Vorlesung Formale Sprachen und Berechenbarkeit

Version vom SS 2024*

Universität Mainz
Institut für Physik
Theorie der kondensierten Materie
Prof. Dr. Friederike Schmid†

Inhalt: Mathematische Grundlagen
- **Elementare Strukturen**
- **Grundbegriffe der Logik**
- **Beweistechniken**

Formale Sprachen
- **Grammatiken**
- **Reguläre Sprachen**
- **Kontextfreie Sprachen**

Berechenbarkeit
- **Rechnermodelle**
- **Entscheidbarkeit**

---

*Elektronisch: Letzte Änderung am 01.08.24

†Staudingerweg 9, 03-534, Tel. (06131-)3920365, <Friederike.Schmid@uni-mainz.de>

Inhaltsverzeichnis

Vorbemerkungen

„Formale Sprachen und Berechenbarkeit“: Grundvorlesung der Theoretischen Informatik, zusammen mit „Komplexitätstheorie”

Konkrete **Struktur** der Vorlesung

- Mathematische Grundlagen (kurz)
- Formale Sprachen
  - Grammatiken
  - Reguläre Sprachen
  - Kontextfreie Sprachen
- Berechenbarkeit
  - Rechnermodelle und Berechenbarkeit
  - Entscheidungsprobleme

Einige empfohlene Bücher:

A. Asteroth, C. Baier *Theoretische Informatik*

J.E. Hopcroft, R. Motwani, J.D. Ullmann: *Einführung in Automatentheorie, Formale Sprachen und Berechenbarkeit*

D.W. Hoffmann: *Theoretische Informatik*

G. Schnitger: *Formale Sprachen und Berechenbarkeit*

Kapitel 1

Mathematische Grundlagen

Dieses Kapitel: Kurze Wiederholung mathematischer Grundlagen
aus der Vorlesung "diskrete Mathematik"
Vorab etwas Notation:
- $\forall$: "Für alle"
- $\exists$: "Es existiert", $\exists!$: "Es existiert genau ein"

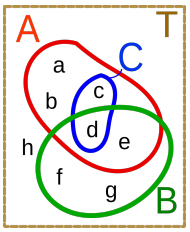
1.1 Elementare Strukturen

1.1.1 Mengen

1.1.1.1 Definitionen

Menge: Verbund von Objekten (Elementen), z.B. $A = \{a, b, c, d, e\}$

- Elemente: $a \in A$
- Leere Menge: $\varnothing = \{\}$ (keine Elemente)
- Teilmenge: z.B. $C = \{c, d\} \subset A$
  (falls auch "gleich" erlaubt ist: $\subseteq$)
- Obermenge: z.B. $A \supset C$ (bzw. $\supseteq$)

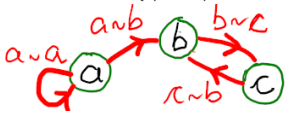


1.1.1.2 Mengenoperationen

- Vereinigungsmenge: $A \cup B = \{x : x \in A \text{ oder } x \in B\}$ $(\{a, b, c, d, e, f, g\})$
- Schnittmenge: $A \cap B = \{x : x \in A \text{ und } x \in B\}$ $(\{d, e\})$
- Differenzenmenge: $A \backslash C = \{x : x \in A \text{ und } x \notin C\}$ $(\{a, b, e\})$
- Komplement: Gegeben eine allumfassende "'Trägermenge"' T.
  Dann ist $\overline{M} = T \backslash M = \{x \in T : x \notin M\}$
  Es gilt: $\overline{M_1 \cup M_2} = \overline{M_1} \cap \overline{M_2}$, $\overline{M_1 \cap M_2} = \overline{M_1} \cup \overline{M_2}$
  (de Morganschen Gesetze)
- Potenzmenge: $2^M = \{M' : M' \subseteq M\}$
  (Menge aller Teilmengen von M)

1.1.1.3 Relationen

⋆ Definition: Beziehung zwischen Elementen einer Menge M
$R \subseteq M \times M$: Teilmenge der Produktmenge $M \times M = \{(x, y) : x, y, \in M\}$
Schreibweise: $x \sim_R y$ oder $x \sim y$, z.B.



⋆ Mögliche Attribute einer Relation

- reflexiv, falls $x \sim x \; \forall \, x$
- symmetrisch, falls mit $x \sim y$ immer auch $y \sim x$ gilt
- transitiv, falls aux $x \sim y$ und $y \sim z$ folgt $x \sim z$
- Äquivalenzrelation, falls sie reflexiv, symmetrisch und transitiv ist

Beispiele:
– Rationale Zahlen $\mathbb{Q}$: Menge der Brüche p/q
"$=$": Äquivalenzrelation $(2/4 = 4/8 = ...)$
"$\leqslant$": reflexiv, transitiv
"$<$": transitiv

– Facebook -Nutzer:
"befreundet": symmetrisch, nicht transitiv

⋆ Verknüpfungen von Relationen

– Gegeben zwei Relationen R, S auf einer Menge M
Dann ist $R \cdot S = \{(x, y) : \exists z$ mit $x \sim_R z$ und $z \sim_S y\}$
$R^{-1} = \{(y, x) : (x, y) \in R\}$

– Potenzen von Relationen
$R^2 = R \cdot R$, $R^3 = R \cdot R^2$, ... $R^n := R \cdot R^{n-1}$ (rekursiv definiert)
$R^1 := R$ $R^0 := \{(x, x) : x \in M\}$ (Identität)

⋆ Reflexiv-transitive Hülle R^* einer Relation R

Kleinste Relation, die $R^+ = \bigcup_{i=1}^{\infty} R^i$ einschließt und reflexiv ist.

$\rightsquigarrow$ Für Relation $\sim$ ist $\sim^*$ definiert durch:
$x \sim^* x \; \forall \, x$; $\quad x \sim^* y \Leftrightarrow \exists z_1, ...z_k$ mit $x \sim z_1, z_1 \sim z_2, ...z_k \sim y$

Relationen werden in dieser Vorlesung eine wichtige Rolle spielen.

1.1.1.4 Funktionen

⋆ Definition: Gegeben zwei Mengen M, N

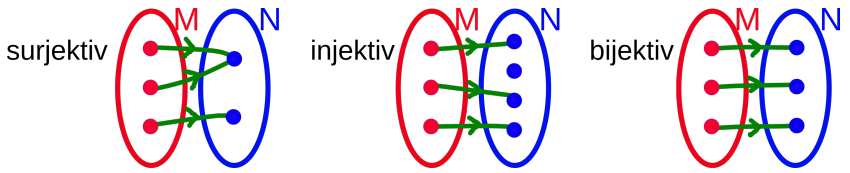
Eine Funktion ist eine Vorschrift, die den Elementen $a \in M$ ein Element $\varphi(a) \in N$ zuordnet.

Notation: $\varphi : \begin{array}{ccc} M & \to & N \\ a & \to & \varphi(a) \end{array}$ mit $\begin{array}{ll} a: & \text{Urbild} \\ \varphi(a): & \text{Bild} \end{array}$

Speziell: Totale Funktion: $\varphi(x)$ definiert für alle $x \in M$
Partielle Funktion: $\varphi(x)$ für manche $x \in M$ undefiniert
Notation in diesem Fall: $\varphi(x) = \perp$

⋆ Eine totale Funktion heißt

surjektiv: Falls alle Elemente von N ein Urbild haben
injektiv: Falls Elemente von N maximal ein Urbild haben (auch: eindeutig)
bijektiv: Injektiv und surjektiv (auch: eineindeutig)



1.1.1.5 Mächtigkeit

Die Einführung der bijektiven Funktionen erlaubt den Vergleich von Mengen

⋆ Zwei Mengen M, N heißen gleichmächtig, ($|M| = |N|$), falls es eine bijektive Abbildung zwischen ihnen gibt.

⋆ Falls Mengen endlich sind, sind sie genau dann gleichmächtig, wenn sie gleich viele Elemente haben.
$\rightsquigarrow$ Definiere $|A|$ = Anzahl Elemente

Falls Mengen unendlich sind: Schwieriger (Abschnitt 1.1.2)

⋆ Potenzmenge: Es gilt $|M| < |2^M|$

(Beweis: $|M| \leqslant |2^M|$ ist klar (da für alle $x \in M$ gilt: $\{x\} \in 2^M$)
Wenn $|M| = |2^M|$ wäre, dann gäbe es eine surjektive Abbildung $f : M \to 2^M$
Definiere $M' = \{x \in M : x \notin f(x)\}$. Dann gilt natürlich $M' \in 2^M$
Da f surjektiv und $M' \in 2^M$, gibt es ein $y \in M$ mit $f(y) = M'$
Aber: Falls $y \notin f(y) = M'$, folgt $y \in M'$ laut Definition von M'
Falls $y \in f(y) = M'$, folgt $y \notin M'$ laut Definition von M'
Damit ergibt sich in jedem Falle ein Widerspruch! ▫)

1.1.2 Zahlen

Spezielle Mengen, die im Folgenden diskutiert werden.

1.1.2.1 Natürliche Zahlen $\mathbb{N}$

$\mathbb{N} = \{1, 2, 3, ...\}$: "natürlich", da von Alters her zum Zählen benutzt

⋆ Axiomatische Definition über Peano Axiome (G. Peano, 1858–1932)

I: 1 ist eine natürliche Zahl
II: Jede natürliche Zahl $a \in \mathbb{N}$ hat genau einen Nachfolger $a^+ \in \mathbb{N}$
III: Es gibt keine Zahl mit dem Nachfolger 1
IV: Aus $a^+ = b^+$ folgt $a = b$
V: Enthält eine Teilmenge von $\mathbb{N}$ die Zahl 1 und zu jeder Zahl a auch deren Nachfolger a^+, so enthält sie alle natürlichen Zahlen.

NB: Axiom V begündet die Beweismethode der vollständigen Induktion, siehe Kapitel 1.1.3.

⋆ Auf $\mathbb{N}$ kann man eine Addition und Multiplikation definieren.

(Addition: über $x+1=x^+,\ x+y^+=(x+y)^+\ \forall\, x,y\in\mathbb{N}$)
Multiplikation: über $x\cdot 1=x,\ x\cdot y^+=x\cdot y+x\ \forall\, x,y\in\mathbb{N}$)

⋆ Häufig erweitert man: $\mathbb{N}_0=\mathbb{N}\cup\{0\}$ mit $0+a=a\ \forall a\in\mathbb{N}_0$

1.1.2.2 Weitere Zahlenmengen

⋆ Ganze Zahlen $\mathbb{Z}=\{...,-2,-1,0,1,2,...\}=\mathbb{N}_0\cup\{-a:a\in\mathbb{N}\}$
Ergänze $\mathbb{N}_0$ so, dass die Gleichung $a+x=b$ lösbar wird für alle $a,b\in\mathbb{Z}$

⋆ Rationale Zahlen $\mathbb{Q}=\{p/q:p\in\mathbb{Z},q\in\mathbb{N}\}$
Ergänze $\mathbb{Z}$ so, dass $a\cdot x=b$ lösbar wird für alle $a\in\mathbb{Q},\ b\in\mathbb{Q}\backslash\{0\}$

⋆ Reelle Zahlen $\mathbb{R}$
Vervollständigt die Zahlengerade, schließt Lücken
"Menge aller Dezimalzahlen mit unendliche vielen Stellen "

1.1.2.3 Abzählbarkeit

Problem: Was ist die Mächtigkeit der Zahlenmengen $\mathbb{N},\mathbb{Z},\mathbb{Q},\mathbb{R}$?
Sind sie gleichmächtig? (schwierig, da unendlich groß !)
Diese Überlegungen sind wichtig für Kapitel 3.

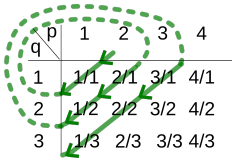
Ausgangspunkt: Unsere Referenz ist $|\mathbb{N}|$: "Abzählbar unendlich"

Frage: Wie steht es mit $|\mathbb{Z}|$, $|\mathbb{Q}|$, $|\mathbb{R}|$?

⋆ $|\mathbb{Z}|$: Sortiere Elemente um und nummeriere durch: $\left|\begin{array}{cccccc} 0, & -1, & 1, & -2, & 2, & \cdots \\ 1, & 2, & ,3, & 4, & 5, & \cdots \end{array}\right|$

$\rightsquigarrow$ Definiert bijektive Abbildung $\mathbb{Z}\leftrightarrow\mathbb{N}\ \Rightarrow\ |\mathbb{Z}|=|\mathbb{N}|$ ist abzählbar!

⋆ $|\mathbb{Q}|$: Cantorsche Konstruktion
(hier für $p,q\in\mathbb{N}$
Erweiterung ist einfach.)



$\rightsquigarrow$ Bijektive Abbildung $\mathbb{Q}\leftrightarrow\mathbb{N}\ \Rightarrow\ |\mathbb{Q}|=|\mathbb{N}|$ ist abzählbar!

⋆ $|\mathbb{R}|$? Nicht mehr abzählbar (Cantor, 1874)

(Beweis: (Hoffmann, S. 62) Betrachte nur Intervall $I=[0,1[$
Angenommen, es gäbe eine bijektive Abbildung $f(n)=x\in I,\ n\in\mathbb{N}$
Betrachte
$$\begin{aligned} f(1) &= 0,p_{11}p_{12}p_{13}\cdots \\ f(2) &= 0,p_{21}p_{22}p_{23}\cdots \\ &\vdots \end{aligned}$$
mit $p_{ij}\in\{0,1,...,9\}$: Ziffern
Konstruiere $r=0,q_1q_2q_3\cdot$ mit $q_j\neq p_{jj}$ (z.B. $q_j=p_{jj}\pm 1$)
$\rightsquigarrow$ Dann gibt es kein n mit $f(n)=r$ (da $p_{nn}\neq q_n\ \forall\, n$) $\rightsquigarrow$ Widerspruch! □)

$\Rightarrow |\mathbb{R}|>|\mathbb{N}|$: $|\mathbb{R}|$ ist überabzählbar unendlich!

Anmerkung: Man kann sich fragen, ob es eine Zwischenstufe zwischen "abzählbar unendlich" und "überabzählbar" gibt: Gibt es eine Teilmenge $M \subset \mathbb{R}$ mit $|\mathbb{N}| < |M| < |\mathbb{R}|$?

Es stellt sich heraus: Diese Frage ist innerhalb der Axiome der Mengenlehre nicht entscheidbar (Paul Cohen, 1963)
$\rightsquigarrow$ Zusätzliches Postulat: Kontinuumshypothese

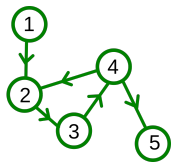
⋆ Potenzmenge: Wegen $|\mathbb{N}| < |2^{\mathbb{N}}|$ (siehe 1.1.1.5) ist $|2^{\mathbb{N}}|$ überabzählbar.

1.1.3 Graphen und Bäume

Graphen werden in dieser Vorlesung eine wichtige Rolle spielen.
Im Wesentlichen: Endliche Menge von "Knoten" mit einer Relation ("Kanten")
Beispiele: – Straßennetze (Kreuzungen und Strassen)
– Internet (Webseiten und Links)
– Soziale Netzwerke (Personen und Freundschaften)
– Darstellungen von Gleichungssystemen in der Mathematik

1.1.3.1 Graphen

z.B.



⋆ Formale Definition eines Graphen

Ein Graph $G = (V, E)$ ist definiert durch:
– Eine endliche Menge V von Knoten
– Eine Menge $E \subseteq \{(x, y) : x, y \in V\}$ von Kanten $\triangleq$ Relation auf V

Man unterscheidet zwischen
– Ungerichteten Graphen: Relation symmetrisch
(Pfeile in graphischen Darstellungen können weggelassen werden)
– Gerichteten Graphen: Reihenfolge in Relation wichtig.

⋆ Nomenklatur
– Eine Kante (u, v) heißt inzident zu u und v
– Falls es eine Kante (u, v) gibt, ist u adjazent zu v
– Grad eines Knotens:
Ungerichtete Graphen: Anzahl inzidenter Kanten
Gerichtete Graphen: Ausgangs- bzw. Eingangsgrad eines Knotens:
$\rightsquigarrow$ Anzahl auslaufender bzw. einlaufender Kanten
– Schleife: Kante (u, u)

⋆ Pfade: Zusammenhängende Wege entlang der Kanten eines Graphs:
$(v_0, \cdots, v_k)$ mit $v_j \in V \; \forall \; j, \; (v_{j-1}, v_j) \in E$ für $1 < j \leqslant k$
Auch möglich ist der "leere Pfad" (v_0) von einem Knoten zu sich selbst.

Speziell: – Einfacher Pfad: Knoten werden maximal einmal durchlaufen.
– Zyklus: $v_k = v_0$ (für $k \geqslant 1$)
– Einfacher Zyklus: Zyklus und einfacher Pfad
– Kürzester Pfad von u nach v:
Es gibt keinen Pfad von u nach v mit weniger Kanten.

⋆ Weitere Definitionen. Ein Graph heißt ...

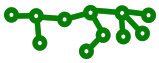
- Azyklisch, wenn es keinen einfachen Zyklus $(v_0, ..., v_k)$ gibt mit $k \geqslant 1$ (gerichtete Graphen) bzw. $k \geqslant 3$ (ungerichtete Graphen).
- Zusammenhängend, wenn alle Knoten $u, v \in G$ mit einem Pfad verbunden werden können (im Fall gerichteter Graphen: gerichtete Pfade)

Anmerkung: Ungerichtete Graphen, die nicht zusammenhängend sind, kann man in Zusammenhangskomponenten aufteilen: Maximale zusammenhängende Teilgraphen.

- Stark zusammenhängend von einem Knoten r aus, wenn es von r zu jedem Knoten $v \in V$ einen gerichteten Pfad gibt (nur gerichtete Graphen).

1.1.3.2 Bäume

⋆ Ein ungerichteter Baum ist ein ungerichteter Graph, der azyklisch und zusammenhängend ist.



Es gelten folgende Theoreme:

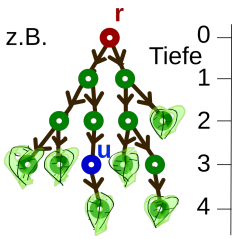
1) Um zu zeigen, dass ein ungerichteter Graph G ein Baum ist, reicht es, zwei der drei folgenden Aussagen zu zeigen:
   a) G ist zusammenhängend
   b) G ist azyklisch
   c) G hat bei n Knoten $(n-1)$ Kanten
2) Zwischen je zwei Knoten eines Baums gibt es genau einen einfachen Pfad.

⋆ Gewurzelte Bäume: Azyklische gerichtete Graphen, die stark zusammenhängend sind von einem Wurzel-knoten r aus.

Man erhält sie aus ungerichteten Bäumen, indem man einen Knoten r auszeichnet (die Wurzel) und alle Kanten so orientiert, dass sie von r wegführen ("Out-tree").

Nomenklatur: Sei B ein Baum, u ein Knoten,
$(v_0 = r, v_1, ...v_k = u)$ der einfache Pfad von r nach u
Dann ist/sind:

- $v_0, ..., v_{k-1}$: Vorfahren von u
- v_{k-1}: Eltern oder Vorgänger von u
- u: Kind von v_{k-1}
- k: Tiefe des Knotens u (im Bild: 3)



Weiterhin:

- Tiefe des Baums B: Maximale Tiefe der Knoten im Baum (im Bild: 4)
- Blatt: Knoten ohne Kinder
- Geordneter Baum: Alle Kinder jedes Knotens sind geordnet
- Binärbaum: Geordneter Baum mit maximal zwei Kindern pro Knoten ("linkes" und "rechtes" Kind)

1.2 Grundbegriffe der Logik

1.2.1 Aussagenlogik

(siehe Buch von Hoffmann, Kapitel 3.1)

1.2.1.1 Aussagen und logische Operatoren

⋆ Aussagen (nach Aristoteles)

> "Eine Aussage ist ein sprachliches Gebilde, von dem es sinnvoll ist, zu sagen, es sei "wahr" oder "falsch"

$\rightsquigarrow$ Aussagen haben einen Wahrheitswert "0" oder "1"

Beispiele: – "1+1 = 2" (wahr: Wahrheitswert 1)
– "1+1 = 3" (falsch: Wahrheitswert 0)
– "XY wird Kanzler/in" (Wahrheitswert unbekannt)

Aber: Keine Aussagen sind:
– "Donald Trump war ein guter Präsident"
– "Dieser Satz ist falsch"

NB: Nach unserer Definition ist Logik zweiwertig. Eine Aussage ist entweder wahr oder falsch. Damit folgen wir der klassischen Logik.
$\rightsquigarrow$ "Nicht falsch" ist gleichbedeutend mit "wahr"

Es gibt alternative Ansätze, z.B. die "intuitionistische" Logik. Dort ist "wahr" gleichbedeutend mit "beweisbar".
$\rightsquigarrow$ Eine Aussage kann dann gleichzeitig "nicht wahr" und "nicht falsch" sein ($\triangleq$ unentscheidbar)
$\rightsquigarrow$ Manche der unten diskutierten Regeln gelten dann nicht mehr!

⋆ Verknüpfungen von Aussagen

Aussagen können zu neuen Aussagen zusammengesetzt werden.
z.B. $F \equiv$ "Wenn es nicht regnet und wir Hunger haben, grillen wir!"
lässt sich schreiben als $F \equiv (\neg A \wedge B) \rightarrow C$
mit $A \equiv$ "Es regnet"
$B \equiv$ "Wir haben Hunger"
$C \equiv$ "Wir grillen"

Dazu setzt man logische Operatoren ein (hier: $\neg, \wedge, \rightarrow$)

⋆ Tabelle möglicher Operatoren und Verknüpfungen (Auswahl)

Negation:

A	$\neg A$
0	1
1	0

Verknüpfungen:

A	B	$A \wedge B$	$A \vee B$	$A \to B$	$A \leftrightarrow B$	$A \oplus B$
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	0	0	1
1	1	1	1	1	1	0
		Konjunktion (AND)	Disjunktion (OR)	Implikation	Äquivalenz (XAND)	Antivalenz (XOR)

Es gelten die folgenden Klammerregeln: $\neg$ vor $\wedge$ vor $\vee$ vor $(\to, \leftrightarrow, \oplus)$

NB: Implikation $A \to B \Leftrightarrow \neg(A \wedge \neg B) \Leftrightarrow \neg A \vee B$

1.2.1.2 Aussagenlogische Formeln

Gegenstand der Logik ist die Erforschung allgemeingültiger Zusammenhänge zwischen logischen Ausdrücken. $\rightsquigarrow$ Erfordert abstrahieren von konkreten Aussagen (ähnlich Algebra: Übergang von Ausdrücken wie "1+1=2" zu Ausdrücken wie "$a^2 + b^2 = c^2$")

⋆ Übergang von der Objektebene zur Metaebene mit

- Logischen Variablen: Platzhalter für Aussagen
- Logischen Formeln: Zusammenhängende Ausdrücke über einer Menge von Variablen
- Belegung: Zuweisung konkreter Aussagen an Variablen
- Modell für logische Formel F: Eine Belegung, in der F wahr ist.

⋆ Eigenschaften: Eine aussagenlogische Formel F heißt

- Erfüllbar, wenn es mindestens ein Modell für F gibt
- Kontradiktion, wenn F nicht erfüllbar ist (z.B. $F \equiv A \wedge \neg A$)
- Tautologie, wenn F für alle Belegungen wahr ist (z.B. $F \equiv A \vee \neg A$)
- Logische Folgerung aus G, $G \Rightarrow F$, wenn F wahr ist für jedes Modell von G
- Äquivalent zu G, $F \Leftrightarrow G$, wenn $F \Rightarrow G$ und $G \Rightarrow F$

⋆ Umformungsregeln

- Doppelnegation: $\neg(\neg A) \Leftrightarrow A$
- Regeln von de Morgan: $\begin{cases} \neg(A \vee B) & \Leftrightarrow & \neg A \wedge \neg B \\ \neg(A \wedge B) & \Leftrightarrow & \neg A \vee \neg B \end{cases}$
- Kontraposition $A \to B \Leftrightarrow \neg B \to \neg A \qquad (\Leftrightarrow \neg A \vee B)$
- Distributivität $\begin{cases} A \vee (B \wedge C) & \Leftrightarrow & (A \vee B) \wedge (A \vee C) \\ A \wedge (B \vee C) & \Leftrightarrow & (A \wedge B) \vee (A \wedge C) \end{cases}$

Beispiel: Vereinfachung einer aussagenlogischen Formel

$$\neg(\neg A \wedge B) \wedge (A \vee B) \underset{\text{de Morgan}}{\Leftrightarrow} (A \vee \neg B) \wedge (A \vee B) \underset{\text{Distributivges.}}{\Leftrightarrow} A \vee \underbrace{(\neg B \wedge B)}_{\varnothing} \Leftrightarrow A$$

1.2.2 Prädikatenlogik

(sehr rudimentär, mehr siehe Hoffmann Kapitel 3.2)

Die Aussagenlogik aus 1.2.1 ist zur Beschreibung vieler logischen Aussagen und Zusammenhänge noch nicht gut geeignet.

Berühmtes Beispiel: $A1 \equiv$ "Sokrates ist ein Mensch"
$A2 \equiv$ "Alle Menschen sind sterblich"
$A3 \equiv$ "Also ist Sokrates sterblich" (Folgerung)

$\rightsquigarrow$ Brauche logische Aussagen über Mengen $\Rightarrow$ Prädikatenlogik

$\star$ Neue logische Konstrukte

- Prädikat: Abbildung $P : \Omega \to \{0, 1\}$ (Ω: Beliebige Menge)
- Quantoren:
  Existenzquantor $\exists x\, P(x)$ wahr, wenn $P(x) = 1$ für ein $x \in \Omega$
  Allquantor $\forall x\, P(x)$ wahr, wenn $P(x) = 1$ für alle $x \in \Omega$

Beispiel Sokrates (s.o.)
Sei $M(x) :=$"x ist ein Mensch", $S(x) :=$ "x ist sterblich"
Dann ist $A1 \equiv M(\text{Sokrates})$, $A2 \equiv \forall x\, (M(x) \to S(x))$;
$\{A1, A2\} \Rightarrow A3 \equiv S(\text{Sokrates})$

$\star$ Umformungsregeln

- Negationsregel: $\neg(\exists x\, P(x)) \;\Leftrightarrow\; \forall x\, \neg P(x)$
  $\neg(\forall x\, P(x)) \;\Leftrightarrow\; \exists x\, \neg P(x)$
- Ausklammern: $(\forall x\, P(x)) \wedge (\forall x\, Q(x)) \;\Leftrightarrow\; \forall x\, (P(x) \wedge Q(x))$
  $(\exists x\, P(x)) \vee (\exists x\, Q(x)) \;\Leftrightarrow\; \exists x\, (P(x) \vee Q(x))$
- Vertauschen: $\forall x\, \forall y\, P(x, y) \;\Leftrightarrow\; \forall y\, \forall x\, P(x, y)$
  $\exists x\, \exists y\, P(x, y) \;\Leftrightarrow\; \exists y\, \exists x\, P(x, y)$
  (aber: $\forall x\, \exists y$ und $\exists y\, \forall x$ dürfen nicht vertauscht werden!)

1.3 Beweistechniken

1.3.1 Beweise und Beweisregeln

⋆ Beweis: Herleitung einer logischen Formel (Konklusion) aus einer Menge von logischen Formeln $\{A_1, ...A_N\}$ (Prämissen) mittels einer Reihe logischer Umformungen und Schlussfolgerungen.

Notation: $\frac{A_1,...,A_N}{B}$

⋆ Beweisregeln (Auswahl)

- Modus ponens: $\frac{A, A\to B}{B}$
- Logische Schlusskette: $\frac{A\to B, B\to C}{A\to C}$
- Kontrapositionsregel: $\frac{\neg B\to\neg A}{A\to B}$
- Widerspruchsregel: $\frac{\neg B, \neg A\to B}{A}$

Beispiel: "Wenn a^2 gerade ist, ist a gerade"

Beweis mittels Kontraposition: $A \equiv$ "a^2 gerade", $B \equiv$ "a gerade"

$\neg B \equiv$ "a ungerade" $\leftrightarrow \exists k \in \mathbb{N}_0 : a = 2k + 1$

$\neg B \Rightarrow a^2 = (2k + 1)^2 = 2k' + 1$ mit $k' = 2k^2 + 2k \in \mathbb{N}_0$

$\Rightarrow \exists k' \in \mathbb{N}_0 : a^2 = 2k' + 1 \leftrightarrow$ "a^2 ungerade" $\equiv \neg A$

Also haben wir gezeigt: $\neg B \to \neg A$ und damit $A \to B$

⋆ Formalisierung von Beweisen ist möglich über Beweissysteme

$\rightsquigarrow$ Entwicklung geeigneter Regelsysteme oder "Kalküle"

Beispiele: Hilbertkalkül, Resolutionskalkül, Tableaukalkül

$\rightsquigarrow$ Beweise werden programmierbar!

Diese Kalküls sind hier nicht Gegenstand der Vorlesung. Eine Einführung findet sich in dem Buch von Hoffmann. Im Folgenden sollen nur einige wichtige, in der Vorlesung benutzte, Beweistechniken vorgestellt werden.

Speziell: Drei wichtige Klassen von Beweisen

– Direkte Deduktion (klar)

– Widerspruchsbeweis (→ 1.3.2)

– Induktion (→ 1.3.3)

1.3.2 Widerspruchsbeweis

⋆ Idee: "Reductio ad absurdum"

Um zu zeigen, dass A falsch sein muss:

– Nimm an, A sei wahr

– Folgere daraus eine Kontradiktion

$\Rightarrow$ Also muss A falsch sein!

⋆ Beispiele

1) $A \equiv$ "*Der Barbier von Sevilla rasiert genau die Menschen in Sevilla, die sich nicht selbst rasieren!*"

| Teste Aussage $B \equiv$ "*Der Barbier rasiert sich selbst.*"
| $A \Rightarrow (B \to \neg B) \wedge (\neg B \to B) \Leftrightarrow \neg B \wedge B = \varnothing$ (Kontradiktion)

Also muss A falsch sein. (Quizfrage: Was ist $\neg A$?)

2) Beweis, dass $\sqrt{2}$ irrational ist ($\sqrt{2} \notin \mathbb{Q}$)

| Angenommen, $\sqrt{2}$ sei rational
| $\Rightarrow \exists\, p \in \mathbb{Z}, q \in \mathbb{N}$ mit $\sqrt{2} = p/q$
| $\Rightarrow \sqrt{2q} = p \;\Rightarrow\; 2q^2 = p^2$
| Wende Satz der eindeutigen Primfaktorzerlegung an auf $2q^2 = p^2$
| Linke Seite: Ungerade Zahl Primfaktoren
| Rechte Seite: Gerade Zahl Primfaktoren → Widerspruch!

⋆ Umkehrung: Verfahren, um zu zeigen, dass A wahr ist

- Nimm an, dass A falsch ist.
- Folgere daraus eine Kontradiktion.

$\Rightarrow$ Also muss A wahr sein.

NB: In der klassischen Logik ist diese Beweisführung äquivalent zur "Reductio ad absurdum".
In der intuitistischen Logik wird sie nicht akzeptiert, da "nicht falsch" nicht notwendig gleichbedeutend ist mit "wahr".

