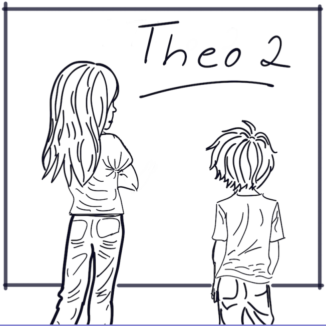
Theoretische Informatik 2

SoSe 2024

Berechenbarkeit und Komplexität

Prof. Dr. Sebastian Siebertz
AG Theoretische Informatik
MZH, Raum 3160
siebertz@uni-bremen.de

Universität Bremen



WHILE-Programme

- WHILE: stark eingeschränkte, modellhafte imperative Sprache.
- Beispiel: Multipliziere $\mathtt{x}_1$ mit $\mathtt{x}_2$, so dass das Ergebnis in $\mathtt{x}_3$ steht.
  - $\mathtt{x}_3 := 0$;
  - **while** $\mathtt{x}_2 \neq 0$ **do**
  - **loop** $\mathtt{x}_1$ **do** $\mathtt{x}_3 := \mathtt{x}_3 + 1$ **end**; Berechne $\mathtt{x}_3 + \mathtt{x}_1$
  - $\mathtt{x}_2 := \mathtt{x}_2 \dot{-} 1$
  - **end**

WHILE-Programme

- Syntax der WHILE-Programme
- Semantik der WHILE-Programme
- WHILE ist Turing-vollständig: Jede Turing-berechenbare Funktion ist WHILE-berechenbar.
- Auf **while**-Schleifen kann nicht verzichtet werden:
  nur mit **loop**-Schleifen erhalten wir die schwächere Sprache LOOP.

Syntax der WHILE-Programme

- Syntax: “Lehre vom Satzbau”
  - Spezifikation der zulässigen Sprachelemente und formale Regeln für den Aufbau des Codes.
- Semantik: Bedeutung der Zeichenfolgen.

Unser Problembegriff

- Menge von *Eingaben* A, Menge von *Ausgaben* B.
- *Problem:* (Partielle) Zuordnung der Eingaben zu den Ausgaben, d.h. eine (partielle) Funktion $P\colon A \to B$.
- Ein Programm soll zu einer gegebenen Eingabe $v \in A$ eine Ausgabe $w \in B$ berechnen.
- Eingaben/Ausgaben von Turingmaschinen: Wörter aus Σ^*/Γ^*.
- Eingaben und Ausgaben von WHILE-Programmen: natürliche Zahlen.
  - Beziehung zwischen $\mathbb{N}_0$ und Σ^* über d-adische Kodierung, wobei $d = |\Sigma| + 1$, z.B. jedes Wort über $\{a, b\}$ wird als Ternärzahl interpretiert.
  - Wir arbeiten mit $d = |\Sigma| + 1$ um ohne weitere Tricks z.B. a von aa zu unterscheiden, diese sollten nicht auf 0 und $0 \cdot 0 = 0$ abgebildet werden.

Syntax der WHILE-Programme

Syntax: "Lehre vom Satzbau"

- Spezifikation der zulässigen Sprachelemente und formale Regeln für den Aufbau des Codes.
- Rekursiv definiert.

Semantik: Bedeutung eines Programms P.

- Das Programm benutzt Variablen $\mathtt{x}_1, \mathtt{x}_2, \ldots$
- Definiert die partielle Funktion $[\![\mathrm{P}]\!]$, die eine Belegung der Variablen auf eine neue Belegung von Variablen abbildet.

Syntax der WHILE-Programme

- Rekursionsanfang: atomare Programme:
  - Var := Const
  - Var := Var + Const
  - Var := Var $\dot{-}$ Const
- Schlüsselwörter Var und Const sind Platzhalter für Variablen $\mathtt{x}_1, \mathtt{x}_2, \dots$ und Konstanten (feste Zahlen aus $\mathbb{N}_0$).
- Beispiele:
  - $\mathtt{x}_7 := 42$
  - $\mathtt{x}_5 := \mathtt{x}_1 + 3$
  - $\mathtt{x}_1 := \mathtt{x}_2 \dot{-} 5$

Syntax der WHILE-Programme

- Rekursionsschritt:

  Wenn Prog ein gültiges WHILE-Programm ist (rekursiv definiert), dann auch

  - **loop** Var **do** Prog **end** — **for**/**loop**-Schleife

    und

  - **while** Var $\neq 0$ **do** Prog **end** — **while**-Schleife

- Wenn Prog_1, Prog_2 gültige WHILE-Programm sind, dann auch

  - $\text{Prog}_1 \, ; \text{Prog}_2$ — Hintereinanderausführung

Semantik WHILE-Programme

Semantik: Bedeutung eines Programms P

- Das Programm benutzt Variablen $\mathtt{x}_1, \mathtt{x}_2, \ldots$
- Definiert die partielle Funktion $[\![\mathrm{P}]\!]$, die eine Belegung der Variablen auf eine neue Belegung von Variablen abbildet.
- Belegung = Initialisierung der Variablen als Eingabe.
- Konstanten repräsentieren auf natürliche Weise Werte aus $\mathbb{N}_0$.
- Die Variablen erhalten durch die Programmausführung neue Werte aus $\mathbb{N}_0$.
- Wenn das Programm terminiert, haben die Variablen neue Werte $\rightarrow$ die Ausgabe (wenn das Programm nicht terminiert ist die Ausgabe undefiniert, es werden partielle Funktionen definiert).

Semantik WHILE-Programme (anschaulich)

- $\mathtt{x_i} \coloneqq \mathtt{j}$ für $j \in \mathbb{N}_0$ weist der Variablen $\mathtt{x_i}$ den Wert j zu.
- $\mathtt{x_i} \coloneqq \mathtt{x_j} + \mathtt{k}$ weist der Variablen $\mathtt{x_i}$ die Summe der Werte der Variablen $\mathtt{x_j}$ und k zu.
- $\mathtt{x_i} \coloneqq \mathtt{x_j} \dot{-} \mathtt{k}$ weist der Variable $\mathtt{x_i}$ den Wert der Variablen $\mathtt{x_j}$ minus k zu, oder 0 falls dieser Wert negativ ist.

Semantik WHILE-Programme (anschaulich)

- **loop** $\mathtt{x_i}$ **do** Prog **end** führt das Programm P x_i mal aus.
  - Änderungen des Wertes von $\mathtt{x_i}$ während der Ausführung von P haben keinen Einfluss auf die Anzahl der Schleifendurchläufe.
- **while** $\mathtt{x_i} \neq 0$ **do** P **end** führt das Programm P so lange aus, bis $\mathtt{x_i}$ den Wert 0 annimmt. Geschieht das nicht, so terminiert die Schleife nicht.
- Die Hintereinanderausführung $\mathrm{P}_1;\mathrm{P}_2$ führt zuerst das Programm P_1 und dann das Programm P_2 aus.

Semantik der WHILE-Programme (formal)

- Sei $X = \{\mathtt{x_1}, \mathtt{x_2}, \ldots\}$ die Menge der Variablen.

▷ Eine Belegung oder Interpretation der Variablen aus X mit Werten aus $\mathbb{N}_0$ ist eine Abbildung $\bar{a} : X \to \mathbb{N}_0$.

  - Wir schreiben $\bar{a}_i$ für den Wert der Variablen $\mathtt{x_i}$.
  - Wenn ein Programm nur die Variablen $\mathtt{x_1}, \ldots, \mathtt{x_k}$ benutzt, so müssen wir nur für diese die Werte der Belegung angeben.

- Semantik:
  - Sei P ein WHILE-Programm.
  - P definiert partielle Abbildung $[\![\mathrm{P}]\!]$,
    die Belegung eine $\bar{a}$ auf eine neue Belegung $\bar{b} = [\![\mathrm{P}]\!](\bar{a})$ abbildet,
  - oder undefiniert ist, falls P mit Belegung $\bar{a}$ nicht terminiert.

Semantik der WHILE-Programme: Formal

- Sei $\bar{a}$ Belegung, $\mathtt{x_i} \in X$ und $a \in \mathbb{N}_0$. Definiere $\bar{a}[\mathtt{x_i} \mapsto a]$ als die Belegung mit
$$(\bar{a}[\mathtt{x_i} \mapsto a])_j = \begin{cases} a & \text{falls } j = i \\ \bar{a}_j & \text{falls } j \neq i. \end{cases}$$

- Sei $f\colon A \to B$ eine partielle Funktion. Definiere die n-fache Ausführung von f induktiv:
  - $f^0(a) = a$ für alle $a \in A$ und
  - $f^{i+1}(a) = f(f^i(a))$ für $i \geq 1$ und alle $a \in A$. Falls f auf $f^i(a)$ undefiniert ist, so ist $f^{i+1}(a)$ undefiniert.

