Grundlegende Definitionen

Alphabet
Eine endliche, nichtleere Menge von Symbolen.

Konvention
Alphabet wird durch $\sum$ dargestellt

Beispiel
$\sum = \{a, b, c\}$ ist die Menge der drei Symbole a, b und c

Wörter
Eine endliche Folge von Symbolen eines bestimmten Alphabetes. (Vergleichbar mit einem String)

Konvention
Man sagt über dem Alphabet $\sum$
Wörter werden als Kleinbuchstaben dargestellt.

Beispiel
alain ist ein Wort über dem Alphabet $\sum = \{a, b, c, \dots, z\}$

Leeres Wort
Ein Wort, das keine Symbole enthält. Es wird durch das Symbol ε dargestellt und ist ein Wort über jedem Alphabet.

Länge eines Wortes
Anzahl Symbole eines Wortes. $|w|$
$|1001101| = 7$
Leerzeichen sind auch Symbole

Häufigkeit eines Symbols in einem Wort
Absolute Häufigkeit eines Symbols x in einem Wort w.
$|1001101|_1 = 4$

Spiegelung
Mit w^R wird das Spiegelwort zu w bezeichnet.
$(abc)^R = cba$

Palindrom
Wenn $w = w^R$ bsp. $w = 101$ $w^R = 101$

Teilwort (Infix)
$w = abba$ $Infix = \{\varepsilon, a, b, ab, abb, bb, abba, bba, ba\}$

Echtes Teilwort
Teilwort darf nicht identisch mit dem Wort sein.
$w = abba$ $Infix = \{\varepsilon, a, b, ab, abb, bb, \cancel{abba}, bba, ba\}$

Präfix
$w = abba$ $Präfix = \{\varepsilon, a, ab, abb, abba\}$

Echtes Präfix
Präfix darf nicht identisch mit dem Wort sein.
$w = abba$ $Präfix = \{\varepsilon, a, ab, abb, \cancel{abba}\}$

Suffix
$w = abba$ $Suffix = \{abba, bba, ba, a, \varepsilon\}$

Echtes Suffix
Suffix darf nicht identisch mit dem Wort sein.
$w = abba$ $Suffix = \{\cancel{abba}, bba, ba, a, \varepsilon\}$

Menge aller Wörter der Länge k
$\sum = \{a, b, c\}$ ist $\sum^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$
Es gilt immer $\sum^0 = \{\varepsilon\}$

Kleenesche Hülle
Die Menge aller Wörter über einem Alphabet $\sum$ wird mit $\sum^*$ bezeichnet
Für $\{0,1\}$ ist $\sum^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$

Positive Hülle
$\sum^+ = \sum^* \setminus \{\varepsilon\}$ ist die Menge aller nichtleeren Wörter.

Konkatenation (Verkettung)
$x = adc$ und $y = def$ Dann ist $xy = abcdef$

Wortpotenz
$a^3 = aaa$
$bbababababbaaaabab = b^2(ab)^4ba^4bab$

Sprache
Eine Teilmenge $L \subseteq \sum^*$ von Wörtern über einem Alphabet $\sum$ wird als Sprache über $\sum$ bezeichnet.
$L = \{w | w \text{ enthält nur Einsen}\}$
Wichtig:
- Sprachen können aus unendlich vielen Wörter bestehen.
- Wörter müssen au einem festen, endlichen Alphabet gebildet werden.
- Wörter selber haben eine endliche Länge.
- $\emptyset$ beschreibt die leere Sprache.

Konkatenation
A ist eine Sprache, aller Binärwörter die mit 1 beginnen.
B ist die Sprache, aller Binärwörter die mit 0 enden.
$AB = \{10, 1010, 1100110010, \dots\}$

Kleenesche Hülle
A^* einer Sprache A ist durch $\{\varepsilon\} \cup A \cup AA \cup AAA \dots$ definiert
$(A^*)^* = A^*$
$A = \{aa, ab, ba, bb\}^*$ $\{\varepsilon, abab, abaaab, aabb, \dots\}$

Entscheidungsproblem
Sei eine Sprache L über einem Alphabet Σ gegeben. Das **Entscheidungsproblem** (Σ, L) ist die folgende Berechnungsaufgabe:

Input: Eine Sprache L und ein Wort $x \in \Sigma^*$
Output: JA, falls $x \in L$, und
NEIN, falls $x \notin L$

Beispiel (gerade Zahl)
Gegeben:
- Alphabet $\Sigma = \{0, 1\}$
- Sprache $L_g = \{1w0 \mid w \in \Sigma^*\}$
- Wörter x aus Σ^*

Der Test, ob x eine gerade Zahl grösser Null ist, ist äquivalent zu der Entscheidung, ob $x \in L_g$ gilt.

