Grundlagen Theoretischer Informatik

Zusammenfassung wichtiger Elemente der GTI

von Alexander Köster
Student der Universität Siegen, GTI 2017
Letzte Aktualisierung: 6. September 2017

Dieser Kurs der Theoretischen Informatik beschäftigt sich
mit den Eigenschaften von Sprachen und deren Automatenmodellen,
sowie Berechenbarkeit und Berechnungsmodelle, u.a. die bekannte Turing-Maschine.

Einzig für Lernzwecke erstellt.

Nicht geeignet als Klausurhilfe.
Das Erstellen einer eigenen Klausurhilfe führt zu einem besonders guten Lerneffekt.
Dieses Dokument sollte nur als Orientierung oder Vergleich dienen.
Jeder sollte seine Klausurhilfe individuell auf seinen Lernstand und seine eigenen Probleme anpassen,
gut bekannte Dinge auslassen und schlecht merkbare Dinge hinzufügen.
Dieses Dokument beinhaltet viel mehr, als für eine tatsächliche Klausur durchschnittlich benötigt wird.
Für einen guten Ansatz, welche Inhalte man braucht, sollte sich an der Probeklausur orientiert werden.

1 Wörter & Sprachen

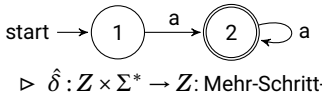
- **Alphabet**: nicht-leere Menge
- **Wort** über dem Alphabet Σ ist eine endliche Zeichenkette $w = a_1 a_2 \dots a_n$ mit $a_i \in \Sigma$.
  Die Länge des Wortes ist $n = |w|$.
  Konvention: Wörter Kleinbuchstaben $u-z$
- ε: **Leeres Wort** mit $|\varepsilon| = 0$
- Σ^*: **Menge aller Wörter** über Alphabet Σ,
  Σ^+: **Menge aller nicht-leeren Wörter**
- **Konkatenation** von Wörtern $u = a_1 \dots a_n$ und $v = b_1 \dots b_n$: $u \circ v = a_1 \dots a_n b_1 \dots b_n = uv$
  - $a^n = aa \dots a$ (n-mal)
  - $\prod_{i=0}^{n} w_i = w_0 w_1 \dots w_n$
- $(\Sigma, \circ)$: Das von Σ erzeugte **freie Monoid**
  - $(u \circ v) \circ w = u \circ (v \circ w)$ (assoz.)
  - $\varepsilon \circ u = u = u \circ \varepsilon$ (neutr.)
- **Formale Sprache** L über Σ: Menge $L \subseteq \Sigma^*$
  - es gibt $L = \emptyset$ und $|L| = \infty$
- **Grammatik**: $G = (V, \Sigma, P, S)$
  - V: Alphabet der Nicht-Terminale (Variablen) **Konvention:** Großbuchstaben
  - Σ: Alphabet der Terminale, $V \cap \Sigma = \emptyset$
    Konvention: Kleinbuchstaben $a-z$
  - $P \subseteq \left((V \cup \Sigma)^+ \setminus \Sigma^*\right) \times (V \cup \Sigma)^*$:
    Produktionen: Tupel (l, r) oft $l \to r$
  - $S \in V$: Startvariable („Axiom“)
  - $(V \cup \Sigma)^*$ nennt man auch Satzform
- **Ableitung**: $u \Rightarrow_G v$
  wenn u in einem Schritt nach v übergeht
  - binäre Relation auf $(V \cup \Sigma)^*$
    $\Rightarrow_G = \{(u, v) | \exists (l \to r) \in P : \exists x, y \in (V \cup \Sigma)^* : u = xly, v = xry\}$
    nichtdeterministisch (keine Funktion)
  - $u \Rightarrow_G^* v$: wenn u in mehreren Schritten nach v übergeht (reflexiv-transitive Hülle)
  - Folge $w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_n$ mit $w_0 = S$ heißt Ableitung von w_n aus S
- Von einer Grammatik G **erzeugte Sprache**:
  $L(G) = \left\{w \in \Sigma^* \middle| S \Rightarrow_G^* w\right\}$

2 Chomsky-Hierarchie

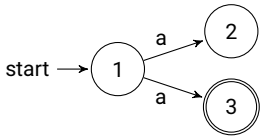
Menge aller Sprachen vgl. $\mathbb{R} \supset \mathbb{N}$
$\supset$ Typ-0-Sprachen (semi-entscheidbare Sprachen)
$\supset$ Typ-1-Sprachen (kontextsensitive Sprachen)
$\supset$ Typ-2-Sprachen (kontextfreie Sprachen)
$\supset$ Typ-3-Sprachen (reguläre Sprachen)

2.1 Typ-3-Sprachen (regulär)

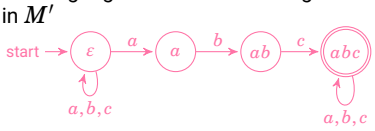
- Alle Produktionen der Form $A \to a \mid A \to Ba$
- ε-**Sonderregelung**: Produktion $S \to \varepsilon$ ist erlaubt für Startsymbol S, als Ausnahme, wenn S in keiner rechten Seite vorkommt
- **Deterministischer endlicher Automat (DFA)** $M = (Z, \Sigma, \delta, z_0, E)$
  - Z: endliche Zustandsmenge
  - Σ: Eingabealphabet ($Z \cap \Sigma = \emptyset$)
  - $z_0 \in Z$: Startzustand
  - $E \subseteq Z$: Menge der Endzustände
  - $\delta : Z \times \Sigma \to Z$: Überführungsfunktion