1.3.3 Induktive Beweise

1.3.3.1 Vollständige Induktion

⋆ Hintergrund: Peano Axiom V der natürlichen Zahlen (1.1.2.1). "*Wenn man, beginnend von der Zahl 1, wiederholt 1 addiert, erhält man die Menge* $\mathbb{N}$ *der natürlichen Zahlen*"

→ Beweismethode zum Beweis einer Eigenschaft $P(n)$ für alle $n \in \mathbb{N}$

- Induktionsanfang: Beweise $P(1)$ (oder $P(0)$)
- Induktionsannahme: Nimm an, $P(n)$ sei wahr.
- Induktionsschritt: Beweise $P(n) \to P(n+1)$

⋆ Beispiele:

1) Zeige: Für endliche Mengen M ($|M| < \infty$) ist $|2^M| = 2^{|M|}$

| –Induktionsanfang: $|M| = 1 \;\Rightarrow\; 2^M = \{\varnothing, M\},\ |2^M| = 2$ ✓
| –Induktionsannahme: $|2^{M'}| = 2^n$ für $|M'| = n$
| –Induktionsschritt: Sei M mit $|M| = n+1$
| Betrachte $M' \subset M$ mit $|M'| = n,\ M \backslash M' = \{x\}$,
| Dann gilt: $2^M = 2^{M'} \cup \Omega'$
| mit $\Omega' = \left\{A \cup \{x\} : A \in 2^{M'}\right\},\ \Omega' \cap 2^{M'} = \varnothing,\ |\Omega'| = |2^{M'}|$
| $\Rightarrow\ |2^M| = |2^{M'}| + |2^{M'}| = 2^n + 2^n = 2^{n+1}$ ✓

2) Ein offensichtlich falscher "Beweis": Wir zeigen folgenden Satz:
"*Wenn sich in einer Gruppe M von n Menschen eine Frau befindet, dann sind die Menschen alle Frauen*"

–Induktionsanfang: $n = 1$: Trivial ✓
–Induktionsannahme: Die Aussage gilt für Mengen $|M'| = n$
–Induktionsschritt: Betrachte eine Menge M von $n+1$ Menschen,
darunter mindestens eine Frau
Stelle alle Menschen in einer Reihe auf, die Frau zuerst.
Betrachte die Menge M' der ersten n Menschen in dieser Reihe.
$\rightsquigarrow$ laut Induktionsannahme sind alle weiblich.
Betrachte nun die Menge M'' der letzten n Menschen.
Nach obigem Argument sind die ersten $n-1$ weiblich.
$\rightsquigarrow$ Laut Induktionsannahme sind auch in M'' alle weiblich.
$\Rightarrow$ Also sind alle Menschen in M weiblich !

Frage: Wo liegt der Denkfehler? (wird nicht verraten!)

1.3.3.2 Varianten der vollständigen Induktion

⋆ Verallgemeinerte Induktion

- Induktionsanfang: $P(1)$
- Induktionsannahme: $P(j)$ gilt für alle $j \leqslant n$
- Induktionsschritt: Beweise $P(1) \wedge ...P(n) = \bigwedge_{i=1}^{n} P(i) \rightarrow P(n+1)$

Beispiel: Beweise "Jedes $n \in \mathbb{N}$, $n \geqslant 2$, ist ein Produkt von Primzahlen."

–Induktionsanfang: $n = 2$ ist eine Primzahl ✓
–Induktionsannahme: Die Aussage gilt für alle j, $j \leqslant n$
–Induktionsschritt: Sei $(n+1) \geqslant 3$ beliebig.
Fallunterscheidung:
– $(n+1)$ ist eine Primzahl ✓
– $(n+1)$ ist keine Primzahl
$\Rightarrow\ n+1 = n_1 \cdot n_2$ mit $n_i \leqslant n \wedge n_i \geqslant 2$
$\rightsquigarrow$ Induktionsannahme: n_i ist ein Produkt von Primzahlen
$\Rightarrow (n+1)$ ist ein Produkt von Primzahlen ✓

⋆ Prinzip des kleinsten Verbrechers

- Induktionsanfang $P(1)$
- Induktionsschritt: Widerspruchsbeweis
  Nimm an, $P(n)$ sei falsch für ein $n \in \mathbb{N}$
  → Dann müsste es ein kleinstes $n \in \mathbb{N}$ geben,
  für das $P(n)$ nicht gilt!
  Zeige: Ein solches n kann es nicht geben!

Leicht problematisches Beispiel: Zeige "*Alle Zahlen sind interessant*"

–Induktionsanfang: $n = 1$ ist klarerweise interessant
(z.B. weil $1 \cdot x = x \; \forall x$) ✓

–Induktionsschritt: Nimm an, n sei nicht nicht interessant.
Dann müsste es ein kleinstes $n_0 \in \mathbb{N}$ geben,
das gerade eben nicht interessant wäre.
Aber: Genau deshalb wäre n_0 doch interessant!

Frage: Wo liegt der Denkfehler?
Meine Antwort: "*Alle Zahlen sind interessant*" ist keine Aussage!
(Alternative: Kein Denkfehler, Zahlen sind nun mal interessant!)

1.3.3.3 Strukturelle Induktion

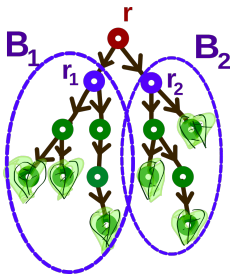
Idee: Ähnlich wie vollständige Induktion, aber diesmal angewendet auf beliebige rekursiv definierte Strukturen, z.B. gewurzelte Bäume!

Beispiel: Beweise folgenden Satz
"Sei B ein Binärbaum mit mindestens 2^k Blättern. Dann gibt es in B einen einfachen Pfad, startend von der Wurzel, mit der Länge $\geqslant k$. "

– Induktionsanfang: $k = 0$ – Bäume mit mehr als $2^0 = 1$ Blättern haben Pfade der Länge $\geqslant 0$: Trivial ✓

– Induktionsannahme:
Die Aussage gelte für Bäume mit mindestens 2^k Blättern.

– Induktionsschritt: Sei nun ein Baum mit $\geqslant 2^{k+1}$ Blättern



Konstruiere, von der Wurzel ausgehend,
den linken und rechten Teilbaum B_1, B_2.

Betrachte den Teilbaum B_j mit der größeren Anzahl Blätter.

$\rightsquigarrow$ B_j enthält mindestens $2^{k+1}/2 = 2^k$ Blätter.

$\rightsquigarrow$ Die Induktionsannahme kann angewendet werden.

$\rightsquigarrow$ B_j hat einen Pfad der Länge $\geqslant k$, der von seiner Wurzel startet.

Führe diesen Pfad weiter bis zur Wurzel von B
$\rightarrow$ Pfadlänge $\geqslant (k+1)$ ✓

Kapitel 2

Formale Sprachen

2.1 Grammatiken

2.1.1 Motivation

2.1.1.1 Sprache und Lyrik

Wir beginnen mit einigen Textbeispielen (zur Informatik kommen wir später)

(a) Babysprache

"viele Buchen" (viele Bücher)
"Jacke aus-e-zieht" (Jacke ausgezogen)
"Anna rutschen gewest" (Anna war rutschen)

(Warum machen die Kinder genau diese "Fehler"?)

(b) Lewis Carroll (*Jabberwocky*)

Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe...

(Were the toves gyring or moming?
und: Woher wissen wir das eigentlich?)

(c) Noam Chomsky

Colorless green ideas sleep furiously.
Furiously sleep ideas green colorless.

(Welcher dieser Sätze macht mehr "Sinn"?)

2.1.1.2 Compilerbau

Gegeben ein Quellprogramm. Dann hat der Compiler folgende Aufgaben:

1) Lexikalische Analyse (Scanner)
   - Erkennung von Grundsymbolen (<,+,`if`, `then`, ...)
   - Überprüfung von Identifiern (z.B. Variablennamen)
     und Literalen (z.B. Formate von Zahlen)

2) Syntaktische Analyse (Parser)
   - Analyse der Struktur des Programms
   - Überprüfung der syntaktischen Korrektheit
     (z.B.: Sind syntaktische Regeln erfüllt wie
     `if` *boolean* `then` *statements* `fi`

3) Semantische Analyse
   - Vielfältig, fortgeschrittenste Stufe
     (z.B. Erstellung Definitionsmodul
     Überprüfen von Typverträglichkeiten
     Identifizierung von Operatoren etc.)

4) Code-Erzeugung und Optimierung

⇒ 1)-3): Verschiedene Stufen der Sprachanalyse
und Tests auf korrekte Sprachstruktur

1) ⇝ Einfache Wortanalyse ↔ "Reguläre Grammatik"

2) ⇝ Einfache Syntaxanalyse ↔ "Kontextfreie Grammatik"

3) ⇝ Kontextsensitive Analyse ↔ "Kontextsensitive Grammatik"

⇒ Klassifizierung von Grammatiken basierend auf Konzepten aus der Linguistik, vor allem basierend auf Noam Chomsky (geboren 1928).

⇒ Gegenstand dieses Kapitels

2.1.1.3 Sprachphilosophie

Rudolf Carnap (1891–1970)

1934: "Logische Syntax der Sprache"

Wichtiges Anliegen: Wendet sich gegen bestimmte Philosophierichtungen der Metaphysik, vor allem Martin Heidegger, in denen seiner Ansicht nach zu viel Wind um Scheinprobleme gemacht wird.

Zum Beispiel sollte der zentrale Satz von Heidegger – *"Das Nichts nichtet"* – laut Carnap grammatikalisch verboten sein, da er sprachliche Strukturelemente unzulässig vermischt. Das Wort "nichts" sollte nicht zum Subjekt oder Prädikat gemacht werden dürfen.

(Aufsatz 1932: "Überwindung der Metaphysik durch logische Analyse der Sprache)

Ansatz: In einer idealen Sprache sollte es überhaupt nicht möglich sein, Gedanken zu formulieren, die sinnlos oder unlogisch sind.

Später wurde diese Auffassung zurückgedrängt zugunsten der Idee, dass Sprache lebt und sich weiterentwickelt.
(Wittgenstein Spätwerk: "Philosophische Untersuchungen")

Allerdings wäre die Forderung von Carnap an die "logische Sprache" eine durchaus sinnvolle Forderung an Programmiersprachen: Ist es möglich, eine Grammatik so zu konzipieren, dass ein grammatikalisch korrekter Programmiercode automatisch fehlerfrei ist?

Antwort: Im Prinzip geht das. Das Problem ist nur, dass eine solche Sprache nicht sehr ausdrucksstark wäre. Sie kann nie alle potentiell sinnvollen Sätze enthalten. Auf Programmiersprachen bezogen heißt das: Sie kann nicht alle potentiellen sinnvollen Programmiercodes enthalten.

$\rightsquigarrow$ Mehr dazu in Kapitel 3: Berechenbarkeit

2.1.2 Elementare Begriffe: Worte und Sprachen

2.1.2.1 Alphabete und Worte

⋆ Definitionen

- Alphabet Σ: Eine endliche Menge von Zeichen σ
- Wort über Σ: Eine endliche Folge $(\sigma_1, ...\sigma_n)$ von Zeichen $\sigma_i \in \Sigma$
  Speziell leeres Wort ε: Kein Zeichen (auch erlaubt)
- Σ^n: Menge aller Worte der Länge n
- Σ^+: Menge aller nichtleeren Worte $\Sigma^+ = \bigcup_{n=1}^{\infty} \Sigma^n$
- Σ^*: Menge aller Worte $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$ (mit $\Sigma^0 = \{\varepsilon\}$)

⋆ Operationen

- Länge eines Wortes: Für $w \in \Sigma^n$ ist $|w| = n$
- Konkatenation zweier Worte: Für $u = (u_1...u_n)$ und $v = (v_1...v_m)$ ist $u \circ v = (u_1...u_n v_1...v_m) \equiv uv$

  (Anmerkung: Die Konkatenation definiert eine Halbgruppe in Σ^*
  da mit $u, v \in \Sigma^*$ auch $uv \in \Sigma^*$, $u(vw) = (uv)w$, $u\varepsilon = \varepsilon u = u$)

2.1.2.2 Formale Sprachen

⋆ Definition Formale Sprache L: Teilmenge von Σ^* ($L \subseteq \Sigma^*$)

⋆ Operationen mit formalen Sprachen

- Übliche Mengenoperationen: ($\cup$, $\cap$, Komplement $\overline{L} = \Sigma^* \backslash L$)
- Konkatenation: $L_1 \circ L_2 = \{u \circ v : u \in L_1, v \in L_2\}$
- Potenzen: $L^0 = \{\varepsilon\}$, $L^n = L \circ L^{n-1} = \underbrace{L \circ L \circ ...L}_{n \text{ Mal}}$
- Kleenesche Hülle: $L^* = \bigcup_{n=0}^{\infty} L^n$
- Analog $L^+ = \bigcup_{n=1}^{\infty} L^n$

⋆ Beispiele für formale Sprachen

- Binärdarstellungen von Zahlen $n \in \mathbb{N}_0$: $\Sigma = \{0,1\}$, $L = \{0,1\}^*$
  (führende Nullen erlaubt)
- Binärdarstellungen von Zahlen $n \in \mathbb{N}$: $\Sigma = \{0,1\}$, $L = \{1\} \circ \{0,1\}^*$
  (führende Nullen nicht erlaubt)
- Binärdarstellungen von Zahlen $n \in \mathbb{N}$: $\Sigma = \{0,1\}$, $L = \{0\}^* \circ \{1\} \circ \{0,1\}^*$
  (führende Nullen erlaubt)
- Uhrzeiten: $\Sigma = \{0,1,..,9,:\}$
  $L = \Big(\underbrace{\{2\} \circ \{0,..4\}}_{20-24} \cup \underbrace{(\{\varepsilon\} \cup \{1\}) \circ \{0,...9\}}_{0-19} \Big) \circ \{:\} \circ \underbrace{\{0,...5\} \circ \{0,...9\}}_{00-59}$
- Dyck-Sprache D_2: Menge aller korrekt geklammerten Ausdrücke
  (z.B. "()", "([]([]))")
  $\Sigma = \{(,),[,]\}$, $L =$? (Wie formalisiert man diese Sprache?)
- Vereinfachte Version: "$a^n b^n$-Sprache": $\Sigma = \{a,b\}; L = \{a^n b^n : n \in \mathbb{N}\}$
- Java: $\Sigma =$ { nötigen Zeichen }, $L =$ funktionierende Quellcodes
  NB: Ein "Wort" ist hier ein ganzer Java-Quellcode!
- "Deutsch": $\Sigma =$ { Buchstaben, Umlaute, Satzzeichen },
  $L =$ grammatikalisch korrekte deutsche Texte
  NB: Ein "Wort" ist hier ein ganzer Text.

2.1.2.3 Typische Fragestellungen in Formalen Sprachen

⋆ Zentral in dieser Vorlesung

a) Wortproblem: Gegeben $w \in \Sigma^*$ und eine Sprache L: Ist $w \in L$?
   (also z.B.: Sollte ein Compiler einen gegebenen Code akzeptieren?)
   Lässt sich dieses Problem effizient lösen? (in polynomialer Zeit)?

b) Konstruktion: Gibt es für eine Sprache L eine Beschreibung aus der sich alle Worte systematisch ableiten lassen?

⋆ Weitere wichtige strukturelle Fragen

- Leerheit: Enthält die Sprache L ein Element? (d.h., ist $L \neq \varnothing$)?
- Endlichkeit: Ist $|L|$ endlich?
- Äquivalenz: Sind zwei Sprachen L_1, L_2 identisch?

Im Folgenden werden wir das Konzept der Grammatiken einführen.
$\rightsquigarrow$ Lösungsvorschlag für Problem b)
Formalisiert einige der Gedanken aus 2.1.1

2.1.3 Grammatiken und Chomsky-Hierarchie

"Und wir beeilten uns, den Jungen zu erzählen, wir hätten von Anfang an gewusst, daß es nur eine Erfindung von Lasse gewesen sei. Und da sagte Lasse, die Jungen hätten gewusst, daß wir gewusst hätten, es sei nur eine Erfindung von ihm. Das war natürlich gelogen, aber vorsichtshalber sagten wir, wir hätten gewusst, die Jungen hätten gewusst, dass wir gewusst hätten, es sei nur eine Erfindung von Lasse. Und da sagten die Jungen - ja - jetzt schaffe ich es nicht mehr aufzuzählen, aber es waren so viele gewusst, dass man ganz verwirrt davon werden konnte, wenn man es hörte. "

(Astrid Lindgren, Wir Kinder aus Bullerbü)

Idee einer Grammatik: Konstruktionsvorschrift für Sprachen durch Einführung von Ersetzungsregeln: Simpel, aber ungeheuer mächtig!

2.1.3.1 Definitionen

⋆ Eine Grammatik ist ein Viertupel $G = (V, \Sigma, P, S)$, bestehend aus

- V: Einer endlichen Menge von "Variablen"
- Σ: Einem endlichen Alphabet (sogenannte "Terminale"), wobei $V \cap \Sigma = \varnothing$
- P: Einer endlichen Menge von Produktionen (Regeln), der Form $l \to r$ mit $l \in \Omega^* V \Omega^*$, $r \in \Omega^*$ und $\Omega = V \cup \Sigma$ $(P \subseteq \Omega^* V \Omega^* \times \Omega^*)$
- S: Einem Startsymbol $S \in V$

Notation: Die Regel $\{V \to w_1 | w_2\}$ steht für $\{V \to w_1, V \to w_2\}$

⋆ Ableitung: beschreibt die Anwendung einer Grammatik

z.B.

x	l	y	→	x	r	y
Der	⟨*Adj.*⟩	Hund ⟨*Prädikat*⟩	→	Der	blöde	Hund ⟨*Prädikat*⟩

Gegeben eine Grammatik $G = (V, \Sigma, P, S)$ und $u, v \in (\Sigma \cup V)^*$.
Dann bedeutet

- $u \Rightarrow_G v$ oder $u \Rightarrow v$: Es gibt $x, y \in (V \cup \Sigma)^*$ und $(l \to r) \in P$ mit $u = xly$, $v = xry$
  NB: "⇒" definiert eine Relation.
- $u \Rightarrow^*_G v$ bzw. $u \Rightarrow^* v$: Reflexiv-transitive Hülle (siehe 1.1.1.3)
  (d.h. $u \Rightarrow^* u$ und $u \Rightarrow^* v \leftrightarrow \exists w_1, ...w_k \in (V \cup \Sigma)^* : u \Rightarrow w_1 \Rightarrow ... \Rightarrow w_k \Rightarrow v$).

⋆ Sprache einer Grammatik

Gegeben eine Grammatik $G = (V, \Sigma, P, S)$
Dann ist $\mathcal{L}(G) = \{w \in \Sigma^* : S \Rightarrow^* w\}$ die von G erzeugte Sprache

NB: Die Worte in $\mathcal{L}(G)$ enthalten keine Variablen mehr, nur Zeichen des Alphabets Σ (Terminale): $\mathcal{L}(G)$ ist die Gesamtheit aller solchen Worte, die sich aus dem Startsymbol S sukzessive herleiten lassen.

⋆ Äquivalenz zweier Grammatiken

Zwei Grammatiken heißen äquivalent, wenn sie dieselbe Sprache erzeugen.

2.1.3.2 Klassifizierung und Chomsky-Hierarchie

In 2.1.3.1: Konstruktionsverfahren für Sprachen über Grammatiken

In 2.1.1: Offensichtlich gibt es verschiedene Stufen von Grammatiken

Nun: Formalisiert über Chomsky-Hierarchie (Noam Chomsky)

$\rightsquigarrow$ Klassifizierung über erlaubte Produktionen $l \to r$ der Grammatik

⋆ Definitionen: Betrachte eine Grammatik $G = (V, \Sigma, P, S)$. G heißt vom

> Typ 0: Keine Einschränkungen an $(l \to r) \in P$
>
> Typ 1 (kontextsensitiv): $|l| \leqslant |r|$ mit "ε-Sonderregel" (siehe unten)
>
> Typ 2 (kontextfrei): $l \in V$
>
> Typ 3 (regulär): $l \in V$ und $r \in \Big(\{\varepsilon\} \cup \Sigma \cup (\Sigma V)\Big)$
>
> (d.h., erlaubt ist $l \to \varepsilon$, $l \to \sigma$, $l \to \sigma B$ mit $\sigma \in \Sigma, B \in V$)

Eine formale Sprache L heißt Sprache vom Typ n, wenn es eine Typ-n Grammatik G gibt, die L erzeugt ($L = \mathcal{L}(G)$)

⋆ Ergänzungen

a) ε-Sonderregel: Wird eingeführt, damit auch Typ-1 Grammatiken das leere Wort ε erzeugen können
Zusätzlich erlaubt ist evtl. eine Regel $S \to \varepsilon$.
Aber: Wenn $(S \to \varepsilon) \in P$, darf S in $(l \to r) \in P$ nie Teil von r sein!

b) ε-Freiheit: Typ-2 und Typ-3 Grammatiken heißen ε-frei, wenn sie die ε-Sonderregel erfüllen und abgesehen von (evtl.) $S \to \varepsilon$ keine Regeln $l \to \varepsilon$ enthalten. Aus jeder Typ-2/Typ-3 Grammatik lässt sich eine äquivalente ε-freie Grammatik des gleichen Typs konstruieren.

(Beweis über Konstruktionsvorschrift:
Gegeben sei eine reguläre oder kontextfreie Grammatik $G = (V, \Sigma, P, S)$
Konstruiere äquivalente ε-freie Grammatik $G' = (V', \Sigma, P', S')$ wie folgt:
(1) Führe neues Startsymbol $S' \neq V \cup \Sigma$ ein. Ergänze V zu $V' = V \cup \{S'\}$ und P zu P' so, dass in P' für jede Regel $(S \to r) \in P$ auch eine Regel $(S' \to r)$ hinzugefügt wird
(2) Bestimme $V_\varepsilon = \{A \in V' : A \overset{*}{\underset{G}{\Rightarrow}} \varepsilon\}$ über Markierungsalgorithmus
(2-i) Markiere alle $A \in V'$ mit $(A \to \varepsilon) \in P'$
(2-ii) Markiere alle $A \in V'$, für die gilt:
$\exists$ Regel $A \to B_1 ... B_n$, in der alle B_i markiert sind.
Wiederhole (2-ii), bis es nichts mehr zu markieren gibt.
(2-iii) $V_\varepsilon = \{A \in V' : A$ markiert $\}$
(3) Bestimme alle Regeln $A \to xBy$ mit $B \in V_\varepsilon$, $(A \to xy) \notin P'$, $x, y \in (V \cup \Sigma)^*$, $|xy| \geqslant 1$ und füge zu P' die Regel $(A \to xy)$ hinzu. Wiederhole (3), bis es keine solchen Regeln mehr gibt.
(4) Entferne alle Regeln $A \to \varepsilon$ aus P'
(5) Falls $S' \in V_\varepsilon$, füge $S' \to \varepsilon$ zu P' hinzu.
Bemerkung: Der Typ der Grammatik (Typ-2, Typ-3) ändert sich nicht!)

Folgerung: Typ-n Sprachen schließen Typ-(n+1) Sprachen ein für n=0,1,2

⋆ Beispiele

- "a^n"-Sprache $L = \{a^n : n \in \mathbb{N}_0\}$ (z.B. $a, aa, aaa, ...$)
  regulär: $\Sigma = \{a\},\ V = \{S, A\},\ P = \{S \to \varepsilon|a|aA,\ A \to a|aA\}$
- "$a^n b^n$"-Sprache $L = \{a^n b^n : n \in \mathbb{N}_0\}$ (z.B. $ab, aabb, aaabbb, ...$)
  kontextfrei: $\Sigma = \{a, b\},\ V = \{S, X\},\ P = \{S \to \varepsilon|X,\ X \to ab|aXb\}$
  aber nicht regulär. Beweis siehe 2.2
- "$a^n b^n c^n$"-Sprache $L = \{a^n b^n c^n : n \in \mathbb{N}_0\}$ (z.B. $abc, aabbcc, ...$)
  kontextsensitiv: $\Sigma = \{a, b, c\}, ; V = \{S, X, B, C\},$
  $P = \{S \to \varepsilon|X,\ X \to aBC|aXBC,\ CB \to BC,$
  $aB \to ab,\ bB \to bb,\ bC \to bc,\ cC \to cc\}$
  (Probieren Sie es aus: Es klappt!)
  aber nicht kontextfrei. Beweis siehe 2.3

2.1.3.3 Abschlusseigenschaften

Satz: Die Sprachen L_1, L_2 seien vom Typ i $(i = 0, ..., 3)$. Dann gilt:

(i) $L_1 \cup L_2$ ist vom Typ i
(ii) $L_1 \circ L_2$ ist vom Typ i
(iii) L_i^* ist vom Typ i

Beweis für Grammatiken vom Typ $i \leqslant 2$:

Wir verknüpfung die zugehörigen Grammatiken. $G_i = (V_i, \Sigma, P_i, S_i)$ mit Hilfe eines neuen Symbols $S \notin V_i\ \forall V_i$

(a) Vereinigung: $G_1 \uplus G_2 = (V, \Sigma, P, S)$
  mit $V = V_1 \cup V_2 \cup \{S\},\ P = P_1 \cup P_2 \cup \{S \to S_1|S_2\}$
  Es gilt: • $\mathcal{L}(G_1 \uplus G_2) = \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$
  • Falls G_1, G_2 vom Typ $i \leqslant 2$ ist, dann auch $G_1 \uplus G_2$

(b) Konkatenation: $G_1 \circ G_2 = (V, \Sigma, P, S)$
  mit $V = V_1 \cup V_2 \cup \{S\},\ P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$
  Es gilt: • $\mathcal{L}(G_1 \circ G_2) = \mathcal{L}(G_1) \circ \mathcal{L}(G_2)$
  • Falls G_1, G_2 vom Typ $i \leqslant 2$ ist, dann auch $G_1 \circ G_2$.

(c) Kleene-Abschluss: $G_i^* = (V_i', \Sigma, P_i', S_i')$
  mit $V_i' = V_i \cup \{S_i'\},\ P_i' = P_i \cup \{S \to \varepsilon|S_i S\}$
  Es gilt: • $\mathcal{L}(G_i^*) = \mathcal{L}(G_i)^*$
  • Falls G_i vom Typ 0,2 ist, dann auch G_i^*.
  (NB: Letzteres gilt nicht für Typ 1 wegen der ε-Sonderregel, aber man kann leicht eine äquivalente ε-freie Grammatik konstruieren, siehe oben.)

Für Grammatiken vom Typ 3 ist der Beweis über Grammatiken etwas aufwändiger (Übungsaufgabe). Einen alternativen Beweis (über Automaten) werden wir in Abschnitt 2.2.2.1 zeigen. Grammatiken vom Typ 3 sind auch unter Schnittbildung und Komplementbildung abgeschlossen (siehe 2.2.2.1).

2.1.4 Automaten

Zum Abschluss noch ein paar Worte zu Automaten, die im Folgenden auch eine wichtige Rolle spielen werden.

Grammatiken sind Regeln zur Erzeugung von Worten, i.e., Sprache

Automaten sind Sprachakzeptoren

- Theoretische Modelle für digitale zeitdiskrete Rechner
  (besonders prominent: Turingmaschinen)
  $\rightsquigarrow$ nützlich für Beweisführungen in der theoretischen Informatik
- Praktische Bedeutung: Compilerdesign (Validierung von Eingaben), Steuerungshardware

Konkret werden wir den Kategorien von Sprachen verschieden Kategorien von Automaten zuordnen können

Typ 3	$\leftrightarrow$	endliche Automaten (Kapitel 2.2)
Typ 2	$\leftrightarrow$	Kellerautomaten (Kapitel 2.3)
Typ 1	$\leftrightarrow$	linear beschränkte Automaten (lassen wir hier weg)
Typ 0	$\leftrightarrow$	z.B. Turingmaschinen oder Registermaschinen

2.1.5 Zusammenfassung

Wichtigste Ergebnisse dieses Teilkapitels

- Zentrales Vokabular: Alphabete, Worte, Sprachen
- Konzept der Grammatiken
- Klassifizierung, Chomsky-Hierarchie

Im Folgenden (Kapitel 2.2 und 2.3): Diskussion von Typ-3/Typ-2 Sprachen

Zentrale Fragestellungen

- Wortproblem und: Kann man es effizient lösen?
- Eigenschaften von Typ-n Sprachen
  Abschlusseigenschaften: Seien L_1, L_2 Sprachen vom Typ n:
  Wie verhält es sich dann mit $L_1 \cap L_2$, $L_1 \cup L_2$, $L_1 \circ L_2$?
  Wie kann man den Typ einer Sprache testen? ($\rightarrow$ Pumping Lemmata)
- Was ist der zugehörige Automat?

2.2 Reguläre Sprachen

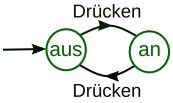
2.2.1 Endliche Automaten

Kurzgefasst: Endliche Automaten sind einfache Rechnermodelle ohne Speicher, in denen nichts überschrieben wird.
Die Eingabe wird von links nach rechts gelesen.

(Vergleiche Turingmaschinen (Kapitel 3) – überschreibt und liest in beider Richungen. Registermaschinen (Kapitel 3) und Kellerautomaten (Kapitel 2.3) verfügen über einen unendlichen Speicher.)

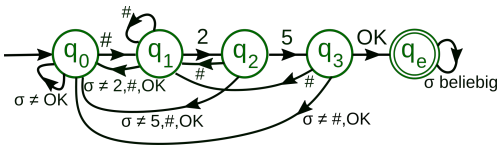
Einfache Beispiele

1) Kippschalter



2) Einfaches Code – Lesegerät

Akzeptiert z.B. den Code `2 5`, nur ein Versuch
Eingabe: $\sigma \in \{$ `0,...9,#,OK` $\}$, Start mit `#`, Stop mit `OK`



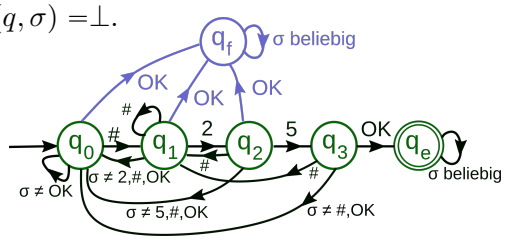
2.2.1.1 Deterministische endliche Automaten (DEAs)

(Notation: Asteroth/Baier)

⋆ Definition: Ein DEA ist ein 5-Tupel $M = (Q, \Sigma, \delta, q_0, F)$ mit

- Q: Endliche Menge von Zuständen (des Automaten)
- Σ: Endliches Alphabet
- $\delta : Q \times \Sigma \to Q$: Übergangsfunktion
- $q_0 \in Q$: Startzustand
- $F \subseteq Q$: Menge der akzeptierenden oder Endzustände

NB: Im Prinzip darf δ auch eine partielle Funktion sein, d.h. $\delta(q, \sigma)$ ist nicht notwendig definiert für alle $q \in Q$, $\sigma \in \Sigma$. Wir werden aber i.A. annehmen, dass δ eine totale Übergangsfunktion ist. Dazu ergänzen wir Q notfalls gedanklich um einen Fangzustand q_F und setzen $\delta(q, \sigma) = q_F$, falls $\delta(q, \sigma) = \perp$.
z.B. Code-Lesegerät



Graphische Darstellung:

Zustände: ○ oder ◎ (akzeptierend)
Übergangsfunktion: ○→○ ; der Fangzustand wird i.A. weggelassen.

⋆ Erweiterte Übergangsfunktion und akzeptierte Sprache

Gegeben sei ein DEA $M = (Q, \Sigma, \delta, q_0, F)$. Eine typische Fragestellung wäre: Befindet sich M nach Eingabe eines Worts wie Σ^* in einem akzeptierenden Zustand? (d.h., akzeptiert M dieses Wort)?

