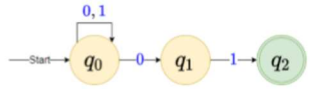
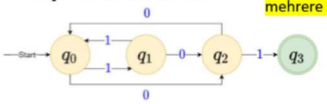
NEA (nicht eindeutig)

- Sprache: $L(M) = \{x01 \mid x \in \{0,1\}^*\}$



- Äquivalenter NEA

Der Automat kann also von einem Zustand in einen Zustand, mehrere Zustände oder auch in keinen Zustand übergehen.

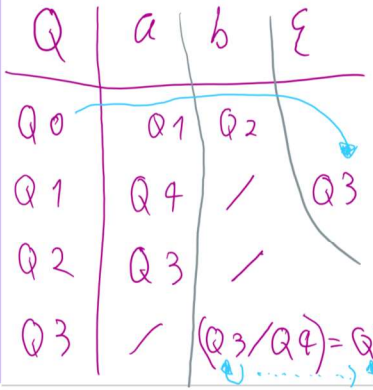


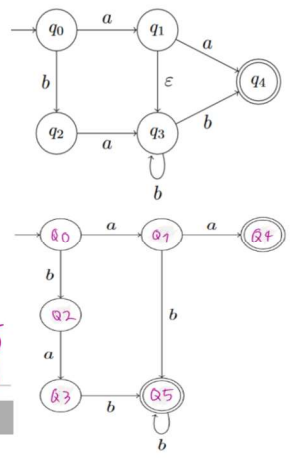
gemeinsame Menge bilden bei nicht deterministisch

Teilmengenkonstruktion

Jeder NEA kann in einen DEA umgewandelt werden. (gleichmächtig)

1. $Q_{NEA} \to P(Q_{NEA}) = Q_{DEA}$ (Potenzmenge)
2. Verbinden mit Vereinigung aller möglichen Zielzustände
3. Nicht erreichbare Zustände eliminieren
4. Enthält akzeptierenden Zustand $= F_{NEA} \to$ akzeptierend

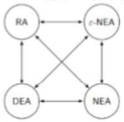




Reguläre Sprachen

Äquivalente Mechanismen

- Akzeptierender Mechanismus DEA, NEA, ε-NEA
- Beschreibender Mechanismus RA



RA = Regulärer Ausdruck = Regex

Äquivalenz DEA und RA

- Es gibt einen DEA, der die Sprache L akzeptiert
- Es gibt einen RA, der die Sprache L akzeptiert.

Bsp. Kontextfreier Grammatik KFG:

(b) $L_1 = \{w \mid w \text{ ist eine Hexadezimalzahl } \geq 32\}$

(b) $G = (\{K, D, Z, A\}, \{0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f\}, P, A)$ mit den Produktionen P:

$K \to 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid a \mid b \mid c \mid d \mid e \mid f$
$D \to 1 \mid K$
$Z \to 0 \mid D \mid ZZ$
$A \to KZ \mid DZZ$

Asymptotische Komplexitätsmessung

O-Notation (Landau Symbole)

- $f \in O(g)$: Es existiert ein $n_0 \in \mathbb{N}$ und ein $c \in \mathbb{N}$, so dass für alle $n \geq n_0$ gilt
  - $f(n) \leq c \cdot g(n) f$ wächst asymptotisch nicht schneller als g
- $f \in \Omega(g)$: Es existiert ein $n_0 \in \mathbb{N}$ und ein $d \in \mathbb{N}$, so dass für alle $n \geq n_0$ gilt
  - $f(n) \geq \frac{1}{d} \cdot g(n) f$ wächst asymptotisch mindestens so schnell wie g
- $f \in \Theta(g)$: Es gilt $f(n) \in O(g(n))$ und $f(n) \in \Omega(g(n))$
  - f und g sind asymptotisch gleich

O Notation

Schranken für die Zeitkomplexität von U

- $O(f(n))$ ist eine obere Schranke, falls

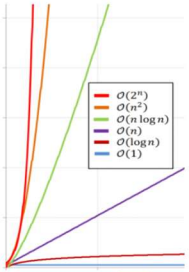
Eine TM existiert, die U löst und eine Zeitkomplexität in $O(f(n))$ hat.

- $\Omega(g(n))$ ist eine untere Schranke, falls

Für alle TM M, die U lösen, gilt dass $\text{Time}_M(n) \in \Omega(g(n))$

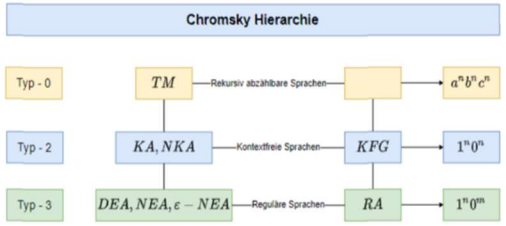
Rechenregeln

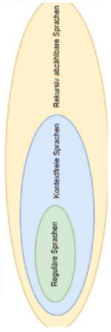
- Konstante Vorfaktoren kann man ignorieren: $c \cdot f(n) \in \mathcal{O}(f(n))$
- Für eine Konstante c gilt: $c \in \mathcal{O}(1)$
- Bei Polynomen ist nur die höchste Potenz entscheidend: $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \in \mathcal{O}(n^k)$
- Die $\mathcal{O}$-Notation ist transitiv: Aus $f(n) \in \mathcal{O}(g(n))$ und $g(n) \in \mathcal{O}(h(n))$ folgt $f(n) \in \mathcal{O}(h(n))$



