SS 2005

Einführung in die Informatik IV

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2005SS/info4/index.html.de

6. Juni 2005

1.1 Turing-Berechenbarkeit

Definition 109

Eine (partielle) Funktion

$$f : \mathbb{N}_0^k \supseteq D \to \mathbb{N}_0$$

heißt Turing-berechenbar, falls es eine deterministische Turingmaschine gibt, die für jede Eingabe $(n_1, \ldots, n_k) \in D$ nach endlich vielen Schritten mit dem Bandinhalt

$$f(n_1, \ldots, n_k) \in \mathbb{N}_0$$

hält. Falls $(n_1, \ldots, n_k) \notin D$, hält die Turingmaschine nicht!

Dabei nehmen wir an, dass Tupel wie $(n_1, \ldots, n_k)$ geeignet codiert auf dem Band der Turingmaschine dargestellt werden.

1.1 Turing-Berechenbarkeit

Definition 109

Eine (partielle) Funktion

$$f : \mathbb{N}_0^k \supseteq D \to \mathbb{N}_0$$

heißt Turing-berechenbar, falls es eine deterministische Turingmaschine gibt, die für jede Eingabe $(n_1, \ldots, n_k) \in D$ nach endlich vielen Schritten mit dem Bandinhalt

$$f(n_1, \ldots, n_k) \in \mathbb{N}_0$$

hält. Falls $(n_1, \ldots, n_k) \notin D$, hält die Turingmaschine nicht!

Dabei nehmen wir an, dass Tupel wie $(n_1, \ldots, n_k)$ geeignet codiert auf dem Band der Turingmaschine dargestellt werden.

Eine beliebte Modellvariante ist die k-Band-Turingmaschine, die statt einem Band k, $k \geq 1$, Arbeitsbänder zur Verfügung hat, deren Lese-/Schreibköpfe sie unabhängig voneinander bewegen kann.

Oft existiert auch ein spezielles Eingabeband, das nur gelesen, aber nicht geschrieben werden kann (read-only). Der Lesekopf kann jedoch normalerweise in beiden Richtungen bewegt werden.

Ebenso wird oft ein spezielles Ausgabeband verwendet, das nur geschrieben, aber nicht gelesen werden kann (write-only). Der Schreibkopf kann dabei nur nach rechts bewegt werden.

Eine beliebte Modellvariante ist die k-Band-Turingmaschine, die statt einem Band k, $k \geq 1$, Arbeitsbänder zur Verfügung hat, deren Lese-/Schreibköpfe sie unabhängig voneinander bewegen kann.

Oft existiert auch ein spezielles Eingabeband, das nur gelesen, aber nicht geschrieben werden kann (read-only). Der Lesekopf kann jedoch normalerweise in beiden Richtungen bewegt werden.

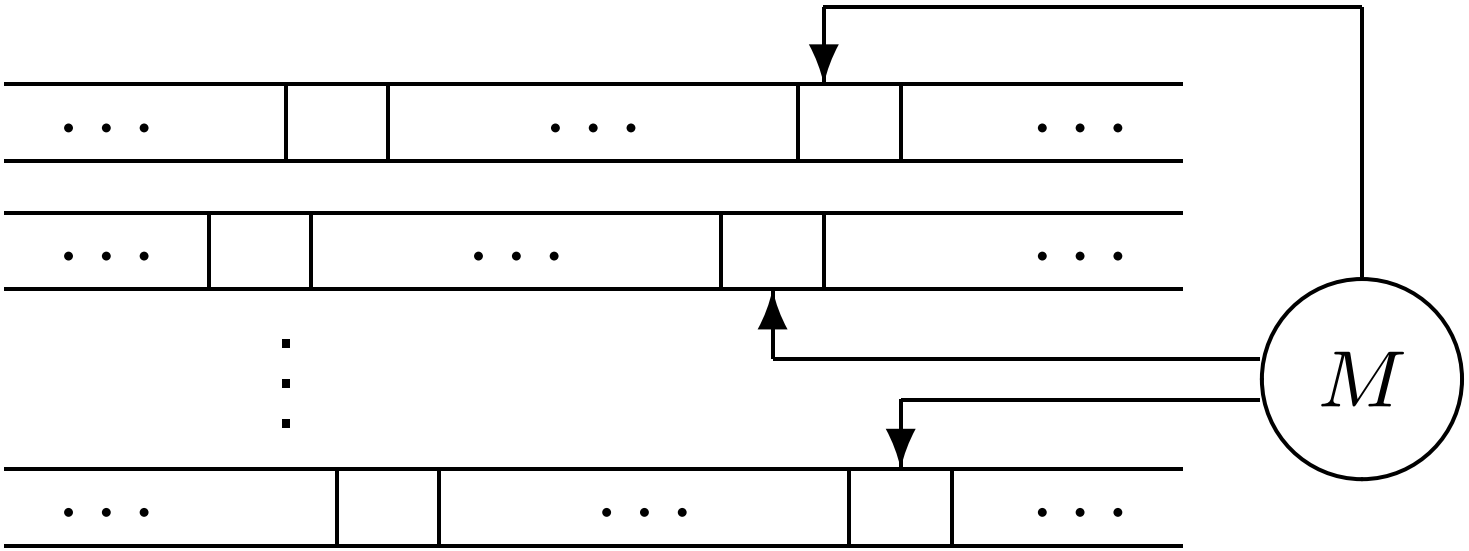
Ebenso wird oft ein spezielles Ausgabeband verwendet, das nur geschrieben, aber nicht gelesen werden kann (write-only). Der Schreibkopf kann dabei nur nach rechts bewegt werden.