Dazu: Erweitere $\delta : Q \times \Sigma \to Q$ zu $\delta : Q \times \Sigma^* \to Q$
über $\delta(q, \varepsilon) := q$, $\delta(q, \sigma w') := \delta(\delta(q, \sigma), w')$ für alle $\sigma \in \Sigma$, $w' \in \Sigma^*$
$\rightsquigarrow$ M akzeptiert $w \in \Sigma^*$, wenn $\delta(q_0, w) \in F$

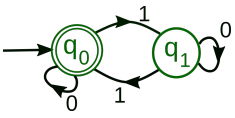
Definition: Die von M akzeptierte (oder erkannte) Sprache $\mathcal{L}(M)$
ist die Menge aller von M akzeptierten Worte.
$\mathcal{L}(M) = \{w \in \Sigma^* : \delta(q_0, w) \in F\}$

NB: Das Wortproblem lässt sich hier effizient lösen in der Zeit $\mathcal{O}(|w|)$

⋆ Beispiele

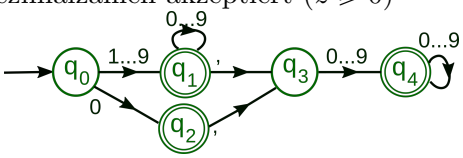
(a) Automat für die Sprache $L = \{w \in \{0,1\}^*$, Anzahl Einser ist gerade$\}$

$\Sigma = \{0,1\}$, M:



(b) Automat, der positive Dezimalzahlen akzeptiert ($z \geqslant 0$)

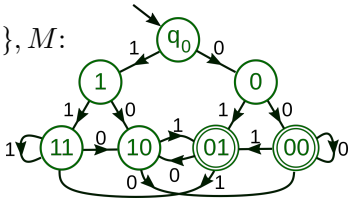
$\Sigma = \{0, ..., 9, ', '\}$, M:



(c) Automat zur Sprache $L = \{w \in \{0,1\}^*$, Vorletztes Zeichen is 0$\}$

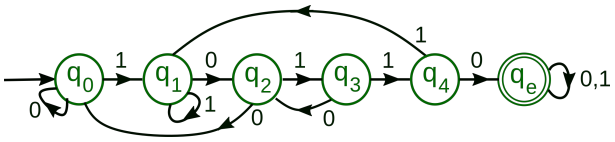
Konstruiere DEA z.B. so, dass er sich die die letzten beiden Zeichen "merkt":
$Q = \{q_0, 0, 1, 00, 01, 10, 11\}$
$\rightsquigarrow$ wird kompliziert!

$\Sigma = \{0,1\}$, M:



(d) Mustererkennung, z.B. $L = \{w \in \{0,1\}^*$, w enhält Folge 10110$\}$

$\Sigma = \{0,1\}$, M:



$\rightsquigarrow$ Die Topologie des konkreten Automaten hängt von der zu erkennenden Zeichenfolge ab. Damit wird das Design sehr fehleranfällig.

$\rightsquigarrow$ Besser wäre es, die Konstruktion könnte automatisiert werden!

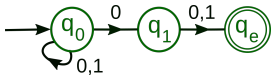
2.2.1.2 Nichtdeterministische endliche Automaten (NEAs)

Problem: Manchmal sind DEAs unnötig kompliziert
(z.B. Beispiele (c,d) in 2.2.1.1)

Ausweg (hilft oft): Lasse auch Übergangsfunktionen zu, die nicht eindeutig sind: $\delta(q,\sigma) = \{q_1, ...q_n\}$: Menge möglicher Zielzustände.
Der Automat akzeptiert ein Wort w, wenn mindestens ein möglicher Lauf in einem akzeptierten Zustand landet.

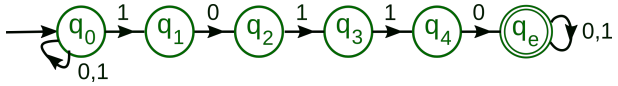
$\rightsquigarrow$ Die Beispiele (c) und (d) aus 2.2.1.1 können einfacher dargestellt werden

(c) $L = \{w \in \{0,1\}^*$, Vorletztes Zeichen is $0\}$ $\triangleq$ M:



(d) $L = \{w \in \{0,1\}^*$, w enhält Folge 10110$\}$:

$\triangleq$ M :



$\rightsquigarrow$ Die Darstellung von Automaten wird sehr viel einfacher, aber es ist noch nicht klar, wie man sie in der Praxis implementiert.

Zum Glück werden wir zeigen können: Aus jedem solchen "nichtdeterministischen" endlichen Automaten lässt sich systematisch ein äquivalenter DEA konstruieren!

⋆ Definitionen

– Ein NEA ist ein 5-Tupel $M = (Q, \Sigma, \delta, Q_0, F)$ mit

- Q: Endliche Menge von Zuständen (analog DEA)
- Σ: Endliches Alphabet (analog DEA)
- $\delta : Q \times \Sigma \to 2^Q$: Übergangsfunktion (in die Potenzmenge 2^Q)
- $Q_0 \subseteq Q$: Menge von Startzuständen
- $F \subseteq Q$: Menge der akzeptierenden Zustände (analog DEA)

– Die erweiterte Übergangsfunktion $\widehat{\delta} : 2^Q \times \Sigma^* \to 2^Q$ ist definiert über
$\widehat{\delta}(P,\varepsilon) = P$, $\widehat{\delta}(P,\sigma) = \bigcup_{p \in P} \delta(p,\sigma)$, $\widehat{\delta}(P,\sigma w') = \widehat{\delta}(\widehat{\delta}(P,\sigma), w')$ $\forall \sigma \in \Sigma$, $w' \in \Sigma^*$

$\rightsquigarrow$ $\widehat{\delta}(P,w)$ ist die Menge aller Zustände, die, ausgehend von Zuständen $p \in P \subseteq Q$, nach Eingabe von w erreicht werden können.

NB: Oft wird dasselbe Symbol (δ) sowohl für die Übergangsfunktion δ als auch die erweiterte Übergangsfunktion $\widehat{\delta}$ verwendet, obwohl das mathematisch nicht ganz sauber ist. (Die Urbildmenge von δ, $Q \times \Sigma$, kann nicht ohne weiteres zu Urbildmenge von $\widehat{\delta}$, $2^Q \times \Sigma$, "erweitert" werden).

– Definition: Die akzeptierte Sprache eines NEA M ist

$\mathcal{L}(M) = \{w \in \Sigma^* : \widehat{\delta}(Q_0, w) \cap F \neq \varnothing\}$

⋆ Äquivalenz von NEAs und DEAs

Satz (Rabin, Scott, 1959):

> Für jeden NEA lässt sich ein DEA konstruieren, der die gleiche Sprache akzeptiert, das heißt, zu jedem NEA M gibt es einen DEA M' mit $\mathcal{L}(M) = \mathcal{L}(M')$.

Konstruktionsvorschrift: über Potenzmenge

Gegeben sei ein NEA $M = (Q, \Sigma, \delta, Q_0, F)$
mit erweiterter Übergangsfunktion $\hat{\delta}$
Konstruiere folgenden DEA: $M' = (Q', \Sigma, \delta', Q_0, F')$
mit Q' : $Q' = 2^Q$,
δ' : $\delta'(P \in 2^Q, \sigma) = \hat{\delta}(P, \sigma)$,
F' : $F' = \{P \in 2^Q : P \cap F \neq \varnothing\}$
Dann gilt $\mathcal{L}(M) = \mathcal{L}(M')$

Beweis, dass die Konstruktion funktioniert:

Zeige per Induktion nach $n = |w|$,
dass gilt: $\delta'(Q_0, w) = \hat{\delta}(Q_0, w)$ für alle $w \in \Sigma^*$

$n = 0$ ($|w| = 0 \Leftrightarrow w = \varepsilon$): $\delta'(Q_0, \varepsilon) = Q_0 = \hat{\delta}(Q_0, \varepsilon)$ ✓

$n = 1$ ($|w| = 1 \Leftrightarrow w = \sigma \in \Sigma$): $\delta'(Q_0, \sigma) = \hat{\delta}(Q_0, \sigma)$ ✓

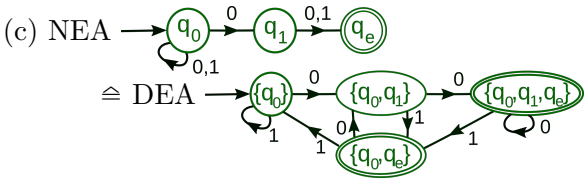
Verwende im Folgenden die Notation $\delta'(Q_0, \sigma) =: \bar{Q}_\sigma$

$n \to n+1$: Nimm an, die Behauptung gelte für alle w mit $|w'| = n$
Betrachte ein Wort w der Länge $|w| = n + 1$.
Zerlege $w = \sigma w'$, $\sigma \in \Sigma, |w'| = n$
$\Rightarrow \delta'(Q_0, \sigma w') = \delta'(\delta'(Q_0, \sigma), w') = \delta'(\bar{Q}_\sigma, w')$
$\hat{\delta}(Q_0, \sigma w') = \hat{\delta}(\hat{\delta}(Q_0, \sigma), w') = \hat{\delta}(\bar{Q}_\sigma, w')$
Laut Induktionsannahme ist $\hat{\delta}(\bar{Q}_\sigma, w') = \delta'(\bar{Q}_\sigma, w')$
$\Rightarrow \delta'(Q_0, \sigma w') = \hat{\delta}(Q_0, \sigma w')$ ✓

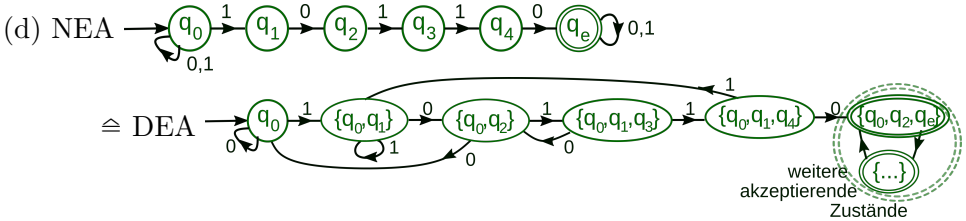
⋆ Beispiele für die Konstruktion äquivalenter DEAs aus NEAs

(Beispiele aus 2.2.1.2)

(c) NEA ≙ DEA



(d) NEA ≙ DEA



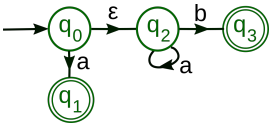
⇝ systematisches, automatisierbares Verfahren!

⋆ Erweiterung: ε-Übergänge

Erweiterung des NEA-Gedankens: Lasse auch spontane Übergänge zwischen Zuständen zu, die ohne Eingabe, einfach so, stattfinden können.

Verwendung: Vor allem in theoretischen Beweisen (z.B. 2.2.2.1)

- Definition: Ein ε-erweiterter NEA ist ein Tupel $M = (Q, \Sigma, \delta, Q_0, F)$
  || mit den Eigenschaften eines NEAs, außer dass die Übergangsfunktion definiert ist auf $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$
- Beispiel: Die Sprache $L = \{a\} \cup \{a^n b : n \in \mathbb{N}_0\}$ wird erkannt von dem folgenden ε-erweiterten NEA:



- Satz: Zu jedem ε-erweiterten NEA M lässt sich ein gewöhnlicher NEA M' konstruieren, der die gleiche Sprache akzeptiert: $\mathcal{L}(M) = \mathcal{L}(M')$

  (Konstruktionsvorschrift:
  - Definiere erst $\Delta(P) = \bigcup_{i=0}^{\infty} \Delta_i(P)$
    mit $\Delta_0(P) = P,\ \Delta_{i+1}(P) = \Delta_i(P) \cup \big(\bigcup_{q \in \Delta_i(P)} \delta(q, \varepsilon) \big)$
    $\triangleq$ Menge der Zustände, die von P mit reinen ε-Übergängen erreicht werden können.
  - Dann ist die erweiterte Übergangsfunktion $\widehat{\delta}$ von M gegeben durch:
    $\widehat{\delta}(P, \varepsilon) = \Delta(P), \quad \widehat{\delta}(P, \sigma) = \bigcup_{q \in \Delta(P)} \Delta\big(\delta(q, \sigma)\big),$
    $\widehat{\delta}(P, \sigma w) = \widehat{\delta}(\widehat{\delta}(P, \sigma), w)$ für alle $\sigma \in \Sigma,\ w \in \Sigma^*$
  - Konstruiere den NEA $M' = (Q, \Sigma, \delta', Q_0', F)$
    mit $Q_0' = \Delta(Q_0)$
    und $\delta'(q, \sigma) = \widehat{\delta}(\{q\}, \sigma)$

  $\rightsquigarrow$ Dann ist per Konstruktion $\widehat{\delta'}(Q_0', w) = \widehat{\delta}(Q_0, w)$ für alle $w \in \Sigma^*$, wobei $\widehat{\delta'}$ die erweiterte Übergangsfunktion zu δ' in M' ist.)

2.2.1.3 Zusammenhang mit regulären Grammatiken

Satz: Sei $L \subseteq \Sigma^*$ eine Sprache. Dann sind folgende Aussagen äquivalent:

|| (I) Es gibt einen DEA M', der L akzeptiert: $L = \mathcal{L}(M')$
|| (II) Es gibt eine reguläre Grammatik G, die L erzeugt: $L = \mathcal{L}(G)$
|| (III) Es gibt einen NEA M, der L akzeptiert: $L = \mathcal{L}(M)$

Beweis: Wir zeigen (I) $\Rightarrow$ (II) $\Rightarrow$ (III); $\big($(III) $\Rightarrow$ (I) wurde in 2.2.1.2 gezeigt.$\big)$

(I) $\Rightarrow$ (II):

Gegeben ein DEA $M' = (Q', \Sigma, \delta', q_0, F')$ mit totaler Übergangsfunktion

- Konstruiere eine Grammatik $G = (V, \Sigma, P, S)$ über
  $V = Q',\ S = q_0,\ P = \{q \to \varepsilon : q \in F'\} \cup \{q \to \sigma q' : \delta'(q, \sigma) = q'\}$
- Zeige: $\mathcal{L}(M') = \mathcal{L}(G)$, das heisst, $(w \in \mathcal{L}(M') \leftrightarrow w \in \mathcal{L}(G))$

  (Vorab: Klar für $w = \varepsilon$, da $\varepsilon \in \mathcal{L}(M') \leftrightarrow q_0 = S \in F' \leftrightarrow \varepsilon \in \mathcal{L}(G)$
  Im folgenden können wir daher $w \neq \varepsilon$ annehmen.

$\underline{\mathcal{L}(M') \subseteq \mathcal{L}(G)}$: Sei $w = \sigma_1 \cdots \sigma_n \in \mathcal{L}(M')$
$\rightsquigarrow$ Die Eingabe w erzeugt eine Zustandsfolge
$q_0 \overset{\sigma_1}{\to} Z_1 \overset{\sigma_2}{\to} Z_2 \overset{\sigma_3}{\to} ... Z_n \in F'$ $(Z_i \in Q')$
$\rightsquigarrow$ Diese entspricht in G einer Ableitung $\Rightarrow$ (bzw. $\Rightarrow_G$)
$S \Rightarrow \sigma_1 Z_1 \Rightarrow \sigma_1\sigma_2 Z_2 ... \Rightarrow \sigma_1\sigma_2 \cdots \sigma_n Z_n \Rightarrow \sigma_1\sigma_2 \cdots \sigma_n = w$
$\rightsquigarrow w \in \mathcal{L}(G)$ ✓
$\underline{\mathcal{L}(G) \subseteq \mathcal{L}(M')}$: Sei $w = \sigma_1 \cdots \sigma_n \in \mathcal{L}(G)$
$\rightsquigarrow$ Dann existiert in G eine Ableitung $\Rightarrow$ (bzw. $\Rightarrow_G$)
$S = q_0 \Rightarrow_G \sigma_1 Z_1 \Rightarrow \sigma_1\sigma_2 Z_2 \Rightarrow ... \Rightarrow \sigma_1 \cdots Z_n \Rightarrow \sigma_1 \cdots \sigma_n$
mit $Z_i \in V = Q'$
$\rightsquigarrow$ Also ist $Z_1 = \delta(q_0, \sigma_1), Z_2 = \delta(Z_1, \sigma_2), ... Z_n = \delta(Z_{n-1}, \sigma_n)$
mit $Z_n \in F'$
$\rightsquigarrow \delta(q_0, w) \in F' \rightsquigarrow w \in \mathcal{L}(M')$ ✓)

$\underline{\text{(II)} \Rightarrow \text{(III)}}$ Gegeben sei eine ε-freie Grammatik $G = (V, \Sigma, P, S)$
(falls G nicht ε-frei ist, können wir eine äquivalente ε-freie Grammatik konstruieren mit der Konstruktionsvorschrift aus 2.1.3.2.)

- $\underline{\text{Konstruiere}}$ einen NEA: $M = (Q, \Sigma, \delta, Q_0, F)$ über
  $Q = V \cup \{q_e\}$ (q_e: Neues Symbol); $Q_0 = \{S\}$
  $F = \left\{ \begin{array}{ll} \{S, q_e\} & \text{falls } (S \to \varepsilon) \in P \\ \{q_e\} & \text{sonst} \end{array} \right\}$;
  $\delta(A, \sigma)$ enthält: $\left\{ \begin{array}{ll} B \in \delta(A, \sigma) & \text{falls } (A \to \sigma B) \in P \\ q_e \in \delta(A, \sigma) & \text{falls } (A \to \sigma) \in P \end{array} \right.$
- $\underline{\text{Zeige}}$ $\mathcal{L}(G) = \mathcal{L}(M)$, das heisst, $(w \in \mathcal{L}(G) \leftrightarrow w \in \mathcal{L}(M))$

  (Vorab: Klar für $w = \varepsilon$, da $\varepsilon \in \mathcal{L}(G) \leftrightarrow \{S\} \in F \leftrightarrow \varepsilon \in \mathcal{L}(M)$
  Im folgenden können wir daher $w \neq \varepsilon$ annehmen.
  $\underline{\mathcal{L}(G) \subseteq \mathcal{L}(M)}$: Sei $w = \sigma_1 \cdots \sigma_n \in \mathcal{L}(G)$
  $\rightsquigarrow$ Dann existiert in G eine Ableitung
  $S =: Z_1 \Rightarrow \sigma_1 Z_2 \Rightarrow \sigma_1\sigma_2 Z_3 \Rightarrow ... \Rightarrow \sigma_1 \cdots Z_n \Rightarrow \sigma_1 \cdots \sigma_n$
  mit $Z_i \in V$, speziell $Z_1 = S$.
  $\rightsquigarrow \exists$ Regeln: $\left\{ \begin{array}{llllll} Z_k & \to \sigma_k Z_{k+1} & \in P & \Leftrightarrow & Z_{k+1} & \in \delta(Z_k, \sigma_k) \\ Z_n & \to \sigma_n & \in P & \Leftrightarrow & q_e & \in \delta(Z_n, \sigma_n) \end{array} \right.$
  $\rightsquigarrow q_e \in \widehat{\delta}(Q_0, w) \rightsquigarrow w \in \mathcal{L}(M)$ ✓
  $\underline{\mathcal{L}(M) \subseteq \mathcal{L}(G)}$: Sei $w = \sigma_1 \cdots \sigma_n \in \mathcal{L}(M)$
  $\rightsquigarrow q_e \in \widehat{\delta}(Q_0 = \{S\}, w)$
  $\rightsquigarrow$ Entweder $|w| = 1$ (d.h. $w = \sigma_1$) und $q_e \in \delta(Q_0, \sigma_1)$
  oder es existieren Zwischenzustände $Z_k \in \delta(Z_{k-1}, \sigma_{k-1})$
  für $1 < k \leqslant n$ (mit $Z_1 := S$)
  $\rightsquigarrow$ Es existieren Regeln: $Z_n \to \sigma_n$ und $Z_{k-1} \to \sigma_{k-1} Z_k$
  $\rightsquigarrow$ Es existiert eine Ableitung:
  $S \Rightarrow \sigma_1 Z_2 \Rightarrow \sigma_1\sigma_2 Z_2 \Rightarrow ... \sigma_1 \cdots \sigma_{n-1} Z_{n-1} \Rightarrow \sigma_1 \cdots \sigma_n$
  $\rightsquigarrow w \in \mathcal{L}(G)$ ✓)

$\underline{\text{Fazit}}$: Typ-3 Sprachen können alternativ über Typ-3 Grammatiken oder über endliche Automaten definiert werden!

2.2.2 Eigenschaften regulärer Sprachen

2.2.2.1 Abschlusseigenschaften

Frage: Kann man reguläre Sprachen kombinieren?

Satz (Antwort): Die Sprachen $L_1, L_2 \subseteq \Sigma^*$ seien regulär. Dann gilt

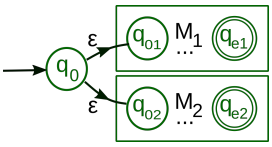
(i) $\overline{L}_i = \Sigma^* \backslash L_i$ ist regulär
(ii) $L_1 \cup L_2$ und $L_1 \cap L_2$ sind regulär
(iii) $L_1 \circ L_2$ ist regulär
(iv) L_i^* ist regulär (Kleene-Abschluss)

Beweis: Da L_i regulär ist, gibt es DEAs $M_i = (Q_i, \Sigma, \delta_i, q_{0i}, F_i)$ mit $L_i = \mathcal{L}(M_i)$
OBdA nehmen wir an, dass M_i eine totale Übergangsfunktion hat (anderenfalls führen wir einen Fangzustand ein).

Für die Beweise konstruieren wir ε-erweiterte NEAs, d.h. wir transformieren die DEAs M_i zunächst formal in NEAs $M_i' = (Q_i, \Sigma, \delta_i', \{q_{0i}\}, F_i)$ mit $\delta_i'(q, \sigma) = \{\delta_i(q, \sigma)\}$

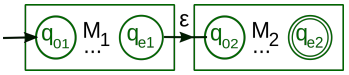
(i) $\overline{M}_i = (Q_i, \Sigma, \delta_i, q_{0i}, \bar{F}_i = Q \backslash F_i)$ akzeptiert $\overline{L}_i$. $\rightsquigarrow \overline{L}_i$ ist regulär.

(ii) Konstruieren einen ε-erweiterten NEA $M_1 \cup M_2$



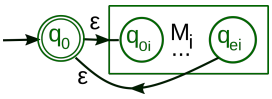
$\rightsquigarrow M_1 \cup M_2$ akzeptiert $L_1 \cup L_2$.
$\rightsquigarrow L_1 \cup L_2$ ist regulär. ✓
Wegen $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ und (i) ist dann auch $L_1 \cap L_2$ regulär. ✓

(iii) Konstruiere einen ε-erweiterten NEA $M_1 \circ M_2$



Verbinde akzeptierende Zustände von M_1 mit Startzuständen von M_2 über ε-Übergänge
$\rightsquigarrow M_1 \circ M_2$ akzeptiert $L_1 \circ L_2$. $\rightsquigarrow L_1 \circ L_2$ ist regulär. ✓

(iv) Konstruiere einen Hüllenautomaten M_i^*. Dazu führen wir einen neuen Startzustand q_0 ein.



Verbinde akzeptierende Zustände von M_i mit q_0, und q_0 mit dem Startzustand q_{0i} von M_i über ε-Übergänge.
$\rightsquigarrow M_i^*$ akzeptiert L_i^*. $\rightsquigarrow L_i^*$ ist regulär. ✓

2.2.2.2 Effizienz

Satz: Seigen M, M_1, M_2 DEAs und $w \in \Sigma^*$. Dann lässt sich das ..

(i) Leerheitsproblem ($\mathcal{L}(M) = \varnothing$?) in der Zeit $\mathcal{O}(|Q||\Sigma|)$ entscheiden.
(ii) Endlichkeitsproblem ($|\mathcal{L}(M)| < \infty$?) in $\mathcal{O}(|Q||\Sigma|)$.
(iii) Wortproblem ($w \in \mathcal{L}(M)$?) in $\mathcal{O}(|w|)$.
(iv) Äquivalenzproblem ($\mathcal{L}(M_1) = \mathcal{L}(M_2)$?) in $\mathcal{O}(|Q_1||Q_2||\Sigma|)$.

Beweis: z.B. Asteroth/Baier, 6.2.2., oder Übungsaufgabe

2.2.2.3 Das Pumping-Lemma für reguläre Sprachen

Frage: Gegeben eine Sprache $L \subseteq \Sigma^*$. Kann ich entscheiden, ob L regulär ist?

Antwort: Man kann zumindest ein notwendiges Kriterium angeben:

Satz: (Pumping Lemma für reguläre Sprachen)

> Sei $L \subseteq \Sigma^*$ eine reguläre Sprache. Dann gibt es ein $n \in \mathbb{N}$, so dass sich alle Worte $w \in L$ mit $|w| \geqslant n$ zerlegen lassen als $w = xyz$ mit $|y| \geqslant 1,\ |xy| \leqslant n,\ xy^k z \in L$ für alle $k \in \mathbb{N}_0$.

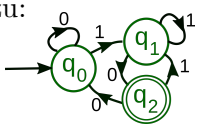
Beweis: Es folgt letztlich daraus, dass der zugehörige DEA endlich ist!

- Wenn L regulär ist, gibt es einen DEA $M = (Q, \Sigma, \delta, q_0, F)$, der L akzeptiert. Nimm an, der sei bekannt. (Ob er wirklich bekannt ist, ist für den Beweis egal!) Setze $n = |Q|$
- Betrachte $w = \sigma_1 \cdot \sigma_s \in L$ mit $s = |w| \geqslant n$
  - $\rightsquigarrow$ generiert bei Eingabe eine Kette von Zustandsänderungen in M
    $$Z_0 = q_0 \overset{\sigma_1}{\to} Z_1 \overset{\sigma_2}{\to} Z_2 \ldots \overset{\sigma_i}{\to} Z_i \overset{\sigma_{i+1}}{\to} \ldots \overset{\sigma_j}{\to} Z_j \overset{\sigma_{j+1}}{\to} \ldots \overset{\sigma_n}{\to} Z_n \to \ldots$$
  - $\rightsquigarrow$ Da $Z_j \in Q$ und es nur n verschiedene Zustände in Q gibt, muss nach Eingabe von maximal n Zeichen ein Z_i doppelt vorkommen.
  - $\rightsquigarrow$ Es existiert ein Paar (i, j) mit $0 \leqslant i < j \leqslant n$ und $Z_i = Z_j$
- Zerlege das Wort w in $x = \sigma_1 \cdot \sigma_i,\ y = \sigma_{i+1} \cdot \sigma_j,\ z = \sigma_{j+1} \cdot \sigma_s$
  $\rightsquigarrow$ $|xy| \leqslant n$ und $|y| \geqslant 1$ per Konstruktion.
  Weiterhin ist $\delta(q_0.x) = Z_i = Z_j = \delta(q_0, xy) = \delta(\delta(q_0, x), y)$
  $\rightsquigarrow$ $\delta(Z_i, y) = Z_i$ $\rightsquigarrow$ y kann beliebig oft eingeschoben werden.
  $\rightsquigarrow$ Mit xyz ist auch $xy^k z \in L$ für alle $k \in \mathbb{N}_0$ ✓

Illustrationsbeispiel: $L = \{0,1\}^* \circ \{1\} \circ \{0\}$. Automat dazu:

Wort $w = 1010 \triangleq$

$q_0 \overset{1}{\to}$	$q_1 \overset{0}{\to} q_2 \overset{1}{\to} q_1$	$\overset{0}{\to} q_2$
x	y	z
1	0 1	0



$\Rightarrow\ |Q| = 3$

Anwendungsbeispiel: "$a^n b^n$"-Sprache aus 2.1.3.2

Frage: Könnte $L = \{a^n b^n : n \in \mathbb{N}_0\}$ regulär sein?

Antwort: Widerspruchsbeweis – Nimm an, L sei regulär.
→ Dann gibt es ein $m \in \mathbb{N}$, so dass das Pumping-Lemma gilt.
Betrachte nun $w = a^m b^m$. Offensichtlich ist $w \in L$
Zerlege $w = xyz$ mit $|xy| \leqslant m$, $|y| \geqslant 1$
$\rightsquigarrow$ xy enthält nur Zeichen $a \rightarrow$ $xy^k z = a^{m+(k-1)|y|} b^m$
$\rightsquigarrow$ $xy^k z \notin L$ für $k \neq 1$ Widerspruch!

$\rightsquigarrow$ Also ist L nicht regulär!

Bemerkung: Die Gültigkeit des Pumping-Lemmas ist ein notwendiges, aber nicht hinreichendes Kriterium dafür dass $L \subseteq \Sigma^*$ regulär ist.
Gegenbeispiel: $L = \{c^m a^n b^n : n, m \in \mathbb{N}\} \cup (\{a\}^* \circ \{b\}^*)$ ist nicht regulär, aber eine Pumping-Zerlegung ist möglich! ($w = xyz$ mit $x = \varepsilon, |y| = 1$)

2.2.3 Reguläre Ausdrücke

Weitere äquivalente Möglichkeit, reguläre Sprachen zu charakterisieren.

2.2.3.1 Definitionen

Gegeben sei ein Alphabet Σ

⋆ Die Menge der regulären Ausdrücke ist rekursiv definiert über

(i) $\varnothing$, ε, $\sigma \in \Sigma$ sind reguläre Ausdrücke.
(ii) Mit α, β sind auch $(\alpha\beta)$, $(\alpha + \beta)$, (α^*) reguläre Ausdrücke.
(iii) Weitere reguläre Ausdrücke gibt es nicht.

⋆ Die Sprache $\mathcal{L}(\gamma)$ zu einem regulären Ausdruck γ ist definiert über

(i) $\mathcal{L}(\varnothing) = \varnothing$, $\mathcal{L}(\varepsilon) = \{\varepsilon\}$, $\mathcal{L}(\sigma) = \{\sigma\}$ für $\sigma \in \Sigma$
(ii) $\mathcal{L}(\alpha\beta) = \mathcal{L}(\alpha) \circ \mathcal{L}(\beta)$, $\mathcal{L}(\alpha + \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$, $\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*$

⋆ Zwei reguläre Ausdrücke heißen äquivalent, $\alpha_1 \equiv \alpha_2$, wenn $\mathcal{L}(\alpha_1) = \mathcal{L}(\alpha_2)$

⋆ Beispiele: $L_1 = \{1\} \circ \{1,0\}^* = \mathcal{L}(\gamma_1)$ mit $\gamma_1 = 1(1+0)^*$
$L_2 = \{w \in \{0,1\}^*$: w enthält $\{1101\}\} = \mathcal{L}(\gamma_2)$ mit $\gamma_2 = (1+0)^*\,1101\,(1+0)^*$
$L_3 = \{w \in \{0,1\}^*$: Anzahl 1er in w ist gerade $\} = \mathcal{L}(\gamma_3)$ mit $\gamma_3 = 0^*(10^*10^*)^*$

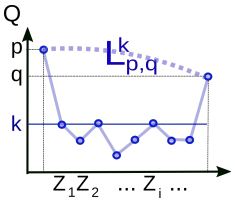
2.2.3.2 Zusammenhang mit regulären Sprachen

Satz: $L \subseteq \Sigma^*$ ist genau dann regulär, wenn es einen regulären Ausdruck γ dazu gibt.

Beweis: Zeige $L = \mathcal{L}(\gamma) \Rightarrow L$ regulär und L regulär $\Rightarrow \big(\exists\gamma$ mit $L = \mathcal{L}(\gamma)\big)$

"$\Leftarrow$" (i) Offensichtlich sind $\varnothing$, $\{\varepsilon\}$, $\{\sigma\}$ mit $\sigma \in \Sigma$ regulär
(ii) Nach 2.2.2.1 sind mit L_1, L_2 auch $L_1 \circ L_2$, $L_1 \cup L_2$, L_i^* regulär.
Das sind genau die Sprachen, die durch reguläre Ausdrücke mit den Basisbausteinen aus (i) erzeugt werden können. ✓

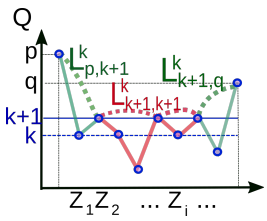
"$\Rightarrow$" L sei regulär, d.h. $L = \mathcal{L}(M)$ für einen DEA $M = (Q, \Sigma, \delta_0, q_0, F)$.



