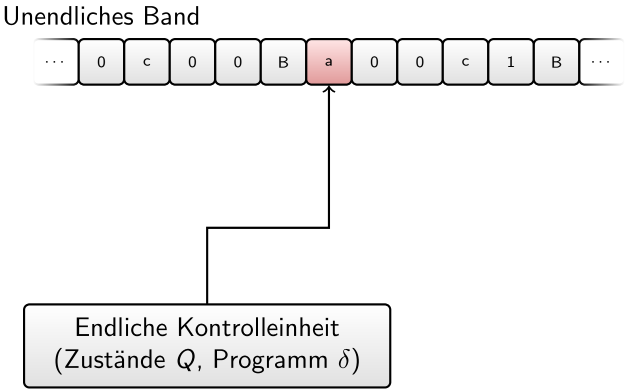
Mächtigkeit von WHILE-Programmen

Prof. Dr. Berthold Vöcking
Lehrstuhl Informatik 1
Algorithmen und Komplexität
RWTH Aachen

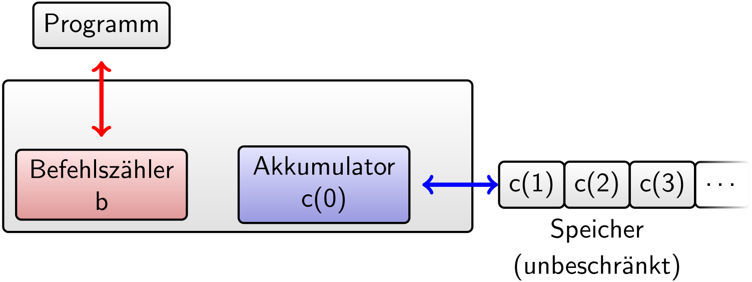
November 2011

Turingmaschine (TM)

$$M = (Q, \Sigma, \Gamma, B, q_0, \bar{q}, \delta)$$



Registermaschine (RAM)



Befehlssatz:
LOAD, STORE, ADD, SUB, MULT, DIV,
INDLOAD, INDSTORE, INDADD, INDSUB, INDMULT, INDDIV,
CLOAD, CADD, CSUB, CMULT, CDIV,
GOTO,
IF c(0) ? x THEN GOTO (wobei ? aus $\{=, <, \leq\}$),
END

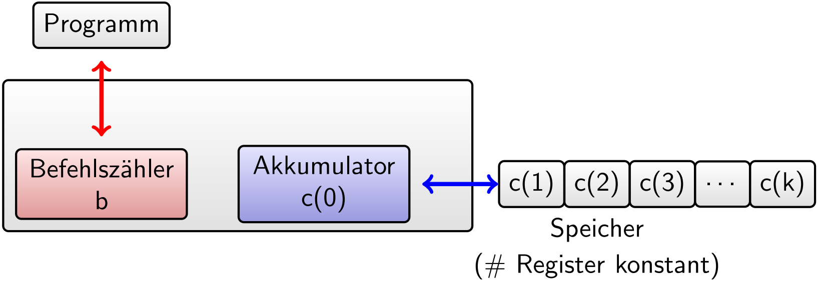
Turing-mächtige Rechnermodelle

Definition

Ein Rechnermodell wird als *Turing-mächtig* bezeichnet, wenn jede Funktion, die durch eine TM berechnet werden kann, auch durch dieses Rechnermodell berechnet werden kann.

Da die Registermaschine die Turingmaschine simulieren kann, ist sie Turing-mächtig.

Eingeschränkte Registermaschine (eingeschränkte RAM)



Befehlssatz:
LOAD, STORE
CLOAD, CADD, CSUB
GOTO,
IF $c(0) \neq 0$ THEN GOTO,
END

Übungsaufgabe: Zeige, dass die eingeschränkte RAM Turing-mächtig ist.

Turing-mächtige Programmiersprachen

Definition

Ein Programmiersprache wird als *Turing-mächtig* bezeichnet, wenn jede Funktion, die durch eine TM berechnet werden kann, auch durch ein Programm in dieser Programmiersprache berechnet werden kann.

Welche Elemente benötigt eine Programmiersprache, um Turing-mächtig zu sein.

