
Automaten, formale Sprachen und Berechenbarkeit

Markus Holzer

Erstellt von Benjamin Gufler

Inhaltsverzeichnis

1	Berechenbarkeit	3
1.1	Grundlagen	3
1.1.1	Kardinalitätsargumente	4
1.2	Syntax und Semantik von WHILE – Programmen	5
1.2.1	Aufzählbarkeit von WHILE – Programmen	7
1.2.2	Techniken der Theorie der Berechenbarkeit	8
1.2.3	Das Rekursionstheorem und Eigenschaften von Aufzählungen	11
1.3	LOOP – Programme — einige Bemerkungen	14
1.4	Berechenbare Eigenschaften von Mengen	16
1.5	Die Sätze von Rice	19
1.6	Alternative Modelle für Berechenbarkeit	22
2	Automaten und formale Sprachen	25
2.1	Grundlagen	25
2.2	(Chomsky –) Grammatiken	26
2.3	Maschinenmodelle	29
2.3.1	Turing-Maschinen	29
2.4	Endliche Automaten	31
2.4.1	Alternative Darstellungen von regulären Mengen	35
	Syntaxdiagramme oder beschriftete Myhill-Graphen	35
	Reguläre Ausdrücke	36
	Erkennbarkeit	37
	Rechnen mit regulären Ausdrücken und Gleichungssysteme	39
2.5	Endliche Automaten über unendlichen Wörtern	41
2.5.1	Büchi-Automaten	41
2.5.2	Muller-Automaten	44
2.5.3	Rabin-Automaten	45

Automaten, formale Sprachen und Berechenbarkeit

Automaten, formale Sprachen und Berechenbarkeit sind Teilgebiete der theoretischen Informatik:

Automaten, formale Sprachen: beschäftigt sich mit Modellierung von Problemen (durch Grammatiken und / oder Automaten). Beschreibung von unendlichen Objekten wie Sprachen durch *endliche* Beschreibungsmechanismen.

Berechenbarkeit und Komplexitätstheorie: Berechenbarkeitstheorie versucht den Begriff der Berechenbarkeit zu präzisieren (verschiedene Modelle sind äquivalent — Church – Turing – These).

Komplexitätstheorie klassifiziert Probleme nach Ressourcenaufwand (Zeit, Platz bei Turingmaschine)

Struktur der Vorlesung: zuerst Berechenbarkeit, dann Automaten und formale Sprachen.

Kapitel 1

Berechenbarkeit

Grundfrage: »Was können (idealisierte) Rechner alles?«

Eingabe → Black box: Turingmaschine, While-Programm → Ausgabe

Typische Fragestellungen

1. Die korrekte Parameterversorgung in Algol 60 ist unentscheidbar zur Compilezeit.
2. Es gibt keinen Algorithmus, der zu einem gegebenen Paar aus Programm und Daten anhält oder nicht (Halteproblem).
3. Die Frage, ob eine Menge von Wang – Teilen die Ebene vollständig parkettieren, ist unentscheidbar.
4. Betrachte folgende Funktion:

$$f(n) = \begin{cases} 3n + 1 & \text{falls } n \text{ gerade} \\ \frac{n}{2} & \text{sonst} \end{cases}$$

Frage: Terminiert die Folge $n, f(n), f(f(n)), \dots$ zum Wert 1?
Status der Entscheidbarkeit ist offen.

5. Wir betrachten die Goodstein – Sequenz:

$$(n)_0 = n$$
$$(n)_{k+1} = \begin{cases} B_{k+2}[(n)_k] & \text{falls } (n)_k > 0 \\ 0 & \text{sonst} \end{cases}$$

wobei B_k die »base pumping« – Funktion ist. Intuitiv funktioniert »base pumping« wie folgt: z.B.

$$266 = 2^8 + 2^3 + 2 = 2^{2^{2+1}} + 2^{2+1} + 2$$
$$B_2(266) = 3^{3^{3+1}} + 3^{3+1} + 3$$

Frage: Terminiert die Folge $(n)_0, (n)_1, \dots$ zum Wert 0 für gegebenes n ?

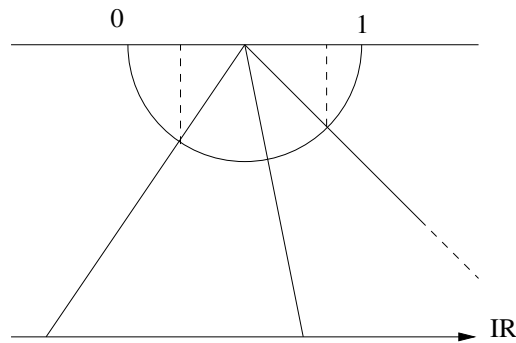
Die Folge für $n = 4$ terminiert für einen Wert k (Anzahl der Schritte) $3 \cdot 2^{402653211} - 1 \approx 10^{10^8}$.

Die Folge terminiert immer. Daher ist das Problem trivialerweise entscheidbar.

1.1 Grundlagen

Bezeichnungen:

- A, B, \dots Mengen (nicht notwendigerweise endlich) von Objekten.

Abbildung 1.1: Abbildung von \mathbb{R} auf $(0, 1)$

- Eine Relation R ist eine Menge von geordneten Tupeln. Bild und Urbild einer Relation:

$$y \in \text{range}(R) \Leftrightarrow \exists x : (x, y) \in R$$

$$x \in \text{dom}(R) \Leftrightarrow \exists y : (x, y) \in R$$

- Eine partielle Funktion ist eine Relation f so, dass für jedes $x \in \text{dom}(f)$ genau ein y mit xfy existiert. Wir schreiben $y = f(x)$ oder $f(x) = y$. Es ist $f \subseteq A \times B$; wir schreiben $f : A \rightarrow B$.
- Eine partielle Funktion f heißt total, falls $\text{dom}(f) = A$.
- Komposition von Funktionen wie gewohnt: $(g \circ f)(x) = g(f(x))$.
- Sei f eine totale Funktion.
 1. f ist INJEKTIV, falls gilt: $x_1, x_2 \in A, x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)$.
 2. f ist SURJEKTIV, falls $B = \text{range}(f)$
 3. f ist BIJEKTIV, falls f injektiv und surjektiv ist.
- Zwei Mengen A und B sind gleichmächtig, falls es eine Bijektion von A nach B gibt. Wir schreiben $A \approx B$. Die Menge B hat gleiche oder größere Kardinalität als A , falls es eine Injektion von A nach B gibt. In Zeichen: $A \preceq B$.

Folgender Satz ist manchmal sehr hilfreich:

Theorem (Schröder – Bernstein): Wenn $A \preceq B$ und $B \preceq A$ gilt, so folgt $A \approx B$.

- Die Menge A ist abzählbar, falls $A \preceq \mathbb{N}$ gilt, wobei \mathbb{N} die natürlichen Zahlen (mit 0) sind. A ist überabzählbar, falls $\mathbb{N} \succ A$ gilt ($A \preceq \mathbb{N}$ gilt nicht).

1.1.1 Kardinalitätsargumente

Offensichtlich ist \mathbb{N} abzählbar. Dies gilt auch für die Menge der Quadratzahlen $\{0, 1, 4, 9, \dots\} \subseteq \mathbb{N}$. Die Bijektion ist $f(x) = x^2$. Kardinalitätsgleiche Mengen zu \mathbb{N} sind $\mathbb{N} \times \mathbb{N}$ und \mathbb{Q} ; hierbei sei \mathbb{Q} die Menge der rationalen Zahlen. Dass $\mathbb{N} \approx \mathbb{N} \times \mathbb{N}$ gilt, sieht man mit Hilfe des Cantorschen Diagonalschemas, das eine Bijektion $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ realisiert. Man kann sich überlegen, dass

$$f(x, y) = \frac{1}{2} [(x + y)^2 + 3x + y]$$

Mit ähnlichem Schema kann man $\mathbb{N} \approx \mathbb{Q}$ zeigen.

Sei \mathbb{R} die Menge der reellen Zahlen. Das offene Einheitsintervall $(0, 1) = \{x \in \mathbb{R} \mid 0 < x < 1\}$ ist kardinalitätsgleich zu \mathbb{R} . Offensichtlich ist $(0, 1) \subseteq \mathbb{R}$ und damit $(0, 1) \preceq \mathbb{R}$. Die Konstruktion aus Abbildung 1.1 liefert $\mathbb{R} \preceq (0, 1)$. Mit Schröder – Bernstein folgt $(0, 1) \approx \mathbb{R}$.

Bemerkung: $\mathbb{R} \approx \mathbb{R} \times \mathbb{R}$ (Raumfüllende Kurven: Hilbert, Peano).

Das Verhältnis von \mathbb{N} zu \mathbb{R} lässt sich wie folgt beschreiben:

Theorem (Cantor): Es gibt keine Bijektion $f : \mathbb{N} \rightarrow \mathbb{R}$.

Beweis. Wir zeigen, dass für jede Funktion $f : \mathbb{N} \rightarrow \mathbb{R}$ ein Element in \mathbb{R} existiert, das nicht im Bild von f liegt. Stellen wir uns eine Auflistung vor:

$$\begin{aligned} f(0) &= 5.0000\dots \\ f(1) &= 3.1415\dots \\ f(2) &= -7.7777\dots \\ f(3) &= 0.0010\dots \end{aligned}$$

Wir konstruieren eine Zahl z . Der ganzzahlige Anteil von z sei 0 und die $(n + 1)$ te Dezimale ist $x + 1$, falls die $(n + 1)$ te Dezimale von $f(n)$ gleich x ist und $1 \leq x < 9$. Sonst ist die $(n + 1)$ te Dezimale gleich 0. Wir erhalten:

$$z = 0.1581\dots$$

Die reelle Zahl z liegt nicht in $\text{range}(f)$, da sie von $f(n)$ an der $(n + 1)$ ten Dezimalen verschieden ist. \square

Die im Beweis verwendete Methode heißt »Diagonalisierung«. Damit ist gezeigt, dass \mathbb{R} überabzählbar ist.

Was hat das mit Berechenbarkeit zu tun? Wir werden sehen:

1. Die Menge aller Programme (Turing – Maschinen, ...), die Funktionen von \mathbb{N} nach \mathbb{N} berechnen, ist abzählbar.
2. Die Menge aller Funktionen $\mathbb{N} \rightarrow \mathbb{N}$ ist überabzählbar.

Aussage 1 wird später bewiesen. Aussage 2 folgt aus der Überlegung:

$$(0, 1) \preceq \{f \mid f \text{ ist Funktion von } \mathbb{N} \rightarrow \mathbb{N}\}$$

mit der injektiven Abbildung, die jeder Zahl z eine Funktion f_z zuordnet. Sei z.B. $z = 0.37658$, dann sei $f_z(n)$ gleich der $(n + 1)$ ten Dezimalen: $f_z(0) = 3, f_z(1) = 7, \dots$ Hieraus folgt, dass die Menge aller Funktionen $\mathbb{N} \rightarrow \mathbb{N}$ mindestens so groß ist wie \mathbb{R} , d.h. mindestens überabzählbar. Es gilt sogar Kardinalitätsgleichheit mit \mathbb{R} ($= 2^{\mathbb{N}}$).

Wir folgern: Es gibt Funktionen, die nicht berechenbar sind.

1.2 Syntax und Semantik von WHILE – Programmen

Wir benötigen einen Begriff der Berechenbarkeit (Entscheidbarkeit) präzise formalisiert. Hierbei gibt es folgende Möglichkeiten:

- Programme / Flussdiagramme
- Maschinen (TM)
- Grammatiken
- λ – Kalkül
- Markov – Algorithmen
- Gleichungssysteme
- partielle rekursive Funktionen

Alle diese Ansätze sind äquivalent! Dies führt zur **Church – Turing – These**: »Die durch diese Ansätze charakterisierte Klasse von Funktionen stimmt mit der Klasse der von uns als intuitiv berechenbar angesehenen Funktionen überein.«

Bemerkung: Es gibt Modelle, die die Church – Turing – These brechen (z.B. interaktive Turing – Maschinen, Berechnungen mit Hilfe von Schwarzen Löchern).

Wir konzentrieren uns auf Programme. Ein WHILE – Programm ist ein Wort über dem Zeichenvorrat

x_1, x_2, \dots	... Variablen
0	... Null
succ, pred	... Nachfolger und Vorgänger
(,)	... Klammerung
:=	... Zuweisung
;	... Hintereinanderausführung
while, \neq , do, od	... While-Anweisung

Die Menge der WHILE – Programme hat folgende Syntax:

$$\begin{aligned} \langle \text{Prog} \rangle &::= \langle \text{Zuweisung} \rangle \mid \langle \text{Prog} \rangle ; \langle \text{Prog} \rangle \mid \text{while } \langle \text{Var} \rangle \neq \langle \text{Var} \rangle \text{ do } \langle \text{Prog} \rangle \text{ od} \\ \langle \text{Zuweisung} \rangle &::= \langle \text{Var} \rangle := 0 \mid \langle \text{Var} \rangle := \text{succ}(\langle \text{Var} \rangle) \mid \langle \text{Var} \rangle := \text{pred}(\langle \text{Var} \rangle) \\ \langle \text{Var} \rangle &::= x_1 \mid x_2 \mid x_3 \mid \dots \end{aligned}$$

Bemerkung: pred realisiert die modifizierte Subtraktion von 1:

$$\text{pred}(n) = \begin{cases} n - 1 & \text{falls } n \geq 1 \\ 0 & \text{sonst} \end{cases}$$

Vor Programmablauf ist eine Anfangsbelegung der Variablen definiert worden. Die Semantik der Programme ist offensichtlich. Für die While – Anweisung `while x != y do P od` gilt, dass das Programm P so lange ausgeführt wird, bis die Werte von x und y gleich sind. Damit berechnet jedes WHILE – Programm mit k Variablen eine j – stellige Funktion $\varphi_P : \mathbb{N}^j \rightarrow \mathbb{N}$ bei Eingabevektor (a_1, a_2, \dots, a_j) vermöge:

1. $\varphi_P(a_1, a_2, \dots, a_j, 0, \dots, 0) = b$, wobei b der Inhalt der Variablen x_1 ist, falls P terminiert und $j \leq k$.
2. $\varphi_P(a_1, a_2, \dots, a_k) = b$, wobei b der Inhalt der Variablen x_1 ist, falls P terminiert und $k < j$ ist.

Falls P auf der Menge *nicht* terminiert, dann ist der Wert von $\varphi_P(a_1, a_2, \dots, a_j)$ undefiniert (\perp).

Da die Syntax sehr eingeschränkt ist, ist es sinnvoll, Makros zuzulassen, z.B.

- a. `z := x`
- b. `z := n`
- c. `z := x + y`
- d. `z := x - y` (modifizierte Subtraktion)
- e. `z := x * y`
- f. ...

Makros für Kontrollflussanweisungen:

- a. `while C do P od`
- b. `if C then P1 else P2 fi`
- c. `if C then P fi`
- d. `repeat P until C`, wobei C eine beliebige Boolesche Formel über den Variablen ist.

Beispiel:

1. WHILE – Programme können unter Umständen nicht terminieren:

```

1  x1 := 1;
2  while x1 != 0 do x1 := succ( x1 ) od

```

2. Berechnung von $z := x + y$:

```

1  z := succ( x );
2  z := pred( z );
3  u := 0;
4  while u != y do
5    z := succ( z );
6    u := succ( u );
7  od

```

Definition: Eine Funktion $\psi : \mathbb{N}^j \rightarrow \mathbb{N}$ heißt **EFFEKTIV BERECHENBAR** oder **BERECHENBAR**, falls es ein WHILE – Programm P gibt mit $\psi = \varphi_P$. Wenn wir die Anzahl der Eingabevariablen benötigen, schreiben wir $\varphi_P^{(j)} : \mathbb{N}^j \rightarrow \mathbb{N}$.

Beispiel:

```

1  while x1 != x2 + x3 do
2    x1 := succ( x1 )
3    x2 := pred( x2 )
4  od

```

Es gilt $\varphi_P^{(3)}(3, 2, 1) = 3$, $\varphi_P^{(3)}(1, 2, 3) = 3$, $\varphi_P^{(3)}(1, 1, 1) = \perp$. Andererseits ist x_3 zu Null zu setzen bei $\varphi_P^{(2)} : \mathbb{N}^2 \rightarrow \mathbb{N}$. Damit ist $\varphi_P^{(2)}(1, 2) = \perp$.

1.2.1 Aufzählbarkeit von WHILE – Programmen

Man kann sich leicht überzeugen, dass man jedem While-Programm eine Zahl zuweisen kann, z.B. `while` \rightarrow 100000 (Binärzahl)
`do` \rightarrow 100001
 ...

Offensichtlich kann man jedes Zeichen durch die entsprechende Binärzahl ersetzen, um eine Zahl für das gesamte Programm zu erhalten. Damit können wir eine Standardaufzählung von WHILE – Programmen konstruieren: P_n ist das Programm, welches entsteht, wenn man n in Binärdarstellung umwandelt und jedem Block gemäß obiger Tabelle übersetzt. Nicht jede Zahl entspricht einem syntaktisch korrekten Programm. Jedes »Nicht – Programm« wird durch das leere Programm ersetzt. Dieses Verfahren wird **ARITHMETISIERUNG DER SYNTAX** oder **GÖDELISIERUNG** genannt. Damit erhalten wir eine Aufzählung aller berechenbaren Funktionen.

Bemerkung: Es gibt andere Formen der Aufzählung. Diese sind irgendwie »äquivalent« (Rogers Theorem).

Theorem (Nicht – Berechenbarkeit des Halteproblems) Es gibt keine totale und berechenbare Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ mit

$$f(i) = \begin{cases} 1 & \text{falls } \varphi_i(i) \text{ anhält} \\ 0 & \text{sonst} \end{cases}$$

Beweis. Annahme: f ist berechenbar, d.h. es gibt ein j mit $\varphi_j = f$. Was passiert bei $f(j)$? Betrachte folgendes Programm mit f als Makro:

```

1  while f( x1 ) = 1 do
2    x1 := x1
3  od
4  x1 := 1

```

Dieses Programm hat einen Index e . Was passiert, wenn obiges Programm auf seinem Index eine Berechnung durchführt? $\varphi_e(e)$ hält an, falls $f(e) = 0$. Dies impliziert, dass $\varphi_e(e)$ nicht anhält. Andererseits: hält $\varphi_e(e)$ nicht an, so muss $f(e) = 1$ sein. Dies ist wieder ein Widerspruch.

In beiden Fällen erhalten wir einen Widerspruch. Damit ist f nicht berechenbar. \square

Nachfolgendes Theorem ist das Hauptresultat dieses Abschnitts, welches zeigt, dass es »universelle« Funktionen gibt. Das heißt, es gibt einen Index u mit

$$\varphi_u^{(2)}(x, y) = \varphi_x(y) \quad \forall x, y$$

Hierbei ist $\varphi_u^{(2)}$ eine Art Interpreter für WHILE – Programme.

Theorem (Enumerationstheorem) Für jedes $j \geq 1$ existiert eine berechenbare Funktion $\Phi : \mathbb{N}^{j+1} \rightarrow \mathbb{N}$, welche universell bezüglich der Aufzählung j – stelliger Funktionen $\varphi_0^{(j)}, \varphi_1^{(j)}, \dots$ ist, d.h. für alle $e \in \mathbb{N}$ und $(a_1, a_2, \dots, a_j) \in \mathbb{N}^j$ gilt:

$$\Phi(e, a_1, a_2, \dots, a_j) = \varphi_e^{(j)}(a_1, a_2, \dots, a_j)$$

Beweis. (Skizze) Informelle Beschreibung von Φ :

1. Bei Eingabe $(e, a_1, a_2, \dots, a_j)$ »schreibe« das Programm P_e auf.
2. Führe P_e auf der Eingabe (a_1, a_2, \dots, a_j) aus.
3. Falls P_e terminiert, gib als Ergebnis den Wert der ersten Variablen zurück.

Die exakte Ausführung dieser Schritte ist sehr aufwändig. \square

Mitteilung: Es gibt ein Makro (bei WHILE – Programmen) $x := \Phi(y, z)$ mit folgender Eigenschaft: Seien i und j die Werte von y und z , dann gilt: Falls $\varphi_i(j)$ undefiniert ist, dann terminiert $\Phi(y, z)$ nicht. Sonst bekommt x den Wert $\varphi_i(j)$ zugewiesen und y und z bleiben unverändert.

