Skript zur Vorlesung

Grundlagen der Theoretischen Informatik

Inhaltsverzeichnis

1 Sprachen und Grammatiken

Ein *Alphabet* ist eine endliche, nichtleere Menge. Die Elemente eines Alphabets heißen auch *Zeichen* oder *Symbole*.

Beispiel 1. $\{0, 1\}$, $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots, \mathsf{z}, \mathsf{A}, \ldots, \mathsf{Z}\}$

Wie üblich: Ist M eine Menge, so bezeichnet $|M|$ die Anzahl der Elemente von M.

Sei Σ ein Alphabet. Ein *Wort über* Σ ist eine Folge von Symbolen aus Σ. Ein Wort entsteht also durch *Hintereinanderschreiben* (*Konkatenation*) von Symbolen aus Σ.

Beispiel 2. $w = \mathsf{abbab}$ ist ein Wort über dem Alphabet $\{\mathsf{a}, \mathsf{b}\}$.

Spezialfall: leeres Wort bzw. leere Folge; Bezeichnung: ε

Die Menge aller Wörter über dem Alphabet Σ bezeichnen wir mit Σ^*.

Beispiel 3. Für $\Sigma = \{\mathsf{a}, \mathsf{b}\}$ ist $\Sigma^* = \{\varepsilon, \mathsf{a}, \mathsf{b}, \mathsf{aa}, \mathsf{ab}, \mathsf{ba}, \mathsf{bb}, \mathsf{aaa}, \mathsf{aab}, \mathsf{aba}, \ldots\}$.

Eine *Sprache über* Σ ist eine Menge von Wörtern über Σ, also eine Teilmenge von Σ^*.

Beispiel 4. Die Menge der deutschen Wörter ist eine Sprache über

$$\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots, \mathsf{z}, \mathsf{A}, \ldots, \mathsf{Z}, \ä, \ö, \ü, \ldots, \ß\}$$

Die Menge aller C-Programme ist eine Sprache über

$$\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots, \mathsf{z}, \mathsf{A}, \ldots, \mathsf{Z}, \{, \}, (,), [,], +, *, -, /, =, ., \_, \sqcup\}$$

Operation auf Wörtern: Konkatenation bzw. Hintereinanderschreiben. Schreibweise: $u \circ v$ oder kurz uv für Konkatenation der Wörter u und v.

Beispiel 5. Ist $u = \mathsf{ab}$ und $v = \mathsf{bab}$, so ist $u \circ v = \mathsf{abbab}$.

Die *Länge* eines Wortes w ist die Anzahl der Symbole in w. Schreibweise: $|w|$

Beispiel 6. $|\mathsf{aba}| = 3$, $|\varepsilon| = 0$, $|u \circ v| = |u| + |v|$

Für ein Wort w und $n \in \mathbb{N}$ ist w^n die Konkatenation $w^n = \underbrace{w \circ w \circ \cdots \circ w}_{\text{n-mal}}$.

Wir definieren: $w^0 = \varepsilon$. Es ist $|w^n| = n \cdot |w|$. *Schreibweise:* $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

Sprachen sind im Allgemeinen unendliche Objekte. Wir möchten sie auf endliche Weise beschreiben, zum Beispiel durch *Grammatiken*.

Beispiel 7 (für eine Grammatik).

⟨Satz⟩	→	⟨Subjekt⟩⟨Prädikat⟩⟨Objekt⟩
⟨Subjekt⟩	→	⟨Artikel⟩⟨Attribut⟩⟨Substantiv⟩
⟨Artikel⟩	→	der
⟨Artikel⟩	→	die
⟨Substantiv⟩	→	Hund
⟨Substantiv⟩	→	Katze
⟨Attribut⟩	→	⟨Adjektiv⟩
⟨Attribut⟩	→	⟨Adjektiv⟩⟨Attribut⟩
⟨Adjektiv⟩	→	kleine
⟨Adjektiv⟩	→	große
⟨Prädikat⟩	→	jagt

Ableitbare Sätze:

Der große Hund jagt die kleine Katze.

Es können unendlich viele Sätze abgeleitet werden:

Der große Hund jagt die kleine kleine … Katze.

Bestandteile einer Grammatik:

- Regeln ($\ldots \to \ldots$)
- Variablen ($\langle \ldots \rangle$)
- Startvariable (Startsymbol, ⟨Satz⟩)
- Eigentliche Symbole, die in abgeleiteten Sätzen vorkommen (sogenannte *Terminalsymbole*: der, die, Hund, …)

Formale Fassung:

Definition 8. Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei:

- V ist eine endliche Menge, die so genannte Menge der *Variablen*.
- Σ ist ein Alphabet, das so gennante *Terminalalphabet*, mit $V \cap \Sigma = \emptyset$.
- P ist die endliche Menge der *Produktionen*, $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$.
- $S \in V$ ist die so genannte *Startvariable*.

Zu den Produktionen: Eine Produktion $u \to v$, wobei u und v Folgen über $V \cup \Sigma$ sind, wird formal als Paar (u, v) aufgefasst.

Die Menge der Produktionen ist also eine Relation

$$P \quad \subseteq \quad \underbrace{(V \cup \Sigma)^+}_{\substack{\text{linke Regelseite,} \\ \text{nicht leeres Wort über } (V \cup \Sigma)}} \quad \times \quad \underbrace{(V \cup \Sigma)^*}_{\text{rechte Regelseite}}$$

Schreibweise: $u \to v \in P$ statt $(u, v) \in P$.

Definition 9. Sei $G = (V, \Sigma, P, S)$ eine Grammatik und seien $u, v \in (V \cup \Sigma)^*$. Wir definieren eine Relation $\Rightarrow_G$ wie folgt:

$u \Rightarrow_G v$, falls u, v zerlegt werden können in Teilwörter $u = xyz$ und $v = xy'z$ mit $x, z \in (V \cup \Sigma)^*$

und $y \to y'$ ist Regel in P.
„u geht unter (Anwendung einer Regel in) G unmittelbar über in v."

$u \Rightarrow_G^* v$, falls $u = v$ oder es Wörter $w_1, \ldots, w_k \in (V \cup \Sigma)^*$ gibt mit $u = w_1$, $w_i \Rightarrow_G w_{i+1}$ für $i = 1, 2, \ldots, k-1$ und $v = w_k$.

Wir lassen den Index G weg, falls dieser eindeutig ist.

Die von G *erzeugte Sprache* ist $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$

Eine *Ableitung* von $w \in L(G)$ in k Schritten ist eine Folge $(w_0, w_1, \ldots, w_k)$ mit $w_0 = S$, $w_k = w$ und $w_i \Rightarrow_G w_{i+1}$ für $i = 0, 1, \ldots, k-1$.

Beispiel 10. $G = (V, \Sigma, P, S)$ mit

$$\begin{array}{lcll} V & = & \{S, B, C\}, & \\ \Sigma & = & \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}, & \\ P & = & \{ \quad S \to \mathsf{a}SBC, & (1) \\ & & \quad S \to \mathsf{a}BC, & (2) \\ & & \quad CB \to BC, & (3) \\ & & \quad \mathsf{a}B \to \mathsf{ab}, & (4) \\ & & \quad \mathsf{b}B \to \mathsf{bb}, & (5) \\ & & \quad \mathsf{b}C \to \mathsf{bc}, & (6) \\ & & \quad \mathsf{c}C \to \mathsf{cc} \quad \} & (7) \end{array}$$

Beispiel für eine Ableitung:

$$\begin{array}{lllll} S & \overset{(1)}{\Longrightarrow} \mathsf{a}SBC & \overset{(1)}{\Longrightarrow} \mathsf{aa}SBCBC & \overset{(2)}{\Longrightarrow} \mathsf{aaa}BCBCBC & \\ & \overset{(3)}{\Longrightarrow} \mathsf{aaa}BBCCBC & \overset{(3)}{\Longrightarrow} \mathsf{aaa}BBCBCC & \overset{(3)}{\Longrightarrow} \mathsf{aaa}BBBCCC & \\ & \overset{(4)}{\Longrightarrow} \mathsf{aaab}BBCCC & \overset{(5)}{\Longrightarrow} \mathsf{aaabb}BCCC & \overset{(5)}{\Longrightarrow} \mathsf{aaabbb}CCC & \\ & \overset{(6)}{\Longrightarrow} \mathsf{aaabbbc}CC & \overset{(7)}{\Longrightarrow} \mathsf{aaabbbcc}C & \overset{(7)}{\Longrightarrow} \mathsf{aaabbbccc} & \end{array}$$

Behauptung $L(G) = \{\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n \mid n \geq 1\}$.

Beweis 11.

„$\supseteq$": Sei $n \geq 1$. Es gilt:

$$\begin{array}{lll} S & \Rightarrow^* \mathsf{a}^n(BC)^n & \text{(Regeln 1 und 2)} \\ & \Rightarrow^* \mathsf{a}^n B^n C^n & \text{(Regel 3)} \\ & \Rightarrow^* \mathsf{a}^n\mathsf{b}^n\mathsf{c}^n & \text{(Regeln 4 bis 7), also } \mathsf{a}^n\mathsf{b}^n\mathsf{c}^n \in L(G) \end{array}$$

„$\subseteq$": Sei $w \in L(G)$, also $S \Rightarrow^* w$. Es gilt: Anzahl von a's in w = Anzahl b's in w = Anzahl c's in w, da aufgrund der Form der Regeln in P in jedem einzelnen Schritt der Ableitung nur Wörter erzeugt werden können mit der Eigenschaft *Anzahl a's = Anzahl b's + Anzahl B's = Anzahl c's + Anzahl C's.*

Außerdem gilt: Alle a's stehen ganz links. Aufgrund der Regeln 4–7 müssen sich in einem Wort, das nur aus Terminalzeichen besteht, daran die b's und dann die c's anschließen. ■

2 Die Chomsky-Hierarchie

Noam Chomsky (Pionier der Linguistik) teilte Grammatiken in vier Typen ein.

Definition 12.

- Jede Grammatik ist vom *Typ 0* (d. h. keine Einschränkungen).
- Eine Grammatik ist vom *Typ 1* (oder: *kontextsensitiv*), falls für alle ihre Regeln $u \to v$ gilt: $|u| \leq |v|$.
- Eine Typ-1-Grammatik ist vom *Typ 2* (oder: *kontextfrei*), falls für alle ihre Regeln $u \to v$ gilt, dass u eine einzelne Variable ist (d. h. $u \in V$).
- Eine Typ-2-Grammatik ist vom *Typ 3* (oder: *regulär*), falls für alle ihre Regeln $u \to v$ gilt, dass v ein einzelnes Terminalzeichen ist ($v \in \Sigma$) oder v aus einem Terminalzeichen gefolgt von einer Variablen besteht.

Eine Sprache $L \subseteq \Sigma^*$ heißt vom Typ 0 (Typ 1, Typ 2, Typ 3), falls es eine Typ-0-Grammatik (Typ-1-Grammatik, Typ-2-Grammatik, Typ-3-Grammatik) G gibt mit $L = L(G)$.

Beispiel 13. Die Grammatik aus obigem Beispiel für $\{\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n \mid n \geq 1\}$ ist vom Typ 1.

Beispiel 14 (für eine kontextfreie Grammatik). $G = (V, \Sigma, P, S)$ mit $V = \{S\}$, $\Sigma = \{\mathsf{a}, \mathsf{b}\}$ und $P = \{S \to \mathsf{a}S\mathsf{b}, S \to \mathsf{ab}\}$. Es ist $L(G) = \{\mathsf{a}^n\mathsf{b}^n \mid n \geq 1\}$.

Zu den Namen „kontextfrei"/„kontextsensitiv": Bei einer kontextfreien Regel $A \to x$ kann in einer Ableitung stets (unabhängig vom Kontext) Variable A durch x ersetzt werden.
Bei kontextsensitiven Grammatiken kann man durch die Regel $uAv \to uxv$ erzwingen, dass diese Ersetzung nur im „Kontext" zwischen u und v erfolgen darf.

Spezialfall des leeren Wortes Das leere Wort kann bei Typ-1-, Typ-2- und Typ-3-Grammatiken nicht abgeleitet werden, d. h. es ist immer $\varepsilon \notin L(G)$ (wegen $|u| \leq |v|$ für $u \to v \in P$).

Deshalb: Abänderung obiger Definition um folgende *Sonderregelung*: Bei einer Grammatik (V, Σ, P, S) vom Typ 1, 2 oder 3 ist unabhängig von den oben genannten Restriktionen die Regel $S \to \varepsilon$ zugelassen.
Ist aber $S \to \varepsilon \in P$, so darf es keine Regel in P geben, in der S auf der rechten Seite vorkommt.[1]

Es gilt: Sprachen vom Typ 3 $\subseteq$ Sprachen vom Typ 2
$\subseteq$ Sprachen vom Typ 1
$\subseteq$ Sprachen vom Typ 0

Alle Inklusionen sind strikt (dazu später).

Frage: Kann man überhaupt feststellen, ob $w \in L(G)$, d. h. $S \Rightarrow^* w$?

Wortproblem für Typ-i-Grammatiken ($i = 0, 1, 2, 3$)

Gegeben: Grammatik G vom Typ i und ein Wort w.
Frage: Ist $w \in L(G)$?

[1] Dies ist keine Beschränkung der Allgemeinheit. Siehe hierzu auch „Theoretische Informatik – kurzgefasst", Uwe Schöning, Seite 18

Satz 15. Das Wortproblem für Typ-1-Sprachen ist *entscheidbar*, d. h. es gibt einen Algorithmus, der bei Eingabe einer kontextsensitiven Grammatik $G = (V, \Sigma, P, S)$ und eines Wortes $w \in \Sigma^*$ nach endlicher Zeit mit der Ausgabe „$w \in L(G)$“ oder „$w \notin L(G)$“ anhält.

Beweis 16. Für $m, n \in \mathbb{N}$, definiere

$$T_m^n = \left\{ w \in (V \cup \Sigma)^* \;\middle|\; \begin{array}{l} |w| \leq n \text{ und } w \text{ lässt sich aus } S \text{ in} \\ \text{höchstens } m \text{ Schritten ableiten} \end{array} \right\}$$

Für alle n gilt:

$$T_0^n = \{S\}$$
$$T_{m+1}^n = \mathrm{Abl}_n(T_m^n),$$

wobei $\mathrm{Abl}_n(X) = X \cup \{w \in (V \cup \Sigma)^* \mid |w| \leq n \text{ und } w' \Rightarrow_G w \text{ für ein } w' \in X\}$.

Es gilt:

- $T_0^n \subseteq T_1^n \subseteq T_2^n \subseteq \cdots$
- $T_m^n = T_{m+1}^n \Rightarrow T_m^n = T_k^n$ für alle $k > m$, also $T_m^n = \bigcup_{k \geq 0} T_k^n$
- Es gibt nur endlich viele Wörter der Länge $\leq n$ in $(V \cup \Sigma)^*$, also ist $T_m^n = \bigcup_{k \geq 0} T_k^n$ endlich.

Zusammengenommen: Für jedes n gibt es ein m mit $T_m^n = T_{m+1}^n = T_{m+2}^n = \cdots$. Falls $w \in L(G)$ und $|w| = n$ gilt, so folgt also: $w \in T_m^n$ für das obige m.

Algorithmus:

Eingabe: $G = (V, \Sigma, P, S)$, $w \in \Sigma^*$
$n := |w|$
$T := \{S\}$
Wiederhole
  $T_1 := T$
  $T := \mathrm{Abl}_n(T)$
solange, bis $w \in T$ oder $T = T_1$
Falls $w \in T$, *dann* Ausgabe „$w \in L(G)$“
sonst Ausgabe „$w \notin L(G)$“

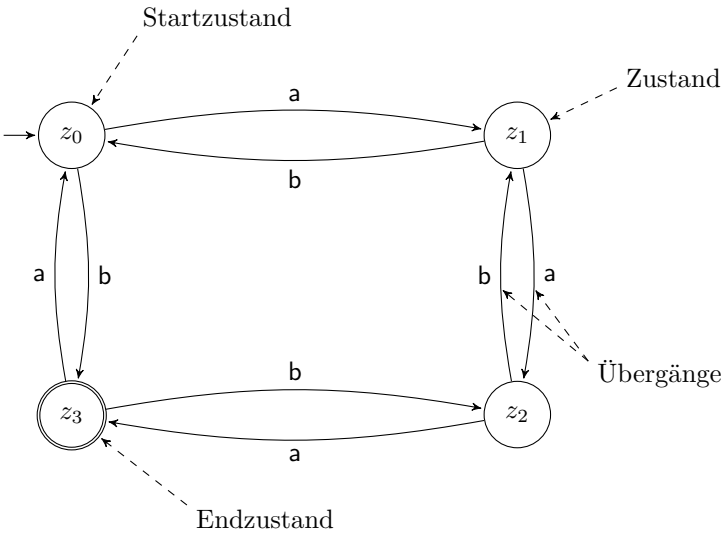
■

Bemerkung 17.

