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

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.
   $\rightsquigarrow$ 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.
   $\rightsquigarrow$ 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?

prinzipielle Lösbarkeit
$\downarrow$
effiziente Lösbarkeit
$\downarrow$
algorithmischer Entwurf
$\downarrow$
P: Programm einer HPS

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).
  $|\{0, \ldots, n-1\}| := n \qquad A$ **endlich** $\mathrm{gdw}\ \exists n : |A| = n.$
- 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**:
  $f : A \to B : \mathbf{dom(f)} = \{\mathbf{a} \in \mathbf{A} : \mathbf{f(a)} \downarrow\}$.
- **Wertebereich**:
  $f : A \to B : \mathbf{im(f)} = \{\mathbf{f(a)} \in \mathbf{B} : \mathbf{a} \in \mathbf{A}, \mathbf{f(a)} \downarrow\}$.
- $f : A \to B$ 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 $(\chi_R(a) = 1 \text{ gdw } a \in R)$.
- **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: $a = a_1 \cdots a_n \to a^{\text{mi}} = a_n \cdots a_1$.
- **Sprachen** sind Teilmengen von Σ^*.
  $L, M \subseteq \Sigma^* : L \cup M$ Vereinigung.
  $LM = \{uv : u \in L, v \in M\}$ Konkatenation.
  $L^* = \{v_1 \cdots v_n : n \in \mathbb{N}, v_i \in L, i = 1, \ldots, n\}$
  $= \bigcup_{n \geq 0} L^n$ Iteration.
- **Beachte**: Σ^* ist abzählbar.
  Programme sind Wörter (Zeichenreihen) über Alphabet $\rightsquigarrow$ 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 $f : A \longrightarrow B$ „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 $f : A \longrightarrow B$ 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 $\Pi_1, \ldots, \Pi_n$ $(n \geq 0)$ (Prämissen) bereits vorhanden.

Schreibweise: $R :: \frac{\Pi_1, \ldots, \Pi_n}{\kappa}$.

(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. $\Sigma = \{a_1, \ldots, a_n\}$ Regelmenge: $\frac{}{\varepsilon}$, $\frac{u}{ua_1}$, $\cdots$, $\frac{u}{ua_n}$ $u \in \Sigma^*$ (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 $(\varphi_1, \varphi_2, \ldots, \varphi_n)$ von Objekten, so dass für alle $i = 1, \ldots, n$ φ_i die Konklusion einer Regel von K ist, deren Prämissen alle in $\{\varphi_1, \ldots, \varphi_{i-1}\}$ enthalten sind.

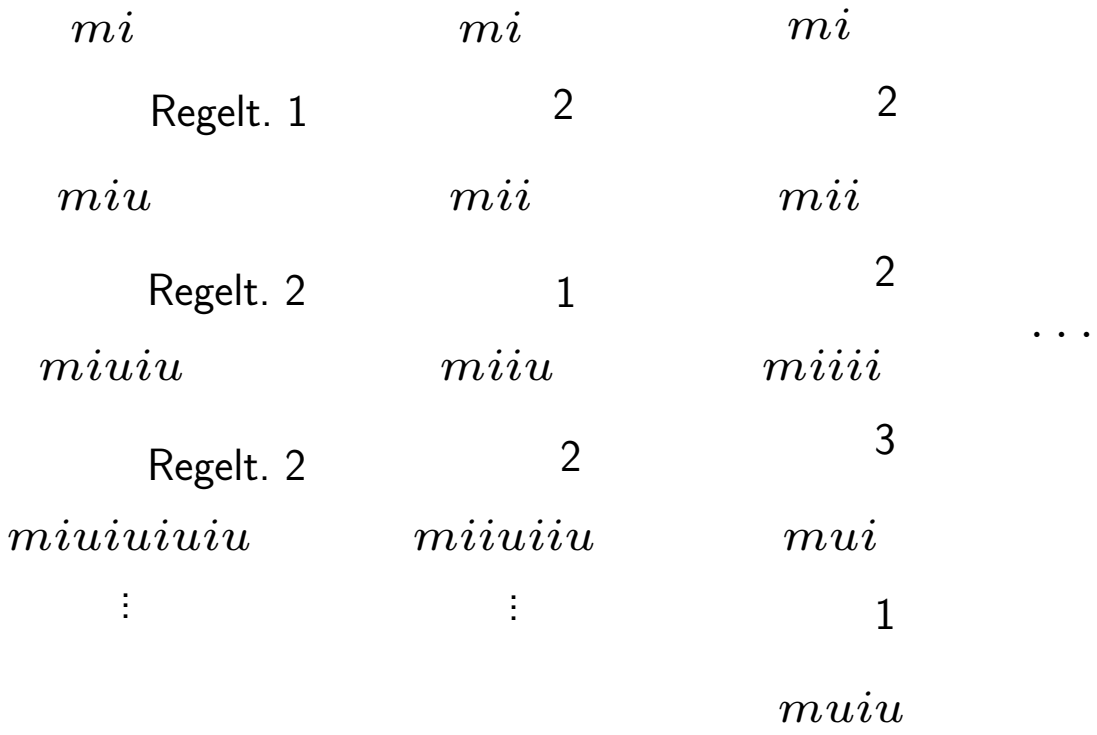
b) Ein Objekt φ ist in K **ableitbar**, falls es eine Ableitung in K mit letztem Objekt φ gibt.
   Schreibweise: $\underset{K}{\vdash} \varphi$.