○ Bezeichne die Zustände mit Nummern: $Q = \{1,\cdot,n\}$ (mit $q_0 = 1$)
○ Betrachte die Sprache $L_{pq} := \{w \in \Sigma^* : \delta(p,w) = q\}$
Übergänge $p \to q$ haben Zwischenzustände $p \to Z_1 \to Z_2 \ldots \to q$
Definiere $L_{pq}^k = \{w \in L_{pq}$: Zwischenzustände $Z_i \in \{1,\cdot,k\}\}$
○ Zeige per Induktion: Es gibt einen regulären Ausdruck zu $L_{p,q}^k$
k=0: Keine Zwischenzustände möglich (da $0 < 1$)

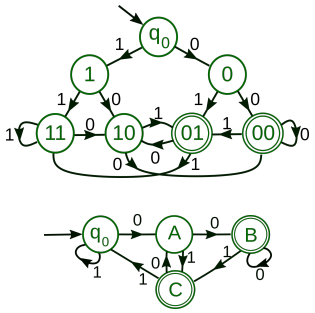
$$\rightsquigarrow L_{p,q}^0 = \begin{cases} \{\sigma \in \Sigma : \delta(p,\sigma) = q\} & \text{für } p \neq q \\ \{\sigma \in \Sigma : \delta(p,\sigma) = p\} \cup \{\varepsilon\} & \text{für } p = q \end{cases}$$

$\rightsquigarrow$ Enthält endlich viele Zeichen σ und evtl. ε
$\rightsquigarrow$ Kann als regulärer Ausdruck dargestellt werden.



$k \to k+1$: Sei $\gamma_{p,q}^k$ ein regulärer Ausdruck für $L_{p,q}^k$
Dann gilt: $L_{p,q}^{k+1} = L_{p,q}^k \cup (L_{p,k+1}^k \circ (L_{k+1,k+1}^k)^* \circ L_{k+1,q}^k)$
$= \mathcal{L}(\gamma_{p,q}^k + \gamma_{p,k+1}^k(\gamma_{k+1,k+1}^k)^*\gamma_{k+1,q}^k)$ ✓
○ Also gibt es auch zu $L = \bigcup_{p \in F} L_{1p}^n$ einen regulären Ausdruck.

2.2.4 Minimierung endlicher Automaten



Motivation: DEAs können unnötig groß sein.

Betrachte etwa Beispiel (c) aus 2.2.1: DEA zu der Sprache $L = \{w \in \{0,1\}^*,$ Vorletztes Zeichen is $0\}$

- Naive Konstruktion (2.2.1.1) $\rightsquigarrow$ $|Q| = 7$ Zustände
- In 2.2.1.2 haben wir für die gleiche Sprache einen kleineren DEA konstruiert $\rightsquigarrow$ nur $|Q| = 4$ Zustände

Für Anwendungen und für den Vergleich von Sprachen wäre es oft günstig, die Größe von Automaten zu minimieren.

Fragen: Gegeben sei eine formale Sprache $L \subseteq \Sigma^*$

(a) Welche minimale Größe muss ein DEA haben, der L akzeptiert?

(b) Wie kann man konkret aus einem DEA einen minimalen Automaten, konstruieren, der die gleiche Sprache akzeptiert?

$\rightsquigarrow$ Diese Fragen sollen nun der Reihe nach beantwortet werden.

2.2.4.1 Der Satz von Myhill und Nerode

Wir beantworten zunächst die Frage (a): Sei $L \subseteq \Sigma^*$ eine reguläre Sprache: Wie viele Zustände muss ein DEA mindestens haben, der L akzeptiert?

Wir bestimmen zunächst eine Untergrenze für diese Mindestanzahl.

⋆ Definitionen: Sei $L \subseteq \Sigma^*$ eine formale Sprache und $x, y \in \Sigma^*$

- Nerode-Relation: $x \sim_L y \Leftrightarrow (\forall\, w \in \Sigma^* : xw \in L \leftrightarrow yw \in L)$
  $\sim_L$ ist eine Äquivalenzrelation (reflexiv, symmetrisch transitiv)
  $\Rightarrow$ definiert Äquivalenzklassen $[x]_L = \{y \in \Sigma^* : y \sim_L x\}$
- Index von L (Index(L)): Anzahl der Äquivalenzklassen bzgl. $\sim_L$ in Σ^*

⋆ Satz: Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DEA mit totaler Übergangsfunktion, || der L akzeptiert ($\mathcal{L}(M) = L$). Dann gilt $|Q| \geqslant \text{Index}(L)$

Beweis: Über zweite Äquivalenzrelation $x \sim_M y \Leftrightarrow \delta(q_0, x) = \delta(q_0, y)$ mit Äquivalenzklassen: $[x]_M = \{y \in \Sigma^* : y \sim_M x\}$

- ○ Die Relation $x \sim_M y$ impliziert $x \sim_L y$
  ($x \sim_M y$ und $xw \in L \Rightarrow \delta(q_0, yw) = \delta(\delta(q_0, y), w) = \delta(\delta(q_0, x), w) = \delta(q_0, xw) \in F$)

$\Rightarrow$ Die Anzahl der Äquivalenzklassen bzgl. $\sim_L$ (d.h. Index(L)) ist kleiner oder gleich der Anzahl der Äquivalenzklassen bzgl. $\sim_M$
(Wegen $x \sim_M y \rightarrow x \sim_L y$ teilt $\sim_M$ feiner in Äquivalenzklassen ein als $\sim_L$)

- ○ Die Anzahl der Äquivalenzklassen bzgl. $\sim_M$ ist kleiner oder gleich $|Q|$
  (Offensichtlich: Es kann maximal $|Q|$ verschiedene Zustände q, q' geben mit $\delta(q_0, x) = q \neq \delta(q_0, y) = q'$ für $x, y \in \Sigma^*$)

$\Rightarrow$ Damit folgt die Behauptung: Index(L) $\leqslant |Q|$

Nun zeigen wir: Index(L) ist tatsächlich die Minimalgröße eines DEA zu L, d.h. es gibt einen DEA, der L akzeptiert und die Größe Index(L) hat.

⋆ Konstruiere dazu den "Nerode-Automaten" $M_L = (Q_L, \Sigma, \delta_L, q_{0L}, F_L)$
über: $Q_L = \{[x]_L : x \in \Sigma^*\},\ q_{0L} = [\varepsilon]_L,\ F_L = \{[w]_L : w \in L\}$,
$\delta_L : \delta_L([x]_L, \sigma) = [x\sigma]_L$

⋆ Zeige: (i) M_L ist wohldefiniert, d.h. die Definition der Abbildung δ_L ist unabhängig vom Repräsentanten.
(ii) $\mathcal{L}(M_L) = L$

(zu (i): Zu zeigen ist $(x \sim_L y \to (x\sigma) \sim_L (y\sigma))$ für alle $x, y \in \Sigma^*,\ \sigma \in \Sigma$
Es gilt: $x \sim_L y \Leftrightarrow\ (\forall w \in \Sigma^* : xw \in L \leftrightarrow yw \in L)$
$\Rightarrow\ (\forall w' \in \Sigma^* : x\sigma w' \in L \leftrightarrow y\sigma w' \in L) \Leftrightarrow\ x\sigma \sim_L y\sigma$ ✓
zu (ii): Analog zu (i) kann man für die erweiterte Übergangsfunktion zeigen:
$\delta_L([x]_L, w) = [xw]_L$ für alle $w \in \Sigma^*$
Speziell: $\delta_L(q_{0L}, w) = \delta_L([\varepsilon]_L, w) = [w]_L \in F_L \leftrightarrow w \in L$ ✓)

Fazit und Schlussfolgerungen

A: Frage (a) wurde beantwortet: Ein minimaler DEA $M = (Q, \Sigma, \delta, q_0, F)$ mit totaler Übergangsfunktion, der die Sprache L akzeptiert, hat die Größe $|Q| = \text{Index}(L)$

B: Satz von Myhill und Nerode: Eine Sprache $L \subseteq \Sigma^*$ ist genau dann regulär, wenn ihr Index endlich ist.
(Genau dann gibt es endliche Automaten, die L akzeptieren!)

2.2.4.2 Der Minimierungsalgorithmus

Nun kommen wir zu Frage (b) von 2.2.4. Der Nerode-Automat ist ein theoretisches Konstrukt. Wie kann man in der Praxis aus einem vorgegebenen DEA einen minimalen Automaten konstruieren? Um in der Praxis Zustände in DEAs zusammenzufassen, brauchen wir eine Äquivalenzrelation für Zustände.

⋆ Vorab: Zustände, die von q_0 aus nicht erreicht werden können heißen überflüssig und werden vorab entfernt (d.h. Zustände $q : \nexists\, w \in \Sigma^* :\ \delta(q_0, w) = q$)

⋆ Definitionen: Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DEA und $p, q \in Q$

∘ Verschmelzungsrelation: $p \equiv_M q \ \Leftrightarrow\ \forall w \in \Sigma^* :\ \delta(p, w) \in F \leftrightarrow \delta(q, w) \in F$
$\equiv_M$ ist eine Äquivalenzrelation (reflexiv, symmetrisch, transitiv) und definiert Äquivalenzklassen $[p]_M = \{q \in Q :\ q \equiv_M p\}$

∘ Der Äquivalenzklassenautomat zu M ist $\tilde{M} = (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{q}_0, \tilde{F})$
mit $\tilde{Q} = \{[p]_M :\ p \in Q\},\ \tilde{q}_0 = [q_0]_M,\ \tilde{F} = \{[p]_M : p \in F\}$,
$\tilde{\delta}([p]_M, \sigma) = [\delta(p, \sigma)]_M$

NB: ∘ Es muss wieder überprüft werden, ob $\tilde{M}$ wohldefiniert ist,
d.h. $p \equiv_M q \ \Rightarrow\ \delta(p, \sigma) \equiv_M \delta(q, \sigma)\ \forall\, p, q \in Q,\ \sigma \in \Sigma$
∘ Für die erweiterte Übergangsfunktion gilt dann:
$\tilde{\delta}([p]_M, w) = [\delta(p, w)]_M\ \forall w \in \Sigma^*$

(Zur Wohldefiniertheit von $\tilde{M}$:

$p \equiv_M q \Leftrightarrow (\forall w \in \Sigma^* : \delta(p,w) \in F \leftrightarrow \delta(q,w) \in F)$

$\Rightarrow (\forall\, w' \in \Sigma^*,\ \sigma \in \Sigma :\ \delta(\delta(p,\sigma),w') = \delta(p,\sigma w') \in F$

$\leftrightarrow \delta(\delta(q,\sigma),w') = \delta(q,\sigma w') \in F$

$\Leftrightarrow\ \delta(p,\sigma) \equiv_M \delta(q,\sigma)$ ✓

Zur erweiterten Übergangsfunktion: Beweis per Induktion über $|w|$ – Übungsaufgabe)

⋆ Satz: Sei $\tilde{M}$ der Äquivalenzklassenautomat zu M ohne überflüssige Zustände.
Dann gilt (i) $\mathcal{L}(\tilde{M}) = \mathcal{L}(M)$
und (ii) $\tilde{M}$ ist minimal.

Beweis:

(i) $\mathcal{L}(\tilde{M}) \supseteq \mathcal{L}(M)$: $w \in \mathcal{L}(M) \Rightarrow \delta(q_0,w) \in F \Rightarrow \tilde{\delta}([q_0]_M, w) = [\delta(q_0,w)]_M \in \tilde{F}$ ✓

$\mathcal{L}(\tilde{M}) \subseteq \mathcal{L}(M)$: $w \in \mathcal{L}(\tilde{M}) \Rightarrow \tilde{\delta}(\tilde{q}_0, w) = [\delta(q_0,w)]_M \in \tilde{F} = \{[p]_M : p \in F\}$

Für Zustände $\delta(q_0,w) \in [\delta(q_0,w)]_M \in \tilde{F}$ gilt: $\exists\, p \in F$ mit $\delta(q_0,w) \equiv_M p$

$\Rightarrow\ (\forall\, w' \in \Sigma^* : \delta(p,w') \in F \leftrightarrow \delta(\delta(q_0,w),w') \in F)$.

Wähle $w' = \varepsilon$. Mit $\delta(p,\varepsilon) = p \in F$ folgt dann $\delta(\delta(q_0,w),\varepsilon) = \delta(q_0,w) \in F$.

Also gilt: Aus $[\delta(q_0,w)]_M \in \tilde{F}$ folgt $\delta(q_0,w) \in F\ \Rightarrow\ w \in \mathcal{L}(M)$ ✓

(ii) Wir zeigen: $x \sim_L y \Leftrightarrow \tilde{\delta}(\tilde{q}_0,x) = \tilde{\delta}(\tilde{q}_0,y)$ für $L = \mathcal{L}(M) = \mathcal{L}(\tilde{M})$ (mit der Nerode-Relation $\sim_L$ aus 2.2.4.1). Da jeder Zustand von $\tilde{M}$ von $\tilde{q}_0$ aus erreichbar ist, ist die Zahl der unterschiedlichen Zustände dann genau die Zahl der Äquivalenzklassen bzgl. $\sim_L$ d.h. $|\tilde{Q}| = \text{Index}(L)$

$x \sim_L y\ \Leftrightarrow\ (\forall\, w \in \Sigma^* :\ xw \in L \leftrightarrow yw \in L)$

$\Leftrightarrow\ (\forall\, w \in \Sigma^* :\ F \ni \delta(q_0,xw) = \delta(\delta(q_0,x),w)$

$\leftrightarrow F \ni \delta(q_0,yw) = \delta(\delta(q_0,y),w)$

$\Leftrightarrow\ \delta(q_0,x) \equiv_M \delta(q_0,y)\ \Leftrightarrow\ \tilde{\delta}(\tilde{q}_0,x) = \tilde{\delta}(\tilde{q}_0,y)$ ✓

NB: Wenn wir den Äquivalenzklassenautomaten und dem Nerode-Automaten M_L aus 2.2.4.1 vergleichen, zeigt sich wegen: $[x]_L = \delta_L(q_0,x)$, dass $\delta_L(q_0,x) = \delta_L(q_0,y) \Leftrightarrow \tilde{\delta}(\tilde{q}_0,x) = \tilde{\delta}(\tilde{q}_0,y)$. Damit sind M_L und $\tilde{M}$ isomorph, d.h. sie unterscheiden sich nur durch die Bezeichnung der Zustände!

⋆ Minimierungsalgorithmus:

Wir beantworten endlich die Frage (b) aus 2.2.4: M kann minimiert werden über die Konstruktion des Äquivalenzautomaten $\tilde{M}$.

1. Schritt: Entferne überflüssige Zustände (Tiefensuche, Kosten $\mathcal{O}(|Q||\Sigma|)$)

2. Schritt: Identifiziere äquivalente Zustandspaare (p,q) über ein Ausschlussprinzip: Sortiere nichtäquivalente Paare aus.

Dazu: – Markiere Paare ($p' \in F$, $q' \notin F$) (sicher nicht äquivalent)

– Gehe rückwärts: Bestimme Paare (p'',q''), für die es ein $\sigma \in \Sigma$ und ein bereits markiertes Paar (p',q') gibt mit $p' = \delta(p'',\sigma)$ und $q' = \delta(q'',\sigma)$.
Markiere diese Paare (p'',q''), falls noch unmarkiert.

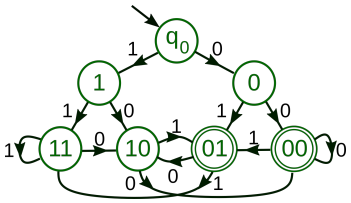
– Wiederhole so lange, bis keine unmarkierten Paare mehr erreicht werden.

→ Die verbliebenen nicht markierten Paare sind äquivalent.

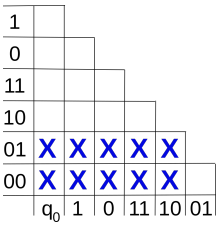
⇝ Automatisierbares Verfahren, Kosten $\mathcal{O}(|Q|^2|\Sigma|)$

⋆ Beispiel: Minimierung des DEA aus Beispiel (c) in 2.2.1.1
(DEA zu der Sprache $L = \{w \in \{0,1\}^* : \text{Vorletztes Zeichen ist } 0\}$)

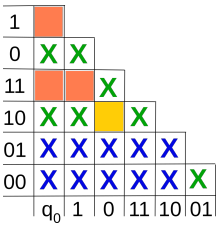
Ausgangspunkt:



Markiere
$p' \in F, q' \notin F$

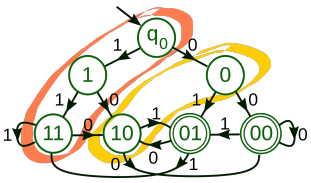


Markiere weiter
bis nichts mehr geht.

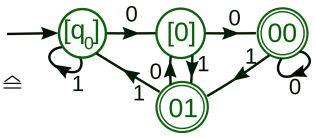


⇝ Aus den unmarkierten Paaren konstruiert man die Äquivalenzklassen $q_0 \equiv_M 1 \equiv_M 11$ (orange) und $0 \equiv_M 10$ (gelb)

⇒ Äquivalenzautomat:
(entspricht hier der Konstruktion aus 2.2.1.2!)



≙



2.2.5 Zusammenfassung

In dem Abschnitt 2.2 wurden reguläre Sprachen behandelt.
Wichtige Ergebnisse sind:

⋆ Charakterisierung regulärer Sprachen
Sei $L \subseteq \Sigma^*$ eine Sprache. Folgende Aussagen sind äquivalent:

(I) $L = \mathcal{L}(G)$ für eine Typ-3 Grammatik G
(II) $L = \mathcal{L}(M)$ für einen DEA M
(III) $L = \mathcal{L}(M')$ für einen NEA M'
(IV) $L = \mathcal{L}(\gamma)$ für einen regulären Ausdruck
(V) Index(L) ist endlich.

⋆ Abschlusseigenschaften

Reguläre Sprachen sind abgeschlossen unter Vereinigung, Durchschnitt, Komplement, Konkatenation, und Kleene-Abschluss.

⋆ Praktische Aspekte

– Das Pumping-Lemma: Kann helfen, zu entscheiden, ob eine Sprache regulär sein könnte (bzw. dass sie es garantiert nicht ist).
– Minimierungsalgorithmus: Ermöglicht die Minimierung von Automaten in polynomialer Zeit $\mathcal{O}(|Q|^2|\Sigma|)$
– Wichtige Probleme in formalen Sprachen lassen sich ebenfalls effizient lösen, z.B. das Wortproblem in der Zeit $\mathcal{O}(|w|)$.

2.3 Kontextfreie Sprachen

Erinnerung vorab

Kapitel 2.1.3: Chomsky-Klassifizierung von Grammatiken $G = (V, \Sigma, P, S)$ anhand der erlaubten Formen von Produktionen $(l \to r) \in P$

- Typ 0: Keine Einschränkungen an $(l \to r) \in P$
- Typ 1 (kontextsensitiv): $|l| \leqslant |r|$ mit "ε-Sonderregel" (siehe unten)
- Typ 2 (kontextfrei): $l \in V$
- Typ 3 (regulär): $l \in V$ und $r \in \Big(\{\varepsilon\} \cup \Sigma \cup (\Sigma V)\Big)$

Kapitel 2.3: Typ-3 Grammatiken und reguläre Sprachen

Vorteil: Das Wortproblem ("ist $w \in \mathcal{L}(G)$?") kann in der Zeit $\mathcal{O}(|w|)$ gelöst werden.

Nachteil: Wenig ausdrucksstark, scheitert schon an der $a^n b^n$-Sprache bzw. der Klammersprache D_2.

Thema dieses Kapitels 2.3: Typ-2 Grammatiken, kontextfreie Sprachen

$\rightsquigarrow$ Sehr viel mächtiger, das Wortproblem lässt sich immer noch in polynomialer Zeit lösen ($\mathcal{O}(|w|^3)$, wie wir sehen werden).

Beispiele für kontextfreie Sprachen

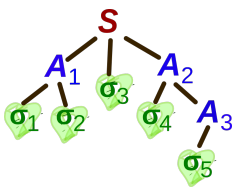
1) Alle regulären Sprachen sind auch kontextfrei.
2) "$a^n b^n$"-Sprache: $\Sigma = \{a, b\}$, $V = \{S, X\}$, $P : S \to \varepsilon | X,\ X \to ab | aXb$
3) D_2-Sprache $\Sigma = \{(,), [,]\}$, $V = \{S, X\}$, $P : S \to \varepsilon | X,\ X \to () | [] | (X) | [X] | XX$
4) Scigen: https://pdos.csail.mit.edu/archive/scigen (generiert wissenschaftliche Artikel)

2.3.1 Ableitungsbäume

Frage: Gegeben eine Grammatik $G = (V, \Sigma, P, S)$ und ein Wort $w \in \mathcal{L}(G)$. Was ist die "grammatikalische Struktur" von w, d.h. wie kann man die Ableitung $S \Rightarrow^* w$ näher charakterisieren?

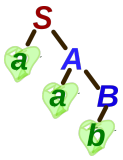
Lösungsvorschlag: Graphische Darstellung mit geordneten gewurzelten Bäumen

Wurzel: Startsymbol S
Innere Knoten: Variable $A \in V$
Blätter: Terminale $\sigma \in \Sigma$
$\rightsquigarrow$ Wort w: Blätter, von links nach rechts gelesen



2.3.1.1 Beispiele

(a) Reguläre Sprache $L = \{a^n b^m : n, m \in \mathbb{N}_0\}$,
Grammatik $G = (V, \Sigma, P, S)$, mit $\Sigma = \{a, b\}$, $V = \{S, A, B\}$,
$P : S \to \varepsilon|a|b|aA|aB|bB$, $A \to a|aA|aB$, $B \to b|bB$



Betrachte das Wort $w = aab$. Die Ableitung ist
$S \Rightarrow aA \Rightarrow aaB \Rightarrow aab$

Binärbaum: Gilt allgemein für reguläre Grammatiken

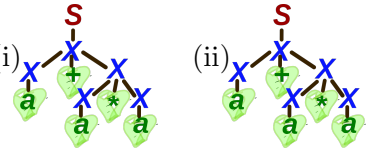
(b) Kontextfreie Sprache $L = \{$ Arithmetische Ausdrücke in Variablen $a, b\}$
Grammatik $G = (V, \Sigma, P, S)$, mit $\Sigma = \{a, b, +, -, *, (,)\}$, $V = \{S, X\}$,
$P : S \to \varepsilon|X$, $X \to X + X|X * X|(X)|a|b$

Betrachte das Wort $a + a * b$. Mögliche Ableitungen sind
(ersetzte Variablen sind unterstrichen)

(i) $S \Rightarrow \underline{X} \Rightarrow \underline{X} + X \Rightarrow a + \underline{X} \Rightarrow a + \underline{X} * X \Rightarrow a + a * \underline{X} \Rightarrow a + a * b$

(ii) $S \Rightarrow \underline{X} \Rightarrow X + \underline{X} \Rightarrow X + X * \underline{X} \Rightarrow X + \underline{X} * b \Rightarrow \underline{X} + a * b \Rightarrow a + a * b$

$\rightsquigarrow$ Ableitungsbäume (i) (ii)

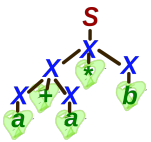


(i) ist eine Linksableitung: Ersetze immer die Variable, die am weitesten links ist.

(ii) ist entsprechend eine Rechtsableitung

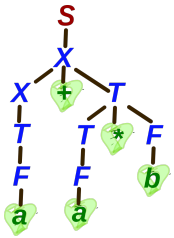
Aber: Beide Ableitungen erzeugen denselben Ableitungsbaum.

(iii) $S \Rightarrow \underline{X} \Rightarrow X * \underline{X} \Rightarrow \underline{X} * b \Rightarrow X + \underline{X} * b$
$\Rightarrow \underline{X} + a * b \Rightarrow a + a * b$
$\rightsquigarrow$ Rechtsableitung mit neuem Ableitungsbaum:



(c) Selbe Sprache wie (b), aber andere Grammatik, $G = (V, \Sigma, P, S)$
mit $\Sigma = \{a, b, +, -, *, (,)\}$, $V = \{S, X, T, F\}$,
$P : S \to \varepsilon|X$, $X \to X + T|T$, $T \to T * F|F$, $F \to (X)|a|b$

$\rightsquigarrow$ Der Ableitungsbaum zu $a + a * b$ wird eindeutig!



Generell: In dieser Grammatik sind die Ableitungsbäume aller Worte eindeutig!

(d) Sprache $L = \{a^i b^j c^k : i = j \text{ oder } j = k\} = L_1 \cup L_2$
mit $L_1 = \{a^i b^i\} \circ \{c^k\}$, $L_2 = \{a^i\} \circ \{b^k c^k\}$

Die Sprache ist kontextfrei. Eine mögliche Grammatik ist z.B.
$G = (V, \Sigma, P, S)$ mit $\Sigma = \{a, b, c\}$, $V = \{X_a, X_c, X_{ab}, X_{bc}\}$,
$P : S \to X_a X_{bc}|X_{ab} X_c$, $X_{ab} \to aX_{ab}b|\varepsilon$, $X_{bc} \to bX_{bc}c|\varepsilon$, $X_a \to aX_a|\varepsilon$, $X_c \to cX_c|\varepsilon$

Aber: Das Wort $a^n b^n c^n$ hat in keiner Grammatik einen eindeutigen Ableitungsbaum (ohne Beweis)

Anschaulicher Grund: Es gibt Ableitungsbäume für $w \in L_1$ und für $w \in L_2$, aber sie sind strukturell verschieden. Für Worte $w \in L_1 \cap L_2$ kann man immer beide benutzen!

2.3.1.2 Definitionen

Die Beispiele aus dem vorigen Abschnitt motivieren folgende formale Definitionen:
Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik

- ⋆ Rechtsableitung/Linksableitung
  - ○ Rechtsableitung $\Rightarrow_R$
    $u \Rightarrow_R u'$, falls es eine Zerlegung $u = xAz$, $u' = xyz$ gibt
    mit $z \in \Sigma^*$, $A \in V$, $x \in (V \cup \Sigma)^*$ und $(A \to y) \in P$
    $\overset{*}{\Rightarrow}_R$ ist die reflexive transitive Hülle von $\Rightarrow_R$
    (d.h. $u \overset{*}{\Rightarrow}_R u$ und $u \overset{*}{\Rightarrow}_R v \leftrightarrow \exists w_1, ...w_k \in (V \cup \Sigma)^* : u \Rightarrow_R w_1 \Rightarrow_R ... \Rightarrow_R w_k \Rightarrow v$)
  - ○ Linksableitung $\Rightarrow_L$: analog
- ⋆ Eindeutigkeit/Mehrdeutigkeit
  - ○ Eine Grammatik G heißt eindeutig, wenn es zu jedem Wort $w \in \mathcal{L}(G)$ genau einen Ableitungsbaum gibt (z.B. (a,c) in 2.3.1.1)
    Anderenfalls heißt G mehrdeutig (z.B. (b,d) in 2.3.1.1)
    Sprachliches Beispiel: "*Ich sehe den Mann mit dem Fernglas*"
  - ○ Eine Sprache L heißt eindeutig, wenn es eine eindeutige Grammatik G gibt mit $L = \mathcal{L}(G)$ (z.B. (a,b,c) in 2.3.1.1)
    Anderenfalls heißt L inhärent mehrdeutig (z.B. (d) in 2.3.1.1).
- ⋆ Nutzlose Variablen (vollständigkeitshalber)

  Nutzlose Variablen sind Variablen, die nicht zur Erzeugung der Sprache beitragen. Die Eliminierung nutzloser Variablen (außer S) ändert $\mathcal{L}(G)$ also nicht. Eine Variable $A \in V$ kann aus zwei Gründen nutzlos sein:
  - ○ Es gibt kein $w \in \Sigma^*$ mit $A \overset{*}{\Rightarrow} w$ (A kann nicht "aufgelöst" werden)
    Beispiel: $\Sigma = \{a\}$, $V = \{S, X, A\}$, $P : S \to \varepsilon|X$, $X \to a|aA$, $A \to aA$
    $\rightsquigarrow$ Hier ist A nutzlos.
    Eliminierungsverfahren:
    (i) Markiere alle $\{B \in V : B \Rightarrow w$ für ein $w \in \Sigma^*\}$
    (ii) Markiere alle $B' \in V$, für die es eine Regel $B' \to B_1...B_n$ gibt, in der alle B_i entweder markierte Variablen oder Terminale sind. Wiederhole dieses, bis es nichts mehr zu markieren gibt.
    (iii) Die verbliebenen unmarkierten Variablen sind nutzlos. Entferne sie und alle damit verbundenen Produktionen.
  - ○ Es gibt keine Ableitung $S \overset{*}{\Rightarrow} xAy$ mit $x, y \in (V \cup \Sigma)^*$
    (d.h. A kann nicht aus dem Startsymbol S erzeugt werden)
    Eliminierungsverfahren (analog oben):
    (i) Markiere S
    (ii) Markiere alle $B' \in V$, für die es eine Regel $B \to xB'y$ gibt, in der B markiert ist und $x, y \in (\Sigma \cup V)^*$.
    Wiederhole dieses, bis es nichts mehr zu markieren gibt.
    (iii) Entferne die verbliebenen unmarkierten Variablen und ihre Produktionen.

2.3.2 Lösung des Wortproblems für kontextfreie Sprachen

Wir wenden uns wieder dem Wortproblem zu: Sei $L \subseteq \Sigma^*$ eine Sprache und $w \in \Sigma^*$ ein Wort. Können wir herausfinden, ob $w \in L$ ist?
Falls L regulär ist, kann man dieses Problem in der Zeit $\mathcal{O}(|w|)$ lösen (z.B. mit Hilfe des DEA zu L). Für kontextfreie Sprachen L werden wir zeigen, dass dann immerhin noch eine Lösung in der Zeit $\mathcal{O}(|w|^3)$ möglich ist. Das Verfahren dazu ist Gegenstand dieses Kapitels. Ausgangspunkt ist eine kontextfreie Grammatik G mit $L = \mathcal{L}(G)$.

- G wird in die "Chomsky-Normalform" (CNF) überführt (→ 2.3.2.1)
- Für CNF Grammatiken gibt es einen Algorithmus, der das Wortproblem in der Zeit $\mathcal{O}(|w|^3$ löst: Der CYK Algorithmus (→ 2.3.2.2)

2.3.2.1 Die Chomsky-Normalform (CNF)

⋆ Definition: Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik
|| G ist in Chomsky-Normalform (CNF), wenn sie die ε-Sonderregel erfüllt und alle Produktionen die Form $S \to \varepsilon$, $A \to \sigma$ mit $\sigma \in \Sigma$ oder $A \to BC$ mit $B, C \in V$ haben.

Anmerkung: In Asteroth/Baier und Hopcroft/Motwani/Ullmann sind Produktionen $S \to \varepsilon$ nicht erlaubt und $\mathcal{L}(G)$ kann demnach das leere Wort nicht enthalten.

⋆ Satz: Die Sprache L sei kontextfrei
|| Dann gibt es eine CNF-Grammatik, die L erzeugt.

Beweis: Da L kontextfrei ist, gibt es eine ε-freie Typ 2-Grammatik $G = (V, \Sigma, P, S)$ mit $L = \mathcal{L}(G)$ (Konstruktion siehe 2.1.3.2).
Daraus kann man wie folgt eine CNF Grammatik konstruieren:

1. Schritt: Eliminierung von Kettenregeln $A \to B$ $(A, B \in V)$
(i) Entferne Zyklen $A_1 \to A_2 \to \ldots A_k \to A_1$ (d.h. $A_1 \to A_2$, $A_2 \to A_3 \ldots$)
· Finde Zyklen durch eine Tiefensuche
· Für den Zyklus $A_1 \to A_2, \ldots A_k \to A_1$:
Ersetze in allen Regeln $A_2, \ldots A_k$ durch A_1
Entferne die Regel $A_1 \to A_1$
(ii) Nummeriere alle Variablen so (um), dass $A_i \to A_j \leftrightarrow i < j$
(z.B. über einen topologischen Sortierungsalgorithmus)
(iii) Eliminiere die restlichen Kettenregeln
· Ausgehend von den sortierten Variablen $A_1 \ldots A_m$
· FOR $j = m, m-1, \ldots 1$ DO
| Ermittle alle Regeln $A_i \to A_j$ $(i < j)$
| Entferne die Regel $A_i \to A_j$
| Füge für jede Regel $A_j \to X$ eine Regel $A_i \to X$ hinzu
DONE
NB: Dabei werden maximal $\mathcal{O}(|P|^2)$ neue Regeln erzeugt.