Die Programmiersprache WHILE – Syntax

Elemente eines WHILE-Programms

- Variablen $x_0 \quad x_1 \quad x_2 \ldots$
- Konstanten $-1 \quad 0 \quad 1$
- Symbole $; \quad := \quad + \quad \neq$
- Schlüsselwörter WHILE DO END

Induktive Definition – Induktionsanfang

Zuweisung

Für jedes $c \in \{-1, 0, 1\}$ ist die Zuweisung

$$x_i \ := \ x_j + c$$

ein WHILE-Programm.

Induktive Definition – Induktionsschritte:

Hintereinanderausführung

Falls P_1 und P_2 WHILE-Programme sind, dann ist auch

$$P_1; P_2$$

ein WHILE-Programm.

WHILE-Konstrukt

Falls P ein WHILE-Programm ist, dann ist auch

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

ein WHILE-Programm.

Die Programmiersprache WHILE – Semantik

Ein While-Programm P berechnet eine k-stellige Funktionen der Form $f : \mathbb{N}^k \to \mathbb{N}$.

- Die Eingabe ist in den Variablen $x_1, \ldots, x_k$ enthalten.
- Alle anderen Variablen werden mit 0 initialisiert.
- Das Resultat eines WHILE-Programms ist die Zahl, die sich am Ende der Rechnung in der Variable x_0 ergibt.

- Programme der Form $x_i := x_j + c$ sind Zuweisungen des Wertes $x_j + c$ an die Variable x_i (wobei $0 - 1 = 0$).
- In einem WHILE-Programm P_1; P_2 wird zunächst P_1 und dann P_2 ausgeführt.
- Das Programm WHILE $x_i \neq 0$ DO P END hat die Bedeutung, dass P solange ausgeführt wird, bis x_i den Wert 0 erreicht.

Beispiel eines WHILE-Programms

Was berechnet dieses WHILE-Programm?

WHILE $x_2 \neq 0$ DO

$\quad x_1 := x_1 + 1;$

$\quad x_2 := x_2 - 1$

END;

$x_0 := x_1$

Satz

Die Programmiersprache WHILE ist Turing-mächtig.

Beweis:

Wir zeigen, dass jede Funktion, die durch eine eingeschränkte RAM berechnet werden kann, auch durch ein WHILE-Programm berechnet werden kann.

Da die eingeschränkte RAM Turing-mächtig ist, ist somit auch die Programmiersprache WHILE Turing-mächtig.

Sei Π ein beliebiges Programm der eingeschränkten RAM. Sei ℓ die Anzahl der Zeilen in Π und k die Anzahl der verwendeten Register.

Beweis Turing-Mächtigkeit von WHILE-Programmen

Wir speichern den Inhalt von Register $c(i)$, für $0 \leq i \leq k$, in der Variable x_i des WHILE-Programms.

In der Variable x_{k+1} speichern wir zudem den Befehlszähler b der RAM ab.

Die Variable x_{k+2} verwenden wir, um eine Variable zu haben, die immer den initial gesetzen Wert 0 enthält.

Die einzelnen RAM-Befehle werden nun in Form von konstant vielen Zuweisungen der Form $x_i := x_j + c$ mit $c \in \{0, 1\}$ implementiert.

Beweis Turing-Mächtigkeit von WHILE-Programmen

RAM vs. **WHILE**

RAM

- LOAD, STORE
- CLOAD, CADD, CSUB, GOTO
- IF c(0) $\neq$ 0 GOTO
- END

WHILE

- $x_i \;:=\; x_j + c$ für $c \in \{-1, 0, 1\}$
- $P_1; P_2$
- WHILE $x_i \neq 0$ DO P END

Der RAM-Befehl LOAD i wird beispielsweise ersetzt durch

$$x_0 := x_i + 0;\;\; x_{k+1} := x_{k+1} + 1$$

Beweis Turing-Mächtigkeit von WHILE-Programmen

RAM vs. **WHILE**

RAM

- LOAD, STORE ✓
- CLOAD, CADD, CSUB, GOTO
- IF c(0) $\neq$ 0 GOTO
- END

WHILE

- $x_i \ := \ x_j + c$ für $c \in \{-1, 0, 1\}$
- $P_1; P_2$
- WHILE $x_i \neq 0$ DO P END

Der RAM-Befehl CLOAD i wird ersetzt durch

$$x_0 := x_{k+2} + 0; \ \underbrace{x_0 := x_0 + 1; \ \ldots; \ x_0 := x_0 + 1;}_{i \text{ mal}} \ x_{k+1} := x_{k+1} + 1$$

