Script generated by TTT

Title: Nipkow: Theo (13.06.2019)

Date: Thu Jun 13 14:14:04 CEST 2019

Duration: 90:32 min

Pages: 104

Analog zur Fallunterscheidung kann man auch eine TM für eine Schleife konstruieren

$$\longrightarrow \quad \text{Band } i = 0? \quad \overset{ja}{\longrightarrow}$$
$$\uparrow\downarrow\, nein$$
$$M$$

die sich wie `while` Band $i \neq 0$ `do` M verhält.

Moral: Mit TM kann man imperativ programmieren:

`:=`
`;`
`if`
`while`

5.4 WHILE- und GOTO-Berechenbarkeit

WHILE ≡ strukturierte Programme mit `while`-Schleifen
GOTO ≡ Assembler

5.4 WHILE- und GOTO-Berechenbarkeit

WHILE ≡ strukturierte Programme mit `while`-Schleifen
GOTO ≡ Assembler

Wir definieren WHILE- und GOTO-Berechenbarkeit und zeigen ihre Äquivalenz mit Turing-Berechenbarkeit.

5.4 WHILE- und GOTO-Berechenbarkeit

WHILE $\equiv$ strukturierte Programme mit `while`-Schleifen

GOTO $\equiv$ Assembler

Wir definieren WHILE- und GOTO-Berechenbarkeit und zeigen ihre Äquivalenz mit Turing-Berechenbarkeit.

Syntax von WHILE-Programmen:

$$
\begin{aligned}
P \;\to\; & X \texttt{ := } X \texttt{ + } C \\
\mid\; & X \texttt{ := } X \texttt{ - } C \\
\mid\; & P \texttt{ ; } P \\
\mid\; & \texttt{IF } X = 0 \texttt{ DO } P \texttt{ ELSE } Q \texttt{ END} \\
\mid\; & \texttt{WHILE } X \neq 0 \texttt{ DO } P \texttt{ END}
\end{aligned}
$$

wobei X eine der Variablen $x_0, x_1, \ldots$
und C eine der Konstanten $0, 1, \ldots$ sein kann.

Beispiel 5.17

`WHILE` $x_2 \neq 0$ `DO` x_1 `:=` x_0`+1 END`

Die modifizierte Differenz ist $m \dot{-} n := \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Die modifizierte Differenz ist $m \dot{-} n := \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

x_i `:=` x_j `+` n Neuer Wert von x_i ist $x_j + n$.

Die modifizierte Differenz ist $m \dot{-} n := \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

x_i `:=` x_j `+` n Neuer Wert von x_i ist $x_j + n$.

x_i `:=` x_j `-` n Neuer Wert von x_i ist $x_j \dot{-} n$.

Die modifizierte Differenz ist $m \dot{-} n := \begin{cases} m-n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

x_i := x_j + n	Neuer Wert von x_i ist $x_j + n$.
x_i := x_j - n	Neuer Wert von x_i ist $x_j \dot{-} n$.
P_1 ; P_2	Führe zuerst P_1 und dann P_2 aus.
WHILE $x_i \neq 0$ DO P END	Führe P bis die Variable x_i (wenn je) den Wert 0 annimmt.

Beispiel 5.18

WHILE $x_1 \neq 0$ DO x_2 := x_2 + 1; x_1 := x_1 - 1 END
simuliert
x_2 := x_2 + x_1

Die modifizierte Differenz ist $m \dot{-} n := \begin{cases} m-n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

x_i := x_j + n	Neuer Wert von x_i ist $x_j + n$.
x_i := x_j - n	Neuer Wert von x_i ist $x_j \dot{-} n$.
P_1 ; P_2	Führe zuerst P_1 und dann P_2 aus.
WHILE $x_i \neq 0$ DO P END	Führe P bis die Variable x_i (wenn je) den Wert 0 annimmt.

Beispiel 5.18

WHILE $x_1 \neq 0$ DO x_2 := x_2 + 1; x_1 := x_1 - 1 END
simuliert
x_2 := x_2 + x_1

Die modifizierte Differenz ist $m \dot{-} n := \begin{cases} m-n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

x_i := x_j + n	Neuer Wert von x_i ist $x_j + n$.
x_i := x_j - n	Neuer Wert von x_i ist $x_j \dot{-} n$.
P_1 ; P_2	Führe zuerst P_1 und dann P_2 aus.
WHILE $x_i \neq 0$ DO P END	Führe P bis die Variable x_i (wenn je) den Wert 0 annimmt.

Beispiel 5.18

WHILE $x_1 \neq 0$ DO x_2 := x_2 + 1; x_1 := x_1 - 1 END
simuliert
x_2 := x_2 + x_1; $X_1 := 0$

Zu Beginn der Ausführung stehen die Eingaben in $x_1, \ldots, x_k$.
Alle anderen Variablen sind 0. Die Ausgabe wird in x_0 berechnet.

Die modifizierte Differenz ist $m \dot{-} n := \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

x_i := x_j + n	Neuer Wert von x_i ist $x_j + n$.
x_i := x_j - n	Neuer Wert von x_i ist $x_j \dot{-} n$.
P_1 ; P_2	Führe zuerst P_1 und dann P_2 aus.
WHILE $x_i \neq 0$ DO P END	Führe P bis die Variable x_i (wenn je) den Wert 0 annimmt.