- Das Wortproblem für Typ-2- und Typ-3-Sprachen ist entscheidbar.
- Obiger Beweis funktioniert nicht für Typ-0-Sprachen, da hier nicht $T_{m+1}^n = \mathrm{Abl}_n(T_m^n)$ gilt. (Wort der Länge n kann aus längerem Wort durch verkürzende Regel entstehen.)
  Tatsächlich: Das Wortproblem für Typ-0-Sprachen ist nicht entscheidbar (dazu später).

3 Reguläre Sprachen

3.1 Endliche Automaten



Definition 18. Ein *(deterministischer) endlicher Automat* (kurz: DEA) ist ein 5-Tupel

$$M = (Z, \Sigma, \delta, z_0, E),$$

wobei für die einzelnen Komponenten gilt:

- Z ist eine endliche Menge, die so genannte *Zustandsmenge*,
- Σ ist ein Alphabet, das sogenannte *Eingabealphabet*, $Z \cap \Sigma = \emptyset$,
- $z_0 \in Z$ ist der so genannte *Startzustand*,
- $E \subseteq Z$ ist die Menge der so genannten *Endzustände*,
- $\delta\colon Z \times \Sigma \to Z$ ist die so genannte *Überführungsfunktion*.
  Bemerkung: δ ist total, d. h. für alle $z \in Z$ und $a \in \Sigma$ gibt es ein $z' \in Z$, so dass $\delta(z, a) = z'$.

Beispiel 19 (zum obigen Automaten). Für $M = (Z, \Sigma, \delta, z_0, E)$ gilt:

- $Z = \{z_0, z_1, z_2, z_3\}$
- $\Sigma = \{\mathsf{a}, \mathsf{b}\}$
- $E = \{z_3\}$
- $\delta(z, \mathsf{a}) = z'$, falls es eine mit a beschriftete Kante von z nach z' gibt und $\delta(z, \mathsf{b}) = z'$, falls es eine mit b beschriftete Kante von z nach z'. Also:

$$\begin{aligned} \delta(z_0, a) &= z_1, \\ \delta(z_0, b) &= z_3, \\ \delta(z_1, a) &= z_2, \text{ usw.} \end{aligned}$$

Der Automat ist anfangs im Zustand z_0. Bei Eingabe eines Wortes ändert sich Zeichen für Zeichen der Zustand. Ein Wort heißt *akzeptiert*, falls der Automat sich nach Eingabe aller Zeichen eines Wortes in einem Endzustand befindet.

Also gilt im obigen Beispiel:

$$\begin{aligned} w = \mathsf{aabaa} \quad & \text{wird akzeptiert (Zustand } z_3\text{)} \\ w = \mathsf{aabba} \quad & \text{wird nicht akzeptiert (Zustand } z_1\text{)} \end{aligned}$$

Formaler ausgedrückt gilt:

Definition 20. Sei $M = (Z, \Sigma, \delta, z_0, E)$ ein DEA. Die erweiterte Überführungsfunktion $\hat{\delta}\colon Z \times \Sigma^* \to Z$ ist (induktiv) definiert wie folgt:

$$\begin{aligned} \hat{\delta}(z, \varepsilon) &= z \text{ für alle } z \in Z \\ \hat{\delta}(z, ax) &= \hat{\delta}(\delta(z, a), x) \text{ für alle } z \in Z, a \in \Sigma \text{ und } x \in \Sigma^* \end{aligned}$$

Die von M akzeptierte Sprache ist

$$L(M) = \{x \in \Sigma^* \mid \hat{\delta}(z_0, x) \in E\}.$$

Beispiel 21. Für das obige Beispiel gilt:

$$\mathsf{aabaa} \in L(M) \text{ und } \mathsf{aabba} \notin L(M).$$

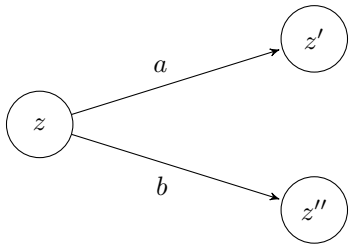
Allgemeiner gilt:

$$w \in L(M) \Leftrightarrow (\text{Anzahl } \mathsf{a}\text{'s in } w - \text{Anzahl } \mathsf{b}\text{'s in } w) \equiv 3 \mod 4.$$

Wobei gilt: $m \equiv n \mod l$ genau dann, wenn $m - n$ durch l teilbar ist.

3.2 Nichtdeterministische endliche Automaten

Von einem Zustand z aus dürfen mehrere (oder auch keine) mit demselben Buchstaben a markierte Pfeile ausgehen (siehe Grafik unten).



Ein nichtdeterministischer endlicher Automat hat bei Eingabe a im Zustand z mehrere Möglichkeiten. Bei Eingabe eines Wortes gibt es also im Allgemeinen mehrere Rechenwege/Zustandsfolgen. Ein Wort heißt *akzeptiert*, falls einer von diesen in einem Endzustand endet.

Definition 22. Ein *nichtdeterministischer endlicher Automat* (kurz: NEA) ist ein 5-Tupel

$$M = (Z, \Sigma, \delta, z_0, E),$$

wobei für die einzelnen Komponenten gilt:

- Z, Σ, z_0 und E sind wie bei deterministischen endlichen Automaten definiert.
- Für die Überführungsfunktion gilt: $\delta\colon Z \times \Sigma \to \mathcal{P}(Z)$.
  $\mathcal{P}(Z)$ ist die Potenzmenge von Z. Für $z \in Z$ und $a \in \Sigma$ ist also $\delta(z, a)$ eine *Menge* von möglichen Folgezuständen.

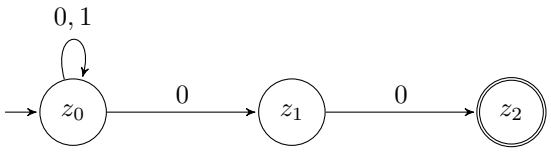
Wir definieren $\hat{\delta}\colon \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$ wie folgt:

$$\hat{\delta}(Z', \varepsilon) = Z' \text{ für alle } Z' \subseteq Z$$
$$\hat{\delta}(Z', ax) = \bigcup_{z \in Z'} \hat{\delta}(\delta(z, a), x) \text{ für alle } Z' \subseteq Z, a \in \Sigma \text{ und } x \in \Sigma^*.$$

Die von M akzeptierte Sprache ist

$$L(M) = \{x \in \Sigma^* \mid \hat{\delta}(\{z_0\}, x) \cap E \neq \emptyset\}.$$

Beispiel 23. Der folgende nichtdeterminischte Automat akzeptiert alle Wörter über $\{0, 1\}$, die mit 00 enden:



Satz 24. Zu jedem NEA M existiert ein DEA M' mit $L(M) = L(M')$.

Beweis 25 (Potenzmengenkonstruktion). Sei $M = (Z, \Sigma, \delta, z_0, E)$ ein NEA. Konstruiere einen DEA, der sich in seinen Zuständen merkt, welche verschiedenen Rechenwege der NEA durchlaufen könnte, genauer:

Setze $M' = (Z', \Sigma, \delta', Z_0', E')$, wobei:

$$Z' = \mathcal{P}(Z),$$
$$z_0' = \{z_0\},$$
$$\delta'(X, a) = \bigcup_{z \in X} \delta(z, a) \text{ für alle } X \subseteq Z,$$
$$E' = \{X \subseteq Z \mid X \cap E \neq \emptyset\}$$

Dann gilt für alle $w = a_1 a_2 \dots a_n \in \Sigma^*$:

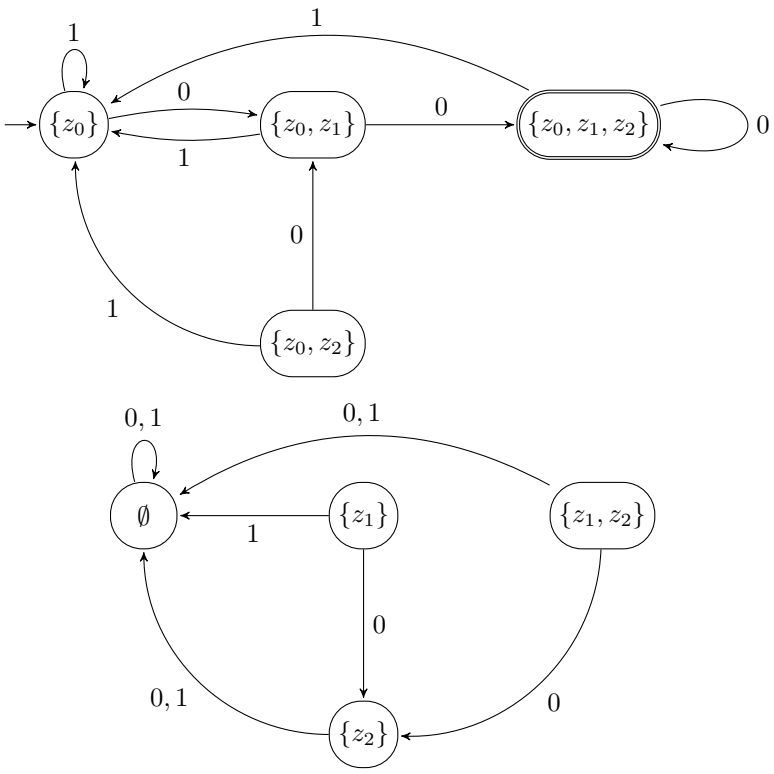
$$\begin{aligned}
w \in L(M) &\Leftrightarrow \hat{\delta}(\{z_0\}, w) \cap E \neq \emptyset \\
&\Leftrightarrow \text{es gibt eine Folge von Zustandsmengen } Z_1, \dots, Z_n \subseteq Z \text{ mit } \hat{\delta}(\{z_0\}, a_1) = Z_1, \hat{\delta}(Z_1, a_2) = Z_2, \dots, \hat{\delta}(Z_{n-1}, a_n) = Z_n \text{ und } Z_n \cap E \neq \emptyset \\
&\Leftrightarrow \text{es gibt } Z_1, \dots, Z_n \subseteq Z \text{ mit } \delta'(\{z_0\}, a_1) = Z_1, \delta'(Z_1, a_2) = Z_2, \dots, \delta'(Z_{n-1}, a_n) = Z_n \text{ und } Z_n \cap E \neq \emptyset \text{ (beachte } \delta'(X, a) = \hat{\delta}(X, a)) \\
&\Leftrightarrow \hat{\delta}'(\{z_0\}, w) \cap E \neq \emptyset \\
&\Leftrightarrow \hat{\delta}'(\{z_0\}, w) \in E' \\
&\Leftrightarrow w \in T(M')
\end{aligned}$$

■

Beispiel 26. M sei gewählt wie im vorherigen Beispiel. Definiere M' wie folgt:

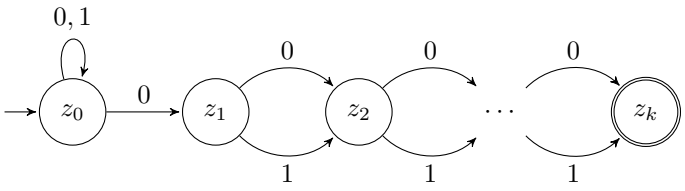
$$M' = (Z', \Sigma, \delta', z_0', E')$$

mit $Z' = \{\emptyset, \{z_0\}, \{z_1\}, \{z_2\}, \{z_0, z_1\}, \{z_0, z_2\}, \{z_1, z_2\}, \{z_0, z_1, z_2\}\}$ und $z_0' = \{z_0\}$.



Bemerkung Man kann alle Zustände bis auf auf $\{z_0\}$, $\{z_0, z_1\}$ und $\{z_0, z_1, z_2\}$ streichen.

Beispiel 27. $L_k = \{w \in \{0,1\}^* \mid |w| \geq k \text{ und der } k\text{-letzte Buchstabe von } w \text{ ist } 0\}$ mit $k \geq 0$.
Folgender NEA mit $k+1$ Zuständen akzeptiert L_k:



Man kann zeigen, dass es keinen DEA gibt, der L_k akzeptiert und weniger als 2^k Zustände besitzt.

3.3 Endliche Automaten und Typ-3-Grammatiken

Satz 28. Sei $L \subseteq \Sigma^*$ eine Sprache. Es gibt einen DEA M mit $L = L(M)$ genau dann, wenn es eine reguläre Grammatik G mit $L = L(G)$ gibt.

Beweis 29.

„⇒“: Sei $L = L(M)$ für den DEA $M = (Z, \Sigma, \delta, z_0, E)$. Definiere die Grammatik $G = (V, \Sigma, P, S)$ wie folgt:

$$V = \{A_z \mid z \in Z\}$$
$$S = A_{z_0}$$

P enthält folgende Regeln:

- Falls $\varepsilon \in L(M)$ (d. h. falls $z_0 \in E$) gilt, so ist $A_{z_0} \to \varepsilon \in P$.
- Falls $\delta(z_1, a) = z_2$ ein Übergang in M ist, so ist $A_{z_1} \to aA_{z_2} \in P$.
  Ist außerdem $z_2 \in E$, so ist zusätzlich $a_{z_1} \to a \in P$.

Dann gilt für alle $w = a_1 \dots a_n \in \Sigma^*$:

$$\begin{aligned} w \in L(M) &\Leftrightarrow \text{es gibt } z_1, \dots, z_n \in Z \text{ mit } Z_n \in E \\ &\quad \text{und } \delta(z_{i-1}, a_i) = z_i \text{ für } i = 1, \dots, n \\ &\Leftrightarrow \text{es gibt } z_1, \dots, z_n \in V \text{ mit} \\ &\quad z_0 \Rightarrow a_1 A_{z_1} \Rightarrow a_1 a_2 A_{z_2} \Rightarrow \dots \Rightarrow a_1 \dots a_{n-1} A_{z_{n-1}} \Rightarrow a_1 \dots a_n \\ &\Leftrightarrow w \in L(G) \end{aligned}$$

Wir müssen eventuell (falls $A_{z_0} \to \varepsilon \in P$) dafür sorgen, dass A_{z_0} auf keiner rechten Regelseite vorkommt (siehe Übungsaufgabe).

„⇐“: Sei $L = L(G)$ für $G = (V, \Sigma, P, S)$. Es reicht zu zeigen, dass ein NEA M mit $L = L(M)$ existiert. Definiere NEA $M = (Z, \Sigma, \delta, z_0, E)$ wie folgt:

$$Z = \{z_A \mid A \in V\} \cup \{z_+\}$$
$$z_0 = z_S$$
$$E = \begin{cases} \{z_S, z_+\}, & \text{falls } S \to \varepsilon \in P \\ \{z_+\}, & \text{falls } S \to \varepsilon \notin P \end{cases}$$
$$\delta(z_A, a) \ni \begin{cases} z_B, & \text{falls } A \to aB \in P \\ z_+, & \text{falls } A \to a \in P \end{cases}$$

Dann gilt für $w = a_1 \dots a_n \in \Sigma^*$ mit $n \geq 1$:

$$\begin{aligned} w \in L(G) &\Leftrightarrow \text{es gibt } A_1, \dots, A_{n-1} \in V \text{ mit} \\ &\quad S \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow a_1 \dots a_{n-1} A_{n-1} \Rightarrow a_1 \dots a_n \\ &\Leftrightarrow \text{es gibt } z_{A_1}, \dots, z_{A_{n-1}} \in Z \text{ mit} \\ &\quad \delta(z_S, a_1) \ni z_{A_1}, \delta(z_{A_1}, a_2) \ni z_{A_2}, \dots, \delta(z_{A_{n-1}}, a_n) \ni z_+ \\ &\Leftrightarrow w = a_1 \dots a_n \in L(M) \end{aligned}$$

■

3.4 Das Pumping-Lemma

Wichtigstes Hilfsmittel, um für eine Sprache nachzuweisen, dass sie nicht regulär ist:

Satz 30 (Pumping-Lemma, uvw-Theorem). Sei L eine reguläre Sprache. Dann gibt es eine Zahl n, sodass sich alle Wörter $x \in L$ mit $|x| \geq n$ zerlegen lassen in $x = uvw$, sodass folgende Eigenschaften gelten:

(1) $|v| \geq 1$

(2) $|uv| \leq n$

(3) Für alle $i \geq 0$ gilt: $uv^i w \in L$.

Beweis 31. Sei $M = (Z, \Sigma, \delta, z_0, E)$ ein DEA mit $L = L(M)$. Wähle $n = |Z|$. Sei nun $x \in L$ beliebig mit $|x| \geq n$, $x = a_1 a_2 \dots a_m$, $m \geq n$ und $a_1, \dots, a_m \in \Sigma$.

Definiere $z_i = \delta(z_{i-1}, a_i)$ für $i = 1, \dots, m$. Da $m+1 > |Z|$, gilt: In $z_0, z_1, \dots, z_m$ kommt mindestens ein Zustand zweimal vor. Sei z_l der erste solche Zustand, und sei $r > l$ minimal mit $z_l = z_r$.