2. Schritt: Eliminierung von Terminalen $\sigma \in \Sigma$ aus Regeln $(l \to r)$ mit $|r| \geqslant 2$

(i) Füge für jedes $\sigma \in \Sigma$ eine neue Variable X_σ zu V
und eine Regel $X_\sigma \to \sigma$ zu P hinzu.

(ii) Ersetze in Regeln $(l \to r)$ mit $|r| \geqslant 2$ alle $\sigma \in \Sigma$ durch X_σ

3. Schritt: Verkürzung von langen Regeln $(l \to r)$ mit $|r| > 2$
Für alle Regeln $A \to B_1 ... B_n$ mit $n > 2$

· Füge neue Variablen $C_1, ... C_{n-2}$ zu V hinzu

· Ergänze die Regeln um $A \to B_1 C_1,\ C_1 \to B_2 C_2,\ C_{n-2} \to B_{n-1} B_n$

· Eliminiere die Regel $A \to B_1 ... B_n$

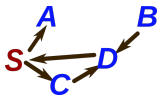
Bemerkung: Die neue Grammatik $G' = (V', \Sigma, P', S)$ ist größer als G.
Size$(G) \triangleq$ Summe der Länge aller $p \in P$ und $|V|$
Size$(G') \sim \mathcal{O}($ Size$(G)^2)$ – im Wesentlichen wegen Schritt 1-(iii)

⋆ Beispiele

(a) Kontextfreie Grammatik $G = (V, \Sigma, P, S)$
mit $\Sigma = \{a, b\},\ V = \{S, A, B, C, D\}$
und $P : S \to A|C,\ A \to aB,\ B \to b|bb|bA|D,\ C \to aCb|D,\ D \to S$

1. Schritt: Eliminiere Kettenregeln

(i) • Finde Zyklen
→ Erstelle Graphen für Kettenregeln
⇝ Es gibt einen Zyklus $S \to C \to D \to S$



• Ersetze überall C, D durch S, eliminiere Regeln $S \to S$
⇝ $V' = \{S, A, B\}$,
$P' : S \to A|aSb,\ A \to aB,\ B \to b|bb|bA|S$

(ii) Nummeriere bzw. sortiere Variablen ⇝ Reihenfolge B, S, A

(iii) Eliminiere verbliebene Kettenregeln (neue Regeln sind unterstrichen)
$S \to A$: Entferne $S \to A$, ergänze $S \to aSb|\underline{aB}$
$B \to S$: Entferne $B \to S$, ergänze $B \to b|bb|bA|\underline{aSb}|\underline{aB}$

2. Schritt: Eliminiere Terminale aus langen Regeln $l \to r$ mit $|r| \geqslant 2$

(i) Neue Variablen X_a, X_b, neue Regeln $X_a \to a,\ X_b \to b$

(ii) Ersetze: $S \to X_a S X_b | X_a B,\ A \to X_a B,$
$B \to b|X_b X_b|X_b A|X_a S X_b|X_a B$

3. Schritt: Verkürze lange Regeln $l \to r$ mit $|r| > 2$

• Lange Regeln sind: $S \to X_a S X_b$ und $B \to X_a S X_b$
⇝ Es reicht *eine* neue Variable Y

• Streiche lange Regeln, ergänze $S \to X_a Y,\ B \to X_a Y$ mit $Y \to S X_b$

⇒ CNF Grammatik: $G' = (V', \Sigma, P, S)$ mit $V' = \{S, A, B, X_a, X_b, Y\}$
und $P' : S \to X_a Y | X_a B,\ A \to X_a B,\ B \to b|X_b X_b|X_b A|X_a Y|X_a B,$
$Y \to S X_b,\ X_a \to a,\ X_b \to b$

(b) $a^n b^n$-Sprache, Grammatik $G = (V, \Sigma, P, S)$
mit $V = \{S, X\},\ P : S \to X,\ X \to ab|aXb$

1. Schritt (keine Zyklen) Eliminiere Kettenregel $S \to X$ nach (iii)
⇝ $P' : S \to ab|aXb,\ X \to ab|aXb$

2. Schritt: Eliminiere Terminale aus langen Regeln
Betroffen sind die Regeln $S \to aXb,\ X \to aXb$
⇝ $V' = \{S, X, X_a, X_b\}$
$P' : S \to X_a X_b | X_a X X_b,\ X \to X_a X_b | X_a X X_b,\ X_a \to a,\ X_b \to b$

3. Schritt: Verkürze lange Regeln
Betroffen sind die Regeln: $S \to X_a X X_b,\ X \to X_a X X_b$
$\rightsquigarrow\ V' = \{S, X, X_a, X_b, Y\}$
$P' : S \to X_a X_b | X_a Y,\ X \to X_a X_b | X_a Y,\ Y \to X X_b,\ X_a \to a,\ X_b \to b$

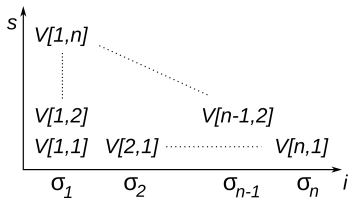
$\Rightarrow$ CNF Grammatik: $G' = (V', \Sigma, P, S)$ mit $V' = \{S, X, X_a, X_b, Y\}$
und $P' : S \to X_a X_b | X_a Y,\ X \to X_a X_b | X_a Y,$
$Y \to X X_b,\ X_a \to a,\ X_b \to b$

2.3.2.2 Der CYK Algorithmus (Cocker, Young, Kasami)

Wir lösen nun das Wortproblem für Grammatiken in Chomsky-Normalform:
Gegeben sei eine CNF-Grammatik $G = (V, \Sigma, P, S)$, $w \in \Sigma^*$: Ist $w \in \mathcal{L}(G)$?

Algorithmus (Dynamische Programmierung) Sei $w = \sigma_1 ... \sigma_n$, $\sigma_i \in \Sigma$

- Betrachte Teilworte $(\sigma_i ... \sigma_{i+s-1})$ der Länge s $(1 \leqslant i \leqslant i+s-1 \leqslant n)$
  und bestimme $V[i,s] = \{A \in V : A \Rightarrow^* (\sigma_i ... \sigma_{i+s-1})\}$ rekursiv
  (i) Initialisiere: $V[i,1] = \{A \in V : A \Rightarrow^* \sigma_i\} = \{A \in V : A \Rightarrow \sigma_i\}$
  (geht, da G Chomsky Normalform hat) Kosten: $\mathcal{O}(|P|\,|w|)$
  (ii) FOR $s = 2, 3, ...|w|$ DO
  Bestimme $V[i,s]$; alle $V[i,s']$ mit $s' < s$ bereits bekannt:
  $A \in V[i,s] \Leftrightarrow A \Rightarrow^* \sigma_i ... \sigma_{i+s-1}$
  $\Leftrightarrow \exists l \in \{1, ..., s-1\}$ und Regel $A \to BC \in P$
  mit $B \Rightarrow^* \sigma_i ... \sigma_{i+l-1}$, d.h. $B \in V[i,l]$
  und $C \Rightarrow^* \sigma_{i+l} ... \sigma_{i+s-1}$, d.h. $C \in V[i+l, s-l]$
  Bestimme alle $A \in V$, für die dieses gilt,
  also $V[i,s] = \bigcup_{l=1}^{s-1} \{A \in V : \exists$ Regel $(A \to BC) \in P$
  mit $B \in V[i,l] \wedge C \in V[i+l, s-l]$
  DONE
  Darstellung als Tabelle
  NB: Kosten (Zeit): $\mathcal{O}(|P|\,|w|^3)$,
  Platzbedarf $\mathcal{O}(|w|^2 |V|)$



- Es gilt: $w \in \mathcal{L}(G) \Leftrightarrow S \in V[1,n]$

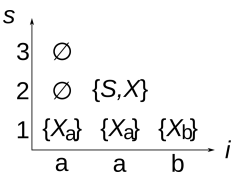
⋆ Beispiel: $a^n b^n$-Sprache mit Grammatik $G = (V, \Sigma, P, S)$
mit $\Sigma = \{a, b\}$, $V = \{S, X, X_a, X_b, Y\}$,
$P : S \to X_a X_b | X_a Y,\ X \to X_a X_b | X_a Y,\ Y \to X X_b,\ X_a \to a,\ X_b \to b$

Teste das Wort $w = aab$

$s = 1$: $V[i,1] = \{X_{\sigma_i}\}$: klar

$s = 2$: $V[1,2] = \{A \in V : \exists A \to BC$ mit $B \in V[1,1] \wedge C \in V[2,1]\}$
(l=1) $= \{A \in V : \exists A \to X_a X_a\} = \varnothing$
$V[2,2] = \{A \in V : \exists A \to X_a X_b\} = \{S, X\}$

$s = 3$: $V[1,3] = \{A \in V : \exists A \to BC$ mit $B \in V[1,1] \wedge C \in V[2,2]$
(l=1,2) oder $B \in V[1,2] \wedge C \in V[2,1]\}$
$= \{A \in V : \exists A \to X_a S$ oder $A \to X_a X\} = \varnothing$



$\Rightarrow\ S \notin V[1,r] (= \varnothing)$ und damit ist $w \in aab \notin \mathcal{L}(G)$ ✓

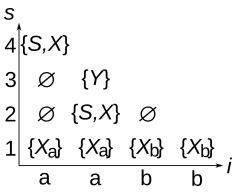
Teste das Wort $w = aabb$

$s = 1$: $V[i,1] = X_{\sigma_i}$: klar

$s = 2$: $V[1,2] = \{A \in V : \exists A \to BC \text{ mit } B \in V[1,1] \wedge C \in V[2,1]\}$
(l=1) $= \{A \in V : \exists A \to X_a X_a\} = \varnothing$
$V[2,2] = \{A \in V : \exists A \to X_a X_b\} = \{S, X\}$
$V[3,2] = \{A \in V : \exists A \to X_b X_b\} = \varnothing$

$s = 3$: $V[1,3] = \{A \in V : \exists A \to BC \text{ mit } B \in V[1,1] \wedge C \in V[2,2]$
(l=1,2) oder $B \in V[1,2] \wedge C \in V[2,1]\}$
$= \{A \in V : \exists A \to X_a S \text{ oder } A \to X_a X\} = \varnothing$
$V[2,3] = \{A \in V : \exists A \to BC \text{ mit } B \in V[2,1] \wedge C \in V[3,2]$
oder $B \in V[2,2] \wedge C \in V[4,1]\}$
$= \{A \in V : \exists A \to S X_b \text{ oder } A \to X X_b\} = \{Y\}$

$s = 4$: $V[1,4] = \{A \in V : \exists A \to BC \text{ mit } B \in V[1,1] \wedge C \in V[2,3]$
(l=1,2,3) oder $B \in V[1,2] \wedge C \in V[3,2]$
oder $B \in V[1,3] \wedge C \in V[4,1]\}$
$= \{A \in V : \exists A \to X_a Y\} = \{S, X\}$



$\Rightarrow$ $S \in V[1,r]$ und damit ist $w \in aabb \in \mathcal{L}(G)$ ✓

2.3.3 Eigenschaften kontextfreier Sprachen

2.3.3.1 Abschlusseigenschaften

Frage (analog 2.2.1.1): Kann man kontextfreie Sprachen kombinieren?

Antwort: Größtenteils bereits in Abschnitt 2.1.3.3 behandelt.

Satz über die Verknüpfung kontextfreier Sprachen

Die Sprachen L_1, L_2 seien kontextfrei. Dann gilt:

(i) $L_1 \cup L_2$ ist kontextfrei (siehe 2.1.3.3)

(ii) $L_1 \circ L_2$ ist kontextfrei (siehe 2.1.3.3)

(iii) L_i^* ist kontextfrei (siehe 2.1.3.3)

Aber:

- $L_1 \cap L_2$ ist nicht unbedingt kontextfrei
  Gegenbeispiel: $L_1 = \{a^n b^n c^m : n, m \in \mathbb{N}_0\}$,
  $L_2 = \{a^m b^n c^n : n, m \in \mathbb{N}_0\}$
  Die Sprachen L_i sind kontextfrei, da sie jeweils Konkatenationen von kontextfreien Sprachen sind
  ($\{a^n b^n\}$ und $\{c^m\}$ bzw. ($\{a^m\}$ und $\{b^n c^n\}$)
  Aber: $L_1 \cap L_2 = \{a^n b^n c^n : n \in \mathbb{N}_0\}$ ist nicht kontextfrei!
  (Beweis gleich in 2.3.3.2 mittels Pumping-Lemma)
- $\bar{L}_i = \Sigma^* \backslash L_i$ ist nicht notwendig kontextfrei
  (sonst wäre $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ kontextfrei)

2.3.3.2 Das Pumping Lemma für kontextfreie Sprachen

Analog 2.2.2.3: Notwendiges Kriterium dafür, dass eine Sprache kontextfrei ist

⋆ Satz: Pumping Lemma für kontextfreie Sprachen

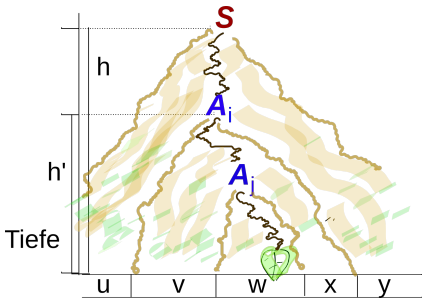
Sei $L \subseteq \Sigma^*$ eine kontextfreie Sprache. Dann gibt es ein $n \in \mathbb{N}$, so dass sich alle Worte $z \in L$ mit $|z| \geqslant n$ zerlegen lassen als $z = uvwxy$ mit $|vx| \geqslant 1$, $|vwx| \leqslant n$, $uv^k wx^k y \in L$ für alle $k \in \mathbb{N}_0$

⋆ Beweis: Da L kontextfrei ist, gibt es eine CNF-Grammatik $G = (V, \Sigma, P, S)$ mit $L = \mathcal{L}(G)$.

- Setze $n = 2^{|V|+1}$.
- Betrachte ein Wort $z \in L$ mit $|z| \geqslant n$ und seinen Ableitungsbaum

(i) Da es eine CNF-Grammatik ist, ist der Ableitungsbaum binär: Jede Variable hat als Kinder entweder zwei Variablen oder genau ein Blatt.

(ii) Die Tiefe des Baums ist h, er hat $|z|$ Blätter $\Rightarrow$ $|z| < 2^h$



$\Rightarrow$ Es gibt mindestens einen Pfad von S zu einem Blatt, der länger als $|V|$ ist! (denn: Es gibt einen Pfad der Länge h und $n = 2^{|V|+1} \leqslant |z| < 2^h \Rightarrow h > (|V|+1)$)

$\Rightarrow$ In diesem Pfad tauchen Variablen doppelt auf: $A_i = A_j$ mit $i < j$. Wähle das das Paar mit maximalem i (das unterste solche Paar). Dann sind ab $A_{i+1} ... A_{n-1}$ alle Variablen auf diesem Pfad paarweise verschieden.

$\Rightarrow$ Der Teilpfad von A_i bis zum Blatt hat die Höhe $h' \leqslant |V|$

- Zerlege z wie in der Skizze angedeutet in $z = uvwxy$
  mit w: Teilbaum, der aus A_j abgeleitet wurde ($A_j \Rightarrow^* w$)
  vwx: Teilbaum, der aus A_i abgeleitet wurde ($A_i \Rightarrow^* vwx$)
  $\rightsquigarrow$ Dann gilt per Konstruktion $|vx| \geqslant 1$
  und $|vwx| \leqslant 2^{h'} \leqslant 2^{|V|} < n$
  Ableitungen: $S \Rightarrow^* uA_iy = uA_jy$ (da $A_i = A_j$)
  $A_j \Rightarrow^* w$, aber auch $A_j = A_i \Rightarrow^* vA_jx$
  $\Rightarrow$ $S \Rightarrow^* uA_iy \Rightarrow^* uvA_ixy \Rightarrow^* uv^2A_ix^2y \Rightarrow^* ... \Rightarrow^* uv^kA_ix^ky$
  und $A_i \Rightarrow^* w$ ✓

⋆ Beispiel: Wie zeigen: $L = \{a^m b^m c^m : m \in \mathbb{N}_0\}$ kann nicht kontextfrei sein.

Beweis: Anderenfalls gäbe es ein $n \in \mathbb{N}$, so dass das Pumping Lemma gilt.
Wähle $z = a^n b^n c^n$ und betrachte Zerlegungen $z = uvwxy$ mit $|vwx| \leqslant n$.
$\rightsquigarrow$ vwx enthält entweder kein a oder kein c
$\Rightarrow$ $uv^kwx^ky \notin L$ für $k \neq 1$ (Auf-/Abpumpen ändert das Verhältnis $a : b : c$)

2.3.4 Kellerautomaten ("pushdown automata")

2.3.4.1 Motivation und Vorüberlegungen

⋆ Aus theoretischer Sicht: Automatenmodell für kontextfreie Sprachen.

⋆ Aus praktischer Sicht: Lösung des Wortproblem für kontextfreie Sprachen

Der CYK Algorithmus löst das Wortproblem, aber skaliert schlecht ($\mathcal{O}(|w|^3)$)
Frage: Kann man mit Hilfe von Automaten die Skalierung verbessern?

Leider stellt sich heraus, dass das nicht allgemein möglich ist.
Aber: Es gibt zumindest für eine Untermenge der kontextfreien Sprachen deterministische Automaten, die das Wortproblem in $\mathcal{O}(|w|)$ lösen.

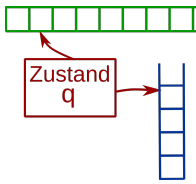
Die nichtdeterministische Variante akzeptiert alle kontextfreien Sprachen.

⋆ Wie müsste solch ein Automat aussehen?

– Offensichtlich nicht endlich

– Beispiel Klammersprache D_2 (korrekt geklammerte Ausdrücke)
Der Automat muss sich merken, welche Klammern in welcher Reihenfolge noch geschlossen werden müssen
(z.B. Wortanfang "([()[" erfordert Fortsetzung "...]...]...)...")
$\rightsquigarrow$ Nicht geschlossene Klammern müssen gespeichert werden.
$\rightsquigarrow$ "Keller" (engl. "Stack" $\triangleq$ Stapel) mit unendlich viel Platz!

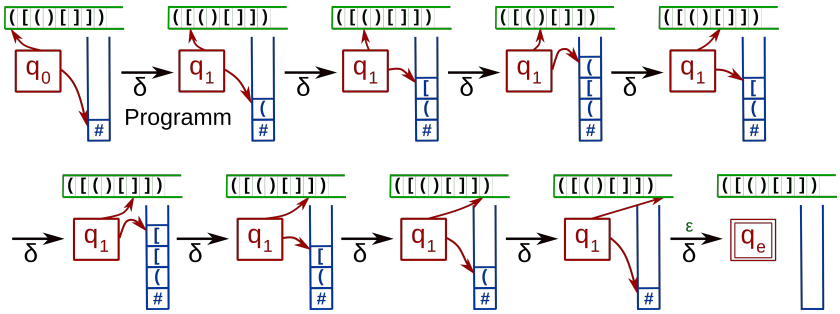
⋆ Idee: Ergänze Automaten um "Keller" bzw. "Ablagestapel"



Eingabeband ("Lesekopf" liest von links nach rechts)

Keller – Automat sieht nur oberstes Kellersymbol ("Stapel" trifft es besser!)

• Beispiel 1: Klammersprache D_2 und Wort $w = ([()[]])$
Der Automat legt Klammern auf dem Stapel ab und entfernt sie dort wieder, wenn sie von einer eingelesenen Klammer geschlossen werden.



NB: Der letzte Schritt ist ein nichtdeterministischer ε-Übergang.
Der Automat weiss ja nicht, ob das Wort noch weitergeht.

• Beispiel 2: $L = \{\text{Palindrome}\}$ (z.B. ANNA, TAT, RELIEFPFEILER)
Schwieriger. Der Automat weiß nicht, wo die Mitte des Wortes ist.
Er muss die Mitte des Wortes nichtdeterministisch "erraten".

2.3.4.2 Nichtdeterministische Kellerautomaten (NKA oder NPDA)

Wir formalisieren und verallgemeinern die Ideen aus 2.3.4.1

⋆ Definition Eine NKA ist ein 7-Tupel $K = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ mit

- Q: Endliche Menge von Zuständen
- Σ: Endliches Eingabealphabet
- Γ: Endliches Kelleralphabet
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$: Übergangsfunktion
- $q_0 \in Q$: Startzustand
- $\# \in \Gamma$: Keller-Startsymbol (oft auch $\perp$)
- $F \subseteq Q$: Menge akzeptierender Zustände (Endzustände)

Wirkungsweise der Übergangsfunktion δ:

- $\delta(q, \sigma, \gamma) = \{(q_1, G_1), ...(q_n, G_n)\}$ für $\sigma \in \Sigma,\ \gamma \in \Gamma,\ G_i \in \Gamma^*,\ q, q_i \in Q$
  - wird aufgerufen, falls K im Zustand q ist, σ das aktuelle Eingabezeichen, und γ das oberste Zeichen auf dem Kellerstapel ist
  - wählt (nichtdeterministisch) einen Übergang (q_i, G_i)
    führt aktuellen Zustand q in q_i über
    ersetzt γ im Stapel durch G_i
    G_i kann leer sein oder eine Zeichenfolge, z.B. $G_i = \gamma_{i_1}\gamma$
  - Lesekopf rückt auf dem Eingabeband einen Platz weiter.
- $\delta(q, \varepsilon, \gamma) = \{(q_1, G_1), ...(q_n, G_n)\}$
  - kann spontan eintreten, falls K im Zustand q ist
  - NKA rückt auf dem Eingabeband nicht weiter
  - ansonsten wie oben

⋆ Konfigurationen und Konfigurationswechsel

Formale Beschreibung dessen, was ein NKA bewirkt
≙ Analogon zur "erweiterten Übergangsfunktion" in Kapitel 2.2

- Eine Konfiguration für einen NKA K ist ein Tripel $\kappa = (q, x, \xi)$
  mit $q \in Q$ (aktueller Zustand), $x \in \Sigma^*, : \xi \in \Gamma^*$
  wobei x : noch einzulesender Teil eines Wortes
  ξ : aktueller Inhalt des Kellers
  Speziell Startkonfiguration: $\kappa_0 = (q_0, w, \#)$
  Die Menge aller Konfigurationen ist $\mathrm{Conf}(K) \subseteq Q \times \Sigma^* \times \Gamma^*$
- Die Konfigurationsrelation
  verknüpft κ mit den möglichen Nachfolgekonfigurationen, d.h.
  $(q, \sigma X, \gamma\xi) \vdash (q_i, X, G_i\xi)$, falls $(q_i, G_i) \in \delta(q, \sigma, \gamma) \quad (\sigma \in \Sigma \cup \{\varepsilon\})$
  Definiert wieder eine reflexive transitive Hülle $\vdash^*$
  (mit $\kappa \vdash^* \kappa,\ \kappa \vdash^* \kappa' \Leftrightarrow \exists \kappa_1 ... \kappa_k$ mit $\kappa \vdash \kappa_1 ... \kappa_k \vdash \kappa'$)

⋆ Akzeptierte Sprache: Es gibt zwei Varianten. Sei K ein NKA

- Die durch leeren Keller akzeptierte Sprache $\mathcal{L}_\varepsilon(K)$ ist
  $\mathcal{L}_\varepsilon(K) = \{w \in \Sigma^* : (q_0, w, \#) \vdash^* (q, \varepsilon, \varepsilon) \text{ für ein } q \in Q\}$
- Die durch Endzustände akzeptierte Sprache $\mathcal{L}(K)$ ist
  $\mathcal{L}(K) = \{w \in \Sigma^* : (q_0, w, \#) \vdash^* (q_F, \varepsilon, \xi) \text{ für ein } q_F \in F \text{ und } \xi \in \Gamma^* \}$

NB: – Außer am Schluss darf der Keller zu keinem Zeitpunkt leer sein, sonst wird der Lauf verworfen ($\delta(q, \sigma, \varepsilon)$ ist nicht definiert!)
– Im Allgemeinen ist $\mathcal{L}_\varepsilon(K) \neq \mathcal{L}(K)$. Aber wir werden zeigen, dass die Akzeptanzbedingungen trotzdem gleichmächtig sind: Zu jedem NKA K gibt es einen NKA K' mit $\mathcal{L}(K) = \mathcal{L}_\varepsilon(K')$ und umgekehrt.

⋆ Beispiel: NKA $K = (Q, \Sigma, \Gamma, \delta, q_0, \#, \cdot)$ für die Palindromsprache

Wähle $Q = \{+, -, f\}$, $q_0 = +$ und akzeptiere mit leerem Keller ($f \triangleq$ fail)

$$\delta(+, \sigma, \gamma) = \{\underbrace{(+, \sigma\gamma)}_{\substack{\text{Mitte nicht}\\ \text{erreicht}}}, \underbrace{(-, \sigma\gamma)}_{\substack{\text{Mitte errreicht,}\\ |w| \text{ gerade}}}, \underbrace{(-, \gamma)}_{\substack{\text{Mitte errreicht,}\\ |w| \text{ ungerade}}}\}$$

$$\delta(-, \sigma, \gamma) = \begin{cases} (-, \varepsilon) & \sigma = \gamma \\ (f, \gamma) & \text{sonst} \end{cases},$$

$$\delta(f, \sigma, \gamma) = \{(f, \gamma)\}, \quad \delta(q, \varepsilon, \#) = \{(q, \varepsilon)\}\ \forall q$$

Beispiele: Akzeptierende Läufe für (i) $w = abba$ und (ii) $w = aba$

(i) $\kappa_0 = (+, abba, \#) \vdash (+, bba, a\#) \vdash (-, ba, ba\#) \vdash (-, a, a\#) \vdash (-, \varepsilon, \#) \vdash (-, \varepsilon, \varepsilon)$

(ii) $\kappa_0 = (+, aba, \#) \vdash (+, ba, a\#) \vdash (-, a, a\#) \vdash (-, \varepsilon, \#) \vdash (-, \varepsilon, \varepsilon)$

2.3.4.3 Zusammenhang mit kontextfreien Sprachen

Wir zeigen nun: Die NKAs akzeptieren genau die kontextfreien Sprachen

Satz: Sei $L \subseteq \Sigma^*$ eine Sprache. Dann sind folgende Aussagen äquivalent

(I) Es gibt eine Typ 2-Grammatik, die L erzeugt: $L = \mathcal{L}(G)$

(II) Es gibt einen NKA K, der L mit Endzuständen akzeptiert: $L = \mathcal{L}(K)$

(III) Es gibt einen NKA K', der L mit leerem Keller akzeptiert: $L = \mathcal{L}_\varepsilon(K')$

Beweis: Wir zeigen (I) ⇒ (II) ⇒ (III) ⇒ (I)

⋆ Beweis (I) ⇒ (II): Zu jeder Typ 2-Grammatik G gibt es einen NKA K mit $\mathcal{L}(K) = \mathcal{L}(G)$

Da G kontextfrei ist, kann es in Chomsky-Normalform gebracht werden. Gegeben sei eine CNF-Grammatik $G = (V, \Sigma, P, S)$. Wir konstruieren den NKA $K = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ zu der Grammatik G über:

$Q = \{q_0, q_1\},\ F = \{q_1\}$,
$\Gamma = V \cup \{\$, \#\}$ mit $\$, \# \notin V$: neue Symbole

δ: (i) $\delta(q_0, \varepsilon, \#) = \{(q_0, S\$)\}$
(ii) $\delta(q_0, \varepsilon, A) \ni (q_0, BC)$, falls $A \to BC \in P$ $(A, B, C \in V)$
(iii) $\delta(q_0, \sigma, A) \ni (q_0, \varepsilon)$, falls $A \to \sigma \in P$ $(A \in V : \sigma \in \Sigma \cup \{\varepsilon\})$
(iv) $\delta(q_0, \varepsilon, \$) = \{(q_1, \varepsilon)\}$

Erläuterung: Ganz zu Beginn wird das neue Symbol \$ am Grund des Kellers abgelegt und kann nicht mehr entfernt werden. Ansonsten enthält der Keller Variablen der Grammatik G und wird nach den Produktionen von G befüllt. Erst wenn alle Variablen ersetzt werden, ist \$ oben und K kann in den akzeptierenden Zustand übergehen.

Zeige, dass $\mathcal{L}(K) = \mathcal{L}(G)$

Vorab: Für $A \in V$, $w \in \Sigma^*$ gilt: $A \Rightarrow_G^* w \Leftrightarrow (q_0, w, A\$) \vdash^* (q_0, \varepsilon, \$)$
Beweis über Induktion nach $|w|$

- Das Symbol \$ ist inert und kann nicht ersetzt oder entfernt werden, ohne den Zustand q_0 zu verlassen. Also ist
  $(q_0, w, A\$) \vdash^* (q_0, \varepsilon, \$) \Leftrightarrow (q_0, w, A) \vdash^* (q_0, \varepsilon, \varepsilon)$
  $\Leftrightarrow (q_0, wx, AX) \vdash^* (q_0, x, X)$ für alle $x \in \Sigma^*$, $X \in \Gamma^*$.
  Wir beweisen das Letztere.
- Induktionsanfang: $|w| = 0$ und $|w| = 1$. In diesem Fall ist $w = \sigma \in \Sigma$ oder $w = \varepsilon$ und es wird einmal Übergangsfunktion (iii) angewendet bzw. die Regel $A \to w$. ✓
- Induktionsannahme (IA): Die Behauptung gelte für $|w| \leqslant n$
- Induktionsschritt: Das Wort w habe die Länge $|w| = n + 1$

"⇒" $A \Rightarrow_G^* w$ mit $|w| = n + 1$
⇒ Da $|w| > 1$, muss die erste Ableitung die Form $A \to BC$ haben mit $B, C \in V$ und ε-Sonderregel.
⇒ Es gibt eine Zerlegung $w = w_B w_C$ mit $B \Rightarrow_G^* w_B$, $C \Rightarrow_G^* w_C$, und $|w_B|, |w_C| \leqslant n$.
⇒ Es gibt Konfigurationsübergänge
$(q_0, w_B w_C x, AX) \overset{\varepsilon}{\underset{\text{(ii)}}{\vdash}} (q_0, w_B w_C x, BCX)$
$\overset{\varepsilon}{\underset{\text{IA}}{\vdash}} (q_0, w_C x, CX) \overset{\varepsilon}{\underset{\text{IA}}{\vdash}} (q_0, x, X)$ ✓

"⇐" $(q_0, wx, AX) \vdash^* (q_0, x, X)\ \forall\, x \in \Sigma^*,\ X \in \Gamma^*$
⇒ Wegen $|w| > 1$, muss es $(q_0, BC) \in \delta(q_0, \varepsilon, A)$ geben mit $(q_0, wx, AX) \vdash (q_0, wx, BCX) \vdash^* (q_0, x, X)$
⇒ Es gibt eine Zerlegung $w = w_B w_C$ mit $(q_0, wx, BCX) \vdash^* (q_0, w_C x, CX) \vdash^* (q_0, x, X)$.
Da $B \in V$ nur über Regeln (iii) aus dem Stapel entfernt werden kann und die ε-Sonderregel gilt, ist $|w_B|, |w_C| \leqslant n$.
Laut Induktionsannahme folgt $B \Rightarrow_G^* w_B$, $C \Rightarrow_G^* w_C$
⇒ $A = BC \Rightarrow_G^* w_B w_C = w$ ✓

Dann folgt:
$w \in \mathcal{L}(G) \Leftrightarrow S \Rightarrow_G^* w$
⇔ ∃ Konfigurationsfolge in K mit
$(q_0, w, \#) \overset{\varepsilon}{\underset{\text{(i)}}{\vdash}} (q_0, w, S\$) \vdash^* (q_0, \varepsilon, \$) \overset{\varepsilon}{\underset{\text{(iv)}}{\vdash}} (q_1, \varepsilon, \varepsilon)$
⇔ $w \in \mathcal{L}(K)$ ✓

⋆ Beweis (II) ⇒ (III): Zu jedem NKA K gibt es einen NKA K' mit $\mathcal{L}(K) = \mathcal{L}_\varepsilon(K')$

Sei $K = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$. Wir konstruieren $K' = (Q', \Sigma, \Gamma', \delta', q_0', \#, \cdot)$ mit

$Q' = Q \cup \{q_0', q_\varepsilon\}$ mit $q_0', q_\varepsilon \notin Q$: neuer Zustand
$\Gamma' = \Gamma \cup \{\$\}$ mit $\$ \notin \Gamma$: neues Symbol

δ': (i) $\delta'(q_0', \varepsilon, \#) = \{(q_0, \#\$)\}$
(ii) $\delta'(q, \sigma, \gamma) = \delta(q, \sigma, \gamma)$ für $\sigma \in \Sigma,\ \gamma \in \Gamma,\ q \in Q$
(ii') $\delta'(q, \varepsilon, \gamma) = \delta(q, \varepsilon, \gamma)$ für $\gamma \in \Gamma,\ q \in Q \backslash F$
(iii) $\delta'(q, \varepsilon, \gamma) = \delta(q, \varepsilon, \gamma) \cup \{(q_\varepsilon, \varepsilon)\}$ für $\gamma \in \Gamma,\ q \in F$
(iv) $\delta'(q_\varepsilon, \varepsilon, \gamma) = \{(q_\varepsilon, \varepsilon)\}$ für $\gamma \in \Gamma'$

Erläuterungen

- Das neue Symbol \$ wird wieder ganz zu Anfang am Grund des Kellers abgelegt und blockiert dort. Das verhindert, dass K' "aus Versehen" das falsche Wort mit leerem Keller akzeptiert. Deshalb wird auch der neue Anfangszustand q_0' benötigt und die neue Konfigurationsrelation $(q_0, w, \#) \vdash_{K'} (q_0, w, \#\$)$
- Abgesehen von (i,iii,iv) hat K' dieselben Übergänge wie K, d.h. $(q_0, w, \#) \vdash_K^* (q, \tilde{w}, \xi) \Leftrightarrow (q_0, w, \#\$) \vdash_{K'}^* (q, \tilde{w}, \$)$ für alle $q \in Q,\ \xi \in \Gamma^*,\ w, \tilde{w} \in \Sigma^*$
- Sobald K' eine Konfiguration $(q_F, \varepsilon, \$)$ erreicht mit $q_F \in F$, kann ein Übergang $(q_F, \varepsilon, \xi\$) \vdash_{K'} (q_\varepsilon, \varepsilon, \xi\$)$ stattfinden. Von diesem aus kann der Keller *via* $(q_\varepsilon, \varepsilon, \xi\$) \vdash_{K'}^* (q_\varepsilon, \varepsilon, \varepsilon)$ entleert werden.