1.2.2 Techniken der Theorie der Berechenbarkeit

Bisherige Möglichkeiten, um zu zeigen, dass eine Funktion (nicht) berechenbar ist:

1. explizites Programmieren (eventuell mit dem Enumerationstheorem)
2. informelle Beschreibung (Verweis auf Church – Turing – These)
3. Diagonalisierung

Weitere Möglichkeiten:

1. Parametrisierung: Sei $\Theta(x, y)$ berechenbar. Halte $x = c$ fest. Welchen Index hat $\Theta'(y) = \Theta(c, y)$?
2. Reduktion: Löse φ mit Hilfe von ψ .

Beispiel

i. Betrachte

$$\psi_1(x, y) = \begin{cases} y & \text{falls } \varphi_x(x) \text{ terminiert} \\ \perp & \text{sonst} \end{cases}$$

Funktion $\psi_1(x, y)$ ist berechenbar mit folgendem Programm:

```

1  x1 := PHI( x1, x1 );
2  x1 := x2

```

ii. Betrachte

$$\psi_2(x, y) = \begin{cases} y & \text{falls } \varphi_x(x) \text{ terminiert} \\ 0 & \text{sonst} \end{cases}$$

Funktion $\psi_2(x, y)$ ist nicht berechenbar, da sonst das Halteproblem lösbar wäre — setze $y = 1$.

Bei (ii) wurde Parametrisierung und Reduktion verwendet.

Betrachte die Berechnung

$$\varphi_i^{(m+n)}(y_1, y_2, \dots, y_m, z_1, z_2, \dots, z_n)$$

falls y_1, y_2, \dots, y_m festgehalten werden.Dann kann man $\varphi_i^{(m+n)}$ an der Stelle $(y_1, y_2, \dots, y_m, z_1, z_2, \dots, z_n)$ mit dem Programm $P_{i'} = Q_i P_i$ berechnen, wobei Q folgendes Programm ist:

```

1  x[m+n+1] := x[1];
2  ...
3  x[m+n+n] := x[n];
4  x[1] := y[1];
5  ...
6  x[m] := y[m];
7  x[m+1] := x[m+n+1];
8  ...
9  x[m+n] := x[m+n+n]
10 x[m+n+1] := 0;
11 ...
12 x[m+n+n] := 0;

```

das als Eingabe z_1, \dots, z_n erhält. Falls (y_1, y_2, \dots, y_m) und i gegeben sind, dann lässt sich i' berechnen, d.h.

$$i' = S_n^m(i, y_1, y_2, \dots, y_m)$$

für eine totale und berechenbare Funktion $S_n^m : \mathbb{N}^{m+1} \rightarrow \mathbb{N}$. Offensichtlich gilt

$$\varphi_{S_n^m(i, y_1, y_2, \dots, y_m)}^{(n)}(z_1, \dots, z_n) = \varphi_i^{(m+n)}(y_1, \dots, y_m, z_1, \dots, z_n)$$

Damit gilt folgendes Theorem:

Theorem (s-m-n – Theorem) Es gibt eine totale und berechenbare Funktion

$$i' = S_n^m(i, y_1, y_2, \dots, y_m)$$

so dass

$$\varphi_{S_n^m(i, y_1, y_2, \dots, y_m)}^{(n)}(z_1, \dots, z_n) = \varphi_i^{(m+n)}(y_1, \dots, y_m, z_1, \dots, z_n)$$

gilt.

Beispiel

i. Betrachte Komposition von Funktionen. Sei $\Theta : \mathbb{N}^3 \rightarrow \mathbb{N}$, definiert vermöge

$$\Theta(m, n, x) = \varphi_m \circ \varphi_n(x)$$

Θ ist berechenbar und hat Index e , d.h.

$$\Theta(m, n, x) = \varphi_e(m, n, x) = \varphi_m \circ \varphi_n(x)$$

Das s-m-n – Theorem liefert

$$\varphi_{S_1^2(e, m, n)}(x) = \Theta(m, n, x) = \varphi_m \circ \varphi_n(x)$$

ii. Finde eine effektive Aufzählung der Indizes für folgende Sequenz von Funktionen: $\{1, x, x^2, x^3, \dots\}$. Angenommen, $f(y, x) = x^y$ hat Index n . Dann gibt es eine totale und berechenbare Funktion $S_1^1 : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit der Eigenschaft

$$f(y, x) = \varphi_n^{(2)}(y, x) = \varphi_{S_1^1(n, y)}^{(1)}(x)$$

Die gesuchten Indizes sind $S_1^1(n, 0), S_1^1(n, 1), \dots$

Unentscheidbare Probleme

Ein Ja / Nein – Problem (darstellbar als Funktion $f : \mathbb{N} \rightarrow \{0, 1\}$) ist entscheidbar, falls es einen Algorithmus (WHILE – Programm) gibt, der bei *jeder* Eingabe terminiert und die richtige Antwort (ja / nein bzw. 1 / 0) gibt (Stichwort: totale berechenbare Funktion). Beispiel für ein unentscheidbares Problem ist das Halteproblem.

Theorem Es gibt eine totale Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$, die *nicht* berechenbar ist.

Beweis. Per Diagonalisierung (Trick: $f(i) = f_i(i) + 1$, wobei f_0, f_1, \dots eine Aufzählung der totalen berechenbaren Funktionen ist). \square

Theorem Sei f_0, f_1, \dots eine effektive Aufzählung, in der jede Funktion f_i eine totale und berechenbare Funktion von \mathbb{N} nach \mathbb{N} ist. Dann gibt es eine totale und berechenbare Funktion, die nicht in obiger Aufzählung erscheint.

Beweis. Per Diagonalisierung \square

Korollar Es gibt *keine effektive* Aufzählung aller totalen und berechenbaren Funktionen $\mathbb{N} \rightarrow \mathbb{N}$.

Hinweis: Partielle Funktionen sind das »RICHTIGE« Konzept für Berechenbarkeit.

Reduktion funktioniert wie folgt: Als Beispiel lösen wir das Aufzählungsproblem totaler und berechenbarer Funktionen mit Hilfe des Totalitätsproblems:

Theorem Es gibt keinen Algorithmus (WHILE – Programm), der entscheidet, ob für einen gegebenen Index i die Funktion $\varphi_i : \mathbb{N} \rightarrow \mathbb{N}$ total ist oder nicht.

Beweis. Definiere

$$\text{total}(i) = \begin{cases} 1 & \text{falls } \varphi_i : \mathbb{N} \rightarrow \mathbb{N} \text{ total} \\ 0 & \text{sonst} \end{cases}$$

Angenommen, die Funktion total ist berechenbar. Betrachte $g : \mathbb{N} \rightarrow \mathbb{N}$:

$g(0)$: kleinstes i , so dass $\text{total}(i) = 1$

$g(n+1)$: kleinstes i größer als $g(n)$, so dass $\text{total}(i) = 1$.

Für g kann man sich ein WHILE – Programm überlegen, das $\text{total}(\cdot)$ als Subroutine enthält. Damit erhalten wir eine effektive Aufzählung aller totalen und berechenbaren Funktionen

$$\varphi_{g(0)}, \varphi_{g(1)}, \varphi_{g(2)}, \dots$$

Dies steht im Widerspruch zu obigem Korollar. \square

Ein weiteres Beispiel:

Theorem (Unentscheidbarkeit des allgemeinen Halteproblems) Es gibt *keinen* Algorithmus, der entscheidet, ob bei gegebenem Index i und $a \in \mathbb{N}$ die Funktion $\varphi_i : \mathbb{N} \rightarrow \mathbb{N}$ auf Eingabe a definiert ist oder nicht.

Beweis. Sei

$$h(i, j) = \begin{cases} 1 & \text{falls } \varphi_i(j) \text{ definiert} \\ 0 & \text{sonst} \end{cases}$$

Es gilt $\text{halt}(i) = h(i, i)$, damit reduziert sich das Halteproblem auf das allgemeine Halteproblem. Hieraus folgt, dass das allgemeine Halteproblem unentscheidbar ist. \square

Folgende Probleme sind ebenfalls unentscheidbar (ohne Beweis):

Theorem Die Funktion

$$e(i) = \begin{cases} 1 & \text{falls } \varphi_i \equiv \text{id} \\ 0 & \text{sonst} \end{cases}$$

ist nicht berechenbar.

Bemerkung: Dieses Theorem gilt für jede beliebige totale und berechenbare Funktion anstelle von id .

Theorem (Unentscheidbarkeit der Äquivalenz) Es gibt keinen Algorithmus, der entscheidet, ob für gegebene Indizes i und j $\varphi_i \equiv \varphi_j$ gilt.

Wie verwende ich Reduktion?

Problem A reduziert auf B (A wird mit Hilfe von B gelöst). Dann gilt:

- B berechenbar $\Rightarrow A$ berechenbar
- B entscheidbar $\Rightarrow A$ berechenbar (A entscheidbar, falls Totalität gegeben ist)

Beachte »Kontraposition«.

1.2.3 Das Rekursionstheorem und Eigenschaften von Aufzählungen

Frage: Gibt es eine Funktion $\varphi_e : \mathbb{N} \rightarrow \mathbb{N}$ mit der Eigenschaft $\varphi_e(x) = e$ für alle x ?

Das Problem ist, dass eine direkte Berechnung von e zu einer zu großen Programmnummer führt. Hierbei hilft folgendes Theorem:

Theorem (Rekursionstheorem) Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine beliebige totale und berechenbare Funktion. Dann gibt es für jedes j ein n so, dass P_n und $P_{f(n)}$ (möglicherweise verschieden) die gleiche j -stellige Funktion ausrechnen, d.h. es gilt

$$\varphi_n^{(j)} = \varphi_{f(n)}^{(j)}$$

Bemerkung: $\varphi_n = \varphi_{f(n)}$ ist ein »Fixpunkt«. Beachte: Im Allgemeinen gilt $n \neq f(n)$.

Beweis. Wir beweisen das Theorem für $j = 1$. Sei $P_{g(i)} = x_2 := \Phi(i, i); x_1 := \Phi(x_2, x_1);$, wobei $x := \Phi(y, z)$ der »universelle« Interpreter ist. Beachte, dass die Funktion g wegen des s-n-m – Theorems total und berechenbar ist. Man sieht:

$$\varphi_{g(i)}(j) = \begin{cases} \varphi_{\varphi_i(i)}(j) & \text{falls } \varphi_i(i) \text{ terminiert} \\ \perp & \text{sonst} \end{cases}$$

Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine totale und berechenbare Funktion. Betrachte $f \circ g$. Diese Funktion ist wieder total und berechenbar, d.h. es gibt einen Index m mit $\varphi_m = f \circ g$. Dann ist

$$\varphi_{g(m)}(j) = \varphi_{f(m)}(j)$$

da φ_m total ist (daher terminiert $\varphi_m(m)$). Da $\varphi_m = f \circ g$ ist, folgt

$$\varphi_{g(m)}(j) = \varphi_{\varphi_m(m)}(j) = \varphi_{f \circ g(m)}(j)$$

Setze $n := g(m)$. Dann gilt

$$\varphi_n = \varphi_{f(n)}$$

□

Was nützt das?

Beispiel Funktion f sei wie folgt definiert:

1. Ersetze jede `pred` - Anweisung durch eine `succ` - Anweisung und umgekehrt.
2. Ersetze `succ(x)` durch das Makro `2 * x`.
3. Aweisungen umordnen.

In allen Fällen liefert das Rekursionstheorem, dass es ein n gibt mit $\varphi_n = \varphi_{f(n)}$.

Die Funktion f heißt auch PROGRAMMTRANSFORMATIONSFUNKTION (PROGRAM REWRITING FUNCTION).

Beachte: Im Fall $\varphi_n = \varphi_{f(n)}$ (n aus dem Rekursionstheorem) gilt im allgemeinen *nicht*

$$\varphi_{n'} = \varphi_{f(n')} \quad \text{für} \quad \varphi_n = \varphi_{n'}$$

Es gilt:

Theorem Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ total und berechenbar. Dann gibt es unendlich viele Indizes n mit $\varphi_n = \varphi_{f(n)}$.

Beweis. (für $j = 1$) Betrachte die $(k + 1)$ sten Funktionen. Wähle ein l mit $\varphi_l \neq \varphi_0, \varphi_l \neq \varphi_1, \dots, \varphi_l \neq \varphi_k$. Definiere $g : \mathbb{N} \rightarrow \mathbb{N}$ vermöge

$$g(x) = \begin{cases} l & \text{falls } x \leq k \\ f(x) & \text{falls } x > k \end{cases}$$

Die Funktion g ist total und berechenbar und hat einen Fixpunkt n (bezüglich des Rekursionstheorems), d.h. $\varphi_n = \varphi_{g(n)}$. Wenn $n \leq k$ ist, gilt $\varphi_{g(n)} = \varphi_l \neq \varphi_n$. Dies ist Widerspruch zu $\gg n$ ist Fixpunkt«. Also ist $n > k$ und damit $g(n) = f(n)$, d.h. n ist auch Fixpunkt für f , d.h. $\varphi_n = \varphi_{g(n)} = \varphi_{f(n)}$. Beachte, dass n beliebig groß werden kann, da k beliebig wählbar ist. □

Beispiele für unerwartete Programmeigenschaften:

1. Es gibt ein Programm, das seine eigene Beschreibung ausgibt, also $\varphi_e(x) = e$. Betrachte $f : \mathbb{N} \rightarrow \mathbb{N}$ mit $P_{f(i)} = x_1 := i$; ($f(i)$ ist Gödelnummer des Programms $x_1 := i$). Das Rekursionstheorem sagt, dass es ein n gibt mit $\varphi_n = \varphi_{f(n)} \equiv n$.
2. Es gibt eine Zahl i^* so, dass für alle j gilt

$$\varphi_{i^*}(j) = \varphi_j(i^*)$$

Definiere $P_{f(i)} = x_1 := \Phi(x_1, i)$; Dann gilt

$$\varphi_{f(i)}(j) = \begin{cases} \varphi_j(i) & \text{falls } \varphi_j(i) \text{ terminiert} \\ \perp & \text{sonst} \end{cases}$$

Das Rekursionstheorem liefert einen Fixpunkt i^* für f , d.h.

$$\varphi_{i^*}(j) = \varphi_{f(i^*)}(j) = \varphi_j(i^*)$$

Obige Beispiele beinhalten den Begriff der SELBSTREFERENZIERUNG (eigene Gödelnummer kommt rechts und links in der Definition vor).

Beachte: Das Rekursionstheorem folgt aus dem Enumerations- und dem s-n-m – Theorem.

Wir führen den Begriff des »akzeptablen Programmiersystems« ein. Sei A_0, A_1, A_2, \dots eine effektive Aufzählung von Algorithmen (bezüglich einer algorithmischen Spezifikation). Für jedes j assoziieren wir eine effektive Indizierung der j - stelligen Funktionen $\alpha_0^{(j)}, \alpha_1^{(j)}, \dots$, wobei A_i die Funktion $\alpha_i^{(j)}$ berechne. Dann ist A_0, A_1, \dots ein akzeptables Programmiersystem und die dazugehörige Indizierung eine akzeptable Indizierung, falls gilt:

1. Jede intuitiv berechenbare Funktion $\mathbb{N} \rightarrow \mathbb{N}$ findet sich in $\{\alpha_i^{(j)} : i, j \in \mathbb{N}\}$ (Stichwort: Church – Turing – These)
2. Für jedes k hat die Folge $\alpha_0^{(k)}, \alpha_1^{(k)}, \dots$ eine universelle Funktion $\alpha_u^{(k+1)} \in \{\alpha_i^{(k+1)} : i, j \in \mathbb{N}\}$.
3. Für jedes m und n gibt es eine totale Funktion s_n^m in $\alpha_0^{(m+1)}, \alpha_1^{(m+1)}, \dots$ so, dass

$$\alpha_{s_n^m(i, y_1, \dots, y_m)}^{(n)}(z_1, \dots, z_n) = \alpha_i^{(m+n)}(y_1, \dots, y_m, z_1, \dots, z_n)$$

Es gilt:

Theorem Seien A_0, A_1, A_2, \dots und B_0, B_1, B_2, \dots akzeptable Programmiersysteme. Für jedes j gibt es eine totale und berechenbare (injektive) Funktion $t : \mathbb{N} \rightarrow \mathbb{N}$ so, dass

$$\alpha_i^{(j)} = \beta_{t(i)}^{(j)}$$

Beweis. (für $j = 1$). Dass t injektiv ist, wird *nicht* bewiesen (aufwändig). Betrachte $\alpha_u^{(2)} : \mathbb{N}^2 \rightarrow \mathbb{N}$, die universelle Funktion für $\alpha_0, \alpha_1, \alpha_2, \dots$. Wegen Bedingung (1) gibt es einen Index k mit $\alpha_u^{(2)} = \beta_k^{(2)}$. Dann gilt:

$$\alpha_i(x) = \alpha_u^{(2)}(i, x) = \beta_k^{(2)}(i, x) \stackrel{(3)}{=} \beta_{s_1^1(k, i)}(x)$$

Setze $t(i) = s_1^1(k, i)$ und das Ergebnis folgt. \square

Das folgende Theorem zeigt, dass die Theorie der Berechenbarkeit unabhängig vom gewählten Modell ist.

Theorem (Rogers Isomorphietheorem) Seien A_0, A_1, A_2, \dots und B_0, B_1, B_2, \dots akzeptable Programmiersysteme. Für jedes j gibt es eine bijektive, totale und berechenbare Funktion $g : \mathbb{N} \rightarrow \mathbb{N}$ mit

$$\alpha_i^{(j)} = \beta_{g(i)}^{(j)}$$

Mitteilung: Rogers Isomorphietheorem ist das Schröder – Bernstein – Theorem der Berechenbarkeit.

Beweis. Eine endliche Menge von geordneten Paaren $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ ist eine endliche Korrespondenz zwischen den A_i s und B_i s, falls:

1. $i \neq j \Rightarrow x_i \neq x_j \wedge x_j \neq y_j$
2. $\alpha_{x_i} = \beta_{y_i}$

für alle $1 \leq i, j \leq n$.

Es gibt eine effektive Prozedur, die bei Eingabe einer endlichen Korrespondenz $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ und einem Wert $x' \notin \{x_1, x_2, \dots, x_n\}$ ein y' berechnet, so dass $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n), (x', y')$ eine endliche Korrespondenz ist. Sei $t : \mathbb{N} \rightarrow \mathbb{N}$ eine injektive, totale und berechenbare Funktion mit $\alpha_x = \beta_{t(x)}$ — die gibt es nach obigem Theorem. Für $x' \notin \{x_1, \dots, x_n\}$ betrachte $t(x')$. Falls $\forall i \in \{1, \dots, n\} : t(x') \neq y_i$ ist, dann setze $y' = t(x')$. Anderenfalls gibt es ein y_{i_1} mit $t(x') = y_{i_1}$. Falls $\forall i \in \{1, \dots, n\} : t(x_{i_1}) \neq y_i$, dann setze $y' = t(x_{i_1})$. Anderenfalls gibt es ein y_{i_2} mit $t(x_{i_1}) = y_{i_2}$ etc.

Da t eine injektive Abbildung und die Korrespondenz endlich ist, findet der Algorithmus ein $y' \notin \{y_1, \dots, y_n\}$ in endlicher Zeit mit $\alpha_{x'} = \beta_{y'}$. Beachte, dass $i_1 \neq i_2 \neq i_3 \neq \dots$ gilt.

O.E. sei $j = 1$. Funktion t eine injektive, totale und berechenbare Funktion mit $\alpha_x = \beta_{t(x)}$ und Funktion u ist eine injektive, totale und berechenbare Funktion mit $\beta_y = \alpha_{u(y)}$. Beachte, dass die obige Aussage über die Berechnung von y' und x' und der endlichen Korrespondenz symmetrisch ist. Anstelle von t kann man u verwenden, um zu gegebenem y' und endlicher Korrespondenz ein x' zu berechnen. Wir konstruieren eine Bijektion g wie folgt:

0. Produziere $(0, t(0))$.
1. Falls $t(0) = 0$, gehe zu Schritt 2. Anderenfalls bestimme ein Paar, dessen zweite Komponente 0 ist (mit Hilfe von Funktion u und obigem Algorithmus).
- 2n. Falls n schon in einem Paar als erste Komponente vorkommt, gehe zu Schritt $2n + 1$. Anderenfalls bestimme ein Paar, dessen erste Komponente n ist (mit obigem Algorithmus und t).
- 2n+1. Falls n schon in einem Paar als zweite Komponente vorkommt, gehe zu Schritt $2n + 2$. Anderenfalls bestimme ein Paar, dessen zweite Komponente n ist (mit obigem Algorithmus und u).

