Theoretische Informatik

Elmar Eder
Universität Salzburg

6. Mai 2021

Inhaltsverzeichnis

1 Mathematische Hilfsbegriffe **4**
- 1.1 Natürliche Zahlen . . . 4
- 1.2 Mengenlehre . . . 4
- 1.3 Symbole der Logik . . . 5
- 1.4 Wörter und Sprachen . . . 5
- 1.5 Induktion und Rekursion . . . 8
  - 1.5.1 Das Prinzip der vollständigen Induktion . . . 8
  - 1.5.2 Wertverlaufsinduktion und -rekursion . . . 10
  - 1.5.3 Der Begriff der induktiven Definition . . . 11
  - 1.5.4 Formale Systeme . . . 14

2 Berechenbarkeitstheorie **16**
- 2.1 Berechenbarkeit und Entscheidbarkeit . . . 16
  - 2.1.1 Entscheidbarkeit . . . 16
- 2.2 μ-rekursive Funktionen . . . 18
  - 2.2.1 Zahlentheoretische Funktionen . . . 18
  - 2.2.2 Primitivrekursive Funktionen . . . 19
  - 2.2.3 Existenz von berechenbaren nicht primitivrekursiven Funktionen . . . 39
  - 2.2.4 Partielle zahlentheoretische Funktionen . . . 40
  - 2.2.5 μ-partiellrekursive Funktionen . . . 41
- 2.3 Logikkalküle . . . 44
  - 2.3.1 Aussagenlogik . . . 44
  - 2.3.2 Prädikatenlogik erster Stufe . . . 57
  - 2.3.3 Allgemeingültigkeitsdefinierbarkeit durch eine Formel . . . 75
  - 2.3.4 Gödelisierung . . . 77
  - 2.3.5 Existenzdarstellung für allgemeingültigkeitsdefinierbare Prädikate . . . 81
  - 2.3.6 Der Normalformsatz von Kleene (Formulierung für allgemeingültigkeitsdefinierbare partielle Funktionen) . . . 82
  - 2.3.7 Äquivalenz von μ-Partiellrekursivität und Allgemeingültigkeitsdefinierbarkeit 83
  - 2.3.8 Unentscheidbarkeit der Existenz eines Wertes . . . 83
  - 2.3.9 Unentscheidbarkeitssätze der Logik . . . 83
- 2.4 Kalküle . . . 84
  - 2.4.1 Grundbegriffe zu den Kalkülen . . . 84
  - 2.4.2 Einige Eigenschaften des Kalkülbegriffs . . . 92
  - 2.4.3 Kalküle und Logik . . . 94
  - 2.4.4 Berechenbarkeit der μ-partiellrekursiven Funktionen durch Kalküle . . . 96
  - 2.4.5 Kodierung von Kalkülen . . . 100

2.4.6 Ein Kalkül für die Herleitbarkeitsrelation in Kalkülen . . . 102
2.4.7 Kalkül-Unentscheidbarkeit der Herleitbarkeit in Kalkülen . . . 103
2.4.8 μ-Partiellrekursivität der kalkülberechenbaren Funktionen . . . 104
2.4.9 Normalformsatz von Kleene . . . 108
2.5 Chomsky-Grammatiken . . . 109
2.5.1 Semi-Thue-Systeme . . . 109
2.5.2 Chomsky-Grammatiken . . . 110
2.5.3 Berechenbarkeit durch eine Grammatik . . . 112
2.6 Turingmaschinen . . . 113
2.6.1 Grundbegriffe . . . 113
2.6.2 Zusammensetzung von Turingmaschinen . . . 117
2.6.3 Weitere Beispiele für Turingmaschinen . . . 119
2.6.4 Turing-Berechenbarkeit . . . 120
2.7 Äquivalenz von μ-Partiellrekursivität, Kalkülberechenbarkeit, Grammatikberechenbarkeit und Turingberechenbarkeit . . . 123
2.8 Partiellrekursive und rekursive Funktionen . . . 124
2.9 Churchsche These . . . 124
2.10 Normalformsatz von Kleene . . . 125
2.11 Rekursiv aufzählbare und rekursive Mengen . . . 126
2.12 Varianten von Turingmaschinen . . . 129
2.12.1 Nichtdeterministische Turingmaschinen . . . 129
2.13 Universelle Turingmaschinen . . . 130
2.14 Unentscheidbare Probleme . . . 131
2.14.1 Das Halteproblem für Turingmaschinen . . . 131
2.14.2 Terminierung von Computerprogrammen . . . 132
2.14.3 Weitere unentscheidbare Probleme . . . 132

3 Komplexitätstheorie **134**
3.1 Einleitung . . . 134
3.2 P und NP . . . 135
3.3 Das $P = NP$-Problem . . . 136
3.4 Das Erfüllbarkeitsproblem der Aussagenlogik . . . 137
3.5 NP-Vollständigkeit . . . 138
3.6 NP-Vollständigkeit von SAT . . . 139
3.7 Nachweis der NP-Vollständigkeit einer Sprache durch Polynomialzeitreduktion . . . 140
3.8 Kryptologie . . . 141

4 Formale Semantik **143**
4.1 Semantik in der logischen Programmierung . . . 143
4.2 Andere Programmiersprachen . . . 144
4.2.1 Funktionale Programmierung . . . 144
4.2.2 While-Programme . . . 144

Einleitung

In diesem Skript werden drei Fragenkomplexe, die Algorithmen oder Computerprogramme betreffen, behandelt.

Definition 1 Unter einem *Algorithmus* versteht man eine formalisierbare Rechenvorschrift.

Ein Algorithmus kann zu gegebenen Eingaben (englisch 'input') eine Ausgabe (englisch 'output') produzieren.

Beispiel 1
für einen Algorithmus. Der *Euklidische Algorithmus* ist ein Algorithmus, der zu zwei eingegebenen natürlichen Zahlen (Eingaben) den größten gemeinsamen Teiler (ggT) dieser beiden Zahlen als Ausgabe liefert.

Definition 2 Ein *Computerprogramm* ist ein in computerlesbarer Form formalisierter Algorithmus.

Die folgenden drei Fragenkomplexe werden uns beschäftigen.
********** ACHTUNG!!! Die Nummerierungen haben sich geändert und werden sich noch ändern. Der folgende Text muss später noch berichtigt werden!!! **********

1. Was kann ein Algorithmus leisten?
   Anders ausgedrückt: Welche Probleme sind prinzipiell algorithmisch lösbar?
   Diese Fragen werden in Kapitel 1 **Berechenbarkeitstheorie** behandelt.

2. Mit welchem Aufwand (Kosten) kann er das leisten?
   Hiermit ist der Verbrauch an Resourcen wie Rechenzeit oder Speicherplatz gemeint.
   Diese Fragen werden in Kapitel 2 **Komplexitätstheorie** behandelt.

3. Wie beschreibt man das, was ein Computerprogramm leistet?
   Dies ist die Frage nach der Bedeutung eines Computerprogramms.
   Diese Frage wird in Kapitel 3 **Formale Semantik** behandelt.

Kapitel 1

Mathematische Hilfsbegriffe

1.1 Natürliche Zahlen

Unter den *natürlichen Zahlen* verstehen wir die nichtnegativen ganzen Zahlen, also die Zahlen $0, 1, 2, 3, \ldots$. Die Menge der natürlichen Zahlen bezeichnet man mit $\mathbb{N}$. Es ist also $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$. Die Menge der postiven natürlichen Zahlen $1, 2, 3, \ldots$ ist $\mathbb{N} \setminus \{0\} = \{1, 2, 3, \ldots\}$.

Zu jeder natürlichen Zahl n gibt es eine nächste natürliche Zahl n', die man den *Nachfolger* von n nennt. Es gilt also $n' = n + 1$. Jede natürliche Zahl kann durch wiederholte Bildung des Nachfolgers aus der Null erzeugt werden:

(Regel 1) Die Zahl 0 ist eine natürliche Zahl.

(Regel 2) Wenn x eine natürliche Zahl ist, dann ist auch x' eine natürliche Zahl.

Die natürlichen Zahlen sind genau diejenigen Dinge, die sich durch – möglicherweise wiederholte – Anwendung der *Regeln* (Regel 1) und (Regel 2) als natürliche Zahlen nachweisen lassen.

Zum Beispiel lässt sich die Zahl 3, also $0'''$, folgendermaßen als natürliche Zahl nachweisen:

1. Aufgrund von (Regel 1) ist 0 eine natürliche Zahl.
2. Aus 1. folgt mit (Regel 2), dass $0'$ eine natürliche Zahl ist.
3. Aus 2. folgt mit (Regel 2), dass $0''$ eine natürliche Zahl ist.
4. Aus 3. folgt mit (Regel 2), dass $0'''$ eine natürliche Zahl ist.

Wir sagen, die Regeln (Regel 1) und (Regel 2) bilden ein *Erzeugungssystem* für die Menge der natürlichen Zahlen.

1.2 Mengenlehre

Wir verwenden die üblichen Notationen der Mengenlehre. Wenn A und B Mengen sind, dann ist $A \cup B$ die Vereinigung von A und B, $A \cap B$ der Durchschnitt von A und B und $A \setminus B$ die Differenz von A und B. Mit $\{x_1, \ldots, x_n\}$ bezeichnen wir die Menge, die genau die Elemente $x_1, \ldots, x_n$ enthält, und mit $\{x \mid P(x)\}$ die Menge aller Dinge x mit der Eigenschaft $P(x)$. Weiter bedeutet $a \in A$, dass a ein Element der Menge A ist. Und $A \subseteq B$ bedeutet, dass die Menge A

eine Teilmenge der Menge B ist. Wenn A eine Menge ist, so ist die *Potenzmenge* $\mathfrak{P}(A)$ definiert als die Menge aller Teilmengen von A, also

$$\mathfrak{P}(A) = \{X \mid X \subseteq A\}.$$

Wenn A eine Menge von Mengen ist, so bezeichnet man mit $\bigcup A$ die Vereinigung von A. Das ist die Menge aller Objekte x derart, dass es ein $X \in A$ gibt mit $x \in X$. Dies ist also die Menge aller Elemente von Elementen von A. Wenn A eine nichtleere Menge von Mengen ist, so bezeichnet man mit $\bigcap A$ den Durchschnitt von A. Das ist die Menge aller Objekte x derart, dass für alle $X \in A$ gilt $x \in X$.

Mit (x_1, x_2) bezeichnen wir das Paar, das aus den Komponenten x_1 und x_2 besteht. Allgemein bezeichnen wir mit $(x_1, \ldots, x_n)$ das n-Tupel, das aus den Komponenten $x_1, \ldots, x_n$ besteht. Wenn X und Y Mengen sind, dann bedeutet $f\colon X \to Y$, dass f eine Funktion von X nach Y ist. Der Definitionsbereich X von f wird mit $\mathrm{Def}(f)$ bezeichnet und das Ziel Y von f wird mit $\mathrm{Ziel}(f)$ bezeichnet. Es gilt also $f\colon \mathrm{Def}(f) \to \mathrm{Ziel}(f)$. Eine Funktion wird manchmal auch bezeichnet durch Angabe eines allgemeinen Argumentwertes und des zugehörigen Funktionswertes mit dem Symbol $\mapsto$ dazwischen. So kann man eine Funktion f auch schreiben als $x \mapsto f(x)$, genauer: $x \mapsto f(x)\colon \mathrm{Def}(f) \to \mathrm{Ziel}(f)$. Auch die Notation $\lambda x.f(x)$ ist üblich. Die Menge $\{(x, f(x)) \mid x \in \mathrm{Def}(f)\}$ heißt der *Graph* der Funktion f und wird mit $\mathrm{Graph}(f)$ bezeichnet. Wenn $f\colon X \to Y$ eine Funktion ist und $A \subseteq X$ ist, dann ist die *Einschränkung* $f|_A$ der Funktion f auf die Menge A definiert durch $f|_A\colon A \to Y$ und $f|_A(x) = f(x)$ für $x \in A$. Die Menge aller Funktionen $f\colon X \to Y$ von X nach Y wird mit Y^X bezeichnet.

1.3 Symbole der Logik

Wir werden die folgenden Symbole der Logik verwenden.

- $\neg$ ist das logische Zeichen für „nicht“. $\neg F$ ist genau dann wahr, wenn F falsch ist. Z.B. bedeutet $\neg 2 < 3$, dass 2 nicht kleiner als 3 ist (eine falsche Aussage).
- $\wedge$ ist das logische Zeichen für „und“. $F \wedge G$ ist genau dann wahr, wenn F wahr ist und G wahr ist.
- $\vee$ ist das logische Zeichen für „oder“. $F \vee G$ ist genau dann wahr, wenn mindestens eine der beiden Aussagen F und G wahr ist (auch wenn beide wahr sind).
- $\Longrightarrow$ oder $\to$ ist das logische Zeichen für „impliziert“ („wenn ..., dann ...“; materielle Implikation). $F \Longrightarrow G$ ist genau dann wahr, wenn F falsch ist oder G wahr ist.
- $\forall$ oder $\bigwedge$ ist das logische Zeichen für „für alle“. $\forall x\, F$ oder $\bigwedge_x F$ ist genau dann wahr, wenn F für jeden Wert von x aus einem vorgegebenen Individuenbereich wahr ist.
- $\exists$ oder $\bigvee$ ist das logische Zeichen für „es gibt“. $\exists x\, F$ oder $\bigvee_x F$ ist genau dann wahr, wenn F für mindestens einen Wert von x aus einem vorgegebenen Individuenbereich wahr ist.

1.4 Wörter und Sprachen

Definition 3

1. Unter einem *Anfangsabschnitt* von $\mathbb{N}\setminus\{0\}$ versteht man eine endliche Teilmenge $\mathcal{A} \subseteq \mathbb{N}\setminus\{0\}$ so, dass gilt
$$\bigwedge_{y\in\mathcal{A}} \bigwedge_{x\in\mathbb{N}\setminus\{0\}} (x < y \Longrightarrow x \in \mathcal{A}).$$
2. Für $n \in \mathbb{N}$ wird $\mathcal{A}_n$ definiert als
$$\mathcal{A}_n := \{x \in \mathbb{N} \setminus \{0\} \mid x \leq n\}$$

Beispiel 2

$$\begin{aligned}\mathcal{A}_0 &= \emptyset\\ \mathcal{A}_1 &= \{1\}\\ \mathcal{A}_2 &= \{1,2\}\\ \mathcal{A}_3 &= \{1,2,3\}\end{aligned}$$

Alle diese Mengen sind Anfangsabschnitte von $\mathbb{N} \setminus \{0\}$.

Es gilt: Die Anfangsabschnitte von $\mathbb{N} \setminus \{0\}$ sind genau die Mengen $\mathcal{A}_n$ mit $n \in \mathbb{N}$.

Definition 4

1. Ein *Alphabet* ist eine endliche Menge Σ von Zeichen.
2. Ein *Wort* (Zeichenreihe, Zeichenkette, englisch string, word) über einem Alphabet Σ ist eine Funktion $w\colon \mathcal{A} \to \Sigma$, wobei $\mathcal{A}$ ein Anfangsabschnitt von $\mathbb{N} \setminus \{0\}$ ist.

Beispiel 3
$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$. Das Wort $w\colon \mathcal{A}_5 \to \Sigma$ sei definiert durch

$$\begin{aligned}w(1) &= \mathtt{g}\\ w(2) &= \mathtt{e}\\ w(3) &= \mathtt{h}\\ w(4) &= \mathtt{e}\\ w(5) &= \mathtt{n}\end{aligned}$$

Definition 5 Sei Σ ein Alphabet. Dann bezeichnet man mit Σ^* die Menge aller Wörter über Σ:

$$\Sigma^* = \{w \mid w \text{ ist ein Wort über } \Sigma\}.$$

Notation 1 Wenn Σ ein Alphabet ist und $w\colon \mathcal{A}_n \to \Sigma$ ein Wort über Σ ist, so schreibt man w auch als $w(1)w(2)\ldots w(n)$. Im obigen Beispiel kann man das Wort w also auch schreiben als `gehen`.

Das *leere Wort* ε ist durch $\varepsilon\colon \emptyset \to \Sigma$ definiert.

Frage: Wozu betrachtet man Alphabete und Wörter (Zeichenreihen) über einem Alphabet?

1. Allgemeinheit, d.h. jede Information, die sich formalisieren und einem Computer mitteilen lässt, ist als Zeichenreihe darstellbar.
2. Mathematische Einfachheit.

Operationen für Wörter

1. *Einbettung* $\iota\colon \Sigma \to \Sigma^*$ mit $\iota(\sigma)\colon \mathcal{A}_1 \to \Sigma$ und $\iota(\sigma)(1) = \sigma$ für $\sigma \in \Sigma$.

2. *Konkatenation* $\cdot\colon \Sigma^* \times \Sigma^* \to \Sigma^*$. Sei $u\colon \mathcal{A}_m \to \Sigma$ und $v\colon \mathcal{A}_n \to \Sigma$. Dann ist $u \cdot v\colon \mathcal{A}_{m+n} \to \Sigma$ definiert durch
$$(u \cdot v)(k) = \begin{cases} u(k) & \text{für } k \leq m \\ v(k-m) & \text{für } m < k \leq m+n. \end{cases}$$
Zum Beispiel gilt $\texttt{ge} \cdot \texttt{hen} = \texttt{gehen}$.
Anstatt $u \cdot v$ schreibt man auch einfach uv.

3. Länge $|\cdot|\colon \Sigma^* \to \mathbb{N}$. Wenn $u\colon \mathcal{A}_m \to \Sigma$, dann ist $|u| = m$.

4. n-tes Zeichen $\pi(n, w)$ von w mit
$\pi\colon \{(n, w) \mid n \in \mathbb{N} \setminus \{0\} \wedge w \in \Sigma^* \wedge n \leq |w|\} \to \Sigma$ und $\pi(n, w) = w(n)$.

5. n-te Potenz $\mathrm{pot}(n, w) = w^n$ von w mit $\mathrm{pot}\colon \mathbb{N} \times \Sigma^* \to \Sigma^*$. Definition durch vollständige Induktion nach n (siehe nächster Abschnitt):
Wenn $w \in \Sigma^*$ ist, so ist
$$\begin{aligned} w^0 &= \varepsilon \\ w^{n'} &= w^n w \qquad \text{für } n \in \mathbb{N}. \end{aligned}$$

6. Spiegelung $\cdot^R\colon \Sigma^* \to \Sigma^*$. Sei $w\colon \mathcal{A}_n \to \Sigma$. Dann ist $w^R\colon \mathcal{A}_n \to \Sigma$ definiert durch $w^R(k) = w(n-k+1)$. Zum Beispiel gilt
$\texttt{gehen}^R(1) = \texttt{gehen}(5-1+1) = \texttt{gehen}(5) = \texttt{n}$
$\texttt{gehen}^R(2) = \texttt{gehen}(4) = \texttt{e}$
$\ldots$
$\texttt{gehen}^R = \texttt{neheg}$

Definition 6 Unter einer *Sprache* (englisch *language*) L über einem Alphabet Σ versteht man eine beliebige Teilmenge von Σ^*.

Beispiel 4
$\Sigma = \{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}, \texttt{Ä}, \texttt{Ö}, \texttt{Ü}, \texttt{a}, \texttt{b}, \ldots, \texttt{z}, \texttt{ä}, \texttt{ö}, \texttt{ü}, \texttt{ß}\}$ mit
$L =$ Menge der Wörter der deutschen Sprache.
Weitere Beispiele für Sprachen sind

1. $\Sigma = \{\texttt{a}\}$
$L = \{w \mid w \in \Sigma^* \wedge |w| \text{ ist gerade}\} = \{\varepsilon, \texttt{aa}, \texttt{aaaa}, \ldots\}$

2. $\Sigma = \{\texttt{a}, \texttt{b}\}$
$L = \{w \mid w \in \Sigma^* \wedge w \text{ enthält gleich viele } \texttt{a} \text{ und } \texttt{b}\} =$
$\{\varepsilon, \texttt{ab}, \texttt{ba}, \texttt{aabb}, \texttt{abab}, \ldots\}$

3. $\Sigma = \{\texttt{a}, \texttt{b}\}$
$L = \{w \mid w \in \Sigma^* \wedge \neg \bigvee_{u,v \in \Sigma^*} w = u\texttt{aa}v\} =$
$\{\varepsilon, \texttt{a}, \texttt{b}, \texttt{ab}, \texttt{ba}, \texttt{bb}, \ldots\}$

4. $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{+}, \texttt{*}, \texttt{(}, \texttt{)}\}$. Sei L die Menge der arithmetischen Ausdrücke über $\texttt{0}$ und $\texttt{1}$. Hier gilt zum Beispiel $\texttt{(1+0+(1+1)*1)*1} \in L$, aber $\texttt{)(1+1)} \notin L$ und $\texttt{10} \notin L$.

1.5 Induktion und Rekursion

1.5.1 Das Prinzip der vollständigen Induktion

Definition 7 Unter einem *Prädikat* auf einer Menge A versteht man eine Teilmenge $P \subseteq A$ von A. Wenn $x \in P$ ist, dann schreibt man auch $P(x)$ und man sagt, es *gelte* $P(x)$ oder, das Prädikat P *treffe auf* x *zu*.

Beispiel 5
Sei P die Menge der Primzahlen. Dann ist P ein Prädikat auf $\mathbb{N}$. Das Prädikat P trifft auf jede Primzahl zu und trifft auf alle natürlichen Zahlen, die nicht Primzahlen sind, nicht zu. Man sagt auch, P sei das Prädikat '... ist Primzahl'.

Ein Prädikat P auf einer Menge A wird dadurch eindeutig definiert, dass man für jedes $x \in A$ angibt, ob $P(x)$ gilt oder nicht. Im obigen Beispiel etwa könnte man das Prädikat P auch folgendermaßen definieren.

$$P(n) : \Longleftrightarrow \ n \text{ ist Primzahl}$$

Prinzip 1 (vollständige Induktion)
Sei P ein Prädikat auf $\mathbb{N}$ mit den folgenden Eigenschaften

INDUKTIONSANFANG $P(0)$

INDUKTIONSSCHRITT $\bigwedge_{n\in\mathbb{N}}\big(P(n) \Longrightarrow P(n')\big)$.

Dann gilt $\bigwedge_{n\in\mathbb{N}} P(n)$.

Im Induktionsschritt nennt man die Aussage $P(n)$ die *Induktionsvoraussetzung* und die Aussage $P(n')$ die *Induktionsbehauptung*.

Definition 8 Unter einem *Beweis* von $P(n)$ *durch vollständige Induktion* nach n versteht man einen Beweis von $P(n)$, der folgendermaßen abläuft.

Beweis von $P(n)$ durch Induktion nach n

INDUKTIONSANFANG: $P(0)$.

Beweis: *** Hier folgt der Beweis von $P(0)$ ***

INDUKTIONSVORAUSSETZUNG: $P(n)$

INDUKTIONSBEHAUPTUNG: $P(n')$

Beweis: *** Hier folgt der Beweis von $P(n')$ unter der Voraussetzung $P(n)$ ***

Nach dem Prinzip der vollständigen Induktion gilt $P(n)$ für alle $n \in \mathbb{N}$.

Beispiel 6
Eine natürliche Zahl n heißt *gerade*, wenn gilt $\bigvee_{x\in\mathbb{N}} n = 2x$. Eine natürliche Zahl n heißt *ungerade*, wenn gilt $\bigvee_{x\in\mathbb{N}} n = 2x + 1$. Wir wollen beweisen, dass jede natürliche Zahl n gerade oder ungerade ist. Hierzu betrachten wir das folgende Prädikat P auf $\mathbb{N}$

$$P(n) : \Longleftrightarrow \ n \text{ ist gerade} \vee n \text{ ist ungerade}$$

und beweisen $P(n)$ durch vollständige Induktion nach n. Das schaut für dieses Prädikat wie folgt aus.

Beweis von

$$n \text{ ist gerade} \vee n \text{ ist ungerade}$$

durch vollständige Induktion nach n

INDUKTIONSANFANG: 0 ist gerade $\vee$ 0 ist ungerade.

Beweis: Wegen $0 = 2 \cdot 0$ ist 0 gerade. Also gilt die Behauptung.

INDUKTIONSVORAUSSETZUNG: n ist gerade $\vee$ n ist ungerade

INDUKTIONSBEHAUPTUNG: n' ist gerade $\vee$ n' ist ungerade

Beweis: Nach Induktionsvoraussetzung ist n gerade oder ungerade. Entsprechend unterscheiden wir zwei Fälle:

1. **Fall** n ist gerade:
   Dann gibt es ein $x \in \mathbb{N}$ so, dass $n = 2x$. Dann ist $n' = 2x + 1$ und damit ist n' ungerade und daher gilt die Induktionsbehauptung.
2. **Fall** n ist ungerade:
   Dann gibt es ein $x \in \mathbb{N}$ so, dass $n = 2x + 1$. Dann ist $n' = (2x + 1) + 1 = 2x + 2 = 2(x + 1)$. Damit ist n' gerade und es folgt die Induktionsbehauptung.

Nach dem Prinzip der vollständigen Induktion gilt

$$n \text{ ist gerade} \vee n \text{ ist ungerade}$$

für alle $n \in \mathbb{N}$.

Satz 1 (Definition durch vollständige Induktion) *Sei A eine Menge, $a \in A$ und $g\colon A \times \mathbb{N} \to A$. Dann gibt es genau eine Funktion $h\colon \mathbb{N} \to A$, die folgende Gleichungen erfüllt.*

$$h(0) = a \tag{1.1}$$
$$h(x') = g(h(x), x). \tag{1.2}$$

Diese beiden Gleichungen werden *Rekursionsgleichungen* genannt. Man sagt, dass sie die Funktion h *durch vollständige Induktion definieren*. Eine solche Definition ist eine spezielle Art von *Rekursion*, d.h. einer Definition eines Begriffs durch sich selbst.

Beispiel 7
Seien $a_0, a_1, a_2, \dots \in \mathbb{R}$. Dann wird durch die Rekursionsgleichungen

$$h(0) = a_0$$
$$h(x') = h(x) + a_{x'}$$

eine Funktion $h\colon \mathbb{N} \to \mathbb{R}$ definiert. Dies ist ein Beispiel für eine Definition durch vollständige Induktion, wobei $A = \mathbb{R}$, $a = a_0$ und g die Funktion $(s, x) \mapsto s + a_{x'}\colon \mathbb{R} \times \mathbb{N} \to \mathbb{R}$ ist. Für $h(x)$ schreibt man üblicherweise $\sum_{i=0}^{x} a_i$. Die Rekursionsgleichungen schauen dann so aus.

$$\sum_{i=0}^{0} a_i = a_0$$
$$\sum_{i=0}^{x'} a_i = \sum_{i=0}^{x} a_i + a_{x'}$$

Man sagt, dass diese Rekursionsgleichungen die Zahl $\sum_{i=0}^{x} a_i$ für jedes $x \in \mathbb{N}$ *durch vollständige Induktion nach x definieren.*

Beispiel 8
Nehmen wir an, wir wissen, was die Addition von natürlichen Zahlen ist. Dann kann man die Multiplikation folgendermaßen definieren. Für $x, y \in \mathbb{N}$ wird $x \cdot y$ durch vollständige Induktion nach y definiert durch die folgenden Rekursionsgleichungen.

$$x \cdot 0 = 0$$
$$x \cdot y' = x \cdot y + x$$

Dies ist ebenfalls ein Beispiel für eine Definition durch vollständige Induktion. Hier ist $A = \mathbb{N}$, $a = 0$ und g ist die Funktion $(p, y) \mapsto p + x \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

1.5.2 Wertverlaufsinduktion und -rekursion

Eine Variante des Prinzips der vollständigen Induktion ist das

Satz 2 (Prinzip der Wertverlaufsinduktion) *Sei P ein Prädikat auf $\mathbb{N}$ mit der folgenden Eigenschaft*

INDUKTIONSSCHRITT:
$$\bigwedge_{n \in \mathbb{N}} \big(\bigwedge_{x<n} P(x) \Longrightarrow P(n) \big).$$

Dann gilt $\bigwedge_{n \in \mathbb{N}} P(n)$.

Im Induktionsschritt nennt man die Aussage $\bigwedge_{x<n} P(x)$ die *Induktionsvoraussetzung* und die Aussage $P(n)$ die *Induktionsbehauptung*.

Im Gegensatz zum Prinzip der vollständigen Induktion ist bei der Wertverlaufsinduktion kein Induktionsanfang nötig, da der Induktionsschritt für $n = 0$ den Induktionsanfang bereits liefert.

Beispiel 9
Unter einer *Primzahl* versteht man eine natürliche Zahl > 1, die nur durch 1 und durch sich selbst teilbar ist. Wir beweisen durch Wertverlaufsinduktion nach n:

Jede natürliche Zahl $n > 1$ ist durch eine Primzahl teilbar.

Beweis durch Wertverlaufsinduktion nach n

INDUKTIONSVORAUSSETZUNG: Jede natürliche Zahl x mit $1 < x < n$ ist durch eine Primzahl teilbar.

INDUKTIONSBEHAUPTUNG: n ist durch eine Primzahl teilbar, falls $n > 1$ ist.

Beweis: Wenn n eine Primzahl ist, dann ist n durch sich selbst teilbar und damit gilt die Behauptung. Ist aber n keine Primzahl, so ist n nach Definition des Begriffs der Primzahl durch eine Zahl x mit $1 < x < n$ teilbar. Nach Induktionsvoraussetzung ist x durch eine Primzahl teilbar. Damit ist auch n durch dieselbe Primzahl teilbar und somit gilt auch in diesem Fall die Induktionsbehauptung.

Nach dem Prinzip der Wertverlaufsinduktion ist daher jede natürliche Zahl $n > 1$ durch eine Primzahl teilbar.

Das Prinzip der vollständigen Induktion ist ein Spezialfall der Wertverlaufsinduktion. Umgekehrt lässt sich das Prinzip der Wertverlaufsinduktion mit dem Prinzip der vollständigen Induktion beweisen.

Wie das Prinzip der vollständigen Induktion, so lässt sich auch das Prinzip der Wertverlaufsinduktion zur Definition verwenden. Man spricht dann von *Wertverlaufsrekursion*. Bei der Wertverlaufsrekursion einer Funktion $h : \mathbb{N} \to A$ wird $h(n)$ durch Bezugnahme auf die Werte $h(x)$ von h mit $x < n$, definiert. Die Definition von $h(n)$ nimmt also Bezug auf die Einschränkung $h|_{\{x|x\in\mathbb{N}\wedge x<n\}}$.

Satz 3 (Wertverlaufsrekursion) *Sei A eine Menge und sei F eine Funktion, die jeder Funktion $f\colon \{x \mid x \in \mathbb{N} \wedge x < n\} \to A$ ein $F(f) \in A$ zuordnet. Dann gibt es genau eine Funktion $h\colon \mathbb{N} \to A$ mit*

$$\bigwedge_{n\in\mathbb{N}} h(n) = F(h|_{\{x|x\in\mathbb{N}\wedge x<n\}}). \tag{1.3}$$

Beispiel 10
Die Fibonaccizahlen $\mathrm{fib}(n)$ für $n = 0, 1, 2, \ldots$ sind definiert durch

$$\mathrm{fib}(n) = \begin{cases} 1, & \text{wenn } n = 0 \\ 1, & \text{wenn } n = 1 \\ \mathrm{fib}(n-2) + \mathrm{fib}(n-1) & \text{sonst.} \end{cases}$$

Dies ist eine Wertverlaufsrekursion mit $A = \mathbb{N}$ und $F(f) := f(n-2)+f(n-1)$ für alle $n \in \mathbb{N}$ mit $n > 1$ und für alle $f\colon \{x \mid x \in \mathbb{N} \wedge x < n\} \to \mathbb{N}$ sowie $F(f) := 1$ für $f\colon \emptyset \to \mathbb{N}$ oder $f\colon \{0\} \to \mathbb{N}$.

1.5.3 Der Begriff der induktiven Definition

Definition 9 Sei A eine Menge und sei $R \subseteq \mathfrak{P}(A) \times A$. Weiter sei $B \subseteq A$ derart, dass gilt

$$\bigwedge_{(X,y)\in R} (X \subseteq B \Longrightarrow y \in B).$$

Dann sagt man, die Menge B sei *abgeschlossen* gegenüber der Relation R.

Beispiel 11
Sei Σ das Alphabet $\{\mathtt{0}, \mathtt{1}, \mathtt{+}, \mathtt{*}, \mathtt{(}, \mathtt{)}\}$ und sei $A = \Sigma^*$ die Menge der Wörter über dem Alphabet Σ. Sei B die Menge der voll geklammerten arithmetischen Ausdrücke über `0` und `1`. Die Relation $R_+ \subseteq \mathfrak{P}(\Sigma^*) \times \Sigma^*$ sei folgendermaßen definiert. Es gelte $(X, y) \in R_+$ genau dann, wenn $u, v \in \Sigma^*$ existieren so, dass $X = \{u, v\}$ und $y = \mathtt{(}u\mathtt{+}v\mathtt{)}$. Dann ist die Menge B abgeschlossen gegenüber der Relation R_+. Man sagt auch, die Menge B sei *abgeschlossen* gegenüber der zweistelligen Operation $(u, v) \mapsto \mathtt{(}u\mathtt{+}v\mathtt{)}\colon \Sigma^* \times \Sigma^* \to \Sigma^*$ auf Σ^*.

Beispiel 12
Seien Σ, A und B wie im letzten Beispiel. Die Relation R sei folgendermaßen definiert. Es gelte $(X, y) \in R$ genau dann, wenn eine der folgenden Aussagen gilt.

1. $X = \emptyset$ und $y = \mathtt{0}$
2. $X = \emptyset$ und $y = \mathtt{1}$
3. $X = \{u, v\}$ mit $u, v \in \Sigma^*$ und $y = \mathtt{(}u\mathtt{+}v\mathtt{)}$.

4. $X = \{u, v\}$ mit $u, v \in \Sigma^*$ und $y = \texttt{(}u\texttt{*}v\texttt{)}$.

Dann ist B abgeschlossen gegenüber der Relation R. Äquivalent zu dieser Aussage ist die folgende Aussage:

Die Menge B der voll geklammerten arithmetischen Ausdrücke über `0` und `1` enthält die Wörter `0` und `1` und ist abgeschlossen gegenüber den zweistelligen Operationen $(u, v) \mapsto \texttt{(}u\texttt{+}v\texttt{)}$ und $(u, v) \mapsto \texttt{(}u\texttt{*}v\texttt{)}$ auf Σ^*.

Satz 4 (induktive Definition) *Sei A eine Menge und sei $R \subseteq \mathfrak{P}(A) \times A$. Dann gibt es eine kleinste Menge $B \subseteq A$, die abgeschlossen ist gegenüber der Relation R.*

Man sagt, diese Menge sei die durch die Relation R *induktiv definierte Menge.*

Beweis des Satzes: Offenbar gibt es eine Menge B, die abgeschlossen ist gegenüber R, nämlich die Menge A. Daher existiert der Durchschnitt B_0 aller Mengen B, die abgeschlossen sind gegenüber R. Die Menge B_0 ist ebenfalls abgeschlossen gegenüber R, denn sei $(X, y) \in R$ mit $X \subseteq B_0$. Dann gilt für alle Mengen B, die abgeschlossen gegenüber R sind, nach Definition von B_0 die Aussage $B_0 \subseteq B$ und daher $X \subseteq B$. Da die Menge B abgeschlossen gegenüber R ist, gilt $y \in B$. Da dies für alle Mengen B, die abgeschlossen gegenüber R sind, gilt, folgt aus der Definition von B_0, dass $y \in B_0$ ist. Also ist auch B_0 abgeschlossen gegenüber R. Aus der Definition von B_0 folgt die Behauptung. q.e.d.

Beispiel 13
Sei R wie im letzten Beispiel definiert. Die durch die Relation R induktiv definierte Menge ist die Menge der voll geklammerten arithmetischen Ausdrücke über `0` und `1`. Die induktive Definition lässt sich auch so schreiben.

1. `0` ist ein voll geklammerter arithmetischer Ausdruck.
2. `1` ist ein voll geklammerter arithmetischer Ausdruck.
3. Wenn u und v voll geklammerte arithmetische Ausdrücke sind, dann ist auch $\texttt{(}u\texttt{+}v\texttt{)}$ ein voll geklammerter arithmetischer Ausdruck.
4. Wenn u und v voll geklammerte arithmetische Ausdrücke sind, dann ist auch $\texttt{(}u\texttt{*}v\texttt{)}$ ein voll geklammerter arithmetischer Ausdruck.

Man sagt, dass diese vier Regeln eine *induktive Definition des Begriffs* eines voll geklammerten arithmetischen Ausdrucks bilden. Ein Wort ist genau dann ein voll geklammerter arithmetischer Ausdruck, wenn es sich durch endlichmalige Anwendung der Regeln als ein solcher nachweisen lässt.

Wir können eine solche induktive Definition auch im Formalismus der Logik ausdrücken durch die folgende Formel F:

$$\begin{gathered} A(\texttt{0}) \wedge \\ A(\texttt{1}) \wedge \\ \forall u \forall v \big(A(u) \wedge A(v) \Longrightarrow A(\texttt{(}u\texttt{+}v\texttt{)}) \big) \wedge \\ \forall u \forall v \big(A(u) \wedge A(v) \Longrightarrow A(\texttt{(}u\texttt{*}v\texttt{)}) \big) \end{gathered}$$

Das Prädikat A ein voll geklammerter arithmetischer Ausdruck zu sein wird hierdurch induktiv definiert, d.h. A ist die kleinste Menge derart, dass die obige Aussage gilt. Ein Wort u über Σ ist genau dann ein voll geklammerter arithmetischer Ausdruck, wenn die Formel $A(u)$ logisch aus F und aus den Gesetzen, denen Wörter und ihre Konkatenationen gehorchen, folgt.

Aufgabe 1 Schreiben Sie die obige Formel F als definites logisches Programm (reines Prologprogramm). Das Programm soll von Zeichenreihen über Σ feststellen können, ob sie voll geklammerte arithmetische Ausdrücke sind oder nicht. Achten Sie dabei darauf, dass Sie den Begriff der Konkatenation von Wörtern in Ihrem Programm auch als geeignetes Prädikat definieren müssen. Testen Sie Ihr Programm für einige kurze Zeichenreihen über Σ.

Die induktiven Definitionen, die wir in dieser Vorlesung machen werden, werden alle von der Art sein, dass sie sich wie oben auf den Begriff der logischen Folgerung zurückführen lassen, und zwar sogar wie oben als definites logisches Programm formulieren lassen.

Die Definition durch vollständige Induktion stellt einen Spezialfall der induktiven Definition dar. Denn wenn eine Funktion h durch vollständige Induktion mittels der Rekursionsgleichungen (1.1) und (1.2) definiert ist, dann ist $\mathrm{Graph}(h)$ die kleinste Teilmenge von $\mathbb{N} \times A$, die das Paar $(0, a)$ als Element enthält und die abgeschlossen gegenüber der einstelligen Operation $(x, y) \mapsto (x', g(y, x))$ auf $\mathbb{N} \times A$ ist. Mit anderen Worten, die Menge $\mathrm{Graph}(h)$ ist gegeben durch die induktive Definition

1. $(0, a) \in \mathrm{Graph}(h)$
2. $(x, y) \in \mathrm{Graph}(h) \Longrightarrow (x', g(y, x)) \in \mathrm{Graph}(h)$.

Aufgabe 2 Finden Sie eine Relation R so, dass $\mathrm{Graph}(h)$ die durch R induktiv definierte Menge ist.

Entsprechend stellt die Wertverlaufsrekursion einen Spezialfall der induktiven Definition dar. Denn sei eine Funktion h durch Wertverlaufsrekursion mittels (1.3) definiert. Dann definieren wir eine Relation $R \subseteq \mathfrak{P}(\mathbb{N} \times A) \times (\mathbb{N} \times A)$ folgendermaßen. $(G, (n, a)) \in R$ gelte genau dann, wenn G der Graph einer Funktion $g\colon \{x \mid x \in \mathbb{N} \wedge x < n\} \to A$ ist und $a = F(g)$ ist. Dann ist die Menge $\mathrm{Graph}(h)$ die durch die Relation R induktiv definierte Menge.

Satz 5 (Induktion über eine induktive Definition) *Sei A eine Menge und sei $B \subseteq A$ induktiv definiert durch eine Relation $R \subseteq \mathfrak{P}(A) \times A$. Weiter sei P ein Prädikat auf A mit der folgenden Eigenschaft*

Induktionsschritt:

$$\bigwedge_{(X,y)\in R} \big(\bigwedge_{x\in X} P(x) \Longrightarrow P(y) \big).$$

Dann gilt $\bigwedge_{b\in B} P(b)$.

Im Induktionsschritt nennt man die Aussage $\bigwedge_{x\in X} P(x)$ die *Induktionsvoraussetzung* und die Aussage $P(y)$ die *Induktionsbehauptung*.

Beweis des Satzes: Die Menge P ist eine Teilmenge von A. Nach Definition von B ist B die kleinste Menge, die abgeschlossen gegenüber R ist. Aber nach dem Induktionsschritt ist auch die Menge P abgeschlossen gegenüber R. Daher ist $B \subseteq P$. Somit gilt $\bigwedge_{b\in B} P(b)$. q.e.d.

Beispiel 14
Jeder voll geklammerte arithmetische Ausdruck über `0` und `1` hat gleichviele öffnende wie schließende Klammern.

Beweis durch Induktion über die Definition des Begriffs eines voll geklammerten arithmetischen Ausdrucks über `0` und `1`: Das Prädikat P auf Σ^* sei folgendermaßen definiert. Für $u \in \Sigma^*$ gelte $P(u)$ genau dann, wenn u gleich viele öffnende wie schließende Klammern hat. Der zu führende Beweis benutzt Satz 5 mit dem Prädikat P.

1. `0` hat gleichviele öffende wie schließende Klammern, nämlich 0.

2. `1` hat gleichviele öffende wie schließende Klammern, nämlich 0.

3. Als Induktionsvoraussetzung nehmen wir an, dass u gleichviele öffnende wie schließende Klammern hat (sagen wir m) und v ebenfalls (sagen wir n). Dann hat `(`u`+`v`)` ebenfalls gleichviele öffnende wie schließende Klammern, nämlich $m + n + 1$.

4. Als Induktionsvoraussetzung nehmen wir an, dass u gleichviele öffnende wie schließende Klammern hat (sagen wir m) und v ebenfalls (sagen wir n). Dann hat `(`u`*`v`)` ebenfalls gleichviele öffnende wie schließende Klammern, nämlich $m + n + 1$.

q.e.d.

Beispiel 15
Die *Nachfolgerfunktion* $N\colon \mathbb{N} \to \mathbb{N}$ ist gegeben durch $N(n) = n' = n + 1$. Mit Hilfe der Nachfolgerfunktion lässt sich der Begriff der natürlichen Zahlen induktiv definieren.

Eine induktive Definition der natürlichen Zahlen:

1. 0 ist eine natürliche Zahl.

2. Wenn n eine natürliche Zahl ist, dann auch $N(n)$.

Ein Beweis durch Induktion über diese induktive Definition ist dann nichts anderes als ein Beweis durch vollständige Induktion.

1.5.4 Formale Systeme

Es ist schwer den Begriff des Formalen Systems exakt zu definieren, da verschiedene Leute darunter je nach ihrer Sichtweise leicht unterschiedliche Dinge verstehen.

Der Begriff taucht wohl zum ersten Mal auf im Zusammenhang mit mathematischen Logik. Der bekannteste Logiker des zwanzigsten Jahrhunderts Kurt Gödel schreibt dazu am Anfang seiner bahnbrechenden Arbeit

> Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I

aus dem Jahre 1931 das Folgende.

> Die Entwicklung der Mathematik in der Richtung zu größerer Exaktheit hat bekanntlich dazu geführt, daß weite Gebiete von ihr formalisiert wurden, in der Art, daß das Beweisen nach einigen wenigen mechanischen Regeln vollzogen werden kann. Die umfassendsten derzeit aufgestellten formalen Systeme sind das System der Principia mathematica (PM) einerseits, das Zermelo-Fraenkelsche (von J. von Neumann weiter ausgebildete) Axiomensystem der Mengenlehre andererseits. Diese beiden Systeme sind so weit, daß alle heute in der Mathematik angewendeten Beweismethoden in ihnen formalisiert, d. h. auf einige wenige Axiome und Schlußregeln zurückgeführt sind.

Diese damals ausschließlich verwendeten logischen formalen Systeme sind folgendermaßen aufgebaut. Logische Aussagen werden ausgedrückt in einer sogenannten formalen Sprache als Wörter über einer Menge von Zeichen. Die derart gebildeten sinnvollen Wörter heißen (logische)

Formeln. Es werden nun rein mechanische Regeln vorgegeben, nach denen Formeln erzeugt werden können oder aus bereits erzeugten Formeln neue Formeln erzeugt werden können. Wenn eine Formel in möglicherweise mehreren Schritten erzeugt wird, sagen wir auch, sie werde hergeleitet. Dies Systeme sind so gemacht, dass darin nur wahre Formeln hergeleitet werden können. So eignen sich diese Systeme zum Beweisen von Formeln. Ein Beispiel für eine solche Regel ist

$$F \vee \neg F,$$

d.h. wenn F eine Formel ist, dann kann die Formel $F \vee \neg F$ erzeugt (hergeleitet) werden. Eine solche Formel, die sozusagen in einem Schritt aus dem Nichts hergeleitet werden kann, nennt man auch Axiom. Ein weiteres Beispiel für eine Regel ist

$$\frac{F \quad F \to G}{G},$$

d.h. aus Formeln F und $F \to G$ kann die Formel G erzeugt werden. Wenn also F und $F \to G$ herleitbar sind, so ist auch G herleitbar. Man sieht hier also, dass die Herleitbarkeit sich induktiv definieren lässt.

