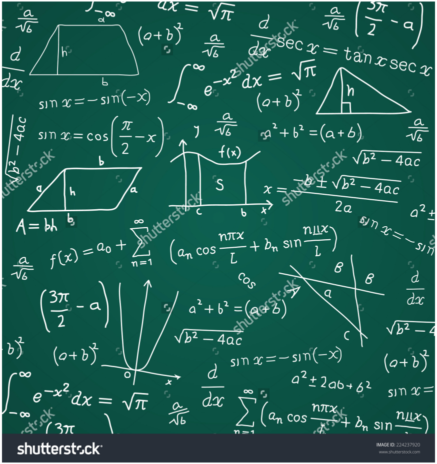
Teil II:
Berechenbarkeit und Komplexität

Algorithmen und Komplexität

22. November 2016

Berechenbarkeitstheorie



RAM-Maschine

	$\vdots$	$\vdots$
1: $M_{-1} \leftarrow 1$	M_{-2}:	$\perp$
2: $M_0 \leftarrow 1$	M_{-1}:	1
3: $M_0 \leftarrow M_0 * M_1$	M_0:	243
4: $M_2 \leftarrow M_2 - M_{-1}$	M_1:	3
5: GOTO 3 IF $M_2 > 0$	M_2:	10
	$\vdots$	$\vdots$

Programmzähler: 5

Die Random Access Machine (RAM) mit einem Programm der Länge 5, ausgeführt auf Eingabe $M_1 = 3$ und $M_2 = 15$.

Programme

Definition:

Ein Programm ist eine endliche Folge von zulässigen Befehlen.

- Details und formale Definition: Skript, Kapitel 1.1.1
- Ein Programm muss nicht zwingend endliche Laufzeit haben!

Funktionen und Programme

- Wir wollen Programme benützen, um Funktionen zu berechnen.

Funktionen und Programme

- Wir wollen Programme benützen, um Funktionen zu berechnen.
- RAM-Programme arbeiten mit Zahlen aus $\mathbb{Z}$, während wir Funktionen über $\{0,1\}^*$ definiert haben.

Funktionen und Programme

- Wir wollen Programme benützen, um Funktionen zu berechnen.
- RAM-Programme arbeiten mit Zahlen aus $\mathbb{Z}$, während wir Funktionen über $\{0,1\}^*$ definiert haben.
- Ist das ein Problem?

Funktionen und Programme

- Wir wollen Programme benützen, um Funktionen zu berechnen.
- RAM-Programme arbeiten mit Zahlen aus $\mathbb{Z}$, während wir Funktionen über $\{0,1\}^*$ definiert haben.
- Ist das ein Problem?
- Es ist leicht, eine Bijektion zwischen $\{0,1\}^*$ und $\mathbb{Z}$ zu finden:

x	$\ldots$	000	01	00	0	λ	1	10	11	100	$\ldots$
$b(x)$	$\ldots$	-4	-3	-2	-1	0	1	2	3	4	$\ldots$

Programme berechnen Funktionen

Sei A ein Programm, das immer endliche Laufzeit hat, und sei $x \in \{0,1\}^*$.

1. Schreibe $b(x) \in \mathbb{Z}$ als Eingabe in M_0.

Programme berechnen Funktionen

Sei A ein Programm, das immer endliche Laufzeit hat, und sei $x \in \{0,1\}^*$.

1. Schreibe $b(x) \in \mathbb{Z}$ als Eingabe in M_0.
2. Lasse das Programm A auf dieser Eingabe laufen, bis es hält.

Programme berechnen Funktionen

Sei A ein Programm, das immer endliche Laufzeit hat, und sei $x \in \{0,1\}^*$.

1. Schreibe $b(x) \in \mathbb{Z}$ als Eingabe in M_0.
2. Lasse das Programm A auf dieser Eingabe laufen, bis es hält.
3. Betrachte die Zahl, die dann in M_0 steht:

Programme berechnen Funktionen

Sei A ein Programm, das immer endliche Laufzeit hat, und sei $x \in \{0,1\}^*$.

1. Schreibe $b(x) \in \mathbb{Z}$ als Eingabe in M_0.
2. Lasse das Programm A auf dieser Eingabe laufen, bis es hält.
3. Betrachte die Zahl, die dann in M_0 steht:
   - $M_0 > 0$: interpretieren wir als 1

Programme berechnen Funktionen

Sei A ein Programm, das immer endliche Laufzeit hat, und sei $x \in \{0, 1\}^*$.

