Was bisher geschah

Turing = Menge aller durch eine DTM berechenbaren Funktionen

GOTO = Menge aller durch Goto-Programme berechenbaren Funktionen

- Syntax: Goto-Programm = Liste mit Elementen
  Inc i, Dec i, Goto k, GotoZ i k, Stop
- Semantik: small step (Zeilen, Anweisungen)
  Konfigurationen $(c, s) \in C = \mathbb{N} \times \mathbb{N}^{\mathbb{N}}$ (Programmzähler, Speicherbelegung)
  Konfigurationsübergänge $\text{step}_p^* \subseteq C^2$ für Goto-Programm p

WHILE = Menge aller durch While-Programme ber.-baren Funktionen

- Syntax: While-Programm = Baum mit
  Blättern: Inc i, Dec i, Skip
  inneren Knoten: While i p, Seq p q, IfZ i p q
- Semantik: big step (Teilbäume, Unterprogramme)
  Konfigurationen $s \in \mathbb{N}^{\mathbb{N}}$ (Speicherbelegung)
  Konfigurationsübergänge $\text{sem}_p \subseteq \mathbb{N}^{\mathbb{N}} \times \mathbb{N}^{\mathbb{N}}$ für While-Programm p

Satz: TURING = WHILE = GOTO

Universelle Programme

schon gezeigt:

- Existenz universeller TM
  (können Ausführung jeder TM simulieren / interpretieren)
- Turing = GOTO = WHILE

Also existieren auch (lassen sich durch Compiler konstruieren)

- universelle Goto-Programme
- universelle While-Programme

zur Interpretation von DTM, Goto- und While-Programmen

Loop-Programme – Motivation

- While i q bedeutet: solange $s(i) > 0$ ist, q ausführen
- es gibt While-Programme, die nicht für jede Eingabe terminieren (es gibt $f \in \mathsf{WHILE}$ mit f nicht total)
- Loop i q bedeutet: q genau $s(i)$ mal ausführen (der Wert von i vor Beginn der Schleife)
- Loop-Programme terminieren ($\mathsf{LOOP} \subseteq \mathsf{TOTAL}$) Das ist softwaretechnisch nützlich.

1926 Vermutung von David Hilbert (andere Formulierung): $\mathsf{WHILE} \cap \mathsf{TOTAL} \subseteq \mathsf{LOOP}$

1927 Gabriel Sudan: Vermutung ist falsch und erstes veröffentlichtes Gegenbeispiel $f \in (\mathsf{WHILE} \cap \mathsf{TOTAL}) \setminus \mathsf{LOOP}$

Loop-Programme

Syntax und Semantik wie While-Programme, außer:

Syntax Loop(i, q) statt While(i, q),

Semantik wenn $p = \text{Loop}(i, q)$,
dann $\text{sem}_p(s, s') = \text{sem}_q^{s(i)}(s, s')$
Befehl q wird genau $s(i)$ mal ausgeführt
(der Wert von i, wenn die Schleife zu erstenmal betreten wird — egal, was später mit i passiert)

Beispiel: $\text{Seq}(\text{Loop}(1, \text{Inc}\,1), \text{Loop}(2, \text{Inc}\,0))$

Jede durch ein Loop-Programm berechenbare Funktion heißt Loop-berechenbar.

LOOP = Menge aller Loop-berechenbaren Funktionen

Beispiele: Addition, Subtraktion, Multiplikation, Potenz,
$n \mapsto n$ ist gerade, $n \mapsto n$ ist Quadratzahl, $n \mapsto n$ ist prim,...

Beispiel: Semantik von Loop-Programmen

$$p = \text{Seq}(\text{Loop } 1\ (\text{Inc } 1))(\text{Loop } 2\ (\text{Inc } 0))$$

$$\begin{aligned}
\text{sem}_p &= \text{sem}_{\text{Loop } 1\ (\text{Inc } 1)} \circ \text{sem}_{\text{Loop } 2\ (\text{Inc } 0)} \\
&= \text{sem}^{s(1)}_{\text{Inc } 1} \circ \text{sem}^{s(2)}_{\text{Inc } 0} \\
&= \{(s, s[1 := s(1) + 1]) \mid s \in \mathbb{N}^{\mathbb{N}}\}^{s(1)} \\
&\quad \circ \{(s', s'[0 := s'(0) + 1]) \mid s' \in \mathbb{N}^{\mathbb{N}}\}^{s(2)} \\
&= \{(s, s[1 := s(1) + s(1)]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \\
&\quad \circ \{(s', s'[0 := s'(0) + s(2)]) \mid s' \in \mathbb{N}^{\mathbb{N}}\} \\
&= \{(s, s[1 := s(1) + s(1)][0 := s(0) + s(2)]) \mid s \in \mathbb{N}^{\mathbb{N}}\}
\end{aligned}$$