Setze $u = a_1 \dots a_l$, $v = a_{l+1} \dots a_r$ und $w = a_{r+1} \dots a_m$ ($u = \epsilon$, falls $l = 0$ und $w = \epsilon$, falls $r = m$).

Dann gilt:

(1) $|v| \geq 1$, da $r > l$.

(2) $|uv| \leq n$, da l und r am weitesten links gewählt sind mit $z_l = z_r$.

(3) Da $z_l = z_r$ gilt, folgt: $\hat{\delta}(z_0, u) = \hat{\delta}(z_0, uv)$, also

$$\hat{\delta}(z_0, uvw) = \hat{\delta}(\hat{\delta}(z_0, uv), w) = \hat{\delta}(\hat{\delta}(z_0, u), w) = \hat{\delta}(z_0, uw), \text{ also } uw \in L.$$

Weiterhin gilt:

$$\hat{\delta}(z_0, uvvw) = \hat{\delta}(\hat{\delta}(z_0, uv), vw) = \hat{\delta}(\hat{\delta}(z_0, u), vw) = \hat{\delta}(z_0, uvw), \text{ also } uvvw = uv^2w \in L.$$

Analog: $uv^3w \in L$ und $uv^4w \in L$, …, allgemein: $uv^iw \in L$ für alle i.[2]

■

Logische Struktur der Aussage des Pumping-Lemmas:

$$(L \text{ regulär}) \Rightarrow \underbrace{(\exists n)(\forall x \in L, |x| \geq n)(\exists u, v, w)\ [x = uvw \text{ und } (1)\text{-}(3) \text{ gelten}]}_{\text{Aussage } (\star)}$$

Nach dem Pumping-Lemma gilt: „L regulär $\Rightarrow (\star)$“.
Die Umkehrung (d. h. „$(\star) \Rightarrow L$ regulär“) gilt im Allgemeinen nicht!

Aber: $(\star)$ gilt nicht $\Rightarrow L$ nicht regulär. In dieser Form wird das Pumping-Lemma meistens verwendet.

Beispiel 32. $L = \{\mathsf{a}^i\mathsf{b}^i \mid i \geq 1\}$ ist nicht regulär.

Angenommen L wäre regulär. Dann gibt es eine Zahl n, sodass sich alle $x \in L$ mit $|x| \geq n$ zerlegen lassen in $x = uvw$, sodass die Aussagen (1)–(3) des Pumping-Lemmas gelten.

Wir betrachten speziell $x = \mathsf{a}^n\mathsf{b}^n$, $|x| = 2n \geq n$. Man kann also x zerlegen in $x = uvw$, sodass $|v| \geq 1$ (wegen (1)) und $|uv| \leq n$ (wegen (2)), d. h. v besteht nur aus a's.

Dann gilt: uv^2w hat mehr a's als b's, also $uv^2w \notin L$ und das ist ein Widerspruch.

[2] Siehe hierzu auch J. Hopcroft, R. Motwani, J. Ullman „Einführung in die Automatentheorie, Formale Sprache und Komplexitätstheorie“, Seite 136 f.

Beispiel 33. $L = \{0^m \mid m \text{ ist eine Quadratzahl}\}$ ist nicht regulär.

Angenommen L wäre regulär. Dann existiert eine Zahl n wie im Pumping-Lemma. Betrachte nun $x = 0^{n^2}$, $|x| = n^2 \geq n$. Man kann x also zerlegen in $x = uvw$ mit (1)–(3) wie im Pumping-Lemma. Es gilt dann: $1 \leq |v| \leq |uv| \leq n$ (wegen (1) und (2)) und $uv^2w \in L$

Aber: $n^2 = |x| = |uvw| < |uv^2w| \leq n^2 + n < n^2 + 2n + 1 = (n+1)^2$, also gilt: $|uv^2w|$ liegt echt zwischen den benachbarten Quadratzahlen n^2 und $(n+1)^2$; $|uv^2w|$ kann also keine Quadratzahl sein, also $uv^2w \notin L$ und das ist ein Widerspruch.

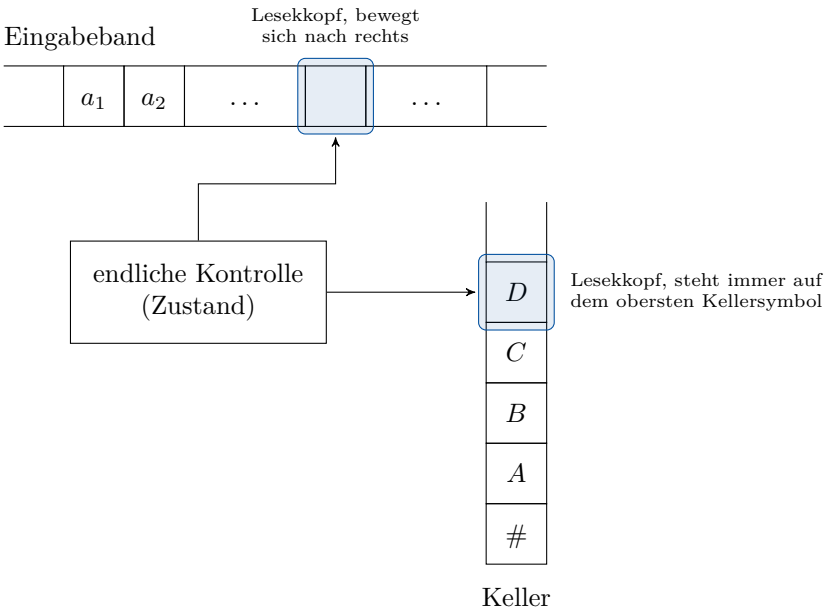
4 Kontextfreie Sprachen

4.1 Kellerautomaten

Endliche Automaten haben keinen Speicher. Ein endlicher Automat kann daher die Sprache $\{\mathsf{a}^n\mathsf{b}^n \mid n \geq 0\}$ nicht akzeptieren, da er beim Lesen des ersten b's nicht mehr „weiß“, wieviele a's er gelesen hat.

Die einzige gespeicherte Information ist der momentane Zustand, wobei es jedoch nur endlich viele Zustände gibt. DEAs und NEAs können daher nur eine endliche Information speichern.

Wir erweitern nun dieses Modell um einen unbeschränkten Speicher, auf den aber nur in eingeschränkter Form zugegriffen werden kann.



In Abhängigkeit vom

(1) aktuellen Zustand der endlichen Kontrolle

(2) gelesenen Zeichen auf dem Eingabeband

(3) gelesenen obersten Symbol im Keller

führt der Automat folgende Aktionen durch:

(4) Die endliche Kontrolle wechselt in einen neuen Zustand.

(5) Das oberste Kellersymbol wird durch eine Folge von Kellersymbolen ersetzt.

Bei endlichen Automaten besteht ein Schritt nur aus (1), (2) und (3).

Definition 34. Ein *(nichtdeterministischer) Kellerautomat* (NKA, Pushdown Automat (PDA)) ist ein 7-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E),$$

wobei für die einzelnen Komponenten gilt:

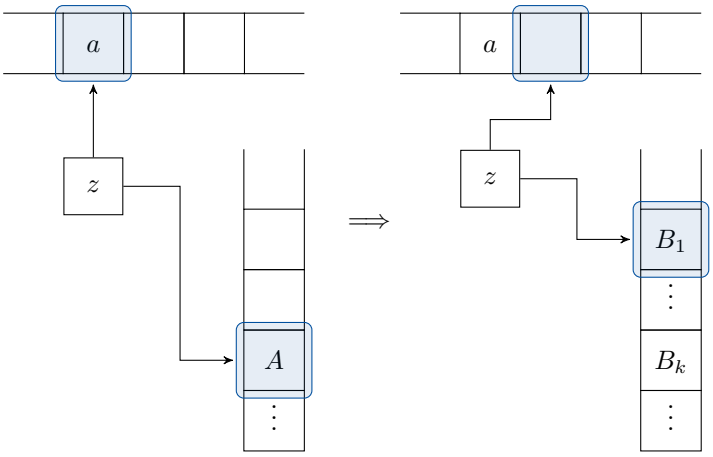
- Z ist die endliche Menge der *Zustände*,

- Σ ist das *Eingabealphabet*,
- Γ ist das *Kelleralphabet*,
- $\delta\colon Z \times \Sigma \times \Gamma \to \mathcal{P}(Z \times \Gamma^*)$ ist die *Überführungsfunktion*. Es gilt: $\delta(z, a, A)$ ist endlich für alle $z \in Z$, $a \in \Sigma$ und $A \in \Gamma$.
- $z_0 \in Z$ ist der *Startzustand*,
- $\# \in \Gamma$ ist das *unterste Kellersymbol*,
- $E \subseteq Z$ ist die Menge der *Endzustände*.

Intuitive Erläuterung der Arbeitsweise M befindet sich am Anfang im Zustand z_0. Der Eingabekopf steht auf dem ersten Zeichen der Eingabe. Der Keller enthält lediglich das Symbol $\#$.

$\delta(z, a, A) \ni (z', B_1, \ldots, B_k)$ (für $z, z' \in Z$, $a \in \Sigma$, $A, B_1, \ldots, B_k \in \Gamma$) bedeutet:

Ist M im Zustand z, liest das Eingabezeichen a und ist A das oberste Kellersymbol, so kann M in den Zustand z' übergehen und das Kellersymbol A durch die Symbole $B_1, \ldots, B_k$ (B_1 wird oberstes Kellersymbol) ersetzen. Der Eingabekopf wandert eine Position nach rechts.



Rechnung endet, falls

- die Eingabe ganz gelesen wurde sind
- oder keine Einträge in δ zur aktuellen Situation passen, z. B. dadurch, dass der Keller geleert wurde.

Ein Eingabewort wird *akzeptiert*, falls ein Zustand aus E angenommen wird, nachdem die Eingabe ganz gelesen wurde. Genauer: Falls es eine Folge von nichtdeterministischen Wahlmöglichkeiten gibt, sodass M einen Endzustand annimmt, nachdem die Eingabe ganz gelesen wurde.

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

Schreibweise: $zaA \to z'B_1 \ldots B_k$ statt $\delta(z, a, A) \ni (z', B_1, \ldots, B_k)$.

Beispiel 35. $L = \{\mathsf{a}^n\mathsf{b}^n \mid n \geq 1\}$. Es gilt $L = L(M)$ für

$$M = \big(\{z_0, z_1, z_2\}, \{\mathsf{a}, \mathsf{b}\}, \{\#, \mathsf{A}, \underline{\mathsf{A}}\}, \delta, z_0, \{z_2\}\big),$$

wobei gilt:

$$z_0\mathsf{a}\# \to z_0\underline{\mathsf{A}} \quad (1)$$
$$z_0\mathsf{a}\underline{\mathsf{A}} \to z_0\mathsf{A}\underline{\mathsf{A}} \quad (2)$$
$$z_0\mathsf{aA} \to z_0\mathsf{AA} \quad (3)$$

$$z_0\mathsf{bA} \to z_1\varepsilon \quad (4)$$
$$z_0\mathsf{b}\underline{\mathsf{A}} \to z_2\varepsilon \quad (5)$$

$$z_1\mathsf{bA} \to z_1\varepsilon \quad (6)$$
$$z_1\mathsf{b}\underline{\mathsf{A}} \to z_2\varepsilon \quad (7)$$

M arbeitet wie folgt auf Eingabe $w = \mathsf{aaabbb}$:

Zustand	Rest der Eingabe	Kellerinhalt	Befehl
z_0	aaabbb	#	(1)
z_0	aabbb	$\underline{\mathsf{A}}$	(2)
z_0	abbb	$\mathsf{A}\underline{\mathsf{A}}$	(3)
z_0	bbb	$\mathsf{AA}\underline{\mathsf{A}}$	(4)
z_1	bb	$\mathsf{A}\underline{\mathsf{A}}$	(6)
z_1	b	$\underline{\mathsf{A}}$	(7)
z_2	ε	ε	

Damit gilt also $\mathsf{aaabbb} \in L(M)$.

Für die Eingabe $w = \mathsf{aaabb}$ erhält man:

Zustand	Rest der Eingabe	Kellerinhalt	Befehl
z_0	aaabb	#	(1),(2),(3)
z_0	bb	$\mathsf{AA}\underline{\mathsf{A}}$	(4)
z_1	b	$\mathsf{A}\underline{\mathsf{A}}$	(6)
z_1	–	$\underline{\mathsf{A}}$	

An dieser Stelle ist die Eingabe ganz gelesen und kein Endzustand erreicht worden, also gilt: $\mathsf{aaabb} \notin L(M)$.

Für die Eingabe $w = \mathsf{abb}$ erhält man:

Zustand	Rest der Eingabe	Kellerinhalt	Befehl
z_0	abb	#	(1)
z_0	bb	$\underline{\mathsf{A}}$	(5)
z_2	b	ε	

An dieser Stelle ist kein weiterer Befehl möglich und die Eingabe ist noch nicht vollständig gelesen worden, also gilt: $\mathsf{abb} \notin L(M)$.

Beispiel 36. $L = \{w\$w^R \mid w \in \{\mathsf{a},\mathsf{b}\}^+\}$, wobei w^R = „w rückwärts“, also $w^R = a_n \dots a_2 a_1$, falls $w = a_1 a_2 \dots a_n$.

$L = L(M)$ für den NKA

$$M = \big(\{z_0, z_1, z_2\}, \{\mathsf{a}, \mathsf{b}, \$\}, \{\#, \mathsf{A}, \mathsf{B}, \underline{\mathsf{A}}, \underline{\mathsf{B}}\}, \delta, z_0, \{z_2\}\big),$$

wobei δ wie folgt definiert ist:

$z_0\mathsf{a}\# \to z_0\underline{\mathsf{A}}$	$z_0\mathsf{a}\underline{\mathsf{A}} \to z_0\mathsf{A}\underline{\mathsf{A}}$	$z_0\mathsf{a}\mathsf{A} \to z_0\mathsf{A}\mathsf{A}$	$z_0\mathsf{a}\underline{\mathsf{B}} \to z_0\mathsf{A}\underline{\mathsf{B}}$	$z_0\mathsf{a}\mathsf{B} \to z_0\mathsf{A}\mathsf{B}$
$z_0\mathsf{b}\# \to z_0\underline{\mathsf{B}}$	$z_0\mathsf{b}\underline{\mathsf{A}} \to z_0\mathsf{B}\underline{\mathsf{A}}$	$z_0\mathsf{b}\mathsf{A} \to z_0\mathsf{B}\mathsf{A}$	$z_0\mathsf{b}\underline{\mathsf{B}} \to z_0\mathsf{B}\underline{\mathsf{B}}$	$z_0\mathsf{b}\mathsf{B} \to z_0\mathsf{B}\mathsf{B}$
	$z_0\$\underline{\mathsf{A}} \to z_1\underline{\mathsf{A}}$	$z_0\$\mathsf{A} \to z_1\mathsf{A}$	$z_0\$\underline{\mathsf{B}} \to z_1\mathsf{B}$	$z_0\$\mathsf{B} \to z_1\mathsf{B}$
	$z_1\mathsf{a}\underline{\mathsf{A}} \to z_2\varepsilon$	$z_1\$\mathsf{A} \to z_1\varepsilon$		
			$z_1\mathsf{b}\underline{\mathsf{B}} \to z_2\varepsilon$	$z_1\mathsf{b}\mathsf{B} \to z_1\varepsilon$

M arbeitet wie folgt auf der Eingabe $w = \mathsf{ab\$ba}$:

Zustand	Rest der Eingabe	Kellerinhalt
z_0	ab\$ba	$\#$
z_0	b\$ba	$\underline{\mathsf{A}}$
z_0	\$ba	$\mathsf{B}\underline{\mathsf{A}}$
z_1	ba	$\mathsf{B}\underline{\mathsf{A}}$
z_1	a	$\underline{\mathsf{A}}$
z_2	ε	ε

Also ist $\mathsf{ab\$ba} \in L(M)$.

Bei Eingabe $\mathsf{ab\$bb}$ arbeitet M wie folgt:

Zustand	Rest der Eingabe	Kellerinhalt
z_0	ab\$bb	$\#$
z_0	b\$bb	$\underline{\mathsf{A}}$
z_0	\$bb	$\mathsf{B}\underline{\mathsf{A}}$
z_1	bb	$\mathsf{B}\underline{\mathsf{A}}$
z_1	b	$\underline{\mathsf{A}}$

keine weitere Bewegung möglich

Also ist $\mathsf{ab\$bb} \notin L(M)$.

Beispiel 37. $L = \{ww^R \mid w \in \{\mathsf{a}, \mathsf{b}\}^+\}$

Im vorherigen Beispiel ist die dritte Zeile zu ersetzen durch:

$$z_0\mathsf{a}\mathsf{A} \to z_1\varepsilon \qquad z_0\mathsf{b}\mathsf{B} \to z_1\varepsilon \qquad z_0\mathsf{a}\underline{\mathsf{A}} \to z_2\varepsilon \qquad z_0\mathsf{b}\underline{\mathsf{B}} \to z_2\varepsilon$$

Satz 38. Eine Sprache L ist kontextfrei genau dann, wenn es einen NKA M gibt mit $L = L(M)$.