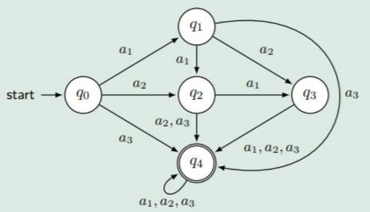
Beispiel (Primzahl)
Gegeben:
- Alphabet $\Sigma = \{0, 1\}$
- Sprache $L_p = \{w \mid w \text{ ist eine Primzahl}\}$
- Wörter x aus Σ^*

Der Test, ob x eine Primzahl darstellt, ist äquivalent zu der Entscheidung, ob $x \in L_p$ gilt.

Reguläre Ausdrücke
Reguläre Ausdrücke sind Wörter, die Sprachen beschreiben.
Die Sonderzeichen ε und $\emptyset$ sind reguläre Ausdrücke.
Jedes Symbol aus dem Alphabet sind auch regexe.
- $\emptyset$ beschreibt die leere Sprache.
- ϵ beschreibt die Sprache $\{\varepsilon\}$, also die Sprache, die nur das leere Wort ε enthält.
- Jedes Symbol $a \in \Sigma$ beschreibt die Sprache $\{a\}$.
- (R^*) beschreibt alle durch Konkatenation kombinierten Wörter, die von R beschrieben werden (kleenesche Hülle).
- $(R|S)$ beschreibt alle Wörter, die von R oder von S beschrieben werden.
- (RS) beschreibt die Wörter, die durch Konkatenation aus einem von R beschriebenen Wort gefolgt von einem durch S beschriebenen Wort entstehen.

Beispiel
Ein regex, der die Sprache aller Binärwörter der Länge 4 beschreibt:
⇨ $(0|1)(0|1)(0|1)(0|1)$
Ein regex für die Sprache der Binärwörter, die das Teilwort 00 enthalten.
⇨ $(0|1)^*00(0|1)^*$

Endliche Automaten (EA)
Ein (deterministischer) endlicher Automat (EA) ist ein Quintupel: $M = (Q, \sum, \delta, q_0, F)$
M = endlicher Automat
Q = endliche Menge von Zuständen $\{q_0, q_1, \dots, q_n\} (n \in \mathbb{N})$
$\sum$ = Eingabealphabet $\{a_1, a_2, \dots, a_m\} (m \in \mathbb{N})$
δ = Übergangsfunktion $Q \times \sum \to Q$
q_0 = Startzustand
$F \subseteq Q$ = Menge der akzeptierenden Zustände

Automat A:


Zustände:
q_0 = Start
$q_1 = 0.50$
$q_2 = 1.–$
$q_3 = 1.50$
$q_4 \geq 2$

Eingabealphabet:
$a_1 = 0.50$
$a_2 = 1.–$
$a_3 = 2.–$

Übergangsfunktionen:
$\delta(q_0, a_1) = q_1, \dots$

Konfiguration
Wenn M sich in einer Konfiguration $(q, w) \in Q \times \sum^*$ befindet, bedeutet dies, dass M im Zustand q ist und noch das Suffix w des Eingabewortes lesen soll.
Wie M in die aktuelle Konfiguration gelangt ist, spielt keine Rolle.

Startkonfiguration
Eine Konfiguration $(q_0, w) \in \{q_0\} \times \sum^*$ nennen wir eine **Startkonfiguration von M auf w.**

Beispiel
Die Startkonfiguration mit $w = a_1, a_2, a_2, a_1$ lautet:
$(\boldsymbol{q_0, a_1 a_2 a_2 a_1})$

Endkonfiguration
Jede Konfiguration aus $Q \times \{\varepsilon\}$ nennen wir eine **Endkonfiguration**.

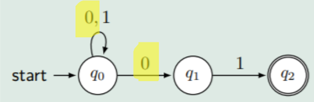
Beispiel
Endkonfiguration von M: $(q_0, \varepsilon), (q_1, \varepsilon), (q_2, \varepsilon), (q_3, \varepsilon)$

Berechnungsschritt
Ein Berechnungsschritt $\vdash_M$ von M ist die Anwendung der Übergangsfunktion auf die aktuelle Konfiguration und ist definiert durch: $(q, w) \vdash_M (p, x)$

