^{(1)}(f_{\pi_a}^{(n)}(\vec{x})) = m+1$

d. h. $\pi_a =$ KOMP(SUCC,KOMP(SUCC,... KOMP(SUCC,NULL) ...))
a-mal $KOMP(SUCC, \ldots)$

Beispiel (Forts.)

- Die Vorgänger Funktion auf $\mathbb{N}$ $pred : \mathbb{N} \to \mathbb{N}$ ist primitiv rekursiv:
  $pred(0) = 0$
  $pred(y+1) = y$ (total)
  $pred(0) = f_{NULL}^{(1)}(0)$
  $pred(y+1) = f_{PROJ(2)}^{(2)}(pred(y), y) = y,$
  d. h. mit $PRED = REK(NULL, PROJ(2))$ gilt
  $pred = f_{PRED}^{(1)} = f_{REK(NULL,PROJ(2))}^{(1)}.$

6.5 Lemma

Die Menge der primitiv rekursiven Funktionen ist abgeschlossen gegenüber Komposition und primitiver Rekursion.

Sind $g : \mathbb{N}^m \to \mathbb{N}, h_1, \ldots, h_m : \mathbb{N}^n \to \mathbb{N} \in \mathcal{P}(\mathbb{N})$, so auch $g \circ (h_1, \ldots, h_m) : \mathbb{N}^n \to \mathbb{N} \in \mathcal{P}(\mathbb{N})$.

Sind $g : \mathbb{N}^{n+1} \to \mathbb{N}, h : \mathbb{N}^{n+2} \to \mathbb{N} \in \mathcal{P}(\mathbb{N})$, so auch $R(g, h)$.

Beweis: Seien G und $H_1, \ldots, H_m$ primitiv rekursive Ausdrücke für g und $h_1, \ldots, h_m$. Dann ist $KOMP(G, H_1, \ldots, H_m)$ ein primitiv rekursiver Ausdruck für $g \circ (h_1, \ldots, h_m)$.

Analog ist $REK(G, H)$ primitiv rekursiver Ausdruck für $R(g, h)$, falls G, H primitiv rekursive Ausdrucke für g bzw. h sind.

Primitiv rekursive Funktionen (Fort.)

Die Menge der primitiv rekursiven Funktionen $\mathcal{P}(\mathbb{N})$ ist also charakterisiert als die kleinste Menge von Funktionen $f : \mathbb{N}^n \to \mathbb{N}$ $(n > 0)$ die die Grundfunktionen enthält und abgeschlossen ist gegenüber Komposition und primitiver Rekursion.

In der Literatur findet man oft die Betrachtung von Funktionen $f : \mathbb{N}^n \to \mathbb{N}^m$ $(n, m > 0)$. Diese lassen sich über die **Parallelausführung** $f := \langle g, h \rangle$ von $g : \mathbb{N}^n \to \mathbb{N}^s, h : \mathbb{N}^n \to \mathbb{N}^t$ die erklärt ist durch

$$f : \mathbb{N}^n \to \mathbb{N}^{s+t} \qquad \langle g, h \rangle(x) = (g(x), h(x))$$

aus den obigen Funktionen gewinnen.

Offenbar sind die primitiv rekursiven Ausdrücke sehr einfache Programme. Sie sind aufgebaut aus $NULL, SUCC, PROJ(i)$ und den variadischen Operator $KOMP$ und den binären Operator REK. Die Interpretation der atomaren Ausdrücke durch die Grundfunktionen und der Operatoren durch die offensichtlich „effektiven" Operationen Komposition und primitive Rekursion machen deutlich, dass diese Programme effektive Berechnungen darstellen. Jedes Programm erlaubt es für jedes $n > 0$ eine Funktion der Stelligkeit n zu berechnen. D. h. ein Programm berechnet unendlich viele Funktionen.

Beachte: Ein primitiv rekursiver Ausdruck stellt stets für jedes n eine Funktion dar. Diese können recht unterschiedlich sein. Siehe z. B. $f_{PROJ(i)}^{(n)}$. Welche Funktion ist $f_{REK(NULL,PROJ(2))}^{(2)}$?

Nachweis von Eigenschaften primitiv rekursiver Ausdrücke oder primitiv rekursiver Funktionen

Erneut: Induktion über Aufbau der primitiv rekursiven Ausdrücke (strukturelle Induktion) bzw. für die Menge $\mathcal{P}(\mathbb{N})$ die sogenannte **Induktion über den Aufbau**: Zeige die Eigenschaft gilt für die Grundfunktionen und die Eigenschaft bleibt erhalten bei Komposition und primitiver Rekursion.

6.6 Lemma

Jede primitiv rekursive Funktion ist total.

6.7 Beispiel Weitere primitiv rekursive Funktionen

$add : \mathbb{N}^2 \to \mathbb{N} \qquad add(x, y) = x + y$ primitiv rekursiv

$add(x, 0) = f_{PROJ(1)}^{(2)}(x, 0)$

$add(x, y+1) = f_{SUCC}^{(1)}(f_{PROJ(2)}^{(3)}(x, add(x, y), y))$

$ADD \;::\; REK(PROJ(1), KOMP(SUCC, PROJ(2))$

ist ein primitiv rekursiver Ausdruck für add, d. h. $add = f_{ADD}^{(2)}$.

Die Multiplikation $mult : \mathbb{N}^2 \to \mathbb{N}$ mit $mult(x, y) = x \cdot y$ ist primitiv rekursiv:

$mult(x, 0) = f_{NULL}^{(2)}(x, 0)$

$mult(x, y+1) = add(f_{PROJ(1)}^{(3)}(x, mult(x, y), y), f_{PROJ(2)}^{(3)}(x, mult(x, y), y))$

d. h. $REK(NULL, KOMP(ADD, PROJ(1), PROJ(2)))$ repräsentiert folglich $mult$.

Beispiele und Vereinfachungen

Die Funktion $sgn : \mathbb{N} \to \mathbb{N}$ mit

$$sgn(x) = \begin{cases} 0 & x = 0 \\ 1 & \text{sonst} \end{cases}$$

$sgn(0) = f^{(1)}_{NULL}(0)$

$sgn(y+1) = f^{(2)}_{KOMP(SUCC,NULL)}(sgn(y), y)$

d. h. $REK(NULL, KOMP(SUCC, NULL))$ repräsentiert sgn.

Analog die Funktion $\overline{sgn} : \mathbb{N} \to \mathbb{N}$ mit

$$\overline{sgn}(x) = \begin{cases} 1 & x = 0 \\ 0 & \text{sonst} \end{cases}$$

Vereinfachungen: Auflockerung des strengen Schemas der primitiven Rekursion.

- Variablen permutieren, mehrfache Verwendung, oder Nicht-Verwendung von Variablen.
- Weitere Abschlusseigenschaften.
- Verwendung bereits als primitiv rekursiv nachgewiesener Funktionen.

Vereinfachungen

6.8 Lemma

Sei $g : \mathbb{N}^m \to \mathbb{N}$ primitiv rekursiv, $m \geq n$ und seien $1 \leq i_1 \leq n, \ldots, 1 \leq i_m \leq n$ Indizes. Dann ist auch die Funktion $h : \mathbb{N}^n \to \mathbb{N}$ mit

$$h(x_1, \ldots, x_n) = g(x_{i_1}, \ldots, x_{i_m})$$

primitiv rekursiv.

Beweis: Es gilt

$h(x_1, \ldots, x_n) =$
$g(f^{(n)}_{PROJ(i_1)}(x_1, \ldots, x_n), \ldots, f^{(n)}_{PROJ(i_m)}(x_1, \ldots, x_n))$

6.9 Beispiel Es genügt in Zukunft, den Nachweis der primitiven Rekursion einer Funktion auf der Basis einer Rekursionsgleichung wie bei den folgenden Funktionen zu führen.

- Nicht negative Differenz: $\dot{-} : \mathbb{N}^2 \to \mathbb{N}$
  $x \dot{-} 0 = x$
  $x \dot{-} (y+1) = pred(x \dot{-} y)$
- Fakultät $fac : \mathbb{N} \to \mathbb{N}$
  $fac(0) = 1 \; (= f^{(1)}_{KOMP(SUCC,NULL)}(0))$
  $fac(y+1) = fac(y) \cdot (y+1)$
  $(= f^{(2)}_{KOMP(MULT(PROJ(1),KOMP(SUCC,PROJ(2)))}(fac(y), y))$

Weitere Abschlusseigenschaften

- Insbesondere ist die Funktion $|x - y| : \mathbb{N}^2 \to \mathbb{N}$, mit

  $$|x - y| \;=\; (x \dot{-} y) + (y \dot{-} x)$$

  primitiv rekursiv.
- Einfache Fallunterscheidung. Oft werden Funktionen nur in bestimmten Bereichen benötigt. Sei etwa $h : \mathbb{N}^2 \to \mathbb{N}$ primitiv rekursiv, dann ist auch die Funktion
  $$F(x, y) = \begin{cases} h(x, y) & \text{falls } x > y \\ 0 & \text{sonst} \end{cases}$$ primitiv rekursiv.
  Es ist $F(x, y) = sgn(x \dot{-} y) \cdot h(x, y)$
  Später werden wir allgemeinere Formen der Fallunterscheidung kennenlernen.

6.10 Lemma Abschluss von $\mathcal{P}(\mathbb{N})$ gegenüber Iteration.

Sei $f : \mathbb{N} \to \mathbb{N}$ primitiv rekursiv, dann ist auch $g : \mathbb{N}^2 \to \mathbb{N}$ mit $g(x, t) = f^t(x)$ primitiv rekursiv.

Beweis: Es ist

$g(x, 0) = x$
$g(x, t+1) = f(g(x, t))$

Frage: Lässt sich jede „Rekursionsgleichung“ durch primitive Rekursion simulieren?

Andere Rekursionsformate

- Das folgende Format ist erlaubt:
  $f(0, y) = g(y)$
  $f(x+1, y) = h(x, f(x, y), y)$
  Mit primitiv rekursiven Funktionen g, h. Dann ist f auch primitiv rekursiv (Argumente vertauscht).
- Hingegen ist die alternative Definition der **Iteration**:
  $g(x, 0) = x$
  $g(x, t+1) = g(f(x), t)$
  nicht vom Format einer primitiven Rekursion, da der Parameter $f(x)$ statt x in der Rekursion verwandt wird.
  Wir werden gleich zeigen, dass auch dieses Format erlaubt ist.
- Allerdings ist die **Ackermannfunktion** $A : \mathbb{N}^2 \to \mathbb{N}$, die durch folgende Rekursionsgleichung definiert wird
  $A(0, y) = y + 1$
  $A(x+1, 0) = A(x, 1)$
  $A(x+1, y+1) = A(x, A(x+1, y))$
  keine primitiv rekursive Funktion (Beweis später).

A ist eine totale Funktion, die sehr schnell wächst. Überzeugen Sie sich!

Andere Rekursionsformate (Fort.)

6.11 Lemma

Seien $g : \mathbb{N} \to \mathbb{N}$, $h : \mathbb{N}^3 \to \mathbb{N}$, $w : \mathbb{N} \to \mathbb{N}$ primitiv rekursiv und $f : \mathbb{N}^2 \to \mathbb{N}$ durch die Gleichungen

$$\begin{array}{lcl} f(x,0) & = & g(x) \\ f(x,y+1) & = & h(x, f(w(x),y), y) \end{array} \quad \text{definiert.}$$

Dann ist auch f primitiv rekursiv.

Beweis:

Sei $F(t,x,y) = \begin{cases} f(w^{t-y}(x),y) & \text{falls } t \geq y \\ 0 & \text{sonst} \end{cases}$

Es gilt $F(t,x,0) = f(w^t(x),0) = g(w^t(x))$ und für $t \geq y+1$

$$\begin{aligned} F(t,x,y+1) &= f(w^{t-y-1}(x), y+1) \\ &= h(w^{t-y-1}(x), f(w(w^{t-y-1}(x)),y),y) \\ &= h(w^{t-y-1}(x), f(w^{t-y}(x),y),y) \\ &= h(w^{t-y-1}(x), F(t,x,y),y) \end{aligned}$$

Für $t < y+1$ gilt

$$F(t,x,y+1) = 0$$

F ist also primitiv rekursiv.
Wegen $f(x,y) = F(y,x,y)$ ist auch f primitiv rekursiv.

Fallunterscheidung

6.12 Lemma Abschluss gegenüber Fallunterscheidung

Seien $g_i : \mathbb{N}^n \to \mathbb{N}$, $h_i : \mathbb{N}^n \to \mathbb{N}$ mit h_i totale Funktionen für $i = 1, \ldots, r$, so dass es zu jedem $\vec{x} \in \mathbb{N}^n$ es genau ein i gibt mit $h_i(\vec{x}) = 0$. Die Fallunterscheidung mit den Funktionen g_i, h_i ist die Funktion

$f = FU(g_i, h_i \;\; i = 1, \ldots, k)$, mit

$$f(\vec{x}) = \begin{cases} g_1(\vec{x}) & \text{falls } h_1(\vec{x}) = 0 \\ \vdots & \\ g_k(\vec{x}) & \text{falls } h_k(\vec{x}) = 0 \end{cases}$$

Offenbar gilt:
$f(\vec{x}) = \overline{sgn}(h_1(\vec{x})) \cdot g_1(\vec{x}) + \cdots + \overline{sgn}(h_k(\vec{x})) \cdot g_k(\vec{x})$,
d. h. sind $g_i, h_i \in \mathcal{P}(\mathbb{N})$, so ist auch $f \in \mathcal{P}(\mathbb{N})$.

Die Fallunterscheidung wird meistens mit $k = 2$ angewendet. Sei h gegeben und $h_1(x) = h(x), h_2(x) = \overline{sgn}(h(x))$. Dann gilt

$$f(x) = \begin{cases} g_1(x) & h_1(x) = 0 \\ g_2(x) & h_2(x) = 0 \end{cases} = \begin{cases} g_1(x) & h(x) = 0 \\ g_2(x) & \text{sonst} \end{cases}$$

Primitiv rekursive Relationen

6.13 Definition

Eine **Relation** $R \subseteq \mathbb{N}^n$ **heißt primitiv rekursiv**, falls ihre charakteristische Funktion $\chi_R \in \mathcal{P}(\mathbb{N})$.

Erinnerung für $\vec{x} \in \mathbb{N}^n$ gilt:

$$\chi_R(\vec{x}) = \begin{cases} 1 & \text{falls } \vec{x} \in R \\ 0 & \text{falls } \vec{x} \notin R \end{cases}$$

6.14 Beispiel

Primitiv rekursive Relationen sind:

- Die Gleichheitsrelation: $= \;\; \subseteq \mathbb{N} \times \mathbb{N}$
  $\chi_=(x,y) = 1 - |x-y|$
- Kleinerrelation: $< \;\; \subseteq \mathbb{N} \times \mathbb{N}$
  $\chi_<(x,y) = sgn(y-x)$
- Kleinergleichrelation: $\leq \;\; \subseteq \mathbb{N} \times \mathbb{N}$
  $\chi_\leq(x,y) = \chi_=(x,y) + \chi_<(x,y)$
- Analog Größerrelation und Größergleichrelation.

Abschlusseigenschaften primitiv rekursiver Relationen

Erinnerung:

Seien $R, S \subseteq \mathbb{N}^n$ Relationen.

Dann

- $\neg R$ **Komplement** von R — $\mathbb{N}^n - R$
- $R \wedge S$ **Durchschnitt** von R und S — $R \cap S$
- $R \vee S$ **Vereinigung** von R und S — $R \cup S$

6.15 Lemma

Sind $R, S \subseteq \mathbb{N}^n$ primitiv rekursiv, so auch $\neg R$, $R \wedge S$, $R \vee S$.

Beweis: Es ist

$\chi_{\neg R} = 1 - \chi_R$, $\chi_{R \wedge S} = \chi_R \cdot \chi_S$ und $R \vee S = \neg(\neg R \wedge \neg S)$.

Frage: Gilt $\chi_{R \vee S} = \chi_R + \chi_S$?

Insbesondere sind $\neq, \leq, >, \geq$ primitiv rekursiv.

Reduzierbarkeit

6.16 Lemma

Seien $S \subseteq \mathbb{N}^n$, $h_i : \mathbb{N}^n \to \mathbb{N}$ $(1 \leq i \leq n)$ primitiv rekursiv. Dann ist auch die Relation $R \subseteq \mathbb{N}^m$ mit

$$R\vec{x} \quad \text{gdw} \quad Sh_1(\vec{x}) \cdots h_n(\vec{x})$$

primitiv rekursiv.

R ist auf S **primitiv rekursiv reduzierbar**, falls es primitiv rekursive Funktionen $h_i : \mathbb{N}^m \to \mathbb{N}$ $(1 \leq i \leq n)$ gibt mit obiger Eigenschaft.

Beweis:

$$\begin{aligned}\chi_R(\vec{x}) &= \chi_S(h_1(\vec{x}), \ldots, h_n(\vec{x})) \\ &= \chi_S \circ (h_1, \ldots, h_n)(\vec{x})\end{aligned}$$

6.17 Beispiel Sei $R \subseteq \mathbb{N}^2$ mit

Rxy gdw x ist ganzzahliger Anteil der Quadratwurzel von y.

Dann ist R primitiv rekursiv

$$Rxy \quad \text{gdw} \quad x \cdot x \leq y \wedge (x+1) \cdot (x+1) > y$$

Wende Lemma an mit

$$Suvw \text{ gdw } u \leq v \wedge w > v$$

$$h_1(x,y) = x \cdot x, h_2(x,y) = y, h_3(x,y) = (x+1) \cdot (x+1)$$

Fallunterscheidung mit primitiv rekursiven Relationen

6.18 Lemma

$R_i \subseteq \mathbb{N}^n$ $1 \leq i \leq m$ sind paarweise disjunkte primitiv rekursive Relationen und $h_1, \ldots, h_{m+1}$ n-stellige Funktionen mit $h_i \in \mathcal{P}(\mathbb{N})$ $i = 1, \ldots, m+1$.

Dann gilt für $f : \mathbb{N}^n \to \mathbb{N}$ mit

$$f(\vec{x}) = \begin{cases} h_1(x) & \text{falls } R_1\vec{x} \\ \vdots & \\ h_m(\vec{x}) & \text{falls } R_m\vec{x} \\ h_{m+1}(\vec{x}) & \text{sonst} \end{cases}$$

$f \in \mathcal{P}(\mathbb{N})$.

Beweis:

$$f(\vec{x}) = \chi_{R_1}(\vec{x}) \cdot h_1(\vec{x}) + \cdots + \chi_{R_m}(x) \cdot h_m(\vec{x}) + (1 - (\chi_{R_1 \vee \cdots \vee R_m}(\vec{x})) \cdot h_{m+1}(\vec{x})$$

6.19 Beispiel Maximum und Minimum von (n)-zwei Zahlen

$$\max(x,y) = \begin{cases} x & \text{falls } x > y \\ y & \text{sonst} \end{cases}$$

Weitere Abschlusseigenschaften von $\mathcal{P}(\mathbb{N})$

6.20 Definition

1. Die **beschränkte Summation** und **beschränkte Multiplikation** mit der Funktion $g : \mathbb{N}^{n+1} \to \mathbb{N}$ sind erklärt durch

$$f, h : \mathbb{N}^{n+1} \to \mathbb{N}$$
$$f(\vec{x}, y) = \sum_{z \leq y} g(\vec{x}, z) \qquad h(\vec{x}, y) = \prod_{z \leq y} g(\vec{x}, z)$$

2. Die **beschränkte Minimierung** mit der Funktion $g : \mathbb{N}^{n+1} \to \mathbb{N}$ ist erklärt durch die Funktion $f : \mathbb{N}^{n+1} \to \mathbb{N}$ mit

$$f(\vec{x}, y) = \begin{cases} u & u \leq y,\ g(\vec{x}, u) = 0 \text{ und } g(\vec{x}, z) > 0 \text{ für } z < u \\ 0 & g(\vec{x}, z) > 0 \text{ für alle } z \leq y \\ \uparrow & \text{es gibt } u \leq y \text{ mit } g(\vec{x}, u)\uparrow,\ g(\vec{x}, z) > 0 \text{ für } z < u \end{cases}$$

Bezeichnung $f(\vec{x}, y) = \mu_{z \leq y}[g(\vec{x}, z) = 0]$

3. Die **beschränkte Minimierung** mit einer Relation $R \subseteq \mathbb{N}^{n+1}$ ist erklärt durch die Funktion $f : \mathbb{N}^{n+1} \to \mathbb{N}$ mit

$$f(\vec{x}, y) = \begin{cases} \text{kleinstes } z \text{ mit } z \leq y \wedge R\vec{x}z & \text{falls } z \text{ existiert} \\ y & \text{sonst} \end{cases}$$

Schreibe $f(\vec{x}, y) = \mu z \leq y . R\vec{x}z$ **f ist total.**

Beispiele

6.21 Beispiel

1. Mit $g(y) = y + 1$ ergibt sich

$$f(y) = \sum_{z \leq y} g(z) = \frac{(y+1) \cdot (y+2)}{2}$$
$$h(y) = \prod_{z \leq y} g(z) = fac(y+1)$$

Mit $g(x, y) = x$ ergibt sich

$$f(x, y) = \sum_{z \leq y} g(x, z) = x \cdot (y+1)$$
$$h(x, y) = \prod_{z \leq y} g(x, z) = x^{y+1}$$

2. Sei $f(x, y) = \mu_{z \leq y}[g(x, z) = 0]$.
Wir betrachten zwei Funktionen g:

$$g(x, y) = x - y \quad \text{liefert} \quad f(x, y) = \begin{cases} 0 & x > y \\ x & x \leq y \end{cases}$$

$$g(x, y) = \begin{cases} x \cdot y & x + y > 0 \\ \uparrow & \text{sonst} \end{cases} \quad \text{liefert} \quad f(x, y) = \begin{cases} 0 & x > 0 \\ \uparrow & x = 0 \end{cases}$$

3. Beschränkte Minimierung mit einer Relation
   - $f(x) =$ ganzzahliger Anteil der Quadratwurzel von x
     $= \mu y \leq x.(y \cdot y \leq x \wedge (y+1) \cdot (y+1) > x)$
   - $x \text{ div } y = \begin{cases} \mu t \leq x.(t+1) \cdot y > x & \text{falls } y > 0 \\ 0 & \text{sonst} \end{cases}$
   - $x \bmod y = \begin{cases} x - (x \text{ div } y) \cdot y & \text{falls } y > 0 \\ 0 & \text{sonst} \end{cases}$

6.22 Lemma $\mathcal{P}(\mathbb{N})$ ist abgeschlossen bezüglich beschränkter Summation, beschränkter Multiplikation und den beschränkten Minimierungen.

Beweis

1. Sei $g \in \mathcal{P}(\mathbb{N})$; zu zeigen ist $f, h \in \mathcal{P}(\mathbb{N})$, wobei

$$f(x,y) = \sum_{z \leq y} g(x,z)$$
$$h(x,y) = \prod_{z \leq y} g(x,z)$$

Es gilt

$$f(x,0) = g(x,0) \quad f(x,y+1) = f(x,y) + g(x,y+1)$$
$$h(x,0) = g(x,0) \quad h(x,y+1) = h(x,y) \cdot g(x,y+1)$$

also

$$f = R(g, h_1) \quad \text{mit} \quad h_1(x,y,z) = y + g(x, z+1)$$
$$h = R(g, h_2) \quad \text{mit} \quad h_2(x,y,z) = y \cdot g(x, z+1)$$

Es gilt $g_i, h_i \in \mathcal{P}(\mathbb{N})$, also $f, h \in \mathcal{P}(\mathbb{N})$

2. Sei $g \in \mathcal{P}(\mathbb{N})$ und $f(x,y) = \mu_{z \leq y}[g(x,z) = 0]$. Für den Nachweis, dass $f \in \mathcal{P}(\mathbb{N})$ gilt, muss f in der funktionalen Programmiersprache zu $\mathcal{P}(\mathbb{N})$ programmiert werden. Die Idee hierzu spiegelt die natürliche Berechnung von $f(x,y)$ wider: Wir bilden die Funktion

$$f_0(x,z) = \begin{cases} 1 & g(x,u) > 0 \text{ für alle } u \leq z \\ 0 & \text{sonst} \end{cases}$$

und summieren die Werte $f_0(x,z)$ für $z = 0, 1, \ldots, y$ auf. Setze also

$$f_0(x,z) = sgn\left(\prod_{u \leq z} g(x,u)\right)$$

dann gilt $f_0 \in \mathcal{P}(\mathbb{N})$ nach 1. und eine kurze Überlegung zeigt

$$f(x,y) = \begin{cases} \sum_{z \leq y} f_0(x,z) & f_0(x,y) = 0 \\ 0 & \overline{sgn}(f_0(x,y)) = 0 \end{cases}$$

Also gilt $f \in \mathcal{P}(\mathbb{N})$ nach 1. und Lemma 6.12.

Der Beweis zur beschränkten Minimierung soll noch an folgendem Beispiel verdeutlicht werden. Sei

$$g(x,y) = x \dot{-} y^2$$

Die folgende Tabelle verdeutlicht die Berechnung von $f(5,y)$

y	$g(5,y)$	$f_0(5,y)$	$f(5,y)$
0	5	1	0
1	4	1	0
2	1	1	0
3	0	0	3
4	0	0	3

3. Ist R primitiv rekursiv und f durch beschränkte Minimierung aus R wie eben definiert, so ist $f \in \mathcal{P}(\mathbb{N})$. Es gilt nämlich
$f(\vec{x}, 0) = 0$
$$f(\vec{x}, y+1) = \begin{cases} f(\vec{x}, y) & \text{falls } R\vec{x}f(\vec{x},y) \\ y+1 & \text{sonst} \end{cases}$$

Beschränkte Quantifizierung

6.23 Definition

Sei $R \subseteq \mathbb{N}^{n+1}$. Definiere Relationen $T \subseteq \mathbb{N}^{n+1}$ und $S \subseteq \mathbb{N}^{n+1}$ durch **beschränkte All-/Existenz-Quantifizierung** durch

$S\vec{x}b$ gdw es gibt $y \leq b$ mit $R\vec{x}y$ (Schreibe $\exists y \leq b.R\vec{x}y$)

$T\vec{x}b$ gdw für alle $y \leq b$ gilt $R\vec{x}y$ (Schreibe $\forall y \leq b.R\vec{x}y$)

6.24 Lemma

Die primitiv rekursiven Relationen sind abgeschlossen gegenüber beschränkter Quantifizierung.

Beweis: Sei $S\vec{x}b$ gdw $\exists y \leq b.R\vec{x}y$

Dann gilt $\chi_S(\vec{x}, 0) = \chi_R(\vec{x}, 0)$
$\chi_S(\vec{x}, b+1) = \max(\chi_R(\vec{x}, b+1), \chi_S(\vec{x}, b))$

Wegen $T\vec{x}b$ gdw $\neg\exists y \leq b.\neg R\vec{x}y$ folgt die Behauptung.

6.25 Beispiel

Die Teilbarkeitsrelation $\mid$ ist primitiv rekursiv.

- $x \mid y$ gdw $\exists t \leq y.t \cdot x = y$
  $Rxyz$ gdw $z \cdot x = y$
  Dann ist $x \mid y$ gdw $Sxyy$ gdw $\exists t \leq y.Rxyt$ gdw
  $\exists t \leq y.t \cdot x = y$
- $\{p : p$ ist Primzahl$\}$ ist primitiv rekursiv.

Primitiv rekursive Codier- und Decodierfunktionen

Paarungsfunktionen, Codierung von Zahlenfolgen

6.26 Definition

Die **Cauchysche Paarungsfunktion** $\langle \cdot, \cdot \rangle : \mathbb{N}^2 \to \mathbb{N}$ wird definiert durch

$$\langle x, y \rangle = ((x+y)(x+y+1) \text{ div } 2) + y$$

Sie ist primitiv rekursiv und bijektiv.

(x, y)	$(0,0)$	$(1,0)$	$(0,1)$	$(2,0)$	$(1,1)$	$(0,2)\cdots$
$\langle x, y \rangle$	0	1	2	3	4	$5\cdots$

Abzählen auf geeigneten Diagonalen: $x + y$ konstant.
(x, y) kommt auf der Diagonalen $x + y + 1$ vor.

- $x + y$ komplett gefüllte Diagonalen: $1 + 2 + \cdots + (x+y) = (x+y)(x+y+1) \text{ div } 2$
- In der Diagonalen, in der (x, y) steht, kommen noch y viele Punkte vor dem Punkt (x, y).

Also ist $\langle \cdot, \cdot \rangle : \mathbb{N}^2 \to \mathbb{N}$ bijektiv

Definiere folgende **Umkehrfunktionen** first, rest: $\mathbb{N} \to \mathbb{N}$ mit $\langle \text{first}(z), \text{rest}(z) \rangle = z$ und

$\text{first}(\langle x, y \rangle) = x$
$\text{rest}(\langle x, y \rangle) = y$

Paarungsfunktionen Codierung von Zahlenfolgen

6.27 Lemma Die Funktionen $\langle \cdot, \cdot \rangle : \mathbb{N}^2 \to \mathbb{N}$ und first, rest: $\mathbb{N} \to \mathbb{N}$ sind primitiv rekursiv.

Beweis:

Nach Def. von $\langle x, y \rangle = z$, folgt $x \leq z$ und $y \leq z$.

Dann

$$\text{first}(z) = \mu x \leq z . \exists y \leq z . \langle x, y \rangle = z$$

$$\text{rest}(z) = \mu y \leq z . \exists x \leq z . \langle x, y \rangle = z$$

Nach Lemma 6.24 und Lemma 6.22 sind first und rest primitiv rekursiv.

Codierung endlicher Zahlenfolgen

6.28 Definition Sei $x_0, \ldots, x_n$ Zahlenfolge. Die **Codierung** $[x_0, \ldots, x_n] \in \mathbb{N}$ wird induktiv über n definiert durch

- $[\,] = 0$ (Codierung der leeren Folge)
- $[x_0, \ldots, x_n] = \langle x_0, [x_1, \ldots, x_n] \rangle$

Die **Folgenzugriffsfunktion** get : $\mathbb{N}^2 \to \mathbb{N}$ sei definiert durch

- $\text{get}(z, 0) = \text{first}(z)$
- $\text{get}(z, i+1) = \text{get}(\text{rest}(z), i)$

Die Folgenzugriffsfunktion liegt in $\mathcal{P}(\mathbb{N})$.

Wertverlaufsrekursion

Beachte: Die Folgenkodierung ist eindeutig bis auf Nullen am rechten Folgenende, d. h. es gilt $[x_0, \ldots, x_n] = [x_0, \ldots, x_n, 0, \ldots, 0]$. Die Elemente x_0 bis x_n mit $x_n \neq 0$ können eindeutig bestimmt werden.

6.29 Lemma Wertverlaufsrekursion

Sind $g : \mathbb{N}^{n+1} \to \mathbb{N}$ und $h : \mathbb{N}^{n+2} \to \mathbb{N}$ primitiv rekursiv, dann auch $f : \mathbb{N}^{n+1} \to \mathbb{N}$ mit

- $f(\vec{x}, 0) = g(\vec{x}, 0)$
- $f(\vec{x}, y+1) = h(\vec{x}, [f(\vec{x}, y), \ldots, f(\vec{x}, 0)], y)$

Hier greift die Rekursion auf beliebig viele Vorgängerwerte zurück.

Beweis:

Die Hilfsfunktion $F(\vec{x}, y) = [f(\vec{x}, y), \ldots, f(\vec{x}, 0)]$ ist primitiv rekursiv, da

- $F(\vec{x}, 0) = \langle g(\vec{x}, 0), 0 \rangle$
- $F(\vec{x}, y+1) = \langle h(\vec{x}, F(\vec{x}, y), y), F(\vec{x}, y) \rangle$

und somit ist

$f(\vec{x}, y) = \text{get}(F(\vec{x}, y), 0) = \text{first}(F(\vec{x}, y))$

primitiv rekursiv.

Beispiel

6.30 Beispiel

1. Sei f definiert durch
   - $f(x, 0) = 1$
   - $f(x, y+1) = f(x, y \text{ div } 2) + 1$

   Dann ist $f \in \mathcal{P}(\mathbb{N})$: Wähle im obigen Lemma $h(x, u, y) = succ(\text{get}(u, y \text{ div } 2))$.
2. **Folgenverkettung $*$: $\mathbb{N}^2 \to \mathbb{N}$**
   (nicht Multiplikation von Zahlen!)
   $[x_0, \ldots, x_n] * [y_0, \ldots, y_m] = [x_0, \ldots, x_n, y_0, \ldots, y_m]$
   wobei $x_n \neq 0$ (falls $n > 0$).

   Behauptung: $*$ ist primitiv rekursiv.
   Beweis: Betrachte

   $$0 * v = v$$
   $$(u+1) * v = \langle \text{first}(u+1), \text{rest}(u+1) * v \rangle$$

   wegen $\text{rest}(u+1) \leq u$ folgt die Behauptung durch Wertverlaufsrekursion (hier wird die Voraussetzung $x_n \neq 0$ benötigt).
   Es gilt nämlich:

   $$[i] = \langle i, 0 \rangle = \underbrace{\frac{i \cdot (i+1)}{2}}_{\geq i} > 0 \quad (i \neq 0)$$

Beispiel (Fort.)

$$\underbrace{[x_0, \ldots, x_n]}_{u+1, x_n \neq 0} = \langle x_0, [x_1, \ldots, x_n] \rangle =$$

$$\underbrace{\frac{(x_0 + [x_1 \ldots x_n])(x_0 + [x_1 \ldots x_n] + 1)}{2}}_{\geq 1} + [x_1, \ldots, x_n]$$

D. h.

- $\text{first}(u + 1) = x_0 < u$ (Listenlänge ≥ 2)
- $\text{rest}(u + 1) = [x_1, \ldots, x_n] \leq u$

Wir werden diese Funktionen noch benötigen, und zwar bei der Arithmetisierung der While-Programme. Programme sind Folgen von Anweisungen. Wir werden Anweisungen durch Zahlen codieren und dementsprechend Programme durch die Codierungen der Zahlenfolgen darstellen. Um die Interpreterfunktion zu simulieren benötigen wir die Folgenzugriffsfunktion und die Folgenverkettung.