Ein formales System von diesem Typ ist also gegeben durch eine Menge A (im Beispiel die Menge der Formeln) und eine Relation $R \subseteq \mathfrak{P}(A) \times A$ (im Beispiel gilt $(X, y) \in R$ genau dann, wenn X eine Menge von Formeln ist und das System eine Regel enthält, die aus den Formeln in X die Formel y zu erzeugen gestattet). Dann ist die Menge der herleitbaren Elemente von A genau die durch R induktiv definierte Menge.

Unter einer Herleitung in einem solchen formalen System verstehen wir eine endliche oder unendliche Folge $(x_1, x_2, \dots)$ von Elementen von A derart, dass für jedes Element x_k der Folge eine Menge $X \subseteq \{x_1, \dots, x_{k-1}\}$ existiert mit $(X, x_k) \in R$. Wenn x letztes Element der Herleitung ist, sprechen wir auch von einer *Herleitung von* x. Wenn R die Eigenschaft besitzt, dass X endlich ist für alle $(X, y) \in R$, dann ist ein $x \in A$ genau dann herleitbar, wenn es eine Herleitung besitzt. Wir werden hier praktische ausschließlich solche Systeme von diesem Typ betrachten, wo dies der Fall ist und wo sich für alle $X \subseteq A$ und $y \in A$ explizit feststellen lässt, ob $(X, y) \in R$.

Es gibt auch noch andere rein formale Systeme. Wir werden hier praktisch ausschließlich solche Systeme betrachten, die wie oben Aktionen nach rein mechanischen Regeln durchzuführen gestatten, ohne aber notwendigerweise etwas mit dem Begriff der Wahrheit einer logischen oder mathematischen Aussage zu tun zu haben. Die formalen Systeme, die wir hier betrachten werden, lassen sich praktisch alle als Systeme vom obigen Typus formulieren.

Systeme, bei denen $(X, y) \in R$ existiert mit einer unendlichen Menge R werden meist als halbformale Systeme bezeichnet, so etwa Logiksysteme, die die ω-Regel

$$\frac{F[0] \quad F[1] \quad F[2] \quad \dots}{\forall x F[x]}$$

enthalten. Diese Regel hat unendlich viele Prämissen.

Andere Systeme, die wir hier nicht als rein formale Systeme betrachten, sind Orakelmaschinen.

Kapitel 2

Berechenbarkeitstheorie

2.1 Berechenbarkeit und Entscheidbarkeit

In der *Berechenbarkeitstheorie* oder *Rekursionstheorie* befasst man sich mit den Begriffen der Berechenbarkeit und der Entscheidbarkeit.

- Der Begriff der *Berechenbarkeit* bezieht sich auf Funktionen. Grob gesagt ist eine Funktion f *berechenbar*, wenn man zu einem gegebenen Argumentwert x den Funktionswert $f(x)$ explizit ausrechnen kann.
- Der Begriff der *Entscheidbarkeit* bezieht sich auf Prädikate. Grob gesagt ist ein Prädikat P *entscheidbar*, wenn man zu einem gegebenen Argumentwert x feststellen kann, ob $P(x)$ gilt oder nicht.

Statt x können auch mehrere Argumentwerte auftreten (mehrstellige Funktionen bzw. Prädikate).

Die Begriffe der Berechenbarkeit und der Entscheidbarkeit hängen eng miteinander zusammen. Denn wenn man z.B. $2+3$ berechnen kann, dann kann man auch entscheiden, ob $2+3=6$ ist. Umgekehrt: wenn man für jede natürliche Zahl n entscheiden kann, ob $2+3=n$ ist, dann kann man auch $2+3$ berechnen, indem man für $n=0,1,2\ldots$ jeweils testet, ob $2+3=n$ ist.

2.1.1 Entscheidbarkeit

Es wurden verschiedene Begriffe der Entscheidbarkeit definiert, die aber alle miteinander zusammenhängen. Gödel schreibt in der oben angeführten Arbeit weiter:

> Es liegt daher die Vermutung nahe, daß diese Axiome und Schlußregeln dazu ausreichen, *alle* mathematischen Fragen, die sich in den betreffenden Systemen überhaupt formal ausdücken lassen, auch zu entscheiden. Im folgenden wird gezeigt, daß dies nicht der Fall ist, sondern daß es in den beiden angeführten Systemen sogar relativ einfache Probleme aus der Theorie der gewöhnlichen Zahlen gibt, die sich aus den Axomen nicht entscheiden lassen.

Mehrere verschiedene Entscheidbarkeitsbegriffe

- Ursprünglich bedeutete Entscheidbarkeit einer Aussage F, dass man entweder beweisen kann, dass F gilt oder beweisen kann, dass $\neg F$ gilt.

- Kurt Gödel hat 1931 den Begriff der *Definierbarkeit* oder *Entscheidungsdefinitheit* einer Relation R in einem formalen logischen System eingeführt. Darunter ist grob gesagt zu verstehen, dass es eine Formel $F[x_1, \ldots, x_n]$ gibt, sodass für alle $r_1, \ldots, r_n$ die Formel $F[r_1, \ldots, r_n]$ in diesem logischen System beweisbar ist, wenn $R(r_1, \ldots, r_n)$ gilt und die Negation $\neg F[r_1, \ldots, r_n]$ in diesem logischen System beweisbar ist, wenn $R(r_1, \ldots, r_n)$ nicht gilt.
- Unter *Entscheidbarkeit* einer Relation R schlechthin ist demnach die Existenz eines solchen logischen Systems zu verstehen.
- Heute versteht man in der Informatik und in der Rekursionstheorie unter *Entscheidbarkeit* von R die Existenz eines Algorithmus, der entscheiden kann, ob $R(x_1, \ldots, x_n)$ gilt.

Algorithmus: 'mechanisches' Verfahren, das zu einer Eingabe $x \in A$ eine Ausgabe $y \in B$ zu berechnen versucht. Die Menge A der möglichen Eingaben und die Menge B der möglichen Ausgaben kann sein

- Eine Zahlenmenge (z.B. $\mathbb{N}$ oder $\mathbb{Z}$)
- Die Menge der Wörter (oder Zeichenketten, englisch 'strings')
- Bilder, Filme, Audioaufnahmen, etc.
- Zusammengesetzte Datenstrukturen (z.B. $\mathbb{N} \times \mathbb{N}$).

Jede mögliche Ein- oder Ausgabe lässt sich im Prinzip als Wort über einem geeigneten Zeichensatz darstellen. Ebenso lässt sie sich, wie wir später (bei den Gödel-Nummerierungen) sehen werden, durch eine natürliche Zahl darstellen. Der Berechenbarkeitsbegriff lässt sich daher, wie wir sehen werden, unabhängig vom Datentyp formulieren und dann auf die anderen Datentypen übertragen.

Definition 10 Eine Funktion $f : A \to B$ heißt *berechenbar*, wenn es einen Algorithmus gibt, der bei Eingabe von $x \in A$ stets die Ausgabe $f(x)$ liefert. Man sagt dann, die Funktion f werde durch diesen Algorithmus *berechnet*.

Die Theorie der Berechenbarkeit befasst sich mit der Frage, welche Funktionen berechenbar sind.

Systeme zur Beschreibung des Berechenbarkeitsbegriffes
Um den Begriff der berechenbaren Funktion formal zu fassen, wurden, vor allem in der ersten Hälfte des 20. Jahrhunderts, verschiedene Formalismen entwickelt, darunter die folgenden.

- μ-partiellrekursive Funktionen
- Logikkalküle
- Chomsky-Grammatiken
- Turing-Maschinen
- Kalküle
- λ-Kalül
- Kombinatoren

⋮

- Verschiedene Programmiersprachen

Jeder dieser Formalismen definiert eine Klasse von berechenbaren Funktionen und gibt an, wie diese Funktionen zu berechnen sind. Wir werden in dieser Vorlesung einige dieser Formalismen kennen lernen.

Es hat sich herausgestellt, dass all diese Formalismen die gleiche Menge von berechenbaren Funktionen beschreiben. Da man trotz aller Anstrengungen bisher keine Prinzipien zur Formulierung von Algorithmen gefunden hat, die sich nicht in jedem dieser Formalismen darstellen ließen, liegt die folgende These nahe.

Churchsche These Jede berechenbare Funktion ist in jedem dieser Formalismen berechnenbar.

2.2 μ-rekursive Funktionen

Dieser Abschnitt befasst sich mit den berechenbaren n-stelligen zahlentheoretischen Funktion, d.h. Funktionen $f\colon \mathbb{N}^n \to \mathbb{N}$. Es wird eine Klasse von zahlentheoretischen Funktionen — die Klasse der *μ-rekursiven Funktionen* — formal eingeführt. Es hat sich herausgestellt, dass die μ-rekursiven Funktionen genau diejenigen zahlentheoretischen Funktionen sind, die sich mit den bis heute bekannten Methoden berechnen lassen. Nach der Churchschen These sind die μ-rekursiven Funktionen genau die berechenbaren zahlentheoretischen Funktionen.

2.2.1 Zahlentheoretische Funktionen

Die *Zahlentheorie* ist diejenige mathematische Disziplin, die sich mit den *natürlichen* Zahlen befasst. In der Zahlentheorie sind die einzigen Grundobjekte die Zahl 0 und die Nachfolgerfunktion N, die jeder natürlichen Zahl a eine natürliche Zahl $N(a) = a' = a + 1$ zuordnet. Die Eigenschaften dieser zwei Grundobjekte werden durch die *Peano-Axiome* beschrieben.

Peano-Axiome

1. 0 ist eine natürliche Zahl.
2. Jeder natürlichen Zahl n ist eine und nur eine natürliche Zahl n' zugeordnet, die der *Nachfolger* von n genannt wird.
3. 0 ist kein Nachfolger.
4. Für natürliche Zahlen m und n gilt $m \neq n \Longrightarrow m' \neq n'$.
5. Enthält eine Menge P von natürlichen Zahlen die Zahl 0 und folgt aus $n \in P$ stets $n' \in P$, so besteht P aus allen natürlichen Zahlen.

Das fünfte Peano-Axiom ist das Prinzip der vollständigen Induktion, das wir schon kennen. Mit Hilfe der den Peano-Axiomen zugrunde liegenden Begriffe der Null und der Nachfolgerfunktion $N = n \mapsto n'$ lassen sich die komplexeren Begriffe aus der Zahlentheorie definieren. So ist z.B. die 3 definiert als $N(N(N(0)))$ und die Addition ist definiert durch die Rekursionsgleichungen

$$x + 0 := x$$
$$x + N(y) := N(x + y).$$

Durch Induktion nach y zeigt man, dass $x + y$ dadurch eindeutig definiert ist.

Wir werden die Begriffe der Null und der Nachfolgerfunktion benutzen, um nicht nur die natürlichen Zahlen formal zu beschreiben, sondern auch die Klasse der μ-rekursiven Funktionen auf den natürlichen Zahlen, die sich genau als die Klasse der — zumindest mit den bis heute bekannten Methoden — berechenbaren zahlentheoretischen Funktionen herausgestellt hat.

Definition 11 Eine *zahlentheoretische Funktion* ist eine Funktion $f\colon \mathbb{N}^n \to \mathbb{N}$, wobei n eine natürliche Zahl ist. Die Zahl n wird *Stelligkeit* von f genannt und man sagt, f sei eine *n-stellige* zahlentheoretische Funktion.

Wenn wir in diesem Abschnitt von n-stelligen Funktionen sprechen, meinen wir immer zahlentheoretische Funktionen von $\mathbb{N}^n$ in $\mathbb{N}$. Für $n \in \mathbb{N}$ werden wir das 1-Tupel (n) mit der Zahl n identifizieren, also auch $\mathbb{N}^1$ mit $\mathbb{N}$.

Beispiele für 1-stellige zahlentheoretische Funktionen sind die *Verdopplungsfunktion* $x \mapsto 2x\colon \mathbb{N} \to \mathbb{N}$, die *Quadratfunktion* $x \mapsto x^2\colon \mathbb{N} \to \mathbb{N}$, die *Fakultätsfunktion* $\mathrm{fak} = x \mapsto x!\colon \mathbb{N} \to \mathbb{N}$ und die *Vorgängerfunktion* $V\colon \mathbb{N} \to \mathbb{N}$ mit

$$V(x) = \begin{cases} x - 1, & \text{wenn } x > 0 \\ 0, & \text{wenn } x = 0. \end{cases}$$

Beispiele für 2-stellige zahlentheoretische Funktionen sind die *Addition* $\mathrm{add}\colon \mathbb{N}^2 \to \mathbb{N}$ oder $+\colon \mathbb{N}^2 \to \mathbb{N}$, die *Multiplikation* $\mathrm{mult}\colon \mathbb{N}^2 \to \mathbb{N}$ oder $(x, y) \mapsto xy\colon \mathbb{N}^2 \to \mathbb{N}$, die *Potenzierung* $\mathrm{pot}\colon \mathbb{N}^2 \to \mathbb{N}$ oder $(x, y) \mapsto x^y\colon \mathbb{N}^2 \to \mathbb{N}$ und die *modifizierte Differenz* $\mathrm{diff}\colon \mathbb{N}^2 \to \mathbb{N}$ oder $\dot{-}\colon \mathbb{N}^2 \to \mathbb{N}$ mit

$$\begin{aligned} \mathrm{add}(a, b) &= a + b \\ \mathrm{mult}(a, b) &= ab \\ \mathrm{pot}(a, b) &= a^b \\ \mathrm{diff}(a, b) = a \dot{-} b &= \begin{cases} a - b, & \text{wenn } a \geq b \\ 0 & \text{sonst.} \end{cases} \end{aligned}$$

Beispiele für n-stellige zahlentheoretische Funktionen sind die *Konstantenfunktionen* $K^n_c\colon \mathbb{N}^n \to \mathbb{N}$ mit

$$K^n_c(x_1, \ldots, x_n) := c.$$

Die Stelligkeit einer Funktion kann auch 0 sein. Eine nullstellige zahlentheoretische Funktion ist eine Funktion $f\colon \mathbb{N}^0 \to \mathbb{N}$. Eine solche Funktion ist eindeutig bestimmt durch ihren Wert $f()$, der eine natürliche Zahl c ist. Dann ist $f = K^0_c$.

Wir werden Elemente von $\mathbb{N}^n$, also Tupel von natürlichen Zahlen auch mit kleinen deutschen Buchstaben $\mathfrak{a}$, $\mathfrak{b}$, $\mathfrak{c}$, …, $\mathfrak{x}$, $\mathfrak{y}$, $\mathfrak{z}$, … bezeichnen. Wir werden mit der Notation sehr liberal umgehen und etwa, wenn $\mathfrak{x} = (x_1, \ldots, x_m)$ und $\mathfrak{z} = (z_1, \ldots, z_n)$ ist, für $f(a, x_1, \ldots, x_m, b, c, z_1, \ldots, z_n)$ einfach schreiben $f(a, \mathfrak{x}, b, c, \mathfrak{z})$.

2.2.2 Primitivrekursive Funktionen

In diesem Abschnitt wird eine Klasse von berechenbaren zahlentheoretischen Funktionen – die Klasse der *primitivrekursiven Funktionen* – eingeführt.

Primitivrekursive Funktionale

Dem Formalismus, der hier definiert wird, liegt die Menge Σ zugrunde, die aus den folgenden Zeichen besteht:

1. $\mathtt{O}$
2. $\mathtt{N}$
3. $\mathtt{P}_i^n$ für alle $i, n \in \mathbb{N}$ mit $1 \leq i \leq n$
4. $\mathtt{(}$
5. $\mathtt{)}$
6. $\mathtt{R}$

Definition 12 (Primitivrekursive Funktionale) *Primitivrekursive Funktionale* sind spezielle Wörter über der Zeichenmenge Σ. Jedes primitivrekursive Funktional ϕ hat eine *Stelligkeit* n. Diese ist eine natürliche Zahl und wir sagen dann, das Funktional ϕ sei *n-stellig*. Diese Begriffe sind induktiv folgendermaßen definiert.

1. $\mathtt{O}$ ist ein 0-stelliges primitivrekursives Funktional.
   $\mathtt{O}$ heißt das *Nullfunktional*.
2. $\mathtt{N}$ ist ein 1-stelliges primitivrekursives Funktional.
   $\mathtt{N}$ heißt das *Nachfolgerfunktional*.
3. Für $i, n \in \mathbb{N}$ mit $1 \leq i \leq n$ ist $\mathtt{P}_i^n$ ein n-stelliges primitivrekursives Funktional.
   $\mathtt{P}_i^n$ heißt das i-te n-stellige *Projektionsfunktional*.
4. Ist ϕ ein m-stelliges primitivrekursives Funktional und sind $\psi_1, \ldots, \psi_m$ n-stellige primitivrekursive Funktionale, so ist $\mathtt{(}\phi\psi_1 \ldots \psi_m\mathtt{)}$ ein n-stelliges primitivrekursives Funktional.
   $\mathtt{(}\phi\psi_1 \ldots \psi_m\mathtt{)}$ heißt die *Komposition* von ϕ mit $\psi_1, \ldots, \psi_m$.
5. Ist ϕ ein n-stelliges primitivrekursives Funktional und ist ψ ein $(n+2)$-stelliges primitivrekursives Funktional, so ist $\mathtt{(R}\phi\psi\mathtt{)}$ ein $(n+1)$-stelliges primitivrekursives Funktional.
   $\mathtt{(R}\phi\psi\mathtt{)}$ heißt das durch *primitive Rekursion* aus ϕ und ψ gebildete Funktional.

Definition 13 (Wert eines p.r. Funktionals) Wir definieren induktiv, was es heißt, dass ein n-stelliges primitivrekursives Funktional ϕ an einer Stelle $\mathfrak{x} \in \mathbb{N}^n$ den *Wert* $y \in \mathbb{N}$ hat, in Zeichen $\mathfrak{x}\ W[\phi]\ y$. Wir definieren also induktiv eine zweistellige Relation $W[\phi] \subseteq \mathbb{N}^n \times \mathbb{N}$ für jedes n-stellige primitivrekursive Funktional ϕ.

1. Es gelte $()\ W[\mathtt{O}]\ 0$.
2. Wenn $x \in \mathbb{N}$, dann gelte $(x)\ W[\mathtt{N}]\ x'$.
3. Wenn $i, n \in \mathbb{N}$ mit $1 \leq i \leq n$ und $x_1, \ldots, x_n \in \mathbb{N}$, dann $(x_1, \ldots, x_n)\ W[\mathtt{P}_i^n]\ x_i$.
4. Wenn ϕ ein m-stelliges primitivrekursives Funktional ist, $\psi_1, \ldots, \psi_m$ n-stellige primitivrekursive Funktionale sind, $\mathfrak{x}\ W[\psi_1]\ y_1$, $\ldots$, $\mathfrak{x}\ W[\psi_m]\ y_m$ gelten und $(y_1, \ldots, y_m)\ W[\phi]\ z$ gilt, dann gelte $\mathfrak{x}\ W[\mathtt{(}\phi\psi_1 \ldots \psi_m\mathtt{)}]\ z$.
5. Wenn ϕ ein n-stelliges primitivrekursives Funktional ist, ψ ein $(n+2)$-stelliges primitivrekursives Funktional ist und $\mathfrak{x}\ W[\phi]\ z$, dann gelte $(\mathfrak{x}, 0)\ W[\mathtt{(R}\phi\psi\mathtt{)}]\ z$.

6. Wenn ϕ ein n-stelliges primitivrekursives Funktional ist, ψ ein $(n+2)$-stelliges primitivrekursives Funktional ist, $(\mathfrak{x}, y)\ W[\texttt{(R}\phi\psi\texttt{)}]\ u$ gilt und $(u, \mathfrak{x}, y)\ W[\psi]\ z$ gilt, dann gelte $(\mathfrak{x}, y')\ W[\texttt{(R}\phi\psi\texttt{)}]\ z$.

Die so definierte Relation $W[\phi] \subseteq \mathbb{N}^n \times \mathbb{N}$ ist rechtseindeutig, d.h. es gilt $\mathfrak{x}\ W[\phi]\ y \wedge \mathfrak{x}\ W[\phi]\ z \Longrightarrow y = z$.

Aufgabe 3 Zeigen Sie dies durch Wertverlaufsinduktion nach der Länge von ϕ, mit geschachtelter vollständiger Induktion nach der letzten Komponente von $\mathfrak{x}$ im Falle der primitiven Rekursion.

Die Relation $W[\phi] \subseteq \mathbb{N}^n \times \mathbb{N}$ ist linkstotal, d.h. für jedes $\mathfrak{x} \in \mathbb{N}^n$ existiert ein $y \in \mathbb{N}$ mit $\mathfrak{x}\ W[\phi]\ y$.

Aufgabe 4 Zeigen Sie dies durch Wertverlaufsinduktion nach der Länge von ϕ, mit geschachtelter vollständiger Induktion nach der letzten Komponente von $\mathfrak{x}$ im Falle der primitiven Rekursion.

Es folgt, dass die Relation $W[\phi]$ der Graph einer Funktion von $\mathbb{N}^n$ nach $\mathbb{N}$, also einer n-stelligen zahlentheoretischen Funktion ist.

Definition 14 (primitivrekursive Funktionen) Wenn ϕ ein n-stelliges primitivrekursives Funktional ist, dann nennt man die n-stellige Funktion $f\colon \mathbb{N}^n \to \mathbb{N}$ mit $\mathrm{Graph}(f) = W[\phi]$, deren Graph die oben definierte Relation $W[\phi]$ ist, die durch das primitivrekursive Funktional ϕ *bezeichnete* Funktion. Eine zahlentheoretische Funktion $f\colon \mathbb{N}^n \to \mathbb{N}$ heißt *primitivrekursiv*, wenn sie durch ein primitivrekursives Funktional bezeichnet wird.

Beispiel 16
Die durch `N` bezeichnete Funktion ist die Nachfolgerfunktion $N\colon \mathbb{N} \to \mathbb{N}$. Sei f die durch das Funktional `(NN)` bezeichnete Funktion. Dann ist f eine 1-stellige primitivrekursive Funktion mit $f(a) = N(N(a)) = a + 2$ für alle $n \in \mathbb{N}$.

Wir werden generell die durch ein Funktional ϕ bezeichnete Funktion f folgendermaßen schreiben. Das Wort ϕ besteht aus Zeichen im `Schreibmaschinenzeichensatz`. Wir schreiben nun die Funktion f, indem wir die gleichen Zeichen im *Mathematikzeichensatz* hinschreiben.
Es gilt also

Definition 15

1. Die 0-stellige Funktion O ist definiert durch
$$O() := 0.$$
Sie heißt die *Nullfunktion*.

2. Die 1-stellige Funktion N ist definiert durch
$$N(x) := x'.$$
Sie heißt die *Nachfolgerfunktion*

3. Für natürliche Zahlen i und n mit $1 \leq i \leq n$ ist die n-stellige Funktion P_i^n definiert durch
$$P_i^n(x_1, \ldots, x_n) := x_i.$$
Sie heißt die i-te n-stellige *Projektionsfunktion*.

4. Ist f eine m-stellige Funktion mit $m \geq 1$ und sind $g_1, \ldots, g_m$ n-stellige Funktionen, so ist die n-stellige Funktion $(fg_1 \ldots g_m)$ definiert durch

$$(fg_1 \ldots g_m)(x_1, \ldots, x_n) := f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)).$$

   Sie heißt die *Komposition* von f mit $g_1, \ldots, g_m$.

5. Ist f eine n-stellige Funktion und g eine $(n+2)$-stellige Funktion, so ist die $(n+1)$-stellige Funktion (Rfg) definiert durch

$$\begin{aligned}(Rfg)(x_1, \ldots, x_n, 0) &:= f(x_1, \ldots, x_n)\\ (Rfg)(x_1, \ldots, x_n, y') &:= g((Rfg)(x_1, \ldots, x_n, y), x_1, \ldots, x_n, y).\end{aligned}$$

   Sie heißt die sich durch *primitive Rekursion* aus f und g ergebende Funktion.

Für 1.-4. ist trivial, dass die dort definierten Funktionen wohldefiniert sind. Für 5. zeigt man durch Induktion nach y, dass $(Rfg)(x_1, \ldots, x_n, y)$ wohldefiniert ist.

Aufgrund unserer abkürzenden Schreibweise kann man die Definitionen dieser Funktionen auch so schreiben:

$$\begin{aligned}O() &:= 0\\ N(x) &:= x'\\ P_i^n(x_1, \ldots, x_n) &:= x_i\\ (fg_1 \ldots g_m)(\mathfrak{x}) &:= f(g_1(\mathfrak{x}), \ldots, g_m(\mathfrak{x}))\\ (Rfg)(\mathfrak{x}, 0) &:= f(\mathfrak{x})\\ (Rfg)(\mathfrak{x}, y') &:= g((Rfg)(\mathfrak{x}, y), \mathfrak{x}, y).\end{aligned}$$

Die Menge der *primitivrekursiven Funktionen* ist die kleinste Menge von zahlentheoretischen Funktionen, die die Nullfunktion, die Nachfolgerfunktion und die Projektionsfunktionen enthält und abgeschlossen ist gegenüber Komposition und primitiver Rekursion.

Beispiel 17
Sei add $:= (RP_1^1(NP_1^3))$. Dann ist add eine 2-stellige primitivrekursive Funktion. Es ist

$$\begin{aligned}\mathrm{add}(x, 0) &= P_1^1(x)\\ &= x\end{aligned}$$

und

$$\begin{aligned}\mathrm{add}(x, y') &= (NP_1^3)((RP_1^1(NP_1^3))(x, y), x, y)\\ &= (NP_1^3)(\mathrm{add}(x, y), x, y)\\ &= N(P_1^3(\mathrm{add}(x, y), x, y))\\ &= N(\mathrm{add}(x, y)).\end{aligned}$$

Es folgt, dass add die Additionsfunktion + ist (Beweis von $\mathrm{add}(x, y) = x + y$ durch Induktion nach y).

$$\begin{aligned}K_0^0 &= O\\ K_{c'}^0 &= (NK_c^0)\end{aligned}$$

Beweis, dass K_c^0 primitivrekursiv ist: durch vollständige Induktion nach c.

$$K_c^1 = (RK_c^0 P_1^2), \qquad K_c^n = (K_c^1 P_1^n) \quad \text{für } n > 1.$$

Folgerung: K_c^n ist eine primitivrekursive Funktion.

Eine Rechtfertigung für die primitive Rekursion: Betrachten wir nun nochmals den Satz 1 zur Definition durch vollständige Induktion für den Fall, dass die dort erwähnte Menge A die Menge $\mathbb{N}$ der natürlichen Zahlen ist. Dann besagt Satz 1, dass es zu jedem $a \in \mathbb{N}$ und $g\colon \mathbb{N}^2 \to \mathbb{N}$ genau eine Funktion $h\colon \mathbb{N} \to \mathbb{N}$ gibt, die folgende Gleichungen erfüllt.

$$\begin{aligned} h(0) &= a \\ h(y') &= g(h(y), y). \end{aligned}$$

Diese Funktion h ist genau die Funktion $(RK_a^0 g)$. Der Satz 1 rechtfertigt also die Definition einer einstelligen zahlentheoretischen Funktion durch primitive Rekursion in dem Sinne, dass er garantiert, dass durch die primitive Rekursion eine eindeutig bestimmte Funktion definiert ist. Der Satz 1 liefert aber auch eine Rechtfertigung für die Definition einer mehrstelligen zahlentheoretischen Funktion durch primitive Rekursion. Hierzu seien $\hat{a}\colon \mathbb{N}^n \to \mathbb{N}$ und $\hat{g}\colon \mathbb{N}^{n+2} \to \mathbb{N}$ zwei zahlentheoretischen Funktionen. Für beliebige natürliche Zahlen $x_1, \ldots, x_n \in \mathbb{N}$ definieren wir eine natürliche Zahl $a_{x_1 \ldots x_n} \in \mathbb{N}$ und eine zweistellige zahlentheoretische Funktion $g_{x_1 \ldots x_n}\colon \mathbb{N}^2 \to \mathbb{N}$ durch

$$\begin{aligned} a_{x_1 \ldots x_n} &:= \hat{a}(x_1, \ldots, x_n) \\ g_{x_1 \ldots x_n}(w, y) &:= \hat{g}(w, x_1, \ldots, x_n, y). \end{aligned}$$

Nach Satz 1 gibt es dann genau eine Funktion $h_{x_1 \ldots x_n}\colon \mathbb{N} \to \mathbb{N}$, die folgende Gleichungen erfüllt.

$$\begin{aligned} h_{x_1 \ldots x_n}(0) &= a_{x_1 \ldots x_n} \\ h_{x_1 \ldots x_n}(y') &= g_{x_1 \ldots x_n}(h_{x_1 \ldots x_n}(y), y). \end{aligned}$$

Da dies für alle $x_1, \ldots, x_n \in \mathbb{N}$ gilt, gibt es genau eine Funktion $\hat{h}\colon \mathbb{N}^{n+1} \to \mathbb{N}^n$, die die folgenden Gleichungen erfüllt:

$$\begin{aligned} \hat{h}(x_1, \ldots, x_n, 0) &= \hat{a}(x_1, \ldots, x_n) \\ \hat{h}(x_1, \ldots, x_n, y') &= \hat{g}(\hat{h}(x_1, \ldots, x_n, y), x_1, \ldots, x_n, y), \end{aligned}$$

und für diese Funktion $\hat{h}$ gilt

$$\hat{h}(x_1, \ldots, x_n, y) = h_{x_1 \ldots x_n}(y).$$

Die Funktion $\hat{h}$ ist genau die Funktion $(R\hat{a}\hat{g})$, die sich aus den Funktionen $\hat{a}$ und $\hat{g}$ durch primitive Rekursion ergibt.

Explizite Transformation: Der Begriff des *primitivrekursiven Terms* über Variablen $x_1, \ldots, x_n$ sei folgendermaßen induktiv definiert. Ein primitivrekursiver Term ist ein Wort über dem Alphabet, das aus den Variablen $x_1, \ldots, x_n$, aus den Ziffern $0, \ldots, 9$, aus Symbolen $f_1, \ldots, f_r$ für primitivrekursive Funktionen und aus runden Klammern und Beistrich besteht. Und zwar

1. Die Variablen $x_1, \ldots, x_n$ sind primitivrekursive Terme.

2. Die Dezimaldarstellungen natürlicher Zahlen sind primitivrekursive Terme.

3. Wenn f ein Symbol für eine k-stellige primitivrekursive Funktion ist und $t_1, \ldots, t_k$ primitivrekursive Terme sind, dann ist das Wort $f(t_1, \ldots, t_k)$ ein primitivrekursiver Term.

Satz 6 (Explizite Transformation) *Eine n-stellige zahlentheoretische Funktion h sei folgendermaßen definiert.*

$$h(x_1, \ldots, x_n) := t.$$

Dabei seien $x_1, \ldots, x_n$ paarweise verschiedene Variablen und t sei ein primitivrekursiver Term über den Variablen $x_1, \ldots, x_n$. Dann ist h primitivrekursiv.

Beweis durch Induktion nach t über die induktive Definition des Begriffs eines primitivrekursiven Terms:

1. Ist t eine Variable x_i, dann ist $h = P_i^n$ und damit ist h primitivrekursiv.

2. Ist t die Dezimaldarstellung einer natürlichen Zahl c, dann ist $h = K_c^n$ und damit primitivrekursiv.

3. Hat t die Form $f(t_1, \ldots, t_k)$, wobei f ein Symbol für eine k-stellige primitivrekursive Funktion ist und $t_1, \ldots, t_k$ primitivrekursive Terme sind, dann sind nach Induktionsvoraussetzung die durch $h_i(x_1, \ldots, x_n) := t_i$ definierten n-stelligen zahlentheoretischen Funktionen $h_1, \ldots, h_k$ primitivrekursiv. Weiter ist dann $h = (f h_1 \ldots h_k)$ und damit ist h primitivrekursiv.

q.e.d.

Beispiel 18
Sei f eine 5-stellige primitivrekursive Funktion und sei

$$h(x, y, z) := f(z, x, 5, z, z).$$

Dann ist h primitivrekursiv, da

$$h = (f P_3^3 P_1^3 K_5^3 P_3^3 P_3^3).$$

Beispiel 19
Seien f und g zwei zweistellige primitivrekursive Funktionen und sei

$$h(x, y) := f(g(5, x), f(y, y)).$$

Dann ist h primitivrekursiv, da

$$h = (f(g K_5^2 P_1^2)(f P_2^2 P_2^2)).$$

Die Multiplikationsfunktion $\mathrm{mult}\colon \mathbb{N}^2 \to \mathbb{N}$, die Potenzfunktion $\mathrm{pot}\colon \mathbb{N}^2 \to \mathbb{N}$, die Vorgängerfunktion $V\colon \mathbb{N} \to \mathbb{N}$, die modifizierte Differenz $\mathrm{diff}\colon \mathbb{N}^2 \to \mathbb{N}$, die Signumfunktion $\mathrm{sg}\colon \mathbb{N} \to \mathbb{N}$ mit

$$\mathrm{sg}(x) := \begin{cases} 0, & \text{wenn } x = 0 \\ 1 & \text{sonst} \end{cases}$$

und die Funktion $\overline{\mathrm{sg}}\colon \mathbb{N} \to \mathbb{N}$ mit

$$\overline{\mathrm{sg}}(x) := \begin{cases} 1, & \text{wenn } x = 0 \\ 0 & \text{sonst} \end{cases}$$

sind primitivrekursive Funktionen, weil gilt

$$\begin{aligned}\text{mult} &= (RK_0^1(+P_1^3P_2^3))\\ \text{pot} &= (RK_1^1(*P_1^3P_2^3))\\ V &= (ROP_2^2)\\ \text{diff} &= (RP_1^1(VP_1^3))\\ \text{sg} &= (ROK_1^2)\\ \overline{\text{sg}} &= (\text{diff}\, K_1^1P_1^1).\end{aligned}$$

Für eine n-stellige zahlentheoretische Funktion f sei

$$\begin{aligned}(\Sigma f) &= (Rg(+P_1^{n+2}h))\\ (\Pi f) &= (Rg(*P_1^{n+2}h)),\end{aligned}$$

wobei

$$\begin{aligned}g &= (fP_1^n \dots P_n^nK_0^n)\\ h &= (fP_2^{n+2} \dots P_{n+1}^{n+2}(NP_{n+2}^{n+2}))).\end{aligned}$$

Dann sind die Funktionen $(\Sigma f)\colon \mathbb{N}^{n+1} \to \mathbb{N}$ und $(\Pi f)\colon \mathbb{N}^{n+1} \to \mathbb{N}$ primitivrekursiv und gegeben durch

$$\begin{aligned}(\Sigma f)(\mathfrak{x}, y) &= \sum_{z=0}^{y} f(\mathfrak{x}, z)\\ (\Pi f)(\mathfrak{x}, y) &= \prod_{z=0}^{y} f(\mathfrak{x}, z)\end{aligned}$$

Aufgabe 5 Zeigen Sie, dass diese Gleichungen äquivalent zu obiger Definition von (Σf) und (Πf) sind.

Damit ist auch die Fakultätsfunktion $\text{fak}\colon \mathbb{N} \to \mathbb{N}$ primitivrekursiv, weil gilt

$$\text{fak} = (\Pi(+P_1^1\,\overline{\text{sg}})).$$

Methoden zum Nachweis, dass eine Funktion primitivrekursiv ist
Ist eine Funktion h durch vollständige Induktion nach ihrem letzten Argument definiert, macht man in der Regel einen Ansatz $h = (Rfg)$ und versucht die Funktionen f und g zu bestimmen. Zum Beispiel ist die Multiplikation durch vollständige Induktion nach dem zweiten Argument definiert durch die folgenden Rekursionsgleichungen.

$$\begin{aligned}\text{mult}(x, 0) &= 0\\ \text{mult}(x, y') &= \text{mult}(x, y) + x.\end{aligned}$$

Macht man den Ansatz $\text{mult} = (Rfg)$, so schreibt sich die Definition von (Rfg)

$$\begin{aligned}(Rfg)(x, 0) &= f(x)\\ (Rfg)(x, y') &= g((Rfg)(x, y), x, y)\end{aligned}$$

als

$$\begin{aligned}\text{mult}(x,0) &= f(x)\\ \text{mult}(x,y') &= g(\text{mult}(x,y),x,y).\end{aligned}$$

Man muss die Funktionen f und g so wählen, dass diese Gleichungen gelten. Umgekehrt folgt aus diesen Gleichungen, dass mult $= (Rfg)$ gilt, wie man durch vollständige Induktion nach dem zweiten Argument beweist. Die Gleichungen sind offenbar dann erfüllt, wenn man die Funktionen f und g folgendermaßen definiert.

$$\begin{aligned}f(x) &:= 0\\ g(p,x,y) &:= p + x.\end{aligned}$$

Im Beispiel weist man diese Funktionen f und g einfach durch eine explizite Transformation als primitivrekursiv nach:

$$\begin{aligned}f &= K_0^1\\ g &= (\text{add } P_1^3\, P_2^3).\end{aligned}$$

Also ist mult $= (Rfg) = (R\ K_0^1\ (\text{add } P_1^3\, P_2^3))$, wobei add $= (RP_1^1(NP_1^3))$ ist, wie wir schon wissen. Damit ist mult primitivrekursiv.

Ist eine Funktion h durch vollständige Induktion nicht nach dem letzten, sondern nach einem anderen Argument definiert, so hilft eine Vertauschung der Argumente, was ein Spezialfall einer expliziten Transformation ist. Ist zum Beispiel $h(x,y)$ durch vollständige Induktion nach x definiert, so definiert man eine zweistellige Funktion k durch $k(y,x) := h(x,y)$. Dann weist man wie oben nach, dass k primitivrekursiv ist. Da $h = (kP_2^2P_1^2)$ ist, ist deshalb auch h primitivrekursiv.

Frage: Ist jede zahlentheoretische Funktion primitivrekursiv?

Antwort: Nein, denn es gibt nur abzählbar viele primitivrekursive Funktionen, aber überabzählbar viele zahlentheoretische Funktionen.

Eine Menge A heißt *abzählbar*, wenn eine injektive Abbildung von A in $\mathbb{N}$ existiert. Es gilt: Eine Menge A ist genau dann abzählbar, wenn sie leer ist oder eine surjektive Abbildung $\theta\colon \mathbb{N} \to A$ existiert.

Beispiel 20
Die Menge $\mathbb{N}$ der natürlichen Zahlen ist abzählbar. Eine surjektive Abzählungsfunktion $\theta\colon \mathbb{N} \to \mathbb{N}$ ist etwa gegeben durch

$$\theta(n) := n.$$

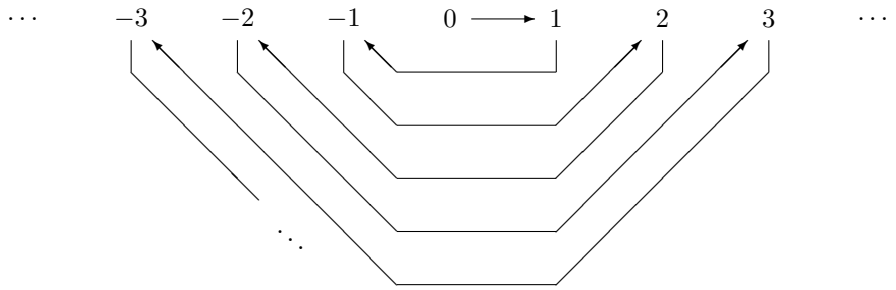
Man kann die Reihenfolge der Abzählung auch durch ein Bild veranschaulichen, indem man jeweils einen Pfeil von $\theta(0)$ nach $\theta(1)$, von $\theta(1)$ nach $\theta(2)$, von $\theta(2)$ nach $\theta(3)$, usw. zeichnet. Das schaut dann so aus.

$$0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow \cdots$$

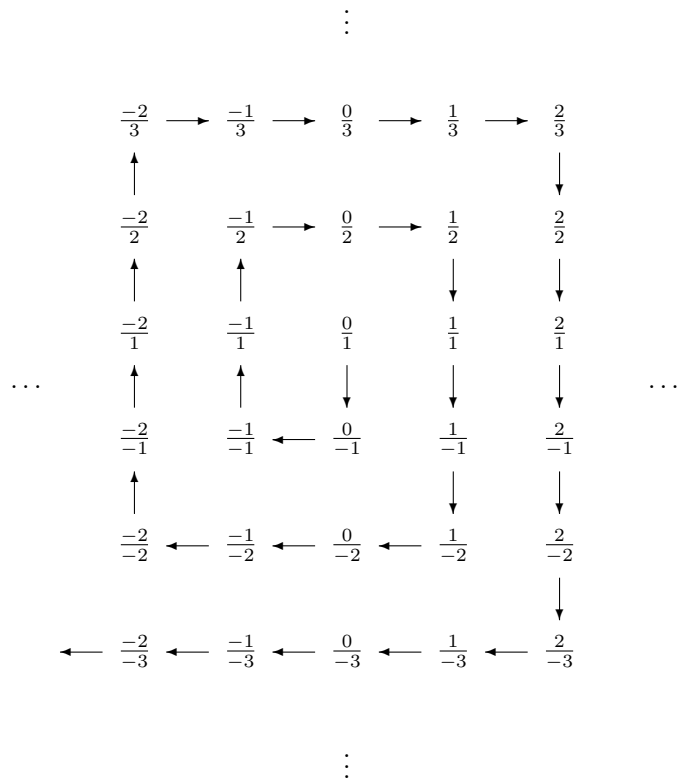
Beispiel 21
Die Menge $\mathbb{Z}$ der ganzen Zahlen ist abzählbar. Eine surjektive Abzählungsfunktion $\theta\colon \mathbb{N} \to \mathbb{Z}$ ist etwa gegeben durch

$$\theta(2n) := -n$$
$$\theta(2n+1) := n+1.$$

In bildlicher Darstellung schaut diese Abzählung so aus.



Beispiel 22
Die Menge $\mathbb{Q}$ der rationalen Zahlen ist abzählbar. Die folgende Abbildung zeigt eine Abzählung von $\mathbb{Q}$. Hierzu schreibt man jede rationale Zahl q als Quotient $\frac{x}{y}$ mit $x, y \in \mathbb{Z}$ und $y \neq 0$. Dann kann man die rationalen Zahlen zweidimensional wie folgt anornden (wobei jede rationale Zahl mehrfach vorkommt, was aber nicht stört).



Lemma 1 *Jede Teilmenge einer abzählbaren Menge ist wieder abzählbar.*

Satz 7 *Die Menge $\mathbb{R}$ der reellen Zahlen ist nicht abzählbar.*

Beweis (Cantor'sches Diagonalverfahren):
Wir führen den Beweis indirekt. Angenommen, $\mathbb{R}$ wäre abzählbar. Dann wäre das Intervall $[0,1) = \{x \in \mathbb{R} \mid 0 \leq x < 1\}$ als Teilmenge von $\mathbb{R}$ ebenfalls abzählbar. Es gäbe also eine surjektive Funktion $\theta\colon \mathbb{N} \to [0,1)$. Also $[0,1) = \{\theta(0), \theta(1), \theta(2), \dots\}$. Wir schreiben jedes $\theta(k)$ in Dezimalbruchdarstellung $0, a_{k0}a_{k1}a_{k2}\dots$ mit Ziffern $a_{kj} \in \{0, \dots, 9\}$. Schreibt man alle diese Dezimalbrüche untereinander, so erhält man ein Schema

$$\begin{aligned}
&0, a_{00}a_{01}a_{02}a_{03}\dots\\
&0, a_{10}a_{11}a_{12}a_{13}\dots\\
&0, a_{20}a_{21}a_{22}a_{23}\dots\\
&0, a_{30}a_{31}a_{32}a_{33}\dots\\
&\vdots
\end{aligned}$$

Betrachten wir nun die Diagonale $a_{00}a_{11}a_{22}a_{33}\dots$ der Nachkommastellen. Ändert man jede Ziffer dieser Folge ab, etwa indem man jedes Vorkommen der Ziffer 6 in dieser Folge durch eine 3 ersetzt und alle anderen Ziffernvorkommen dieser Folge durch eine 6, so erhält man eine neue Folge von Ziffern $b_0b_1b_2b_3\dots$ mit

$$b_k := \begin{cases} 3, & \text{falls } a_{kk} = 6 \\ 6, & \text{falls } a_{kk} \neq 6, \end{cases}$$

also mit

$$b_k \neq a_{kk} \qquad \text{für alle } k \in \mathbb{N}.$$

Folglich kann der Dezimalbruch $0, b_0b_1b_2b_3\dots$ nicht identisch sein mit einem der Dezimalbrüche $0, a_{n0}a_{n1}a_{n2}a_{n3}\dots$. Andernfalls müsste nämlich $b_n = a_{nn}$ sein. Sei nun x die reelle Zahl mit der Dezimalbruchdarstellung $0, b_0b_1b_2b_3\dots$. Dann kann x folglich nicht in der Menge $\{\theta(0), \theta(1), \theta(2), \dots\}$ vorkommen, da die Dezimalbruchdarstellung von x eindeutig ist (eine reelle Zahl mit einer Dezimalbruchdarstellung, die keine Ziffern 0 oder 9 enthält, hat stets eine eindeutige — d.h. nur eine — Dezimalbruchdarstellung). Dies steht im Widerspruch zur Surjektivität der Funktion θ. q.e.d.