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 $\frac{}{\kappa}$ $(\kappa \in M)$ ist. Schreibweise: $M \underset{K}{\vdash} \varphi$.

Beispiel 3.2 (Fort.)

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

b) Ableitbar aus mi sind z. B.



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

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.

$$\Pi_1 \qquad \cdots \qquad \Pi_n$$

Regel:

$$K$$

Ableitung $(\varphi_1, \ldots, \varphi_n)$: 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 $B_0 = B$, $B_{i+1} := B_i \cup \Gamma^1_K(B_i)$, wobei

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

„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 $l \longrightarrow r$ „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

$$x = ulv \text{ und } y = urv$$

$x \overset{n}{\underset{\Pi}{\vdash}} y$ 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 $\Sigma = \{a, b\}$

   $$aba ::= baab$$

   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.

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

2. Grammatiken als Wortersetzungssysteme

leftside::=rightside		(EBNF) Schreibweise für Produktionen
Statement Expression:		$A ::= B$
Assignment	$\rightsquigarrow$	$A ::= C$
MethodInvocation		$A ::= D$
ClassInstanceCreationExpression		⋮
⋮		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 $mi \underset{\Pi}{\vdash} w \rightsquigarrow 3 \not| \, |w|_i$

(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: $S : \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+ \qquad (\mathbb{N}^+ = \mathbb{N} - \{0\})$.

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

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 $f, g : A \to B$ $\quad f \sqsubseteq g$

„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: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\exists$, $=$
- 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

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

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.

$X : s \in V$ $\quad X$ 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 $c :\longrightarrow s$ in Σ eine Konstante $c_A \in s_A$, jedem Funktionssymbol $f : s_1 \times \cdots \times s_n \longrightarrow s$ in Σ eine totale Funktion $f_A : s_{1A} \times \cdots \times s_{nA} \longrightarrow s_A$ 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: $z : V \longrightarrow A$ (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		$0_N = 0 \in \mathbb{N}$
$succ$		$succ_N(x) = x + 1$

- 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
  $(\{b\}, \{\text{true}, \text{false} :\rightarrow b, \text{not} : b \rightarrow b, \text{and}, \text{or} : b \times b \rightarrow b\})$
  Standard-Interpretation: z. B.
  $b_{Boolean} = \{W, F\}$
  $\text{true}_{Boolean} = W$ und $\text{false}_{Boolean} = F$
  $\text{or}_{Boolean}(F, F) = F$
  $\text{or}_{Boolean}(x, y) = W$ 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 $\varnothing :\longrightarrow set_d$ und $insert : set_d \times d \rightarrow set_d$
  Interpretation A erweitert um
  $(set_d)_{set_d(A)} :=$ alle endlichen Teilmengen von d_A
  $\varnothing_{set_d(A)} :=$ 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
  $z_1(X) = 0, \quad z_1(Y) = 3, \quad z_1(X_1) = \varnothing,$
  $z_1(X_2) = \{0, 1, 3\}$ und
  $z_2(X) = 1, \quad z_2(Y) = 0, \quad z_2(X_1) = \{0, 1, 3\},$
  $z_2(X_2) = \varnothing$ Zustände.

