1 Reguläre Sprachen

Definition 1.1 Alphabet

Eine endliche, nicht leere Menge.

Definition 1.2 Wort
Ein Wort w über einem Alphabet Σ ist eine endliche Folge $\sigma_1 \dots \sigma_n$ von Zeichen aus Σ.

Definition 1.3 Wortlänge $|w|$

Ist w=$\sigma_1 \dots \sigma_n$ so ist $|w| := n$. ϵ ist das leere Wort mit $|\epsilon| = 0$.

Definition 1.4 Konkatenation $\circ$ von Worten
Sind u und v Wörter, so bezeichnet $u \circ v$ das Wort, das entsteht, wenn v hinter u geschrieben wird.

Definition 1.5 n-malige Wiederholung
u^n bezeichnet die n-malige Wiederholung von u. Induktiv also

$$u^0 = \epsilon, u^{n+1} = u^n \circ u$$

Definition 1.6 Sprache

Menge von Wörtern über einem Alphabet Σ.

Definition 1.7 Konkatenation von Sprachen
Seien L_1 und L_2 Sprachen, dann ist

$$L_1 \circ L_2 =_{def} \{u \circ v | u \in L_1, v \in L_2\}$$

Definition 1.8 Wiederholung von Sprachen
Ist L_1 eine Sprache so L_1^* auch eine Sprache mit:

$$L_1^* =_{def} \{u_1 \dots u_n | u_1, \dots, u_n \in L_1, n \in \mathbb{N}\}$$

Bemerkung Konventionen
Griechischen Großbuchstaben stehen meist für Alphabete, Kleinbuchstaben hingegen für Worte oder Zeichen aus einem Alphabet.

1.1 Reguläre Ausdrücke

Definition 1.9 Syntax Regulärer Ausdrücke
Sei Σ ein Alphabet. Dann definieren folgenden Regeln induktiv die Menge der regulären Audrücke über Σ:

1. (a) Das Zeichen $\emptyset$ ist ein regulrärer Ausdruck.
   (b) Das Zeichen ϵ ist ein regulärer Audruck.
   (c) Für jedes $\sigma \in \Sigma$ ist *sigma* ein regulärer Audruck.
2. Sind α und β reguläre Ausdrücke, so auch
   (a) $(\alpha\beta)$, und
   (b) $(\alpha + \beta)$
3. Ist α ein regulärer Ausdruck, so auch (α^*)

Bemerkung D
ie Menger aller regulären Ausdrücke über Σ ist selbst auch wieder eine Sprache $\Sigma \cup \{\emptyset, \epsilon, +, *,), (\}$

Definition 1.10 Semantik Regulärer Ausdrücke
Für jeden regulären Ausdruck α sei die Sprache $L(\alpha)$ wie folgt definiert:

- – $L(\emptyset) = \emptyset$
  – $L(\epsilon) = \{\epsilon\}$
  – $L(\sigma) = \{\sigma\}$, für jedes $\sigma \in \Sigma$
- Sind α und β reguläre Ausdrücke so ist
  – $L((\alpha\beta)) = L(\alpha) \circ L(\beta)$
  – $L((\alpha + \beta)) = L(\alpha) \cup L(\beta)$
- Ist α ein regulärer Ausdruck, so ist $L((\alpha^*)) = L(\alpha)^*$

Definition 1.11 Reguläre Sprache

Eine Sprache L heißt regulär, falls es einen regulären Ausdruck α gibt mit $L = L(\alpha)$

Satz 1.12 Äquivalenzen für + und $\circ$
Seien α, β, γ reguläre Ausdrücke:

- Assoziativität bezüglich + und $\circ$
  – $\alpha + (\beta + \gamma) \equiv (\alpha + \beta) + \gamma$
  – $\alpha(\beta\gamma) \equiv (\alpha\beta)\gamma$
- Kommutativität bezüglich +
  – $\alpha + \beta \equiv \beta + \alpha$
- Neutrale Elemente bezüglich + und $\circ$
  – $\emptyset + \alpha \equiv \alpha \equiv \alpha + \emptyset$
  – $\epsilon\alpha \equiv \alpha \equiv \alpha\epsilon$
  – Nullement bezüglich $\circ$
    * $\emptyset\alpha \equiv \emptyset \equiv \alpha\emptyset$
- Distributivität
  – $\alpha(\beta + \gamma) \equiv \alpha\beta + \alpha\gamma$
  – $(\alpha + \beta)\gamma \equiv \alpha\gamma + \beta\gamma$
  – Indepotenz
  – $(\alpha^*)^* \equiv \alpha^*$
  – $\emptyset^* \equiv \epsilon$
  – $\epsilon^* \equiv \epsilon$

1.2 Endliche Automaten

Definition 1.13 E
ndlicher Automat $\mathcal{A} = (Q, \Sigma, \delta, s, F)$

- endliche Menge Q von Zuständen
- Eingabealphabet Σ
- Überführungsfuntion $\delta : Q \times \Sigma \to Q$
- Startzustand $s \in Q$
- akzeptierende Zustände $F \subseteq Q$