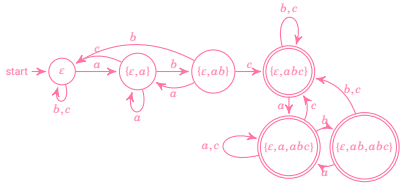
  - $\hat{\delta} : Z \times \Sigma^* \to Z$: Mehr-Schritt-Übergänge
  - $T(M) = \left\{x \in \Sigma^* \middle| \hat{\delta}(z_0, x) \in E\right\}$:
    vom DFA M **akzeptierte Sprache**
  - $T(M)$ von einem DFA ist immer regulär.
- **Nichtdeterministischer endl. Automat (NFA)** $M = (Z, \Sigma, \delta, S, E)$
  - Z: endliche Zustandsmenge
  - Σ: Eingabealphabet ($Z \cap \Sigma = \emptyset$)
  - $S \subseteq Z$: Menge der Startzustände
  - $E \subseteq Z$: Menge der Endzustände
  - $\delta : Z \times \Sigma \to 2^Z$: Überführungsfunktion



  - $\hat{\delta} : 2^Z \times \Sigma^* \to 2^Z$: **M.-S.-Übergänge**
    $\hat{\delta}(A, w)$ = alle Zustände, die man von Zuständen aus der Menge A durch Einlesen von w erreichen kann
  - $T(M) = \left\{x \in \Sigma^* \middle| \hat{\delta}(S, x) \cap E \neq \emptyset\right\}$:
    vom NFA M **akzeptierte Sprache** (Wort wird akzeptiert, wenn es *min. einen* Pfad zu einem Endzustand in M gibt)
  - Es gibt auch ε-Kanten in NFAs (Sofortü.)
- NFAs und DFAs sind **gleich mächtig**.
  → **Potenzmengenkonstruktion**
  NFA $M \to$ DFA $M' = \left(2^Z, \Sigma, \gamma, S, F\right)$
  γ: neue Übergangsfunktion, vereinige alle Ziele eines Übergangs in δ in einen einzigen Zustand in M'

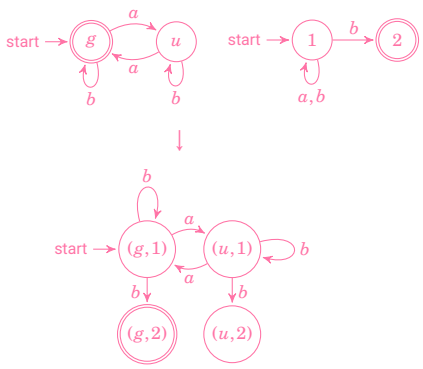


Bsp.:

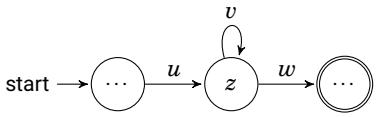


- **Reguläre Ausdrücke**
  - $\mathrm{Reg}(\Sigma)$: Menge aller regulären Ausdr.
  - $L(\emptyset) = \emptyset, L(\varepsilon) = \{\varepsilon\}$
  - $\forall a \in \Sigma : L(a) = \{a\}$
  - $L(\alpha\beta) = L(\alpha)L(\beta)$, wobei
    $L_1 L_2 = \{w_1 w_2 | w_1 \in L_1, w_2 \in L_2\}$
  - $L((\alpha|\beta)) = L(\alpha) \cup L(\beta)$
  - $L((\alpha)^*) = (L(\alpha))^*$,
    wobei $L^* = \{w_1 \dots w_n | n \in \mathbb{N}_0, w_i \in L\}$
    - *„Kleenesche Hülle“* L^* (s.u.)
  - Reguläre Ausdrücke sind **gleichmächtig wie NFAs/DFAs** (reguläre Sprachen).
- **Konkatenation** regulärer Sprachen:
  Alle Zustände, die in DFA/NFA 1 in Endzustände übergehen, erhalten zusätzliche Übergänge zu allen Startzuständen in DFA/NFA 2 → neuer NFA für $1 \circ 2$
  - abgeschlossen ($L_1 L_2$ immer regulär für L_1, L_2 regulär)
- **Vereinigung** regulärer Sprachen:
  Zwei DFAs/NFAs können einfach parallel als neuer NFA mit mehreren Startzuständen geschrieben werden.
  - abgeschlossen ($L_1 \cup L_2$ immer regulär für L_1, L_2 regulär)
- **Kleenesche Hülle** L^* regulärer Sprachen:
  - L^* enthält immer ε (s. Def.)
  - nur so kann man unendl. Spr. erzeugen
  - Alle Zustände, die im DFA/NFA in einen Endzustand übergehen, erhalten zusätzliche Übergänge zu allen Startzuständen, evtl. einen getrennten Start- + Endzustand für ε-Erkennung hinzufügen, falls L das bisher nicht akzeptierte
  - abgeschlossen (L^* regulär $\forall L$ regulär)
- **Komplement** $\overline{L} = \Sigma^* \setminus L$ regulärer Sprachen:
  - Alle End- und Nicht-Endzustände in einem *DFA* vertauschen
  - abgeschlossen ($\overline{L}$ regulär $\forall L$ regulär)
- **Schnitt** regulärer Sprachen:
  - abgeschlossen ($\overline{L}$ regulär $\forall L$ regulär)
  - wegen $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ (DeMorgan)
  - **Kreuzproduktautomat**
    - 2 DFAs/NFAs „parallel“ ausführen
    - neuer NFA mit Zuständen der Form (a, b) mit $a \in M_1, b \in M_2$
    - Start-/Endzustand, wenn a und b in ihren Automaten jeweils Start-/Endzustand sind

Bsp.:



- **Pumping-Lemma** (auch „uvw-Theorem“)
  Beweis, dass eine Sprache *nicht* regulär ist



  Die Schleife v kann gar nicht, einmal oder mehrmals durchlaufen werden.
  Es muss gelten:
  - $|v| \geq 1$: Schleife v ist nicht trivial
  - $|uv| \leq n$ (n ist Anzahl Zustände eines NFA): d.h. spätestens nach n Eingaben wird der Zustand z ein 2. Mal erreicht
  - $\forall i \in \mathbb{N}_0 : uv^i w \in L$