Beispiel
Berechnung des Wortes $w = a_1 a_2 a_2 a_1$ auf den Automaten M.
$(q_0, a_1 a_2 a_2 a_1) \vdash_M (q_1, a_2 a_2 a_1) \vdash_M (q_3, a_2 a_1) \vdash_M (q_4, a_1) \vdash_M (q_4, \varepsilon)$

Sprache eines endlichen Automaten
Die **Sprache** $L(M)$ eines endlichen Automaten M ist die Menge aller von M akzeptierenden Wörter.
$L(M) = \{w \in \sum^* \mid \text{Berechnung von } M \text{ auf } w \text{ ist akzeptierend}\}$

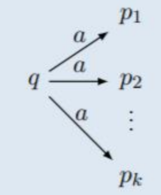
Nichtdeterministischer endlicher Automat
Ein nichtdeterministischer endlicher Automat (NEA) ist ein Quintupel: $M = (Q, \sum, \delta, q_0, F)$
$\delta: Q \times \sum \to P(Q)$
$P(Q)$ = die Potenzmenge von Q, also die Menge aller Teilmengen von Q.
Der Automat kann also von einem Zustand in einen Zustand, mehrere Zustände oder auch in keinen Zustand übergehen.

M_3:


Zustand	Eingabe 0	Eingabe 1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\emptyset$

$L(M_3) = \{x01 \mid x \in \{0,1\}^*\}$

Berechnungsschritt
Ein Berechnungsschritt in einem NEA ist die Anwendung der Übergangsfunktion auf die aktuelle Konfiguration.
Dargestellt wird ein Berechnungsschritt durch k viele Zweige



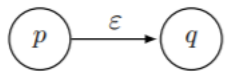
wenn $\delta(q, a) = \{p_1, p_2, \dots, p_k\}$ für ein $k \in \mathbb{N}$ gilt.

Berechnung
Die Berechnung eines NEA auf w startet in der Starkonfiguration.
Anschliessend wird in jedem Zweig des ersten Berechnungsschrittes der nächste Berechnungsschritt durchgeführt.
Das wird für jeden Zweig fortgesetzt, bis der NEA entweder in einer Sackgasse landet oder das ganze Wort gelesen worden ist.

Akzeptierende Berechnung
Eine Berechnung von M auf einer Eingabe w ist akzeptierend, wenn für mindestens ein weg das Wort ganz gelesen wurde.

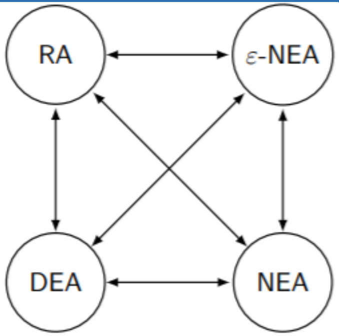
Äquivalenz von (D)EA und NEA
Es gibt einen DEA, der die Sprache L akzeptiert. $\iff$ *Es gibt einen NEA, der die Sprache L akzeptiert.*

NEA mit ε-Übergängen
Der NEA kann spontan den Zustand wechseln, ohne ein Eingabesymbol zu lesen.



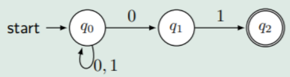
Äquivalenz von (D)EA und ε-NEA
Es gibt einen DEA, der die Sprache L akzeptiert. $\iff$ *Es gibt einen ε-NEA, der die Sprache L akzeptiert.*

Übersicht

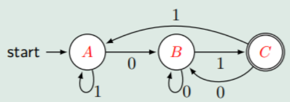


Teilmengenkonstruktion

Ursprünglicher **NEA** M_3:



Resultierender **DEA** M_3':



Zustand	Eingabe 0	Eingabe 1
$\to \{q_0\} = A$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\} = B$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\} = C$	$\{q_0, q_1\}$	$\{q_0\}$

Zustandsklassen

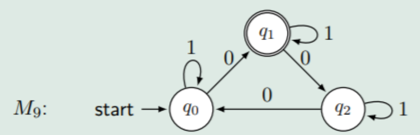
Klasse$[p] = \{w \in \Sigma^* \mid M$ endet nach dem Lesen von w in $p\}$

Jedes Wort landet in einem Zustand.

Kein Wort landet in 2 Zuständen.

$L(M_9) = \{w \in \{0,1\}^* \mid |w|_0 \bmod 3 = 1\}$

M_9:



Zustandsklassen von M_9:

- $[q_0] = \{w \in \{0,1\}^* \mid (\text{Anzahl Nullen in } w) \bmod 3 = 0\}$
- $[q_1] = \{w \in \{0,1\}^* \mid (\text{Anzahl Nullen in } w) \bmod 3 = 1\}$
- $[q_2] = \{w \in \{0,1\}^* \mid (\text{Anzahl Nullen in } w) \bmod 3 = 2\}$

Grenze endlicher Automaten

Wenn ein EA M nach dem Lesen zweier Präfixe x und y im gleichen Zustand landet, kann er nicht mehr zwischen x und y unterscheiden.