Definition 1.14 Semantik endlicher Automaten

- Die erweiterte Überführungsfunktion $\delta^* : Q \times \Sigma^* \to Q$ eines Automaten $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ ist wie folgt induktiv definiert:
  - $\delta(q, \epsilon) =_{def} q$,
  - $\delta(q, u\sigma) =_{def} \delta(\delta^*(q, u), \sigma,)$ für $u \in \Sigma^*, \sigma \in \Sigma$
- $\mathcal{A}$ akzeptiert $w \Leftrightarrow_{def} \delta^*(s, w) \in F$
- Von $\mathcal{A}$ entschiedene Sprache:

$$L(\mathcal{A}) =_{def} \{w \in \Sigma^* | \mathcal{A} \text{ akzeptiert } w\}$$

Bemerkung Notation

- $\#_\tau(w)$: Häufigkeit des Vorkommes des Zeichens τ im String w
- $k|n :$ k ist ein Teiler von n

Definition 1.15 Nichtdeterministischer endlicher Automat
Der NEA $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ besteht aus [a]

- Zustandmenge Q,
- einem Eingabealphabet Σ,
- einem Anfangszustand $s \in Q$,
- einer akzeptierenden Menge F von Zuständen, sowie
- einer Übführungsfunktion $\delta : Q \times \Sigma \to \mathcal{P}(Q)$.

[a] Bei NEAs sind $\delta(q, \sigma)$ Mengen von Zuständen.

Definition 1.16 Semantik von NEAs

- $\delta^*(q, w) =_{def}$ Menge der Zustände, die $\mathcal{A}$ von q aus nach Lesen von w erreichen kann induktiv:
  - $\delta^*(q, \epsilon) = \{q\}$
  - $\delta^*(q, u\sigma) = \bigcup_{p \in \delta^*(q,u)} \delta(p, \sigma)$
- $\mathcal{A}$ akzeptiert w, falls $\delta^*(s, w) \cap F \neq \emptyset$
- $L(\mathcal{A}) =_{def} \{w \in \Sigma^* | \mathcal{A} \text{ akzeptiert } w\}$

Satz 1.17
Zu jeden nichtdeterministischen endlichen Automaten $\mathcal{A}$ gibt es einen (deterministischen) endlichen Automaten $\mathcal{A}_D$ mit $L(\mathcal{A}_D) = L(\mathcal{A})$

Definition 1.18 Nichtdeterministischer endlicher Automat
Ein **ein nichtdeterministischer endlicher Automat** $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ **mit ϵ-Übergängen** besteht aus

- einer Zustandsmenge Q,
- einem Eingabealphabet Σ,
- einem Anfangszustand $s \in Q$,
- einer Menge F von akzeptierenden zustanden
- sowie einer Überführungsfunktion

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$$

Definition 1.19

- **ϵ-closure**$(q) =_{def}$ Menge aller von q aus ohne Lesen eines Symbols erreichbaren Zustände
- $\delta^*(q, \epsilon) =_{def}$ ϵ-closure(q)
- $\delta^*(q, u\sigma) =_{def} \bigcup_{q' \in \delta^*(q,u)} \epsilon\text{-closure}(\delta(q', \sigma))$

Satz 1.20

Zu jeden ϵ-NEA $\mathcal{A}$ gibt es einen EA $\mathcal{A}_D$ mit $L(\mathcal{A}_\mathcal{D}) = L(\mathcal{A})$

Satz 1.21

Zu jeden RA α gibt es einen ϵ-NEA $\mathcal{A}$ mit $L(\alpha) = L(\mathcal{A})$

Satz 1.22

Zu jeden EA $\mathcal{A}$ gibt es einen RA α mit $L(\alpha) = L(\mathcal{A})$

Satz 1.23
Für eine Sprache $L \subseteq \Sigma^*$ sind äquivalent:

(a) $L = L(\alpha)$ für einen RA α

(b) $L = L(\mathcal{A})$ für einen EA $\mathcal{A}$

(c) $L = L(\mathcal{A})$ für einen NEA $\mathcal{A}$

(d) $L = L(\mathcal{A})$ für einen ϵ-NEA $\mathcal{A}$

Satz 12.1 Pumping-Lemma für Kontextfreie Grammatiken
Ist L kontextfrei, so gibt es ein $n \in \mathbb{N}$, so dass sich jeder String $z \in L$ mit $|z| \geq n$ so in $z = uvwxy$ zerlegen lässt, das gelten:

(1) $vx \neq \epsilon$,

(2) $|vwx| \leq n$,

(3) $uv^k wx^k y \in L$, für alle $k \geq 0$

Korollar 12.2
- Sei L eine Sprache
- Angenommen, für jedes $n > 0$ gibt es einen String $z \in L$ mit $|z| \geq n$ so dass für jede Zerlegung $z = uvwxy$ mit
  (1) $vx \neq \epsilon$,
  (2) $|vwx| \leq n$
  ein $k \geq 0$ existiert mit $uv^k wx^k y \notin L$
- Dann ist L nicht kontextfrei.