p berechnet $f : \mathbb{N}^2 \to \mathbb{N}$ mit $\forall (x_1, x_2) : f(x_1, x_2) = x_2$
(mit Nebenwirkung $s(1) \mapsto 2s(1)$)
Modifikation:

$$q = \text{Seq}(\text{Loop } 1\ (\text{Inc } 0))(\text{Loop } 2\ (\text{Inc } 0))$$

q berechnet $f : \mathbb{N}^2 \to n$ mit $\forall (x_1, x_2) : f(x_1, x_2) = x_1 + x_2$

LOOP $\subseteq$ WHILE

$$\begin{aligned}\mathsf{Loop}(i,p) \mapsto q = \mathsf{Seq}(\mathsf{Seq}(\quad & \mathsf{While}(i, \mathsf{Seq}(\mathsf{Seq}(\mathsf{Inc}\, j, \mathsf{Inc}\, k), \mathsf{Dec}\, i)),\\ & \mathsf{While}(k, \mathsf{Seq}(\mathsf{Inc}\, i, \mathsf{Dec}\, k))))),\\ & \mathsf{While}(j, \mathsf{Seq}(p, \mathsf{Dec}\, j))))\end{aligned}$$

mit in p nicht verwendeten Registern j, k

$$\begin{aligned}\mathsf{sem}_q &= \mathsf{sem}_{\mathsf{While}(i,\mathsf{Seq}(\mathsf{Seq}(\mathsf{Inc}\, j,\mathsf{Inc}\, k),\mathsf{Dec}\, i))}\\ &\quad \circ \mathsf{sem}_{\mathsf{While}(k,\mathsf{Seq}(\mathsf{Inc}\, i,\mathsf{Dec}\, k))} \circ \mathsf{sem}_{\mathsf{While}(j,\mathsf{Seq}(p,\mathsf{Dec}\, j))}\\ &= \{(s, s[j := s(i), k := s(i), i := 0]) \mid s \in \mathbb{N}^{\mathbb{N}}\}\\ &\quad \circ\{(s, s[i := s(k), k := 0]) \mid s \in \mathbb{N}^{\mathbb{N}}\}\\ &\quad \circ \mathsf{sem}_{\mathsf{While}(j,\mathsf{Seq}(p,\mathsf{Dec}\, j))}\\ &= \{(s, s[j := s(i)]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \circ \mathsf{sem}_{\mathsf{While}(j,\mathsf{Seq}(p,\mathsf{Dec}\, j))}\\ &= \{(s, s[j := s(i)]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \circ \mathsf{sem}^{s(j)}_{\mathsf{Seq}(p,\mathsf{Dec}\, j)} \ (j \text{ kommt nicht in } p \text{ vor})\\ &= \{(s, s[j := s(i)]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \circ (\mathsf{sem}_p \circ \mathsf{sem}_{\mathsf{Dec}\, j})^{s(j)}\\ &= \{(s, s[j := s(i)]) \mid s \in \mathbb{N}^{\mathbb{N}}\} \circ \mathsf{sem}^{s(i)}_p \circ\{(s', s'[j := 0]) \mid s \in \mathbb{N}^{\mathbb{N}}\}\\ &= \mathsf{sem}^{s(i)}_p = \mathsf{sem}_{\mathsf{Loop}(i,p)}\end{aligned}$$

LOOP und WHILE

schon gezeigt: $\mathrm{LOOP} \subseteq \mathrm{WHILE} \cap \mathrm{TOTAL}$
$\mathrm{WHILE} \cap \mathrm{TOTAL} \not\subseteq \mathrm{LOOP}$