Die Konstruktion soll an einem Beispiel demonstriert werden. Nehmen wir an, die Zahlen $\theta(0), \theta(1), \theta(2), \dots$ seien

$$\begin{aligned}
&0,6589732\dots\\
&0,1092845\dots\\
&0,3211691\dots\\
&0,2448123\dots\\
&0,5677980\dots\\
&0,4482761\dots\\
&0,5001799\dots\\
&\vdots
\end{aligned}$$

Dann lautet die Diagonale der Nachkommastellen 6018969 ... . Verändert man hierin die Ziffern 6 zu 3 und alle übrigen Ziffern zu 6, so erhält man 3666636 ... . Die Zahl $x = 0,3666636\ldots$ kommt unter den obigen Zahlen $\theta(0), \theta(1), \ldots$ nicht vor. Also haben wir eine reelle Zahl gefunden, die durch die angenommene Abzählung aller reller Zahlen nicht abgezählt wird, was ein Widerspruch zur Annahme bedeutet.

Satz 8 *Die Menge $\mathbb{N}^{\mathbb{N}}$ der Funktionen $f\colon \mathbb{N} \to \mathbb{N}$ ist nicht abzählbar.*

Beweis (Cantor'sches Diagonalverfahren):
$\mathbb{N}^{\mathbb{N}}$ ist nicht leer, da $x \mapsto 0\colon \mathbb{N} \to \mathbb{N} \in \mathbb{N}^{\mathbb{N}}$ ist.
Angenommen es gäbe eine surjektive Abbildung

$$\theta = k \mapsto f_k\colon \mathbb{N} \to \mathbb{N}^{\mathbb{N}}.$$

Dann sei $f\colon \mathbb{N} \to \mathbb{N}$ definiert durch

$$\bigwedge_{k\in\mathbb{N}} f(k) := f_k(k) + 1.$$

Dann ist $f \in \mathbb{N}^{\mathbb{N}}$. Wegen der Surjektivität von θ existiert ein $n \in \mathbb{N}$ mit $f = f_n$. Dann gilt

$$f(n) = f_n(n) + 1 = f(n) + 1.$$

Widerspruch. q.e.d.

Folgerung: Für jedes $n > 0$ ist die Menge $\mathbb{N}^{\mathbb{N}^n}$ der n-stelligen zahlentheoretischen Funktionen nicht abzählbar. Entsprechend sind für ein nichtleeres Alphabet Σ die Menge ${\Sigma^*}^{\Sigma^*}$ und für $n > 0$ die Menge ${\Sigma^*}^{{\Sigma^*}^n}$ nicht abzählbar.

Abzählbarkeit der Menge der primitivrekursiven Funktionale

Satz 9 *Sei Σ ein Alphabet. Dann ist Σ^* abzählbar.*

Beweis: Sei n die Anzahl der Zeichen in Σ. Dann gibt es zu jedem $k \in \mathbb{N}$ nur endliche viele – nämlich n^k – Wörter aus Σ^*, die die Länge k haben. Seien $w_{k1}, \ldots, w_{kn^k}$ diese Wörter, also $w_{01} = \varepsilon$ und $w_{11}, \ldots, w_{1n}$ die Wörter aus Σ^* der Länge 1, und so weiter. Dann ist eine Abzählung von Σ^* gegeben durch

$$w_{01}, w_{11}, \ldots, w_{1n}, w_{21}, \ldots, w_{2n^2}, w_{31}, \ldots, w_{3n^3}, \ldots .$$

q.e.d.

Satz 10 *Sei Σ eine abzählbare Menge von Zeichen. Dann ist die Menge Σ^* der Wörter über Σ abzählbar.*

Beweis: Sei $\Sigma = \{z_0, z_1, z_2, \ldots\}$. Seien nun z und $'$ zwei Zeichen. Sei $f\colon \Sigma^* \to \{z,'\}^*$ definiert durch $f(z_{n_1} \ldots z_{n_k}) := z'^{n_1} \ldots z'^{n_k}$. Dann ist f injektiv. Nach dem vorigen Satz gibt es eine injektive Funktion g von $\{z,'\}^*$ nach $\mathbb{N}$. Daher ist $g \circ f\colon \Sigma^* \to \mathbb{N}$ injektiv. q.e.d.

Insbesondere gilt für $\Sigma = \{\mathtt{0}, \mathtt{N}, \mathtt{(}, \mathtt{)}, \mathtt{R}, \mathtt{P}^1_1, \mathtt{P}^2_1, \mathtt{P}^2_2, \mathtt{P}^3_1, \mathtt{P}^3_2, \mathtt{P}^3_3, \ldots\}$, dass Σ^* abzählbar ist. Da jede Teilmenge einer abzählbaren Menge abzählbar ist, ist auch die Menge der primitivrekursiven

Funktionale abzählbar. Für jede natürliche Zahl n ist auch die Menge der n-stelligen primitivrekursiven Funktionale abzählbar.

Man kann sogar eine explizite Abzählung für Wörter aus Σ^* angeben. Um diese Wörter abzuzählen, müssen wir sie in irgendeiner Reihenfolge anordnen. Dazu verwenden wir die Idee, die hinter der Anordnung in einem Lexikon steht. Dort werden erst alle Wörter aufgelistet, die mit einem `a` beginnen, dann alle, die mit einem `b` beginnen, und so weiter. Man nennt das die *lexikographische Reihenfolge*. Leider funktioniert das in dieser Form aber nur bei einem Lexikon, das nur endlich viele Wörter enthält. Wenn wir alle Wörter über dem lateinischen Alphabet so anordnen, dann ergibt sich etwa folgende Reihenfolge.

$$\varepsilon, \mathtt{a}, \mathtt{aa}, \mathtt{aaa}, \ldots, \mathtt{aab}, \ldots, \mathtt{ab}, \mathtt{aba}, \ldots, \mathtt{b}, \ldots.$$

Es kommen also unendlich viele Wörter vor dem Wort `b` und das Wort `b` würde daher in der Aufzählung überhaupt nicht vorkommen. Als Ausweg bietet sich aber wie oben gezeigt an, erst alle Wörter der Länge 0 (das ist nur ε) aufzuzählen, dann alle der Länge 1, dann alle der Länge 2, und so weiter. Die Wörter einer festen Länge kann man dabei in lexikographischer Reihenfolge anordnen, weil es jeweils nur endlich viele sind. Wir wollen diese Idee auf primitivrekursive Funktionale übertragen. Dazu bilden wir zunächst die primitivrekursiven Funktionale vermöge der im Beweis des letzten Satzes definierten Funktion f auf Wörter über dem Alphabet $\{z,'\}$ ab. Nun müssen wir wie beim lateinischen Alphabet die Zeichen von $\{z,'\}$ in einer Reihenfolge anordnen, also eine lineare Ordnungsrelation $<$ auf Σ definieren, etwa durch

$$z < '$$

Hiermit kann man nun eine lineare Ordnungsrelation $<$ auf $\{z,'\}^*$ induktiv definieren durch:

1. Wenn $u, v \in \{z,'\}^*$ und $|u| < |v|$ ist, dann ist $u < v$.
2. Wenn $a \in \{z,'\}$, $u, v \in \{z,'\}^*$, $|u| = |v|$ und $u < v$, dann ist $au < av$.
3. Wenn $a, b \in \{z,'\}$, $a < b$, $u, v \in \{z,'\}^*$ und $|u| = |v|$, dann ist $au < bv$.

Jede nichtleere Teilmenge von $\{z,'\}^*$ hat ein bezüglich $<$ kleinstes Element. Nun sei eine Ordnungsrelation $<$ auf Σ^* definiert durch $u < v :\iff f(u) < f(v)$ für alle $u, v \in \Sigma^*$. Dann hat jede nichtleere Teilmenge von Σ^* ein bezüglich $<$ kleinstes Element.

Zu jeder unendlichen Teilmenge A von Σ^* existiert ein Ordnungsisomorphismus $\theta\colon \mathbb{N} \to A$, d.h. eine bijektive Funktion mit

$$j < k \Longrightarrow \theta(j) < \theta(k).$$

Man nennt $\theta(k)$ auch das *(k+1)-te Element* von A. Die Funktion θ heißt die *Ordnungsfunktion* von A.

Folglich ist jede Teilmenge von Σ^* abzählbar, insbesondere die Menge der primitivrekursiven Funktionale und auch die Menge der n-stelligen primitivrekursiven Funktionale.

Gleiches gilt für die Menge der (n-stelligen) primitivrekursiven Funktionen.

Folgerung: Es gibt 1-stellige zahlentheoretische Funktionen, die nicht primitivrekursiv sind. Allgemein gilt für jedes $n > 0$: Es gibt n-stellige zahlentheoretische Funktionen, die nicht primitivrekursiv sind.

Primitivrekursive Prädikate

Definition 16 Unter einem n-stelligen *zahlentheoretischen Prädikat* verstehen wir eine n-stellige Relation auf $\mathbb{N}$, also eine Teilmenge von $\mathbb{N}^n$. Wenn P ein n-stelliges zahlentheoretisches Prädikat ist und $\mathfrak{x} = (x_1, \ldots, x_n) \in \mathbb{N}^n$ ist, dann schreibt man statt $\mathfrak{x} \in P$ auch $P(\mathfrak{x})$ oder $P(x_1, \ldots, x_n)$. Bei einem zweistelligen zahlentheoretischen Prädikat P verwendet man oft die Infixschreibweise xPy für $(x, y) \in P$.

Beispiel 23
$\leq$ ist ein zweistelliges zahlentheoretisches Prädikat. $\leq$ trifft auf x und y genau dann zu, wenn $x \leq y$ ist. Als Menge betrachtet ist $\leq$ die Menge $\{(x, y) \mid x \leq y\}$.

Definition 17 Sei $n \in \mathbb{N}$ und sei $A \subseteq \mathbb{N}^n$. Dann ist die *charakteristische Funktion* χ_A der Menge A definiert durch $\chi_A \colon \mathbb{N}^n \to \mathbb{N}$ und

$$\chi_A(\mathfrak{x}) := \begin{cases} 1, & \text{falls } \mathfrak{x} \in A \text{ ist} \\ 0, & \text{falls } \mathfrak{x} \in \mathbb{N}^n \setminus A \text{ ist.} \end{cases}$$

Da ein n-stelliges zahlentheoretisches Prädikat P eine Teilmenge von $\mathbb{N}^n$ ist, besitzt es eine charakteristische Funktion χ_P, die eine n-stellige zahlentheoretische Funktion mit Funktionswerten aus $\{0, 1\}$ ist. Umgekehrt gibt es zu jeder n-stelligen zahlentheoretischen Funktion f mit Funktionswerten aus $\{0, 1\}$ genau ein n-stelliges zahlentheoretisches Prädikat P mit $f = \chi_P$, nämlich $P = \{\mathfrak{x} \in \mathbb{N}^n \mid f(\mathfrak{x}) = 1\}$.

Definition 18 Ein n-stelliges zahlentheoretisches Prädikat heißt *primitivrekursiv*, wenn seine charakteristische Funktion primitivrekursiv ist.

Lemma 2 *Ein n-stelliges zahlentheoretisches Prädikat P ist genau dann primitivrekursiv, wenn es eine n-stellige primitivrekursive Funktion f gibt mit*

$$\bigwedge_{\mathfrak{x} \in \mathbb{N}^n} \big(P(\mathfrak{x}) \iff f(\mathfrak{x}) = 0\big).$$

Beispiel 24
$\leq$ ist ein zweistelliges primitivrekursives Prädikat, da

$$\bigwedge_{x, y \in \mathbb{N}} \big(x \leq y \iff \operatorname{diff}(x, y) = 0\big)$$

gilt.

Definition 19 Sind P und Q zwei n-stellige zahlentheoretische Prädikate, so sind die *Negation* $\neg P$ von P und die *Konjunktion* $P \wedge Q$ und die *Disjunktion* $P \vee Q$ von P und Q diejenigen n-stelligen zahlentheoretischen Prädikate, die definiert sind durch

$$\begin{aligned} (\neg P)(\mathfrak{x}) &: \iff \neg P(\mathfrak{x}) \\ (P \wedge Q)(\mathfrak{x}) &: \iff P(\mathfrak{x}) \wedge Q(\mathfrak{x}) \\ (P \vee Q)(\mathfrak{x}) &: \iff P(\mathfrak{x}) \vee Q(\mathfrak{x}) \end{aligned}$$

für alle $\mathfrak{x} \in \mathbb{N}^n$.

Die Negation eines n-stelligen primitivrekursiven Prädikats ist primitivrekursiv. Die Konjunktion oder Disjunktion zweier n-stelliger primitivrekursiver Prädikate ist primitivrekursiv.

Beispiel 25
Die Prädikate $<$, $>$, $=$, $\leq$, $\geq$, $\neq$ sind primitivrekursiv.

Die explizite Transformation ist ebenso wie für zahlentheoretische Funktionen auch für zahlentheoretische Prädikate möglich. Dabei wird ein n-stelliges Prädikat P definiert, indem man für $P(x_1, \ldots, x_n)$ einen äquivalenten logischen Ausdruck angibt. Dieser Ausdruck muss aufgebaut sein aus den Variablen $x_1, \ldots, x_n$, aus natürlichen Zahlen, aus zahlentheoretischen Funktionen und Prädikaten und aus den aussagenlogischen Junktoren $\neg$, $\wedge$ und $\vee$. Sind die darin auftretenden zahlentheoretischen Funktionen und Prädikate alle primitivrekursiv, so ist auch P primitivrekursiv. So folgt zum Beispiel aus der Primitivrekursivität des zweistelligen zahlentheoretischen Prädikats $\leq$, dass auch das zweistellige zahlentheoretische Prädikat $\neq$ primitivrekursiv ist, da gilt: $x \neq y \iff \neg(x \leq y \wedge y \leq x)$.

Beschränkte Quantifizierung

Definition 20 Sei P ein $(n+1)$-stelliges zahlentheoretisches Prädikat. Dann sind die *beschränkte Allquantifizierung* $(\forall_b P)$ und die *beschränkte Existenzquantifizierung* $(\exists_b P)$ von P die $(n+1)$-stelligen Prädikate, die definiert sind durch

$$(\forall_b P)(\mathfrak{x}, y) : \iff \bigwedge_{z \leq y} P(\mathfrak{x}, z)$$
$$(\exists_b P)(\mathfrak{x}, y) : \iff \bigvee_{z \leq y} P(\mathfrak{x}, z).$$

Satz 11 *Wenn P ein primitivrekursives $(n+1)$-stelliges Prädikat ist, dann sind auch die Prädikate $(\forall_b P)$ und $(\exists_b P)$ primitivrekursiv.*

Beweis: Nach Voraussetzung ist P, d.h. auch ihre charakteristische Funktion χ_P primitivrekursiv. Nun gilt

$$\chi_{(\forall_b P)} = (\Pi \chi_P)$$
$$\chi_{(\exists_b P)} = (\mathrm{sg}(\Sigma \chi_P)).$$

Damit sind auch die charakteristischen Funktionen $\chi_{(\forall_b P)}$ und $\chi_{(\exists_b P)}$ und damit auch die Prädikate $(\forall_b P)$ und $(\exists_b P)$ primitivrekursiv. q.e.d.

Beschränkter μ-Operator

Definition 21 Für ein $(n+1)$-stelliges zahlentheoretisches Prädikat P ist die $(n+1)$-stellige zahlentheoretische Funktion $(\mu_b P)$ definiert durch

$$(\mu_b P)(\mathfrak{x}, y) := \begin{cases} \min\{z \leq y \mid P(\mathfrak{x}, z)\}, & \text{falls so ein } z \text{ existiert} \\ 0 & \text{sonst.} \end{cases}$$

Für eine $(n+1)$-stellige zahlentheoretische Funktion f ist die $(n+1)$-stellige zahlentheoretische Funktion $(\mu_b f)$ definiert durch

$$(\mu_b f)(\mathfrak{x}, y) := \begin{cases} \min\{z \leq y \mid f(\mathfrak{x}, z) = 0\}, & \text{falls so ein } z \text{ existiert} \\ 0 & \text{sonst.} \end{cases}$$

Wir sprechen von Anwendung des *beschränkten μ-Operators* oder von *beschränkter Minimalisierung*. Statt $(\mu_b P)(\mathfrak{x}, y)$ werden wir auch schreiben $\mu_b z \leq y \colon P(\mathfrak{x}, z)$.

Satz 12 *Wenn P ein $(n+1)$-stelliges primitivrekursives Prädikat ist, dann ist $(\mu_b P)$ eine $(n+1)$-stellige primitivrekursive Funktion. Wenn f eine $(n+1)$-stellige primitivrekursive Funktion ist, dann ist $(\mu_b f)$ eine $(n+1)$-stellige primitivrekursive Funktion.*

Beweis: Es ist $(\mu_b P) = (RK_0^n g)$, wobei

$$g(u, \mathfrak{x}, y) := \begin{cases} y', & \text{falls } P(\mathfrak{x}, y') \wedge \neg \bigvee_{z \leq y} P(\mathfrak{x}, z) \\ u & \text{sonst.} \end{cases}$$

Ist f eine $(n+1)$-stellige primitivirekursive Funktion, so ist $P := \{(\mathfrak{x}, y) \mid f(\mathfrak{x}, y) = 0\}$ ein $(n+1)$-stelliges primitivrekursives Prädikat und nach dem eben Gezeigten ist daher $(\mu_b P)$ primitivrekursiv. Wegen $(\mu_b f) = (\mu_b P)$ ist also $(\mu_b f)$ primitivrekursiv. q.e.d.

div, wur, kgV, ggT sind primitivrekursiv.

$$\operatorname{div}(x, y) = \mu_b d \leq x : (d+1)y > x.$$

$$\operatorname{div}(x, y) = \begin{cases} \lfloor \frac{x}{y} \rfloor, & \text{wenn } y \neq 0 \\ 0 & \text{sonst.} \end{cases}$$

$$\operatorname{div}(x, y) = (\mu_b P)(x, y, x)$$

$$\operatorname{div} = ((\mu_b P) P_1^2 P_2^2 P_1^2)$$

$$x \bmod y = x \dot{-} y \cdot \operatorname{div}(x, y)$$

$$\operatorname{wur}(x) = \lfloor \sqrt{x} \rfloor$$

$$\operatorname{kgV}(x, y) = \mu_b z \leq xy : (x|z \wedge y|z)$$

$$x|z \iff \bigvee_{u \leq z} xu = z$$

Für $(\mu_b P)(\mathfrak{x}, y)$ schreiben wir auch $\mu_b z \leq y : P(\mathfrak{x}, z)$.
$(\mu_b f) = (\mu_b P)$, wenn $\bigwedge_{\mathfrak{x} \in \mathbb{N}^n, y \in \mathbb{N}} (P(\mathfrak{x}, y) \iff f(\mathfrak{x}, y) = 0)$.
$\operatorname{ggT}(x, y)\operatorname{kgV}(x, y) = xy$

$$\operatorname{ggT}(x, y) = \frac{xy}{\operatorname{kgV}(x, y)}$$

$$\mathrm{ggT}(x,y) = (x+y) \dot{-} \mu_b s \leq x+y : (x+y \dot{-} s | x \wedge x+y \dot{-} s | y)$$

$$\mathrm{ggT}(x,y) = \mu_b t \leq x+y : (t|x \wedge t|y \wedge \neg \bigvee_{u \leq x+y} (u > t \wedge u|x \wedge u|y))$$

Die Cantor'sche Paarfunktion

Definition 22 Die *Paarfunktion* $(x,y) \mapsto \langle x,y \rangle \colon \mathbb{N}^2 \to \mathbb{N}$ ist definiert durch

$$\langle x,y \rangle = \frac{(x+y)(x+y+1)}{2} + y.$$

Die Paarfunktion ist bijektiv und primitivrekursiv. Es gilt für alle $x, y \in \mathbb{N}$:

$$x \leq \langle x,y \rangle$$
$$y \leq \langle x,y \rangle$$

Definition 23 Für $n \in \mathbb{N} \setminus \{0\}$ ist die *n-Tupelfunktion* $(x_1, \ldots, x_n) \mapsto \langle x_1, \ldots, x_n \rangle \colon \mathbb{N}^n \to \mathbb{N}$ durch Induktion nach n definiert durch

$$\langle x \rangle := x$$
$$\langle x_1, \ldots, x_{n+1} \rangle := \langle \langle x_1, \ldots, x_n \rangle, x_{n+1} \rangle$$

Da die Definition für $\langle x_1, x_2 \rangle$ der obigen Definition der Paarfunktion nicht widerspricht, wird hierdurch die n-Tupelfunktion eindeutig definiert. Sie ist primitivrekursiv, da sie sich aus der Paarfunktion mittels wiederholter expliziter Transformation ergibt, und sie ist bijektiv. Für $n \in \mathbb{N} \setminus \{0\}$, $1 \leq k \leq n$ und $x_1, \ldots, x_n \in \mathbb{N}$ gilt

$$x_k \leq \langle x_1, \ldots, x_n \rangle.$$

Definition 24 Die *Projektionen* $\pi^n_k \colon \mathbb{N} \to \mathbb{N}$ für $k, n \in \mathbb{N}$ mit $k \leq n$ sind definiert durch

$$\pi^n_k(\langle x_1, \ldots, x_n \rangle) = x_k.$$

Die Projektionen sind primitivrekursiv. Denn für jedes $x \in \mathbb{N}$ ist $\pi^n_k(x)$ die kleinste natürliche Zahl x_k so, dass natürliche Zahlen $x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n$ existieren so, dass $x = \langle x_1, \ldots, x_n \rangle$. Da die Zahlen x^n_k kleinergleich x sind, gilt

$\pi^n_k(x) = \mu_b x_k \leq x \colon$

$$\bigvee_{x_1 \leq x} \cdots \bigvee_{x_{k-1} \leq x} \bigvee_{x_{k+1} \leq x} \cdots \bigvee_{x_n \leq x} x = \langle x_1, \ldots, x_n \rangle.$$

Die Projektionen lassen sich also mittels expliziter Transformation, beschränkter Existenzquantifizierung und beschränkter Minimalisierung durch primitivrekursive Funktionen und Prädikate ausdrücken und sind damit selbst primitivrekursiv.

Die Projektionen spielen die Rolle von Umkehrfunktionen der n-Tupelfunktion.

Definition 25 Eine Funktion $\mathfrak{f} \colon \mathbb{N}^m \to \mathbb{N}^n$ heißt *primitivrekursiv*, wenn es n m-stellige primitivrekursive Funktionen $f_1, \ldots, f_n$ gibt so, dass für alle $\mathfrak{x} \in \mathbb{N}^m$ gilt

$$\mathfrak{f}(\mathfrak{x}) = (f_1(\mathfrak{x}), \ldots, f_n(\mathfrak{x})).$$

Aufgabe 6 Für eine einstellige zahlentheoretische Funktion f ist die *Iteration* $(\mathrm{It}\, f)$ von f eine zweistellige zahlentheoretische Funktion, die gegeben ist durch

$$(\mathrm{It}\, f)(x, y) = \underbrace{f(f(\dots f(f}_{y \text{ mal das } f}(x))\dots)).$$

1. Definieren Sie $(\mathrm{It}\, f)$ durch Rekursionsgleichungen.
2. Zeigen Sie, dass $(\mathrm{It}\, f)$ primitivrekursiv ist, wenn f primitivrekursiv ist.

Definition 26 Für eine Funktion $\mathfrak{f}\colon \mathbb{N}^n \to \mathbb{N}^n$ ist die *Iteration* $(\mathrm{It}\,\mathfrak{f})\colon \mathbb{N}^{n+1} \to \mathbb{N}^n$ gegeben durch

$$(\mathrm{It}\,\mathfrak{f})(\mathfrak{x}, y) = \underbrace{\mathfrak{f}(\mathfrak{f}(\dots \mathfrak{f}(\mathfrak{f}}_{y \text{ mal das } \mathfrak{f}}(\mathfrak{x}))\dots)).$$

Die exakte Definition durch Rekursionsgleichungen ist analog wie bei Teil 1 der obigen Aufgabe und bleibt hier als Übung für den Leser.

In Teil 2 der obigen Aufgabe wurde für $n = 1$ gezeigt, dass $(\mathrm{It}\,\mathfrak{f})$ primitivrekursiv ist, wenn $\mathfrak{f}$ primitivrekursiv ist. Dies gilt auch für beliebiges $n \in \mathbb{N} \setminus \{0\}$.

Lemma 3 *Sei $n \in \mathbb{N} \setminus \{0\}$ und sei $\mathfrak{f}\colon \mathbb{N}^n \to \mathbb{N}^n$ primitivrekursiv. Dann ist auch $(\mathrm{It}\,\mathfrak{f})\colon \mathbb{N}^{n+1} \to \mathbb{N}^n$ primitivrekursiv.*

Beweis: Die Idee des Beweises besteht darin ein n-Tupel $\mathfrak{x} = (x_1, \dots, x_n)$ durch die natürliche Zahl $\langle x_1, \dots, x_n\rangle$ zu codieren. Wir wollen diese Zahl auch mit $\langle\mathfrak{x}\rangle$ bezeichnen. Um $(\mathrm{It}\,\mathfrak{f})(\mathfrak{x}, y)$ zu berechnen kann man nun – anstatt y mal die Funktion $\mathfrak{f}$ auf $\mathfrak{x}$ anzuwenden – zunächst das Tupel $\mathfrak{x}$ durch die natürliche Zahl $\langle\mathfrak{x}\rangle$ codieren, dann auf diese Zahl y mal die Funktion $g = \langle\mathfrak{x}\rangle \mapsto \langle\mathfrak{f}(\mathfrak{x})\rangle\colon \mathbb{N} \to \mathbb{N}$ anwenden und schließlich die erhaltene Zahl wieder zu einem n-Tupel decodieren. Wir werden zeigen, dass g eindeutig definiert und primitivrekursiv ist. Dass die Iteration einer primitivrekursiven Funktion $g\colon \mathbb{N} \to \mathbb{N}$ wieder primitivrekursiv ist, haben wir ja schon in der letzten Aufgabe gezeigt. Man kann sich den Prozess durch das folgende Bild veranschaulichen.

$$\begin{array}{lll} \mathfrak{x} & & \underbrace{\mathfrak{f}(\dots \mathfrak{f}}_{y \text{ mal}}(\mathfrak{x})\dots) \\ \updownarrow & & \updownarrow \\ \langle\mathfrak{x}\rangle \mapsto \langle\mathfrak{f}(\mathfrak{x})\rangle \mapsto \langle\mathfrak{f}(\mathfrak{f}(\mathfrak{x}))\rangle \mapsto \cdots \mapsto & & \langle\underbrace{\mathfrak{f}(\dots \mathfrak{f}}_{y \text{ mal}}(\mathfrak{x})\dots)\rangle \end{array}$$

Die Funktion g kann man sich durch das folgende Bild veranschaulichen.

$$\begin{array}{lccll} \mathfrak{f}: & \mathfrak{x} & \mapsto & \mathfrak{f}(\mathfrak{x}) & n\text{-Tupel von natürlichen Zahlen} \\ & \updownarrow & & \updownarrow & \\ g: & \langle\mathfrak{x}\rangle & \mapsto & \langle\mathfrak{f}(\mathfrak{x})\rangle & \text{natürliche Zahlen (‘Codierungen’)} \end{array}$$

Da $\mathfrak{f}$ nach Voraussetzung primitivrekursiv ist, gibt es nach Definition n n-stellige primitivrekursive Funktionen $f_1, \dots, f_n$ so, dass $\mathfrak{f}(\mathfrak{x}) = (f_1(\mathfrak{x}), \dots, f_n(\mathfrak{x}))$ ist für alle $\mathfrak{x} \in \mathbb{N}^n$. Nun gilt für $x = \langle\mathfrak{x}\rangle$, dass $\mathfrak{x} = (\pi_1^n(x), \dots, \pi_n^n(x))$ ist. Da die n-Tupelfunktion bijektiv ist, ist g als $\langle\mathfrak{x}\rangle \mapsto \langle\mathfrak{f}(\mathfrak{x})\rangle$ eindeutig definiert und es ist

$$\begin{aligned} g(x) &= \langle\mathfrak{f}(\pi_1^n(x), \dots, \pi_n^n(x))\rangle \\ &= \langle f_1(\pi_1^n(x), \dots, \pi_n^n(x)), \dots, f_n(\pi_1^n(x), \dots, \pi_n^n(x))\rangle. \end{aligned}$$

Da die π_k^n, die f_k und die n-Tupelfunktion primitivrekursiv sind, ist g primitivrekursiv. Nach der letzten Aufgabe ist damit auch $(\mathrm{It}\, g)$ primitivrekursiv. Nun zeigt man leicht durch vollständige Induktion nach y, dass für alle $\mathfrak{x} \in \mathbb{N}^n$ und alle $y \in \mathbb{N}$ gilt

$$\langle (\mathrm{It}\, \mathfrak{f})(\mathfrak{x}, y)\rangle = (\mathrm{It}\, g)(\langle \mathfrak{x}\rangle, y).$$

Es folgt

$$(\mathrm{It}\, \mathfrak{f})(\mathfrak{x}, y) = (\pi_1^n((\mathrm{It}\, g)(\langle \mathfrak{x}\rangle, y)), \ldots, \pi_n^n((\mathrm{It}\, g)(\langle \mathfrak{x}\rangle, y))).$$

Also ist $(\mathrm{It}\, \mathfrak{f})$ primitivrekursiv. q.e.d.

Die Petersche Funktion

Die Petersche Funktion $p\colon \mathbb{N}^2 \to \mathbb{N}$ ist definiert durch die Rekursionsgleichungen

$$p(0, y) = y' \tag{2.1}$$
$$p(x', 0) = p(x, 1) \tag{2.2}$$
$$p(x', y') = p(x, p(x', y)). \tag{2.3}$$

Wir werden zeigen, dass $p(x, 0)$ mit x schneller als jede primitivrekursive Funktion anwächst. Damit ist insbesondere p nicht primitivrekursiv, obwohl p berechenbar ist.

Lemma 4 *Die Petersche Funktion p ist durch diese Rekursionsgleichungen eindeutig definiert.*

Dieses Lemma kann man beweisen durch zweimalige Anwendung von Satz 1:

Beweis: Nach Satz 1 wird für jedes $r\colon \mathbb{N} \to \mathbb{N}$ und $x \in \mathbb{N}$ eine Funktion $g_{rx}\colon \mathbb{N} \to \mathbb{N}$ eindeutig definiert durch die Rekursionsgleichungen

$$g_{rx}(0) = r(1) \tag{2.4}$$
$$g_{rx}(y') = r(g_{rx}(y)). \tag{2.5}$$

Nun wird, ebenfalls nach Satz 1, eine Funktion $q\colon \mathbb{N} \to \mathbb{N}^{\mathbb{N}}$ eindeutig definiert durch die Rekursionsgleichungen

$$q(0) = N \tag{2.6}$$
$$q(x') = g_{q(x)x}. \tag{2.7}$$

Dabei sei N die Nachfolgerfunktion auf $\mathbb{N}$. Setzt man in (2.4) und (2.5) $r := q(x)$, so besagt die Eindeutigkeit, dass für eine einstellige zahlentheoretische Funktion f genau dann $f = g_{q(x)x}$ gilt, wenn für alle $y \in \mathbb{N}$ die Gleichungen

$$f(0) = q(x)(1)$$
$$f(y') = q(x)(f(y))$$

gelten. Dies gilt insbesondere für die Funktion $f := q(x')$. Die Gleichung 2.7 ist also äquivalent zu den Gleichungen.

$$q(x')(0) = q(x)(1)$$
$$q(x')(y') = q(x)(q(x')(y)).$$

Insgesamt sind also die Rekursionsgleichungen (2.6) und (2.7) äquivalent zu den Gleichungen

$$\begin{aligned} q(0)(y) &= y' \\ q(x')(0) &= q(x)(1) \\ q(x')(y') &= q(x)(q(x')(y)). \end{aligned}$$

Durch diese drei Gleichungen wird also eindeutig eine Funktion $q\colon \mathbb{N} \to \mathbb{N}^{\mathbb{N}}$ definiert. Da jeder Funktion $q\colon \mathbb{N} \to \mathbb{N}^{\mathbb{N}}$ genau eine Funktion $p\colon \mathbb{N}^2 \to \mathbb{N}$ entspricht und umgekehrt, wobei $p(x,y) = q(x)(y)$ gilt, ist daher die Funktion p eindeutig definiert durch die Gleichungen (2.1), (2.2) und (2.3). q.e.d.

Eine andere Möglichkeit das Lemma zu beweisen erfolgt, indem man durch geschachtelte vollständige Induktion (Induktion nach x mit geschachtelter Induktion nach y) zeigt, dass $p(x,y)$ eindeutig definiert ist. Mathematisch etwas unbefriedigend bei einem solchen Beweis ist, dass mathematisch nicht ganz klar ist, was es bedeutet, $p(x,y)$ sei eindeutig definiert durch die Rekursionsgleichungen, wo doch durch die Rekursionsgleichungen die gesamte Funktion p definiert wird. Man muss also genauer präzisieren, was man meint mit „$p(x,y)$ ist eindeutig definiert“. Hierzu bemerken wir, dass man die Rekursionsgleichungen als induktive Definition des Wertes von $p(x,y)$ auffassen kann. Genauer gesagt wird der Graph von p induktiv definiert. Wir sagen nun $p(x,y)$ sei eindeutig definiert, wenn es genau ein z gibt so, dass (x,y,z) in dem so definierten Graphen liegt. Man kann dann den Beweis so führen:

Beweis durch vollständige Induktion nach x:

INDUKTIONSANFANG: $x = 0$: $p(0,y)$ ist eindeutig definiert.

Beweis: trivial

INDUKTIONSVORAUSSETZUNG: $p(x,y)$ sei eindeutig definiert für alle $y \in \mathbb{N}$.

INDUKTIONSBEHAUPTUNG: $p(x',y)$ ist eindeutig definiert für alle $y \in \mathbb{N}$.

Beweis: *Beweis* durch vollständige Induktion nach y

INDUKTIONSANFANG: $y = 0$: $p(x',0)$ ist eindeutig definiert.

Beweis: Nach Induktionsvoraussetzung bezüglich x ist $p(x,1)$ eindeutig definiert und daher auch $p(x',0)$.

INDUKTIONSVORAUSSETZUNG: $p(x',y)$ sei eindeutig definiert.

INDUKTIONSBEHAUPTUNG: $p(x',y')$ ist eindeutig definiert.

Beweis: Nach Induktionsvoraussetzung bezüglich y ist $p(x',y)$ eindeutig definiert. Nach Induktionsvoraussetzung bezüglich x ist daher auch $p(x,p(x',y))$ und damit auch $p(x',y')$ eindeutig definiert.

Nach dem Prinzip der vollständigen Induktion (über y) ist daher $p(x',y)$ eindeutig definiert für alle $y \in \mathbb{N}$.

Nach dem Prinzip der vollständigen Induktion (über x) ist daher $p(x,y)$ eindeutig definiert für alle $x,y \in \mathbb{N}$.

Die Petersche Funktion p ist berechenbar. Man kann für sie in jeder höheren Programmiersprache ein kurzes Programm angeben. Die Petersche Funktion hat einige nützliche Eigenschaften.

Strikte Monotonie im ersten Argument:

$$x < y \Longrightarrow p(x,z) < p(y,z)$$

Strikte Monotonie im zweiten Argument:

$$\begin{aligned} y < z &\Longrightarrow p(x,y) < p(x,z) \\ p(x,p(y,z)) &\leq p(\max(x+2,y+1),z) \\ m \cdot p(x,y) &\leq p(x+m,y). \end{aligned}$$

Um zu zeigen, dass die Petersche Funktion nicht primitivrekusiv ist, verwenden wir zwei Notationen. Die *Norm* (genauer die *1-Norm*) $\|\mathfrak{x}\|$ eines Tupels $\mathfrak{x}$ von natürlichen Zahlen ist definiert durch $\|(x_1,\ldots,x_n)\| := x_1 + \cdots + x_n$. Eine zahlentheoretische Funktion $f\colon \mathbb{N}^n \to \mathbb{N}$ heiße *k-beschränkt* genau dann, wenn gilt

$$\bigwedge_{\mathfrak{x}\in\mathbb{N}^n} f(\mathfrak{x}) \leq p(k, \|\mathfrak{x}\|).$$

Lemma 5 *Zu jeder primitivrekursiven Funktion f gibt es ein k so, dass f k-beschränkt ist.*

Beweis durch Induktion nach der Definition des Begriffs der primitivrekursiven Funktion:

1. O ist 0-beschränkt.
2. N ist 0-beschränkt.
3. P_i^n ist 0-beschränkt.
4. Wenn f m-stellig und k_0-beschränkt ist und g_j n-stellig und k_j-beschränkt ist für $j = 1,\ldots,m$, dann ist $(fg_1\ldots g_m)$ k-beschränkt mit $k := \max(k_0,k_1,\ldots,k_m)+m+1$. Denn

$$\begin{aligned} (fg_1\ldots g_m)(\mathfrak{x}) &= f(g_1(\mathfrak{x}),\ldots,g_m(\mathfrak{x})) \\ &\leq p(k_0, \|(g_1(\mathfrak{x}),\ldots,g_m(\mathfrak{x}))\|) \\ &= p(k_0, g_1(\mathfrak{x}) + \cdots + g_m(\mathfrak{x})) \\ &\leq p(k_0, p(k_1,\|\mathfrak{x}\|) + \cdots + p(k_m,\|\mathfrak{x}\|)) \\ &\leq p(k_0, m \cdot p(\max(k_1,\ldots,k_m),\|\mathfrak{x}\|)) \\ &\leq p(k-2, m \cdot p(k-m-1,\|\mathfrak{x}\|)) \\ &\leq p(k-2, p(k-1,\|\mathfrak{x}\|)) \\ &\leq p(k,\|\mathfrak{x}\|). \end{aligned}$$

5. Wenn f n-stellig und k_1-beschränkt ist und g $(n+2)$-stellig und k_2-beschränkt ist, dann zeigt man durch vollständige Induktion nach y, dass gilt

$$(Rfg)(\mathfrak{x},y) \leq p(k, \|\mathfrak{x}\| + y)$$

mit $k := \max(k_1,k_2)+2$. Damit ist (Rfg) k-beschränkt.

q.e.d.

Satz 13 *p ist nicht primitivrekursiv.*

Beweis: Angenommen, p wäre primitivrekursiv. Die Funktion $f\colon \mathbb{N} \to \mathbb{N}$ sei definiert durch

$$f(x) := p(x, x) + 1.$$

Dann ist auch f primitivrekursiv. Nach dem Lemma gibt es daher ein $k \in \mathbb{N}$ so, dass f k-beschränkt ist. Das heißt, für alle $x \in \mathbb{N}$ gilt $f(x) \leq p(k, x)$. Insbesondere gilt also $f(k) \leq p(k, k)$. Aber nach Definition von f ist $f(k) = p(k, k) + 1$. Dies ist ein Widerspruch. q.e.d.

Für festes x ist die Funktion $y \mapsto p(x, y)$ primitivrekursiv, wie man leicht durch Induktion nach x nachweist. Aber für kein $y \in \mathbb{N}$ ist $x \mapsto p(x, y)$ primitivrekursiv. Selbst die Funktion $x \mapsto p(x, 0)$ wächst mit x stärker an als jede primitivrekursive Funktion.

2.2.3 Existenz von berechenbaren nicht primitivrekursiven Funktionen

Wir haben auf der Menge der primitivrekursiven Funktionale eine lineare Ordnungsrelation definiert so, dass die Menge der primitivrekursiven Funktionale mit dieser Ordnungsrelation ordnungsisomorph zur Menge der natürlichen Zahlen mit der üblichen Ordnungsrelation ist. Wir haben gesehen, dass für $n \in \mathbb{N}$ ein Ordnungsisomorphismus von $\mathbb{N}$ in die Menge der n-stelligen primitivrekursiven Funktionale existiert, der jeder natürlichen Zahl k das $(k + 1)$-te n-stellige Funktional (bezüglich der auf der Menge der primitivrekursiven Funktionale definierten Ordnungsrelation) zuordnet. Wir bezeichnen im Folgenden das $(k+1)$-te n-stellige primitivrekursive Funktional mit ϕ_k^n. Zu gegebenem $n \in \mathbb{N}$ und $k \in \mathbb{N}$ kann man explizit das Funktional ϕ_k^n ermitteln.

Aufgabe 7 Geben Sie einen Algorithmus an, der zu je zwei natürlichen Zahlen n und k das Funktional ϕ_k^n liefert.

Aufgabe 8 Die Funktion f sei folgendermaßen definiert. Für jede natürliche Zahl k sei $f(k)$ der Wert des Funktionals ϕ_k^1 an der Stelle k. Geben Sie einen Algorithmus an, der bei Eingabe einer beliebigen natürlichen Zahl k die Zahl $f(k)$ als Ausgabe liefert.

Aufgabe 9 Zeigen Sie, dass die Funktion f nicht primitivrekursiv ist.
Hinweis: Verwenden Sie das Cantor'sche Diagonalvorfahren.

Aufgabe 10 Sei Σ ein Alphabet und es sei eine totale irreflexive Ordnung auf Σ gegeben. Schreiben Sie einen Algorithmus, der zu jedem Wort aus Σ^* das bezüglich der früher definierten zu $\mathbb{N}$ ordnungsisomorphen totalen irreflexiven Ordnung nächste Element von Σ^* berechnet. Dieses ist das bezüglich der lexikographischen Ordnung nächste von gleicher Länge, falls dieses existiert, sonst das lexikographisch kleinste von größerer Länge.

Wie das Beispiel der Peterschen Funktion und die Aufgaben dieses Abschnitts zeigen, gibt es berechenbare zahlentheoretische Funktionen, die nicht primitivrekursiv sind. Die Argumentation dieser Aufgaben mit dem **Cantor'schen Diagonalverfahren** funktioniert nicht nur für das System der primitivrekursiven Funktionen, sondern für beliebige formale Systeme von berechenbaren Funktionen.

Satz 14 *Sei Σ ein Alphabet und sei L eine Sprache über Σ, d.h. $L \subseteq \Sigma^*$. Weiter sei jedem Wort ϕ aus L eine zahlentheoretische Funktion W_ϕ zugeordnet mit den folgenden Eigenschaften.*

1. *Es ist entscheidbar, ob ein Wort aus Σ^* in L liegt.*

2. *Zu einem Wort ϕ aus L ist die Stelligkeit der zugeordneten zahlentheoretischen Funktion W_ϕ berechenbar.*

3. *Zu einem Wort ϕ aus L und einem n-Tupel $\mathfrak{x}$ von natürlichen Zahlen, wobei n die Stelligkeit der dem Wort ϕ zugeordneten Funktion W_ϕ ist, ist der Wert $W_\phi(\mathfrak{x})$ dieser Funktion für dieses Tupel $\mathfrak{x}$ berechenbar.*

Dann gibt es eine berechenbare zahlentheoretische Funktion, die keinem Wort der Sprache L zugeordnet ist.

Beweis: Wie oben definieren wir eine totale Ordnung $<$ auf Σ^*, die ordnungsisomorph zur üblichen Ordnung auf $\mathbb{N}$ ist. Sei A die Menge der Wörter aus L, deren zugeordnete Funktion einstellig ist.

Wenn A endlich ist, dann ist die Menge der Elementen von A zugeordneten Funktionen auch endlich. Da es aber unendlich viele berechenbare einstellige zahlentheoretische Funktionen gibt, gibt es eine berechenbare einstellige zahlentheoretische Funktion, die nicht darunter ist. Daher kann sie auch keinem Wort aus L zugeordnet sein.

Sei also nun A unendlich. Sei θ die Ordnungsfunktion von A. Dann ist $A = \{\theta(0), \theta(1), \theta(2), \dots\}$. Die Ordnungsfunktion θ ist berechenbar, da A wegen der Eigenschaften 1. und 2. entscheidbar ist und da zu jedem Wort aus Σ^* das bezüglich $<$ nächste Wort aus Σ^* berechenbar ist (vgl. die vorige Aufgabe). Wegen der Eigenschaft 3. ist daher auch die Funktion $f\colon \mathbb{N} \to \mathbb{N}$ mit

$$f(x) := W_{\theta(x)}(x) + 1$$

berechenbar. Die Funktion f ist keinem Wort aus A zugeordnet. Denn andernfalls gäbe es ein $k \in \mathbb{N}$ so, dass dem Wort $\theta(k)$ die Funktion f zugeordnet wäre, also $f = W_{\theta(k)}$. Dann wäre

$$f(k) = W_{\theta(k)}(k) + 1 = f(k) + 1,$$

was ein Widerspruch ist. Die Funktion f ist also keinem Wort aus A und damit auch keinem Wort aus L zugeordnet. q.e.d.

Folgerung: Für jedes sinnvolle System zur Darstellung **aller** berechenbaren zahlentheoretischen Funktionen ist es unentscheidbar, ob einem Wort eine zahlentheoretische Funktion zugeordnet ist.

Für Programmiersprachen bedeutet dies, dass es unentscheidbar ist, ob ein Programm terminiert.

2.2.4 Partielle zahlentheoretische Funktionen

Definition 27 Sei $n \in \mathbb{N}$, $A \subseteq \mathbb{N}^n$ und $f\colon A \to \mathbb{N}$. Dann sagt man, f sei eine *n-stellige partielle zahlentheoretische Funktion*. Wir schreiben dann auch $f\colon \mathbb{N}^n \xrightarrow{\text{part}} \mathbb{N}$. Wenn $A = \mathbb{N}^n$ ist, sprechen wir von einer *totalen* Funktion.

Idee: Für $\mathfrak{x} \in \mathbb{N}^n \setminus A$ ist $f(\mathfrak{x})$ undefiniert. Dem entspricht Nichtterminierung eines Algorithmus.