K' "rät" also, wann das Eingabewort w zu Ende ist, und wechselt dann ggf. in den Zustand q_ε (falls ein akzeptierender Zustand von K erreicht wurde). Falls K' sich "irrt" und zu früh nach q_ε wechselt, entleert sich der Keller zwar auch. Da aber die Eingabe noch nicht abgeschlossen ist (also noch ein Teilwort von w übrig ist), wird dieser Lauf trotzdem verworfen.

Bemerkung – Wir haben hier gezeigt: Zu jedem NKA K gibt es einen NKA K', so dass $\mathcal{L}_\varepsilon(K') = \mathcal{L}(K)$. Die umgekehrte Aussage (zu jedem NKA K gibt es einen NKA K' mit $\mathcal{L}(K') = \mathcal{L}_\varepsilon(K)$) beweisen wir indirekt mit dem Ringschluss (III) ⇒ (I) ⇒ (II). Ein direkter Beweis ist jedoch auch möglich. Er geht im Prinzip wie oben und ist sogar noch etwas einfacher (→ Übungsaufgabe)

⋆ Beweis (III) ⇒ (I): Zu jedem NKA K gibt es eine Typ 2-Grammatik G mit $\mathcal{L}(G) = \mathcal{L}_\varepsilon(K)$

Gegeben $K = (Q, \Sigma, \Gamma, \delta, q_0, \#, \cdot)$. Wir konstruieren $G = (V, \Sigma, P, S)$.

Erinnerung vorab: In 2.2.1.3 haben wir ein ähnliches Problem gelöst: Die Konstruktion eine Typ 3-Grammatik aus einem DEA.
Ansatz dort: Setze $V = Q$, konstruiere Regeln aus $\delta(q, \sigma) = q'$
(für alle $\delta(q, \sigma) = q'$ generiere $q \to \sigma q'$, für $q \in F$ auch $q \to \varepsilon$)

Hier: Der Ansatz $V = Q$ reicht nicht, denn $\delta(q, \sigma, \gamma)$ hängt auch von γ (Keller) ab. Die Variablen $A \in V$ müssen also auch Information

über den Kellerinhalt enthalten. Ideal wäre es, wenn sie den gesamten Kellerinhalt speichern könnten, aber da die Menge V aller Variablen endlich ist, geht das leider nicht.

Ansatz: Tripelkonstruktion: $V = \{[qXp]\}$ mit $p, q \in Q$, $x \in \Gamma$

$[qXp]$ charakterisiert (zum Teil) einen Zustand des NKAs über

$q \in Q$:	Aktueller Zustand des NKAs
$X \in \Gamma$:	Aktuell oberstes Kellersymbol
$p \in Q$:	Prognose für den Zustand, den der NKA haben wird, wenn der aktuell oberste Kellerplatz geräumt wurde

Konkret: (mit $\sigma \in (\Sigma \cup \{\varepsilon\})$)

(i) Für alle $(q', \varepsilon) \in \delta(q, \sigma, X)$ generiere die Regel $[qXq'] \to \sigma$

(ii) Für $(q', \gamma) \in \delta(q, \sigma, X)$ mit $\gamma \in \Gamma$ (Symbol γ ersetzt X) generiere eine Schar von Regeln $[qXp] \to \sigma[q'\gamma p] \; \forall \; p \in Q$

($\rightsquigarrow$ Analog 2.2.1.3: $q \to q'$, $X \to \gamma$, p zunächst offen. Nur Variablen mit korrekter Prognose für p können vollständig in Terminale aufgelöst werden. Anderenfalls sind sie "nutzlos" (siehe 2.3.1.2) und tragen nicht zu $\mathcal{L}(G)$ bei.)

(ii') Verallgemeinerung: Für $(q', \gamma_1 ... \gamma_k) \in \delta(q, \sigma, X)$ (mit $\gamma_i \in \Gamma$) generiere Regeln $[qXp] \to \sigma[q'\gamma_1 r_1][r_1\gamma_2 r_2]...[r_{k-1}\gamma_k p]$ für alle möglichen Kombinationen $(p, r_1 ... r_{k-1}) \in V^k$

(Anschaulich: $r_1 ... r_{k-1}$ sind geratene Zwischenzustände, die K hat, wenn $\gamma_1 ... \gamma_{k-1}$ gerade abgebaut wurden.)

(iii) Start: $S \to [q_0 \# p]$ für alle $p \in Q$

(p ist der geratene Endzustand nach Eingabe eines Wortes $w \in \mathcal{L}(K)$)

Wir zeigen nun: $\mathcal{L}(G) = \mathcal{L}_\varepsilon(K)$

bzw. die stärkere Aussage $\big((q, w, X) \vdash^* (p, \varepsilon, \varepsilon)\big) \Leftrightarrow \big([qXp] \Rightarrow_G^* w\big)$

(denn: Dann ist auch $\big((q_0, w, \#) \vdash^* (p, \varepsilon, \varepsilon)\big) \Leftrightarrow \big(S \Rightarrow_G [q_0 \# p] \Rightarrow_G^* w\big)$)

„$\Rightarrow$" Zeige: Aus $(q, w, X) \vdash^* (p, \varepsilon, \varepsilon)$ folgt $[qXp] \Rightarrow_G^* w$
über Induktion nach der Anzahl der Schritte im NKA K

Induktionsanfang: Ein Schritt

Sei $(q, w, X) \vdash (p, \varepsilon, \varepsilon) \;\Rightarrow\; w \in (\Sigma \cup \{\varepsilon\})$ und $\delta(q, w, X) \ni (p, \varepsilon)$
$\Rightarrow \; \exists$ Regel $[qXp] \to \sigma = w \;\Rightarrow\; [qXp] \Rightarrow_G w \quad \checkmark$

Induktionsschritt: Die Behauptung gelte für n Schritte.

Betrachte eine Sequenz von $(n+1)$ Konfigurationen

$$(q, w, X) \underset{\text{1. Schritt}}{\vdash} (p_0, \tilde{w}, \gamma_1 ... \gamma_k) \underset{\text{n Schritte}}{\vdash^*} (p, \varepsilon, \varepsilon) \qquad (k \geqslant 1)$$

Hier ist $w = \sigma\tilde{w}$ mit $\sigma \in (\Sigma \cup \{\varepsilon\})$.

(i) Im ersten Schritt wird σ eingelesen und das Kellersymbol X durch $\gamma_1 ... \gamma_k$ ersetzt. Da $(p_0, \tilde{w}, \gamma_1 ... \gamma_k) \vdash^* (p, \varepsilon, \varepsilon)$, werden in den folgenden n Schritten die $\gamma_1 ... \gamma_k$ wieder abgebaut, während Teilworte w_i von $\tilde{w}$ sukzessive eingelesen werden. In dem Schritt, in dem γ_i entfernt wird, gehe der Automat in den Zustand p_i über, konkret ist also $p_k = p$.

$\Rightarrow \exists$ Konfigurationsfolgen $(p_{i-1}, w_i, \gamma_i) \vdash^* (p_i, \varepsilon, \varepsilon)$
mit n oder weniger Schritten.
Laut Induktionsannahme folgt daraus $[p_{i-1}\gamma_i p_i] \Rightarrow_G^* w_i$
Weiterhin ist $w = \sigma w_1 ... w_k$.

(ii) Wegen $(q, w, X) \vdash (p_0, \gamma_1 ... \gamma_k)$ gilt $(p_0, \gamma_1 ... \gamma_k) \in \delta(q, \sigma, X)$
$\Rightarrow \exists$ Regeln $[qXp'] \to \sigma[p_0\gamma_1 p_1'][p_1'\gamma_2 p_2']...[p_{k-1}'\gamma_k p']$
für alle $(p', p_1'..p_{k-1}') \in V^k$ insbesondere für $p_j' = p_j$, $p' = p$.

Aus der Kombination von (i) und (ii) folgt:
$[qXp] \Rightarrow_G \sigma[p_0\gamma_1 p_1]...[p_{k-1}\gamma_k p] \vdash^* \sigma w_1 ... w_k = w$ ✓

„$\Leftarrow$ “ Zeige: Aus $[qXp] \Rightarrow_G^* w$ folgt $(q, w, X) \vdash^* (p, \varepsilon, \varepsilon)$
über Induktion nach der Anzahl der Ableitungsschritte in G

Induktionsanfang: Eine Ableitung
$([qXp] \Rightarrow_G w) \Rightarrow \exists$ Regel $[qXp] \to w$ und $w \in (\Sigma \cup \{\varepsilon\})$
$\Rightarrow (p, \varepsilon) \in \delta(q, w, X) \Rightarrow (q, w, X) \vdash (p, \varepsilon, \varepsilon)$ ✓

Induktionsschritt: Die Behauptung gelte für n Ableitungsschritte.
Wir betrachten eine Ableitung $[qXp] \Rightarrow_G^* w$ mit $n+1$ Schritten
und zerlegen Sie gemäß
$[qXp] \Rightarrow_G \sigma[p_0\gamma_1 p_1][p_1\gamma_2 p_2]...[p_{k-1}\gamma_k p_k] \Rightarrow_G^* w$ (mit $p_k = p$)
(Der erste Ableitungsschritt kann nur diese Form haben!)
$\Rightarrow (p_0, \gamma_1 ... \gamma_k) \in \delta(q, \sigma, X)$, d.h. $(q, w, X) \vdash (p_0, \tilde{w}, \gamma_1 ... \gamma_k)$
mit $\sigma \in \Sigma \cup \{\varepsilon\}$ und $\sigma\tilde{w} = w$
Wir betrachten die Zerlegung $\tilde{w} = w_1 ... w_k$ mit $[p_{i-1}\gamma_i p_i] \Rightarrow_G^* w_i$
Dann gilt: $[p_{i-1}\gamma_i p_i] \Rightarrow_G^*$ erfordert weniger als n Ableitungen
$\Rightarrow$ (Induktionsannahme) $(p_{i-1}, w_i, \gamma_i) \vdash^* (p_i, \varepsilon, \varepsilon)$
Kombiniere alles:
$(q_0, w, X) \vdash (p_0, w_1 ... w_k, \gamma_1 ... \gamma_k) \vdash^* (p_1, w_2 ... w_k, \gamma_2 ... \gamma_k)$
$\vdash^* (p_2, w_3 ... w_k, \gamma_3 ... \gamma_k) \vdash^* (p_{k-1}, w_k, \gamma_k) \vdash^* (p_k, \varepsilon, \varepsilon) = (p, \varepsilon, \varepsilon)$ ✓

2.3.4.4 Deterministische Kellerautomaten

In 2.3.4.3 wurde gezeigt, dass NKAs und kontextfreie Grammatiken äquivalent sind. Da sie nichtdeterministisch sind, eignen sie sich jedoch nicht wirklich als praktische Hilfsmittel zur Lösung des Wortproblems. Das kann allgemein nur mit dem CYK Algorithmus (2.3.2.2) in $\mathcal{O}(|w|^3)$ gelöst werden.
→ für Anwendungen (z.B. Parser in Compilern) ziemlich unpraktisch!

Ausweg: Beschränkung auf deterministische Kellerautomaten (DKAs) und Sprachen, die von diesen akzeptiert werden.

⋆ Definition: Ein DKA ist ein NKA $K = (Q, \Sigma, P, \delta, q_0, \#, F)$ mit
- $|\delta(q, \sigma, \gamma)| \leqslant 1$ für alle $q \in Q,\ \gamma \in \Gamma,\ \sigma \in (\Sigma \cup \{\varepsilon\})$
- $\delta(q, \varepsilon, \gamma) \neq \varnothing$ impliziert $\delta(q, \sigma, \gamma) = \varnothing$ für alle $\sigma \in \Sigma$

⋆ Eine Sprache L heißt deterministisch kontextfrei,
|| wenn es einen DKA gibt, der sie mit Endzuständen akzeptiert.

Bemerkungen:

- Grund für die Zulassung von ε-Übergängen: Ermöglicht z.B. das partielle Leeren des Kellers, ohne dass ein Zeichen eingelesen werden muss. Da es im Falle von $\delta(q, \varepsilon, \gamma) \neq \varnothing$ zum εÜbergang für dieses Paar (q, γ) keine Alternative gibt, bleibt der DKA deterministisch.
- Man kann aus einem DKA, der mit leerem Keller akzeptiert, immer einen DKA konstruieren, der mit Endzuständen akzeptiert. Deshalb fordern wir $L = \mathcal{L}(K)$ und nicht $L = \mathcal{L}_\varepsilon(K)$.
- In DKAs lsst sich das Wortproblem in der Zeit $\mathcal{O}(|w|)$ lösen.

⋆ Satz:
- Jede reguläre Sprache L ist deterministisch kontextfrei (als DKA kann einfach der DEA zu L benutzt werden.)
- Jede deterministisch kontextfreie Sprache ist kontextfrei (klar).
- Umgekehrt ist nicht jede kontextfreie Sprache deterministisch kontextfrei. (Gegenbeispiel ohne Beweis: Palyndromsprache)

NB: Deterministisch kontextfreie Sprachen sind unter Komplementbildung abgeschlossen (man ersetzt im DKA einfach F durch $\overline{F}$. Für die Klasse der allgemeinen kontextfreie Sprachen gilt das nicht.

⋆ Satz: Jede deterministisch kontextfreie Sprache ist eindeutig
(ohne Beweis)

2.3.5 Zusammenfassung und Ausblick

In dem Kapitel 2.3 wurden kontextfreie Sprachen behandelt. Sie sind sehr viel mächtiger als die regulären Sprachen aus 2.2. Wichtige Eigenschaften sind:

⋆ Charakterisierung kontextfreier Sprachen

Sei $L \subseteq \Sigma^*$ eine Sprache. Folgende Aussagen sind äquivalent:

(I) $L = \mathcal{L}(G)$ für eine kontextfreie Grammatik G
(II) $L = \mathcal{L}(K)$ für einen NKA K
(III) $L = \mathcal{L}_\varepsilon(K')$ für einen NKA K'

⋆ Abschlusseigenschaften: Kontextfreie Sprachen sind

- abgeschlossen und Vereinigung, Konkatenation, Kleene-Abschluss
- nicht abgeschlossen unter Durchschnitt, Komplement

⋆ Praktische Aspekte:

- Konzept der Ableitungsbäume
- Chomsky-Normalform und CYK-Algorighmus
  $\rightsquigarrow$ Das Wortproblem kann in der Zeit $\mathcal{O}(|w|^3)$ gelöst werden.
- Pumping-Lemma: Kriterium, ob eine Sprache kontextfrei sein könnte

⋆ Weiterführende Themen

- Deterministisch kontextfreie Sprachen
  Unterklasse der kontextfreien Sprachen
  Lösung des Wortproblems ist möglich in $\mathcal{O}(|w|)$
- Was passiert eigentlich, wenn man zwei Keller zulässt?
  $\rightsquigarrow$ Diese Klasse von Automaten wird Turing-mächtig:
  Man kann Typ 0-Sprachen damit erkennen!
- Zum Konzept der Turing-Mächtigkeit und zu Turing-Maschinen kommen wir im nächsten Kapitel.

Kapitel 3

Berechenbarkeit

Erinnerung an Kapitel 2:

Wir können nun Algorithmen für einige typische Fragestellungen in formalen Sprachen konstruieren (vgl. 3.1.2.3).

Gegeben seien zwei Grammatiken G, G' und ein Wort w

Problem	regulär	kontextfrei	Typ 0
Ist $w \in \mathcal{L}(G)$?	✓ (DEA)	✓ (CNF & CYK)	?
Ist $\mathcal{L}(G) = \varnothing$?	✓ (DEA)†	✓ (Markierungsalgorithmus)*	?
Ist $\lvert\mathcal{L}(G)\rvert = \infty$?	✓ (DEA)††	✓ (CNF & CYK)**	?
Ist $\mathcal{L}(G) = \mathcal{L}(G')$?	✓ (DEAs)†††	?	?

(Mögliche Algorithmen:

†: Erreichbarkeitsanalyse für $q \in F$ von q_0, z.B. über Markierungsalgorithmus

†† Identifiziere Zyklen auf Pfaden von q_0 nach $q \in F$

††† z.B. über Minimierung der DEAs und Vergleich der resultierenden Nerode-Automaten. Schneller geht es mit dem sogenannten Produktautomaten, siehe Asteroth/Baier Kapitel 6.2.2.

*: Bestimme via Markierungsalgorithmus (siehe 2.3.1.2), ob S nutzlos ist,

** $\lvert\mathcal{L}(G)\rvert = \infty$ genau dann, wenn es Worte gibt, die so lang sind, dass sie nach dem Pumping Lemma aufgepumpt werden können. In CNF Grammatiken kann man nach 2.3.3.2 die Pumpingzahl $n = 2^{\lvert V\rvert+1}$ wählen. Worte w, die größer als $2n$ sind, können auf eine Länge $n \leqslant \lvert w\rvert < 2n$ abgepumpt werden. Also genügt es, mit dem CYK Algorithmus alle Worte der Länge $n \leqslant \lvert w\rvert < 2n$ zu testen. Das kann zwar dauern, aber es ist machbar.)

Frage: Kann man alle Felder dieser Tabelle befüllen?

Oder fundamentaler: Kann man alle "sinnvollen" Fragen mit Hilfe von Algorithmen / eines Computerprogramms beantworten?
→ Problem der Berechenbarkeit und Entscheidbarkeit

Hinweis, warum das eventuell problematisch sein könnte:
Die Menge aller Funktionen $\{0,1\}^* \to \{0,1\}$ hat die Mächtigkeit $\lvert 2^{\mathbb{N}}\rvert$
Aber: Programme haben nur abzählbar viele Zeichen!

Plan für dieses Kapitel

3.1 Erst einmal müssen wir den Begriff der Berechenbarkeit spezifizieren. Dafür führen wir Rechnermodelle ein, allen voran den Klassiker, die Turingmaschine, aber auch solche, die unseren modernen Vorstellungen von Rechnern und Programmen näher kommen. Wir werden feststellen, dass alle unsere Rechnermodelle äquivalent sind, d.h. das gleiche können.

3.2 Auf dieser Basis können wir diskutieren, was Entscheidbarkeit ist und ob alle Probleme entschieden werden können. Der Klassiker ist hier das sogenannte Halteproblem.

3.1 Konzept der Berechenbarkeit

3.1.1 Intuitive Berechenbarkeit

Wir haben alle eine Vorstellung davon, was "berechenbar" heißt.

Eine mögliche Formulierung könnte lauten:

Gegeben eine partielle Funktion $f : \mathbb{N} \to \Lambda$

Die Funktion heißt berechenbar, wenn es einen Algorithmus gibt, der $f(n)$ *ausgibt, falls* $f(n)$ *definiert ist, und anderenfalls verwirft oder endlos weiterläuft (das ist auch erlaubt).*

Beispiele für Funktionen, die nach diesem Verständnis berechenbar sein sollten:

- $f(n) = n$te Nachkommastelle von π
- Die Ackermannfunktion (Ackermann 1926, Péter 1935) $\mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$
  $\text{ack}(0, m) = m + 1$
  $\text{ack}(n, 0) = \text{ack}(n - 1, 1)$ für $n \geqslant 1$
  $\text{ack}(n, m) = \text{ack}(n - 1,\text{ack}(n, m - 1))$ für $n, m \geqslant 1$

  Die Berechnung ist in der Praxis schwierig für größere n, m, weil die Funktion *extrem schnell* anwächst, deshalb lässt sich die Dauer der Berechnung im Voraus nicht abschätzen.
  Aber: Berechenbar ist diese Funktion
- $f_\perp(n) = \perp$ (die überall undefinierte Funktion)

Intuitiv ist daher "klar", was mit "berechenbar" gemeint ist. Um mit diesem Begriff sauber arbeiten zu können, ist trotzdem eine Formalisierung notwendig. $\rightsquigarrow$ Rechnermodelle

Rechnermodelle sollten maximal "mächtig"sein, d.h. alle denkbaren Algorithmen sollten implementierbar sein.
$\rightsquigarrow$ Sie müssen klarerweise mächtiger sein als DEAs oder NKAs.

3.1.2 Turingmaschinen (TM)

Die Turingmaschine (Alan Turing, 1936) ist das ultimative abstrakte Rechnermodell und ein Standardformalismus für Berechenbarkeit und Komplexität

Prinzip (Einfachste Version, Einband-TM)

Ähnlich den Automaten aus Kapitel 2, aber ohne Keller. Dafür kann das Eingabeband in beide Richtungen gelesen und auch überschrieben werden. Der Speicherplatz ist in beide Richtungen unendlich. Die Menge der Zustände und das Bandalphabet sind nach wie vor endlich, ebenso das "Programm" (Einträge in der Übergangsfunktion).

$\rightsquigarrow$ Simpel, aber wie wir sehen werden, überaus mächtig!

Hintergrund (Turings Argumentation)

Modelliert die Art, wie ein intelligentes Wesen mit "Papier und Bleistift" rechnet. Der Automat entspricht sozusagen dem Zustand des Gehirns inklusive seinem endlichen Gedächtnis. Das Band ist das "Papier".

3.1.2.1 Deterministische Turingmaschinen (DTMs)

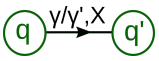
⋆ Definition: eine (Einband-)DTM ist ein 7-Tupel $T = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ mit

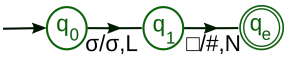
- Q: Endliche Menge von Zuständen (des Automaten)
- Σ: Endliches Eingabealphabet
- $\Gamma \supseteq \Sigma$: Endliches Bandalphabet
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$: Übergangsfunktion
- $q_0 \in Q$: Startzustand
- $\square \in \Gamma \backslash \Sigma$: Blanksymbol
- $F \subseteq Q$: Menge der akzeptierenden oder Endzustände

⋆ Wirkungsweise der TM (vgl. Automaten in Kapitel 2)

- Beginn: Startzustand q_0, das Eingabeband enthält das Eingabewort w (von links nach rechts), ansonsten enthält das Band nur $\square$s.
  Der "Kopf" der TM steht auf dem ersten Zeichen von w.
- Leseschritt:
  - Der Kopf liest das Bandzeichen γ darunter.
  - Die TM berechnet $\delta(q, \gamma) = (q', \gamma', X)$ mit $X \in \{L, R, N\}$
  - Die TM überschreibt γ mit γ' und wechselt vom Zustand q nach q'.
    Der Kopf bewegt sich gemäß dem Wert von X:
    L : 1 nach links, R : 1 nach rechts, N : gar nicht
- Abschluss (Zwei Möglichkeiten)
  - $q \in F$: Ausgabe: Bandinhalt zwischen dem Kopf (erstes Zeichen) und dem ersten $\square$.
  - $\delta(q, \gamma) = \perp$: Ausgabe: Verwerfend ($\perp$)

⋆ Graphische Darstellung (analog DEAs/NEAs in Kapitel 2.2

Zustände: ○ oder ◎ (akzeptierend)
Übergangsfunktion: $\delta(q,\gamma) = (q',\gamma',X) \;\triangleq$ 

Beispiel: TM, die einen String w durch $\#w$ ersetzt:
$\delta(q_0,\sigma) = (q_1,\sigma,L)\ \forall\sigma\in\Sigma,\ \delta(q_1,\square) = (q_e,\#,N)$ 

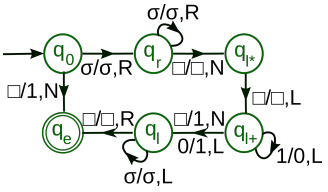
⋆ Konfigurationen und Konfigurationswechsel
Formalisierung der obigen Vorschrift (analog 2.3.4.2)

- Eine Konfiguration einer TM T ist ein Tripel $(\alpha,q,\beta)\in\Gamma^*\times Q\times\Gamma^*$
  mit $q\in Q$: aktueller Zustand der TM
  $\alpha\in\Gamma^*$ aktueller Bandinhalt links vom Kopf ohne äußere □s
  $\beta\in\Gamma^*$ aktueller Bandinhalt rechts vom Kopf ohne äußere □s, beginnend mit dem Zeichen unter dem Kopf
  Falls α oder β nur □s enthält, dann notiere $\alpha,\beta = \square$
  Alternative Notation $(\alpha,q,\beta) \triangleq \alpha q\beta$ (ganz ohne äußere □s)
  Speziell Startkonfiguration: $\kappa_0 = (\square,q_0,w) \triangleq q_0w$
  Die Menge aller Konfigurationen ist $\mathrm{Conf}(T)\subseteq\Gamma^*\times Q\times\Gamma^*$

- Die Konfigurationsrelation $\vdash\subseteq \mathrm{Conf}(T)\times\mathrm{Conf}(T)$
  $\kappa\vdash\kappa'$ verknüpft κ mit möglichen Nachfolgekonfigurationen κ':
  Sei $\kappa = (\nu\xi,q,\gamma w)$ mit $\xi,\gamma\in\Gamma,\ \nu,w\in\Gamma^*$.
  $$\text{Dann ist } \kappa\vdash \begin{cases} (\nu\xi,q',\gamma' w) & \text{falls } \delta(q,\gamma) = (q',\gamma',N) \\ (\nu,q',\xi\gamma' w) & \text{falls } \delta(q,\gamma) = (q',\gamma',L) \\ (\nu\xi\gamma',q',w) & \text{falls } \delta(q,\gamma) = (q',\gamma',R) \end{cases}$$
  $\vdash$ definiert wie gewohnt auch eine reflexive transitive Hülle $\vdash^*$
  (mit $\kappa\vdash^*\kappa,\ \kappa\vdash^*\kappa' \Leftrightarrow \exists\kappa_1...\kappa_k$ mit $\kappa\vdash\kappa_1...\kappa_k\vdash\kappa'$)

- Ausgabe (formalisiert): bei Eingabe w ist die Ausgabe der TM das Wort $w'\in(\Gamma\backslash\{\square\})^*$, falls $q_0w\vdash^*\alpha q_e w'\square\beta$ mit $q_e\in F,\ \alpha,\beta\in\Gamma^*$

- Beispiel: TM, die 1 zu einer Binärzahl w addiert.
  $\Sigma = \{0,1\},\ \Gamma = \Sigma\cup\{\square\},\ \sigma$ steht für alle $\sigma\in\Sigma$
  Eingabe: $w = 1011\ \Rightarrow\ \kappa_0 = q_01011$
  $q_01011\vdash 1q_r011\vdash 10q_r11\vdash 101q_r1\vdash 1011q_r$
  $\vdash 1011q_{l*}\vdash 101q_{l+}1\vdash 10q_{l+}10\vdash 1q_{l+}000$
  $\vdash q_l1100\vdash q_l\square1100\vdash q_e1100$
  Eingabe: $w = 11,\ \Rightarrow\ \kappa_0 = q_011$
  $q_011\vdash 1q_r1\vdash 11q_r\vdash 11q_{l*}\vdash 1q_{l+}1\vdash q_{l+}10\vdash q_{l+}\square00$
  $\vdash q_l100\vdash q_l\square100\vdash q_e100$



⋆ Berechnete Funktion einer TM

Die Ausgabe einer TM T bei Eingabe w definiert eine partielle Funktion

$$T(w) = \begin{cases} w' & \text{falls } T \text{ bei Eingabe } w \text{ das Wort } w' \text{ ausgibt} \\ \bot & \text{falls es keine Ausgabe gibt} \end{cases}$$

Alternativ werden wir statt $T(w)$ manchmal auch $f_T(w)$ schreiben.

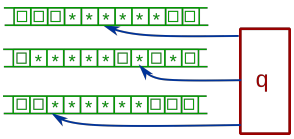
⋆ Sprache einer TM (Zusammenhang mit formalen Sprachen)

Die von einer Turingmaschine T akzeptierte Sprache ist
$\mathcal{L}(T) = \{w \in \Sigma^* : q_0 w \vdash^* \alpha q_e \beta \text{ mit } q_e \in F,\ \alpha, \beta \in \Gamma^*\}$

3.1.2.2 Erweiterung: Mehrband-Turingmaschinen

Mehrband-DTMs sind Varianten der DTM mit mehreren Bändern und mehreren Köpfen. Wie wir zeigen werden, sind sie der Einband-DTM gleichmächtig, aber etwas leichter zu programmieren und damit ein günstiger Ausgangspunkt für Vergleiche zwischen Rechnermodellen.

⋆ Definition: Eine k-Band-TM hat k Bänder, k zugehörige Köpfe und daher eine Übergangsfunktion $\delta : Q \times \Gamma^k \to Q \times (\Gamma \times \{L, R, N\})^k$.
Alle übrigen Definitionen (Konfigurationen, Konfigurationswechsel etc.) sind wie gehabt. Eingabe- und Ausgabeband müssen explizit spezifiziert werden.



⋆ Satz: Jede k-Band-DTM kann durch eine Einband-DTM simuliert werden.

Beweisidee: Wir konzipieren die Einband-DTM so, dass sie alle relevante Information über den aktuellen Zustand der Mehrband-DTM enthält. Um Mehrband-Übergänge nachzustellen, muss die Einband-DTM Zwischenschritte durchführen und sich kurzzeitig Informationen merken, z.B. über eingelesene Zeichen γ_k. Das ist aber mit einem endlichen "Gedächtnis" (einer endlichen Zustandsmenge) möglich.