- $O(n)$ $7n+4$
- $O(n^3)$ $25n^2 + n^3 + 100n$
- $O(n^2 \cdot \log(n))$ $n^2 + n \cdot n \cdot (\log(n)) + 20n^2 + 50n \cdot 100$
- $O(2^n)$ $10^{20} + 3n^3 + 2^n + 2^{10} \cdot 2^{30}$

Chromsky Hierarchie





Kontextfreie Grammatiken

Kontextfreie Grammatik

Eine KontextFreie Grammatik $G(KFG)$ ist ein 4-Tupel (N, Σ, P, A) mit

- N ist das Alphabet der Nichtterminale (Variablen)
- Σ ist das Alphabet der Terminale
- P ist eine endliche Menge von Produktionen mit der Form $X \to \beta$

Mit Kopf $X \in N$ und Rumpf $\beta \in (N \cup \Sigma)^*$

- A ist das Startsymbol, wobei $A \in N$

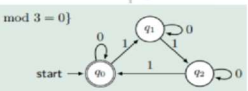
Ein Wort $\beta \in (N \cup \Sigma)^*$ nennen wir Satzform.
Seien α, β und γ Satzformen und $A \to \gamma$ eine Produktion.

- Ableitungsschritt mit Produktion $A \to \gamma$ $\alpha A \beta \to \alpha \gamma \beta$
- Ableitung Folge von Ableitungsschritten $\alpha \to \cdots \to \omega$

DEA zu KFG:

1. Für jeden Zustand q_i gibt es ein Nichtterminal Q_i.
2. Für jede Transition $\delta(q_i, a) = q_j$ erstellen wir die Produktion $Q_i \to aQ_j$.
3. Für jeden akzeptierenden Zustand $q_i \in F$ erstellen wir die Produktion $Q_i \to \varepsilon$.
4. Das Nichtterminal Q_0 wird zum Startsymbol.

$L_5 = \{w \in \{0,1\}^* \mid |w|_1 \mod 3 = 0\}$



Nichtterminale:
Q_0, Q_1, Q_2

Produktionen:
$Q_0 \to 0Q_0 \mid 1Q_1 \mid \varepsilon$
$Q_1 \to 0Q_1 \mid 1Q_2$
$Q_2 \to 0Q_2 \mid 1Q_0$

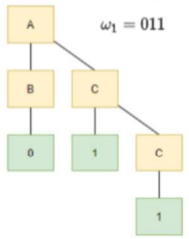
Ableitungsbaum

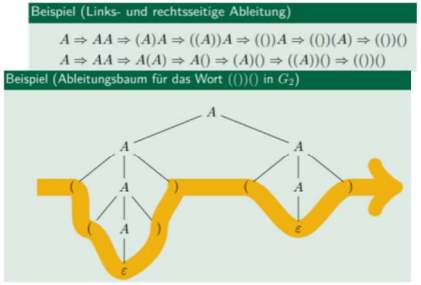
Eine Ableitung kann als Ableitungsbaum / Parsebaum dargestellt werden. KGF G_1 für die Sprache $\{0^n 1^m \mid n, m \in N\}$

- $G_1 = \{\{A, B, C\}, \{0,1\}, P, A\}$
- $P = \{A \to BC, B \to 0B|0|\varepsilon, C \to 1C|1|\varepsilon\}$ ABC = Nichtterminale

Ableitung von $\omega_1 = 011$

- $A \to BC \to 0AA \to 01C \to 011 \to \ldots \to 011$





Mehrdeutigkeit

Eine KFG nennen wir mehrdeutig, wenn es ein Wort gibt, das mehrere Ableitungsbäume besitzt.

Mehrdeutigkeiten eliminieren:

- Korrekte Klammerung vom Benutzer erzwingen
- Grammatik anpassen
- Den Produktionen einen Vorrang vergeben

KFG für Sprache L

Jede reguläre Sprache kann durch eine kontextfreie Grammatik beschrieben werden. Sei L eine reguläre Sprache. Dann gibt es einen DEA $M = (Q, \Sigma, \delta, q_0, F)$ mit $L(M) = L$
Dann können wir einen KFG für L wie folgt bauen:

- Für jeden Zustand q_i gibt es ein Nichtterminal Q_i
- Für jede Transition $\delta(q_i, a) = q_j$ erstellen wir die Produktion $Q_i \to aQ_j$
- Für jeden akzeptierenden Zustand $q_i \in F$ erstellen wir die Produktion $Q_i \to \varepsilon$
- Das Nichtterminal Q_0 wird zum Startsymbol A.

Kellerautomaten

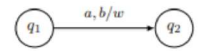
Kellerautomaten haben einen «Speicher». PDA = Push Down Automat. Ist nur im akzeptierten Zustand akzeptiert Stack spielt keine rolle

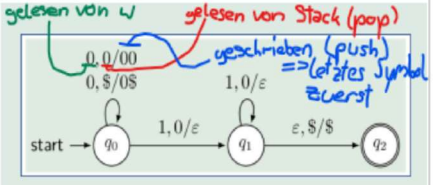
Ein deterministischer Kellerautomat KA ist ein 7-Tupel

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \$, F)$$

- Menge von Zuständen: Q
- Alphabet der Eingabe: Σ
- Alphabet des Kellers: Γ
- Übergangsfunktion: $\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \to Q \times \Gamma^*$
- Anfangszustand: $q_0 \in Q$
- Symbol vom Alphabet des Kellers: $\$ \in \Gamma$
- Akzeptierende Zustände: $F \subseteq Q$