Semantik der WHILE-Programme: Formal

- Sei P ein WHILE-Programm.
- Die Abbildung $[\![\mathrm{P}]\!]$ ist auf Belegungen $\bar{a}$ induktiv definiert:
  - $[\![\mathtt{x_i} := \mathtt{j}]\!](\bar{a}) = \bar{a}[\mathtt{x_i} \mapsto j]$,
  - $[\![\mathtt{x_i} := \mathtt{x_j} + \mathtt{k}]\!](\bar{a}) = \bar{a}[\mathtt{x_i} \mapsto \bar{a}(\mathtt{x_j}) + k]$,
  - $[\![\mathtt{x_i} := \mathtt{x_j} \dot{-} \mathtt{k}]\!](\bar{a}) = \bar{a}[\mathtt{x_i} \mapsto \bar{a}(\mathtt{x_j}) \dot{-} k]$,

Semantik der WHILE-Programme: Formal

- $[\![\textbf{loop}\ \ \mathtt{x_i}\ \ \textbf{do}\ \ \mathrm{P}\ \ \textbf{end}]\!]\,(\bar{a}) = [\![\mathrm{P}]\!]^{\bar{a}_i}\,(\bar{a})$,
- $[\![\textbf{while}\ \ \mathtt{x_i} \neq 0\ \ \textbf{do}\ \ \mathrm{P}\ \ \textbf{end}]\!]\,(\bar{a}) =$

$$\begin{cases} [\![\mathrm{P}]\!]^{n}\,(\bar{a}) & \text{falls } n \text{ minimal ist, so dass } [\![\mathrm{P}]\!]^{n}\,(\bar{a}) \text{ de-} \\ & \text{finiert ist und } ([\![\mathrm{P}]\!]^{n}\,(\bar{a}))_i = 0 \\ \text{undefiniert} & \text{falls ein solches } n \text{ nicht existiert,} \end{cases}$$

- $[\![\mathrm{P}_1\,;\mathrm{P}_2]\!]\,(\bar{a}) = [\![\mathrm{P}_2]\!]\,([\![\mathrm{P}_1]\!]\,(\bar{a}))$

  (hier ist $[\![\mathrm{P}_2]\!]\,([\![\mathrm{P}_1]\!]\,(\bar{a}))$ undefiniert, wenn $[\![\mathrm{P}_1]\!]\,(\bar{a})$ undefiniert ist).

WHILE-berechenbare Funktionen

- Angenommen ein Programm P benutzt die Variablen $\mathtt{x}_1, \ldots, \mathtt{x}_\ell$ für ein ℓ (wir können Variablen umbenennen um Lücken zu schließen).
- Wir schreiben $(a_1, \ldots, a_\ell)$ für die Belegung $\bar{a}(\mathtt{x_i}) = a_i$ für alle $1 \leq i \leq \ell$.
- Dann können wir $[\![\mathrm{P}]\!](a_1, \ldots, a_\ell) = (b_1, \ldots, b_\ell)$ schreiben, die Belegung aller nicht vorkommenden Variablen spielt keine Rolle.
- WHILE-Programme berechnen in diesem Sinne partielle Funktionen $f\colon \mathbb{N}_0^\ell \to \mathbb{N}_0^\ell$.
- Viele der Variablen sind Hilfsvariablen, deren Belegung in der Ein- oder Ausgabe uns nicht interessiert.
- Wir möchten partielle Funktionen $f\colon \mathbb{N}_0^k \to \mathbb{N}_0^m$ für $k, m \leq \ell$ berechnen.

WHILE-berechenbare Funktionen (informell)

- Wir möchten partielle Funktionen $f\colon \mathbb{N}_0^k \to \mathbb{N}_0^m$ für $k, m \le \ell$ berechnen.
- Initialisiere $\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{k}$ mit dem Eingabe-Tupel $(a_1, \ldots, a_k)$.
- Alle anderen Variablen werden mit 0 initialisiert.
- Gebe zusätzlich an, welche der Variablen die m-stellige Ausgabe bilden sollen.

WHILE-berechenbare Funktionen (informell)

- Beispiel: Multipliziere $\mathtt{x}_1$ mit $\mathtt{x}_2$, so dass das Ergebnis in $\mathtt{x}_3$ steht.
  - $\mathtt{x}_3 := 0;$
  - **while** $\mathtt{x}_2 \neq 0$ **do**
  -     **loop** $\mathtt{x}_1$ **do** $\mathtt{x}_3 := \mathtt{x}_3 + 1$ **end**;   Berechne $\mathtt{x}_3 + \mathtt{x}_1$
  -     $\mathtt{x}_2 := \mathtt{x}_2 \dot{-} 1$
  - **end**
- Mit Eingabevariablen $\mathtt{x}_1, \mathtt{x}_2$ und Ausgabevariable $\mathtt{x}_3$ berechnet das Programm die Multiplikationsfunktion
  $\text{add} : \mathbb{N}_0^2 \to \mathbb{N}_0 : (\mathtt{x}_1, \mathtt{x}_2) \mapsto \mathtt{x}_1 \cdot \mathtt{x}_2$.