Konkretes Verfahren: Zur Modellierung der Mehrband-DTM verwenden wir eine Einband-DTM mit einem $2k$-spurigen Band. Dazu erweitern wir das Bandalphabet so, dass $(2k)$-Tupel $(x_1, ...x_{2k}) \in (\Gamma \cup \{*\})^{2k}$ durch ein Zeichen $X \in \Gamma'$ dargestellt werden können. Die Information der Mehrband-DTM wird auf der mehrspurigen Einband-DTM folgendermaßen abgelegt:

	·	·	·	·	·	Inhalt Band 1
	□	□	*	□	□	Position Kopf 1
	·	·	·	·	·	Inhalt Band 2
	*	□	□	□	□	Position Kopf 2
						etc.

Die Spur $2j$–1 enthält den Inhalt des j-ten Bandes
Die Spur $2j$ markiert die Position des Kopfes j

Die Einband-DTM geht folgendermaßen vor:

1) Sie besucht nacheinander alle k Kopfspuren und "merkt" sich das darüberliegende Symbol γ_k, indem sie einen entsprechenden Zustand $\bar{q} \triangleq [q, \{\gamma_j\}]$ einnimmt (Notation: $\{\gamma_j\} = \gamma_1, ...\gamma_k$).
Das erfordert zwar eine gewaltige Erweiterung der Zustandsmenge, aber die Menge der Zustände bleibt endlich.

2) Sie vollzieht einen Übergang von $\bar{q}$ nach $\bar{q}' \triangleq [q', \{(\gamma'_j, X'_j)\}]$ entsprechend der Übergangsfunktion der Mehrband-DTM.

3) Sie besucht wieder alle Spuren und implementiert die gewünschten Änderungen: Überschreiben von $\gamma_i \to \gamma'_i$ und Verschiebung der Markierungssymbole in den Kopfspuren.

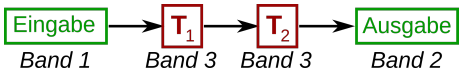
⋆ Anwendungen: Mit Mehrband-DTMs können Konstruktionen entwickelt werden, die die Programmierung von DTMs sehr erleichtern.

Im Folgenden nennen wir ein paar Beispiele. Das Eingabeband ist stets Band 1, das Ausgabeband Band 2, die restlichen Bänder werden zum Speichern, für Kopiervorgänge, und für Nebenrechnungen verwendet.

- Hintereinanderschalten von TMs

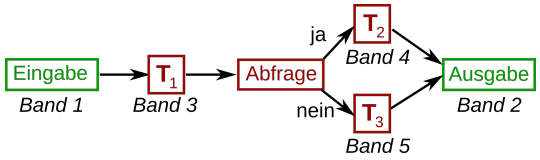
  Gegeben zwei TMs T_1, T_2 mit $T_j = (Q_j, \Sigma, \Gamma, \delta_j, q_{0j}, \square, F_j)$

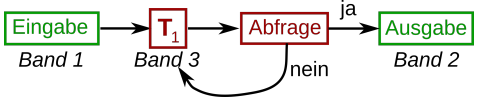
  Wir konstruieren $T = T_1; T_2 = (Q_1 \cup Q_2, \Sigma, \Gamma, \delta, q_{01}, \square, F_2)$

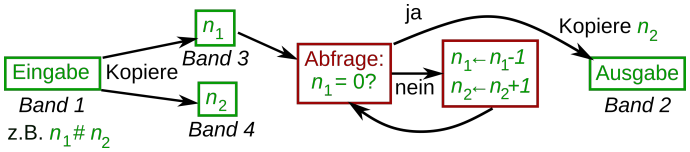
  über 

  – Eingabe auf Band 1
  – Kopiere die Eingabe auf Band 3, lasse T_1 laufen.
  – Erfolgreicher Abschluss liefert $\alpha_3 q_e w' \square \beta_3$ auf Band 3 mit $q_e \in F_1$.
    Lösche α_3 und β_3, lasse T_2 laufen.
  – Bei erfolgreichem Abschluss kopiere das Ergebnis auf Band 2

  $\rightsquigarrow$ Die Ausgabe ist $T(w) = T_2(T_1(w))$

- Verzweigungen 

- Schleifen 

- Arithmetische Operationen, z.B. Addieren, $n = n_1 + n_2$ 

  (NB: Sehr ineffizientes Programm. Eine effizientere Programmierung, z.B. nach den Rechenregeln, die man in der Grundschule lernt, ist natürlich auch erlaubt – Übungsaufgabe!)

3.1.2.3 Nichtdeterministische Turingmaschinen (NTMs)

Analog zu den NEAs und NKAs in Kapitel 2 kann man auch NTMs definieren.

⋆ Definition: Eine NTM ist ein 7-Tupel $T = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$
mit Komponenten $Q, \Sigma, \Gamma, q_0, \square, F$ wie beim DTM
und einer totalen Übergangsfunktion $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R,N\}}$

⋆ Funktionsweise: Wie bei anderen nichtdeterministischen Automaten.
Die NTM wählt "zufällig" die Folgekonfiguration von κ aus einer Menge möglicher Folgekonfigurationen. Die Konfigurationsrelation $\vdash$ verknüpft κ mit allen möglichen Nachfolgekonfigurationen.

⋆ Bedeutung (unter anderem): Intuitiver Sprachakzeptor für Typ-0 Sprachen

Definition: $\mathcal{L}(T) = \{w \in \Sigma^* : q_0 w \vdash^* \alpha q_e \beta \text{ für ein } q_e \in F\}$

Satz: Sei L eine Sprache. Dann sind äquivalent:

(I) $L = \mathcal{L}(T)$ für eine NTM T

(II) L ist eine Sprache vom Typ 0.

Beweisskizze

- Konstruiere eine NTM aus der Grammatik mit $\Gamma = V \cup \Sigma \cup \{\square\}$
  Für alle Regeln $l \to r$ führt die NTM folgende Schritte durch
  - Sie sucht Fragmente r
  - Falls sie ein Fragment r findet, führt sie es nichtdeterministisch über in l. (In der Regel muss sie dafür die Bandinhalte rechts und/oder links von r über Kopiervorgänge verschieben.)
  - Falls $q_0 w \vdash^* q_f S$ für ein q_f, geht sie nichtdeterministisch über in $q_e S$ mit $q_e \in F$.
- Konstruiere eine Typ-0 Grammatik aus einer NTM mit Variablen $q, S, A, \square$ und $\binom{\gamma}{\gamma'}$, d.h. $V = (Q \cup \{S, A, \square\}) \cup (\Gamma \times \Gamma)$,
  (i) Konstruiere Platzhalter $\binom{\square}{\square}...\binom{\square}{\square} q_0 \binom{\sigma_1}{\sigma_1}...\binom{\sigma_n}{\sigma_n}\binom{\square}{\square}...\binom{\square}{\square}$ für alle möglichen Worte $w = \sigma_1...\sigma_n \in \Sigma^*$
  Regeln: $S \to q_0 A,\ A \to A\binom{\square}{\square} | \varepsilon | \binom{\sigma}{\sigma} A \,\forall\, \sigma \in \Sigma,\ q_0 \to \binom{\square}{\square} q_0$
  (ii) Teste anhand der oberen Einträge γ in $\binom{\gamma}{\tilde{\gamma}}$, ob es Konfigurationsübergänge $q_0 w \vdash^* \alpha q_e \beta$ mit $q_e \in F$ gibt. Die unteren Einträge bleiben unangetastet und "merken" sich das Ausgangswort w.
  Regeln: Für alle $\tilde{\gamma}, \xi, \tilde{\xi} \in \Gamma$ ergänze:
  für $\delta(q,\gamma) \ni (q', \gamma', N)$ Regel $\quad q\binom{\gamma}{\tilde{\gamma}} \to q'\binom{\gamma'}{\tilde{\gamma}}$
  $\delta(q,\gamma) \ni (q', \gamma', R)$ Regel $\quad q\binom{\gamma}{\tilde{\gamma}} \to \binom{\gamma'}{\tilde{\gamma}} q'$
  $\delta(q,\gamma) \ni (q', \gamma', L)$ Regeln $\quad \binom{\xi}{\tilde{\xi}} q \binom{\gamma}{\tilde{\gamma}} \to q' \binom{\xi}{\tilde{\xi}} \binom{\gamma'}{\tilde{\gamma}}$
  (iii) Ersetze Variablen $\binom{\gamma}{\tilde{\gamma}}$ durch Terminale $\tilde{\gamma}$ und eliminiere $\square$s. Falls (ii) erfolgreich war, eliminiere auch q_e. Nur in diesem Fall können alle Variablen aufgelöst werden und das Wort w wird erzeugt. Anderenfalls kann es nicht erzeugt werden.
  Regeln: Für $\tilde{\gamma} \in \Sigma \cup \{\square\},\ q_e \in F$
  $\binom{\gamma}{\tilde{\gamma}} \to \tilde{\gamma},\ q_e \to \varepsilon,\ \square \to \varepsilon$

⋆ Satz: Zu jeder NTM T gibt es eine DTM T', die die gleiche Sprache akzeptiert: $\mathcal{L}(T) = \mathcal{L}(T')$.

Beweisskizze: Die DTM simuliert eine Breitensuche durch alle möglichen Nachfolgekonfigurationen der NTM (Asteroth/Baier S. 125)

Folgerung: NTMs und DTMs sind gleichmächtig!

Anmerkung: NTMs sind Kandidaten als Rechnermodelle für Quantencomputer. Quantencomputer wären damit zwar schneller, aber nicht mächtiger als klassische Computer.

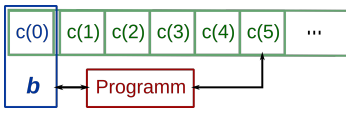
3.1.3 Alternative Rechnermodelle

In Abschnitt 3.1.2 haben wir Turingmaschinen eingeführt als einfaches und sehr mächtiges, aber auch ziemlich abstraktes Rechnermodell.

Frage: Wie verhalten sich echte Rechner und Programme zu TMs?
Dazu sollen jetzt weitere, realistischere Rechnermodelle diskutiert werden.

3.1.3.1 Registermaschinen (RMs)

Die Registermaschine ist ein abstraktes Rechnermodell, das jedoch relativ nahe an echten Assemblersprachen (für ein-Adress-Maschinen) ist. Die Registermaschine arbeitet mit natürlichen Zahlen. Ihre Komponenten sind:



- Ein endliches *Programm* mit Befehlen $\{P(1), ..., P(N)\}$
- Der *Befehlszähler* b, der festlegt, welcher Befehl als nächstes durchgeführt werden soll
- Ein unendlicher *Datenspeicher* mit Registerzellen $\{c(1), c(2), ...\}$
- Der *Akkumulator* $c(0)$, eine spezielle Registerzelle, die an allen Berechnungen teilnimmt und in der Zwischenergebnisse gespeichert werden.

Die Kontrolleinheiten (Programm und Befehlszähler) sind endlich. Der Speicher ist unendlich. Das gilt auch für den Akkumulator, der zwar nur eine einzige Zahl speichern kann – diese kann aber beliebig groß sein.

⋆ Befehlssatz (mögliche Programmbefehle) mit Notation $x = \#k, j$ oder $*j$
und $v(\#k) := k$ (Zahl)
$v(j) := c(j)$ (Inhalt der Registerzelle j)
$v(*j) := c(c(j))$ (Indirekte Addressierung. $c(j) = 0$ führt zum Abbruch!)

		Wirkung auf Akkumulator/Speicher	Wirkung auf Steuerwerk	
Grund-Befehle	`LOAD` x	$c(0) \leftarrow v(x)$	$b \leftarrow b{+}1$	(i)
	`STORE` j	$c(j) \leftarrow c(0)$	$b \leftarrow b{+}1$	(iia)
	`STORE` $*j$	$c(c(j)) \leftarrow c(0)$	$b \leftarrow b{+}1$	(iib)
Arithmet. Befehle	`ADD` x	$c(0) \leftarrow c(0) + v(x)$	$b \leftarrow b{+}1$	
	`SUB` x	$c(0) \leftarrow \max(0, c(0){-}v(x))$	$b \leftarrow b{+}1$	
	`MULT` x	$c(0) \leftarrow c(0) \cdot v(x)$	$b \leftarrow b{+}1$	
	`DIV` x	$c(0) \leftarrow [c(0)/v(x)]$	$b \leftarrow b{+}1$	(iii)
Sprung-Befehle	`GOTO` j		$b \leftarrow j$	(iva)
	`JZERO` j		$b \leftarrow j$: $c(0){=}0$ $b \leftarrow b{+}1$: sonst	(ivb)
	`END`		$b \leftarrow \infty$	(Ende)

Kommentare:

(i) Lädt $v(x)$ in den Akkumulator.
(iia) Speichert den Inhalt des Akkumulators in der Registerzelle $c(j)$.
(iia) Indirekte Addressierung. Abbruch, falls $c(j) = 0$.
(iii) Ganzzahlige Division ohne Rest. Abbruch, falls $v(x) = 0$.
(iva) Unbedingter Sprung.
(ivb) Bedingter Sprung (Verzweigung).

⋆ Funktionsweise der Registermaschine und berechnete Funktion

- Eingabe: Einträge $(n_1, ...n_k)$ in den Registerzellen $c(1)...c(k)$
- Die Befehle $P(b)$ werden der Reihe nach abgearbeitet, bis das Programm endet oder abbricht
- Ausgabe: Eintrag $c(i)$ in einem festgelegten Ausgaberegister i

Eine Registermaschine R definiert eine partielle Funktion $f_R : \mathbb{N}_0^k \to \mathbb{N}_0$
$f_R(n_1...n_k) = n$, wobei $n_1...n_k$ die Eingabe ist, n die Ausgabe
$f_R(n_1...n_k) = \perp$, falls das Programm abbricht oder endlos weiterläuft.

⋆ Satz: Zu jeder RM R gibt es eine DTM T, die die gleiche Funktion berechnet, d.h. $f_T = f_R$. Umgekehrt gibt es zu jeder DTM T eine RM R, die die gleiche Funktion berechnet.

Beweisskizze RM → DTM: Simuliere die RM durch eine Mehrband-TM.

Die Befehle sind in den Zuständen enthalten: $Q = \underbrace{Q_0}_{\text{Initialisierung}} \cup \underbrace{Q_1 \cup ...Q_N}_{\text{Befehle}} \cup \underbrace{Q_{N+1}...}_{\text{Kopieren etc.}}$

Band 1: Eingabe in Binärform
Band 2: Ausgabe
Band 3: Belegung $c(0), c(1)...$ in Binärform, z.B.
$\#\#\#0\#\text{bin}(c(0))\#\#\text{bin}(i_1)\#\text{bin}(c(i_1))\#\#...$
wobei die i_k Registernummern von Registern sind, auf die schon einmal zugegriffen wurde
Band 4ff: Nebenrechnungen

Vorgehen der DTM

Befehle im Zustandssatz Q_b werden abgearbeitet, indem
– Auf Band 3 j bzw. $*j$ aufgesucht und auf Hilfsbänder kopiert wird.
– Operationen auf den Hilfsbändern durchgeführt werden, z.B. mit Hilfe der Konstruktionen aus 3.1.2.2
– Ergebnisse auf Band 3 zurück übertragen werden.

Speziell Sprungbefehle: Direkte Übergänge $Q_b \to Q_j$

Beweisskizze DTM → RM: Simuliere die DTM durch eine RM.

Wir ordnen jedem Element $a \in \Gamma$ eine Zahl $n_a \leqslant d := |\Gamma|$ zu und kodieren Konfigurationen $\kappa = \alpha q \beta$ mit $\alpha = a_m...a_1, \beta = b_1...b_n$ durch "Kellerregister" für α, β, z.B. $\alpha = \sum_j n_{a_j} d^j$, $\beta = \sum_j n_{b_j} d^j$,

⇝ Man kann Kelleroperationen durch RM-Funktionen darstellen, z.B.

$\text{TOP}(\alpha) = \alpha - d \cdot [\alpha/d]$: Liest das oberste Zeichen
$\text{POP}(\alpha) = [\alpha/d]$: Entfernt das oberste Zeichen
$\text{PUSH}(x_0, \alpha) = x_0 + \alpha \cdot d$: Fügt das Zeichen x_0 dazu

Damit können Übergangsfunktionen der TM in RM-Befehle übersetzt werden, z.B. $\delta(q, \gamma) = \delta(q', \gamma', L)$ mit dem RM Programm rechts: (IF-Befehl wird mit JZERO realisiert)

```
y ← TOP(β)
IF (y = γ) THEN
    z ← TOP(α)
    α ← POP(α)
    β ← POP(β)
    β ← PUSH(γ′, β)
    β ← PUSH(z, β)
END IF
q ← q′
```

3.1.3.2 WHILE-Programme

Im Gegensatz zu den sehr hardware-nahen Registermaschinen sind WHILE-Programme ein Rechnermodell, dass sich an (einfachen) Programmiersprachen orientiert, also nahe an Software ist.

⋆ Elemente eines WHILE-Programms
- Konstanten 0,1,2,...
- Variablen $x_0, x_1, x_2, ...$ (in unbegrenzter Anzahl)
  Sie bilden den *Speichervektor* und beschreiben den Zustand des Programms zu einem bestimmten Zeitpunkt.
- Operationen +, -
- Den Zuweisungsoperator :=
- Das Trennsymbol ;
- Das Schleifenkonstrukt WHILE DO END

⋆ Syntax und *Semantik*

- $x_i := n,\ x_i := x_i \pm c$ sind WHILE-Programme.
  Bedeutung: $x_i \pm c$ erhöht/erniedrigt den Wert von x_i um c.
- Falls P_1 und P_2 WHILE-Programme sind, dann auch $P_1;\ P_2$
  Bedeutung: Erst wird P_1 durchgeführt, dann P_2.
- Mit P ist auch WHILE $x_i \neq 0$ DO P END ein WHILE-Programm.
  Bedeutung: Wiederhole P so lange, bis $x_i = 0$.

Eingabe: $x_1, ...x_k$; (Sonstige Initialisierung: $x_0 = x_{k+1} = x_{k+2}... = 0$)

Ausgabe: $x_0 = n$

⋆ Berechnete Funktion

Ein WHILE-Programm definiert eine partielle Funktion $f_W : \mathbb{N}_0^k \to \mathbb{N}_0$
$f_W(n_1...n_k) = n$, wobei $n_1...n_k$ die Eingabe ist, n die Ausgabe
$f_W(n_1...n_k) = \perp$, falls das Programm abbricht oder endlos weiterläuft.

⋆ Satz: Zu jedem WHILE-Programm W gibt es eine DTM T, die die gleiche Funktion berechnet, d.h. $f_T = f_W$. Zu jeder DTM gibt es ein WHILE-Programm, das die gleiche Funktion berechnet.

Beweisskizze WHILE-Programm → TM

Simulation mit Hilfe einer Mehrband-TM und Programmkomponenten aus 3.1.2.2

Beweisskizze TM → WHILE-Programm

Simulation einer TM mit einem WHILE-Programm mit ähnlichen Methoden wie im Falle der Registermaschine (Genaueres siehe z.B. Hoffmann)

⋆ Bemerkung: Dass WHILE-Programme gleich viel können wie Turingmaschinen, ist a priori nicht selbstverständlich. Die recht ähnlichen LOOP-Programme, bei denen die Anzahl der Schleifendurchläufe fest ist, können das z.B. nicht (siehe z.B. Hoffmann).

3.1.4 Berechenbarkeit und Churchsche These

⋆ Definitionen

- Gegeben sei eine partielle Funktion $f : \Sigma^* \to \Lambda^*$

  f heisst Turing-berechenbar, wenn es eine DTM T gibt mit Eingabealphabet Σ, Bandalphabet Λ, so dass $\Lambda \subseteq \Gamma \backslash \{\square\}$ und $f = f_T$.

- Gegeben sei eine partielle Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0$ $(f(n_1, ...n_k) = n)$

  - f heisst Turing-berechenbar, wenn es eine DTM T gibt mit $f_T = f$, (z.B. via binärer Kodierung $\text{bin}(n_1)\#\text{bin}(n_2)...\text{bin}(n_k)$)
  - f heißt Registermaschinen-berechenbar, wenn es eine RM R gibt mit $f_R = f$
  - f heißt WHILE-berechenbar, wenn es ein WHILE-Programm W gibt mit $f_W = f$

⋆ Satz: Alle drei Berechenbarkeitsbegriffe sind äquivalent!

Beweis: Wir haben in 3.1.3 gezeigt, dass eine wechselseitige Simulation der Rechner möglich ist.

Dieses Ergebnis motiviert die

> Churchsche These: (Church, 1936)
> Die Turing-berechenbaren Funktionen sind genau die "intuitiv berechenbaren" Funktionen.

Im Wesentlichen besagt die These: Mächtiger als eine Turingmaschine kann ein Rechnermodell / ein Computer nicht sein. Mit Turing-Maschinen kann man alles berechnen, was überhaupt berechenbar ist.
$\rightsquigarrow$ Ausgangspunkt für das nächste Kapitel: Entscheidungsprobleme

3.2 Entscheidungsprobleme

Nachdem wir nun den Begriff der Berechenbarkeit geklärt haben, können wir auf die Ausgangsfrage dieses Kapitels zurückkommen: Kann man alle Problem auf dem Rechner lösen? Zur Vereinfachung fokussieren wir auf JA/NEIN-Fragen: Entscheidungsprobleme

Ansatz: Wir formulieren JA/NEIN Fragen als Wortproblem einer Sprache, z.B.

- "Ist n eine Primzahl?"
  $\hat{=}$ $L = \{$Darstellungen von Primzahlen$\}$ über $\Sigma = \{0, ...9\}$
- "Ist das Vorlesungsskript fehlerfrei?"
  $\hat{=}$ $L = \{$Fehlerfreie Skripte$\}$ über $\Sigma = \{$nötige Zeichen$\}$
- "Gibt es in dem Programm P potentiell Endlosschleifen?"
  $\hat{=}$ $L = \{$ Programme ohne Endlosschleifen $\}$

3.2.1 Entscheidbarkeit und Semientscheidbarkeit

3.2.1.1 Zentrale Definitionen

Sei $L \subseteq \Sigma^*$ eine Sprache, die ein JA/NEIN Problem repräsentiert.

⋆ Charakteristische Funktionen

Die totale charakteristische Funktion $\chi_L : \Sigma^* \to \{0, 1\}$ ist definiert als

$$\chi_L(w) = \begin{cases} 1: & w \in L \quad \text{(JA)} \\ 0: & w \notin L \quad \text{(NEIN)} \end{cases}$$

Die partielle charakteristische Funktion $\chi_L{}' : \Sigma^* \to \{1\}$ ist definiert als

$$\chi'_L(w) = \begin{cases} 1: & w \in L \quad \text{(JA)} \\ \perp: & w \notin L \quad \text{(NEIN)} \end{cases}$$

⋆ Entscheidbarkeit

L heisst entscheidbar, wenn χ_L berechenbar ist.

L heisst semientscheidbar, wenn χ'_L berechenbar ist.

NB: Offensichtlich impliziert Entscheidbarkeit Semientscheidbarkeit.

⋆ Entscheidungsverfahren

Turing-Maschinen, die χ_L bzw. χ'_L berechnen, heissen Entscheidungs- bzw. Semi-Entscheidungsverfahren.

NB:
- Semientscheidbarkeit impliziert: Es gibt eine Turingmaschine T mit $\mathcal{L}(T) = L$ (ein Semi-Entscheidungsverfahren). Diese hält im JA-Fall immer an. Im NEIN-Fall verwirft sie oder läuft endlos weiter.
- Entscheidungsverfahren akzeptieren immer, d.h. sie halten immer an mit einer Ausgabe.

3.2.1.2 Wichtige Zusammenhänge

⋆ Satz: Es gilt

(i) L entscheidbar $\leftrightarrow \overline{L} = \Sigma^* \backslash L$ entscheidbar.

(ii) $L, \overline{L}$ semientscheidbar $\leftrightarrow$ L entscheidbar.

Beweis:

(i) Tausche im Entscheidungsverfahren für L die Ausgaben 0, 1 aus ✓

(ii) $\leftarrow$: Klar aus (i)

$\rightarrow$: Betrachte ein Semi-Entscheidungsverfahren $T', \overline{T}'$ zu $L, \overline{L}$. Kombiniere sie zu einem Entscheidungsverfahren T für L über eine Vier-Band-Turingmaschine: Die Eingabe auf Band eins wird zunächst auf die Bänder zwei und drei kopiert. Im zweiten Band läuft T', im dritten Band $\overline{T}'$. Das vierte Band verfolgt die beiden anderen Bänder und akzeptiert, sobald T' oder $\overline{T}'$ akzeptiert, mit Ausgabe 1 oder 0. Da $L \cup \overline{L} = \Sigma^*$, hält T immer. ✓

⋆ Definition: $L \subseteq \Sigma^*$ heisst rekursiv aufzählbar, wenn entweder $L = \varnothing$ oder eine totale berechenbare Funktion $f : \mathbb{N} \rightarrow \Sigma^*$ existiert mit
$L = \{f(n) : n \in \mathbb{N}\}$ ($f(n)$ muss nicht injektiv sein).

Beispiel: $L = \{$Menge aller Primzahlen $\}$ ist rekursiv aufzählbar,
z.B. über $f(n) = n$te Primzahl.

⋆ Satz: $L \subseteq \Sigma^*$ ist semientscheidbar $\leftrightarrow$ L ist rekursiv aufzählbar.

Beweis: Für $L = \varnothing$ offensichtlich, nimm also an $L \neq \varnothing$.

$\leftarrow$: Konstruiere aus dem Berechnungsverfahren für $f(n)$ ein Semi-Entscheidungsverfahren für L z.B.

```
n = 0
WHILE w ≠ f(n)
    n ← n + 1
END
```

$\rightarrow$: Konstruiere eine berechenbare surjektive Abbildung $f(n) : \mathbb{N} \rightarrow L$ aus einem bekannten Semi-Entscheidungsverfahren T' für L

- Nummeriere erst die Elemente w von Σ^* durch ($\Sigma = \{\sigma_1, ...\sigma_N\}$)
  (z.B. via $w_m = \sigma_{j_1}...\sigma_{j_l}$ für $m = j_1 + j_2 N + ...j_l N^{l-1}$)
- Algorithmus zur Bestimmung von $f(n) \in L$:

```
n' = 0
  FOR k = 1, 2, ...{
    FOR m = 1, 2, ..., k {
      w = w_m
      Lasse T' über k Schritte laufen mit Eingabe w_m
      Falls T' akzeptierende hält: {n' ← n' + 1}
      Falls n' = n: Verlasse FOR Schleifen
      }
    }
  f(n) = w
```

NB: Wir wissen, dass alle Sprachen abzählbar sind. Das heisst aber nicht, dass sie alles semientscheidbar sind. Es muss eine total berechenbare Abzählvorschrift $f(n)$ geben.

3.2.2 Hilfsmittel zur Lösung von Entscheidungsproblemen

3.2.2.1 Die Methode der Reduktion

Reduktion wird im Folgenden ein wichtiges Beweisprinzip sein. Es gibt viele Varianten. Wir verwenden hier konkret die sogenannte "many-one" Reduktion.

⋆ Definition: Seien Σ, Γ Alphabete und $L \subseteq \Sigma^*$, $K \subseteq \Gamma^*$ Sprachen

> L heisst auf K (many-one-)reduzierbar, wenn es eine total berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt mit $x \in L \leftrightarrow f(x) \in K$.
> Notation: $L \preceq K$

⋆ Satz:

> (i) Falls $L \preceq K$ und K entscheidbar ist, folgt: L ist entscheidbar.
>
> (ii) Falls $L \preceq K$ und K semientscheidbar ist, folgt: L ist semientscheidbar.

Beweis: Die TM T^K sei ein (Semi)-Entscheidungsverfahren für K und die TM T^f die TM, die $f(x)$ berechnet.

Wir konstruieren eine TM T, die L (semi)-entscheidet über

⋆ Korollar: Falls $L \preceq K$ und L unentscheidbar ist, dann ist K unentscheidbar.

3.2.2.2 Universelle Turingmaschinen

Schwierige Entscheidungsprobleme betreffen oft Aussagen auf einer Meta-Ebene, z.B. "Welche Eigenschaften hat ein Programm"?

Bezogen auf Berechenbarkeit und Turing-Maschinen bedeutet das: Wir brauchen eine TM, die ganze Klassen von Turingmaschinen simulieren kann.

(Analog Universalrechner ≙ Computer mit unendlich großem Speicher. Die Eingabe besteht aus dem Programmcode und später noch der eigentlichen Eingabe.)

Notwendig dafür ist eine Codierung von Turingmaschinen.

⋆ Definition: Gödelisierung

Unter Gödelisierung versteht man eine berechenbare Vorschrift, die jeder Turingmaschine eindeutig eine Binärzahl *code* (T) zuordnet (die "Gödelnummer"), und deren Umkehrung ebenfalls berechenbar ist.

Es sind verschiedene Verfahren der Gödelisierung denkbar. Eine Möglichkeit für die Klasse von TMs mit $\Sigma = \{0, 1\}$ wäre etwa:

Gegeben eine TM $T = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit $\Sigma = \{0, 1\}$ mit $|Q| = N_Q$, $|\Gamma| = N_\Gamma$
Codierung: Binärcode für $\#\#N_Q\#\#N_\Gamma\#\#code(\delta)\#\#\ code(F)\#$
mit $code(\delta)$: enthält für jedes $\delta(q_i, \gamma_m) = \delta(q_j, \gamma_n, X)$
die Zeichenkette $\#\#i\#j\#m\#n\#X\#\#$
und $code(F)$ für $F = \{q_{i_1}, ...q_{i_l}\}$ beinhaltet $\#i_1\#i_2\#...\#i_l\#$
Daraus lässt sich die TM T eindeutig rekonstruieren
(bis auf die "Namen" für die Zustände q_i und die Bandalphabet-Zeichen γ_m).

⋆ Definition: Universelle Turingmaschine

Wir betrachten die Klasse der Turingmaschinen T auf $\Sigma = \{0,1\}$ und definieren

$$T_w = \begin{cases} T: & \text{falls } w = code(T) \\ T_\perp: & \text{sonst,} \end{cases}$$

mit $T_\perp$: Turingmaschine, die nie anhält (unbedingte Endlosschleife).

Die universelle Turingmaschine T_u ist auf dem Alphabet $\Sigma = \{0,1,\#\}$ definiert als $T_u(w\#x) := T_w(x)$. Das heisst, sie gibt auf die Eingabe $w\#x$ mit $w, x \in \{0,1\}^*$ die Ausgabe von T_w zur Eingabe x aus.

Implementierung z.B. über eine Mehrband TM mit
Band 1: Eingabe
Band 2: Aktueller Bandzustand
Band 3: Aktueller Zustand q_i der TM
Weitere Bänder: Nebenrechnungen
Initialisiere: Kopiere x auf Band 2
Dann lies Befehle von T_w von Band 1 ab und verfahre entsprechend.
(siehe Asteroth/Baier)

3.2.3 Das Halteproblem

Das Halteprogramm ist ein paradigmatisches Entscheidungsproblem.
Frage: Gegeben eine Turingmaschine T und eine Eingabe.
Kann man entscheiden, ob T anhält?

3.2.3.1 Die formale Sprache des Halteproblems

Mit Hilfe der Werkzeuge aus 3.2.2 kann das Halteproblem als Wortproblem einer formalen Sprache dargestellt werden.

⋆ Allgemeines Halteproblem: $\Sigma = \{0,1,\#\}$

$H = \{w\#x : w, x \in \{0,1\}^* : T_w \text{ hält bei Eingabe } x \text{ an}\}$

⋆ Spezielles Halteproblem: $\Sigma = \{0,1\}$

$H' = \{w \in \{0,1\}^* : T_w \text{ hält bei Eingabe } w \text{ an}\}$

$H^\varepsilon = \{w \in \{0,1\}^* : T_w \text{ hält bei Eingabe } \varepsilon \text{ an}\}$

Satz: H, H', H^ε sind semientscheidbar.