Proposition 12.3
Die beiden folgenden Sprachen sind nicht kontextfrei:

(a) $L_{abc} = \{a^m b^m c^m \mid m \geq 1\}$

(b) $L_{\text{doppel}} = \{ww \mid w \in \{a, b\}^*\}$

Bemerkung Umwandlungsalgorithmen
- Die folgenden Umwandlungen sind in **linearer Zeit** möglich und erzeugen Objekte **linearer Größe**:
  - kontextfreie Grammatik → Kellerautomat
  - Kellerautomat mit leerem Keller → Kellerautomat mit akzeptierenden Zuständen
  - Kellerautomat mit akzeptierenden Zuständen → Kellerautomat mit leerem Keller
- Die Umwandlung eines **Kellerautomanten $\mathcal{A}$ in eine Grammatik** ist in Zeit $\mathcal{O}(|\mathcal{A}^4|)$ möglich. (Dabei bezeichnet $|\mathcal{A}|$ die Größe der Überführungsfunktion, wobei jede Transition 1+ Länge des neuen Kellerstring beiträgt.)
- Die Umwandlung einer **kontextfreien Grammatik G in eine CNF-Grammatik** ist in Zeit $\mathcal{O}(|G^2|)$ möglich.
- Es gibt einen Algorithmus der eine Grammatik in eine **Greibach-Normalform** der Größe $\mathcal{O}(|G^3|)$ umwandeln kann.

Satz 12.4 Leerheitsproblem für kontextfreie Grammatiken
Zu einer gegebenen kontextfreien Grammatik G lässt sich in linearer Zeit $\mathcal{O}(|G|)$ entscheiden, ob $L(G) \neq \emptyset$ gilt.

Satz 12.5 Endlichkeitsproblem für kontextfreie Grammatiken
Das Endlichkeitsproblem für kontextfreie Grammatiken in CNF kann in linearer Zeit entschieden werden.

Bemerkung Unentscheidbare Probleme für kontextfreie Grammatiken
Folgende Probleme sind nicht Algorithmisch lösbar:

(1) Ist $L(G)$ eindeutig oder inhärent mehrdeutig?

(2) Ist G eindeutig?

(3) Ist $L(G)$ deterministisch kontextfrei?

(4) Ist $L(G)$ regulär?

(5) Ist $L(G_1) \cap L(G_2) = \emptyset$?

(6) Ist $L(G_1) \cap L(G_2)$ kontextfrei?

(7) Ist $L({}_1G) \subseteq L(G_2)$?

(8) Ist $L(G_1) = L(G_2)$?

(9) Ist $L(G) = \Sigma^*$?

Satz 12.6 Abschlusseigenschaften von kontextfreien Sprachen
Die Klasse der kontextfreien Sprachen ist abgeschlossen unter

- Vereinigung,
- Konkatenation,
- dem $*-$Operator und
- dem $+-$Operator.

Satz 12.7
(a) Ist L kontextfrei, so auch $\{w^R | w \in L\}$

(b) Ist $L \subseteq \Sigma^*$ kontextfrei, $h : \Sigma^* \to \Gamma^*$ ein Homomorphismus, so ist auch $h(L)$ kontextfrei.

(c) Ist $L \subseteq \Sigma^*$ kontextfrei, $h : \Gamma^* \to \Sigma^*$ ein Homomorphismus, so ist auch $h^{-1}(L)$ kontextfrei.

Satz 12.8 Weitere Abschlusseigenschaften
Die Klasse der kontextfreien Sprachen ist **nicht** abgeschlossen unter

(a) Durchschnitt,

(b) Komplement,

(c) Mengendifferenz

Satz 12.9
Ist L_1 kontextfrei und L_2 regulär, so ist $L_1 \cup L_2$ kontextfrei.

Satz 12.10
Die Klasse der **deterministisch** kontextfreien Sprachen ist abgeschlossen unter Komplementbildung.

Satz 12.11
Die Klasse der **deterministisch** kontextfreien Sprachen **nicht** abgeschlossen unter Vereinigung und Durchschnitt.

Bemerkung
Die folgenden Sprachen sind kontextfrei, aber nicht deterministisch kontextfrei:

- $L_{\text{doppel}} := \{ww \mid w \in \{0,1\}^*\}$
- $L_{\text{undoppel}} := \overline{L_{\text{doppel}}}$
  Sprich L_{undoppel} enthält alle String ungerader Länge sowie alle Strings aus $L_{\text{diff}} = \{w_1 w_2 \mid |w_1| = |w_2|, w_1 \neq w_2\}$

Satz 12.12
L_{undoppel} ist kontextfrei, aber nicht deterministisch kontextfrei.

Algorithmus 13.1 CYK-Algorithmus
Eingabe: $w \in \Sigma^*, G$ in CNF
Ausgabe: "ja", falls $w \in L(G)$

```
1:  for i := 1 TO n do
2:      V_{i,i} := {X ∈ V | X → w[i,i] in P}
3:  for l := 1 TO n − 1 do
4:      for i := 1 TO n − l do
5:          V_{i,i+1} := ∅{Teilstrings der Längel + 1}
6:          for k := i TO i + l − 1 do
7:              V_{i,i+l} := V_{i,i+l} ∪ {X | X → YZ, Y ∈ V_{i,k}, Z ∈ V_{k+1,i+l}}
8:  if S ∈ V_{1,n} then
9:      Akzeptieren
10: else
11:     Ablehnen
```