Beispiel 5.18

WHILE $x_1 \neq 0$ DO x_2 := x_2 + 1; $x_1 := x_1$ - 1 END
simuliert
x_2 := x_2 + x_1

Zu Beginn der Ausführung stehen die Eingaben in $x_1, \ldots, x_k$.
Alle anderen Variablen sind 0. Die Ausgabe wird in x_0 berechnet.

Syntaktische Abkürzungen („Zucker“):

x_i := x_j $\equiv$ x_i := x_j + 0

Syntaktische Abkürzungen („Zucker“):

x_i := x_j $\equiv$ x_i := x_j + 0

x_i := n $\equiv$ x_i := x_j + n

Syntaktische Abkürzungen („Zucker“):

x_i := x_j $\equiv$ x_i := x_j + 0

x_i := n $\equiv$ x_i := x_j + n

(wobei an x_j nirgends zugewiesen wird)

Syntaktische Abkürzungen („Zucker“):

x_i := x_j $\equiv$ x_i := x_j + 0

x_i := n $\equiv$ x_i := x_j + n

(wobei an x_j nirgends zugewiesen wird)

x_i := x_j + x_k $\equiv$ x_i := x_j;

(falls $i \neq k$)

```
WHILE x_k ≠ 0 DO
  x_i := x_i + 1; x_k := x_k - 1
END
```


Syntaktische Abkürzungen („Zucker“):

x_i := x_j $\equiv$ x_i := x_j + 0

x_i := n $\equiv$ x_i := x_j + n

(wobei an x_j nirgends zugewiesen wird)

x_i := x_j + x_k $\equiv$ x_i := x_j;

(falls $i \neq k$)

```
WHILE x_k ≠ 0 DO
  x_i := x_i + 1; x_k := x_k - 1
END
```

x_i := x_j * x_k $\equiv$ x_i := 0;

(falls $i \neq j, k$)

```
WHILE x_k ≠ 0 DO
  x_i := x_i + x_j; x_k := x_k - 1
END
```


Syntaktische Abkürzungen („Zucker“):

x_i := x_j $\equiv$ x_i := x_j + 0

x_i := n $\equiv$ x_i := x_j + n

(wobei an x_j nirgends zugewiesen wird)

x_i := x_j + x_k $\equiv$ x_i := x_j;

(falls $i \neq k$)

```
WHILE x_k ≠ 0 DO
  x_i := x_i + 1; x_k := x_k - 1
END
```

x_i := x_j * x_k $\equiv$ x_i := 0;

(falls $i \neq j, k$)

```
WHILE x_k ≠ 0 DO
  x_i := x_i + x_j; x_k := x_k - 1
END
```

DIV, MOD, geschachtelte Ausdrücke

Definition 5.19

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

Definition 5.19

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.)

Definition 5.19

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.)

- terminiert mit $f(n_1, \ldots, n_k)$ in x_0, falls $f(n_1, \ldots, n_k)$ definiert ist,

Definition 5.19

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.)

- terminiert mit $f(n_1, \ldots, n_k)$ in x_0, falls $f(n_1, \ldots, n_k)$ definiert ist,
- terminiert nicht, falls $f(n_1, \ldots, n_k)$ undefiniert ist.

Definition 5.19

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.)

- terminiert mit $f(n_1, \ldots, n_k)$ in x_0, falls $f(n_1, \ldots, n_k)$ definiert ist,
- terminiert nicht, falls $f(n_1, \ldots, n_k)$ undefiniert ist.

Turingmaschinen können WHILE-Programme simulieren:

Definition 5.19

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.)

- terminiert mit $f(n_1, \ldots, n_k)$ in x_0, falls $f(n_1, \ldots, n_k)$ definiert ist,
- terminiert nicht, falls $f(n_1, \ldots, n_k)$ undefiniert ist.

Turingmaschinen können WHILE-Programme simulieren:

Satz 5.20 (WHILE $\to$ TM)

Jede WHILE-berechenbare Funktion ist auch Turing-berechenbar.

Definition 5.19

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.)

- terminiert mit $f(n_1, \ldots, n_k)$ in x_0, falls $f(n_1, \ldots, n_k)$ definiert ist,
- terminiert nicht, falls $f(n_1, \ldots, n_k)$ undefiniert ist.

Turingmaschinen können WHILE-Programme simulieren:

Satz 5.20 (WHILE $\to$ TM)

Jede WHILE-berechenbare Funktion ist auch Turing-berechenbar.

Beweis:

Jede Programmvariable wird auf einem eigenen Band gespeichert.

257

Übersetzungen

Ein GOTO-Programm ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$$

(wobei alle Marken verschieden und optional sind)

Ein GOTO-Programm ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$$

(wobei alle Marken verschieden und optional sind)

Ein GOTO-Programm ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$$

(wobei alle Marken verschieden und optional sind)

Mögliche Anweisungen A_i sind:

x_i := x_j + n
x_i := x_j - n

Ein GOTO-Programm ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$$

(wobei alle Marken verschieden und optional sind)

Mögliche Anweisungen A_i sind:

x_i := x_j + n
x_i := x_j - n
GOTO M_i

Ein GOTO-Programm ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$$

(wobei alle Marken verschieden und optional sind)

Mögliche Anweisungen A_i sind:

x_i := x_j + n
x_i := x_j - n
GOTO M_i
IF $x_i = n$ GOTO M_j
HALT

Ein GOTO-Programm ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$$

(wobei alle Marken verschieden und optional sind)

Mögliche Anweisungen A_i sind:

x_i := x_j + n
x_i := x_j - n
GOTO M_i
IF $x_i = n$ GOTO M_j
HALT

Die Semantik ist wie erwartet.