Beweis 39. Wird in der Vorlesung „Formale Sprachen“ gebracht.[3]

[3] Siehe hierzu auch J. Hopcroft, R. Motwani, J. Ullman. „Einführung in die Automatentheorie, Formale Sprachen und Komplextätstheorie“ S. 248 ff.

Bemerkung 40. Es gibt auch deterministische Kellerautomaten (DKA). Obiger Satz gilt nicht für deterministische Kellerautomaten.[3]

Zum Beispiel kann die kontextfreie Sprache $\{ww^R \mid w \in \{\mathsf{a},\mathsf{b}\}^+\}$ von keinem DKA akzeptiert werden. DKAen akzeptieren also nur eine echte Teilmenge der kontextfreien Sprachen.

4.2 Das Pumping-Lemma für kontextfreie Sprachen

Wichtigstes Hilfsmittel, um nachzuweisen, dass eine Sprache *nicht* kontextfrei ist.

Satz 41 (Pumping-Lemma, $uvwxy$-Theorem). Sei L eine kontextfreie Sprache. Dann gibt es eine Zahl n, sodass sich alle Wörter $z \in L$ mit $|z| \geq n$ zerlegen lassen in $z = uvwxy$, sodass folgende Eigenschaften erfüllt sind:

(1) $|vx| \geq 1$

(2) $|vwx| \leq n$

(3) Für alle $i \geq 0$ gilt: $uv^iwx^iy \in L$

Struktur des Satzes:

$$(L \text{ kontextfrei}) \Rightarrow \underbrace{(\exists n \in \mathbb{N})(\forall z \in L, |z| \geq n)(\exists u,v,w,x,y)\ [z = uvwxy \wedge (1)\text{–}(3) \text{ gelten}]}_{(\star)}$$

Anwendung: Kontraposition des Satzes, also:

$$(\star) \text{ gilt nicht} \Rightarrow L \text{ ist nicht kontextfrei}^4$$

Beispiel 42. $L = \{\mathsf{a}^i\mathsf{b}^i\mathsf{c}^i \mid i \geq 1\}$ ist nicht kontextfrei.

Angenommen L wäre kontextfrei. Dann gibt es eine Zahl n, sodass sich alle $z \in L$ mit $|z| \geq n$ zerlegen lassen in $z = uvwxy$, sodass die Aussagen (1)–(3) des Pumping-Lemmas gelten.

Wir betrachten speziell $z = \mathsf{a}^n\mathsf{b}^n\mathsf{c}^n$, $|z| = 3n \geq n$. Man kann also z zerlegen in $z = uvwxy$, sodass

- $|vx| \geq 1$
- $|vwx| \leq n$, d. h. vwx kann höchstens zwei der 3 Buchstaben a, b oder c enthalten.

$$z = \boxed{\mathsf{a}\ldots\mathsf{a}}\boxed{\mathsf{b}\ldots\mathsf{b}}\boxed{\mathsf{c}\ldots\mathsf{c}} = \mathsf{a}^n\mathsf{b}^n\mathsf{c}^n$$
$$uwx = \boxed{\phantom{\mathsf{a}\ldots\mathsf{a}}}$$

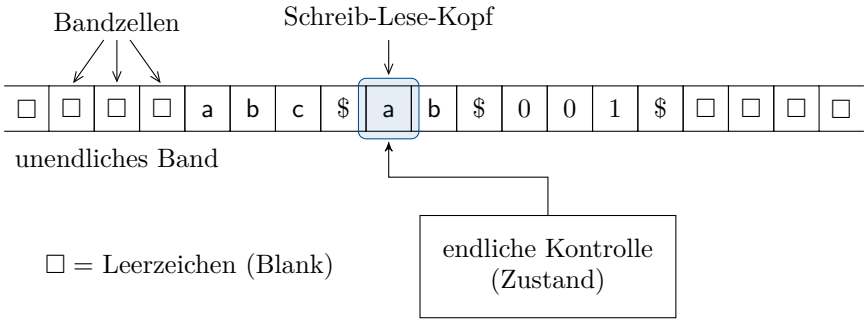
Dann gilt: In uv^2wx^2y ist die Anzahl der a's, b's und c's nicht gleich. Also ist $uv^2wx^2y \notin L$ und das ist ein Widerspruch zur Annahme.

[4] Siehe hierzu U. Schöning „Theoretische Informatik kurz gefasst“ Seite 57 ff.

5 Typ-1- und Typ-0-Sprachen

Kellerautomaten haben unbegrenzten Speicher, auf den sie aber nur eingeschränkt zugreifen können. Verallgemeinerung: Hilfsspeicher an beliebiger Stelle lesbar und modifizierbar.

Turingmaschine:



In Abhängigkeit vom

(1) aktuellen Zustand der endlichen Kontrolle

(2) gelesenen Zeichen auf dem Eingabeband

führt die Maschine folgende Aktionen durch:

(1) Sie wechselt in einen neuen Zustand.

(2) Sie ersetzt das gelesene Zeichen durch ein neues Zeichen.

(3) Sie bewegt den Schreib-Lese-Kopf.

Definition 43. Eine *Turingmaschine* (TM) ist ein 7-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E),$$

wobei für die einzelnen Komponenten gilt:

- Z ist die Menge der *Zustände*,
- Σ ist das *Eingabealphabet*,
- $\Gamma \supset \Sigma$ ist das *Arbeitsalphabet*,
- $z_0 \in Z$ ist der *Startzustand*,
- $\square \in \Gamma \setminus \Sigma$ ist das *Leerzeichen* bzw. *Blank*
- $E \subseteq Z$ ist die Menge der Endzustände.
- δ ist die *Übergangsfunktion*.

  Bei deterministischen Turingmaschinen (DTM, TM) gilt:

  $$\delta\colon Z \times \Gamma \to Z \times \Gamma \times \{\mathrm{L}, \mathrm{N}, \mathrm{R}\}$$

  Bei nichtdeterministischen Turingmaschinen (NTM) gilt:

  $$\delta\colon Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{\mathrm{L}, \mathrm{N}, \mathrm{R}\})$$

Informelle Arbeitsweise: $\delta(z,a) = (z',b,X)$ und $X \in \{\mathrm{L},\mathrm{N},\mathrm{R}\}$ bedeutet:

Befindet sich M im Zustand z und liest der Schreib-Lese-Kopf das Zeichen a, so geht M in den Zustand z' über, ersetzt das Zeichen a durch b und bewegt den Kopf

$$\begin{cases} \text{nach rechts,} & \text{falls } X = \mathrm{R} \\ \text{nach links,} & \text{falls } X = \mathrm{L} \\ \text{nicht,} & \text{falls } X = \mathrm{N} \text{ (neutral)} \end{cases}$$

Für eine NTM gilt: $\delta(z,a) \ni (z',bX)$ bedeutet:

Befindet sich M im Zustand z und liest der Schreib-Lese-Kopf das Zeichen a, so kann M in den Zustand z' übergehen, ersetzt das Zeichen a durch b und bewegt den Kopf

$$\begin{cases} \text{nach rechts,} & \text{falls } x = \mathrm{R} \\ \text{nach links,} & \text{falls } x = \mathrm{L} \\ \text{nicht,} & \text{falls } x = \mathrm{N} \text{ (neutral)} \end{cases}$$

M hält an, sobald ein Zustand aus E angenommen wird.

Wir fassen als Nächstes die Arbeitsweise einer Turingmaschine formaler:

Definition 44. Eine *Konfiguration* einer TM $M = (Z,\Sigma,\Gamma,\delta,z_0,\square,E)$ ist ein Wort $k = uzv$, wobei $u,v \in \Gamma^*$ und $z \in Z$.

Eine Konfiguration entspricht einer „Momentanaufnahme“ der TM, also der vollständigen Beschreibung ihrer Situation zu einem Zeitpunkt.

$k = uzv$ bedeutet:

- M ist im Zustand z.
- Der Teil des Bandes, der die Eingabe enthält oder bisher vom Kopf von M besucht wurde, ist uv.
- Der Kopf steht auf dem ersten Zeichen von v.

Start der Arbeitsweise:

- Die Kontrolle im Zustand z_0.
- Auf dem Band befindet sich ein Wort x.
- Der Kopf steht auf dem ersten Zeichen von x.

Formal: Die *Startkonfiguration* von M bei Eingabe x ist z_0x.

Seien k_1 und k_2 Konfigurationen.
Frage: Wann geht k_2 aus k_1 durch einen Rechenschritt von M hervor (in Zeichen: $k_1 \vdash k_2$)?

Definition 45. Sei $M = (Z,\Sigma,\Gamma,\delta,z_0,\square,E)$ eine TM. Wir definieren eine zweistellige Relation $\vdash$ auf der Menge der Konfigurationen wie folgt für $z \in Z \setminus E$:

$$a_1 \dots a_m z b_1 \dots b_n = \begin{cases} a_1 \dots a_m z' c b_2 \dots b_n, & \text{falls } \delta(z,b_1) = (z',c,\mathrm{N}),\ m \geq 0,\ n \geq 1 \\ a_1 \dots a_m c z' b_2 \dots b_n, & \text{falls } \delta(z,b_1) = (z',c,\mathrm{R}),\ m \geq 0,\ n \geq 2 \\ a_1 \dots z' a_m c b_2 \dots b_n, & \text{falls } \delta(z,b_1) = (z',c,\mathrm{L}),\ m \geq 1,\ n \geq 1 \end{cases}$$

Sonderfälle:

- $n = 1$, Maschine läuft nach rechts:

$$a_1 \dots a_m z b_1 \vdash a_1 \dots a_m c z' \square, \quad \text{falls } \delta(z, b_1) = (z', c, \mathrm{R}),\ m \geq 0$$

- $m = 0$, Maschine läuft nach links:

$$z b_1 \dots b_n \vdash z' \square c b_2 \dots b_n, \quad \text{falls } \delta(z, b_1) = (z', c, \mathrm{L}),\ n \geq 1$$

Für $z \in E$ gibt es keine Konfiguration k mit

$$a_1 \dots a_m z b_1 \dots b_n \vdash k.$$

Für NTM M gilt: Überall „$\delta(z, b_1) = (\dots)$“ ersetzen durch „$\delta(z, b_1) \ni (\dots)$“.

Also gilt:

- Ist M eine DTM und k eine Konfiguration, so gibt es höchstens eine Konfiguration k' mit $k \vdash k'$.
- Ist M eine NTM und k eine Konfiguration, so gibt es eine beliebige endliche Anzahl von k' mit $k \vdash k'$.

Konfigurationen verlängern sich, falls der Kopf eine noch nicht besuchte Bandzelle betritt.

Schreibweise: Statt $\delta(z, a) = (z', b, x)$ oder $\delta(z, a) \ni (z', b, x)$ kurz: $za \to z'bx$.

Beispiel 46. Folgende TM addiert zu einer auf dem Band befindlichen Zahl x in Binärdarstellung ($x \in \{0,1\}^*$) die Zahl 1 und positioniert anschließend den Kopf auf das erste Zeichen der Zahl:

$$M = (\{z_0, z_1, z_2, z_e\}, \{0,1\}, \{0,1,\square\}, \delta, z_0, \square, \{z_e\})$$

wobei:

$$\left.\begin{array}{lcl} z_0 0 & \to & z_0 0\mathrm{R} \\ z_0 1 & \to & z_0 1\mathrm{R} \\ z_0 \square & \to & z_1 \square\mathrm{L} \end{array}\right\} \text{Kopf ans rechte Eingabeende positionieren}$$

$$\left.\begin{array}{lcl} z_1 0 & \to & z_2 1\mathrm{L} \\ z_1 1 & \to & z_1 0\mathrm{L} \\ z_1 \square & \to & z_e 1\mathrm{N} \end{array}\right\} \begin{array}{l}\text{Nach links laufen bis zur ersten Null, dabei} \\ \text{alle 1 durch 0 ersetzen. Falls keine 0 gefunden} \\ \text{wird, links 1 anhängen und anhalten.}\end{array}$$

$$\left.\begin{array}{lcl} z_2 0 & \to & z_2 0\mathrm{L} \\ z_2 1 & \to & z_2 1\mathrm{L} \\ z_2 \square & \to & z_e \square\mathrm{R} \end{array}\right\} \text{Kopf ans linke Eingabeende positionieren}$$

Beispiel für eine Rechnung:

$$\begin{array}{lllllllll} z_0 101 & \vdash & 1 z_0 01 & \vdash & 10 z_0 1 & \vdash & 101 z_0 \square & \vdash & 10 z_1 1 \square \quad 1 z_1 00 \square \\ & \vdash & z_2 110 \square & \vdash & z_2 \square 110 \square & \vdash & \square z_e 110 \square & & \end{array}$$

Definition 47. Die von einer Turingmaschine $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ *akzeptierte Sprache* ist

$$L(M) = \{w \in \Sigma^* \mid z_0 w \vdash^* uzv \text{ für ein } z \in E \text{ und } u, v \in \Gamma^*\}.$$

Dabei ist $k_a \vdash k_e$, falls $k_a = k_e$ oder es $k_1, \ldots, k_n$ gibt mit

$$k_a \vdash k_1 \vdash \cdots \vdash k_n \vdash k_e.$$

Also: Ein Wort wird akzeptiert, falls *irgendwann* ein Endzustand angenommen wird.

Definition 48. Ein *linear-beschränkter Automat* (LBA) ist eine NTM

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$$

mit folgenden Eingenschaften:

- $\Gamma \setminus \Sigma$ enthält zwei spezielle Symbole $\triangleright$ und $\triangleleft$, die so genannte *linke* bzw. *rechte Bandendemarkierung*.
- Falls M $\triangleright$ liest, ist keine Kopfbewegung nach links erlaubt.
- Falls M $\triangleleft$ liest, ist keine Kopfbewegung nach rechts erlaubt.
- Die Bandsymbole $\triangleright$ und $\triangleleft$ dürfen nicht durch andere Zeichen überschrieben werden.

Die von M *akzeptierte Sprache* ist

$$L(M) = \{w \in \Sigma^* \mid z_0 \triangleright w \triangleleft \vdash^* uzv \text{ für ein } z \in E \text{ und } u, v \in \Gamma^*\}.$$

Also: Ein Wort w heißt akzeptiert, falls bei Eingabe $\triangleright w \triangleleft$ die Maschine irgendwann einen Endzustand annimmt. Der Kopf bewegt sich niemals aus dem Bereich des Wortes $\triangleright w \triangleleft$ heraus.

Die Anzahl der besuchten Bandzellen ist also $|\triangleright w \triangleleft|$, also linear abhängig von der Eingabelänge, daher die Bezeichnung „LBA“.

Beispiel 49. Sei $L = \{\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n \mid n \geq 1\}$. Folgender LBA

$$M = (\{z_0, z_1, z_2, z_3, z_4, z_\mathsf{a}, z_\mathsf{b}, z_\mathsf{c}, z_L\}, \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}, \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \triangleleft, \triangleright, \square\}, \delta, z_0, \square, \{z_4\})$$

akzeptiert L, wobei die Überführungsfunktion δ wie folgt gegeben ist:

$$\left.\begin{array}{lcl} z_0\triangleright & \to & z_0 \triangleright \mathrm{R} \\ z_0\mathsf{a} & \to & z_\mathsf{a}\mathrm{aR} \\ z_\mathsf{a}\mathsf{a} & \to & z_\mathsf{a}\mathrm{aR} \\ z_\mathsf{a}\mathsf{b} & \to & z_\mathsf{b}\mathrm{bR} \\ z_\mathsf{b}\mathsf{b} & \to & z_\mathsf{b}\mathrm{bR} \\ z_\mathsf{b}\mathsf{c} & \to & z_\mathsf{c}\mathrm{cR} \\ z_\mathsf{c}\mathsf{c} & \to & z_\mathsf{c}\mathrm{cR} \\ z_\mathsf{c}\triangleleft & \to & z_L \triangleleft \mathrm{L} \end{array}\right\} \text{ Test, ob Eingabe die Form } a^+b^+c^+ \text{ hat}$$

$$\left.\begin{array}{lcl} z_L\mathrm{c} & \to & z_L\mathrm{cL} \\ z_L\mathrm{b} & \to & z_L\mathrm{bL} \\ z_L\mathrm{a} & \to & z_L\mathrm{aL} \\ z_L\square & \to & z_L\square\mathrm{L} \\ z_L\triangleright & \to & z_1 \triangleright \mathrm{R} \end{array}\right\} \text{ Durchlauf nach links}$$

$$\left.\begin{array}{lcl} z_1\square & \to & z_1\square\mathrm{R} \\ z_1\mathrm{a} & \to & z_2\square\mathrm{R} \\ z_1\triangleleft & \to & z_4 \triangleleft \mathrm{L} \end{array}\right\} \text{ Suche und Löschen des ersten } \mathsf{a}\text{s}$$

$$\left.\begin{array}{lcl} z_2\mathsf{a} & \to & z_2\mathsf{a}R \\ z_2\square & \to & z_2\square R \\ z_2\mathsf{b} & \to & z_3\square R \end{array}\right\} \text{Suche und Löschen des ersten } \mathsf{b}\text{s}$$

$$\left.\begin{array}{lcl} z_3\mathsf{b} & \to & z_3\mathsf{b}R \\ z_3\square & \to & z_3\square R \\ z_3\mathsf{c} & \to & z_L\square L \end{array}\right\} \text{Suche und Löschen des ersten } \mathsf{c}\text{s}$$

Begründung:

Sei w die Eingabe. Zuerst durchläuft M das Wort w von links nach rechts und testet, ob w die Form $\mathsf{a}^+\mathsf{b}^+\mathsf{c}^+$ hat. Ist dies nicht der Fall, so bleibt M stecken und akzeptiert nicht. Natürlich gilt dann auch $w \notin L$.