Simultane Rekursion

6.31 Definition

Eine Funktion $f = (f_1, \ldots, f_m) : \mathbb{N}^n \to \mathbb{N}^m$ heißt primitiv rekursiv, falls jede Komponentenfunktion $f_i : \mathbb{N}^n \to \mathbb{N}$ $i = 1, \ldots, m$ primitiv rekursiv ist.

6.32 Lemma Simultane Rekursion

Sind $g : \mathbb{N}^{n+1} \to \mathbb{N}^m$ und $h : \mathbb{N}^{n+m+1} \to \mathbb{N}^m$ primitiv rekursiv, dann ist auch die Funktion $f : \mathbb{N}^{n+1} \to \mathbb{N}^m$ mit

- $f(\vec{x}, 0) \rightrightarrows g(\vec{x}, 0)$
- $f(\vec{x}, y + 1) \rightrightarrows h(\vec{x}, f(\vec{x}, y), y)$

($\rightrightarrows$ ist als Gleichheit von Vektoren zu lesen).

Beweis:

Die Hilfsfunktion

$$F(\vec{x}, y) = [f_1(\vec{x}, y), \ldots, f_m(\vec{x}, y)]$$

ist primitiv rekursiv.

Simultane Rekursion (Forts.)

Da $F(\vec{x}, 0) = [g_1(\vec{x}, 0), \ldots, g_m(\vec{x}, 0)]$

und

$F(\vec{x}, y + 1) = [h_1(\vec{x}, F(\vec{x}, y), y), \ldots, h_m(\vec{x}, F(\vec{x}, y), y)]$

und für festes m die Funktion $[k_1, \ldots, k_m] : \mathbb{N}^n \to \mathbb{N}$ primitiv rekursiv ist für $k_i : \mathbb{N}^n \to \mathbb{N}, k_i \in \mathcal{P}(\mathbb{N})$.

Aus $f_i(\vec{x}, y) = \text{get}(F(\vec{x}, y), i)$ $i = 1, \ldots, m$ folgt die Behauptung.

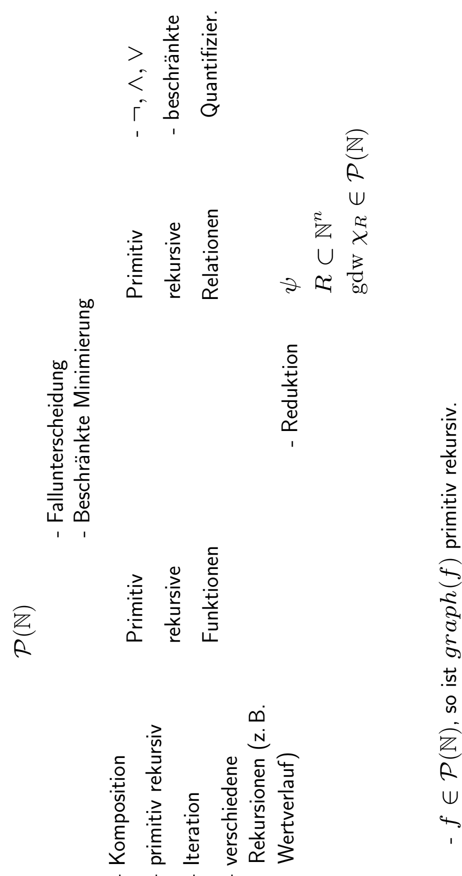
6.33 Beispiel

Sei $paar : \mathbb{N} \to \mathbb{N}^2$ die Umkehrung der Cauchyschen Paarungsfunktion. Es gilt

$paar(0) = (0, 0)$ und

$$paar(n+1) = \begin{cases} (y + 1, 0) & \text{falls } paar(n) = (0, y) \\ (x - 1, y + 1) & \text{sonst, wobei } paar(n) = (x, y) \end{cases}$$

Zeige: $paar(n) = (\text{first}(n), \text{rest}(n))$.



Sind alle totalen „berechenbaren“ Funktionen primitiv rekursiv?

- Numeriere effektiv alle primitiv rekursiven Ausdrücke, z. B. lexikographische Anordnung der Wörter über dem Alphabet $\{NULL, SUCC, PROJ, KOMP, REK, (,), 0, \ldots, 9, ,\}$
- Sei π_i der i-te primitiv rekursive Ausdruck in dieser Nummerierung (Länge + Lexikographisch).
- Definiere Diagonalfunktion $d : \mathbb{N} \to \mathbb{N}$ durch

$$d(n) = f_{\pi_n}^{(1)}(n) + 1$$

Dann ist d total und „effektiv berechenbar“.

6.34 Satz Diagonalschluss für primitiv rekursive Funktionen
Die Funktion d ist nicht primitiv rekursiv.

Beweis:
Wäre d primitiv rekursiv, dann gäbe es einen primitiv rekursiven Ausdruck π_n, so dass $d = f_{\pi_n}^{(1)}$. Dann aber
$f_{\pi_n}^{(1)}(n) = d(n) = f_{\pi_n}^{(1)}(n) + 1$ ↯

Universelle Funktionen für $\mathcal{P}(\mathbb{N})$

6.35 Folgerung Aufzählung der einstelligen Funktionen in $\mathcal{P}(\mathbb{N})$
Es gibt keine **universelle** primitiv rekursive Funktion $f : \mathbb{N}^2 \to \mathbb{N}$ für $\mathcal{P}(\mathbb{N})$, d. h. eine Funktion $f \in \mathcal{P}(\mathbb{N})$ mit der Eigenschaft
$\forall g \in \mathcal{P}(\mathbb{N}) \exists i \in \mathbb{N} : f(i, \cdot) = g(\cdot)$ (i heißt **Index** für g).

Beweis: Sei $f : \mathbb{N}^2 \to \mathbb{N}$ universelle Funktion für $\mathcal{P}(\mathbb{N})$.
Sei $d'(x) = f(x, x) + 1$.
Angenommen $f \in \mathcal{P}(\mathbb{N})$. Dann $d' \in \mathcal{P}(\mathbb{N})$ und es gibt $i \in \mathbb{N}$ mit $d'(x) = f(i, x)$ für $x \in \mathbb{N}$.
Insbesondere $f(i, i) = d'(i) = f(i, i) + 1$ ↯

Gesucht: Konkrete Funktionen, die nicht primitiv rekursiv sind.

6.36 Beispiel Ackermann Funktion

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

A ist eine totale Funktion.
A ist „eine Art“ universelle Funktion für $\mathcal{P}(\mathbb{N})$.

Eigenschaften der Ackermann Funktion

1. $A(x, y) > y$.
2. $A(x, y_1) > A(x, y_2)$, falls $y_1 > y_2$ (Monotonie).
3. $A(x + 1, y) \geq A(x, y + 1)$.
4. $A(x + 2, y) > A(x, 2y)$.
5. A ist total. (wie zeigt man dies!)
6. A ist effektiv berechenbar.

```
proc AKM(in x, y : nat, out z : nat)
  {true}call AKM(x, y, z){z = A(x, y)}
  begin
    if x = 0
      then z := y + 1;
       else
             if y = 0
                then call AKM(pred(x), 1, z);
                 else
                      call AKM(x, pred(y), z1);
                      call AKM(pred(x), z1, z);
             end;
    end;
  end.
```

Die Prozedur „sollte“ korrekt kommentiert sein! (Nachweis!) über $(Nat, \ldots, A)$.

Eigenschaften der Ackermann Funktion (Forts.)

7. Für jede primitiv rekursive Funktion $f : \mathbb{N}^n \to \mathbb{N}$ gibt es ein $r \in \mathbb{N}$, so dass

$$f(\vec{x}) \leq A(r, \max\{x_1, \ldots, x_n\})$$

Beweis: Induktion über Aufbau von $\mathcal{P}(\mathbb{N})$.

- Grundfunktionen: wähle $r = 0$
  $$\left.\begin{matrix} 0 \\ x_1 + 1 \\ x_i \end{matrix}\right. \leq A(0, \max\{x_1, \ldots, x_n\}) = \max\{x_1, \ldots, x_n\} + 1$$
- Sei $f = g \circ (h_1, \ldots, h_m)$ und $y = \max\{x_1, \ldots, x_n\}$.
  wobei für g durch r, h_i durch s_i beschränkt sei, d. h.
  $g(\vec{x}) \leq A(r, y)$, $h_i(\vec{x}) \leq A(s_i, y)$.

$$\begin{aligned} f(\vec{x}) &= g \circ (h_1, \ldots, h_m)(\vec{x}) = g(h_1(\vec{x}), \ldots, h_m(\vec{x})) = \\ &\leq A(r, \max\{h_1(\vec{x}), \ldots, h_m(\vec{x})\}) \\ &\overset{(2)}{\leq} A(r, \max\{A(s_1, y), \ldots, A(s_m, y)\}) \\ &\overset{(3)}{=} A(r, A(\max\{s_1, \ldots, s_m\}, y)) \\ &\overset{(2)(3)}{\leq} A(\max\{s_1, \ldots, s_m, r\}, A(\max\{s_1, \ldots, s_m, r\} \\ &\qquad\qquad +1, y)) \\ &\overset{def}{=} A(\max\{s_1, \ldots, s_m, r\} + 1, y + 1) \\ &\overset{(3)}{\leq} A(\max\{s_1, \ldots, s_m, r\} + 2, y) \end{aligned}$$

Wähle $\max\{s_1, \ldots, s_m, r\} + 2$ als Konstante für f.

Eigenschaften der Ackermann Funktion (Forts.)

- Sei $f = R(g, h)$.
  g sei durch r, h durch s beschränkt.
  Hilfsbehauptung:
  $f(\vec{x}, z) \leq A(\max\{r, s\} + 1, y + z)$ mit
  $y = \max\{x_1, \ldots, x_n\}$.
  Beweis: Induktion über z. (einfach).
  Setze $p := \max\{r, s\} + 1$. Dann

$$A(p, y+z) \overset{(2)}{\leq} A(p, 2\max\{y, z\}) \overset{(4)}{\leq} A(p+2, \max\{y, z\})$$

  $p + 2$ kann als Konstante für f gewählt werden.

8. Die Ackermannfunktion ist nicht primitiv rekursiv, d. h.
   $A \notin \mathcal{P}(\mathbb{N})$.

Beweis:

Angenommen $A \in \mathcal{P}(\mathbb{N})$. Dann ist f mit $f(x, y) = A(x, y) + 1$ primitiv rekursiv. Es gibt (wegen 7.) ein $r \in \mathbb{N}$ mit
$f(x, y) = A(x, y) + 1 \leq A(r, \max\{x, y\})$.
Insbesondere für $x = y = r$

$$f(r, r) = A(r, r) + 1 \leq A(r, r) \qquad ↯$$

6.2 μ-Rekursive Funktionen $\mathcal{R}_p(\mathbb{N})$ (partiell rekursive Funktionen)

- Primitiv rekursive Funktionen $\mathcal{P}(\mathbb{N})$
  Die bisher betrachteten Operationen auf Funktionen bilden totale Funktionen wieder in totalen Funktionen ab.
  z. B. Komposition, primitive Rekursion, Fallunterscheidung, beschränkte Minimierung, Wertverlaufsrekursion.
- Diagonalisierung liefert für jede effektiv aufzählbare Menge von totalen Funktionen eine Diagonalfunktion d, die total, effektiv und nicht in der Menge liegt.
  Will man alle effektiv berechenbaren Funktionen charakterisieren, so benötigt man partielle Funktionen.

Welche Operationen führen zu partiellen Funktionen?

Vergleichbar dazu: While Konstrukt der Programmiersprache.

Idee: Unbeschränkte Minimierung: Suchen nach „erster Nullstelle".

Hat eine Funktion keine Nullstelle, so ist eine solche Suche nicht erfolgreich sein und führt somit zur Partialität.

Minimierung

6.37 Definition Minimierungsoperator

Sei $f : \mathbb{N}^{n+1} \to \mathbb{N}$ eine Funktion $n \geq 1$.

$g : \mathbb{N}^n \to \mathbb{N}$ entsteht aus f durch **Minimierung**, falls gilt

$$g(\vec{x}) \downarrow \ \text{gdw} \ \exists y \ (f(\vec{x}, y) \downarrow \wedge f(\vec{x}, y) = 0 \ \wedge \\ \forall z \ (z < y \to (f(\vec{x}, z) \downarrow \wedge f(\vec{x}, z) > 0)))$$

In diesem Fall ist $g(\vec{x})$ als das eindeutig bestimmte y definiert.

Schreibe: $g(\vec{x}) = \mu y. f(\vec{x}, y) = 0$,
(kleinste y, so dass $f(\vec{x}, y)$ null ist)

6.38 Beispiel

- $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$g(x) = \mu y.(x + y = 0) = \begin{cases} 0 & x = 0 \\ \uparrow & \text{sonst} \end{cases}$$

- $* : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$
  $g(x) = \mu y.(x * y = 0) = 0$

Beachte:

Die Minimierung wird auf Funktionen mit Stelligkeit ≥ 2 angewendet.

Offenbar ist $\mathcal{P}(\mathbb{N})$ **nicht** abgeschlossen gegen Minimierung
(siehe $\mu y.x + y = 0$).

μ-rekursive Ausdrücke und Funktionen: Die Klasse $\mathcal{R}_p(\mathbb{N})$

6.39 Definition Erweiterung der Ausdrücke

- **Syntax:** **μ-rekursive Ausdrücke** entstehen durch Hinzunahme der Regel

$$\frac{G}{MIN(G)}$$

  zum Kalkül der primitiv rekursiven Ausdrücke.
- **Semantik:** Jeder μ-rekursive Ausdruck π repräsentiert für beliebige Stelligkeit $n \geq 1$ eine Funktion $f_\pi^{(n)} : \mathbb{N}^n \to \mathbb{N}$ durch:
  $f_{MIN(G)}^{(n)}(x_1, \ldots, x_n) = \mu y. f_G^{(n+1)}(x_1, \ldots, x_n, y) = 0$,
  falls $\pi = MIN(G)$.
  Sonst bleibt $f_\pi^{(n)}$ wie im primitiv rekursiven Fall unter Beobachtung, dass nun partielle Funktionen vorkommen dürfen, d. h. ist beim rekursiven Auswerten eine Teilfunktion an der betrachteten Stelle nicht definiert, so ist auch die gesamte Funktion an der betrachteten Stelle nicht definiert.
- Eine Funktion $f : \mathbb{N}^n \to \mathbb{N}$ heißt μ-**rekursiv** (oder **partiell rekursiv**), falls $f = f_\pi^{(n)}$ für einen μ-rekursiven Ausdruck π gilt.

  Sei $\mathcal{R}_p(\mathbb{N})$ die Menge der partiell rekursiven Funktionen.

Beispiel

6.40 Beispiel

Sei $\pi = REK(SUCC, MIN(PROJ(1)))$.

Bestimme $f_\pi^{(2)} : \mathbb{N}^2 \to \mathbb{N}$.

$$\begin{aligned} f_\pi^{(2)}(x,0) &= f_{SUCC}^{(2)}(x,0) = x+1 \\ f_\pi^{(2)}(x,y+1) &= f_{MIN(PROJ(1))}^{(3)}(x, f_\pi^{(2)}(x,y), y) \\ &= \mu z. f_{PROJ(1)}^{(4)}(x, f_\pi^{(2)}(x,y), y, z) = 0 \\ &= \begin{cases} 0 & x = 0 \\ \uparrow & \text{sonst} \end{cases} \\ f_\pi^{(2)}(x,y) &= \begin{cases} x+1 & \text{falls } y = 0 \\ 0 & \text{falls } x = 0 \wedge y > 0 \\ \uparrow & \text{sonst} \end{cases} \end{aligned}$$

6.41 Satz

Die Klasse der μ-rekursiven Funktionen enthält die Grundfunktionen und ist abgeschlossen gegenüber Komposition, primitiver Rekursion und Minimierung. Sie ist die kleinste Klasse von Funktionen mit dieser Eigenschaft.

Es gilt $\mathcal{P}(\mathbb{N}) \subsetneq \mathcal{R}_p(\mathbb{N})$.

Beweisprinzip für Eigenschaften μ-rekursiver Funktionen:

- Eigenschaft E gilt für die Grundfunktionen.
- E bleibt erhalten bei Komposition, primitiver Rekursion, Minimierung.

Entscheidbare Mengen Abschlusseigenschaften

6.42 Definition

Eine Relation $R \subset \mathbb{N}^n$ heißt **(rekursiv-) entscheidbar,** falls ihre charakteristische Funktion $\chi_R : \mathbb{N}^n \to \mathbb{N}$ μ-rekursiv ist, d. h.

$$\chi_R \in \mathcal{R}_p(\mathbb{N})$$

Beachte: Jede primitiv rekursive Relation ist entscheidbar.

Es gelten für die μ-rekursiven Funktionen und entscheidbaren Relationen zum primitiv rekursiven Fall analoge Abschlusseigenschaften.

Insbesondere:

6.43 Lemma Reduzierbarkeit

Ist $S \subseteq \mathbb{N}^n$ entscheidbar und sind $f_1, \ldots, f_n : \mathbb{N}^m \to \mathbb{N}$ **totale** μ-rekursive Funktionen (z.B. wenn die f_i primitiv rekursiv sind).

Dann ist auch $R \subseteq \mathbb{N}^m$ mit
$R\vec{x} \text{ gdw } Sf_1(\vec{x}) \ldots f_n(\vec{x})$ entscheidbar.

6.44 Beispiel

Sei f eine totale μ-rekursive Funktion, dann ist ihr Graph entscheidbar.

Beweis:

$$\begin{aligned} R(\vec{x}, y) &\text{ gdw } (\vec{x}, y) \in graph(f) \\ &\text{ gdw } f(\vec{x}) = y \end{aligned}$$

Lemma mit $f_1(\vec{x}, y) = f(\vec{x}), f_2(\vec{x}, y) = y, S \equiv \; =$.

Fallunterscheidung mit entscheidbaren Relationen

6.45 Lemma Fallunterscheidung

Seien $R_1, \ldots, R_m$ paarweise disjunkte entscheidbare Relationen und $h_1, \ldots, h_{m+1}$ μ-rekursive Funktionen auf $\mathbb{N}^n$. Dann ist die Funktion $f : \mathbb{N}^n \to \mathbb{N}$ mit

$$f(\vec{x}) = \begin{cases} h_1(\vec{x}) & \text{falls } R_1\vec{x} \\ \vdots & \\ h_m(\vec{x}) & \text{falls } R_m\vec{x} \\ h_{m+1}(\vec{x}) & \text{sonst} \end{cases}$$

μ-rekursiv.

Sind die Funktionen h_i auf R_i definiert und ist h_{m+1} auf $\mathbb{N}^n \setminus \bigcup_1^m R_i$ definiert, so ist f total.

Beweis:

Der Beweis geht nicht wie im primitiv rekursiven Fall!

Damals:

$$f(\vec{x}) = \chi_{R_1}(\vec{x}) \cdot h_1(\vec{x}) + \cdots + \chi_{R_m}(x) \cdot h_m(\vec{x}) + (1 - (\chi_{R_1 \vee \cdots \vee R_m}(\vec{x})) \cdot h_{m+1}(\vec{x})$$

Diese Gleichung gilt mit partiellen Funktionen h_i nicht!

Fallunterscheidung (Fort.)- Weitere Abschlusseigenschaften

Definiere μ-rekursive Hilfsfunktionen H_i für $1 \leq i \leq m+1$ auf $\mathbb{N}^{n+1}$ durch

$H_i(\vec{x}, 0) = 0 \qquad H_i(\vec{x}, y+1) = h_i(\vec{x})$

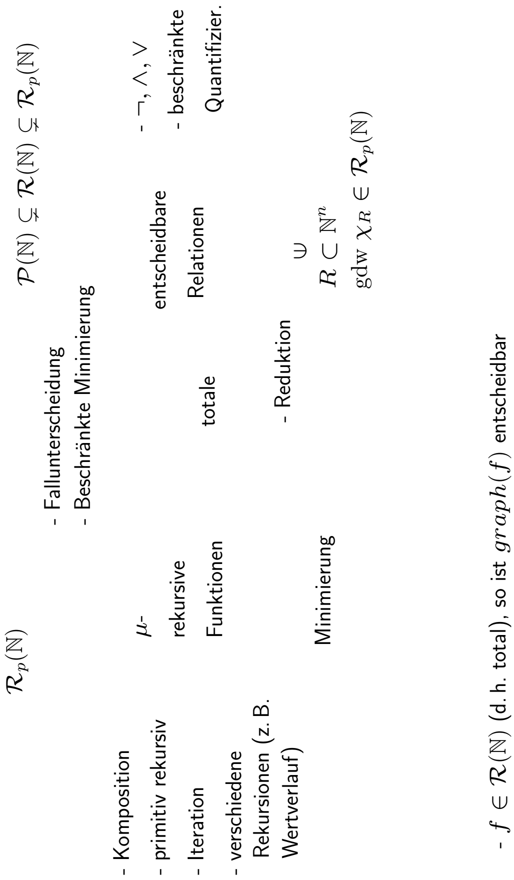
Dann gilt $H_i \in \mathcal{R}_p(\mathbb{N})$ und

$$f(\vec{x}) = H_1(\vec{x}, \chi_{R_1}(\vec{x})) + \cdots + H_m(\vec{x}, \chi_{R_m}(\vec{x})) + H_{m+1}(\vec{x}, 1 - (\chi_{R_1} + \cdots + \chi_{R_m})(\vec{x}))$$

(Beachte $H_i(\vec{x}, y)$ ist für $y > 0$ definiert gdw $h_i(\vec{x})$ definiert ist).

6.46 Lemma

- Die entscheidbaren Relationen sind abgeschlossen gegen $\neg, \wedge, \vee$ und beschränkte Quantifizierung.
- Die Klasse $\mathcal{R}_p(\mathbb{N})$ ist abgeschlossen gegen beschränkte und unbeschränkte Minimierung mit Relationen. Ist $R \subseteq \mathbb{N}^{n+1}$ entscheidbar, so ist die Funktion
  $$f(\vec{x}, b) = \mu y \leq b. R\vec{x}y$$
  und
  $$g(\vec{x}) = \mu y. R\vec{x}y$$
  μ-rekursiv.
  (Das kleinste y mit $R\vec{x}y$, falls es ein solches gibt, sonst $\uparrow$).
  Beachte dabei f ist stets total.



6.3 Universalität der μ-rekursiven Funktionen

Ziel:

Äquivalenz der μ-rekursiven und der durch while-Programme berechenbaren Funktionen.

6.47 Satz

Jede μ-rekursive Funktion ist durch ein while-Programm über N programmierbar.

Beweis: **Simulationstechnik** Durch Induktion über Aufbau der μ-rekursiven Ausdrücken zeige, dass jede partiell rekursive Funktion durch ein while-Programm berechenbar ist.

Somit sind die Grundfunktionen programmierbar. Komposition, primitive Rekursion, Minimierung von programmierbaren Funktionen liefern programmierbare Funktionen.

Voraussetzung: Werden mehrere Programme benötigt, so Verwendung unterschiedlicher Variablen (ggf. Umbenennung von Variablen).

π μ-rekursiver Ausdruck, $n \geq 1$.
Induktive Konstruktion eines While-Programms $\alpha_\pi^{(n)}$, das die Funktion $f_\pi^{(n)}$ mit Eingabevariable $\vec{X}_\pi = (X_\pi^{(1)}, \ldots, X_\pi^{(n)})$ und der Ausgabevariable Y_π berechnet.

Verwende dabei als Abkürzung

$\vec{X} := \vec{t};$ für $X^1 := t^1; \ \ldots, X^n := t^n;$

Programmierbarkeit der μ-rekursiven Funktionen

- **Grundfunktionen**:

$\alpha_{NULL}^{(n)}$	ist das Programm	$Y_{NULL} := 0;$	
$\alpha_{SUCC}^{(n)}$	ist das Programm	$Y_{SUCC} := succ(X_{SUCC}^1);$	
$\alpha_{PROJ(i)}^{(n)}$	ist das Programm	$Y_{PROJ(i)} := X_{PROJ(i)}^i$	$i \leq n$
		$Y_{PROJ(i)} := 0$	$i > n$

- Sei $F = KOMP(G, H_1, \ldots, H_m)$.

  Dann ist $\alpha_F^{(n)}$ das Programm:

$$
\begin{array}{ll}
\{\text{true}\} & \vec{X}_{H_1} := \vec{X}_F;\ \alpha_{H_1}^{(n)} \qquad \{Y_{H_1} = f_{H_1}^{(n)}(\vec{X}_F)\} \\
& \ldots \\
& \ldots \\
& \vec{X}_{H_m} := \vec{X}_F;\ \alpha_{H_m}^{(n)} \qquad \{Y_{H_m} = f_{H_m}^{(n)}(\vec{X}_F)\} \\
& \vec{X}_G := (Y_{H_1}, \ldots, Y_{H_m});\ \alpha_G^{(m)} \\
& \qquad \{Y_G = f_G^{(m)}(Y_{H_1}, \ldots, Y_{H_m})\} \\
& Y_F := Y_G; \\
& \qquad \{Y_F = f_G^{(m)}(f_{H_1}^{(n)}(\vec{X}_F), \ldots, f_{H_m}^{(n)}(\vec{X}_F))\}
\end{array}
$$

Programmierbarkeit der μ-rekursiven Funktionen

- Sei $F = REK(G, H)$. Dann ist $\alpha_F^{(n)}$ das Programm:

$$
\begin{array}{ll}
I := 0; & \\
\vec{X}_G := (X_F^1, \ldots, X_F^n, I);\ \alpha_G^{(n)} & \{Y_G = g(\vec{X}, 0)\} \\
Y_H := Y_G; & \{Y_H = g(\vec{X}, 0)\} \\
\underline{\textbf{while}}\ \neg I = X_F^{n+1}\ \underline{\textbf{do}} & \\
\quad \vec{X}_H := (X_F^1, \ldots, X_F^n, Y_H, I);\ \alpha_H^{(n+2)} & \\
& \{Y_H = h(\ldots)\} \\
\quad I := succ(I); & \\
\underline{\textbf{end}}; & \\
Y_F := Y_H; &
\end{array}
$$

- Sei $F = MIN(G)$. Dann ist $\alpha_F^{(n)}$ das Programm

$$
\begin{array}{ll}
I := 0; & \\
\vec{X}_G := (\vec{X}_F, I);\ \alpha_G^{(n+1)} & \{Y_G = g(\vec{X}, 0)\} \\
\underline{\textbf{while}}\ \neg Y_G = 0\ \underline{\textbf{do}} & \\
\quad I := succ(I); & \\
\quad \vec{X}_G := (\vec{X}_F, I);\ \alpha_G^{(n+1)} & \{Y_G = g(\vec{X}, I)\} \\
\underline{\textbf{end}}; & \\
Y_F := I; &
\end{array}
$$

Universalität der μ-rekursiven Funktionen

D. h. alle partiell-rekursiven Funktionen sind while-berechenbar.

Dies gilt für jede praktisch anwendbare Programmiersprache, die $0, succ$ enthält und Zuweisung, Fallanweisung und Whileanweisungen zulässt. Insbesondere gilt die Behauptung auch für $N = (\mathbb{N}, 0, succ)$.

Wie zeigt man die Umkehrung: Simulation der Berechnung eines while-Programms durch eine μ-rekursive Funktion.

$I_A(\alpha, z)$: Interpreterfunktion, als primitiv-rekursive Funktion
Iteration
Minimierung.

da Semantik $z[[\alpha]]_A z'$ gdw $\exists n \in \mathbb{N}\ I_A^n(\alpha, z) = (\varepsilon, z')$

Problem: Die μ-rekursiven Funktionen sind arithmetische Funktionen, also muss man eine Arithmetisierung aller Konstrukte die bei den While-Programmen vorkommen durchführen.

Codierung syntaktischer Objekte, die in der Definition der Interpreterfunktion vorkommen

$code : syn_obj \to \mathbb{N}$

Vereinbarungen:

- Variablen sind aus Menge $Var = \{V_0, V_1, \dots\}$ zu wählen
- Terme enthalten nur $Var, 0, succ$ (d.h. Programme über N).
- Boolesche Formeln nur mit $\wedge, \neg$

Codierungsfunktion $code$

6.48 Definition Codierungsfunktion $code : syn_obj \to \mathbb{N}$

Induktiv über Aufbau der syn_obj, definiert durch:

- $code(\varepsilon) = 0$
- $code(0) = [1]$
- $code(V_i) = [2, i]$
- $code(succ(t)) = [3, code(t)]$
- $code(s = t) = [4, code(s), code(t)]$
- $code(\neg B) = [5, code(B)]$
- $code(B \wedge C) = [6, code(B), code(C)]$
- $code(V_i := t;\ ) = [7, i, code(t)]$
- $code($**if** B **then** β **else** γ **end**$;\ ) = [8, code(B), code(\beta), code(\gamma)]$
- $code($**while** B **do** β **end**$;\ ) = [9, code(B), code(\beta)]$
- $code(A_1 \dots A_n) = [code(A_1), \dots, code(A_n)]$ (Anweisungen $A_i, n \geq 2$)
- $z : V \to \mathbb{N}$ Zustand mit $V \subseteq \{V_0, \dots, V_m\}$ so $code(z) = [x_0, \dots, x_m]$ mit $x_i = z(V_i)$, falls $V_i \in V$, sonst $x_i = 0$.

Codierungsfunktion $code$ (Forts.)

Beachte:

- $[\cdot]$ ist die Codierung von endlichen Zahlenfolgen mit Folgenzugriffsfunktion

  $$get(z, i) = \begin{cases} x_i & \text{für } [x_0, \dots, x_n] = z \ \ (i \leq n) \\ 0 & \text{sonst} \end{cases}$$

  get ist primitiv rekursiv.

  Abkürzung: $z[i]$ für $get(z, i)$.

- $code(z)$ ist wohldefiniert, wegen der Invarianz der Folgencodierung in Bezug auf Nullen am rechten Folgenende.
- Codierung ist nicht injektiv:
  Termcodes, Formelcodes und Programmcodes können gleich sein.

Codierungsfunktion $code$ Beispiel

6.49 Beispiel

Sei α das Programm:

$V_0 := V_1;$
$V_3 := 0;$
while $\neg V_2 = V_3$ **do**
  $V_0 := succ(V_0);$
  $V_3 := succ(V_3);$
end;

$$code(\alpha) = [code(A_1), code(A_2), code(A_3)] = \langle code(A_1), \langle code(A_2), \langle code(A_3), 0\rangle\rangle\rangle$$

$$code(A_1) = [7, 0, [2, 1]] = [7, 0, 7] = 97895$$

$$code(A_2) = [7, 3, 1] = 182$$

$$code(A_3) = [9, \underbrace{code(\neg V_2 = V_3)}, code(V_0 := succ(V_0); V_3 := succV_3))]$$

$$= [5, code(V_2 = V_3)] = [5, [4, [2, 2], [2, 3]]] \ \vdots$$

Primitiv rekursive Simulation der Termauswertung

- Definiere **Termauswertungsfunktion** $\tau : \mathbb{N}^2 \longrightarrow \mathbb{N}$, so dass für alle Terme t und Zustände z gilt

$$(*) \qquad \tau(code(t), code(z)) = val_{N,z}(t)$$

τ ist somit auf den Codes von Termen und Zuständen eindeutig definiert. (Dies genügt!)
Definiere τ durch Fallunterscheidung wie folgt

$$\tau(x,y) = \begin{cases} 0 & \text{falls } x[0] = 1 \quad (\text{konst } 0) \\ y[x[1]] & \text{falls } x[0] = 2 \quad (\text{var}) \\ \tau(x[1], y) + 1 & \text{falls } x[0] = 3 \quad (succ) \\ 2001 & \text{sonst} \end{cases}$$

Offenbar ist $\tau \in \mathcal{P}(\mathbb{N})$ (Fallunterscheidung primitiv rekursiver Relation) und τ erfüllt $(*)$.

Primitiv rekursive Simulation der Auswertung B-Ausdrücke

- Definiere **Auswertungsfunktion Boolescher Formeln** $\beta : \mathbb{N}^2 \longrightarrow \mathbb{N}$, so dass für alle B-Formeln B und Zustände z gilt

$$(**) \qquad \beta(code(B), code(z)) = \begin{cases} 1 & \text{falls } \mathbb{N} \models_z B \\ 0 & \text{sonst} \end{cases}$$

Definiere β durch Fallunterscheidung + Wertverlaufsrekursion.

$$\beta(x,y) = \begin{cases} \chi_=(\tau(x[1], y), \tau(x[2], y)) & \text{falls } x[0] = 4 \\ 1 - \beta(x[1], y) & \text{falls } x[0] = 5 \\ \beta(x[1], y) \cdot \beta(x[2], y) & \text{falls } x[0] = 6 \\ 2001 & \text{sonst} \end{cases}$$

Offenbar ist $\beta \in \mathcal{P}(\mathbb{N})$ und β erfüllt $(**)$.

Primitiv rekursive Simulation der Speicheränderungen + Interpreterfunktion

- Definiere **Funktion für Speicheränderungen** (bei Zuweisungen): $\sigma : \mathbb{N}^3 \longrightarrow \mathbb{N}$, so dass für alle Zustände z, Variablen V_i und Werte $a \in \mathbb{N}$ gilt:

$$(***) \qquad \sigma(code(z), i, a) = code(z(V_i/a))$$

Definiere σ rekursiv über i durch

$$\sigma(x, 0, a) = \langle a, rest(x) \rangle$$
$$\sigma(x, i+1, a) = \langle \text{first}(x), \sigma(\text{rest}(x), i, a) \rangle$$

Offenbar ist $\sigma \in \mathcal{P}(\mathbb{N})$ und erfüllt $(***)$.