Eine beliebte Modellvariante ist die k-Band-Turingmaschine, die statt einem Band k, $k \geq 1$, Arbeitsbänder zur Verfügung hat, deren Lese-/Schreibköpfe sie unabhängig voneinander bewegen kann.

Oft existiert auch ein spezielles Eingabeband, das nur gelesen, aber nicht geschrieben werden kann (read-only). Der Lesekopf kann jedoch normalerweise in beiden Richtungen bewegt werden.

Ebenso wird oft ein spezielles Ausgabeband verwendet, das nur geschrieben, aber nicht gelesen werden kann (write-only). Der Schreibkopf kann dabei nur nach rechts bewegt werden.

Beispiel 110 (k-Band-Turingmaschine)



Satz 111

Jede k-Band-Turingmaschine kann effektiv durch eine 1-Band-TM simuliert werden.

Beweis:

Wir simulieren die k Bänder auf k Spuren eines Bandes, wobei wir das Teilalphabet für jede Spur auch noch so erweitern, dass die Kopfposition des simulierten Bandes mit gespeichert werden kann. □

Satz 111

Jede k-Band-Turingmaschine kann effektiv durch eine 1-Band-TM simuliert werden.

Beweis:

Wir simulieren die k Bänder auf k Spuren eines Bandes, wobei wir das Teilalphabet für jede Spur auch noch so erweitern, dass die Kopfposition des simulierten Bandes mit gespeichert werden kann. □

Definition 112

Eine Sprache $A \subseteq \Sigma^*$ heißt rekursiv oder entscheidbar, falls es eine deterministische TM M gibt, die auf allen Eingaben $\in \Sigma^*$ hält und A erkennt.

Definition 113

Eine Sprache $A \subseteq \Sigma^*$ heißt rekursiv aufzählbar (r.a.) oder semi-entscheidbar, falls es eine TM N gibt, für die

$$L(N) = A\,,$$

also, falls A Chomsky-0 ist.

Definition 112

Eine Sprache $A \subseteq \Sigma^*$ heißt rekursiv oder entscheidbar, falls es eine deterministische TM M gibt, die auf allen Eingaben $\in \Sigma^*$ hält und A erkennt.

Definition 113

Eine Sprache $A \subseteq \Sigma^*$ heißt rekursiv aufzählbar (r.a.) oder semi-entscheidbar, falls es eine TM N gibt, für die

$$L(N) = A\,,$$

also, falls A Chomsky-0 ist.

Definition 114

Sei $A \subseteq \Sigma^*$. Die charakteristische Funktion χ_A von A ist

$$\chi_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ 0 & \text{sonst} \end{cases}$$

Definition 115

Sei $A \subseteq \Sigma^*$. χ'_A ist definiert durch

$$\chi'_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Definition 114

Sei $A \subseteq \Sigma^*$. Die charakteristische Funktion χ_A von A ist

$$\chi_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ 0 & \text{sonst} \end{cases}$$

Definition 115

Sei $A \subseteq \Sigma^*$. χ'_A ist definiert durch

$$\chi'_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Satz 116

A ist rekursiv $\Leftrightarrow$ χ_A ist berechenbar.

Beweis:

Folgt aus der Definition von rekursiv: es gibt eine TM, die `ja` oder `nein` liefert. Wandle das Ergebnis in 1 oder 0. □

Satz 116

A ist rekursiv $\Leftrightarrow$ χ_A ist berechenbar.