Aufgabe 11 In Programmiersprachen kommt oft ein `if then else`-Konstrukt vor. Sei P ein n-stelliges primitivrekursives Prädikat und seien f und g zwei n-stellige primitivrekursive Funktionen. Zeigen Sie, dass die folgendermaßen definierte n-stellige zahlentheoretische Funktion h ebenfalls primitivrekursiv ist.

$$h(\mathfrak{x}) := \begin{cases} f(\mathfrak{x}), & \text{falls } P(\mathfrak{x}) \text{ gilt} \\ g(\mathfrak{x}) & \text{sonst.} \end{cases}$$

Aufgabe 12 In Programmiersprachen kommt oft ein `while`-Konstrukt vor. Sei P ein einstelliges zahlentheoretisches Prädikat und f eine einstellige zahlentheoretische Funktion. Betrachten Sie den folgenden Pseudocode.

$$x := a; \quad \texttt{while } P(x) \texttt{ do } x := f(x); \quad \texttt{return } x;$$

Nehmen wir weiter an, wir wissen, dass die `while`-Schleife immer terminiert und dass die Anzahl der Schleifendurchläufe $\leq g(a)$ ist.

1. Drücken Sie die Anzahl der Schleifendurchläufe aus durch das Prädikat P, und die Funktionen f und g mit Hilfe von expliziter Transformation, Iteration und beschränktem μ-Operator.
2. Drücken Sie die gelieferte Ausgabe x entsprechend aus.

2.2.5 μ-partiellrekursive Funktionen

In diesem Abschnitt wird das System der primitivrekursiven Funktionale und Funktionen erweitert durch Hinzunahme des sogenannten (unbeschränkten) μ-Operators. Das resultierende System ist das System der *μ-partiellrekursiven Funktionale* und die darin ausdrückbaren Funktionen sind die *μ-partiellrekursiven Funktionen*. Mit dieser Erweiterung wird das System so mächtig, dass man darin alle Mechanismen zur Beschreibung berechenbarer Funktionen, die wir heute kennen, ausdrücken kann. Es sind darin also alle nach unserem heutigen Wissen berechenbaren Funktionen erfasst.

μ-partiellrekursive Funktionale

Dem Formalismus, der hier definiert wird, liegt die Menge Σ zugrunde, die aus den folgenden Zeichen besteht:

1. `O`
2. `N`
3. $\texttt{P}_i^n$ für alle $i, n \in \mathbb{N}$ mit $1 \leq i \leq n$
4. `(`
5. `)`
6. `R`
7. μ

Definition 28 (μ-partiellrekursive Funktionale) *μ-patiellrekursive Funktionale* sind spezielle Wörter über der Zeichenmenge Σ. Jedes μ-partiellrekursive Funktional ϕ hat eine *Stelligkeit* n. Diese ist eine natürliche Zahl und wir sagen dann, das Funktional ϕ sei *n-stellig*. Diese Begriffe sind induktiv folgendermaßen definiert.

1. `O` ist ein 0-stelliges μ-partiellrekursives Funktional.
   `O` heißt das *Nullfunktional*.
2. `N` ist ein 1-stelliges μ-partiellrekursives Funktional.
   `N` heißt das *Nachfolgerfunktional*.

3. Für $i, n \in \mathbb{N}$ mit $1 \leq i \leq n$ ist $\mathtt{P}_i^n$ ein n-stelliges μ-partiellrekursives Funktional.
   $\mathtt{P}_i^n$ heißt das i-te n-stellige *Projektionsfunktional*.

4. Ist ϕ ein m-stelliges μ-partiellrekursives Funktional und sind $\psi_1, \ldots, \psi_m$ n-stellige μ-partiellrekursive Funktionale, so ist $(\phi\psi_1 \ldots \psi_m)$ ein n-stelliges μ-partiellrekursives Funktional.
   $(\phi\psi_1 \ldots \psi_m)$ heißt die *Komposition* von ϕ mit $\psi_1, \ldots, \psi_m$.

5. Ist ϕ ein n-stelliges μ-partiellrekursives Funktional und ist ψ ein $(n+2)$-stelliges μ-partiellrekursives Funktional, so ist $(\mathtt{R}\phi\psi)$ ein $(n+1)$-stelliges μ-partiellrekursives Funktional.
   $(\mathtt{R}\phi\psi)$ heißt das durch *primitive Rekursion* aus ϕ und ψ gebildete Funktional.

6. Ist ϕ ein $(n+1)$-stelliges μ-partiellrekursives Funktional, so ist $(\mu\phi)$ ein n-stelliges μ-partiellrekursives Funktional.
   $(\mu\phi)$ heißt das durch *Minimalisierung* (oder mit dem *μ-Operator*) aus ϕ gebildete Funktional.

Definition 29 (Wert eines μ-partiellrekursiven Funktionals) Wir definieren induktiv, was es heißt, dass ein n-stelliges μ-partiellrekursives Funktional ϕ an einer Stelle $\mathfrak{x} \in \mathbb{N}^n$ den *Wert* $y \in \mathbb{N}$ hat, in Zeichen $\mathfrak{x}\ W[\phi]\ y$. Wir definieren also induktiv eine zweistellige Relation $W[\phi] \subseteq \mathbb{N}^n \times \mathbb{N}$ für jedes n-stellige μ-partiellrekursive Funktional ϕ.

1. Es gelte $()\ W[\mathtt{O}]\ 0$.

2. Wenn $x \in \mathbb{N}$, dann gelte $(x)\ W[\mathtt{N}]\ x'$.

3. Wenn $i, n \in \mathbb{N}$ mit $1 \leq i \leq n$ und $x_1, \ldots, x_n \in \mathbb{N}$, dann $(x_1, \ldots, x_n)\ W[\mathtt{P}_i^n]\ x_i$.

4. Wenn ϕ ein n-stelliges μ-partiellrekursives Funktional ist, $\psi_1, \ldots, \psi_m$ n-stellige μ-partiellrekursive Funktionale sind, $\mathfrak{x}\ W[\psi_1]\ y_1$, $\ldots$, $\mathfrak{x}\ W[\psi_m]\ y_m$ gelten und $(y_1, \ldots, y_m)\ W[\phi]\ z$ gilt, dann gelte $\mathfrak{x}\ W[(\phi\psi_1 \ldots \psi_m)]\ z$.

5. Wenn ϕ ein n-stelliges μ-partiellrekursives Funktional ist, ψ ein $(n+2)$-stelliges μ-partiellrekursives Funktional ist und $\mathfrak{x}\ W[\phi]\ z$, dann gelte $(\mathfrak{x}, 0)\ W[(\mathtt{R}\phi\psi)]\ z$.

6. Wenn ϕ ein n-stelliges μ-partiellrekursives Funktional ist, ψ ein $(n+2)$-stelliges μ-partiellrekursives Funktional ist, $(\mathfrak{x}, y)\ W[(\mathtt{R}\phi\psi)]\ u$ gilt und $(u, \mathfrak{x}, y)\ W[\psi]\ z$ gilt, dann gelte $(\mathfrak{x}, y')\ W[(\mathtt{R}\phi\psi)]\ z$.

7. Wenn ϕ ein $(n+1)$-stelliges μ-partiellrekursives Funktional ist, $(\mathfrak{x}, z)\ W[\phi]\ u_z$ für alle $z < y$ gilt, wobei $u_z > 0$ für alle $z < y$ ist, und $(\mathfrak{x}, y)\ W[\phi]\ 0$, dann gelte $\mathfrak{x}\ W[(\mu\phi)]\ y$.

Die so definierte Relation $W[\phi] \subseteq \mathbb{N}^n \times \mathbb{N}$ ist rechtseindeutig, d.h. es gilt $\mathfrak{x}\ W[\phi]\ y \wedge \mathfrak{x}\ W[\phi]\ z \Longrightarrow y = z$.

Aufgabe 13 Zeigen Sie dies durch Wertverlaufsinduktion nach der Länge von ϕ, mit geschachtelter vollständiger Induktion nach der letzten Komponente von $\mathfrak{x}$ im Falle der primitiven Rekursion.

Die Relation $W[\phi] \subseteq \mathbb{N}^n \times \mathbb{N}$ ist im Allgemeinen nicht linkstotal, d.h. im Allgemeinen gilt nicht, dass für jedes $\mathfrak{x} \in \mathbb{N}^n$ ein $y \in \mathbb{N}$ existiert mit $\mathfrak{x}\ W[\phi]\ y$. Zum Beispiel ist $(\mu\mathtt{N})$ ein nullstelliges μ-partiellrekursives Funktional mit $W[(\mu\mathtt{N})] = \emptyset$.

Es folgt, dass für ein n-stelliges μ-partiellrekursives Funktional ϕ die Relation $W[\phi]$ der Graph einer n-stelligen partiellen zahlentheoretischen Funktion ist, aber im Allgemeinen nicht der Graph einer n-stelligen (totalen) zahlentheoretischen Funktion.

Definition 30 (μ-partiellrekursive Funktionen) Wenn ϕ ein n-stelliges μ-partiellrekursives Funktional ist, dann nennt man die n-stellige partielle Funktion $f\colon \mathbb{N}^n \xrightarrow{\text{part}} \mathbb{N}$ mit $\text{Graph}(f) = W[\phi]$, deren Graph die oben definierte Relation $W[\phi]$ ist, die durch das μ-partiellrekursive Funktional ϕ *bezeichnete* partielle Funktion. Eine partielle zahlentheoretische Funktion $f\colon \mathbb{N}^n \xrightarrow{\text{part}} \mathbb{N}$ heißt *μ-partiellrekursiv*, wenn sie durch ein μ-partiellrekursives Funktional bezeichnet wird.

Ähnlich wie bei den primitivrekursiven Funktionalen und Funktionen verwenden wir hier den Schreibmaschinenzeichensatz für μ-partiellrekursive Funktionale und den Mathematikzeichensatz für die dadurch bezeichneten Funktionen.

Definition 31 Zusätzlich zu den bereits definierten Grundfunktionen Nullfunktion, Nachfolgerfunktion und Projektionsfunktionen definieren wir also

4. Ist f eine m-stellige partielle zahlentheoretische Funktion und sind $g_1, \ldots, g_m$ n-stellige partielle zahlentheoretische Funktionen, so ist die n-stellige partielle zahlentheoretische Funktion $(fg_1 \ldots g_m)$ definiert durch

   $$(fg_1 \ldots g_m)(\mathfrak{x}) := f(g_1(\mathfrak{x}), \ldots, g_m(\mathfrak{x})),$$

   falls $g_1(\mathfrak{x})$, $\ldots$, $g_m(\mathfrak{x})$ definiert sind und $f(g_1(\mathfrak{x}), \ldots, g_m(\mathfrak{x}))$ definiert ist. Sonst ist $(fg_1 \ldots g_m)(\mathfrak{x})$ undefiniert.
   Sie heißt die *Komposition* von f mit $g_1, \ldots, g_m$.

5. Ist f eine n-stellige partielle zahlentheoretische Funktion und g eine $(n+2)$-stellige partielle zahlentheoretische Funktion, so ist die $(n+1)$-stellige partielle zahlentheoretische Funktion (Rfg) definiert durch

   $$(Rfg)(\mathfrak{x}, 0) := f(\mathfrak{x}),$$

   falls $f(\mathfrak{x})$ definiert ist. Andernfalls ist $(Rfg)(\mathfrak{x}, 0)$ undefiniert. Weiter ist

   $$(Rfg)(\mathfrak{x}, y') := g((Rfg)(\mathfrak{x}, y), \mathfrak{x}, y),$$

   falls $(Rfg)(\mathfrak{x}, y)$ definiert ist und $g((Rfg)(\mathfrak{x}, y), \mathfrak{x}, y)$ definiert ist. Andernfalls ist $(Rfg)(\mathfrak{x}, y')$ undefiniert.
   Sie heißt die sich durch *primitive Rekursion* aus f und g ergebende Funktion.

6. Ist f eine $(n+1)$-stellige partielle zahlentheoretische Funktion, so ist die n-stellige partielle zahlentheoretische Funktion (μf) definiert durch

   $$(\mu f)(\mathfrak{x}) := \text{kleinstes } z \in \mathbb{N} \text{ mit } f(\mathfrak{x}, z) = 0,$$

   falls es so ein z gibt und außerdem $f(\mathfrak{x}, u)$ definiert ist für alle $u \leq z$. Andernfalls ist $(\mu f)(\mathfrak{x})$ undefiniert.

   Sie heißt die sich durch Anwendung des *μ-Operators* (oder durch *Minimalisierung*) aus f ergebende Funktion.

Die Menge der μ-partiellrekursiven Funktionen ist die kleinste Menge von partiellen zahlentheoretischen Funktionen, die die Nullfunktion, die Nachfolgerfunktion und die Projektionsfunktionen enthält und abgeschlossen ist gegenüber Komposition, primitiver Rekursion und μ-Operator.

Mit dem μ-Operator lässt sich die Wirkung einer Schleife in einem Computerprogramm nachvollziehen. So entspricht etwa einem Pseudocode

$$\mathfrak{x} := \mathfrak{a}; \quad \texttt{while } P(\mathfrak{x}) \texttt{ do } \mathfrak{x} := \mathfrak{f}(\mathfrak{x}); \quad \texttt{return } \mathfrak{x};$$

eine Minimalisierung ähnlich der von Aufgabe 12, wobei aber die beschränkte Minimalisierung durch eine unbeschränkte Minimalisierung ersetzt ist.

Definition 32 Unter einer *μ-rekursiven* Funktion versteht man eine totale μ-partiellrekursive Funktion.

Aufgabe 14 Zeigen Sie, dass die Petersche Funktion μ-rekursiv ist.
Hinweis: Wenn man den Wert der Peterschen Funktion p an einer Stelle ausrechnen will, muss man fortlaufend die Rekursionsgleichungen anwenden, etwa so:

$$\begin{aligned} p(1,2) &= p(0,p(1,1)) \\ &= p(0,p(0,p(1,0))) \\ &= p(0,p(0,p(0,1))) \\ &= p(0,p(0,2)) \\ &= p(0,3) \\ &= 4. \end{aligned}$$

Man kann den Status einer solchen Berechnung zu einem Zeitpunkt durch einen Kellerspeicher darstellen, der ein Tupel von Zahlen enthält. Im Beispiel sind die Zahltupel nacheinander $(1,2)$, $(0,1,1)$, $(0,0,1,0)$, $(0,0,0,1)$, $(0,0,2)$, $(0,3)$ und (4). Solche Zahlentupel kann man wieder mittels der Cantor'schen Tupelfunktion als natürliche Zahlen kodieren. Iteration und μ-Operator sind die weiteren benötigten Werkzeuge zur Lösung der Aufgabe.

2.3 Logikkalküle

2.3.1 Aussagenlogik

Die Sprache der Aussagenlogik

Grundzeichen:

1. Aussagenvariablen (P, Q, R,... auch mit Indizes, ...)
2. Junktoren:
   $\neg$ 'nicht'
   $\wedge$ 'und'
   $\vee$ 'oder'
3. Runde Klammern ( )

Die Menge der Aussagenvariablen bezeichnen wir mit $\mathbb{A}$.

Definition 33 (induktive Definition der *Formeln*) Formeln sind spezielle Zeichenreihen aus Grundzeichen, und zwar

1. Jede Aussagenvariable ist eine Formel.[1]
2. Wenn F eine Formel ist, dann ist auch $\neg F$ eine Formel.
3. Wenn F und G Formeln sind, dann sind auch $(F \wedge G)$ und $(F \vee G)$ Formeln.

Wenn F und G Formeln sind, dann nennen wir die Formel $\neg F$ die *Negation* von F, die Formel $(F \wedge G)$ die *Konjunktion* von F und G und die Formel $(F \vee G)$ die *Disjunktion* der Formeln F und G.

Wenn wir über Formeln reden, lassen wir meist Klammern weg, wo dies nicht zu Mißverständnissen führt. Für Formeln verwenden wir Zeichen F, G, H, ..., auch mit Indizes, ...

Definition 34 (induktive Definition der *Teilformeln* einer Formel)

1. F ist eine Teilformel von F.
2. Wenn $\neg G$ eine Teilformel von F ist, dann ist auch G eine Teilformel von F.
3. Wenn $(G \wedge H)$ eine Teilformel von F ist, dann sind auch G und H Teilformeln von F.
4. Wenn $(G \vee H)$ eine Teilformel von F ist, dann sind auch G und H Teilformeln von F.

Semantik

Die Semantik ist die Lehre von der Bedeutung. In der mathematischen Logik interessiert man sich insbesondere dafür, wann eine Formel wahr ist und wann sie falsch ist. Hierzu betrachten wir die Menge der Wahrheitswerte (auch Boole'sche Werte genannt)

$$\mathbb{B} = \{\text{wahr}, \text{falsch}\}.$$

Definition 35 Eine *Variablenbelegung* (oder *Interpretation*) ist eine Funktion $i\colon \mathbb{A} \to \mathbb{B}$.

Definition 36 des *Wahrheitswertes* iF einer Formel F bei einer Variablenbelegung i.

1. $iP := i(P)$ für jede Aussagenvariable P.
2. $i\neg F := \begin{cases} \text{wahr}, & \text{wenn } iF = \text{falsch} \\ \text{falsch} & \text{sonst} \end{cases}$
3. $i(F \wedge G) := \begin{cases} \text{wahr}, & \text{wenn } iF = \text{wahr und } iG = \text{wahr} \\ \text{falsch} & \text{sonst} \end{cases}$
4. $i(F \vee G) := \begin{cases} \text{wahr}, & \text{wenn } iF = \text{wahr oder } iG = \text{wahr} \\ \text{falsch} & \text{sonst} \end{cases}$

Das 'oder' ist ein nicht ausschließendes 'oder', d.h., wenn iF und iG beide = wahr sind, dann ist auch $i(F \vee G) =$ wahr. Für jede Variablenbelegung i und jede Formel F gilt $iF \in \mathbb{B}$.

[1] Wir identifizieren hier eine Zeichenreihe, die aus nur einem Zeichen besteht, mit diesem einen Zeichen.

Definition 37 Ein *Modell* einer Formel F ist eine Variablenbelegung i so, daß $iF =$ wahr ist.

Definition 38 Eine Formel F heißt *allgemeingültig*, wenn für jede Variablenbelegung i gilt $iF =$ wahr. Eine Formel F heißt *erfüllbar*, wenn es eine Variablenbelegung i gibt mit $iF =$ wahr (also genau dann, wenn sie ein Modell hat). Andernfalls heißt sie *unerfüllbar*.

Definition 39 Wir sagen, eine Formel G *folge semantisch* aus einer Formel F (in Zeichen: $F \models G$), wenn jedes Modell von F auch Modell von G ist. Zwei Formeln F und G heißen *semantisch äquivalent* (in Zeichen: $F \sim G$), wenn $iF = iG$ ist für alle Variablenbelegungen i (also genau dann, wenn sie die gleichen Modelle haben).

Die Relation $\sim$ ist eine Äquivalenzrelation auf der Menge der Formeln.

Definition 40 Sei S eine Menge von Formeln. Dann heißt i ein *Modell* von S, wenn i Modell jeder Formel $F \in S$ ist. Wenn S eine Formelmenge und G eine Formel ist, dann sagen wir, G *folge semantisch* aus S (in Zeichen: $S \models G$), wenn jedes Modell von S ein Modell von G ist. Wir sagen, eine Formelmenge S sei *erfüllbar*, wenn sie ein Modell hat. Andernfalls heißt sie *unerfüllbar*.

Beispiel 26
Sei $\mathbb{A} := \{P, Q\}$ und $F :\equiv P \wedge \neg Q$. Diese Formel F hat genau ein Modell i, das gegeben ist durch

$$\begin{aligned} i(P) &= \text{wahr} \\ i(Q) &= \text{falsch.} \end{aligned}$$

Denn wenn eine Variablenbelegung i ein Modell von F sein soll, dann muß $iF =$ wahr sein. Also $i(P \wedge \neg Q) =$ wahr. Also gilt nach Definition 36 $iP =$ wahr und $i\neg Q =$ wahr. Also gilt weiter nach Definition 36 $iQ =$ falsch und damit $i(P) =$ wahr und $i(Q) =$ falsch.

Umgekehrt sei $i(P) =$ wahr und $i(Q) =$ falsch. Dann ist nach Definition 36 $iP =$ wahr und $iQ =$ falsch und damit $i\neg Q =$ wahr und folglich $i(P \wedge \neg Q) =$ wahr.

Beispiel 27
Sei $\mathbb{A} := \{P, Q\}$ und $F :\equiv P \vee \neg Q$. Dann hat F genau 3 Modelle i_1, i_2 und i_3, die gegeben sind durch

$$\begin{aligned} i_1(P) &= \text{wahr} \\ i_1(Q) &= \text{wahr} \end{aligned}$$

$$\begin{aligned} i_2(P) &= \text{wahr} \\ i_2(Q) &= \text{falsch} \end{aligned}$$

$$\begin{aligned} i_3(P) &= \text{falsch} \\ i_3(Q) &= \text{falsch} \end{aligned}$$

In jedem der beiden Beispiele ist die Formel F erfüllbar, aber nicht allgemeingültig.

Beispiel 28
Sei $\mathbb{A} := \{P\}$ und $F :\equiv P \wedge \neg P$. Dann hat F kein Modell und ist daher unerfüllbar.

Beispiel 29
Sei $\mathbb{A} := \{P\}$ und $F :\equiv P \vee \neg P$. Dann ist jede Variablenbelegung über der Menge $\mathbb{A}$ der Aussagenvariablen ein Modell von F. Damit ist F allgemeingültig.

Satz 15 *Für alle Formeln F, G und H gelten die folgenden semantischen Äquivalenzen.*

$F \wedge G \sim G \wedge F$	*Kommutativität von* $\wedge$
$F \vee G \sim G \vee F$	*Kommutativität von* $\vee$
$\neg\neg F \sim F$	
$(F \wedge G) \wedge H \sim F \wedge (G \wedge H)$	*Assoziativität von* $\wedge$
$(F \vee G) \vee H \sim F \vee (G \vee H)$	*Assoziativität von* $\vee$
$F \wedge (G \vee H) \sim (F \wedge G) \vee (F \wedge H)$	*Distributivgesetz*
$F \vee (G \wedge H) \sim (F \vee G) \wedge (F \vee H)$	*Distributivgesetz*
$F \wedge F \sim F$	
$F \vee F \sim F$	
$\neg(F \wedge G) \sim \neg F \vee \neg G$	*DeMorgan'sche Regel*
$\neg(F \vee G) \sim \neg F \wedge \neg G$	*DeMorgan'sche Regel*

Satz 16 *Sei F eine Formel und sei G eine Teilformel von F. Weiter sei G' eine Formel mit $G \sim G'$ und F' sei aus F entstanden durch Ersetzung eines Vorkommens der Teilformel G durch die Formel G'. Dann ist $F \sim F'$.*

Beispiel 30
Seien G und H zwei Formeln und sei $F :\equiv G \vee H$. Da $G \sim \neg\neg G$ gilt, läßt sich der Satz anwenden, wenn man $G' :\equiv \neg\neg G$ und $F' :\equiv \neg\neg G \vee H$ setzt. Es ist also $G \vee H \sim \neg\neg G \vee H$.

Satz 17 *Sei F eine Formel. Dann gilt:*

$$F \text{ ist allgemeingültig} \iff \neg F \text{ ist unerfüllbar}$$
$$F \text{ ist unerfüllbar} \iff \neg F \text{ ist allgemeingültig}$$

Satz 18 *Sei S eine Formelmenge und F eine Formel. Dann gilt:*

$$S \models F \iff S \cup \{\neg F\} \text{ ist unerfüllbar.}$$

Dieser Satz, der auch in der später in der Vorlesung einzuführenden Erweiterung der Aussagenlogik, nämlich in der Prädikatenlogik, gilt, ist ein für die logische Programmierung grundlegender Satz. Denn eine Klausel eines Prologprogramms kann man als Formel auffassen, das Prologprogramm selbst als Menge S von Formeln. Die Anfrage kann man als Formel F auffassen. Die Aufgabe von Prolog ist es dann zu beweisen, daß $S \models F$ gilt. Prolog tut dies, indem es aus $S \cup \{\neg F\}$ einen Widerspruch ableitet, also indem es zeigt, daß diese Menge unerfüllbar ist. Nach dem Satz muß daher $S \models F$ gelten.

Normalformen

Definition 41 Eine Formel heißt ein *Literal*, wenn sie eine Aussagenvariable P oder die Negation $\neg P$ einer Aussagenvariablen P ist.

Definition 42 der Formeln in *Negations-Normalform* (induktive Definition)

1. Jedes Literal ist in Negations-Normalform.

2. Wenn F und G Formeln in Negations-Normalform sind, dann sind auch die Formeln $F \wedge G$ und $F \vee G$ in Negations-Normalform.

Eine derartige induktive Definition ist so zu verstehen, daß eine Formel genau dann in Negations-Normalform ist, wenn sich durch endlich häufige Anwendung der beiden in der Definition formulierten Regeln ableiten läßt, daß sie in Negations-Normalform ist. Man kann das auch mengentheoretisch folgendermaßen formulieren.

Definition 43 der Formeln in *Negations-Normalform* (alternative Definition). Die Menge der Formeln in Negations-Normalform ist die kleinste Menge S mit

1. Jedes Literal ist Element von S.

2. $F, G \in S \Longrightarrow F \wedge G, F \vee G \in S$.

Für 'Negations-Normalform' schreiben wir auch kurz 'NNF'.

Satz 19 *Zu jeder Formel F gibt es eine Formel F' in NNF mit $F \sim F'$.*

Beweis: durch Induktion nach der Länge von F.
Induktionsvoraussetzung: Zu jeder Formel G, die kürzer als F ist, existiere eine Formel G' in NNF mit $G \sim G'$.
Induktionsbehauptung: Es gibt eine Formel F' in NNF mit $F \sim F'$.
Beweis (Induktionsschritt): Wir machen eine Fallunterscheidung.

1. F ist eine Aussagenvariable P. Dann sei $F' :\equiv P$.

2. F ist die Negation einer Formel. In diesem Fall müssen wir wieder vier Unterfälle unterscheiden.

   (a) $F \equiv \neg P$. Dann sei $F' :\equiv \neg P$.

   (b) $F \equiv \neg\neg G$. Dann ist G kürzer als F. Daher existiert nach Induktionsvoraussetzung ein G' in NNF mit $G \sim G'$. Nun ist $F \sim G$, also $F \sim G'$. Sei nun $F' :\equiv G'$.

   (c) $F \equiv \neg(G \wedge H)$. Dann sind $\neg G$ und $\neg H$ kürzer als F. Nach Induktionsvoraussetzung gibt es also Formeln I und J in NNF mit $\neg G \sim I$ und $\neg H \sim J$. Daher ist

   $$F \sim \neg G \vee \neg H \sim I \vee \neg H \sim I \vee J.$$

   Sei nun $F' :\equiv I \vee J$. Dann ist $F \sim F'$, und F' ist in NNF.

   (d) $F \equiv \neg(G \vee H)$. Für diesen Fall geht der Beweis analog.

3. $F \equiv G \wedge H$. Dann sind G und H kürzer als F. Nach Induktionsvoraussetzung existieren Formeln G' und H' in NNF mit $G \sim G'$ und $H \sim H'$. Also

   $$F \equiv G \wedge H \sim G' \wedge H \sim G' \wedge H'.$$

   Hiermit ist $F \sim G' \wedge H'$. Sei nun $F' :\equiv G' \wedge H'$. Dann ist F' in NNF, da G' und H' in NNF sind. Außerdem ist $F \sim F'$.

4. $F \equiv G \vee H$. In diesem Fall erfolgt der Beweis analog.

q.e.d.

Definition 44 der *Konjunktion* $F_1 \wedge \cdots \wedge F_n$ von Formeln $F_1, \ldots, F_n$.

1. Die Konjunktion von F_1 ist F_1.
2. $F_1 \wedge \cdots \wedge F_{n+1} :\equiv (F_1 \wedge \cdots \wedge F_n) \wedge F_{n+1}$.

Definition 45 der *Disjunktion* $F_1 \vee \cdots \vee F_n$ von Formeln $F_1, \ldots, F_n$.

1. Die Disjunktion von F_1 ist F_1.
2. $F_1 \vee \cdots \vee F_{n+1} :\equiv (F_1 \vee \cdots \vee F_n) \vee F_{n+1}$.

Definition 46 Eine Formel F ist in *konjunktiver Normalform* (*KNF*), wenn sie eine Konjunktion von Disjunktionen von Literalen ist.

Definition 47 Eine Formel F ist in *disjunktiver Normalform* (*DNF*), wenn sie eine Disjunktion von Konjunktionen von Literalen ist.

Beispiel 31

$(P \vee \neg Q) \wedge (\neg P \vee Q) \wedge \neg R$	ist in KNF.
$(P \wedge \neg Q \wedge \neg P) \vee Q$	ist in DNF.
$(P \vee Q) \wedge (\neg P \vee Q)$	ist in KNF.
$P \wedge Q \wedge \neg R$	ist in KNF und in DNF.
$P \vee Q$	ist in KNF und in DNF.
$\neg P$	ist in KNF und in DNF.

Jede Formel in KNF oder DNF ist in NNF.

Lemma 6 *Zu jeder Formel F in NNF gibt es eine Formel F' in KNF mit $F \sim F'$.*

Beweis: durch Induktion nach der Länge von F.
Induktionsvoraussetzung: Die Behauptung gelte für alle Formeln in NNF, die kürzer als F sind.
Induktionsbehauptung: Die Behauptung gilt für F.
Beweis (Induktionsschritt): Wir machen eine Fallunterscheidung.

1. F ist ein Literal. Dann sei $F' :\equiv F$.
2. $F \equiv G \wedge H$. Dann sind G und H kürzer als F und ebenfalls in NNF. Nach Induktionsvoraussetzung gibt es ein $G' \equiv G_1 \wedge \cdots \wedge G_m$ in KNF, wobei jedes G_i eine Disjunktion von Literalen ist, mit $G \sim G'$. Analog gibt es ein $H' \equiv H_1 \wedge \cdots \wedge H_n$ in KNF, wobei jedes H_j eine Disjunktion von Literalen ist, mit $H \sim H'$. Nun gilt

$$F \equiv G \wedge H \sim G' \wedge H' \sim G_1 \wedge \cdots \wedge G_m \wedge H_1 \wedge \cdots \wedge H_n.$$

Sei nun $F' :\equiv G_1 \wedge \cdots \wedge G_m \wedge H_1 \wedge \cdots \wedge H_n$. Dann ist F' in KNF und $F \sim F'$.

3. $F \equiv G \vee H$. Wie oben bekommen wir wieder

$$\begin{array}{ll} G' \equiv G_1 \wedge \cdots \wedge G_m & G \sim G' \\ H' \equiv H_1 \wedge \cdots \wedge H_n & H \sim H' \end{array}$$

Hieraus ergibt sich

$$\begin{array}{rcl} F & \equiv & G \vee H \\ & \sim & G' \vee H' \\ & \equiv & (G_1 \wedge \cdots \wedge G_m) \vee (H_1 \wedge \cdots \wedge H_n) \\ & \sim & (G_1 \vee H_1) \wedge (G_1 \vee H_2) \wedge \cdots \wedge (G_1 \vee H_n) \wedge \\ & & (G_2 \vee H_1) \wedge (G_2 \vee H_2) \wedge \cdots \wedge (G_2 \vee H_n) \wedge \\ & & \vdots \\ & & (G_m \vee H_1) \wedge (G_m \vee H_2) \wedge \cdots \wedge (G_m \vee H_n) \end{array}$$

Letztere semantische Äquivalenz ergibt sich durch die wiederholte Anwendung eines der beiden Distributivgesetze sowie des Assoziativgesetzes für $\wedge$. Jedes G_i und jedes H_j ist eine Disjunktion von Literalen. Durch wiederholte Anwendung des Assoziativgesetzes für $\vee$ kann man also $G_i \vee H_j$ in eine Disjunktion I_{ij} von Literalen umwandeln, d.h. $G_i \vee H_j \sim I_{ij}$. Es folgt, daß F semantisch äquivalent ist zu der folgenden Formel F'.

$$\begin{array}{c} I_{11} \wedge I_{12} \wedge \cdots \wedge I_{1n} \wedge \\ I_{21} \wedge I_{22} \wedge \cdots \wedge I_{2n} \wedge \\ \vdots \\ I_{m1} \wedge I_{m2} \wedge \cdots \wedge I_{mn} \end{array}$$

Außerdem ist die so definierte Formel F' ist in KNF.

q.e.d.

Satz 20 *Zu jeder Formel F gibt es eine Formel F' in KNF mit $F \sim F'$.*

Beweis: Nach Satz 19 gibt es zu F eine semantisch äquivalente Formel G in NNF. Nach dem Lemma gibt es zu G eine semantisch äquivalente Formel F' in KNF. Also $F \sim G$ und $G \sim F'$ und somit $F \sim F'$. q.e.d.

Satz 21 *Zu jeder Formel F gibt es eine Formel F' in DNF mit $F \sim F'$.*

Beweis: analog. q.e.d.

Man kann aber diesen Satz auch anders beweisen. Die Idee ist die folgende. Man bestimmt zunächst alle Modelle der Formel F, z.B. durch eine Wahrheitswerttabelle. Als Beispiel hierzu betrachten wir die folgende Formel F.

$$\neg(\neg P \vee Q) \vee \neg P.$$

Dazu stellt man eine Tabelle der Wahrheitswerte aller ihrer Teilformeln für alle Variablenbelegungen auf. Jede Zeile der folgenden Tabelle entspricht einer Variablenbelegung.

P	Q	$\neg P$	$\neg P \vee Q$	$\neg(\neg P \vee Q)$	$\neg(\neg P \vee Q) \vee \neg P$
wahr	wahr	falsch	wahr	falsch	falsch
wahr	falsch	falsch	falsch	wahr	wahr
falsch	wahr	wahr	wahr	falsch	wahr
falsch	falsch	wahr	wahr	falsch	wahr

Nun ist eine Variablenbelegung i genau dann ein Modell der Formel F, wenn in dieser Wahrheitswerttabelle am Schnittpunkt der zu i gehörigen Zeile mit der letzten Spalte der Eintrag 'wahr' steht. Da im Beispiel in der letzten Spalte genau dreimal der Eintrag 'wahr' steht, hat F genau drei Modelle. Das erste dieser Modelle ist dadurch definiert, daß bei ihm P wahr ist und Q falsch ist, oder anders ausgedrückt, daß bei ihm $P \wedge \neg Q$ wahr ist. Das zweite Modell ist dadurch gekennzeichnet, daß bei ihm $\neg P \wedge Q$ wahr ist und das dritte dadurch, daß bei ihm $\neg P \wedge \neg Q$ wahr ist. Insgesamt ist also eine Variablenbelegung i genau dann ein Modell von F, wenn bei i die folgende Formel F'

$$(P \wedge \neg Q) \vee (\neg P \wedge Q) \vee (\neg P \wedge \neg Q)$$

wahr wird. Also ist $F \sim F'$ und offensichtlich ist F' in DNF.

Formeln in KNF spielen in der logischen Programmierung eine zentrale Rolle. Betrachten wir etwa das Prologprogramm

```
p :- q, r.
q.
r.
```

und dazu die Anfrage

```
?- p.
```

Jede der Klauseln des Programms kann man als Formel schreiben. Die erste Programmklausel entspricht einer logischen Implikation $\mathtt{q} \wedge \mathtt{r} \rightarrow \mathtt{p}$. Diese Implikation kann man als Abkürzung auffassen für die Formel $\neg(\mathtt{q} \wedge \mathtt{r}) \vee \mathtt{p}$, die nach einer der DeMorgan'schen Regeln semantisch äquivalent ist zu $\neg\mathtt{q} \vee \neg\mathtt{r} \vee \mathtt{p}$, also auch zu $\mathtt{p} \vee \neg\mathtt{q} \vee \neg\mathtt{r}$. Insgesamt entsprechen die drei Programmklauseln den folgenden drei Formeln.

$$\mathtt{p} \vee \neg\mathtt{q} \vee \neg\mathtt{r}$$
$$\mathtt{q}$$
$$\mathtt{r}$$

Das Programm entspricht also der Formelmenge S, die aus diesen drei Formeln besteht. Die Anfrage entspricht einer Formel F. Im Beispiel ist $F \equiv \mathtt{p}$. Prolog muß versuchen nachzuweisen, daß $S \models F$ gilt. Nach Satz 18 gilt dies genau dann, wenn $S \cup \{\neg F\}$ unerfüllbar ist. Genau die Unerfüllbarkeit dieser Formelmenge bestehend aus den Formeln

$$\mathtt{p} \vee \neg\mathtt{q} \vee \neg\mathtt{r}$$
$$\mathtt{q}$$
$$\mathtt{r}$$
$$\neg\mathtt{p}$$

wird von Prolog versucht nachzuweisen. Jede dieser Formeln ist eine Disjunktion von Literalen. Die Formelmenge ist genau dann unerfüllbar, wenn die Konjunktion

$$(\mathtt{p} \vee \neg \mathtt{q} \vee \neg \mathtt{r}) \wedge \mathtt{q} \wedge \mathtt{r} \wedge \neg \mathtt{p}$$

dieser Formeln unerfüllbar ist. Diese Formel ist aber in konjunktiver Normalform. Prolog ist ein automatisches Beweissystem, das gewisse Formeln in KNF auf Unerfüllbarkeit testet.

Klauseln in der Logik

In der Logik wird eine Disjunktion von Literalen dargestellt als Menge dieser Literale. So wird die Formel $P \vee \neg Q \vee \neg R$ dargestellt als die Menge $\{P, \neg Q, \neg R\}$. So eine Menge wird in der Logik *Klausel* genannt.

Definition 48 Eine *Klausel* in der Logik ist eine endliche Menge von Literalen.

Definition 49 Eine Klausel ist *wahr* bei einer Variablenbelegung i genau dann, wenn mindestens eines ihrer Literale wahr ist bei i. Andernfalls ist sie *falsch* bei i.

Eine Disjunktion D von Literalen ist bei einer Variablenbelegung i genau dann wahr, wenn die zu D gehörige Klausel bei der Variablenbelegung i wahr ist.

Analog wie für Formeln oder Formelmengen sind für Klauseln die semantischen Begriffe *Modell*, *Erfüllbarkeit*, *Unerfüllbarkeit*, *semantische Folgerung*, etc. definiert.

Nach unserer Definition ist auch die leere Menge als Klausel erlaubt. Sie wird mit $\square$ bezeichnet. Die leere Klausel ist bei jeder Variablenbelegung falsch.

Resolution

Resolution allgemein Der Resolutionskalkül ist ein Kalkül zum automatischen Nachweis der Unerfüllbarkeit einer Menge S von Klauseln. Hierzu werden, ausgehend von S solange weitere Klauseln, die semantisch aus S folgen, mit der sogenannten *Resolutionsregel* abgeleitet, bis die leere Klausel und damit ein Widerspruch abgeleitet ist.

Definition 50 Zwei Literale heißen zueinander *komplementär*, wenn eines der beiden Literale die Negation des anderen ist. Das zu einem Literal L komplementäre Literal wird mit $\overline{L}$ bezeichnet. Es gilt also $\overline{L} \equiv \neg L$ oder $L \equiv \neg \overline{L}$.

Definition 51 (*Resolutionsregel*):
Seien c und d zwei Klauseln und sei L ein Literal mit $L \in c$ und $\overline{L} \in d$. Dann heißt die Klausel

$$(c \setminus \{L\}) \cup (d \setminus \{\overline{L}\})$$

eine *Resolvente* von c und d. Die Klauseln c und d heißen *Elternklauseln*. Die Literale L und $\overline{L}$ in den Klauseln c bzw. d heißen die *wegresolvierten* Literale. Eine Menge S' von Klauseln ist durch einen *Resolutionsschritt* aus einer Menge S von Klauseln entstanden, wenn $S' = S \cup \{e\}$ ist, wobei e eine Resolvente zweier Klauseln $c, d \in S$ ist. Eine *Resolutionsableitung* aus einer Klauselmenge S ist eine Folge $(c_1, c_2, c_3, \dots)$ von Klauseln so, daß jedes c_k ein Element von S ist oder eine Resolvente von c_i und c_j mit $i, j < k$. Eine *Resolutionsableitung* einer Klausel c aus S ist eine Resolutionsableitung $(c_1, \dots, c_n)$ aus S, die mit c endet (also mit $c_n = c$). Eine *Resolutionswiderlegung* einer Klauselmenge S ist eine Resolutionsableitung von $\square$ aus S.

Eine Klauselmenge S heißt mit Resolution *widerlegbar* oder *resolutionswiderlegbar*, wenn es eine Resolutionswiderlegung von S gibt. Die Resolutionsregel schreiben wir auch so:

$$\frac{c \qquad d}{(c \setminus \{L\}) \cup (d \setminus \{\overline{L}\})} \quad , \quad \text{wenn } L \in c \text{ und } \overline{L} \in d.$$

Die Resolutionsregel erlaubt aus zwei Klauseln eine neue abzuleiten. Bei Regeln allgemein spricht man davon, daß man aus den *Prämissen* die *Konklusion* ableitet. Speziell bei der Resolutionsregel werden die Prämissen *Elternklauseln* genannt und die Konklusion wird *Resolvente* genannt.

Lemma 7 *Seien c und d zwei Klauseln und sei e eine Resolvente von c und d. Dann gilt $\{c, d\} \models e$.*

Beweis: Sei i ein Modell von $\{c, d\}$. Dann sind c und d bei i wahr. Es gibt also Literale $J \in c$ und $K \in d$ so, daß J und K wahr sind bei der Variablenbelegung i. Nach Voraussetzung ist e eine Resolvente von c und d. Es gibt also ein Literal L so, daß $L \in c$ und $\overline{L} \in d$ ist und daß $e = (c \setminus \{L\}) \cup (d \setminus \{\overline{L}\})$ gilt. Wir unterscheiden nun drei (sich nicht gegenseitig ausschließende) Fälle.

1. $J \not\equiv L$. Dann ist $J \in c \setminus \{L\}$, also $J \in e$. Damit ist e wahr bei der Variablenbelegung i, also i ein Modell von e.
2. $K \not\equiv \overline{L}$. Dann ist $K \in c \setminus \{\overline{L}\}$, also $K \in e$. Damit ist e wahr bei der Variablenbelegung i, also i ein Modell von e.
3. $J \equiv L$ und $K \equiv \overline{L}$. Dieser Fall kann nicht auftreten, da J und K beide wahr sind bei der Variablenbelegung i.

Also ist in jedem Fall i ein Modell von e. Damit gilt $\{c, d\} \models e$. q.e.d.

Satz 22 *(Korrektheitssatz): Sei S eine Klauselmenge, die mit Resolution widerlegbar ist. Dann ist S unerfüllbar.*

Beweis: Sei $(c_1, \ldots, c_n)$ mit $c_n = \square$ eine Resolutionswiderlegung von S. Angenommen, S wäre erfüllbar. Dann würde S ein Modell i haben. Aus dem Lemma folgt durch Induktion nach j, daß i ein Modell von jedem c_j ist $(j = 1, \ldots, n)$. Für $j := n$ würde dies bedeuten, daß i ein Modell der leeren Klausel $\square$ ist. Da wir wissen, daß die leere Klausel kein Modell hat, folgt ein Widerspruch. q.e.d.

Satz 23 *(Vollständigkeitssatz): Sei S eine unerfüllbare Klauselmenge. Dann besitzt S eine Resolutionswiderlegung.*

Eine Resolutionsableitung aus einer Formelmenge S läßt sich auch als knotenmarkierter Binärbaum darstellen. Dabei sind die Blätter mit Elementen aus S markiert und die Markierung eines jeden inneren Knotens ist die Resolvente der Markierungen seiner beiden Nachfolgerknoten. Der Baum wird üblicherweise so gezeichnet, daß die Wurzel unten ist und die Blätter oben sind.

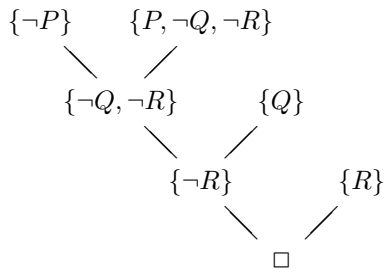
Beispiel 32
Sei

$$S = \{\{P, \neg Q, \neg R\}, \{Q\}, \{R\}, \{\neg P\}\}.$$

Dann ist

$$(\{\neg P\}, \{P, \neg Q\, \neg R\}, \{\neg Q, \neg R\}, \{Q\}, \{\neg R\}, \{R\}, \square)$$

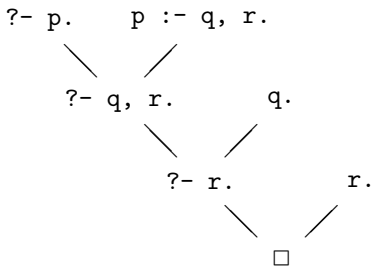
eine Resolutionswiderlegung von S. Diese Resolutionswiderlegung wird durch den folgenden Baum dargestellt.



Schreibt man die Klauseln in Prolog-Notation, so schaut die Klauslemenge folgendermaßen aus

```
p :- q, r.
q.
r.
?- p.
```

und der Widerlegungsbaum folgendermaßen.



Beispiel für Prologs Suche nach einer Resolutionswiderlegung Wenn man eine Anfrage an Prolog stellt, so wählt sich Prolog zunächst das erste Ziel der Anfrage aus und versucht eine Resolution mit einer Programmklausel, die dieses Ziel als Kopf hat. Prolog nimmt die erste Programmklausel, die paßt. Dadurch erzeugt Prolog eine neue Anfrage (die Resolvente), die es zu lösen versucht. Findet Prolog zu dieser neuen Anfrage keine Lösung (oder keine Lösung mehr, wenn es mit Strichpunkt aufgefordert wird mehrere Lösungen zu suchen), dann macht Prolog die Beweisschritte, die es seit der Resolution der ursprünglichen Anfrage mit der ersten passenden Programmklausel gemacht hat, wieder rückgängig (backtracking) und versucht eine Resolution mit der nächsten passenden Programmklausel. Wenn es keine passende Programmklausel mehr gibt, schlägt die Anfrage fehl und Prolog antwortet mit `No`.