Satz 5.22 (GOTO → WHILE)

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden.

Satz 5.22 (GOTO → WHILE)

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden.

Beweis: Simuliere $M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$ durch

pc := 1;
WHILE $pc \neq 0$ DO
  IF $pc - 1 = 0$ THEN P_1 ELSE
  ⋮
  IF $pc - k = 0$ THEN P_k ELSE $pc := 0$ END ... END
END

wobei $A_i \mapsto P_i$ wie folgt definiert ist:

x_i := x_j +/- n	$\mapsto$	x_i := x_j +/- n; pc := pc + 1
GOTO M_i	$\mapsto$	pc := i
IF $x_j = 0$ GOTO M_i	$\mapsto$	IF $x_j = 0$ THEN pc := i ELSE pc := pc + 1 END
HALT	$\mapsto$	pc := 0

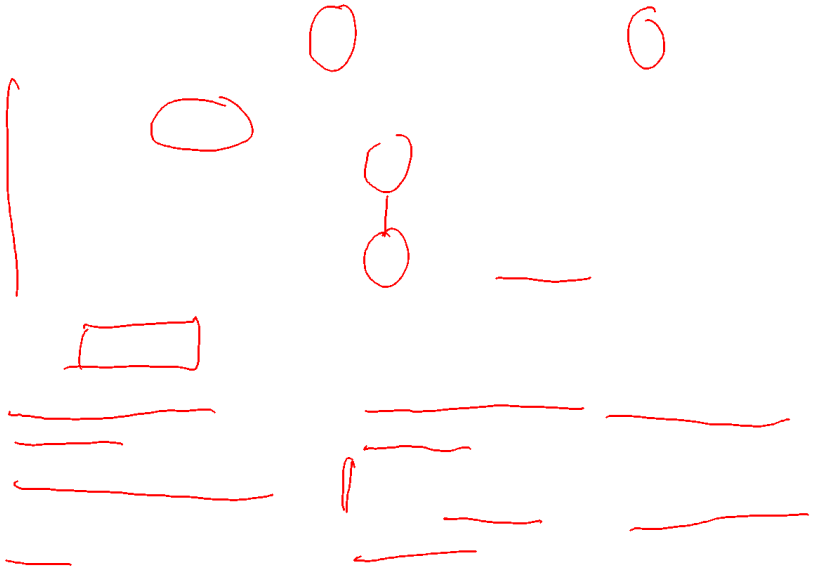
□

Satz 5.22 (GOTO → WHILE)

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden.

Satz 5.22 (GOTO $\rightarrow$ WHILE)

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden.



Korollar 5.23

WHILE- und GOTO-Berechenbarkeit sind äquivalent.

Korollar 5.23

WHILE- und GOTO-Berechenbarkeit sind äquivalent.

Korollar 5.24 (Kleenesche Normalform)

Jedes WHILE-Programm ist zu einem WHILE-Programm mit genau einer WHILE-Schleife äquivalent.

Korollar 5.23

WHILE- und GOTO-Berechenbarkeit sind äquivalent.

Korollar 5.23

WHILE- und GOTO-Berechenbarkeit sind äquivalent.

Satz 5.25 (TM → GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Satz 5.25 (TM → GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Übersetzung: TM $(Q, \Sigma, \Gamma, \delta, q_0, \square, F) \rightarrow$ GOTO-Programm.

Satz 5.25 (TM → GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Übersetzung: TM $(Q, \Sigma, \Gamma, \delta, q_0, \square, F) \rightarrow$ GOTO-Programm.

$$Q = \{q_0, \ldots, q_k\} \qquad \Gamma = \{a_0 (= \square), \ldots, a_n\}$$

Eine Konfiguration

$$(a_{i_p} \ldots a_{i_1}, q_l, a_{j_1} \ldots a_{j_q})$$

wird durch die Programmvariablen x, y, z wie folgt repräsentiert:

$$x = (i_p \ldots i_1)_b, \quad y = (j_q \ldots j_1)_b, \quad z = l$$

wobei $(i_p \ldots i_1)_b$ die Zahl $i_p \ldots i_1$ zur Basis $b := n + 1$ ist:

$$x = \sum_{r=1}^{p} i_r b^{r-1}$$

Der Kern des GOTO-Programms ist die iterierte Simulation von δ:

M: IF $z \in F$ GOTO M_{end};
a := y MOD b;
IF $z = 0$ AND $a = 0$ GOTO M_{00};
IF $z = 0$ AND $a = 1$ GOTO M_{01};
...
IF $z = k$ AND $a = n$ GOTO M_{kn};
M_{00}: P_{00}; GOTO M;
M_{01}: P_{01}; GOTO M;
...
M_{kn}: P_{kn}; GOTO M

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist.