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

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 $c :\longrightarrow s$ in Σ, so ist c ein Term vom Typ s.
- Ist $f : s_1 \times \cdots \times s_n \longrightarrow s$ in Σ, t_i ein Term über $(S, \Sigma), V$ 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 $Typ : Term(S, \Sigma, V) \longrightarrow S$

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

$\frac{}{c}\ (c :\longrightarrow s \in \Sigma) \qquad Typ(c) = s$

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

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

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-, $(\varphi \rightarrow \psi)$ -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 $(\varphi * \psi)$ $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.
- 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 $\mathbf{Form(S, \Sigma, V)}$

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

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

Substitution (Forts.)

4.11 Lemma

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

Beweis: einfache strukturelle Induktion.

4.12 Beispiel

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

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

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

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

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

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:**

$$\begin{aligned} A \quad &\models \mathsf{and}(X, Y) = \mathsf{and}(Y, X) \\ &\models \mathsf{or}(X, \mathsf{not}(X)) = \mathsf{true} \\ &\models \mathsf{and}(X, \mathsf{not}(X)) = \mathsf{false} \\ &\models \mathsf{not}(\mathsf{and}(\mathsf{not}(X), \mathsf{not}(Y))) = \mathsf{or}(X, Y) \end{aligned}$$

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

$$\begin{aligned} A \quad &\models (\varphi \vee \neg\varphi) \\ &\models \varphi \leftrightarrow \neg\neg\varphi \\ &\models (\varphi \vee \psi) \leftrightarrow (\neg\varphi \rightarrow \psi) \\ &\models (\varphi \wedge \psi) \leftrightarrow \neg(\varphi \rightarrow \neg\psi) \end{aligned}$$

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:

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

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“

$$
\begin{aligned}
&= \exists Z[Z = succ(Y)]\{Y/X\} \\
&= \exists Z\ Z = succ(X)
\end{aligned}
$$

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

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

$$\begin{array}{ll}
\text{gdw } A \models_{z(Y/b)} [[\psi]\{Z/Y\}]\{\vec{X}/\vec{t}\} & \text{für alle } b \text{ in Typ}(Y)_A \\
\overset{(IV)}{\text{gdw}}\ A \models_{z(Y/b)(\vec{X}/\vec{a})} [\psi]\{Z/Y\} & \text{für alle } b \\
\overset{(IV)}{\text{gdw}}\ A \models_{z(Y/b)(\vec{X}/\vec{a})(Z/b)} \psi & \text{für alle } b \\
\text{gdw } A \models_{z(\vec{X}/\vec{a})(Z/b)} \psi & \text{für alle } b \\
\text{gdw } A \models_{z(\vec{X}/\vec{a})} \forall Z\psi &
\end{array}$$

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.

$\overline{\varepsilon}$ — ε **leeres Programm**

$\overline{X := t;}$ — $X : s \in V, t \in \text{Term}(S, \Sigma, V)$, $\text{Typ}(t) = s$ **Zuweisung**

$\dfrac{\beta, \gamma}{\underline{\textbf{if}}\ B\ \underline{\textbf{then}}\ \beta\ \underline{\textbf{else}}\ \gamma\ \underline{\textbf{end}};}$ — B Boolesche-Formel über $(S, \Sigma), V$ **Test**

$\dfrac{\beta}{\underline{\textbf{while}}\ B\ \underline{\textbf{do}}\ \beta\ \underline{\textbf{end}};}$ — 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 $\mathcal{Z}(A, V)$.

- $z[[\varepsilon]]_A z' \text{ gdw } z = z'$.
- $z[[X := t]]_A z' \text{ gdw } z' = z(X/val_{A,z}(t))$
- $z[[$**if** B **then** β **else** γ **end**; $]]_A z' \text{ gdw}$
  $(A \models_z B$ und $z[[\beta]]_A z')$ oder $(A \not\models_z B$ und $z[[\gamma]]_A z')$.
- $z[[$**while** B **do** β **end**; $]]_A z' \text{ gdw}$
  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 $0 \leq i < n$
  - $A \not\models_{z_n} B$
  - $z_n = z'$
- $z[[\alpha\beta]]_A z' \text{ gdw}$ 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:

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

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

Beachte in $I_A(\underline{\textbf{while}}\ B\ \underline{\textbf{do}}\ \gamma\ \underline{\textbf{end}};\ \beta, z)$ ist „Restprogramm“ strukturell komplexer als Ausgangsprogramm (im Fall $A \models_z B$).