  „Kochrezept“:
  - Nehme eine fest aber beliebige Zahl n.
  - Wähle ein geeignetes Wort $x \in L$ mit $|x| \geq n$. Hilfreich: n sollte in der Definition auftauchen (z.B. im Exponenten).
  - Betrachte alle möglichen Zerlegungen $x = uvw$ mit o.g. Einschränkungen.
  - Wähle für jede Zerlegung i, sodass $uv^i w \notin L$. Meist $i = 0$ oder $i = 2$.
- **Erkennungsäquivalenz**: Wenn für einen DFA-Zustand gilt: Startet man von diesem, führen alle Eingaben zum gleichen Ergebniszustand.
  Formal: $z_1, z_2 \in Z$ sind erkennungsäquivalent, gdw. $\forall w \in \Sigma^* : \hat{\delta}(z_1, w) \in E \Leftrightarrow \hat{\delta}(z_2, w) \in E$
  - Äquivalenzrelation (refl., symm., trans.)
- **Myhill-Nerode-Äquivalenz**:
  Erweiterung der Erkennungsäquivalenz auf Sprachen L allgemein
  $x R_L y \Leftrightarrow \forall w \in \Sigma^* (xw \in L \Leftrightarrow yw \in L)$
  d.h.: $\hat{\delta}(z_0, x)$ erkennungsäquiv. zu $\hat{\delta}(z_0, y)$
  - **Äquivalenzklassen**: Teilmengen von Σ^*, deren Elemente untereinander alle Myhill-Nerode-äquivalent sind
  - Anzahl der Äquivalenzkl.: $\mathrm{index}(R_L)$
  - Der **Minimalautomat** (kleinster DFA) hat genau $\mathrm{index}(R_L)$ viele Zustände.
    Finden: **Äquivalenzklassenautomat** M_L:
    Äquivalenzklassen = Zustände
    „Kochrezept“:
    - Starte Tabelle aller Zustandspaare (treppenförmig, horiz. letzten Zst. weglassen, vert. 1. Zst. wegl.)
    - Markiere $\{z, z'\}$ mit $z \in E \wedge z' \notin E$.
    - $\forall$ unmarkierten Paare: teste $\forall a \in \Sigma$, ob $\left\{\delta(z, a), \delta\left(z', a\right)\right\}$ bereits markiert. Ja ⇒ markiere $\{z, z'\}$.

- Wdh., bis keine Änderung der Tabelle mehr.
- $\forall$ jetzt noch unmarkierten Paare gilt: z erkennungsäq. z'
- „Markieren“ sollte mit Reihenfolgenkommentar erfolgen (Zahl in Tabelle statt nur Häkchen o.ä.)

▷ L regulär $\Leftrightarrow \text{index}(R_L) < \infty$ (M.-N.)

▷ Wenn man unendlich viele Äquivalenzklassen (Wörter in versch. Äq.-kl.) finden kann, ist die Sprache nicht regulär.

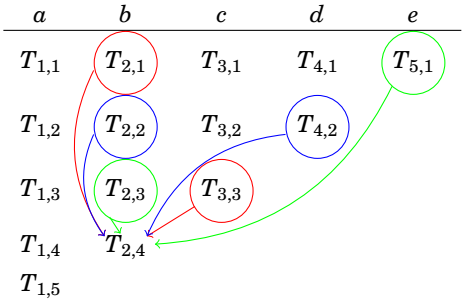
▷ „Pfadäquivalenz“ in DFA M:
$xR_M y \Leftrightarrow \hat{\delta}(z_0,x) = \hat{\delta}(z_0,y)$
M ist minimal $\Leftrightarrow R_M = R_L$

- Es gibt keinen *eindeutigen* minimalen NFA.
- **Leerheitsproblem** ($T(M) = \emptyset$) ist entscheidbar (kann man an Automatenzeichnung sehen)
- **Endlichkeitsproblem** ($|T(M)| < \infty$) ist entscheidbar (siehe Pumping-L./Zyklus im Automaten)
- **Inklusionsproblem** ist entscheidbar:
$T(M_1) \subseteq T(M_2) \Leftrightarrow \left(\overline{T(M_2)} \cap T(M_1)\right) = \emptyset$
- **Äquivalenzproblem**: $T(M_1) = T(M_2)$
$\Leftrightarrow T(M_1) \subseteq T(M_2) \wedge T(M_2) \subseteq T(M_1)$
oder: minimaler DFA der beiden ist isomorph
- Zum Beweisen der Abgeschlossenheit dieser Probleme: Primzahlen helfen oft!

2.2 Typ-2-Sprachen (kontextfrei)

- Alle Produktionen der Form $A \to$ *irgendwas*, keine verkürzende Regeln (*irgendwas* ≥ 1) außer ε-Produktionen (mit und ohne ε-Sonderregelung, s. 2.1)
- **Chomsky-Normalform (CNF)** für kontextfreie Grammatik G mit $\varepsilon \notin L(G)$: nur Produktionen der Form $A \to BC$ und $A \to a$ erlaubt
  - ▷ Zu jeder solchen Grammatik gibt es CNF.
  - ▷ Umwandlung in CNF – „Kochrezept“:
    - Für jedes Terminalsymbol $a \in \Sigma$ eine neue Variable A_a mit einziger Produktion $A_a \to a$ einführen.
    - Jedes Vorkommen von jedem a auf einer rechten Seite mit dem jeweiligen neuen A_a ersetzen.
    - Kettenregeln eliminieren: $A \to B$, falls $B \to$ *irgendwas*, ersetzen mit $A \to$ *irgendwas*
    - Produktionen d.F. $A \to A_1 \ldots A_n$ ersetzen mit $A \to A_1 B_1$, $B_1 \to A_2 B_2, \ldots, B_{n-1} \to A_{n-1} A_n$
  - ▷ Für ε ein Zwischen-Startsymbol $S' \to \varepsilon$ und alle Produktionen vom „originalen“ S hinzufügen (ε-Sonderregel, s. 2.1)
- **Greibach-Normalform**: Alle Produktionen der Form $A \to aB_1B_2 \ldots B_k$ mit $k \geq 0$
- **Pumping-Lemma** (auch „$uvwxy$-Theorem“):
Alle Wörter z mit $|z| \geq n$ mit $n = 2^k$ (k ist die Anzahl Variablen einer CNF) lassen sich für kontextfreie Spr. zerlegen als $z = uvwx$ mit:
  - ▷ $|vx| \geq 1$
  - ▷ $|vwx| \leq n$
  - ▷ $\forall i \geq 0 : uv^i wx^i y \in L$