$F = \{3, 5\}$

IF $z = 3$ GOTO M_{end}

IF $z = 5$ —"—

Satz 5.25 (TM → GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Übersetzung: TM $(Q, \Sigma, \Gamma, \delta, q_0, \square, F) \rightarrow$ GOTO-Programm.

$$Q = \{q_0, \ldots, q_k\} \qquad \Gamma = \{a_0 (= \square), \ldots, a_n\}$$

Eine Konfiguration

$$(a_{i_p} \ldots a_{i_1},\ q_l,\ a_{j_1} \ldots a_{j_q})$$

wird durch die Programmvariablen x, y, z wie folgt repräsentiert:

$$x = (i_p \ldots i_1)_b, \quad y = (j_q \ldots j_1)_b, \quad z = l$$

wobei $(i_p \ldots i_1)_b$ die Zahl $i_p \ldots i_1$ zur Basis $b := n + 1$ ist:

$$x = \sum_{r=1}^{p} i_r b^{r-1}$$

Der Kern des GOTO-Programms ist die iterierte Simulation von δ:

M: IF $z \in F$ GOTO M_{end};
a := y MOD b;
IF $z = 0$ AND $a = 0$ GOTO M_{00};
IF $z = 0$ AND $a = 1$ GOTO M_{01};
...
IF $z = k$ AND $a = n$ GOTO M_{kn};
M_{00}: P_{00}; GOTO M;
M_{01}: P_{01}; GOTO M;
...
M_{kn}: P_{kn}; GOTO M

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist.

Der Kern des GOTO-Programms ist die iterierte Simulation von δ:

M: IF $z \in F$ GOTO M_{end};
a := y MOD b;
IF $z = 0$ AND $a = 0$ GOTO M_{00};
IF $z = 0$ AND $a = 1$ GOTO M_{01};
...
IF $z = k$ AND $a = n$ GOTO M_{kn};
M_{00}: P_{00}; GOTO M;
M_{01}: P_{01}; GOTO M;
...
M_{kn}: P_{kn}; GOTO M

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist. Für $D = L$:

Der Kern des GOTO-Programms ist die iterierte Simulation von δ:

M: IF $z \in F$ GOTO M_{end};
a := y MOD b;
IF $z = 0$ AND $a = 0$ GOTO M_{00};
IF $z = 0$ AND $a = 1$ GOTO M_{01};
...
IF $z = k$ AND $a = n$ GOTO M_{kn};
M_{00}: P_{00}; GOTO M;
M_{01}: P_{01}; GOTO M;
...
M_{kn}: P_{kn}; GOTO M

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist. Für $D = L$:

Der Kern des GOTO-Programms ist die iterierte Simulation von δ:

M: IF $z \in F$ GOTO M_{end};
a := y MOD b;
IF $z = 0$ AND $a = 0$ GOTO M_{00};
IF $z = 0$ AND $a = 1$ GOTO M_{01};
...
IF $z = k$ AND $a = n$ GOTO M_{kn};
M_{00}: P_{00}; GOTO M;
M_{01}: P_{01}; GOTO M;
...
M_{kn}: P_{kn}; GOTO M

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist. Für $D = L$:

z := r; Zustand aktualisieren
y := y DIV b; Löschen von a_j

264

Der Kern des GOTO-Programms ist die iterierte Simulation von δ:

M: IF $z \in F$ GOTO M_{end};
a := y MOD b;
IF $z = 0$ AND $a = 0$ GOTO M_{00};
IF $z = 0$ AND $a = 1$ GOTO M_{01};
...
IF $z = k$ AND $a = n$ GOTO M_{kn};
M_{00}: P_{00}; GOTO M;
M_{01}: P_{01}; GOTO M;
...
M_{kn}: P_{kn}; GOTO M

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist. Für $D = L$:

z := r; Zustand aktualisieren
y := y DIV b; Löschen von a_j

y := b*y + s; Schreiben von a_s

264

Der Kern des GOTO-Programms ist die iterierte Simulation von δ:

M: IF $z \in F$ GOTO M_{end};
a := y MOD b;
IF $z = 0$ AND $a = 0$ GOTO M_{00};
IF $z = 0$ AND $a = 1$ GOTO M_{01};
...
IF $z = k$ AND $a = n$ GOTO M_{kn};
M_{00}: P_{00}; GOTO M;
M_{01}: P_{01}; GOTO M;
...
M_{kn}: P_{kn}; GOTO M

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist. Für $D = L$:

z := r; Zustand aktualisieren
y := y DIV b; Löschen von a_j
y := b*y + s; Schreiben von a_s

y := b*y + (x MOD b); Bewegung L (I): Einfügen von a_{i_1} in y

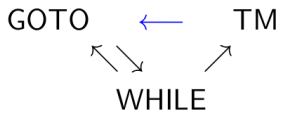
264

Der Kern des GOTO-Programms ist die iterierte Simulation von δ:

M: IF $z \in F$ GOTO M_{end};
IF a := y MOD b;
IF $z = 0$ AND $a = 0$ GOTO M_{00};
IF $z = 0$ AND $a = 1$ GOTO M_{01};
...
IF $z = k$ AND $a = n$ GOTO M_{kn};
M_{00}: P_{00}; GOTO M;
M_{01}: P_{01}; GOTO M;
...
M_{kn}: P_{kn}; GOTO M

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist. Für $D = L$:

z := r;	Zustand aktualisieren
y := y DIV b;	Löschen von a_j
y := b*y + s;	Schreiben von a_s
y := b*y + (x MOD b);	Bewegung L (I): Einfügen von a_{i_1} in y
x := x DIV b	Bewegung L (II): Löschen von a_{i_1} aus x



Übersetzungen

5.5 Unentscheidbarkeit des Halteproblems

Ziel: Es ist unentscheidbar ob ein Programm terminiert.