Gilt aber $w = \mathsf{a}^k\mathsf{b}^l\mathsf{c}^m$ mit $k, l, m \geq 1$ so läuft M wieder auf die linke Begrenzung. Ab jetzt ist die Arbeit von M in Durchgänge eingeteilt: In jedem Durchgang läuft M nach rechts und ersetzt jeweils ein a, b und c durch $\square$. Am Ende des Durchgangs läuft M dann wieder auf die linke Begrenzung.

Gilt nicht $k = l = m$, d. h. $w \notin L$, so bleibt M in Durchgang $1 + \min\{k, l, m\}$ stecken, weil er eines der Zeichen a, b oder c nicht mehr findet.

Gilt aber $k = l = m$, d. h. $w \in L$, so steht nach Durchgang k das Wort $\square^k\square^k\square^k$ zwischen den Begrenzungen. In Durchgang $k+1$ läuft dann M im Zustand z_1 komplett durch das Wort nach rechts bis zur rechten Begrenzung „$\triangleleft$“. Dann geht M in den akzeptierenden Zustand z_4 über.

Satz 50.

(1) Eine Sprache L ist kontextsensitiv (Typ 1) genau dann, wenn es einen LBA gibt mit $L(M) = L$.

(2) Eine Sprache L ist vom Typ 0 genau dann, wenn es eine TM M gibt mit $L(M) = L$ genau dann, wenn es eine NTM M gibt mit $L(M) = L$.

Beweis 51. In der Vorlesung „Formale Sprachen“. ■

Bemerkung 52. Es ist unbekannt, ob deterministische LBAs nicht schon die Klasse der Typ-1-Sprachen akzeptieren.

LBA-Problem: Gibt es für jede Typ-1-Sprache einen deterministischen LBA, der sie akzeptiert?

6 Der intuitive Berechenbarkeitsbegriff

Was ist berechenbar?
Wir schränken uns (zunächst) auf Funktionen über den natürlichen Zahlen ein.

Eine Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ heißt *berechenbar*, falls es ein Rechenverfahren bzw. einen Algorithmus gibt (z. B. Scheme-Programm, C++-Programm), das f berechnet, d. h. gestartet mit Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$ hält der Algorithmus nach endlich vielen Schritten mit Ausgabe $f(n_1, \ldots, n_k)$.

Wir fordern nicht, dass f total sein muss, d. h. für gewisse $(n_1, \ldots, n_k) \in \mathbb{N}^k$ darf $f(n_1, \ldots, n_k)$ undefiniert sein. In diesem Fall soll der Algorithmus nicht stoppen (Endlosschleife).

Beispiel 53. Addition, Multiplikation und die „gewöhnlichen“ zahlentheoretischen Funktionen sind berechenbar.

Beispiel 54.

$$f_1(n) = \begin{cases} 1, & \text{falls } n \text{ ein Anfangsabschnitt der Nachkommastellen von } \pi \text{ ist} \\ 0, & \text{sonst} \end{cases}$$

$f_1(1) = f_1(14) = f_1(141) = f_1(1415) = 1$, da $\pi = 3{,}1415\ldots$; $f(2) = 0$

f_1 ist berechenbar, da es Näherungsalgorithmen zur Berechnung von π gibt, die immer weitere Ziffern von π ausgeben.

Beispiel 55.

$$f_2(n) = \begin{cases} 1, & \text{falls } n \text{ irgendwo in den Nachkommastellen von } \pi \text{ vorkommt} \\ 0, & \text{sonst} \end{cases}$$

$f_2(141) = f_2(415) = 1$, da $\pi = 3{,}1415\ldots$. $f(1010101010) = ?$

Es ist nicht bekannt, ob f_2 berechenbar ist.

Beispiel 56.

$$f_3(n) = \begin{cases} 1, & \text{falls 7 in den Nachkommastellen von } \pi \text{ irgendwo} \\ & \text{mindestens } n\text{-mal hintereinander vorkommt} \\ 0, & \text{sonst} \end{cases}$$

Wir wissen nicht, wie lange Blöcke von „7“ in der Darstellung von π vorkommen. Trotzdem ist f_3 berechenbar, denn:

(1) Entweder kommen beliebig lange Blöcke von „7“ vor, also $f_3(n) = 1$ für alle n und f_3 berechenbar.

(2) oder es gibt eine maximale solche Blocklänge c, also:

$$f_3(n) = \begin{cases} 1, & \text{falls } n \leq c \\ 0, & \text{sonst} \end{cases}$$

Diese Funktion ist auch berechenbar.

Beispiel 57.

$$f_4(n) = \begin{cases} 1, & \text{falls die Antwort auf das LBA-Problem „ja“ ist} \\ 0, & \text{sonst} \end{cases}$$

f_4 ist die Konstant-1- oder Konstant-0-Funktion, also berechenbar.

f_3 und f_4 sind berechenbar, jedoch wissen wir nicht, wie der entsprechende Algorithmus aussieht.

Ziel: Präzisierung des Berechenbarkeitsbegriffs.
Nur so ist es möglich, zu beweisen, dass eine Funktion *nicht* berechenbar ist.

7 Berechenbarkeit durch Maschinen

7.1 Turing-Berechenbarkeit

Wir verwenden TMn nicht nur, um Sprachen zu akzeptieren, sondern auch, um Funktionen zu berechnen.

Definition 58. Eine Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ heißt *Turing-berechenbar*, falls es eine DTM M gibt, sodass für alle $n_1, \ldots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \ldots, n_k) = m \quad \Rightarrow \quad M \text{ mit Eingabe } \mathrm{bin}(n_1)\#\ldots\#\mathrm{bin}(n_k) \text{ hält mit } \square\cdots\square\,\mathrm{bin}(m)\square\cdots\square \text{ auf dem Arbeitsband}$$

$$f(n_1, \ldots, n_k) \text{ undefiniert} \quad \Rightarrow \quad M \text{ mit Eingabe } \mathrm{bin}(n_1)\#\,\mathrm{bin}(n_2)\#\ldots\#\,\mathrm{bin}(n_k) \text{ stoppt nicht}$$

$\mathrm{bin}(n)$ für $n \in \mathbb{N}$ bezeichnet die Binärdarstellung von n ohne führende Nullen.

Bemerkung 59. Das Eingabealphabet einer TM, die eine Funktion über $\mathbb{N}$ im obigen Sinne berechnet, ist stets $\{0, 1, \#\}$.

Definition 60. Eine Funktion $f\colon \Sigma^* \to \Delta^*$ heißt *Turing-berechenbar*, falls es DTM M gibt, sodass für alle $x \in \Sigma^*$ und $y \in \Delta^*$ gilt:

$$f(x) = y \quad \Rightarrow \quad M \text{ mit Eingabe } x \text{ hält mit } \square\cdots\square y\square\cdots\square \text{ auf dem Ausgabeband}$$

$$f(x) \text{ undefiniert} \quad \Rightarrow \quad M \text{ mit Eingabe } x \text{ stoppt nicht}$$

Beispiel 61. Die Nachfolgerfunktion $s\colon \mathbb{N} \to \mathbb{N}$, $s(n) = n + 1$ ist Turing-berechenbar.

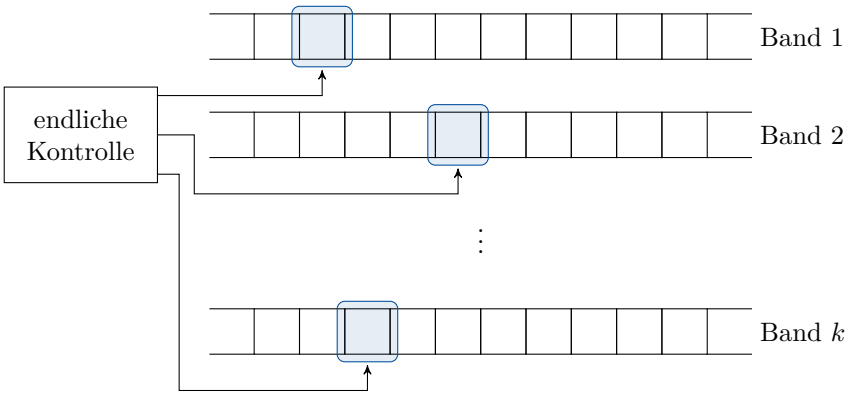
(Siehe Abschnitt 5.)

Beispiel 62. Die überall undefinierte Funktion Ω wird berechnet von folgender Turingmaschine: $z_0 a \to z_0 a\mathrm{N}$ für alle $a \in \Gamma$.

Beispiel 63. Die Funktion $f\colon \mathbb{N} \to \mathbb{N}$, $f(n) = 2n$ ist Turing-berechenbar vermöge folgender TM:

$$\begin{aligned} z_0 a &\to z_0 a\mathrm{R} \text{ für } a \in \{0, 1\} \\ z_0\square &\to z_1 0\mathrm{L} \\ z_1 a &\to z_1 a\mathrm{L} \text{ für } a \in \{0, 1\} \\ z_1\square &\to z_e\square\mathrm{R} \end{aligned}$$

7.2 Mehrband-Maschinen



Bei einer Mehrband-Turingmaschine hängt die durchzuführende Aktion vom aktuellen Zustand und den k gelesen Zeichen auf den k Arbeitsbändern ab.

Definition 64. Eine k-Band-DTM ist ein 7-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E),$$

wobei für die einzelnen Komponenten gilt:

- Z, Σ, Γ, z_0, $\square$ und E sind wie bei einer 1-Band-DTM definiert.
- $\delta\colon \underbrace{Z}_{(1)} \times \underbrace{\Gamma^k}_{(2)} \to \underbrace{Z}_{(3)} \times \underbrace{\Gamma^k}_{(4)} \times \underbrace{\{\mathrm{L}, \mathrm{R}, \mathrm{N}\}^k}_{(5)}$ mit
  - (1) aktueller Zustand
  - (2) gelesene Zeichen auf den k Bändern
  - (3) neuer Zustand
  - (4) geschriebene Zeichen auf den k Bändern
  - (5) Kopfbewegungen auf den k Bändern

Arbeitsweise: Die Eingabe steht zunächst auf Band 1. Die Bänder 2 bis k sind zunächst leer. Die Maschine führt einzelne Schritte durch, analog zu gewöhnlichen DTMn.

Akzeptierte Sprache: Das Eingabewort x wird genau dann akzeptiert, wenn M irgendwann einen Endzustand erreicht.

Berechnete Funktion: $f(n_1, \ldots, n_k) = m$ gdw. M mit Eingabe $\mathrm{bin}(n_1)\# \ldots \# \mathrm{bin}(n_k)$ erreicht irgendwann einen Endzustand mit $\mathrm{bin}(m)$ auf Band 1.
(Berechnung von Funktionen $f\colon \Sigma^* \to \Delta^*$ analog.)

Schreibweise: Statt $\delta(z, a_1, \ldots, a_k) = (z', b_1, \ldots, b_k, X_1, \ldots, X_k)$ kurz:

$$za_1 \ldots a_k \to z'b_1 \ldots b_k X_1 \ldots X_k$$

Beispiel 65. Folgende 2-Band-Turingmaschine akzeptiert $\{w\#w \mid w \in \{0,1\}^*\}$:

$$M = (\{z_0, z_1, z_2, z_e\}, \{0, 1, \#\}, \{0, 1, \#, \square\}, \delta, z_0, \square, \{z_e\}),$$

wobei für die Überführungsfunktion gilt:

$z_0 0\square$	$\to$	$z_0 00\mathrm{RR}$	0, 1 auf Band 1 werden auf Band 2 kopiert
$z_0 1\square$	$\to$	$z_0 11\mathrm{RR}$	
$z_0 \#\square$	$\to$	$z_1 \#\square\mathrm{NL}$	# auf Band 1 $\Rightarrow$ Zustand z_1
$z_0 \square\square$	$\to$	$z_2 \square\square\mathrm{NN}$	Endlosschleife, falls kein # gefunden wird
$z_1 \#0$	$\to$	$z_1 \#0\mathrm{NL}$	Kopf auf Band 2 nach links
$z_1 \#1$	$\to$	$z_1 \#1\mathrm{NL}$	Kopf auf Band 1 bleibt auf #
$z_1 \#\square$	$\to$	$z_2 \#\square\mathrm{RR}$	$\square$ auf Band 2 $\Rightarrow$ Zustand z_2
$z_2 00$	$\to$	$z_2 00\mathrm{RR}$	auf beiden Bändern nach rechts gehen,
$z_2 11$	$\to$	$z_2 11\mathrm{RR}$	solange gleiche Zeichen gefunden werden
$z_2 01$	$\to$	$z_2 01\mathrm{NN}$	verschiedene Zeichen $\Rightarrow$ Endlosschleife
$z_2 10$	$\to$	$z_2 10\mathrm{NN}$	
$z_2 \square\square$	$\to$	$z_e \square\square\mathrm{NN}$	Alles gleich, daher fertig
$z_2 0\square$	$\to$	$z_2 0\square\mathrm{NN}$	unterschiedliche Länge $\Rightarrow$ Endlosschleife
$z_2 1\square$	$\to$	$z_2 1\square\mathrm{NN}$	
$z_2 \square 0$	$\to$	$z_2 \square 0\mathrm{NN}$	
$z_2 \square 1$	$\to$	$z_2 \square 1\mathrm{NN}$	
$z_2 \#0$	$\to$	$z_2 \#0\mathrm{NN}$	Endlosschleife, falls zweites # gefunden wird
$z_2 \#1$	$\to$	$z_2 \#1\mathrm{NN}$	
$z_2 \#\square$	$\to$	$z_2 \#\square\mathrm{NN}$	

Andere Einträge in δ sind beliebig.

Beispiel für eine Rechnung:

Konfigurationen von k-Band-DTMn:

$$(u_1 z v_1, u_2 z v_2, \ldots, u_k z v_k)$$

mit $z \in Z$, $u_1 \ldots, u_k, v_1, \ldots, v_k \in \Gamma^*$. Bedeutung wie bei Konfigurationen von 1-Band-Turingmaschinen.

Eingabe $w = 101\#101$:

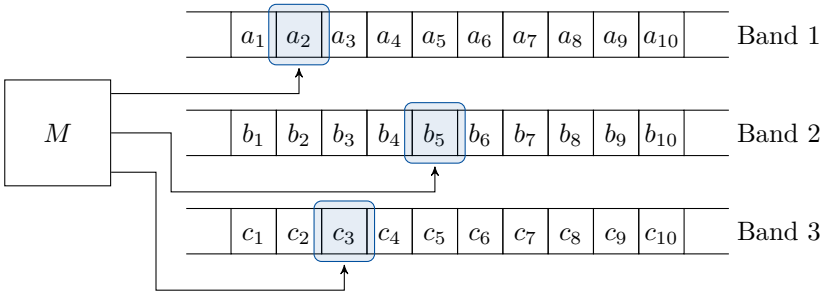
$$\begin{aligned}(z_0101\#101, z_0\square) &\vdash (1z_001\#101, 1z_0\square)\\ &\vdash (10z_01\#101, 10z_0\square)\\ &\vdash (101z_0\#101, 101z_0\square)\\ &\vdash (101z_1\#101, 10z_11\square)\\ &\vdash (101z_1\#101, 1z_101\square)\\ &\vdash (101z_1\#101, z_1101\square)\\ &\vdash (101z_1\#101, z_1\square101\square)\\ &\vdash (101\#z_2101, \square z_2101\square)\\ &\vdash (101\#1z_201, \square1z_201\square)\\ &\vdash (101\#10z_21, \square10z_21\square)\\ &\vdash (101\#101z_2\square, \square101z_2\square)\\ &\vdash (101\#101z_e\square, \square101z_e\square), \text{also } w \in L(M)\end{aligned}$$

Satz 66. Sei $k > 1$. Zu jeder k-Band-DTM M gibt es eine (1-Band-)DTM M', sodass $L(M) = L(M')$ bzw. dass M und M' dieselbe Funktion berechnen.