Ein einfaches Beispiel soll dies demonstrieren. Gegeben sei das folgende Prologprogramm und die folgende Anfrage.

```
p :- q, r.
p :- s.
q.
?- p.
```

Prolog macht nun die folgenden Schritte

1. Die Anfrage (Zielklausel) '`?- p.`' wird resolviert mit der ersten Programmklausel '`p:- q, r.`' und als Resolvente ergibt sich die neue Zielklausel '`?- q, r.`'. Prolog merkt sich, mit welcher Programmklausel es resolviert hat.

2. Die Zielklausel '`?- q, r.`' wird resolviert mit der dritten Programmklausel '`q.`' und als Resolvente ergibt sich die neue Zielklausel '`?- r.`' Prolog merkt sich wieder, mit welcher Programmklausel es resolviert hat.

3. Die Zielklausel '`?- r.`' wird versucht mit einer der Programmklauseln zu resolvieren, was scheitert.

4. Prolog macht die Beweisschritte bis zu Schritt 2[2] rückgängig und sucht eine Alternative zu Schritt 2, also eine andere Möglichkeit die Zielklausel '`?- q, r.`' zu lösen (backtracking). Da Prolog in einer Zielklausel grundsätzlich das erste Literal zum Wegresolvieren auswählt, versucht es eine alternative Klausel mit dem Kopf `q` zu finden. Im Programm paßt da aber nur die dritte Programmklausel, von der Prolog sich gemerkt hat, daß es sie schon im Schritt 2 ausgewählt hat. Daher scheitert der Versuch die Zielklausel '`?- q, r.`' zu lösen.

5. Prolog macht die Beweisschritte bis zu Schritt 1 rückgängig und versucht zu Schritt 1 eine Alternative, also eine andere Möglichkeit die Zielklausel '`?- p.`' zu lösen (backtracking). Prolog hat sich in Schritt 1 gemerkt, daß es schon versucht hat diese Zielklausel mit der ersten Programmklausel zu resolvieren. Daher resolviert es die Zielklausel '`?- p.`' diesmal mit der zweiten Programmklausel '`p :- s.`' und als Resolvente ergibt sich die neue Zielklausel '`?- s.`'. Prolog merkt sich wieder, mit welcher Programmklausel es resolviert hat.

6. Prolog versucht die Zielklausel '`?- s.`' mit einer der Programmklauseln zu resolvieren, was scheitert.

7. Prolog macht die Beweisschritte bis zu Schritt 5 rückgängig und sucht eine Alternative zu Schritt 5, also eine andere Möglichkeit die Zielklausel '`?- p.`' zu lösen (backtracking). Prolog hat sich in Schritt 5 gemerkt, daß es dort versucht hat mit der zweiten Programmklausel zu resolvieren und da es keine weitere Programmklausel mit dem Kopf `p` mehr gibt, scheitert der Versuch die Zielklausel '`?- p.`' zu lösen.

8. Damit ist die ursprüngliche Anfrage gescheitert und Prolog antwortet mit `No`.

Hornklauseln und SLD-Resolution

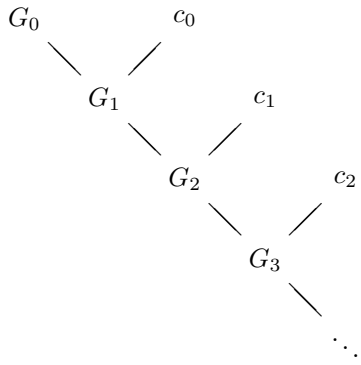
Definition 52 Ein *positives Literal* ist eine Aussagenvariable (also ein Literal, das kein Negationszeichen $\neg$ enthält). Ein *negatives Literal* ist die Negation einer Aussagenvariablen (also ein Literal, das das Negationszeichen $\neg$ enthält).

[2] bis zum letzten Schritt, bei dem es möglicherweise eine alternative Wahlmöglichkeit gegeben hätte

Definition 53 Eine *Hornklausel* ist eine Klausel, die höchstens ein positives Literal enthält. Eine *Programmklausel* ist eine Klausel, die genau ein positives Literal enthält. Eine *Anfrageklausel* oder *Zielklausel* (englisch *query clause* oder *goal clause*) ist eine Klausel, die kein positives Literal enthält.

Jede Hornklausel ist entweder eine Programmklausel oder eine Zielklausel. Prolog arbeitet mit Hornklauseln. Wir werden für Hornklauseln oft Prolognotation verwenden.

Definition 54 *SLD-Resolution* ist Resolution für Hornklauseln mit folgenden Einschränkungen: $S = P \cup \{G\}$. P ist eine Menge von Programmklauseln. G ist eine Zielklausel. Eine SLD-Resolutionsableitung hat die Form



wobei $G_0 \equiv G$ ist und $c_0, c_1, c_2, \cdots \in P$ sind. Dabei muß jeder Resolutionsschritt die Form haben.

$$\frac{\texttt{?-}\ A_1, \ldots, A_j, \ldots, A_m. \qquad\qquad A_j\ \texttt{:-}\ B_1, \ldots, B_n.}{\texttt{?-}\ A_1, \ldots, A_{j-1}, B_1, \ldots, B_n, A_{j+1}, \ldots, A_m.}$$

Das Literal A_j heißt dabei das *ausgewählte Literal*. Es wird dabei angenommen, daß eine *Auswahlfunktion* (englisch: *selection funktion*) gegeben sei, die zu einer Zielklausel das ausgewählte Literal findet.

Prolog wählt eimmer das jeweils erste Literal einer Zielklausel aus (es ist also $j = 1$). Den Resolutionsschritt kann man dann auch so schreiben.

$$\frac{\texttt{?-}\ A_1, \ldots, A_m. \qquad\qquad A_1\ \texttt{:-}\ B_1, \ldots, B_n.}{\texttt{?-}\ B_1, \ldots, B_n, A_2, \ldots, A_m.}$$

Andere Systeme der logischen Programmierung können sich auch ein anderes Literal als ausgewähltes Literal aussuchen.

Der Name 'SLD-Resolution' kommt aus dem Englischen:
Selection function
Linear resolution
Definite clauses.

Unter *linearer* Resolution versteht man Resolution, die auf solche Resolutionsableitungen beschränkt ist, für die der rechte Nachfolger eines jeden inneren Knotens des Ableitungsbaumes ein Blatt ist. Unter einer *definiten* Klausel versteht man eine Hornklausel.

2.3.2 Prädikatenlogik erster Stufe

Die Sprache der Prädikatenlogik erster Stufe

Im Gegensatz zur Aussagenlogik erlaubt die Prädikatenlogik auch über Objekte und darüber zu sprechen, ob eine Eigenschaft auf alle Objekte zutrifft und ob es ein Objekt gibt, das die Eigenschaft erfüllt. Ein Beispiel für eine prädikatenlogische Formel in der Mathematik ist die Formel

$$\forall x \exists y \; x < y.$$

oder, in Präfixschreibweise, $\forall x \exists y <(x, y)$. Wie in der Aussagenlogik ist eine derartige Formel aus Grundzeichen aufgebaut.

Grundzeichen:

1. Abzählbar unendlich viele *Variablen* (auch Individuenvariablen oder Objektvariablen genannt) (u, v, w, x, y, z, ...).
2. n-stellige *Funktionszeichen* für $n = 0, 1, 2, \ldots$ (f, g, h, ...). Nullstellige Funktionszeichen heißen *Konstanten* (a, b, c, d, ...).
3. n-stellige Prädikatszeichen für $n = 0, 1, 2, \ldots$ (P, Q, R, ...).
4. *Junktoren* $\neg$, $\wedge$, $\vee$.
5. *Quantoren* $\forall$ ('für alle', *Allquantor*), $\exists$ ('existiert', *Existenzquantor*).
6. Runde Klammern ( ) und Beistrich ,.

Definition 55 (induktive Definition der *Terme*) Terme sind spezielle Zeichenreihen aus Grundzeichen, und zwar

1. Jede Variable ist ein Term.
2. Wenn f ein n-stelliges Funktionszeichen ist und $t_1, \ldots, t_n$ Terme sind, dann ist $f(t_1, \ldots, t_n)$ ein Term.

Im Fall $n = 0$ ist mit $f(t_1, \ldots, t_n)$ einfach die Konstante f gemeint.

Für Terme verwenden wir Zeichen r, s, t, ..., auch mit Indizes, ...

Definition 56 (induktive Definition der *Formeln*) Formeln sind spezielle Zeichenreihen aus Grundzeichen, und zwar

1. Wenn P ein n-stelliges Prädikatszeichen ist und $t_1, \ldots, t_n$ Terme sind, dann ist $P(t_1, \ldots, t_n)$ ein Formel. Eine solche Formel nennen wir eine *Atomformel*.
2. Wenn F eine Formel ist, dann ist auch $\neg F$ eine Formel.
3. Wenn F und G Formeln sind, dann sind auch $(F \wedge G)$ und $(F \vee G)$ Formeln.
4. Wenn F eine Formel und x eine Variable ist, dann sind auch $\forall x F$ und $\exists x F$ Formeln.

Wenn F und G Formeln sind, dann nennen wir die Formel $\neg F$ die *Negation* von F, die Formel $(F \wedge G)$ die *Konjunktion* von F und G und die Formel $(F \vee G)$ die *Disjunktion* der Formeln F und G.

Wenn wir über Formeln reden, lassen wir meist Klammern weg, wo dies nicht zu Mißverständnissen führt. Für Formeln verwenden wir Zeichen F, G, H, ..., auch mit Indizes, ...

Definition 57 (induktive Definition der *Teilformeln* einer Formel)

1. F ist eine Teilformel von F.
2. Wenn $\neg G$ eine Teilformel von F ist, dann ist auch G eine Teilformel von F.
3. Wenn $(G \wedge H)$ eine Teilformel von F ist, dann sind auch G und H Teilformeln von F.
4. Wenn $(G \vee H)$ eine Teilformel von F ist, dann sind auch G und H Teilformeln von F.
5. Wenn $\forall x G$ eine Teilformel von F ist, dann ist auch G eine Teilformel von F.
6. Wenn $\exists x G$ eine Teilformel von F ist, dann ist auch G eine Teilformel von F.

Definition 58 Sei F eine Formel und sei $\forall x G$ oder $\exists x G$ eine Teilformel von F. Dann heißt jedes Vorkommen der Variablen x in F, das in dieser Teilformel liegt, ein *gebundenes* Vorkommen von x in F. Ein nicht gebundenes Vorkommen einer Variablen in einer Formel heißt ein *freies* Vorkommen dieser Variablen in der Formel. Eine Formel, in der keine Variablen frei vorkommen, heißt eine *geschlossene* Formel.

Beispiel 33
In der Formel

$$P(x) \vee \neg \forall y \, Q(x, f(y))$$

sind beide Vorkommen der Variablen x frei und beide Vorkommen der Variablen y gebunden. Da in dieser Formel x frei vorkommt, ist die Formel nicht geschlossen. In der Formel

$$P(x, y) \vee \neg \forall y \, Q(x, f(y))$$

sind beide Vorkommen der Variablen x sowie das erste Vorkommen der Variablen y frei, aber das zweite und das dritte Vorkommen der Variablen y gebunden.

Definition 59 Sei F eine Formel und seien $x_1, \ldots, x_n$ diejenigen Variablen, die in F frei vorkommen. Dann heißt die Formel $\forall x_1 \ldots \forall x_n F$ ein *Allabschluß* (auch $\forall$-*Abschluß*) von F und die Formel $\exists x_1 \ldots \exists x_n F$ heißt ein *Existenzabschluß* (auch $\exists$-*Abschluß*) von F. Einen Allabschluß von F bezeichnen wir auch mit $\forall[F]$, einen Existenzabschluß von F mit $\exists[F]$.

Semantik

Definition 60 Eine *Interpretation* ist ein Tripel $i = (D, i_f, i_p)$, wobei gilt

1. D ist eine nichtleere Menge, genannt der *Individuenbereich* (englisch: domain).
2. i_f ist eine Abbildung, die jedem n-stelligen Funktionszeichen f eine n-stellige Funktion $i_f(f)\colon D^n \to D$ zuordnet.
3. i_p ist eine Abbildung, die jedem n-stelligen Prädikatszeichen P eine n-stellige Relation (= Prädikat) $i_p(P) \subseteq D^n$ zuordnet.

Definition 61 Eine *Variablenbelegung* ist eine Abbildung $\mathcal{V}$, die jeder Variablen x einen Wert $\mathcal{V}(x) \in D$ in einem Individuenbereich D zuordnet.

Definition 62 des *Wertes* $i_{\mathcal{V}}t$ eines Terms t bei einer Interpretation i und einer Variablenbelegung $\mathcal{V}$. Dieser Wert ist ein Element des Individuenbereichs.

1. $i_{\mathcal{V}} x := \mathcal{V}(x)$.
2. $i_{\mathcal{V}} f(t_1, \ldots, t_n) := i_f(f)(i_{\mathcal{V}} t_1, \ldots, i_{\mathcal{V}} t_n)$.

Definition 63 Sei $\mathcal{V}$ eine Variablenbelegung über dem Individuenbereich D, sei x eine Variable und sei $\xi \in D$. Dann bezeichnen wir mit $\mathcal{V}_\xi^x$ die Variablenbelegung, die gegeben ist durch

$$\mathcal{V}_\xi^x(y) := \begin{cases} \xi, & \text{wenn } y \equiv x \\ \mathcal{V}(y) & \text{sonst} \end{cases}$$

Definition 64 des *Wahrheitswertes* $i_{\mathcal{V}} F$ einer Formel F bei einer Interpretation i und einer Variablenbelegung $\mathcal{V}$.

1. $i_{\mathcal{V}} P(t_1, \ldots, t_n) := \begin{cases} \text{wahr}, & \text{wenn } (i_{\mathcal{V}} t_1, \ldots, i_{\mathcal{V}} t_n) \in i_p(P) \\ \text{falsch} & \text{sonst} \end{cases}$
2. $i_{\mathcal{V}} \neg F := \begin{cases} \text{wahr}, & \text{wenn } i_{\mathcal{V}} F = \text{falsch} \\ \text{falsch} & \text{sonst} \end{cases}$
3. $i_{\mathcal{V}}(F \wedge G) := \begin{cases} \text{wahr}, & \text{wenn } i_{\mathcal{V}} F = \text{wahr und } i_{\mathcal{V}} G = \text{wahr} \\ \text{falsch} & \text{sonst} \end{cases}$
4. $i_{\mathcal{V}}(F \vee G) := \begin{cases} \text{wahr}, & \text{wenn } i_{\mathcal{V}} F = \text{wahr oder } i_{\mathcal{V}} G = \text{wahr} \\ \text{falsch} & \text{sonst} \end{cases}$
5. $i_{\mathcal{V}} \forall x F := \begin{cases} \text{wahr}, & \text{wenn } i_{\mathcal{V}_\xi^x} F = \text{wahr für alle } \xi \in D \\ \text{falsch} & \text{sonst} \end{cases}$
6. $i_{\mathcal{V}} \exists x F := \begin{cases} \text{wahr}, & \text{wenn es ein } \xi \in D \text{ gibt so, daß } i_{\mathcal{V}_\xi^x} F = \text{wahr ist} \\ \text{falsch} & \text{sonst} \end{cases}$

Der Wahrheitswert einer geschlossenen Formel F hängt nicht von der Variablenbelegung, sondern nur von der Interpretation i ab. Wir bezeichnen ihn mit iF.

Definition 65 Ein *Modell* einer geschlossenen Formel ist eine Interpretation i so, daß $iF = \text{wahr}$ ist.

Die Begriffe *semantische Folgerung*, *semantische Äquivalenz*, *Allgemeingültigkeit* sowie *Erfüllbarkeit* von Formeln und von Formelmengen sind entsprechend wie in der Aussagenlogik definiert.

Die Sätze, die wir in der Aussagenlogik kennengelernt haben, gelten auch in der Prädikatenlogik. Für die logische Programmierung sind insbesondere die folgenden beiden Sätze von Bedeutung.

Satz 24 *Sei F eine Formel. Dann gilt:*

$$\begin{aligned} F \text{ ist allgemeingültig} &\iff \neg F \text{ ist unerfüllbar} \\ F \text{ ist unerfüllbar} &\iff \neg F \text{ ist allgemeingültig} \end{aligned}$$

Satz 25 *Sei S eine Formelmenge und F eine Formel. Dann gilt:*

$$S \models F \iff S \cup \{\neg F\} \text{ ist unerfüllbar.}$$

Prologklauseln als Formeln

Wir definieren $F \to G$ als Abkürzung für $\neg F \vee G$.

Beispiel 34

```
grossvater(X,Z) :-
     vater(X,Y),
     elternteil(Y,Z).
```

entspricht der Formel

$$\forall x \forall y \forall z \big(V(x,y) \wedge E(y,z) \to G(x,z) \big).$$

„Für alle x, y und z gilt: wenn x Vater von y ist und y Elternteil von z ist, dann ist x Großvater von z."

Allgemein entspricht eine Prologklausel

$$A\texttt{:-}B_1, \ldots, B_n.$$

der Formel

$$\forall [B_1 \wedge \cdots \wedge B_n \to A].$$

Dabei ist $B_1 \wedge \cdots \wedge B_n \to A$ eine Abkürzung für $\neg(B_1 \wedge \cdots \wedge B_n) \vee A$, was wiederum semantisch äquivalent ist zur Formel $A \vee \neg B_1 \vee \cdots \vee \neg B_n$. Die Prologklausel entspricht also auch der Formel

$$\forall [A \vee \neg B_1 \vee \cdots \vee \neg B_n].$$

Ein Faktum

$$A.$$

entspricht der Formel

$$\forall [A].$$

Beispiel 35

```
gleich(X,X).
```

entspricht der Formel

$$\forall x\, {=}(x,x).$$

Eine Anfrage

$$\texttt{?-}B_1, \ldots, B_n.$$

entspricht der Formel

$$\forall [\neg B_1 \vee \cdots \vee \neg B_n].$$

Beispiel 36

```
?- grossvater(X,gabi).
```

entspricht der Formel

$$\forall x \neg G(x,g).$$

Prolog versucht aus dem Programm und aus dieser Formel einen Widerspruch abzuleiten. Es zeigt also indirekt (durch Widerspruchsbeweis) die Existenz eines x mit $G(x,g)$, also eines Großvaters von Gabi. Dieser Beweis geschieht konstruktiv, d.h. Prolog konstruiert einen Wert für x so, daß dann $G(x,g)$ gilt.

Hier ist ein Beispiel für ein komplettes Prologprogramm und eine dazugehörige Anfrage.

Beispiel 37

```
elternteil(hans,peter).
elternteil(peter,gabi).
maennlich(hans).
vater(X,Y) :- elternteil(X,Y), maennlich(X).
grossvater(X,Z) :- vater(X,Y), elternteil(Y,Z).
?- grossvater(X,gabi).
```

Diese sechs Klauseln entsprechen den folgenden sechs Formeln.

$$
\begin{gathered}
E(h,p)\\
E(p,g)\\
M(h)\\
\forall x\forall y\big(V(x,y)\vee\neg E(x,y)\vee\neg M(x)\big)\\
\forall x\forall y\forall z\big(G(x,z)\vee\neg V(x,y)\vee\neg E(y,z)\big)\\
\forall x\neg G(x,g).
\end{gathered}
$$

Prolog startet mit der letzten dieser sechs Formeln (der Formel, die der Anfrage entspricht) und verwendet die übrigen fünf Formeln, um daraus einen Widerspruch abzuleiten. Wenn die ersten fünf Formeln gelten, kann also die letzte nicht gelten, d.h. es muß einen Wert für x geben so, daß $G(x,g)$ gilt. Im Beispiel gilt dies für $x=h$, also Hans. Prolog findet diesen Wert und gibt aus

```
X = hans.
```

Wir können die sechs Formeln zu einer zusammenfassen entweder so

$$
\begin{gathered}
E(h,p)\wedge\\
E(p,g)\wedge\\
M(h)\wedge\\
\forall x\forall y\big(V(x,y)\vee\neg E(x,y)\vee\neg M(x)\big)\wedge\\
\forall x\forall y\forall z\big(G(x,z)\vee\neg V(x,y)\vee\neg E(y,z)\big)\wedge\\
\forall x\neg G(x,g).
\end{gathered}
$$

oder, was semantisch äquivalent dazu ist, so

$$
\begin{gathered}
\forall x\forall y\forall z\Big(\\
E(h,p)\wedge\\
E(p,g)\wedge\\
M(h)\wedge\\
\big(V(x,y)\vee\neg E(x,y)\vee\neg M(x)\big)\wedge\\
\big(G(x,z)\vee\neg V(x,y)\vee\neg E(y,z)\big)\wedge\\
\neg G(x,g)\Big).
\end{gathered}
$$

Das Ergebnis ist ein Allabschluß einer Konjunktion von Disjunktionen von Literalen, wobei wir unter einem Literal eine Atomformel oder die Negation einer Atomformel verstehen. Von einer solchen Formel sagen wir, sie sei in *konjunktiver Normalform* (KNF).

Normalformen

Ähnlich wie in der Aussagenlogik gibt es auch in der Prädikatenlogik den Begriff der Normalformen. Auch hier gibt es Negationsnormalform, konjunktive Normalform, disjunktive Normalform und eine Reihe weiterer Normalformen. Wir interessieren uns besonders für die konjunktive Normalform.

Definition 66 Eine Formel heißt ein *Literal*, wenn sie eine Atomformel oder die Negation einer Atomformel ist. Im ersteren Fall heißt sie ein *positives Literal*, im letzteren Fall ein *negatives Literal*.

Beispiel 38
Seien x eine Variable, a und b zwei Konstanten, P ein zweistelliges Prädikatszeichen und f ein zweistelliges Funktionszeichen. Dann ist $P(a, f(x, b))$ ist ein positives Literal und $\neg P(f(x, x), f(b, x))$ ein negatives Literal.

Definition 67 Eine Formel F ist in *konjunktiver Normalform* (*KNF*), wenn sie ein Allabschluß einer Konjunktion von Disjunktionen von Literalen ist.

Beispiel 39
Die Formel

$$\begin{aligned}\forall x \forall y \forall z \Big(& \\ & E(h,p) \wedge \\ & E(p,g) \wedge \\ & M(h) \wedge \\ & \big(V(x,y) \vee \neg E(x,y) \vee \neg M(x)\big) \wedge \\ & \big(G(x,z) \vee \neg V(x,y) \vee \neg E(y,z)\big) \wedge \\ & \neg G(x,g)\Big).\end{aligned}$$

ist in konjunktiver Normalform.

Zu einer Formel der Prädikatenlogik gibt es im allgemeinen keine semantisch äquivalente Formel in KNF. Zum Beispiel gibt es keine Formel in KNF, die semantisch äquivalent ist zu $\exists x\, P(x)$. Es gilt aber der folgende Satz.

Satz 26 *Zu jeder Formel F kann man effektiv eine Formel F' in KNF konstruieren derart, daß F genau dann erfüllbar ist, wenn F' erfüllbar ist.*[3]

Wenn für zwei Formeln F und F' gilt, daß F genau dann erfüllbar ist, wenn F' erfüllbar ist, so sagen wir auch, F und F' seien *erfüllbarkeitsäquivalent*. In der Prädikatenlogik gilt also im Gegensatz zur Aussagenlogik nicht die semantische Äquivalenz einer Formel mit ihrer KNF, sondern lediglich die Erfüllbarkeitsäquivalenz. Da ein Prologsystem aber ein Theorembeweiser ist, der die Unerfüllbarkeit einer Formel nachzuweisen versucht, genügt die Erfüllbarkeitsäquivalenz um zu sehen, daß bei Prolog die Beschränkung auf KNF keine so gravierende Einschränkung ist.

[3] Auf diesen Satz soll hier nicht näher eingegangen werden. Die Idee ist die, daß man die Existenzquantoren in F irgendwie eliminieren muß, da eine Formel in KNF keine Existenzquantoren enthalten darf. Wenn nun etwa $\forall x \exists y\, P(x, y)$ gilt, so muß es eine Funktion geben, die jedem Wert für x einen entsprechenden Wert für y zuordnet, es muß also die Formel $\forall x\, P(x, f(x))$ erfüllbar sein. Diese Einführung von Funktionszeichen nennt man *Skolemisierung*. Das neu eingeführte Funktionszeichen f nennt man *Skolemfunktionszeichen*, oft auch einfach *Skolemfunktion*.

Klauseln

Definition 68 Eine *Klausel* ist eine endliche Menge von Literalen.

Eine Formel in KNF läßt sich durch eine endliche Menge von Klauseln darstellen. Und zwar hat ja nach Definition die Formel die Form eines Allabschlusses einer Konjunktion von Disjunktionen von Literalen. Jeder dieser Disjunktionen von Literalen ordnen wir eine Klausel zu, nämlich die Menge aller Literale der Disjunktion.

Beispiel 40
Die obige Formel in KNF entspricht der Klauselmenge, die aus den folgenden Klauseln besteht.

$$\{E(h,p)\}$$
$$\{E(p,g)\}$$
$$\{M(h)\}$$
$$\{V(x,y), \neg E(x,y), \neg M(x)\}$$
$$\{G(x,z), \neg V(x,y), \neg E(y,z)\}$$
$$\{\neg G(x,g)\}.$$

Definition 69 Eine *Hornklausel* ist eine Klausel, die höchstens ein positives Literal enthält. Eine *Programmklausel* ist eine Klausel, die genau ein positives Literal enthält. Eine *Zielklausel* ist eine Klausel, die kein positives Literal enthält.

Im Beispiel sind alle Klauseln Hornklauseln, und zwar sind die ersten fünf Klauseln Programmklauseln und die sechste Klausel ist eine Zielklausel. Prolog arbeitet stets mit Hornklauseln.

Resolution

Betrachten wir nochmal das Großvater-Beispiel.

```
...
grossvater(X,Z) :- vater(X,Y), elternteil(Y,Z).
?- grossvater(X,gabi).
```

Es soll grob skizziert werden, wie eine solche Anfrage von Prolog gelöst wird. Prolog startet mit der Anfrage. Das Ziel `grossvater(X,gabi)` ist für einen geeigneten Wert von `X` zu beweisen. Hierzu sucht Prolog im Programm eine Klausel, deren Kopf das Prädikatszeichen `grossvater` hat und findet

```
grossvater(X,Z) :- vater(X,Y), elternteil(Y,Z).
```

`grossvater(X,gabi)` ist also dann bewiesen, wenn man `vater(X,Y)` und `elternteil(Y,gabi)` für einen geeigneten Wert von `Y` beweist. Prolog erzeugt also eine neue Anfrage

```
?- vater(X,Y), elternteil(Y,gabi).
```

die es seinerseits zu lösen versucht. Wie hat sich diese neue Anfrage ergeben? Hierzu mußte das Ziel `grossvater(X,Z)` der ursprünglichen Anfrage und der Kopf `grossvater(X,Z)` der Klausel des Prologprogramms einander gleich gemacht (wir sagen *unifiziert*) werden durch Ersetzung (wir sagen *Substitution*) der Variablen `Z` durch `gabi`. Also

```
grossvater(X,gabi) :- vater(X,Y), elternteil(Y,gabi).
?- grossvater(X,gabi).
```

Die neue Anfrage ergibt sich nun durch Resolution dieser beiden Prologklauseln in der Form, in der wir sie bereits aus der Aussagenlogik kennen. Wir werden später die Resolution für die Prädikatenlogik genauer definieren. Hier halten wir nur grob fest, daß sich in der Prädikatenlogik einen Resolutionsschritt aufgeteilt in drei Schritte vorstellen kann: eine *Unifikation*, Anwendung einer *Substitution* auf die Elternklauseln und eine einfache aussagenlogische Resolution, wie wir sie bereits kennengelernt haben.

Unifikation Es geht darum zwei Terme durch eine Substitution gleich zu machen (Gleichungslösen).

Beispiel 41
Das zweistellige Prädikat = ist in Prolog vordefiniert durch

```
=(X,X).
```

Stellt man nun an Prolog die Anfrage

```
?- =( f(X,g(X)) , f(g(a),Y) ).
```

so muß Prolog die Gleichung

```
f(X,g(X)) = f(g(a),Y)
```

lösen. Für `X` und `Y` sollen Terme gefunden werden, die diese Gleichung erfüllen, egal durch was für Funktionen die Funktionszeichen `f` und `g` interpretiert werden. Dies ist nur möglich, wenn gleichzeitig die beiden Gleichungen

```
X = g(a)          g(X) = Y
```

gelöst werden. Die Lösung lautet

```
X = g(a)
Y = g(g(a)).
```

Die ursprüngliche Gleichung wird also gelöst, indem man darin die Variable `X` durch den Term `g(a)` und die Variable `Y` durch den Term `g(g(a))` ersetzt. Wir sprechen von einer *Substitution*, die wir folgendermaßen schreiben.

$$\{\texttt{X} \leftarrow \texttt{g(a)}, \texttt{Y} \leftarrow \texttt{g(g(a))}\}$$

Mathematisch ist eine Substitution dadurch gegeben, daß wir zu jeder Variablen den Term angeben, durch den die Variable bei der Substitution ersetzt wird.

Definition 70 Eine *Substitution* ist eine Funktion σ, die jeder Variablen x einen Term $\sigma(x)$ zuordnet, wobei nur für endlich viele Variablen x gilt: $\sigma(x) \neq x$.

Beispiel 42
Sei

$$\begin{aligned}
&\sigma(x) = g(a)\\
&\sigma(y) = g(g(a))\\
&\sigma(z) = z \qquad \text{für alle } z, \text{ die nicht identisch mit } x \text{ oder } y \text{ sind.}
\end{aligned}$$

Diese Substitution σ wird geschrieben als $\{x \leftarrow g(a), y \leftarrow g(g(a))\}$.

Definition 71 Seien $x_1, \ldots, x_n$ Variablen und $t_1, \ldots, t_n$ Terme und die Substitution σ sei definiert durch

$$\begin{aligned} \sigma(x_i) &= t_i \\ \sigma(x) &= x \qquad \text{für alle } x, \text{ die nicht identisch mit einem der } x_i \text{ sind.} \end{aligned}$$

Dann wird σ bezeichnet mit $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$.

Die Ausgabe von Prolog ist, sofern sie nicht '`No`' lautet, eine Substitution. Wenn man zum Beispiel auf die Anfrage

```
?- grossvater(X,gabi).
```

von Prolog die Antwort

```
X = hans
```

bekommt, so so bedeutet dies die Substitution $\{$`X` $\leftarrow$ `hans`$\}$. Die Ausgabe `Yes` bedeutet die leere Substitution $\{\}$, die auch mit ε bezeichnet wird. Es ist $\varepsilon(x) = x$ für alle Variablen x.

Definition 72 Die *Anwendung* einer Substitution σ auf einen Term t ergibt einen Term $t\sigma$, der folgendermaßen definiert ist.

$$\begin{aligned} x\sigma &:= \sigma(x) \\ f(t_1, \ldots, t_n)\sigma &:= f(t_1\sigma, \ldots, t_n\sigma). \end{aligned}$$

Beispiel 43
$f(x, g(x))\ \{x \leftarrow g(a), y \leftarrow g(g(a))\} = f(g(a), g(g(a)))$.

Definition 73 Seien σ und τ zwei Substitutionen. Dann ist die *Komposition* $\sigma\tau$ von σ und τ folgendermaßen definiert: $(\sigma\tau)(x) := (\sigma(x))\tau$.

Beispiel 44
Sei $\sigma = \{x \leftarrow f(x, y), y \leftarrow g(z)\}$ und $\tau = \{y \leftarrow g(y), z \leftarrow a\}$. Dann ist $\sigma\tau = \{x \leftarrow f(x, g(y)), y \leftarrow g(a), z \leftarrow a\}$.

Wenn σ und τ zwei Substitutionen sind und t ein Term ist, dann gilt $t(\sigma\tau) = (t\sigma)\tau$.

Definition 74 Seien s und t zwei Terme. Unter einem *Unifikator* von s und t verstehen wir eine Substitution σ so, daß $s\sigma = t\sigma$ ist. Wenn zwei Terme s und t einen Unifikator haben, dann sagen wir, sie seien *unifizierbar*.

Beispiel 45
Die Terme $f(x, b)$ und $f(a, y)$ haben den Unifikator $\{x \leftarrow a, y \leftarrow b\}$.
Die Terme a und $f(x)$ haben keinen Unifikator.
Die Terme x und $f(x)$ haben keinen Unifikator.
Jede Substitution ist Unifikator der Terme x und x.
Die Terme x und $f(y)$ haben den Unifikator $\{x \leftarrow f(y)\}$, aber auch den Unifikator $\{x \leftarrow f(g(z)), y \leftarrow g(z), u \leftarrow g(f(z))\}$.

Definition 75 Eine Substitution σ ist *allgemeiner* als eine Substitution τ, wenn es eine Substitution ρ gibt mit $\tau = \sigma\rho$.

Wenn ein Unifikator σ zweier Terme s und t allgemeiner ist als eine Substitution τ, dann ist auch τ ein Unifikator von s und t. Im letzten Beispiel ist der erste angegebene Unifikator von x und $f(y)$ allgemeiner als der zweite.

Definition 76 Eine Substitution σ heißt *allgemeinster Unifikator* (englisch: *most general unifier, mgu*) zweier Terme s und t, wenn σ ein Unifikator von s und t ist und außerdem σ allgemeiner als jeder Unifikator von s und t ist.

Satz 27 *Zwei unifizierbare Terme haben immer auch einen allgemeinsten Unifikator.*

Unter einem *Unifikationsalgorithmus* versteht man einen Algorithmus, der von zwei Termen festellt, ob sie unifizierbar sind, und im Falle der Unifizierbarkeit einen allgemeinsten Unifikator liefert.

Es gibt effiziente Unifikationsalgorithmen, deren Laufzeit linear mit der Länge der Terme anwächst. Da aber Prolog bei jedem Berechnungsschritt eine Unifikation durchführen muß und die Terme oft äußerst komplexe Datenstrukturen darstellen und daher sehr lang sind, genügt diese Effizienz in der Praxis noch nicht. Prolog verwendet einen in vielen Fällen noch effizienteren Algorithmus. Will Prolog zum Beispiel eine Variable x mit einem langen Term t unifizieren, so ermittelt es als Unifikator einfach die Substitution $\{x \leftarrow t\}$. Es hat den Term t bereits im Speicher und braucht ihn nicht erneut aufzubauen. Dies funktioniert allerdings nur, wenn x in t nicht vorkommt. Wenn die Möglichkeit besteht, daß x in t vorkommen könnte, muß für korrekte Unifikation t danach durchsucht werden, ob x darin vorkommt (englisch: occurs). Man nennt das den 'occur check'. In den meisten praktischen Anwendungen ist der 'occur check' nicht nötig. Prolog verwendet in seinem Unifikationsalgorithmus standardmäßig keinen 'occur check', in einigen wenigen Prologversionen läßt er sich aber optional einschalten.

Da für Prolog ein Prädikatszeichen sich syntaktisch nicht von einem Funktionszeichen unterscheidet und daher eine Atomformel sich syntaktisch nicht von einem Term unterscheidet, kann man die Begriffe der Anwendung einer Substitution und der Unifizierbarkeit sowie den Unifikationsalgorithmus genausogut für Atomformeln wie für Terme verwenden. So ist zum Beispiel $P(t_1, \ldots, t_n)\sigma \equiv P(t_1\sigma, \ldots, t_n\sigma)$. Wenn A eine Atomformel und σ eine Substitution ist, so bezeichne $(\neg A)\sigma$ die Formel $\neg(A\sigma)$. Wenn $\{L_1, \ldots, L_n\}$ eine Klausel ist, so werden wir auch $\{L_1, \ldots, L_n\}\sigma$ schreiben für $\{L_1\sigma, \ldots, L_n\sigma\}$. In Prolog-Notation schreiben wir `?-` $A_1, \ldots, A_n.\sigma$ für `?-` $A_1\sigma, \ldots, A_n\sigma.$ und A `:-` $B_1, \ldots, B_n.\sigma$ für $A\sigma$ `:-` $B_1\sigma, \ldots, B_n\sigma$.

SLD-Resolution Der Resolutionskalkül läßt sich von der Aussagenlogik auf die Prädikatenlogik übertragen. Da wir es in Prolog mit Hornklauseln und dem speziellen Begriff der SLD-Resolution zu tun haben, werden wir hier die allgemeine Resolution nicht einführen, sondern gleich zur SLD-Resolution kommen.

Definition 77 Zwei Klauseln heißen *variablendisjunkt*, wenn sie keine gemeinsamen Variablen haben.

Beispiel 46
Die beiden Klauseln

```
grossvater(X,Z) :- vater(X,Y), elternteil(Y,Z).
?- grossvater(X,gabi).
```

sind nicht variablendisjunkt, da beide Klauseln die Variable `X` enthalten. Dagegen sind die beiden Klauseln

```
grossvater(X',Z) :- vater(X',Y), elternteil(Y,Z).
?- grossvater(X,gabi).
```

variablendisjunkt.

Der Begriff eines SLD-Resolutionsschrittes läßt sich am einfachsten erklären, wenn man zunächst annimmt, daß die beteiligten Elternklauseln variablendisjunkt sind. Dann schaut ein SLD-Resolutionsschritt folgendermaßen aus.

$$\frac{\texttt{?-}\ A_1, \ldots, A_j, \ldots, A_m. \qquad A\ \texttt{:-}\ B_1, \ldots, B_n.}{\texttt{?-}\ A_1, \ldots, A_{j-1}, B_1, \ldots, B_n, A_{j+1}, \ldots, A_m.\sigma}.$$

Dabei müssen A und A_j unifizierbar sein und σ muß ein allgemeinster Unifikator von A und A_j sein. Sind die Elternklauseln nicht variablendisjunkt, müssen vorher in der Programmklausel Variablen umbenannt werden wie im Beispiel oben.

Definition 78 Eine *Variante* einer Klausel c ist das Ergebnis einer Variablenumbenennung in c, d.h. einer Ersetzung von Variablen durch andere Variablen, wobei gleiche Variablen durch gleiche, verschiedene durch verschiedene ersetzt werden.

Beispiel 47
`grossvater(X',Z) :- vater(X',Y), elternteil(Y,Z).`
ist eine Variante der Klausel

`grossvater(X,Z) :- vater(X,Y), elternteil(Y,Z).`

In der Prädikatenlogik stellt eine Klausel einen Allabschluß dar. Daher ist eine Variante einer Klausel genau dann wahr, wenn die ursprüngliche Klausel wahr ist.

Die allgemeine SLD-Resolutionsregel schaut nun so aus.

$$\frac{\texttt{?-}\ A_1, \ldots, A_j, \ldots, A_m. \qquad A\ \texttt{:-}\ B_1, \ldots, B_n.}{\texttt{?-}\ A_1, \ldots, A_{j-1}, B'_1, \ldots, B'_n, A_{j+1}, \ldots, A_m.\sigma},$$

wobei $A'\texttt{:-}B'_1, \ldots, B'_n.$ eine Variante von $A\texttt{:-}B_1, \ldots, B_n.$ und variablendisjunkt zu $\texttt{?-}A_1, \ldots, A_m.$ ist und σ ein mgu von A_j und A' ist.

Da in Prolog das ausgewählte Literal immer das erste Literal der Zielklausel ist, hat die SLD-Resolutionsregel für Prolog die Form

$$\frac{\texttt{?-}\ A_1, \ldots, A_m. \qquad A\ \texttt{:-}\ B_1, \ldots, B_n.}{\texttt{?-}\ B'_1, \ldots, B'_n, A_2, \ldots, A_m.\sigma},$$

wobei $A'\texttt{:-}B'_1, \ldots, B'_n.$ eine Variante von $A\texttt{:-}B_1, \ldots, B_n.$ und variablendisjunkt zu $\texttt{?-}A_1, \ldots, A_m.$ ist und σ ein mgu von A_1 und A' ist.

In anderen Systemen der logischen Programmierung kann das ausgewählte Literal auch ein anderes Literal der Zielklausel sein.

Der Begriff der *SLD-Resolutionsableitung* ist analog definiert wie in der Aussagenlogik. Eine *SLD-Widerlegung* einer Klauselmenge ist eine SLD-Resolutionsableitung der leeren Klausel $\square$ aus dieser Klauselmenge.

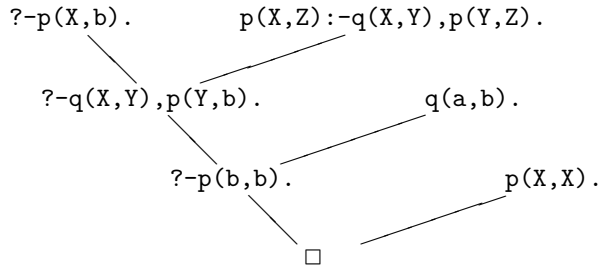
Beispiel 48
Betrachten wir das folgende Prologprogramm

```
p(X,Z) :- q(X,Y), p(Y,Z).
p(X,X).
q(a,b).
```

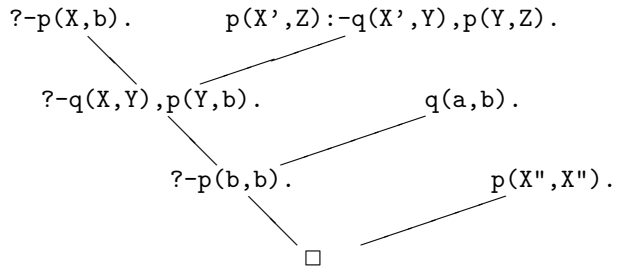
mit der Anfrage

```
?- p(X,b).
```

Dann ist eine mögliche SLD-Widerlegung der Klauselmenge, die aus diesen vier Klauseln besteht, die folgende.



Bei jedem Resolutionsschritt wird zunächst jeweils von der im abgebildeten Baum rechten Elternklausel, also von der beteiligten Programmklausel, eine Variante gebildet. Die Darstellung der SLD-Widerlegung als Baum unter Verwendung von Varianten der Klauseln des Programms schaut im Beispiel so aus.



In diesem Baum stehen jeweils rechts Varianten von Klauseln des Prologprogramms. Wir wählen die Variante dabei so, daß alle auftretenden Variablen jeweils neu sind, also im Baum in einem früheren Resolutionsschritt noch nicht vorgekommen sind. Diese Darstellung einer SLD-Widerlegung als Baum ist zum Verstehen des Vorgehens von Prolog geeigneter als der zuerst abgebildete Baum. Denn die Antwort, die Prolog schließlich liefert, ergibt sich aus den jeweiligen allgemeinsten Unifikatoren zu allen Resolutionsschritten. In der Baumdarstellung mit Varianten der Programmklauseln kann man diese Unifikatoren leicht direkt aus dem Baum ablesen. So werden in dieser Darstellung. Im ersten Schritt werden die Klauseln

```
?- p(X,b).
p(X',Z) :- q(X',Y), p(Y,Z).
```

miteinander resolviert. Dazu muß das Ziel `p(X,b)` der Zielklausel mit dem Kopf `p(X',Z)` der Programmklausel —, die eine Variante der ersten Klausel des ursprünglichen Programms ist, —

unifiziert werden. Ein allgemeinster Unifikator dieser beiden Atomformeln ist die Substitution

$$\sigma_1 = \{\texttt{Z} \leftarrow \texttt{b}, \texttt{X'} \leftarrow \texttt{X}\}.$$

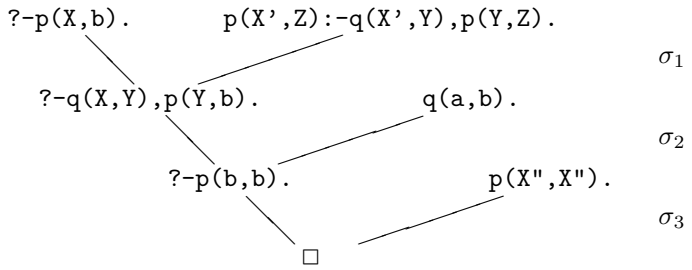
Beim zweiten Resolutionsschritt ist ein mgu die Substitution

$$\sigma_2 = \{\texttt{X} \leftarrow \texttt{a}, \texttt{Y} \leftarrow \texttt{b}\}.$$

Beim dritten Resolutionsschritt ist ein mgu die Substitution

$$\sigma_3 = \{\texttt{X"} \leftarrow \texttt{b}\}.$$

Schreibt man die jeweiligen allgemeinsten Unifikatoren der Resolutionsschritte rechts neben den Baum, so erhält man das folgende Bild.



Dabei sind σ_1, σ_2 und σ_3 wie oben definiert. Insgesamt wurde die in der ursprünglichen Anfrage vorkommende Variable `X` durch `a` ersetzt. Wir sagen, die Substitution $\{\texttt{X} \leftarrow \texttt{a}\}$ sei die *berechnete Antwortsubstitution*. Die berechnete Antwortsubstitution wird von Prolog ausgegeben in der Form

```
X = a
```

Die berechnete Antwortsubstitution ergibt sich, indem man die Substitutionen σ_1, σ_2 und σ_3 hintereinander ausführt, also die Komposition

$$\sigma_1\sigma_2\sigma_3 = \{\texttt{X} \leftarrow \texttt{a}, \texttt{X'} \leftarrow \texttt{a}, \texttt{Y} \leftarrow \texttt{b}, \texttt{Z} \leftarrow \texttt{b}, \texttt{X"} \leftarrow \texttt{b}\}$$

bildet. Da für die Antwort, die Prolog liefert aber nur von Interesse ist, durch welche Terme diejenigen Variablen ersetzt werden, die in der ursprünglichen Anfrage vorkommen (im Beispiel lediglich die Variable `X`, muß diese Komposition auf diese Variablen eingeschränkt werden (siehe unten), d.h. die Antwort, die Prolog liefert, stellt die Substitution

$$\{\texttt{X} \leftarrow \texttt{a}\}$$

dar.