  Beweis von nicht-kontextfrei – „Kochrezept“:
  - ▷ Wähle eine fest aber beliebige Zahl n.
  - ▷ Wähle ein geeignetes Wort $z \in L$ mit $|z| \geq n$. Hilfreich: n sollte in der Definition auftauchen (z.B. im Exponenten)
  - ▷ Alle Zerlegungen $z = uvwxy$ betrachten mit den o.g. Einschränkungen.
  - ▷ „Pumpen“: Finde $uv^i wx^i y \notin L$.
Meist hilft $i = 0$ oder $i = 2$.
- **Unäre Sprachen:** Sprachen über einem einelementigen Alphabet
  - ▷ Jede unäre kontextfreie Sprache ist automatisch auch regulär.
- **Vereinigung, Konkatenation, Stern-Operator**
  - ▷ abgeschlossen ($L_1 \cup L_2, L_1L_2, L_1^*$ immer kontextfrei für L_1, L_2 kontextfrei)
- **Schnitt** und **Komplement** sind *nicht* abgeschlossen unter kontextfreien Sprachen!
  - ▷ Aber: kontextfrei $\cap$ regulär $\in$ kontextfrei
- **Wortproblem – CYK-Algorithmus**:
  - ▷ Wir gehen die folgende Tabelle von oben nach unten durch und suchen immer die Variablen, die in ihrer Produktionsregel die Variablenkombination der vorigen Zeile erzeugen können. **Bsp.:** Haben wir die Felder A und B in der vorigen Zeile, so suchen wir für das neue Feld eine Variable, die die Produktion $X \to AB$ hat.



- ▷ erste Zeile: Alle Variablen, die das darüber stehende Terminal produzieren
- ▷ ergibt ein Feld $\emptyset$, kann man dessen Kombinationen überspringen
- ▷ sind in einem Feld mehrere Variablen, prüft man distributiv
- ▷ wenn im untersten Feld (hier $T_{1,5}$) ein Startsymbol ist, ist das Wort $\in L$
- ▷ Komplexität: $\mathcal{O}(n^3)$

- **Kellerautomaten (PDA)** (nichtdeterministisch) $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$
  - ▷ Z: endliche Zustandsmenge
  - ▷ Σ: Eingabealphabet ($Z \cap \Sigma = \emptyset$)
  - ▷ Γ: Kelleralphabet
  - ▷ z_0: Startzustand
  - ▷ $\delta : Z \times (\Sigma \cup \{\varepsilon\}) \to \wp_e(Z \times \Gamma^*)$: Überf.f. ($\wp_e(M)$: Menge aller endl. Teilmengen)
  - ▷ $\# \in \Gamma$: Kellerbodenzeichen

  akzeptieren genau die kontextfreien Sprachen
  - ▷ nichtdet. PDA > det. PDA !
  - ▷ Zu Beginn enthält PDA nur #
  - ▷ Kellerspeicher kann unendlich wachsen
  - ▷ Unser PDA akzeptiert mit leerem Keller
- **PDA-Konfiguration**: $(z, w, \gamma) \in Z \times \Sigma^* \times \Gamma^*$
  - ▷ $z \in Z$: aktueller Zustand
  - ▷ $w \in \Sigma^*$: noch zu lesende Eingabe
  - ▷ $\gamma \in \Gamma^*$: aktueller Kellerinhalt („oben“ = links)
  - ▷ Übergang: $(z, aw, A\gamma) \vdash (z', w, \gamma'\gamma)$ (A oben im K.) und $(z', \gamma') \in \delta(z, a, A)$
- **Akzeptierte Sprache eines PDA** M: $N(M) = \{x \in \Sigma^* | (z_0, x, \#) \vdash^* (z, \varepsilon, \varepsilon)\}$ für ein $z \in Z$
  - ▷ Alle Wörter, mit denen man den Keller vollständig leeren kann
  - ▷ nichtdet.: es kann auch zus. Berechnungen geben, die den Keller nicht leeren
- **Deterministisch kontextfreie Sprache**
  - ▷ Akzeptiert durch det. Kellerautomaten: $M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$
    - $D(M)$ = akzeptierte Sprache
    - det. PDA akzeptiert mit Endzustand $z_e \in E$ und beliebigem Keller $\gamma \in \Gamma^*$
  - ▷ det. kontextfrei $\subset$ kontextfrei
  - ▷ Wortproblem in $\mathcal{O}(n)$ lösbar
  - ▷ abgeschlossen unter Komplement
  - ▷ nicht abgeschlossen unter Schnitt und Vereinigung
  - ▷ det. kontextfreie Grammatiken sind komplex, z.B. $\text{LR}(k)$-Grammatiken
  - ▷ Äquivalenzproblem entscheidbar
  - ▷ Inklusionsproblem unentscheidbar
- **Leerheitsproblem** $L = \emptyset$:
  - ▷ Best.: $W = \{A \in V | \exists w \in \Sigma^* : A \Rightarrow_G^* w\}$
Menge der produktiven Variablen (aus denen man ein Terminalsymbol erzeugen kann) Dazu zuerst: alle Variablen, aus denen sofort Terminale entstehen, dann alle Variablen, die diese V. erzeugen, usw.
  - ▷ $L \neq \emptyset \Leftrightarrow S \in W$
- **Endlichkeitsproblem**
  - ▷ Graph aufstellen: Wenn $(A \to BC) \in P$ und $A, B \in W$ (s.o.), im Graph:

  - ▷ $|L| = \infty \Leftrightarrow$ man kommt von S zu einer Variable A, von der man in einem echten Zyklus zurück nach A kommen kann
- **Äquivalenzproblem** und **Schnittproblem** sind unentscheidbar

2.3 Typ-1-Sprachen (kontextsensitiv)

- Keine verkürzenden Produktionen
($l \to r$ mit $|l| \leq |r|$)
- ε-**Sonderregelung**: $S \to \varepsilon$ erlaubt (s. 2.1)
- **Linear beschränkte Automaten (LBA)**: Turingmaschine, die $\square$ nicht mit anderen Symbolen überschreiben kann (nur so viel Band benutzen, wie das Eingabewort lang war) (s. 2.4)
  - ▷ $A = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$, wie TM
  - ▷ $\forall a \in \Gamma \setminus \{\square\} : \square \not\to a \wedge a \not\to \square$
  - ▷ Akzeptierte Sprache: $T(M)$
$= \{w \in \Sigma^* | \exists k \in \Gamma^* E \Gamma^+ : z_0 w \square \vdash_A^* k\}$
  - ▷ rechtes $\square$ zum Erkennen des Bandendes
  - ▷ nichtdet. LBA: genau Typ-1-Sprachen; ob det. gleich mächtig sind ist *nicht bekannt*
  - ▷ Beweis durch „Simulation“ der Grammatik rückwärts (wie TM)
- Man kann von jeder Typ-1-Grammatik, deren Worte alle mit dem gleichen Symbol enden oder beginnen, dieses Symbol wegschneiden und hat wieder eine Typ-1-Grammatik
- **Komplement**: abgeschlossen
- **Wortproblem** ist entscheidbar.

2.4 Typ-0-Sprachen (semi-entsch.)

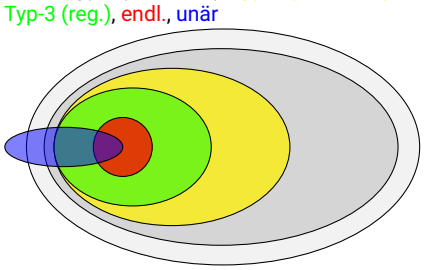
- Keine Einschränkungen der Produktionen. Jede Grammatik ist vom Typ 0.
- **Turingmaschine (TM)**:
  - ▷ Maschine mit unendlichem Band, endlich vielen Zuständen und einem Schreib-Lese-Kopf (steht zu Anfang ganz links)
  - ▷ Det.: $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$
    - Z: endliche Zustandmenge
    - Σ: Eingabealphabet
    - $\Gamma \supset \Sigma$: Bandalphabet
    - $z_0 \in Z$: Startzustand
    - $E \subseteq Z$: Endzustände
    - $\delta : (Z \setminus E) \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$
Überführungsfunktion (det.):
aktueller Zustand, gelesenes Bandsymbol $\to$ neuer Zustand, neues Bandsymbol, Kopfbewegung nach rechts/links/keine

- $\delta : (Z \setminus E) \times \Gamma \to 2^{Z \times \Gamma \times \{L,R,N\}}$ Überführungsfunktion (nichtdet.)
- $\square$: Blanksymbol („leer“)
- det. äquiv. nichtdet.

▷ **Konfiguration** ist ein Wort $k \in \Gamma^* Z \Gamma^+$
- Bedeutung: in zwei Hälften geteiltes Wort auf dem Band, $z \in Z$ stellt Zustand und die Stelle des Schreib-Lese-Kopfes dar (steht auf dem Symbol rechts von z)
- Übergangsrelation: $\vdash_M$
- Rechts von z muss immer mindestens ein Symbol (sei es $\square$) stehen

▷ **Akzeptierte Sprache** $T(M)$
$= \{x \in \Sigma^* \mid \exists k \in \Gamma^* E \Gamma^+ : z_0 x \vdash_M^* k\}$
$\cup \{\varepsilon \mid \exists k \in \Gamma^* E \Gamma^+ : z_0 \square \vdash_M^* k\}$
Alle Wörter, mit denen die TM in einen Endzustand gelangen kann

- **Komplement**: nicht abgeschlossen
- **Wortproblem** ist nicht entscheidbar (nur semi-entscheidbar).
- **Zusammenfassung**:
Typ-0, Typ-1 (kontexts.), Typ-2 (kontextfr.), Typ-3 (reg.), endl., unär