1. Schreibe $b(x) \in \mathbb{Z}$ als Eingabe in M_0.
2. Lasse das Programm A auf dieser Eingabe laufen, bis es hält.
3. Betrachte die Zahl, die dann in M_0 steht:
   - $M_0 > 0$: interpretieren wir als 1
   - $M_0 \leq 0$: interpretieren wir als 0

Programme berechnen Funktionen

Sei A ein Programm, das immer endliche Laufzeit hat, und sei $x \in \{0,1\}^*$.

1. Schreibe $b(x) \in \mathbb{Z}$ als Eingabe in M_0.
2. Lasse das Programm A auf dieser Eingabe laufen, bis es hält.
3. Betrachte die Zahl, die dann in M_0 steht:
   - $M_0 > 0$: interpretieren wir als 1
   - $M_0 \leq 0$: interpretieren wir als 0

Wir stellen fest:
Unser Programm A berechnet eine Funktion $A : \{0,1\}^* \to \{0,1\}$.

Berechenbare Sprachen und Funktionen

Definition:

- Eine Sprache L heisst *berechenbar*, falls es ein Programm A gibt, so dass für alle $x \in \{0, 1\}^*$ die Gleichung $A(x) = L(x)$ gilt.

Berechenbare Sprachen und Funktionen

Definition:

- Eine Sprache L heisst *berechenbar*, falls es ein Programm A gibt, so dass für alle $x \in \{0,1\}^*$ die Gleichung $A(x) = L(x)$ gilt.
- Eine Funktion f heisst *berechenbar*, falls es ein Programm A gibt, so dass für alle $x \in \{0,1\}^*$ die Gleichung $A(x) = f(x)$ gilt.

Berechenbare Sprachen und Funktionen

Definition:

- Eine Sprache L heisst *berechenbar*, falls es ein Programm A gibt, so dass für alle $x \in \{0,1\}^*$ die Gleichung $A(x) = L(x)$ gilt.
- Eine Funktion f heisst *berechenbar*, falls es ein Programm A gibt, so dass für alle $x \in \{0,1\}^*$ die Gleichung $A(x) = f(x)$ gilt.

Serie 10, Aufgabe 3:

Es existieren nicht-berechenbare Sprachen.

Andere Rechenmodelle?

Offensichtlich ist unsere Definition von berechenbar abhängig von der RAM-Maschine. Ist das eine Einschränkung?

Andere Rechenmodelle?

Offensichtlich ist unsere Definition von berechenbar abhängig von der RAM-Maschine. Ist das eine Einschränkung?

- es gibt weniger mächtige Rechenmodelle

Andere Rechenmodelle?

Offensichtlich ist unsere Definition von berechenbar abhängig von der RAM-Maschine. Ist das eine Einschränkung?

- es gibt weniger mächtige Rechenmodelle
- es gibt aber auch mächtigere Rechenmodelle

Andere Rechenmodelle?

Offensichtlich ist unsere Definition von berechenbar abhängig von der RAM-Maschine. Ist das eine Einschränkung?

- es gibt weniger mächtige Rechenmodelle
- es gibt aber auch mächtigere Rechenmodelle
- die meisten anderen Modelle wie z.B. Turing-Maschinen sind gleich mächtig wie die RAM-Maschine!

Andere Rechenmodelle?

Offensichtlich ist unsere Definition von berechenbar abhängig von der RAM-Maschine. Ist das eine Einschränkung?

- es gibt weniger mächtige Rechenmodelle
- es gibt aber auch mächtigere Rechenmodelle
- die meisten anderen Modelle wie z.B. Turing-Maschinen sind gleich mächtig wie die RAM-Maschine!

Church-Turing-These:

Die Klasse der RAM-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein. (nicht beweisbar!)

lesenswert:
'History of the Church-Turing thesis' auf englischer Wikipedia

Codieren in $\{0,1\}^*$

- Wir haben die Begriffe Sprache, Funktion, Berechenbarkeit über $\{0,1\}^*$ eingeführt.

Codieren in $\{0,1\}^*$

- Wir haben die Begriffe Sprache, Funktion, Berechenbarkeit über $\{0,1\}^*$ eingeführt.
- Ist dies hinreichend flexibel?