WHILE-berechenbare Funktionen (formal)

Definition WHILE-berechenbare Funktion

Eine partielle Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0^m$ heißt WHILE-berechenbar, wenn es ein WHILE-Programm P gibt mit

- den Variablen $\mathtt{x}_1, \ldots, \mathtt{x}_\ell$ für ein $\ell \geq k$, so dass
- $[\![\mathrm{P}]\!](a_1, \ldots, a_k, \underbrace{0, \ldots, 0}_{\ell - k \text{ mal}})$ definiert ist für alle $(a_1, \ldots, a_k) \in \mathbb{N}_0^k$ auf denen f definiert ist, und so dass
- $\big([\![\mathrm{P}]\!](a_1, \ldots, a_k, \underbrace{0, \ldots, 0}_{\ell - k \text{ mal}})\big)_i = f(a_1, \ldots, a_k)_i$ ist, für $1 \leq i \leq m$.

Subprogramme

- Bequemlichkeiten:
  - Zuweisung $\mathtt{x_i} := \mathtt{x_j}$ entspricht dem Programm
  - $\mathtt{x_i} := 0;$
  - **loop** $\mathtt{x_j}$ **do** $\mathtt{x_i} := \mathtt{x_i} + 1$
- Jede bereits konstruierte WHILE-berechenbare Funktion kann als atomare Operation benutzt werden.
- Beispiel: Multiplikation
  - $\mathtt{x_1} := \mathtt{x_2} \cdot \mathtt{x_3}$

▷ $\mathtt{x_1} := 0;$ ▷ **loop** $\mathtt{x_2}$ **do** $\mathtt{x_1} := \mathtt{x_1} + \mathtt{x_3}$ **end**	▷ $\mathtt{x_1} := 0;$ **loop** $\mathtt{x_2}$ **do** ▷ **loop** $\mathtt{x_3}$ **do** $\mathtt{x_1} := \mathtt{x_1} + 1$ **end** ▷ **end**

Beispiel: If-Abfragen

- **if** $\mathtt{x_i} = 0$ **then** P **end**

  als Abkürzung für

  - $\mathtt{x_j} := 1$;
  - **loop** $\mathtt{x_i}$ **do** $\mathtt{x_j} := 0$ **end**;
  - **loop** $\mathtt{x_j}$ **do** P **end**

loop-Schleifen werden nicht gebraucht

- loop-Schleifen werden in einer minimalen Definition von WHILE-Programmen nicht gebraucht:
  - **loop** $\mathtt{x_i}$ **do** P **end**

    kann simuliert werden durch
  - $\mathtt{x_j} := \mathtt{x_i}$;
while $\mathtt{x_j} \neq 0$ **do**
    $\mathtt{x_j} := \mathtt{x_j} \dot{-} 1$;
    P
end
  - Hier ist $\mathtt{x_j}$ eine Variable, die in $\mathtt{P}$ nicht vorkommt.

WHILE-Programme vs Turingmaschinen

- Turingmaschinen berechnen partielle Funktionen $f : \Sigma^* \to \Gamma^*$.
  - $\Sigma \subseteq \Gamma$
- Wir möchten Wörter mit Zahlen identifizieren:
  - Setze $d = |\Gamma| + 1$.
  - Ordne die Symbole von Γ als $a_1, \dots, a_{d-1}$.
  - Betrachte die d-adische Zahlendarstellung: ein Wort $v_1 \dots v_\ell = a_{i_1} \dots a_{i_\ell}$ wird interpretiert als

$$(i_1, \dots, i_\ell)_d \quad := \sum_{j=1}^{\ell} i_j \cdot d^{\ell-j}$$

- Beispiel: $\Sigma = \Gamma = \{a, b\}$
  - Sei a das erste und b das zweite Symbol.
  - $(abaa)_3 = 1 \cdot 3^0 + 1 \cdot 3^1 + 2 \cdot 3^2 + 1 \cdot 3^3 = 1 + 3 + 18 + 27 = 49$

WHILE-Programme vs Turingmaschinen

Satz

Jede WHILE-berechenbare Funktion $f\colon \mathbb{N}_0 \to \mathbb{N}_0$ ist Turing-berechenbar.

Beweis (Skizze, $\Rightarrow$).