Kontextfreie Grammatik (KFG)

Definition

Eine kontextfreie Grammatik G ist ein 4-Tupel $(N, \sum, P, A)$

Mit:

N= Ist das Alphabet der Nichtterminale (Variablen)

$\sum$ = Ist das Alphabet der Terminale

P = Ist eine endliche Menge von Produktionen (Regeln), Jede Produktion hat die Form: $X \to \beta$

A = Ist das Startsymbol

Die kontextfreien Sprachen enthält die regulären Sprachen

Linksseitige- und rechtsseitige-Ableitung

- Eine **linksseitige Ableitung** ersetzt bei jedem Ableitungsschritt das Nichtterminal, das am weitesten *links* in der Satzform auftritt.
- Eine **rechtsseitige Ableitung** ersetzt bei jedem Ableitungsschritt das Nichtterminal, das am weitesten *rechts* in der Satzform auftritt.

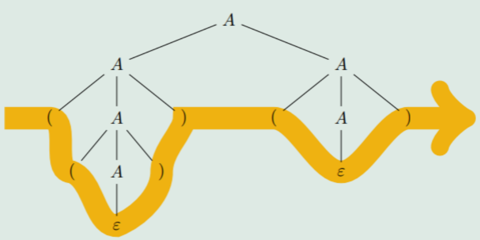
Beispiel (Links- und rechtsseitige Ableitung)

$$A \Rightarrow AA \Rightarrow (A)A \Rightarrow ((A))A \Rightarrow (())A \Rightarrow (())(A) \Rightarrow (())()$$
$$A \Rightarrow AA \Rightarrow A(A) \Rightarrow A() \Rightarrow (A)() \Rightarrow ((A))() \Rightarrow (())()$$

Ableitungsbaum

Ein **Ableitungsbaum** ist eine graphische Darstellung einer Ableitung.

Beispiel (Ableitungsbaum für das Wort $(())()$ in G_2)



Beispiel

Kontextfreie Grammatik für die nicht-reguläre Sprache $\{0^n 1^n \mid n \in \mathbb{N}\}$:

$$A \to 0A1$$
$$A \to \varepsilon$$

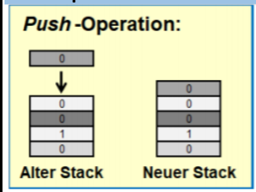
Eine Ableitung des Wortes 000111 in der Grammatik:

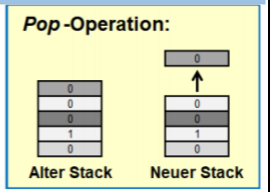
$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000111$$

Kellerautomaten

Automatenmodell für die Erkennung von kontextfreien Sprachen.

Kelleroperationen





Deterministischer Kellerautomat

Ein **deterministischer Kellerautomat (KA)** M ist ein 7-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, \$, F)$, wobei

- Q ist eine endliche Menge von Zuständen.
- Σ ist das Alphabet der Eingabe.
- Γ ist das Alphabet des Kellers.
- $\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \to Q \times \Gamma^*$ ist eine (partielle) Übergangsfunktion.
- $q_0 \in Q$ ist der Startzustand.
- $\$ \in \Gamma$ ist ein ausgezeichnetes Symbol vom Alphabet des Kellers.
- $F \subseteq Q$ ist die Menge der akzeptierenden Zustände.

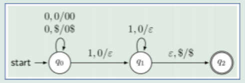
Anfangs enthält der Keller eine Instanz des Symbols \$.

Damit der KA tatsächlich deterministisch ist bedingt: Für die Übergangsfunktion gilt zusätzlich folgende Einschränkung: Für jeden Zustand q und alle Symbole x, b gilt, wenn $\delta(q, b, x)$ definiert ist, dann ist $\delta(q, \varepsilon, x)$ undefiniert.