Codieren in $\{0, 1\}^*$

- Wir haben die Begriffe Sprache, Funktion, Berechenbarkeit über $\{0, 1\}^*$ eingeführt.
- Ist dies hinreichend flexibel?
- Bereits gesehen: $\{0, 1\}^*$ ist bijektiv zu $\mathbb{Z}$.

Codieren in $\{0,1\}^*$

- Wir haben die Begriffe Sprache, Funktion, Berechenbarkeit über $\{0,1\}^*$ eingeführt.
- Ist dies hinreichend flexibel?
- Bereits gesehen: $\{0,1\}^*$ ist bijektiv zu $\mathbb{Z}$.
- Wir wollen auch Paare (x, y) als Elemente von $\{0,1\}^*$ codieren können, oder auch ganze Programme.

Codierung von Paaren

Seien $x, y \in \{0,1\}^*$. Wir möchten das Paar (x, y) als Element von $\{0,1\}^*$ auffassen. Wie geht das?

Codierung von Paaren

Seien $x, y \in \{0,1\}^*$. Wir möchten das Paar (x, y) als Element von $\{0,1\}^*$ auffassen. Wie geht das?

Idee:

- verdopple jedes Bit b von x und y
- dies ergibt x' und y'
- füge x' und y' mit 'Trennsymbol' 10 zusammen

Codierung von Paaren

Seien $x, y \in \{0,1\}^*$. Wir möchten das Paar (x, y) als Element von $\{0,1\}^*$ auffassen. Wie geht das?

Idee:

- verdopple jedes Bit b von x und y
- dies ergibt x' und y'
- füge x' und y' mit 'Trennsymbol' 10 zusammen

Beispiel:

Sei $x = 100$ und $y = 11$. Dann codieren wir das Paar (x, y) als 110000101111.

Codierung von Programmen

Serie 1, Aufgabe 2:

$$
\begin{array}{ll}
1: & M_{-2} \leftarrow 1 \\
2: & M_{-1} \leftarrow M_{M_0} \\
3: & M_{-1} \leftarrow M_{-1} + M_{-1} \\
4: & M_{M_0} \leftarrow M_{-1} \\
5: & M_0 \leftarrow M_{M_0} - M_2 \\
6: & \text{GOTO 2 IF } M_0 > 0
\end{array}
$$

Codierung von Programmen

Serie 1, Aufgabe 2:

$$
\begin{array}{rl}
1: & M_{-2} \leftarrow 1 \\
2: & M_{-1} \leftarrow M_{M_0} \\
3: & M_{-1} \leftarrow M_{-1} + M_{-1} \\
4: & M_{M_0} \leftarrow M_{-1} \\
5: & M_0 \leftarrow M_{M_0} - M_2 \\
6: & \text{GOTO 2 IF } M_0 > 0
\end{array}
$$

Beobachtung:

Wir können jedes Symbol als Bitstring codieren (z.B. ASCII). Dies ergibt einen endlichen Bitstring, somit können wir das Programm als Element von $\{0,1\}^*$ codieren.

Gödelsche Unvollständigkeitssätze

Sei V ein Programm, das Beweise verifizieren kann. Wir nehmen an, dass V widerspruchsfrei ist, d.h. V kann nicht gleichzeitig S und $\neg S$ beweisen.

Gödelsche Unvollständigkeitssätze

Sei V ein Programm, das Beweise verifizieren kann. Wir nehmen an, dass V widerspruchsfrei ist, d.h. V kann nicht gleichzeitig S und $\neg S$ beweisen.

1. UV-Satz

Sei V so dass für alle A, x die Aussage '$A(x)$ hält' V-beweisbar ist. Dann gibt es ein Programm A und eine Eingabe x, so dass $A(x)$ nicht hält, dies aber nicht V-beweisbar ist.

Gödelsche Unvollständigkeitssätze

Sei V ein Programm, das Beweise verifizieren kann. Wir nehmen an, dass V widerspruchsfrei ist, d.h. V kann nicht gleichzeitig S und $\neg S$ beweisen.

1. UV-Satz
Sei V so dass für alle A, x die Aussage '$A(x)$ hält' V-beweisbar ist. Dann gibt es ein Programm A und eine Eingabe x, so dass $A(x)$ nicht hält, dies aber nicht V-beweisbar ist.

2. UV-Satz
Kein widerspruchsfreies Programm V kann die eigene Widerspruchsfreiheit beweisen.