- Sei P ein WHILE-Programm, das f berechnet.
- Annahme: P benutzt keine **loop**-Schleifen.
- Annahme: Die Variablen von P sind $\mathtt{x}_1, \ldots, \mathtt{x}_\ell$.
- Konstruiere Turingmaschine M mit ℓ Bändern.
  - Das i-te Band dient zum Speichern von $\mathtt{x}_\mathtt{i}$.
  - Die Zustände von M repräsentieren den Fortschritt der Programmausführung.

WHILE-Programme vs Turingmaschinen

Satz

Jede Turing-berechenbare Funktion $f\colon \mathbb{N}_0 \to \mathbb{N}_0$ ist WHILE-berechenbar.

Beweis (Skizze, $\Leftarrow$).

- Sei umgekehrt M eine deterministische Turingmaschine, die f berechnet.
- Annahme: $\Gamma = \{a_1, \ldots, a_{d-1}\}$, also $d = |\Gamma| + 1$.
- Annahme: $Q = \{q_0, \ldots, q_k\}$ und q_0 ist der akzeptierende Zustand.
  - Alphabetssymbole und Zustände können als natürliche Zahl interpretiert werden.

WHILE-Programme vs Turingmaschinen

- Kodiere Konfiguration

$$\alpha = (q, v, k) \in Q \times \Gamma^{\omega} \times \mathbb{N}$$

wobei $n \geq k$ minimal ist mit $v_i = \sqcup$ für alle $i > n$ durch vier natürliche Zahlen.

  - x_1 repräsentiert $n - 1$ (die Wortlänge -1 ist die Ausgabe der TM, $v_1 = [$ wird nicht mitgezählt),
  - x_2 repräsentiert $v_1 \dots v_{k-1}$,
  - x_3 repräsentiert q,
  - x_4 repräsentiert $v_k \dots v_n$.

WHILE-Programme vs Turingmaschinen

- Falls $v_1 \dots v_{k-1} = a_{i_1} \dots a_{i_{k-1}}$, so setzen wir $x_2 = (i_1, \dots, i_{k-1})_d$, d. h. wir betrachten $i_1 \dots i_{k-1}$ als eine Zahl in d-ärer Zahlendarstellung.
- Falls $q = q_m$, so setzen wir $x_3 = m$.
- Falls $v_k \dots v_n = a_{i_k} \dots a_{i_n}$, so setzen wir

$$x_4 = (i_n, \dots, i_k)_d,$$

d.h. wir betrachten den zweiten Teil des Wortes in umgekehrter d-ärer Zahlendarstellung.

WHILE-Programme vs Turingmaschinen

- Berechnungsschritte → arithmetische Operationen
  - Lesen des aktuellen Symbols a_{j_1}:
    - ▷ $x_4 = (i_n, \ldots, i_k)_d \quad \Rightarrow \quad i_k = x_4 \mod d$.
  - Ändern des aktuellen Symbols zu a_j:
    - ▷ Neuer Wert von x_4 ist $(i_n, \ldots, i_{k+1}, j)_d$.
    - ▷ $(i_n, \ldots, i_k)_d \operatorname{div} d$
      (Division nach unten abgerundet, es verschwindet die letzte Stelle),
    - ▷ Multiplikation mit d und
    - ▷ Addition von j.
  - Verschieben des Schreib-Lese-Kopfes und
    Update der Wortlänge: ähnliche arithmetische Operation.
- All diese Operationen sind WHILE-berechenbar.

WHILE-Programme vs Turingmaschinen

WHILE-Programm für M:

1) Erzeuge aus Eingabe die Kodierung der Startkonfiguration in den Variablen x_1, x_2, x_3, x_4.
2) In einer **while**-Schleife (**while** $\mathrm{x}_3 \neq 0$) wird bei jedem Durchlauf ein Schritt der Berechnung simuliert:
   - In Abhängigkeit vom aktuellen Zustand (Wert von x_3) und
   - dem gelesenen Symbol, d. h. von $x_4 \bmod d$
   - wird mit den oben dargestellten arithmetischen Operationen das aktuelle Symbol verändert,
   - der Schreib-Lese-Kopf bewegt
   - und die Wortlänge (die Ausgabe in x_1) aktualisiert.

Die **while**-Schleife terminiert, wenn der $x_3 = 0$,
wenn also der akzeptierende Zustand q_0 erreicht wird.

Zusammenfassung

- WHILE: stark eingeschränkte, modellhafte imperative Sprache.

Satz

Eine partielle Funktion $f\colon \mathbb{N}_0 \to \mathbb{N}_0$ ist WHILE-berechenbar genau dann wenn sie Turing-berechenbar ist.

- Wir verstehen immer besser, was berechenbar ist!