Beispiel

Ein KA für die kontextfreie Sprache $\{0^n 1^n \mid n > 0\}$:

Formal lässt sich der Automat als $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, \$, F)$ darstellen, mit:



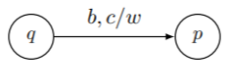
- $Q = \{q_0, q_1, q_2\}$,
- $\Sigma = \{0, 1\}, \Gamma = \{0, \$\}$,
- $F = \{q_2\}$,
- und δ ist wie folgt gegeben:

$$\delta(q_0, 0, 0) = (q_0, 00) \quad \delta(q_0, 0, \$) = (q_0, 0\$) \quad \delta(q_0, 1, 0) = (q_1, \varepsilon)$$
$$\delta(q_1, 1, 0) = (q_1, \varepsilon) \quad \delta(q_1, \varepsilon, \$) = (q_2, \$).$$

Berechnungsschritt

$\delta(q, b, c) = (p, w)$ wird wie folgt interpretiert:

1. Der Automat befindet sich im Zustand q
2. Der Automat liest das Symbol b von der Eingabe falls $b \neq \varepsilon$
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



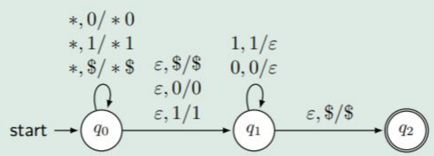
Nichtdeterministischer Kellerautomat

Unterscheidet sich von dem deterministischen KA nur in der Übergangsfunktion.

Die Zustandsbedingung an die Übergangsfunktion fällt beim NKA weg.

Beispiel

Kellerautomat für die Sprache $\{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$:



Das * steht im Diagramm für ein beliebiges Zeichen aus $\sum$. In diesem Beispiel für 0 oder 1 und dient der Vereinfachung.

Sprache eines Kellerautomaten

Die Sprache $L(M)$ des KA M ist definiert durch:

$L(M) = \{w \in \sum^* \mid (q_0, w, \$) \vdash (q, \varepsilon, \gamma)$ für ein $q \in F$ und ein $\gamma \in {}^*\}$

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

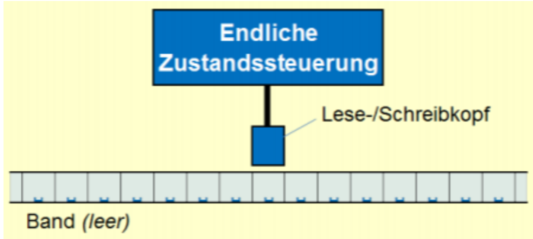
Äquivalenz

Eine Sprache ist kontextfrei, genau dann, wenn es einen NKA gibt, der die Sprache erkennt.

Es gibt auch Sprachen, die nicht erkannt werden können.

Turingmaschinen (TM)

Eine TM ist informell ein endlicher Automat, ergänzt mit einem Lese-/schreib-kopf und einem unendliches Band von Zellen.



Definition (TM)

Eine **(deterministische) Turing-Maschine** (TM) ist ein 7-Tupel

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

mit einer bzw. einem:

- *endlichen* Menge von **Zuständen** $Q = \{q_0, q_1, \ldots, q_n\}$ $(n \in \mathbb{N})$,
- **Eingabealphabet** $\Sigma = \{a_1, a_2, \ldots, a_m\}$ $(m \in \mathbb{N})$,
- **Übergangsfunktion** $\delta : Q \times \Gamma \to Q \times \Gamma \times D$, $D = \{L, R\}$,
- **Startzustand** $q_0 \in Q$,
- Menge von **akzeptierenden Zuständen** $F \subseteq Q$,
- **Bandalphabet** Γ (endliche Menge von Symbolen) und $\Sigma \subset \Gamma$ und
- **Leerzeichen** $\sqcup$, mit $\sqcup \in \Gamma$ und $\sqcup \notin \Sigma$.

Definition (Übergangsfunktion)

Die **Übergangsfunktion** δ ist eine partielle Funktion

$$\delta : Q \times \Gamma \to Q \times \Gamma \times D.$$

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

- $q, p \in Q$ und $X, Y \in \Gamma$
- D beschreibt die Bewegung des Lese-/Schreibkopfes über dem Band. D kann die Werte L für *links* (bzw. *left*) und R für *rechts* (bzw. *right*) annehmen.

Definition (Band)

Das **Band**

- ist in **einzelne Zellen** unterteilt, die jeweils ein **beliebiges Symbol** aus Γ enthalten können, und
- beinhaltet zu Beginn die ***Eingabe***, d. h. ein **endliches Wort** aus Σ^*. Alle anderen Zellen enthalten das besondere Symbol $\sqcup$[2].

Der **Lese-/Schreibkopf** kann jeweils **genau eine Zelle** des Bandes **lesen** und **beschreiben**.