Berechnete und korrekte Antwortsubstitutionen

Definition 79 Sei σ eine Substitution und V eine Menge von Variablen. Dann ist die *Einschränkung* $\sigma\big|_V$ von σ auf V definiert durch

$$\sigma\big|_V(x) = \begin{cases} \sigma(x) & \text{, wenn } x \in V \\ x & \text{sonst.} \end{cases}$$

Definition 80 Sei P eine Menge von Programmklauseln und G eine Zielklausel. Seien $\sigma_1, \ldots, \sigma_n$ die jeweiligen allgemeinsten Unifikatoren der Resolutionsschritte einer SLD-Widerlegung von $P \cup \{G\}$. Dann heißt die Einschränkung der Komposition $\sigma_1 \ldots \sigma_n$ auf die Menge der in der Anfrage G auftretenden Variablen eine *berechnete Antwortsubstitution* für $P \cup \{G\}$.

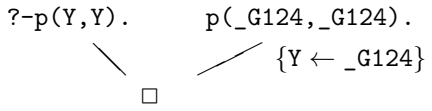
Im Beispiel ist die Substitution $\sigma = \{\texttt{X} \leftarrow \texttt{a}\}$ nicht nur eine berechnete Antwortsubstitution. Die Antwort ist auch korrekt in dem Sinne, daß die Formel `p(a,b)`, die sich aus dem Ziel der Anfrage dadurch ergibt, daß man darin `X` durch `a` ersetzt, semantisch aus dem Programm folgt. In Zeichen kann man das so ausdrücken. $P \models \texttt{p(X,b)}\sigma$. In diesem Beispiel kommt in der Formel $\texttt{p(X,b)}\sigma$ keine Variable mehr vor. Betrachten wir aber nun das folgende Prolog-Programm und die folgende Prolog-Anfrage

```
p(X,X).
?- p(Y,Y).
```

Hier liefert Prolog etwa folgende Antwort.

```
Y = _G124
```

Dies stellt eine Substitution $\{\texttt{Y} \leftarrow \texttt{\_G124}\}$ dar. Was ist geschehen? Prolog hat zunächst versucht die Zielklausel mit der Programmklausel zu resolvieren. Hierzu hat es eine Variante `p(_G124,_G124).` der Programmklausel `p(X,X).` gebildet. Hierzu wiederum mußte Prolog eine neue Variable `_G124` erzeugen. Die Resolution hat als Resolvente die leere Klausel $\square$ und als allgemeinsten Unifikator die Substitution $\sigma = \{\texttt{Y} \leftarrow \texttt{\_G124}\}$ geliefert, die Prolog dann ausgegeben hat. Die Darstellung der SLD-Widerlegung als Baum unter Verwendung von Varianten der Klauseln des Programms schaut dabei so aus.



Wendet man hier die berechnete Antwortsubstitution σ auf das Ziel `p(Y,Y)` an, so erhält man die Formel `p(_G124,_G124)`. Auch hier folgt die resultierende Formel aus dem Programm, und zwar egal, was man für die Variable `_G124` einsetzt. Genauer gesagt, der Allabschluß der Formel folgt semantisch aus dem Programm. In diesem Sinne ist die von Prolog gelieferte Antwort korrekt.

Definition 81 Sei P eine Menge von Programmklauseln und sei G eine Zielklausel $\texttt{?-}A_1, \ldots, A_n$. Dann heißt eine Substitution σ eine *korrekte Antwortsubstitution* für $P \cup \{G\}$, wenn gilt $P \models \forall[(A_1 \wedge \cdots \wedge A_n)\sigma]$.

Satz 28 *Sei P eine Menge von Programmklauseln und G eine Zielklausel. Dann ist jede berechnete Antwortsubstitution für $P \cup \{G\}$ auch eine korrekte Antwortsubstitution.*

Im letzten Beispiel ist $\{\texttt{Y} \leftarrow \texttt{\_G124}\}$ die berechnete Antwortsubstitution und daher auch eine korrekte Antwortsubstitution. Darüberhinaus ist aber jede Substitution $\{\texttt{Y} \leftarrow t\}$ ist eine korrekte Antwortsubstitution.

Satz 29 *Sei P eine Menge von Programmklauseln und G eine Zielklauel. Dann gibt es zu jeder korrekten Antwortsubstitution σ eine berechnete Antwortsubstitution, die allgemeiner als σ ist.*

Resolution für KNF allgemein. $(c\sigma \setminus c_0\sigma) \cup (d'\sigma \setminus d'_0\sigma)$. Korrektheits- und Vollständikeitssatz. Mit Normalformtransformation hat man Verfahren zum Nachweis der Unerfüllbarkeit / Allgemeingültigkeit für die volle Prädikatenlogik erster Stufe mit Funktionszeichen.

Der Kalkül des natürlichen Schließens

Der Kalkül des natürlichen Schließens versucht den in der Mathematik üblichen Stil des Beweisens von Aussagen zu formalisieren. Betrachten wir als Beispiel für einen mathematischen Beweis den folgenden Beweis für die Formel $A \wedge B \to B \wedge A$.

1. Wir nehmen an, $A \wedge B$ gelte.
2. Wegen 1. gilt B.
3. Wegen 1. gilt A.
4. Wegen 2. und 3. gilt $B \wedge A$.
5. Da wir unter der Annahme 1. die Aussage 4. bewiesen haben, gilt die Implikation $A \wedge B \to B \wedge A$.

Man kann diesen Beweis als Baum darstellen:

$$\dfrac{\dfrac{\dfrac{A \wedge B}{B} \quad \dfrac{A \wedge B}{A}}{B \wedge A}}{A \wedge B \to B \wedge A} \ .$$

Die Blätter des Baumes sind Annahmen. Im Beispiel wird die Annahme $A \wedge B$ zweimal verwendet – einmal um B zu beweisen und einmal um A zu beweisen. Schaun wir uns die Schritte des Beweises genauer an. Zunächst machen wir die Annahme $A \wedge B$. Daraus schließen wir auf B. Diesen Schluss schreiben wir so.

$$\dfrac{A \wedge B}{B}$$

Weiters schließen wir aus derselben Annahme $A \wedge B$ auf A mit dem Schluss

$$\dfrac{A \wedge B}{A}.$$

Aus B und A schließen wir auf $B \wedge A$ mit dem Schluss

$$\dfrac{B \quad A}{B \wedge A}.$$

Bis hierher hängen alle Aussagen, die wir bewiesen haben, von der gemachten Annahme $A \wedge B$ ab, d.h. sie sind nur unter der Voraussetzung, dass die Annahme gilt, als wahr bewiesen worden. Als Letztes schließen wir daraus, dass $B \wedge A$ unter der Annahme $A \wedge B$ gilt, auf $A \wedge B \to B \wedge A$. Wir schreiben den Schluss folgendermassen.

$$\dfrac{\overset{[A \wedge B]}{B \wedge A}}{A \wedge B \to B \wedge A}.$$

Dabei bedeutet die in eckigen Klammern eingeschlossene Formel $A \wedge B$ über der Prämisse $B \wedge A$, dass die Konklusion $A \wedge B \to B \wedge A$ des Schlusses nun nicht mehr von der Annahme $A \wedge B$ abhängt. Wir können auch in einem Schluss mehrere Prämissen haben, über denen jeweils höchstens eine Formel in eckigen Klammern stehen darf. Ein Beispiel dafür ist die Oder-Beseitigung

$$\dfrac{A \vee B \quad \overset{[A]}{C} \quad \overset{[B]}{C}}{C} \ .$$

Kommt in einem formalen Beweis (wir sprechen hierbei von einer Herleitung) eine Oder-Beseitigung vor, so hängt die Konklusion C von den folgenden Annahmen ab.

1. von allen Annahmen, von denen die erste Prämisse $A \vee B$ abhängt
2. von allen Annahmen, von denen die zweite Prämisse C abhängt, mit Ausnahme aller Annahmen, die identisch mit der Formel A sind
3. von allen Annahmen, von denen die dritte Prämisse C abhängt, mit Ausnahme aller Annahmen, die identisch mit der Formel B sind

Allgemein sind die Annahmen in einer Herleitung diejenigen Blätter, die nicht Axiome sind. Annahmen hängen von sich selbst ab. Kommt eine Formel B als Konklusion eines Schlusses in der Herleitung vor und ist A eine der Prämissen des Schlusses, so hängt B von allen Annahmen ab, von denen A abhängt – mit Ausnahme aller solcher Annahmen, die formal gleich der im Schluss über der Prämisse in eckigen Klammern stehenden Formel ist, falls eine solche Formel dort in eckigen Klammern steht.

Das Ziel ist es nun die zu beweisende Formel zu erschließen, wobei man sich von allen gemachten Annahmen unabhängig macht.

Im Folgenden verwenden wir A, B, C, D, ... als Mitteilungszeichen für Formeln. Mit Fx bezeichnen wir eine Formel, in der außer x keine Variablen frei vorkommen. Für einen variablenfreien term t bezeichnen wir mit Ft die Formel, die sich aus Fx ergibt durch Ersetzung aller freien Vorkommen der Variablen x durch den Term t. Mit a bezeichnen wir eine Konstante.

Axiome des Kalküls

$$A \vee \neg A$$

Die Regeln des Kalküls

Und-Einführung (UE)

$$\frac{A \quad B}{A \wedge B}$$

Und-Beseitigung (UB)

$$\frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B}$$

Oder-Einführung (OE)

$$\frac{A}{A \vee B} \qquad \frac{B}{A \vee B}$$

Oder-Beseitigung (OB)

$$\frac{A \vee B \quad \overset{[A]}{C} \quad \overset{[B]}{C}}{C}$$

All-Einführung (AE)

$$\frac{Fa}{\forall x\, Fx}$$ mit Eigenvariablenbedingung

a darf weder in $\forall x\, Fx$ vorkommen noch in einer Annahmeformel, von der diese abhängt.

All-Beseitigung (AB)

$$\frac{\forall x\, Fx}{Ft}$$

Existenz-Einführung (EE)

$$\frac{Ft}{\exists x\, Fx}$$

Existenz-Beseitigung (EB)

$$\frac{\exists x\, Fx \qquad \overset{[Fa]}{C}}{C} \qquad \text{mit Eigenvariablenbedingung}$$

a darf weder in $\exists x\, Fx$ vorkommen noch in der Oberformel C noch in einer Annahmeformel, von der diese abhängt außer in dem angegebenen Fa.

Folgt-Einführung (FE)

$$\frac{\overset{[A]}{B}}{A \to B}$$

Folgt-Beseitigung (FB)

$$\frac{A \quad A \to B}{B}$$

Nicht-Einführung (NE)

$$\frac{\overset{[A]}{\bot}}{\neg A}$$

Nicht-Beseitigung (NB)

$$\frac{A \quad \neg A}{\bot}$$

Ex-falso-quodlibet

$$\frac{\bot}{D}$$

Definition 82 Unter einer *Herleitung* im Kalkül des natürlichen Schließens versteht man einen Baum, dessen Knoten mit Formeln markiert sind wie oben beschrieben, in dem jeder innere Knoten Konklusion eines der oben angeführten Schlüsse ist und dessen Nachfolger die Prämissen desselben Schlusses sind. Dabei hänge die Wurzel des Baumes von keiner Annahme mehr ab. Wir sprechen dann von einer Herleitung der Formel an der Wurzel des Baumes und sagen dann auch, diese Formel sei *herleitbar* im Kalkül des natürlichen Schließens.

Es gilt dann der Adäquatheitssatz (Korrektheitssatz und Vollständigkeitssatz).

Satz 30 *Eine geschlossene Formel ist genau dann allgemeingültig, wenn sie im Kalkül des natürlichen Schließens herleitbar ist.*

Der Sequenzenkalkül

Hat man eine Formel B im Kalkül des natürlichen Schließens unter gewissen Annahmen $A_1, \ldots, A_m$ bewiesen, so kann man dies als sogenannte *Sequenz* $A_1, \ldots, A_m \Rightarrow B$ notieren. Eine solche Sequenz hat dann in etwa die gleiche Bedeutung wie die Formel $A_1 \wedge \cdots \wedge A_m \to B$. Man kann nun einen Kalkül der Sequenzen aufstellen, bei dem Sequenzen bewiesen werden ohne Annahmen machen zu müssen. Im Sequenzenkalkül wird das noch verallgemeinert, indem man statt B mehrere Formeln zulässt. Eine Sequenz $A_1, \ldots, A_m \Rightarrow B_1, \ldots, B_n$ hat dann die gleiche Bedeutung wie die Formel $A_1 \wedge \cdots \wedge A_m \to B_1 \vee \cdots \vee B_n$ oder wie die Formel $\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n$. Die kommaseparierte Formelfolge $A_1, \ldots, A_m$ vor dem Sequenzenpfeil heißt das *Antezedens* der Sequenz und die kommaseparierte Formelfolge $B_1, \ldots, B_n$ hinter dem Sequenzenpfeil heißt das *Sukzedens* der Sequenz. Das Antezedens einer Sequenz kann auch leer sein. Dann ist $m = 0$. Eine

solche Sequenz $\Rightarrow B_1, \ldots, B_n$ hat die gleiche Bedeutung wie die Formel $B_1 \vee \cdots \vee B_n$. Ebenso kann das Sukzedens einer Sequenz leer sein. Dann ist $n = 0$. Eine solche Sequenz $A_1, \ldots, A_m \Rightarrow$ hat die gleiche Bedeutung wie die Formel $\neg(A_1 \wedge \cdots \wedge A_m)$ oder wie die Formel $\neg A_1 \vee \cdots \vee \neg A_m$. Die leere Sequenz $\Rightarrow$, bei der sowohl das Antezedens als auch das Sukzedens leer ist, hat die gleiche Bedeutung wie das Falsum $\bot$, ist also immer falsch. Im Folgenden bezeichnen wir kommaseparierte (eventuell leere) endliche Formelfolgen mit großen griechischen Buchstaben.

Axiome

$$D \Rightarrow D$$

Regeln

Verdünnung

$$\frac{\Gamma \Rightarrow \Theta}{D, \Gamma \Rightarrow \Theta} \qquad \frac{\Gamma \Rightarrow \Theta}{\Gamma \Rightarrow \Theta, D}$$

Zusammenziehung

$$\frac{D, D, \Gamma \Rightarrow \Theta}{D, \Gamma \Rightarrow \Theta} \qquad \frac{\Gamma \Rightarrow \Theta, D, D}{\Gamma \Rightarrow \Theta, D}$$

Vertauschung

$$\frac{\Delta, D, E, \Gamma \Rightarrow \Theta}{\Delta, E, D, \Gamma \Rightarrow \Theta} \qquad \frac{\Gamma \Rightarrow \Theta, E, D, \Lambda}{\Gamma \Rightarrow \Theta, D, E, \Lambda}$$

Schnitt

$$\frac{\Gamma \Rightarrow \Theta, D \quad D, \Delta \Rightarrow \Lambda}{\Gamma, \Delta \Rightarrow \Theta, \Lambda}$$

UES

$$\frac{\Gamma \Rightarrow \Theta, A \quad \Gamma \Rightarrow \Theta, B}{\Gamma \Rightarrow \Theta, A \wedge B}$$

UEA

$$\frac{A, \Gamma \Rightarrow \Theta}{A \wedge B, \Gamma \Rightarrow \Theta} \qquad \frac{B, \Gamma \Rightarrow \Theta}{A \wedge B, \Gamma \Rightarrow \Theta}$$

OEA

$$\frac{A, \Gamma \Rightarrow \Theta \quad B, \Gamma \Rightarrow \Theta}{A \vee B, \Gamma \Rightarrow \Theta}$$

OES

$$\frac{\Gamma \Rightarrow \Theta, A}{\Gamma \Rightarrow \Theta, A \vee B} \qquad \frac{\Gamma \Rightarrow \Theta, B}{\Gamma \Rightarrow \Theta, A \vee B}$$

AES

$$\frac{\Gamma \Rightarrow \Theta, Fa}{\Gamma \Rightarrow \Theta, \forall x\, Fx}$$

mit Eigenvariablenbedingung

EEA:

$$\frac{Fa, \Gamma \Rightarrow \Theta}{\exists x\, Fx, \Gamma \Rightarrow \Theta}$$

mit Eigenvariablenbedingung

AEA

$$\frac{Ft, \Gamma \Rightarrow \Theta}{\forall x\, Fx, \Gamma \Rightarrow \Theta}$$

EES

$$\frac{\Gamma \Rightarrow \Theta, Ft}{\Gamma \Rightarrow \Theta, \exists x\, Fx}$$

NES

$$\frac{A, \Gamma \Rightarrow \Theta}{\Gamma \Rightarrow \Theta, \neg A}$$

NEA

$$\frac{\Gamma \Rightarrow \Theta, A}{\neg A, \Gamma \Rightarrow \Theta}$$

Ein Frege-Hilbert-Kalkül

Axiome

$$A \to (B \to A)$$
$$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$$
$$(\neg A \to \neg B) \to (B \to A)$$
$$\forall x\, Fx \to Ft$$
$$\forall x (A \to B) \to (A \to \forall x\, B),$$

wenn x in A nicht frei vorkommt.

Regeln

$$\frac{A \quad A \to B}{B}$$
$$\frac{Fa}{\forall x\, Fx},$$

wenn a nicht in der Konklusion auftritt.

2.3.3 Allgemeingültigkeitsdefinierbarkeit durch eine Formel

Definition 83 Das *Herbrand-Universum* ist die Menge aller variablenfreien Terme.

Definition 84 Wir wollen mit $F(x_1, \ldots, x_n)$ eine Formel der Prädikatenlogik erster Stufe (mit Funktionszeichen) bezeichnen, die außer $x_1, \ldots, x_n$ keine Variablen frei enthält. Für variablenfreie Terme $t_1, \ldots, t_n$ wollen wir mit $F(t_1, \ldots, t_n)$ die Formel bezeichnen, die aus $F(x_1, \ldots, x_n)$ dadurch entsteht, dass darin für alle $i = 1, \ldots, n$ alle freien Vorkommen der Variablen x_i durch den Term t_i ersetzt werden. Die n-stellige Relation R auf dem Herbrand-Universum sei folgendermaßen definiert.

$$R = \{(t_1, \ldots, t_n) \mid \models F(t_1, \ldots, t_n)\}$$

Wir sagen dann, die Formel $F(x_1, \ldots, x_n)$ *definiere* die Relation R *bezüglich Allgemeingültigkeit.* Wir sagen dann auch, die Relation R sei durch eine Formel *allgemeingültigkeitsdefinierbar.*

Beispiel 49
Gegeben sei die Logiksprache mit den folgenden Funktions- und Prädikatszeichen: der Konstanten 0, dem einstelligen Funktionszeichen f und dem dreistelligen Prädikatszeichen P. Dann

besteht das Herbranduniversum aus den Termen 0, $f(0)$, $f(f(0))$, ... Dann definiert die Formel

$$\begin{gathered}\forall x\, P(x,0,x)\\ \wedge \forall x \forall y \forall z \left(P(x,y,z) \rightarrow P(x,f(y),f(z))\right)\\ \rightarrow P(x_1,x_2,x_3)\end{gathered}$$

die dreistellige Relation

$$\{(f^i(0), f^j(0), f^k(0)) \mid i+j=k\}$$

auf dem Herbrand-Universum. Betrachtet man einen Term $f^n(0)$ aus dem Herbrand-Universum als Codierung für die natürliche Zahl n (f stellt die Nachfolgerfunktion dar), so entspricht diese Relation gerade dem Graphen der Additionsfunktion. Wir wollen den Term $f^n(0)$, der die natürliche Zahl n codiert, auch mit $\underline{n}$ bezeichnen.

Die Allgemeingültigkeit der obigen Formel ist äquivalent zur Unerfüllbarkeit der Formel

$$\begin{gathered}\forall x\, P(x,0,x)\\ \wedge \forall x \forall y \forall z \left(\neg P(x,y,z) \vee P(x,f(y),f(z))\right)\\ \wedge \neg P(x_1,x_2,x_3),\end{gathered}$$

die in Prolognotation folgendermaßen ausschaut.

```
p(X,0,X).
p(X,f(Y),f(Z)) :- p(X,Y,Z).
?- p(X1,X2,X3).
```

Allgemein gilt der

Satz 31 *Sei P ein reines Prologprogramm (aus Hornklauseln) und G eine Anfrageklausel. Seien $x_1, \ldots, x_n$ paarweise verschiedene Prologvariablen derart, dass in G außer $x_1, \ldots, x_n$ keine weiteren Variablen vorkommen. Dann wird die Menge aller n-Tupel $(t_1, \ldots, t_n)$ von Termen derart, dass $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ eine korrekte Antwortsubstitution für P und G ist, durch eine Formel definiert.*

Da, wie wir gesehen haben, die μ-partiellrekursiven Funktionen durch reine Prologprogramme implementiert werden können, ist also der Graph einer μ-partiellrekursiven Funktion stets durch eine Formel allgemeingültigkeitsdefinierbar.

Definition 85 Wir sagen, eine n-stellige partielle Funktion auf dem Herbrand-Universum sei durch eine Formel *allgemeingültigkeitsdefinierbar*, wenn ihr Graph durch eine Formel allgemeingültigkeitsdefinierbar ist.

Definition 86 Wir sagen, eine n-stellige Relation R auf $\mathbb{N}$ sei durch eine Formel *allgemeingültigkeitsdefinierbar*, wenn die n-stellige Relation $\{(\underline{i_1}, \ldots, \underline{i_n}) \mid (i_1, \ldots, i_n) \in R\}$ auf dem Herbrand-Universum einer geeigneten Sprache der Logik erster Stufe, die zumindest die Konstante 0 und das einstellige Funktionszeichen f enthält, durch eine Formel allgemeingültigkeitsdefinierbar ist. Wir sagen, eine n-stellige partielle zahlentheoretische Funktion sei durch eine Formel *allgemeingültigkeitsdefinierbar*, wenn ihr Graph durch eine Formel allgemeingültigkeitsdefinierbar ist.

Satz 32 *Sei h eine partielle zahlentheoretische Funktion. Dann sind die folgenden drei Aussagen zeinander äquivalent.*

1. *h ist μ-partiellrekursiv.*

2. *h ist in reinem Prolog implementierbar.*

3. *h ist durch eine Formel allgemeingültigkeitsdefinierbar.*

Die Implikationen 1. $\Longrightarrow$ 2. und 2. $\Longrightarrow$ 3. ergeben sich nach dem eben Gesagten. Die Implikation 3. $\Longrightarrow$ 1. ist etwas aufwendiger zu beweisen.

Die Idee beim Beweis von 3. $\Longrightarrow$ 1. ist die folgende. Sei h durch eine Formel $F(x_1, \ldots, x_n, y)$ allgemeingültigkeitsdefinierbar.

****

EVENTUELL NOCH DAS FOLGENDE AN GEEIGNETER STELLE EINFÜGEN!

Dann gilt für $i_1, \ldots, i_n, j \in \mathbb{N}$ und für einen geeigneten Kalkül, etwa den Frege-Hilbert-Kalkül:

$$\begin{aligned} h(i_1, \ldots, i_n) = j &\iff (i_1, \ldots, i_n, j) \in \mathrm{Graph}(h) \\ &\iff \models F(\underline{i_1}, \ldots, \underline{i_n}, \underline{j}) \\ &\iff \vdash F(\underline{i_1}, \ldots, \underline{i_n}, \underline{j}) \\ &\iff \text{es gibt eine Herleitung von } F(\underline{i_1}, \ldots, \underline{i_n}, \underline{j}) \\ &\iff \text{es gibt ein } k \in \mathbb{N} \text{ so, dass } k \text{ Gödelnummer einer Herleitung von } F(\underline{i_1}, \ldots, \underline{i_n}, \underline{j}) \text{ ist.} \end{aligned}$$

****

Dann für jedes $(i_1, \ldots, i_n, j) \in \mathrm{Graph}(h)$ die Formel $F(\underline{i_1}, \ldots, \underline{i_n}, \underline{j})$ allgemeingültig und es gibt daher eine Herleitung von $F(\underline{i_1}, \ldots, \underline{i_n}, \underline{j})$ in einem geeigneten Kalkül der Logik erster Stufe, etwa in einem Frege-Hilbert-Kalkül. Betrachten wir nun die syntaktischen Objekte der Sprache der Logik erster Stufe und des Kalküls. Das sind etwa Terme, Formeln, Herleitungen, usw. Jedes solche syntaktische Objekt X kann codiert werden als natürliche Zahl $\ulcorner X \urcorner$, genannt die *Gödelnummer* von X nach Kurt Gödel, der das Verfahren 1931 erstmalig einführte. Nun kann man zu gegebenen $i_1, \ldots, i_n \in \mathbb{N}$ den Funktionswert $h(i_1, \ldots, i_n)$ folgendermaßen berechnen, falls er existiert.

Suche die kleinste natürliche Zahl k, die Gödelnummer einer Herleitung einer Formel der Form $F(\underline{i_1}, \ldots, \underline{i_n}, \underline{j})$ ist mit $j \in \mathbb{N}$. Extrahiere daraus die letzte Formel $F(\underline{i_1}, \ldots, \underline{i_n}, \underline{j})$ der Herleitung und aus dieser wiederum die Zahl j. Diese Zahl ist das gesuchte $h(i_1, \ldots, i_n)$. Wenn es keine solche Herleitung gibt, ist auch $h(i_1, \ldots, i_n)$ undefiniert.

Definieren wir also ein $(n+1)$-stelliges zahlentheoretisches Prädikat P dadurch, dass $P(i_1, \ldots, i_n, k)$ genau dann gelte, wenn es ein $j \in \mathbb{N}$ gibt so, dass k Gödelnummer einer Herleitung der Formel $F(\underline{i_1}, \ldots, \underline{i_n}, \underline{j})$ ist. Weiter definieren wir eine $(n+1)$-stellige zahlentheoretische Funktion g derart, dass in diesem Fall $g(k) = j$ gilt. Dann ist $h(i_1, \ldots, i_n) = g((\mu P)(i_1, \ldots, i_n))$. Man kann zeigen, dass das Prädikat P primitivrekursiv ist und dass man es auch so einrichten kann, dass die Funktion g primitivrekursiv ist. Damit ist h μ-partiellrekursiv.

2.3.4 Gödelisierung

Für $k \in \mathbb{N} \setminus \{0\}$ sei p_k die k-te Primzahl.

Wir ordnen den Grundzeichen des Frege-Hilbert-Kalküls eineindeutig natürliche Zahlen zu:

- Den Zeichen „$\neg$“, „$\rightarrow$“, „(“, „)“, „$\forall$“ bzw. „,“ die Zahlen 1, 3, 5, 7, 9, bzw. 11.

- Der k-ten Variablen x_k mit $k = 1, 2, \ldots$ die Zahl 13^k
- Dem k-ten n-stelligen Funktionszeichen die Zahl p_{2k+5}^{n+1}, insbesondere der Konstanten 0 die Zahl 17 und dem Zeichen f für die Nachfolgerfunktion die Zahl 289.
- Dem k-ten n-stelligen Prädikatszeichen die Zahl p_{2k+6}^{n+1}.

Einer endlichen Folge von Objekten, denen die Zahlen $n_1, n_2 \ldots, n_k$ zugeordnet sind, ordnen wir die Zahl $2^{n_1} \cdot 3^{n_2} \cdot \cdots \cdot p_k^{n_k}$ zu. So wird jedem Wort über der Menge der Grundzeichen, insbesondere also auch jeder Formel, eine natürliche Zahl zugeordnet. Weiter wird jeder endlichen Folge von Formeln, insbesondere also auch jeder Herleitung im Kalkül, eine natürliche Zahl zugeordnet.

Einige primitivrekursive Prädikate und Funktionen

1. $x|y :\Longleftrightarrow \exists z \leq y\colon xz = y$.
   x ist Teiler von y.
2. $\mathrm{Prim}(x) :\Longleftrightarrow \neg\exists z \leq x\colon (z \neq 1 \wedge z \neq x \wedge z|x) \wedge x > 1$.
   x ist Primzahl.
3. $\mathrm{pr}(0, x) := 0$.
   $\mathrm{pr}(n+1, x) := \mu y \leq x\colon (\mathrm{Prim}(y) \wedge y|x \wedge y > \mathrm{pr}(n, x))$.
   $\mathrm{pr}(n, x)$ ist die n-te Primzahl, die Teiler von x ist.
4. $0! := 1$.
   $(n+1)! := (n+1) \cdot n!$.
5. $p_0 := 0$.
   $p_{n+1} := \mu y \leq p_n! + 1\colon (\mathrm{Prim}(y) \wedge y > p_n)$.
   p_n ist die n-te Primzahl.
6. $\mathrm{gl}(n, x) := \mu y \leq x\colon (p_n^y | x \wedge \neg p_n^{y+1} | x)$
   Ist x die Gödelnummer einer endlichen Folge von Objekten, so ist $\mathrm{gl}(n, x)$ die Gödelnummer des n-ten Gliedes dieser Folge, falls $n > 0$ und n nicht größer als die Länge der Folge ist.
7. $l(x) := \mu y \leq x\colon (p_y | x \wedge \neg p_{y+1} | x)$.
   Ist x die Gödelnummer einer endlichen Folge von Objekten, so ist $l(x)$ ist die Länge dieser Folge.
8. $x * y := \mu z \leq p_{l(x)+l(y)}^{x+y}\colon \big(\forall n \leq l(x)\colon \mathrm{gl}(n, z) = \mathrm{gl}(n, x) \wedge \forall n \leq l(y)\colon (n > 0 \Rightarrow \mathrm{gl}(n + l(x), z) = \mathrm{gl}(n, y))\big)$.
   Sind x und y die Gödelnummern zweier Folgen von Objekten, so ist $x * y$ die Gödelnummer der Konkatenation dieser beiden Folgen.
9. $r(x) := 2^x$.
   Ist x die Gödelnummer eines Objekts, so ist $r(x)$ die Gödelnummer der Folge, die nur aus diesem einen Objekt besteht.
10. $\mathrm{Var}(x) :\Longleftrightarrow \exists k \leq x\colon (x = 13^k \wedge k > 0)$.
    x ist Gödelnummer einer Variablen.
11. $\mathrm{Fktz}(n, x) :\Longleftrightarrow \exists k \leq x\colon (x = p_{2k+5}^{n+1} \wedge k > 0)$.
    x ist Gödelnummer eines n-stelligen Funktionszeichens.

12. $\mathrm{Prdz}(n,x) \iff \exists k \le x\colon (x = p_{2k+6}^{n+1} \wedge k > 0)$.
x ist Gödelnummer eines n-stelligen Prädikatszeichens.

13. $\mathrm{ksFlg}(x,1,y) :\iff \exists z \le l(x)\colon (z > 0 \wedge y = \mathrm{gl}(z,x))$.
$\mathrm{ksFlg}(x,n+1,y) :\iff \exists u \le p_{xn+n}^{x+\ulcorner,\urcorner}\colon \exists z \le l(x)\colon (z > 0 \wedge \mathrm{ksFlg}(x,n,u) \wedge y = u * r(\ulcorner,\urcorner) * \mathrm{gl}(z,x))$.
Wenn x Gödelnummer einer endlichen Folge X von Wörtern ist und n eine natürliche Zahl ist, dann gilt $\mathrm{ksFlg}(x,n,y)$ genau dann, wenn y Gödelnummer einer kommaseparierten nichtleeren Folge von n Wörtern aus X ist. Dabei darf ein Wort aus X auch mehrfach vorkommen.

14. $\mathrm{Trm}(x,y) :\iff \exists n \le y\colon \exists f \le x\colon \big(\mathrm{Fktz}(n,f) \wedge ((n = 0 \wedge y = r(f)) \vee (n > 0 \wedge \exists u \le p_{xn+n}^{x+\ulcorner,\urcorner}\colon (\mathrm{ksFlg}(x,n,u) \wedge y = r(f) * r(\ulcorner(\urcorner) * u * r(\ulcorner)\urcorner)))))\big)$.
Wenn x Gödelnummer einer endlichen Folge X von Termen ist, dann gilt $\mathrm{Trm}(x,y)$ genau dann, wenn y Gödelnummer eines Terms ist, der aus Termen aus X durch Anwendung eines Funktionszeichens gebildet ist.

15. $\mathrm{TermF}(x) :\iff l(x) > 0 \wedge \forall z \le l(x)\colon \big(z > 0 \Rightarrow \exists x_1, y, x_2 \le x\colon (x = x_1 * r(y) * x_2 \wedge l(x_1) + 1 = z \wedge (\mathrm{Var}(y) \vee \mathrm{Trm}(x_1,y)))\big)$.
x ist Gödelnummer einer endlichen nichtleeren Folge von Termen, deren jeder eine Variable ist oder aus vorausgehenden Termen mit einem Funktionszeichen gebildet ist.

16. $\mathrm{Term}(x) :\iff \exists u \le p_x^x\colon (\mathrm{TermF}(u) \wedge \mathrm{gl}(l(u),u) = x)$.
x ist Gödelnummer eines Terms.

17. $\mathrm{Atomf}(x) :\iff \exists n,p,y,f \le x\colon (x = r(p) * y \wedge \mathrm{Prdz}(n,p) \wedge \mathrm{Fktz}(n,f) \wedge \mathrm{Term}(r(f) * y))$.
x ist Gödelnummer einer Atomformel.

18. $\mathrm{neg}(x) := r(\ulcorner\neg\urcorner) * x$.
Wenn x Gödelnummer einer Formel ist, dann ist $\mathrm{neg}(x)$ die Gödelnummer ihrer Negation.

19. $\mathrm{imp}(x,y) := r(\ulcorner(\urcorner) * x * r(\ulcorner\to\urcorner) * y * r(\ulcorner)\urcorner)$.
Wenn x und y Gödelnummern zweier Formeln sind, dann ist $\mathrm{imp}(x,y)$ die Gödelnummer der daraus gebildeten Implikation.

20. $\mathrm{gen}(x,y) := r(\ulcorner\forall\urcorner) * r(x) * y$.
Wenn x die Gödelnummer einer Varibalen ist und y die Gödelnummer einer Formel, dann ist $\mathrm{gen}(x,y)$ die Gödelnummer der daraus durch Generalisation (Anwendung des Allquantors) gebildeten Formel.

21. $\mathrm{zahl}(0) := r(\ulcorner 0\urcorner)$.
$\mathrm{zahl}(n+1) := r(\ulcorner f\urcorner) * r(\ulcorner(\urcorner) * \mathrm{zahl}(n) * r(\ulcorner)\urcorner)$.
$\mathrm{zahl}(n)$ ist die Gödelnummer des Terms, der die natürliche Zahl n darstellt.

22. $\mathrm{Op}(x,y,z) :\iff x = \mathrm{neg}(y) \vee x = \mathrm{imp}(y,z) \vee \exists v \le x\colon (\mathrm{Var}(v) \wedge x = \mathrm{gen}(v,y))$.
x ist Gödelnummer einer Formel, die aus der Formel mit der Gödelnummer y und ggf. der Formel mit der Gödelnummer z durch eine logische Operation Negation, Implikation oder Generalisation entstanden ist.

23. $\mathrm{FormF}(x) :\iff l(x) > 0 \wedge \forall n \le l(x)\colon \big(n > 0 \Rightarrow \mathrm{Atomf}(\mathrm{gl}(n,x)) \vee \exists p,q < n\colon (p > 0 \wedge q > 0 \wedge \mathrm{Op}(\mathrm{gl}(n,x),\mathrm{gl}(p,x),\mathrm{gl}(q,x)))\big)$.
x ist Gödelnummer einer endlichen Folge von Formeln, deren jede eine Atomformel ist oder aus vorausgehenden Formeln mit einem der logischen Zeichen $\neg$, $\to$, $\forall$ gebildet ist.

24. $\mathrm{Form}(x) :\Longleftrightarrow \exists y \leq p_x^x \colon (\mathrm{FormF}(y) \wedge x = \mathrm{gl}(l(y), y))$.
x ist Gödelnummer einer Formel.

25. $\mathrm{Geb}(v, n, x) :\Longleftrightarrow \mathrm{Var}(v) \wedge \mathrm{Form}(x) \wedge \exists a, b, c \leq x \colon (x = a * \mathrm{gen}(v, b) * c \wedge \mathrm{Form}(b) \wedge l(a) + 2 \leq n \leq l(a) + l(\mathrm{gen}(v, b)))$.
Die Variable mit der Gödelnummer v ist in der Formel mit Gödelnummer x an n-ter Stelle gebunden.

26. $\mathrm{Fre}(v, n, x) :\Longleftrightarrow \mathrm{Var}(v) \wedge \mathrm{Form}(x) \wedge v = \mathrm{gl}(n, x) \wedge \neg \mathrm{Geb}(v, n, x)$.
Das Zeichen an n-ter Stelle in der Formel mit der Gödelnummer x ist die Variable mit der Gödelnummer v und sie ist dort frei.

27. $\mathrm{Fr}(v, x) :\Longleftrightarrow \exists n \leq l(x) \colon \mathrm{Fre}(v, n, x)$.
Die Variable mit der Gödelnummer v kommt in der Formel mit der Gödelnummer x frei vor.

28. $\mathrm{su}(x, n, y) := \mu z \leq p_{l(x)+l(y)}^{x+y} \colon \exists u, v \leq x \colon (x = u * r(\mathrm{gl}(n, x)) * v \wedge z = u * y * v \wedge n = l(u) + 1)$.
$\mathrm{su}(x, n, y)$ ist die Gödelnummer des Wortes, das aus dem Wort mit der Gödelnummer x entsteht, wenn man das Zeichen an der Stelle n durch das Wort mit der Gödelnummer y ersetzt.

29. $\mathrm{st}(0, v, x) := \mu n \leq l(x) \colon (\mathrm{Fre}(v, n, x) \wedge \neg \exists p \leq l(x) \colon (p > n \wedge \mathrm{Fre}(v, p, x)))$.
$\mathrm{st}(k + 1, v, x) := \mu n < \mathrm{st}(k, v, x) \colon (\mathrm{Fre}(v, n, x) \wedge \neg \exists p < \mathrm{St}(k, v, x) \colon (p > n \wedge \mathrm{Fre}(v, p, x)))$.
$\mathrm{st}(k, v, x)$ ist die $(k + 1)$-tletzte Stelle in der Formel mit der Gödelnummer x, an der die Variable mit der Gödelnummer v frei vorkommt (0, falls es keine solche Stelle gibt).

30. $a(v, x) := \mu n \leq l(x) \colon \mathrm{st}(n, v, x) = 0$.
Die Anzahl der Stellen, an denen die Variable mit der Gödelnummer v in der Formel mit der Gödelnummer x frei vorkommt.

31. $\mathrm{sb}_0(x_y^v) := x$.
$\mathrm{sb}_{k+1}(x_y^v) := \mathrm{su}(\mathrm{sb}_k(x_y^v), \mathrm{st}(k, v, x), y)$.
$\mathrm{sb}_k(x_y^v)$ ist die Gödelnummer der Formel, die aus der Formel mit der Gödelnummer x entsteht durch Ersetzung der letzten k freien Vorkommen der Variable mit der Gödelnummer v durch den Term mit der Gödelnummer y.

32. $\mathrm{sb}(x_y^v) := \mathrm{sb}_{a(v,x)}(x_y^v)$.
Die Gödelnummer der Formel, die aus der Formel mit der Gödelnummer x entsteht durch Ersetzung aller freien Vorkommen der Variable mit der Gödelnummer v durch den Term mit der Gödelnummer y.
Statt $\mathrm{sb}(\ldots \mathrm{sb}(\mathrm{sb}(x_{y_1}^{v_1})_{y_2}^{v_2}) \ldots {}_{y_n}^{v_n})$ schreiben wir kurz $\mathrm{sb}(x_{y_1\, y_2}^{v_1\, v_2} \ldots {}_{y_n}^{v_n})$.

33. $\mathrm{Ax}_1(x) :\Longleftrightarrow \exists a, b \leq x \colon (\mathrm{Form(a)} \wedge \mathrm{Form}(b) \wedge x = \mathrm{imp}(a, \mathrm{imp}(b, a)))$.
x ist Gödelnummer eines Axioms 1.

34. $\mathrm{Ax}_2(x) :\Longleftrightarrow \exists a, b, c \leq x \colon (\mathrm{Form}(a) \wedge \mathrm{Form}(b) \wedge \mathrm{Form}(c) \wedge x = \mathrm{imp}(\mathrm{imp}(a, \mathrm{imp}(b, c)), \mathrm{imp}(\mathrm{imp}(a, b), \mathrm{imp}(a, c))))$.
x ist Gödelnummer eines Axioms 2.

35. $\mathrm{Ax}_3(x) :\Longleftrightarrow \exists a, b \leq x \colon (\mathrm{Form}(a) \wedge \mathrm{Form}(b) \wedge x = \mathrm{imp}(\mathrm{imp}(\mathrm{neg}(a), \mathrm{neg}(b)), \mathrm{imp}(b, a)))$.
x ist Gödelnummer eines Axioms 3.

36. $\mathrm{Ax}_4(w) :\Longleftrightarrow \exists x, u, t \leq w \colon (\mathrm{Var}(x) \wedge \mathrm{Form}(u) \wedge \mathrm{Term}(t) \wedge x = \mathrm{imp}(\mathrm{gen}(x, u), \mathrm{sb}(u_t^x)))$.
x ist Gödelnummer eines Axioms 4.

37. $\mathrm{Ax}_5(w) \iff \exists x, a, b \leq w\colon (\mathrm{Var}(x) \wedge \mathrm{Form(a)} \wedge \mathrm{Form}(b) \wedge w = \mathrm{imp}(\mathrm{gen}(x, \mathrm{imp}(a, b)), \mathrm{imp}(a, \mathrm{gen}(x, b))) \wedge \neg \mathrm{Fr}(x, a))$.
w ist Gödelnummer eines Axioms 5.

38. $\mathrm{Ax}(w) :\iff \mathrm{Ax}_1(w) \vee \mathrm{Ax}_2(w) \vee \mathrm{Ax}_3(w) \vee \mathrm{Ax}_4(w) \vee \mathrm{Ax}_5(w)$.
w ist Gödelnummer eines Axioms.

39. $R_1(b, a, x) :\iff x = \mathrm{imp}(a, b)$.
Die Formel mit der Gödelnummer b folgt aufgrund von Regel 1 unmittelbar aus der Formel mit der Gödelnummer a und der Formel mit der Gödelnummer x.

40. $R_2(w, u) :\iff \exists a \leq u\colon \exists x, y \leq w\colon (\mathrm{Fktz}(0, a) \wedge \mathrm{Var}(x) \wedge \mathrm{Form}(y) \wedge u = \mathrm{sb}(y_a^x) \wedge w = \mathrm{gen}(x, y) \wedge \neg \exists w_1, w_2 \leq w\colon w = w_1 * r(a) * w_2)$.
Die Formel mit der Gödelnummer w folgt aufgrund von Regel 2 unmittelbar aus der Formel mit der Gödelnummer u.

41. $\mathrm{Fl}(x, y, z) :\iff R_1(x, y, z) \vee R_2(x, y)$.
Die Formel mit der Gödelnummer x folgt unmittelbar aus den Formeln mit den Gödelnummern y und z gemäß Regel 1 oder Regel 2.

42. $\mathrm{Bw}(x) :\iff l(x) > 0 \wedge \forall n \leq l(x)\colon \big(n > 0 \Rightarrow \mathrm{Ax}(\mathrm{gl}(n, x)) \vee \exists p, q < n\colon (p, q > 0 \wedge \mathrm{Fl}(\mathrm{gl}(n, x), \mathrm{gl}(p, x), \mathrm{gl}(q, x)))\big)$.
x ist Gödelnummer einer Herleitung (eines Beweises) im Frege-Hilbert-Kalkül.

43. $B(x, y) :\iff \mathrm{Bw}(x) \wedge \mathrm{gl}(l(x), x) = y$.
x ist Gödelnummer eines Beweises der Formel mit der Gödelnummer y.

44. $\mathrm{Bew}(x) :\iff \exists y\, B(y, x)$.
x ist Gödelnummer einer beweisbaren Formel.

Alle diese zahlentheoretischen Funktionen und Prädikate mit Ausnahme des Prädikats Bew sind primitivrekursiv.