- Definiere **Funktion zur Simulation der Interpreterfunktion** I_N, $i : \mathbb{N} \longrightarrow \mathbb{N}$, so dass für alle Programme α und α' und Zustände z und z' gilt:

$$(****) \qquad I_N(\alpha, z) = (\alpha', z') \qquad \text{gdw}$$
$$i(\langle code(\alpha), code(z) \rangle) = \langle code(\alpha'), code(z') \rangle$$

i muss für Codierungen der Form $\langle p, y \rangle$ richtig arbeiten.

Beachte es gilt
$p = \langle \text{first}(p), \text{rest}(p) \rangle$ (Paarungsfunktion) und für $p > 0$ die erste Anweisung des von p codierten Programms den Code first(p) und das Restprogramm den Code rest(p) haben.

Interpreterfunktion (Forts.)

$i(\langle p, y \rangle)$ wird durch Fallunterscheidung definiert:
$i(\langle p, y \rangle) =$

$$\begin{cases} \langle p, y \rangle & \text{falls } p = 0 \\ \langle rest(p), \sigma(y, \text{first}(p)[1], \\ \qquad \tau(\text{first}(p)[2], y)) \rangle & \text{falls first}(p)[0] = 7 \\ \langle \text{first}(p)[2] * \text{rest}(p), y \rangle & \text{falls first}(p)[0] = 8 \\ & \text{und } \beta(\text{first}(p)[1], y) = 1 \\ \langle \text{first}(p)[3] * \text{rest}(p), y \rangle & \text{falls first}(p)[0] = 8 \\ & \text{und } \beta(\text{first}(p)[1], y) = 0 \\ \langle \text{first}(p)[2] * p, y \rangle & \text{falls first}(p)[0] = 9 \\ & \text{und } \beta(\text{first}(p)[1], y) = 1 \\ \langle \text{rest}(p), y \rangle & \text{falls first}(p)[0] = 9 \\ & \text{und } \beta(\text{first}(p)[1], y) = 0 \\ 2001 & \text{sonst} \end{cases}$$

Offenbar ist $i \in \mathcal{P}(\mathbb{N})$ und erfüllt $(****)$.

Simulation der Speicherinitialisierung und Ausgabefunktion

- **Speicherinitialisierung** für Programm mit Code p, $\mathsf{inp}^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$.
  Initialisierung bei Eingabe von n-Zahlen $x_1, \ldots, x_n$ in den Variablen $V_1, \ldots, V_n$. V_0 dient als Ausgabevariable.
  $$\mathsf{inp}^{(n)}(p, x_1, \ldots, x_n) = \langle p, [0, x_1, \ldots, x_n] \rangle$$
- **Ausgabe** des berechneten Wertes für Programm mit Code p, $\mathsf{out} : \mathbb{N} \to \mathbb{N}$
  $$\mathsf{out}(\langle p, x \rangle) = x[0]$$
- $\mathsf{inp}^{(n)}$ und out sind in $\mathcal{P}(\mathbb{N})$.

Wir haben nun alle Bestandteile zusammen um die Simulation der while-berechenbaren Funktion durch die partiell rekursiven Funktionen nachzuweisen. Die Interpreterfunktion ist so lange zu iterieren, bis als Restprogramm das leere Programm (mit Codezahl 0) entsteht. Diese Iterationszahl kann als Zeitkomplexitätsfunktion betrachtet werden. Sie misst nämlich die Anzahl der ausgeführten Anweisungen sowie die Anzahl der ausgewerteten B-Formeln.

Laufzeitfunktionen
Universelle Funktionen

6.50 Definition Zeitkomplexitätsfunktion

Sei p Code eines Programms mit Eingabevariablen $V_1, \ldots, V_n$. Die Laufzeit (Anzahl der Rechenschritte) bei Eingabe $x_1, \ldots, x_n$ sei definiert durch folgende μ-rekursive Funktion $\Phi^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$ mit

$$\Phi^{(n)}(p, x_1, \ldots, x_n) = \mu t.\mathsf{first}(i^t(\mathsf{inp}^{(n)}(p, x_1, \ldots, x_n)) = 0$$

Φ heißt **Zeitkomplexitätsfunktion.**

Simulation der Berechnung des Programms mit Codezahl p bei Eingabe $x_1, \ldots, x_n$ durch μ-rekursive Funktion $\varphi^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$.

$$\varphi^{(n)}(p, x_1, \ldots, x_n) = \mathsf{out}(i^{\Phi^{(n)}(p, x_1, \ldots, x_n)}(\mathsf{inp}^{(n)}(p, x_1, \ldots, x_n)))$$

$\varphi^{(n)}(p, x_1, \ldots, x_n) \downarrow$ gdw Programm mit Code p terminiert bei Eingabe $x_1, \ldots, x_n$.

$\varphi^{(n)}(p, x_1, \ldots, x_n)$ ist dann die Ausgabe (in V_0) vom Programm mit Codezahl p bei Eingabe $x_1, \ldots, x_n$ in $V_1, \ldots, V_n$.

$\rightsquigarrow$ **Universeller Rechner für While-Programme**

Schreibweisen: $\varphi_p^{(n)}(x_1, \ldots, x_n)$ für $\varphi^{(n)}(p, x_1, \ldots, x_n)$
bzw. $\Phi_p^{(n)}(x_1, \ldots, x_n)$ für $\Phi^{(n)}(p, x_1, \ldots, x_n)$.

(n) weglassen, falls $n = 1$.

Offenbar gilt $\Phi^{(n)}, \varphi^{(n)} \in \mathcal{R}_p(\mathbb{N})$.

Äquivalenz while-berechenbaren und μ-rekursiven Funktionen

6.51 Satz

Ist $f : \mathbb{N}^n \to \mathbb{N}$ mit $n \geq 1$ durch ein while-Programm in N berechenbar, dann ist f μ-rekursiv.

Beweis: o.b.d.A. Eingabevariable $V_1, \ldots, V_n$, Ausgabe V_0.

Alle anderen benutzten Variablen mit 0 initialisiert (d. h. Codezahl für Zustand der $V_0, \ldots, V_n$ belegt ist auch Codezahl für Zustand der $V_0, \ldots, V_n, V_{n+1}, \ldots, V_m$ belegt, wobei $V_{n+1}, \ldots, V_m$ mit 0 belegt werden).

Sei α ein solches Programm, α berechne $f : \mathbb{N}^n \to \mathbb{N}$, dann gilt
$$f(x_1, \ldots, x_n) = \varphi^{(n)}(code(\alpha), x_1, \ldots, x_n)$$
Da $\varphi^{(n)} \in \mathcal{R}_p(\mathbb{N})$ gilt auch $f \in \mathcal{R}_p(\mathbb{N})$.

These: Die Klasse der berechenbaren Funktionen ist $\mathcal{R}_p(\mathbb{N})$.

6.52 Folgerung Jede Funktion $f \in \mathcal{R}_p$ lässt sich mittels maximal einmaliger Anwendung der Minimierung angewandt auf eine totale Funktion definieren.

Beweis: $f(x_1, \ldots, x_n) = \varphi^{(n)}(code(\alpha_f), x_1, \ldots, x_n)$, wobei f vom Programm α_f mit Eingabevariablen $V_1, \ldots, V_n$ und Ausgabe V_0 berechnet wird.

Beachte: Zu $f \in \mathcal{R}_p$, $f : \mathbb{N}^n \to \mathbb{N}$ gibt es ein $p \in \mathbb{N}$, so dass $f(x_1, \ldots, x_n) = \varphi_p^{(n)}(x_1, \ldots, x_n)$. p **ist Index für** f.

Universelle μ-rekursive Funktionen

Es gibt Funktionen $f : \mathbb{N}^2 \to \mathbb{N}$ mit $f \in \mathcal{R}_p(\mathbb{N})$, so dass es für jede μ-rekursive Funktion $g : \mathbb{N} \to \mathbb{N}$ einen Index i gibt, mit $f(i, _) = g$.

Beachte: Es gibt stets ∞-viele i für festes g. (wähle z. B. $f = \varphi^{(1)}$)

Das Ergebnis für While-Programme lässt sich leicht auf rekursive Programme übertragen.

$code(call \ldots)$

Für die erweiterte Interpreterfunktion I_Ω lässt sich wiederum eine primitiv rekursive Simulation i_Ω leicht angeben. Man erhält entsprechend:

$$\exists t\ I_\Omega^t(\alpha, z) = (\varepsilon, z') \Leftrightarrow \exists t'\ i_\Omega^{t'}(\langle code(\alpha), code(z) \rangle) = \langle 0, code(z') \rangle$$

Insbesondere gilt:

Rekursive While-Programme können nicht mehr berechnen als While-Programme.

Beide Klassen sind $\mathcal{R}_p(\mathbb{N})$.(Für die Algebra N)

6.4 Grundzüge der Rekursionstheorie

Folgerungen aus der Existenz universeller Funktionen

$\Phi^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$ Zeitkomplexitätsfunktion

$\varphi^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$ Universelle Funktion

6.53 Lemma Eigenschaften von $\Phi^{(n)}$ bzw. $\varphi^{(n)}$:

$\Phi^{(n)}$ und $\varphi^{(n)}$ haben denselben Definitionsbereich.

Die Relationen **Beschränkte Laufzeit** $BLZ \subset \mathbb{N}^{n+2}$

$BLZ = \{(p, \vec{x}, b) \in \mathbb{N}^{n+2} : \Phi^{(n)}(p, \vec{x}) \leq b\}$

und **Beschränkte Berechenbarkeit** $BBER \subset \mathbb{N}^{n+3}$

$BBER =$
$\{(p, \vec{x}, b, y) \in \mathbb{N}^{n+3} : \Phi^{(n)}(p, \vec{x}) \leq b \wedge \varphi^{(n)}(p, \vec{x}) = y\}$

sind primitiv rekursiv.

Beweis:

Definitionsbereiche gleich folgt aus Definition von $\Phi^{(n)}$ bzw. $\varphi^{(n)}$.

Relationen primitiv rekursiv, da

$$BLZ = \exists t \leq b.\ \underbrace{\mathsf{first}(i^t(\mathsf{inp}^{(n)}(p, \vec{x}))) = 0}_{rel(p,\vec{x},t)\ \text{primitiv-rekursiv}}$$

bzw.

$$BBER = \exists t \leq b.\ \underbrace{(\mathsf{first}(i^t(\mathsf{inp}^{(n)}(p, \vec{x}))) = 0 \wedge \mathsf{out}(i^t(\mathsf{inp}^{(n)}(p, \vec{x}))) = y)}_{rel(p,\vec{x},t,y)\ \text{primitiv-rekursiv}}$$

Folgerungen

6.54 Satz s-m-n Theorem

Zusammenhang zwischen universellen Funktionen. Zu jedem Paar $m, n \in \mathbb{N}$ mit $n \geq 1$ gibt es eine primitiv rekursive Funktion $s_{m,n} : \mathbb{N}^{m+1} \to \mathbb{N}$, so dass für alle $p \in \mathbb{N}, \vec{x} \in \mathbb{N}^n$ und $\vec{y} \in \mathbb{N}^m$ gilt:

$$\varphi^{(n+m)}(p, \vec{x}, \vec{y}) \quad = \varphi^{(n)}(s_{m,n}(p, \vec{y}), \vec{x}) \quad = \varphi^{(n)}_{s_{m,n}(p,\vec{y})}(\vec{x})$$

Beweis:

In Anhängigkeit von $\vec{y}$ ist der Programmcode p so abzuändern, dass die Werte $\vec{y}$ nicht über die Eingabe eingelesen, sondern im Programm zugewiesen werden. Hierbei ist zu beachten, dass die Eingabezahlen durch geeignete Terme dargestellt werden.

Ist $p = code(\alpha)$ muss $s_{m,n}(p, \vec{y})$ der Code zum Programm

$V_{n+1} := succ^{y_1}(0);\ \ldots;\ V_{n+m} := succ^{y_m}(0);\ \alpha$ sein.

Sei

$s_{m,n}(p, \vec{y}) := [[7, n+1, h(y_1)], \ldots, [7, n+m, h(y_m)]] * p$

Mit $*$ die Folgenverkettungsfunktion von Beispiel 6.30 und $h : \mathbb{N} \to \mathbb{N}$ wobei

$h(0) = [1]$ ($code$ von 0), $\quad h(y+1) = [3, h(y)]$,

d. h. $h(y) = code(succ^y(0))$

Folgerungen - Programmtransformatoren

Beachte:

Bei obiger Situation gilt auch für beliebige $n \in \mathbb{N}$ und l mit $1 \leq l \leq m$ die Aussage

$\varphi^{(n+m)}(p, \vec{x}, \vec{y}) = \varphi^{(n+l)}(s_{m,n}(p, \vec{y}), \vec{x}, \vec{z})$

für alle $\vec{z} \in \mathbb{N}^l$.

6.55 Lemma

Es gibt eine primitiv rekursive Funktion $compose : \mathbb{N}^2 \to \mathbb{N}$, so dass für alle $p, q \in \mathbb{N}$ gilt

$$\varphi_{compose(p,q)}(x) = \varphi_p(\varphi_q(x))$$

Beweis:

Die Funktion $f(x, p, q) = \varphi_p(\varphi_q(x))$ ist μ-rekursiv. Sei a Index zu einem f berechnenden Programm. D. h.

$\varphi^{(3)}_a(x, p, q) = f(x, p, q)$ für $(x, y, q) \in \mathbb{N}^3$
$\qquad = \varphi^{(1)}_{s_{2,1}(a,p,q)}(x)$

D. h. $compose(p, q) = s_{2,1}(a, p, q)$ ist primitiv rekursiv.

Lässt sich auf andere Operationen übertragen:

Primitive Rekursion, Beschränkte Minimierung, Minimierung, Wertverlaufsrekursion.

Rekursiv aufzählbare Relationen

Wir haben bisher Relationen betrachtet, die entweder primitiv rekursiv oder rekursiv entscheidbar waren. Eine weitere Klasse von Relationen sind die effektiv aufzählbaren Relationen.

6.56 Definition

Eine Relation $R \subseteq \mathbb{N}^n$ heißt **rekursiv aufzählbar**, wenn es eine berechenbare (μ-rekursive) Funktion f mit Definitionsbereich R gibt, d. h.

$R \subseteq \mathbb{N}^n$ rekursiv aufzählbar $\text{gdw}\ \ \exists f \in \mathcal{R}_p(\mathbb{N}) : dom(f) = R$
also $(f(\vec{x}) \downarrow\ \ \text{gdw}\ R\vec{x}\ \ (x \in \mathbb{N}^n))$

6.57 Lemma

$R \subseteq \mathbb{N}^n$ ist entscheidbar gdw R und $\neg R$ rekursiv aufzählbar.

Beweis:

Sei R entscheidbar. Dann ist auch $\neg R$ entscheidbar.

R ist Definitionsbereich der Funktion

$$f(\vec{x}) = \begin{cases} 1 & \text{falls } R\vec{x} \\ \uparrow & \text{sonst} \end{cases}$$

$f \in \mathcal{R}_p(\mathbb{N})$ (Beachte dabei $\uparrow (x) = \uparrow$ alle x ist in $\mathcal{R}_p(\mathbb{N})$).

R.a. Relationen - Charakterisierungen

Seien R und $\neg R$ rekursiv aufzählbar und a bzw. b Programmindizes von Funktionen, die als Definitionsbereich R und $\neg R$ haben.

Sei

$$f(\vec{x}) = \mu t.(\Phi_a^{(n)}(\vec{x}) \leq t \vee \Phi_b^{(n)}(\vec{x}) \leq t)$$

$f \in \mathcal{R}_p(\mathbb{N})$ und total, da jedes $\vec{x}$ entweder in $R = dom(\varphi_a^{(n)})$ oder in $\neg R = dom(\varphi_b^{(n)})$ liegt.

Weiterhin ist $\quad R\vec{x} \;\; \mathrm{gdw} \;\; \Phi_a^{(n)}(\vec{x}) \leq f(\vec{x})$ entscheidbar.

6.58 Lemma R.a. Relationen und entscheidbare Relationen

$R \subseteq \mathbb{N}^n$ ist rekursiv aufzählbar gdw es gibt eine entscheidbare Relation $S \subseteq \mathbb{N}^{n+1}$ mit $\quad R\vec{x} \;\; \mathrm{gdw} \;\; \exists y \; S\vec{x}y$.

Beweis:

- Sei R rekursiv aufzählbar. Sei a Index mit $R = dom(\varphi_a^{(n)})$. Dann gilt

  $$R\vec{x} \;\; \mathrm{gdw} \;\; \varphi_a^{(n)}(\vec{x}) \downarrow \;\; \mathrm{gdw} \;\; \exists y \; \Phi_a(\vec{x}) \leq y$$

  Die Relation $S\vec{x}y \; \mathrm{gdw} \; \Phi_a(\vec{x}) \leq y$ ist entscheidbar (sogar primitiv rekursiv entscheidbar).
- Gelte $R\vec{x} \; \mathrm{gdw} \; \exists y \; S\vec{x}y$ mit S entscheidbar. Dann ist $f(\vec{x}) = \mu y.S\vec{x}y$ berechenbar und hat als Definitionsbereich genau R.

(Lemma gilt auch, wenn entscheidbar durch primitiv rekursiv ersetzt wird.)

Zusammenhänge

6.59 Lemma

$f : \mathbb{N}^n \rightarrow \mathbb{N}$ ist berechenbar gdw der Graph von f rekursiv aufzählbar ist. D. h.

$graph(f) = \{(\vec{x}, y) \in \mathbb{N}^{n+1} : f(\vec{x}) \downarrow \wedge f(\vec{x}) = y\}$ rekursiv aufzählbar.

Beweis:

• Sei $f = \varphi_a^{(n)}$. Dann gilt für alle $\vec{x}, y$.

$$(\vec{x}, y) \in graph(f) \;\mathrm{gdw}\; \exists b \underbrace{(\Phi_a^{(n)}(\vec{x}) \leq b \wedge \varphi_a^{(n)}(\vec{x}) = y)}_{S\vec{x}yb \text{ entscheidbar nach 6.53}}$$

also rekursiv aufzählbar nach Lemma 6.58.

• Sei $graph(f)$ rekursiv aufzählbar: $graph(f) = dom(\varphi_b^{(n+1)})$, dann gilt

$$f(\vec{x}) = \mathsf{first}(\mu y.(\Phi_b^{(n+1)}(\vec{x}, \mathsf{first}(y)) \leq \mathsf{rest}(y)))$$

Dafür betrachte man $y = \langle z, t\rangle$ als Codierung eines Paares (z, t). Ist $(\vec{x}, z) \in graph(f)$, so gibt es ein t mit $\Phi_b^{(n+1)}(\vec{x}, z) = t$ und der μ-Operator liefert y mit $y = \langle z, t\rangle$. Für $f(\vec{x}) = \uparrow$ gibt es kein solches y.

Folgerungen - Weitere Charakterisierungen

6.60 Lemma

Sei f eine totale Funktion. Dann gilt $f \in \mathcal{R}_p(\mathbb{N}) \; \mathrm{gdw} \; graph(f)$ ist entscheidbar.

Beweis:

Sei f total, berechenbar. Dann ist die Menge $\{(\vec{x}, y) : f(\vec{x}) = y\}$ entscheidbar, d. h. $graph(f)$ ist entscheidbar (siehe auch Beispiel 6.44).

Ist $graph(f)$ entscheidbar, so ist auch $graph(f)$ rekursiv aufzählbar und somit f berechenbar.

Einfacher: $f(\vec{x}) = \mu y.\chi_{graph(f)}(x, y) = 1$.

6.61 Lemma Effektive Aufzählungen r.a. Relationen

$R \subseteq \mathbb{N}^n$ ist rekursiv aufzählbar gdw $R = \varnothing$ oder es gibt totale berechenbare Funktionen $f_1, \ldots, f_n : \mathbb{N} \rightarrow \mathbb{N}$ mit $R = \{(f_1(i), \ldots, f_n(i)) : i \in \mathbb{N}\}$.

Man kann also R mit Hilfe der Funktionen f_i „effektiv“ aufzählen.

Ist $n = 1$, so ist $R = im(f_1)$, d. h. R ist Bild einer totalen berechenbaren Funktion.

Weitere Charakterisierungen (Fort.)

Beweis:

Sei $R \subseteq \mathbb{N}^n$ rekursiv aufzählbar, $R \neq \varnothing$. Sei $R = dom(\varphi_a^{(n)})$, $\vec{b} \in R$ fest. Definiere Funktionen $f_i \quad i = 1, \ldots, n$ durch

$$f_i(x) = \begin{cases} x[i] & \text{falls } \Phi_a^{(n)}(x[1], \ldots, x[n]) \leq x[0] \\ b_i & \text{sonst} \end{cases}$$

Durchläuft x alle natürlichen Zahlen, so werden alle möglichen Parameter für $\Phi_a^{(n)}$ durchlaufen. Die Relation $\Phi_a^{(n)}(x[1], \ldots, x[n]) \leq x[0]$ ist nach Lemma 6.53 primitiv rekursiv. Somit sind die f_i primitiv rekursiv, also total, und liefern alle Werte von R.

Umgekehrt ist $R = \varnothing$, so ist R rekursiv aufzählbar als Definitionsbereich der nirgends definierten Funktion. Sei also $R = \{(f_1(i), \ldots, f_n(i)) : i \in \mathbb{N}\}$ $f_i \in \mathcal{R}_p(\mathbb{N})$ totale Funktionen, dann liefert

$$R\vec{x} \;\; \mathrm{gdw} \;\; \exists i(x_1 = f_1(i) \wedge \cdots \wedge x_n = f_n(i))$$

R ist rekursiv aufzählbar nach Lemma 6.58.

Abschlusseigenschaften r.a. Relationen

6.62 Lemma Abschlusseigenschaften
Die r.a. Relationen sind

a) Abgeschlossen gegen **existentielle Quantifizierung**
$R \subseteq \mathbb{N}^{n+1}$ $n \geq 1$ sei rekursiv aufzählbar. Dann auch S mit

$$S\vec{x} \text{ gdw } \exists y R\vec{x}y$$

b) Abgeschlossen gegen **Vereinigung und Durchschnitt**
Sind $R, S \subseteq \mathbb{N}^n$ rekursiv aufzählbar. Dann sind auch $R \cup S$, $R \cap S$ rekursiv aufzählbar.

Beweis:

a) Sei R rekursiv aufzählbar. Also $R\vec{a}$ gdw $\exists z T\vec{a}z$ für eine entscheidbare Relation T. Dann ist $S\vec{x}$ gdw
$\exists y R\vec{x}y$ gdw $\exists y \exists z T\vec{x}yz$ gdw $\exists b\, T\vec{x}\text{first}(b) rest(b)$

b) Sei $R\vec{x}$ gdw $\exists y\, T\vec{x}y$ und $S\vec{x}$ gdw $\exists y\, V\vec{x}y$ mit T, V entscheidbare Relationen. Dann gilt
$(R \wedge S)\vec{x}$ gdw
$\exists y \exists z\, (T\vec{x}y \wedge V\vec{x}z)$ gdw $\exists u\, (T\vec{x}\text{first}(u) \wedge V\vec{x}rest(u))$
und
$(R \vee S)\vec{x}$ gdw
$\exists y \exists z\, (T\vec{x}y \vee V\vec{x}z)$ gdw $\exists u\, (T\vec{x}\text{first}(u) \wedge V\vec{x}rest(u))$

Zusammenfassung Charakterisierungen der r.a Relationen

Sei $R \subseteq \mathbb{N}^n$, dann sind äquivalent:

- $R \subseteq \mathbb{N}^n$ ist rekursiv aufzählbar.
- $R = dom(f)$ für ein $f \in \mathcal{R}_p(\mathbb{N})$.
- $R = \emptyset \vee R = im(f_1, \ldots, f_n)$ für $f_i \in \mathcal{R}(\mathbb{N})$.
- $R = \emptyset \vee R = im(f_1, \ldots, f_n)$ für $f_i \in \mathcal{P}(\mathbb{N})$.
- R endlich $\vee$ $R = im(f_1, \ldots, f_n)$ für $f_i \in \mathcal{R}(\mathbb{N})$ mit f_i injektiv.
- $R\vec{x}$ gdw $\exists y\, S\vec{x}y$ für eine entscheidbare Relation S.
- $R\vec{x}$ gdw $\exists y\, S\vec{x}y$ für eine primitiv rekursive Relation S.

R ist entscheidbar gdw R und $\neg R$ sind rekursiv aufzählbar.
Rekursiv aufzählbare Relationen sind abgeschlossen gegen $\cup, \cap, \exists$.

Was mit $\neg$ und $\forall$?

Weitere Funktionsdefinitionen

6.63 Lemma Funktiondefinition auf r.a. Relationen
Sei $g : \mathbb{N}^n \to \mathbb{N}$ berechenbar, $R \subseteq \mathbb{N}^n$ rekursiv aufzählbar. Dann ist auch

$$f(\vec{x}) = \begin{cases} g(\vec{x}) & \text{falls } R\vec{x} \\ \uparrow & \text{sonst} \end{cases}$$

berechenbar.

Beweis: Es gilt $f(\vec{x}) = sgn(succ(h(\vec{x}))) \cdot g(\vec{x})$, sofern $dom(h) = R$.

Da R rekursiv aufzählbar ist, gibt es ein $h \in \mathcal{R}_p(\mathbb{N})$ mit $dom(h) = R$, also auch $f \in \mathcal{R}_p(\mathbb{N})$.

Führt jede Definition einer Funktion durch Fallunterscheidung mit paarweise disjunkten r.a. Relationen wieder zu berechenbaren Funktionen?

Sind die r.a. Relationen eine echte Obermenge der entscheidbaren Relationen. Gibt es r.a. Relationen die nicht rekursiv entscheidbar sind?

Unentscheidbare Relationen

Wichtige Probleme

- Das **allgemeine Halteproblem:** $K_0 \subseteq \mathbb{N}^2$

$$K_0 = \{(a, x) \in \mathbb{N}^2 : \varphi_a(x) \downarrow\}$$

- Das **spezielle Halteproblem (Selbstanwendungsproblem):** $K \subseteq \mathbb{N}$

$$K = \{a \in \mathbb{N} : \varphi_a(a) \downarrow\}$$

- Sei $f : \mathbb{N} \to \mathbb{N}$ mit $f \in \mathcal{R}(\mathbb{N})$, d. h. total.
$\mathbf{Spez}(f) = \{a \in \mathbb{N} : \varphi_a = f\}$ Menge der „Indizes von f"
- $\mathbf{\ddot{A}quiv} \subseteq \mathbb{N}^2$ Äquivalenz von Indizes
$\ddot{A}quiv = \{(a, b) \in \mathbb{N}^2 : \varphi_a = \varphi_b\}$

Beachte: K_0 und K sind rekursiv aufzählbar. Wie stehen diese Probleme in Beziehung?

Erinnerung: $R \subseteq \mathbb{N}^n$, $S \subseteq \mathbb{N}^l$. S kann auf R **rekursiv reduziert** werden (schreibe $\mathbf{S} \leq_\mathbf{m} \mathbf{R}$), falls es totale berechenbare Funktionen $f_1, \ldots, f_n : \mathbb{N}^l \to \mathbb{N}$ gibt, so dass gilt

$$S\vec{x} \text{ gdw } Rf_1(\vec{x}) \cdots f_n(\vec{x})$$

Wir hatten gezeigt: Ist R entscheidbar, so auch S.

Eigenschaften der Many-one Reduzierbarkeit

6.64 Lemma Es gilt

- $\leq_m$ ist reflexiv und transitiv.
- Ist $S \leq_m R$, R entscheidbar, so ist S entscheidbar.
- Ist $S \leq_m R$, R rekursiv aufzählbar, so ist S rekursiv aufzählbar.
- $S \leq_m R$, so ist $\neg S \leq_m \neg R$.

Beweis:

Sei $R = dom(f)$, dann ist $S = dom(f \circ (f_1, \ldots, f_n))$.

Die anderen Eigenschaften sind leicht zu beweisen.

6.65 Lemma

Es gilt

• $K \leq_m K_0$ • $K \leq_m spez(f)$ • $Spez(succ) \leq_m \ddot{A}quiv$

Beweis:

• Es ist $a \in K$ gdw $(a, a) \in K_0$ $(f_1, f_2$ Identität auf $\mathbb{N})$.

• Sei $f : \mathbb{N} \longrightarrow \mathbb{N}$ total berechenbar. Definiere

$$\psi(x,p) = \begin{cases} f(x) & \text{falls } p \in K \\ \uparrow & \text{sonst} \end{cases}$$

ψ ist berechenbar, also gibt es ein $a \in \mathbb{N}$ mit $\varphi_a^{(2)} = \psi$.

Rekursiv aufzählbare Relationen, die nicht entscheidbar sind

Nach Definition von ψ gilt

$p \in K \Rightarrow \varphi_a^{(2)}(x,p) = f(x)$

$p \notin K \Rightarrow \varphi_a^{(2)}(x,p) \uparrow$

Nach s-m-n Theorem gibt es primitiv rekursive Funktion

$g(p) = s_{1,1}(a,p) : \varphi_a^{(2)}(x,p) = \varphi_{g(p)}(x)$.

• Dann ist $p \in K$ gdw $g(p) \in Spez(f)$.

• Sei $a \in \mathbb{N}$ mit $\varphi_a = succ$:

$b \in Spez(succ)$ gdw $(a,b) \in \ddot{A}quiv$.

6.66 Satz

$K = \{a \in \mathbb{N} : \varphi_a(a) \downarrow\}$ ist nicht entscheidbar.

Beweis: Angenommen K wäre entscheidbar. Wende Diagonalisierungsargument an: Sei $f : \mathbb{N} \longrightarrow \mathbb{N}$ mit

$$f(x) = \begin{cases} \varphi_x(x) + 1 & \text{falls } x \in K \\ 0 & \text{sonst} \end{cases}$$

$f \in \mathcal{R}_p(\mathbb{N})$ und f ist total. Es gibt einen Index $p \in \mathbb{N}$ für f, d. h. $f = \varphi_p$. Insbesondere $f(p) = \varphi_p(p)$. Dies ist ein Widerspruch, da

- Ist $p \in K$, so $\varphi_p(p) \downarrow$ und $f(p) = \varphi_p(p) + 1$ ↯
- Ist $p \notin K$, so $\varphi_p(p) \uparrow$ und $f(p) = 0$ ↯

Folgerungen

6.67 Folgerung Es gilt

- $K_0, Spez(f)(f \in \mathcal{R})$ und $\ddot{A}quiv$ sind unentscheidbar.
- $\neg K, \neg K_0$ sind nicht rekursiv aufzählbar.
- Die rekursiv aufzählbaren Relationen sind nicht abgeschlossen gegenüber Komplementbildung und Allquantifizierung.
- Die entscheidbaren Relationen sind nicht abgeschlossen gegenüber existentielle- und Allquantifizierung.

6.68 Lemma

Sei R rekursiv aufzählbar. Dann gilt $R \leq_m K_0$.

Beweis: Sei $R \subseteq \mathbb{N}^n$ rekursiv aufzählbar. $R = dom(\varphi_a^{(n)})$ für ein $a \in \mathbb{N}$.

Es gilt

$$\begin{aligned} \vec{x} \in R \quad & \text{gdw} \ \varphi_a^{(n)}(\vec{x}) \downarrow \\ & \text{gdw} \ \varphi_{s_{n-1,1}(a,x_2,\ldots,x_n)}(x_1) \downarrow \\ & \text{gdw} \ (s_{n-1,1}(a,x_2,\ldots,x_n), x_1) \in K_0 \end{aligned}$$

Seien $f_1(x_1,\ldots,x_n) = s_{n-1,1}(a,x_2,\ldots,x_n)$ und $f_2(x_1,\ldots,x_n) = x_1$. $f_1, f_2 \in \mathcal{P}(\mathbb{N})$, also $R \leq_m K_0$.

Die Relation K_0 ist also die „schwerste“ r.a. Relation.

Vollständige rekursiv aufzählbare Relationen

6.69 Definition

Eine Relation $S \subseteq \mathbb{N}^n$ heißt vollständig bzgl. $\leq_m$, falls S rekursiv aufzählbar ist, und für jede andere rekursiv aufzählbare Relation $R \subseteq \mathbb{N}^m$ $(m \geq 1)$, $R \leq_m S$ gilt.

S ist eine „schwerste“ rekursiv aufzählbare Relation.

6.70 Satz Existenz vollständiger, r.a. Relationen.

K_0 ist vollständig für die rekursiv aufzählbaren Relationen.

6.71 Lemma

Ist S vollständig für rekursive aufzählbare Relationen und gilt $S \leq_m R$ für R rekursiv aufzählbar, dann ist auch R vollständig.

Beweis:

Sei T rekursiv aufzählbar, dann ist $T \leq_m S \leq_m R$. D. h. $T \leq_m R$.

6.72 Folgerung K ist auch vollständig.

Anwendung des s-m-n-Theorems. Zu zeigen:

$K_0 = \{(a,x) \in \mathbb{N}^2 : \varphi_a(x) \downarrow\} \leq_m K = \{a : \varphi_a(a) \downarrow\}$

Betrachte $Rzax$ gdw K_0ax. Dann ist R rekursiv aufzählbar, d. h. $R = dom(\varphi_b^{(3)})$. Das s-m-n-Theorem liefert:

$$\varphi_b^{(3)}(z,a,x) \downarrow \ \text{gdw} \ \varphi_{s_{2,1}(b,a,x)}(z) \downarrow$$

Vollständige rekursiv aufzählbare Relationen
Die Sätze von Rice

Dann gilt mit $z = s_{2,1}(b, a, x)$

$$\begin{aligned} \varphi_a(x) \downarrow \quad & \text{gdw } \varphi_b^{(3)}(s_{2,1}(b, a, x), a, x) \downarrow \\ & \text{gdw } \varphi_{s_{2,1}(b,a,x)}(s_{2,1}(b, a, x)) \downarrow \\ & \text{gdw } K s_{2,1}(b, a, x) \end{aligned}$$

Da $h(a, x) = s_{2,1}(b, a, x)$ primitiv rekursiv ist, gilt die Behauptung $K_0 \leq_m K$.