Kellerautomat (KA):
$\delta(q_1, a, b) = (q_2, w)$:

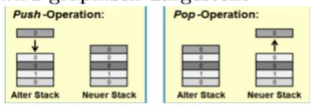




Zusätzliche Einschränkungen für DKAs

Für jeden Zustand q und alle Symbole x, b gilt, wenn $\delta(q, b, c)$ definiert ist, dann ist $\delta(q, \varepsilon, x)$ undefiniert.
Ein Übergang $\delta(q, b, c) = (p, \omega)$ wird graphisch dargestellt

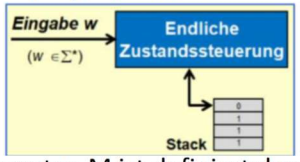
$$q - b, c/\omega \longrightarrow p$$



Berechnungsschritte

Ein Berechnungsschritt $\delta(q, b, c) = (p, \omega)$ wird wie folgt interpretiert

1. Der Automat befindet sich im Zustand q.
2. Der Automat liest das Symbol b von der Eingabe (falls $b = \varepsilon$, wird nichts gelesen).
3. Der Automat entfernt das oberste Kellersymbol c.
4. Der Automat schreibt das Wort w auf den Stack (von hinten nach vorne).
5. Der Automat wechselt in den Zustand p.



Sprache eines Kellerautomaten

Die Sprache L(M) des Kellerautomaten M ist definiert durch

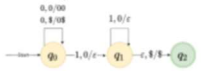
$$L(M) = \left\{ \omega \in \Sigma^* \mid \left(q_0, \omega, \$\right) \vdash^* (q, \varepsilon, \gamma) \text{ für ein } q \in F \text{ und ein } \gamma \in \Gamma^* \right\}$$

Elemente von $L(M)$ werden von M akzeptierte Wörter genannt.

Kellerautomat für eine Sprache erstellen

Ein Kellerautomat für die kontextfreie Sprache $\left\{0^n 1^n \mid n > 0\right\}$

- $0, 0/00$ Read 0 Add 0 $(00 - 0) = 0$
- $0, \$/0\$$ Read 0 Add 0 $(\$0 - \$) = 0$
- $1, 0/\varepsilon$ Read 1 Remove 0 Read $(\varepsilon - 0) = -0$
- $\varepsilon, \$/\$$ Read ε - $(\$ - \$) = \varepsilon$



Eine Sprache ist kontextfrei, wenn sie von einem NKA erkannt wird (nicht unbedingt von einem DKA).
Kontextfreie Sprachen, welche von einem DKA erkannt werden, sind eindeutig.

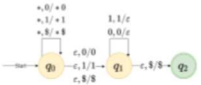
- $\omega_1 = 011 : \left(q_0, 011, \$\right) \vdash \left(q_1, 11, 0\$\right) \vdash \left(q_1, 1, \$\right) \rightarrow \omega_1$ verwerfend

Das Zeichen \$ zeigt an, dass der «Stack» leer ist.

NKA: Übergangsfunktion

- $\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \rightarrow P(Q \times \Gamma*)$

Kellerautomat für die Sprache $\left\{\omega\omega^R \mid \omega \in \{0,1\}^*\right\}$



> Berechnung Beispiel:

$(q_0, 0011, \$) \vdash (q_0, 011, 0\$) \vdash (q_0, 11, 00\$) \vdash (q_1, 1, 0\$)$
$\vdash (q_1, \epsilon, \$) \vdash (q_2, \epsilon, \$)$ ist notwendig, Stack muss nicht leer sein

$\Rightarrow$ Die Berechnung ist akzeptierend.

Eine **Konfiguration von** M ist ein Element (q, w, γ) aus $Q \times \Sigma^* \times \Gamma^*$, wobei

- q für den Zustand steht,
- w die verbleibende Eingabe repräsentiert,
- γ für den Inhalt des Kellers steht.
  (Dabei steht das Symbol ganz links für das oberste Symbol.)

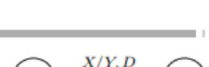
> Beispiel: NKA und/oder DKA erkennbar?

$L_2 = \{waw^R \mid w \in \{0,1\}^*\}, \Sigma = \{0,1,a\}$ ->DKA
$L_3 = \{ww \mid w \in \{0,1\}^*\}, \Sigma = \{0,1\}$ -> /
$L_4 = \{0^n 1^n 0^n \mid n > 0\}, \Sigma = \{0,1\}$ -> /

Turingmaschinen

Turing-Maschine (TM):

$\delta(q_1, X) = (q_2, Y, D)$:



Turingmaschinen (TM)

- Einen Lese- / Schreib-Kopf
- Ein unendliches Band von Zellen

Eine deterministischer Turing-Maschine TM ist ein 7-Tupel

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$$

- Menge von Zuständen: Q
- Alphabet der Eingabe: Σ
- Bandalphabet: Γ und $\Sigma \subset \Gamma$
- Übergangsfunktion: $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times D, D = \{L, R\}$
- Anfangszustand: $q_0 \in Q$
- Akzeptierende Zustände: $F \subseteq Q$
- Leerzeichen $\sqcup$, mit $\mu \in \Gamma$ und $\mu \notin \Sigma$

Sie bildet das 2-Tupel (q, X) auf das Tripel (p, Y, D)

- $q, p \in Q$ und $X, Y \in \Gamma$
- D = Direction
- X = Read
- Y = Overwrite

$q - X/Y, D \rightarrow p$

Wenn TM anhält, dann fertig (akzeptierend falls Zustand akzeptierend). Bandinhalt ist dann das Resultat.