Algorithmus 13.2 Erweiterter CYK-Algorithmus
Eingabe: $w \in \Sigma^*, G$ in CNF
Ausgabe: Ableitungsbaum, falls $w \in L(G)$

```
1:  for i := 1 TO n do
2:      V_{i,i} := {(X,i) ∈ V | X → w[i,i] in P}
3:  for l := 1 TO n − 1 do
4:      for i := 1 TO n − l do
5:          V_{i,i+1} := ∅{Teilstrings der Längel + 1}
6:          for k := i TO i + l − 1 do
7:              V_{i,i+l} := V_{i,i+l} ∪ {(X,k) | X → YZ, ∃m : (Y,m) ∈ V_{i,k}, Z ∈ V_{k+1,i+l}}
8:  if es gibt kein k mit (S,k) ∈ V_{1,n} then
9:      Ablehnen
10: Konstruiere Ableitungsbaum rekursiv durch Aufruf von Tree(S,1,n)
```

Algorithmus Tree
Eingabe: X, i, j **Ausgabe**: Ableitungsbaum für $w[i,j]$ aus X

```
1:  if i = jthen
2:
3:
4:
5:
```

Bemerkung $p_k(v)$
Mit $p_k(v)$ bezeichnen wir das Präfix der Länge k ei**Eingabe**ne String v:

$$p_k(v) =_{\text{def}} \begin{cases} v_1 \cdots v_k & , \text{ falls } k \leq n \\ v_1 \cdots v_n & , \text{ falls } k > n \end{cases}$$

Definition LR(k)-Grammatik
Sei $k \geq 0$. Eine Grammatik heißt **LL(k)-Grammatik**, falls die folgenden Bedingung, für alle $X \in V,\ z, x, y \in \Sigma^*$ und $\alpha, \beta, \gamma \in (\Sigma \cup V)^*$, erfüllt ist:

- Falls
  * $S \Rightarrow_l^* uX\alpha \Rightarrow_l u\beta\alpha \Rightarrow_l^* ux$,
  * $S \Rightarrow_l^* uX\alpha \Rightarrow_l u\gamma\alpha \Rightarrow_l^* uy$, und
  * $p_k(u) = p_k(y)$
- so ist $\beta = \gamma$

Bemerkung
- Für einen String $\alpha \in (V \cup \Sigma)^*$ sei

$$\text{FIRST}(\alpha) =_{\text{def}} \{\sigma \in \Sigma \mid \alpha \Rightarrow^* \sigma v, v \in \Sigma^*\} \cup \{\epsilon \mid \alpha \Rightarrow^* \epsilon\}$$

- Für eine Variable X sei

$$\text{FOLLOW}(X) =_{\text{def}} \{\sigma \in \Sigma \mid S \Rightarrow^* uX\sigma v,\ \ u, v \in \Sigma^*\}$$

Satz 13.3
Eine kontextfreie Grammtik G ohne nutzlose Variablen und ohne Linksrekursion ist genau dann eine LL(1)-Grammatik, wenn für alle Variablen X von G und alle Regeln $X \to \alpha$ und $X \to \beta$ mit $\alpha \neq \beta$ gilt:

- FIRST(α)$\cap$FIRST(β)=$\emptyset$
- Falls $\alpha \Rightarrow^* \epsilon$ so ist FOLLOW(X)$\cap$FIRST(β)=$\emptyset$

Definition LR(k)-Grammatik
Sei $k \geq 0$. Eine Grammatik heißt **LR(k)-Grammatik**, falls die folgenden Bedingung, für alle $X, Y \in V,\ z, x, y \in \Sigma^*$ und $\alpha, \beta, \gamma \in (\Sigma \cup V)^*$, erfüllt ist:

- Falls
  * $S \Rightarrow_r^* \alpha X u \Rightarrow_r \alpha\beta u$,
  * $S \Rightarrow_r^* \gamma Y u \Rightarrow_r \alpha\beta u$, und
  * $p_k(u) = p_k(y)$
- so gelten:
  * $x = y$
  * $\alpha = \gamma$, und
  * $X = Y$

Bemerkung
In den Bezeichnungen LL(k) und LR(k) steht der erste Buchstabe (L) für die Leserichtung des Worts. Der zweite Buchstabe für die Ableitungsrichtung (Links oder Rechts).

Satz 13.4
Sei $k \geq 1$, dann sind äquivalent:

- L ist deterministisch kontextfrei
- $L = L(G)$ für eine LR(k)-Grammatik G

Satz 13.5 Folgerungen
(a) Für jedes $k \geq 1$ sind LR(k)-Grammatiken und LR(1)-Grammatiken gleich ausdrucksstark

(b) Das Syntaxanalyseproblem für LR(k)-Grammatiken lässt sich in linearer Zeit lösen.

Satz 13.6 Folgerungen

Jede deterministische kontextfreie Sprache hat eine eindeutige Grammatik.

Bemerkung zu LL(k)- und LR(k)-Grammatiken

- LR(0)-Grammatiken entsprechen gerade den deterministischen Kellerautomanten, die mit llrem Keller akzeptieren.
- Jede LL(k)-Grammatik ist auch eine LR(k)-Grammatik, jedoch **nicht** umgekehrt!
- LR(k)-Grammatiken sind eindeutig.

