

# Theoretische Informatik 2

## Berechenbarkeit und Komplexität

Inoffizielles Skript  
Marvin Borner

**WARNUNG WIP: Fehler zu erwarten!**  
**Stand: 6. Juni 2023, 13:28**  
Bitte meldet euch bei mir, falls ihr Fehler findet.

Vorlesung gehalten von  
**Ulrike von Luxburg**  
Sommersemester 2023

## Inhalt

<b>1</b>	<b>Reguläre Sprachen und endliche Automaten</b>	<b>1</b>
1.1	Wörter und Sprachen . . . . .	2
1.2	Endlicher, deterministischer Automat . . . . .	3
1.3	Reguläre Sprachen und Abschlusseigenschaften . . . . .	5
1.4	Nicht-deterministische Automaten . . . . .	7
1.5	Mächtigkeit . . . . .	8
1.6	Reguläre Ausdrücke . . . . .	9
1.7	Pumping-Lemma . . . . .	10
1.8	Pushdown automaton . . . . .	11
1.9	Grammatiken . . . . .	12
<b>2</b>	<b>Turingmaschinen</b>	<b>12</b>
2.1	Varianten . . . . .	14
<b>3</b>	<b>Entscheidbarkeit von Sprachen vs. Berechenbarkeit von Funktionen</b>	<b>14</b>
3.1	Gödelnummer . . . . .	15
3.2	Die universelle Turingmaschine . . . . .	17
3.3	Abzählbar unendliche Mengen . . . . .	17
3.3.1	Spaß mit Abzählbarkeit . . . . .	18
3.4	Wie groß ist $\Sigma^*$ ? . . . . .	19
3.5	Überabzählbare Mengen . . . . .	20
3.5.1	$2^{\mathbb{N}}$ ist überabzählbar . . . . .	20
3.5.2	Indikatorfunktion einer abzählbaren Menge . . . . .	20
3.5.3	Mächtigkeit von Potenzmengen . . . . .	20
3.5.4	Russels Paradox . . . . .	20
3.6	Sprachen, die nicht semi-entscheidbar sind . . . . .	21
3.6.1	Diagonalsprache . . . . .	21
3.6.2	Game of Life . . . . .	22
3.7	Sprachen, die semi-entscheidbar, aber nicht entscheidbar sind . . . . .	22
3.7.1	Komplementbildung . . . . .	23
3.8	Abbildungs-Reduktionen in der Berechenbarkeitstheorie . . . . .	24
3.9	Das Halteproblem und viele weitere Probleme in $RE \setminus R$ . . . . .	26
3.9.1	Reduktion $\overline{D_{TM}} \preceq A_{TM}$ . . . . .	26
3.9.2	Reduktion $A_{TM} \preceq H$ (allgemeines Halteproblem) . . . . .	27
3.9.3	Reduktion $H \preceq H_0$ (spezielles Halteproblem) . . . . .	27
3.9.4	Reduktion $H_0 \preceq K$ . . . . .	27
3.9.5	Schlussfolgerung . . . . .	28
<b>4</b>	<b>Der Satz von Rice</b>	<b>28</b>

## 1 Reguläre Sprachen und endliche Automaten

### Motivation

- Eingabe
- Verarbeitung (Berechnungen, Zustände)
- Ausgabe

## 1.1 Wörter und Sprachen

### Definition

Ein *Alphabet*  $\Sigma$  sei eine nicht-leere, endliche Menge. Ein *Wort*  $w$  ist entsprechend eine Folge von Elementen aus  $\Sigma$ .

### Beispiel

- $\Sigma = \{a, \dots, z\}$ ,  $w = \text{luxburg}$ ,  $|w| = 7$

### Definition

$\Sigma^n$  ist die Menge aller Wörter der Länge  $n$ . Die *Kleene'sche Hülle* ist  $\Sigma^* := \bigcup_{n=0}^{\infty} \Sigma^n$ .  
 $\Sigma^+ := \bigcup_{n=1}^{\infty} \Sigma^n$ .  
*Sprache*  $L$  über  $\Sigma$  ist eine Teilmenge von  $\Sigma^*$ .

### Definition

Eine *Konkatenation* ist eine Aneinanderhängung zweier Wörter  $u$  und  $w$ . Eine Konkatenation zweier *Sprachen*  $L_1, L_2$  ist  $L_1 \circ L_2 := \{uw \mid u \in L_1, w \in L_2\}$ . Die Kleene'sche Hülle einer Sprache  $L$  ist dann  $L^* := \{\underbrace{x_1 \dots x_k}_{\text{Konkatenation von } k \text{ Wörtern}} \mid x_i \in L, k \in \mathbb{N}_0\}$ .

Eine  $k$ -fache Aneinanderhängung von Wörtern ist  $w_k = \underbrace{w \dots w}_{k\text{-mal}}$ .

### Beispiel

- $w = 010$ ,  $u = 001$ ,  $wu = \underbrace{010}_w \underbrace{001}_u$ ,  $uwu = \underbrace{001}_u \underbrace{010}_w \underbrace{001}_u$
- $w^3 = 010 \ 010 \ 010$

### Bemerkung

Die Konkatenation auf  $\Sigma^*$  hat die Eigenschaften:

- assoziativ:  $a(bc) = (ab)c$
- nicht kommutativ:  $ab \neq ba$
- neutrales Element  $\varepsilon$ :  $\varepsilon a = a\varepsilon = a$
- ein inverses Element

### Definition

Ein Wort  $x$  heißt *Teilwort* eines Wortes  $y$ , falls es Wörter  $u$  und  $v$  gibt, sodass  $y = uxv$ .

- Falls  $u = \varepsilon$ ,  $x$  *Präfix* von  $y$
- Falls  $v = \varepsilon$ ,  $x$  *Suffix* von  $y$

### Beispiel

- 01 ist Teilwort von **00111**
- 10 ist Präfix von **1010011**
- 011 ist Suffix von **10101110011**

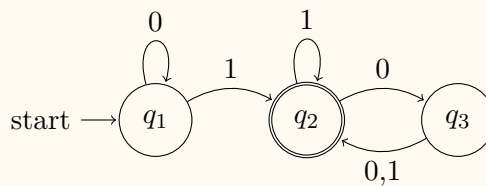
## 1.2 Endlicher, deterministischer Automat

### Definition

Für einen *endlichen, deterministischen Automat*  $(Q, \Sigma, \delta, q_0, F)$  ist

- $Q$  eine endliche Menge der *Zustände*
- $\Sigma$  das *Alphabet*
- $\delta : Q \times \Sigma \rightarrow Q$  die *Übergangsfunktion*
- $q_0 \in Q$  der *Startzustand*
- $F \subset Q$  die Menge der *akzeptierenden Zustände*

### Beispiel



$Q = \{q_1, q_2, q_3\}$ ,  $\Sigma = \{0, 1\}$ ,  $q_1$  Startzustand,  $F = \{q_2\}$ .

$\delta$  kann dargestellt werden durch

/	0	1
$q_1$	$q_1$	$q_2$
$q_2$	$q_3$	$q_2$
$q_3$	$q_2$	$q_2$

Die Zustandsfolge ist mit  $w = 001$

$$q_1 \xrightarrow{0} q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2.$$

### Definition

- partielle Übergangsfunktion: nicht alle Übergänge sind definiert
- totale Übergangsfunktion: alle Übergänge sind definiert

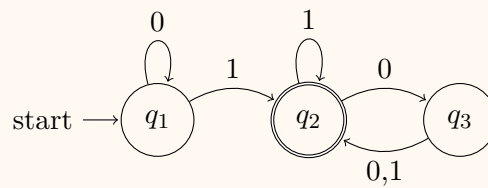
### Definition

Eine Folge  $s_0, \dots, s_n \in Q$  von Zuständen heißt *Berechnung* des Automaten  $M = (Q, \Sigma, \delta, q_0, F)$  auf dem Wort  $w = w_1 \dots w_n$ , falls

- $s_0 = q_0$
- $\forall i = 0, \dots, n-1 : s_{i+1} = \delta(s_i, w_{i+1})$

Es ist also eine "gültige" Folge von Zuständen, die man durch Abarbeiten von  $w$  erreicht.