Hält an wenn keine Übergangs Funktion mehr übrig

Band

- Unterteilt in einzelne Zellen mit jeweils einem beliebigen Symbol
- Beinhaltet zu Beginn die Eingabe, d.h. ein endliches Wort aus Σ^*. Alle anderen Zellen enthalten das besondere Symbol 4 .

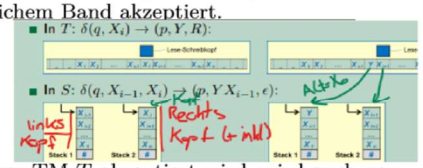
Konfiguration

einer Turing-Maschine M ist durch die folgenden Angaben eindeutig spezifiziert

- Zustand der Zustandssteuerung
- Position des Lese- / Schreibkopfes
- Bandinhalt

Semi-Unendliches Band

Das Band der Turingmaschine ist nur in eine Richtung unendlich.
Jede Sprache L die von einer TM T akzeptiert wird, wird auch von einer TM mit semi-unendlichem Band akzeptiert.



Mehrere Stacks

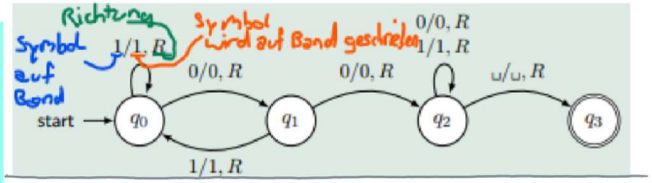
Jede Sprache L die von einer TM T akzeptiert wird, wird auch von einer 2Stack-Maschine S akzeptiert.

$\Rightarrow$ 2 Stack DKA gleichmächtig wie TM.

Zähler-Maschinen

Eine Zähler-Maschine (Counter Machine) mit k Zählern entspricht einer k Stack-Maschine mit dem Unterschied, dass die Stacks durch einfache Zähler ersetzt werden.
Jede Sprache L die von einer TM T akzeptiert wird, wird auch von einer 2Zähler-Maschine Z mit 2 Zählern akzeptiert.

$\Rightarrow$ Zählermaschine mit 2 Zählern kann eine mit 3 Zählern simulieren. Diese kann eine Maschine mit 2 Stack simulieren.



> Universelle TM:

Gödel Nummer

Ein Übergang $\delta(q_i, X_j) = (q_k, X_l, D_m)$ einer TM wird codiert über die Zeichenreihe: lesen schreiben

$0^i 1 0^j 1 0^k 1 0^l 1 0^m$ mit $(i, j, k, l, m \in \mathbb{N})$

$\Rightarrow$ Mit 111 = w (TM#w).
$\Rightarrow$ Mit 11 Abstände zwischen Übergänge.

Wie wird der Übergang $\delta(q_1, 1) = (q_3, 0, R)$ kodiert?

- der Zustand q_1 wird über 0 kodiert
- das Bandsymbol 1 über 00 kodiert
- der Zustand q_3 über 000 kodiert
- das Bandsymbol 0 über 0 kodiert
- und die Bewegung R über 00 kodiert

Das ergibt zusammengesetzt für $\delta(q_1, 1) = (q_3, 0, R)$: 0100100010100

TM mit Speicher

In der endlichen Zustandssteuerung einer TM können ausser dem SteuerZustand zusätzlich endlich viele Daten-Zustände gespeichert werden.

Mehrere Spuren

- Das Band der TM setzt sich aus mehreren «Spuren» zusammen.
- Jede Spur kann ein Symbol des Bandalphabets speichern.

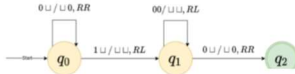
Mehrere Bänder

- TM mit endlich vielen Bändern und Lese- / Schreibköpfen
- Jeder Lese- / Schreibkopf kann unabhängig auf ein Band zugreifen

Bsp:

Mehrband-Maschine

Spezifizieren Sie eine TM M_4, welche die Subtraktion von zwei natürlichen Zahlen $(a - b$, mit $a \geq b)$ realisiert.



Beispiel: $4 - 2 = 2$

			1 2 3 4 5 6 7 8 9
1	$q_0 0000100 \vdash$	$0 \sqcup / \sqcup 0, RR$	0 0 0 0 1 0 0
2	$q_0 \sqcup \vdash$		
1	$\sqcup q_0 00100 \vdash$	$0 \sqcup / \sqcup 0, RR$	0 0 0 1 0 0
2	$0 q_0 \sqcup \vdash$		0
1	$\sqcup\sqcup q_0 0100 \vdash$	$0 \sqcup / \sqcup 0, RR$	0 0 1 0 0
2	$00 q_0 \sqcup \vdash$		0 0
1	$\sqcup\sqcup\sqcup q_0 100 \vdash$	$0 \sqcup / \sqcup 0, RR$	0 1 0 0
2	$000 q_0 \sqcup \vdash$		0 0 0
1	$\sqcup\sqcup\sqcup\sqcup q_0 00 \vdash$	$1 \sqcup / \sqcup\sqcup, RL$	1 0 0
2	$0000 q_0 \sqcup \vdash$		0 0 0 0
1	$\sqcup\sqcup\sqcup\sqcup\sqcup q_1 00 \vdash$	$00 / \sqcup\sqcup, RL$	0 0
2	$000 q_1 0 \vdash$		0 0 0 0
1	$\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup q_1 0 \vdash$	$00 / \sqcup\sqcup, RL$	0
2	$00 q_1 0 \vdash$		0 0 0
1	$\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup q_1$	$\sqcup 0 / \sqcup 0, RR$	
2	$0 q_1 0 \vdash$		0 0
1	$\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup q_2 \sqcup$		
2	$00 q_2 \sqcup \vdash$		0 0

Berechnungsmodelle

Turing-berechenbar

Jedes algorithmisch lösbare Berechnungsproblem kann von einer Turing-Maschine gelöst werden.