5.5 Unentscheidbarkeit des Halteproblems

Ziel: Es ist unentscheidbar ob ein Programm terminiert.

5.5 Unentscheidbarkeit des Halteproblems

Ziel: Es ist unentscheidbar ob ein Programm terminiert.

Definition 5.26
Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt entscheidbar gdw ihre charakteristische Funktion

$$\chi_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

5.5 Unentscheidbarkeit des Halteproblems

Ziel: Es ist unentscheidbar ob ein Programm terminiert.

Definition 5.26
Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt entscheidbar gdw ihre charakteristische Funktion

$$\chi_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

Eine Eigenschaft/Problem $P(x)$ heißt entscheidbar gdw $\{x \mid P(x)\}$ entscheidbar ist.

5.5 Unentscheidbarkeit des Halteproblems

Ziel: Es ist unentscheidbar ob ein Programm terminiert.

Definition 5.26
Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt entscheidbar gdw ihre charakteristische Funktion

$$\chi_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

Eine Eigenschaft/Problem $P(x)$ heißt entscheidbar gdw $\{x \mid P(x)\}$ entscheidbar ist.

Fakt 5.27
Die entscheidbaren Mengen sind abgeschlossen unter Komplement: Ist A entscheidbar, dann auch $\overline{A}$.

Kodierung einer TM als Wort über $\Sigma = \{0, 1\}$, exemplarisch:

Kodierung einer TM als Wort über $\Sigma = \{0, 1\}$, exemplarisch:

Kodierung einer TM als Wort über $\Sigma = \{0, 1\}$, exemplarisch:

- Sei $\Gamma = \{a_0, \ldots, a_k\}$ und $Q = \{q_0, \ldots, q_n\}$.

Kodierung einer TM als Wort über $\Sigma = \{0, 1\}$, exemplarisch:

- Sei $\Gamma = \{a_0, \ldots, a_k\}$ und $Q = \{q_0, \ldots, q_n\}$.
- $\delta(q_i, a_j) = (q_{i'}, a_{j'}, d)$ wird kodiert als

$$\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#bin(m)$$

wobei $bin : \mathbb{N} \to \{0, 1\}^*$ die Binärkodierung einer Zahl ist und $m = 0/1/2$ falls $d = L/R/N$.

Kodierung einer TM als Wort über $\Sigma = \{0, 1\}$, exemplarisch:

- Sei $\Gamma = \{a_0, \ldots, a_k\}$ und $Q = \{q_0, \ldots, q_n\}$.
- $\delta(q_i, a_j) = (q_{i'}, a_{j'}, d)$ wird kodiert als

$$\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#bin(m)$$

wobei $bin : \mathbb{N} \to \{0, 1\}^*$ die Binärkodierung einer Zahl ist und $m = 0/1/2$ falls $d = L/R/N$.

- Kodierung von δ: Konkatenation der Kodierungen aller $\delta(.,.) = (.,.,.)$, in beliebiger Reihenfolge.

Kodierung einer TM als Wort über $\Sigma = \{0, 1\}$, exemplarisch:

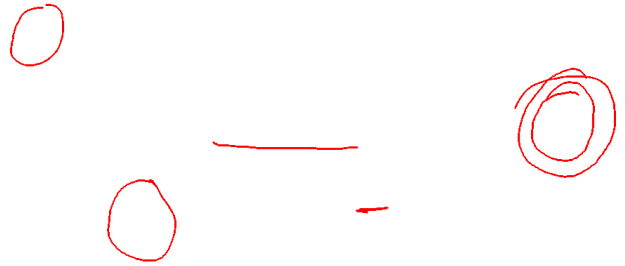
- Sei $\Gamma = \{a_0, \ldots, a_k\}$ und $Q = \{q_0, \ldots, q_n\}$.
- $\delta(q_i, a_j) = (q_{i'}, a_{j'}, d)$ wird kodiert als

$$\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#bin(m)$$

  wobei $bin : \mathbb{N} \to \{0, 1\}^*$ die Binärkodierung einer Zahl ist und $m = 0/1/2$ falls $d = L/R/N$.
- Kodierung von δ: Konkatenation der Kodierungen aller $\delta(.,.) = (.,.,.)$, in beliebiger Reihenfolge.
- Kodierung von $\{0, 1, \#\}^*$ in $\{0, 1\}^*$:

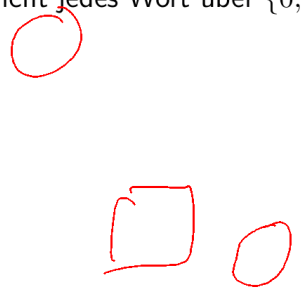
$$\begin{array}{rcl} 0 & \mapsto & 00 \\ 1 & \mapsto & 01 \\ \# & \mapsto & 11 \end{array}$$

Nicht jedes Wort über $\{0, 1\}^*$ kodiert eine TM.



Nicht jedes Wort über $\{0, 1\}^*$ kodiert eine TM.

Nicht jedes Wort über $\{0, 1\}^*$ kodiert eine TM.