## Beispiel



- $w = 001$  ergibt die Zustandsfolge  $q_1q_1q_1q_2$

## Definition

Eine Berechnung *akzeptiert* das Wort  $w$ , falls die Berechnung in einem akzeptierten Zustand endet.

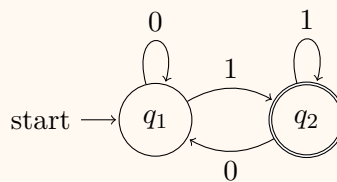
Die von einem endlichen Automaten  $M$  *akzeptierte* (erkannte) Sprache  $L(M)$  ist die Menge der Wörter, die von  $M$  akzeptiert werden:

$$L(M) := \{w \in \Sigma^* \mid M \text{ akzeptiert } w\}$$

## Bemerkung

Eine Berechnung kann mehrmals in akzeptierenden Zuständen eintreten/austreten. Wichtig ist der Endzustand, nachdem der letzte Buchstabe des Eingabewortes verarbeitet wurde.

## Beispiel



- $w = 1101 \rightarrow q_1q_2q_2q_1q_2 \rightarrow w$  wird akzeptiert
- $w = 010 \rightarrow q_1q_1q_2q_1 \rightarrow w$  wird **nicht** akzeptiert

Es folgt:

$$L(M) = \{w \in \Sigma^* \mid w = \varepsilon \text{ oder } w \text{ endet mit } 0\}$$

## Definition

Sei  $\delta : Q \times \Sigma \rightarrow Q$  eine Übergangsfunktion. Die *erweiterte Übergangsfunktion*  $\delta^* : Q \times \Sigma^* \rightarrow Q$  sei induktiv definiert:

- $\delta^*(q, \varepsilon) = q$  für alle  $q \in Q$
- Für  $w \in \Sigma^*$ ,  $a \in \Sigma$  ist:

$$\delta^*(q, wa) = \delta(\underbrace{\delta^*(q, w)}_{\text{Zustand nach Lesen von } w}, \overbrace{a}^{\text{Lesen von Buchstabe } a}).$$

### 1.3 Reguläre Sprachen und Abschlusseigenschaften

#### Definition

Eine Sprache  $L \subset \Sigma^*$  heißt *reguläre Sprache*, wenn es einen endlichen Automaten  $M$  gibt, der diese Sprache akzeptiert.

Die Menge aller regulären Sprachen ist *REG*.

#### Satz

Sei  $L$  eine reguläre Sprache über  $\Sigma$ . Dann ist auch  $\bar{L} := \Sigma^* \setminus L$  eine reguläre Sprache.

#### Beweis

- $L$  regulär  $\implies$  es gibt Automaten  $M = (Q, \Sigma, \delta, q_0, F)$ , der  $L$  akzeptiert
- Definiere “Komplementautomat”  $\bar{M} = (Q, \Sigma, \delta, q_0, \bar{F})$  mit  $\bar{F} := Q \setminus F$ .
- Dann gilt:

$$\begin{aligned} w \in \bar{L} &\iff M \text{ akzeptiert } w \text{ nicht} \\ &\iff \bar{M} \text{ akzeptiert } w. \end{aligned}$$

Q.E.D.

#### Satz

Die Menge der regulären Sprachen ist abgeschlossen bezüglich der Vereinigung:

$$L_1, L_2 \in \text{REG} \implies L_1 \cup L_2 \in \text{REG}.$$

#### Beweis

Sei  $M_1 = (Q_1, \Sigma_1, \delta_1, s_1, F_1)$  ein Automat, der  $L_1$  erkennt,  $M_2 = (Q_2, \Sigma_2, \delta_2, s_2, F_2)$  ein Automat, der  $L_2$  erkennt.

Wir definieren den Produktautomaten  $M := M_1 \times M_2$ :  $M = (Q, \Sigma, \Delta, s, F)$  mit

- $Q = Q_1 \times Q_2$
- $\Sigma = \Sigma_1 \cup \Sigma_2$ ,
- $\underbrace{s}_{\text{neuer Startzustand}} = (s_1, s_2)$ ,  $F = \{(f_1, f_2) \mid f_1 \in F_1 \text{ oder } f_2 \in F_2\}$ ,

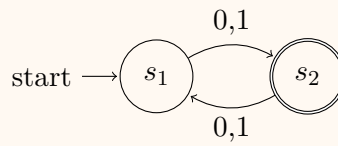
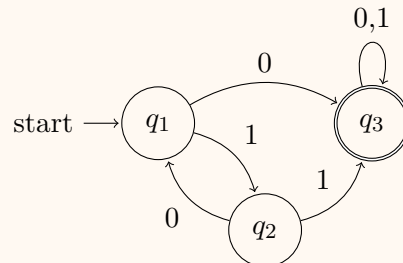
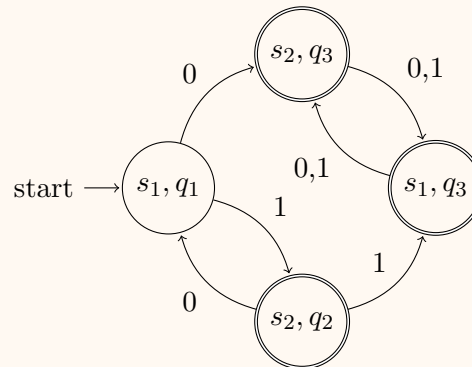
- $\underbrace{\Delta}_{\text{neue Übergangsfunktion}} : Q \times \Sigma \rightarrow Q$ ,

$$\Delta(\underbrace{(r_1, r_2)}_{\substack{\in Q_1 \quad \in Q_2}}, \underbrace{a}_{\in \Sigma}) = (\delta_1(r_1, a), \delta_2(r_2, a)).$$

Übertragung der Definition auf erweiterte Übergangsfunktionen: Beweis durch Induktion (ausgelassen).

Nach Definition von  $F$  akzeptiert  $M$  ein Wort  $w$ , wenn  $M_1$  oder  $M_2$  das entsprechende Wort akzeptieren. Der Satz folgt. Q.E.D.

## Beispiel

 $M_1$ : $M_2$ : $M_1 \times M_2$ :

## Satz

Seien  $L_1, L_2$  zwei reguläre Sprachen. Dann sind auch  $L_1 \cap L_2$  und  $L_1 \setminus L_2$  reguläre Sprachen.

## Beweis

- $L_1 \cap L_2$ : Beweis funktioniert analog wie für  $L_1 \cup L_2$ , nur mit

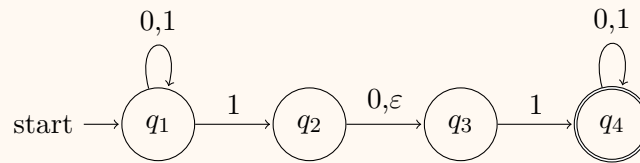
$$F := \{(q_1, q_2) \mid q_1 \in F_1 \text{ und } q_2 \in F_2\}.$$

- $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$

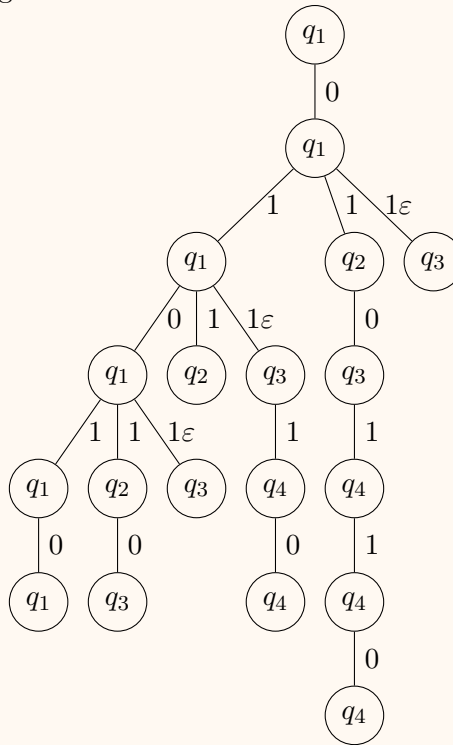
Q.E.D.