- Computer und Turing-Maschinen sind äquivalent.

Turing-berechenbare Funktion: Turing-Maschine $T = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$

$$T : \Sigma^* \to \delta^*$$

$$T(\omega) = \begin{cases} u & \text{falls T auf } \omega \in \Sigma^* \text{ angesetzt, nach endlich vielen Schritten mit u auf dem Band anhält} \\ \uparrow & \text{falls T bei Input } \omega \in \Sigma^* \text{ nicht hält} \end{cases}$$

Primitiv rekursive Grundfunktionen

Für jedes $n \in \mathbb{N}$ und jede Konstante $k \in \mathbb{N}$ die n-stellige konstante Funktion:

$$c_k^n = \mathbb{N}^n \to \mathbb{N} \text{ mit } c_k^n(x_1, ..., x_n) = k$$

Nachfolgerfunktion:

$$\eta : \mathbb{N} \to \mathbb{N} \text{ mit } \eta(x) = x + 1$$

Für jedes $n \in \mathbb{N}$ und jedes $1 < k < n$ die n-stellige Projektion auf die k-te Komponente:

$$\pi_k^n : \mathbb{N}^n \to \mathbb{N} \text{ mit } \pi_k^n(x_1, ..., x_k, ..., x_n) = k$$

n = Anzahl der Argumente, k = Position des Arguments

Loop (primitiv-rekursiv)

- Zuweisungen: $x = y + c$ und $x = y - c$
- Sequenzen: P und $Q \to P; Q$
- Schleifen: $P \to$ Loop x do P until End

Addition von natürlichen Zahlen Add(x, y) = x + y

```
LOOP x1 DO
    x2 = x2 + 1
END
x0 = x2 + 0
```

Nicht gleichmächtig wie TM

X0 ist Return wert

While (Turing vollständig)

Erweiterung deer Sprache Loop

- While $x_i > 0$ do ... until End

Multiplikation von natürlichen Zahlen Mul(x, y) = x * y

```
WHILE x1 > 0 DO
    x1 = x1 - 1
    LOOP x2 DO
        x0 = x0 + 1
    END
END
```

Terminieren nicht immer.
Gleichmächtig zu TM und GOTO.
Zähler ist im While Rumpf änderbar!

Bsp. Primitive Rekursion

$$Add(0, y) = y$$
$$Add(x+1, y) = Add(x, y) + 1$$

$$Add(0, y) = \pi_1^1(y)$$
$$Add(x+1, y) = \eta(\pi_1^3(Add(x, y), x, y)).$$

GoTo (Turing vollständig)

- Zuweisungen: $x_i = x_j + c$ und $x_i = x_j - c$
- Sprunganweisung: IF $x_i = c$ THEN GOTO L_k ELSE GOTO L_t – or simple: GOTO L_k
- Schleifen: WHILE $x_i > 0$ DO ... HALT

Beispiel (Addition)

Das GOTO-Programm

```
M1: x0 = x1 + 0;
M2: If x2 = 0 Then Goto M6;
M3: x2 = x2 - 1;
M4: x0 = x0 + 1;
M5: Goto M2;
M6: Halt
```

berechnet die Addition $Add(x, y) = x + y$.

> Bemerkungen:
- Jede entscheidbare Sprache ist auch semi-entscheidbar.
- Sprache A entscheidbar wenn A und Komplement von A semi-entscheidbar.
- Sprache A entscheidbar, dann auch Komplement von A entscheidbar. (Semi-entscheidbare nicht unbedingt.)
- Alle rekursiven Sprachen sind rekursiv aufzählbar.
- Primitiv rekursive Sprachen sind eine Teilmenge der rekursiven Sprachen.

Entscheidbarkeit

Entscheidbarkeit

- Ein Problem ist entscheidbar, wenn es einen Algorithmus gibt, der für jede Eingabe eine Antwort liefert.
- Ein Problem ist semi-entscheidbar, wenn es einen Algorithmus gibt, der für jede Eingabe eine Antwort liefert, falls die Antwort ja ist.

Eine Sprache $A \subset \Sigma^*$ ist genau dann entscheidbar, wenn sowohl A als auch $\bar{A}$ semi-entscheidbar ist.

- $\bar{A}$ steht für das Komplement von A in Σ^* : $\bar{A} = \Sigma^* \setminus A = \{\omega \in \Sigma^* \mid \omega \notin A\}$

Entscheidbarkeit und Turingmaschinen Eine Sprache $A \subset \Sigma^*$ heisst entscheidbar, wenn eine TM T existiert, die sich wie folgt verhält:

- Bandinhalt $x \in A$ T hält mit Bandinhalt «1» (Ja) an
- Bandinhalt $x \in \Sigma^* \setminus A$ T hält mit Bandinhalt «0» (Nein) an

Äquivalente Aussagen:

- $A \subset \Sigma^*$ ist entscheidbar
- Es existiert eine TM, die das Entscheidungsproblem $T(\Sigma, A)$ löst
- Es existiert ein WHILE-Programm, dass bei einem zu A gehörenden Wort stets terminiert → Entscheidungsverfahren für A