Beweis: Diagonalisierung

- $p_0, p_1, \ldots$ (z.B. quasi-lexikografische) Aufzählung aller LOOP-Programme, die einstellige Funktionen berechnen, $f_0, f_1, \ldots : \mathbb{N} \to \mathbb{N}$
- für $d : x \mapsto f_x(x) + 1$ gilt: $d \in \mathrm{WHILE} \cap \mathrm{TOTAL}$
  Begründung: Interpreter für LOOP-Programme (universelle TM / universelles While-Programm)
- Diese Funktion d kommt in der Aufzählung $f_0, f_1, \ldots$ nicht vor (also $d \notin \mathrm{LOOP}$)
  Begründung (indirekt):
  - Annahme $d \in \mathrm{LOOP}$
  - damit existiert ein $k \in \mathbb{N}$ mit $d = f_k$
  - Widerspruch bei $d(k) = f_k(k)$ (nach Def. von f_k) und $d(k) = f_k(k) + 1$ (nach Def. von d)
  - also ist die Annahme falsch und $d \notin \mathrm{LOOP}$

Ackermann-Funktion

(Wilhelm Friedrich Ackermann, 1928)

$A : \mathbb{N}^2 \to \mathbb{N}$ mit $\forall (x, y) \in \mathbb{N}^2$:

$$\begin{aligned} A(0, y) &= y + 1 \\ A(x + 1, 0) &= A(x, 1) \\ A(x + 1, y + 1) &= A(x, A(x + 1, y)) \end{aligned}$$

Beispiele: $A(0, 0), A(0, 1), A(0, 2),$
$A(1, 0), A(1, 1), A(1, 2),\ A(1, 0), A(2, 1), A(2, 2)$

- bestimme $A(2, 4), A(3, 3), A(4, 2)$ (ÜA)

Eigenschaften der Ackermann-Funktion

1. $\forall x, y \in \mathbb{N} : A(x, y) > y$
   Beweis durch Induktion nach x, innen Induktion nach y
2. $\forall x, y \in \mathbb{N} : A(x, y + 1) > A(x, y)$
   (Monotonie im zweiten Argument)
   Beweis (ohne Induktion): Fallunterscheidung $x = 0$, $x > 0$
3. $\forall x, y \in \mathbb{N} : A(x + 1, y) \geqslant A(x, y + 1)$
   Beweis durch Induktion nach y (ÜA)
4. $\forall x, y \in \mathbb{N} : A(x + 1, y) > A(x, y)$
   (Monotonie im ersten Argument)
5. $\forall x, x', y, y' \in \mathbb{N} : ((x \leqslant x' \wedge y \leqslant y') \rightarrow (A(x, y) \leqslant A(x', y')))$
   (Monotonie in beiden Argumenten)

Eigenschaften der Ackermann-Funktion (1)

1. $\forall x, y \in \mathbb{N} : A(x, y) > y$

Beweis durch Induktion nach x, innen Induktion nach y:

IAx: $x = 0$: $\forall y \in \mathbb{N} : A(0, y) = y + 1 > y$

ISx: IH x: $\forall y \in \mathbb{N} : A(x, y) > y$
IB x: $\forall y \in \mathbb{N} : A(x + 1, y) > y$
Beweis x: $\forall y \in \mathbb{N} : A(x + 1, y) > y$ durch Induktion über y

IAy: $y = 0$: $A(x + 1, 0) = A(x, 1) \overset{(IHx)}{>} 1 > 0$

ISy: IH y: $A(x + 1, y) > y$
IB y: $A(x + 1, y + 1) > y + 1$
Beweis y:

$A(x + 1, y + 1) = A(x, A(x + 1, y)) \overset{(IHx)}{>} A(x + 1, y) \overset{(IHy)}{>} y$
und damit $A(x + 1, y + 1) > y + 1$

Eigenschaften der Ackermann-Funktion (2)

2. $\forall x, y \in \mathbb{N} : A(x, y+1) > A(x, y)$

Beweis (ohne Induktion) durch Fallunterscheidung $x = 0$, $x > 0$:

$$x = 0: \; A(0, y+1) = y + 2 > y + 1 = A(0, y)$$

$$x > 0: \; A(x, y+1) = A(x-1, A(x, y)) \overset{1.}{>} A(x, y)$$

Eigenschaften der Ackermann-Funktion (4)

4. $\forall x, y \in \mathbb{N} : A(x+1, y) > A(x, y)$

Beweis:
$A(x+1, y) \overset{3}{\geqslant} A(x, y+1) \overset{2}{>} A(x, y)$

1-4 ergeben zusammengefasst:

5. Monotonie in beiden Argumenten:
$\forall x, x', y, y' \in \mathbb{N} : ((x \leqslant x' \wedge y \leqslant y') \rightarrow (A(x, y) \leqslant A(x', y')))$