Beweis Turing-Mächtigkeit von WHILE-Programmen

RAM vs. **WHILE**

RAM

- LOAD, STORE ✓
- CLOAD, CADD, CSUB, GOTO ✓
- IF c(0) $\neq$ 0 GOTO
- END

WHILE

- $x_i \;:=\; x_j + c$ für $c \in \{-1, 0, 1\}$
- $P_1; P_2$
- WHILE $x_i \neq 0$ DO P END

Den RAM-Befehl IF $c(0) \neq 0$ GOTO j ersetzen wir durch das WHILE-Programm:

$$
\begin{array}{ll}
x_{k+1} := x_{k+1} + 1; & (b := b + 1) \\
x_{k+3} := x_0 + 0; & (\mathit{help} := c(0)) \\
\text{WHILE } x_{k+3} \neq 0 \text{ DO} & (\text{while } \mathit{help} \neq 0) \\
\quad x_{k+1} := x_{k+2} + 0; \underbrace{x_{k+1} := x_{k+1} + 1; \cdots + 1;}_{j \text{ mal}} & (b := j) \\
\quad x_{k+3} := x_{k+2} + 0 & (\mathit{help} := 0) \\
\text{END} & (\text{end of while})
\end{array}
$$

RAM vs. **WHILE**

- LOAD, STORE ✓
- CLOAD, CADD, CSUB, GOTO ✓
- IF c(0) $\neq$ 0 GOTO ✓
- END

- $x_i \ := \ x_j + c$ für $c \in \{-1, 0, 1\}$
- $P_1; P_2$
- WHILE $x_i \neq 0$ DO P END

Den RAM-Befehl END ersetzen wir durch das WHILE-Programm

$$x_{k+1} = 0$$

Jede Zeile des RAM-Programms wird nun wie oben beschrieben in ein WHILE-Programm transformiert.

Das WHILE-Programm für Zeile i bezeichnen wir mit P_i.

Aus P_i konstruieren wir nun ein WHILE-Programm P_i' mit der folgenden Semantik:

Falls $x_{k+1} = i$ dann führe P_i aus.

Übungsaufgabe: Implementiere das WHILE-Programm P_i' mit Unterprogramm P_i.

Beweis Turing-Mächtigkeit von WHILE-Programmen

Nun fügen wir die WHILE-Programme $P'_1, \ldots, P'_\ell$ zu einem WHILE-Programm P zusammen:

$x_{k+1} := 1;$

WHILE $x_{k+1} \neq 0$ DO

$\quad P'_1; \ldots; P'_\ell$

END

P berechnet dieselbe Funktion wie Π. $\square$

Ausblick: Die Programmiersprache LOOP

Syntax

Änderung im Vergleich zu WHILE-Programmen:

Wir ersetzen das WHILE-Konstrukt durch ein LOOP-Konstrukt der folgenden Form:

$$\text{LOOP } x_i \text{ DO } P \text{ END } ,$$

wobei die Variable x_i nicht in P vorkommen darf.

Semantik

Das Programm P wird x_i mal hintereinander ausgeführt.

Frage

Sind LOOP-Programme Turing-mächtig?

Ausblick: Mächtigkeit von LOOP-Programmen

TM $\equiv$ RAM $\equiv$ WHILE $\equiv$ rekursiv

$$\supsetneq$$

LOOP $\equiv$ primitiv rekursiv

$+ \quad - \quad \times \quad \frac{a}{b} \quad x^k \quad \binom{n}{k} \quad \ldots$

Ackermannfkt.

Ausblick: Ackermann-Funktion

Definition

Die Ackermannfunktion $A : \mathbb{N}^2 \to \mathbb{N}$ ist folgendermaßen definert:

$$
\begin{array}{lcll}
A(0, n) & = & n + 1 & \text{für } n \geq 0 \\
A(m+1, 0) & = & A(m, 1) & \text{für } m \geq 0 \\
A(m+1, n+1) & = & A(m, A(m+1, n)) & \text{für } m, n \geq 0
\end{array}
$$

Ausblick: Ackermann-Funktion

Wenn man den ersten Parameter fixiert …

- $A(1, n) = n + 2$,
- $A(2, n) = 2n + 3$,
- $A(3, n) = 8 \cdot 2^n - 3$,
- $A(4, n) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{\substack{n+2 \text{ viele} \\ \text{Potenzen}}} - 3$,

Bereits $A(4, 2) = 2^{65536} - 3$ ist größer als die (vermutete) Anzahl der Atome im Weltraum.