2 Berechenbarkeit

2.1 LOOP-Programme

Definition Syntax von LOOP-Programmen
Die Syntax von **LOOP-Programmen** ist wie folgt definiert:
Wertzuweisung:

- $x_i := c$
- $x_i := x_j$
- $x_i := x_j + c$
- $x_i := x_j - c$

sind LOOP-Programme.
Reihung: Sind P_1 und P_2 LOOP-Programme, so auch

$$P_1; P_2$$

Wiederholung: Falls P ein LOOP-Programm ist, so auch

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

Definition Speicherinhalte

- Ein **Speicherinhalt** X ist eien Funktion $(\mathbb{N} - 0) \to \mathbb{N}$, für die $X(i) \neq 0$ nur für endliche viele $i \in (\mathbb{N} - 0)$ gilt:

  $$X(i) \text{ repräsentiert den Wert von } x_i$$

- Der Speicherinhalt X_b bei Eingabe $b \in \mathbb{N}$ ist definiert durch:

$$X_b(i) =_{def} \begin{cases} b & \text{für } i = 1 \\ 0 & \text{sonnst} \end{cases}$$

Definition Semantik von LOOP-Programmen

- Ist X ein Speicherinhalt und P ein LOOP-Programm, so bezeichne $P(X)$ den **Speicherinhalt nach Bearbeitung von P**
- Dabei ist $X' = P(X)$ induktiv wie folgt definiert:
  - Falls P von der Form $x_i := x_j + c$ ist, ist

    $$X'(k) =_{def} \begin{cases} X(j) + c & \text{für } k = j \\ X(k) & \text{sonst} \end{cases}$$

    Analog für $x_i := c$ und $x_i := x_j$
  - Falls P von der Form $x_i := x_j - c$ ist, ist

    $$X'(k) =_{def} \begin{cases} \max(X(j) - c, 0) & \text{für } k = i \\ X(k) & \text{sonst} \end{cases}$$

  - Falls P von der Form $P_1; P_2$ ist, so ist

    $$P(X) =_{def} P_2(P_1(X))$$

  - Falls P von der Form $\texttt{LOOP}\ x_j\ \texttt{DO}\ P_1\ \texttt{END}$ ist, so ist $P(X) =_{def} P_1^{X(i)}(X)$ ($X(i)$-malige Wiederholung).
- Die durch ein LOOP-Programm P berechnete Funktion f_P ist wie folgt definiert:
  Für jedes $b \in \mathbb{N}$ ist $f_P(b) =_{def} X'(1)$ für die Speicherbelegung $X' = P(X_b)$

Definition LOOP-berechenbar
Eine Funktion $f : \mathbb{N} \to \mathbb{N}$ heißt LOOP-berechenbar, falls $f = f_P$ für ein LOOP-Programm P

2.2 WHILE-Programme

Definition WHILE-Programme
Die Synatax von WHILE-Programmen entspricht der von LOOP-Programmen, mit der zusätzlichen Regel:

- Ist P ein WHILE-Programm, so auch

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

Proposition 15.1

f LOOP-berechnebar $\Rightarrow$ f WHILE-berechenbar

Definition Semantik von WHILE-Schleifen

- Ist P von der Form $\texttt{WHILE}\ x_i \neq 0\ \texttt{DO}\ P_1\ \texttt{END}$ und X ein Speicherinhalt, so sei

$$P(X) =_{def} \begin{cases} X & \text{falls } X(i) = 0 \\ P(P_1(X)) & \text{sonst} \end{cases}$$

- Die Funktion f_P ist Analog zu LOOP-Programmen definiert
- $f : \mathbb{N} \to \mathbb{N}$ heißt **WHILE-berechenbar**, falls $f = f_P$ für ein WHILE-Programm P

2.3 GOTO-Programme

Definition GOTO-Programm

- Ein **GOTO-Programm** besteht aus einer Folge von Anweisungen A_i und Sprungmarken M_i
  - $M_1 : A_1$;
  - $M_2 : A_2$;
  - ...
  - $M_k : A_k$;
- Mögliche Anweisungen
  - **Wertzuweisungen:**
    - $x_i := c$
    - $x_i := x_j$
    - $x_i := x_j + c$
    - $x_i := x_j - c$
  - **Bedingter Sprung:** $\texttt{IF}\ x_j = c\ \texttt{THEN GOTO}\ M_j$
  - **Stopanweisung:** HALT

Definition Konfiguration GOTO-Programme

- Eine **Konfiguration eines GOTO-Programmes** P ist ein $(l; X)$, wobeil M_l eine Sprungmarke von P und X ein Speicherinhalt ist.
- **Start-Konfiguration** bei Eingabe $b : (1; X_b)$
- Die **Nachfolge-Konfiguration** (l', X') einer Konfigartion $(l; X)$ ist wie folgt definiert:
  - Ist A_l eine Wertzuweisung, so ist $l' = l + 1$ und X' ist definert wie bei LOOP-Programmen
  - Falls A_l ein bedingert Sprung `IF` $x_i = c$ `THEN GOTO` M_j ist, so ist

$$X' =_{def} X \text{ und } l' = \begin{cases} j & \text{falls } X(i) = c \\ l+1 & \text{sonst} \end{cases}$$

- **Halte-Konfiguration** $(l; X)$: $l = k + 1$ oder A_l ist `HALT`
  $f_P(b)$ ist dann wieder $X(1)$
- **GOTO-berechenbar:** analog WHILE-berechenbar

HIER FOLIE 15.17

Satz 15.2

Jede GOTO-berechenbare Funktion ist auch WHILE-berechenbar.