Beweis:

Folgt aus der Definition von rekursiv: es gibt eine TM, die `ja` oder `nein` liefert. Wandle das Ergebnis in 1 oder 0. □

Satz 117

A ist rekursiv aufzählbar $\Leftrightarrow$ χ'_A ist berechenbar.

Beweis:

Folgt unmittelbar aus der Definition. □

Satz 118

A ist rekursiv $\Leftrightarrow$ χ'_A und $\chi'_{\bar{A}}$ sind berechenbar ($\Leftrightarrow$ A und $\bar{A}$ sind r.a.)

Beweis:

Nur $\Leftarrow$ ist nichttrivial. Wir lassen hier eine TM für A und eine TM für $\bar{A}$ Schritt für Schritt parallel laufen. □

Satz 117

A ist rekursiv aufzählbar $\Leftrightarrow$ χ'_A ist berechenbar.

Beweis:

Folgt unmittelbar aus der Definition. □

Satz 118

A ist rekursiv $\Leftrightarrow$ χ'_A und $\chi'_{\bar{A}}$ sind berechenbar ($\Leftrightarrow$ A und $\bar{A}$ sind r.a.)

Beweis:

Nur $\Leftarrow$ ist nichttrivial. Wir lassen hier eine TM für A und eine TM für $\bar{A}$ Schritt für Schritt parallel laufen. □

Satz 117

A ist rekursiv aufzählbar $\Leftrightarrow$ χ'_A ist berechenbar.

Beweis:

Folgt unmittelbar aus der Definition. □

Satz 118

A ist rekursiv $\Leftrightarrow$ χ'_A und $\chi'_{\bar{A}}$ sind berechenbar ($\Leftrightarrow$ A und $\bar{A}$ sind r.a.)

Beweis:

Nur $\Leftarrow$ ist nichttrivial. Wir lassen hier eine TM für A und eine TM für $\bar{A}$ Schritt für Schritt parallel laufen. □

Satz 117

A ist rekursiv aufzählbar $\Leftrightarrow$ χ'_A ist berechenbar.

Beweis:

Folgt unmittelbar aus der Definition. □

Satz 118

A ist rekursiv $\Leftrightarrow$ χ'_A und $\chi'_{\bar{A}}$ sind berechenbar ($\Leftrightarrow$ A und $\bar{A}$ sind r.a.)

Beweis:

Nur $\Leftarrow$ ist nichttrivial. Wir lassen hier eine TM für A und eine TM für $\bar{A}$ Schritt für Schritt parallel laufen. □

1.2 WHILE-Berechenbarkeit

WHILE-Programme sind wie folgt definiert:

Variablen: $x_1, x_2, x_3, \ldots$

Konstanten: 0, 1, 2, ...

Trennsymbole: ; $:=$ $\neq$

Operatoren: $+$ -

Schlüsselwörter: WHILE DO END

Der Aufbau von WHILE-Programmen:

- $x_i := c$, $x_i := x_j + c$, $x_i := x_j - c$ sind WHILE-Programme
  Die Interpretation dieser Ausdrücke erfolgt, wie üblich, mit der Einschränkung, dass $x_j - c$ als 0 gewertet wird, falls $c > x_j$.
- Sind P_1 und P_2 WHILE-Programme, so ist auch

  $$P_1; P_2$$

  ein WHILE-Programm.
  Interpretation: Führe zuerst P_1 und dann P_2 aus.
- Ist P ein WHILE-Programm, so ist auch

  $$\text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}$$

  ein WHILE-Programm.
  Interpretation: Führe P solange aus, bis x_i den Wert 0 hat.
  Achtung: Zuweisungen an x_i im Innern von P beeinflussen dabei den Wert von x_i!

Der Aufbau von WHILE-Programmen:

- $x_i := c$, $x_i := x_j + c$, $x_i := x_j - c$ sind WHILE-Programme
  Die Interpretation dieser Ausdrücke erfolgt, wie üblich, mit der Einschränkung, dass $x_j - c$ als 0 gewertet wird, falls $c > x_j$.
- Sind P_1 und P_2 WHILE-Programme, so ist auch

  $$P_1; P_2$$

  ein WHILE-Programm.
  Interpretation: Führe zuerst P_1 und dann P_2 aus.
- Ist P ein WHILE-Programm, so ist auch

  $$\text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}$$

  ein WHILE-Programm.
  Interpretation: Führe P solange aus, bis x_i den Wert 0 hat.
  Achtung: Zuweisungen an x_i im Innern von P beeinflussen dabei den Wert von x_i!