□

1.3 LOOP – Programme — einige Bemerkungen

Ein LOOP – Programm ist ein Wort über dem Zeichenvorrat

x_1, x_2, \dots	... Variablen
0	... Null
succ, pred	... Nachfolger und Vorgänger
(,)	... Klammerung
$:=$... Zuweisung
;	... Hintereinanderausführung
$\text{loop}, \text{do}, \text{od}$... Loop-Anweisung

Die Menge der LOOP – Programme hat folgende Syntax:

$$\begin{aligned} \langle \text{Prog} \rangle &::= \langle \text{Zuweisung} \rangle \mid \langle \text{Prog} \rangle ; \langle \text{Prog} \rangle \mid \text{loop} \langle \text{Var} \rangle \text{do} \langle \text{Prog} \rangle \text{od} \\ \langle \text{Zuweisung} \rangle &::= \langle \text{Var} \rangle := 0 \mid \langle \text{Var} \rangle := \text{succ}(\langle \text{Var} \rangle) \mid \langle \text{Var} \rangle := \text{pred}(\langle \text{Var} \rangle) \\ \langle \text{Var} \rangle &::= x_1 \mid x_2 \mid x_3 \mid \dots \end{aligned}$$

Anfangsbelegung und Programmablauf sind wie bei WHILE – Programmen definiert. Die Semantik der LOOP – Programme ist offensichtlich. Die LOOP – Anweisung $\text{loop } x \text{ do } P \text{ od}$ bedeutet, dass das Programm P genau t Mal ausgeführt wird, falls x den Wert t vor Eintritt in die LOOP – Anweisung hat. Wie im Fall von WHILE – Programmen berechnet jedes LOOP – Programm P mit k Variablen eine (j – stellige) Funktion $\psi_P : \mathbb{N}^j \rightarrow \mathbb{N}$ mit dem Ergebnis in der ersten Variablen.

Theorem Alle LOOP – Programme terminieren für beliebige Anfangsbelegungen.

Beweis. (Idee) Strukturelle Induktion. □

Definition: Eine Funktion $f : \mathbb{N}^j \rightarrow \mathbb{N}$ heißt LOOP – BERECHENBAR, falls es ein LOOP – Program P gibt mit $f = \psi_P$. Wenn wir die Stelligkeit benötigen, schreiben wir $f^{(j)} = \psi_P^{(j)}$.

Korollar Jede LOOP – berechenbare Funktion ist total.

Alle »normalen« Funktionen sind LOOP – berechenbar (z.B. Addition, Multiplikation, Potenz, ...). Es gibt totale Funktionen, aber *nicht* LOOP – berechenbar sind. Anbei zwei Beispiele.

Die »busy beaver« – Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ für **LOOP – Programme** ist definiert vermöge $f(n) =$ »größte Zahl, welche ein LOOP – Programm P mit Länge $1 \leq j \leq n$ und den Anfangsbelegungen $0 \leq x_1, x_2, \dots \leq j$ berechnet«, wobei die Länge eines LOOP – Programms wie folgt definiert ist:

1. Falls P eine Zuweisung ist, dann $|P| = 1$.
2. Falls $P = P_1; P_2$ ist, dann $|P| = |P_1| + |P_2|$.
3. Falls $P = \text{loop } x \text{ do } P_1 \text{ od}$, dann $|P| = |P_1| + 1$.

Die Funktion f ist WHILE – berechenbar (zähle alle Programme bis zur Länge n auf — das sind endlich viele — und lasse diese auf allen Belegungen $0 \leq x_1, x_2, \dots \leq j$ laufen). Sammle alle Ergebnisse und bestimme das Maximum.

Theorem Die »busy beaver« – Funktion für LOOP – Programme ist nicht LOOP – berechenbar.

Beweis. Die Funktion f ist total und wächst streng monoton. Annahme: f ist LOOP – berechenbar. Dann ist auch $g(n) = f(n + 1)$ LOOP – berechenbar mit dem Programm

```
1  x1 := succ( x1 );
2  x1 := f( x1 );
```

Sei P das LOOP – Programm, welches g berechnet, und sei k die Länge von P . Dann gilt

$$g(k) = f(k + 1) \quad \text{und} \quad g(k) \leq f(k)$$

Hieraus folgt $f(k) \geq g(k) = f(k + 1)$. Dies ist ein Widerspruch zur strengen Monotonie von f . Daher ist f nicht LOOP – berechenbar. \square

Mitteilung: Die »busy beaver« – Funktion für WHILE – Programme ist nicht WHILE – berechenbar. Beweis: analog zu oben.

Ein weiteres Beispiel für eine totale und nicht LOOP – berechenbare Funktion ist die Ackermann – Funktion $A : \mathbb{N}^2 \rightarrow \mathbb{N}$. Sie hat folgende Definition:

$$\begin{aligned} A(0, y) &= y + 1 \\ A(x + 1, y) &= A(x, 1) \\ A(x + 1, y + 1) &= A(x, A(x + 1, y)) \end{aligned}$$

Wenn man die letzte Zeile entwickelt, erhält man

$$A(x + 1, y + 1) = \underbrace{A(x, A(x, \dots A(x, 1) \dots))}_{y+2 \text{ mal } A} \quad (1.1)$$

Betrachte $A(x, y)$ als Familie von einstelligigen Funktionen $A_x(y)$. Dann ist (1.1) äquivalent zu

$$A_{x+1} = A_x^{y+1}(1)$$

Einige Werte:

$$\begin{aligned} A_0(y) &= A(0, y) = y + 1 && \text{succ} \\ A_1(y) &= A_0^{y+1}(1) = y + 2 && (\text{erhöhe } 1 \text{ } y + 1 \text{ mal um } 1 \text{ (succ)}) \\ A_2(y) &= A_1^{y+1}(1) = \text{»erhöhe } 1 \text{ um } 2 \text{ genau } y + 1 \text{ mal«} = 2y + 3 \\ A_3(y) &= A_2^{y+1}(1) = \text{»starte mit } 1, \text{ dann verdoppeln und } 3 \text{ addieren, genau } y + 1 \text{ mal«} \\ &= 2(2^{y+1} - 1) + 1 \end{aligned}$$

Lemma Für jedes LOOP – Programm P gibt es eine Konstante k , so dass für alle $n \geq k$ gilt:

$$\psi_P(n) < A(k, n)$$

Beweis. (Idee): strukturelle Induktion über den Aufbau von LOOP – Programmen. Schwieriger Fall ist die LOOP – Schleife: Wenn P die Funktion $f(n)$ berechnet, dann berechnet $\text{loop } x \text{ do } P \text{ od}$ die Funktion $f \circ f \circ \dots \circ f = f^n(n)$ (n -fache Anwendung). \square

Theorem Die Ackermannfunktion ist nicht LOOP – berechenbar.

Beweis. Annahme: A ist LOOP – berechenbar. Dann ist auch $g(n) = A(n, n)$ LOOP – berechenbar. Sei P das Programm, welches g berechnet. Wegen obigem Lemma gibt es ein k mit $g(n) = \psi_P(n) < A(k, n)$ für alle $n \geq k$. Für $n = k$ folgt $g(k) = \psi_P(k) < A(k, k) = g(k)$. Das ist ein Widerspruch. \square

Mitteilung: Mit ähnlichen Mitteln kann man zeigen, dass es kein LOOP – universelles LOOP – Programm, d.h. $\psi_u^{(2)}(x, y) = \psi_x^{(1)}(y)$, gibt.

1.4 Berechenbare Eigenschaften von Mengen

Sei $A \subseteq \mathbb{N}^k$. Die einfachste Möglichkeit, A darzustellen, ist die CHARAKTERISTISCHE FUNKTION $\chi_A : \mathbb{N}^k \rightarrow \{0, 1\}$, die wie folgt definiert ist:

$$\chi_A(x) = \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{sonst} \end{cases}$$

Definition: Eine Menge $A \subseteq \mathbb{N}^k$ heißt ENTSCHEIDBAR (LÖSBAR, REKURSIV), falls die charakteristische Funktion χ_A eine totale und berechenbare Funktion ist.

Beispiele

1. Die Menge $\{i : \varphi_i : \mathbb{N} \rightarrow \mathbb{N}\}$ ist entscheidbar.
2. Die Menge $\{i : \varphi_i : \mathbb{N} \rightarrow \mathbb{N} \text{ total}\}$ ist nicht entscheidbar (s. Abschnitt 1.2).

Es folgen Eigenschaften rekursiver Mengen. Eine Menge $A \subseteq \mathbb{N}^k$ ist co-endlich, falls $\bar{A} := \mathbb{N}^k \setminus A$ endlich ist.

Theorem (Eigenschaften rekursiver Mengen)

1. Falls die Menge A endlich oder co-endlich ist, dann ist A rekursiv.
2. Falls A rekursiv ist, dann ist auch \bar{A} rekursiv.
3. Wenn A_1 und A_2 rekursiv sind, dann sind auch $A_1 \cup A_2$ und $A_1 \cap A_2$ rekursiv.

Beweis. \rightarrow Hausaufgabe \square

Beachte: Abschlusseigenschaften enthalten meistens Implikationen.

Eine Menge ist rekursiv aufzählbar, falls es einen Algorithmus, der alle Elemente der Menge aufzählt (ausgibt). Dies führt zur Definition:

Definition: Eine Menge $A \subseteq \mathbb{N}^k$ ist SEMIENTSCHEIDBAR oder REKURSIV AUFZÄHLBAR, falls entweder $A = \emptyset$ oder $A = \text{range}(f)$ für eine totale und berechenbare Funktion $f : \mathbb{N} \rightarrow \mathbb{N}^k$. f wird als ENUMERATIONSFUNKTION bezeichnet; eine Auflistung der Elemente von A ist $f(0), f(1), f(2), \dots$

Beispiele

1. Menge \mathbb{N} : Enumerationsfunktion $f \equiv \text{id}$
2. Die Menge aller Primzahlen: Enumerationsfunktion $f(i) = p_i$, wobei p_i die i -te Primzahl ist.

Theorem Jede rekursive Menge ist rekursiv aufzählbar.

Beweis. Falls $A = \emptyset$, dann ist A rekursiv aufzählbar per Definition. Sei $A \neq \emptyset$ und $a \in A$. Dann berechnet das WHILE – Programm `if $\chi_A(x_1) \neq 1$ then $x_1 := a$ fi` unter der Annahme, dass χ_A total und berechenbar ist, die Funktion

$$f(n) = \begin{cases} n & \text{falls } n \in A \\ a & \text{falls } n \notin A \end{cases}$$

Offensichtlich gilt $A = \text{range}(f)$. □

Folgendes Theorem beinhaltet eine fundamentale Technik (\gg dovetailing \ll).

Theorem Die Menge $K = \{i : \varphi_i(i) \text{ terminiert}\}$ ist rekursiv aufzählbar, aber nicht rekursiv.

Beweis. K ist nicht rekursiv, da ansonsten das Halteproblem lösbar wäre. Betrachte die bijektive Abbildung $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ aus Abschnitt 1.2 und die Projektionen f_1 und f_2 auf die erste und zweite Komponente ($n = f(x, y); f_1(n) = x$ und $f_2(n) = y$). Wir schreiben folgendes Programm:

```

1  n := 0;
2  while true do
3    x := f1( n );
4    y := f2( n );
5    if Px haelt auf Eingabe x in y Schritten then
6      output x;
7    fi
8    n := succ( n );
9  od

```

Offensichtlich testet der Algorithmus jedes Programm P_i für j Schritte mit $i, j \in \mathbb{N}$ (\gg dovetailing \ll). Definiere eine totale und berechenbare Funktion

$$g(i) = i+1\text{-tes Element in der Ausgabeliste des Algorithmus}$$

Dann sieht man leicht: $K = \text{range}(g)$. □

Theorem (ohne Beweis) Die Menge $\{i : \varphi_i : \mathbb{N} \rightarrow \mathbb{N} \text{ total}\}$ ist nicht rekursiv aufzählbar (und damit auch nicht rekursiv).

Weitere Eigenschaften rekursiver Mengen:

Theorem Die Menge $A \subseteq \mathbb{N}$ ist rekursiv genau dann, wenn A und $\bar{A} (=_{\text{def}} \mathbb{N} \setminus A)$ rekursiv aufzählbar sind.

Beweis. Die Richtung \gg von links nach rechts \ll folgt aus dem Komplementabschluss rekursiver Mengen und der Tatsache, dass jede rekursive Menge auch rekursiv aufzählbar ist. Für die Richtung \gg von rechts nach links \ll argumentieren wir wie folgt: Wenn entweder A oder \bar{A} gleich \emptyset ist, dann ist A offensichtlich rekursiv. Sei nun weder $A = \emptyset$ noch $\bar{A} = \emptyset$. Dann gibt es totale und berechenbare Funktionen f, g mit $A = \text{range}(f)$ und $\bar{A} = \text{range}(g)$. Folgender Algorithmus entscheidet $x \in A$:

Berechne $f(0), g(0), f(1), g(1), \dots, f(i), g(i), \dots$ für $i \in \mathbb{N}$ Wenn x als Wert von f vorkommt, dann $x \in A$, und wenn x als Wert von g vorkommt, dann $x \notin A$.

Die Korrektheit folgt aus $A \cup \bar{A} = \mathbb{N}$ und $A \cap \bar{A} = \emptyset$. □

Korollar Die Menge $\bar{U} = \{i : \varphi_i(i) \text{ divergiert}\}$ ist nicht rekursiv aufzählbar.

Beweis. Folgt unmittelbar aus obigem Theorem. □

Für folgendes Theorem benötigen wir noch eine Definition.

Definition Eine Menge $A \subseteq \mathbb{N}$ heißt rekursiv aufzählbar in aufsteigender Reihenfolge, wenn sie durch eine totale und berechenbare Funktion f rekursiv aufzählbar ist und für $i < j$ folgt $f(i) < f(j)$.

Theorem Eine unendliche Menge $A \subseteq \mathbb{N}$ ist rekursiv genau dann, wenn A rekursiv aufzählbar in aufsteigender Reihenfolge ist.

Beweis. Wenn A rekursiv ist, dann folgt sofort, dass A rekursiv aufzählbar in aufsteigender Reihenfolge ist: Zähle die natürlichen Zahlen auf und berechne dazu $\chi_A(0), \chi_A(1), \chi_A(2), \dots$. Falls $\chi_A(i) = 1$ ist, dann nimm i in die A - Liste auf. Aus dieser Liste kann man eine totale und berechenbare Funktion f definieren:

$$f(i) = \gg(i+1)\text{tes Element der } A \text{ - Liste}\ll$$

mit den notwendigen Eigenschaften ($A = \text{range}(f)$ und $\forall i < j : f(i) < f(j)$).

Für die Umkehrung argumentieren wir wie folgt: Sei f die Funktion, die A in aufsteigender Reihenfolge aufzählt. Um zu entscheiden, ob $x \in A$ ist, genügt es, nacheinander Elemente aus dem Bildbereich zu erzeugen, bis ein Wert $> x$ erscheint. Dann gilt $x \in A$, falls x in der Liste der erzeugten Elemente vorkommt; ansonsten gilt $x \notin A$. \square

Eine einfache Anwendung obigen Theorems ist (ohne Beweis)

Korollar Jede unendliche rekursiv aufzählbare Menge hat eine unendliche rekursive Teilmenge. Es folgt eine alternative Charakterisierung der rekursiv aufzählbaren Mengen.

Theorem

- i. Eine Menge A ist rekursiv aufzählbar genau dann, wenn A Domain einer berechenbaren Funktion f ist, d.h. $A = \text{dom}(f)$.
- ii. Eine Menge A ist rekursiv aufzählbar genau dann, wenn A Range einer berechenbaren Funktion f ist, d.h. $A = \text{range}(f)$.

Beweis. (nur i): Wenn $A = \emptyset$, dann ist A Domain der überall undefinierten Funktion. Sei $A \neq \emptyset$ und $A = \text{range}(f)$ für eine totale und berechenbare Funktion f . Definiere

$$\Theta(i) = \begin{cases} 1 & \text{falls } i \in \text{range}(f) \\ \perp & \text{sonst} \end{cases}$$

Offensichtlich gilt $\text{dom}(\Theta) = \text{range}(f) = A$ und Θ ist berechenbar durch

```

1  x2 := x1;
2  x1 := 1;
3  n := 0;
4  while x2 != f( n ) do
5    n := succ( n );
6  od
```

Für die Umkehrung argumentieren wir wie folgt: Sei $A = \text{dom}(\Theta)$ für eine berechenbare Funktion $\Theta : \mathbb{N} \rightarrow \mathbb{N}$. Wenn $\text{dom}(\Theta) = \emptyset$, dann ist A rekursiv und damit rekursiv aufzählbar. Anderenfalls sei $a \in A$. Betrachte das folgende Programm (f1, f2 Projektionsfunktionen für die Dovetailing - Technik):

```

1  x := f1( x1 );
2  y := f2( x1 );
3  if Theta( x ) haelt in y Schritten then
4    x1 := x
5  else
6    x1 := a
7  fi
```

Wie man leicht sieht, berechnet das Programm eine totale Funktion mit $\text{range}(f) = \text{dom}(\Theta)$. \square

Das Theorem führt zu einer Aufzählung der rekursiv aufzählbaren Mengen der Form

$$\text{dom}(\varphi_0), \text{dom}(\varphi_1), \text{dom}(\varphi_2), \dots$$

oder

$$\text{range}(\varphi_0), \text{range}(\varphi_1), \text{range}(\varphi_2), \dots$$

Diese Aufzählungen stehen in Beziehung zueinander:

Theorem Es gibt totale und berechenbare Funktionen $t_1 : \mathbb{N} \rightarrow \mathbb{N}$ und $t_2 : \mathbb{N} \rightarrow \mathbb{N}$, so dass für alle $i \in \mathbb{N}$ gilt:

- i. $\text{dom}(\varphi_i) = \text{range}(\varphi_{t_1(i)})$
- ii. $\text{range}(\varphi_i) = \text{dom}(\varphi_{t_2(i)})$

Beweis. Hausaufgabe \square

Dies motiviert folgende Definition:

Definition Sei $\varphi_0, \varphi_1, \varphi_2, \dots$ die Standardaufzählung der berechenbaren Funktionen $\mathbb{N} \rightarrow \mathbb{N}$. Dann ist W_0, W_1, W_2, \dots mit $W_i = \text{dom}(\varphi_i)$ eine (Standard-)Aufzählung der rekursiv aufzählbaren Mengen. Diese Definition lässt sich auf j -stellige Funktionen erweitern. Wir schreiben dann $W_i^{(j)} = \text{dom}(\varphi_i^{(j)})$.

Einfache Operationen auf rekursiv aufzählbaren Mengen: Sei $A \subseteq \mathbb{N}$ und $\Theta : \mathbb{N} \rightarrow \mathbb{N}$, dann ist

$$\Theta(A) = \{\Theta(a) : a \in A \wedge a \in \text{dom}(\Theta)\}$$

und

$$\Theta^{-1}(A) = \{b : b \in \text{dom}(\Theta) \wedge \Theta(b) \in A\}$$

Für die Indizierung rekursiv aufzählbarer Mengen gilt folgendes

Theorem Es gibt totale und berechenbare Funktionen $g_1, g_2 : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit

1. $\varphi_i(W_j) = W_{g_1(i,j)}$
2. $\varphi_i^{-1}(W_j) = W_{g_2(i,j)}$

für alle $i, j \in \mathbb{N}$.