2.3.5 Existenzdarstellung für allgemeingültigkeitsdefinierbare Prädikate

Sei nun $F(x_1, \ldots, x_n)$ eine Formel wie in Definition 84. Sei e die Gödelnummer dieser Formel und seien $u_1, \ldots, u_n$ die Gödelnummern der Variablen $x_1, \ldots, x_n$. Seien nun $i_1, \ldots, i_n$ natürliche Zahlen. Dann sind $\mathrm{zahl}(i_1), \ldots, \mathrm{zahl}(i_n)$ die Gödelnummern der diese natürlichen Zahlen darstellenden Terme $\underline{i_1}, \ldots, \underline{i_n}$. Die Formel $F(\underline{i_1}, \ldots, \underline{i_n})$ hat dann die Gödelnummer $\mathrm{sb}(e^{u_1}_{\mathrm{zahl}(i_1)} \cdots {}^{u_n}_{\mathrm{zahl}(i_n)})$. Für das durch die Formel F definierte n-stellige zahlentheoretische Prädikat R gilt also

$$R(i_1, \ldots, i_n) \iff \mathrm{Bew}(\mathrm{sb}(e^{u_1}_{\mathrm{zahl}(i_1)} \cdots {}^{u_n}_{\mathrm{zahl}(i_n)})),$$

also

$$R(i_1, \ldots, i_n) \iff \exists y\, B(y, \mathrm{sb}(e^{u_1}_{\mathrm{zahl}(i_1)} \cdots {}^{u_n}_{\mathrm{zahl}(i_n)})).$$

Definiert man hier

$$P(i_1, \ldots, i_n, y) :\iff B(y, \mathrm{sb}(e^{u_1}_{\mathrm{zahl}(i_1)} \cdots {}^{u_n}_{\mathrm{zahl}(i_n)})),$$

so ist P ein primitivrekursives Prädikat und es gilt

$$R(i_1, \ldots, i_n) \iff \exists y\, P(i_1, \ldots, i_n, y).$$

Es folgt der

Satz 33 *Für jedes durch eine Formel allgemeingültigkeitsdefinierbare n-stellige zahlentheoretische Prädikat R gibt es ein $(n+1)$-stelliges primitivrekursives Prädikat P derart, dass für alle $i_1, \ldots, i_n \in \mathbb{N}$ gilt*

$$R(i_1, \ldots, i_n) \iff \exists y\, P(i_1, \ldots, i_n, y).$$

2.3.6 Der Normalformsatz von Kleene (Formulierung für allgemeingültigkeitsdefinierbare partielle Funktionen)

Sei nun eine r-stellige partielle zahlentheoretische Funktion h allgemeingültigkeitsdefinierbar durch eine Formel $F(x_1, \ldots, x_r, x_{r+1})$. Sei e die Gödelnummer der Formel $F(x_1, \ldots, x_r, x_{r+1})$. Seien $u_1, \ldots, u_r, v$ die Gödelnummern der Variablen $x_1, \ldots, x_r, x_{r+1}$. Weiter seien $i_1, \ldots, i_r, j$ natürliche Zahlen. Dann gilt

$$(i_1, \ldots, i_r, j) \in \mathrm{Graph}(h) \iff \exists y\, B(y, \mathrm{sb}(e^{u_1}_{\mathrm{zahl}(i_1)} \cdots {}^{u_r}_{\mathrm{zahl}(i_r)}{}^{v}_{\mathrm{zahl}(j)})).$$

Wenn $B(y, \mathrm{sb}(e^{u_1}_{\mathrm{zahl}(i_1)} \cdots {}^{u_r}_{\mathrm{zahl}(i_r)}{}^{v}_{\mathrm{zahl}(j)}))$ gilt, dann ist y die Gödelnummer eines Beweises der Formel $F(\underline{i_1}, \ldots, \underline{i_r}, \underline{j})$. Nun ist $\underline{j}$ ein Teilwort des letzten Gliedes dieses Beweises[4]. Daher ist die Gödelnummer von $\underline{j}$ und damit erst recht die Zahl j kleinergleich der Zahl y.

Nun definieren wir für alle $e, i_1, \ldots, i_r, y \in \mathbb{N}$

$$P_r(e, i_1, \ldots, i_r, y) :\iff \exists j \leq y\colon B(y, \mathrm{sb}(e^{u_1}_{\mathrm{zahl}(i_1)} \cdots {}^{u_r}_{\mathrm{zahl}(i_r)}{}^{v}_{\mathrm{zahl}(j)})).$$

Dann ist P_r ein primitivrekursives $(r+2)$-stelliges Prädikat und es gilt

$$(i_1, \ldots, i_r) \in \mathrm{Def}(h) \iff \exists y\, P_r(e, i_1, \ldots, i_r, y).$$

Nun wird eine $(r+2)$-stellige primitivrekursive Funktion T_r definiert durch

$$T_r(e, i_1, \ldots, i_r, y) := \begin{cases} 0, & \text{wenn } P_r(e, i_1, \ldots, i_r, y) \\ 1 & \text{sonst.} \end{cases}$$

Dann ist $(\mu T_r)(e, i_1, \ldots, i_r)$ die Gödelnummer eines Beweises der Formel $F(\underline{i_1}, \ldots, \underline{i_r}, \underline{j})$ mit $j = h(i_1, \ldots, i_r)$, falls $(i_1, \ldots, i_r) \in \mathrm{Def}(h)$ ist, und undefiniert sonst.

Wenn h durch eine Formel $G(x_1, \ldots, x_{r+1})$ bezüglich Allgemeingültigkeit definiert wird, so auch durch die Formel $\forall x_1 \ldots \forall x_{r+1}(G(x_1 \ldots, x_{r+1}) \to P(x_1, \ldots, x_{r+1})) \to P(x_1, \ldots, x_{r+1})$, wobei P ein $(r+1)$-stelliges Prädikatszeichen ist, das in G nicht vorkommt. Wir können also in obigen Ausführungen ohne Beschränkung der Allgemeinheit annehmen, dass $F(x_1, \ldots, x_{n+1})$ die Form $(A \to P(x_1, \ldots, x_{n+1}))$ hat, wobei A eine geschlossene Formel ist und P ein $(r+1)$-stelliges Prädikatszeichen ist. Unter dieser Annahme lässt sich der Funktionswert j leicht aus dem Beweis ablesen.

$U(y) := \mu j \leq y\colon \exists x \leq y\colon \mathrm{gl}(l(y), y) = x * \mathrm{zahl}(j) * r(\ulcorner)\urcorner) * r(\ulcorner)\urcorner)$.

Nun gilt

$$h(i_1, \ldots, i_r) = \begin{cases} (U(\mu T_r))(e, i_1, \ldots, i_r), & \text{falls dies definiert ist} \\ \text{undefiniert} & \text{sonst.} \end{cases}$$

[4] Denn wegen der Rechtseindeutigkeit der Funktion h muss die Variable x_{r+1} in der Formel $F(x_1, \ldots, x_r, x_{r+1})$ frei vorkommen.

Satz 34 *Es gibt eine 1-stellige primitivrekursive Funktion U und zu jedem $r \in \mathbb{N}$ eine $(r+2)$-stellige primitivrekursive Funktion T_r so, dass für jede r-stellige allgemeingültigkeitsdefinierbare partielle zahlentheoretische Funktion h eine natürliche Zahl e existiert so, dass für alle $\mathfrak{x} \in \mathbb{N}^r$ gilt*

$$h(\mathfrak{x}) = \begin{cases} (U(\mu T_r))(e, \mathfrak{x}), & \textit{falls dies definiert ist} \\ \textit{undefiniert} & \textit{sonst.} \end{cases}$$

Diese Zahl nennt man einen *Index* der Funktion h. Man schreibt dann oft $h = \{e\}^r$.

2.3.7 Äquivalenz von μ-Partiellrekursivität und Allgemeingültigkeitsdefinierbarkeit

Aus dem letzten Satz folgt, dass jede allgemeingültigkeitsdefinierbare partielle zahlentheoretische Funktion μ-partiellrekursiv ist. Hiermit haben wir die Äquivalenz der Begriffe der μ-Partiellrekursivität, der Allgemeingültigkeitsdefinierbarkeit und der Implementierbarkeit in reinem Prolog für partielle zahlentheoretische Funktionen.

Die zweistellige partielle zahlentheoretische Funktion $g := (U(\mu T_1))$ ist eine *universelle* μ-partiellrekursive Funktion. Dies bedeutet, sie erlaubt jede beliebige einstellige (und damit mit Hilfe der Cantor-Tupel-Codierung auch jede beliebigstellige) μ-partiellrekursive Funktion h zu berechnen, da $h(x) = g(e, x)$ ist, wobei e ein Index von h ist.

Etwas ähnliches macht ein Interpreter für eine Programmiersprache. Der Interpreter (entspricht dem g) bekommt als Eingabe das Programm (entspricht dem e) und die Eingabe für das Programm (entspricht dem x). Ein Interpreter ist in diesem Sinne ein universelles Programm.

Weiter bemerken wir, dass man aus einem μ-partiellrekursiven Funktional den Index der zugehörigen partiellen zahlentheoretischen Funktion explizit berechnen kann.

2.3.8 Unentscheidbarkeit der Existenz eines Wertes

Die zweistellige zahlentheoretische Funktion g sei folgendermaßen definiert.

$$g(e, x) := \begin{cases} 1, & \text{wenn } \{e\}^1(x) \text{ definiert ist} \\ 0 & \text{sonst.} \end{cases}$$

Dann ist die Frage, ob eine gegebene μ-partiellrekursive Funktion an einer gegebenen Stelle einen Wert hat, genau dann entscheidbar, wenn die Funktion g berechenbar ist.

Wir zeigen, dass g nicht μ-rekursiv ist (also nicht im System der μ-partiellrekursiven Funktionen berechenbar ist). Wäre nämlich g μ-rekursiv, so wäre die partielle zahlentheoretische Funktion h, die definiert ist durch

$$h(x) := \mu y \, (g(x, x))$$

μ-partiellrekursiv. Sie würde also einen Index e besitzen. Dann ist $h(e)$ genau dann definiert, wenn $g(e, e) = 0$ ist, also genau dann, wenn $\{e\}^1(e)$ undefiniert ist, also genau dann, wenn $h(e)$ undefiniert ist. Dies wäre ein Widerspruch.

2.3.9 Unentscheidbarkeitssätze der Logik

Hier einige Sätze oder Abwandlungen von solchen von Kurt Gödel zur Unentscheidbarkeit in der Logik.

Satz 35 *Zu jedem korrekten formalen System der Zahlentheorie gibt es eine geschlossene zahlentheoretische Formel F so, dass weder F noch $\neg F$ in dem System beweisbar ist.*

Beweisidee. Man definiert eine Formel Fx_1 so, dass $F^{\ulcorner}Gx_1{}^{\urcorner}$ besagt, dass $G^{\ulcorner}Gx_1{}^{\urcorner}$ nicht beweisbar ist. Also

$$Fx_1 := \neg\mathrm{Bew}(\mathrm{sb}(x_1{}^{u_1}_{\mathrm{zahl}(x_1)})).$$

Weiter sei H die Formel $F^{\ulcorner}Fx_1{}^{\urcorner}$. Dann besagt die Formel H, dass die Formel $F^{\ulcorner}Fx_1{}^{\urcorner}$, d.h. die Formel H selbst nicht beweisbar ist. Die Formel H besagt also so etwas wie „Ich bin nicht beweisbar“. Dann ist H im formalen System nicht beweisbar, denn sonst wäre H wegen der Korrektheit des Systems wahr, was aber gerade bedeutet, dass H nicht beweisbar ist und daher zu einem Widerspruch führt. Da nun H nicht beweisbar ist, folgt, dass H wahr ist, da es ja gerade seine Nicht-Beweisbarkeit behauptet. Daher kann wegen der Korrektheit des Systems auch die Formel $\neg H$ nicht beweisbar sein.

Dieser Satz besagt also, dass es für die Zahlentheorie keinen Kalkül gibt, der korrekt und vollständig ist. Dies steht im Gegensatz zur reinen Prädikatenlogik erste Stufe, wo der vorgestellte Logikkalkül ein korrekter und vollständiger Kalkül ist. In der reinen Prädikatenlogik erster Stufe gilt jedoch

Satz 36 *Die Menge aller natürlichen Zahlen, die nicht Gödelnummer einer allgemeingültigen Formel sind, ist nicht allgemeingültigkeitsdefinierbar: Es gibt keine Formel Ax_1 der Prädikatenlogik erster Stufe, die nur x_1 als freie Variable enthält, so, dass für alle geschlossenen Formeln G gilt, dass $A^{\ulcorner}G^{\urcorner}$ genau dann allgemeingültig ist, wenn G nicht allgemeingültig ist.*

Satz 37 *Die Widerspruchsfreiheit eines korrekten hinreichend ausdrucksstarken Kalküls der Zahlentheorie ist in dem Kalkül selbst nicht beweisbar.*

2.4 Kalküle

2.4.1 Grundbegriffe zu den Kalkülen

In diesem Abschnitt nehmen wir an, dass wir eine abzählbar unendliche Menge $\mathcal{S} = \{a_0, a_1, a_2, \dots\}$ von Zeichen haben, die wir *Symbole* nennen, sowie eine abzählbar unendliche Menge $\mathcal{V} = \{x_0, x_1, x_2, \dots\}$ von Zeichen, die wir *Variablen* nennen. Weiter nehmen wir an, dass $\mathcal{S}$ und $\mathcal{V}$ dijunkt sind.

Definition 87 (Regel, Kalkül)

1. Eine *Regel* ist ein $(n+1)$-Tupel $(P_1, \dots, P_n, K)$ von Wörtern über $\mathcal{S} \cup \mathcal{V}$ mit $n \in \mathbb{N}$ und derart, dass jede Variable, die im Wort K vorkommt, auch in einem der Wörter $P_1, \dots, P_n$ vorkommt. Die Wörter $P_1, \dots, P_n$ heißen die *Prämissen* und das Wort K die *Konklusion* der Regel.

2. Ein *Kalkül* ist eine endliche Folge von Regeln.

Wir schreiben eine Regel $(P_1, \dots, P_n, K)$ meist als $P_1, \dots, P_n \to K$ oder als $\dfrac{P_1 \ \dots \ P_n}{K}$. Wir schreiben einen Kalkül, indem wir seine Regeln untereinander oder (mit etwas Abstand) nebeneinander hinschreiben. Dazu müssen wir noch angeben, welche in diesen Regeln vorkommenden Zeichen Symbole und welche Variablen sind.

Beispiel 50
Die Wortfolge (a_0), die nur aus dem einen Wort a_0 besteht, ist eine Regel mit 0 Prämissen und der Konklusion a_0. Diese Regel wird geschrieben als $\to a_0$ oder als $\dfrac{}{a_0}$. Ebenso ist (x_0, x_0x_0) eine Regel mit der einzigen Prämisse x_0 und der Konklusion x_0x_0. Sie wird geschrieben als $x_0 \to x_0x_0$ oder als $\dfrac{x_0}{x_0x_0}$. Das Paar $((a_0), (x_0, x_0x_0))$ ist ein Kalkül, der auch geschrieben wird als

$$\begin{aligned} &\to a_0 \\ x_0 &\to x_0x_0 \quad . \end{aligned}$$

Beispiel 51
Seien $a, b, c \in \mathcal{S}$ und $v, x, y, z \in \mathcal{V}$. Dann ist (abc) eine Regel mit 0 Prämissen und der Konklusion abc. Diese Regel wird geschrieben als $\to abc$ oder als $\dfrac{}{abc}$. Ebenso ist $(vxyz, vyxz)$ eine Regel mit der einzigen Prämisse $vxyz$ und der Konklusion $vyxz$. Sie wird geschrieben als $vxyz \to vyxz$ oder als $\dfrac{vxyz}{vyxz}$. Die Regel (x, y, xy) hat die Prämissen x und y und die Konklusion xy. Sie wird geschrieben als $x, y \to xy$ oder als $\dfrac{x \quad y}{xy}$. Das Tripel $((abc), (vxyz, vyxz), (x, y, xy))$ ist ein Kalkül, der auch geschrieben wird als

$$\begin{aligned} &\to abc \\ vxyz &\to vyxz \\ x, y &\to xy \end{aligned}$$

An dieser Stelle möchte ich daran erinnern, dass der Begriff der Regel, allerdings in einem informellen Sinne, in dieser Vorlesung bereits an früherer Stelle aufgetaucht ist, und zwar im Abschnitt 1.5. Dort wurde im Beispiel 13 der Begriff des voll geklammerten arithmetischen Ausdrucks über `0` und `1` durch vier Regeln (im informellen Sinne) mittels induktiver Definition definiert. Durch leichte Modifikation kann man daraus Regeln in dem hier definierten formalen Sinne und damit einen Kalkül konstruieren wie folgt.

Beispiel 52
Wir nehmen an, dass a,...,z,ä,ö,ü,ß,A,...,Z,Ä,Ö,Ü, `,0,1,+,*,(,)` Symbole, also Elemente von $\mathcal{S}$ seien und dass u und v Variablen, also Elemente von $\mathcal{V}$ seien. Die vier Regeln des Kalküls sind nun die folgenden.

$$\frac{}{\texttt{0} \text{ ist ein Ausdruck}}$$

$$\frac{}{\texttt{1} \text{ ist ein Ausdruck}}$$

$$\frac{u \text{ ist ein Ausdruck} \qquad v \text{ ist ein Ausdruck}}{\texttt{(}u\texttt{+}v\texttt{)} \text{ ist ein Ausdruck}}$$

$$\frac{u \text{ ist ein Ausdruck} \qquad v \text{ ist ein Ausdruck}}{\texttt{(}u\texttt{*}v\texttt{)} \text{ ist ein Ausdruck}}$$

Wie in diesem Beispiel kann man oft induktive Definitionen fast wörtlich durch einen Kalkül wiedergeben. Oft werden durch eine induktive Definition mehrere Begriffe gleichzeitig definiert. Wenn jedoch bei einer induktiven Definition nur ein einziger Begriff (wie im Beispiel 13 der Begriff des voll geklammerten Ausdrucks über `0` und `1`) definiert wird oder zumindest im Vordergrund

des Interesses steht, ist es sinnvoll und üblich in den Regeln den diesen Begriff bezeichnenden Text wegzulassen. Man erhält dann ebenfalls einen Kalkül zur Beschreibung des induktiv definierten Begriffs wie im folgenden Beispiel.

Beispiel 53
Seien $\mathtt{0}, \mathtt{1}, \mathtt{+}, \mathtt{*}, \mathtt{(}, \mathtt{)} \in \mathcal{S}$ und $u, v \in \mathcal{V}$. Dann ist

$$\frac{}{\mathtt{0}}$$

$$\frac{}{\mathtt{1}}$$

$$\frac{u \qquad v}{\mathtt{(}u\mathtt{+}v\mathtt{)}}$$

$$\frac{u \qquad v}{\mathtt{(}u\mathtt{*}v\mathtt{)}}$$

ein Kalkül, den man auch so schreiben kann

$$\begin{aligned} &\to \mathtt{0} \\ &\to \mathtt{1} \\ u, v &\to \mathtt{(}u\mathtt{+}v\mathtt{)} \\ u, v &\to \mathtt{(}u\mathtt{*}v\mathtt{)}\ . \end{aligned}$$

Definition 88 (Instanziierung, Instanz)

1. Eine *Instanziierung* ist eine Abbildung $\iota\colon \mathcal{V} \to \mathcal{S}^*$.
2. Für $w \in (\mathcal{S} \cup \mathcal{V})^*$ und $\iota\colon \mathcal{V} \to \mathcal{S}^*$ bezeichnen wir mit $w\iota$ das Wort über $\mathcal{S}$, das sich aus dem Wort w dadurch ergibt, dass darin jedes $x \in \mathcal{V}$ durch das Wort $\iota(x)$ ersetzt wird.[5]
3. Eine *Instanz* einer Regel $P_1, \ldots, P_n \to K$ ist eine Regel $P_1\iota, \ldots, P_n\iota \to K\iota$, wobei ι eine Instanziierung ist. Wir bezeichnen sie mit $(P_1, \ldots, P_n \to K)\iota$.

Betrachten wir das Beispiel 51, so ist etwa eine Instanziierung $\iota\colon \mathcal{V} \to \mathcal{S}^*$ gegeben durch $\iota(v) := aba$, $\iota(x) := b$, $\iota(y) := ac$, $\iota(z) := \varepsilon$ und $\iota(u) := \varepsilon$ für alle anderen Variablen u. Dann gilt $(\to abc)\iota = \to abc$. Also ist die Regel $\to abc$ eine Instanz der Regel $\to abc$. Entsprechend gilt $(vxyz \to vyxz)\iota = ababac \to abaacb$. Also ist die Regel $ababac \to abaacb$ eine Instanz der Regel $vxyz \to vyxz$. Weiter gilt $(x, y \to xy)\iota = b, ac \to bac$. Also ist die Regel $b, ac \to bac$ eine Instanz der Regel $x, y \to xy$. Eine Regel, die mindestens eine Variable enthält, hat unendlich viele Instanzen.

Definition 89 (Herleitbarkeit, erzeugte Sprache) Sei K ein Kalkül. Für eine Menge $W \subseteq \mathcal{S}^*$ von Wörtern über $\mathcal{S}$ und für ein Wort $w \in \mathcal{S}^*$ wird induktiv definiert, was es heißt, dass im Kalkül K das Wort w aus W *herleitbar* sei, in Zeichen $W \vdash_K w$.

1. Wenn $w \in W$ ist, dann gilt $W \vdash_K w$.

[5] Das heißt: Wenn $w = c_1 \ldots c_n$ ist mit $c_1, \ldots, c_n \in \mathcal{S} \cup \mathcal{V}$, dann ist $w\iota$ die Konkatenation $w_1 \cdots w_n$ der Wörter w_i $(i = 1, \ldots, n)$, die definiert sind durch $w_i := \begin{cases} c_i, & \text{wenn } c_i \in \mathcal{S} \\ \iota(c_i), & \text{wenn } c_i \in \mathcal{V}. \end{cases}$

2. Wenn $W \vdash_K w_1, \ldots, W \vdash_K w_n$ gelten und $w_1, \ldots, w_n \to w$ Instanz einer Regel von K ist, dann gilt auch $W \vdash_K w$.

Statt $\emptyset \vdash_K w$ schreibt man auch $K \vdash w$ und sagt, w sei in K *herleitbar*. Die von K *erzeugte Sprache* $L(K)$ ist definiert durch $L(K) := \{w \in \mathcal{S}^* \mid K \vdash w\}$. Die von K *auf einem Alphabet* $\Delta \subseteq \mathcal{S}$ *erzeugte Sprache* $L(K, \Delta)$ ist definiert durch $L(K, \Delta) := L(K) \cap \Delta^*$. In dieser Vorlesung wollen wir eine von einem Kalkül K auf einem Alphabet Δ erzeugte Sprache $L(K, \Delta)$ kurz eine *kalkülerzeugte* Sprache nennen.

Die Bedingung in der Definition des Begriffs der Regel, dass jede in der Konklusion vorkommende Variable auch in einer der Prämissen vorkommen muss, garantiert, dass die in einem Kalkül herleitbaren Wörter nur Symbole enthalten, die auch in den Regeln des Kalküls vorkommen. Daher ist die von einem Kalkül erzeugte Sprache auch wirklich eine Sprache (über einem endlichen Alphabet). Hätte man diese Bedingung nicht, würde zum Beispiel die einzige Regel $\to x$ mit $x \in \mathcal{V}$ bereits ganz $\mathcal{S}^*$ erzeugen. Eine Alternative wäre solche Regeln zuzulassen, aber dann den Kalkül explizit auf eine endliche Menge von Symbolen einzuschränken. Die Klasse der von Kalkülen auf Alphabeten erzeugten Sprachen ist für beide möglichen Definitionen aber die gleiche Klasse. Man gewinnt also nichts durch Zulassung solcher Regeln.

Beispiel 54
Im Kalkül K von Beispiel 51 gilt $\{a, ab\} \vdash_K abcbaa$. Denn nach 1. gilt $\{a, ab\} \vdash_K a$ und $\{a, ab\} \vdash_K ab$. Nun ist aber $a, ab \to aab$ eine Instanz der dritten Regel des Kalküls (mit einer Instanziierung ι mit $\iota(x) := a$ und $\iota(y) := ab$). Nach 2. gilt daher $\{a, ab\} \vdash_K aab$. Nun ist $aab \to baa$ eine Instanz der zweiten Regel des Kalküls (mit $\iota(v) := \varepsilon$, $\iota(x) := aa$, $\iota(y) := b$ und $\iota(z) := \varepsilon$). Nach 2. folgt daher, dass $\{a, ab\} \vdash_K baa$ gilt. Nun ist $\to abc$ eine Instanz der ersten Regel des Kalküls (mit ι beliebig). Nach 2. folgt daher, dass $\{a, ab\} \vdash_K abc$ gilt. Nun ist $abc, baa \to abcbaa$ eine Instanz der dritten Regel des Kalküls (mit $\iota(x) := abc$ und $\iota(y) := baa$). Aus $\{a, ab\} \vdash_K abc$ und $\{a, ab\} \vdash_K baa$ folgt daher, dass $\{a, ab\} \vdash_K abcbaa$ gilt.

Die von K erzeugte Sprache ist die Menge aller nichtleeren Wörter über $\{a, b, c\}$, die gleich viele a's wie b's wie c's enthalten.

Nun ein Beispiel eines Kalküls, dessen erzeugte Sprache leer ist, obwohl die Menge seiner Regeln nichtleer ist.

Beispiel 55
Seien $a, b, c \in \mathcal{S}$ und $x, y \in \mathcal{V}$. Sei K der Kalkül mit der einzigen Regel

$$x, y \to xy\ .$$

Dann kann man wie folgt zeigen, dass $\{a, b\} \vdash_K aba$ gilt:
Nach 1. gilt $\{a, b\} \vdash_K a$ und $\{a, b\} \vdash_K b$. Nun ist $a, b \to ab$ eine Instanz der Regel (mit $\iota(x) := a$ und $\iota(y) := b$). Aus $\{a, b\} \vdash_K a$ und $\{a, b\} \vdash_K b$ folgt daher nach 2., dass $\{a, b\} \vdash_K ab$ gilt. Nun ist $ab, a \to aba$ ebenfalls eine Instanz der Regel (mit $\iota(x) := ab$ und $\iota(y) := a$). Aus $\{a, b\} \vdash_K ab$ und $\{a, b\} \vdash_K a$ folgt daher nach 2., dass $\{a, b\} \vdash_K aba$ gilt.

In K ist aus der leeren Menge $\emptyset$ kein Wort w herleitbar. Weder durch Anwendung von 1. noch von 2. kann man ein Wort als herleitbar nachweisen, da in 1. vorausgesetzt ist, dass $w \in \emptyset$, was immer falsch ist, und da in 2. vorausgesetzt ist, dass Wörter w_1, w_2 bereits aus $\emptyset$ herleitbar seien. Also ist $L(K) = \emptyset$.

Nach der Definition ist $L(K)$ die kleinste Menge, die abgeschlossen gegenüber Instanzen von Regeln von K ist. Das heißt $L(K)$ ist die kleinste Menge M derart, dass für jede Instanz

$w_1, \ldots, w_n \to w$ einer Regel von K gilt: Wenn $w_1, \ldots, w_n \in M$, dann $w \in M$. Die Menge $L(K)$ ist genau dann nichtleer, wenn es in K mindestens eine Regel gibt, die keine Prämisse hat, also eine Regel der Form $\to w$. Ist $\to w$ eine Regel von K, so nennt man das Wort w ein *Axiom* von K. Ein Axiom ist stets ein Wort über $\mathcal{S}$. Jedes Axiom eines Kalküls ist Element der von diesem Kalkül erzeugten Sprache.

Definition 90 (Herleitung) Sei K ein Kalkül und sei $W \subseteq \mathcal{S}^*$.

1. Eine *Herleitung* in K aus W ist eine endliche oder unendliche Folge $(w_1, w_2, \ldots)$ von Wörtern aus $\mathcal{S}^*$ so, dass für jedes $j \in \mathbb{N} \setminus \{0\}$, das kleinergleich der Anzahl der Glieder der Folge ist, $w_j \in W$ gilt oder natürliche Zahlen n und $j_1, \ldots, j_n < j$ existieren so, dass $w_{j_1}, \ldots, w_{j_n} \to w_j$ eine Instanz einer Regel von K ist.
2. Bei einer endlichen Herleitung $(w_1, \ldots, w_n)$ spricht man auch von einer *Herleitung des Wortes w_n aus der Wortmenge W* in K. Im Fall $W = \emptyset$ spricht man einfach von einer *Herleitung des Wortes w_n* in K. Die Zahl n heißt die *Länge* der Herleitung. Man spricht bei einer Herleitung der Länge n auch von einer Herleitung in n *Schritten*.

Ein Wort w ist genau dann in einem Kalkül K aus einer Wortmenge W herleitbar, wenn es eine Herleitung von w aus W in K gibt. Die Axiome eines Kalküls K sind genau die Wörter, die in K in einem Schritt herleitbar sind.

Als Beispiel betrachten wir den Kalkül von Beispiel 53. In diesem Kalkül ist die endliche Folge (`0`, `1`, `(0+1)`, `((0+1)*1)`) von Wörtern eine Herleitung des Wortes `((0+1)*1)`. Denn sei $w_1 :=$ `0`, $w_2 :=$ `1`, $w_3 :=$ `(0+1)` und $w_4 :=$ `((0+1)*1)`. Dann ist zu zeigen, dass es zu jedem $j \in \{1, 2, 3, 4\}$ ein $n \in \mathbb{N}$ und $j_1, \ldots, j_n < j$ gibt so, dass $w_{j_1}, \ldots, w_{j_n} \to w_j$ eine Instanz einer Regel des Kalküls ist. Für $j = 1$ folgt dies, indem man $n = 0$ wählt, da $\to$ `0` eine Instanz der ersten Regel des Kalküls ist. Für $j = 1$ folgt dies ebenfalls, indem man $n = 0$ wählt, da $\to$ `1` eine Instanz der ersten Regel des Kalküls ist. Für $j = 3$ folgt es, indem man $n = 2$ und $j_1 = 1$ und $j_2 = 2$ wählt, da `0`, `1` $\to$ `(0+1)` eine Instanz der dritten Regel des Kalküls ist. Für $j = 4$ folgt es, indem man $n = 2$ und $j_1 = 3$ und $j_2 = 2$ wählt, da `(0+1)`, `1` $\to$ `((0+1)*1)` eine Instanz der vierten Regel des Kalküls ist. Man leitet also die Glieder der Herleitung eines nach dem anderen jeweils aus bereits hergeleiteten, also in der Herleitung bereits an früherer Stelle auftretenden Gliedern der Herleitung her. Die Axiome dises Kalküls sind `0` und `1`.

Beispiel 56
Seien $a, b, c, S, T \in \mathcal{S}$ und $x, y \in \mathcal{V}$. Sei K der Kalkül

$$\begin{aligned}
&\to Sa && \text{(1. Regel)}\\
&\to Sb && \text{(2. Regel)}\\
&\to Sc && \text{(3. Regel)}\\
&\to T && \text{(4. Regel)}\\
Sx &\to Tx && \text{(5. Regel)}\\
Sx, Ty &\to Txyx && \text{(6. Regel)}\\
Tx &\to x\ . && \text{(7. Regel)}
\end{aligned}$$

Dann sind in K genau die folgenden Wörter herleitbar:

1. Sa, Sb, Sc.
2. Alle Wörter Tw mit $w \in \{a, b, c\}^*$ und $w^R = w$.[6]

[6] Zur Erinnerung: Mit w^R hatten wir die Spiegelung des Wortes w bezeichnet.

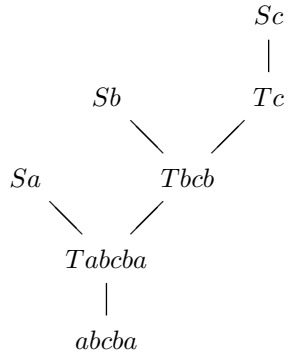
3. Alle Wörter $w \in \{a, b, c\}^*$ mit $w^R = w$.

Nun sei $\Delta = \{a, b, c\}$. Dann ist $L(K, \Delta) = \{w \in \Delta^* \mid w^R = w\}$. Z.B. ist $abcba \in L(K, \Delta)$. Eine Herleitung der Länge 7 von $abcba$ ist $(Sc, Tc, Sb, Tbcb, Sa, Tabcba, abcba)$. Denn zunächst leitet man Sc nach der dritten Regel her, dann daraus Tc nach der fünften Regel, dann Sb nach der zweiten Regel. Aus Sb und Tc ergibt sich $Tbcb$ nach der sechsten Regel, wobei die Variable x durch das Wort b und die Variable y durch das Wort c ersetzt wird. Aus der ersten Regel erhält man Sa. Aus Sa und $Tbcb$ ergibt sich $Tabcba$ nach der sechsten Regel, wobei die Variable x durch das Wort a und die Variable y durch das Wort bcb ersetzt wird. Aus $Tabcba$ erhält man schließlich $abcba$ mit der siebenten Regel, wobei die Variable x durch das Wort $abcba$ ersetzt wird.

Man schreibt eine Herleitung oft als eine Aufeinanderfolge von Zeilen, wobei jedes Glied der Herleitung in eine eigene Zeile geschrieben wird, mit einer Nummer versehen wird und mit Informationen darüber versehen wird, aus welchen darüber gelegenen Zeilen es sich als Konklusion mittels welcher Regel ergibt. Im letzten Beispiel schaut das dann so aus.

(1)	Sc	nach der 3. Regel
(2)	Tc	aus (1) nach der 5. Regel
(3)	Sb	nach der 2. Regel
(4)	$Tbcb$	aus (3) und (2) nach der 6. Regel
(5)	Sa	nach der 1. Regel
(6)	$Tabcba$	aus (5) und (4) nach der 6. Regel
(7)	$abcba$	aus (6) nach der 7. Regel

Die Struktur einer endlichen Herleitung kann man auch in Baumform darstellen, etwa im letzten Beipiel als



Dabei ist jeder Knoten des Baumes mit der Konklusion einer Instanz einer Regel des Kalküls markiert und jeweils seine Söhne (in der Abbildung die mit ihm verbundenen Knoten darüber) mit den Prämissen.

Beispiel 57
Seien $|, +, *, =$, $NAT \in \mathcal{S}$ und $u, x, y, z \in \mathcal{V}$. Sei K der Kalkül

$$\begin{aligned} &\rightarrow NAT \\ NAT\, x &\rightarrow NAT\, x| \\ NAT\, x,\ NAT\, y &\rightarrow x + y = xy \\ NAT\, x &\rightarrow x * = \\ x * y = z, z + x = u &\rightarrow x * y| = u \end{aligned}$$

Dann besteht $L(K)$ aus den Wörtern $NAT\,|^n$ mit $n \in \mathbb{N}$, den Wörtern $|^i + |^j = |^k$ mit $i + j = k$ und den Wörtern $|^i * |^j = |^k$ mit $i \cdot j = k$. Die im Kalkül herleitbaren Wörter können als mathematische Aussagen über natürliche Zahlen interpretiert werden. Dabei ist eine natürliche Zahl n dargestellt als Strichfolge aus n Strichen, NAT steht für das Prädikat „... ist natürliche Zahl“, $+$ steht für die Addition, $*$ für die Multiplikation und $=$ für die Gleichheitsrelation auf den natürlichen Zahlen. Das im Kalkül herleitbare Wort $||| * || = ||||||$ ist dann zu interpretieren als die mathematische Aussage $3 \cdot 2 = 6$. Seine Herleitung (NAT , $NAT\,|$, $NAT\,||$, $NAT\,|||$, $|||* =$, $+||| = |||, ||| * | = |||, ||| + ||| = ||||||, ||| * || = ||||||$) kann als Beweis für die Aussage $3 \cdot 2 = 6$ betrachtet werden. Diese Herleitung kann betrachtet werden als Formalisierung des folgenden mathematischen Beweises.

> $0 \in \mathbb{N}$ (1. Peanoaxiom). Also ist nach dem 2. Peanoaxiom $1 \in \mathbb{N}$, da 1 der Nachfolger von 0 ist. Entsprechend ist $2 \in \mathbb{N}$ (der Nachfolger von 1) und daher $3 \in \mathbb{N}$ (der Nachfolger von 2). Also ist $3 \cdot 0 = 0$ nach der ersten Rekursionsgleichung für die Multiplikation. Weiter gilt $0 + 3 = 3$, wie man sich leicht ausrechnet. Nach der zweiten Rekursionsgleichung für die Multiplikation ergibt sich aus den letzteren beiden Gleichungen $3 \cdot 1 = 3 \cdot 0 + 3 = 3$. Weiter gilt $3 + 3 = 6$, wie man sich leicht ausrechnet. Nach der zweiten Rekursionsgleichung für die Multiplikation ergibt sich aus den letzteren beiden Gleichungen $3 \cdot 2 = 3 \cdot 1 + 3 = 6$.[7]

Ein wichtiges, aber nicht das einzige Anwendungsgebiet von Kalkülen ist die Formalisierung von logischen oder mathematischen Beweisen wie im letzten Beispiel. Die Idee ist, dass man zunächst eine formale Sprache aufstellt, in der man eine Klasse von Aussagen als Wörter formalisiert, die dann als diese Aussagen interpretiert werden. Dann wird ein Kalkül aufgestellt für die Herleitung solcher Wörter. Natürlich muss man dafür sorgen, dass in diesem Kalkül nur solche Wörter herleitbar sind, die einer im beabsichtigten Sinne „wahren“ Aussage entsprechen. Ein solcher Kalkül wird ein *korrekter* Kalkül (englisch *sound* calculus) genannt. Der Kalkül im Beispiel ist ein korrekter Kalkül zum Beweisen von arithmetischen Gleichungen. Einen korrekten Kalkül kann man zum rein formalen Beweisen verwenden und kann sicher sein, dass dies nie zu falschen Aussagen führen kann. Solche formalen Beweise sind auch einer Überprüfung durch einen Computer zugänglich und können oft sogar durch einen Computer automatisch gefunden werden (automatisches Theorembeweisen, automatisches Beweisen, automatische Deduktion).

Statt Herleitung sagt man auch *Ableitung*, im Englischen meist *derivation*. Nur bei einem für eine Klasse von logischen oder mathematischen Aussagen korrekten Kalkül spricht man auch von einem *(formalen) Beweis*, im Englischen *(formal) proof*. Eher auf den Prozess des Generierens einer Herleitung beziehen sich die Wörter *Inferenz*, *Deduktion*, *(formales) Beweisen* und *(formales) Schließen*, im Englischen *(formal) reasoning*.

[7] Die Addition von Zahlen ist im Kalkül einfach durch Konkatenation von Wörtern realisiert, weshalb sie hier nicht in Einzelschritte aufgelöst werden muss.

Definition 91 (Durch Kalküle entscheidbare Sprachen) Eine Sprache L heißt *durch Kalküle entscheidbar*, wenn es ein Alphabet Δ und Kalküle K_1 und K_2 gibt mit $L = L(K_1, \Delta)$ und $\Delta^* \setminus L = L(K_2, \Delta)$. Wir sagen dann auch, es sei *durch Kalküle entscheidbar, ob* ein Wort in L liegt.

Beispiel 58
Die Menge der Wörter über $\{a, b\}$ mit gerader Länge ist durch Kalküle entscheidbar. Denn sei $\Delta := \{a, b\}$, sei K_1 der Kalkül mit den Regeln $\to \varepsilon$, $x \to xaa$, $x \to xab$, $x \to xba$ und $x \to xbb$ und sei K_2 der Kalkül mit den Regeln $\to a$, $\to b$, $x \to xaa$, $x \to xab$, $x \to xba$ und $x \to xbb$. Dann ist $L(K_1, \Delta)$ die Menge der Wörter über $\{a, b\}$ mit gerader Länge und $L(K_2, \Delta)$ die Menge der Wörter über $\{a, b\}$ mit ungerader Länge. Es ist also durch Kalküle entscheidbar, ob die Länge eines Wortes über $\{a, b\}$ gerade ist.

Jede durch Kalküle entscheidbare Sprache ist auch entscheidbar, d.h. es gibt einen Algorithmus, der bei Eingabe eines Wortes die Ausgabe „ja“ liefert, wenn das Wort in der Sprache liegt, und andernfalls die Antwort „nein“ liefert. Denn die Menge der endlichen Folgen von Wörtern über $\mathcal{S}$ ist abzählbar und lässt sich auch effektiv von einem Algorithmus abzählen. Für jede solche Folge ist weiterhin entscheidbar, ob sie eine Herleitung eines gegebenen Wortes $w \in \Delta^*$ im Kalkül K_1 ist und ob sie eine Herleitung von w im Kalkül K_2 ist. Man muss nun nur nacheinander alle endlichen Wortfolgen der Abzählung daraufhin überprüfen, ob sie eine Herleitung von w in K_1 oder in K_2 sind. Ist $w \in L$, so findet man so nach endlicher Zeit eine Herleitung von w in K_1. Andernfalls, wenn also $w \in \Delta^* \setminus L$ ist, findet man nach endlicher Zeit eine Herleitung von w in K_2. Entsprechend muss $w \in L$ sein, wenn die gefundene Herleitung eine Herleitung in K_1 ist, und es muss $w \in \Delta^* \setminus L$ sein, wenn die gefundene Herleitung eine Herleitung in K_2. Hiermit hat man einen Entscheidungsalgorithmus für L.

Definition 92 (Berechnung einer partiellen Funktion durch einen Kalkül) Sei K ein Kalkül, sei $\# \in \mathcal{S}$, sei $\Delta \subseteq \mathcal{S}$ endlich mit $\# \notin \Delta$ und seien $m, n \in \mathbb{N}$. Weiter sei $f\colon (\Delta^*)^m \xrightarrow{\text{part}} (\Delta^*)^n$ so, dass gilt

$$L(K, \Delta \cup \{\#\}) = \{u_1\# \dots \#u_m\#w_1\# \dots \#w_n \mid ((u_1, \dots, u_m), (w_1, \dots, w_n)) \in \text{Graph}(f)\}.$$

Dann sagen wir, der Kalkül K *berechne* die partielle Funktion f. Wir nennen das ausgezeichnete Zeichen # das *Leerzeichen*.

Beispiel 59
Die Spiegelungsfunktion auf $\{a, b\}^*$ wird berechnet vom Kalkül

$$\to \#\qquad\qquad x\#y \to xa\#ay\qquad\qquad x\#y \to xb\#by$$

Jede durch einen Kalkül berechenbare partielle Funktion ist eine partiellberechenbare Funktion, d.h. es gibt einen Algorithmus, der bei Eingabe eines Elementes ihres Definitionsbereiches den zugehörigen Funktionswert liefert und andernfalls nicht terminiert. Denn die Menge der endlichen Folgen von Wörtern über $\mathcal{S}$ ist abzählbar und lässt sich auch effektiv von einem Algorithmus abzählen. Für jede solche Folge ist weiterhin entscheidbar, ob sie bei gegebenen Wörtern $u_1, \dots, u_m \in \Delta^*$ eine Herleitung eines Wortes $u_1\# \dots \#u_m\#w_1\# \dots \#w_n$ mit $w_1, \dots, w_n \in \Delta^*$ ist. Hiermit hat man einen Algorithmus, der zu Eingabewörtern $u_1, \dots, u_m$ Ausgabewörter $w_1, \dots, w_n$ liefert mit $(w_1, \dots, w_n) = f(u_1, \dots, u_m)$, wenn $(u_1, \dots, u_m) \in \text{Def}(f)$, und der andernfalls nicht terminiert.

2.4.2 Einige Eigenschaften des Kalkülbegriffs

Wenn ein Alphabet $\Delta \subseteq \mathcal{S}$ und zwei Kalküle K_1 und K_2 gegeben sind, so ist es oft von Interesse diese beiden Kalküle so zusammenzusetzen, dass ein neuer Kalkül K entsteht mit $L(K, \Delta) = L(K_1, \Delta) \cup L(K_2, \Delta)$. Die naheliegende Idee einfach die Regeln von beiden Kalkülen in einem neuen Kalkül zusammenzufassen funktioniert deshalb nicht, weil ein Wort w im Kalkül K_1 herleitbar sein könnte, auf das eine Regel von K_2 anwendbar ist oder umgekehrt. Man muss also die beiden Kalküle irgendwie trennen und erst im letzten Herleitungsschritt zusammenführen. Hierzu und für weitere ähnliche Anwendungen führen wir ein paar Notationen ein.

Zunächst führen wir für eine Sprache L und ein Zeichen a die folgenden Notationen ein. $La := \{wa \mid w \in L\}$ und $aL := \{aw \mid w \in L\}$. Für Kalküle führen wir die folgenden Notationen ein.

Notation 2 Seien K, K_1, K_2 Kalküle, sei $\Delta \subseteq \mathcal{S}$ endlich, sei $a \in \mathcal{S}$ und sei R eine Regel.

1. Mit $\mathcal{S}_K$ bezeichnen wir die Menge der Symbole, die in K vorkommen. Mit $\mathcal{V}_K$ bezeichnen wir die Menge der Variablen, die in K vorkommen.
2. Mit K_{Δ^*} bezeichnen wir den Kalkül mit den Regeln $\to \varepsilon$ sowie $x \to xa$ für jedes $a \in \Delta^*$.
3. Mit aK bezeichnen wir den Kalkül mit der Regel $aw_1, \ldots, aw_n \to aw$ für jede Regel $w_1, \ldots, w_n \to w$ des Kalküls K. Mit Ka bezeichnen wir den Kalkül mit der Regel $w_1a, \ldots, w_na \to wa$ für jede Regel $w_1, \ldots, w_n \to w$ des Kalküls K.
4. Mit $K_1 + K_2$ bezeichnen wir den Kalkül mit den Regeln von K_1 gefolgt von den Regeln von K_2. Mit $K + (R)$ bezeichnen wir den Kalkül mit den Regeln von K gefolgt von der Regel R. Mit $(R) + K$ bezeichnen wir den Kalkül mit der Regel R gefolgt von den Regeln von K.

Sei nun K ein Kalkül und $a \in \mathcal{S}$. Dann ist $L(Ka) = L(K)a$ und $L(aK) = aL(K)$. Außerdem gilt dann für jedes endliche $W \subseteq \mathcal{S}^*$ und jedes $w \in \mathcal{S}^*$ mit $w \notin W$:

$$W \vdash_{Ka} w \iff \bigvee_{v \in \mathcal{S}^*} (\{u \in \mathcal{S}^* \mid ua \in W\} \vdash_K v \text{ und } w = va)$$
$$W \vdash_{aK} w \iff \bigvee_{v \in \mathcal{S}^*} (\{u \in \mathcal{S}^* \mid au \in W\} \vdash_K v \text{ und } w = av).$$

Sind nun K_1 und K_2 zwei Kalküle und sind $a, b \in \mathcal{S} \setminus (\mathcal{S}_{K_1} \cup \mathcal{S}_{K_2})$ mit $a \neq b$ und ist K der Kalkül $K_1a + K_2b + (xa \to x) + (xb \to x)$, so ist $L(K) = L(K_1)a \cup L(K_2)b \cup L(K_1) \cup L(K_2)$ und $L(K, \mathcal{S}_{K_1} \cup \mathcal{S}_{K_2}) = L(K_1) \cup L(K_2)$. Die Vereinigung zweier kalkülerzeugter Sprachen ist somit wieder eine kalkülerzeugte Sprache.