Der Aufbau von WHILE-Programmen:

- $x_i := c$, $x_i := x_j + c$, $x_i := x_j - c$ sind WHILE-Programme
  Die Interpretation dieser Ausdrücke erfolgt, wie üblich, mit der Einschränkung, dass $x_j - c$ als 0 gewertet wird, falls $c > x_j$.
- Sind P_1 und P_2 WHILE-Programme, so ist auch

  $$P_1; P_2$$

  ein WHILE-Programm.
  Interpretation: Führe zuerst P_1 und dann P_2 aus.
- Ist P ein WHILE-Programm, so ist auch

  $$\text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}$$

  ein WHILE-Programm.
  Interpretation: Führe P solange aus, bis x_i den Wert 0 hat.
  Achtung: Zuweisungen an x_i im Innern von P beeinflussen dabei den Wert von x_i!

Der Aufbau von WHILE-Programmen:

- $x_i := c$, $x_i := x_j + c$, $x_i := x_j - c$ sind WHILE-Programme
  Die Interpretation dieser Ausdrücke erfolgt, wie üblich, mit der Einschränkung, dass $x_j - c$ als 0 gewertet wird, falls $c > x_j$.
- Sind P_1 und P_2 WHILE-Programme, so ist auch

  $$P_1; P_2$$

  ein WHILE-Programm.
  Interpretation: Führe zuerst P_1 und dann P_2 aus.
- Ist P ein WHILE-Programm, so ist auch

  $$\text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}$$

  ein WHILE-Programm.
  Interpretation: Führe P solange aus, bis x_i den Wert 0 hat.
  Achtung: Zuweisungen an x_i im Innern von P beeinflussen dabei den Wert von x_i!

Satz 119

Ist eine Funktion WHILE-berechenbar, so ist sie auch Turing-berechenbar.

Beweis:
Die Turingmaschine merkt sich den Programmzähler des WHILE-Programms sowie die aktuellen Werte aller Variablen. Der Platz dafür muss notfalls immer wieder angepasst werden. □

Wir werden später auch die umgekehrte Aussage des obigen Satzes zeigen.

Satz 119

Ist eine Funktion WHILE-berechenbar, so ist sie auch Turing-berechenbar.

Beweis:

Die Turingmaschine merkt sich den Programmzähler des WHILE-Programms sowie die aktuellen Werte aller Variablen. Der Platz dafür muss notfalls immer wieder angepasst werden. □

Wir werden später auch die umgekehrte Aussage des obigen Satzes zeigen.

Satz 119

Ist eine Funktion WHILE-berechenbar, so ist sie auch Turing-berechenbar.

Beweis:

Die Turingmaschine merkt sich den Programmzähler des WHILE-Programms sowie die aktuellen Werte aller Variablen. Der Platz dafür muss notfalls immer wieder angepasst werden. □

Wir werden später auch die umgekehrte Aussage des obigen Satzes zeigen.

1.3 GOTO-Berechenbarkeit

GOTO-Programme sind wie folgt definiert:

Variablen: $x_1, x_2, x_3, \ldots$

Konstanten: 0, 1, 2, ...

Trennsymbole: ; :=

Operatoren: + - =

Schlüsselwörter: IF THEN GOTO HALT

Ein GOTO-Programm ist eine Folge von markierten Anweisungen

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

wobei die A_i Anweisungen und die M_i Zielmarken für Sprünge sind.
Als Anweisungen können auftreten:

- Wertzuweisung: $x_i := x_j \pm c$
- unbedingter Sprung: GOTO M_i
- bedingter Sprung: IF $x_j = c$ THEN GOTO M_i
- Stopanweisung: HALT

Dabei ist c jeweils eine (beliebige) Konstante.

Ein GOTO-Programm ist eine Folge von markierten Anweisungen

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

wobei die A_i Anweisungen und die M_i Zielmarken für Sprünge sind.
Als Anweisungen können auftreten:

- Wertzuweisung: $x_i := x_j \pm c$
- unbedingter Sprung: GOTO M_i
- bedingter Sprung: IF $x_j = c$ THEN GOTO M_i
- Stopanweisung: HALT

Dabei ist c jeweils eine (beliebige) Konstante.