Nicht jedes Wort über $\{0,1\}^*$ kodiert eine TM.

Sei $\hat{M}$ eine beliebige feste TM.

Definition 5.28
Die zu einem Wort $w \in \{0,1\}^*$ gehörige TM M_w ist

$$M_w := \begin{cases} M & \text{falls } w \text{ Kodierung von } M \text{ ist} \\ \hat{M} & \text{sonst} \end{cases}$$

Die Kodierung von syntaktischen Objekten (Programmen, Formeln, etc) als Zahlen nennt man Gödelisierung und die Zahlen Gödelnummern.

Definition 5.29

$M[w]$ ist Abk. für „Maschine M mit Eingabe w“

Definition 5.29

$M[w]$ ist Abk. für „Maschine M mit Eingabe w“

$M[w]\downarrow$ bedeutet, dass $M[w]$ terminiert/hält.

Definition 5.29

$M[w]$ ist Abk. für „Maschine M mit Eingabe w“

$M[w]\downarrow$ bedeutet, dass $M[w]$ terminiert/hält.

Definition 5.30 (Spezielles Halteproblem)

Gegeben: Ein Wort $w \in \{0,1\}^*$.

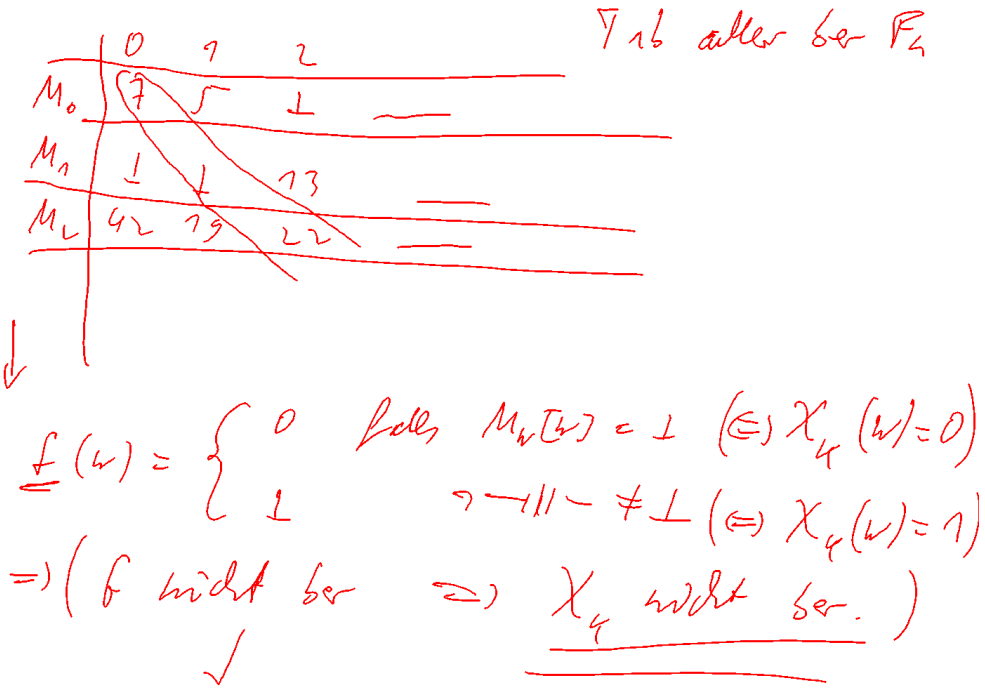
Problem: Hält M_w bei Eingabe w ?

Als Menge:

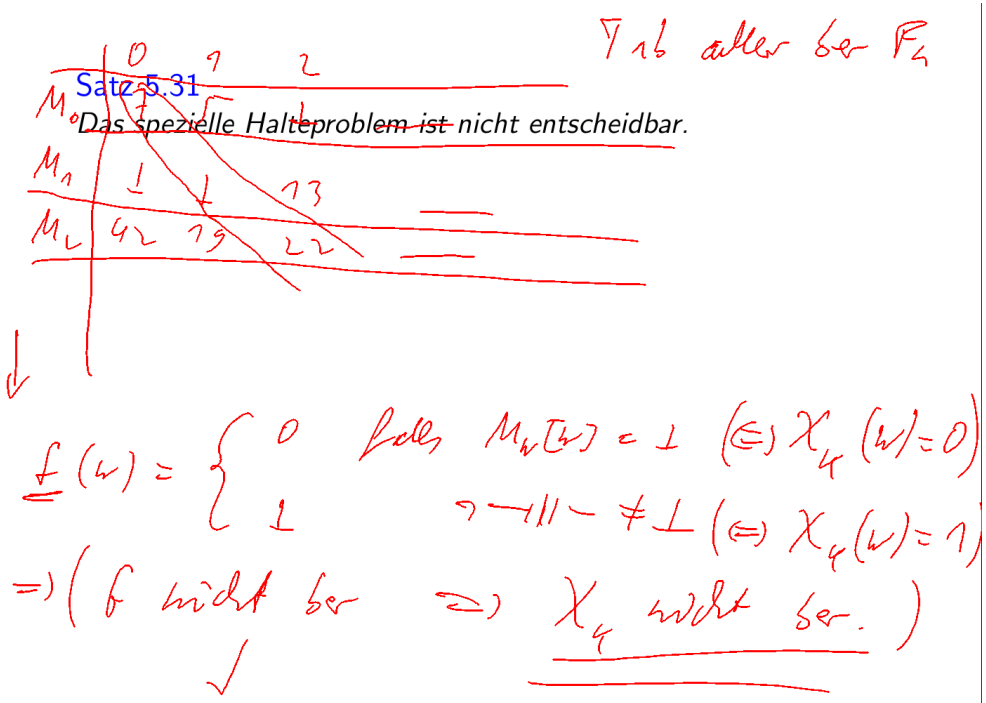
$$K := \{w \in \{0,1\}^* \mid M_w[w]\downarrow\}$$

Menge aller TM, die auf sich selbst angewandt halten

Satz 5.31
Das spezielle Halteproblem ist nicht entscheidbar.