Beweis. (nur 1) Es gilt $\varphi_i(W_j) = \{\varphi_i(a) : a \in W_j \cap W_i\}$. Betrachte das Programm

$$P_{\hat{g}(i,j)} = x_2 := \Phi(j, x_1); x_1 := \Phi(i, x_1)$$

Offensichtlich berechnet $P_{\hat{g}(i,j)}$ die Funktion $\varphi_i(a)$, falls $a \in W_i \cap W_j$. Es gilt $\varphi_i(W_j) = \text{range}(\varphi_{\hat{g}(i,j)})$. Wir verwenden die totale und berechenbare Funktion $t_2 : \mathbb{N} \rightarrow \mathbb{N}$ mit der Eigenschaft $\text{range}(\varphi_i) = \text{dom}(\varphi_{t_2(i)})$ für alle $i \in \mathbb{N}$. Dann erhalten wir

$$\varphi_i(W_j) = \text{range}(\varphi_{\hat{g}(i,j)}) = \text{dom}(\varphi_{t_2 \circ \hat{g}(i,j)}) = \text{dom}(\varphi_{g_1(i,j)})$$

Mit $g_1(i, j) = t_2 \circ \hat{g}(i, j)$ folgt die Behauptung. \square

1.5 Die Sätze von Rice

Der Satz von Rice zeigt, dass jede nicht triviale Partitionierung von \mathbb{N} mit einer bestimmten Eigenschaft nicht entscheidbar ist. Die erwähnte Eigenschaft ist wie folgt definiert:

Definition Eine Menge $I \subseteq \mathbb{N}$ von Indizes RESPEKTIERT Funktionen, wenn $i \in I$ und $\varphi_i = \varphi_j$ $j \in I$ impliziert.

Theorem (Satz von Rice) Sei $I \subseteq \mathbb{N}$ eine Menge von Indizes, die Funktionen respektiert. Dann ist I rekursiv genau dann, wenn $I = \emptyset$ oder $I = \mathbb{N}$.

Beweis. Wenn $I = \emptyset$ oder $I = \mathbb{N}$, dann ist I offensichtlich rekursiv. Sei also $\emptyset \neq I \neq \mathbb{N}$. Annahme: I sei rekursiv. O.E. enthalte I alle Indizes einer berechenbaren Funktion Θ , welche nicht die überall undefinierte Funktion \perp ist — anderenfalls betrachte $\bar{I} = \mathbb{N} \setminus I$.

Definiere das Programm

$$P_{f(i)} = x_2 := \Phi(i, i); x_1 := \Theta(x_1)$$

Dann gilt

$$\varphi_{f(i)} = \begin{cases} \Theta & \text{falls } \varphi_i(i) \text{ terminiert} \\ \perp & \text{sonst} \end{cases}$$

Es gilt $f(i) \in I$ genau dann, wenn $\varphi_i(i)$ terminiert und damit

$$\chi_I(f(i)) = \begin{cases} 1 & \text{falls } \varphi_i(i) \text{ terminiert} \\ 0 & \text{sonst} \end{cases}$$

Die Funktion $\chi_I \circ f$ ist nicht berechenbar, da sonst das Halteproblem lösbar wäre. Daher ist I nicht rekursiv. \square

Beispiel Folgende Mengen sind unentscheidbar:

1. $A_1 = \{i : \varphi_i : \mathbb{N} \rightarrow \mathbb{N} \text{ total}\}$
2. $\bar{A}_1 = \{i : \varphi_i : \mathbb{N} \rightarrow \mathbb{N} \text{ nicht total}\}$
3. $A_2 = \{i : \text{range}(\varphi_i) = \mathbb{N}\}$

Beachte, dass auf die Menge $K = \{i : \varphi_i(i) \text{ terminiert}\}$ der Satz von Rice nicht anwendbar ist: Sei $W_e = \{e\}$. Dann ist $e \in K$, aber für $e' \neq e$ mit $\varphi_{e'} \neq \varphi_e$ folgt nicht $e' \in K$.

Für das folgende Theorem benötigen wir eine neue Notation: Für zwei Funktionen f und g schreiben wir $f \leq g$, falls als $f(x)$ definiert $g(x)$ definiert und $f(x) = g(x)$ folgt.

Theorem (Satz von Rice, Variation 1) Sei I eine Indexmenge, die Funktionen respektiert. Wenn eine berechenbare Funktion Θ existiert mit

1. $\{i : \varphi_i = \Theta\} \subseteq I$ und
2. es gibt eine berechenbare Funktion $\hat{\Theta}$ mit $\Theta \leq \hat{\Theta}$ und $\{i : \varphi_i = \hat{\Theta}\} \subseteq \bar{I}$.

Dann ist I nicht rekursiv aufzählbar.

Beachte: Aus Bedingung 2 folgt $\hat{\Theta} \neq \Theta$. Deshalb ist Θ nicht total.

Beweis. Betrachte die berechenbare Funktion $\mu : \mathbb{N}^2 \rightarrow \mathbb{N}$, definiert als

$$\mu(i, j) = \begin{cases} \Theta(j) & \text{falls } \varphi_i(i) \text{ nicht terminiert} \\ \hat{\Theta}(j) & \text{falls } \varphi_i(i) \text{ terminiert} \end{cases}$$

d.h. $\mu(i, j) = \varphi_e(i, j)$ für ein e . Mit Hilfe des s-n-m - Theorems folgt, dass es eine totale und berechenbare Funktion f gibt mit $\mu(i, j) = \varphi_{f(i)}(j)$. Dann gilt $f(i) \in I$ genau dann, wenn $\varphi_i(i)$ nicht terminiert. Dann lässt sich $\bar{K} = \{i : \varphi_i(i) \text{ terminiert nicht}\}$ schreiben als $\bar{K} = f^{-1}(I)$. Wenn I rekursiv aufzählbar wäre, dann auch \bar{K} wegen der Abschlusseigenschaft unter Inversenbildung. Das ist ein Widerspruch. Es bleibt zu zeigen, dass μ berechenbar ist. Aus $\Theta \leq \hat{\Theta}$ folgt: Wenn $\Theta(x)$ definiert ist, dann auch $\hat{\Theta}(x) = \Theta(x)$. Starte $\Theta(j)$ und $\varphi_i(i)$ gemeinsam (dovetailing). Falls $\Theta(j)$ hält, stoppe und gib das Ergebnis aus — dann gilt nämlich $\Theta(j) = \hat{\Theta}(j)$ unabhängig von $\varphi_i(i)$. Falls $\varphi_i(i)$ vor $\Theta(j)$ hält, stoppe $\Theta(j)$ und starte $\hat{\Theta}(j)$. \square

Beispiel $\overline{A_1} = \{i : \varphi_i : \mathbb{N} \rightarrow \mathbb{N} \text{ ist nicht total}\}$
 Für Funktion

$$\Theta(n) = \begin{cases} 0 & \text{falls } n = 0 \\ \perp & \text{sonst} \end{cases}$$

ist Bedingung 1 erfüllt. Setze $\hat{\Theta}(n) = n$ für $n \geq 0$. Dann ist $\Theta \leq \hat{\Theta}$ und Bedingung 2 ist erfüllt. Damit ist $\overline{A_1}$ nach dem Satz von Rice, Variation 1 nicht rekursiv aufzählbar. Die Unentscheidbarkeit der Menge A_1 lässt sich mit dem Satz von Rice, Variation 1, nicht zeigen.

Theorem (Satz von Rice, Variation 2) Sei I eine Indexmenge, die Funktionen respektiert. Wenn eine berechenbare Funktion Θ existiert mit

1. $\{i : \varphi_i = \Theta\} \subseteq I$ und
2. $\{i : \varphi_i \leq \Theta \wedge \text{dom}(\varphi_i) \text{ endlich}\} \subseteq \overline{I}$

dann ist die Menge I nicht rekursiv aufzählbar.

Beweis. Betrachte die Funktion $\mu : \mathbb{N}^2 \rightarrow \mathbb{N}$, definiert vermöge

$$\mu(i, j) = \begin{cases} \Theta(j) & \text{falls } P_i \text{ auf Eingabe } i \text{ in höchstens } j \text{ Schritten nicht terminiert} \\ \perp & \text{sonst} \end{cases}$$

Die Funktion μ ist berechenbar, daher gibt es einen Index e mit $\mu = \varphi_e$. Mit Hilfe des s-n-m – Theorems folgt die Existenz einer totalen und berechenbaren Funktion f mit $\mu(i, j) = \varphi_e(i, j) = \varphi_{f(i)}(j)$. Dann gilt

$$\begin{aligned} i \in K &\Leftrightarrow P_i \text{ hält auf Eingabe } i \text{ (Definition von } K) \\ &\Leftrightarrow P_i \text{ hält auf Eingabe } i \text{ in genau } j \text{ Schritten, für ein } j \\ &\Leftrightarrow \varphi_{f(i)}(x) = \Theta(x) \text{ für } x < j \wedge \varphi_{f(i)}(x) = \perp \text{ für } x \geq j \text{ für obiges } j \\ &\Leftrightarrow \varphi_{f(i)} \leq \Theta \wedge \text{dom}(\varphi_{f(i)}) \text{ endlich} \\ &\Rightarrow f(i) \in \overline{I} \end{aligned}$$

Andererseits gilt

$$\begin{aligned} i \in \overline{K} &\Leftrightarrow P_i \text{ hält auf Eingabe } i \text{ nicht} \\ &\Leftrightarrow \varphi_{f(i)} = \Theta \\ &\Rightarrow f(i) \in I \end{aligned}$$

Hieraus schließen wir $i \in \overline{K} \Leftrightarrow f(i) \in I$. Wenn I rekursiv aufzählbar wäre, dann auch \overline{K} . Dies ist ein Widerspruch.

Was haben wir getan? Reduktion von \overline{K} auf I ! □

Beispiel $A_1 = \{i : \varphi_i : \mathbb{N} \rightarrow \mathbb{N} \text{ total}\}$. Mit $\Theta(n) = n$ für alle n folgt die nicht rekursive Aufzählbarkeit.

Abschließend definieren wir den Begriff der Reduktion formal.

Definition (many-one – Reduktion) Eine Menge A ist MANY-ONE – REDUZIERBAR AUF B , in Zeichen $A \leq_m B$, falls es eine totale und berechenbare Funktion f gibt, so dass $\forall x : x \in A \Leftrightarrow f(x) \in B$. A ist many-one – äquivalent zu B , in Zeichen $A \equiv_m B$, falls $A \leq_m B \wedge B \leq_m A$.

Es folgen einfache Eigenschaften.

Theorem

- i. \leq_m ist reflexiv und transitiv.
- ii. $A \leq_m B$ impliziert $\bar{A} \leq_m \bar{B}$.
- iii. $A \leq_m B$ und B rekursiv impliziert A rekursiv.
- iv. $A \leq_m B$ und B rekursiv aufzählbar impliziert A rekursiv aufzählbar.

Beweis. Hausaufgabe □

Beachte, dass aus $A \leq_m B$ im allgemeinen nicht $A \leq_m \bar{B}$ folgt. Betrachte K und \bar{K} . Die Menge K ist rekursiv aufzählbar, aber \bar{K} nicht. Es folgt $\bar{K} \not\leq_m K$ und $K \not\leq_m \bar{K}$. Für eine »schwächere« Reduktion, die Turing – Reduktion, gilt $\bar{K} \leq_T K$.

Definition (Turing – Reduktion) Eine Menge A ist TURING – REDUZIERBAR AUF B , in Zeichen $A \leq_T B$, falls A rekursiv mit Hilfe von B ist, d.h. falls es ein WHILE – Programm mit Orakel B gibt, welches χ_A berechnet. Ein WHILE – Programm mit Orakel B ist ein »normales« WHILE – Programm, welches zusätzlich folgende Kontrollflussanweisungen enthalten darf:

- while $x \in B$ do ... od
- while $x \notin B$ do ... od

Theorem

- i. \leq_T ist reflexiv und transitiv.
- ii. \leq_m impliziert \leq_T .

Das letzte Theorem des Abschnittes

Theorem Die Menge $K = \{i : \varphi_i(i) \text{ hält}\}$ ist \leq_m – vollständig. Hierbei ist \leq_m – vollständig wie folgt definiert: A ist \leq_m – vollständig, falls

1. A ist rekursiv aufzählbar
2. für jede rekursiv aufzählbare Menge B gilt $B \leq_m A$.

Bemerkung: vollständig \sim typischer Vertreter.

Beweis. Sei $K_0 = \{(i, j) : \exists t : P_i \text{ hält auf Eingabe } j \text{ in höchstens } t \text{ Schritten}\}$. Für eine rekursiv aufzählbare Menge B mit $B = w_i$ für ein $i \in \mathbb{N}$ gilt $j \in B \Leftrightarrow (i, j) \in K_0$ mit Funktion $h(j) = (i, j)$ für alle $j \in \mathbb{N}$. Im zweiten Schritt zeigen wir $K_0 \leq_m K$. Betrachte

$$\varphi_{g(x)} = \begin{cases} 1 & \text{falls } \varphi_{f_1(x)}(f_2(x)) \text{ hält} \\ \perp & \text{sonst} \end{cases}$$

wobei f_1 und f_2 die »Cantorprojektionsfunktionen« sind. Bemerkung: g ist total und berechenbar. Dann gilt $(i, j) \in K_0 \Leftrightarrow g(f(i, j)) \in K$, wobei $f_1(f(i, j)) = i$ und $f_2(f(i, j)) = j$. Die gesuchte Reduktion ist $g \circ f$. □

1.6 Alternative Modelle für Berechenbarkeit

Es folgen abschließende Bemerkungen zur Theorie der Berechenbarkeit (ohne Beweise).

Definition Die Klasse der primitiv rekursiven Funktionen $\mathbb{N}^n \rightarrow \mathbb{N}$ ist wie folgt induktiv definiert:

1. Die nullstelligen Funktionen (Konstanten) sind primitiv rekursiv.
2. Die Nachfolgerfunktion $s(n) = n + 1$ ist primitiv rekursiv.
3. Die Vorgängerfunktion

$$p(n) = \begin{cases} 0 & \text{falls } n = 0 \\ n - 1 & \text{sonst} \end{cases}$$

ist primitiv rekursiv.

4. Die Projektionsfunktionen $u_j^n : \mathbb{N}^n \rightarrow \mathbb{N}$, definiert durch $u_j^n(x_1, x_2, \dots, x_n) = x_j$ ist primitiv rekursiv.
5. Die Funktionskomposition ist primitiv rekursiv, d.h. für alle primitiv rekursiven Funktionen $g : \mathbb{N}^m \rightarrow \mathbb{N}$ und $h_1, \dots, h_m : \mathbb{N}^n \rightarrow \mathbb{N}$ ist die Funktion $f : \mathbb{N}^n \rightarrow \mathbb{N}$, definiert durch $f(x_1, \dots, x_n) = g(h_1(x_1, \dots, x_n), \dots, h_m(x_1, \dots, x_n))$ primitiv rekursiv.
6. Das Rekursionsschema ist primitiv rekursiv, d.h. für alle primitiv rekursiven Funktionen $g : \mathbb{N}^n \rightarrow \mathbb{N}$ und $h : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ ist die Funktion $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$, definiert vermöge

$$\begin{aligned} f(x_1, \dots, x_n, 0) &= g(x_1, \dots, x_n) \\ f(x_1, \dots, x_n, y + 1) &= h(x_1, \dots, x_n, y, f(x_1, \dots, x_n, y)) \end{aligned}$$

primitiv rekursiv.

Theorem Eine Funktion f ist primitiv rekursiv genau dann, wenn f LOOP – berechenbar ist. Primitiv rekursive Funktionen lassen sich mit Hilfe der »Minimierung« erweitern.

Definition Erweitert man die Definition der primitiv rekursiven Funktionen um die μ – Rekursion, so erhält man die Klasse der μ – rekursiven Funktionen. Die μ – Rekursion ist wie folgt definiert: Sei $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ eine (μ – rekursive) Funktion, dann ist die Funktion

$$f(x_1, \dots, x_n) = \mu y [h(x_1, \dots, x_n, y) = 0] = \begin{cases} z & \text{falls } h(x_1, \dots, x_n, y) \text{ für } y \leq z \text{ definiert ist,} \\ & h(x_1, \dots, x_n, z) = 0 \text{ und} \\ & h(x_1, \dots, x_n, y) \leq 0 \text{ für } y < z \\ \perp & \text{sonst} \end{cases}$$

Theorem Eine Funktion f ist μ – rekursiv genau dann, wenn f WHILE – berechenbar ist. Eine alternative Charakterisierung der berechenbaren Funktionen liest sich wie folgt:

Theorem Eine Funktion f ist WHILE – berechenbar genau dann, wenn f von einer Turingmaschine berechnet wird (bei Eingabe der Parameter x_1, \dots, x_n berechnet die Turingmaschine $f(x_1, \dots, x_n)$ in geeigneter Codierung).

Ein klassisches unentscheidbares Problem, basierend auf Turingmaschinen – Berechnungen, ist das Postsche Korrespondenzproblem (PCP), das wie folgt definiert ist:

Gegeben eine Liste $(u_1, v_1), (u_2, v_2), \dots, (u_k, v_k)$ von Wortpaaren über einem Alphabet. Frage: Gibt es eine Folge i_1, i_2, \dots, i_m mit $m \geq 1$ und $1 \leq i_j \leq k$ so, dass $u_{i_1} u_{i_2} \dots u_{i_m} = v_{i_1} v_{i_2} \dots v_{i_m}$?

Beispiel

- $(0, 10), (101, 01)$ hat keine Lösung
- $(a, aaa), (abaaa, ab), (ab, b)$ hat Lösung 2, 1, 1, 3.

Theorem Das PCP ist unentscheidbar.

Beweis. Reduktion des allgemeinen Halteproblems auf PCP. □

Kapitel 2

Automaten und formale Sprachen

Formale Sprachen beschäftigen sich mit der Beschreibung unendlicher Objekte (Sprachen über endlichen / unendlichen Wörtern, Sprachen über endlichen / unendlichen Bäumen, ...) durch endliche Beschreibungsmechanismen (Automaten, Grammatiken, Logiken). Der Beschreibungsmechanismus hängt stark davon ab, was man beschreiben will (Sequentialität, Verteiltheit, Unabhängigkeit, ...). Während Automaten akzeptierende Systeme sind, sind Grammatiken meist generierende Systeme. Zu Beginn einige Grundlagen.

2.1 Grundlagen

Sei Σ ein Alphabet (die Elemente von Σ sind unteilbare atomare Einheiten), z.B. ist $\Sigma = \{0, 1\}$ ein Alphabet, $\{a, ab\}$ (mit ab teilbar) nicht. Ein Wort x über einem Alphabet ist eine endliche Σ -Sequenz, d.h. $x = a_1 a_2 \dots a_k$ mit $k \geq 0$ und $a_i \in \Sigma$ für $1 \leq i \leq k$. Falls $k = 0$ ist, beschreibt x das leere Wort λ oder ε . Die Länge von x ist k und wird mit $|x|$ bezeichnet; $|\lambda| = 0$.

Σ^* bezeichnet die Menge aller endlichen Wörter über Σ (inklusive λ). Für $x, y \in \Sigma^*$ sei $x \cdot y = x \circ y = xy$ die Konkatenation (Hintereinanderschreiben) von x und y . Das Tupel $(\Sigma^*, \cdot, \lambda)$ ist ein Monoid. Das Monoid ist sogar »frei«, d.h. jedes Element aus Σ^* hat eine eindeutige Zerlegung bezüglich der Konkatenation aus Σ -Elementen.

Für eine formale Sprache $L \subseteq \Sigma^*$ definieren wir Potenzen von L wie folgt:

$$L^0 = \{\lambda\}$$
$$L^{i+1} = L \cdot L^i = \{x \cdot y : x \in L \wedge y \in L^i\} \text{ für } i \geq 0$$

wobei für $X, Y \subseteq \Sigma^*$ gilt: $X \cdot Y = \{x \cdot y : x \in X \wedge y \in Y\}$.

Beispiel

- $L = \{a\} \subseteq \{a, b\}^*$, dann ist $L^2 = L \cdot L = \{a \cdot a\} = \{a^2\}$
- Für $L = \{ab, a, \lambda\} \subseteq \{a, b\}^*$ ist $L^2 = \{abab, aba, ab, aab, aa, a, ab, a, \lambda\}$.

Ferner setzen wir

$$L^* = \bigcup_{i \geq 0} L^i \quad \text{und} \quad L^+ = \bigcup_{i \geq 1} L^i = L \cdot L^*$$

Hierbei heißt L^* die KLEENSCHES HÜLLE von L .