Methoden für den Nachweis von Unentscheidbarkeit und nicht rekursive Aufzählbarkeit.

6.73 Satz (Rice) Entscheidbare Indexmengen

Sei $S \subset \mathcal{R}_p^{(1)}(\mathbb{N})$ Menge einstelliger Funktionen. Dann ist die **Indexmenge** der Funktionen in S, die Menge $S_\mu = \{a \in \mathbb{N} : \varphi_a \in S\}$, genau dann entscheidbar, wenn $S = \varnothing$ oder $S = \mathcal{R}_p^{(1)}(\mathbb{N})$.
Also ist eine Indexmenge $R \subseteq \mathbb{N}$ entscheidbar gdw $R = \varnothing$ oder $\mathbb{N}$.

Beweis:

Ist $S = \varnothing$ oder $S = \mathcal{R}_p^{(1)}(\mathbb{N})$, so ist $S_\mu = \varnothing$ oder $S_\mu = \mathbb{N}$ und somit entscheidbar.
Sei $S \neq \varnothing, \neq \mathcal{R}_p^{(1)}(\mathbb{N})$. Angenommen S_μ ist entscheidbar.
O.b.d.A. $\uparrow \notin S$ (sonst statt S_μ wähle $\neg S_\mu$).

Die Sätze von RICE (Fort.)

Nach Vor. gibt es eine Funktion $f \in S$.
Definiere

$$F(x, y) = \begin{cases} f(y) & \text{falls } Kx \\ \uparrow & \text{sonst} \end{cases}$$

F ist berechenbar, also gibt es ein $a \in \mathbb{N}$ mit $F(x, y) = \varphi_a^{(2)}(y, x) = \varphi_{s_{1,1}(a,x)}^{(1)}(y)$. Die Funktion $g(x) = s_{1,1}(a, x)$ ist primitiv rekursiv und es gilt

$x \in K \Rightarrow \varphi_{g(x)} = f \in S \text{ gdw } g(x) \in S_\mu$
$x \notin K \Rightarrow \varphi_{g(x)} = \uparrow \notin S \text{ gdw } g(x) \notin S_\mu$

Es gilt somit $x \in K \text{ gdw } g(x) \in S_\mu$, d. h. $K \leq_m S_\mu$ ↯

Nichttriviale Indexmengen sind also nicht rekursiv entscheidbar. Sind sie überhaupt rekursiv aufzählbar?

6.74 Definition

$f : \mathbb{N} \to \mathbb{N}$ heißt **endliche Restriktion** von $g : \mathbb{N} \to \mathbb{N}$, falls $dom(f)$ endlich ist und $f \sqsubseteq g$, d. h.

$$dom(f) \text{ endlich und } f(x) \downarrow \Rightarrow (g(x) \downarrow \wedge f(x) = g(x))$$

Die Sätze von Rice (Fort.)

6.75 Satz (Rice-Shapiro) Rekursiv aufzählbare Indexmengen

Sei $S \subset \mathcal{R}_p^{(1)}(\mathbb{N})$. Ist die Menge $S_\mu = \{a \in \mathbb{N} : \varphi_a \in S\}$ rekursiv aufzählbar, dann folgt für $f \in \mathcal{R}_p^{(1)}(\mathbb{N})$:

- $f \in S \text{ gdw}$ es gibt eine endliche Restriktion g von f in S.

Insbesondere sind $spez(f)$ (f total berechenbar) und somit auch $\ddot{A}quiv$ nicht rekursiv aufzählbar.

Es gibt keine effektive Aufzählung aller while-Programme, die nur primitiv rekursive oder nur totale μ-rekursive-Funktionen berechnen.

Beweis:

Sei S_μ rekursiv aufzählbar und $f \in \mathcal{R}_p^{(1)}(\mathbb{N})$.

- Sei g endliche Restriktion von f und $g \in S$.

Angenommen $f \notin S$. Betrachte die Funktion:

$$F(x, y) = \begin{cases} f(y) & \text{falls } Kx \\ g(y) & \text{sonst} \end{cases}$$

Behauptung: F ist berechenbar. Da

$$F(x, y) = \begin{cases} g(y) & \text{falls } y \in dom(g) \\ sgn(succ(\varphi_x(x))) \cdot f(y) & \text{sonst} \end{cases}$$

Beweisfortsetzung

Sei $a \in \mathbb{N}$ mit $F(x, y) = \varphi_a^{(2)}(y, x) = \varphi_{s_{1,1}(a,x)}(y)$

$x \in K \Rightarrow F(x, \cdot) = f$

$x \notin K \Rightarrow F(x, \cdot) = g$

d. h. $x \notin K \text{ gdw } s_{1,1}(a, x) \in S_\mu$, d.h. S_μ ist nicht r.a. ↯

- Sei umgekehrt $f \in S$. Angenommen es gebe keine endliche Restriktion von f in S.

Betrachte die Funktion:

$$F(x, y) = \begin{cases} f(y) & \text{falls } \neg\Phi_x(x) \leq y \\ \uparrow & \text{sonst} \end{cases}$$

F ist berechenbar.
Sei $a \in \mathbb{N}$ mit $F(x, y) = \varphi_a^{(2)}(y, x) = \varphi_{s_{1,1}(a,x)}(y)$.
Es gilt:

- $x \in K \Rightarrow F(x, \cdot)$ ist eine endlichen Restriktion von f:

Da $\varphi_x(x) \downarrow$, d. h. $\Phi_x(x) = t_0$ und für

$y < t_0 \quad F(x, y) = f(y)$

$y \geq t_0 \quad F(x, y) \uparrow$

- $x \notin K \Rightarrow F(x, \cdot) = f$, da $\neg\Phi_x(x) \leq y$ gültig.

$x \notin K \text{ gdw } s_{1,1}(a, x) \in S_\mu$, d.h. S_μ ist nicht r.a. ↯

Nicht rekursiv aufzählbare Mengen

6.76 Folgerung Der Satz erlaubt uns zu zeigen, dass gewisse Indexmengen nicht r.a. sind.
Ist nämlich A eine Indexmenge partiell rekursiver Funktionen und $p \in A$ für die gilt:
a) Es gibt ein $q \in \neg A$ mit $\varphi_p \sqsubseteq \varphi_q$ oder
b) Es gibt kein Index einer endliche Restriktion von φ_p in A.

Insbesondere sind folgende Mengen A_i nicht rekursiv aufzählbar:

$$\begin{aligned}
A_0 &= \{x \in \mathbb{N} \mid \varphi_x = \uparrow, \text{d. h. } dom(\varphi_x) = \varnothing\} \\
A_1 &= \{x \in \mathbb{N} \mid dom(\varphi_x) \text{ endlich}\} \\
A_2 &= \{x \in \mathbb{N} \mid im(\varphi_x) \text{ endlich}\} \\
A_3 &= \{x \in \mathbb{N} \mid dom(\varphi_x) = \mathbb{N}, \text{d. h. } \varphi_x \text{ total}\} \\
A_4 &= \{x \in \mathbb{N} \mid im(\varphi_x) = \mathbb{N}, \text{d. h. } \varphi_x \text{ surjektiv}\} \\
\neg A_5 &= \{x \in \mathbb{N} \mid a \in dom(\varphi_x)\ a \text{ fest, d. h. } \varphi_x(a) \downarrow\} \\
\neg A_6 &= \{x \in \mathbb{N} \mid a \in im(\varphi_x)\ a \text{ fest, d. h. } \exists y \varphi_x(y) = a\}
\end{aligned}$$

Wende Satz 6.75 an.

Beachte $\neg A_0, \neg A_5, \neg A_6$ sind rekursiv aufzählbar.

Existenz von Programmen (berechenbare Funktionen) mit bestimmten Eigenschaften

6.77 Satz Fixpunktsatz - Rekursionssatz

a) **FPS.** Zu jedem $f \in \mathcal{R}^{(1)}(\mathbb{N})$ (totale μ-rekursive Funktion) gibt es ein $p \in \mathbb{N}$ mit $\varphi_{f(p)} = \varphi_p$.
D.h. die „Programme" p und $f(p)$ berechnen die gleiche Funktion.
Es muss **nicht** $f(p) = p$ gelten.

b) **RS.** Zu jeder Funktion $G \in \mathcal{R}^{(2)}(\mathbb{N})$, $G : \mathbb{N}^2 \to \mathbb{N}$, gibt es ein $q \in \mathbb{N}$ mit $\varphi_q = G(\cdot, q)$.
D. h. $\varphi_q(x) = G(x, q)$ für alle $x \in \mathbb{N}$.

Beweis:

a) Betrachte die Funktion $F(x, y) = \varphi_{f(s_{1,1}(y,y))}(x)$

$F \in \mathcal{R}^{(2)}(\mathbb{N})$. Sei q Index von F, d. h.

$$F(x, y) = \varphi_q^{(2)}(x, y) \underset{s-m-n}{=} \varphi_{s_{1,1}(q,y)}(x)$$

Setze $p = s_{1,1}(q, q)$. Dann gilt
$\varphi_{f(p)}(x) = \varphi_{f(s_{1,1}(q,q))}(x) = F(x, q) = \varphi_{s_{1,1}(q,q)}(x)$
$= \varphi_p(x)$.
Beachte: wählt man für $f(y) = s_{1,1}(y, y)$, so gibt es ein p mit $\varphi_{s_{1,1}(p,p)} = \varphi_p$.

Anwendungen

b) Sei $G \in \mathcal{R}_p^{(2)}(\mathbb{N})$ und a Index von G, d. h.
$G(x, y) = \varphi_a^{(2)}(x, y) \underset{s-m-n}{=} \varphi_{s_{1,1}(a,y)}(x)$.
Sei $f(y) = s_{1,1}(a, y)$. Nach a) gibt es $q \in \mathbb{N}$ mit

$$\varphi_q(x) = \varphi_{f(q)}(x) = \varphi_{s_{1,1}(a,q)}(x) = G(x, q)$$

6.78 Folgerung Existenz spezieller Programme!

• Es gibt ein while-Programm, das für jede Eingabe seine eigene Codezahl ausgibt.
Sei $G(x, y) = f^{(2)}_{proj(2)}(x, y) = y$. Dann liefert RS $q \in \mathbb{N}$ mit

$$\varphi_q(x) = G(x, q) = q$$

• Es gibt eine primitiv rekursive Funktion f mit $f : \mathbb{N} \to \mathbb{N}$, so dass $\forall x \in \mathbb{N}.\ dom(\varphi_{f(x)}) = \{x^2\}$.
„Implizite Definition von Programmen"

Sei $h(x, y) = \mu z.(\chi_{\{x^2\}}(y) = 1) = \begin{cases} 0 & y = x^2 \\ \uparrow & \text{sonst} \end{cases}$

$h \in \mathcal{R}_p$, d. h. $h(x, y) = \varphi_a^{(2)}(y, x) = \varphi_{s_{1,1}(a,x)}(y)$.
Wähle $f(x) = s_{1,1}(a, x)$.

• FPS liefert sogar Existenz von $p_0 \in \mathbb{N}$ mit
$\varphi_{p_0} = \varphi_{f(p_0)}$, d. h. $dom(\varphi_{p_0}) = dom(\varphi_{f(p_0)}) = \{p_0^2\}$

6.5 Die Churchsche These

Maschinennahe Programme: Register- und Turing-Maschinen.

Bisher: $\mathcal{R}_p(\mathbb{N})$=While-programmierbar über N.
=While-rekursiv programmierbar über N.

Wichtige Ergebnisse

Existenz **universeller** berechenbarer-Funktionen:
$\Phi^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$ **Zeitkomplexitätsfunktion.**

- $\Phi^{(n)}(p, \vec{x}) = \mu t.\text{first}(i^t(\text{inp}^{(n)}(p, \vec{x}))) = 0$
  $\text{inp}^{(n)}, i, \text{first} \in \mathcal{P}(\mathbb{N})$

$\varphi^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$ **universelle Funktion** für n-stellige Funktionen.

- $\varphi^{(n)}(p, \vec{x}) = \text{out}(i^{\Phi^{(n)}(p,\vec{x})}(\text{inp}^{(n)}(p, \vec{x})))$
  Kleenesche Normalform für μ-rekursive Funktionen.
- $\Phi^{(n)}, \varphi^{(n)} \in \mathcal{R}_p(\mathbb{N})$ (d. h. berechenbar)
- $\forall f \in \mathcal{R}_p^{(n)}(\mathbb{N}) \quad \exists p \in \mathbb{N} : f = \varphi_p^{(n)}$

Jede berechenbare Funktion hat einen „Index".

Sätze: „Programmtransformationen"

s-m-n Theorem: $\exists s_{m,n} \in \mathcal{P}^{m+1} \varphi_p^{n+m}(\vec{x}, \vec{y}) = \varphi^n_{s_{m,n}(p,\vec{y})}(\vec{x})$

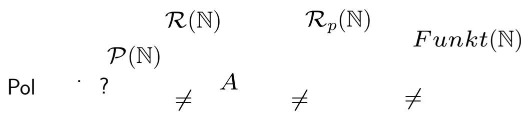
Fixpunktsatz: $\forall f \in \mathcal{R}^{(1)}(\mathbb{N}) \ \ \exists p \in \mathbb{N} \ \ \varphi_{f(p)} = \varphi_p$

Rekursionstheorem: $\forall G \in \mathcal{R}_p^{(2)}(\mathbb{N}) \ \ \exists p \in \mathbb{N} \ \ \varphi_p = G(\cdot, p)$

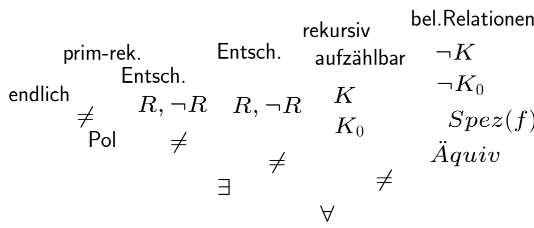
Die Sätze von Rice: Entscheidbarkeit und r.a. von Indexmengen.

Klassifizierung von Funktionen und Relationen

$f : \mathbb{N}^n \to \mathbb{N}$



$R \subset \mathbb{N}^n$ Relationen:



Churchsche These

Die Klasse der effektiv berechenbaren Funktionen ist genau die Klasse der μ-rekursiven Funktionen. Jede Formalisierung von berechenbaren Funktionen liefert die gleiche Klasse.

Wir werden einige dieser Formalisierungen kurz vorstellen.

6.79 Definition Register-Maschinen (goto-Programme über N)

Goto-Programme über der Variablenmenge $V = \{V_0, \ldots, V_n\}$ sind markierte Befehlsfolgen der Form

$$\begin{array}{ll} \mathcal{P} :: & 0 : B_0 \\ & 1 : B_1 \\ & \quad \vdots \\ & L : B_L \end{array}$$

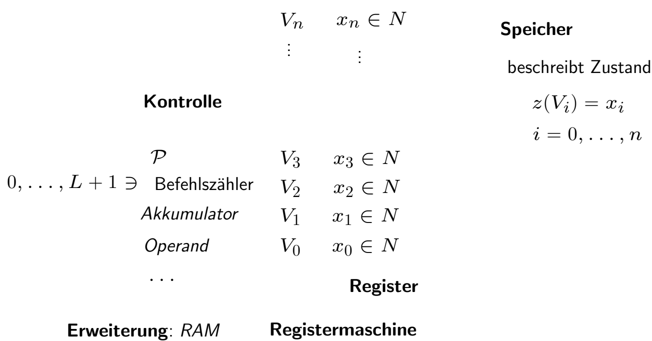
Mit **Befehlen** $B_i, i \in \{0, \ldots, L\}$ einer der Formen

$\bullet V_i := s(V_i)$ $\bullet V_i := p(V_i)$ $\bullet$**if** $V_i = 0$ **then goto** l_1 **else goto** l_2

mit $V_i \in V, l_1, l_2 \in \{0, \ldots, L+1\}$ (**Marken**).

Die intendierte Semantik von $\boldsymbol{s}, \boldsymbol{p}$ ist die Nachfolger- bzw. die Vorgängerfunktion auf $\mathbb{N}$.

Register-Maschinen Semantik



Interpretersemantik:

$I_{\mathcal{P}}(l, z) : \{0, \ldots, L+1\} \times \mathcal{Z} \to \{0, \ldots, L+1\} \times \mathcal{Z}$

Startzustand: $(0, z)$, Eingaben $z(V_i) = x_i \in \mathbb{N}$

$$I_{\mathcal{P}}(l, z) = \begin{cases} (l+1, z(V_i / z(V_i) + 1)) & l : V_i := s(V_i) \in \mathcal{P} \\ (l+1, z(V_i / z(V_i) \dot{-} 1)) & l : V_i := p(V_i) \in \mathcal{P} \\ (l_1, z) & l : \textbf{if } V_i = 0 \textbf{ then goto } l_1 \textbf{ else goto } l_2 \in \mathcal{P} \\ & \wedge z(V_i) = 0 \\ (l_2, z) & l : \textbf{ if } V_i = 0 \textbf{ then goto } l_1 \textbf{ else goto } l_2 \in \mathcal{P} \\ & \wedge z(V_i) \neq 0 \\ (l, z) & l = L+1 \text{ oder } l \text{ kein Label in } \mathcal{P} \ \textbf{(Stopp)} \end{cases}$$

Register-Maschinen berechenbare Funktionen

Programm $\mathcal{P}$ stoppt aus Startzustand z gdw keine Befehlsausführung mehr möglich.

Ein- Ausgabevereinbarungen für die Berechnung von Funktionen $f : \mathbb{N}^l \to \mathbb{N}$: $\mathcal{P}$ **berechnet** f gdw

i) Die Rechnung stoppt aus Anfangszustand $z(V_i) = x_i$, $i = 1, \ldots, l$, $z(V_i) = 0$ sonst gdw $(x_1, \ldots, x_l) \in dom(f)$.

ii) Gilt $(x_1, \ldots, x_l) \in dom(f)$, $y = f(x_1, \ldots, x_l)$, so stoppt $\mathcal{P}$ in einem Zustand z' mit $z'(V_0) = y$. Also gilt:

$$\exists t \in \mathbb{N} : I^t_{\mathcal{P}}(0, z) = (L+1, z')$$

6.80 Beispiel Einfache RM bzw. goto-Programme

Sei S festes Register mit Inhalt 0, d. h. $z(V_s) = 0$

a) Register „leeren"

$$\begin{array}{ll} V \Leftarrow 0 :: & 0 : V := p(V) \\ & 1 : \textbf{if } V = 0 \textbf{ then goto } 2 \textbf{ else goto } 0 \end{array}$$

b) $Z \Leftarrow Z + 1$, $Z \Leftarrow Z \dot{-} 1$ sind leicht anzugeben.

Einfache RM bzw. goto-Programme

c) „Inhalt umspeichern“

$Z \Leftarrow Y$ ::	$0 : Z \Leftarrow 0$
copy Y nach Z	1 : **if** $Y = 0$ **then goto** 5 **else goto** 2
Hilfsregister U	$2 : Y := p(Y)$
initialisiert mit 0	$3 : U := s(U)$
unbedingter Sprung:	4 : **if** $V_s = 0$ **then goto** 1 **else goto** 1
goto 1 (Abkürzung)	
	5 : **if** $U = 0$ **then goto** 10 **else goto** 6
	$6 : U := p(U)$
	$7 : Z := s(Z)$
	$8 : Y := s(Y)$
	9 : **goto** 5

6.81 Lemma

- Jede μ-rekursive Funktion ist goto-berechenbar.
- Jede goto-berechenbare Funktion ist μ-rekursiv.

Beweisidee:

- Zeige die Grundfunktionen sind goto-berechenbar.

$f = go(h_1, \ldots, h_m), \quad f = R(g, h), \quad f = \mu.g$

Lassen sich durch Goto-Programme berechnen, falls $g, h_1, \ldots, h_m, h$ goto-berechenbar.

- Zeige die Funktion $I_{\mathcal{P}}$ lässt sich durch eine primitiv rekursive Funktion simulieren. Dann Iteration und Minimierung.

Turingmaschinen (nach A. Turing)

$\Sigma = \{b_1, \ldots, b_r\}$ Alphabet, Leersymbol $\Box \notin \Sigma, a \in \Sigma$

Band Arbeitsfeld

a ...

q_0 q_1 q_2 Steuereinheit $Q = \{q_0, q_1, \ldots\}$ endliche Zustandsmenge

Zu jedem Zeitpunkt sind nur endlich viele Felder nicht mit $\Box$ belegt. Es gibt somit stets zusammenhängenden Block endlicher Länge, der das A-Feld enthält und außerhalb davon nur Leerzeichen vorkommen.

Erlaubte Operationen:

In Abhängigkeit vom Zustand und Inhalt des A-Felds schreibe Zeichen ins A-Feld, bewege Lese-Schreibkopf um ein Feld nach links (L), rechts (R) oder bleibe darauf (S), ändere Zustand.

Beschreibung durch „**Übergangsfunktion**“
$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$, wobei Q endliche Zustandsmenge und Γ Bandalphabet sind.

Turingmaschinen (Forts.)

6.82 Definition

Eine Turingmaschine T ist ein 6 -Tupel $T = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit folgenden Bestandteilen:

- Q ist endliche **Zustandsmenge**.
- Σ Eingabealphabet mit $\Box \notin \Sigma$. **Eingabezeichen.**
- Γ Bandalphabet mit $\Sigma \subseteq \Gamma$ und $\Box \in \Gamma$. **Bandzeichen**.
- q_0 ist der **Startzustand**.
- $F \subseteq Q$ Menge der **Endzustände**.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ (oft als total verlangt) genügt $dom(\delta) = (Q \backslash F) \times \Gamma$. **Übergangsfunktion**.
  Wird oft als Tafel oder Tabelle angegeben.

Ein **Bandzustand** von T ist ein Tripel (q, x, β) mit $q \in Q$ (aktueller Zustand), $x \in \mathbb{Z}$ (aktuelle Kopfposition), $\beta : \mathbb{Z} \to \Gamma$ totale Funktion (aktueller Bandinhalt) mit $\beta(y) = \Box$ für alle bis auf endlich viele $y \in \mathbb{Z}$.

Turingmaschinen (Forts.)

T überführt den Bandzustand (q, x, β) in den Bandzustand (q', x', β') (**Folgezustand**), falls

- $\delta(q, \beta(x)) = (q', \beta'(x), M)$
- $\beta'(y) = \beta(y)$ für alle $y \neq x$ — Folgezustand
- $x' = \begin{cases} x - 1 & \text{falls } M = L \\ x + 1 & \text{falls } M = R \\ x & \text{falls } M = S \end{cases}$

Eine **Rechnung** von T ist eine endliche Folge von Bandzuständen $(z_0, \ldots, z_n)$, so dass T für alle $0 \leq i < n$ den Zustand z_i in z_{i+1} überführt.

Eine Rechnung heißt haltend, falls $z_n = (q, x, \beta) \wedge q \in F$.

6.83 Beispiel

$\Sigma = \{1, 2\}$, $\Gamma = \Sigma \cup \{\Box\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$

δ	$\Box$	1	2
q_0	$(q_0, \Box, R)$	$(q_1, \Box, R)$	$(q_1, \Box, R)$
q_1	$(q_2, \Box, S)$	$(q_1, \Box, R)$	$(q_1, \Box, R)$
q_2	$(q_2, \Box, S)$	$(q_2, 1, S)$	$(q_2, 2, S)$

Andere Beschreibungen von δ möglich: z.B.

Fünftupel $\{q\ b\ q'\ b'\ M : q \in Q, b \in \Gamma\}$

Beispiele von Turingmaschinen

Beispiel Rechnung:

-2 -1 0 1 2 3 4 5 6 7

1 2 2

q_0

q_2

q_1

Anfangszustand $z_0 = (q_0, 0, \beta)$ wobei $\beta(2) = 1$
$\beta(3) = 2$
$\beta(5) = 2$
sonst $\Box$

$z_1 = (q_0, 1, \beta)$
$z_2 = (q_0, 2, \beta)$
$z_3 = (q_1, 3, \beta_1)$ mit $\beta_1(3) = 2 = \beta_3(5)$ sonst $\Box$
$z_4 = (q_1, 4, \beta_2)$ mit $\beta_3(5) = 0$ sonst $\Box$
$z_5 = (q_2, 4, \beta_2)$ $\ni q_2$ haltend oder „Haltezustand"
$z_6 = (q_2, 4, \beta_2)$ „Endzustand"

Wirkung: TM sucht rechts vom A-Feld $w \in \Sigma^*$ als Block und löscht es. Bleibt auf Leerzeichen hinter w stehen, falls $w \in \Sigma^+$ existiert. Stoppt nicht, falls auf AFeld und rechts davon lauter $\Box$-Zeichen sind.

Turing-berechenbare Funktionen

Unäre Codierung von Zahlen $n \to \underbrace{|||\ldots|}_{n}$

6.84 Definition

Eine Funktion $f : \mathbb{N}^n \to \mathbb{N}$ heißt **Turing-berechenbar**, falls es eine TM T mit Eingabealphabet $\{|, \$\}$ gibt, so dass der Bandzustand $(q_0, 0, \beta)$ mit

- $\beta(i) = \Box$ für $i < 0$ und $i > x_1 + x_2 + \cdots + x_n + n$
- $\beta(0) = \beta(x_1 + 1) = \beta(x_1 + x_2 + 2) = \cdots = \beta(x_1 + \cdots + x_n + n) = \$$
- $\beta(i) = |$ für alle anderen i

genau dann zu einer haltenden Rechnung ergänzt werden kann, wenn $f(x_1, \ldots, x_n) \downarrow$ und, ist in diesem Fall (q, i, β') der Zustand, in dem die Rechnung hält, dann ist die Anzahl der Striche $|$, die in $\beta'(i + 1), \beta'(i + 2), \ldots$ unmittelbar aufeinanderfolgen, gleich $f(x_1, \ldots, x_n)$.

Turing-berechenbare Funktionen (Forts.)

.... \$ x_1 viele Striche \$ .... \$ x_n viele Striche \$ ....

q_0

T

.... $f(x_1, \ldots, x_n)$ viele Striche kein Strich ....

$q \in F$

Beispiele

6.85 Beispiel

1. Vorgänger und Nachfolger: $v(x) = n - 1,\ s(n) = n + 1$

   $v ::$ \$ n-Striche \$ $\rightsquigarrow$ \$ $n \dot{-} 1$-Striche \$

   q_0

   $q \in F$

   $$\begin{array}{llll} \delta(q_0, \$) & = (q_1, \Box, R) & \Sigma & = \{|, \$\} \\ \delta(q_1, |) & = (q_3, \$, S) & \Gamma & = \{|, \$, \Box\} \\ \delta(q_1, \$) & = (q_2, \$, L) & Q & = \{q_0, q_1, q_2, q_3\} \\ \delta(q_2, \Box) & = (q_3, \$, S) & F & = \{q_3\} \end{array}$$

   $s ::$ \$ n-Striche \$ $\rightsquigarrow$ \$ $n + 1$-Striche \$

   q_0

   $q \in F$

   $$\begin{array}{llll} \delta(q_0, \$) & = (q_0, |, L) & \Sigma & = \{|, \$\} \\ \delta(q_0, \Box) & = (q_1, \$, S) & \Gamma & = \{|, \$, \Box\} \\ & & Q & = \{q_0, q_1\} \\ & & F & = \{q_1\} \end{array}$$

Beispiele (2)

2. Suche rechts vom A-Feld erstes Vorkommen von \$, bleibe dort stehen.

   nicht \$ \$ $\rightsquigarrow$ nicht \$ \$

   q_0

   $q \in F$

   $\delta(q_0, b) = (q_1, b, R) \quad b \in \Gamma$
   $\delta(q_1, b) = (q_1, b, R) \quad b \in \Gamma \backslash \$$
   $\delta(q_1, \$) = (q_2, \$, S)$
   SL\$. Analog SR\$

3. Verschiebe Block \$$n$-Striche\$ um ein Feld nach links

   $0 \; 1 \; 2 \; \cdots n+2 \; \cdots$ $\quad 0 \quad n+1$
   \$ n Striche \$ Zeichen $\rightsquigarrow$ \$ n Striche \$ Zeichen

   q_0

   $q \in F$

Beispiele (2) (Forts.)

$\Sigma = \{|, \$\}$
$\Gamma = \Sigma \cup \{\Box\}$
$Q = \{q_0, \ldots, q_5\}$
$F = \{q_5\}$

$q_0 \; b \; \$ \; R \; q_1 \quad b \in \Gamma$
$q_1 \; b \; b \; R \; q_2$
$q_2 \; | \; | \; L \; q_3$
$q_2 \; \$ \; \Box \; L \; q_4$
$q_3 \; b \; | \; R \; q_1$
$q_4 \; b \; \$ \; R \; q_5$
$q_5 \; b \; b \; S \; q_5$

VL. Analog VR (verschiebe nach rechts).
$\delta(q_2, \$) = (q_4, \Box, L)$
q_3 merkt sich |
q_4 merkt sich \$

Strategie:

$q_0 \quad 0$
q_1
$q_3 \quad q_2$
$q_4 \quad q_1$

Simulation von RM-Programme durch TM

6.86 Lemma

• Jede RM (goto)-berechenbare Funktion lässt sich durch eine TM berechnen. Also ist jede μ-rekursiv Funktion TM-berechenbar.

• Jede Turing-berechenbare Funktion ist μ-rekursiv.

Beweisidee:

Simuliere Berechnung des goto-Programms über $V_0, \ldots, V_m$, $f : \mathbb{N} \to \mathbb{N}$. Speichere Zustand $z : V \to \mathbb{N}$ als

\$ \$ x_1-Striche \$ x_2-Striche \$ $\cdots$ \$ x_n-Striche \$ \$ $\cdots$ \$

$m + 2$ \$-Zeichen

$z(V_2) = x_2$

q_0

Ein-Befehl wird durch mehrere TM-Schritte simuliert.
Die Zustände entsprechen Marken im Goto-Programm.
$V_i := s(V_i)$

- Verschiebe die Blöcke vor V_i jeweils um ein Feld nach links wie oben.
- Wende s TM an.
- SL\$ i-mal.

Simulation von TM durch μ rekursive Funktionen

Simulation der Überführungsfunktion einer Turing-Maschine durch eine primitiv-rekursive Funktion $i_T : \mathbb{N} \to \mathbb{N}$, die auf geeignet codierten Bandzuständen arbeitet (q, x, β).

Dann wie üblich.

Wir haben somit weitere Charakterisierungen der μ-rekursiven Funktionen, die die Churchsche These untermauern.

Man kann für $\mathcal{P}(\mathbb{N})$ (primitiv-rekursiven Funktionen) ebenfalls eine Charakterisierung mit Hilfe einfacher Programmiersprachen finden.

z. B. For-Programme über $\mathbb{N}$

Anweisungsfolgen:

Anweisung: Zuweisung, Test oder For-Schleife der Form:

for $I = 0$ **to** J **do** α **end**;

$I, J \in V$ α For-Programm über V, das keine Zuweisung der Form $I := t$ oder $J := t$ mit Term t enthält (Schleife wird genau $z(J)$ mal ausgeführt, dabei wird stets α ausgeführt und I um 1 erhöht).

6.6 Berechenbarkeit auf Zeichenreihen Wortfunktionen

Wortfunktionen: $f : (\Sigma^*)^n \to \Sigma^*$

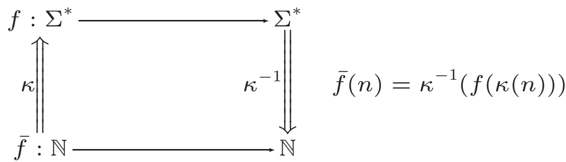
Wortrelationen $R \subseteq (\Sigma^*)^n$ **Sprachen**$R \subseteq \Sigma^*$

Bisher: Funktionen, Relationen auf $\mathbb{N}$: μ-rekursive Funktionen.

Turing-Maschinen und While-Programme sind für beliebige Alphabete bzw. beliebige Strukturen definiert.

Verallgemeinerung der Ergebnisse der Rekursionstheorie, insbesondere über Entscheidbarkeit und Nichtentscheidbarkeit auf Wortfunktionen und Relationen.

1. Möglichkeit: Codierung von Σ^* in $\mathbb{N}$. Einfache effektive Codierungen: z. B. Folgencodierungsfunktion oder Interpretation als Zahl (binäre-, dezimale-Darstellung).



$\bar{f}(n) = \kappa^{-1}(f(\kappa(n)))$

Berechenbarkeit auf Zeichenreihen (2)

Definition $f \in \mathcal{R}_p(\Sigma)$ gdw $\bar{f} \in \mathcal{R}_p(\mathbb{N})$.

Zeige: Unabhängig von der gewählten effektiven Codierung.

2. Möglichkeit: $\Sigma = \{a_1, \ldots, a_n\}$ $(n \geq 1)$

While-Programme:

Betrachte die Algebra

$String = (\Sigma^*, \varepsilon, succ_{a_1}, \ldots, succ_{a_n}, pred)$ mit

- $succ_a(u) = au \quad (a \in \Sigma)$
- $pred(au) = u \quad pred(\varepsilon) = \varepsilon$

Ordnungen auf Σ^*: $\leq_{llex}$ Länge-Lexikographisch,

d. h. $u \leq_{llex} v$ gdw $|u| < |v|$ oder
$|u| = |v| \wedge u \leq_{lex} v$

Wobei $\leq_{lex}$ lex. Ordnung, die von lin. Ordnung auf Σ induziert wird (z. B. $a_1 < a_2 < a_3 \cdots < a_n$).