## 1.4 Nicht-deterministische Automaten

### Beispiel



Der komplette Berechnungsbaum:



### Definition

Ein *nicht-deterministischer Automat* besteht aus einem 5-Tupel  $(Q, \Sigma, \delta, q_0, F)$ .

- $Q, \Sigma, q_0, F$  wie beim deterministischen Automat,

- $\delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow \overset{(*)}{\mathcal{P}(Q)}$  Übergangsfunktion

(\*): Die Funktion definiert die **Menge** der möglichen Zustände, in die man von einem Zustand durch Lesen eines Buchstabens gelangen kann.

### Definition

Sei  $M = (Q, \Sigma, \delta, q_0, F)$  ein nicht-deterministischer endlicher Automat,  $w = w_1 \dots w_n \in \Sigma^*$ . Eine Folge von Zuständen  $s_0, s_1, \dots, s_m \in Q$  heißt *Berechnung von M auf w*, falls man  $w$  schreiben kann als  $w = u_1 u_2 \dots u_m$  mit  $u_i \in \Sigma \cup \{\varepsilon\}$ , sodass

Übergänge  $\varepsilon$ , hier  $u_i = \varepsilon$

- $s_0 = q_0$ ,
- für alle  $0 \leq i \leq m-1 : s_{i+1} \in \delta(s_i, u_{i+1})$ .

Die Berechnung heißt *akzeptierend*, falls  $s_m \in F$ .

Der nicht-deterministische Automat  $M$  *akzeptiert Wort w*, falls es eine akzeptierende Berechnung von  $M$  auf  $w$  gibt.



## Bemerkung

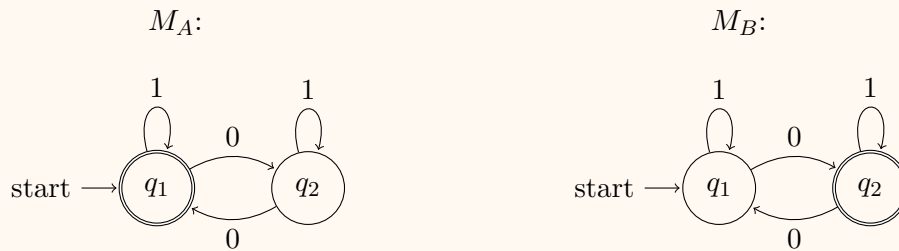
$\varepsilon$ -Transitionen: Ein nicht-deterministischer Automat kann bei "Lesen" des leeren Wortes  $\varepsilon$  einen Übergang machen, falls es so in der Übergangsfunktion definiert ist.

## Beispiel

Betrachte die regulären Sprachen

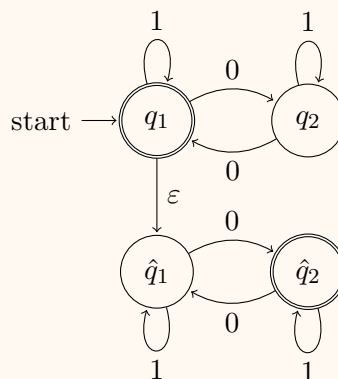
- $A := \{x \in \{0, 1\}^* \mid \text{Anzahl } 0 \text{ gerade}\}$
- $B := \{x \in \{0, 1\}^* \mid \text{Anzahl } 0 \text{ ungerade}\}$

Zugehörige Automaten:



Nun betrachte *Konkatenation*  $AB$ . Um die Sprache zu erkennen, müsste der Automat bei einer Eingabe zunächst einen ersten Teil  $A$  des Wortes betrachten und schauen, ob die Anzahl der 0 gerade ist. **Irgendwann** müsste er beschließen, dass nun der zweite Teil  $B$  des Wortes anfängt und er müsste schauen, ob dort die Anzahl der 0 ungerade ist.

"Irgendwann"  $\implies$  nicht-deterministisch.



## 1.5 Mächtigkeit

## Bemerkung

Die Mächtigkeit eines Automaten wird hierbei beschrieben durch die Anzahl an Sprachen, die dieser erkennen kann.

## Definition

Zwei Automaten  $M_1, M_2$  heißen *äquivalent*, wenn sie die gleiche Sprache erkennen:

$$L(M_1) = L(M_2)$$

## Satz

Zu jedem nicht-deterministischen endlichen Automaten gibt es einen äquivalenten deterministischen endlichen Automaten.

## Beweis

Lang aber trivial. Basically konstruiert man einfach eine deterministische Übergangsfunktion auf den nicht-deterministischen Verzweigungen.

## Satz

Es folgt:

Eine Sprache  $L$  ist regulär  $\iff$  es gibt einen nicht-deterministischen Automaten, der  $L$  akzeptiert.

## Satz

Die Klasse der regulären Sprachen ist abgeschlossen unter Konkatenation:

$$L_1, L_2 \in \text{REG} \implies L_1 L_2 \in \text{REG}$$

## Satz

Die Klasse REG ist abgeschlossen unter Bildung der Kleene'schen Hülle, d.h.:

$$L \in \text{REG} \implies L^* \in \text{REG}$$

## 1.6 Reguläre Ausdrücke

## Definition

Sei  $\Sigma$  ein Alphabet. Dann:

- $\underbrace{\emptyset}_{\text{leere Sprache}}$  und  $\underbrace{\varepsilon}_{\text{leeres Wort}}$  sind reguläre Ausdrücke.
- Alle Buchstaben aus  $\Sigma$  sind reguläre Ausdrücke.
- Falls  $R_1, R_2$  reguläre Ausdrücke sind, dann sind auch die folgenden Ausdrücke regulär:
  - $R_1 \cup R_2$ ,
  - $R_1 \circ R_2$ ,
  - $R_1^*$ .

## Definition

Sei  $R$  ein regulärer Ausdruck. Dann ist die *von  $R$  induzierte Sprache*  $L(R)$  wie folgt definiert:

- $R = \emptyset \implies L(R) = \emptyset$
- $R = \varepsilon \implies L(R) = \{\varepsilon\}$
- $R = \sigma$  für ein  $\sigma \in \Sigma \implies L(R) = \{\sigma\}$
- $R = R_1 \cup R_2 \implies L(R) = L(R_1) \cup L(R_2)$
- $R = R_1 \circ R_2 \implies L(R) = L(R_1) \circ L(R_2)$
- $R = R_1^* \implies L(R) = (L(R_1))^*$

## Satz

Eine Sprache ist genau dann regulär, wenn sie durch einen regulären Ausdruck beschrieben wird.

## Beweis

Strukturelle Induktion. Tja.

## 1.7 Pumping-Lemma

## Motivation

Frage: Gibt es Sprachen, die nicht regulär sind?

## Beispiel

$$L = \{0^n 1^n \mid n \in \mathbb{N}\} = \{01, 0011, 00111, \dots\}$$

Ein Automat, der  $L$  erkennt, müsste vermutlich “zählen” können. Mit endlich vielen Zuständen scheint dies für beliebig große Zahlen nicht möglich zu sein.

Aber: Wir kann man das formal beweisen?

## Satz

Sei  $A$  eine reguläre Sprache über das Alphabet  $\Sigma$ . Dann gibt es eine natürliche Zahl  $n$ , sodass sich alle Wörter  $s \in A$  mit Länge  $|s| \geq n$  zerlegen lassen in drei Teilworte  $s = xyz$ , mit  $x, y, z \in \Sigma^*$ , sodass gilt:

- $|y| > 0$ ,
- $|xy| \leq n$ ,
- $\forall i \geq 0 : xy^i z \in A$

## Beweis

Idee: Ein Automat mit  $n$  Zuständen besucht für die Verarbeitung eines Wortes  $w$  mit Länge  $> n$  immer  $n + 1$  Zustände  $\implies$  es gibt einen Zustand, der mindestens zweimal besucht wird.