Literatur

- Michael A. Harrison: Introduction to Formal Language Theory, Addison-Wesley 1978
- Alexander Asteroth, Christel Baier: Theoretische Informatik, Pearson-Studium 2002
- (Nicholas Pippenger: Theories of Computability, Cambridge University Press, 1997)

- Uwe Schöning: Theoretische Informatik — kurzgefasst, BI-Wissenschaftsverlag
- Hopcraft, Ullman: Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979
- Dominique Perrin, Jean-Reic Prin: Infinite Words — Automata, Semigroups, Logic and Games, Elsevier 2004

2.2 (Chomsky –) Grammatiken

Die Grundidee ist ein »Ersetzungsprozess«, bei dem einzelne Zeichen oder Zeichenketten durch andere Wörter ersetzt werden. Formal ist eine Chomsky – Grammatik wie folgt definiert:

Definition (Chomsky – Grammatik) Eine Grammatik ist ein 4-Tupel $G = (N, T, P, S)$, wobei

- N eine endliche Menge von Nichtterminalen (Variablen),
- T eine endliche Menge von Terminalen mit $N \cap T = \emptyset$,
- P eine endliche Menge aus $(N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ und
- $S \in N$ das Startsymbol oder Axiom ist.

Die Elemente von P werden Produktionen genannt. Anstelle von $(\alpha, \beta) \in P$ schreiben wir $\alpha \rightarrow_G \beta$ oder einfach $\alpha \rightarrow \beta$.

Sei $G = (N, T, P, S)$ eine Grammatik. Die Relation \Rightarrow_G oder \Rightarrow ist wie folgt definiert: Seien $x, y \in (N \cup T)^*$. Dann gilt $x \Rightarrow y$, falls $x = x_0 \alpha x_1$, $y = x_0 \beta x_1$ und $\alpha \rightarrow \beta$ Produktion in P ist.

Die Relation \Rightarrow_G^* oder \Rightarrow^* ist die reflexive, transitive Hülle von \Rightarrow_G oder \Rightarrow . Diese ist gegeben durch $x \Rightarrow^* y$ ($\Rightarrow x$ leitet in endlich vielen Schritten y ab) genau dann, wenn es x_0, x_1, \dots, x_n gibt mit $x = x_0 \Rightarrow x_1 \Rightarrow \dots \Rightarrow x_n = y$. Der Spezialfall $n = 0$ ist zugelassen und liefert $\forall x \in (N \cup T)^* : x \Rightarrow^* x$.

Die durch die Grammatik $G = (N, T, P, S)$ erzeugte Sprache ist definiert vermöge

$$L(G) = \{w \in T^* : S \Rightarrow_G^* w\}$$

Bemerkung: Anstelle von generierenden Chomsky – Grammatiken hätte man auch reduzierende (akzeptierende) definieren können.

Zwei Grammatiken G_1 und G_2 heißen ÄQUIVALENT, falls $L(G_1) = L(G_2)$ ist. Die Chomsky – Hierarchie ist wie folgt definiert:

Definition (Grammatik – Typ) Sei $G = (N, T, P, S)$ eine Grammatik. Dann heißt G

- PHRASENSTRUKTUR – GRAMMATIK oder vom Typ 0
- KONTEXT-SENSITIV oder vom Typ 1, wenn für alle Regeln (außer $S \rightarrow \lambda$) gilt: $\alpha_0 A \alpha_1 \rightarrow \alpha_0 \beta \alpha_1$ für $\alpha_0, \alpha_1 \in (N \cup T)^*$, $A \in N$ und $\beta \in (N \cup T)^+$. Kommt das Axiom in keiner rechten Seite vor, dann ist auch $S \rightarrow \lambda$ zugelassen.
- MONOTON, wenn für alle Regeln (außer $S \rightarrow \lambda$) gilt: $\alpha \rightarrow \beta$ mit $|\alpha| \leq |\beta|$. Kommt das Axiom in keiner rechten Seite vor, dann ist auch $S \rightarrow \lambda$ zugelassen.
- KONTEXTFREI oder vom Typ 2, wenn $P \subseteq N \times (N \cup T)^*$
- REGULÄR oder vom Typ 3, wenn $P \subseteq N \times (T^* N \cup T^*)$ (oder $P \subseteq N \times (NT^* \cup T^*)$)

Eine Sprache $L \subseteq T^*$ heißt vom Typ i, falls es eine Typ i – Grammatik G mit $L = L(G)$ gibt.

Beispiele

- $S \rightarrow A, A \rightarrow \lambda$ ist vom Typ 3, Typ 2, nicht monoton, nicht vom Typ 1, aber vom Typ 0.
- Die Sprache $L = \{a^n b^n c^n : n \geq 1\}$ wird durch die Grammatik $G = (\{S, B, C\}, \{a, b, c\}, P, S)$ mit der Produktionsmenge

$$\begin{aligned} S &\rightarrow aSBC|aBC \\ CB &\rightarrow BC \\ aB &\rightarrow ab \\ bB &\rightarrow bb \\ bC &\rightarrow bc \\ cC &\Rightarrow cc \end{aligned}$$

erzeugt. Dies ist eine Typ 1 – Grammatik.

Wir zeigen, dass die Klassen der Sprachen, induziert durch die Chomsky – Grammatiktypen eine Hierarchie bilden (bezüglich Sprachinklusion).

Theorem

- Sprachen vom Typ i sind zugleich Sprachen vom Typ $i - 1$ für $i = 1, 2, 3$.
- Monotone Sprachen sind vom Typ 1 und umgekehrt.

Beweis. (Idee)

- Die einzige nicht offensichtliche Inklusion ist: Jede kontextfreie Sprache ist kontextsensitiv. Idee. Bestimme alle Nonterminale A mit $A \Rightarrow^* \lambda$ (rekursiver Algorithmus) und füge Regeln geeignet hinzu. Falls z.B. $A \Rightarrow^* \lambda$ und es eine Regel $B \rightarrow xAyA$ gibt, dann füge die Regeln $B \rightarrow xyA|xAy|xy$ ein. Lösche Regeln der Form $C \rightarrow \lambda$. Füge $S \rightarrow \lambda$ hinzu, falls $S \Rightarrow^* \lambda$ gilt.
- Zu zeigen: Simulation von monotonen Regeln mit Hilfe kontextsensitiver Regeln. Ersetze Regeln der Form

$$p = A_1 A_2 \dots A_m \rightarrow B_1 \dots B_n$$

mit $m \leq n$ und $n \geq 3$ wie folgt:

Falls $m = n$, dann

$$\begin{aligned} A_1 A_2 &\rightarrow B_1 Y_{p,1} \\ Y_{p,i} A_{i+2} &\rightarrow B_{i+1} Y_{p,i+1} \quad \text{für } 1 \leq i \leq m-2 \\ Y_{p,m-2} A_m &\rightarrow B_{m-1} B_m \end{aligned}$$

wobei die $Y_{p,i}$ s neue Nichtterminale sind.

Falls $m < n$, dann

$$\begin{aligned} A_1 \dots A_m &\rightarrow B_1 \dots B_{m-1} Y_{p,1} \\ Y_{p,m-2} &\rightarrow B_m \dots B_n \end{aligned}$$

Abschließend simuliere Regeln der Form $p = AB \rightarrow CD$ wie folgt:

$$\begin{aligned} AB &\rightarrow AX_p \\ AX_p &\rightarrow Y_p X_p \\ Y_p X_p &\rightarrow Y_p D \\ Y_p D &\rightarrow CD \end{aligned}$$

wobei X_p, Y_p neue Nichtterminale sind.

□

Damit erhalten wir die Inklusionskette

$$\mathcal{L}_3 \subseteq \mathcal{L}_2 \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_{\text{mon}} \subseteq \mathcal{L}_0$$

wobei $\mathcal{L}_i = \{L : \text{es gibt ein } G \text{ vom Typ } i \text{ mit } L = L(G)\}$. Es bleibt zu zeigen, dass diese Inklusionskette echt ist.

Für Typ 1 – und Typ 0 – Sprachen wird Diagonalisierung verwendet. Alternative Möglichkeit über das Wortproblem, welches definiert ist als: Gegeben eine Grammatik G und ein Wort w : Ist $w \in L(G)$?

Es gilt das folgende Theorem:

Theorem Das Wortproblem für Typ 1 – Sprachen ist entscheidbar, für Typ 0 – Sprachen unentscheidbar.

Beweis. (Idee) Die Typ 0 – Aussage folgt aus der Berechenbarkeitstheorie. Für die Typ 1 – Aussage bestimme die Mengen

$$T_n^0 = \{S\}$$

$$T_n^{m+1} = T_n^m \cup \{x \in (N \cup T)^* : |x| \leq n \wedge y \Rightarrow x \text{ für } y \in T_n^m\}$$

und entscheide $w \in T_n^m$ für ein m mit $T_n^m = T_n^{m+1}$. \square

Für Typ 3 und Typ 2 werden Iterationslemmata verwendet. Es folgt Ogden's Lemma für Typ 2 – Sprachen.

Lemma (Ogden's Lemma) Sei L eine kontextfreie Sprache. Dann gibt es eine natürliche Zahl n , so dass man jedes Wort $z \in L$, in dem n oder mehr Positionen markiert sind, als $z = uvwx^i y$ schreiben kann, wobei folgendes gilt:

1. v und x haben zusammen zumindest eine markierte Position
2. vw hat höchstens n markierte Positionen
3. $uv^iwx^i y \in L$ für alle $i \geq 0$.

Beweis. (Idee) Sei $G = (N, T, P, S)$ mit $L = L(G)$. Setze $n = l^{2m+3}$, wobei $l = \max\{|\alpha| : A \rightarrow \alpha \in P\}$ und $m = |N|$. Betrachte den Ableitungsbaum T für ein $z \in L(G)$, wobei mindestens n Positionen in z markiert sind. Wähle einen Pfad von Knoten n_1, n_2, \dots in T wie folgt:

1. n_1 ist Wurzel.
2. Wenn n_i bekannt ist, wähle n_{i+1} so, dass n_{i+1} Nachfolger von n_i ist und die meisten markierten Blätter aller Geschwister besitzt.
3. Falls n_i Blatt ist, dann endet der Pfad hier.

Sei n_1, \dots, n_p der konstruierte Pfad. Ein Knoten heißt VERZWEIGEND, falls mindestens zwei Nachfolgeknoten markierte Blätter besitzen. Per Induktion zeigt man folgendes: Wenn n_1, \dots, n_i r verzweigende Knoten hat, dann hat n_{i+1} mindestens l^{2m+3-r} markierte Blätter. Da n_1 l^{2m+3} markierte Blätter hat, gilt für den Pfad $n_1 \dots n_p$, dass mindestens $2m + 3$ verzweigende Knoten auftreten müssen. Seien b_1, \dots, b_{2m+3} die letzten $2m + 3$ verzweigenden Knoten. Knoten b_i ist LINKSVERZWEIGEND, falls ein linker Nachfolger (bezüglich des Pfades) mindestens ein markiertes Blatt enthält. O.E. gibt es in $b_1 \dots b_{2m+3}$ mindestens $m + 2$ linksverzweigende Knoten l_1, \dots, l_{m+2} , (die letzten $m + 2$ Knoten von $b_1 \dots b_{2m+3}$). Da $m = |N|$, findet man in l_2, \dots, l_{m+2} zwei Knoten mit gleichen Nonterminalen. Diese sind l_f und l_g . Damit ist das Ogden – Lemma erfüllt. \square

Spezialfall:

Theorem (*uvwxy*-Theorem, Pumping-Lemma) Sei L eine kontextfreie Sprache. Dann gibt es eine von L abhängige Konstante n , so dass sich z für $z \in L$ und $|z| \geq n$ als $z = uvwxy$ schreiben lässt mit

1. $|vw| \geq 1$
2. $|vwx| \leq n$
3. $\forall i \geq 0 : uv^iwx^iy \in L$

Beispiel $L = \{a^ib^jc^id^j : i, j \geq 1\}$. Wir zeigen, dass L *nicht* kontextfrei ist.

Widerspruchsbeweis: Angenommen, L ist kontextfrei. Demnach ist das Pumping-Lemma anwendbar. n sei die Konstante aus dem Pumping-Lemma. Betrachte das Wort $a^n b^n c^n d^n \in L$. Gemäß dem Pumping-Lemma setzen wir $a^n b^n c^n d^n = uvwxy$.

- Fall 1: vwx enthält nur $as \Rightarrow uvw \in L$. Dieses Wort hat mindestens ein a weniger $\Rightarrow uvw$ hat weniger as als cs . Widerspruch.
- Fall 2-4 (vwx enthält nur bs , cs oder ds): analog
- Fall 5: v enthält as , x enthält ein $b \Rightarrow uvw \in L \Rightarrow$ Zahl der as , bs reduziert, Zahl der cs und ds gleich. Widerspruch.
- Fall 6-7: analog

Pumping-Lemma für Typ 3 – Sprachen Sei L eine reguläre Menge. Dann gibt es eine Konstante m , so dass sich jedes Wort $z \in L$ mit $|z| \geq m$ als $z = uvw$ mit $|v| \geq 1$ und $|uv| \leq m$ schreiben lässt und für alle $i \geq 0$ gilt: $uv^iw \in L$.

Bemerkung: Es gibt nicht-kontextfreie Sprachen, für die man mit dem *uvwxy*-Theorem nicht nachweisen kann, dass sie nicht kontextfrei sind.

2.3 Maschinenmodelle

Zu jeder Chomsky – Stufe gibt es ein entsprechendes Maschinenmodell.

- Typ 0: DTM, NTM (Turing-Maschine)
- Typ 1: NTM mit linear beschränktem Platz
- Typ 2: nichtdeterministischer Kellerautomat
deterministische Kellerautomaten entsprechen deterministischen kontextfreien Sprachen
- Typ 3: DFA, NFA

2.3.1 Turing-Maschinen

Turing-Maschinen bestehen aus einer endlichen Kontrolle und einem (beidseitig) unendlichen Band:

$$\mathcal{M} = (K, \Sigma, \delta, F, s)$$

mit

K endliche Menge von Zuständen

Σ Bandsymbolmenge

δ Übergangsrelation ($\delta \subseteq (K \times \Sigma) \times (K \times \Sigma \times \{\rightarrow, \leftarrow, \downarrow\})$)

F Endzustände ($F \subseteq K$)

s Startzustand ($s \in K$)

Eine Berechnung ist eine Sequenz von »Konfigurationen«. Sie beschreibt Zustand, Bandinhalt und die Position des Lesekopfs: (α, q, β) mit $\alpha, \beta \in \Sigma^*, q \in K$.

Startkonfiguration: (λ, s, w) .

Mittels Übergangsrelation kann man eine Relation \vdash auf den Konfigurationen definieren: $k_1 \vdash k_2$; \vdash^* : reflexiver, transitiver Abschluss von \vdash .

$$(w, q, tw') \vdash (wt', q', w') \Leftrightarrow ((q, t), (q', t', \rightarrow)) \in \delta$$

Eine Turing-Maschine \mathcal{M} akzeptiert L genau dann, wenn $(\lambda, s, w) \vdash^* (w', f, w'')$ mit $f \in F$.

Bemerkung: Viele Varianten von Turing-Maschinen sind denkbar: einseitig unendliches Band, mehrere Bänder, zweidimensionales Band, mehrere Leseköpfe, Beschränkung des Bandalphabets auf drei und/oder der Zustandsmenge auf zwei Elemente, . . . Alle dieses Varianten sind äquivalent.

MPCP Gegeben sei eine Folge von Paaren $(a_1, b_1), \dots, (a_k, b_k)$. Gesucht: Folge i_1, \dots, i_n , so dass $a_1 a_{i_1} a_{i_2} \dots a_{i_n} = b_1 b_{i_1} b_{i_2} \dots b_{i_n}$.

Satz Das MPCP ist unentscheidbar.

Beweis. Reduktion auf das Halteproblem: Wir nehmen eine Maschine M , ein Wort w und konstruieren eine Instanz I des MPCP, so dass für I Indexf. existiert. Paare für das MPCP:

- Anfangspaar: $(\#, \#q_0, w\#)$
- Dummypaare: $(\#, \#), (c, c) \forall c \in \Sigma$
- Zustandsübergänge:
 - $\delta(q, X) \rightarrow (p, Y, \rightarrow) \rightsquigarrow (qX, Yp)$
 - $\delta(q, X) \rightarrow (p, Y, \leftarrow) \rightsquigarrow (ZqX, pZY) \forall Z \in \Sigma$
 - $\delta(q, \sqcup) \rightarrow (p, Y, \rightarrow) \rightsquigarrow (p\#, Yp\#)$
 - $\delta(q, \sqcup) \rightarrow (p, Y, \leftarrow) \rightsquigarrow (Zq\#, pZY\#)$
 - $\delta(q, X) \rightarrow (p, Y, \downarrow) \rightsquigarrow (qX, pY)$

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

Übergänge:

- $\delta(q_1, 0) = (q_2, 1, \rightarrow) \rightsquigarrow (q_1 0, 1 q_2)$
- $\delta(q_2, 0) = (q_3, 0, \leftarrow) \rightsquigarrow (1 q_2 0, q_3 1 0), (0 q_2 0, q_3 0 0), (\sqcup q_2 0, q_3 \sqcup 0)$
- $\delta(q_1, 1) = (q_2, 0, \leftarrow) \rightsquigarrow (0 q_1 1, q_2 0 0), (1 q_1 1, q_2 1 0), (\sqcup q_1 1, q_2 \sqcup 0)$
- $\delta(q_2, 1) = (q_1, 1, \rightarrow) \rightsquigarrow (q_2 1, 0 q_1)$
- $\delta(q_1, \sqcup) = (q_2, 1, \leftarrow) \rightsquigarrow (0 q_1 \sqcup, q_2 0 1 \#), (1 q_1 \sqcup, q_2 1 1 \#), (\sqcup q_2 \sqcup, q_2 \sqcup 1 \#)$
- $\delta(q_2, \sqcup) = (q_2, 0, \rightarrow) \rightsquigarrow (q_2 \sqcup, 0 q_2 \#)$

Endpaare: $(XqY, q), (Xq, q), (qY, q), (q\#\#, \#)$ für $q \in F, X, Y \in \Sigma$

Eine Lösung von I existiert genau, dann, wenn $\langle M, w \rangle$ hält. □

Bemerkung: MPCP \leq PCP

Schnittproblem kontextfreier Grammatiken

Gegeben zwei kontextfreie Grammatiken G_1, G_2 . Ist $L(G_1) \cap L(G_2) = \emptyset$?

Diese Frage ist unentscheidbar.

Beweis. (durch Reduktion auf das PCP)

Gegeben sei eine Instanz des PCP $(a_1, b_1), \dots, (a_k, b_k)$. Transformation in zwei kontextfreie Grammatiken G_1, G_2 , so dass $L(G_1) \cap L(G_2) = \emptyset \Leftrightarrow I$ keine Lösung besitzt.

Idee: Generiere »obere Strings« in Grammatik G_1 und »untere Strings« in Grammatik G_2 .

$$P_1 = \{S_1 \rightarrow c_i a_i\} \cup \{S_1 \rightarrow c_i S_1 a_i\}$$

$$P_2 = \{S_2 \rightarrow c_i b_i\} \cup \{S_2 \rightarrow c_i S_2 b_i\}$$

Die oberen Regeln generieren Wörter $c_{i_1} \dots c_{i_n} a_{i_1} \dots a_{i_n}$, die unteren Regeln $c_{i_1} \dots c_{i_n} b_{i_1} \dots b_{i_n}$. Die hinteren Teile dieser Wörter würden das PCP lösen. \square

Abschließend einige Abschlusseigenschaften der Klassen der Chomsky-Hierarchie:

OPERATION	TYP 3	TYP 2	TYP 1	TYP 0
Vereinigung	ja	ja	ja	ja
Konkatenation	ja	ja	ja	ja
Kleenesche Hülle	ja	ja	ja	ja
Reversal	ja	ja	ja	ja
Durchschnitt	ja	nein ¹	ja	ja
Komplement	ja	nein	ja ²	nein
Schnitt mit regulären Mengen	ja	ja	ja	ja
Homomorphismus	ja	ja	nein ³	ja
inverser Homomorphismus	ja	ja	ja	ja
Quotient mit regulärer Menge	ja	ja	nein ³	ja

Schnitt mit regulären Mengen, Homomorphismus und inverser Homomorphismus werden als Trio-Operationen bezeichnet.

Beachte, dass die in der Tabelle angeführten Operationen *nicht* unabhängig voneinander sind; z.B. Konkatenation mit einer regulären Menge lässt sich durch die Trio-Operationen darstellen.

Bemerkung: Z.B. Abschluss unter Vereinigung bedeutet folgendes: »Wenn L_1, L_2 Typ-3 – Sprachen sind, dann ist auch $L_1 \cup L_2$ eine Typ-3 – Sprache.