Beachte: $|\cdot| : \Sigma^* \to \Sigma^*$ $|u| = a_1^{|u|}$ und $\chi_{\leq llex}$ sind while-berechenbar.

3. Möglichkeit: **μ-rekursive Funktionen über Σ^***: $a \in \Sigma$

- $f^{(n)}_{NULL}(\vec{w}) = \varepsilon, f^{(n)}_{SUCC_a}(\vec{w}) = aw_1$ $(\vec{w} = (w_1, \ldots, w_n))$
- $f^{n}_{PROJ(i)}$ wie bisher.

Berechenbarkeit auf Zeichenreihen (3)

Komposition: $f \circ (g_1, \ldots, g_n)$ wie bisher.

Primitive Rekursion: $f = R_\Sigma(g, h_1, \ldots, h_n)$, falls

- $f(\vec{u}, \varepsilon) = g(\vec{u}, \varepsilon)$
- $f(\vec{u}, a_i v) = h_i(\vec{u}, f(\vec{u}, v), v)$

Minimierung: $f(\vec{u}) = \mu_{llex} v \cdot g(\vec{u}, v) = \varepsilon$

$f(\vec{u}) = w$ $llex$-minimal mit $g(\vec{u}, w) = \varepsilon$, sofern ein solches existiert.

4. Möglichkeit: **RM (Goto-Programme)**: $z(V_i) \in \Sigma^*$

Befehle:

- $X := s(a, X)$ Wirkung wie $succ_a$ in Σ^*
- $X := p(X)$ Wirkung wie pred in Σ^*
- **if** $X = \varepsilon$ **then** **goto** l_1 **else** **goto** l_2
- Test (Anfangsbuchstabe ist $a \in \Sigma$):
  if $AB(X) = a$ **then** **goto** l_1 **else** **goto** l_2

Berechenbarkeit auf Zeichenreihen (4)

5. Möglichkeit:**Turing-Maschinen**:
$T = (Q, \Sigma, \Gamma \supseteq \Sigma \cup \{\Box\}, \delta, q_0, F \subseteq \Gamma)$

… u a v …

q

$u, v \in \Sigma^*$

Block $u\ a\ v \in \Gamma^*$

außerhalb nur $\Box$-Zeichen

Konfiguration: $u\ q\ a\ v \in \Gamma^* \cdot Q \cdot \Gamma^+ \quad (\Gamma^+ = \Gamma^* \backslash \{\varepsilon\})$

Zwei Konfigurationen heißen **äquivalent**, falls sie sich nur durch Blöcke von $\Box$-Zeichen davor und danach unterscheiden.

Anfangskonfigurationen: $q_0 \Box w \qquad w \in \Sigma^*$

w

q_0

Berechenbarkeit auf Zeichenreihen (5)

Folgekonfigurationen

k' ist Folgekonfiguration von $k : k \underset{T}{\vdash} k'$, falls gilt

k	$\delta(q,a)$	k'	
$u\ q\ a\ v$	(q', a', S)	$u\ q'\ a'\ v$	
$u\ q\ a\ v$	(q', a', R)	$u\ a'\ q'\ v$	$v \in \Gamma^+$
$u\ q\ a$	(q', a', R)	$u\ a'\ q'\ \Box$	
$u\ b\ q\ a\ v$	(q', a', L)	$u\ q'\ b\ a'\ v$	$b \in \Gamma$
$q\ a\ v$	(q', a', L)	$q'\ \Box\ a'\ v$	

Eine **Rechnung** einer TM T ist Folge von Konfigurationen $(k_0, \ldots, k_n)$ mit $k_i \underset{T}{\vdash} k_{i+1}$. Sie ist haltend, falls k_n eine Endkonfiguration ist, d. h. $k_n = u\ q\ v$ mit $q \in F$. Schreibe $k_0 \overset{*}{\vdash}_T k_n$

Eine **Berechnung** einer TM T ist eine Rechnung wobei k_0 eine Anfangskonfiguration $(k_0 = q_0 \Box w) w \in \Sigma^*$ ist.

TM-berechenbare Funktionen: $f : (\Sigma^*)^n \to \Sigma^*$ ist TM-berechenbar, falls es eine TM T gibt die f berechnet, d. h.

a) T stoppt für Anfangskonfiguration $k_0 = q_0 \Box x_1 \Box x_2 \Box \cdots \Box x_n \Box$ $\mathrm{gdw}\ (x_1, \ldots, x_n) \in dom(f)$

b) Gilt $(x_1, \ldots, x_n) \in dom(f)$ und $y = f(x_1, \ldots, x_n)$, so hat T beim Stopp die Konfiguration $\Box^i\ q \Box x_1 \Box \cdots \Box x_n \Box y \Box^j$, für geeignete $i, j \in \mathbb{N}$.

Berechenbarkeit auf Zeichenreihen (6)

6.87 Satz

$f : (\Sigma^*)^n \to \Sigma^*$, dann sind äquivalent

- $f \in \mathcal{R}_p(\Sigma)$, d. h. f ist μ-rekursiv.
- f ist while-programmierbar über String.
- f ist RM-(goto)-berechenbar.
- f ist TM-berechenbar.

Existenz universeller Funktionen, universeller Programme und universeller Maschinen wie bisher.

- Relationen: Entscheidbarkeit, rek-Aufzählbarkeit $R \subseteq (\Sigma^*)^n$
- R entscheidbar $\mathrm{gdw}\ \chi_R \in \mathcal{R}_p(\Sigma), \chi_R(\vec{w}) = \begin{cases} \varepsilon & w \notin R \\ a_1 & w \in R \end{cases}$
- R rekursiv-aufzählbar $\mathrm{gdw}\ R = dom(f),\ f \in \mathcal{R}_p(\Sigma)$.
- Halteproblem: $K_0 = \{(T, w) \mid T$ mit Anfangskonfiguration $q_0 \Box w$ hält, d. h. Berechnung mit Endkonfiguration$\}$

Ist nicht entscheidbar.

Bisherige Ergebnisse lassen sich übertragen.

Insbesondere: Reduzierbarkeit $\leq_m$.

Turing-Maschinen als Akzeptoren von Sprachen und als entscheidende Automaten

6.88 Definition Akzeptierende und erkennende TM

Sei $T = (Q, \Sigma, \Gamma \supseteq \Sigma \cup \{\Box\}, \delta, q_0, F)$

- T **akzeptiert** die Sprache $L \subseteq \Sigma^*$ gdw für $w \in \Sigma^* : q_0 \Box w \overset{*}{\vdash}_T u\ q\ v$ mit $q \in F$ $\mathrm{gdw}\ w \in L$, d. h. es gibt haltende Berechnung aus $q_0 \Box w$ $\mathrm{gdw}\ w \in L$, $L = L(T)$.
- T **entscheidet** die Sprache $L \subseteq \Sigma^*$ gdw für jede Eingabe $w \in \Sigma^*$ hält T: $q_0 \Box w \overset{*}{\vdash}_T u\ q\ v$ mit $q \in F$ und $w \in L$, so $q = q_y$ $\quad w \notin L$, so $q = q_n$ wobei $q_y, q_n \in F$ spezielle „Ja"-, „Nein"- Zustände sind.

6.89 Lemma

- $L \subseteq \Sigma^*$ ist entscheidbar gdw es gibt eine TM T, die L entscheidet.
- $L \subseteq \Sigma^*$ ist rekursiv aufzählbar gdw es gibt eine TM T, die L akzeptiert, d. h. $L = L(T)$.

Beachte: Andere Konventionen sind möglich. Andere TM: Mehrband TM, δ unvollständig, Band einseitig unendlich, mehrspurig, nicht deterministisch.

Beispiele

Turing-Programme

- Turing Befehl hat die Form
  $B \equiv Op$ $\quad Op \in \Gamma \dot\cup \{R, L, \text{stopp}\}$
  $B \equiv q$ $\quad q \in Q$ unbedingter Sprung
  $B \equiv a, q$ $\quad a \in \Gamma, q \in Q$ bedingter Sprung nach q, falls a in A-Feld
- Turing Programm ist endliche Folge markierter Befehle
  $Q = \{q_0, q_1, \ldots, q_n\}$, $q_i \neq q_j$ für $i \neq j$
  TP:: $q_0 : B_0$ $\quad B_i$ Turing-Befehl
  $q_1 : B_1$
  $\vdots$
  $q_n : B_n$
- Semantik eines T-Programms durch Angabe der TM
  $T = (Q', \Sigma, \Gamma, \delta, q_0, F)$, $a \in \Gamma$, $Q' = Q \cup \{q_{n+1}\}$
  $$\begin{aligned} \delta(q_i, a) &= (q_{i+1}, a', S) & B_i &\equiv a' \in \Gamma \\ &= (q_{i+1}, a, M) & B_i &\equiv M \in \{L, R\} \\ &= (q_{i+1}, a, S) & B_i &\equiv a', q \quad a \neq a' \\ &= (q, a, S) & B_i &\equiv a, q \\ &= (q_{n+1}, a, S) & B_i &\equiv \text{ stopp oder } i = n+1 \end{aligned}$$
  $F = \{q_{n+1}\}$
- Eigenschaft:Jede TM kann durch ein äquivalentes T-Programm beschrieben werden.

Beispiele

Suche Links von AF das erste Vorkommen von $\Box$ ... Rechts ...

$SL\,\Box$:	L	$SR\,\Box$:	R
	$\Box, Fin$		$\Box, Fin$
	$SL\,\Box$		$SR\,\Box$
Fin :	Stopp	Fin :	Stopp

TM, die die Menge der Palindrome über $\{a,b\}^*$ entscheidet
$L = \{w \in \{a,b\}^* : w = w^{mi}\}$

q_0:	R	q_b:	$\Box$
	$\Box : q_y$		$SR\,\Box$
	$a : q_a$		L
	$b : q_b$		$\Box : q_y$
			$b : q$
			$a : q_n$
q_a:	$\Box$	q:	$\Box$
	$SR\,\Box$		$SL\,\Box$
	L		q_0
	$\Box : q_y$		
	$a : q$		
	$b : q_n$		

Diese Turing Programm hält für jede Eingabe $w \in \Sigma^*$ und entscheidet die Menge der Palindrome.

Simulation von TM-Berechnungen durch Wortersetzungssystemen (Σ, Π)

Π ist Menge von Produktionen $l ::= r$, mit $l \in \Delta^+$, $r \in \Delta^*$

Kalkül: $\frac{u\ l\ v}{u\ r\ v}$ für $l ::= r \in \Pi$, $u, v \in \Delta^*$.

Sei

$T = (Q, \Sigma, \Gamma, \delta, q_0, F)$ und $\Delta = Q \dot\cup \Gamma \dot\cup \{\#\}$

Produktionen Π_T:

Für jedes $\delta(q,a) = (q', a', S)$: $\quad q\ a ::= q'\ a' \in \Pi_T$.

Für jedes $\delta(q,a) = (q', a', R)$ und $b \in \Gamma$:

$$q\ a\ b ::= a'\ q'\ b \in \Pi_T$$
$$q\ a\ \# ::= a'\ q'\ \Box\ \# \in \Pi_T$$

Für jedes $\delta(q,a) = (q', a', L)$ und $b \in \Gamma$

$$b\ q\ a ::= q'\ b\ a' \in \Pi_T$$
$$\#q\ a ::= \#q'\ \Box\ a' \in \Pi_T$$

Offenbar gilt:

$k = u\ q\ v \underset{T}{\vdash} u'\ q'\ v' = k'$, d. h. k' ist Folgekonfiguration von k

$\text{gdw} \quad \#u\ q\ v\# \overset{1}{\underset{\Pi_T}{\vdash}} \#\ u'\ q'\ v'\ \#$,

d. h. Rechnungen der TM T können in Π_T simuliert werden.

$\#q_0\ \Box\ w\ \# \underset{\Pi_T}{\vdash} \#\ u\ q\ v\ \# \quad \text{gdw} \quad q_0\ \Box\ w \overset{*}{\underset{T}{\vdash}} u\ q\ v$

für $w \in \Sigma^*$, $u, v \in \Gamma^*$, $q \in Q$.

Das Ableitbarkeitsproblem

6.90 Definition Sei (Σ, Π) ein Wortersetzungssystem.

Das **Ableitbarkeitsproblem** $Abl \subset \Sigma^* \times \Sigma^*$ für (Σ, Π) ist gegeben durch

$$Abl\ x\ y \qquad \text{gdw} \qquad x \underset{\Pi}{\vdash} y$$

(für $x, y \in \Sigma^*$) d.h. „y lässt sich aus x mit Hilfe der Produktionen aus Π ableiten".

6.91 Satz Unentscheidbarkeit des Ableitbarkeitsproblems

Das Ableitbarkeitsproblem für beliebige Wortersetzungssysteme ist nicht entscheidbar.

Beweis:

Reduziere das Halteproblem für TM auf das Ableitbarkeitsproblem. Die Konstruktion TM $T \longrightarrow$ simulierendes Wortersetzungssystem Π_T ist effektiv. Für $q \in F$ füge noch die Produktionen

$a\ q ::= q$, $q\ a ::= q$, für $a \in \Delta \backslash \{\#\}$ und $\#\ q\ \# ::= q$ hinzu.

Dann gilt: T hält mit Eingabe w

gdw $\exists u, v \in \Gamma^*, q \in F$ mit $q_0\ \Box\ w \overset{*}{\underset{T}{\vdash}} u\ q\ v$

gdw $\exists u, v \in \Gamma^*, q \in F$ mit $\#\ q_0\ \Box\ w\ \# \underset{\Pi_T}{\vdash} \#\ u\ q\ v\ \#$

gdw $\exists q \in F$ mit $\#\ q_0\ \Box\ w\ \# \underset{\Pi_T}{\vdash} q$.

Also ist das Halteproblem auf das Ableitbarkeitsproblem reduzierbar.

Speziellere Ergebnisse (z.B. spezielles Wort ableitbar) sind möglich.

Das Postsche Korrespondenzproblem (PCP)

6.92 Definition

Das Postsche Korrespondenzproblem (**PCP**) besteht aus allen Listen von Wortpaaren

$$\mathcal{L} = (x_1 \sim y_1, \ldots, x_k \sim y_k) \qquad k \geq 1$$

mit nichtleeren Wörtern $x_i, y_i \in \Sigma^* (1 \leq i \leq k)$ zu denen es eine Indexfolge $i_1, \ldots, i_n \in \{1, \ldots, k\}$ mit $n \geq 1$ gibt, so dass

$(*) \qquad x_{i_1} \cdots x_{i_n} = y_{i_1} \ldots y_{i_n} \qquad$ gilt.

Schreibe: PCP($\mathcal{L}$). Die Folge $(i_1, \ldots, i_n)$ ist Lösung, falls $(*)$ gilt.

Einschränkungen: z. B. $i_1 = 1$ spezielle PCP (**SPCP**).

Beachte Parameter: $\Sigma, k, x_i, y_i \in \Sigma^+, 1 \leq i \leq k$

Lösung: Liste natürlicher Zahlen aus $\{1, \ldots, k\}$.

Beachte: Zu gegebener Liste $i_1, \ldots, i_n$ ist es einfach zu überprüfen, ob sie eine Lösung ist.

Beispiel

6.93 Beispiel $\Sigma = \{0, 1\}$

- $\mathcal{L}_1 = (0 \sim 0\,0\,0,\ 0\,1\,0\,0 \sim 0\,1,\ 0\,0\,1 \sim 1) \quad k = 3$
  Lösungen können nur mit $i_1 = 1$ oder $i_1 = 2$ beginnen.
  1. Lösung: $i_1 = 1, i_2 = 3 : \underline{0}\ \ \underline{0\,0\,1} = \underline{0\,0\,0}\ \ \underline{1}$.
  2. Lösung: $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, $i_4 = 3$.
  $x_2x_1x_1x_3 \qquad \underline{0\,1\,0\,0}\,\underline{0}\,\underline{0}\,\underline{0\,0\,1}$
  $y_2y_1y_1y_3 \qquad \underline{0\,1}\,\underline{0\,0\,0}\,\underline{0\,0\,0}\,\underline{1}$
  d. h. PCP($\mathcal{L}_1$), SPCP($\mathcal{L}_1$)
- $\mathcal{L}_2 = (\underline{1} \sim \underline{1}\,1\,1, \underline{1\,0}\,1\,1\,1 \sim \underline{1\,0},\ 1\,0 \sim 0) \quad k = 3$
  Lösung: $2, 1, 1, 3$ (muss mit 1 oder 2 beginnen).
  $x_2x_1x_1x_3 \quad \underline{1\,0\,1\,1\,1}\,\underline{1}\,\underline{1}\,\underline{1\,0} \quad \underline{1}\,\underline{1}\cdots$ keine
  $y_2y_1y_1y_3 \quad \underline{1\,0}\,\underline{1\,1\,1}\,\underline{1\,1\,1}\,\underline{0} \quad \underline{1\,1\,1}\,\underline{1\,1\,1}\cdots$ Lösung
  d. h. PCP($\mathcal{L}_2$), $\neg$ SPCP($\mathcal{L}_2$)
- $\mathcal{L}_3 = (\underline{0\,1} \sim \underline{0\,1}\,0, 1\,\underline{0\,0} \sim \underline{0\,0}, 0\,1\,0 \sim 1\,0\,0)$
  Behauptung: $\neg$ PCP($\mathcal{L}_3$):: Lösung muss mit 1 beginnen und mit 2 enden. $t \in \{1, 2, 3\}^+$ $t = 1t'2$.
  $\underline{0\,1}\,\underline{0\,1\,0}\,\underline{0\,1\,0}\cdots$ $\qquad t'$ muss mit 3 beginnen
  $\underline{0\,1\,0}\,\underline{1\,0\,0}\,\underline{1\,0\,0}\cdots$

  Keine Fortsetzung $\qquad \cdots\underline{1\,0\,0}$
  möglich, da kein y mit 1 endet $\qquad \cdots\underline{0\,0}$

Beispiel (Forts.)

- $\mathcal{L}_4 = (0\,0\,1 \sim 0, 0\,1 \sim 0\,1\,1, 0\,1 \sim 1\,0\,1, 1\,0 \sim 0\,0\,1)$
  Es gilt PCP($\mathcal{L}_4$) aber $\neg$ SPCP($\mathcal{L}_4$).
  Lösung etwas länger: mit 2 beginnen, mit 3 enden, 1 muss verwendet werden.

$\underline{0\,1}\,\underline{1\,0}\,0\,1\,0\,1\,1\,0$ keine Forts.
$\underline{0\,1}\,\underline{1}\,\underline{0\,0\,1}\,0\,1\,1\,0\,1\,1\,0\,0\,1$
$\underline{0\,1}\,\underline{1}\,\underline{0\,0\,1}\,1\,0\,1\,0\,0\,1\,0\,0\,1\,0\,1\,1\,0\,0\,1\,1\,0\cdots$
$\underline{0\,1\,1}\,\underline{0\,0\,1}\,\underline{1\,0\,1}\,0\,0\,1\,0\,0\,1\,0\,1\,1\,0\,\underline{0}\,\underline{1}\,\underline{1}\,0\,0\,1\,1\,0\,1\,0\,0\,1$

Unentscheidbarkeit von PCP

6.94 Satz Das PCP ist unentscheidbar.

Beweisidee: Reduziere Ableitbarkeitsproblem für WES auf PCP.

- Sei $\gamma = (\Sigma, \Pi)$ ein Wortersetzungssystem ohne ε-Regeln.
  $Abl_\gamma(u, v)$ gdw $u \underset{\Pi}{\vdash} v$
- Konstruktion: Aus $\gamma, u, v \rightsquigarrow \mathcal{L}_{u,v}$ mit $u \underset{\Pi}{\vdash} v$ gdw PCP($\mathcal{L}_{u,v}$).
  Sei $\gamma = (\Sigma, \Pi)$, $\Pi = \{u_i ::= v_i \mid i = 1, \ldots, n\}$,
  $\Sigma = \{a_1, \ldots, a_r\}$. $\hat{\Sigma} = \{\hat{a}_1, \ldots, \hat{a}_r\}$ Kopie von Σ und $\Gamma = \Sigma \mathbin{\dot\cup} \hat{\Sigma} \mathbin{\dot\cup} \{+, \hat{+}, [,]\}$.
- $u, v \in \Sigma^*$. Definiere $\mathcal{L}_{u,v}$ über Γ durch

X_i ::	$[u+$,	$]$,	$+$,	$\hat{+}$,	a_1,	$\ldots$,	a_r,	$\hat{a}_1$,	$\ldots$,	$\hat{a}_r$,	v_1,	$\ldots$,	v_n,	$\hat{v}_1$,	$\ldots$,	$\hat{v}_n$
	$\wr$	$\wr$	$\wr$	$\wr$	$\wr$		$\wr$	$\wr$		$\wr$	$\wr$		$\wr$	$\wr$		$\wr$
Y_i ::	$[$	$,\hat{+}v]$,	$\hat{+}$,	$+$,	$\hat{a}_1$,	$\ldots$,	$\hat{a}_r$,	a_1,	$\ldots$,	a_r,	$\hat{u}_1$,	$\ldots$,	$\hat{u}_n$,	u_1,	$\ldots$,	u_n
NR :	1	2	3	4	5	$\ldots$	l	m								n

$l = 4 + r$, $m = 4 + r + 1$, $n = 4 + 2r + 2n$

Behauptung: PCP($\mathcal{L}_{u,v}$) gdw $u \underset{\Pi}{\vdash} v$ (o.b.d.A. $u \neq v$).

„$\Leftarrow$" $u \underset{\Pi}{\vdash} v$ etwa $u = w_0 \overset{(1)}{\underset{\Pi}{\vdash}} w_1 \overset{(1)}{\underset{\Pi}{\vdash}} \cdots \overset{(1)}{\underset{\Pi}{\vdash}} w_k = v \quad k > 0$

Es gibt $t_j \in J^*$, $j = 1, \ldots, k$ mit

j-gerade $\quad X(t_j) = [w_0 + \hat{w}_1 \hat{+} w_2 + \cdots + \hat{w}_{j-1} \hat{+} w_j$
$\qquad\qquad Y(t_j) = [w_0 + \hat{w}_1 \hat{+} w_2 + \cdots + \hat{w}_{j-1}$

j-ungerade $\quad X(t_j) = [w_0 + \hat{w}_1 \hat{+} w_2 + \cdots \hat{+} w_{j-1} + \hat{w}_j$
$\qquad\qquad Y(t_j) = [w_0 + \hat{w}_1 \hat{+} w_2 + \cdots \hat{+} w_{j-1}$

7 Die Chomsky-Hierarchie
Formale Sprachen, Grammatiken, Automaten

Σ Alphabet, $L \subseteq \Sigma^*$ formale Sprachen.

- Terme über Signatur (S, Sigma)
- Formeln
- While Programme
- Partielle Korrektheitsformeln
- Ausdrücke (primitiv rekursiv, μ-rekursiv)

Wie beschreibt man Sprachen ?

- Durch **Grammatiken** $G = (N, T, \Pi, Z)$ (spezielle Kalküle)
  N, T disjunkte Alphabete, Π Produktionen über $N \cup T$
  $Z \in N$ Startsymbol. Von G **erzeugte** Sprache:
  $L(G) = \{w \in T^* : Z \underset{\Pi}{\vdash} w\}$, d. h.
  $Z \overset{1}{\underset{\Pi}{\vdash}} w_1 \overset{1}{\underset{\Pi}{\vdash}} \cdots \overset{1}{\underset{\Pi}{\vdash}} w_n = w \qquad n \geq 1$

Problem: Wie entscheidet man $w \in L(G)$?

- Durch **Automaten** $A = (Q, N, T, \Pi, i, F)$
  Q endliche Zustandsmenge, Π Produktionen über $N \cup T$, die Übergang zwischen Konfigurationen beschreiben, i Initialkontext, F Finalkonfigurationen. Von A **akzeptierte** Sprache:
  $L(A) = \{w \in T^* : \exists f \in F \;\; i(w) \underset{\Pi}{\vdash} f\}$

Problem: Wie entscheidet man $w \in L(A)$?

7.1 Grammatiken

7.1 Definition Allgemeine Grammatiken

Eine Grammatik ist ein 4 Tupel

$$G = (N, T, \Pi, Z)$$

- Mit N endliche Menge **Nichtterminalsymbole**,
- T endliche Menge **Terminalsymbole**, $N \cap T = \varnothing$,
- Π endliche Menge von **Produktionen** $l \to r$ mit $l, r \in (N \cup T)^*$, wobei l mindestens ein Zeichen aus N enthält und $Z \in N$ Startsymbol ist.

Die von G **erzeugte Sprache** ist die Menge

$$L(G) = \{w \in T^* : Z \underset{\Pi}{\vdash} w\}$$

D.h. es gibt eine Ableitung $\{Z, w_1, \ldots, w_n = w\}$ für w mit $Z \underset{\Pi}{\overset{1}{\vdash}} w_1 \underset{\Pi}{\overset{1}{\vdash}} w_2 \underset{\Pi}{\overset{1}{\vdash}} \cdots \underset{\Pi}{\overset{1}{\vdash}} w$, d. h. $Z \underset{\Pi}{\overset{n}{\vdash}} w$ im Wortersetzungssystem $(N \cup T, \Pi)$, für ein $n \geq 1$.

Zwei G_1, G_2 Grammatiken sind **äquivalent**, falls $L(G_1) = L(G_2)$.

Beispiele

7.2 Beispiel Schreibweisen

a) $G = (N, T, \Pi, Z)$, $N = \{Z, Z_1\}$, $T = \{a, b\}$
   $\Pi :: Z \to aZ_1, Z_1 \to bZ_1 \mid a$ 3 Produktionen.
   Behauptung: $L(G) = \{ab^n a : n \in \mathbb{N}\}$
   Beweis: „$\supseteq$" Gebe Ableitung an.
   „$\subseteq$" $L(Z_1, G) = \{w \in T^* : Z_1 \underset{\Pi}{\vdash} w\} = \{b^n a : n \in \mathbb{N}\}$
   Induktion nach $i : Z_1 \underset{\Pi}{\overset{i}{\vdash}} w, w \in T^*$
   $i = 1 \leadsto w = a$
   $i \to i + 1 \quad Z_1 \overset{i}{\vdash} b^i Z_1 \vdash b^i a$

b) $G = (N, T, \Pi, Z)$, $N = \{Z\}$, $T = \{a, b\}$
   $\Pi :: Z \to aZb \mid \varepsilon$
   Behauptung: $L(G) = \{a^n b^n : n \in \mathbb{N}\}$
   Sei $\alpha \in V^* = (N \cup T)^*$, $\alpha \notin T^*$, $Z \underset{\Pi}{\overset{n}{\vdash}} \alpha$, so $\alpha = a^n Z b^n$.
   Induktion nach n.
   Dann „$\subseteq$" klar, „$\supseteq$" Angabe einer Ableitung.

c) $N = \{Z, T, S, A, B\}$, $T = \{a, b\}$
   $\Pi :: Z \to TS$, $T \to aTA \mid bTB \mid \varepsilon$, $S \to \varepsilon$
   $Aa \to aA, \quad Ab \to bA, \quad AS \to aS$
   $Ba \to aB, \quad Bb \to bB, \quad BS \to bS$

Beispiele (Fort.)

Beispiel einer Ableitung:
$Z \overset{1}{\vdash} TS \overset{1}{\vdash} aTAS \overset{1}{\vdash} abTBAS \overset{1}{\vdash} abBAS \overset{1}{\vdash} abBaS \overset{1}{\vdash} abaBS \overset{1}{\vdash} ababS \overset{1}{\vdash} abab$
Behauptung: $L(G) = \{ww : w \in T^*\}$
Für $w = w(a, b)$, sei $\hat{w} = w(A, B)$ das entsprechende Wort in den Großbuchstaben. Weiterhin sei ρ die Spiegelungsfunktion.
„$\supseteq$" $Z \underset{\Pi}{\vdash} wT\rho(\hat{w})S \underset{T \to \varepsilon}{\vdash} w\rho(\hat{w})S \underset{\Pi}{\vdash} wwS \vdash ww$
„$\subseteq$" Normierte Ableitungen: Erst T-Regeln bis $T \to \varepsilon$
$Z \vdash TS \underset{\Pi}{\vdash} wT\rho(\hat{w})S \vdash w\rho(\hat{w})S \underset{\Pi}{\vdash} ww$
Groß $\to$ klein, Vertauschregeln, mit $AS \to aS$, $BS \to bS$

d) $N = \{Z, A, B\}$, $T = \{a, b\}$
   $\Pi :: Z \to \varepsilon \mid aAbZ \mid bBaZ, \quad A \to \varepsilon \mid aAbA,$
   $B \to \varepsilon \mid bBaB$
   Behauptung: $L(G) = \{w \in T^* : |w|_a = |w|_b\}$
   $Z \underset{\Pi}{\vdash} \alpha \in (N \cup T)^*$, $|w|_a = |w|_b$ klar aus Regeln, also $L(G) \subseteq \{w \in T^* \mid |w|_a = |w|_b\}$
   „$\supseteq$" Ableitung angeben + Induktion $|w|_a = |w|_b$.

   Eine andere Möglichkeit: $\Pi' : Z \to \varepsilon \mid aZb \mid bZa \mid ZZ$, dann $L(G') = L(G)$. Also sind G und G' äquivalent.
   Frage: Einfachste Grammatik, die eine Sprache L erzeugt?

Beispiele (Forts.)

e) $N = \{Z, B, C\}$, $T = \{a, b, c\}$
   $\Pi :: Z \to aZBC \mid aBC, \quad CB \to BC,$
   $aB \to ab, bB \to bb,$
   $bC \to bc, cC \to cc$
   Behauptung: $L(G) = \{a^n b^n c^n : n \geq 1\}$
   „$\supseteq$" $Z \overset{n-1}{\vdash} a^{n-1}S(BC)^{n-1} \underset{S \to aBC}{\overset{1}{\vdash}} a^n(BC)^n \underset{\Pi}{\vdash} a^n B^n C^n \underset{\Pi}{\vdash} a^n b^n C^n \underset{\Pi}{\vdash} a^n b^n c^n$

   „$\subseteq$" Jede Ableitung lässt sich „normieren", erst alle Anwendungen von Z-Regeln (d. h. keine $CB \to BC$ Anwendung), dann die restlichen Regeln.
   $Z \underset{\Pi}{\vdash} a^n ZW(B, C) \overset{1}{\vdash} a^{n+1}BCW(B, C) \vdash a^{n+1}b^{n+1}c^{n+1}$
   mit $|W(B, C)|_B = |W(B, C)|_C = n$

   Aus $aW(B, C)$ mit $|W(B, C)|_B = |W(B, C)|_C$ lässt sich nur $ab^n c^n$ ableiten (als terminales Wort).

7.2 Chomsky Hierarchie

7.3 Definition Klassifikation nach Form der Produktionen

Sei $G = (N, T, \Pi, Z)$ Grammatik.

0) G ist vom **Typ 0**, falls keine Einschränkungen.
1) G ist vom **Typ 1 (kontext-sensitiv)**, falls $l \to r \in \Pi$, so $l = xAy$, $r = xzy$ mit $x, y \in (N \cup T)^*$, mit $A \in N$, $z \in (N \cup T)^+$ (d.h. $|l| \leq |r|$).
   Ausnahme: $Z \to \varepsilon$ (ε - Regel) erlaubt, falls Z in keiner rechten Seite einer Produktion vorkommt.
2) G ist vom **Typ 2 (kontext-frei)**, falls $l \to r \in \Pi$, so $l = A$, $r = z$ mit $A \in N, z \in (N \cup T)^*$.
3) G ist vom **Typ 3 (rechts-linear)**, falls $l \to r \in \Pi$, so $l = A, r = aB|a|\varepsilon$, $A, B \in N, a \in T$.

Eine Sprache $L \subseteq T^*$ heißt **vom Typ i**, falls es eine Grammatik G vom Typ i gibt mit $L = L(G)$.

Im **Beispiel** 7.2: a) Typ 3, b) Typ 2, c) Typ 0, d) Typ 2, e) Typ 0.

Beachte: G rechts-linear, so G kontext-frei, G kontext-frei ohne ε-Regeln, so G kontext-sensitiv.

Normierungen für Grammatiken

7.4 Bemerkung Normierte Grammatiken - Eigenschaften

- Es gibt stets eine äquivalente Grammatik vom gleichen Typ, für die das Startsymbol in keiner rechten Seite einer Produktion vorkommt.
  $\Pi_1 = \Pi \cup \{Z_1 \to Z\}$
  Für Typ 3 $\{Z_1 \to \alpha$: für $Z \to \alpha \in \Pi\}$
- Für eine kontext-freie Grammatik G und Wörter $x, y, z, u, v \in (N \cup T)^*$ gilt
  $x \underset{\Pi}{\vdash} y$ so $uxv \underset{\Pi}{\vdash} uyv$ (gilt sogar für beliebige G)
  $xy \overset{n}{\underset{\Pi}{\vdash}} z$, so gibt es $z_1, z_2 \in (N \cup T)^*$ mit $z = z_1 z_2$ und $x \overset{\leq n}{\underset{\Pi}{\vdash}} z_1$, $y \overset{\leq n}{\underset{\Pi}{\vdash}} z_2$ (Ind. nach n).
- Für jede kontext-freie Grammatik G gibt es eine ε-freie kontext-freie Grammatik G_1 mit $L(G_1) = L(G) - \{\varepsilon\}$.
  Ist $\varepsilon \in L(G)$, dann gibt es eine kontext-freie Grammatik G' mit $L(G') = L(G)$, wobei die einzige Regel in G', die ε als rechte Seite hat, $Z' \to \varepsilon$ ist. Hierbei ist Z' Startsymbol von G', und Z' kommt in keiner rechten Seite einer Regel vor.

Normierungen - Abschlusseigenschaften

Beweisidee:

Sei $U_1 = \{X : X \to \varepsilon \in \Pi\}$ und
$U_{i+1} = U_i \cup \{X : X \to \alpha \in \Pi, \alpha \in U_i^*\}$.