Beispiel (Fort.)

4.30 Beispiel Im Beispiel 4.26: While-Programm $\alpha = S_1 S_2 S_3$ über Signatur von Nat.

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

$$
\begin{aligned}
I_A^9(\alpha, z) &= I_A^9(S_1 S_2 S_3, z) = I_A^7(S_3, z(Y/0, Z/0)) \\
&= I_A^6(Z := succ(Z + (Y + Y)); \\
&\qquad\qquad Y := succ(Y);\ S_3, z(Y/0, Z/0)) \\
&= I_A^5(Y := succ(Y);\ S_3, z(Y/0, Z/1)) \\
&= I_A^4(S_3, z(Y/1, Z/1)) \\
&= I_A^3(Z := succ(Z + (Y + Y)); \\
&\qquad\qquad Y := succ(Y);\ S_3, z(Y/1, Z/1)) \\
&\ldots \\
&= I_A(S_3, z(Y/2, Z/4)) \\
&= (\varepsilon, z(Y/2, Z/4))
\end{aligned}
$$

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)
  $z[[\alpha]]_A z' \quad \text{gdw} \quad z' = z(X/val_{A,z}(t))$
  $\text{gdw} \quad I_A(\alpha, z) = (\varepsilon, z')$ (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 $z_0, \ldots, z_m$ 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 $n_i (i = 0, \ldots, m-1)$ mit $I_A^{n_i}(\beta, z_i) = (\varepsilon, z_{i+1})$. Mit $n := n_0 + \ldots + n_{m-1}$ 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 $f : s_A^1 \times \cdots \times s_A^n \longrightarrow s_A$ 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:

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

$f(z(X_1), \ldots, z(X_n)) \uparrow$
$\rightsquigarrow$ 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:

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

Regeln für Zuweisungen:

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

Regeln für Testanweisungen:

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

Kalkül zur Ableitung partieller Korrektheitsaussagen (2)

Regeln für Schleifen:

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

ξ wird **Schleifeninvariante** genannt.

Regeln für Anweisungsfolgen:

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

Für eine Algebra A ist $\mathbf{HC}(A)$ 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:

- $\{\varphi\}X := t;\ \{\psi\}$

- $\{\varphi\}$**if** B **then** $\{(\varphi \wedge B)\}\alpha\{\psi\}$ **else** $\{(\varphi \wedge \neg B)\}\beta\{\psi\}$ **end**; $\{\psi\}$

- $\{\varphi\}\{\xi\}$**while** B **do** $\{(\xi \wedge B)\}\alpha\{\xi\}$ **end**; $\{(\xi \wedge \neg B)\}\{\psi\}$

- $\{\varphi\}\alpha\{\xi\}\beta\{\psi\}$

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:
$A \models (\varphi \rightarrow \psi)$,
wenn $\{\varphi\}\{\psi\}$ oder $\{\varphi\}\varepsilon\{\psi\}$ im kommentierten Programmtext vorkommt, bzw.
$A \models (\varphi \rightarrow [\psi]\{X/t\})$,
wenn $\{\varphi\}X := t;\ \{\psi\}$ im kommentierten Programmtext vorkommt.

Notation für Ableitungen im hoareschen Kalkül (Forts.)

Vollständig kommentierte Programme:

Zwischen zwei Anweisungen stets Kommentar.

$$
\begin{array}{l}
\{\varphi\} \\
\quad A_1 \\
\{\ldots\} \\
\{\ldots\} \\
\quad A_2 \\
\{\ldots\} \\
\{\ldots\} \\
\quad A_3 \\
\{\ldots\} \\
\{\ldots\} \\
\quad \ldots \\
\{\ldots\} \\
\{\ldots\} \\
\quad A_n \\
\{\psi\}
\end{array}
$$

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 !