## Bemerkung

Aus der Kontraposition folgt:  $A$  nicht regulär  $\implies \forall n \exists s \forall x, y, z \in \Sigma^* \exists i : xy^i z \notin A$

## Beispiel

Wir zeigen mit dem Pumping-Lemma, dass  $L = \{0^n 1^n \mid n \in \mathbb{N}\}$  nicht regulär ist.

## Beweis

Betrachte ein beliebiges  $n \in \mathbb{N}$ . Wähle das Wort  $s = 0^n 1^n$  (es gilt  $|s| > n$ ). Sei nun  $xyz = s$  eine beliebige Zerlegung (mit  $|y| > 0$ ,  $|xy| \leq n$ ). Man muss nun ein  $i$  finden, sodass  $xy^i z$  nicht in  $L$  ist.

- Fall 1:  $y$  besteht nur aus 0en:  $s = \overbrace{0}^x \overbrace{00}^y \overbrace{\dots 0111 \dots 1}^z$ . Dann ist  $xy^2 z \notin L$  (da es mehr 0en als 1en hat).
- Fall 2:  $y$  besteht nur aus 1en: analog.
- Fall 3:  $y$  hat 0en und 1en:  $s = \overbrace{0 \dots 0}^x \overbrace{011}^y \overbrace{\dots 1}^z$ . Dann ist aber  $xy^2 z \notin L$ .

## 1.8 Pushdown automaton

## Motivation

Endliche Automaten haben nur endlichen Speicher, können also nicht mal zählen. Deshalb ein erweitertes Modell: Automat mit Stack als Speicher.

## Definition

Ein nicht deterministischer *Kellerautomat* besteht aus einem 6-Tupel  $(Q, \Sigma, \Gamma, \delta, q_0, F)$ , wobei gilt:

- $Q$  endliche Zustandsmenge
- $\Sigma$  Eingabealphabet
- $\Gamma$  Stack-Alphabet
- $q_0 \in Q$  Startzustand
- $F \subset Q$  Menge der akzeptierenden Zustände
- Übergangsfunktion:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$$

## Beispiel

$$L = \{ww^R \mid w \text{ Wort über } \Sigma\}$$

wobei  $w^R$  das Wort  $w$  rückwärts ist, kann von einem (nicht-deterministischen) Kellerautomaten erkannt werden.

## Beispiel

$$L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$$

kann von einem Kellerautomaten *nicht* erkannt werden.

## Bemerkung

Die Sprachen, die von einem nicht-deterministischen Kellerautomaten erkannt werden können, heißen *kontextfreie Sprachen*. Bei deterministischen Kellerautomaten heißen sie entsprechend *deterministische kontextfreie Sprachen*.

## 1.9 Grammatiken

Nicht wirklich relevant.

## Bemerkung

- Backus-Naur Schreibweise
- Chomsky-Hierarchie

## 2 Turingmaschinen

## Motivation

Ein allgemeineres Modell eines Computers:

- kann eine Eingabe lesen
- hat beliebig viel Speicherplatz
- kann Dinge an beliebigen Stellen in den Speicher schreiben/lesen
- kann beliebig viele Rechenschritte machen

## Beispiel

Betrachte die Sprache  $L = \{w\#w \mid w \in \{0,1\}^*\}$ .

- lies den ersten Buchstaben und merke
- überschreibe mit Symbol  $x \notin \{0,1,\#\}$
- nach rechts bis  $\#$  erscheint
- vergleiche nächsten ( $\neq y$ ) Buchstabe mit gemerkten
- falls gleich:
  - überschreibe mit  $y \notin \{0,1,\#,x\}$
  - gehe zurück bis  $x$
  - wiederhole

## Definition

Eine *Turingmaschine (TM)* ist ein 7-Tupel  $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ :

- $Q$  ist eine endliche Menge von *Zuständen*
- $\Sigma$  ist eine endliche Menge, das *Eingabealphabet*
- $\Gamma$  ist eine endliche Menge, das *Arbeitsalphabet*, mit  $\Sigma \subset \Gamma$  und einem Leerzeichen  $\sqcup$
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$  die *Übergangsfunktion*
- $q_0 \in Q$  der *Startzustand*
- $q_{\text{accept}} \in Q$  der *akzeptierende Endzustand*
- $q_{\text{reject}} \in Q$ ,  $q_{\text{reject}} \neq q_{\text{accept}}$  der *verwerfende Endzustand*

## Bemerkung

- Es gibt genau einen akzeptierenden und verwerfenden Zustand
- Die TM beendet ihre Berechnung, sobald sie einen dieser beiden Zustände erreicht
- Das Band der TM hat “ein linkes Ende”, nach rechts ist es unbeschränkt

## Beispiel

TM, die  $L = \{0^{2^n} \mid n \in \mathbb{N}_0\}$  erkennt.

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}}\}$
- $\Sigma = \{0\}$
- $\Gamma = \{0, x, \sqcup\}$
- $\delta = \text{tja?}$

## Definition

Eine *Konfiguration* der TM  $M$  wird beschrieben durch den Inhalt des Bandes, die Position des Lesekopfes und den derzeitigen Zustand  $q \in Q$ :

$$uqv$$

- Inhalt des Speicherbandes ist String  $uv$
- Position des Schreibkopfes ist direkt nach  $u$ , auf dem ersten Buchstaben von  $v$
- Zustand ist  $q$

Außerdem:

- Die *Startkonfiguration* von  $M$  auf Eingabe  $w$  ist  $q_0w$
- Eine Konfiguration heißt *akzeptierend/verwerfend*, wenn der Zustand  $q_{\text{accept}}/q_{\text{reject}}$  ist

## Definition

Eine *Berechnung* der TM  $M$  auf Eingabe  $w$  ist eine gültige Folge von Konfigurationen  $C_0, C_1, C_2, \dots$ , sodass  $C_0$  die Startkonfiguration ist und die Konfiguration  $C_{i+1}$  jeweils in der Übergangsfunktion beschrieben aus  $C_i$  hervorgeht.

Eine Berechnung einer TM auf Eingabe  $w$  heißt *akzeptierend/verwerfend*, falls sie im Zustand  $q_{\text{accept}}/q_{\text{reject}}$  endet.

Eine Berechnung heißt *nicht-akzeptierend*, falls sie entweder in  $q_{\text{reject}}$  endet oder *nie beendet wird*.

## Definition

Eine Sprache  $L$  heißt *rekursiv aufzählbar (semi-entscheidbar)*, falls es eine TM  $M$  gibt, die  $L$  akzeptiert. Das heißt:

- $w \in L \implies M$  akzeptiert  $w$
- $w \notin L \implies M$  verwirft oder hält nicht an

## Definition

Eine Sprache  $L$  heißt *(rekursiv) entscheidbar*, falls es eine TM  $M$  gibt, sodass gilt:

- $w \in L \implies M$  akzeptiert  $w$
- $w \notin L \implies M$  verwirft  $w$

**$M$  hält immer an.**

## 2.1 Varianten

### Definition

Zwei Turing-Maschinen  $M_1, M_2$  heißen *äquivalent*, falls sie die gleichen Sprachen akzeptieren:  $L(M_1) = L(M_2)$ .

$$\begin{aligned} M_1 \text{ akzeptiert } w &\implies M_2 \text{ akzeptiert } w \\ M_1 \text{ akzeptiert } w \text{ nicht} &\implies M_2 \text{ akzeptiert } w \text{ nicht} \end{aligned}$$

### Definition

Wir können wie bei endlichen Automaten eine *nicht-deterministische Turingmaschine (NTM)* definieren:

- Zu jedem Zeitpunkt hat die TM mehrere Möglichkeiten, wie sie fortfahren kann
- Formal geht dann die Übergangsfunktion

$$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

### Definition

Eine *Berechnung* der NTM entspricht einem möglichen Pfad im “Baum der möglichen Berechnungen”.