Offenbar $U_i \subseteq N$, $U_i \subseteq U_{i+1}$. D. h. es gibt k mit $U_k = U_{k+1}$ und somit $U_k = U_{k+v}$, für $v = 0, 1, 2, 3 \ldots$

Behauptung: $X \underset{\Pi}{\vdash} \varepsilon \text{ gdw } X \in U_k$. (Beweis: Übung).
Insbesondere: $\varepsilon \in L(G) \text{ gdw } Z \in U_k$.

Definiere: $G_1 = (N, T, \Pi_1, Z)$ mit
$X \to \alpha' \in \Pi_1$ gdw es gibt $X \to \alpha \in \Pi, \alpha' \neq \varepsilon$ entsteht durch Streichen von Buchstaben in U_k (kein Streichen erlaubt).

7.5 Lemma Abschlusseigenschaften von $\mathcal{L}_i$

$\mathcal{L}_i$ ist abgeschlossen bzgl. $\cup, \circ, *$ für $i = 0, 1, 2, 3$.

Beweis:

$L_1 \circ L_2 = \{uv : u \in L_1, v \in L_2\}$

$$L^* = \{u_1 \ldots u_n : n \in \mathbb{N}, u_i \in L\} = \bigcup_{n \geq 0} L^n \quad (L^0 = \{\varepsilon\})$$

Sei L_j erzeugt von $G_j = (N_j, T_j, \Pi_j, Z_j)$. G_j vom Typ i $(i = 0, 1, 2, 3)$, $j = 1, 2$.

Abschlusseigenschaften

O.B.d.A. auf linken Seiten von Produktionen kommen keine terminalen Buchstaben vor. (Für $a \in T$ Platzhalter $A_a \in N$, ersetze Vorkommen von a in linker Seite durch A_a. Hinzunahme von Produktionen $A_a \to a$). $N_1 \cap N_2 = \varnothing$.

a) $\cup$: $G =$
   $(N_1 \cup N_2 \cup \{Z\}, T_1 \cup T_2, \Pi_1 \cup \Pi_2 \cup \{Z \to Z_1 \mid Z_2\})$
   Für Typ (3): $Z \to \alpha$ für $Z_1 \to \alpha \in \Pi_1$ oder $Z_2 \to \alpha \in \Pi_2$.
   G ist vom Typ i und $L(G) = L(G_1) \cup L(G_2)$.
b) $\circ$: $G = (N_1 \cup N_2 \cup \{Z\}, T_1 \cup T_2, \Pi_1 \cup \Pi_2 \cup \{Z \to Z_1 Z_2\})$
   G ist vom Typ i für $i = 0, 1, 2$.
   Behauptung: $L(G) = L(G_1) \circ L(G_2)$.
   „$\supseteq$“ $Z \overset{1}{\underset{\Pi}{\vdash}} Z_1 Z_2 \underset{\Pi}{\vdash} u Z_2 \vdash uv$ für $u \in L(G_1), v \in L(G_2)$.
   „$\subseteq$“ $Z \overset{1}{\underset{\Pi}{\vdash}} Z_1 Z_2 \underset{\Pi}{\vdash} X$ und $X \in (T_1 \cup T_2)^*$.
   Dann $Z_1 \underset{\Pi_1}{\vdash} X_1$ und $Z_2 \underset{\Pi_2}{\vdash} X_2$, $X = X_1 X_2$.
   Da linke Seiten nur aus nichtterminalen Buchstaben und $N_1 \cap N_2 = \varnothing$, d. h. keine Vermischungen.
   Für Typ 3 - Grammatiken:
   Π_1' entstehe aus Π_1 durch Ersetzen von jeder Produktion $X \to a|\varepsilon$ durch $X \to aZ_2$ bzw. $X \to \alpha$ für $Z_2 \to \alpha \in \Pi_2$.
   $G = (N_1 \cup N_2, T_1 \cup T_2, \Pi' \cup \Pi_2, Z_1)$ erfüllt Forderung.

Abschlusseigenschaften (Fort.)

c) $*: L^* = \{w : \exists n \in \mathbb{N}, w \in L^n, w = v_1 \dots v_n, v_i \in L\}$
Sei $G =$
$(N_1 \cup \{Z\}, T_1, \Pi_1 \cup \{Z \to \varepsilon, Z \to Z_1, Z_1 \to Z_1 Z_1\})$.
Dann ist G vom Typ i für $i = 0, 1, 2$ und $L(G) = L(G_1)^*$.
Für Typ 3 Grammatiken: Übung.

7.6 Folgerung

- Jede endliche Sprache ist vom Typ 3:
  $w = a_1 \dots a_n \qquad a_i \in T \qquad n \geq 0$
  $Z \to a_1 A_1, A_1 \to a_2 A_2, \dots, A_{n-1} \to a_n A_n, A_n \to \varepsilon$
  $N = \{Z, A_1, \dots, A_n\}$
- $\mathcal{L}_{\text{endl}} \subsetneq \mathcal{L}_{T_3} \subsetneq \mathcal{L}_{T_2} \subsetneq \mathcal{L}_{T_1} \subsetneq \mathcal{L}_{T_0}$
- Wie ordnen sich die Sprachklassen in Hierarchie ein?

$$\mathcal{L}_{\text{endl}} \subsetneq \mathcal{L}_{\text{prim-rek}} \subsetneq \mathcal{L}_{\text{rek-entsch.}} \subsetneq \mathcal{L}_{\text{rek-aufzb.}}$$

Ist $L(G)$ entscheidbar für beliebiges G ?

7.7 Lemma

Sei $G = (N, T, \Pi, Z)$ Grammatik, dann ist $L(G)$ rekursiv aufzählbar.

Idee: Führe systematisch alle Ableitungen aus Z der Länge nach durch.

Ableitbare Wörter aus $(N \cup T)^*$ in $1, 2, 3 \dots$ Ableitungsschritte.

$L(G)$ ist rekursiv aufzählbar

- Verfahren hält mit Eingabe w gdw $s \overset{i}{\underset{\Pi}{\vdash}} w$ für ein i d. h. w kommt in Stufe i vor.
- Verfahren ist effektiv und hält bei Eingabe w gdw $w \in L(G)$.

Formal: Sei $\Sigma = N \dot\cup T \dot\cup \{\vdash\}, \Pi = \{l_1 \to r_1, \dots, l_n \to r_n\}$ und
$M = \{w \in \Sigma^*$: Es gibt $w_1, \dots, w_m \in (N \cup T)^*$ mit
$w = \ \vdash Z \vdash w_1 \vdash \cdots \vdash w_m \vdash$ und
$Z \overset{1}{\underset{\Pi}{\vdash}} w_1, w_i \overset{1}{\underset{\Pi}{\vdash}} w_{i+1}$ für $i \geq 1\}$

M ist die Menge der Ableitungen in G.

Für $\alpha, \beta \in V^*$ sei

$Q_i(\alpha, \beta)$ gdw $\alpha \underset{l_i \to r_i}{\vdash} \beta$ gdw
$\exists\, \alpha', \alpha'' \leq \alpha.\ \alpha = \alpha' l_i \alpha'' \wedge \beta = \alpha' r_i \alpha''$

$Q(\alpha, \beta)$ gdw $\alpha \underset{\Pi}{\vdash} \beta$ gdw $Q_1(\alpha, \beta) \vee \cdots \vee Q_n(\alpha, \beta)$.

Offenbar $Q_1, \dots, Q_n \in \mathcal{P}(\Sigma), Q \in \mathcal{P}(\Sigma)$.

M ist primitiv rekursiv (verwende Anfangswort, Teil- und Endwort).

•$x \in L(G)$ gdw $\exists w.\ w \in M \wedge$ Endwort$(\vdash x \vdash, w)$.

Umkehrung

7.8 Lemma

$L \subset \Sigma^*$ rekursiv aufzählbar, dann gibt es eine Grammatik $G = (N, \Sigma, \Pi, Z)$ mit $L = L(G)$.

Beweisidee: Simuliere mit der Grammatik die TM-Schritte einer TM die L akzeptiert rückwärts.

Sei o.B.d.A. T eine TM, die L akzeptiert mit nur einem Haltezustand q. D. h. $F = \{q\}$. $T = (Q, \Sigma, \Gamma, \delta, q_0, F)$

Die Konfigurationen von T werden in Klammern eingeschlossen: $[uq_iv]$.

Produktionen von G bewirken:

1-Gruppe: $Z \underset{G}{\vdash} [uqv] \quad u, v \in \Gamma^* \ \ (u, v$ lang genug).

2-Gruppe: $[k_{i+1}] \underset{G}{\vdash} [k_i]$, falls $k_i \underset{T}{\vdash} k_{i+1}$, dabei ist $|k_i| = |k_{i+1}|$.

Dann gilt: $[uqv] \underset{G}{\vdash} [\Box^s q_0 \Box x \Box^t]$, falls $q_0 \Box x \overset{*}{\underset{T}{\vdash}} uqv \quad (x \in \Sigma^*)$.

3-Gruppe: $[\Box^s q_0 \Box x \Box^t] \underset{G}{\vdash} x$ für alle $s, t \in \mathbb{N}, x \in \Sigma^*$.

Wählt man s, t genügend groß, so verlässt die TM bei ihrer Berechnung nie den Block $\Box^s x \Box^t$.

Produktionen (Forts.)

Produktionen 1-Gruppe:
$Z \to [Z_0], Z_0 \to Z_0 b \mid b Z_0 \mid q \ (b \in \Gamma)$.

Produktionen 2-Gruppe: z. B. aus Turing Programm

$q_i : a$	$\rightsquigarrow$	$q_{i+1} a$	$\to$	$q_i b$	$b \in \Gamma$
$q_i : R$	$\rightsquigarrow$	$b q_{i+1}$	$\to$	$q_i b$	$b \in \Gamma$
$q_i : L$	$\rightsquigarrow$	$q_{i+1} b$	$\to$	$b q_i$	$b \in \Gamma$
$q_i : q_k$	$\rightsquigarrow$	q_k	$\to$	q_i	
$q_i : a, q_k$	$\rightsquigarrow$	$q_k a$	$\to$	$q_i a$	und $q_{i+1} b \to q_i b \ (b \neq a)$

Produktionen 3-Gruppe:
$q_0 \to T_1, \Box T_1 \to T_1, [T_1 \Box \to T_2$
$T_2 b \to b T_2, b \in \Sigma, T_2 \to T_3,$
$T_3 \Box \to T_3, \ T_3] \to \varepsilon.$

G ist Typ-0 Grammatik!
Hierbei ist $N = \{Z, Z_0, T_1, T_2, T_3, [,]\} \cup Q \cup (\Gamma - \Sigma)$
Es gilt $Z \underset{G}{\vdash} x \in \Sigma^*$ gdw T akzeptiert x, d. h. $L(G) = L$.

7.9 Satz

$L \subseteq \Sigma^*$ ist rekursiv aufzählbar gdw es gibt eine Typ-0 Grammatik $G = (N, \Sigma, \Pi, Z)$ mit $L = L(G)$.

Insbesondere sind Typ-0-Sprachen abgeschlossen gegenüber $\cap$ aber nicht gegen $\neg$ (Komplement) und es gibt nicht entscheidbare Typ-0-Sprachen.

Wortprobleme

7.10 Definition Wortproblem, uniformes Wortproblem

Sei $G = (N, \Sigma, \Pi, Z)$. Das **Wortproblem** für G ist definiert:

$$WP(x) \text{ gdw } x \in L(G) \quad (x \in \Sigma^*)$$

Ist $\mathcal{G}$ eine Klasse von Grammatiken, so ist das **uniforme Wortproblem** für $\mathcal{G}$ definiert durch

$$UWP(G, x) \text{ gdw } x \in L(G) \quad (G \in \mathcal{G}, x \in T_G^*)$$

7.11 Folgerung

- UWP ist nicht entscheidbar für Typ-0 Grammatik.
- Es gibt Typ-0 Grammatik mit unentscheidbaren WP.
- Das uniforme WP für Typ 1 Grammatiken ist primitiv rekursiv.

$$\mathcal{L}_{\text{endl.}} \subsetneq \mathcal{L}_{\text{Typ}-3} \subseteq \mathcal{L}_{\text{Typ}-2} \subseteq \mathcal{L}_{\text{Typ}-1} \subseteq \mathcal{L}_{\text{prim-rek}} \subsetneq \mathcal{L}_{\text{Typ}-0} = \mathcal{L}_{\text{rek-aufzb.}}$$

endliche Automaten EA	Kellerautomaten PDA	linear beschränkte Automaten LBA	TM als akzeptierende Automaten

Formale Sprachen und akzeptierende Automaten

Einschränkungen der Turing-Maschinen: Möglichkeiten

$M ::$ Eingabeband

w $\#$

q Ausgabe $w \in L / w \notin L$

u

- nur lesen im EB
- lesen dann rechts
- Endmarkierungen
- Hilfsband als Keller
- akzeptieren/verwerfen durch Zustand

Konfigurationen: $uqw \underset{M}{\vdash} u'q'w'$ mit Hilfe von Produktionen.

7.12 Definition Automaten für Sprachen

Ein **Automat** (oder **Akzeptor**) $A = (Q, N, T, \Pi, i, F)$ mit endlicher **Zustandsmenge** Q, endlicher Menge N von **Hilfssymbolen** und endlichem **Eingabealphabet** T, so dass Q, N, T paarweise disjunkt sind, $i : T^* \to (N \cup T)^* \cdot Q \cdot (N \cup T)^*$: **Initialkonfiguration** zur Eingabe $w \in I^*$, einer endlichen Menge von **Finalkonfigurationen** F der Form $lqr \in (N \cup T)^* q (N \cup T)^*$ und einer endlichen Menge Π von **Produktionen** der Form $lqr \to l'q'r'$ $(l, l', r, r' \in (N \cup T)^* q, q' \in Q)$.

$L(A) = \{w \in T^* : \exists f \in F \quad i(w) \underset{\Pi}{\vdash} f\}$ die von A **akzeptierte Sprache**.

7.3 Endliche Automaten - reguläre Sprachen - Typ 3-Sprachen

Typ-3 Grammatik: $G = (N, T, \Pi, \Sigma)$, Π mit Produktionen der Form $A \to aB|a|\varepsilon$, $A, B \in N, a \in T$

7.13 Definition Endliche Automaten

a) Ein (deterministischer) **endlicher Automat (DEA)** ist ein 5-Tupel $A = (Q, \Sigma, \Pi, q_0, F)$ mit $q_0 \in Q$ **Startzustand**, $F \subset Q$ Menge der **Finalzustände** (akzeptierende Zustände). $\Pi = \{qa \to q' : q, q' \in Q, a \in \Sigma\}$: Für jedes Paar $(q, a) \in Q \times \Sigma$ gibt es genau eine Produktion $qa \to q'$.

b) Ein **indeterministischer endlicher Automat (NEA)** ist ebenfalls ein 5-Tupel A wie eben mit dem Unterschied, dass es für jedes Paar $(q, a) \in Q \times \Sigma$ eine endliche (eventuell leere) Menge von Produktionen der Form $qa \to q'$ sowie Produktionen der Form $q \to q'$ (Spontanübergänge, ε-**Übergänge**) gibt.

c) **Initialkonfiguration** bei Eingabe $w \in \Sigma^* : q_0 w$, d. h. $i(w) = q_0 w$ für $w \in \Sigma^*$. **Finalkonfigurationen**: F.

d) Die von A **akzeptierte Sprache** ist die Menge $L(A) = \{w \in \Sigma^* : q_0 w \underset{\Pi}{\vdash} f$ für ein $f \in F\}$. Schreibe auch $q_0 w \underset{A}{\vdash} f$.

Beispiele - Darstellungsarten Zustandsgraph oder Automatendiagramme

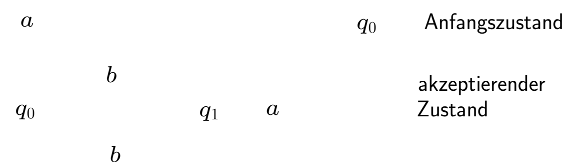
7.14 Beispiel

1. $A = (\{q_0, q_1\}, \{a, b\}, \Pi, q_0, \{q_0\})$
   $\Pi :: q_0 a \to q_0, q_0 b \to q_1, q_1 a \to q_1, q_1 b \to q_0$

   Behauptung: $q_0 w \underset{A}{\vdash} q_0 \text{ gdw } |w|_b$ gerade.

   Beweis: Induktion nach $|w|_b$, d. h. $L(A) = \{w \in \{a, b\}^* : |w|_b$ gerade$\}$.

   Diagramm: Knoten $\leftrightarrow$ Zustand, gerichtete Kante $\leftrightarrow$ Produktion



   Matrix-Tabelle:

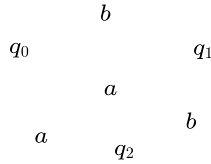
	a	b
q_0	q_0	q_1
q_1	q_1	q_0

Beispiele - Darstellungsarten Zustandsgraph oder Automatendiagramme (Forts.)

Bei indeterminierten Automaten: mehrere Kanten aus Zustand können mit Buchstaben a oder ε markiert sein.

Tabellendarstellung: Zustandsmengen + ε-Spalte.

2. Betrachte



Behauptung: $L(A) = \{ab, aba\}^*$

„$\supseteq$“ klar. „$\subseteq$“ Es gelte: $q_0 w \underset{A}{\vdash} q_0$.

Dann $w = \varepsilon$ oder w fängt mit a an.

$q_0 a w' \vdash q_1 w' \vdash q_0 \leadsto w'$ fängt mit b an.

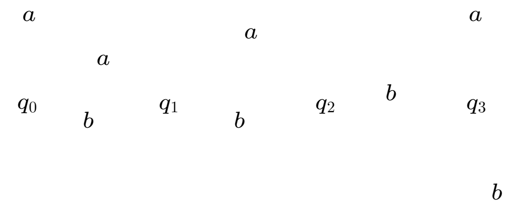
$q_0 w''$ Induktion

$q_1 b w''$

$q_2 w''$ w'' mit a +Ind.

Beispiele (Fort.)

3. $L = \{w \in \{a, b\}^* : w$ enthält nicht bbb als TW$\}$.

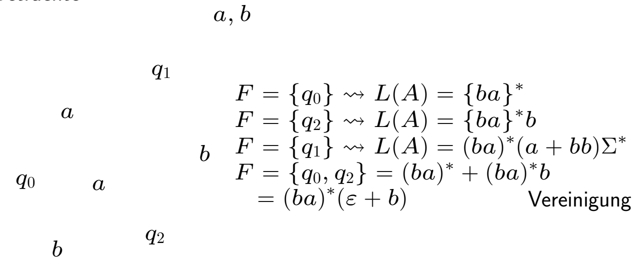


Beschreibung der Wege, die von q_0 nach q_i führen.

$q_0 \leadsto q_0 : \varepsilon, \{a\}^*, \{a\}^*\{ba\}^*\{a\}^*, a^*bbaa^*, \dots$

Reguläre Ausdrücke zur Beschreibung von Sprachen.

4. Betrachte



$F = \{q_0\} \leadsto L(A) = \{ba\}^*$

$F = \{q_2\} \leadsto L(A) = \{ba\}^* b$

$F = \{q_1\} \leadsto L(A) = (ba)^*(a + bb)\Sigma^*$

$F = \{q_0, q_2\} = (ba)^* + (ba)^* b$

$= (ba)^*(\varepsilon + b)$ Vereinigung

Operationen: Verkettung, Vereinigung, Iteration $(*)$.

Beispiele (Fort.)

5. Dezimalzahlen, die durch 5 teilbar sind.

	0	1	2	3	4	5	6	7	8	9
q_0	q_0	q_1	q_2	q_3	q_4	q_0	q_1	q_2	q_3	q_4
q_1	q_0	q_1	q_2	q_3	q_4	q_0	q_1	q_2	q_3	q_4
q_2	q_0	q_1	q_2	q_3	q_4	q_0	q_1	q_2	q_3	q_4
q_3	q_0	q_1	q_2	q_3	q_4	q_0	q_1	q_2	q_3	q_4
q_4	q_0	q_1	q_2	q_3	q_4	q_0	q_1	q_2	q_3	q_4

$q_0 w \vdash q_i$ gdw $w \equiv i (\mathrm{mod}\ 5)$, $F = \{q_0\}$

Automat mit 2 Zustände genügt!

$\leadsto$ **Äquivalente Automaten, minimale Automaten**.

Endliche Automaten und Typ-3-Grammatiken

7.15 Lemma Charakterisierungssatz

Ist $A = (Q, \Sigma, \Pi, q_0, F)$ EA, so ist $L(A)$ eine Typ-3 (rechtslineare) Sprache.

Beweis:

Definiere rl-Grammatik $G = (N, \Sigma, \Pi_G, Z)$ mit $N = Q$, $Z = q_0$, so dass für alle $x \in \Sigma^*$ gilt:

$$(*) \qquad q_0 x \underset{A}{\vdash} q_i \text{ gdw } Z \underset{G}{\vdash} x q_i$$

Endliche Automaten und Typ-3-Grammatiken

Definiere:

$$\begin{aligned} \Pi_G = \quad & \{q_i \to a q_j : q_i a \to q_j \in \Pi\} \\ & \cup \{q_i \to a : q_i a \to q \in \Pi \wedge q \in F\} \\ & \cup \{Z \to \varepsilon : \text{ falls } q_0 \in F\} \end{aligned}$$

G ist rechts-lineare Grammatik.

Behauptung: $(*)$ gilt für G:

Beweis: Induktion nach $|x|$.

„$\Rightarrow$“ $x = \varepsilon$, $q_0 \varepsilon \underset{A}{\vdash} q_0$, $Z = q_0 \underset{G}{\vdash} q_0$

$x \leadsto xa$, $q_0 x \underset{A}{\vdash} q_i$, Ind. Vor $Z \underset{G}{\vdash} x q_i$

Sei $q_i a \to q_j \in \Pi$, dann $q_0 x a \underset{A}{\vdash} q_i a \underset{A}{\vdash} q_j$

Da $q_i \to a q_j \in \Pi_G$ folgt $Z \underset{G}{\vdash} x q_i \overset{1}{\vdash} x a q_j$

„$\Leftarrow$“ $x = \varepsilon$, $Z \underset{G}{\vdash} q_i$, dann $q_i = q_0$

$x \leadsto xa$, $Z \underset{G}{\vdash} x a q_j$. Da Π_G rechts-linear ist, folgt

$Z \underset{G}{\vdash} x q_i \overset{1}{\vdash} x a q_j$ mit Regel $q_i \to a q_j \in \Pi_G$.

Dann aber $q_i a \to q_j \in \Pi$.

Nach Ind. Vor: $q_0 x \underset{A}{\vdash} q_i$ und somit $q_0 x a \underset{A}{\vdash} q_i a \underset{\Pi}{\overset{1}{\vdash}} q_j$.

Endliche Automaten und Typ-3-Grammatiken (2)

Behauptung: $L(A) = L(G)$

„$\subseteq$" $x \in L(A)$

-: $x = \varepsilon$, so ist $q_0 \in F$, $Z \to \varepsilon \in \Pi_G$, d. h. $x \in L(G)$

-: $x = ya$, $q_0 y \underset{A}{\vdash} q_i$, $q_i a \to q$ mit $q \in F$.

Dann folgt aus $(*)$ $Z \underset{G}{\vdash} y q_i \overset{1}{\vdash} ya$, da $q_i \to a \in \Pi_G$,
d. h. $ya \in L(G)$. Also $x \in L(G)$

„$\supseteq$" $x \in L(G)$

-: $x = \varepsilon$, so $Z \to \varepsilon \in \Pi_G \rightsquigarrow q_0 \in F \rightsquigarrow x \in L(A)$

-: $x = ya$, $Z \underset{G}{\vdash} y q_i \underset{G}{\overset{1}{\vdash}} ya$. Wegen $(*)$ ist $q_0 y \underset{A}{\vdash} q_i$ und $q_i a \to q$ mit $q \in F$, d. h. $q_0 ya \underset{A}{\vdash} q_i a \underset{A}{\vdash} q \in F$.
Also $x \in L(A)$.

Beachte:

G ist rechts-linear und „eindeutig", d. h. ist $w \in L(G)$, so gibt es genau eine Ableitung für w.
Falls A NEA, so Problem mit Spontanübergängen, diese würden Regeln der Form $q_i \to q_j$ bedeuten. Sonst ok.

Beispielkonstruktion

7.16 Beispiel Sei $A = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \Pi, q_0, \{q_0\})$.

Π	a	b
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

$G_A = (N, \Sigma, \Pi_G, Z)$, $N = \{q_0, \dots, q_3\}$, $Z = q_0$

$$\begin{aligned} \Pi_G :\quad & q_0 \to a q_2 | b q_1 | \varepsilon \quad (q_0 \in F) \\ & q_1 \to a q_3 | b q_0 | b \quad (q_0 \in F) \\ & q_2 \to a q_0 | a | b q_3 \quad (q_0 \in F) \\ & q_3 \to a q_1 | b q_2 \end{aligned}$$

Beachte: $|\Pi_G| \leq 2 \cdot |\Sigma| \cdot |Q| + 1$.

Frage: Wird jede Typ-3 Sprache von einem DEA akzeptiert?

Problem: Bei Typ-3 Grammatiken ist $A \to aB$ und $A \to aC$ erlaubt, d. h. Indeterminismus.

Endliche Automaten und Typ-3-Grammatiken (3)

7.17 Lemma Charakterisierungssatz

Zu jeder Typ-3 Sprache L gibt es NEA A mit $L = L(A)$.

Beweis: Sei G Typ-3 Grammatik $G = (N, T, \Pi_G, Z)$ mit $L = L(G)$.

Definiere:

$A = (Q, T, \Pi_A, q_0, F)$ mit $Q = N \dot\cup \{S\}$, $q_0 = Z$.

$\Pi_A : \{Xa \to Y :$ für $X \to aY \in \Pi_G\}$
$\quad \cup \{Xa \to S :$ für $X \to a \in \Pi_G\}$

$F = \{S\} \cup \{X \mid X \to \varepsilon \in \Pi_G\}$

Behauptung:

a) $q_0 w \underset{A}{\vdash} X$ gdw $Z \underset{G}{\vdash} wX$ für $X \in N, w \in T^*$.

b) $w \in L(A)$ gdw $w \in L(G)$ gdw $Z \underset{G}{\vdash} w$ für $w \in T^*$.

Beweis:

a) Induktion nach $|w|$ :: -: $w = \varepsilon$

„$\Rightarrow$" $X = q_0 = Z$,

„$\Leftarrow$" dito.

-: $w = va$

„$\Rightarrow$" $q_0 va \underset{A}{\vdash} X$, $x \in N$: Dann $q_0 v \underset{A}{\vdash} Y$, $Y \in N$ und $ya \vdash X$.

D. h. nach Ind. Vor. $Z \underset{G}{\vdash} vY \underset{G}{\overset{1}{\vdash}} vaX$.

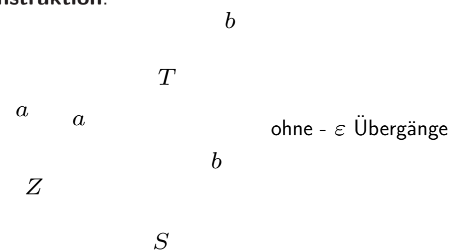
Konstruktion-Beispiele

„$\Leftarrow$" $Z \underset{G}{\vdash} vaX$, $X \in N$. Dann $Z \underset{G}{\vdash} vY$, für ein $Y \in N$ und $Y \to aX \in \Pi_G$. Dann $q_0 va \underset{A}{\vdash} Ya \underset{A}{\vdash} X$.

b) $w \in L(A)$.
Dann $q_0 w \underset{A}{\vdash} S$ oder $q_0 w \vdash X$ mit $X \to \varepsilon \in \Pi_G$. Dann aber $w = va$, $q_0 w \underset{A}{\vdash} Xa \vdash S$.
$X \in N \rightsquigarrow Z \underset{G}{\vdash} vX \underset{G}{\vdash} va \in L(G)$ oder $Z \underset{G}{\vdash} wX \underset{G}{\vdash} w \in L(G)$. $\rightsquigarrow$ Behauptung.

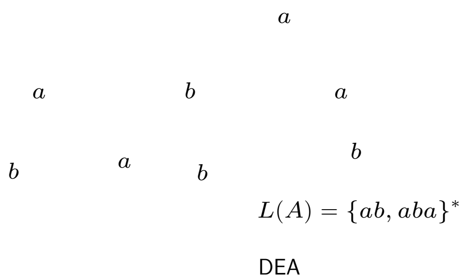
7.18 Beispiel

1. $G = (N, \Sigma, \Pi_G, Z)$, $N = \{Z, T\}$, $\Sigma = \{a, b\}$
   $\Pi_G :: Z \to aZ | aT$, $T \to bT | b$
   Behauptung: $L(G) = \{a^n b^m : n, m \geq 1\}$ (klar).
   Konstruktion:



Beispiele

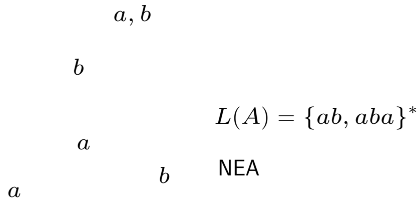
2. Betrachte



$L(A) = \{ab, aba\}^*$

DEA

3. Sei

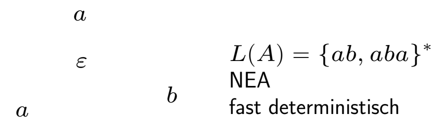


$L(A) = \{ab, aba\}^*$

NEA

Beispiele

4. Sei



$L(A) = \{ab, aba\}^*$
NEA
fast deterministisch

Kann man Spontanübergänge vermeiden?
JA: Idee $q \sim q'$ gdw es gibt $q_0, \ldots, q_n \in Q$ $q_0 = q, q_n = q'$, $q_i \to q_{i+1} \in \Pi$. Lässt sich effektiv berechnen!

$$\Pi^* = \{qa \to q' : \exists q''(q \sim q'' \wedge q''a \to q' \in \Pi)\}$$

$$F^* = \{q : \exists f \in F : q \sim f\}$$

Dann $L(A) = L(A^*)$.

Wir haben somit:

7.19 Lemma
$L \subseteq T^*$ ist Typ-3 Sprache gdw $L = L(A)$ für ein NEA A.

Charakterisierungssatz für r.l. Sprachen

7.20 Satz
Zu jedem NEA A gibt es einen DEA A' mit $L(A) = L(A')$.

Beweis: Sei $A = (Q, \Sigma, \Pi, q_0, F)$ ein NEA. A enthalte keine ε-Übergänge. Definition DEA $A' = (Q', \Sigma, \Pi', q_0', F')$ mit

- Q' = Potenzmenge von $Q = \{T : T \subseteq Q\}$
- $\Pi' = \{Ta \to \{q' \in Q : \exists q \in T \;\; qa \to q' \in \Pi\} : T \in Q', a \in \Sigma\}$
- $q_0' = \{q_0\}$
- $F' = \{T \subseteq Q : T \cap F \neq \varnothing\}$

Behauptung: $L(A') = L(A)$.
Beweis: Es gilt $Ty \underset{A'}{\vdash} \{q' \in Q : \exists q \in T \;\; qy \underset{A}{\vdash} q'\} =: T'$ für $T \subseteq Q, y \in \Sigma^*$.
Ind. nach $|y| : y = \varepsilon$, so $T' = T$, da keine Spontanübergänge.
Sei $y = az$, $a \in \Sigma$, dann

$$\begin{aligned} Taz \;\; & \underset{A'}{\vdash} \{q' : \exists q \in T \;\; qa \to q' \in \Pi\}z \\ & \underset{\text{Ind.Vor.}}{\underset{A'}{\vdash}} \{q'' : \exists q' \exists q \in T \;\; qa \to q' \in \Pi, q'z \underset{A}{\vdash} q''\} \\ & = \{q'' : \exists q \in T \;\; qaz \underset{A}{\vdash} q''\} \end{aligned}$$

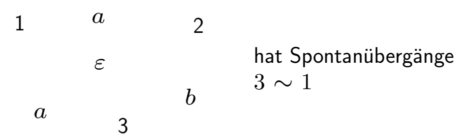
Beispiele

Sei

$$\begin{aligned} y \in L(A') \;\; & \text{gdw } \exists T \in Q \; (T \cap F \neq \varnothing \wedge \{q_0\}y \underset{A'}{\vdash} T) \\ & \text{gdw } \{q \in Q : q_0 y \underset{A}{\vdash} q\} \cap F \neq \varnothing \\ & \text{gdw } y \in L(A) \end{aligned}$$

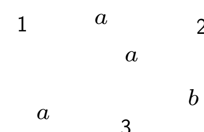
7.21 Beispiel

- Sei



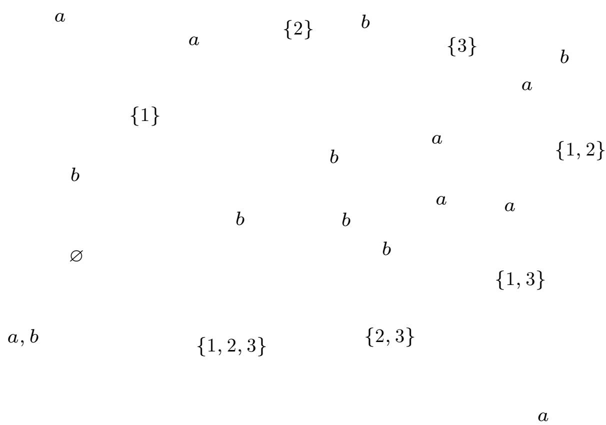
hat Spontanübergänge
$3 \sim 1$

ohne ε-Übergänge



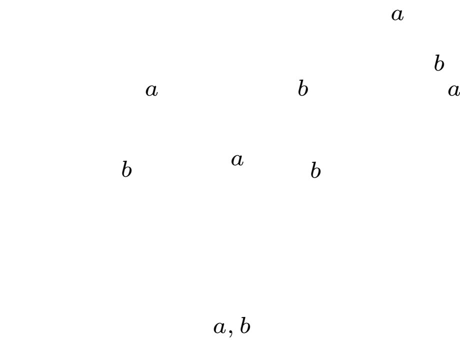
Neue Zustandsmenge:
$\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}$

Beispiele (Fort.)