Konfiguration / Berechnung

- **Rechts-Bewegung** einer Turing-Maschine mit $\delta(q, X_i) = (p, Y, R)$
  - Für $i = n$ gilt: $X_1 \ldots X_{n-1} q X_n \vdash X_1 \ldots X_{n-1} Y p$
  - Für $i = 1$ und $Y = \sqcup$ gilt: $q X_1 \ldots X_n \vdash p X_2 \ldots X_n$
- **Links-Bewegung** einer Turing-Maschine mit $\delta(q, X_i) = (p, Y, L)$
  - Für $i = 1$ gilt: $q X_1 \ldots X_n \vdash p \sqcup Y X_2 \ldots X_n$
  - Für $i = n$ und $Y = \sqcup$ gilt: $X_1 \ldots X_{n-1} q X_n \vdash X_1 \ldots X_{n-2} p X_{n-1}$

Gegeben seien $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ und eine Eingabe $w \in \Sigma$.

Eine **Berechnung auf der Eingabe** w ist eine endliche Folge von Berechnungsschritten

$$K_0 \vdash K_1 \vdash \ldots \vdash K_n,$$

so dass

- K_0 die Startkonfiguration ist und
- aus der Konfiguration K_n keine Bewegung mehr möglich ist.

Beispiel

Universelle TM

Beispiel

Gegeben sei die TM M

$$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, \sqcup\}, \delta, q_1, \sqcup, \{q_2\}) \text{ mit:}$$

$$\delta(q_1, 1) = (q_3, 0, R), \qquad \delta(q_3, 0) = (q_1, 1, L),$$
$$\delta(q_3, 1) = (q_2, 0, R), \qquad \delta(q_3, \sqcup) = (q_3, 0, L).$$

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

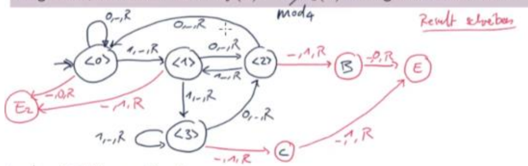
- der Zustand q_1 wird über 0 kodiert
- das Bandsymbol 1 über 00 kodiert
- der Zustand q_3 über 000 kodiert
- das Bandsymbol 0 über 0 kodiert
- und die Bewegung R über 00 kodiert

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

Berechenbarkeit

Aufgabe (mod_3 Funktion)

Zeigen Sie, dass die Funktion $f(n) = \cancel{mod_3}(n)$ Turing-berechnenbar ist.
mod4



Input: 10010 exp: 2 = 10

<0>10010 ⊢ <1>0010 ⊢ <2>010 ⊢ <0>10 ⊢ <1>0 ⊢ <2>_

⊢ 1B_ ⊢ 10E ✓

Loop

Wichtig: 3-5=0 (es gibt keine negativen Zahlen)

LOOP-Programme bestehen aus folgenden syntaktischen Grundelementen:

- Variablen: `x0, x1, x2, ...`
- Konstanten: `0, 1, 2, 3, 4, ...`
- Trennzeichen: `;`
- Zuweisung: `=`
- Operationszeichen: `+` und `-`
- Schlüsselwörter: `Loop, Do, End`

Nach Ablauf eines LOOP-Programms steht der Wert der Berechnung in der Variablen `x0`.

Beispiel (Addition)

Die Addition von natürlichen Zahlen ist LOOP-berechenbar. Das **LOOP-Programm**

```
Loop x1 Do
  x2 = x2 + 1
End;
x0 = x2 + 0
```

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

Beispiel (Multiplikation)

Die Multiplikation von natürlichen Zahlen ist LOOP-berechenbar. Das **LOOP-Programm**

```
Loop x2 Do
  Loop x1 Do
    x0 = x0 + 1
  End
End
```

berechnet die Multiplikation $Mul(x, y) = x \cdot y$.

Beispiel (Modulo Funktion)

Das **LOOP-Programm**:

```
x3 = x1 + 1;
Loop x3 Do
  Loop x3 Do
    x0 = x3 + 0
  End;
  Loop x2 Do
    x3 = x3 - 1
  End
End;
x0 = x0 - 1
```

berechnet die Modulo Funktion $Mod(x, y)$.

Primitiv rekursive Funktionen

Primitiv rekursive Funktionen = Loop berechenbar

Beispiele (Grundfunktionen)

Einige Grundfunktionen:

- $c_5^4 : \mathbb{N}^4 \to \mathbb{N}$ mit $c_5^4(x, y, z, u) = 5$.
- $\pi_1^3 : \mathbb{N}^3 \to \mathbb{N}$ mit $\pi_1^3(x, y, z) = x$.
- $\pi_1^1 : \mathbb{N} \to \mathbb{N}$ mit $\pi_1^1(x) = x$.
- $\pi_5^5 : \mathbb{N}^5 \to \mathbb{N}$ mit $\pi_5^5(x, y, z, u, v) = v$.