Eine Berechnung heißt *akzeptierend/verwerfend*, falls sie in einem *akzeptierenden/verwerfenden* Zustand endet.

Die von der NTM *akzeptierte Sprache* besteht aus den Wörtern, für die es eine akzeptierende Berechnung gibt: Mindestens einer der Pfade im Berechnungsbaum endet im akzeptierenden Zustand.

### Bemerkung

- bei DTMs: nicht akzeptierend  $\iff$  verwerfen oder nicht terminieren
- bei NTMs: nicht akzeptierend  $\iff \forall$  Pfade: Pfad verwirft oder endet nicht

### Satz

$$\text{DTM} \underbrace{\equiv}_{\text{berechenbarkeitsäquivalent}} \text{NTM}$$

#### Beweis

Breitensuche im Berechnungsbaum. Blabla offensichtlich.

## 3 Entscheidbarkeit von Sprachen vs. Berechenbarkeit von Funktionen

### Definition

Eine Funktion  $f : \Sigma^* \rightarrow \Gamma^*$  heißt *(Turing)-berechenbar* oder *totalrekursiv*, falls es eine TM gibt, die bei Eingabe von  $w \in \Sigma^*$  den Funktionswert  $f(w)$  ausgibt (und insbesondere anhält).

## Satz

Eine Sprache  $L \subset \Sigma^*$  ist genau dann *entscheidbar*, wenn ihre charakteristische Funktion

$$\mathbb{1}_L : \Sigma^* \rightarrow \{0, 1\}, \mathbb{1}_L(w) = \begin{cases} 1 & w \in L \\ 0 & \text{sonst} \end{cases}$$

*berechenbar* ist.

## Beweis

- $L$  entscheidbar  $\implies \mathbb{1}_L$  berechenbar
  - TM  $M$ , die  $w$  genau dann akzeptiert, wenn  $w \in L$ . Erweitern zu  $\hat{M}$ :
    - \* falls  $M$  akzeptiert, schreibe 1 auf Band und lösche alles andere
    - \* falls  $M$  verwirft, schreibe 0 aufs Band
- $\mathbb{1}_L$  berechenbar  $\implies L$  entscheidbar
  - $\hat{M}$  berechnet  $\mathbb{1}_L$ 
    - \* TM gibt 1  $\implies$  akzeptierender Zustand
    - \* TM gibt 0  $\implies$  verworfender Zustand

Q.E.D.

## Satz

Eine Funktion  $f : \Sigma^* \rightarrow \Gamma^*$  ist *berechenbar* genau dann, wenn es eine TM gibt, die die folgende Sprache *entscheidet*:

$$L_f = \{w\#g \mid w \in \Sigma^*, g \in \Gamma^*, f(w) = g\}$$

## Beweis

- $f : \Sigma^* \rightarrow \Gamma^*$  berechenbar  $\implies L_f$  entscheidet
  - TM  $M_1$  berechnet bei  $w \in \Sigma^*$  Ausgabe von  $f(w)$
  - TM  $M_2$  bekommt  $w\#g$  und ruft  $M_1$  mit  $w$  auf
  - $M_2$  wartet auf Ergebnis  $g_1$  von  $M_1$
  - $M_2$  vergleicht  $g_1$  mit  $g$
- $L_f$  entscheidet  $\implies f : \Sigma^* \rightarrow \Gamma^*$  berechenbar
  - TM  $M_2$  entscheidet  $L_f$
  - TM  $M_1$  bekommt  $w$
  - $M_1$  probiert *alle* Antworten von  $f(w)$  aus bis die richtige Antwort gefunden wurde
  - sobald  $L_f$  akzeptiert, weiß  $M_1$  die Antwort

Q.E.D.

## 3.1 Gödelnummer

## Motivation

Programmierbare Turingmaschinen?

$\implies$  binäre Kodierung für TM?!



## Beispiel

Eine beispielhafte Kodierung einer TM  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$  mit  $\{0, 1, \#\}$ :

- $\Gamma = \{A_1, \dots, A_r\}$ :  $C(A_j) = 10^j 1$
- $Q = \{q_1, \dots, q_m\}$ :  $C(q_i) = 110^i 11$
- $C(L) = 1110111$  und  $C(R) = 11100111$
- $C(\delta(q, a) = (\hat{q}, \hat{a}, B)) = \#C(q)C(a)C(\hat{q})C(\hat{a})C(B)\#$

$\implies C(M) = \#0^m \#0^r \#C(t_1) \#C(t_2) \# \dots$  mit Transitionen  $t_i$ ,  $m = |Q|$  und  $r = |\Gamma|$ .

Ein reiner binärer String lässt sich durch  $0 \mapsto 00, 1 \mapsto 11, \# \mapsto 01$  bilden.

## Bemerkung

Die Kodierung ist

- injektiv:  $C(M_1) = C(M_2) \implies M_1 = M_2$ .
- *präfixfrei*

## Satz

Es gibt eine TM  $A_{\text{true}}$ , die für einen binären String  $w$  entscheidet, ob er eine gültige Kodierung einer TM ist.

## Definition

Seien  $x, y \in \{0, 1\}^+$  zwei binäre Strings.  $x \leq y$  falls  $n_x, n_y$  die durch die Strings repräsentiert werden,  $n_x \leq n_y$  repräsentieren.

Für  $x = \varepsilon, y \in \{0, 1\}^*$  gelte immer  $\varepsilon \leq y$ .

## Bemerkung

Erfüllt die Bedingungen einer *totalen Ordnung*: Transitiv, anti-symmetrisch und total.

## Definition

Sei  $x \in \{0, 1\}^*$ . Für  $i \in \mathbb{N}$  nennt man  $x$  die Kodierung der  $i$ -ten *Turingmaschine*, falls gilt:

- $x = C(M)$  für eine TM  $M$
- $\{y \in \{0, 1\}^* \mid y \leq x\}$  enthält genau  $i - 1$  Wörter, die Kodierungen von Turingmaschinen sind.

## Satz

Es gibt eine TM  $A$ , die für ein  $i \in \mathbb{N}$  die Kodierung der  $i$ -ten TM berechnet.

## Definition

**Informell:** Die natürliche Zahl (der binäre String), der eine Turingmaschine beschreibt, heißt die *Gödelnummer der TM*. Schreibweise  $\langle M \rangle$ .

**Formell:** Sei  $\mathcal{M}$  die Menge aller Turingmaschinen. Die *Gödelisierung* sei  $g : \mathcal{M} \rightarrow \mathbb{N}$ , falls gilt:

- $g$  ist injektiv
  - $g(\mathcal{M})$  ist entscheidbar (TM konstruierbar, die für jedes  $n \in \mathbb{N}$  entscheidet, ob  $n \in G(\mathcal{M})$  gilt)
  - $g : \mathcal{M} \rightarrow \mathbb{N}$  und  $g^{-1} : g(\mathcal{M}) \rightarrow \mathcal{M}$  sind berechenbar
- $g(M)$  heißt für  $M \in \mathcal{M}$  die *Gödelnummer* von  $M$ .