3 Berechnungen

3.1 Berechenbarkeit

- **Partielle Funktionen**: nicht überall definiert
Bsp.: Subtraktion und Division auf $\mathbb{N}$
- Binärdarstellung einer Zahl $n \in \mathbb{N}$:
$\text{bin}(n) \in 1\{0,1\}^* \cup \{0\}$
- **Intuitive Berechenbarkeit**:
Es gibt ein Verfahren, das in endlichen Schritten das Ergebnis einer Funktion ausgibt (oder nie terminiert, wenn nicht definiert)
  - ▷ Kann nicht formal mathematisch definiert werden (Church'sche These)
- **Turing-Berechenbarkeit**: entspricht genau den intuitiv berechenbaren Funktionen
  - ▷ Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist Turing-berechenbar, wenn es eine det. TM gibt, die nach Eingabe von $\text{bin}(n_1)\#\text{bin}(n_2)\#\ldots$ in endl. Schritten in einen Endzustand übergeht, wenn nur das Ergebnis $f(n_1,n_2,\ldots)$ auf dem Band ist und der Kopf ganz links steht.
  - ▷ Analog dazu für Funktionen auf Wörtern
- Es gibt **abzählbar viele Programme** (berechenbare Funktionen), wenn man Programme als Code (= Wörter $\in \Sigma^*$) darstellen kann, aber **überabzählbar viele Funktionen** (Diagonalisierungsbeweis: es müsste sonst eine Funktion $F : \mathbb{N} \to$ *Menge aller Funktionen* geben)
- Es gibt **nicht berechenbare Funktionen**.
- Die **überall undefinierte Funktion** Ω ist Turing-berechenbar (TM ohne Endzustand)
- **Mehrband-Turingmaschine**:
  - ▷ $\delta : (Z \setminus E) \times \Gamma^k \to Z \times \Gamma^k \times \{L,R,N\}^k$
  - ▷ Ein- und Ausgabe auf dem ersten Band. Schreib-Lese-Köpfe unabhängig.
  - ▷ Gleich mächtig wie normale TM. (Beweis: M-TM mit TM auf Tupel-Alphabet simulieren)
  - ▷ Meist kein sinnvolles Modell.

- **Loop-Programme**
  - ▷ Variablen $x_1, x_2, \ldots$, Konstanten $0, 1, \ldots$
  - ▷ Zuweisungen: $x_i := x_j + c$, $x_i := x_j - c$ mit $i, j \in \mathbb{N}$ ($i = j$ erlaubt)
  - ▷ *Achtung:* $1 - 2 = 0$ (wir sind in $\mathbb{N}$)
  - ▷ Sequentielle Ausführung: $P_1; P_2$
  - ▷ Schleife: Loop x_i Do P End
So oft P ausführen, wie x_i *zu Beginn* angibt (Anzahl ändert sich während der Schleife nicht)
  - ▷ Eingabewerte stehen in den ersten n Variablen, **Konvention:** Ausgabe in x_1
  - ▷ Loop-Pr. **< Turing-Berechenbarkeit**! Loop entspricht LBA (s. 2.3)

Formal:
  - ▷ $[P]_k(n_1,\ldots,n_k) = (m_1,\ldots,m_k)$
Funktion, die von P bei Eingabe von k Werten berechnet wird
  - ▷ Projektionsfunk.: $\pi_i(n_1,\ldots,n_k) = n_i$
$\Rightarrow \pi_1([P]_k(\ldots)) =$ die 1. Variable
  - ▷ „Loop-berechenbar“, wenn es ein Loop-Programm gibt, das eine endliche Zahl Variablen zum Berechnen benutzt

Beweisbar erlaubte Konstrukte:
  - ▷ If $x_i = 0$ Then P End
  - ▷ $x_i := x_j + x_k$ für $i \neq k$
  - ▷ $x_i := x_j \cdot x_k$ für $k \neq i \neq j$

- **While-Programme**
  - ▷ Wie Loop-Programme (gleiches erlaubt)
  - ▷ Zusätzlich: While $x_i \neq 0$ Do P End
So lange P ausführen, bis $x_i = 0$.
  - ▷ Entspricht Turing-Berechenbarkeit (beweisbar mit Mehrband-TM)

- **Goto-Programme**
  - ▷ Besteht aus $M_1 : A_1;\quad M_2 : A_2;\quad \ldots$
(M_i ist ein Sprungmarker, A_i ein Goto-Programm) Sprungmarker darf man weglassen, wenn nicht benötigt.
  - ▷ Wertzuweisung wie Loop/While
  - ▷ Unbedingter Sprung: Goto M_i
  - ▷ Bedingter Sprung:
If $x_i = c$ Then Goto M_i
  - ▷ Stoppanweisung: Halt
Beendet das Programm. Ergebnis in x_1.
  - ▷ Äquivalent zu While und Turing.

- **Primitiv rekursive Funktionen**: Funktionale Programmierung, induktive Definition:
  - ▷ Konstante Funktionen:
$k_m : \mathbb{N} \to \mathbb{N},\quad k_m(n) = m$
  - ▷ Projektionen:
$\pi_i^k : \mathbb{N}^k \to \mathbb{N},\quad \pi_i^k(n_1,\ldots,n_k) = n_i$
  - ▷ Nachfolgerfunktion:
$s : \mathbb{N} \to \mathbb{N},\quad s(n) = n+1$
  - ▷ Ineinander Einsetzen von primitiv rekursiven Funktionen: Sei g, f_i prim. rek., $g(f_1(n_1,\ldots,n_k),\ldots,f_k(n_1,\ldots,n_k))$

ergibt immer primitiv rekursive Funktionen.
Dazu: Primitive Rekursion:
  - ▷ $f(0,n_1,\ldots,n_k) = g(n_1,\ldots,n_k)$
  - ▷ $f(n+1,n_1,\ldots,n_k) =$
$h(f(n,n_1,\ldots,n_k),n,n_1,\ldots,n_k)$

Es gilt:
  - ▷ Primitiv rekursive Berechenbarkeit entspricht genau den Loop-Programmen.
Beweise für das eine können also genauso mit dem anderen geführt werden, falls dies einfacher ist.