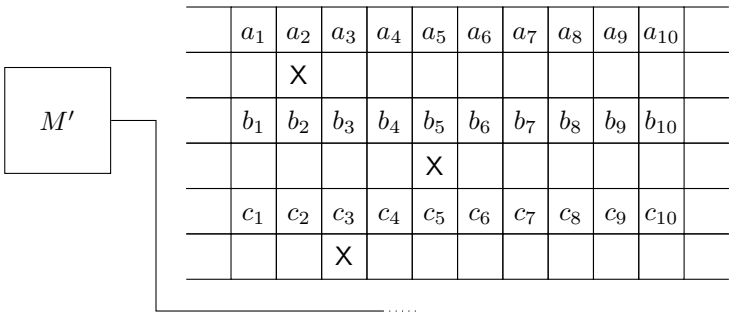
Beweis 67. Sei $M = (Q, \Sigma, \Gamma, \delta, z_0, \square, E)$. Wir unterteilen das Band von M' in $2k$ „Spuren“, in denen wir die Inhalte von k Bändern von M sowie die Position der k Köpfe von M speichern.

Zum Beispiel:

Situation von M ($k = 3$):



Entsprechende Situation von M':



M' hat 6 Spuren:

- Die Spuren 1, 3 und 5 entsprechen den Bänder von M
- Die Spuren 2, 4 und 6 entsprechen den Kopfposition der Bänder von M

Formal: Arbeitsalphabet Γ' von M' besteht neben den Zeichen aus Γ aus $2k$-Tupeln, deren Komponenten jeweils Zeichen von Γ oder die Markierung „X“ ($\mathrm{X} \notin \Gamma$) enthalten:

$$\Gamma' = \Gamma \cup (\Gamma \cup \{\mathrm{X}\}^{2k})$$

Arbeitsweise von M':

Eingabe $x = a_1 a_2 \dots a_n \in \Sigma^*$:

- M' erzeugt die Spurendarstellung der Startkonfiguration von M, also:

	a_1	a_2	a_3	$\cdots$						a_n	
	X										
	□	□	□	$\cdots$						□	
	X										

$\dots$ (weitere Spuren) $\dots$

- M' simuliert jeweils einen Schritt von M durch folgende Operationen:
  - M' bewegt den Kopf auf die linkeste Position, die eine Markierung „X“ enthält.
  - M' läuft nach rechts, bis alle Markierungen gefunden wurden und merkt sich dabei die Symbole aus Γ in den markierten Feldern.
  - M' sieht in δ nach, welche Aktion durchzuführen ist, bewegt den Kopf wieder nach ganz links und läuft sodann nach rechts, um die Bandinhalte und die Markierungen zu aktualisieren.
- Sobald M' bemerkt, dass M einen Endzustand annimmt, erzeugt M' aus der aktuellen Spurendarstellung einen Bandinhalt, der genau dem Inhalt der ersten Spur entspricht. M' bewegt den Kopf nach ganz links und hält an (nimmt Endzustand an). ■

7.3 Zusammensetzung von Turingmaschinen

7.3.1 1-Band nach k-Band

Sei M eine 1-Band-TM. Dann bezeichnet $M(i,k)$ $(1 \le i \le k)$, die k-Band-TM, die auf Band i genau die Aktion ausführt, die M auf seinem Band ausführt, und die Bänder $1, \dots, i-1, i+1, \dots, k$ unverändert lässt.

Ist also z. B. in M $\delta(z,a) = (z', b, X)$ mit $X \in \{\mathrm{L}, \mathrm{N}, \mathrm{R}\}$, so ergibt sich für $M(2,4)$:

$$\delta(z, c_1, a, c_3, c_4) = (z', c_1, b, c_3, c_4, \mathrm{N}, X, \mathrm{N}, \mathrm{N})$$

für alle c_1, c_3 und c_4 aus dem Arbeitsalphabet von M (= Arbeitsalpahbet von $M(2,4)$).

Schreibweise: $M(i)$ statt $M(i,k)$, falls k aus dem Kontext klar.

7.3.2 Einige spezielle Maschinen

Die Maschinen aus Abschnitt 5, die zu einer Zahl (in Binärdarstellung) 1 addiert, bezeichnen wir mit „Band := Band + 1“.

Statt „Band := Band + 1 (i)“ schreiben wir: „Band i := Band i + 1“.

Analog konstruieren wir Maschinen „Band i := Band $i - 1$“ (hier: $0 - 1 = 0$), „Band i := 0“ und „Band i := Band j“.

7.3.3 Hintereinanderschaltung von Turingmaschinen

Seien $M_i = (Z_i, \Sigma, \Gamma_i, \delta_i, z_{0,i}, \square, E_i)$ mit $i = 1, 2$ zwei DTMn mit o. B. d. A. $Z_1 \cap Z_2 = \emptyset$.

Wir definieren daraus die neue Turingmaschine

$$M = (Z_1 \cup Z_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, z_{0,1}, \square, E_2),$$

wobei:

$$\delta(z,a) = \begin{cases} \delta_1(z,a), & \text{falls } z \in Z_1 \setminus E_1 \text{ und } a \in \Gamma_1 \\ \delta_2(z,a), & \text{falls } z \in Z_2 \text{ und } a \in \Gamma_2 \\ (z_{0,2}, a, \mathrm{N}), & \text{falls } z \in E_1 \text{ und } a \in \Gamma_1 \end{cases}$$

Bezeichnungen für M: „M_1;M_2“ oder Start $\to M_1 \to M_2 \to$ Stopp. Dies lässt sich analog definieren für mehr als zwei Maschinen.

Beispiel 68.

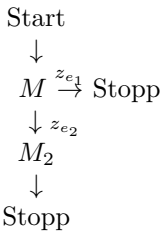
Start
↓
Band := Band + 1
↓
Band := Band + 1
↓
Band := Band + 1
↓
Stopp

Dies ist eine Turingmaschine, die zum Bandinhalt 3 addiert.

Analog:

Start
↓
$M \overset{z_{e_1}}{\to} M_1 \to$ Stopp
$\downarrow z_{e_2}$
M_2
↓
Stopp

bezeichnet die Turingmaschine, die zuerst M simuliert und vom Endzustand z_{e_1} von M nach M_1 und vom Endzustand z_{e_2} von M nach M_2 übergeht.



bezeichnet die Turingmaschine, die zuerst M simuliert und vom Endzustand z_{e_1} von M anhält und vom Endzustand z_{e_2} von M nach M_2 übergeht.

7.3.4 Schleifen

Betrachte folgende Turingmaschine

$$M = (\{z_0, z_1, \text{ja}, \text{nein}\}, \Sigma, \Gamma, \delta, z_0, \square, \{\text{ja}, \text{nein}\})$$

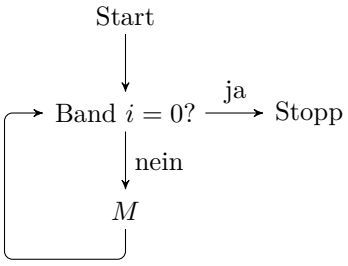
mit

- $\Sigma \supseteq \{0, 1\}$
- $\Gamma \supseteq \{0, 1, \square\}$
- für die Überführungsfunktion δ gilt:

$$\begin{aligned}
\delta(z_0, a) &= (\text{nein}, a, \text{N}) \text{ für } a \in \Gamma \setminus \{0\} \\
\delta(z_0, 0) &= (z_1, 0, \text{R}) \\
\delta(z_1, \square) &= (\text{ja}, \square, \text{L}) \\
\delta(z_1, a) &= (\text{nein}, a, \text{L}) \text{ für } a \in \Gamma \setminus \{0\}
\end{aligned}$$

Bezeichnung für M: „Band $= 0$?“.
Schreibweise: „Band $i = 0$?“ statt „Band $= 0$? (i)“.

Sei nun M eine beliebige Turingmaschine. „WHILE Band $i \neq 0$ DO M“ bezeichnet dann die Turingmaschine



8 Berechenbarkeit in Programmiersprachen

Syntaktische Komponenten von LOOP

- *Variablen*: $x_0, x_1, x_2, \ldots$
  Zur besseren Lesbarkeit werden wir auch Variablennamen wie z. B. $uv, x, y, z, \ldots$ benutzen.
- *Konstanten*: $0, 1, 2, \ldots$
- *Operationszeichen*: $+$ und $-$
- *Trennsymbole*: ; und $:=$
- *Schlüsselwörter*: LOOP, DO und END

Syntax von LOOP

- Sind x_i und x_j Variablen und c eine Konstante, so sind

  $$x_i := x_j + c \quad \text{und} \quad x_i := x_j - c$$

  LOOP-Programme.
- Sind P_1 und P_2 LOOP-Programme, so ist

  $$P_1; P_2$$

  ein LOOP-Programm.
- Ist P ein LOOP-Programm und x_i eine Variable, so ist

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

  ein LOOP-Programm.

Semantik von LOOP

Sei P ein LOOP-Programm. P berechnet eine Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ wie folgt:

Zu Beginn der Rechnung befinden sich Eingabewerte $n_1, \ldots, n_k \in \mathbb{N}$ in den Variablen $x_1, \ldots, x_k$. Alle anderen Variablen haben den Startwert 0. P wird wie folgt ausgeführt:

- Durch das Programm „$x_i := x_j + c$“ erhält x_i als Wert den Wert von $x_j + c$.
- Durch das Programm „$x_i := x_j - c$“ erhält x_i als Wert den Wert von $x_j - c$, falls dieser nichtnegativ ist, ansonsten ist der Wert 0.
- Bei Ausführung des Programms „$P_1; P_2$“ wird zunächst P_1 und dann P_2 ausgeführt.
- Die Ausführung des Programms „LOOP x_i DO P' END“ geschieht wie folgt:

  Das Programm P' wird so oft ausgeführt, wie der Wert der Variablen x_i zu Beginn angibt, d. h. Zuweisungen an x_i in P' haben keinen Einfluss auf die Anzahl der Wiederholungen.

Das Ergebnis der Ausführung von P ist der Wert von x_0 nach Abarbeitung, also $f(n_1, \ldots, n_k) =$ Wert von x_0 am Ende der Ausführung.

Eine Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ heißt *LOOP-berechenbar*, falls es ein LOOP-Programm gibt, das f wie soeben festgelegt berechnet.

Beachte: Jedes LOOP-Programm hält nach endlich vielen Schritten an. Daraus folgt, dass jede LOOP-berechenbare Funktion total ist.

Spezielle LOOP-Programme

„$x_i := x_j$“ steht für: „$x_i := x_j + 0$“.

„$x_i := c$“ (für eine Konstante c) steht für „$x_i := x_j + c$“, wobei x_j eine noch nicht benutzte Variable ist (die also den Wert 0 hat).

„IF $x_i = 0$ THEN P END“ (für ein LOOP-Programm P) steht für folgendes Programm:

„$x_j := 1$;
LOOP x_i DO $x_j := 0$ END;
LOOP x_j DO P END“,
wobei x_j eine Variable ist, die in P nicht vorkommt.

„$x_i := x_j + x_k$“ steht für

„$x_i := x_j$;
LOOP x_k DO $x_i := x_i + 1$ END“.

„$x_i := x_j * x_k$“ steht für

„$x_i := 0$;
LOOP x_k DO $x_i := x_i + x_j$ END“.

Analog: „$x_i := x_j$ DIV x_k“
(Wert von x_i wird größte Zahl kleiner gleich Wert von $\frac{\text{Wert von } x_j}{\text{Wert von } x_k}$, so genannte ganzzahlige Division.)

„$x_i := x_j$ MOD x_k“
(Wert von x_i wird der Rest der ganzzahligen Division des Wertes von x_j durch den Wert von x_k.)

8.1 Die Programmiersprache WHILE

Erweiterung von LOOP:
neues *Schlüsselwort*: WHILE

Syntax: Ist P ein WHILE-Programm und x_i eine Variable, so ist

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

ein WHILE-Programm.

Semantik: Die Ausführung von „WHILE $x_i \neq 0$ DO P END“ geschieht so, dass Programm P so lange wiederholt ausgeführt wird, wie der Wert von x_i ungleich Null ist.

P berechnet $f\colon \mathbb{N}^k \to \mathbb{N}$ wie folgt:
Eingabewerte $n_1, \ldots, n_k$ in Variablen $x_1, \ldots, x_k$, die anderen Variablen haben Startwert 0.

$f(n_1, \ldots, n_k)$ ist der Wert von x_0 nach der Ausführung von P, falls diese stoppt, ansonsten ist $f(n_1, \ldots, n_k)$ undefiniert.

Eine Funktion f heißt *WHILE-berechenbar*, falls es ein WHILE-Programm gibt, das f wie eben festgelegt berechnet.

Beispiel 69. Das LOOP-Programm

LOOP x DO P END

kann simuliert werden durch

$y := x$;
WHILE $y \neq 0$ DO $y := y - 1$; P END,

wobei y eine noch nicht verwendete Variable ist.

Korollar 70. Jedes WHILE-Programm ist äquivalent zu (d. h. berechnet die gleiche Funktion) einem WHILE-Programm, in dem keine LOOP-Schleifen vorkommen.

Satz 71. Jede WHILE-berechenbare Funktion ist Turing-berechenbar.

Beweis 72. Sei P ein WHILE-Programm. P enthalte o. B. d. A. keine LOOP-Schleifen.

Idee: Konstruiere Turingmaschine, die für jede vorkommende Variable ein Band verwendet.

$P =$ „$x_k := x_j + c$“:
Siehe Turingmaschine aus Beispiel 68. Die Werte von x_j und x_k befinden sich auf den Bändern j und k.

$P =$ „$P_1; P_2$“:
Seien M_1 und M_2 Turingmaschinen, welche die von P_1 und P_2 berechneten Funktionen berechnen. Dann simuliert $M_1; M_2$ das Programm P.

$P =$ „WHILE $x_i \neq 0$ DO P' END“:
Sei M' eine Turingmaschine, welche die von P' berechnete Funktion berechnet. Der Wert von x_i befinde sich dabei auf Band i.

Dann entspricht P folgender Turingmaschine: WHILE Band $i \neq 0$ DO M'. ■

Damit folgt:

$$\text{LOOP-berechnbar} \overset{\text{(i)}}{\Rightarrow} \text{WHILE-berechenbar} \overset{\text{(ii)}}{\Rightarrow} \text{Turing-berechenbar},$$

wobei für die Implikation (i) die Umkehrung nicht gilt und für die Implikation (ii) die Umkehrung gilt, aber noch ein Zwischenschritt (im folgenden Abschnitt) durchzuführen ist.

8.2 Die Programmiersprache GOTO

Ein GOTO-Programm ist eine Folge

$$\begin{aligned} &M_1\colon A_1;\\ &M_2\colon A_2;\\ &\quad\vdots\\ &M_k\colon A_k; \end{aligned}$$

wobei die M_i so genannte *Marken* (*Labels*) und die A_i Anweisungen sind.

Mögliche Anweisungen:

- Wertzuweisung: $x_i := x_j \pm c$ (x_i und x_j Variablen, c Konstante)
- Sprung: GOTO M_i
- Bedingter Sprung: IF $x_i = c$ THEN GOTO M_j (x_i Variable, c Konstante)
- Stoppanweisung: HALT

Die letzte Anweisung eines Programms ist eine Stoppanweisung oder ein Sprung.

Semantik
Die Abarbeitung eines GOTO-Programmes beginnt mit der Ausführung der ersten Anweisung. Die Ausführung der Anweisungen der einzelnen Typen geschieht wie folgt:

- $x_i := x_j \pm c$: Wert von x_i wird $x_j \pm c$, danach weiter mit nächster Anweisung.
- GOTO M_i: Weiter mit der Anweisung mit der Marke M_i.
- IF $x_i = c$ THEN GOTO M_j: Falls Wert von x_i gleich c ist, weiter mit der Anweisung mit der Marke M_j, sonst weiter mit nächster Anweisung.
- HALT: Ausführung des Programms stoppen.

Die berechnete Funktion ist wie bei LOOP- bzw. WHILE-Programmen definiert.

Vereinbarung: Marken, die nie hinter einem GOTO vorkommen, dürfen weggelassen werden. Marken müssen nicht in der Reihenfolge $M_1 \ldots M_2 \ldots M_3 \ldots$ benutzt werden.

Beispiel 73. Das WHILE-Programm

WHILE $x_i \neq 0$ DO P END

kann simuliert werden durch

M_1: IF $x_i = 0$ THEN GOTO M_2;
P;
GOTO M_1;
M_2: ...

Korollar 74. Jede WHILE-berechenbare Funktion ist GOTO-berechenbar.

Für die Umkehrung gilt:

Gegeben sei ein GOTO-Programm

M_1: A_1;
M_2: A_2;
⋮
M_k: A_k;

Dies kann durch folgendes WHILE-Programm simuliert werden:

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

Dabei ist x irgendeine Variable, die in $A_1, \ldots, A_k$ nicht verwendet wird, und A'_i $(1 \leq i \leq k)$ ist folgendes Programm:

- „$x_j := x_l \pm c$; $x := x + 1$“, falls $A_i =$ „$x_j := x_l \pm c$“
- „$x := m$“, falls $A_i =$ „GOTO M_m“
- „IF $x_j = c$ THEN $x := m$ ELSE $x := x + 1$ END“,
falls $A_i =$ „IF $x_j = c$ THEN GOTO M_m“
- „$x := 0$“, falls $A_i =$ „HALT“

Das „IF ... THEN ... ELSE ...“-Programm kann durch LOOP-Schleifen ersetzt werden (Übungsaufgabe).