Beachte, dass die Rückrichtung i.a. *nicht* gilt. Wähle z.B. $L_1 = \{a^n b^n : n \geq 0\}$ und $L_2 = \{a, b\}^*$. Dann ist $L_1 \cup L_2$ eine Typ-3 – Sprache, aber L_1 nicht.

Reversal ist wie folgt definiert: $L^R = \{w^R : w \in L\}$ und $\lambda^R = \lambda$ und $(a_1 a_2 \dots a_n)^R = (a_n \dots a_2 a_1)^R = a_n \dots a_2 a_1$.

Homomorphismus ist wie folgt definiert: Seien A und B Monoide mit den Operationen \circ_A und \circ_B . Ein Homomorphismus ist eine Abbildung $h : A \rightarrow B$, der folgende Eigenschaft erfüllt: $h(a \circ_A b) = h(a) \circ_B h(b)$ für alle $a, b \in A$. Für eine Sprache $L \subseteq \Sigma^*$ ist $h(L) = \{h(w) : w \in L\}$ und $h^{-1} = \{w : h(w) \in L\}$.

Quotient mit regulärer Menge ist wie folgt definiert: Sei $L \subseteq \Sigma^*$ eine Sprache und $R \subseteq \Sigma^*$ eine reguläre Sprache. Dann ist $L/R = \{w : \exists u \in R : wu \in L\}$, $R \setminus L = \{w : \exists u \in R : uw \in L\}$.

2.4 Endliche Automaten

Ein endlicher Automat ist wie folgt definiert:

¹ $\{a^n b^n c^n : n \geq 0\} = \{a^n b^n c^m : n, m \geq 0\} \cap \{a^m b^n c^n : m, n \geq 0\}$

²Kann mit Hilfe der Technik des induktiven Zählens bewiesen werden (\rightsquigarrow Komplexitätstheorie); dieses Ergebnis ist auch als Immerman-Szelepcsényi-Theorem bekannt.

³Wie mit folgender Aussage bewiesen: Jede Typ-0 – Sprache ist homomorphes Bild einer Typ-1 – Sprache

Definition (nichtdeterministischer endlicher Automat) Ein nichtdeterministischer endlicher Automat (NEA) ist ein 5-Tupel $A = (Q, \Sigma, \delta, q_0, F)$, wobei

- Q endliche Menge von Zuständen
- Σ endliches Alphabet
- q_0 Startzustand ($\in Q$)
- F Endzustand ($\subseteq Q$)
- δ Überföhrungsfunktion $Q \times \Sigma \rightarrow Q$

Ein NEA heißt deterministisch, wenn für alle $q \in Q$ und $a \in \Sigma$ gilt: $|\delta(q, a)| = 1$.

Endliche Automaten werden meist graphisch dargestellt: Die Zustände sind Knoten eines Graphen. Die Überföhrungsfunktion induziert beschriftete Kanten. Startzustand und Endzustände induzieren spezielle Markierungen.

Für die Akzeptanz erweitern wir δ zu $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$ per Induktion. Dann ist die von A akzeptierte Sprache $L(A) = \{w \in \Sigma^* : \delta^*(q_0, w) \cap F \neq \emptyset\}$

Zwei endliche Automaten A_1 und A_2 sind äquivalent falls $L(A_1) = L(A_2)$.

Theorem: Potenzautomatenkonstruktion Zu jedem nichtdeterministischen endlichen Automaten gibt es einen äquivalenten deterministischen endlichen Automaten.

Beweis. Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein nichtdeterministischer endlicher Automat. Wir konstruieren einen deterministischen endlichen Automaten $B = (2^Q, \Sigma, \delta, \{q_0\}, F')$ mit $F' = \{P \subseteq Q : P \cap F \neq \emptyset\}$ und $\delta(P, a) = \bigcup_{p \in P} \delta(p, a)$. Zu zeigen: $L(A) = L(B)$.

\supseteq : Sei $w = a_1 a_2 \dots a_n \in L(A)$. Dann gibt es Zustände q_0, q_1, \dots, q_n mit q_0 ist Anfangszustand, $q_{i+1} \in \delta(q_i, a_{i+1})$ für $i = 0, 1, \dots, n-1$ und $q_n \in F$. Sei P_0, P_1, \dots, P_n die zu w gehörenden Zustände in B (während der Bearbeitung). Per Induktion zeigt man, dass $q_i \in P_i$ gilt. Insbesondere $P_0 = \{q_0\}$ und $P_n \cap F \neq \emptyset$, da $q_n \in P_n$ gilt. Damit wird w auch von B akzeptiert.

\supseteq : Sei $w \in L(B)$ und $w = a_1 a_2 \dots a_n$. Weiter seien P_0, P_1, \dots, P_n die zu w gehörenden Zustände während der Bearbeitung. Dann ist $P_0 = \{q_0\}$, $P_{i+1} = \delta'(P_i, a_{i+1})$ für $i = 0, 1, \dots, n-1$ und $P_n \cap F \neq \emptyset$. Wir wählen ein $q_n \in P_n \cap F$ (beliebig).

Sei $q_{n-1} \in P_{n-1}$ mit $q_n \in \delta(q_{n-1}, a_n)$.

Sei $q_{n-2} \in P_{n-2}$ mit $q_{n-1} \in \delta(q_{n-2}, a_{n-1})$.

...

Sei $q_0 \in P_0$ mit $q_1 \in \delta(q_0, a_1)$.

Die Zustände q_0, q_1, \dots, q_n mit dem Wort w bilden eine akzeptierende Berechnung in A . Daher folgt $w \in L(A)$. \square

Um die Größe von deterministischen endlichen Automaten abzuschätzen, benötigen wir folgende Definition:

Definition: Äquivalenzrelation \sim_L Sei $L \subseteq \Sigma^*$ eine Sprache. Die Relation \sim_L bezeichnet folgende Äquivalenzrelation auf Σ^* :

Für alle $x, y \in \Sigma^*$ gilt $x \sim_L y$ genau dann, wenn für alle $z \in \Sigma^*$: $xz \in L \Leftrightarrow yz \in L$. Wir schreiben $[x]_{\sim_L}$ bzw. $[x]_L$ für die Äquivalenzklasse von x bezüglich \sim_L . Wir schreiben Σ^*/L für den Quotienten Σ^*/\sim_L .

Beispiel Betrachte die Sprache 0^*1^* . Zum Beispiel gilt: $0 \sim_L 00 \sim_L 0^3 \sim_L \dots$. Man sieht, dass alle Elemente aus 0^* in der gleichen Äquivalenzklasse liegen. Entsprechend gilt $1 \sim_L 0^m 1^n$ für $m \geq 0, n \geq 1$. Alle anderen Wörter x sind paarweise äquivalent, da $xz \notin L$.

Theorem: Satz von Myhill und Nerode Sei L eine Sprache. Dann ist L regulär genau dann, wenn der Index von L (Anzahl der Äquivalenzklassen bzgl. \sim_L) endlich ist.

Um diesen Satz zu beweisen benötigen wir noch eine Definition und Hilfsaussagen.

Definition Für einen deterministischen endlichen Automaten $A = (Q, \Sigma, \delta, q_0, F)$ definieren wir die Äquivalenzrelation \sim_A durch $x \sim_A y$ genau dann, wenn $\delta(q_0, x) = \delta(q_0, y)$. Wir schreiben Σ^*/A für den Quotienten Σ^*/\sim_A .

Hilfsaussage Für jeden deterministischen endlichen Automaten A ist der Index von \sim_A endlich.

Beweis. (Idee)

Betrachte die Menge $K_p = \{x \in \Sigma^* : \delta(q_0, x) = p\}$. Für $x, y \in K_p$ gilt $x \sim_A y$. Andererseits gilt $x \not\sim_A y$ für $p \neq q$, $x \in K_p$ und $y \in K_q$. Es gilt $[x]_A = K_p$, falls $\delta(q_0, x) = p$. Daher gilt $\Sigma^*/\sim_A = \{K_p : p \in Q\}$. Es folgt $|\Sigma^*/\sim_A| \leq |Q|$. \square

Hilfsaussage Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein deterministischer endlicher Automat mit $L = L(A)$. Dann gilt:

1. \sim_A ist Verfeinerung von \sim_L , d.h. für alle $x, y \in \Sigma^*$ gilt: Aus $x \sim_A y$ folgt $x \sim_L y$.
2. $|\Sigma^*/\sim_L| \leq |\Sigma^*/\sim_A| \leq |Q|$

Beweis. (Idee)

Es gelte $x \sim_A y$. Weiterhin sei $z \in \Sigma^*$, dann gilt

$$\delta(q_0, xz) = \delta(\delta(q_0, x), z) = \delta(\delta(q_0, y), z) = \delta(q_0, yz)$$

Somit gilt $x \sim_L y$ (da z beliebig gewählt). Daher ist \sim_A eine Verfeinerung von \sim_L .

Gleiche Argumentation wie bei Hilfslemma a liefert Aussage b. \square

Beweis. (Satz von Myhill und Nerode)

\Rightarrow : folgt aus Hilfsaussage b.

\Leftarrow : Aus $x \sim_L y$ folgt $xy \sim_L ya$ für $a \in \Sigma$. Definiere einen deterministischen endlichen Automaten $A_L = (\Sigma^*/\sim_L, \Sigma, \delta_L, [\lambda]_L, F_L)$ mit $F_L = \{[x] : x \in L\}$ und $\delta_L([x]_L, a) = [xa]_L$. Korrektheit per Induktion. Der Automat A_L wird MINIMALER AUTOMAT VON L genannt. \square

Für die Hauptaussage dieses Abschnittes benötigen wir noch folgende Definition:

Definition Zwei deterministische endliche Automaten $A_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ mit $i = 1, 2$ heißen ISOMORPH, wenn es eine bijektive Abbildung $f : Q_1 \rightarrow Q_2$ gibt mit:

- $q_2 = f(q_1)$
- $F_2 = f(F_1)$
- $\delta_2(f(q), a) = f(\delta_1(q, a))$

Offensichtlich gilt: Wenn A_1 und A_2 isomorph sind, dann $L(A_1) = L(A_2)$.

Es folgt die Hauptaussage dieses Abschnittes.

Theorem Sei L eine reguläre Sprache.

1. Der Minimalautomat A_L für L ist der bis auf Isomorphie eindeutig bestimmte deterministische endliche Automat mit folgenden Eigenschaften:
 - (a) $L(A_L) = L$
 - (b) $\sim_{A_L} = \sim_L$
 - (c) Jeder Zustand ist vom Anfangszustand aus erreichbar.
2. Ist $A = (Q, \Sigma, \delta, q_0, F)$ ein deterministischer endlicher Automat mit $L(A) = L$, dann gilt $|\Sigma^*/\sim_L| \leq |Q|$.

Beweis. 1. Wir wissen: $L(A_L) = L$ und \sim_{A_L} ist Verfeinerung von \sim_L . Es bleibt zu zeigen: \sim_L ist Verfeinerung von \sim_{A_L} . Per Induktion zeigt man $\delta_L^*([\lambda]_L, x) = [x]_L$ für alle $x \in \Sigma^*$. Seien $x \sim_L y$. Es folgt:

$$\delta_L^*([\lambda]_L, x) = [x]_L = [y]_L = \delta_L^*([\lambda]_L, y)$$

Also: $x \sim_{A_L} y$ und damit $\sim_{A_L} = \sim_L$.

Da die Äquivalenzklassen nicht leer sind, folgt (c). Damit erfüllt A_L die drei Eigenschaften. Wir zeigen nun, dass jeder deterministische endliche Automat, der die drei Eigenschaften erfüllt, isomorph zu A_L ist.

Sei $B = (Q, \Sigma, \delta, q_0, F)$ ein deterministischer endlicher Automat mit

- (a) $L(B) = L$
- (b) $\sim_B = \sim_L$ und
- (c) jeder Zustand in B ist aus q_0 erreichbar.

Für jeden Zustand $q \in Q$ sei $K_q = \{x \in \Sigma^* : \delta(q_0, x) = q\}$. Die Menge K_q ist eine Äquivalenzklasse bezüglich $\sim_B = \sim_L$. Also ist $K_q \in \Sigma^* / \sim_L$ ein Zustand aus A_L . Man sieht leicht, dass die Abbildung $f : Q \rightarrow \Sigma^* / \sim_L, q \mapsto K_q$ ein Isomorphismus ist.

2. folgt aus der Tatsache, dass \sim_A eine Verfeinerung von \sim_L ist. □

Mit Hilfe der Charakterisierung des minimalen deterministischen endlichen Automaten lässt sich folgende Aussage beweisen:

Theorem (Blow-up NEA \rightarrow DEA) Sei $L_n = \{w \in \{0, 1\}^* : \text{das } n\text{-letzte Zeichen ist } 1\}$. Die Sprache L_n wird durch einen NEA mit $n + 1$ Zuständen akzeptiert und jeder DEA für L_n benötigt mindestens 2^n Zustände.

Beweis. Wir zeigen, dass die Relation \sim_L mindestens 2^n verschiedene Äquivalenzklassen besitzt. Es gilt: Je zwei Wörter $x \neq y \in \{0, 1\}^n$ sind nicht äquivalent bezüglich \sim_L . Sei $x = a_1 a_2 \dots a_n$ und $y = b_1 b_2 \dots b_n$. Dann gibt es ein i mit $a_i \neq b_i$, etwa $a_i = 0$ und $b_i = 1$. Dann ist

$$x0^{i-1} \notin L_n \quad \text{und} \quad y0^{i-1} \in L_n$$

Daraus folgt die Behauptung. □

Die bisherigen Ausführungen führen uns zum Minimierungsalgorithmus für deterministische endliche Automaten. Wir definieren ein Äquivalenzrelation \equiv auf den Zuständen eines Automaten $A = (Q, \Sigma, \delta, q_0, F)$. Weiter:

$$q \equiv p \Leftrightarrow \forall z \in \Sigma^* : \delta^*(q, z) \in F \Leftrightarrow \delta^*(p, z) \in F$$

Wir definieren den Quotientenautomaten

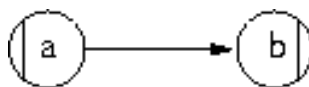
$$A / \equiv = (Q / \equiv, \Sigma, \delta_{\equiv}, [q_0]_{\equiv}, F_{\equiv})$$

wobei

- $F_{\equiv} = \{[q]_{\equiv} : q \in F\}$ und
- $\delta_{\equiv}([q]_{\equiv}, a) = [\delta(q, a)]_{\equiv}$

Man kann leicht zeigen, dass A / \equiv und A äquivalent sind, d.h. $L(A / \equiv) = L(A)$. Weiters gilt: A / \equiv und der Minimalautomat von A sind isomorph (Beweis analog zu \sim_L - und \sim_A -Relationen).

Der Minimalalgorithmus beruht auf der Idee, den Quotientenautomaten zu berechnen. Hierzu werden zukzessive Relationen $R_0 \supseteq R_1 \supseteq R_2 \supseteq \dots$ berechnet, die gegen die Relation \equiv streben. Die initiale Relation R_0 identifiziert alle Endzustände und alle Nicht-Endzustände (zwei Klassen). Im $(i + 1)$ -ten Schritt entfernen wir ein Zustandspaar (q, q') aus der Relation R_i , so dass $(\delta(q, a), \delta(q', a)) \notin R_i$ für ein $a \in \Sigma$ und erhalten die Relation R_{i+1} . Wenn kein solches Paar existiert, dann ist $R_i = \equiv$ (und damit auch gleich zu \sim_L). Die Korrektheit folgt aus den folgenden Fakten:

Abbildung 2.1: Myhill-Graph: $a \in S$, $b \in F$, $(a, b) \in K$

- Aus $p \equiv q$ folgt $p \in F \Leftrightarrow q \in F$
- Für alle $a \in \Sigma$ gilt $\delta(q, a) \equiv \delta(p, a)$

Algorithmus Erstelle eine Tabelle für alle (ungeordneten) Paare (q, q') . Markiere in der Tabelle alle Zustandspaare (q, q') mit $q \in F$ und $q' \notin F$ (oder umgekehrt). Solange es ein unmarkiertes Paar (q, q') und $a \in \Sigma$ gibt, so dass $(\delta(q, a), \delta(q', a))$ markiert ist, wähle dieses Paar und markiere es. Dann bilde die maximalen Mengen paarweise unmarkierter Zustände.

Abschließend einige Bemerkungen zu einfachen Verallgemeinerungen von endlichen Automaten (welche äquivalent zu NEAs und DEAs sind):

- mehrere Anfangszustände
- λ -Kanten (Kanten mit Wörtern)
- modifizierte Akzeptanzbedingungen (z.B. anstelle von $\delta^*(q_0, w) \cap F \neq \emptyset$ jetzt $\delta^*(q_0, w) \subseteq F$)
- ...

2.4.1 Alternative Darstellungen von regulären Mengen

Es folgen einige nicht automatentheoretische Darstellungen. Dazu gehören:

- reguläre Ausdrücke
- Syntaxdiagramme oder beschriftete Myhill-Graphen
- Erkennbarkeit durch Monoide
- Gleichungssysteme
- Logik
- Zweiwege-Automaten

Syntaxdiagramme oder beschriftete Myhill-Graphen

Ein Myhill-Graph ist ein 4-Tupel $G = (X, K, S, F)$, wobei

- X ein endliches Alphabet,
- $S, F \subseteq X$ und
- $K \subseteq X \times X$ sind.

Graphische Darstellung: s. Abb. 2.1

Die Wegemenge $W(G)$ ist wie folgt definiert:

$$W(G) = \{x_1x_2 \dots x_n : x_1 \in S, x_n \in F, (x_i, x_{i+1}) \in K : i = 1, \dots, n-1\}$$

Man kann sich leicht überlegen, dass gilt:

$$W(G) = SX^* \cap X^*F \setminus X^*VX^*$$

wobei $(x, y) \in V \Leftrightarrow (x, y) \notin K$ (verbotene Übergänge). Solche Wortmengen heißen LOKALE MENGEN oder STANDARDMENGEN, da man zum Akzeptieren nur ein »Schiebefenster« der Länge zwei braucht (lokale Eigenschaft). Man beachte, dass nicht jede reguläre Menge lokal ist, z.B. ist $\{ab, bc, ac\}$ keine lokale Menge. — abc ist kein Element der Menge.

Theorem Jede lokale Menge ist regulär.

Auf Myhill-Graphen führe wir eine Knotenbeschriftung ein und erhalten beschriftete Myhill-Graphen, die formal wie folgt definiert sind:

$$G = (X, K, S, F, \Sigma, \beta)$$

wobei $G' = (X, K, S, F)$ ein Myhill-Graph ist und $\beta : X \rightarrow \Sigma \cup \{\lambda\}$ ein alphabetischer Homomorphismus. Die Wegemenge eines beschrifteten Myhill-Graphen $G = (X, K, S, F, \Sigma, \beta)$ ist wie folgt definiert:

$$W(G) = \beta(W(G')) \quad \text{wobei } G' = (X, K, S, F)$$

Beschriftete Myhill-Graphen sind Syntaxdiagramme ohne Rekursion.

Theorem Jede reguläre Menge ist homomorphes Bild (unter einem alphabetischen Homomorphismus) einer lokalen Menge und umgekehrt, d.h. jede reguläre Menge ist Wegemenge eines beschrifteten Myhill-Graphen.

Beweis. Offensichtlich ist jede Wegemenge eines beschrifteten Myhill-Graphen regulär (Abschlusseigenschaften). Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Wir definieren einen Myhill-Graphen $G = (X, K, S, F)$ und einen Homomorphismus $\beta : X \rightarrow \Sigma \cup \{\lambda\}$ wie folgt:

$$\begin{aligned} X &= \{(p, a, q) : p, q \in Q, a \in \Sigma \cup \{\lambda\}\} \\ S &= \{(q_0, a, q) : q \in Q, a \in \Sigma \cup \{\lambda\}\} \\ F &= \{(p, a, p) : p, q \in Q, q \in F, a \in \Sigma \cup \{\lambda\}\} \\ K &= \{(p, a, q')(q', b, q) : p, q', q \in Q, a, b \in \Sigma \cup \{\lambda\}, q' \in \delta^*(p, a), q \in \delta^*(q', b)\} \end{aligned}$$

Der Homomorphismus $\beta : X \rightarrow \Sigma \cup \{\lambda\}$ ist definiert vermöge $\beta((p, a, q')) = a$ für $(p, a, q') \in X$. Man sieht leicht, dass $L(A) = \beta(W(G))$ gilt. \square

Reguläre Ausdrücke

Reguläre Ausdrücke und deren Semantik sind wie folgt definiert:

Definition (regulärer Ausdruck) Sei Σ ein Alphabet. Die Menge der regulären Ausdrücke und deren induzierte Sprachen sind wie folgt definiert:

- \emptyset und λ sind reguläre Ausdrücke mit $L(\emptyset) = \emptyset$ und $L(\lambda) = \{\lambda\}$.
- Für jedes $a \in \Sigma$ ist a ein regulärer Ausdruck mit $L(a) = \{a\}$.
- Mit α und β sind auch $(\alpha + \beta)$, $(\alpha\beta)$, α^* reguläre Ausdrücke. Es gilt $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$, $L(\alpha\beta) = L(\alpha)L(\beta)$ und $L(\alpha^*) = (L(\alpha))^*$.
- Nichts sonst ist ein regulärer Ausdruck.