Konstruktion liefert oft zu viele Zustände. Nicht erreichbare Zustände (vom Startzustand aus) streichen.

Beispiele (Fort.)



Ist dies minimaler DEA der $L(A)$ akzeptiert, d. h. minimale Anzahl von Zuständen? JA.

$x \underset{A}{\sim} y \text{ gdw } (q_0 x \underset{A}{\vdash} q \text{ gdw } q_0 y \underset{A}{\vdash} q)$.

$\underset{A}{\sim}$ ist rechtsinvariant, d.h.

$x \underset{A}{\sim} y \rightarrow xz \sim yz$ für alle $z \in \Sigma^*$.

Index = Anzahl der Äquivalenzklassen.

$L(A)$ ist Vereinigung von Äquivalenzklassen (Myhill-Nerode).

Es gibt Verfahren um einen äquivalenten minimalen DEA zu bestimmen.

Folgerungen

7.22 Folgerung

a) Rechts-lineare Sprachen sind abgeschlossen gegenüber Komplement und Durchschnitt.
$A = (Q, \Sigma, \Pi, q_0, F)$ DEA $L = L(A)$.
$A' = (Q, \Sigma, \Pi, q_0, Q - F)$ DEA mit $L(A') = \neg L$.
$L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}$ oder direkt mit Produktautomaten.
$A_1 \times A_2 = (Q_1 \times Q_2, \Sigma, \Pi_1 \times \Pi_2, (q_{0_1}, q_{0_2}), F_1 \times F_2)$.

b) Jede Typ-3 Sprache kann von Typ-3 Grammatik G erzeugt werden mit: Π enthält für $X \in N, a \in \Sigma$ $X \rightarrow aY$ oder $X \rightarrow a$ (genau eine Produktion $X \rightarrow aY$). D. h. G ist eindeutig und somit ist jede Typ-3 Sprache eindeutig.

c) Das WP für Typ-3 Grammatiken ist in linearer Zeit entscheidbar.

d) **Pumping-Lemma** für Typ-3 Sprachen.
Zu jeder Typ-3 Sprache L gibt es ein $n \in \mathbb{N}$, so dass für alle $y \in L$ gilt: Ist $|y| \geq n$. Dann lässt sich y zerlegen in $y = uvw$ mit $0 < |v| \leq |uv| \leq n$, so dass für alle $i \in \mathbb{N}$ $uv^i w \in L$.
Beweis:
Sei A DEA mit $L(A) = L$ und $n := |Q|$. Ist $y \in L(A)$, $|y| \geq n$. Betrachte
$q_0 y \overset{1}{\vdash} q_1 y_1 \vdash \cdots \overset{1}{\vdash} q_{n-1} y_{n-1} \overset{1}{\vdash} q_n y_n \vdash \cdots \overset{1}{\vdash} q \in F$, $\{q_0, \ldots, q_n\} \subseteq Q$. Es gibt Zustand q', der zweimal vorkommt $q_0 uvw \underset{A}{\vdash} q'vw \underset{A}{\vdash} q'w \vdash q_0, v \neq \varepsilon, |uv| \leq n$. Dann aber $q_0 uv^i w \vdash q$ für alle $i \geq 0$.

Beispiel

7.23 Beispiel

$L = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$ nicht Typ 3 Sprache.

Angenommen, L ist rechts-linear, sei n Konstante für L.
Betrachte $y = a^n b^n \in L$
Pumping-Lemma $\rightsquigarrow a^{k_0}(a^k)^i a^{k_1} b^n \in L$ für alle i $(k_0 + k + k_1 = n, k > 0)$ ↯

Oder: $L \cap \{a\}^*\{b\}^* = \{a^n b^n \mid n \geq 0\}$ wäre rechts-linear, falls L es ist. ↯

e) Für eine Typ-3 Sprache sind folgende Probleme entscheidbar.
Dabei soll L durch eine Typ-3 Grammatik, oder durch einen DEA, oder durch einen NEA gegeben sein.
- Ist L leer?
- Ist $L = \Sigma^*$?
- Ist L endlich?
- Ist $L = L_1$ für eine Typ-3 Sprache L_1?

Es gibt weitere Charakterisierungen von rl-Sprachen, z.B. durch rechtsinvariante Äquivalenzrelationen auf Σ^* von endlichen Index (d.h. nur endlich viele Äquivalenzklassen) oder etwa durch reguläre Ausdrücke.

Andere Charakterisierung von Typ-3 Sprachen

Reguläre Ausdrücke über Σ : $REG(\Sigma)$
Wörter über $\Sigma \cup \{\Lambda, \varepsilon, \cup, *, (,)\}$ (oft $+$ für $\cup$).
Kalkül:

$\overline{\Lambda}, \quad \overline{\varepsilon}, \quad \overline{a}$ für $a \in \Sigma$, $\quad \frac{\alpha, \beta}{(\alpha\beta)}, \quad \frac{\alpha, \beta}{(\alpha \cup \beta)}, \quad \frac{\alpha}{\alpha^*}$

Semantik: Reguläre Sprachen, die durch reg. Ausdrücke über Σ dargestellt werden: $\langle\ \rangle$: reg. Ausdruck $\longrightarrow$ Sprachen über Σ

- $\langle \Lambda \rangle = \varnothing$
- $\langle \varepsilon \rangle = \{\varepsilon\}$
- $\langle a \rangle = \{a\} \quad a \in \Sigma$
- $\langle (\alpha\beta) \rangle = \langle \alpha \rangle \circ \langle \beta \rangle$
- $\langle (\alpha \cup \beta) \rangle = \langle \alpha \rangle \cup \langle \beta \rangle$
- $\langle \alpha^* \rangle = \langle \alpha \rangle^*$

7.24 Satz

L ist Typ-3 Sprache gdw L ist reguläre Sprache, d. h.
es gibt $\alpha \in REG(\Sigma) : \langle \alpha \rangle = L$.

Beweis:

„$\Leftarrow$“ Typ-3 Sprachen enthalten $\varnothing$,$\{\varepsilon\}$, $\{a\}$ für $a \in \Sigma$ und sind abgeschlossen gegen $\cdot, \cup, *$.

„$\Rightarrow$“ Sei $A = (Q, \Sigma, \Pi, q_1, F)$, $Q = \{q_1, \ldots, q_n\}$ DEA mit $L(A) = L$. Für $i, j \in \{1, \ldots, n\}$ und $t \in \{0, \ldots, n\}$ definiere

$$L^t_{ij} = \{y \in \Sigma^* : q_i y \overset{1}{\vdash} q_{i_1} y_1 \overset{1}{\vdash} \cdots \overset{1}{\vdash} q_{i_k} y_k \overset{1}{\vdash} q_j$$
mit Zwischenzuständen
$q_{i_1}, \ldots, q_{i_k} \in \{q_1, \ldots, q_t\}\}$

Behauptung: Jedes L^t_{ij} ist durch regulären Ausdruck darstellbar. Insbesondere auch $L(A)$.

Beweis: Induktion nach t:

$L^0_{ij} = \{y \in \Sigma^* : q_i y \overset{1}{\vdash} q_j\}$ ist endlich.

$L^{t+1}_{ij} = L^t_{ij} \cup L^t_{it+1}(L^t_{t+1t+1})^* L^t_{t+1j}$

$L(A) = \bigcup_{q_j \in F} L^n_{1j}$

7.25 Beispiel

1

q_1 0 q_2 1 q_3

0 0, 1

i	j	$t=$ 0	1	2	3
1	1	ε	ε	$(00)^*$	
1	2	0	0	$0(00)^*$	
1	3	1	1	0^*1	
2	1	0	0	$0(00)^*$	
2	2	ε	$\varepsilon + 00$	$(00)^*$	
2	3	1	$1 + 01$	0^*1	
3	1	$\varnothing$	$\varnothing$	$(0+1)(00)^*0$	
3	2	$0+1$	$0+1$	$(0+1)(00)^*$	
3	3	ε	ε	$\varepsilon + (0+1)0^*1$	

Varianten + Verallgemeinerungen EA

Endliche Automaten mit Ausgaben Mealy und Moore Automaten

$\Sigma = \{0,1\} \times \{0,1\}$
mod 2 Addierer.

0 1 0 1 1
0 0 1 1 0 1 0 0 0 1

0 0/0 1 0/0
1 1/0
q_0 q_1 0 1/0
0 1/1
0 0/1
1 0/1 1 1/1

0 0 1 0
$s_0/0$ 1 1 $s_1/0$ 0 1
0 1, 1 0 0 0 1 1 1 1 0 1, 1 0
0 0
$s_0/1$ $s_1/1$
0 0
0 1, 1 0 1 1

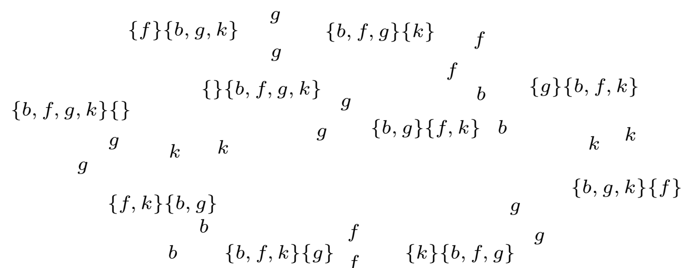
Spezifikation von Prozessen
Dynamisches Verhalten

Statecharts, Petri-Netze, SDL

UML Verhaltensdiagramme (Statecharts, Activity diagrams, MSC)

Event-Condition-Action: $e[c]a$: Übergänge.

Prozess: Bauer/Boot /Fluss, Gans/Fuchs/Korn.



7.4 Kontextfreie Sprachen - Typ2-Sprachen

Erinnerung Sei $G = (N, T, \Pi, Z)$ Grammatik.

G ist vom Typ 2 (kontextfrei), falls $l \to r \in \Pi$, so $l = A, r = z, A \in N, z \in (N \cup T)^*$.

Eine Sprache heißt kontextfrei, falls sie durch eine kontextfreie Grammatik erzeugt werden kann.

Beispiel: $G = (N, T, \Pi, Z)$, $T = \{a, b\}$, $N = \{Z\}$.
$\Pi : Z \to aZb \mid \varepsilon \qquad L(G) = \{a^n b^n \mid n \in \mathbb{N}\}$

Behauptung: $L(G)$ ist nicht rechtslinear. Sei n Konstante für L
$y = a^n b^n$. Pumping-Lemma $\rightsquigarrow (a^{k_0})(a^k)^i(a^{k_1})b^n \in L$
für alle $i \in \mathbb{N}$ $(k_0 + k + k_1 = n, k > 0)$ ↯

Gibt es auch ein Pumping-Lemma für kontextfreie Sprachen?

Es ist $aaabbb \in L(G)$. Ableitung als Baum:

Z

$a \quad Z \quad b$

$a \quad Z \quad b$

$a \quad Z \quad b$

ε

Ableitungsbäume - Strukturbäume

7.26 Definition

Sei G eine kontextfreie Grammatik und $(Z, u_1, \ldots, u_n)$ eine Ableitung in G. Der Strukturbaum zu dieser Ableitung wird induktiv über n definiert:

1. Der Strukturbaum zur Ableitung (Z) besteht aus einem einzigen mit Z beschrifteten Knoten. Blattwort ist Z.
2. Es sei die Ableitung $(Z, u_1, \ldots, u_n, u_{n+1})$ mit $u_n = uAv$, $u_{n+1} = ub_1 \ldots b_m v$ und eine Produktion $A \to b_1 \ldots b_m$ von G mit einzelnen Zeichen b_i gegeben. Sei weiter der Strukturbaum von $(Z, u_1, \ldots, u_n)$ schon konstruiert. Erweitere in diesem Baum den $(|u| + 1)$-ten Knoten (mit dem zu ersetzenden A beschriftet) mit m Folgeknoten, die mit $b_1, \ldots, b_m$ beschriftet sind. (ε als Zeichen erlaubt). Blattwort ist u_{n+1}.

7.27 Beispiel

$G = (N, T, \Pi, Z)$ mit $N = \{Z\}, T = \{a, b, c, +, *\}$,
$\Pi : Z \to Z + Z,\ Z \to Z * Z,\ Z \to a|b|c$

a) Z

$Z \quad + \quad Z$

$a \quad Z \quad * \quad Z$

$b \quad c$

b) Z

$Z \quad * \quad Z$

$Z \quad + \quad Z \quad c$

$a \quad b$

Strukturbäume

a) Z

$Z \quad + \quad Z$

$a \quad Z \quad * \quad Z$

$b \quad c$

Es gibt zu $a + b * c$ verschiedene Ableitungen:

(i) $(\underset{\uparrow}{Z}, \underset{\uparrow}{Z} + Z, a + \underset{\uparrow}{Z}, a + \underset{\uparrow}{Z} * Z, a + b * \underset{\uparrow}{Z}, a + b * c)$

(ii) $(\underset{\uparrow}{Z}, Z + \underset{\uparrow}{Z}, Z + Z * \underset{\uparrow}{Z}, Z + \underset{\uparrow}{Z} * c, \underset{\uparrow}{Z} + b * c, a + b * c)$

Die Ableitungen (i) und (ii) sind unterschiedlich, erzeugen aber denselben Strukturbaum: a).

Desweiteren wird in Ableitung (i) immer das am weitesten links stehende Nichtterminalzeichen ersetzt. (siehe ↑).

Betrachte die Ableitungen:

(iii) $(\underset{\uparrow}{Z}, \underset{\uparrow}{Z} * Z, \underset{\uparrow}{Z} + Z * Z, a + \underset{\uparrow}{Z} * Z, a + b * \underset{\uparrow}{Z}, a + b * c)$

(iv) $(\underset{\uparrow}{Z}, Z * \underset{\uparrow}{Z}, \underset{\uparrow}{Z} * c, Z + \underset{\uparrow}{Z} * c, \underset{\uparrow}{Z} + b * c, a + b * c)$

Strukturbäume

b) Z

$Z \quad * \quad Z$

$Z \quad + \quad Z \quad c$

$a \quad b$

Ableitungen (iii) und (iv) erzeugen Strukturbaum b).

Insgesamt:

1. Ein Strukturbaum repräsentiert eine Menge von Ableitungen.
2. Ein ableitbares Wort kann verschiedene Ableitungen haben, die nicht durch **einen** Strukturbaum dargestellt werden können.

Punkt 2 kann Schwierigkeiten bereiten, wenn einem ableitbaren Ausdruck eine Semantik (etwa ein Wert) zugeordnet werden soll.

Eindeutigkeit der Termsyntax geht verloren, wenn auf Klammern verzichtet wird. Was ist der Wert von $1 + 2 * 3$?
$(1 + 2) * 3 = 6$
$1 + (2 * 3) = 7$

Eindeutigkeit

7.28 Definition

Eine kontextfreie Grammatik G heißt **eindeutig**, falls für jedes $w \in L(G)$ gilt: Alle Ableitungen von w besitzen denselben Strukturbaum.
Eine k.f. Sprache L ist **eindeutig**, falls $L = L(G)$, mit G eindeutig.

7.29 Beispiel Betrachte Grammatik $G = (N, T, \Pi, Z)$ mit
$N = \{Z\}$, $T = \{a, b, c, +, *, (,)\}$,

$$\begin{aligned} \Pi : \quad & Z \to (Z + Z) \\ & Z \to (Z * Z) \\ & Z \to a|b|c \end{aligned}$$

G ist eindeutig und somit die Sprache $L(G)$ auch. $\rightsquigarrow$ Übung.

7.30 Definition

Sei G eine kontextfreie Grammatik und $(u_0, u_1, \ldots, u_n)$ eine Ableitung in G. Die Ableitung heißt **Linksableitung** in G, falls für alle $i < n \;\; u_{i+1}$ aus u_i durch Ersetzen des am weitesten links stehende Nichterminalzeichen mit Hilfe einer Regel in G entsteht.
(**Rechtsableitung** analog).

7.31 Beispiel G aus vorherigem Beispiel

$$(\underset{\uparrow}{Z}, (\underset{\uparrow}{Z} * Z), ((\underset{\uparrow}{Z} + Z) * Z), ((a + \underset{\uparrow}{Z}) * Z),$$
$$((a + b) * \underset{\uparrow}{Z}), ((a + b) * c))$$

Ableitung für $((a + b) * c)$ $\rightsquigarrow$ Linksableitung.

Eindeutigkeit k.f. Grammatiken

7.32 Lemma

Eine kontextfreie Grammatik ist genau dann eindeutig, wenn jedes durch die Grammatik erzeugte Wort genau eine Linksableitung (bzw. Rechtsableitung) besitzt.

Beweis: Übung.

Beachte:

1. Ist $w \in L(G)$, so gibt es eine Linksableitung zu w.
2. Jede rechtslineare Sprache ist eindeutig.
3. Es gibt sogenannte ererbt mehrdeutige kontextfreie Sprachen, etwa
   $$\begin{aligned} L = \quad & \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \\ & \{a^n b^m c^m d^n \mid n, m \geq 1\} \end{aligned}$$
   Man kann zeigen:
   Jede kontextfreie Grammatik G, die L erzeugt, ist mehrdeutig.

Problem: Wie kann man möglichst effizient testen, ob ein Wort aus einer kontextfreien Grammatik ableitbar ist?

$\rightsquigarrow$ Konstruiere Automaten, der den Strukturbaum einer Ableitung in einer bestimmten Weise aufbaut: Top-Down, Preorder.

LL-Automaten zu einer k.f. Grammatik

7.33 Definition

Sei $G = (N, T, \Pi, Z)$ eine kontextfreie Grammatik. Der **LL-Automat** zu G ist das folgende Tupel

$$A_{LL}(G) = (\{\#\}, N, T, \Pi_{LL}(G), Z\#, \{\#\})$$

Mit folgenden Produktionen in $\Pi_{LL}(G)$:
Für alle $t \in T$ und alle Produktionen
$A \to B_1 \ldots B_n \in \Pi$ mit einzelnen Zeichen B_i

$A\# \to B_n \ldots B_1\#$ (Produce) (Beachte die Reihenfolge der B's)
$t\#t \to \#$ (Compare)

Ableitbarkeit in A_{LL} bedeutet Ableitbarkeit in diesem Wortersetzungssystem. Die von A_{LL} akzeptierte Sprache ist die Menge

$$\{x \in T^* : Z\#x \underset{\Pi_{LL}(G)}{\vdash} \#\}$$

Initialkonfiguration bei Eingabe $x \in T^* : Z\#x$, d. h.
$i(X) = Z\#x$.
Finalkonfigurationen: $\{\#\}$

7.34 Lemma Sei G eine kontextfreie Grammatik.
Es ist $x \in L(G)$ gdw $x \in L(A_{LL}(G))$.

Beispielkonstruktion

7.35 Beispiel G aus vorherigem Beispiel,

$$\begin{aligned} \Pi_{LL}(G) : \quad & Z\# \to \;)Z + Z(\# \\ & Z\# \to \;)Z * Z(\# \\ & Z\# \to a\#|\, b\#|\, c\# \\ & a\#a \to \# \\ & b\#b \to \# \\ & \vdots \\ &)\#) \to \# \end{aligned}$$

Wir wissen $((a + b) * c) \in L(G)$.
Betrachte Ableitung (Berechnung)

$$\begin{aligned} (\quad & Z\#((a + b) * c, \\ &)Z * Z(\#((a + b) * c), \\ &)Z * Z\#(a + b) * c), \\ &)Z*)Z + Z(\#(a + b) * c), \\ &)Z*)Z + Z\#a + b) * c), \\ &)Z*)Z + a\#a + b) * c), \\ &)Z*)Z +\#+ b) * c), \\ &)Z*)Z\#b) * c), \\ & \vdots \\ & \#) \end{aligned}$$

Spezielle Eigenschaften kontextfreier Sprachen Pumping-Lemma

Erinnerung: Syntaxanalyse: G Typ-2 Grammatik.

- $w \in L(G)$, so gibt es eine Linksherleitung (Ableitung) für w aus z, d. h.
$$Z \overset{1}{\underset{G}{\vdash}} \alpha_1 \overset{1}{\underset{G}{\vdash}} \alpha_2 \overset{1}{\underset{G}{\vdash}} \cdots \vdash \alpha_n = w$$
- LL-Automat akzeptiert w (simuliert die Linksableitung).
- Zugehöriger Strukturbaum (geordneter markierter Baum, mit Blattwort w).

Z

w

- G ist eindeutig gdw für kein $w \in L(G)$ gibt es zwei verschiedene Strukturbäume.
  gdw keine zwei verschiedene Linksableitungen.
  Es gibt kontextfreie Sprachen, die nicht von eindeutiger kontextfreier Grammatik erzeugt werden können.
  z. B. $\{b^m c^m d^l : m, l \geq 1\} \cup \{b^l c^n d^n : l, n \geq 1\}$
  Alle Wörter der Form $b^i c^i d^i \quad i \geq 1$ sind mehrdeutig.

Beispiel: Pumping Eigenschaft

7.36 Beispiel $G = (\{Z, A\}, \{a, b\}, \Pi, Z)$ mit
$\Pi : Z \to aAZ \mid a \qquad A \to ZbA \mid ZZ \mid ba$

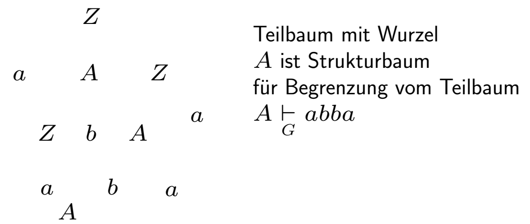
- $Z \vdash aAZ \vdash aZbAZ \vdash aabAZ \vdash aabbaZ \vdash aabbaa$
- Strukturbaum für $aabbaa$

Z

$a \quad A \quad Z$

$Z \quad b \quad A \quad a$

$a \quad b \quad a$



A

$Z \quad b \quad A$

$a \quad b \quad a$

Beachte
$A \underset{G}{\vdash} abA \underset{G}{\vdash} (ab)^n A \underset{G}{\vdash} (ab)^n ba$ oder
$Z \underset{G}{\vdash} aabbaZ \underset{G}{\vdash} (aabba)^n Z \vdash (aabba)^n a$

„Aufpumpen“ von Teilwörter bei Wiederholung nichtterminaler Buchstaben.

Pumping Lemma für k.f. Sprachen

7.37 Lemma $G = (N, T, \Pi, Z)$ kontext-freie Grammatik.
Sei $p = \max\{|\beta_i| : \alpha_i \to \beta_i \in \Pi\}$. Ist $\mathcal{B}$ Strukturbaum für $\alpha \in (N \cup T)^*$ der Tiefe h, so gilt $|\alpha| \leq p^h$.
(Da Anzahl der Blätter $\leq p^h$).

7.38 Satz $uvwxy$-**Theorem** (Bar-Hillel, Perles, Shamir).
Sei L eine kontext-freie Sprache. Dann gibt es ein $n \in \mathbb{N}$, so dass für jedes Wort $z \in L(G)$ mit $|z| \geq n$ gilt:
Es gibt eine Zerlegung von z in $uvwxy$ mit $0 < |vx|$ und $|vwx| \leq n$ und für jedes $i \in \mathbb{N}$ ist auch $uv^i wx^i y \in L(G)$.

- (Beachte: Insbesondere ist auch $uwy \in L(G)$).

Beweis-Idee: o.B.d.A. sei L erzeugt von kontext-freier Grammatik G ohne ε-Regeln (bis auf $Z \to \varepsilon$).
Sei $p = \max\{|\beta| : A \to \beta \in \Pi_G\}$. Betrachte $p^{|N|}$ und $z \in L(G)$ mit $|z| > p^{|N|}$. Ist $\mathcal{B}$ Strukturbaum für z, so ist die Tiefe von $\mathcal{B}$ mindestens $|N| + 1$. Sei $\mathcal{B}$ gewählt von minimaler Tiefe h.

Behauptung: Es gibt $A \in N$ mit
$Z \underset{G}{\vdash} uAy \underset{G}{\vdash} uvAxy \underset{G}{\vdash} uvwxy = z$, wobei
$u, v, w, x, y \in \Sigma^*$, $vx \neq \varepsilon$, $|vwx| \leq p^{|N|}$.
Dann $A \underset{G}{\vdash} vAx$, $A \underset{G}{\vdash} w$, wähle $n = p^{|N|} + 1$.

Beweisargument

Beachte: Analoges Argument führt zu Beweis des Pumping-Lemmas für RL-Grammatiken.

Z

Z kommt auf keiner rechten Seite vor.

$h \geq |N| + 1$ $\quad A$

keine ε-Regeln.

$h' \leq |N|$ $\quad A$

$u \quad v \quad w \quad x \quad y$

- Innere Knoten sind mit Nichtterminalsymbolen (NT) markiert.
- Da $h \geq |N| + 1$, gibt es eine Weg zu Blatt der Länge $\geq |N| + 1$
- NT-Symbol (verschieden von Z) wiederholt sich.
- Wähle NT A maximaler Tiefe, d.h. Teilbaum unter A hat Tiefe $\leq |N|$ und $|vwx| \leq p^{|N|}$.
- Dann $vx \neq \varepsilon$, da $\mathcal{B}$ minimaler Tiefe.

$\rightsquigarrow$ Behauptung.

Anwendungen

7.39 Folgerung und Anwendungen

a) Die Sprache $L = \{a^m b^m c^m \mid m > 0\}$ ist **nicht kontextfrei**. Angenommen L ist kontextfrei, n die Konstante vom $uvwxy$-Theorem. Wähle $m > n/3$.
$z = a^m b^m c^m = uvwxy$, $vx \neq \varepsilon$, $|vwx| \leq n$
Enthält v oder x mindestens zwei Buchstaben aus $\{a, b, c\}$, so $uv^2wx^2y \notin L$, da falsche Reihenfolge der Buchstaben.
Falls v und x nur aus a's, b's oder c's, so falsche Anzahl, da nur zwei gekoppelt.

b) $L = \{a^n : n \text{ Primzahl } \} \subseteq a^*$ ist **nicht kontextfrei**. Angenommen ja. Dann ist L RL-Sprache (warum?). Sei n Konstante des Pumping-Lemmas für RL-Sprachen $a^p \in L$ mit $p > n$. Dann ist $a^p = a^i a^j a^k$, $j > 0$, $a^{i+l\cdot j+k} \in L$, $l \geq 0$. D. h. $i + l \cdot j + k$ ist Primzahl für alle l, insbesondere für $l = i + k$ ↯

c) Kontextfreie-Sprachen (Typ-2 Sprachen) sind nicht abgeschlossen gegen $\cap$ und $\neg$.
Beweis:
$L_1 = \{a^n b^n c^m : n, m \geq 1\}$, $L_2 = \{a^m b^n c^n : n, m \geq 1\}$ sind kontextfrei, aber $L_1 \cap L_2 = \{a^n b^n c^n : n \geq 1\}$ ist nicht kontextfrei, wegen $L_1 \cap L_2 = \Sigma^* - ((\Sigma^* - L_1) \cup (\Sigma^* - L_2))$ folgt Behauptung.

Anwendungen (Forts.)

d) Sei $G = (N, T, \Pi, Z)$ kontextfreie Grammatik
$p = \max\{|\beta| : A \to \beta \in \Pi\}$, $n = p^{|N|}$. $L(G)$ ist unendlich gdw es gibt $z \in L(G) : n < |z| \leq n \cdot (p+1)$.
Beweis:
„$\Leftarrow$" Pumping-Lemma.
„$\Rightarrow$" $z \in L(G)$ minimale Länge mit $|z| > n$. Angenommen $|z| > n \cdot (p+1)$, dann $z = uvwxy \in L(G)$, $0 < |vx| \leq |vwx| \leq n$ und $uwy \in L(G)$ nach Pumping-Lemma. Dann ist $n < |uwy| < |z|$ ↯

Insbesondere ist es entscheidbar, ob $L(G)$ unendliche Sprache für G Typ-2 Grammatik.

e) Beachte: Pumping-Lemma liefern notwendige, jedoch nicht hinreichende Bedingungen für L Typ-2 (3) Sprache:
$\{a^p b^n : p\text{-Primzahl}, n \geq p\}$ ist nicht kontextfrei, dies kann aber nicht mit Pumping-Lemma bewiesen werden.

LL-Automat für G $(\{\#\}, N, T, \Pi_{LL}(G), Z\#, \{\#\})$ kann als Kellerautomat aufgefasst werden. Nur ein Zustand $\#$.

Kontextfreie Sprachen und Kellerautomaten

7.40 Definition

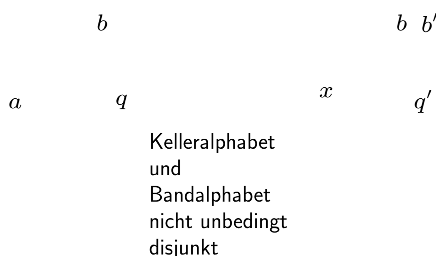
Ein Kellerautomat $K = (Q, N, T, \Pi, iq_0, F)$ mit Q Zustandsmenge, T Eingabealphabet, N Kelleralphabet, $i \in N, q_0 \in Q$, $F \subset Q$. Anfangskonfiguration: Für $x \in T^*$ $i(x) = iq_0x$,
Π Produktionen der Form

$aqb \to xq'$ (Lesen eines Zeichens)
$aq \to xq'$ (Spontanübergang)

mit $x \in N^*, a \in N, q, q' \in Q$ und $b \in T$.
Die von K akzeptierte Sprache ist die Menge

$$L(K) = \{x \in T^* : iq_0x \underset{\Pi}{\vdash} f \text{ für ein } f \in F\}$$

Lesen eines Zeichens und Spontanübergänge erzeugen in Abhängigkeit eines gewissen Buchstabens im Keller ein neues Wort.



Beispiele

Deterministische Kellerautomaten:
Für $(a, q) \in N \times Q$ gibt es entweder genau eine Produktion der Form $aq \to xq'$ oder für jedes $b \in T$ genau eine Produktion der Form $aqb \to xq'$. ⇝ **Deterministische kontextfreie Sprachen**.

7.41 Beispiel

1. $L = \{w \not{c}\, w^{mi} : w \in \{a, b\}^*\}$
k.f. Grammatik für L: $Z \to aZa \mid bZb \mid \not{c}$
$K = (\{q_0, q_1\}, \{Z, a, b\}, \{a, b, \not{c}\}, \Pi, Zq_0, F = \{q_1\})$
$\Pi ::$ $zq_0a \mapsto zaq_0$ $\quad zq_0b \to zbq_0$ $\quad z \in \{Z, a, b\}$
$zq_0 \not{c} \to zq_1$ $\quad z \in \{Z, a, b\}$
$aq_1a \to q_1$ $\quad bq_1b \to q_1$
$Zq_1 \to q_1$
K ist deterministischer Kellerautomat $L(K) = L$. Also ist L eine deterministische k.f. Sprache.

2. $G = (N, T, \Pi, Z)$, $I = \{a, b\}$, $\Pi : Z \to aZa \mid bZb \mid \varepsilon$
Dann gilt $L(G) = \{ww^{mi} : w \in T^*\}$.
Sei K mit $Q = \{q\}$, $N = \{Z, a, b\}$, $q_0 = q$, $i = Z$, und Π_K:
$aqa \to q$, $bqb \to q$
$Zq \to aZaq \mid bZbq \mid q$
(nicht deterministische Produktionen).

Beispiele (Fort.)

Behauptung: $L(K) = L(G)$

„$\supseteq$“ klar.

„$\subseteq$“ $Zqw \vdash q \rightsquigarrow Z$ muss vom Keller gelöscht werden., d. h.

$Zqw \vdash uZqv \overset{1}{\vdash} uqv \vdash q$

$uqv \vdash q$, wobei Z in u nicht enthalten ist.

$\rightsquigarrow$ nur Vergleiche, also $|u| = |v| \wedge u^{mi} = v$

(Ind. $|u|$).

v ist Endwort von w, d. h. $w = xv = xu^{mi}$ und

$Zq_0w \vdash uZu^{mi}qxu^{mi} \vdash uZqu^{mi}$, d. h. $2|u|$ Schritte

und $w = uu^{mi}$

Induktion nach $|u|$.

Charakterisierungssatz

7.42 Satz

Die kontextfreien Sprachen sind genau diejenigen, die durch einen Kellerautomaten akzeptiert werden.

Beweis: „$\curvearrowright$“ LL-Automat.

„$\curvearrowleft$“ Sei K ein Kellerautomat. o.B.d.A. Finalzustand nur ein Zustand $f \in Q$.

Definiere eine kontextfreie Grammatik G mit nichtterminalen
$N_G = \{[xq, q'] : x \in \Gamma, q, q' \in Q\}$,
Startzustand $Z = [iq_0, f]$,
Terminalsymbolen Σ und Produktionen

$[xq, q'] \rightarrow a[x_mq_m, q_{m-1}][x_{m-1}q_{m-1}, q_{m-2}] \cdots [x_2q_2, q_1][x_1q, q']$

Für jeden Befehl $xqa \rightarrow x_1 \cdots x_mq_m$, $a \in \Sigma \cup \{\varepsilon\}$ und alle $q_1, \ldots, q_{m-1}, q' \in Q$.

Es gilt für $x \in \Gamma, q, q' \in Q$ und $w \in \Sigma^*$

$(*) \qquad xqw \underset{K}{\vdash} q' \text{ gdw } [xq, q'] \underset{G}{\vdash} w$

Insbesondere erzeugt also G, die von K akzeptierte Sprache.

Beweis von $(*)$:

Es gelte $[xq, q'] \underset{G}{\vdash} w$. Durch Induktion über die Länge einer Ableitung in G zeige im Kellerautomaten gilt $xqw \underset{K}{\vdash} q'$.