Beachte: Das konstruierte WHILE-Programm hat nur eine WHILE-Schleife! (Aber weitere LOOP-Schleifen.)

Korollar 75. Jede GOTO-berechenbare Funktion ist auch WHILE-berechenbar.

Korollar 76. Jede WHILE-berechenbare Funktion kann durch ein WHILE-Programm mit nur einer WHILE-Schleife ersetzt werden.

Beweis 77. Sei f WHILE-berechenbar vermöge P. Dann simuliere P durch das GOTO-Programm P', und simuliere P' durch das WHILE-Programm P'' wie oben. P'' hat dann nur eine WHILE-Schleife. ■

Sei f Turing-berechenbar. Sei $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ eine 1-Band-Turingmaschine, die f berechnet. Wir wollen zeigen, dass f GOTO-berechenbar ist.

Seien $Z = \{z_1, \ldots, z_k\}$ und $\Gamma = \{a_1, \ldots, a_m\}$. Sei $b = m + 1$.

Wir repräsentieren eine Konfiguration $a_{i_1} a_{i_2} \ldots a_{i_p} z_l a_{j_1} a_{j_2} \ldots a_{j_q}$ durch die Werte der Programmvariablen x,y und z, wobei

- $x = (i_1 \ldots i_p)_b$, d. h. die Zahl $i_1 \ldots i_p$ in b-adischer Darstellung ist. Das leere Wort wird durch 0 kodiert.
- $y = (j_q \ldots j_1)_b$
- $z = l$

Also: $x = \sum_{\mu=1}^{p} i_\mu \cdot b^{p-\mu}$ und $y = \sum_{\mu=1}^{q} i_\mu \cdot b^{\mu-1}$.

Das GOTO-Programm, das f berechnet, sieht folgendermaßen aus:

M_1: P_K;
M_2: P_M;
M_3: P_D;

Dabei ist P_K ein Programmstück, das die Kodierung der Startkonfiguration von M in den Variablen x, y und z erzeugt.
P_M ist ein Programmstück, das M schrittweise simuliert durch Änderungen der Werte von x, y und z.
P_D ist ein Programmstück, das aus der Kodierung der Endkonfiguration von M in x, y und z die Ausgabe von M in x_0 erzeugt.

Wir geben zunächst P_M genauer an:

M_2: $n := y$ MOD b;
IF $z = 1$ THEN GOTO M_{21};
IF $z = 2$ THEN GOTO M_{22};
$\vdots$
IF $z = k$ THEN GOTO M_{2k};
M_{21}: IF $n = 1$ THEN GOTO M_{211};
IF $n = 2$ THEN GOTO M_{212};
$\vdots$
IF $n = m$ THEN GOTO M_{21m};
M_{22}: IF $n = 1$ THEN GOTO M_{221};
IF $n = 2$ THEN GOTO M_{222};
$\vdots$
IF $n = m$ THEN GOTO M_{22m};
$\vdots$
M_{211}: P_{211};
M_{212}: P_{212};
$\vdots$
M_{2km}: P_{2km};

Es bleibt noch, die Programmfragmente P_{2ij} $(1 \leq i \leq k, 1 \leq j \leq m)$ anzugeben.

Wir betrachten die Marke M_{2ij}. Sie wird angesprungen, falls z_i und $n = j$, also falls die Turingmaschine M im Zustand z_i ist und das Zeichen a_j liest (beachte: $n = y$ MOD $b = j_1$, falls $y = (j_q \dots j_1)_b$ ist).

Sei $\delta(z_i, a_j) = (z_r, a_s, X)$. Die Konfiguration von M sei $a_{i_1} a_{i_2} \dots a_{i_p} z_i a_{j_1} a_{j_2} \dots a_{j_q}$ mit $j_1 = j$.

Das Programm P_{2ij} sieht dann wie folgt aus:

$\delta(z_i, a_j) = (z_r, a_s, \mathrm{N})$ wird simuliert durch:

	Zustand z_r	neue Konfiguration
$z := r$;		
$y := y$ DIV b;	$y = (j_q \dots j_2)_b$	$a_{i_1} a_{i_2} \dots a_{i_p} z_r a_s a_{j_2} \dots a_{j_q}$
$y := b * y + s$;	$y = (j_q \dots j_2 s)_b$	

$\delta(z_i, a_j) = (z_r, a_s, \mathrm{L})$ wird simuliert durch:

	Zustand z_r	neue Konfiguration
$z := r$;		
$y := y$ DIV b;	$y = (j_q \dots j_2)_b$	$a_{i_1} a_{i_2} \dots a_{i_{p-1}} z_r a_{i_p} a_s a_{j_2} \dots a_{j_q}$
$y := b * y + s$;	$y = (j_q \dots j_2 s)_b$	
$y := b * y + (x$ MOD $b)$;	$y = (j_q \dots j_2 s i_p)_b$	
$x := x$ DIV b;	$x = (i_1 i_2 \dots i_{p-1})_b$	

$\delta(z_i, a_j) = (z_r, a_s, \mathrm{R})$ wird simuliert durch:

	Zustand z_r	neue Konfiguration
$z := r$;		
$y := y$ DIV b;	$y = (j_q \dots j_2)_b$	$a_{i_1} a_{i_2} \dots a_{i_p} a_s z_r a_{j_2} \dots a_{j_q}$
$x := x * b + s$;	$x = (i_1 \dots i_p s)_b$	

(Im Falle einer Kopfbewegung nach links nehmen wir dabei der Einfachheit halber an, dass $x \neq 0$.) Ist $z_r \in E$, so fügen wir in allen drei Fällen den Befehl „GOTO M_3“ an, ansonsten fügen wir den Befehl „GOTO M_2“ an.

Wir geben als nächstes das GOTO-Programm P_K genau an. Wir nehmen zur Vereinfachung der Präsentation dabei an, dass die betrachtete Turingmaschine M eine einstellige Funktion berechnet. Die Eingabe von M ist also stets eine Zahl in Binärdarstellung und das Eingabealphabet ist

demnach $\Sigma = \{0, 1\}$. Wir nehmen für das Arbeitsalphabet $\Gamma = \{a_1, a_2, \ldots, a_m\}$ an, dass $a_1 = 0$, $a_2 = 1$ und $a_3 = \square$.

Die Eingabe von M ist die Binärdarstellung einer Zahl. Sei v diese Zahl und sei $v_1 \ldots v_n$ die Binärdarstellung dieser Zahl. Für die Eingabe von P_K gilt dann $x_1 = v$ und alle anderen Variablen sind 0. Die Startkonfiguration von M ist $z_1 a_{v_1+1} a_{v_2+1} \ldots a_{v_n+1}$.

Wir müssen also folgendes mit P_K erreichen:

$$\begin{aligned} x &= 0 \\ y &= ((v_n + 1)(v_{n-1} + 1) \ldots (v_1 + 1))_b \\ z &= 1 \end{aligned}$$

D. h. $y = \sum_{\mu=1}^{n} (v_\mu + 1) \cdot b^{\mu-1}$. Folgendes Programm erreicht dies:

```
    x := 0;
    z := 1;
    y := 0;
    IF x_1 = 0 THEN GOTO 2;          // Abfangen des Spezialfalls v = 0
1:  IF x_1 = 0 THEN GOTO 3;
    x_2 := x_1 MOD 2;
    x_2 := x_2 + 1;
    y := y · b;
    y := y + x_2;
    x_1 := x_1 DIV 2;
    GOTO 1;
2:  y := 1;                          // v = 0 → y = 1
3:  HALT
```

Die Konstruktion von P_D verbleibt als Übungsaufgabe.

Es folgt:

Satz 78. Jede Turing-berechenbare Funktion ist GOTO-berechenbar.

9 Die Church'sche These

Wir haben gesehen:

$$\text{Turing-berechenbar} \Leftrightarrow \text{WHILE-berechenbar} \Leftrightarrow \text{GOTO-berechenbar}$$

Auch andere Versuche, den Begriff der Berechenbarkeit formal zu fassen, haben sich als äquivalent herausgestellt, z. B.

- μ-rekursive Funktion,
- λ-definierbare Funktionen,
- Markov-Programme,
- Registermaschinen,
- ...

Alle bekannten Programmiersprachen führen ebenfalls zum gleichen Begriff von Berechenbarkeit.

These von Church

Eine Funktion ist berechenbar im intuitiven Sinne genau dann, wenn sie Turing-berechenbar ist.
(Nicht beweisbar, da „berechenbar im intuitiven Sinne" nicht formal gefasst.)

Wir verwenden also in Zukunft den Begriff „berechenbar" synonym mit „Turing-berechenbar".
Andere Sprechweise: „rekursiv" oder „partiell rekursiv"
(„total rekursiv" = berechenbar und total, d. h. überall definiert.)

> **Bemerkung 79.** Es gibt berechenbare Funktionen, die nicht LOOP-berechenbar sind. Zum Beispiel ist keine nicht-überall definierte Funktion LOOP-berechenbar.

Frage: Gibt es totale und berechenbare Funktionen, die nicht LOOP-berechenbar sind?
Antwort: Ja, zum Beispiel die so genannte Ackermannfunktion (siehe Schöning).

10 Entscheidbarkeit und Aufzählbarkeit

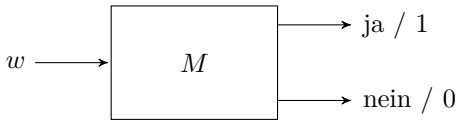
Übertragung des Begriffs „Berechenbarkeit“ aus der Welt der Funktionen in den Kontext von Sprachen:

Definition 80. Eine Sprache $A \subseteq \Sigma^*$ heißt *entscheidbar*, wenn die Funktion $c_A \colon \Sigma^* \to \{0,1\}$ mit

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

berechenbar ist. c_A heißt *charakteristische Funktion* von A.

Also: Eine Sprache A ist entscheidbar, falls ein Algorithmus M existiert, der bei Eingabe w nach endlicher Zeit stoppt und „ja“ ausgibt, falls $w \in A$, und „nein“ ausgibt, falls $w \notin A$.



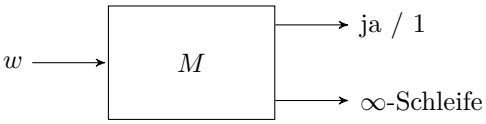
M heißt „Entscheidungsalgorithmus“ für A.

Definition 81. Eine Sprache $A \subseteq \Sigma^*$ heißt *semi-entscheidbar*, wenn die Funktion

$$\chi_A \colon \Sigma^* \to \{0,1\} \text{ mit } \chi_A(w) := \begin{cases} 1, & \text{falls } w \in A \\ \text{undefiniert}, & \text{sonst} \end{cases}$$

berechenbar ist.

Also: Eine Sprache A ist semi-entscheidbar, falls ein Algorithmus M existiert, der bei Eingabe w nach endlicher Zeit stoppt und „ja“ ausgibt, falls $w \in A$, und in eine Endlosschleife gerät, falls $w \notin A$.



M heißt „Semi-Entscheidungsalgorithmus“ für A.

Analoge Definitionen ergeben sich für $A \subseteq \mathbb{N}$.

Für $A \subseteq \Sigma^*$ ist $\overline{A} =_{\text{def}} \Sigma^* \setminus A = \{w \in \Sigma^* \mid w \notin A\}$.

beobachtung Sei $A \subseteq \Sigma^*$. Es gilt:

A ist entscheidbar genau dann, wenn $\overline{A}$ ist entscheidbar.

Satz 82. Sei $A \subseteq \Sigma^*$. Es gilt:

A ist entscheidbar genau dann, wenn A und $\overline{A}$ sind semi-entscheidbar.

Beweis 83.

„⇒“: Sei M ein Entscheidungsalgorithmus für A.

Semi-Entscheidungsalgorithmus für A:

Eingabe: w
Simuliere M bei Eingabe w;
Falls M mit „ja“ stoppt, dann Ausgabe „ja“, stopp;
Sonst: Endlosschleife.

Semi-Entscheidungsalgorithmus für $\overline{A}$:

Eingabe: w
Simuliere M bei Eingabe w;
Falls M mit „nein“ stoppt, dann Ausgabe „ja“, stopp;
Sonst: Endlosschleife.

„⇐“: Seien M_1 und M_2 Turingmaschinen, die χ_A bzw. $\chi_{\overline{A}}$ berechnen. Dann ist folgender Algorithmus ein Entscheidungsalgorithmus für A:

Eingabe: w
$s := 1$;
M_1: Simuliere M_1 für s Schritte;
Falls M_1 dabei stoppt, dann: Ausgabe „1“ und HALT;
Simuliere M_2 für s Schritte;
Falls M_2 dabei stoppt, dann: Ausgabe „0“ und HALT;
$s := s + 1$;
GOTO M.

■

Definition 84. Eine Sprache $A \subseteq \Sigma^*$ heißt *rekursiv-aufzählbar*, falls $A = \emptyset$ oder falls es eine totale berechenbare Funktion $f\colon \mathbb{N} \to \Sigma^*$ gibt, sodass

$$A = \{f(0), f(1), f(2), \ldots\}.$$

Wir sagen: f zählt A auf.

Satz 85. Eine Sprache ist rekursiv-aufzählbar genau dann, wenn sie semi-entscheidbar ist.

Beweis 86.

„⇒“: Sei $A = \{f(0), f(1), f(2), \ldots\}$ für eine totale, berechenbare Funktion f.

Der folgende Algorithmus ist ein Semi-Entscheidungsalgorithmus für f:

Eingabe: w
$n := 0$;
M_1: IF $f(n) = w$ THEN Ausgabe „1“; HALT END;
$n := n + 1$;
GOTO M.

„⇐“: Wir benötigen eine Kodierung von Paaren von natürlichen Zahlen durch eine natürliche Zahl. Dazu definieren wir:

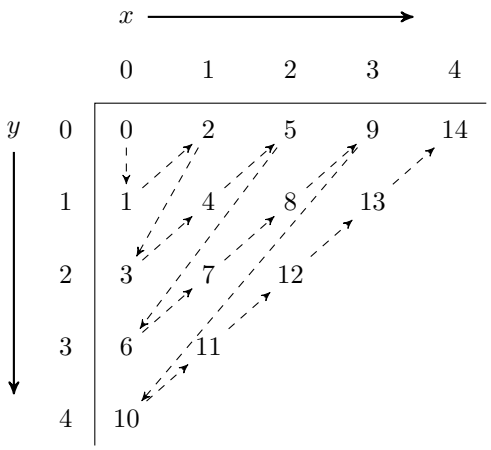
$$c(x, y) = \frac{1}{2}(x + y + 1)(x + y) + x.$$

Schreibweise: $\langle x, y\rangle = c(x, y)$.

Es gilt: Für jede natürliche Zahl n gibt es genau ein Paar x, y, sodass $n = \langle x, y\rangle$.

Mit anderen Worten: Die Funktion: $c\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ ist bijektiv.

Die Behauptung ergibt sich aus folgender Wertetabelle für c:



Sei nun $A \subseteq \Sigma^*$ semi-entscheidbar. Sei M eine Turingmaschine, die χ_A berechnet. Ist $A = \emptyset$, so ist A rekursiv-aufzählbar nach Definition.

Sei also $A \neq \emptyset$ und $a \in A$. Folgender Algorithmus berechnet dann eine Funktion f, die A aufzählt:

Eingabe: n
Seien $k, s \in \mathbb{N}$ so, dass $n = \langle k, s\rangle$;
Sei w das k-te Wort in Σ^* in lexikographischer Reihenfolge;
Simuliere M bei Eingabe w für s Schritte;
Falls M dabei „1“ ausgibt, dann Ausgabe „w“; sonst: Ausgabe „a“
Stopp.

Es gilt: f ist total und berechenbar. Ist w ein Wort, das vom Algorithmus ausgegeben wird, so ist $w \in A$, also:

$$\{f(0), f(1), f(2), \ldots\} \subseteq A.$$

Sei nun $w \in A$. Dann gibt es eine Zahl s, sodass M bei Eingabe w nach s Schritten mit Ausgabe „1“ stoppt.

Sei w das k-te Wort in Σ^*. Sei $n = \langle k, s\rangle$. Aus dem Algorithmus ergibt sich, dass dann $f(n) = w$, also

$$A \subseteq \{f(0), f(1), f(2), \ldots\}.$$

Zusammengenommen:

$$A = \{f(0), f(1), f(2), \ldots\},$$

also ist A rekursiv-aufzählbar. ■

Korollar 87. Eine Sprache A ist entscheidbar genau dann, wenn A und $\overline{A}$ rekursiv-aufzählbar sind.