Bemerkung: Klammern werden oftmals weggelassen, wobei $+$ die schwächste Priorität hat und $*$ die höchste.

Theorem Zu jedem regulären Ausdruck α gibt es einen endlichen Automaten A mit $L(\alpha) = L(A)$ und umgekehrt.

Beweis. \Rightarrow Diese Richtung folgt aus den Abschlusseigenschaften für reguläre Mengen und der Tatsache, dass die Mengen \emptyset , $\{\lambda\}$ und $\{a\}$ regulär sind.

\Leftarrow Es sei $A = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Das (graphische) Verfahren aus Abbildung 2.2 liefert einen regulären Ausdruck mit $L(A) = L(\alpha)$.

Bemerkung: Das vorgestellte Verfahren beruht auf dem Warshall-Algorithmus zur Bestimmung von Wegen in Graphen.

Endliches Anwenden dieser Regeln führt zu der Situation aus Abbildung 2.3. Dann gilt $L(\alpha) = L(A)$. \square

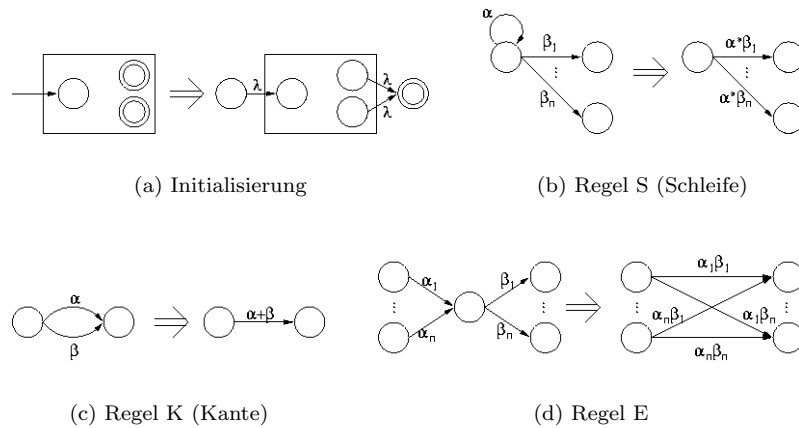


Abbildung 2.2: Äquivalenz von regulären Ausdrücken und endlichen Automaten

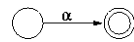


Abbildung 2.3: Ergebnis

Erkennbarkeit

Es folgt die algebraische Sichtweise der formalen Sprachen. Wir definieren Erkennbarkeit wie folgt:

Definition (Erkennbarkeit) : Sei Σ ein Alphabet. Eine Menge $L \subseteq \Sigma^*$ wird von einem Monoid erkannt, wenn es einen Homomorphismus $\varphi : \Sigma^* \rightarrow M$ gibt und ein $E \subseteq M$ mit $L = \varphi^{-1}(E)$. Für reguläre Mengen gilt folgende Aussage:

Theorem Eine Menge $L \subseteq \Sigma^*$ ist regulär genau dann, wenn L von einem endlichen Monoid erkannt wird.

Beweis. Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein deterministischer endliche Automat. Jedes Wort $w \in \Sigma^*$ induziert eine Abbildung $w_A : Q \rightarrow Q$ vermöge $w_A(p) = \delta^*(p, w)$. Wie man leicht sieht, bildet die Menge aller Abbildungen von Q nach Q zusammen mit der Identität ($\text{id} = \lambda_A$) und der Hintereinanderausführung von Abbildungen ein Monoid. Dieses Monoid wird Transitionsmonoid von A genannt. Dann gilt mit $E = \{w_A : Q \rightarrow Q : \delta^*(q_0, w) \in F\}$ und dem Homomorphismus $\varphi : \Sigma^* \rightarrow M$, definiert vermöge $\varphi(a) = a_A$ für $a \in \Sigma$ die Aussage $L = \varphi^{-1}(E)$.

Für die Umkehrung sei $L = \varphi^{-1}(E)$ mit $E \subseteq M$. Definiere einen endlichen Automaten $A = (M, \Sigma, \delta, q_0, E)$ mit $q_0 = \varphi(\delta) = \text{id}_M$ und $\delta(m, a) = m\varphi(a)$ für $m \in M$ und $a \in \Sigma$. Per Induktion folgt $\delta^*(m, x) = m\varphi(x)$ für alle $m \in M$ und $x \in \Sigma^*$. Dann ist $\varphi^{-1}(E) = L(A)$, da

$$x \in L(A) \Leftrightarrow \delta^*(q_0, x) \in E \Leftrightarrow \varphi(\lambda)\varphi(x) \in E \Leftrightarrow \varphi(x) \in E \Leftrightarrow x \in \varphi^{-1}(E)$$

□

Hinweis: Das Transitionsmonoid eines deterministischen endlichen Automaten lässt sich berechnen, da die Menge aller Abbildungen $Q \rightarrow Q$ endlich ist.

Beispiel Berechnen mit Hilfe einer Tabelle (vgl. Abb. 2.4)

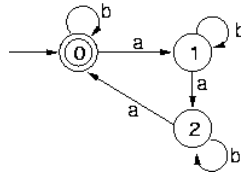


Abbildung 2.4: Beispielautomat

	0	1	2
λ_A	0	1	2
a_A	1	2	0
b_A	$= \lambda_A$		
a_A^2	2	0	1
$a_A b_A$	$= a_A$		
$b_A a_A$	$= a_A$		
$b_A b_A$	$= \text{id}_A$		
a_A^3	$= \text{id}_A$		

Jedes andere Wort der Länge ≥ 3 lässt sich auf ein kürzeres Wort reduzieren. Das Transitionsmonoid ist

$$M = \{\lambda_A, a_A, a_A^2\}$$

mit der Verknüpfungstafel

	λ_A	a_A	a_A^2
λ_A	λ_A	a_A	a_A^2
a_A	a_A	a_A^2	λ_A
a_A^2	a_A^2	λ_A	a_A

und Homomorphismus $\varphi(a) = a_A$ für $a \in \Sigma$. Dann gilt $L = \varphi^{-1}(\{\lambda_A\})$.

Ähnlich wie im Fall von deterministischen endlichen Automaten, wo es ein kanonisches Objekt (minimaler Automat) gibt, so gibt es auch ein kanonisches Monoid. Hierzu definieren wir die syntaktische Relation.

Definition: Syntaktische Relation Für eine Sprache $L \subseteq \Sigma^*$ definieren wir

$$x \approx_L y \Leftrightarrow_{\text{def}} \forall z, z' \in \Sigma^* : z x z' \in L \Leftrightarrow z y z' \in L$$

Die Relation \approx_L ist Äquivalenzrelation und $u \approx_L v$, $u' \approx_L v'$ impliziert $u u' \approx_L v v'$. Aus $u \approx_L v$ folgt $u \sim_L v$ (Myhill-Nerode). Die Umkehrung gilt i.a. nicht. Z.B. $L = b b a^*$. Dann ist $a \sim_L b a$, aber $a \not\approx_L b a$, da $b \cdot a \cdot \lambda \notin L$ und $b \cdot b a \cdot \lambda \in L$.

Das syntaktische Monoid von L ist dann Σ^* / \approx_L mit der Operation $[u]_{\approx_L} \cdot [v]_{\approx_L} = [u v]_{\approx_L}$ und dem neutralen Element $[\lambda]_{\approx_L}$. Die Abbildung $\eta_L : \Sigma^* \rightarrow \Sigma^* / \approx_L$, definiert vermöge $\eta_L(x) = [x]_{\approx_L}$ heißt SYNTAKTISCHER HOMOMORPHISMUS von L . Mit $E_L = \eta_L(L) = \{[x]_{\approx_L} : x \in L\}$ gilt dann $L = \eta_L^{-1}(E_L)$. Die Sprache L wird vom syntaktischen Monoid erkannt.

Folgendes Theorem zeigt, dass Erkennbarkeit immer durch das syntaktische Monoid erreicht werden kann.

Theorem Eine Sprache $L \subseteq \Sigma^*$ ist genau dann von einem Monoid M erkennbar, wenn es ein Untermonoid $M' \subseteq M$, einen surjektiven Homomorphismus $\psi : M^* \rightarrow \Sigma^* / \approx_L$ gibt und einen Homomorphismus $\varphi : \Sigma^* \rightarrow M'$ mit $\eta_L = \psi \circ \varphi$.

Beweis. Es sei $L = \varphi^{-1}(E)$ für einen Homomorphismus $\varphi : \Sigma^* \rightarrow M$ und $E \subseteq M$. Setze $M' = \varphi(\Sigma^*)$. Damit ist $\varphi : \Sigma^* \rightarrow M'$ ein surjektiver Homomorphismus. Ist $m \in M'$, so gibt es ein $x \in \Sigma^*$ mit $\varphi(x) = m$. Dann definieren wir $\psi(m) = \eta_L(x)$. Diese Abbildung ist wohldefiniert, da für $m = \varphi(x) = \varphi(y)$ gilt $\varphi(z x z') = \varphi(z y z')$ für alle $z, z' \in \Sigma^*$. Also

$$z x z' \in L \Leftrightarrow \varphi(z x z') \in E \Leftrightarrow \varphi(z y z') \in E \Leftrightarrow z y z' \in L$$

Hieraus folgt $\eta_L(x) = \eta_L(y)$. Darüber hinaus ist ψ ein Homomorphismus, da für $m = \varphi(x)$ und $m' = \varphi(y)$ folgt:

$$\psi(mm') = \psi(\varphi(x)\varphi(y)) = \psi(\varphi(xy)) = \eta_L(xy) = \eta_L(x)\eta_L(y) = \psi(\varphi(x))\psi(\varphi(y)) = \psi(m)\psi(m')$$

Die Surjektivität von ψ folgt, da für $[x]_{\approx_L} \in \Sigma^*/\approx_L$ gilt $\psi(\varphi(x)) = [x]_{\approx_L}$, da $\eta_L(x) = \psi(\varphi(x))$.

Die Umkehrung sieht man wie folgt: Sei $\eta_L = \psi \circ \varphi$ für einen surjektiven Homomorphismus $\psi : M' \rightarrow \Sigma^*/\approx_L$ und $M' \subseteq M$. Setze $E = \psi^{-1}(E_L)$, wobei $E_L = \eta_L(L)$. Dann gilt

$$\varphi^{-1}(E) = \varphi^{-1}(\psi^{-1}(E_L)) = \eta_L^{-1}(E_L) = L$$

Damit ist L erkennbar. □

Theorem Sei $L \subseteq \Sigma^*$ eine reguläre Sprache und A_L der minimale deterministische endliche Automat für L . Das syntaktische Monoid Σ^*/\approx_L ist isomorph zum Transitionsmonoid von A_L .

Beweis. Sei $T(A_L)$ das Transitionsmonoid von A_L . Definiere $f : \Sigma^*/\approx_L \rightarrow T(A_L)$ vermöge $f([x]_{\approx_L}) = x_{A_L}$ für $x \in \Sigma^*$. Die Abbildung f ist wohldefiniert (und injektiv), da

$$[x]_{\approx_L} = [y]_{\approx_L} \Leftrightarrow x_{A_L} = y_{A_L}$$

gilt. Dies folgt, da

$$\begin{aligned} [x]_{\approx_L} = [y]_{\approx_L} &\Leftrightarrow \forall z, z' \in \Sigma^* : zxz' \in L \Leftrightarrow zyz' \in L \\ &\Leftrightarrow \forall z : [zx]_{\sim_L} = [zy]_{\sim_L} \\ &\Leftrightarrow \forall z : \delta_{A_L}^*([z]_{\sim_L}, x) = \delta_{A_L}^*([z]_{\sim_L}, y) \\ &\Leftrightarrow x_{A_L} = y_{A_L} \end{aligned}$$

Die Surjektivität von f ist unmittelbar, da $x_{A_L} = f([x]_{\approx_L})$ für alle $x \in \Sigma^*$. Es bleibt zu zeigen, dass f ein Homomorphismus ist. Dies folgt, da

$$f([x]_{\approx_L}[y]_{\approx_L}) = f([xy]_{\approx_L}) = (xy)_{A_L} = x_{A_L} \circ y_{A_L} = f([x]_{\approx_L})f([y]_{\approx_L})$$

□

Rechnen mit regulären Ausdrücken und Gleichungssysteme

Die Äquivalenz von regulären Ausdrücken kann man mit Hilfe eines Axiomensystems berechnen. Wir benötigen noch eine Definition.

Definition (Leerworteigenschaft) Ein regulärer Ausdruck α besitzt die Leerworteigenschaft, falls $\lambda \in L(\alpha)$ gilt.

Bemerkung: Die Leerworteigenschaft lässt sich auch induktiv über die regulären Ausdrücke definieren.

Wir führen Axiome und Rechenregeln ein. Die Axiome (nach Saloma und Urponen) lauten:

1. $\lambda = \emptyset^*$
2. $(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$
3. $(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma)$
4. $\alpha + \beta = \beta + \alpha$
5. $\alpha \cdot (\beta + \gamma) = \alpha \cdot \beta + \alpha \cdot \gamma$
6. $(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma$
7. $\alpha \cdot \lambda = \alpha$
8. $\alpha \cdot \emptyset = \emptyset$

$$9. \alpha^* = \alpha \alpha^* + \lambda$$

$$10. \alpha^* = (\alpha + \lambda)^*$$

Die Rechenregeln sind:

1. Kongruenz: Wenn $\alpha = \beta$ gilt, kann in einem regulären Ausdruck γ_1 ein Vorkommen von α durch β ersetzt werden. Für den resultierenden regulären Ausdruck γ_2 gilt $\gamma_1 = \gamma_2$.
2. Symmetrie: Wenn $\alpha = \beta$ gilt, dann auch $\beta = \alpha$.
3. Transitivität: Wenn $\alpha = \beta$ und $\beta = \gamma$ gilt, dann auch $\alpha = \gamma$.
4. Gleichungsauflösung: Wenn die Gleichung $\alpha = \beta \cdot \alpha + \gamma$ gilt, dann gilt auch die Gleichung $\alpha = \beta^* \cdot \gamma$. Wenn β die Leerworteigenschaft *nicht* erfüllt, dann ist $\alpha = \beta^* \cdot \gamma$ eindeutige Lösung.

Offensichtlich sind alle Axiome und Rechenregeln korrekt. Dass $\alpha = \beta^* \cdot \gamma$ eine Lösung der Gleichungsauflösung ist, sieht man durch einsetzen. Beachte, dass die Gleichung $\alpha = \beta \cdot \alpha + \gamma$ eine Fixpunktgleichung ist, deren kleinster Fixpunkt (mengentheoretisch) sich durch Approximation berechnen lässt: Definiere hierzu

$$\begin{aligned} \alpha_0 &= \beta \cdot \emptyset + \gamma \\ \alpha_{i+1} &= \beta \cdot \alpha_i + \gamma = \beta^i \cdot \gamma + \beta^{i-1} \cdot \gamma + \dots + \beta \cdot \gamma + \gamma \quad (i \geq 0) \end{aligned}$$

Lösung: $\alpha = \beta^* \cdot \gamma$. Beachte, dass diese Lösung eindeutig ist, falls β die Leerworteigenschaft nicht erfüllt.

Bemerkung: Falls β die Leerworteigenschaft erfüllt, dann ist jeder reguläre Ausdruck $\alpha' = \beta^* \cdot (\gamma + \delta)$ für beliebigen regulären Ausdruck δ eine Lösung.

Es folgt das Haupttheorem:

Theorem Das oben angegebene Axiomensystem zusammen mit den Rechenregeln ist gültig und vollständig, d.h. alle Äquivalenzen der Form $L(\alpha) = L(\beta)$ für reguläre Ausdrücke α, β lassen sich mit diesen Regeln beweisen.

Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein verallgemeinerter endlicher Automat (auf den Kanten dürfen reguläre Ausdrücke stehen) mit $Q = \{q_0, q_1, \dots, q_n\}$ und r_{ij} der reguläre Ausdruck auf den Kanten von q_i nach q_j . Das zu A gehörende Gleichungssystem sieht wie folgt aus:

$$\begin{aligned} y_0 &= r_{00}y_0 + r_{01}y_1 + \dots + r_{0n}y_n + \delta_0 \\ &\vdots \\ y_n &= r_{n0}y_0 + r_{n1}y_1 + \dots + r_{nn}y_n + \delta_n \end{aligned}$$

wobei y_0, \dots, y_n Variablen sind und $\delta_i = \lambda$, falls $q_i \in F$ und $\delta_i = \emptyset$ sonst. Verkürzend schreiben wir

$$\vec{y} = M\vec{y} + \vec{\delta} \tag{2.1}$$

wobei $M = (r_{ij})$ die $(n+1) \times (n+1)$ -Matrix der r_{ij} ist, $\vec{\delta}$ der Spaltenvektor der δ_i und \vec{y} der Spaltenvektor der y . Folgende Aussage gilt für die Gleichung 2.1 — ohne Beweis.

Theorem Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein verallgemeinerter endlicher Automat mit $Q = \{q_0, q_1, \dots, q_n\}$. Für $k = 0, 1, \dots, n$ sei $L_k = \{w \in \Sigma^* : \delta^*(q_k, w) \cap F \neq \emptyset\}$ und \vec{L} sei der Spaltenvektor der L_i . Ferner sei $\vec{y} = M\vec{y} + \vec{\delta}$ ds zu A gehörige Gleichungssystem. Dann gilt:

1. \vec{L} erfüllt das Gleichungssystem, d.h. $\vec{L} = M\vec{L} + \vec{\delta}$.
2. \vec{L} lässt sich durch Iteration berechnen (ausgehend von $\vec{L}^{(0)} = \vec{\delta}$ und $\vec{L}^{(k+1)} = M\vec{L}^{(k)} + \vec{\delta}$, d.h. $\vec{L} = M^* \vec{\delta}$)

3. Für jede Lösung \vec{L}' des Gleichungssystems gilt $\vec{L} \subseteq \vec{L}'$.
4. Das Gleichungssystem hat die eindeutige Lösung $M^* \vec{\delta}$, falls M nicht die Leerworteigenschaft besitzt. Die Matrix $M = (r_{ij})$ besitzt die Leerworteigenschaft, falls es eine Folge von Indizes i_1, i_2, \dots, i_k mit $k \geq 1$ gibt, so dass $\lambda \in L(r_{i_p i_{p+1}})$ für $1 \leq p \leq k-1$ und $\lambda \in L(r_{i_k i_1})$ gilt.

Beachte: L_0 ist die von A akzeptierte Sprache

$$\vec{L} = \begin{pmatrix} L_0 \\ L_1 \\ \vdots \\ L_n \end{pmatrix}$$

2.5 Endliche Automaten über unendlichen Wörtern

Zuerst einige Definitionen: Ein unendlich langes Wort über einem endlichen Alphabet Σ ist eine unendliche Sequenz von Elementen aus Σ , d.h. $v = a_0 a_1 \dots a_n \dots$. Ein unendlich langes Wort v lässt sich als Abbildung von \mathbb{N} nach Σ darstellen (auffassen), d.h. $v(n) = a_n$ für $n \geq 0$. Mit Σ^ω bezeichnen wir die Menge aller unendlichen Wörter über Σ und wir setzen $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. Die Konkatenation von $u = a_0 a_1 \dots a_n \in \Sigma^*$ und $v = b_1 b_2 \dots \in \Sigma^*$ ist $uv = a_0 a_1 \dots a_n b_1 b_2 \dots \in \Sigma^\omega$. Für $A \subseteq \Sigma^*$ sei $A^\omega = \{x_0 x_1 \dots x_n \dots : x_i \in A \setminus \{\lambda\} \text{ für } i \geq 0\}$. Es gelten folgende Eigenschaften (ohne Beweis).