Charakterisierungssatz (Forts.)

Erster Ableitungsschritt

$$[xq, q'] \overset{1}{\underset{G}{\vdash}} a[x_mq_m, q_{m-1}][x_{m-1}q_{m-1}, q_{m-2}] \cdots [x_2q_2, q_1][x_1q_1, q'] \underset{G}{\vdash} w$$

mit $a \in \Sigma \cup \{\varepsilon\}$. Somit ist w zerlegbar in Teilwörter $aw_m \ldots w_1$ mit der Eigenschaft $[x_iq_i, q_{i-1}] \underset{G}{\vdash} w_i$.

Für $1 < i \leq m$ und $[x_1q_1, q'] \underset{G}{\vdash} w_1$.

Nach Induktion vor folgt $x_iq_iw_i \underset{K}{\vdash} q_{i-1}$ für $1 < i \leq m$ und $x_1q_1w_1 \underset{K}{\vdash} q'$.

Da es die Regel $xqa \rightarrow x_1 \cdots x_mq_m$ im Kellerautomaten gibt, erhält man die Ableitung:

$$\begin{aligned} xqw = xqaw_m \ldots w_1 & \underset{K}{\vdash} x_1 \cdots x_{m-1}x_mq_mw_mw_{m-1} \cdots w_1 \\ & \underset{K}{\vdash} x_1 \cdots x_{m-1}q_{m-1}w_{m-1} \cdots w_1 \\ & \underset{K}{\vdash} x_1 \cdots q_{m-2} \cdots w_1 \\ & \cdots \\ & \underset{K}{\vdash} x_1q_1w_1 \\ & \underset{K}{\vdash} q' \end{aligned}$$

Charakterisierungssatz (Forts.)

Es gelte umgekehrt $xqw \underset{K}{\vdash} q'$. Induktiv über die Länge einer Ableitung im Kellerautomaten zeigt man nun $[xq, q'] \underset{G}{\vdash} w$.

Betrachte ersten Schritt:

$xqw \overset{1}{\underset{K}{\vdash}} x_1 \cdots x_mq_mv \underset{K}{\vdash} q'$

mit $w = av$, $a \in \Sigma \cup \{\varepsilon\}$. Zerlege die Ableitung von $x_1 \cdots x_mq_mv$ nach q' in m-Phasen, die dadurch definiert sind, dass nach der i-ten Phase im Keller nur noch die Zeichen $x_1 \cdots x_{m-i}$ verbleiben. (Da Keller leer gemacht werden muss). Die Ableitung hat somit die Form

$$\begin{aligned} xqw & \overset{1}{\underset{K}{\vdash}} x_1 \cdots x_mq_mv = x_1 \cdots x_mq_mw_m \cdots w_1 \\ & \underset{K}{\vdash} x_1 \cdots x_{m-1}q_{m-1}w_{m-1} \cdots w_1 \\ & \underset{K}{\vdash} x_1 \cdots q_{m-2} \cdots w_1 \\ & \cdots \\ & \underset{K}{\vdash} x_1q_1w_1 \\ & \underset{K}{\vdash} q' \end{aligned}$$

und es gilt $x_iq_iw_i \underset{K}{\vdash} q_{i-1}$ für $1 < i \leq m$, $x_1q_1w_1 \underset{K}{\vdash} q'$.
Ind.vor $\rightsquigarrow$ $[x_iq_i, q_{i-1}] \underset{G}{\vdash} w_i$, $K_i \leq m$ und $[x_1q_1, q'] \underset{G}{\vdash} w_1$,

also $[xq, q'] \overset{1}{\underset{G}{\vdash}} a[x_mq_m, q_{m-1}] \cdots [x_1q_1, q'] \underset{G}{\vdash} aw_m \cdots w_1 = av = w$.

Abschlusseigenschaften

7.43 Lemma

Der Durchschnitt einer kontextfreien Sprache mit einer rechts-linearen Sprache ist eine kontextfreie Sprache.

Beweis: Idee: Lasse gleichzeitig Kellerautomat und endlicher Automat ablaufen.

Sei K mit Produktionen der Form

$$xq_Ka \to x'q'_K \text{ bzw. } yq_K \to y'q'_K$$

und A DEA mit Produktionen

$$q_Aa \to q'_A(:= \delta(q_A, a))$$

Bilde Produktautomat $[K, A]$, d. h. $Q = [Q_K, Q_A] \ni [q_K, q_A]$ als Kellerautomat mit Produktionen
$x[q_K, q_A]a \to x'[q'_K, \delta(q_A, a)]$
bzw.
$y[q_K, q_A] \to y'[q'_K, q_A]$

Startzustand $[i_K, i_A]$, d. h. $i(x) = i[q_{0_K}, q_{0_A}]x$ für x Eingabewort.

Finalzustände $[f_K, f_A]$ $f_K \in$ Finalzustand von K, $f_A \in$ Finalzustand von A.

Abschlusseigenschaften (Forts.)

Es gilt

$$\begin{aligned} w \in L(K) \cap L(A) \text{ gdw } & \exists f_K \in F_K\ i_K q_{0_K} w \underset{K}{\vdash} f_K \wedge \\ & \exists f_A \in F_A\ q_{0_A} w \underset{A}{\vdash} f_A \\ \text{gdw } & \exists [f_K, f_A] \in F_K \times F_A : \\ & i_K[q_{0_K}, q_{0_A}]w \underset{[K,A]}{\vdash} [f_K, f_A] \\ \text{gdw } & w \in L([K, A]) \end{aligned}$$

7.44 Beispiel

$L = \{ww : w \in \{a, b\}^*\}$ ist keine kontextfreie Sprache.

Sei $R = a^+b^+a^+b^+$ rechts-lineare Sprache (warum?)

$L \cap R = \{a^ib^ja^ib^j : i \geq 1, j \geq 1\}$ ist keine kontextfreie Sprache.

Angenommen JA: Pumping-Lemma für kontextfreie Sprachen: Sei $n \in \mathbb{N}$ die Konstante für die kontextfreie Sprache $L \cap R$.

Wähle $i = j = n$ $\quad a^nb^na^nb^n \in L \cap R$.

$uvwxy$ Zerlegung: $\underset{|vwx| \leq n}{\longmapsto\!\!\dashv}$ $a^nb^na^nb^n = uvwxy$

⇝ Kopplung zwischen $a-$EXP und $b-$EXP kann nicht aufrechterhalten werden ↯

(Frage: Ist L kontext-sensitive Sprache?)

Bemerkung zu Pumping Lemmata

Beachte Quantoren bei Pumping Lemmata.

$\forall$	$\exists$	$\forall$	$\exists$	$\forall$	$uv^iw \in L$
$L \in \mathcal{L}_3$	$n \in \mathbb{N}$	$z \in L$	$u, v, w \in \Sigma^*$	i	
		$\lvert z\rvert \geq n$	$z = uvw$		
			$0 < \lvert v\rvert \leq n$		
$L \in \mathcal{L}_2$			$u, v, w,$		
			$x, y \in \Sigma^*$		$uv^iwx^iy \in L$
			$z = uvwxy$		
			$vx \neq \varepsilon$		
			$\lvert vwx\rvert \leq n$		

Es gibt schärfere Versionen dieser Lemmata. z. B. $|uv| \leq n$ oder $|vw| \leq n$ für rechts-lineare Sprachen. Odgen's Lemma für kontextfreie Sprachen (Man darf sogar gewisse Buchstaben markieren).

Wortproblem für kontextfreie Grammatiken

G kontextfreie Grammatik. $w \in \Sigma^*$ $w \in L(G)$? Wortproblem ist primitiv rekursiv entscheidbar. (schlechte obere Schranke!)

Kellerautomat der $L(G)$ akzeptiert Ist dieser effizient?

Problem:

- keine Endeutigkeit (mehrere Strukturbäume)
- Kellerautomat ist nicht-deterministisch.
- Falls deterministischer Kellerautomat möglich, so effizienter.

Beachte Beispiele:

- Boolesche Formeln über Signatur (PL-Formeln)
- Terme über Signatur
- Formeln über Signatur
- While Programme über Signatur

Mehrere Regeln mit gleicher linken Seite!

Verallgemeinerung der deterministischen Kellerautomaten

Mit Vorausschau $n \in \mathbb{N}$, falls in Abhängigkeit vom Kellerinhalt und den n-nächsten Eingabezeichen eindeutig die Möglichkeit besteht, die einzig richtige, als nächstes anzuwendende Produktion zu finden.

1-Vorausschau $\{a^n b^n : n \geq 1\}$

Schlagwort LR(k)-LL(k) Analyse.

7.45 Definition Normalformen für kontext-freie Grammatiken

Sei G eine kontext-freie Grammatik, G ist in

- **Chomsky-Normalform**: Produktionen der Form

$$A \rightarrow BC \text{ oder } A \rightarrow a \qquad A, B, C \in N, a \in T$$

- **Greibach-Normalform**: Produktionen der Form

$$A \rightarrow a\alpha \qquad A \in N, a \in T, \alpha \in N^*$$

7.46 Satz

Zu jeder kontextfreien Grammatik G mit $\varepsilon \notin L(G)$ gibt es eine kontextfreie Grammatik G' in Chomsky-Normalform, mit $L(G) = L(G')$.

Die Transformation $G \rightsquigarrow G'$ ist effektiv.

Beweisidee

Beweisidee: $G = (N, T, \Pi, Z)$ kontextfrei.

1. Schritt: **ε-frei**: Da $\varepsilon \notin L(G)$, gibt es eine äquivalente Grammatik G', die keine Produktionen der Form $A \rightarrow \varepsilon$ enthält.

2. Schritt: **Normierte Terminierung**: Zu $G = (N, T, \Pi, Z)$ gibt es eine äquivalente Grammatik $G' = (\tilde{N}, T, \Pi', Z)$, die nur Produktionen $A \rightarrow a$ mit $a \in T$ und $A \rightarrow \alpha$ mit $\alpha \in \tilde{N}^*$ enthält.
Sei $\tilde{N} = N \cup \{A_a : a \in T\}$. Π' entsteht aus Π indem jedes $a \in T$ in Π durch A_a ersetzt wird, vereinigt mit $\{A_a \rightarrow a : a \in T\}$ Platzhalter.

3. Schritt: **Keine Kettenproduktionen**: Zu einer Grammatik $G = (N, T, \Pi, Z)$ gibt es eine äquivalente Grammatik $G' = (N, T, \Pi', Z)$, die keine Produktionen der Form $A \rightarrow B$ mit $A, B \in N$ enthält (siehe NEA).
Sei $M = \{(A, B) \in N^2 : A \underset{G}{\vdash} B\}$
(lässt sich berechnen: Entferne Zyklen $(A, B), (B, A) \in M$. Beginne mit (A, A).)
$\Pi' = \Pi \backslash \{A \rightarrow B : A, B \in N\}$ vereinigt mit Produktionen $A \rightarrow r', |r'| > 1$, die aus Produktionen $A \rightarrow r \in \Pi$ durch Ersetzen mancher B in r durch C mit $(B, C) \in M$ entstehen, vereinigt mit $A \rightarrow a$ für alle $(A, A_a) \in M$.

Beweisidee (Forts.)

4. Schritt: **Chomsky-Normalform**: Produktionen der Form $A \rightarrow B_1 \ldots B_n$, $n > 2$ ersetzen: Dazu $A \rightarrow B_1 H_1, H_1 \rightarrow B_2 H_2, \cdots, H_{n-3} \rightarrow B_{n-2} H_{n-2}, H_{n-2} \rightarrow B_{n-1} B_n$ mit neuen Nichterminalsymbolen $H_1, \ldots, H_{n-2}$.

Falls $\varepsilon \in L(G)$, so ist $(N \cup \{Z'\}, T, \Pi \cup \{Z' \rightarrow Z, Z' \rightarrow \varepsilon\}, Z')$, wobei (N, T, Π, Z) in Chomsky-Normalform.

7.47 Beispiel Sei $G = (\{Z, A, B\}, \{a, b\}, \Pi, Z)$

$$\begin{array}{lll} \Pi: & Z \rightarrow bA & Z \rightarrow aB \\ & A \rightarrow a & B \rightarrow b \\ & A \rightarrow aZ & B \rightarrow bZ \\ & A \rightarrow bAA & B \rightarrow aBB \end{array}$$

1. Schritt: **ε-frei**: ok.

2. Schritt:

$$\begin{array}{ll} Z \rightarrow A_b A \mid A_a B & \\ A \rightarrow a, B \rightarrow b, A_a \rightarrow a, A_b \rightarrow b & \\ A \rightarrow A_a Z & B \rightarrow A_b Z \\ A \rightarrow A_b A A & B \rightarrow A_a B B \end{array}$$

3. Schritt: **Keine Kettenproduktionen**: ok.

Wortproblemalgorithmen für k.f. Sprachen

4. Schritt: Letzte Zeile oben:

$$A \rightarrow A_b C_1, C_1 \rightarrow AA$$
$$B \rightarrow A_a D_1, D_1 \rightarrow BB$$

Auswirkungen auf Strukturbaum? (binär)

$\rightsquigarrow$ Pumping Lemma Konstante: $2^{|N|} + 1$.

$x \in L(G) \rightsquigarrow x$ ist in höchstens $2|x| + 1$ Schritten in G ableitbar (exponentieller Aufwand für Entscheidung $x \in L(G)$).

7.5 Algorithmus von Cocke-Kasami-Younger

7.48 Satz

Sei G in Chomsky-Normalform. Dann gibt es einen Algorithmus der das Wortproblem für G mit Laufzeit $O(n^3)$ entscheidet.

$w \in L(G) \qquad |w| = n$ Laufzeit $O(n^3)$

Beweis: Sei $w = a_1 \ldots a_n$,
$L_{ij}(w) = \{A \in N : A \underset{G}{\vdash} a_i \ldots a_j\} \qquad (i \leq j)$

Es gilt $w \in L(G)$ gdw $Z \in L_{1n}(w)$.

Wie berechnet man aus w die L_{ij}. **Dynamisches Programmieren**.

Induktiv über $j - i$ Berechnung von L_{ij}:

- $j - i = 0 : L_{jj} = \{A : A \rightarrow a_j \in \Pi\}$
(da Chomsky-Normalform)

Algorithmus von Cocke-Kasami-Younger

- $j - i > 0$: Berechne L_{ij} aus L_{ik-1} und L_{kj} für ein k mit $i < k \leq j$, wobei $A \in L_{ij}$, falls $A \to BC \in \Pi$, $B \in L_{ik-1}$, $C \in L_{kj}$.

In jedem Schritt müssen maximal $2n$ Mengen betrachtet werden und es gibt weniger als n^2 Mengen L_{ij}, daher kann die Laufzeit durch cn^3 beschränkt werden, wobei c eine Konstante ist, die von der Grammatik G abhängt.

Verwaltung mithilfe einer Erkennungs-Matrix

$j-i$ \ i	1	2	...	n
0	{}	{}		{}
1	{}	...	{}	
2	⋮			
⋮	⋮			
$n-1$	{}			

Beispiel:

$Z \to CB \mid FA \mid FB$
$A \to CZ \mid FD \mid a$
$B \to FZ \mid CE \mid b$
$D \to AA, E \to BB, C \to a, F \to b$

$w = aababb$, $|w| = 6$

Teilw. Länge	$j-i$	$i=1$	2	3	4	5	6	
1	0	A,C	A,C	B,F	A,C	B,F	B,F	(n Einträge)
2	1	D	Z	Z	Z	E,Z		($n-1$ Eintr.) Kosten 1
3	2	A	A	B	A,B			($n-1$ Eintr.) Kosten 2
4	3	D	Z	Z,E				⋮
5	4	A	A,B					⋮
6	5	D,Z						1-Eintrag Kosten $n-1$

$$n + \sum_{i=2}^{n}(n-i+1)(i-1) = \frac{n^3+5}{6}$$

Auf Mehrband TM mit Zeit n^3 realisierbar. Siehe z. B. Hopcroft/Ullman Automaten + formale Sprachen.
Viele Verbesserungen: Mit Einschränkungen oft $O(n)$ möglich! (Vorausschau 1 Det.).

7.6 Unentscheidbare Probleme für kontextfreie Grammatiken

Unentscheidbare Probleme für allgemeine Grammatiken

- Wortproblem
- $L(G) = \varnothing$
- $L(G) = \Sigma^*$
- $L(G)$ endlich
- $L(G_1) = L(G_2)$
- $\varepsilon \in L(G)$?

Für rechts-lineare-Grammatiken alle entscheidbar.

Für kontexfreie Grammatiken? Wortproblem, $L(G)$ endlich ?, $L(G) = \varnothing?, \varepsilon \in L(G)?$ entscheidbar.

7.49 Satz

Sind G_1, G_2 kontextfreie Grammatiken.
Es ist unentscheidbar, ob die zugehörigen Sprachen disjunkt sind.

Folgendes Problem ist nicht rekursiv entscheidbar:

Eingabe: kontextfreie Grammatiken G_1, G_2.

Frage: $L(G_1) \cap L(G_2) \neq \varnothing$?

Unentscheidbare Probleme für kontextfreie Grammatiken (Forts.)

Beweis: Reduktion des PCP auf dieses Problem.

Sei $\mathcal{L} = (u_1 \sim v_1, \ldots, u_k \sim v_k)$, $u_i, v_i \in \Gamma^+$, $k \geq 1$.

Sei $J = \{1, \ldots, k\}$, $J \cap \Gamma = \varnothing$.

Definiere Grammatiken $G_j = (N_j, T, \Pi_j, Z_j)$, $j = 1, 2$.

$T = \Gamma \cup J$, $N_j = \{Z_j\}$

$\Pi_1 = \{Z_1 \to u_i Z_1 i \mid u_i i : i = 1, \ldots, k\}$ $2k$-Regeln

$\Pi_2 = \{Z_2 \to v_i Z_2 i \mid v_i i : i = 1, \ldots, k\}$ $2k$-Regeln

$$\begin{aligned} L(G_1) \cap L(G_2) \neq \varnothing \quad & \text{gdw } \exists x \in \Sigma^* \;\; x \in L(G_1) \cap L(G_2) \\ & \text{gdw } \exists t_1, t_2 \in J^* \\ & \qquad x = U(t_1)t_1^{mi} = V(t_2)t_2^{mi} \\ & \text{gdw } \exists t \in J^+ \;\; x = U(t)t^{mi} = V(t)t^{mi} \\ & \text{gdw } \exists t \in J^+ \;\; U(t) = V(t) \\ & \text{gdw } PCP(\mathcal{L}) \end{aligned}$$

Beachte:
Die Konstruktion liefert „einfache" k.f. Grammatiken G_1 und G_2:
Sie sind **linear** ($A \to uBv$ Regeln) und **eindeutig**: nur eine Linksableitung möglich!

Folgerungen

• Es gibt kein effektives Verfahren, um für zwei kontextfreie Grammatiken G_1, G_2 eine kontextfreie Grammatik G zu bestimmen mit $L(G) = L(G_1) \cap L(G_2)$. (Begründung: $L(G) \neq \varnothing$ ist für kontextfreie Grammatiken entscheidbar).

• Man kann jedoch eine kontextsensitive Grammatik berechnen mit $L(G) = L(G_1) \cap L(G_2)$, d. h. $L(G) \neq \varnothing$ ist nicht entscheidbar für kontextsensitive Grammatiken.

7.50 Satz Das Mehrdeutigkeitsproblem für kontextfreie Grammatiken ist unentscheidbar.

Eingabe: G kontextfreie Grammatik.

Frage: Ist G mehrdeutig?

Beweis:

PCP auf Mehrdeutigkeitsproblem reduzieren: Seien G_1 und G_2 die kontextfreien Grammatiken wie oben zu PCP $\mathcal{L}$ konstruiert.

$$G_{\mathcal{L}} := (\{Z, Z_1, Z_2\}, \Gamma \cup J, \Pi_1 \cup \Pi_2 \cup \{Z \to Z_1, Z \to Z_2\}, Z)$$

$\mathcal{L} \rightsquigarrow G_{\mathcal{L}}$ effektiv. $L(G_{\mathcal{L}}) = L(G_1) \cup L(G_2)$

G_1, G_2 sind eindeutig.

$G_{\mathcal{L}}$ ist mehrdeutig $\text{gdw } L(G_1) \cap L(G_2) \neq \varnothing$
$\text{gdw } PCP(\mathcal{L})$

Weitere unentscheidbare Probleme für kontextfreie Grammatiken

7.51 Satz

Folgende Probleme für kontextfreie Grammatiken sind nicht entscheidbar.

1)	$P_1(G)$	$\text{gdw } L(G) = \Sigma^*$ (G über $\Sigma = T$)
2)	$P_2(G)$	$\text{gdw } L(G)$ ist rechts-linear
3)	$P_3(G)$	$\text{gdw } \neg L(G)$ ist kontextfrei (rechts-linear, unendlich)
4)	$P_4(G)$	$\text{gdw } L(G_1) = L(G_2)$ (beide über T)
5)	$P_5(G)$	$\text{gdw } L(G_1) \subseteq L(G_2)$
6)	$P_6(G_1, G_2)$	$\text{gdw } L(G_1) \cap L(G_2)$ ist kontextfrei
7)	$P_7(G_1, G_2)$	$\text{gdw } L(G_1) \cap L(G_2)$ unendlich
8)	$P_8(G_1, G_2)$	$\text{gdw } L(G_1) \cap L(G_2)$ RL-Sprache

$\mathcal{L}_{\text{endl}} \subsetneq \mathcal{L}_3 \subsetneq \mathcal{L}_{\text{det-kf}} \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_{\text{prim-rek}} \subsetneq \mathcal{L}_{\text{entsch.}} \subsetneq \mathcal{L}_0 = \mathcal{L}_{\text{rek-aufzb.}}$

Kontextsensitive Grammatiken und Sprachen

Erinnerung: Die Sprache $\{a^n b^n c^n : n \in \mathbb{N}\}$ ist eine Typ-1 Sprache: Grammatik $(\{Z, A, B, H, C\}, \{a, b, c\}, \Pi, Z)$ mit Produktionen $\Pi = \{Z \to \varepsilon \mid Ac, A \to ab \mid aACB, CB \to CH, CH \to BH, BH \to BC, B \to b, Cc \to cc\}$.
Sie ist nicht kontexfrei.

Das Wortproblem für k.s. Grammatiken ist entscheidbar. Man muss nur Ableitungen bis zur Länge $(|N| + |T| + 1)^{|x|} + 1$ durchsuchen (ansonsten enthält die Ableitung zwei identische Wörter mit Länge $\leq |x|$).

Ein **linear beschränkter Automat (LBA)** ist eine (nichtdeterministische) Turing-Maschine, deren Lese-/Schreibkopf den Bereich, auf dem beim Start die Eingabe steht, nicht verlassen darf.
• Die Typ-1-Sprachen sind genau die Sprachen, die sich mit einem LBA akzeptieren lassen.

Solche Automaten lassen sich auch durch Produktionen Charakterisieren. Sie haben die Gestalt:
$qa \to q'a' \quad q, q' \in Q, a, a' \in N \cup T$
$qa \to aq' \quad q, q' \in Q, a, a' \in N \cup T$
$bqa \to q'ba \quad q, q' \in Q, a, a' \in N \cup T$
letztere für alle $b \in N \cup T$ falls keine andere diese linke Seite hat.

Kontextsensitive Grammatiken und Sprachen (Fort.)

Es gibt weitere Charakterisierungen der k.s. Sprachen durch spezielle Grammatiken. Eine Grammatik $G = (N, T, \Pi, Z)$ heißt **erweiternd**, falls Π nur Produktionen der Form $l \to r$ mit $l \neq \varepsilon$ und $|r| \geq |l|$ enthält. Es gilt: Zu jeder erweiternden Grammatik gibt eine äquivalente k.s. Grammatik.

8 Grundlagen der Programmierung
Zusammenfassung

Zusammenfassung
Ausblick

Grundlagen für die Entwicklung von Software-Systemen
Aktivitäten: **Spezifikation - Entwurf - Implementierung**

Benötigt: Formale Beschreibungstechniken

- Syntax
- Semantik

Spezifikation: Was soll ein SW-System leisten?

Funktionalität: Beschreibung funktionaler Eigenschaften.
Natürliche Sprache, Logik (Vor- Nachbedingungen).
Hier nur Aussagen- und Prädikatenlogik.

Abstrakte Datentypen $\equiv$ Algebren über Signatur.

Was: Axiome. Oft genügen „="-Axiome, d. h. Gleichheitslogik. Bedingte Gleichungen sind standard.

Beispiel: Keller(X) X Parameter $=_X$ definiert

empty: $\longrightarrow$ stack; is_empty: stack $\longrightarrow$ bool;
push: elem, stack $\longrightarrow$ stack;
pop: stack $\longrightarrow$ elem;

Formale Spezifikationstechniken (Forts.)

Axioms:

All-Quantif.: pop(push(x, y)) $= x$
is_empty(empty) = true
is_empty(push(x, y)) = false

$\rightsquigarrow$ **Formale Spezifikationstechniken**

Benötigt: Logik, Modelltheorie

Nicht funktionale Eigenschaften:

z. B. Zeitverhalten, Platzverhalten,. . . „Komplexität"

$\rightsquigarrow$ Klassifikation der berechenbaren Funktionen/Prädikate (Relationen).

Hier: Nicht jede Funktion ist berechenbar. Komplexitätsmaße.

Präzisierung der Berechenbarkeit

- While-berechenbare Funktionen
- μ-rekursive Funktionen $\mathcal{R}_p(\Sigma)$
- RM-berechenbare Funktionen Techniken
- Turing-berechenbare Funktionen **Simulation**

Formale Spezifikationstechniken (Forts.)

Komplexitätsmaß für ein Berechnungskonzept φ ist eine Abbildung $\Phi : \mathbb{N}^2 \longrightarrow \mathbb{N}$ mit den Eigenschaften

$\Phi(i, n) \downarrow$ gdw $\varphi_i(n) \downarrow$

$\{(i, n, m) \in \mathbb{N}^3 : \Phi(i, n) \leq m\}$ ist entscheidbar.

z. B. While Programme: Laufzeit eines While-Programms

$$\Phi(p, x) = \mu t.\text{first}(i^t(inp(p, x))) = 0$$

wobei i Interpreterfunktion.

- **Zeitkomplexität** der i-ten TM bei Eingabe n
  $\Phi(i, n) \downarrow$ gdw $\varphi_i(n) \downarrow$
  $\Phi(i, n) = \max_R\{t :$ Es gibt eine Berechnung R der Länge t der i-ten TM auf Eingabe $n\}$
- **Platzkomplexität** bei TM
  $\Phi(i, n)$ Anzahl der verschiedenen Bandstellen, die die i-te TM auf Eingabe n höchstens besucht.
  $\{\Phi(i, n) \leq m\}$ ist entscheidbar.
  Laufzeit muss $\leq m \cdot |Q||\Gamma|^m + 1$ (Anzahl der Konfigurationen der Länge $\leq m$).
- Bedarf an **Speicherplatz** in einem Goto-Programm.
  z. B. Anzahl der Register (**Einheitskostenmaß**)
  oder Anzahl der Register und Größe der Zahlen (**log-Kostenmaß**)

Wichtige Begriffe

Beachte:

Ist Φ ein Komplexitätsmaß und $B : \mathbb{N} \longrightarrow \mathbb{N}$ eine totale berechenbare Funktion. Dann gibt es eine totale, berechenbare Funktion $f : \mathbb{N} \longrightarrow \{0, 1\}$, so dass jedes Programm i, das f berechnet, für fast alle Werte n eine Komplexität $\Phi(n) \geq B(n)$ hat.

Es gibt beliebig komplexe Funktionen (**Diagonalisierung!**)

$\rightsquigarrow$ - **Komplexitätstheorie, Komplexitätsklassen**

DTime($s(n)$), NTime($s(n)$), DSpace($s(n)$), NSpace($s(n)$)

$P = \bigcup_{pol}$ DTime(pol), $NP = \bigcup_{pol}$ NTime(pol)

- **Reduzierbarkeit**: $P \leq_m Q$

Verfeinerungen: One-one Reduzierbarkeit: Injektion
pm: Pol-Zeit berechenbare Reduktionen.
Wichtig für die Klassen P, NP.

- **Vollständigkeitsbegriff**: z. B.

$$K = \{a \mid \varphi_a(a) \downarrow\}, K_0 = \{(a, x) : \varphi_a(x) \downarrow\}$$

Sind vollständig in der Klasse der rekursiv aufzählbaren Relationen.

- **Rekursionstheorie**: Universelle Funktionen $\varphi_p^{(n)}(x_1, \ldots, x_n)$
  SMN-Theorem: Stelligkeiten der universellen Funktionen.

Wichtige Begriffe (Forts.)

Ergebnisse sind richtig für jede **zulässige Aufzählung** der berechenbaren Funktionen. D.h. beschränkte Berechenbarkeit muss entscheidbar sein und SMN-Satz muss gelten.
$\leadsto$ Charakterisierung der r.a. Relationen, Rekursionssatz, Fixpunktsatz sowie die Sätze von RICE über **Indexmengen**

$$S \subset \mathcal{R}_p(\mathbb{N}) \qquad Ind(S) = \{p : \varphi_p \in S\}$$

- **Existenz berechenbarer Funktionen mit bestimmten Eigenschaften**.
  Verwende Churchsche These, SMN-, Rekursionssatz oder FPS.

- **Nicht Entscheidbarkeit** oder **nicht rekursiv aufzählbar**.
  Verwende Reduktion von K oder K_0 auf Relationen oder Rice Sätze, falls Indexmengen.
- Weitere Klassen: superrekursiv
  subrekursiv

Formale Beschreibungstechniken

Welcher Aspekt eines Systems soll beschrieben werden?

Wie soll beschrieben werden?

Nur das **was** oder **was** und **wie**.

- **Funktionale Aspekte**
- Verhaltensaspekt (Temporale Logik, Statecharts, SDL,...)
- Entwurfsaspekt (logischer Entwurf, Systemstruktur, „Architektur")
- Implementierungsaspekt (Programmiersprache, abstrakte Maschine, Compiler,...)

Zielsetzung: Übergänge zwischen Beschreibungen sollten „natürlich"-sein, Werkzeug-unterstützt.

$\leadsto$ Verifikation, Validierung, Testen sollten ermöglicht werden.

- Ableitung partieller Korrektheitsaussagen
- Verifikationsbedingungen
- Testgeneratoren
- Dokumenterstellung

Programmiersprachen

Deklarativ, Funktional (μ-rekursiver Ausdruck), Prozedural (While-Programm), Maschinennahe TM-, RM-Programme.
$\leadsto$ **Syntax, Semantik**

Syntax Programmiersprachen (allg. Beschreibungstechniken)

Grammatiken

i. Allg. kontextfrei (Teile kontext sensitiv: Prozedurdeklarationen, $\leadsto$ attributierten Grammatiken).

Grundlage: **Kalküle**

- Syntaktische Definition von Objektmengen
  Regeln, Ableitungen
- Müssen nicht immer Zeichenreihen sein
- Können auch graphische Objekte sein
- Oft beides zusammen

Beispiel: Terme, Formeln, Programme, gültige Formeln, gültige partielle Korrektheitsaussagen, Diagramme, UML,...

Speziell für Zeichenreihen: **Wortersetzungssysteme**.

Grundlage für: Grammatiken (RL, KF, KS ...)
Automaten (EA, NDEA, KA, TM ...)
Algebraische Strukturen: Monoide, Gruppen, Algebren

Z.B.: Prozessalgebren: $(a, b;\ ba \rightarrow ab)$ kommutatives Monoid.

Termersetzungssysteme: Signatur (S, Σ)

Listen: nil $:\rightarrow L$, cons $: X, L \rightarrow L$ append $: L, L \rightarrow L$

append(nil, l) $\Rightarrow l$

append(cons(x, l), l') $\Rightarrow$ cons(x, append(l, l'))

Kalküle (Fort.)

Funktionale Programmiersprachen: Applikative Programme.

(Bedingte Gleichungen ...) $\leadsto$ **Programmieren mit Gleichungen**

Beweiskalküle:

Aussagenlogik: Axiome (Schemata)

- $A \rightarrow (B \rightarrow A)$
- $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
- $(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$

Regeln: MP Modus Ponens oder Abtrennungsregel (cut Regel)

$$\frac{\alpha, \alpha \rightarrow \beta}{\beta}$$

$Ax \vdash \alpha$, so α Tautologie.

Beweis: $\alpha_1, \ldots, \alpha_n \qquad \alpha_n = \alpha$
$\alpha_i \in Ax$ oder $\alpha_j, \alpha_k = \alpha_j \rightarrow \alpha_i,\ j, k < i$

Kompaktheitssatz: $\Sigma \vdash \alpha$, so gibt es eine endliche Teilmenge $\Sigma_0 \subset \Sigma : \Sigma_0 \vdash \alpha$.

Verallgemeinerungen: - Prädikatenlogik.
- Andere Logiken. $\leadsto$ Logik Vorlesung.

Logik, Prozessbeschreibungssprachen

Grundlage automatischer Beweiser

Beachte: $Nat = (\mathbb{N}, 0, 1, +, \cdot, <)$ Theorie von Nat (d. h. gültige Formeln in Nat) nicht rekursiv aufzählbar!

Grundlage für Prozess-BT: **Endliche Automaten**

- Endliche Automaten
- Petri-Netze (Verallgemeinerungen)
- Statecharts, SDL $\cdots$
- $\vdots$

Semantik von Beschreibungstechniken

- Operational (Zustandsübergänge)
- Denotional (Programme „bezeichnen“ Funktionen)
- Transformation (Übersetzen in Konstrukte mit wohldefinierter Semantik)

Interpreterfunktionen, Compiler, abstrakte Maschinen, Termersetzung,. . .

Wichtige Eigenschaften von BT:

Modularisierung, **Parametrisierung**, **Kompositionsmöglichkeiten**, **Verfeinerung** (Abstraktionsmöglichkeiten).