## 3.2 Die universelle Turingmaschine

## Motivation

Turingmaschinen können bis jetzt nur genau ein Programm ausführen, aber wir wollen mehr!! :((

## Beispiel

Eine Möglichkeit ist eine *3-Band-TM*:

- Auf Band 1 wird  $M$  simuliert
- Auf Band 2 wird die Gödelnummer  $\langle M \rangle$  geschrieben
- Auf Band 3 wird der aktuelle Zustand von  $M$  vermerkt

Vorbereitung:

- $U$  liest  $\langle M \rangle w$  auf Band 1 und teilt sie in  $\langle M \rangle$  und  $w$  auf
  - bricht ab, falls  $\langle M \rangle$  keine korrekte Kodierung ist
- $U$  kopiert  $\langle M \rangle$  auf Band 2 und löscht sie von Band 1
- $U$  schreibt die Kodierung des Startzustandes von  $M$  auf Band 3

Simulation:

- $U$  weiß mittels Band 3 den aktuellen Zustand
- $U$  liest den aktuellen Buchstaben von Band 1
- $U$  sucht auf Band 2 den zugehörigen Übergang
- $U$  führt Übergang auf Band 1 durch und merkt sich den neuen Zustand in Band 3

Ausgabe:

- $U$  stoppt, sobald Band 3 den akzeptierenden/verwerfenden Zustand erreicht
- Band 1 enthält die Ausgabe der Berechnung

## 3.3 Abzählbar unendliche Mengen

## Motivation

Es gibt **viel** mehr Sprachen als Turingmaschinen  $\implies$  es gibt Sprachen, die nicht von TMs erkannt werden können.

**Definition**

Eine Menge  $M \neq \emptyset$  heißt *endlich*, falls es ein  $n_0 \in \mathbb{N}$  gibt, für das es eine bijektive Abbildung

$$g : \{1, 2, \dots, n_0\} \rightarrow M$$

gibt. Andernfalls heißt  $M$  *unendlich*.

Im endlichen Fall bezeichnet man mit  $|M| := n_0$  die *Kardinalität/Mächtigkeit* der Menge.

**Definition**

Zwei Mengen  $M_1, M_2$  heißen *gleich mächtig*, falls es eine bijektive Abbildung  $M_1 \rightarrow M_2$  gibt.

$M_2$  heißt *mächtiger als*  $M_1$ , falls es eine injektive Abbildung  $f : M_1 \rightarrow M_2$  und keine injektive Abbildung  $g : M_2 \rightarrow M_1$  gibt.

**Definition**

Menge  $M$  heißt *abzählbar unendlich*, wenn sie gleich mächtig wie  $\mathbb{N}$  ist (es existiert Bijektion  $f : M \rightarrow \mathbb{N}$ ).

Menge  $M$  heißt *höchstens abzählbar*, wenn sie endlich oder abzählbar unendlich ist.

Eine Menge, die weder endlich noch abzählbar ist, heißt *überabzählbar*.

**Bemerkung**

Menge  $\mathbb{N}$  ist unendlich und abzählbar:

$$f : \mathbb{N} \rightarrow \mathbb{N}, n \mapsto n \text{ bijektiv}$$

**3.3.1 Spaß mit Abzählbarkeit****Motivation**

Oh nein mein Hotel hat unendlich viele Zimmer und alle sind besetzt!

Jetzt kommt noch ein Gast aber wo soll der hin??

Dann kommt noch ein Bus mit abzählbar vielen Leuten – was soll ich tun?!?

**Beispiel**

- $\mathbb{Z}$  ist abzählbar mit folgender Bijektion:

$\mathbb{N}$	1	2	3	4	5	6	7
$\mathbb{Z}$	0	1	-1	2	-2	3	-3

- $\mathbb{N}^2$  ist abzählbar mit cooler zickidizickzack Bijektion
- $\mathbb{Q}$  ist abzählbar mit noch coolerer zickidizickzack Bijektion

**Satz**

Eine Menge ist genau dann unendlich, wenn es eine echte Teilmenge  $U \subset M, M \neq U$  und eine injektive Abbildung  $f : M \rightarrow U$  gibt.

## Satz

Sei  $M$  eine beliebige unendliche Menge. Dann ist  $\mathbb{N}$  nicht mächtiger als  $M$ .

## Satz

- Sei  $A$  höchstens abzählbar,  $f : A \rightarrow B$  bijektiv. Dann ist auch  $B$  höchstens abzählbar.
- $M$  abzählbar,  $N \subset M$ . Dann ist  $N$  endlich oder abzählbar.
- Seien  $M_1, M_2, \dots$  abzählbare Mengen. Dann ist  $\bigcup_{i \in \mathbb{N}} M_i$  auch abzählbar.
- Endliche Produkte von abzählbaren Mengen sind abzählbar:  $M_1, M_2, \dots, M_n$  abzählbar. Dann  $M_1 \times M_2 \times \dots \times M_n$  abzählbar

## Bemerkung

Gilt nicht für unendliche Produkte! Bspw.  $2^{\mathbb{N}}$  ist überabzählbar.

3.4 Wie groß ist  $\Sigma^*$ ?

## Satz

$\Sigma^*$  ist unendlich.

## Beweis

- $\Sigma$  enthält mindestens einen Buchstaben:  $a \in \Sigma$
- $\Sigma^*$  enthält demnach  $a, a^2, a^3, \dots$
- Also existiert injektive Abbildung  $f : \mathbb{N} \rightarrow \Sigma^*, f(u) = a^i$ , also muss  $\Sigma^*$  unendlich groß sein Q.E.D.

## Satz

$\Sigma^*$  ist abzählbar unendlich.

## Beweis

Bijektive Abbildung  $f : \mathbb{N} \rightarrow \Sigma^*$ :

$$\Sigma^* = \{\text{Wörter mit Länge } 0\} \cup \{\text{Wörter mit Länge } 1\} \cup \dots$$

- $f$  ist wohldefiniert: Jedes  $n \in \mathbb{N}$  erhält genau ein Bildwort  $f(n) \in \Sigma^*$  (klar)
- Surjektiv: Jedes Wort von  $\Sigma^*$  kriegt mindestens eine Nummer (klar)
- Injektiv: Jedes Wort von  $\Sigma^*$  kriegt genau eine Nummer (klar) Q.E.D.

## Satz

Sei  $L$  eine Sprache über einem endlichen Alphabet. Dann ist  $L$  höchstens abzählbar.

**Satz**

Die Menge aller Turingmaschinen ist abzählbar.

**Beweis**

TM  $M$  kann eindeutig durch GN  $\langle M \rangle \in \{0, 1\}^*$  beschrieben werden und  $\{0, 1\}^*$  ist abzählbar. Q.E.D.

**3.5 Überabzählbare Mengen****Satz**

Die Menge der reellen Zahlen ist überabzählbar.

**Beweis**

Über Cantorsches Diagonalisierungsverfahren. Trivial.

**Bemerkung**

- $\mathbb{N}$  abzählbar
- $\mathbb{Q}$  abzählbar
- $\mathbb{R}$  überabzählbar

**3.5.1  $2^{\mathbb{N}}$  ist überabzählbar****Satz**

Die Menge  $2^{\mathbb{N}} := \{0, 1\}^{\mathbb{N}} := \{(a_i)_{i \in \mathbb{N}} \mid a_i \in \{0, 1\}\}$  ist überabzählbar.

**Beweis**

Widerspruchsbeweis mit Cantor. Trivial.

**Bemerkung**

$2^{\mathbb{N}}$  und  $[0, 1] \subset \mathbb{R}$  haben Gemeinsamkeiten:

- Zahlen aus  $[0, 1]$  als Binärfolge darstellbar
- Erzeugt Bijektion zwischen  $2^{\mathbb{N}}$  und  $[0, 1]$  (außer periodische 1)

**3.5.2 Indikatorfunktion einer abzählbaren Menge**

TODO

**3.5.3 Mächtigkeit von Potenzmengen**

TODO

**3.5.4 Russels Paradox**

TODO

### 3.6 Sprachen, die nicht semi-entscheidbar sind

#### Satz

Betrachte das Alphabet  $\Sigma = \{0, 1\}$ . Es gibt Sprachen  $L$  über  $\Sigma$ , die nicht rekursiv aufzählbar sind.

#### Beweis

- Die Menge aller TM/GN ist abzählbar unendlich
- Ergo ist die Menge aller Sprachen, die von einer TM entschieden werden, auch höchstens abzählbar
- Die Menge aller Sprachen über  $\{0, 1\}$  ist überabzählbar
- Ergo muss es Sprachen geben, die nicht von einer TM entschieden werden können

Q.E.D.