Semi-Entscheidbarkeit Turingmaschinen

Eine Sprache $A \subset \Sigma^*$ heisst semi-entscheidbar, wenn eine TM T existiert, die sich wie folgt verhält:

- Bandinhalt $x \in A$ T hält mit Bandinhalt «1» (Ja) an
- Bandinhalt $x \in \Sigma^* \setminus A$ T hält nie an

Äquivalente Aussagen

- $A \subset \Sigma^*$ ist semi-entscheidbar
- $A \subset \Sigma^*$ ist rekursiv aufzählbar
- Es gibt eine TM, die zum Entscheidungsproblem $T(\Sigma, A)$ nur die positiven («Ja») Antworten liefert und sonst gar keine Antwort
- Es gibt ein WHILE-Programm, dass bei einem zu A gehörenden Wort stets terminiert und bei Eingabe von Wörtern die nicht zu A gehören nicht terminiert

> Ackermannfunktionen: (TM-berechenbar)

⇨ Totale Funktion: nicht Loopberechenbar

Die **Ackermannfunktion** $a : \mathbb{N}^2 \to \mathbb{N}$ ist durch die Gleichungen

(exp. nach Parametern)

$$a(0, m) = m + 1$$
$$a(n+1, 0) = a(n, 1)$$
$$a(n+1, m+1) = a(n, a(n+1, m))$$

gegeben.

Nicht primitiv rekursiv

> Loopinterpreter: (TM-berechenbar)

Ein **LOOP-Interpreter** ist eine Funktion $I : \mathbb{N}^2 \to \mathbb{N}$, die für jedes LOOP-Programm P und jede natürliche Zahl x die Gleichung

$$I(\langle P \rangle, x) = P_1(x)$$

Bezeichne also x den Bytecode eines Programmes P, dann soll für jeden Input y die Gleichung

$$I(x, y) = P(y)$$

Reduzierbarkeit

Eine Sprache $A \subset \Sigma^*$ heisst auf eine Sprache $B \subset \Gamma^*$ reduzierbar, wenn es eine totale, Turing-berechenbare Funktion $F : \Sigma^* \to \Gamma^*$ gibt, so dass für alle $\omega \in \Sigma^*$

$$\omega \in A \Leftrightarrow F(\omega) \in B$$

- $A \preccurlyeq B$ A ist reduzierbar auf B
- $A \preccurlyeq B$ und $B \preccurlyeq C \to A \preccurlyeq C$

Satz

Für beliebige Sprachen $A \subset \Sigma^*$ und $B \subset \Gamma^*$ gilt:

- *Ist B entscheidbar und $A \preceq B$, dann ist auch A entscheidbar.*
- *Ist B semi-entscheidbar und $A \preceq B$, dann ist auch A semi-entscheidbar.*

Halteproblem

Das allgemeine Halteproblem H ist die Sprache (# = Delimiter)

- $H := \{\omega \# x \in \{0, 1, \#\}^* \mid T_\omega$ angesetzt auf x hält $\}$

Sprachen der Halteprobleme (HP): leeres HPH_0 und spezielles HP H_S

- $H_0 := \{\omega \in \{0,1\}^* \mid T_\omega$ angesetzt auf das leere Band hält $\}$
- $H_S := \{\omega \in \{0,1\}^* \mid T_\omega$ angesetzt auf ω hält $\}$

H_0, H_S und H sind semi-entscheidbar.

Im Rahmen des **allgemeinen Halteproblems** "wird gefragt", ob eine gegebene Turingmaschine auf einem gegebenen Input anhält. Das

Komplexitätstheorie

Quantitative Gesetze und Grenzen der algorithmischen Informationsverarbeitung

- Zeitkomplexität: Laufzeit des besten Programms, welches das Problem löst
- Platzkomplexität: Speicherplatz des besten Programms
- Beschreibungskomplexität: Länge des kürzesten Programms

Zeitbedarf Der Zeitbedarf von M auf Eingaben der Länge $n \in \mathbb{N}$ im schlechtesten Fall definiert als

$$\text{Time}_M(n) = \max \{\text{Time}_M(\omega) \mid |\omega| = n\}$$

Sei M eine TM, die immer hält und sei $\omega \in \Sigma^*$. Der Zeitbedarf von M auf der Eingabe ω ist

- Time $_M(\omega)$ = Anzahl von Konfigurationsübergängen in der Berechnung von M auf ω

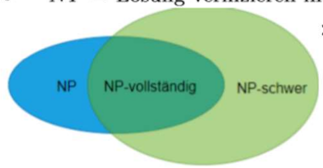
P ≙ *Lösung finden in Polynomzeit*
NP ≙ *Lösung verifizieren in Polynomzeit*

P vs NP Klassifizierung von Problemen

Ein Problem U heisst in Polynomzeit lösbar, wenn es eine obere Schranke $O(n^c)$ gibt für eine Konstante $c \geq 1$.

- $P \doteq$ Lösung finden in Polynomzeit
- $NP \doteq$ Lösung verifizieren in Polynomzeit

NP := Alle polynomzeit endscheidbaren Sprachen mittels einer NTM.