Ein GOTO-Programm ist eine Folge von markierten Anweisungen

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

wobei die A_i Anweisungen und die M_i Zielmarken für Sprünge sind.
Als Anweisungen können auftreten:

- Wertzuweisung: $x_i := x_j \pm c$
- unbedingter Sprung: GOTO M_i
- bedingter Sprung: IF $x_j = c$ THEN GOTO M_i
- Stopanweisung: HALT

Dabei ist c jeweils eine (beliebige) Konstante.

Ein GOTO-Programm ist eine Folge von markierten Anweisungen

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

wobei die A_i Anweisungen und die M_i Zielmarken für Sprünge sind.
Als Anweisungen können auftreten:

- Wertzuweisung: $x_i := x_j \pm c$
- unbedingter Sprung: GOTO M_i
- bedingter Sprung: IF $x_j = c$ THEN GOTO M_i
- Stopanweisung: HALT

Dabei ist c jeweils eine (beliebige) Konstante.

Ein GOTO-Programm ist eine Folge von markierten Anweisungen

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

wobei die A_i Anweisungen und die M_i Zielmarken für Sprünge sind.
Als Anweisungen können auftreten:

- Wertzuweisung: $x_i := x_j \pm c$
- unbedingter Sprung: GOTO M_i
- bedingter Sprung: IF $x_j = c$ THEN GOTO M_i
- Stopanweisung: HALT

Dabei ist c jeweils eine (beliebige) Konstante.

Satz 120

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

Beweis:

Ersetze jede WHILE-Schleife WHILE $x_i \neq 0$ DO P END durch folgendes Konstrukt:

$$
\begin{array}{ll}
M_1\text{:} & \text{IF } x_i = 0 \text{ THEN GOTO } M_2 \\
 & P\text{;} \\
 & \text{GOTO } M_1 \\
M_2\text{:} & \ldots
\end{array}
$$

□

Satz 120

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

Beweis:

Ersetze jede WHILE-Schleife WHILE $x_i \neq 0$ DO P END durch folgendes Konstrukt:

$$
\begin{array}{ll}
M_1\text{:} & \text{IF } x_i = 0 \text{ THEN GOTO } M_2 \\
 & \quad P\text{;} \\
 & \text{GOTO } M_1 \\
M_2\text{:} & \ldots
\end{array}
$$

□

Satz 121

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

Beweis:

Gegeben sei das GOTO-Programm

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

Wir simulieren dies durch ein WHILE-Programm mit genau einer WHILE-Schleife:

$c := 1;$
WHILE $c \neq 0$ DO
  IF $c = 1$ THEN A'_1 END;
  IF $c = 2$ THEN A'_2 END;
  $\vdots$
  IF $c = k$ THEN A'_k END;
END

Beweis:

wobei

$$A'_i := \begin{cases} x_j := x_l \pm b; c := c+1 & \text{falls } A_i = x_j := x_l \pm b \\ c := \ell & \text{falls } A_i = \text{GOTO } M_\ell \\ \text{IF } x_j = b \text{ THEN } c := \ell & \text{falls } A_i = \text{IF } x_j = b \text{ THEN} \\ \text{ELSE } c := c+1 \text{ END} & \qquad\qquad \text{GOTO } M_\ell \\ c := 0 & \text{falls } A_i = \text{HALT} \end{cases}$$

Es bleibt als Übungsaufgabe überlassen, die IF-Anweisungen ebenfalls durch WHILE-Schleifen zu ersetzen. □

Satz 122

Aus Turing-Berechenbarkeit folgt GOTO-Berechenbarkeit.

Beweis:

□

Satz 122

Aus Turing-Berechenbarkeit folgt GOTO-Berechenbarkeit.

Beweis:

Die Konfiguration (α, q, β) einer (det.) 1-Band-TM wird in den Variablen x_l, x_Q, x_r codiert. Die Zeichenreihen α und β werden als Zahlen (zu einer geeigneten Basis) aufgefasst, mit der niedrigstwertigen Ziffer bei der Position des Lese-/Schreibkopfes.

Satz 122

Aus Turing-Berechenbarkeit folgt GOTO-Berechenbarkeit.

Beweis:

Die Konfiguration (α, q, β) einer (det.) 1-Band-TM wird in den Variablen x_l, x_Q, x_r codiert. Die Zeichenreihen α und β werden als Zahlen (zu einer geeigneten Basis) aufgefasst, mit der niedrigstwertigen Ziffer bei der Position des Lese-/Schreibkopfes.

Jedes Tupel der Übergangsfunktion der TM wird durch ein geeignetes kleines Programmstück simuliert. Dabei werden Operationen wie Multiplikation mit, Division durch und Modulorechnung zur Basis benötigt. □