While

Definition (WHILE-Programme)

Erweitert man die Sprache LOOP um den zusätzliche syntaktischen Baustein `While xi > 0 Do ... End` für alle Variablen `xi`, dann erhält man die Menge aller **WHILE-Programme**[4].

Bemerkungen

- Jedes LOOP-Programm ist auch ein WHILE-Programm.
- WHILE-Programme terminieren nicht immer (können unter Umständen unendlich lang laufen).
- Die Semantik von WHILE-Programmen ist analog zur Semantik von LOOP-Programmen gegeben.

GOTO

GOTO-Programme bestehen aus folgenden syntaktischen Grundelementen:

- Variablen: `x0, x1, x2, ...`
- Konstanten: `0, 1, 2, 3, 4, ...`
- Marker: `M1, M2, ...`
- Zuweisung: `=`,
- Trennzeichen: `;`, `:`
- Operationszeichen: `+` und `-`
- Schlüsselwörter: `Goto, If, Then, Halt`

Nach Ablauf eines GOTO-Programms steht der Wert der Berechnung in der Variablen `x0`.

Turing-Vollständig

Satz (Turing-Vollständigkeit)

Für jede natürliche Zahl $k \in \mathbb{N}$ und jede (partielle) Funktion $f : \mathbb{N}^k \to \mathbb{N}$, sind folgende Aussagen äquivalent:

- *Es gibt eine Turing-Maschine T, die f berechnet. D.h., f ist Turing-berechenbar.*
- *Es gibt ein WHILE-Programm P, das f berechnet. D.h., f ist WHILE-berechenbar.*
- *Es gibt ein GOTO-Programm Q, das f berechnet. D.h., f ist GOTO-berechenbar.*

Unvollständig

Unvollständigkeit von LOOP-Programmen und primitiv rekursiver Funktionen:

- Wir haben gesehen, dass die Berechnung jeder Turingmaschine mit einem geeigneten WHILE-Programm simuliert werden kann (und umgekehrt).
- Im folgenden wollen wir uns davon überzeugen, dass dies für LOOP-Programme und primitiv rekursive Funktionen nicht der Fall ist. Wir geben zwei verschiedene Funktionen an, die beide berechenbar sind (im Sinn von WHILE/Turingmaschinen) aber nicht primitiv rekursiv respektive LOOP berechenbar.

Satz

- *Die Ackermannfunktion ist (Turing-) berechenbar.*
- *Die Ackermannfunktion ist nicht LOOP berechenbar/nicht primitiv rekursiv.*
- *Die Ackermannfunktion ist total (überall definiert).*

Bemerkung

- Wie wir anhand der Ackermannfunktion und des LOOP-Interpreters gesehen haben, gibt es totale Turing-berechenbare Funktionen, die nicht primitiv rekursiv und somit auch nicht LOOP-berechenbar sind.
- Somit gilt: Die Berechnungsmodelle der LOOP-Programme und der primitiv rekursiven Funktionen sind **nicht** Turing-vollständig.

Entscheidbarkeit

Definition (Entscheidbarkeit)

Eine Sprache $A \subset \Sigma^*$ heisst **entscheidbar**, wenn eine Turingmaschine T existiert, die das Entscheidungsproblem (Σ, A) löst.

(semi-)Entscheidbar

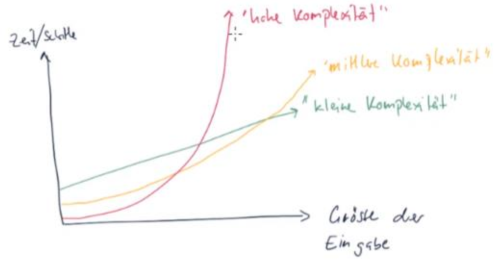
Definition (Semi-Entscheidbarkeit)

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

- Wenn T mit Bandinhalt $x \in A$ gestartet wird, dann hält T nach endlich vielen Schritten mit Bandinhalt "1"(Ja) an.
- Wenn T mit Bandinhalt $x \in \Sigma^* \setminus A$ gestartet wird, dann hält T nie an.