Ein Semi-Entscheidungsverfahren wäre z.B.
H: Eine TM, die bei Eingabe $w\#x$ die Berechnung von $T_w(x)$ simuliert und 1 ausgibt, wenn $T_w(x)$ anhält.
H', H^ε: analog

Diese TMs halten per Definition an, wenn $w\#x \in \mathcal{L}(H)$ bzw. $w \in \mathcal{L}(H')$ bzw. $w \in \mathcal{L}(H^\varepsilon)$

Nun diskutieren wir die Frage: Sind H, H', H^ε entscheidbar?

3.2.3.2 Entscheidbarkeit des Halteproblems

Wir kommen nun zum Höhepunkt dieses Kapitels.

⋆ Satz: Es gilt

(i) Das allgemeine Halteproblem H ist unentscheidbar

(ii) Das spezielle Halteproblem H' ist unentscheidbar

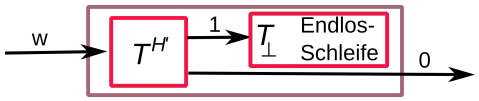
(iii) Das Halteproblem mit leerem Band H^ε ist unentscheidbar.

Beweis

(ii) Widerspruchsbeweis: Nimm an, H' wäre entscheidbar.
⇒ Dann gäbe es ein Entscheidungsverfahren (eine TM) $T^{H'}$ für H'

$$\text{mit } T^{H'}(w) = \begin{cases} 1: & w \in H' \\ 0: & w \notin H' \end{cases}$$

Betrachte nun eine Turingmaschine T, die folgendermaßen arbeitet



und teste die Eingabe $w = code\,(T)$.

Falls $w \in H' \rightarrow T^{H'}(w) = 1 \rightarrow T(w)$ terminiert nicht $\rightarrow w \notin H'$
Falls $w \notin H' \rightarrow T^{H'}(w) = 0 \rightarrow T(w)$ terminiert $\rightarrow w \in H'$

$\rightsquigarrow$ Kontradiktion. Also muss H' unentscheidbar sein !

(i) ergibt sich aus (ii) mit Hilfe einer Reduktion (3.2.2.1)
Es gilt $H' \preceq H$ via $f(w) = w\#w$ (mit $w \in \Sigma^*$, $\Sigma = \{0,1\}$, $\Gamma = \{0,1,\#\}$)
Da H' unentscheidbar ist, muss also auch H unentscheidbar sein.

(iii) ergibt sich ebenfalls aus (ii) mit Hilfe einer Reduktion
Es gilt $H' \preceq H^\varepsilon$ via $f(w) = w'$, wobei w' eine TM codiert, die unabhängig von der Eingabe immer $T_w(w)$ ausführt.

(Implementierung: T'_w ersetzt erst die ursprüngliche Eingabe mit dem Wort w, spult dann zurück und führt die Befehle von T_w aus.)

⋆ Korollar: Es kommt noch schlimmer: Es gibt sogar Sprachen, die nicht einmal semientscheidbar sind. Diese Sprachen sind demnach keine Typ-0 Sprachen, und können durch keine Grammatik erzeugt werden.

Beispiele: $\overline{H}$, $\overline{H'}$, $\overline{H^\varepsilon}$ sind nicht semientscheidbar.

Beweis: Folgt aus 3.2.1.2. Da H, H', H^ε nicht entscheidbar sind, aber semientscheidbar, können $\overline{H}$, $\overline{H'}$, $\overline{H^\varepsilon}$ nicht semientscheidbar sein!

⋆ Anmerkung: Alternativer Beweis über Diagonalsprache.
Da TM durch endliche Zeichenketten $code\,(T)$ codiert werden, ist die Menge aller TMs abzählbar. Wir nummerieren sie durch und konstruieren die Tabelle

$$t_{ij} = \begin{cases} 1 & \text{falls } T_i \text{ die Eingabe } j \text{ (als Binärzahl) nicht hält.} \\ 0 & \text{falls } T_i \text{ bei Eingabe } j \text{ (als Binärzahl) hält} \end{cases}$$

Die Diagonalsprache ist definiert als $L_D = \{n \in \mathbb{N} : t_{nn} = 1\}$.

⇒ Dann kann es kein Semi-Entscheidungsverfahren für L_D geben. Gäbe es eines, dann könnte man es so modifizieren, dass es für $n \in L_D$ t_{nn} ausgibt und ansonsten immer endlos weiterläuft. Die entsprechende TM T hätte eine Nummer k. Die Frage, ob $k \in L_D$ ist, führt analog dem Beweis (ii) oben zu einem Widerspruch.

3.2.4 Der Satz von Rice

Ein noch viel weitreichenderer Satz ist der Satz von Rice (Rice, 1951). Er besagt im Wesentlichen, dass jede nichttriviale Eigenschaft einer Ausgabe von Turingmaschinen unentscheidbar ist!

⋆ Satz: Sei $S \neq \varnothing$ eine echte Teilmenge aller partiell berechenbarer Funktionen $f : \{0,1\}^* \to \{0,1\}^*$. Wir definieren die Sprache L_S über
$L_S = \{w \in \{0,1\}^* : T_w \text{ berechnet eine partielle Fkt. } f \in S\}$.
Dann ist L_S unentscheidbar!

⋆ Beispiele: Unentscheidbar ist z.B.

- Ist $\mathcal{L}(T_w)$ leer? $(L_S = \{w : T_w \text{ terminiert nie}\})$
  $(S = \{f_\perp\}$; Entspricht dem allgemeinen Halteproblem)
- Akzeptiert T_w nach Eingaben von Worten der Maximallänge n?
  $(S = \{\text{berechenbare Funktionen } \bigcup_{j=0}^{n} : \{0,1\}^j \to \{0,1\}\})$
- Berechnet T_w eine monotone Funktion? $(T_w(i+1) \geqslant T_w(i)?)$

NB: Aussagen über Zustände von TM können entscheidbar sein,
z.B.: Schreibt die TM überhaupt je irgendetwas raus?

Beweis des Satzes: Wir unterscheiden zwischen dem Fall $f_\perp \notin S$ und $f_\perp \in S$. und konstruieren je nachdem eine Reduktion $H^\varepsilon \preceq L_S$ oder $\overline{H^\varepsilon} \preceq L_S$.

Dazu wählen wir partiell berechenbare Funktionen $f, \overline{f} : \{0,1\}^* \to \{0,1\}$ mit $f \in S$ und $\overline{f} \notin S$. Solche Funktionen muss es geben, da $S \neq \varnothing$ und S nicht alle partiell berechenbaren Fkt. umfasst. Aus den zugehörigen TMs $T^f, T^{\overline{f}}$ konstruieren wir die TMs

$$T^{f;w}(x) = \begin{cases} f_\perp : w \notin H^\varepsilon \\ f(x) : w \in H^\varepsilon \end{cases}, \qquad T^{\overline{f};w}(x) = \begin{cases} f_\perp : w \notin H^\varepsilon \\ \overline{f}(x) : w \in H^\varepsilon \end{cases},$$

Realisierung z.B. über eine TM, die bei Eingabe x erst T_w mit Eingabe ε simuliert, und dann T^f bzw. $T^{\overline{f}}$ mit Eingabe x.

Nun machen wir die Fallunterscheidung $f_\perp \notin S$ oder $f_\perp \in S$.

⋆ $f_\perp \notin S \rightsquigarrow$ Wir betrachten $T^{f;w}$
$w \in H^\varepsilon \to T_w(\varepsilon)$ hält $\to T^{f,w}$ berechnet $f \in S \to code(T^{f,w}) \in L_S$
$w \notin H^\varepsilon \to T_w(\varepsilon)$ hält nicht $\to T^{f;w}$ berechnet $f_\perp \notin S \to code(T^{f;w}) \notin L_S$
⇒ Die Abbildung $w \to code(T^{f;w})$ generiert eine Reduktion $H^\varepsilon \preceq L_S$

⋆ $f_\perp \in S \rightsquigarrow$ Wir betrachten $T^{\overline{f};w}$
$w \in \overline{H^\varepsilon} \to T_w(\varepsilon)$ hält nicht $\to T^{\overline{f};w}$ berechnet $f_\perp \in S \to code(T^{\overline{f};w}) \in L_S$
$w \notin \overline{H^\varepsilon} \to T_w(\varepsilon)$ hält $\to T^{\overline{f};w}$ berechnet $\overline{f} \notin S \to code(T^{\overline{f};w}) \notin L_S$
⇒ Die Abbildung $w \to code(T^{\overline{f};w})$ generiert eine Reduktion $\overline{H^\varepsilon} \preceq L_S$

In beiden Fällen folgt: Da $H^\varepsilon, \overline{H^\varepsilon}$ unentscheidbar sind, ist L_S unentscheidbar ✓

3.2.5 Weitere Entscheidungsprobleme

Die bis jetzt diskutierten Entscheidungsprobleme bezogen sich alle auf Turingmaschinen. Im folgenden Abschnitt sollen Entscheidungsprobleme diskutiert werden, die nicht direkt mit Turingmaschinen zu tun haben, z.B. einige der eingangs gestellten Fragestellungen zu formalen Sprachen.

Aus dem Satz von Rice können wir jetzt schon folgern, dass die in Abschnitt 3.1.2.3 genannten typischen Fragestellungen in formalen Sprachen für Sprachen vom Typ 0 unentscheidbar sind:

Gegeben seien zwei Grammatiken G, G' und ein Wort w. Dann sagt folgende Tabelle aus, welche Fragen nach unserem aktuellen Kenntnisstand entscheidbar sind (JA, es gibt einen Algorithmus, siehe Kapitel 3 Anfang) bzw. unentscheidbar (NEIN):

Problem	regulär	kontextfrei	Typ 0
Ist $w \in \mathcal{L}(G)$?	JA	JA	NEIN
Ist $\mathcal{L}(G) = \varnothing$?	JA	JA	NEIN
Ist $\lvert\mathcal{L}(G)\rvert = \infty$?	JA	JA	NEIN
Ist $\mathcal{L}(G) = \mathcal{L}(G')$?	JA	?	NEIN

Für kontextfreie Sprachen kennen wir jedoch noch nicht alle Antworten. Darüber hinaus gibt es noch eine Reihe weiterer Entscheidungsprobleme im Zusammenhang mit kontextfreien Sprachen, die ebenfalls von Interesse sind, z.B.: Ist $\mathcal{L}(G) \cap \mathcal{L}(G')$ leer, unendlich, kontextfrei?

Die Brücke zur Beantwortung vieler dieser Fragen wird durch ein auf den ersten Blick ziemlich spezielles Problem gebildet, das sich jedoch gut auf andere Probleme reduzieren läßt: Dem Postschen Korrespondenzproblem (Post, 1946)

3.2.5.1 Das Postsche Korrespondenzproblem (PKP)

⋆ Definition: Postsches Korrespondenzproblem (PKP)

Gegeben sei ein Alphabet Σ

- Eine Instanz $K = p_1 ... p_k$ des PKP ist eine Folge von Wortpaaren $p_i = \binom{u_i}{v_i}, i = 1, ..., k$ mit $u_i, v_i \in \Sigma^+$
- Eine Instanz besitzt eine Lösung, wenn es eine Indexfolge $i_1 ... i_m$ gibt, so dass $u_{i_1} \circ u_{i_2} \circ ... u_{i_m} = v_{i_1} \circ v_{i_2} \circ ... v_{i_m}$.
- Die Sprache des PKP ist PKP $=\{w \in (\Sigma \cup \{(,), , \})^* : w$ kodiert eine Instanz des PKP und die Instanz hat eine Lösung $\}$

⋆ Beispiele ($\Sigma = \{0, 1\}$)

$K = \binom{1}{101}\binom{010}{0}$ hat eine Lösung

$K = \binom{1}{101}\binom{01}{1}\binom{10}{0}$ hat eine Lösung, z.B. $121332 = \binom{1011101001}{1011101001}$

$K = \binom{1}{11}\binom{1}{01}$ kann keine Lösung haben,
denn unten sind immer mehr Zeichen als oben.

$K = \binom{10}{101}\binom{01}{1}$ kann keine Lösung haben,
denn oben gibt es immer exakt gleich viele Einsen wie Nullen, und unten sind die Einsen immer in der Überzahl.

$K = \binom{001}{0}\binom{01}{011}\binom{01}{101}\binom{10}{001}$? (hat eine Lösung, aber mit Mindestlänge 66)

⋆ Satz: Das PKP ist unentscheidbar

Beweis: In zwei Schritten

(i) Betrachte zunächst ein Hilfsproblem, das "modifizierte PKP", kurz MPKP. Es ist definiert wie das PKP, nur wird für die Lösung einer Instanz das erste Indexpaar vorgegeben: K hat eine Lösung, wenn es eine Folge $i_2...i_m$ gibt mit $u_1 \circ u_{i_2} \circ ...u_{i_m} = v_1 \circ v_{i_2} \circ ...v_{i_m}$.
Zeige MPKP $\preceq$ PKP

(ii) Zeige $H^\varepsilon \preceq$ MPKP

Da H^ε nicht entscheidbar ist, folgt, das MPKP und PKP nicht entscheidbar sein können.

Vorbemerkung: Im folgenden zeigen wir $H^\varepsilon \preceq$ MPKP und MPKP $\preceq$ PKP für MPKPs bzw. PKPs mit bestimmtem Alphabet, das noch spezifiert werden wird. Das ist aber keine Einschränkung, denn wir können im PKP/MPKP auch alle Zeichen mit 01 Folgen kodieren, z.B. über die Zuordnung $\sigma_n \triangleq 0\underbrace{1111..1}_{n \text{ Mal}}$ für das $n$te Zeichen des Alphabets Σ.

Beweis zu (i) (MPKP $\preceq$ PKP)

Wir konstruieren eine Funktion $f : K = (p_1...p_k) \to K' = (p'_1...p'_k p'_{k+1} p'_{k+2})$, die berechenbar ist und garantiert, dass $K \in$ MPKP $\leftrightarrow K' \in$ PKP.

Das Problem ist offenbar die Richtung $\leftarrow$: K' muss so konstruiert sein, dass seine Lösungen automatisch mit einem vordefinierten Gegenstück zu p_1 beginnen.

Trick: Erzwinge dies durch Hilfssymbole $\#, \$ \notin \Sigma_{\text{PKP}}$ ($\Sigma_{\text{MPKP}} = \Sigma_{\text{PKP}} \cup \{\#, \$\}$)
- Zu $u_i = \sigma_{i_1}\sigma_{i_2}...\sigma_{i_n}$ definiere $\widehat{u}_i = \sigma_{i_1}\#\sigma_{i_2}\#...\#\sigma_{i_n}$;
  Zu $v_j = \sigma_{i_1}\sigma_{j_2}...\sigma_{j_n}$ definiere $\widehat{v}_j = \sigma_{j_1}\#\sigma_{j_2}\#...\#\sigma_{j_n}$.
- Definiere $p'_i = \binom{\widehat{u}_i\#}{\#\widehat{v}_i}$ für $i = 1...k$ und $p'_{k+1} = \binom{\#\widehat{u}_1\#}{\#\widehat{v}_1}$, $p'_{k+2} = \binom{\$}{\#\$}$

$\rightsquigarrow$ Die Funktion $K = (p_1...p_k) \to K' = (p'_1...p'_{k+2})$ ist berechenbar und erfüllt $K \in$ MPKP $\leftrightarrow K' \in$ PKP.

"$\to$": Klar (Falls K eine Lösung $p_1 p_{i_2}...p_{i_n} = \binom{\sigma_1...\sigma_N}{\sigma_1...\sigma_N}$ hat,
ist $p'_{k+1}p'_{i_2}...p'_{i_n}p'_{k+2} = \binom{\#\sigma_1\#\sigma_2...\sigma_N\#\$}{\#\sigma_1\#\sigma_2...\sigma_N\#\$}$ eine Lösung von K' ✓)

"$\leftarrow$": Alle Lösungen des PKP müssen mit p'_{k+1} beginnen und mit p'_{k+2} enden, da p'_{k+1} das einzige Paar von Worten ist mit demselben Anfangszeichen und p'_{k+2} das einzige Paar von Worten, die mit demselben Zeichen enden.
Sei $p'_{k+1}p'_{i_2}...p'_{i_n}p'_{k+1}$ eine Lsg. des PKP.
Dann ist $p_1 p_{i_2}...p_{i_n}$ per Konstruktion eine Lsg. des MPKP ✓

Beweis zu (ii) ($H^\varepsilon \preceq$ MPKP) (Erinnerung: $H^\varepsilon = \{w : T_w$ hält bei Eingabe ε an $\}$)

Vorab: Definiere $H'^\varepsilon = \{w : T_w$ hält bei Eingabe ε akzeptierend an $\}$

Es gilt $H'^\varepsilon \preceq H^\varepsilon$ via $f(w) = w'$, wobei $T_{w'}$ T_w simuliert und an verwerfende Zustände eine Endlosschleife hängt. Also reicht es, $H'^\varepsilon \preceq$ MPKP zu zeigen.

Gesucht also: Eine berechenbare Funktion f mit $w \in H'^\varepsilon \leftrightarrow K = f(w) \in$ MPKP

Lösungsidee: Wortpaare in K kodieren mögliche Konfigurationswechsel von T_w

Naiver Ansatz: Wähle $\Sigma_{\text{MPKP}} = \Sigma \cup Q \cup \{\#\}$ mit $\# \notin \Gamma$ (Trennzeichen)
Wähle als Startpaar des MPKP $p_1 = \binom{\#}{\#\#\kappa_0}$.

Für alle Konfigurationswechsel $\kappa_i \vdash \kappa_j$ in T_w kreiere ein Paar $\binom{\#\kappa_i}{\#\kappa_j}$

Für alle möglichen Endkonfigurationen κ_e kreiere Endpaare $\binom{\#\kappa_e\#\$}{\#\$}$.
Da das Startpaar unten ein # mehr enthält als das Endpaar, und # als Zeichen in den Konfigurationen κ_j nicht vorkommt, kann die Bilanz nur durch ein Endpaar ausgeglichen werden. Bis zum Erreichen des Endpaares hat die Wortpaar-Folge die Form $\binom{\#\#\kappa_0\#\kappa_1\#\ldots\#\kappa_e\#\$}{\#\#\kappa_0\#\kappa_1\#\ldots\#\kappa_e\#\$}$ und stellt somit eine Lösung dar. Darin sind per Konstruktion alle (κ_n, κ_{n+1}) erlaubte Konfigurationsübergänge von T_w (d.h. $\kappa_n \vdash \kappa_{n+1}$).

Problem mit diesem Ansatz: Da es unendlich viele mögliche Konfigurationen gibt, müsste K unendlich gross sein. Das ist nicht erlaubt.

$\rightsquigarrow$ Wir müssen Konfigurationswechsel intelligenter kodieren!

Möglichkeit einer konkreten Umsetzung (Asteroth/Baier, leicht modifiziert)

Ziel: Konzipiere die Instanz so, dass mögliche Konfigurationswechsel $\kappa_1 \vdash \kappa_2 \vdash \ldots\kappa_e$ in einem akzeptierenden Lauf automatisch Lösungen der Instanz generieren, der Form (sinngemäß) $\binom{\#\#\kappa_0\#\kappa_1\#\kappa_2\ldots\#\kappa_e\#\$}{\#\#\kappa_0\#\kappa_1\#\kappa_2\ldots\#\kappa_e\#\$}$

Zur Vereinfachung modifizieren wir das Ziel leicht:

- Wir lassen auch Repetitionen von κ zu, z.B. $\binom{\ldots\kappa_{i-1}\#\kappa_i\#\kappa_i\#\kappa_{i+1}\#\cdots}{\ldots\kappa_{i-1}\#\kappa_i\#\kappa_i\#\kappa_{i+1}\#\ldots}$
- Wir lassen in der Darstellung der Konfigurationen auch eine endliche Anzahl äußerer Blanks ($\square$) zu (erspart Fallunterscheidungen)
- Am Ende soll in $\kappa_e = \alpha q_e \beta$ das Zeichen q_e noch sukzessive freigestellt werden, so dass die Konfigurationsfolge in $\binom{\cdots\#\kappa_e\#\ldots\#q_e\#\$}{\ldots\#\kappa_e\#\ldots\#q_e\#\$}$ endet.

Ansatz: Die Instanz K enthält folgende Paare:

- Das Startpaar $\binom{\#}{\#\#q_0}$ und Endpaare $\binom{q_e\#\$}{\$}$ für alle $q_e \in F$.
  Wie im naiven Ansatz generiert das Startpaar unten einen Überschuss an #, der nur durch ein Endpaar ausgeglichen werden kann.
- Paare, die die Übergangsfunktion kodieren: $\binom{X}{Y}$ wobei $X = \xi q \gamma$
  und $Y = \begin{cases} \xi q'\gamma' : & \delta(q,\gamma) = (q',\gamma',N) \\ \xi\gamma' q' : & \delta(q,\gamma) = (q',\gamma',R) \\ q'\xi\gamma' : & \delta(q,\gamma) = (q',\gamma',L) \end{cases} \quad (\xi,\gamma,\gamma' \in \Gamma,\ q,q' \in Q)$
  (Zugehörige Konfigurationswechsel: $\alpha X \beta \vdash \alpha Y \beta$ mit $\alpha, \beta \in \Gamma^*$)
- Paare, die kopieren: $\binom{a}{a}$ für alle $a \in \Gamma \cup Q \cup \{\#\}$
  Damit werden die Zeichen einer Konfiguration, die von der Übergangsfunktion nicht betroffen sind, von oben nach unten übertragen.
- Paare, die äußere Blanks einfügen $\binom{\#}{\square\#}, \binom{\square\#}{\#}, \binom{\#}{\#\square}, \binom{\#}{\square\#}$
  Diese ermöglichen Konfigurationswechsel, an denen äußere Blanks beteiligt sind, wie z.B. $q\gamma\beta \vdash q'\square\gamma'\beta$ via $\delta(q,\gamma) = (q',\gamma',L)$ bzw. $\binom{\square q\gamma}{q\square\gamma'}$ oder $\alpha q \vdash \alpha\gamma' q'$ via $\delta(q,\square) = (q',\gamma',R)$ bzw. $\binom{\xi,q,\square}{\xi\gamma' q'}$.
- Paare, die in akzeptierenden Konfigurationen $\kappa_e = \alpha q_e \beta$ mit $q_e \in F$ die Zeichenketten α und β sukzessive entfernen: $\binom{\gamma q_e}{q_e}, \binom{q_e\gamma}{q_e}$ für $\gamma \in \Gamma$, $q_e \in F$. Sie sind nötig, um q_e freizustellen, damit schließlich ein Endpaar die Folge abgeschließen kann.

Beispiel: $q_0 \vdash 1q_e$ mittels $\delta(q_0,\square) = (q_e,1,R) \mathrel{\widehat{=}}$ Paar $\binom{\square q_0\square}{\square 1 q_1}$

Entspricht der Lösung: $\binom{\#}{\#\#q_0}\binom{\#}{\#\square}\binom{q_0}{q_0}\binom{\#}{\square\#}\binom{\square q_0\square}{\square 1 q_1}\binom{\#\square}{\#}\binom{1q_e}{q_e}\binom{\#}{\#}\binom{q_e\#\$}{\$}$
bzw. $\binom{\#\#q_0\#\square q_0\square\#\square 1 q_e\#q_e\#\$}{\#\#q_0\#\square q_0\square\#\square 1 q_e\#q_e\#\$}$

$\rightsquigarrow$ Die so definierte Funktion $w \to K$ ist berechenbar und umkehrbar, d.h. aus K kann die zugehörige TM T_w rekonstruiert werden.
Sie erfüllt $w \in H'^{\varepsilon} \leftrightarrow K \in$ MPKP

"→": Falls $T_w(\varepsilon)$ hält, kann aus K automatisch die zugehörige Konfigurationsfolge erzeugt werden und daraus eine Lösung von K konstruiert werden ✓.

"←": Lösungen von K haben immer einen Anfangsblock, der mit dem Startpaar beginnt, danach eine Konfigurationsfolge der zugehörigen TM kodiert, und in einem Endpaar $\binom{q_e\#\$}{\$}$ mündet. Dies folgt, wie oben beschrieben, aus der Bilanz der # Symbole: Das Startpaar hat unten ein # mehr als oben, alle anderen Paare bis auf die Endpaare haben unten und oben gleich viele #. Nur mit einem Endpaar kann die Bilanz ausgeglichen werden. Der Anfangsblock entspricht einem akzeptierenden Lauf der zugehörigen TM. ✓
NB: An den Anfangsblock können sich z.B. wegen der Paare $\binom{a}{a}$ beliebige weitere Zeichenfolgen anschließen.

3.2.5.2 Anwendungen auf formale Sprachen

Aus der Unentscheidbarkeit des PKP können Unentscheidbarkeitsaussagen über eine Reihe anderer Probleme hergeleitet werden. Wir können nun den Bogen zu den Entscheidungsproblemen in formalen Sprachen schlagen.
Wir beziehen uns dabei in der Regel auf das PKP auf dem Alphabet $\Sigma = \{0, 1\}$: Dem 0/1-PKP.

Satz: Für kontextfreie Grammatiken G_i sind folgende Fragen unentscheidbar:

(a) $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \varnothing$? (Schnittproblem)

(b) $|\mathcal{L}(G_1) \cap \mathcal{L}(G_2)| = \infty$?

(c) $\mathcal{L}(G_1) = \mathcal{L}(G_2)$? (Äquivalenzproblem)

Beweise (Notation: $w^R \triangleq w$ mit umgekehrter Zeichenfolge)

(a) Durch Reduktion 0/1-PKP $\preceq$ Schnittproblem

Wir konstruieren eine berechenbare Funktion $K \to f(K) = (G_1, G_2)$, so dass $K \in$ 0/1-PKP $\leftrightarrow \mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \varnothing$

Konkret: Gegeben sei $K = \binom{x_1}{y_1} ... \binom{x_k}{y_k}$ $(x_i, y_i \in \{0,1\}^+)$

Wähle $\Sigma = \{0, 1, \$, a_1, ...a_k\}$

$G_1 = (V_1, \Sigma, P_1, S)$ mit $V_1 = \{S, A, B\}$

$P_1 : S \to A\$B,\ A \to a_i A x_i | a_i x_i,\ B \to y_i^R B a_i | y_i^R a_i$

$\rightsquigarrow$ erzeugt Worte $a_{i_n}...a_{i_1}x_{i_1}...x_{i_n}\$y_{j_m}^R..y_{j_1}^R a_{j_1}...a_{j_m}$

$G_2 = (V_2, \Sigma, P_2, S)$ mit $V_2 = \{S, C\}$

$P_2 : S \to a_i S a_i | C,\ C \to 0C0|1C1|\$$

$\rightsquigarrow$ erzeugt Worte $a_{i_n}...a_{i_1}v\$v^R a_{i_1}...a_{i_n}$ mit $v \in \{0,1\}^*$ mit $v = x_{i_1}...x_{i_n} = y_{i_1}...y_{i_n}$

$\Rightarrow$ $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \varnothing$ genau dann, wenn K eine Lösung hat.

NB: G_1, G_2 sind sogar deterministisch kontextfrei, denn man kann einen zugehörigen DKA konstruieren (Übungsaufgabe).

(b) Wieder durch Reduktion. Wir können dieselbe Abbildung wie in (a) verwenden. Wenn K eine Lösung hat, dann auch unendlich viele, da man sie aneinanderhängen kann.

(c) Durch Reduktion $\overline{\text{0/1-PKP}} \preceq$ Äquivalenzproblem

Mit 0/1-PKP ist auch $\overline{\text{0/1-PKP}}$ unentscheidbar und damit auch das Äquivalenzproblem, falls eine Reduktion möglich ist.

Die Reduktion basiert auf der gleichen Abbildung wie oben (a). Sie bildet K (siehe oben) auf ein Paar von deterministisch kontextfreie Grammatiken G_1 und G_2 ab. Das nutzen wir hier und verwenden ohne Beweis noch den folgenden Satz:

G_2 determ. kontextfrei $\leftrightarrow \overline{\mathcal{L}(G_2)}$ determ. kontextfrei

Da G_2 deterministisch kontextfrei ist, existiert demnach eine Grammatik $\overline{G_2}$, so dass $\mathcal{L}(\overline{G_2}) = \overline{\mathcal{L}(G_2)}$ kontextfrei ist.

Wir definieren $G_3 = G_1 \uplus \overline{G_2}$, d.h. $\mathcal{L}(G_3) = \mathcal{L}(G_1) \cup \overline{\mathcal{L}(G_2)}$

$$
\begin{aligned}
\Rightarrow\ & \mathcal{L}(\overline{G_2}) = \mathcal{L}(G_3) \\
& \leftrightarrow \overline{\mathcal{L}(G_2)} = \mathcal{L}(G_3) \qquad \left(=\mathcal{L}(G_1) \cup \overline{\mathcal{L}(G_2)}\right) \\
& \leftrightarrow \overline{\mathcal{L}(G_2)} \supseteq \mathcal{L}(G_1) \\
& \leftrightarrow \mathcal{L}(G_1) \cap \mathcal{L}(G_2) = \varnothing \\
& \leftrightarrow K \text{ hat keine Lösung.}
\end{aligned}
$$

Also erfüllt die Funktion $K \to f(K) = (\overline{G_2}, G_3)$ die Bedingung $K \in \overline{\text{0/1-PKP}} \leftrightarrow \mathcal{L}(\overline{G_2}) = \mathcal{L}(G_3)$ und kann die gewünschte Reduktion vermitteln.

Weitere Beispiele unentscheidbarer Probleme (ohne Beweis)

Für kontextfreie Grammatiken G, G_i sind unentscheidbar:

- Ist G mehrdeutig?
- Ist $\overline{\mathcal{L}(G)}$ kontextfrei?
- Ist $\mathcal{L}(G)$ regulär?
- Ist $\mathcal{L}(G)$ deterministisch kontextfrei?
- Ist $\mathcal{L}(G_1) \cap \mathcal{L}(G_2)$ kontextfrei?

3.3 Zusammenfassung

In dem Kapitel 3 wurde das Thema der Berechenbarkeit und Entscheidbarkeit behandelt. Wichtige Konzepte und Ergebnisse sind:

⋆ Berechenbarkeit

Gegeben sei eine partielle Funktion $f : \mathbb{N}_0 \to \mathbb{N}$.
Folgende Aussagen sind äquivalent:
(I) f ist Turing-berechenbar (i.e., kann mit einer TM berechnet werden)
(II) f ist Registermaschinen-berechenbar
(III) f ist WHILE-berechenbar

Churchsche These: "Intuitive berechenbar" $\equiv$ TM-berechenbar

⋆ Zusammenhang mit formalen Sprachen

Sei $L \subseteq \Sigma^*$ eine formale Sprache. Dann sind äquivalent:
(I) $L = \mathcal{L}(T)$ für eine DTM T
(II) $L = \mathcal{L}(T')$ für eine NTM T'
(III) L ist semi-entscheidbar (s.u.)
(IV) L ist eine Typ-0 Sprache

⋆ Entscheidbarkeit

- Definition von Entscheidbarkeit und Semi-Entscheidbarkeit
- Es gibt unentscheidbare Sprachen (allen voran das Halteproblem H)
- Es gibt Sprachen, die nicht einmal semi-entscheidbar sind (z.B. $\overline{H}$). Diese Sprachen sind demnach keine Typ-0 Sprachen und es gibt keine Grammatik, die sie erzeugt.

⋆ Praktische Aspekte und Hilfsmittel

- Beweismethode der Reduktion (hier konkret: "many-one Reduktion")
- Universelle Turingmaschinen
- Postsches Korrespondenzproblem (PKP): Zur Behandlung von Entscheidungsproblemen in formalen Sprachen

⋆ Anwendungen und Folgerungen

- Satz von Rice: Bzgl. der Ausgabe von TM ist fast alles unentscheidbar
- Unentscheidbarkeitsaussagen für formalen Sprachen