#### Motivation

Existenz ist schön und gut, aber können wir tatsächlich eine  $D \notin \text{RE}$  konstruieren?

#### 3.6.1 Diagonalsprache

#### Definition

- Seien  $w_1, w_2, w_3, \dots$  alle Wörter über  $\Sigma = \{0, 1\}$  (abzählbar viele)
- Seien  $M_1, M_2, M_3, \dots$  alle TM (abzählbar viele) TODO: autoformat

/	$w_1$	$w_2$	$w_3$	$\dots$
$M_1$	$d_{11}$	$d_{12}$	$d_{13}$	$\dots$
$M_2$	$d_{21}$	$d_{22}$	$d_{23}$	$\dots$
$M_3$	$d_{31}$	$d_{32}$	$d_{33}$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

## Beispiel

/	$w_1$	$w_2$	$w_3$	...
$M_1$	1	0	0	...
$M_2$	0	1	0	...
$M_3$	1	1	0	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Diagonalsprache  $D = \{w_i \mid d_{ii} = 0\}$ :

- Enthält das Wort  $w_3$
- Enthält nicht die Wörter  $w_1, w_2$

Es gibt keine TM, die  $D$  erkennt:

- $D$  wird nicht von  $M_1$  erkannt
  - $M_1$  akzeptiert  $w_1$ . Aber da  $d_{11} = 1$  ist  $w_1 \notin D$
- $D$  wird nicht von  $M_2$  erkannt (analog)
- $D$  wird nicht von  $M_3$  erkannt
  - $M_3$  akzeptiert  $w_3$  nicht, da  $d_{33} = 0$ . Also würde  $M_3$  ein Wort aus  $D$  nicht akzeptieren
- usw.

## Satz

$D \in \text{RE}$ : Die Diagonalsprache ist nicht rekursiv aufzählbar.

## 3.6.2 Game of Life

## Definition

- Jede schwarze Zelle mit 2/3 schwarzen Nachbarn bleibt schwarz, alle anderen schwarzen Zellen werden weiß
- Jede weiße Zelle mit drei schwarzen Nachbarn wird schwarz, die anderen weißen Zellen werden weiß

## Satz

Das Problem, von einer Startkonfiguration eine bestimmte Zielkonfiguration zu erreichen, ist unentscheidbar.

## 3.7 Sprachen, die semi-entscheidbar, aber nicht entscheidbar sind

## Definition

$$A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ ist Code einer TM und } w \text{ wird von } M \text{ akzeptiert}\} \subset \{0, 1\}^*$$

mit  $\langle M, w \rangle$  Wort, welches die GN der TM  $M$  mit  $w$  konkateniert.

## Satz

$A_{\text{TM}}$  ist semi-entscheidbar:

$$A_{\text{TM}} \in \text{RE}$$

## Satz

$A_{\text{TM}}$  ist nicht entscheidbar:

$$A_{\text{TM}} \notin \text{R}$$

## 3.7.1 Komplementbildung

## Definition

Sei  $L \in \mathcal{L}$  eine Sprache über  $\Sigma$ . Die *Komplement-Sprache*  $L^C$  ist definiert als

$$L^C := \{w \in \Sigma^* \mid w \notin L\}.$$

Manchmal auch Notation  $\bar{L}$  statt  $L^C$ .

## Satz

$$L \in \text{R} \implies L^C \notin \text{R}.$$

## Definition

Die Menge *coRE* ist definiert als

$$\text{coRE} := \{L \mid L^C \in \text{RE}\}.$$

Oder anders aufgeschrieben:

$$L \in \text{coRE} :\iff L^C \in \text{RE}.$$

## Satz

$$L \in \text{R} \iff L \in \text{RE} \wedge L \in \text{coRE}$$

## Beweis

- $L \in \text{R} \implies L \in \text{RE} \wedge L \in \text{coRE}$ :
    - $L \in \text{R} \implies L \in \text{RE}$  klar, da  $\text{R} \subset \text{RE}$
    - Außerdem wie zuvor:  $L \in \text{R} \implies L^C \in \text{R} \subset \text{RE}$ .
  - $L \in \text{RE} \wedge L \in \text{coRE} \implies L \in \text{R}$ 
    - sei  $M$  TM für  $L$  und  $M_C$  TM für  $L^C$ . Neue TM  $\hat{M}$ :
      - \*  $M$  und  $M_C$  parallel auf  $w$  laufen lassen
      - \* falls  $M$  akzeptiert, soll  $\hat{M}$  akzeptieren
      - \* falls  $M_C$  akzeptiert, soll  $\hat{M}$  verwerfen
    - dann:  $w \in L \implies M$  akzeptiert  $\implies \hat{M}$  akzeptiert
    - sowie:  $w \notin L \implies w \in L^C \implies M_C$  akzeptiert  $\implies \hat{M}$  verwirft
- Q.E.D.

## Satz

$$L \in \text{RE} \not\implies L^C \in \text{RE}$$



## Motivation

Also gibt es nun weitere Sprachen  $\notin \text{RE}$ ?

## Satz

$\overline{A_{\text{TM}}} \notin \text{RE}$ .

## Beweis

Bereits bewiesen:  $L \in \text{RE} \wedge L \in \text{coRE} \implies L \in \text{R}$ .

Wissen außerdem, dass  $A_{\text{TM}} \in \text{RE}$  und  $A_{\text{TM}} \notin \text{R}$ . Also muss  $A_{\text{TM}} \notin \text{coRE}$  sein.  
Q.E.D.

TODO: Bildchen wichtig für Klausur. TODO: Bar.

### 3.8 Abbildungs-Reduktionen in der Berechenbarkeitstheorie

## Motivation

Wir wollen  $P_1$  lösen, indem wir es auf ein anderes Problem  $P_2$  reduzieren

- Falls  $P_2$  “leicht” ist, dann ist auch  $P_1$  “leicht”
- Falls  $P_1$  “schwer” ist, dann ist auch  $P_2$  “schwer”

## Definition

Sprache  $L_1 \subset \Sigma_1^*$  ist auf Sprache  $L_2 \subset \Sigma_2^*$  *Abbildungs-reduzierbar*, falls gilt:

Es gibt eine Turing-berechenbare Funktion  $f : \Sigma_1^* \rightarrow \Sigma_2^*$ , sodass für alle  $w \in \Sigma_1^*$  gilt:

$$w \in L_1 \iff f(w) \in L_2.$$

Die Funktion  $f$  heißt *Reduktion von  $L_1$  auf  $L_2$* .

Schreibweise:  $L_1 \preceq_m L_2$ , wobei das  $m$  oft für “mapping reduction” steht.

## Beispiel

TODO: Graphen

- Problem  $k$ -Clique: Gegeben ein ungerichteter Graph  $G = (V, E)$  und  $k \in \mathbb{N}$ . Eine Teilmenge  $V' \subset V$  heißt *Clique*, falls  $V'$  im Graph vollständig verbunden ist. Gibt es eine Clique der Größe  $k$  in  $G$ ?
- Problem  $k$ -Vertex-Cover: Gegeben ein ungerichteter Graph  $G = (V, E)$  und  $k \in \mathbb{N}$ . Eine Teilmenge  $V' \subset V$  heißt *Vertex Cover*, falls jede Kante des Graphen mindestens einen Endpunkt in  $V'$  hat. Gibt es im Graphen ein Vertex Cover mit  $k$  Knoten?

*Reduktion von Clique auf Vertex Cover.* Gegeben: Graph  $G$ ,  $k \in \mathbb{N}$ . Auf  $G$  wollen wir  $k$ -Clique lösen, indem wir Graphen  $\hat{G}$  bauen und  $\hat{k} \in \mathbb{N}$  so wählen, dass gilt:

$$G \text{ hat } k\text{-Clique} \iff \hat{G} \text{ hat } \hat{k}\text{-Vertex-Cover.}$$

Dazu wählen wir  $\hat{G}$  als das “Komplement” von  $G$ :

- $\hat{G}$  hat die gleichen Knoten wie  $G$
- $\hat{G}$  hat Kante  $uv$  genau dann, wenn  $G$  diese Kante **nicht** hat.