Satz 15.3

Jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar.

2.4 Rekursionstheorie

Definition primitiv rekursive Funktionen

Basisfunktionen:

- Alle Funktionen $c^k : \mathbb{N}^k \to \mathbb{N}$ mit $c \in \mathbb{N}$, definiert durch

$$c^k(x_1, \ldots, x_k) =_{def} c$$

  sind primitiv rekursiv (**konstante Funktionen**)

- Alle Funktionen $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$, definiert durch

$$\pi_i^k(x_1, \ldots, x_k) =_{def} x_i$$

  sind primitiv rekursiv (**Projektionen**)

- Die Funktion $s : \mathbb{N} \to \mathbb{N}$, definiert durch

$$s(x_1) =_{def} x_1 + 1$$

  ist primitiv rekursiv (**Nachfolgerfunktion**)

Zusammengesetzte Funktionen:

- Sind $h : \mathbb{N}^l \to \mathbb{N}$ und $g_1, \ldots, g_l : \mathbb{N}^k \to \mathbb{N}$ primitiv rekursive Funktionen, so ist die durch

$$f(x_1, \ldots, x_k) =_{def} h(g_1(x_1, \ldots, x_k), \ldots, g_l(x_1, \ldots, x_k))$$

  definierte Funktion $f : \mathbb{N}^k \to \mathbb{N}$ primitiv rekursiv (**Komposition**)

- Sind $h : \mathbb{N}^{k+1} \to \mathbb{N}$ und $g : \mathbb{N}^{k-1} \to \mathbb{N}$ primitiv rekursive Funktionen, so ist die durch

$$\begin{aligned} f(0, x_2, \ldots, x_k) &=_{def} g(x_2, \ldots, x_k) \\ f(x+1, x_2, \ldots, x_k) &=_{deF} h(f(x, x_2, \ldots, x_k), x, x_2, \ldots, x_k) \end{aligned}$$

  definierte Funktion $f : \mathbb{N}^k \to \mathbb{N}$ primitiv rekursiv (**Primitive Rekursion**)

Satz 15.4
Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist genau dann primitiv rekursiv, wenn sie durch ein LOOP-Programm berechnet werden kann.

Definition μ-rekursive Funktionen

- Ist $f : \mathbb{N}^{k+1} \to \mathbb{N}$ eine partielle Funktion, so sei die partielle Funktion $\mu f : \mathbb{N}^k \to \mathbb{N}$, definiert durch $\mu f(x_1, \ldots, x_k) =_{def}$ kleinstes $n \in \mathbb{N}$, für das gilt:

$$\begin{aligned} f(n, x_1, \ldots, x_k) &= 0 \\ f(m, x_1, \ldots, x_k) &\neq \perp, \text{ für alle } m < n \end{aligned}$$

- Die μ-rekursive Funktionen sind wie die primitiv rekursiven Funktioen definiert mit der zusätzlichen Regel: Ist f μ-rekursiv, so auch μf

Satz 15.5
Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist genau dann μ-rekursiv, wenn sie druch ein WHILE-Programm berechnet werden kann.

3 Komplexitätstheorie

Definition RucksackO
Geben: Gewichtsschranke G und m Gegenstände repräsentiert durch

- Werte $w_1, \ldots, W_m$ und
- Gewichte $g_1, \ldots, g_m$

Gesucht: $I \subseteq \{1, \ldots, m\}$, so dass $\sum_{i \in I} w_i$ maximal ist und $\sum_{i \in I} g_i \leq G$ gilt

Definition Col
Geben: Ungerichteter Graph G, Zahl k.
Frage: Lassen sich die Knoten von G mit k Farben **zulässig färben**, also so, dass durch Kanten verbundene Knoten verschiedene Farben haben?

Definition EulerKreis
Geben: Ungerichteter Graph G
Frage: Gibt es eine geschlossen Kantenfolge in G, die **jede Kante** genau einmal besucht?

Satz 15.1 [Euler]
Ein zusammenhänender Graph G hat genau dann einen Euler-Kreis, wenn jeder Knoten geraden Grad hat

Definition HamiltonKreis
Geben: Ungerichteter Graph G
Frage: Gibt es einen geschlossenen Weg in G, der jeden Knoten genau einemal besucht?

Definition CliqueO
Geben: Graph $G = (V, E)$
Gesucht: Maximales C mit $C \subseteq G$, so dass C ein K^n. Sprich der größte vollständige Teilgraph von G

Definition Sat
Geben: Aussagenlogische Formel φ in KNF
Frage: Ist φ erfüllbar?

Definition 3-Sat
Geben: Aussagenlogische Formel φ in KNF mit je **3** Literalen je Klausel **Frage:** Ist φ erfüllbar?