Bsp.:
  - ▷ Additionsfunktion $\text{add} : \mathbb{N}^2 \to \mathbb{N}$
$\text{add}(0,m) = m$
$\text{add}(n+1,m) = s(\text{add}(n,m))$
  - ▷ Multiplikationsfunktion $\text{mult} : \mathbb{N}^2 \to \mathbb{N}$
$\text{mult}(0,m) = 0$
$\text{mult}(n+1,m) = \text{add}(\text{mult}(n,m),m)$
  - ▷ Dekrementierung $\text{dec}(n)$,
Subtraktion $\text{sub}(n,m)$
  - ▷ $n \mapsto \binom{n}{2}$

Diese Funktionen können auch direkt zum Bauen neuer primitiv rekursiver Funktionen genutzt werden.
  - ▷ **Paarungsfunktion** $c : \mathbb{N}^2 \to \mathbb{N}$
    - Bijektion: ein Tupel kann eindeutig durch eine Zahl $\in \mathbb{N}$ kodiert werden
    - Nacheinanderanwendung: beliebige k-Tupel können kodiert werden
    - Schreibe: $\langle n_1,\ldots,n_k\rangle = c(\ldots) = n$
    - **Dekodierungsfunktion**
$d_i(\langle n_1,\ldots,n_k\rangle) = n_i$
  - ▷ **Prädikate**: Funktionen $P : \mathbb{N}^{k+1} \to \{0,1\}$
    - beschränktes Max.: $q_P(n,n_1,\ldots,n_k)$
(größtes $x < n$, für das $P(x,n_1,\ldots,n_k) = 1$ ist)
ist prim. rek., wenn P prim. rek.
    - beschränkte Exist.: $Q_P(n,n_1,\ldots,n_k)$
(1, falls es ein $x < n$ gibt, für das $P(x,n_1,\ldots,n_k) = 1$ ist, sonst 0)
ist prim. rek., wenn P prim. rek.

- **μ-rekursive Funktionen**
  - ▷ wie prim. rek. Funktionen definiert
  - ▷ zusätzlich: **μ-Operator**
verwandelt Funktion $f : \mathbb{N}^{k+1} \to \mathbb{N}$ in $\mu f : \mathbb{N}^k \to \mathbb{N}$ mit $\mu f(x_1,\ldots,x_k) =$ kleinste Nullstelle von $f(n,x_1,\ldots,x_n)$ bzgl. n; falls keine Nullstelle: undefiniert
Berechne dazu $f(0,\ldots), f(1,\ldots),\ldots$ Gib ersten Wert zurück. Falls Berechnung nicht terminiert, analog While: n.def.
  - ▷ **Bsp.:**
    - $\Omega = \mu f$ mit $f(x,y) = 1\ \forall x,y$
    - $\text{sqrt}(n) = \lceil\sqrt{n}\rceil$ ist μ-rek.
    - $\text{nlog}(b,x) = \lceil\log_b x\rceil$ ist μ-rek.
  - ▷ Klasse der μ-rek. Funktionen entspricht Turing-, While-, Goto-Berechenbarkeit
  - ▷ Ackermannfunktion $a : \mathbb{N}^2 \to \mathbb{N}$
$a(0,y) = y+1$
$a(x,0) = a(x-1,1)$
$a(x,y) = a(x-1,a(x,y-1))$
ist total, wächst schnell, ist While-berechenbar, aber nicht Loop-berechenb.

3.2 Entscheidbarkeit

- Sprache $A \subseteq \Sigma^*$ ist **entscheidbar**, wenn die **charakteristische Funktion** berechenbar ist:
$$\chi_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ 0 & \text{falls } w \notin A \end{cases} \quad \text{mit } w \in \Sigma^*$$
d.h.: nach endlich vielen Schritten sagt TM eindeutig, ob w zur Sprache gehört oder nicht
- Sprache A ist **semi-entscheidbar**, wenn die **halbe charakteristische Funktion** berechenb.:
$$\chi'_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ \text{undefiniert} & \text{falls } w \notin A \end{cases}$$
Das entspricht genau den Typ-0-Sprachen.
  - ▷ Zum Berechnen von χ'_A kann man eine TM der Sprache A nehmen, und statt Endzustand eine 1 aufs Band schreibt

3.3 Probleme

- Probleme auf Wörtern (charakteristische Funktion) kann man durch Kodieren der Wörter als Zahlen auch mit Funktionen berechnen
- **Allgemeines Wortproblem**:
Menge $A = \{(w,G) \mid w \in L(G)\}$
mit G = kodierte Chomsky-Grammatik
  - ▷ Prüfen, ob w zu einer $L(G)$ gehört, mit TM, die w *und* G als Eingabe erhält
  - ▷ ist *unentscheidbar!*