Außerdem setzen wir  $\hat{k} := |V| - k$ .

## Satz

Diese Reduktion ist berechenbar.

## Beweis

Die Reduktion  $f$  transformiert den Graphen  $G$  in den Graphen  $\hat{G}$ . Man kann dann eine TM konstruieren, die bei Eingabe von  $G$  und  $k$  die Ausgabe  $\hat{G}$  und  $\hat{k}$  produziert. Q.E.D.

## Satz

$G$  hat eine Clique der Größe  $k$  genau dann, wenn  $\hat{G}$  einen Vertex Cover der Größe  $\underbrace{n - k}_{:=\hat{k}}$  besitzt (wobei  $n = |V|$  ist).

## Beweis

TODO?

## Satz

Falls  $L_1 \preceq L_2$ , dann gilt:

1. Falls  $L_2$  (semi-)entscheidbar ist, dann ist auch  $L_1$  (semi-)entscheidbar.
2. Falls  $L_1$  nicht (semi-)entscheidbar ist, dann ist auch  $L_2$  nicht (semi-)entscheidbar.

## Beweis

TODO: Bildchen hihi

## 1. Beweis

- Sei  $L_2$  (semi-)entscheidbar, d.h. es gibt eine TM  $M_2$ , die die Sprache (semi-)entscheidet.
- Nun baut man TM  $M_1$ , die  $L_1$  (semi-)entscheidet: Bei der Eingabe von  $w$  wendet  $M_1$  zunächst die TM  $F$  an, die die Reduktion von  $L_1$  auf  $L_2$  realisiert. Auf die Ausgabe von  $F$  wird dann  $M_2$  angewandt.

## 2. Beweis durch Widerspruch

- Annahme:  $L_2$  ist (semi-)entscheidbar
- Mit Instanz von  $L_1$  starten und TM  $M_1$  wie zuvor
- Dann (semi-)entscheidet  $M_1$ , aber auch  $L_1$  :O

Q.E.D.

Andere Aussagen lassen sich nicht definitiv treffen.

3.9 Das Halteproblem und viele weitere Probleme in  $RE \setminus R$ 

## Motivation

Wir wollen eine Kette von Reduktionen zeigen:

$$\overline{D_{TM}} \preceq A_{TM} \preceq H \preceq H_0 \preceq K.$$

3.9.1 Reduktion  $\overline{D_{TM}} \preceq A_{TM}$ 

## Satz

- $\overline{D_{TM}} = \{\langle M \rangle \mid M \text{ akzeptiert } \langle M \rangle\}$
- $A_{TM} = \{\langle M, w \rangle \mid M \text{ akzeptiert } w\}$

$$\overline{D_{TM}} \preceq A_{TM}$$

## Beweis

Definiere  $f(w) := ww$ . Ist berechenbar und offensichtlich gilt:

$$w \in \overline{D_{TM}} \iff M \text{ akzeptiert } \langle M \rangle \iff f(w) = ww \in A_{TM}.$$

Q.E.D.

### 3.9.2 Reduktion $A_{\text{TM}} \preceq H$ (allgemeines Halteproblem)

#### Satz

- $A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ akzeptiert } w\}$
- $H = \{\langle M, w \rangle \mid M \text{ hält bei Eingabe } w \text{ an}\}$

$$A_{\text{TM}} \preceq H$$

#### Beweis

Mit Eingabe  $\langle M, w \rangle$  baut man eine TM  $\hat{M}$ :

- $\hat{M}$  macht die gleichen Berechnungen wie  $M$
- Falls  $M$  in einem nicht-akzeptierenden Zustand stoppt, dann begibt sich  $\hat{M}$  in eine Endlosschleife
- Sei  $\langle \hat{M} \rangle$  der Code dieser TM – dieser Code lässt sich nicht berechnen

Die Reduktionsabbildung ist:

$$f(\langle M, w \rangle) = \langle \hat{M}, w \rangle.$$

Dann gilt:

$$\begin{aligned} \langle M, w \rangle \in A_{\text{TM}} &\iff M \text{ akzeptiert } w \\ &\iff \hat{M} \text{ stoppt bei Eingabe } w \iff \langle \hat{M}, w \rangle \in H \end{aligned}$$

Q.E.D.

### 3.9.3 Reduktion $H \preceq H_0$ (spezielles Halteproblem)

#### Satz

- $H = \{\langle M, w \rangle \mid M \text{ hält bei Eingabe } w \text{ an}\}$
- $H_0 = \{\langle M \rangle \mid M \text{ hält bei Eingabe } \varepsilon \text{ an}\}$

$$H \preceq H_0$$

#### Beweis

TODO

### 3.9.4 Reduktion $H_0 \preceq K$

#### Satz

- $H_0 = \{\langle M \rangle \mid M \text{ hält bei Eingabe } \varepsilon \text{ an}\}$
- $K = \{\langle M \rangle \mid M \text{ hält bei Eingabe von } \langle M \rangle \text{ an}\}$

$$H_0 \preceq K$$

#### Beweis

TODO

### 3.9.5 Schlussfolgerung

#### Satz

Die Sprachen  $\overline{D_{TM}}, A_{TM}, H, H_0, K$  sind alle in  $RE \setminus R$ .

#### Motivation

- Das Halteproblem ist relevant, um die Beendung von Programmen zu zeigen – geht nicht! :(
- Berechnen zwei TM das Gleich? – geht nicht! :(

#### Definition

Problem 1  $\preceq_T$  Problem 2, falls es eine TM zur Lösung von Problem 1 gibt, die eine Turingmaschine für Problem 2 beliebig oft als “Unterprogramm” aufrufen darf.

## 4 Der Satz von Rice

#### Motivation

Viele Eigenschaften von Turingmaschinen/Programmen sind nicht entscheidbar. Sind diese Eigenschaften die Ausnahme?

Rice sagt: Nein! So gut wie alle *interessanten* Eigenschaften von TMs sind unentscheidbar. *Interessante* Eigenschaften sind jene, die sich nur an der von ihr erkannten Sprache festmachen lassen. Dabei interessiert uns nur die Ausgabe der TM, nicht die Art der Berechnung selbst.

#### Definition

Sei  $L$  eine Sprache, deren Wörter nur aus Codes von TMs bestehen:

$$L \subset \{w \in \{0,1\}^* \mid w \text{ ist Code einer TM}\}$$

Die Sprache  $L$  heißt *nicht-trivial*, falls gilt:

- $L \neq \emptyset$  (es gibt TMs, deren Code in  $L$  enthalten ist)
- $L \neq \{w \in \{0,1\}^* \mid w \text{ ist Code einer TM}\}$  (es gibt TMs, deren Code nicht in  $L$  enthalten ist)

#### Definition

Sei  $L$  eine Sprache, deren Wörter nur aus Codes von TMs bestehen:

$$L \subset \{w \in \{0,1\}^* \mid w \text{ ist Code einer TM}\}.$$

Die Sprache  $L$  heißt *semantisch*, falls gilt:

Wenn  $M_1$  und  $M_2$  äquivalente TMs sind, dann sind entweder beide in  $L$  enthalten oder beide nicht in  $L$  enthalten:

$$\langle M_1 \rangle \in L \iff \langle M_2 \rangle \in L$$

**Satz**

Jedes semantische, nichttriviale Entscheidungsproblem ist unentscheidbar. Wow!

**Beweis**

Siehe Vorlesung.

**Bemerkung**

Die Umkehrung des Satzes ist im Allgemeinen falsch, es gilt nur:

$$\text{semantisch} \implies \text{unentscheidbar}.$$

**Motivation**

Allgemeine “Verifikation” von Programmen ist unmöglich! Außerdem sind ohne Einschränkungen der TMs wichtige Eigenschaften nicht beweisbar!

Wir müssen die “Sorte von Programmen” einschränken, um sie verifizieren zu können.