Theorem Für $A, B \subseteq \Sigma^*$ gilt:

1. $(A + B)^\omega = (A^* B)^\omega + (A + B)^* B^\omega$
2. $(AB)^\omega = A(BA)^\omega$
3. $(A^n)^\omega = (A^+)^\omega = A^\omega$ für $n \geq 1$
4. $AA^\omega = A^+ A^\omega = A^\omega$

Wir definieren die ω -regulären Mengen von Σ^∞ : Dies ist die kleinste Menge R mit der Eigenschaft:

1. $\emptyset \in R$ und für alle $a \in \Sigma$ gilt $\{a\} \in R$,
2. R ist unter endlicher Vereinigung abgeschlossen,
3. für alle $A \subseteq \Sigma^*$ und $B \subseteq \Sigma^\infty$ mit $A, B \in R$ folgt $AB \in R$ und
4. für alle $A \subseteq \Sigma^*$ mit $A \in R$ folgt $A^\omega \in R$.

Folgendes Theorem charakterisiert die ω -regulären Mengen von Σ^ω . (Diese Mengen werden im Folgenden ω -reguläre Mengen genannt.)

Theorem Eine Teilmenge von Σ^ω ist ω -regulär genau dann, wenn sie endliche Vereinigung von Mengen der Form AB^ω ist, wobei A und B reguläre Teilmengen von Σ^* sind.

Beweis. per Induktion (komplizierter Fall ist (3) von oben) □

2.5.1 Büchi-Automaten

Büchi-Automaten sind wie folgt definiert: Ein Büchi-Automat ist ein 5-Tupel $A = (Q, \Sigma, \delta, q_0, F)$, wobei $Q, \Sigma, \delta, q_0, F$ wie im Fall endlicher Automaten definiert sind. Eine Berechnung von A auf dem Wort $v = a_0 a_1 \dots a_n \dots \in \Sigma^\omega$ ist eine Folge $\vec{r}(\omega)$ von Konfigurationen

$$(q_0, a_0 a_1 \dots a_n \dots) \vdash_A (q_1, a_1 a_2 \dots a_n \dots) \vdash_A (q_2, a_2 \dots a_n \dots) \vdash_A \dots$$

mit $q_{i+1} \in \delta(q_i a_i)$ für $i \geq 0$. Sei nun $\text{inf}(\bar{r}(\omega))$ die Menge von Zuständen, die unendlich oft in der Berechnung $\bar{r}(\omega)$ vorkommen. Die von A akzeptierte Menge ist dann

$$L^\omega(A) = \{w \in \Sigma^\omega : \text{es gibt eine Berechnung } \bar{r}(\omega) \text{ mit } \text{inf}(\bar{r}(\omega)) \cap F \neq \emptyset\}$$

d.h. mindestens ein Endzustand muss unendlich oft gesehen werden.

Ein Büchi-Automat A mit Zustandsmenge Q ist deterministisch, falls δ eine Abbildung von Q nach Q ist.

Lemma Jede nicht leere Menge, die durch einen Büchi-Automaten akzeptiert wird, enthält ein ultimativ periodisches Wort, d.h. ein Wort der Form uv^ω .

Beweis. Offensichtlich, da es auf Grund der Nichtleerheit von L Wörter u, v geben muss mit $(q_0, u) \vdash_A^* (q, \lambda)$ und $(q, v) \vdash_A^* (q, \lambda)$ und $q \in F$, wobei \vdash_A^* die reflexiv-transitive Hülle von \vdash_A ist. Dann wird aber uv^ω von A akzeptiert. \square

Nachfolgendes Theorem beschreibt die Beziehung von Büchi-akzeptierbaren Mengen und ω -regulären Mengen.

Theorem eine Sprache $L \subseteq \Sigma^\omega$ über einem Alphabet Σ ist Büchi-akzeptierbar genau dann, wenn L eine ω -reguläre Sprache ist.

Beweis. ähnlich wie im endlichen Wortfall \square

Im endlichen Wortfall sind deterministische und nichtdeterministische endliche Automaten gleichwertig. Bei Büchi-Automaten ist dies nicht der all. Um das zu zeigen, führen wir folgende Operation ein: Sei $L \subseteq \Sigma^*$, dann ist $\overrightarrow{L} = \{v \in \Sigma^\omega : \text{unendlich viele Präfixe von } v \text{ sind in } L\}$.

Beispiel

- $L = a^*b, \overrightarrow{L} = \emptyset$
- $L = (ab)^+, \overrightarrow{L} = (ab)^\omega$
- $L = (a^*b)^+ = (a+b)^*b, \overrightarrow{L} = (a^*b)^\omega$

Folgendes Beispiel zeigt, dass sich nicht jede ω -reguläre Menge in der Form \overrightarrow{L} schreiben lässt:

Beispiel Sei $(a+b)^*a^\omega$ die Menge der Wörter, die nur endlich viele bs enthalten. Angenommen, diese Menge lässt sich in der Form $\overrightarrow{L} = (a+b)^*a^\omega$ schreiben für ein $L \subseteq \Sigma^*$. Dann hat das Wort ba^ω ein Präfix ba^{n_1} in L , das Wort $ba^{n_1}ba^\omega$ einen Präfix $ba^{n_1}ba^{n_2}$ in L usw. Daher gibt es eine Folge (unendlich) n_1, n_2, \dots so, dass das Wort $u = ba^{n_1}ba^{n_2}ba^{n_3} \dots$ unendlich viele Präfixe in L hat. Damit muss $u \in \overrightarrow{L} = (a+b)^*a^\omega$ gelten. Dies ist ein Widerspruch zur Annahme, da n unendlich viele bs enthält.

Folgendes Theorem zeigt die Beziehung zwischen deterministischen Büchi-Automaten und der oben eingeführten Operation.

Theorem Sei A ein deterministischer Büchi-Automat. Dann gilt $L^\omega(A) = \overrightarrow{L(A)}$.

Beweis. Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein deterministischer Büchi-Automat und $w = a_0a_1 \dots \in L^\omega(A)$. Dann gibt es eine Berechnung $(q_0, a_0a_1 \dots) \vdash_A (q_1, a_1a_2 \dots) \vdash_A \dots$ und $n_0 < n_1 < n_2 < \dots$ so, dass $q_{n_i} \in F$ für $i \geq 0$. Es gilt $w_k = a_0a_1 \dots a_{n_k-1}$ ist in $L(A)$ und damit ein Präfix von w . Hieraus folgt $L^\omega(A) \subseteq \overrightarrow{L(A)}$. Umgekehrt sei $\omega \in \overrightarrow{L(A)}$. Damit hat w unendlich viele Präfixe in $L(A)$. Da A deterministisch ist, kann man folgern, dass w eine Berechnung induziert, welche unendlich oft durch F läuft. Hieraus folgt $L^\omega(A) \supseteq L(A)$. \square

Aus obigem Theorem folgt unmittelbar eine Charakterisierung deterministischer Büchi-Mengen.

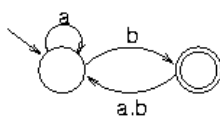


Abbildung 2.5: Büchi-Automat

Korollar Sei $L \subseteq \Sigma^\omega$ für ein Alphabet Σ . Folgende Aussagen sind äquivalent:

1. L wird durch einen deterministischen Büchi-Automaten akzeptiert.
2. Es gibt reguläre Mengen $M \subseteq \Sigma^*$ mit $L = \overrightarrow{M}$.

Darüber hinaus kann man folgende Aussage beweisen:

Theorem Jede Sprache, die von einem deterministischen Büchi-Automaten akzeptiert wird, ist ω -regulär, aber nicht jede ω -reguläre Sprache wird von einem deterministischen Büchi-Automaten akzeptiert.

Beweis. Die Inklusion deterministische Büchi-Automaten-Sprache in Büchi-Automaten-Sprache ist trivial. Die Echtheit folgt mit obigem Beispiel $(a + b)^* a^\omega$. \square

Es folgen einige Abschlusseigenschaften von deterministischen Büchi-Mengen.

Theorem Deterministische Büchi-Sprachen sind nicht unter Komplementbildung abgeschlossen.

Beweis. Betrachte $L = \{v \in \Sigma^\omega : v \text{ enthält unendlich viele } bs\}$. Diese Sprache wird vom deterministischen Büchi-Automaten in Abbildung 2.5 akzeptiert. Wie oben gesehen, ist $\Sigma^* \setminus L$ keine deterministische Büchi-Automaten-Sprache. \square

Theorem Deterministische Büchi-Sprachen sind unter Vereinigung und Schnitt abgeschlossen.

Beweis. Die Vereinigung folgt aus

$$\bigcup_{1 \leq i \leq n} \overrightarrow{L_i} = \overrightarrow{\bigcup_{1 \leq i \leq n} L_i}$$

und obigem Korollar. Für den Schnitt benötigen wir eine spezielle Konstruktion. Seien $A_i = (Q_i, \Sigma, \delta_i, q_{i,0}, F_i)$ deterministische Büchi-Automaten für $i = 1, 2$. Definiere $B = (Q_1 \times Q_2 \times \{0, 1, 2\}, \Sigma, \delta, (q_{1,0}, q_{2,0}, 0), F)$ mit

- $\delta((p_1, p_2, s), a) = (q_1, q_2, t)$, falls $\delta_1(p_1, a) = q_1$ und $\delta_2(p_2, a) = q_2$ und
 1. $t = 0$, falls $s = 2$
 2. $t = 1$, falls $s = 0$ und $q_1 \in F_1$
 3. $t = 2$, falls $s = 0$ und $q_2 \in F_2$
 4. $t = s$ sonst
- $F = Q_1 \times F_2 \times \{2\}$. Damit speichert die dritte Zustandskomponente, ob ein Zustand aus F_1 und dann ein Zustand aus F_2 usw. besucht wird.

Dann lässt sich per Induktion zeigen, dass $L^\omega(B) = L^\omega(A_1) \cap L^\omega(A_2)$ gilt. \square

Bemerkung: Die Abschlusseigenschaften Schnitt und Vereinigung gelten auch für Büchi-Sprachen.

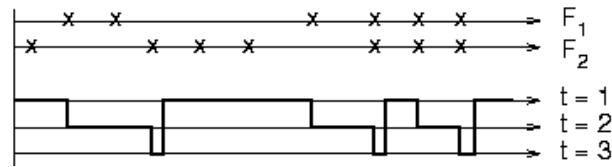


Abbildung 2.6: Ablauf in einem Büchi-Automaten

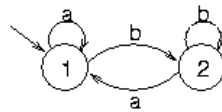


Abbildung 2.7: Muller-Automat

2.5.2 Muller-Automaten

Formal ist ein Muller-Automat ein 5-Tupel $A = (Q, \Sigma, \delta, q_0, \mathcal{F})$, wobei Q, Σ, δ und q_0 wie bei einem Büchi-Automaten definiert sind und $\mathcal{F} \subseteq 2^Q$. Die vom Muller-Automaten akzeptierte Sprache ist

$$L^\omega(A) = \{v \in \Sigma^\omega : \text{es gibt eine Berechnung } \pi(v) \text{ mit } \inf(\pi(v)) \in \mathcal{F}\}$$

Im nachfolgenden betrachten wir hauptsächlich deterministische Muller-Automaten (analoge Definition wie bei Büchi-Automaten).

Beispiel Betrachte den deterministischen Muller-Automaten A aus Abbildung 2.7 mit $\mathcal{F} = \{\{2\}\}$. Es gilt $L^\omega(A) = (a + b)^*b^\omega$.

Theorem Für einen (deterministischen) Muller-Automaten A ist die Sprache $L^\omega(A)$ ω -regulär.

Beweis. Sei $A = (Q, \Sigma, \delta, q_0, \mathcal{F})$ ein Muller-Automat. Ohne Einschränkung können wir annehmen, dass \mathcal{F} einelementig ist (Abschluss unter Vereinigung). Sei $\mathcal{F} = \{\{t_0, t_1, \dots, t_k\}\}$. Sei X die Menge von endlichen Wörtern, die vom Startzustand q_0 nach t_0 führen. Für $0 \leq i < k$ (bzw. $i = k$) sei Y_i die Menge von endlichen Wörtern, die vom Zustand t_i nach t_{i+1} (bzw. von t_k nach t_0) führen und nur Zustände aus $\{t_0, t_1, \dots, t_k\}$ besuchen. Dann gilt $L^\omega(A) = X(Y_0Y_1 \dots Y_k)^\omega$. \square

Theorem Die deterministischen Muller-Automaten-Sprachen sind unter Komplementbildung und Schnitt abgeschlossen (und damit unter Vereinigung).

Es folgt das Haupttheorem für Muller-Automaten.

Theorem Folgende Aussagen für $L \subseteq \Sigma^\omega$ sind äquivalent:

1. L wird von einem deterministischen Muller-Automaten akzeptiert.
2. L ist von der Form $L = \bigcup_{1 \leq i \leq n} (U_i \setminus V_i)$, wobei U_i und V_i deterministische Büchi-Automaten-Sprachen sind.
3. L ist boolesche Kombination von deterministischen Büchi-Automaten-Sprachen.

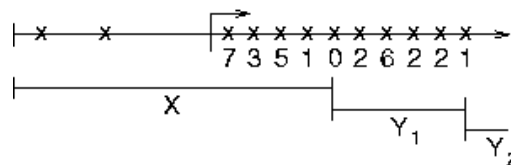


Abbildung 2.8: Ablauf in einem Muller-Automaten

Beweis. $1 \Rightarrow 2$ Sei $A = (Q, \Sigma, \delta, q_0, \mathcal{F})$ ein deterministischer Muller-Automat. Ohne Einschränkung sei \mathcal{F} einelementig, d.h. $\mathcal{F} = \{F\}$. Dann gilt:

$$L^\omega(A) = \bigcap_{t \in F} L^\omega(B_t) \setminus \bigcap_{t \notin F} L^\omega(B_t)$$

wobei B_t der deterministische Büchi-Automat $B_t = (Q, \Sigma, \delta, q_0, \{t\})$ ist. Da deterministische Büchi-Automaten unter Schnitt und Vereinigung abgeschlossen sind, folgt die Behauptung.

$2 \Rightarrow 3$ offensichtlich

$3 \Rightarrow 1$ Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein deterministischer Büchi-Automat. Dann akzeptiert der deterministische Muller-Automat $B = (Q, \Sigma, \delta, q_0, \mathcal{F})$ mit $F = \{T \subseteq Q : T \cap F \neq \emptyset\}$ die gleiche Menge wie A . Der Abschluss unter booleschen Kombinationen von deterministischen Muller-Automaten-Sprachen folgt aus obigem Theorem. \square

2.5.3 Rabin-Automaten

Obiges Theorem (Bedingung 2) motiviert ein neues Automatenmodell. Ein Rabin-Automat ist ein 5-Tupel $A = (Q, \Sigma, \delta, q_0, \mathcal{R})$, wobei Q, Σ, δ, q_0 wie bei Büchi-Automaten definiert sind und $\mathcal{R} = \{(L_i, U_i) : L_i, U_i \subseteq Q, 1 \leq i \leq n\}$. Die vom Rabin-Automaten akzeptierte Sprache ist $L^\omega(A) = \{v \in \Sigma^\omega : \text{es gibt eine Berechnung } \pi(v) \text{ und ein } i \text{ mit } \inf(\pi(v)) \cap L_i = \emptyset \text{ und } \inf(\pi(v)) \cap U_i \neq \emptyset\}$. Wie im Fall von Muller-Automaten betrachten wir hauptsächlich deterministische Rabin-Automaten.

Theorem Jeder deterministische Rabin-Automat ist zu einem deterministischen Muller-Automaten äquivalent und umgekehrt.

Beweis. Sei $A = (Q, \Sigma, \delta, q_0, \mathcal{R})$ ein deterministischer Rabin-Automat. Definiere

$$\mathcal{F} = \{T \subseteq Q : T \cap L = \emptyset \wedge T \cap U \neq \emptyset \text{ für } (L, U) \in \mathcal{R}\}$$

Offensichtlich akzeptiert der deterministische Muller-Automat $B = (Q, \Sigma, \delta, q_0, \mathcal{F})$ die Menge $L^\omega(A)$.

Sei L eine deterministische Muller-Automaten-Sprache. Dann lässt sich L in der Form $L = \bigcup_{1 \leq i \leq n} (U_i \setminus V_i)$ schreiben, wobei U_i, V_i deterministische Büchi-Automaten-Sprachen sind. Betrachte zuerst die deterministischen Büchi-Automaten-Mengen X_1 und X_2 , die durch $A_i = (Q_i, \Sigma, \delta_i, q_{0,i}, F_i)$ akzeptiert werden. Dann akzeptiert der Rabin-Automat

$$B = (Q_1 \times Q_2, \Sigma, \delta, (q_{0,1}, q_{0,2}), \mathcal{R})$$

mit

- $\delta((q_1, q_2), a) = (p_1, p_2)$, falls $\delta(q_1, a) = p_1$ und $\delta(q_2, a) = p_2$ und
- $\mathcal{R} = \{(Q_1 \times F_2, F_1 \times Q_2)\}$

die Menge $X_1 \setminus X_2$.

Sei nun $A_i = (Q_i, \Sigma, \delta_i, q_{0,i}, \mathcal{R}_i)$ der deterministische Rabin-Automat, der $U_i \setminus V_i$ akzeptiert mit $\mathcal{R}_i = \{(L_i, U_i)\}$. Dann akzeptiert der deterministische Rabin-Automat

$$A = (Q_1 \times Q_2 \times \dots \times Q_n, \Sigma, \delta, (q_{0,1}, \dots, q_{0,n}), \mathcal{R})$$

mit

- $\delta((q_1, q_2, \dots, q_n), a) = (p_1, p_2, \dots, p_n)$, falls $\delta_i(q_i, a) = p_i$ für $1 \leq i \leq n$ und
- $\mathcal{R} = \{(Q_1 \times Q_2 \times \dots \times Q_{j-1} \times L_j \times Q_{j+1} \times \dots \times Q_n, Q_1 \times Q_2 \times \dots \times Q_{j-1} \times U_j \times Q_{j+1} \times \dots \times Q_n) : (L_j, U_j) \in \mathcal{R}_j, 1 \leq j \leq n\}$

die Menge L . \square

Abschließend das Haupttheorem für Rabin-Automaten.

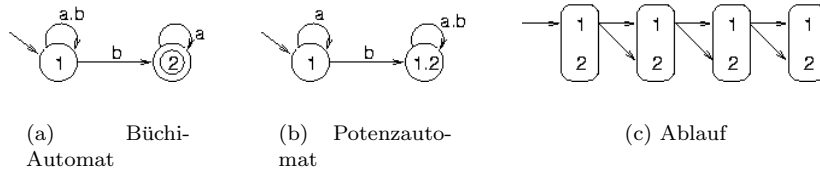


Abbildung 2.9: Büchi-Automaten-Sprachen und Rabin-Automaten

Theorem Jede nichtdeterministische Büchi-Automaten-Sprache wird von einem deterministischen Rabin-Automaten akzeptiert.

Die Beweisidee basiert auf einer Verallgemeinerung der Potenzautomatenkonstruktion. Betrachte z.B. den Büchi-Automaten aus Abbildung 2.9 (a), der die Sprache $(a + b)^*ba^\omega$ akzeptiert. Die Potenzautomatenkonstruktion liefert den Automaten aus Abbildung 2.9 (b). Dieser Automat liefert b^ω ! Das Problem kommt von der Situation aus Abbildung 2.9 (c). Der Endzustand 2 kommt zwar in jeder Menge vor, aber es gibt keine Berechnung, auf der der Endzustand unendlich oft vorkommt. Daher kommen wir zu folgender Idee: Betrachte den Ablauf

$$I = \{q_0\} \xrightarrow{u_0} S_0 \xrightarrow{u_1} S_1 \xrightarrow{u_2} \dots$$

($S_i \subseteq Q$, u_i gelesene Wörter) mit der Eigenschaft

- $S_0 \subseteq \delta(I, u_0)$ und $S_{n+1} \subseteq \delta(s_n, u_{n+1})$ für $n \geq 0$.
- für jedes $n \geq 0$ und $q \in S_{n+1}$ gibt es $p \in S_n$ und einen Pfad $p \xrightarrow{u_{n+1}} q$ im Automaten, der durch einen Endzustand führt.