5.7 Beispiel

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

$$
\begin{array}{ll}
 & \{\text{true}\} \\
 & Y := 0; \\
 & \{Y = 0\} \\
 & Z := 0; \\
\varphi :: & \{Y = 0 \wedge Z = 0\} \\
\xi :: & \{(Y < X \vee Y = X) \wedge Z = (Y * Y)\} \\
 & \underline{\textbf{while}}\ \neg Y = X\ \underline{\textbf{do}} \\
\xi \wedge B :: & \{(Y < X \vee Y = X) \wedge Z = Y * Y \wedge \neg Y = X\} \\
 & Z := succ(Z + (Y + Y)); \\
 & \{(Y < X \wedge Z = (succ(Y) * succ(Y))\} \\
 & Y := succ(Y); \\
\xi :: & \{(Y < X \vee Y = X) \wedge Z = (Y * Y)\} \\
 & \underline{\textbf{end}}; \\
\xi \wedge \neg B :: & \{((Y < X \vee Y = X) \wedge Z = (Y * Y)) \wedge \\
 & \neg\neg Y = X\} \\
 & \{Z = (X * X)\}
\end{array}
$$

Syntaktisch korrekt kommentiert, vollständig.

Beispiel (Forts.)

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

$$
\begin{aligned}
(1)Nat &\models \text{true} \rightarrow 0 = 0 \\
(2)Nat &\models Y = 0 \rightarrow (Y = 0 \wedge 0 = 0) \\
(3)Nat &\models (Y = 0 \wedge Z = 0) \rightarrow ((Y < X \vee Y = X) \wedge \\
&\quad Z = (Y * Y)) \\
(4)Nat &\models (((Y < X \vee Y = X) \wedge Z = Y * Y) \wedge \neg Y = X) \\
&\quad \rightarrow (Y < X \wedge succ(Z + (Y + Y)) = \\
&\quad succ(Y) * succ(Y)) \\
(5)Nat &\models (Y < X \wedge Z = succ(Y) * succ(Y)) \rightarrow \\
&\quad (succ(Y) < X \vee succ(Y) = X) \wedge Z = \\
&\quad succ(Y) * succ(Y) \\
(6)Nat &\models (((Y < X \vee Y = X) \wedge Z = Y * Y) \wedge \\
&\quad \neg\neg Y = X) \rightarrow Z = X * X
\end{aligned}
$$

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 $A \models \varphi \to \xi$)
$A \models_{z_0} (\xi \wedge B)$ (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 $A \models (\xi \wedge \neg B) \to \psi$)
$A \models_{z'} \psi$ 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:

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

Dies folgt, da die Regel durch Anwenden der Regeln

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

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

für $(i = 2, \ldots, n)$ und iteriertes Anwenden der Anweisungsfolgenregel 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

$A \models (\varphi \to [\cdots[\psi]\{X_n/t_n\}\cdots]\{X_1/t_1\})$

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.

$$\{(X = X' \wedge Y = Y')\}$$
$$Z := X;\ X := Y;\ Y := Z;$$
$$\{(Y = X' \wedge X = Y')\}$$

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

$- : nat \times nat \to nat$ 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

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

φ:: $\{X > 0, Y > 0\}$
$A := X;\ B := Y;$
$\{\xi :: \{X > 0, Y > 0, \mathrm{GGT}(X, Y) = \mathrm{GGT}(A, B)\}\}$
while $A \neq B$ **do** $\{\xi, A \neq B\}$
if $B < A$
then $\{\xi, A \neq B, B < A\}$
$A := A - B;$ $\{\xi\}$
else $\{\xi, A \neq B, \neg B < A\}$
$B := B - A;$ $\{\xi\}$
end; $\{\xi\}$
end; $\{\xi, A = B\}$

ψ:: $\{A = \mathsf{GGT}(X, Y)\}$

Beispiel GGT-Berechnung (Forts.)

Die nachzuweisenden Beweisverpflichtungen in Nat sind:

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

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.

$\mathrm{div} : nat \times nat \to nat$ ganzzahlige Division
$\mathrm{mod} : nat \times nat \to nat$ 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 $\rightsquigarrow$ Logik.