11 Unentscheidbare Probleme

11.1 Das Halteproblem

Gibt es rekursiv-aufzählbare Sprachen, die nicht entscheidbar sind?

Beispiel 88.

Eingabe: n
WHILE $n \neq 1$ DO
$n := n - 2$
END
Ausgabe: n

Der Algorithmus hält, falls die Eingabe n eine ungerade Zahl ist, ansonsten hält der Algorithmus nicht.

Beispiel 89.

Eingabe: n
WHILE $n \neq 1$ DO
IF n ist gerade THEN
$n := n$ div 2
ELSE $n := 3 \cdot n + 1$
END
Ausgabe: 1

Beispielsweise erhält man für die Eingabe 3 folgendes:

$$n = 3|10|5|16|8|4|2|1 \qquad \text{Ausgabe 1, halt.}$$

Es ist nicht bekannt, ob der Algorithmus für alle Eingaben hält.

Zunächst: Wir möchten Turingmaschinen durch Wörter über dem Alphabet $\{0, 1\}$ kodieren.

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ mit $\Gamma = \{a_0, a_1, \ldots, a_k\}$ und $Z = \{z_0, z_1, \ldots, z_n\}$. Dann kodieren wir einen Übergang $\delta(z_i, a_j) = (z_{i'}, a_{j'}, X)$ durch das Wort

$$\#\#\,\mathrm{bin}(i)\#\,\mathrm{bin}(j)\#\,\mathrm{bin}(i')\#\,\mathrm{bin}(j')\#\,\mathrm{bin}(m) \quad \text{mit } m := \begin{cases} 0, & \text{falls } X = \mathrm{L} \\ 1, & \text{falls } X = \mathrm{N} \\ 2, & \text{falls } X = \mathrm{R} \end{cases}$$

Wir hängen alle so erhaltenen Wörter für alle Übergänge in δ aneinander (in beliebiger Reihenfolge). Ergebnis: Kodierung von M als Wort über $\{0, 1, \#\}$ oder als natürliche (Binärdarstellung).

In diesem Wort kodieren wir schließlich die einzelnen Buchstaben wie folgt:

0 durch 00
1 durch 01
durch 11

Ergebnis: Kodierung von M als Wort $\{0, 1\}$.

Man nennt diese Kodierung auch *Gödelisierung* von M.

Nicht jedes Wort aus $\{0,1\}^*$ ergibt sich so als Kodierung einer Turingmaschine. Wir legen aber fest:

Sei $w \in \{0,1\}^*$. Dann ist

$$M_w := \begin{cases} M, & \text{falls } w \text{ Gödelisierung von } M \text{ wie oben beschrieben ist} \\ \widehat{M}, & \text{sonst,} \end{cases}$$

wobei $\widehat{M}$ die Turingmaschine aus Beispiel 62 ist, die Ω berechnet.

Definition 90. Das *spezielle Halteproblem* ist die Sprache

$$K = \{w \in \{0,1\}^* \mid M_w \text{ hält bei Eingabe } w\}$$

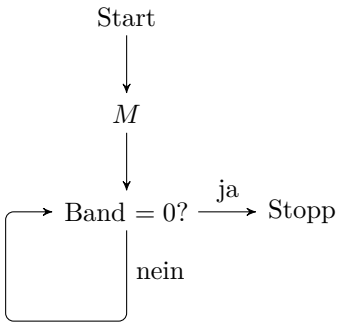
beobachtung K ist rekursiv-aufzählbar.

Beweis 91. Semi-Entscheidungsalgorithmus für K:

Eingabe w:
Konstruiere die δ-Funktion von M_w;
Simuliere M_w auf Eingabe w;
Falls Simulation stoppt, dann Ausgabe „1“; HALT. ■

Satz 92. K ist nicht entscheidbar.

Beweis 93. Annahme: K wäre entscheidbar. Sei M eine Turingmaschine, die c_K berechnet. Definiere Turingmaschine M' wie folgt:



Also: M' stoppt genau dann, wenn M gibt „0“ aus.
M' gerät in Endlosschleife genau dann, wenn M gibt „1“ aus.

Sei w eine Gödelisierung von M'. Nun gilt:

$$\begin{aligned} w \in K &\Leftrightarrow M' \text{ hält bei Eingabe } w \text{ (Definition von } K) \\ &\Leftrightarrow M \text{ hält bei Eingabe } w \text{ und gibt „0“ aus (Konstruktion von } M') \\ &\Leftrightarrow c_K(w) = 0 \text{ (Definition von } M) \\ &\Leftrightarrow w \notin K \end{aligned}$$

Dies ist ein Widerspruch zur Annahme, also ist K nicht entscheidbar. ■

Korollar 94. $\overline{K}$ ist nicht rekursiv-aufzählbar.

Nachweis, dass weitere Probleme nicht entscheidbar sind: Zurückführen auf bekannte unentscheidbare Probleme (das heißt zunächst auf K).

Formal:

Definition 95. Seien $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ Sprachen.

A heißt auf B *reduzierbar*, in Zeichen: $A \leq B$, falls es eine totale, berechenbare Funktion $f\colon \Sigma^* \to \Gamma^*$ gibt, sodass für alle $w \in \Sigma^*$ gilt:

$$w \in A \Leftrightarrow f(w) \in B$$

Lemma 96. Ist $A \leq B$ und B entscheidbar, so ist A entscheidbar.

Beweis 97. Sei $A \leq B$ via f und c_B berechenbar. Dann ist c_A berechenbar mit folgendem Algorithmus:

Eingabe: w
Berechne $z = f(w)$
Berechne $b = c_B(z)$
Ausgabe: b

Es gilt:

$$w \in A \Leftrightarrow f(w) = z \in B \Leftrightarrow b = c_B(z) = 1 \Leftrightarrow \text{Algorithmus gibt „1“ aus.}$$ ■

Kontraposition Ist $A \leq B$ und A nicht entscheidbar, dann ist auch B nicht entscheidbar.

Definition 98. Das *(allgemeine) Halteproblem* ist die Sprache

$$H = \{w\#x \mid M_w \text{ hält bei Eingabe } x\}.$$

Satz 99. H ist nicht entscheidbar.

Beweis 100. Wir zeigen: $K \leq H$.

Wähle $f(w) = w\#w$. Dann gilt: $w \in K \Leftrightarrow w\#w \in H \Leftrightarrow f(w) \in H$.

Also $K \leq H$ via f. ■

11.2 Entscheidbarkeit in der Chomsky-Hierarchie

Wir haben bereits gesehen: Ist A eine Sprache vom Typ 1, 2 oder 3, so ist A entscheidbar (siehe Abschnitt 2).

Es verbleiben die Typ-0-Sprachen.

Satz 101. Eine Sprache ist vom Typ 0 genau dann, wenn sie rekursiv-aufzählbar ist.

Beweis 102.

„⇐“: Zur Erinnerung: Eine Sprache ist vom Typ 0 genau dann, wenn es eine Turingmaschine M gibt mit:

$$w \in A \Rightarrow M \text{ erreicht bei Eingabe } A \text{ einen Endzustand}$$
$$w \notin A \Rightarrow M \text{ gerät bei Eingabe } A \text{ in eine Endlosschleife}$$

Sei nun A rekursiv-aufzählbar via Semi-Entscheidungsverfahren M. Dann akzeptiert M die Sprache A in obigem Sinne, also ist A vom Typ 0.

„⇒“: A werde von Turingmaschine M akzeptiert. Wir modifizieren M so, dass beim Erreichen eines Endzustands eine „1“ ausgegeben wird. Diese neue Turingmaschine berechnet also χ_A, also ist A rekursiv-aufzählbar. ■

Sei A eine Sprache. Aus den bisherigen Resultaten ergibt sich, dass die folgenden Aussagen äquivalent sind:

(1) A ist vom Typ 0.

(2) $A = L(M)$ für eine Turingmaschine M.

(3) A ist rekursiv-aufzählbar.

(4) A ist Wertebereich einer totalen berechenbaren Funktion oder $A = \emptyset$.

(5) A ist semi-entscheidbar.

(6) A ist Definitionsbereich einer berechenbaren Funktion.

Man kann zeigen, dass folgende Aussage zu obigen äquivalent ist:

(7) A ist Wertebereich einer (eventuell partiellen) berechenbaren Funktion.

Korollar 103. Die Klasse der Typ-1-Sprachen ist eine echte Teilmenge der Klasse der Typ-0-Sprachen.

Beweis 104. Das Halteproblem ist rekursiv-aufzählbar, also vom Typ 0, aber nicht entscheidbar und daher nicht vom Typ 1. ■

Die Sprachklassen im Überblick:

$$\text{Typ } 3 \overset{(1)}{\subsetneq} \text{Typ } 2 \overset{(2)}{\subsetneq} \text{Typ } 1 \overset{(3)}{\subsetneq} \text{Typ } 0,$$

das heißt, dass die Inklusionen echt sind. Beispiele dafür, dass die Inklusionen echt sind:

(1) $\{\mathsf{a}^n\mathsf{b}^n \mid n \geq 0\}$.

(2) $\{\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n \mid n \geq 1\}$.

(3) Die Sprachen H und K.

11.3 Der Satz von Rice

Definition 105. Das *Halteproblem auf leerem Band* ist die Sprache

$$H_0 = \{w \mid M_w \text{ angesetzt auf leerem Band hält}\}$$

Satz 106. H_0 ist nicht entscheidbar.

Beweis 107. Wir zeigen: $H \leq H_0$.

Zu jedem Wort $w\#x$ betrachten wir die Turingmaschine $M_{(w\#x)}$, die folgendermaßen arbeitet:

Eingabe: y
Falls Band leer (also $y = \varepsilon$):
Schreibe x auf das Band und simuliere danach M_w auf Eingabe x.
Falls Band nicht leer: Stopp.

Sei f eine Funktion, die $w\#x$ abbildet auf die Gödelisierung von $M_{(w\#x)}$:

$$f(w\#x) = \text{Gödelisierung von } M_{(w\#x)}.$$

Sei f beliebig definiert für Eingaben, die nicht von der Form $w\#x$ sind. f ist berechenbar. (Bei Eingabe $w\#x$ konstruiere zunächst $M_{(w\#x)}$ und bilde dann deren Gödelisierung.)

Es gilt weiterhin:

$$\begin{aligned} w\#x \in H &\Leftrightarrow M_w \text{ hält bei Eingabe } x \\ &\Leftrightarrow M_{(w\#x)} \text{ hält mit leerem Band} \\ &\Leftrightarrow f(w\#x) \in H_0 \end{aligned}$$

Also ist $H \leq H_0$ vermöge f. ■

Der folgende Satz zeigt, dass jede funktionale Eigenschaft von Turingmaschinen unentscheidbar ist.

Satz 108 (von Rice). Sei $\mathcal{R}$ die Klasse aller berechenbaren Funktionen. Sei $\mathcal{S} \subseteq \mathcal{R}$ mit $\mathcal{S} \neq \emptyset$ und $\mathcal{S} \neq \mathcal{R}$.

Dann ist die Sprache

$$C(\mathcal{S}) = \{w \mid \text{die von } M_w \text{ berechnete Funktion ist aus } \mathcal{S}\}$$

nicht entscheidbar.

Beweis 109. Sei Ω die überall undefinierte Funktion.

Fall 1: $\Omega \in \mathcal{S}$

Da $\mathcal{S} \neq \mathcal{R}$, gibt es eine Funktion $g \in \mathcal{R} \setminus \mathcal{S}$. Sei M_g eine Turingmaschine, die g berechnet.

Zu jedem Wort w betrachten wir die Turingmaschine $M(w)$, die folgendermaßen arbeitet:

Eingabe: y
Simuliere M_w bei leerem Band;
Falls diese Simulation stoppt: Simuliere M_g bei Eingabe y;

Sei h die von $M(w)$ berechnete Funktion. Dann gilt:

$$h = \begin{cases} \Omega, & \text{falls } M_w \text{ bei leerem Band nicht stoppt.} \\ g, & \text{sonst.} \end{cases}$$

Sei $f(w)$ die Gödelisierung von $M(w)$. f ist berechenbar.

Es gilt weiterhin:

$$\begin{aligned} w \in H_0 &\Rightarrow M_w \text{ bei leerem Band stoppt} \\ &\Rightarrow M(w) \text{ berechnet die Funktion } g \\ &\Rightarrow \text{die von } M(w) \text{ berechnete Funktion liegt nicht } \mathcal{S} \\ &\Rightarrow f(w) \notin C(\mathcal{S}) \end{aligned}$$

$$\begin{aligned} w \notin H_0 &\Rightarrow M_w \text{ bei leerem Band stoppt nicht} \\ &\Rightarrow M(w) \text{ berechnet die Funktion } \Omega \\ &\Rightarrow \text{die von } M(w) \text{ berechnete Funktion liegt } \mathcal{S} \\ &\Rightarrow f(w) \in C(\mathcal{S}) \end{aligned}$$

Es folgt: $\overline{H_0} \leq C(\mathcal{S})$ vermöge f, also ist $C(\mathcal{S})$ nicht entscheidbar, da $\overline{H_0}$ nicht entscheidbar ist.

Fall 2: $\Omega \notin \mathcal{S}$

Da $\mathcal{S} \neq \emptyset$, gibt es eine Funktion $g \in \mathcal{S}$. Definiere mit diesem g die Maschinen M_g, $M(w)$ und die Funktion f wie oben.

Nun gilt:

$$\begin{aligned} w \in H_0 &\Rightarrow M_w \text{ bei leerem Band stoppt} \\ &\Rightarrow M(w) \text{ berechnet die Funktion } g \\ &\Rightarrow \text{die von } M(w) \text{ berechnete Funktion liegt } \mathcal{S} \\ &\Rightarrow f(w) \in C(\mathcal{S}) \end{aligned}$$

$$\begin{aligned} w \notin H_0 &\Rightarrow M_w \text{ bei leerem Band stoppt nicht} \\ &\Rightarrow M(w) \text{ berechnet die Funktion } \Omega \\ &\Rightarrow \text{die von } M(w) \text{ berechnete Funktion liegt nicht } \mathcal{S} \\ &\Rightarrow f(w) \notin C(\mathcal{S}) \end{aligned}$$

Daraus folgt: $H_0 \leq C(\mathcal{S})$, also ist $C(\mathcal{S})$ nicht entscheidbar. ■

Korollar 110. Die folgenden Sprachen sind nicht entscheidbar:

- $\{w \mid M_w \text{ berechnet eine totale Funktion}\}$
- $\{w \mid M_w \text{ berechnet eine monotone Funktion}\}$
- $\{w \mid M_w \text{ berechnet eine konstante Funktion}\}$
- $\{w \mid M_w \text{ berechnet die Funktion } f(x) = x + 1\}$

Kommentiertes Literaturverzeichnis

[1] John E. Hopcroft, Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Publishing Company, 1979.

Der absolute Klassiker zur Theoretischen Informatik! Enthält den gesamten Stoff der Vorlesung. Nachteil: nur noch antiquarisch erhältlich.

[2] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Einführung in Automatentheorie, Formale Sprachen und Berechenbarkeit, Pearson, 2011.

Neuauflage von [1]. Viele zentrale Themen werden sehr ausführlich motiviert und erklärt. Nachteil: Einige für diese Vorlesung wichtige Themengebiete, die noch in [1] behandelt werden, fallen weg.

[3] Uwe Schöning, Theoeretische Informatik – kurzgefasst, Spektrum Akademischer Verlag, 2008.

Der deutsche Klassiker zum Thema! Enthält den gesamten Stoff der Vorlesung. Die Darstellung ist aber oft sehr knapp.

[4] Hans Hermes, Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit, Springer-Verlag, 1961.

Ältere deutsche Darstellung des Gebietes der Berechenbarkeit. Sehr empfehlenswerte Lektüre, auch wenn Sie sich für Fragen interessieren, die über die Vorlesung hinausgehen.

[5] Hartley Rogers, Jr., Theory of Recursive Functions and Effective Computability, Mc Graw-Hill Book Company, 1967.

Klassisches Lehrbuch der Berechenbarkeitstheorie. Sehr empfehlenswerte Lektüre, auch wenn Sie sich für Fragen interessieren, die über die Vorlesung hinausgehen. Taschenbuchausgabe von 1987 noch erhältlich.

[6] Wolfgang J. Paul, Komplexitätstheorie, B. G. Teubner, 1978.

Erstes deutsches Lehrbuch zur Komplexitätstheorie, das im ersten Teil eine ausgezeichnete Einführung in die Berechenbarkeitstheorie enthält.

[7] Martin Davis, Computability and Unsolvability, Dover Publications Inc., 1958.

Ein Werk der Unterhaltungsmathematik: Davis erklärt in lockerem Plauderton, aber doch mathematisch exakt Fragestellungen, Konzepte und Resultate der Berechenbarkeitstheorie. Dover hat 1983 das Werk neu in einer günstigen, heute noch erhältlichen Taschenbuchausgabe aufgelegt. Sehr empfehlenswert!