Definition TSP
Geben: Gesucht:

Definition TSPO
Geben: Gesucht:

Definition TSPW
Geben: Gesucht:

Definition 3-Col
Geben: Gesucht:

Definition NTIME

Geben: Gesucht:

3.1 Turingmaschinen

Definition Turingmaschiene

Eine Turingmascheine $M = (Q, \Gamma, \delta, s)$ besteht aus

- einer endlichen Menge Q, (**Zustandsmenge**)
- einem Alphabet Γ, mit $\triangleright, \sqcup \in \Gamma$, (**Arbeitsalphabet**)
- einer Funktion $\delta : Q \times \Gamma \to (Q \cup \{\text{ja,nein}, h\}) \times \Gamma \times \{\leftarrow, \uparrow, \rightarrow\}$ (**Überführungsfunktion**)
- einem ausgezeichneten Zustand $s \in Q$, (**Startzustand**)

Definition Konfiguration von Turingmaschienen

Eine **Konfiguration** von **M** ist eine Tupe $(q, (w, z))$ mit

- $q \in Q \cup \{\text{ja,nein}, h\}$ (aktueller Zustand)
- w der aktuelle String
- z die Position des Zeigers der TM (linker Rand ist 0)

Bemerkung String-Zeigerbeschreibung

- In der Konfiguration nenen wir (w, z) die String-Zeigerbeschreibung, mit String $w \in \Gamma^*$ und Zeigerposition $z \in \mathbb{N}, z \leq |w|$, wobei $w[0] =_{def} \triangleright$.
- Eine **alternative** String-Zeigerbeschreibungs Notation ist (u, σ, v) anstelle von (w, z), mit $w = w\sigma v$ und $|u| = z$.

Definition Nachfolgekonfiguration

- Sei $K = (q, (w, z))$ eine Konfiguration mit $w[z] = \sigma$ und sei $\delta(q, \sigma) = (q', \tau, d)$ mit $\tau \in \Gamma, d \in \{\leftarrow, \downarrow, \rightarrow\}$
- Dann ist die **Nachfolgekonfiguration** $K' = (q', (w', z'))$ von K wie folgt definiert: (Schreibweise $K \vdash_M K'$)
  - $z' = z + 1$, falls $d = \rightarrow$,
  - $z' = z$, falls $d = \downarrow$,
  - $z' = z - 1$, falls $d = \leftarrow$,
  - $w' = w[z/\tau]$, falls $z < |w|$ oder $d \neq \rightarrow$,
  - $w' = w[z/\tau]\sqcup$, falls $z = |w|$ und $d = \rightarrow$,
- Wir schreiben $K \vdash_M^* K'$, falls es Konfigurationen $K_1, \ldots, K_m$ gibt mit $K \vdash_M K_1 \vdash_M \ldots \vdash_M K_m \vdash_M K'$

Definition Semantik von Turingmaschienen

- Sei $\Sigma \subseteq \Gamma - \{\triangleright, \sqcup\}$ (**Ein-/Ausgabe-Alphabet**)
- Die **Startkonfiguration** von M bei Eingabe $u \in \Sigma^*$ ist $K_0(u) =_{def} (s, (u, 0))$
- $(q, (w, z))$ heißt **Haltekonfiguration**, falls $q \in \{h, \text{ja,nein}\}$
- $K_0, K_1, \ldots, K_t$ heißt **Berechnung von M bei Eingabe** u, falls
  - $K_0 = K_0(u)$
  - $K_i \vdash_M K_{i+1}$ für alle $i < t$, und
  - K_t eine Haltekonfiguration ist
- M **akzeptiert** u, falls $K_0(u) \vdash_M^* (\text{ja}, (w, z))$
- M **lehnt** u **ab**, falls $K_0(u) \vdash_M^* (\text{nein}, (w, z))$, für gewisse $w \in \Sigma^*, z < |w|$
- $M(u) =_{def}$ die (endliche oder unendliche) Berechnung von M bei Eingabe u

Definition Sprachen von Turingmaschienen

- Eine TM M **entscheidet** eine Sprache L, falls für jedes $u \in \Sigma^*$ gilt:
  - $u \in L \Rightarrow M$ akzeptiert u
  - $u \notin L \Rightarrow M$ lehnt u ab
- $L(M) =_{def}$ Menge aller von M akzeptierten Strings

Definition Turing-berechenbar

- $f_M(u) =_{def} v \in \Sigma^*$, falls
  - $K_0(u) \vdash_M^* (h, (v, 0))$ oder
  - $K_0(u) \vdash_M^* (h, (v\tau w, 0))$ für ein $\tau \in \Gamma - \Sigma, w \in \Gamma^*$
- Eine Funktion $f : \Sigma^* \to \Sigma^*$ heißt **Turing-berechenbar**, falls $f = f_M$, für eine Turingmaschine M

3.1.1 Mehrstring-Turingmaschinen

Definition Syntax von Mehrstring-TM

Eine **Mehrstring-Turingmaschine** $M = (Q, \Gamma, \delta, s)$ besteht aus

- einer Menge Q von **Zuständen**,
- einem **Bandalphabet** Γ mit $\sqcup \in \Gamma$ und $\triangleright \in \Gamma$ (wir nennen $\sqcup$ "Blank" und $\triangleright$ "linker Rand"),
- einem **Anfangszustand** $s \in Q$, und
- einer **Zustandsübergangsfunktion**

$$\delta : Q \times \Gamma^k \to (Q \cup \{h, \text{ja,nein}\}) \times \Gamma^k \times \{\leftarrow, \downarrow, \rightarrow\}^k$$