Satz 5.31
Das spezielle Halteproblem ist nicht entscheidbar.

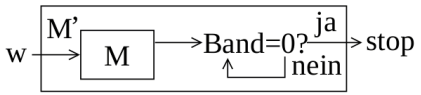


Satz 5.31
Das spezielle Halteproblem ist nicht entscheidbar.

Beweis:
Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.
Dann ist auch folgende Funktion f berechenbar:

$$f(w) := \begin{cases} 0 & \text{falls } \chi_K(w) = 0 \\ \bot & \text{falls } \chi_K(w) = 1 \end{cases}$$

Denn berechnet eine TM M die Funktion χ_K,
so berechnet die folgende TM M' die Funktion f:



Satz 5.31

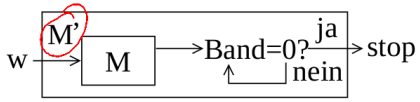
Das spezielle Halteproblem ist nicht entscheidbar.

Beweis:

Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.
Dann ist auch folgende Funktion f berechenbar:

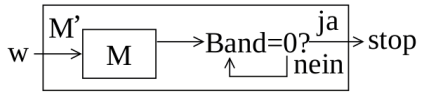
$$f(w) := \begin{cases} 0 & \text{falls } \chi_K(w) = 0 \\ \perp & \text{falls } \chi_K(w) = 1 \end{cases}$$

Denn berechnet eine TM M die Funktion χ_K,
so berechnet die folgende TM M' die Funktion f:

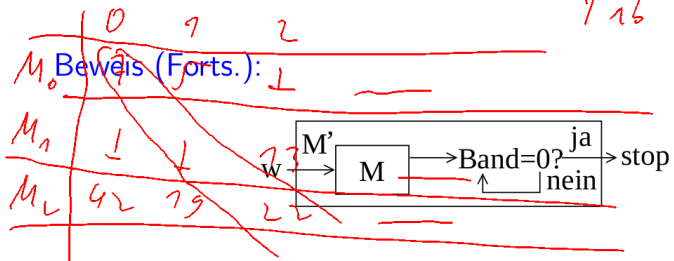


Dann gibt es ein w' mit $M_{w'} = M'$.

Beweis (Forts.):



Beweis (Forts.):



$$\begin{aligned} f(w') = \perp &\Leftrightarrow \chi_K(w') = 1 && \text{(Def. von } f\text{)} \\ &\Leftrightarrow w' \in K && \text{(Def. von } \chi_K\text{)} \\ &\Leftrightarrow M_{w'}[w']\downarrow && \text{(Def. von } K\text{)} \\ &\Leftrightarrow M'[w']\downarrow && (M_{w'} = M') \\ &\Leftrightarrow f(w') \neq \perp && (M' \text{ berechnet } f) \end{aligned}$$

Definition 5.32 ((Allgemeines) Halteproblem)

Gegeben: Wörter $w, x \in \{0,1\}^*$.

Problem: Hält M_w bei Eingabe x ?

Als Menge:

$$H := \{w\#x \mid M_w[x]\downarrow\}$$

Definition 5.32 ((Allgemeines) Halteproblem)

Gegeben: Wörter $w, x \in \{0,1\}^*$.

Problem: Hält M_w bei Eingabe x ?

Als Menge:

$$H := \{w\#x \mid M_w[x]\downarrow\}$$

Satz 5.33

Das Halteproblem H ist nicht entscheidbar.

Definition 5.34 (Reduktion)

Eine Menge $A \subseteq \Sigma^*$ ist reduzierbar auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt mit

$$\forall w \in \Sigma^*.\ w \in A \Leftrightarrow f(w) \in B$$

Definition 5.34 (Reduktion)

Eine Menge $A \subseteq \Sigma^*$ ist reduzierbar auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt mit

$$\forall w \in \Sigma^*.\ w \in A \Leftrightarrow f(w) \in B$$

Wir schreiben dann $A \leq B$.

Definition 5.34 (Reduktion)

Eine Menge $A \subseteq \Sigma^*$ ist reduzierbar auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt mit

$$\forall w \in \Sigma^*.\ w \in A \Leftrightarrow f(w) \in B$$

Wir schreiben dann $A \leq B$.

Intuition:

- B ist mindestens so schwer zu lösen wie A.

Definition 5.34 (Reduktion)

Eine Menge $A \subseteq \Sigma^*$ ist reduzierbar auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt mit

$$\forall w \in \Sigma^*.\ w \in A \Leftrightarrow f(w) \in B$$

Wir schreiben dann $A \leq B$.