Komplexität

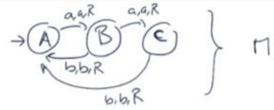
Wieviel Zeit braucht eine TM



Time

Wieviel berechnungsschritte = Time

BSP:



$Time_M(aab) = 3$

A aab ⊢ a B ab ⊢ aa C b ⊢ aab A

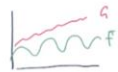
$Time_M(3) = \max\{\underbrace{Time_M(aaa)}_{2}, Time_M(aab), \ldots \underbrace{Time_M(bbb)}_{0}\}$

$= 3$

O-Notation

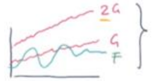
Die Funktion F "wächst nicht schneller" als G

Stufe 0



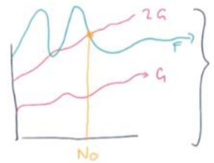
$\forall n\, (F(n) \leq G(n))$

Stufe 1 (vernachlässigen von konstanten Faktoren)
("Funktion F wächst nicht wesentlich schneller")



$\exists c\, \forall n\, (F(n) \leq c \cdot G(n))$

Stufe 2 (Vernachlässigen von "kleinen" Werten)



$\exists N_0, c\, \forall n > N_0\, (F(n) \leq c\, G(n))$

Seien $f, g : \mathbb{N} \to \mathbb{N}$ zwei Funktionen. Dann gilt:

a) $f \in \mathcal{O}(g)$: Es existieren 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)

Beispiel

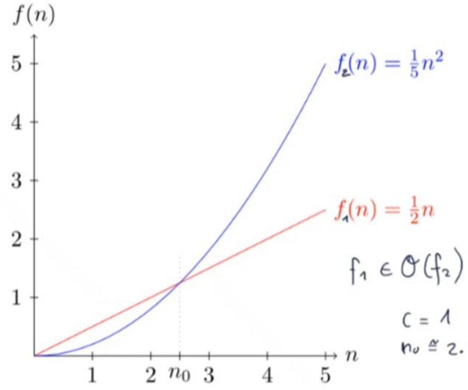
$2n^2 \in \mathcal{O}(n^2)$ Ja $(c = 2, n_0 = 0)$

$2n^2 + n \in \mathcal{O}(n^2)$ Ja $(c = 3, n_0 = 0)$

$15n^5 + 3n^3 + 4n^2 + 1005 \in \mathcal{O}(n^5)$ Ja
$c = 23$

$\sum_{i=0}^{k} a_i n^i \in \mathcal{O}(n^k)$ $c := \boxed{\sum_{i=0}^{k} a_i}$

$n^2 \in \mathcal{O}(500n)$? Nein
$c \cdot 500n = (500c)n < n^2$ für $n > 500c$



$f_1 \in \mathcal{O}(f_2)$

$c = 1$

$n_0 \hat{=} 2.5$

Polynomzeit

Definition (In Polynomzeit lösbar)

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

Definition (Die Klasse P)

Die **Klasse aller in Polynomzeit entscheidbaren Sprachen** wird **P** genannt.

in Polynomzeit lösbar = "schnell lösbar"

Nicht Polynomzeit

Definition (Die Klasse NP)

Die **Klasse aller von einer NTM in Polynomzeit entscheidbaren Sprachen** nennen wir **NP**.

Achtung

NP heisst NICHT „nicht polynomiell" sondern „nichtdeterministisch polynomiell".

b'barkeit	Komplexität
einfach ≙ entscheidbar	einfach ≙ P
schwieriger ≙ semi-entsch'bar	halb einfach ≙ NP
schwer ≙ nicht semi-entsch'bar/Rest	schwer ≙ Rest

Polynomzeit Verifizierer

P ≙ Lösung *finden* in Polynomzeit

NP ≙ Lösung *verifizieren* in Polynomzeit

Clique

Gegeben: Graph, G
Zahl, n

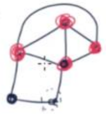
z.B.

, 3

Frage: Enthält G eine Clique der Grösse n?

Ja

 ist Clique der Grösse 3

(, 4) => Ja

(, 4) => Nein

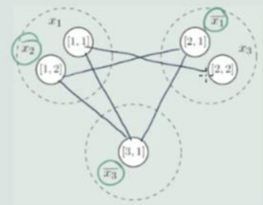
Algorithmus: (brute-force)
durch alle Teilgraphen der Grösse n gehen

Beispiel für die Konstruktion: $\varphi = (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_3})$

0 1 1 0 1

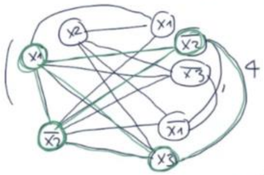
$\underbrace{}_{1}$ $\underbrace{}_{1}$

$\underbrace{}_{1}$



Aufgabe (CLIQUE ist NP-vollständig)

Ist $\varphi'' = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_2})$ erfüllbar?

(, 4) => erfüllbar

 ✓