- Sprache A entscheidbar genau dann, wenn A entscheidbar $\wedge$ $\overline{A}$ entscheidbar
- A **rekursiv aufzählbar**, wenn $\exists f : \mathbb{N} \to \Sigma^*$ mit $A = \mathrm{im}(f)$ (f total und berechenbar)
  - ▷ mathematisch abzählbar ähnlich definiert, f muss nicht berechenbar sein
  - ▷ genau dann rekursiv aufzählbar, wenn sie semi-entscheidbar ist (Typ-0)
- **Kodierung einer TM**: $\mathrm{code}(M)$ $M_w = M$, wenn $\mathrm{code}(M) = w$, oder eine „Rückfall"-TM, falls es kein gültiger Code ist
  - ▷ Es gibt eine TM, die jede andere TM damit simulieren kann (**„Universelle TM"**)
- **Allgemeines Halteproblem**: Sprache $H = \{w\#x | w, x \in \{0,1\}^*, M_w \text{ angesetzt auf } x \text{ hält}\} = \{w\#x | w, x \in \{0,1\}^*, x \in T(M_w)\}$
  - ▷ ist *unentscheidbar* (aus Reduktion des speziellen Halteproblems durch $f(w) = w\#w$, s.u.)
- **Spezielles Halteproblem**: Sprache $K = \{w \in \{0,1\}^* | M_w \text{ angesetzt auf } w \text{ hält}\}$ d.h.: „die TM akzeptiert ihre eigene Kodierung"
  - ▷ ist *unentscheidbar!* (wichtig!)
  - ▷ Beweis durch Widerspruch (Diagonal.): Sei M die TM, die das spezielle Halteproblem berechnet. Dann sei M' die TM, die bei Eingabe von w genau dann in eine Endlosschleife geht, wenn M „1" ausgibt, sonst selbst 0 ausgibt. Gibt man in M' nun w' so ein, dass $M_{w'} = M'$, dann müsste sie genau dann halten, wenn sie selbst nicht hält. $\lightning$
  - ▷ ist semi-entscheidbar
- **Reduktion** eines Problems $B \subseteq \Sigma^*$ auf ein Problem $A \subseteq \Gamma^*$ (schreibe $B \leq A$): Zum Beweis von Unentscheidbarkeit ein Problem auf ein bereits als unentscheidbar bekanntes Problem zurückführen
  - ▷ d.h. $\exists f : \Sigma^* \to \Gamma^*$ mit $\forall x \in \Sigma : x \in B \Leftrightarrow f(x) \in A$
  - ▷ Wenn $B \leq A$: Falls A entscheidbar, ist auch B entscheidbar. Falls B unentscheidbar, ist auch A unentscheidbar.
- **Halteproblem auf leerem Band**: Sprache $H_0 = \{w \in \{0,1\}^* | M_w \text{ hält auf Eingabe } \varepsilon\}$
  - ▷ ist *unentscheidbar* ($H \leq H_0$)
- **Satz von Rice**: Es ist unentscheidbar, ob eine TM eine bestimmte Eigenschaft $\mathscr{S}$ hat. *Formal:* Sei $\mathscr{R}$ die Klasse aller Turing-berechenbaren Funktionen und $\mathscr{S} \subseteq \mathscr{R}$. Dann ist die Sprache $C(\mathscr{S}) = \{w \in \{0,1\}^* | \text{von } M_w \text{ berechnete F. liegt in } \mathscr{S}\}$ unentscheidbar (Reduktion $\overline{H_0} \leq C(\mathscr{S})$)
  - ▷ Kein Programm kann die inhaltliche Korrektheit anderer Software überprüfen.
- Für ein GOTO-Programm mit 1 Variable ist das Halteproblem entscheidbar (simulierbar mit PDA). Für mehr als 1 Variable unentscheidbar.
- **Busy-Beaver-Funktion** $\Sigma(n) =$ die maximale Anzahl „1"en, die von einer TM mit $\Gamma = \{1, \square\}$ und n Zuständen geschrieben werden kann, wenn sie auf dem leeren Band terminiert
  - ▷ nicht berechenbar ($H_0 \leq \Sigma$)
  - ▷ wächst extrem schnell ($\Sigma(6) > 1{,}29 \cdot 10^{865}$), bis heute nicht *genau* bekannt
- **Postsches Korrespondenzproblem (PCP)**: Eingabe: endliche Liste von Wortpaaren $I = ((x_1, y_1), \ldots, (x_k, y_k))$, Frage: Gibt es eine Folge von Indizes dieser Liste $i_1, \ldots i_n \in \{1, \ldots, k\}$, sodass gilt $x_{i_1} x_{i_2} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$?
  - ▷ ist nur semi-entscheidbar (brute force)
  - ▷ **Modifiziertes PCP (MPCP):** Wie PCP, aber es muss $i_1 = 1$ sein
  - ▷ $H \leq \text{MPCP} \leq \text{PCP}$
  - ▷ $\text{PCP}_{m,n}$: Beschränkung auf n-elem. Alphabet und maximal m Wortpaare
    - – bereits $\text{PCP}_{5,2}$ ist unentscheidbar.
    - – $\text{PCP}_{m,1}, \text{PCP}_{2,n}$ sind entscheidbar.
    - – $\text{PCP}_{3,2}, \text{PCP}_{4,2}$ unbekannt.
- **Schnittleerheit**, **Inklusions-** und **Äquivalenzproblem** für **kontextfreie** Grammatiken sind unentscheidbar, weil ≥ PCP

4 Sonstiges

- **Potenzgesetze** für $a, b, x, y \in \mathbb{C}$
  - ▷ $x^a \cdot x^b = x^{a+b}$
  - ▷ $(x^a)^b = x^{a \cdot b}$
  - ▷ $x^a \cdot y^a = (x \cdot y)^a$
  - ▷ $x^{-a} = \frac{1}{x^a}$
  - ▷ $x^{1/a} = \sqrt[a]{x}$
- **Wurzelgesetze** für $x, y, n, m \in \mathbb{C}$:
  - ▷ $\sqrt[n]{x} \cdot \sqrt[n]{y} = \sqrt[n]{x \cdot y}$ wenn $x \cdot y \geq 0$
  - ▷ $\left(\sqrt[n]{x}\right)^m = \sqrt[n]{x^m}$
  - ▷ $\sqrt[m]{\sqrt[n]{x}} = \sqrt[n \cdot m]{x}$
- **Logarithmusgesetze** für $a, b, p, q, n \in \mathbb{C}$
  - ▷ $\log_b b = 1$
  - ▷ $\log_b 1 = 0$
  - ▷ $\log_b (p \cdot q) = \log_b p + \log_b q$
  - ▷ $\log_b \frac{p}{q} = \log_b p - \log_b q$
  - ▷ $\log_b p^n = n \cdot \log_b p$
  - ▷ $\log_b \sqrt[n]{p} = \frac{\log_b p}{n}$
  - ▷ $\log_a p = \frac{\log_b p}{\log_b a}$
- „Gauß-Klammern" für $\mathbb{R} \to \mathbb{Z}$ bzw. $\mathbb{R}_0^+ \to \mathbb{N}$:
  - ▷ $\lfloor x \rfloor = x$ abgerundet („floor function")
  - ▷ $\lceil x \rceil = x$ aufgerundet („ceiling function")
- Gaußsche Summenformel: $\sum_{k=0}^{n} k = \frac{n \cdot (n+1)}{2}$