> Polynomzeit-Verifizierer (Alternative NP Def):

Sei $L \subseteq \Sigma^*$ eine Sprache und $p: \mathbb{N} \to \mathbb{N}$ eine Funktion. Eine TM M ist ein **p-Verifizierer** für L, falls M wie folgt auf allen Eingaben $w\#x$ für $w \in \Sigma^*$ und $x \in \{0,1\}^*$ arbeitet:

- $\text{Time}_M(w\#x) \leq p(|w|)$ für alle Eingaben $w\#x$.
- Für jedes $w \in L$ existiert ein $x \in \{0,1\}^*$ mit $|x| \leq p(|w|)$, so dass M die Eingabe $w\#x$ akzeptiert. x heisst **Zeuge** für $w \in L$.
- Für alle $w \notin L$ existiert kein Zeuge.

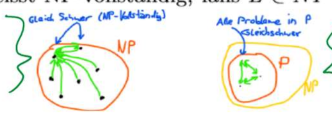
=>Fall $p \in O(n^k)$, $k \in \mathbb{N}$, dann M ein Polynomzeit-V.

NP-schwer

Eine Sprache L heisst NP-schwer, falls für alle Sprachen $L' \in NP$ gilt, dass $L' \preccurlyeq_p L$

NP-Schwer: Wenn alle Sprachen / Probleme in NP auf dieses in polynomieller Zeit reduzierbar sind.

Eine Sprache L heisst NP-vollständig, falls $L \in NP$ und L ist NP-schwer.



Alle Probleme in P gleich schwer

Theoretische Informatik

Lucien Perret, Jil Zerndt
May 2024

Alphabete, Wörter, Sprachen

Alphabete sind endliche, nichtleere Mengen von Symbolen.
- $\Sigma = \{a, b, c\}$ Mengen von drei Symbolen
- $\Sigma_{\text{Bool}} = \{0, 1\}$ Boolsches Alphabet

Keine Alphabete
- $\mathbb{N}, \mathbb{R}, \mathbb{Z}$ usw. (unendliche Mächtigkeit)

Wort ist eine endliche Folge von Symbolen eines bestimmten Alphabets.
- abc Wort über dem Alphabet Σ_{lat} (oder über $\Sigma = \{a, b, c\}$)
- 100111 Wort über dem Alphabet $\{0, 1\}$
- ε Leeres Wort (über jedem Alphabet)

Wortkonventionen

Definition	Beispiel	Beschreibung
$\|w\|$	$\|10011\| = 5$	Wortlänge
$\|w\|_x$	$\|abc\|_a = 1$	Symbolhäufigkeit (X)
w^R	$(abc)^R = cba$	Spiegelwort
$w^R = w$	$(anna)^R = anna$	Palindrom
$x \circ y\ (= xy)$	$ab \circ cd = abcd$	Konkatenation
$\|x \circ y\| = \|x\| + \|y\|$	-	Konkatenationlänge
$w = vy$	$w = \varepsilon abba$ Präfix hier = ε	Präfix v (echt wenn $y \neq \varepsilon$)
$w = xv$	$w = abba\varepsilon$ Suffix hier = ε	Suffix v (echt wenn $x \neq \varepsilon$)
$w = xvy$	$w = aabba$ Infix hier = ab «v» an einem Stück!	Infix (Teilwort) v (echt wenn $\neg(x = \varepsilon \wedge y = \varepsilon)$)
$w^X = www \ldots$	$w^3 = www$ $w^0 = \varepsilon$ $w^{n+1} = w^n \circ w$	Wortpotenz nach X (Achtung: 1. Symbol ist «inkl.» X)
Σ^* $= \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \ldots)$	$\Sigma^+ = \Sigma^* \backslash \{\varepsilon\}$ $(= \Sigma^* - \{\varepsilon\})$	Kleenesche Hülle (immer unendlich)
Σ^k	$\Sigma^0 = \{\varepsilon\}$	Wörter mit Länge k. (nie unendlich)

Konkatenation ist Multiplikation

$AB = \{uv \mid u \in A, v \in B\}$

Bsp: $\{a, ab, aa\}\ \{x, y\}$

$= \{ax, ay, abx, aby, aax, aay\}$

Sprache über einem Alphabet Σ = Eine Teilmenge $L \subseteq \Sigma^*$ von Wörtern.
- $\Sigma_1 \subseteq \Sigma_2 \wedge L$ Sprache über $\Sigma_1 \rightarrow L$ Sprache über Σ_2
- Σ^* Sprache über jedem Alphabet Σ
- $\{\} = \emptyset$ ist die leere Sprache

Konkatenation von zwei Sprachen $A \subset \Sigma^*$ und $B \subset \Gamma^*$

$$AB = \{uv \mid u \in A \text{ und } v \in B\}$$

Die **Kleenesche Hülle** A^* einer Sprache $A = \{\varepsilon\} \cup A \cup AA \cup AAA \cup \ldots$

Das leere Wort ist nicht in der leeren Sprache

Reguläre Ausdrücke

= Regex

Reguläre Ausdrücke sind Wörter, die Sprachen beschreiben.
Die Sprache RA_Σ der Regulären Ausdrücke über einem Alphabet Σ ist wie folgt definiert:
- $\emptyset, \epsilon \in RA_\Sigma$
- $\Sigma \subset RA_\Sigma$
- $R \in RA_\Sigma \Rightarrow (R^*) \in RA_\Sigma$
- $R, S \in RA_\Sigma \Rightarrow (RS) \in RA_\Sigma$
- $R, S \in RA_\Sigma \Rightarrow (R \mid S) \in RA_\Sigma$

$L(R_2)$: Menge der Binärwörter mit abwechselnd Nullen und Einsen

$$R_2 = (1|\varepsilon)(01)^+(0|\varepsilon)$$

Bsp:

Für jeden regulären Ausdruck $R \in RA_\Sigma$ definieren wir die Sprache $L(R)$ von R wie folgt:
- Leere Sprache: $L(\emptyset) = \emptyset$
- Sprache, die nur das leere Wort enthält: $L(\varepsilon) = \{\varepsilon\}$
- Beschreibt die Sprache $\{a\}$: $L(a) = \{a\} \quad \forall a \in \Sigma$
- Kombiniert die Wörter von R: $L(R^*) = L(R)^*$
- Verkettung von Wörtern (R = prefix): $L(RS) = L(R) \circ L(S)$
- Wörter die in R oder S beschrieben werden: $L(R \mid S) = L(R) \cup L(S)$

Reguläre Sprache
Eine Sprache A über dem Alphabet Σ heisst regulär, falls
- $A = L(R)$ für einen regulären Ausdruck $R \in RA_\Sigma$ gilt.

Beispiele
- $R_1 = a^*b \quad L(R_1) = \{b, ab, aab, aaab, \ldots\}$
- $R_2 = (aa)^*b^*aba \quad L(R_2) = \{aba, baba, aaaba, aababa, \ldots\}$
- $R_3 = (a \mid ab)^* \quad L(R_3) = \{\varepsilon, a, ab, aa, abab, \ldots\}$

$L(R_1)$: Menge der ganzen Zahlen in Dezimaldarstellung
- $((- \mid \varepsilon)(1, 2, 3, 4, 5, 6, 7, 8, 9)(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) \mid 0).0$

Eigenschaften und Konventionen Die Menge RA_Σ über dem Alphabet Σ ist eine Sprache über dem Alphabet

$$\{\emptyset, \epsilon, *, (), , |\} \cup \Sigma$$

Priorisierung von Operatoren
- (1) $* =$ Wiederholung $\rightarrow$ (2) Konkatenation $\rightarrow$ (3) $|=$ Oder

Beispiele
- $(aa)^*b^*aba = (aa)^*b^*aba$
- $(ab)|(ba) \quad = ab|ba$
- $a(b(ba))|b \quad = abba|b$

Erweiterte Syntax
- $R^+ = R(R^*)$
- $R? = (R \mid \epsilon)$
- $[R_1, \ldots, R_k] = R_1 |R_2| \ldots | R_k$

Collatz Zahlen sind die die immer auf 4 - 2 - 1 enden
Bildungsvorschrift: Ist n gerade, setze $n = n/2$
Ist n ungerade: setze $n = 3n + 1$

Endliche Automaten

Endliche Automaten entsprechen Maschinen, die Entscheidungsprobleme lösen.
- Links nach rechts
- Keinen Speicher
- Keine Variablen
- Speichert aktuellen Zustand
- Ausgabe über akzeptierende Zustände

Endliche Automat (EA):
$\delta(q_1, a) = (q_2)$:

DEA Ein deterministischer endlicher Automat (DEA) ist ein 5-Tupel $M = (Q, \Sigma, \delta, q_0, F)$
- Q endliche Menge von Zuständen
- Σ endliches Eingabealphabet
- $\delta : Q \times \Sigma \rightarrow Q$ Übergangsfunktion
- $q_0 \in Q$ Startzustand
- $F \subseteq Q$ Menge der akzeptierenden Zustände

DEA Funktionen
$M = (Q, \Sigma, \delta, q_0, F)$ ein EA. **Konfiguration** von M auf ω ist ein Element aus $Q \times \Sigma^*$.
- Startkonfiguration von M auf $\omega \quad \{q_0, \omega\} \in \{q_0\} \times \Sigma^*$
- Endkonfiguration (q_n, ε)

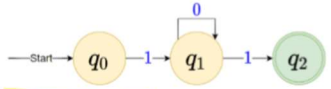
Berechnungsschritt $\vdash_M$ von M

$$(q, \omega) \vdash_M (p, x)$$

Berechnung ist eine endliche Folge von Berechnungsschritten

$(q_a, \omega_1\omega_2 \ldots \omega_n) \vdash_M \ldots \vdash_M (q_e, \omega_j \ldots \omega_n) \rightarrow (q_a, \omega_1\omega_2 \ldots \omega_n) \vdash_M^* (q_e, \omega$

Beispiel DEA (eindeutig)
- Sprache: $L(M) = \{1x1 \mid x \in \{0\}^*\}$



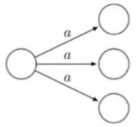
! DEA sind gleichmächtig zu Regex

Konfiguration
- Startkonfiguration auf $\omega = 101 \rightarrow (q_0, 101)$
- Endkonfiguration auf $\omega = 101 \rightarrow (q_2, \varepsilon)$

Berechnung
- $\omega = 101 \rightarrow (q_0, 101) \vdash_M (q_1, 01) \vdash_M (q_1, 1) \vdash_M (q_2, \varepsilon) \rightarrow$ akzeptierend
- $\omega = 10 \rightarrow (q_0, 10) \vdash_M (q_1, 0) \vdash_M (q_1, \varepsilon) \rightarrow$ verwerfend

Nichtdeterministischer endlicher Automat (NEA)
Der einzige Unterschied zum DEA besteht in der Übergangsfunktion δ
- Übergangsfunktion $\delta : Q \times \Sigma \rightarrow P(Q)$

Ein ε-NEA erlaubt zusätzlich noch ε-Übergänge.



> Anmerkung zu NEA:
- Sobald ein Pfad akzeptierend, dann w akzeptierend.
- εNEA: Spontane Zustandsänderung durch ε.