Verwendet werden im Programm nur

$$Y \mathrm{mod}\ 2 \text{ und } Y \mathrm{div}\ 2$$

Programm

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

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

$\{0 = 0\}$
$X := A;\ Y := B;\ Z := 1;$
$\{\xi :: \{X^Y * Z = A^B\}$
while $\neg Y = 0$ **do**
$\{ \qquad \}$
if $Y \bmod 2 = 0$
then
$\{ \qquad \}$
$Y := Y \operatorname{div} 2;\ X := X * X;$
$\{ \qquad \}$
else
$\{ \qquad \}$
$Y := Y - 1;\ Z := Z * X;$
$\{ \qquad \}$
end;
$\{ \qquad \}$
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 $A \models (\xi \to W_{\alpha,\psi})$
  (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 $A \models (S_{\varphi,\alpha} \to \xi)$.
  (d. h. sie impliziert jede andere Nachbedingung).
- $wlp_A(X := t, \psi)$ definierbar durch $W_{X:=t,\psi} = [\psi]\{X/t\}$.
- $S_{\varphi,X:=t} = \exists Y([\varphi]\{X/Y\} \wedge X = t\{X/Y\})$.
  $(Y \notin \text{Var}(X, t, \varphi))$ (Definiert $spc_A(\varphi, X := t)$)

Beispiele

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

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

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 $(nat, 0 \to nat, succ : nat \to nat)$.
$\mathbb{N}$ 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 $z' \in wlp_A(\underline{\textbf{while}}\ B\ \underline{\textbf{do}}\ \beta\ \underline{\textbf{end}};\ , \psi)$.
  Sei also $z'[[\underline{\textbf{while}}\ B\ \underline{\textbf{do}}\ \beta\ \underline{\textbf{end}};\ ]]_A z''$.
  Wegen $A \models_z B$ und $z[[\beta]]_A z'$, gilt auch $z[[\underline{\textbf{while}}\ B\ \underline{\textbf{do}}\ \beta\ \underline{\textbf{end}};\ ]]_A z''$.
  Wegen $A \models_z \xi$ gilt auch $z \in wlp_A(\underline{\textbf{while}}\ B\ \underline{\textbf{do}}\ \beta\ \underline{\textbf{end}};\ , \psi)$
  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 $z[[\underline{\textbf{while}}\ B\ \underline{\textbf{do}}\ \beta\ \underline{\textbf{end}};\ ]]_A z$. Wegen $A \models_z \xi$ kann man $z \in wlp_A(\underline{\textbf{while}}\ B\ \underline{\textbf{do}}\ \beta\ \underline{\textbf{end}};\ , \psi)$ 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 nichtrekursive 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 $0 :\longrightarrow nat, succ : nat \rightarrow nat$

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: $\{\text{true}\}$ **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 $P(t_1, \ldots, t_n, U_1, \ldots, U_m);$ wobei Typ(t_i) = Typ(X_i)
$U_1, \ldots, U_m$ kommen nicht in $t_1, \ldots, t_n$ vor: Nebenwirkungsfrei.

Beispiele für Prozeduren

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

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

proc $PRED'$(**in** $X : nat$, **out** $Z : nat$)
  $\{\text{true}\}$ **call** $PRED'(X, Z)$; $\{(X = 0 \wedge Z = 0) \vee$
                $succ(Z) = X\}$
  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** $Q(\vec{t} \ldots X_i \ldots)$, 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

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

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** $P91(X, Y); \ \{\psi(X, Y)\}$
  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 $\psi(X, Y)$ ::
$((X > 100 \rightarrow Y = (X - 10)) \wedge (X \leq 100 \rightarrow Y = 91))$
Behauptung:

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

Syntaktisch korrekte Kommentierung von β:

$\{\text{true}\}$
if $X > 100$
  then
    $\{X > 100\}Y := X - 10;\ \{\psi(X, Y)\}$
  else $\{\neg X > 100\}$
    call $P91(X + 11, U);$
    $\{\neg X > 100 \wedge [\psi]\{X/X + 11, Y/U\}\}$
    call $P91(U, Y);$
    $\{\psi(X, Y)\}$
end;
$\{\psi(X, Y)\}$

Beweisverpflichtungen

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

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

Fall: $z(X) =$

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

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

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

$\rightsquigarrow$ Behauptung