Definition Semantik von Mehrstring-TM

- Sei $M = (Q, \Gamma, \delta, s)$ eine Mehrstring-TM
- **Ein-/Ausgabealphabet:** $\Sigma \subseteq \Gamma - \{\sqcup, \triangleright\}$,
- **Konfiguration** von $M : k+1$-Tupel $(q, s_1, \ldots, s_k)$, wobei
  - $q \in Q$
  - s_i String-Zeigerbeschreibung $K_0(u)$ von M bei Eingabe $u \in \Sigma^*$: $(s, (u, 0), (\varepsilon, 0), \ldots, (\varepsilon, 0))$
- $(q, s_1, \ldots, s_k)$ ist **Haltekonfiguration**, falls $q \in \{h, \text{ja,nein}\}$
- Sei $K = (q, (u_1, z_1), \ldots, (u_k, z_k))$ eine Konfiguration von M und sei, für jedes i, $\sigma_i =_{def} w[i]$
- Ist $\delta(q, \sigma_1, \ldots, \sigma_k) = (q', \tau_1, \ldots, \tau_k, d_1, \ldots, d_k)$ so ist $K' = (q', (u_1', z_1'), \ldots, (u_k', z_k'))$ die **Nachfolgekonfiguration von** K, wenn für alle i gilt:
  - $z_i' = z_i + 1$, falls $d_i =$ "$\rightarrow$",
  - $z_i' = z_i$, falls $d_i =$ "$\downarrow$",
  - $z_i' = z_i - 1$, falls $d_i =$ "$\leftarrow$",
  - $u_i' = u_i'[z_i/\tau_i]$, falls $z_i < |u_i|$ oder $d_i \neq$ "$\rightarrow$",
  - $z_i' = u_i'[z_i/\tau_i]\sqcup$, falls $d_i = |u_i'|$ und $d_i =$ "$\rightarrow$",
- Schreibweise $K \vdash_M K'$ (Spreichweise M erreicht K' von K aus in einem Schritt)
- $K \vdash_M^* K'$: wie bei Turingmaschinen

3.2 NP-Vollständigkeit

Definition Polynomielle Reduktion

- Eine Funktion f heißt **polynomielle Reduktion** von L auf L', falls
  (1) f in polynomieller Zeit berechenbar ist
  (2) und für alle $w \in \Sigma^*$ gilt:
  $$w \in L \Leftrightarrow f(w) \in L'$$
- L heißt polynomiell reduzierbar auf L', falls es einen polynomielle Reduktion von L auf L' gibt.
- Schreibweise: $L \leq_p L'$

Proposition 22.1

- Seien $L, L' \subseteq \Sigma^*, L \leq_p L'$
- Dann gelten:
  (a) wenn $L' \in$**P** dann $L \in$**P**
  (b) wenn $L' \in$**NP** dann $L \in$**NP**

Lemma 22.2
Seien $L, L' \subseteq \Sigma^*, L \leq_p L'$.
Wenn $L \notin$**P** dann $L' \notin$**P**

Definition NP-schwierig
Eine Sprache L heißt **NP-schwierig**, falls für alle $L' \in$**NP** gilt:
$$L' \leq L$$

Proposition 22.3

Das allgemeine Halteproblem **H** ist **NP**-schwierig.

Definition NP-vollständigkeit

Eine Sprache L heißt **NP-vollständig**, falls L **NP**-schwierig ist und $L \in$**NP** gilt.

Satz 22.4
Sei L eine **NP**-vollständige Sprache
(a) Falls $L \in$ **P**, so ist **P**=**NP**.
(b) Falls $L \notin$ **P**, so ist **P**≠**NP**.

Satz 22.5 [Cook 71]

SAT ist **NP**-vollständig.

Lemma 23.1 $\leq_p$ ist Transitiv

- Seien $L_1, L_2, L_3 \subseteq \Sigma^*$ Sprachen
- Falls $L_1 \leq_p L_2$ und $L_2 \leq_p L_3$, so gilt auch $L_1 \leq_p L_3$

Lemma 23.2
Ist $L' \in$ **NP** und $L \leq_p L'$ für ein **NP**-vollständiges Problem L, so ist auch L' **NP**-vollständig.

Proposition 23.3

$$\text{SAT} \leq_p \text{3-SAT}$$

Lemma 23.4

3-SAT ist **NP**-vollständig.

Proposition 23.5

$$\text{3-SAT} \leq_p \text{3-COL}$$

Proposition 23.6

$$\text{3-SAT} \leq_p \textsc{Clique}$$

Proposition 23.7

$$\text{3-SAT} \leq_p \textsc{HamiltonKreis}$$

Proposition 23.8

$$\textsc{HamiltonKreis} \leq_p \text{TSP}$$

Proposition 23.9

$$\text{3-SAT} \leq_p \textsc{Rucksack}$$

Satz 23.10
Die folgenden Probleme sind **NP**-vollständig:

- 3-SAT
- 3-COL
- Clique
- HamiltonKreis
- TSP
- Ruchsack

Index