Intuition:

- B ist mindestens so schwer zu lösen wie A.
- Ist A unlösbar, dann auch B.

Definition 5.34 (Reduktion)

Eine Menge $A \subseteq \Sigma^*$ ist reduzierbar auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt mit

$$\forall w \in \Sigma^*.\ w \in A \Leftrightarrow f(w) \in B$$

Wir schreiben dann $A \leq B$.

Intuition:

- B ist mindestens so schwer zu lösen wie A.
- Ist A unlösbar, dann auch B.
- Ist B lösbar, dann erst recht A.

273

Lemma 5.35

Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Lemma 5.35

Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:

Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.

274

Lemma 5.35
Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:
Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.
Dann ist $\chi_B \circ f$ berechenbar und $\chi_A = \chi_B \circ f$:

Lemma 5.35
Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:
Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.
Dann ist $\chi_B \circ f$ berechenbar und $\chi_A = \chi_B \circ f$:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x)) \qquad \square$$

$w \in A \iff f(w) \in B$

Lemma 5.35
Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:
Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.
Dann ist $\chi_B \circ f$ berechenbar und $\chi_A = \chi_B \circ f$:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x)) \qquad \square$$

Korollar 5.36
Falls $A \leq B$ und A ist unentscheidbar, dann ist auch B unentscheidbar.

Lemma 5.35
Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:
Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.
Dann ist $\chi_B \circ f$ berechenbar und $\chi_A = \chi_B \circ f$:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x)) \qquad \square$$

Korollar 5.36
Falls $A \leq B$ und A ist unentscheidbar, dann ist auch B unentscheidbar.

Beispiel 5.37
Da $K \leq H$ (mit Reduktion $f(w) := w\#w$) und K unentscheidbar ist, ist auch H unentscheidbar.

Satz 5.38

Das Halteproblem auf leerem Band, H_0, ist unentscheidbar.

$$H_0 := \{w \in \{0,1\}^* \mid M_w[\epsilon]\downarrow\}$$

Satz 5.38

Das Halteproblem auf leerem Band, H_0, ist unentscheidbar.

$$H_0 := \{w \in \{0,1\}^* \mid M_w[\epsilon]\downarrow\}$$

Beweis:
Wir zeigen $K \leq H_0$ mit einer Funktion $f : \{0,1\}^* \to \{0,1\}^*$

Satz 5.38

Das Halteproblem auf leerem Band, H_0, ist unentscheidbar.

$$H_0 := \{w \in \{0,1\}^* \mid M_w[\epsilon]\downarrow\}$$

Beweis:
Wir zeigen $K \leq H_0$ mit einer Funktion $f : \{0,1\}^* \to \{0,1\}^*$

$f(w)$ ist die Kodierung folgender TM:

> Überschreibe die Eingabe mit w; führe M_w aus.

Satz 5.38

Das Halteproblem auf leerem Band, H_0, ist unentscheidbar.

$$H_0 := \{w \in \{0,1\}^* \mid M_w[\epsilon]\downarrow\}$$

Beweis:
Wir zeigen $K \leq H_0$ mit einer Funktion $f : \{0,1\}^* \to \{0,1\}^*$
$f(w)$ ist die Kodierung folgender TM:

> Überschreibe die Eingabe mit w; führe M_w aus.

Dh f berechnet aus w die Kodierung w_1 einer TM, die w schreibt, und gibt die Kodierung von "$w_1; w$" zurück.

Satz 5.38

Das Halteproblem auf leerem Band, H_0, ist unentscheidbar.

$$H_0 := \{w \in \{0,1\}^* \mid M_w[\epsilon]\downarrow\}$$

Beweis:

Wir zeigen $K \leq H_0$ mit einer Funktion $f : \{0,1\}^* \to \{0,1\}^*$
$f(w)$ ist die Kodierung folgender TM:

Überschreibe die Eingabe mit w; führe M_w aus.

Dh f berechnet aus w die Kodierung w_1 einer TM, die w schreibt, und gibt die Kodierung von "$w_1; w$" zurück.
Damit ist f total und berechenbar.

Satz 5.38

Das Halteproblem auf leerem Band, H_0, ist unentscheidbar.

$$H_0 := \{w \in \{0,1\}^* \mid M_w[\epsilon]\downarrow\}$$

Beweis:

Wir zeigen $K \leq H_0$ mit einer Funktion $f : \{0,1\}^* \to \{0,1\}^*$
$f(w)$ ist die Kodierung folgender TM:

Überschreibe die Eingabe mit w; führe M_w aus.

Dh f berechnet aus w die Kodierung w_1 einer TM, die w schreibt, und gibt die Kodierung von "$w_1; w$" zurück.
Damit ist f total und berechenbar.
Es gilt:

$$w \in K \Leftrightarrow M_w[w]\downarrow \Leftrightarrow M_{f(w)}[\epsilon]\downarrow \Leftrightarrow f(w) \in H_0$$

□

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

Fazit:
Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?

Fazit:
Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
  Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.

Fazit:
Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
  Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.
- Kann Variable x_7 bei einer bestimmten Eingabe je den Wert 2^{32} erreichen?

Fazit:
Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
  Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.
- Kann Variable x_7 bei einer bestimmten Eingabe je den Wert 2^{32} erreichen?
  Reduktion: Ein Programm P hält gdw

Quiz

Ist es entscheidbar, ob eine TM

1. mehr als 314 Zustände hat?