Ist K ein Kalkül, $\Delta \subseteq \mathcal{S}$ endlich und $a, b, c \in \mathcal{S} \setminus (\mathcal{S}_K \cup \Delta)$ paarweise verschieden, dann ist $Ka + K_{\Delta^*}b + (xa, xb \to xc)$ ein Kalkül K' mit

$$L(K') = L(K)a \cup \Delta^*b \cup L(K, \Delta)c.$$

Zu jeder kalkülerzeugten Sprache $L(K, \Delta)$ gibt es also einen Kalkül K' und ein $c \in \mathcal{S} \setminus \Delta$ so, dass $L(K, \Delta) = \{w \mid wc \in L(K')\}$ ist.

Sind K_1 und K_2 Kalküle und $a, b \in \mathcal{S} \setminus (\mathcal{S}_{K_1} \cup \mathcal{S}_{K_2})$ mit $a \neq b$, so ist $K_1a + K_2b + (xa, xb \to x)$ ein Kalkül K mit $L(K) \cap (\mathcal{S} \setminus \{a, b\})^* = L(K_1) \cap L(K_2)$. Hieraus folgt, dass der Durchschnitt zweier kaklülerzeugter Sprachen wieder eine kalkülerzeugte Sprache ist.

Satz 38 *Sei Δ eine endliche Teilmenge von $\mathcal{S}$ und es seien $\#, E, A$ drei paarweise verschiedene Symbole aus $\mathcal{S}$ mit $\#, E, A \notin \Delta$. Eine partielle Funktion $f\colon (\Delta^*)^m \xrightarrow{\text{part}} (\Delta^*)^n$ wird genau dann durch einen Kalkül berechnet, wenn es einen Kalkül K gibt, sodass für alle $u_1, \ldots, u_m, w_1, \ldots, w_n \in \Delta^*$ gilt:*

$$((u_1, \ldots, u_m), (w_1, \ldots, w_n)) \in \text{Graph}(f) \iff \{u_1\# \ldots \#u_m E\} \vdash_K w_1\# \ldots \#w_n A.$$

Beweis: Wir beweisen zunächst die „nur-dann"-Richtung, dann die „dann"-Richtung:

1. $f\colon (\Delta^*)^m \xrightarrow{\text{part}} (\Delta^*)^n$ werde durch einen Kalkül K_1 berechnet. O.b.d.A.[8] können wir annehmen, dass E und A nicht in den Regeln von K_1 vorkommen. Andernfalls benennen wir in K_1 die Symbole E und A vorher entsprechend um zu neuen Symbolen E' und A'. Außerdem können wir o.B.d.A. annehmen, dass jede Prämisse einer Regel von K_1 mindestens ein Symbol aus $\mathcal{S} \setminus (\Delta \cup \{\#\})$ enthält. Andernfalls können wir ein neues solches Symbol, sagen wir a, hinten an jede Prämisse und an die Konklusion jeder Regel von K_1 anfügen und die Regel $xa \to x$ hinzufügen und erhalten einen Kalkül, der f berechnet und in dem jede Prämisse einer Regel das Symbol $a \in \mathcal{S} \setminus (\Delta \cup \{\#\})$ enthält. Nun sei K_2 der Kalkül, der aus den Regeln von K_1 und der zusätzlichen Regel $x_1\# \ldots \#x_m E, x_1\# \ldots \#x_m\#y_1\# \ldots \#y_n \to y_1\# \ldots \#y_n A$ besteht. Nun seien $u_1, \ldots, u_m \in \Delta^*$. Dann sind die in K_2 aus der einelementigen Menge $\{u_1\# \ldots \#u_m E\}$ herleitbaren Wörter genau die Wörter aus $L(K_1)$, das Wort $\{u_1\# \ldots \#u_m E\}$ und im Falle, dass $(u_1, \ldots, u_m) \in \text{Def}(f)$ ist, auch das Wort $y_1\# \ldots \#y_n A$, wobei $(y_1, \ldots, y_n) := f(x_1, \ldots, x_m)$ sei.

2. Sei K_2 ein Kalkül, sodass für alle $u_1, \ldots, u_m, w_1, \ldots, w_n \in \Delta^*$ gilt:

   $$((u_1, \ldots, u_m), (w_1, \ldots, w_n)) \in \text{Graph}(f) \iff \{u_1\# \ldots \#u_m E\} \vdash_{K_2} w_1\# \ldots \#w_n A.$$

   Es ist zu zeigen, dass ein Kalkül K_1 existiert, der f berechnet. Die Idee zum Beweis ist die folgende. Sei x eine Variable, die nicht in K_2 auftritt. Sei Σ die Menge, die besteht aus den Symbolen, die in Regeln von K_2 vorkommen, sowie den Symbolen E und A. Seien $h, w \in \mathcal{S} \setminus \Sigma$ mit $h \neq w$. Nun wird ein Kalkül K_1' definiert, der den Herleitbarkeitsbegriff in K_2 formalisiert. Es soll dabei xhy informell interpretierbar sein als Aussage „y ist in K_2 aus $\{x\}$ herleitbar". Hierzu werden als Hilfsmittel Aussagen der Form „x ist ein Wort aus Σ^*" benötigt, die in K_1' durch wx formalisiert werden. Nun sei K_1' der Kalkül mit den folgenden Regeln.

   (a) Die Regel $\to w$ sowie die Regel $wx \to wxa$ für jedes $a \in \Sigma$.

   (b) Die Regel $wx \to xhx$.

   (c) Die Regel $wx, xhP_1, \ldots, xhP_n \to xhK$ für jede Regel $P_1, \ldots, P_n \to K$ des Kalküls K_2.

   (d) Die Regel $xEhyA \to x\#y$.

   Dann sind genau die folgenden Wörter in K_1 herleitbar.

   (a) Die Wörter wv mit $v \in \Sigma^*$.

   (b) Die Wörter $v_1 h v_2$ mit $v_1, v_2 \in \Sigma^*$ und $\{v_1\} \vdash_{K_2} v_2$.

   (c) Die Wörter $v_1\#v_2$ mit $v_1, v_2 \in \Sigma^*$ und $\{v_1 E\} \vdash_{K_2} v_2 A$.

[8] Ohne Beschränkung der Allgemeinheit

Nun könnte es aber sein, dass für zwei Wörter u und v zwar $\{uE\} \vdash_{K_2} vA$ gilt, aber u und v nicht die gewünschte syntaktische Form haben. Wir müssen daher die Regeln für w geeignet einschränken, damit wu nur noch dann herleitbar ist, wenn u die Form $u_1\#\ldots\#u_m$ mit $u_1,\ldots,u_m$ hat. Ebenso müssen wir die Regel $xEhyA \to x\#y$ so einschränken, dass nur solche Instanziierungen von ihr zulässig sind, für die y zu einem Wort $w_1\#\ldots\#w_n$ mit $w_1,\ldots,w_m$ instanziiert wird. Sei also K_1 der Kalkül mit den folgenden Regeln.

(a) Die Regel $\to D$ sowie die Regel $Dx \to Dxa$ für jedes $a \in \Delta$.

(b) Die Regel $Dx_1,\ldots,Dx_m \to wx_1\#\ldots\#x_mE$.

(c) Die Regel $wx \to xhx$.

(d) Die Regel $wx, xhP_1,\ldots,xhP_n \to xhK$ für jede Regel $P_1,\ldots,P_n \to K$ des Kalküls K_2.

(e) Die Regel $Dy_1,\ldots,Dy_n, xEhy_1\#\ldots\#y_nA \to x\#y_1\#\ldots\#y_n$.

Dabei sei $D \in \mathcal{S} \setminus (\Sigma \cup \{h,w\})$. Dann ist K_1 der gewünschte Kalkül.

q.e.d.

Im Satz kann man sich das Wort $u_1\#\ldots\#u_mE$ interpretiert als die Aussage „$(u_1,\ldots,u_m)$ ist die Eingabe“ und das Wort $w_1,\ldots,w_nA$ interpretiert als die Aussage „$(w_1,\ldots,w_n)$ ist die Ausgabe eines Algorithmus“ vorstellen. Tatsächlich stellt ein Kalkül auch einen nichtdeterministischen Algorithmus dar.

2.4.3 Kalküle und Logik

Wie bereits erwähnt, ist die Logik ein wichtiges Anwendungsgebiet der Kalküle. Als einfachstes Beispiel einer Logik betrachten wir die *Aussagenlogik*, auf die wir auch an späterer in dieser Vorlesung wieder bei der Untersuchung von Problemen aus der Komplexitätstheorie stoßen werden. In der Aussagenlogik betrachtet man eine abzählbar unendliche Menge von Zeichen, die man *Aussagenvariablen* nennt. Die Formeln sind Wörter, die außer den Aussagenvariablen noch *Junktoren* (Symbole für aussagenlogische Verknüpfungen) und runde Klammern „(“ und „)“ enthalten dürfen. In der Wahl der Junktoren hat man gewisse Freiheiten, man muss aber die Menge der zu verwendenden Junktoren fest wählen. Eine Möglichkeit ist als Junktoren die folgenden Zeichen zu wählen: „$\neg$“ für „nicht“, „$\wedge$“ für „und“ und „$\vee$“ für „oder“. Die *Formeln* werden induktiv definiert:

1. Jede Aussagenvariable ist eine Formel.
2. Wenn F eine Formel ist, dann ist auch $\neg F$ eine Formel.
3. Wenn F und G Formeln sind, dann ist auch $(F \wedge G)$ eine Formel.
4. Wenn F und G Formeln sind, dann ist auch $(F \vee G)$ eine Formel.

Da wir in dieser Vorlesung nur endliche Alphabete zugelassen haben und Kalküle immer nur endlich viele Regeln haben dürfen, müssen wir zur Beschreibung des Formelbegriffs durch einen Kalkül die Aussagenvariablen durch Wörter über einem endlichen Alphabet ausdrücken. Wir nehmen hierzu an, dass die folgenden Zeichen Symbole aus $\mathcal{S}$ seien: a,...,z,A,...,Z,X,,$\neg$,$\wedge$,$\vee$,(,)

und nehmen als Aussagenvariablen die Wörter X, $X\prime$, $X\prime\prime$, usw. Weiter nehmen wir an $F, G, H \in \mathcal{V}$. Dann ist ein Kalkül zur Beschreibung des Begriffs der Formel gegeben durch

$$\frac{}{\text{Aussagenvariable } X} \qquad \frac{\text{Aussagenvariable } F}{\text{Aussagenvariable } F\prime}$$

$$\frac{\text{Aussagenvariable } F}{\text{Formel } F} \qquad \frac{\text{Formel } F}{\text{Formel } \neg F}$$

$$\frac{\text{Formel } F \qquad \text{Formel } G}{\text{Formel } (F \wedge G)} \qquad \frac{\text{Formel } F \qquad \text{Formel } G}{\text{Formel } (F \vee G)}$$

Eine andere Möglichkeit ist die folgenden Junktoren zu wählen: „$\neg$“ für „nicht“ und „$\rightarrow$“ für „impliziert“. Zur Formalisierung in einem Kalkül nehmen wir an, dass die folgenden Zeichen Symbole aus $\mathcal{S}$ seien: a,...,z,A,...,Z,X,$\prime$,$\neg$,$\rightarrow$,(,) und dass $F, G, H \in \mathcal{V}$ seien. Da das Zeichen $\rightarrow$ dabei ein Symbol aus $\mathcal{S}$ ist, dürfen wir hier natürlich nicht dasselbe Symbol als Trennungssymbol zwischen Prämissen und Konklusion einer Regel verwenden sondern müssen dafür (wie auch schon oben geschehen) den horizontalen Strich verwenden. Die Regeln zur Bildung von Formeln schauen dann so aus.

$$\frac{}{\text{Aussagenvariable } X} \qquad \frac{\text{Aussagenvariable } F}{\text{Aussagenvariable } F\prime}$$

$$\frac{\text{Aussagenvariable } F}{\text{Formel } F} \qquad \frac{\text{Formel } F}{\text{Formel } \neg F}$$

$$\frac{\text{Formel } F \qquad \text{Formel } G}{\text{Formel } (F \rightarrow G)}$$

Eine *Interpretation* (oder *Belegung*) ist eine Abbildung, die jeder Aussagenvariablen einen Wahrheitswert w (wahr) oder f (falsch) zuordnet. Jede Interpretation induziert auch eine Zuordnung eines Wahrheitswertes zu jeder Formel. Eine Formel ist also bei einer Interpretation entweder wahr oder falsch (näheres dazu im Kapitel Komplexitätstheorie). Eine Formel heißt *allgemeingültig*, wenn sie bei jeder Interpretation wahr ist. In der Logik werden Kalküle zum Nachweis der Allgemeingültigkeit von Formeln aufgestellt. Für die Aussagenlogik bei Verwendung der beiden Junktoren $\neg$ und $\rightarrow$ ist ein solcher Kalkül etwa der, der aus den obigen Regeln zusammen mit den folgenden Regeln besteht.

$$\frac{\text{Formel } F \qquad \text{Formel } G}{(F \rightarrow (G \rightarrow F))}$$

$$\frac{\text{Formel } F \qquad \text{Formel } G \qquad \text{Formel } H}{((F \rightarrow (G \rightarrow H)) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H)))}$$

$$\frac{\text{Formel } F \qquad \text{Formel } G}{((\neg F \rightarrow \neg G) \rightarrow (G \rightarrow F))}$$

$$\frac{F \qquad (F \rightarrow G)}{G}$$

Bemerkung: In der Logik ist es üblich, nachdem der Begriff der Formel einmal definiert ist, beim Hinschreiben des Kalküls für die allgemeingültigen Formeln die Regeln für den Begriff der Formel wegzulassen und in den Regeln für allgemeingültige Formeln die Prämissen Formel F, Formel G, Formel H wegzulassen und anstattdessen dazuzusagen, dass F, G und H für Formeln stehen sollen. Es ist in der Logik weiterhin üblich Regeln, die nach Weglassung dieser Prämissen keine Prämissen mehr haben, als Axiomenschemata zu bezeichnen und darin die horizontalen Striche ebenfalls wegzulassen. Ebenso ist es in der Logik üblich bei der Notation Klammern wegzulassen, wo dies nicht zu Missverständnissen führt. So werden üblicherweise Präzedenzregeln zur Klammerersparnis vereinbart, z.B., dass $\neg$ stärker bindet als alle anderen Junktoren und dass $\to$ rechtsassoziativ ist. Man schreibt den obigen Kalkül in der Logik also üblicherweise folgendermaßen:

Axiome:

$$F \to G \to F$$
$$(F \to G \to H) \to (F \to G) \to F \to H$$
$$(\neg F \to \neg G) \to G \to F$$

Regeln:

$$\frac{F \qquad F \to G}{G}$$

Ich möchte aber darauf hinweisen, dass eine solche Schreibweise nicht mit der in dieser Vorlesung vorgestellten übereinstimmt.

2.4.4 Berechenbarkeit der μ-partiellrekursiven Funktionen durch Kalküle

Nun werden wir einen Kalkül angeben zur Berechnung des Wertes eines μ-partiellrekursiven Funktionals an einer Argumentstelle gibt. Hierzu stellen wir hier der Einfachheit halber Funktionale dar als Wörter über dem Alphabet $\{O, N, P, ,, ', (,), R, \mu\}$, wobei die Projektionsfunktionale $\mathtt{P_i^n}$ dargestellt werden als $P, \ldots, \ldots'\ldots'$ jeweils mit i tiefgestellten und n hochgestellten Strichen. In den nun folgenden Kalkülen werden weiter Symbole verwendet zur Beschreibung sowohl von Objekten als auch von Sachverhalten (Aussagen). So wird der senkrechte Strich | zur Darstellung von natürlichen Zahlen verwendet. Eine natürliche Zahl n wird dargestellt durch das Wort $|^n$, das aus n Strichen besteht. Das leere Wort symbolisiert also die Zahl 0. Mit νx wird im folgenden Kalkül die Aussage dargestellt, dass x eine natürliche Zahl ist, mit jti die Aussage, dass j der tiefgestellte Index i ist, mit khn die Aussage, dass k der hochgestellte Index n ist und mit $f\pi i\pi n$ die Aussage, dass f das i-te n-stellige Projektionsfunktional ist. Wir nehmen in diesem Abschnitt der Vorlesung an, dass

$$O, N, P, ,, ', (,), R, \mu, |, \nu, t, h, \pi, \triangle, \square, \#, \tau, \circ, \bullet, * \in \mathcal{S}$$
$$x, y, z, i, m, n, j, k, f, g, \mathfrak{x}, \mathfrak{y}, \mathfrak{g} \in \mathcal{V}$$

gelten mögen. Sei K_1 der Kalkül

$$\to \nu \qquad \nu x \to \nu x| \qquad \to ,t| \qquad jti \to j,ti|$$
$$\to 'h| \qquad khn \to k'hn| \qquad jti, khim \to Pjk\pi i\pi im$$

Dann sind in K_1 etwa die Wörter $\nu|||$ und $P_{\prime\prime}{}^{\prime\prime\prime\prime}\pi||\pi|||||$ herleitbar, deren Interpretationen die Aussagen „3 ist eine natürliche Zahl“ bzw. „$\mathtt{P_2{}^5}$ ist das 2-te 5-stellige Projektionsfunktional“ sind. Informell kann man die fünf Regeln des Kalküls folgendermaßen lesen.

1. 0 ist eine natürliche Zahl.
2. Wenn x eine natürliche Zahl ist, dann ist $x + 1$ eine natürliche Zahl.
3. $\mathtt{{}_1}$ ist der untere Index 1.
4. Wenn $\mathtt{{}_j}$ der untere Index i ist, dann ist $\mathtt{{}_{j+1}}$ der untere Index $i + 1$.
5. $\mathtt{{}^1}$ ist der obere Index 1.
6. Wenn $\mathtt{{}^k}$ der obere Index n ist, dann ist $\mathtt{{}^{k+1}}$ der obere Index $n + 1$.
7. Wenn $\mathtt{{}_j}$ der untere Index i ist und $\mathtt{{}^k}$ der obere Index $i + m$ ist, dann ist $\mathtt{P_j{}^k}$ das i-te $(i + m)$-stellige Projektionsfunktional.

Als nächstes wird ein Kalkül aufgestellt zur Bestimmung der Stelligkeit eines μ-partiellrekursiven Funktionals. Und zwar wird durch $f\triangle n$ die Aussage dargestellt, dass f ein n-stelliges μ-partiellrekursives Funktional sei. Weiter wird mit $\mathfrak{g}\square m\square n$ symbolisiert, dass $\mathfrak{g}$ die Konkatenation von m n-stelligen Funktionalen sei. Wenn wir also m n-stellige Funktionale $g_1, \ldots, g_m$ haben, dann soll „gelten“ $g_1 \ldots g_m\square m\square n$. Sei K_2 der Kalkül mit den Regeln von K_1 sowie den Regeln

$$\begin{array}{cccc} \to O\triangle & \to N\triangle| & f\pi i\pi n \to f\triangle n & g\triangle n \to g\square|\square n \\ \mathfrak{g}\square m\square n, g\triangle n \to \mathfrak{g}g\square m|\square n & f\triangle m, \mathfrak{g}\square m\square n \to (f\mathfrak{g})\triangle n & f\triangle n, g\triangle n|| \to (Rfg)\triangle n| & f\triangle n| \to (\mu f)\triangle n \end{array}$$

Dann ist K_2 ein Kalkül zur Berechnung der Stelligkeit eines μ-partiellrekursiven Funktionals. Auch in diesem Kalkül sind aus wahren Aussagen immer nur wieder wahre Aussagen herleitbar. Betrachtet man das Zeichen $\triangle$ als Leerzeichen, so berechnet dieser Kalkül die partielle Funktion auf $\{O, N, P, ,, ', (,), R, \mu, |\}^*$, die jedem μ-partiellrekursiven Funktional seine Stelligkeit in Strichkodierung zuordnet. Zum Beispiel ist in K_2 das Wort $(NN)\triangle|$ herleitbar, das der Aussage „$\mathtt{(NN)}$ ist ein 1-stelliges μ-partiellrekursives Funktional“ entspricht. Die acht neu hinzugekommenen Regeln des Kalküls kann man informell folgendermaßen lesen.

1. $\mathtt{O}$ ist ein 0-stelliges Funktional.
2. $\mathtt{N}$ ist ein 1-stelliges Funktional.
3. Wenn f das i-te n-stellige Projektionsfunktional ist, dann ist f ein n-stelliges Funktional.
4. Wenn g ein n-stelliges Funktional ist, dann ist g eine Konkatenation von einem einzigen n-stelligen Funktional.
5. Wenn $\mathfrak{g}$ eine Konkatenation von m n-stelligen Funktionalen ist und g ein n-stelliges Funktional, dann ist $\mathfrak{g}g$ eine Konkatenation von $m + 1$ n-stelligen Funktionalen.
6. Wenn f ein m-stelliges Funktional ist und $\mathfrak{g}$ eine Konkatenation von m n-stelligen Funktionalen, dann ist $(f\mathfrak{g})$ ein n-stelliges Funktional.
7. Wenn f ein n-stelliges Funktional ist und g ein $(n+2)$-stelliges Funktional, dann ist (Rfg) ein $(n + 1)$-stelliges Funktional.
8. Wenn f ein $(n + 1)$-stelliges Funktional ist, dann ist (μf) ein n-stelliges Funktional.

Als nächstes stellen wir ein n-Tupel $(x_1, \ldots, x_n)$ von natürlichen Zahlen dar als das Wort $\#|^{x_1}\# \ldots \#|^{x_n}$. Mit $\mathfrak{x}\tau n$ soll die Aussage „$\mathfrak{x}$ ist ein n-Tupel von natürlichen Zahlen“ symbolisiert werden. Hierzu ist der folgendermaßen definierte Kalkül K_3 geeignet: Die Regeln von K_3 seien die Regeln von K_2 und die Regeln

$$\to \tau \qquad\qquad \mathfrak{x}\tau n, \nu x \to \mathfrak{x}\#x\tau n|$$

Dann gilt z.B. $K_3 \vdash \#||||\#||\#|||\tau|||$, da $(3, 2, 3)$ ein 3-Tupel (Tripel) von natürlichen Zahlen ist.

Nun wird durch $f \circ \mathfrak{x} \circ y$ die Aussage dargestellt „Der Wert des Funktionals f an der Stelle $\mathfrak{x}$ ist y“, durch $\mathfrak{g} \bullet \mathfrak{x} \bullet \mathfrak{y}$ die Aussage „$\mathfrak{g}$ ist eine Konkatenation von μ-partiellrekursiven Funktionalen, deren Werte an der Stelle $\mathfrak{x}$ das Tupel $\mathfrak{y}$ bilden“ und durch $f * \mathfrak{x} * y$ die Aussage „Es gibt ein $n \in \mathbb{N}$, sodass $\mathfrak{x}$ ein n-Tupel von natürlichen Zahlen, f ein $(n+1)$-stelliges μ-partiellrekursives Funktional und y eine natürliche Zahl ist und der Wert von f an der Stelle $(\mathfrak{x}, z)$ größer als Null ist für alle $z < y$“. Sei K der Kalkül mit den Regeln vom Kalkül K_3 sowie mit den Regeln

$$\begin{array}{rr}
\to O \circ \circ & \nu x \to N \circ \#x \circ x| \\
f\pi i|\pi i|m, \mathfrak{x}\tau i, \nu x, \mathfrak{y}\tau m \to f \circ \mathfrak{x}\#x\mathfrak{y} \circ x & g \circ \mathfrak{x} \circ y \to g \bullet \mathfrak{x} \bullet \#y \\
\mathfrak{g} \bullet \mathfrak{x} \bullet \mathfrak{y}, g \circ \mathfrak{x} \circ y \to \mathfrak{g}g \bullet \mathfrak{x} \bullet \mathfrak{y}\#y & \mathfrak{g} \bullet \mathfrak{x} \bullet \mathfrak{y}, f \circ \mathfrak{y} \circ z \to (f\mathfrak{g}) \circ \mathfrak{x} \circ z \\
f\triangle n, g\triangle n||, f \circ \mathfrak{x} \circ y \to (Rfg) \circ \mathfrak{x}\# \circ y & \nu x, (Rfg) \circ \mathfrak{x}\#x \circ y, g \circ \#y\mathfrak{x}\#x \circ z \to (Rfg) \circ \mathfrak{x}\#x| \circ z \\
\mathfrak{x}\tau n, f\triangle n| \to f * \mathfrak{x} * & f * \mathfrak{x} * y, f \circ \mathfrak{x}\#y \circ z| \to f * \mathfrak{x} * y| \\
f \circ \mathfrak{x}\#y \circ , f * \mathfrak{x} * y \to (\mu f) \circ \mathfrak{x} \circ y &
\end{array}$$

Dann ist K ein Kalkül zum Berechnen des Werts eines μ-partiellrekursiven Funktionals. Die elf neu hinzugekommenen Regeln des Kalküls kann man informell folgendermaßen lesen.

1. $\mathcal{W}^{\mathtt{0}}() = 0$.
2. Wenn $x \in \mathbb{N}$, dann $\mathcal{W}^{\mathtt{N}}(x) = x + 1$.
3. Wenn $f = \mathtt{P_{i+1}}^{\mathtt{i+1+m}}$, $\mathfrak{x} \in \mathbb{N}^i$, $x \in \mathbb{N}$ und $\mathfrak{y} \in \mathbb{N}^m$, dann ist $\mathcal{W}^f(\mathfrak{x}, x, \mathfrak{y}) = x$.
4. Wenn $\mathcal{W}^g(\mathfrak{x}) = y$, dann ist g ein Funktional, dessen Wert an der Stelle $\mathfrak{x}$ gleich y ist.
5. Wenn $g_1, \ldots, g_m$ Funktionale sind, deren Werte an der Stelle $\mathfrak{x}$ gleich $y_1, \ldots, y_m$ sind, und überdies $\mathcal{W}^g(\mathfrak{x}) = y$ ist, dann sind $g_1, \ldots, g_m, g$ Funktionale, deren Werte an der Stelle $\mathfrak{x}$ gleich $y_1, \ldots, y_m, y$ sind.
6. Wenn $g_1, \ldots, g_m$ Funktionale sind, deren Werte an der Stelle $\mathfrak{x}$ gleich $y_1, \ldots, y_m$ sind, und überdies $\mathcal{W}^f(y_1, \ldots, y_m) = z$ ist, so ist $\mathcal{W}^{(fg_1 \ldots g_m)}(\mathfrak{x}) = z$.
7. Wenn f ein n-stelliges und g ein $(n+2)$-stelliges Funktional ist und $\mathcal{W}^f(\mathfrak{x}) = y$ ist, dann ist $\mathcal{W}^{(Rfg)}(\mathfrak{x}, 0) = y$.
8. Wenn $x \in \mathbb{N}$ und $\mathcal{W}^{(Rfg)}(\mathfrak{x}, x) = y$ und $\mathcal{W}^g(y, \mathfrak{x}, x) = z$ ist, dann ist $\mathcal{W}^{(Rfg)}(\mathfrak{x}, x + 1) = z$.
9. Wenn $\mathfrak{x} \in \mathbb{N}^n$ und f ein $(n+1)$-stelliges Funktional ist, dann ist der Wert von f an der Stelle $(\mathfrak{x}, u)$ positiv für alle $u < 0$.
10. Wenn $\mathfrak{x} \in \mathbb{N}^n$ und $y \in \mathbb{N}$ und der Wert eines $(n+1)$-stelligen Funktionals f an der Stelle $(\mathfrak{x}, u)$ positiv ist für alle $u < y$ und $\mathcal{W}^f(\mathfrak{x}, y) = z + 1$ ist, dann ist der Wert von f an der Stelle $(\mathfrak{x}, u)$ positiv für alle $u < y + 1$.

11. Wenn $\mathcal{W}^f(\mathfrak{x}, y) = 0$ ist und der Wert von f an der Stelle $(\mathfrak{x}, u)$ positiv ist für alle $u < y$, dann ist $\mathcal{W}^{(\mu f)}(\mathfrak{x}) = y$.

Um zu sehen, welche Wörter in K herleitbar sind, bezeichnen wir für ein durch ein Wort aus Symbolen dargestelltes Objekt κ dieses Wort mit $\tilde{\kappa}$. Also ist z.B. $\tilde{n} = |^n$ für $n \in \mathbb{N}$. Man zeigt dann durch Induktion nach der Definition der Herleitbarkeit, dass jedes in K herleitbare Wort eine der folgenden Formen hat.

1. $\nu\tilde{r}$ mit $r \in \mathbb{N}$.
2. $\tilde{\mathfrak{r}}t\tilde{r}$ mit $r \in \mathbb{N}$.
3. $\tilde{\mathfrak{s}}h\tilde{s}$ mit $s \in \mathbb{N}$.
4. $\widetilde{\mathtt{P}_{\mathfrak{r}}^{\mathfrak{s}}}\pi\tilde{r}\pi\tilde{s}$ mit $r, s \in \mathbb{N}$ und $1 \leq r \leq s$.
5. $\tilde{\phi}\triangle\tilde{s}$, wobei ϕ ein s-stelliges μ-partiellrekursives Funktional ist.
6. $\tilde{\phi}_1 \ldots \tilde{\phi}_r \Box \tilde{r} \Box \tilde{s}$, wobei $\phi_1, \ldots, \phi_r$ r s-stellige μ-partiellrekursive Funktionale sind.
7. $\tilde{\mathfrak{z}}\tau\tilde{r}$ mit $\mathfrak{z} \in \mathbb{N}^r$
8. $\tilde{\phi} \circ \tilde{\mathfrak{z}} \circ \tilde{r}$ mit $\mathcal{W}^{\phi}(\mathfrak{z}) = r$.
9. $\tilde{\phi}_1 \ldots \tilde{\phi}_s \bullet \tilde{\mathfrak{z}} \bullet \tilde{r}_1 \ldots \tilde{r}_s$ mit $\mathcal{W}^{\phi_1}(\mathfrak{z}) = r_1, \ldots, \mathcal{W}^{\phi_s} = r_s$.
10. $\tilde{\phi} * \tilde{\mathfrak{z}} * \tilde{r}$, wobei $\mathfrak{z} \in \mathbb{N}^l$ und $r \in \mathbb{N}$ ist, ϕ ein $l+1$-stelliges μ-partiellrekursives Funktional ist und $\mathcal{W}^{\phi}(\mathfrak{z}, s) > 0$ ist für alle $s < r$.

Umgekehrt gilt, dass jedes dieser Wörter auch in K herleitbar ist. Dies zeigt man z.B. für die Werte von Funktionalen durch Induktion nach der Länge der Funktionale, ggf. mit Nebeninduktion nach dem letzten Argument. Es gilt also der

Satz 39 *Sei* $\Sigma := \{O, N, P, \iota, ', (,), R, \mu, |, \#\}$. *Die partielle Funktion* $h\colon (\Sigma^*)^2 \xrightarrow{\text{part}} \Sigma^*$ *sei definiert durch*

$$h(w_1, w_2) := \begin{cases} \widetilde{\mathcal{W}^{\phi}(\mathfrak{x})}, & \text{wenn } \tilde{\phi} = w_1 \text{ und } \tilde{\mathfrak{x}} = w_2 \text{ und } \mathcal{W}^{\phi}(\mathfrak{x}) \text{ existiert} \\ \text{undefiniert}, & \text{wenn es kein solches } \phi \text{ und } \mathfrak{x} \text{ gibt.} \end{cases}$$

Dann gelten die folgenden Aussagen.

1. *Der Kalkül K berechnet die partielle Funktion h mit $\circ$ als Leerzeichen.*
2. *Jede μ-partiellrekursive Funktion wird durch einen Kalkül berechnet.*

Durch Hinzufügung der Regel $\tilde{\phi} \circ \#\mathfrak{x} \circ y \to \mathfrak{x}\#y$ zum Kalkül K für ein μ-partiellrekursives Funktional ϕ sieht man, dass jede μ-partiellrekursive Funktion durch einen Kalkül berechenbar ist.

2.4.5 Kodierung von Kalkülen

Es stellt sich nun die Frage, ob es entscheidbar ist, ob ein Wort w in einem Kalkül K herleitbar ist. Wir untersuchen hierzu zunächst die Frage, ob dieses Problem durch Kalküle entscheidbar ist. Der Begriff der Entscheidbarkeit durch Kalküle ist aber nur definiert für Sprachen über einem endlichen Alphabet, während Kalküle Zeichen aus einem unendlichen Zeichenvorrat verwenden können. Wir müssen also die in Frage kommenden Paare (K, w) irgendwie als Wörter über einem endlichen Alphabet kodieren. Wir werden hierzu Kalküle, Wörter über $\mathcal{S}$ und andere Objekte kodieren als Wörter über dem Alphabet $\{0, 1\}$. Die Kodierung eines Objekts κ bezeichnen wir dabei mit $[\kappa]$.

Die Kodierung $[\kappa]$ eines Objektes κ wird folgendermaßen als ein Wort über dem Alphabet $\{0, 1\}$ definiert.

$$\begin{aligned}
\text{Symbol:} \quad & [a_n] := 1 \underbrace{0 \dots 0}_{2n+2 \text{ mal}} \\
\text{Variable:} \quad & [x_n] := 1 \underbrace{0 \dots 0}_{2n+3 \text{ mal}} \\
\text{Wort aus } (\mathcal{S} \cup \mathcal{V})^*: \quad & [z_1 \dots z_n] := [z_1] \dots [z_n] \qquad \text{für } z_1, \dots, z_n \in \mathcal{S} \cup \mathcal{V} \\
\text{Regel:} \quad & [p_1, p_2, \dots, p_n \to k] := [p_1]1[p_2]1 \dots 1[p_n]1[k]10 \\
\text{Kalkül:} \quad & [(R_1, \dots, R_n)] := [R_1] \dots [R_n]
\end{aligned}$$

Z.B. ist $[a_0 a_0] = 100100$. Ist K der Kalkül mit den beiden Regeln $\to a_0$ und $x_0 \to x_0 x_0$, dann ist $[K] = 10010100011000100010$.

Das Paar (K, w) wird kodiert als das Wort $[K]\#[w]$ über dem Alphabet $\{0, 1, \#\}$ und es stellt sich die Frage, ob die Sprache

$$L = \{[K]\#[w] \mid K \text{ ist ein Kalkül und } K \vdash w\}$$

durch Kalküle entscheidbar ist. Es wird sich herausstellen, dass dies nicht der Fall ist, dass aber diese Sprache auf dem Alphabet $\{0, 1, \#\}$ durch einen Kalkül erzeugt wird.

Den Prozess der Kodierung kann man sich in zwei Schritte aufgeteilt denken. Dazu definieren wir zunächst die Mengen

$$\begin{aligned}
\mathcal{X} &:= \mathcal{S} \cup \mathcal{V} \\
\mathcal{Z} &:= \mathcal{X} \cup \{\text{“,”}, \text{“;”}\}.
\end{aligned}$$

Die Menge $\mathcal{Z}$ besteht also aus den Symbolen, den Variablen, dem Beistrich und dem Strichpunkt. Dabei nehmen wir an, dass der Beistrich und der Strichpunkt Zeichen seien, die weder in $\mathcal{S}$ noch in $\mathcal{V}$ liegen.

1. Schritt: Ein Objekt κ wird zunächst kodiert als ein Wort $\tilde{\kappa}$ über der Zeichenmenge $\mathcal{Z}$.

$$\begin{aligned}
\text{Zeichen aus } \mathcal{X}: \quad & \tilde{z} := z \\
\text{Wort aus } \mathcal{X}^*: \quad & \tilde{w} := w \\
\text{Endliche Folge von Wörtern aus } \mathcal{X}^*: \quad & \widetilde{(w_1, \dots, w_n)} := w_1, \dots, w_n, \\
\text{Regel:} \quad & \widetilde{p_1, \dots, p_n} \to k := p_1, \dots, p_n, k; \\
\text{Kalkül:} \quad & \widetilde{(R_1, \dots, R_n)} := \tilde{R}_1 \dots \tilde{R}_n
\end{aligned}$$

Später werden wir noch die Kodierungen $\tilde{\kappa}$ von weiteren Objekten κ angeben.

2. Schritt: In dem im 1. Schritt erhaltenen Wort wird jedes Zeichen $z \in \mathcal{Z}$ weiter kodiert als Wort $\hat{z} \in \{0,1\}^*$.

$$\begin{aligned}\hat{,} &:= 1\\ \hat{;} &:= 10\\ \hat{a}_n &:= 1\ \underbrace{0\ldots0}_{n+2 \text{ mal}}\\ \hat{x}_n &:= 1\ \underbrace{0\ldots0}_{n+3 \text{ mal}}\ .\end{aligned}$$

Jedes Wort $w = z_1 \ldots z_n \in \mathcal{Z}^*$ wird kodiert als Wort $\hat{w} := \hat{z}_1 \ldots \hat{z}_n \in \{0,1\}^*$ $\quad (z_1, \ldots, z_n \in \mathcal{Z})$.

Die gesamte Kodierung: Für ein Objekt κ sei $[\kappa] := \hat{\tilde{\kappa}}$.

Um die Sprache L durch einen Kalkül zu erzeugen, müssen wir noch weitere Objekte kodieren: Instanziierungen, Herleitungen, usw. Leider sind Instanziierungen nicht endlich beschreibbar (es gibt überabzählbar viele Instanziierungen, da wir unendlich viele Variable haben). Bei der Anwendung einer Instanziierung auf eine Regel werden aber immer nur endlich viele Variable wirklich ersetzt. Daher genügt es die Beschränkung einer Instanziierung auf eine endliche Variablenmenge, etwa $\{x_0, \ldots, x_n\}$ mit $n \in \mathbb{N}$ zu betrachten. Wir definieren also $\mathcal{V}_n := \{x_0, \ldots, x_n\}$ für $n \in \mathbb{N}$. Eine *Instanziierung auf* $\mathcal{V}_n$ ist eine Abbildung $\iota\colon \mathcal{V}_n \to \mathcal{S}^*$. Eine Instanziierung auf $\mathcal{V}_n$ kann man ebenso wie eine Instanziierung (auf ganz $\mathcal{V}$) anwenden auf ein Wort aus $\mathcal{X}^*$ oder auf eine Regel, sofern dieses Wort bzw. diese Regel keine Variablen aus $\mathcal{V} \setminus \mathcal{V}_n$ enthält, und man erhält dann ebenfalls eine Instanz dieses Wortes bzw. dieser Regel. Wir definieren

$\tilde{\iota} := x_0 w_0 \ldots x_n w_n$, wenn $\iota\colon \mathcal{V}_n \to \mathcal{S}^*$ und $\iota(x_i) = w_i$ für $i = 0, \ldots, n$.

Die Paare (x_i, w_i) heißen auch die *(Variablen-)Bindungen* von ι. Sie werden also hier kodiert als $x_i w_i$. Nun sei wieder $[\iota] := \hat{\tilde{\iota}}$.

Auf der Menge $\mathcal{Z}^*$ der Wörter über der Zeichenmenge $\mathcal{Z}$ betrachten wir die folgenden n-stelligen Prädikate mit $n \in \mathbb{N}$ (also Teilmengen von $(\mathcal{Z}^*)^n$).

$=$	2-stellig	$\{(w,w) \mid w \in \mathcal{Z}^*\}$
$\mathcal{Z}$	1-stellig	$\{\tilde{z} \mid z \in \mathcal{Z}\}$ (Das Wort $\tilde{z}$ der Länge 1 identifizieren wir mit dem Zeichen z)
$\mathcal{Z}^*$	1-stellig	$\{w \mid w \in \mathcal{Z}^*\}$
$\mathcal{S}$	1-stellig	$\{\tilde{z} \mid z \in \mathcal{S}\}$ (Das Wort $\tilde{z}$ der Länge 1 identifizieren wir mit dem Zeichen z)
$\mathcal{V}$	1-stellig	$\{\tilde{z} \mid z \in \mathcal{V}\}$ (Das Wort $\tilde{z}$ der Länge 1 identifizieren wir mit dem Zeichen z)
$\mathcal{X}$	1-stellig	$\{\tilde{z} \mid z \in \mathcal{X}\}$ (Das Wort $\tilde{z}$ der Länge 1 identifizieren wir mit dem Zeichen z)
$\mathcal{S}^*$	1-stellig	$\{w \mid w \in \mathcal{S}^*\}$
$\mathcal{X}^*$	1-stellig	$\{w \mid w \in \mathcal{X}^*\}$
$\mathcal{P}$	1-stellig	$\{\tilde{p} \mid p$ ist endliche Folge von Wörtern über $\mathcal{X}\}$
$\mathcal{R}$	1-stellig	$\{\tilde{r} \mid r$ ist Regel$\}$
$\mathcal{K}$	1-stellig	$\{\tilde{k} \mid k$ ist Kalkül$\}$
$\mathcal{I}$	1-stellig	$\{\tilde{\iota} \mid \iota\colon \mathcal{V}_n \to \mathcal{S}^*$ für ein $n \in \mathbb{N}\}$
$\mathcal{B}$	3-stellig	$\{(\tilde{\iota}, \tilde{x}, \tilde{w}) \mid \iota(x) = w\}$
$\mathcal{A}$	3-stellig	$\{(\tilde{\iota}, \tilde{u}, \tilde{w}) \mid u \in \mathcal{X}^* \wedge u\iota = w\}$
$\mathcal{Q}$	3-stellig	$\{(\tilde{\iota}, \tilde{p}, \tilde{w}) \mid p$ ist eine Folge $(p_1, \ldots, p_m)$ von Wörtern aus $\mathcal{X}^*$ und w ist eine Folge $(w_1, \ldots, w_n)$ von Wörtern aus $\mathcal{X}^*$ und für alle $i \in \{1, \ldots, m\}$ gibt es ein $j \in \{1, \ldots, n\}$ mit $p_i\iota = w_j\}$
$\mathcal{H}$	2-stellig	$\{(\tilde{h}, \tilde{k}) \mid h$ ist Herleitung im Kalkül $k\}$

Für jedes n-stellige Prädikat p auf $\mathcal{Z}^*$ definieren wir ein n-stelliges Prädikat $\hat{p}$ auf $\{0,1\}^*$ durch

$$\hat{p} := \{(\hat{w}_1, \ldots, \hat{w}_n) \mid (w_1, \ldots, w_n) \in p\}.$$

2.4.6 Ein Kalkül für die Herleitbarkeitsrelation in Kalkülen

Seien $=, Z, Z^*, S, V, X, S^*, X^*, P, R, K, I, B, A, Q, H$ paarweise verschiedene Symbole, die auch verschieden seien von den Symbolen 0, 1 und #. Wir konstruieren nun einen Kalkül K_0 mit den folgenden Eigenschaften für alle Wörter $u, v, w \in \mathcal{S}^*$:

$$\begin{aligned}
K_0 \vdash v = w &\iff (v, w) \in \hat{=} \\
K_0 \vdash wZ &\iff w \in \hat{\mathcal{Z}} \\
K_0 \vdash wZ^* &\iff w \in \hat{\mathcal{Z}}^* \\
K_0 \vdash wS &\iff w \in \hat{\mathcal{S}} \\
K_0 \vdash wV &\iff w \in \hat{\mathcal{V}} \\
K_0 \vdash wX &\iff w \in \hat{\mathcal{X}} \\
K_0 \vdash wS^* &\iff w \in \hat{\mathcal{S}}^* \\
K_0 \vdash wX^* &\iff w \in \hat{\mathcal{X}}^* \\
K_0 \vdash wP &\iff w \in \hat{\mathcal{P}} \\
K_0 \vdash wR &\iff w \in \hat{\mathcal{R}} \\
K_0 \vdash wK &\iff w \in \hat{\mathcal{K}} \\
K_0 \vdash wI &\iff w \in \hat{\mathcal{I}} \\
K_0 \vdash uBvBw &\iff (u, v, w) \in \hat{\mathcal{B}} \\
K_0 \vdash uAvAw &\iff (u, v, w) \in \hat{\mathcal{A}} \\
K_0 \vdash uQvQw &\iff (u, v, w) \in \hat{\mathcal{Q}} \\
K_0 \vdash vHw &\iff (v, w) \in \hat{\mathcal{H}}
\end{aligned}$$

Seien $x, x', x'', x''', y, y', y'', y''', y'''', z, z', z'', z'''$ paarweise verschiedene Variablen. Dann sei

K_0 der folgende Kalkül.

$$\begin{array}{c}
\to\ = \qquad 1x = 1x \to 1x0 = 1x0 \qquad x = x \to x1 = x1 \\
\to 1Z \qquad xZ \to x0Z \\
\to Z^* \qquad xZ^*, yZ \to xyZ^* \\
\to 100S \qquad xS \to x00S \\
\to 1000V \qquad xV \to x00V \\
xS \to xX \qquad xV \to xX \\
\to S^* \qquad xS^*, yS \to xyS^* \\
\to X^* \qquad xX^*, yX \to xyX^* \\
\to P \qquad xP, yX^* \to xy1P \\
xP \to x10R \qquad x10R, yS \to xy10R \qquad x1y10R, x = x'zx'', zV, x''Z^*, yX^* \to x1yz10R \\
\to K \qquad xK, yR \to xyK \\
yS^* \to 1000yI \qquad xI, x = x'zx'', zV, x''S^*, yS^* \to xz00yI \\
xI, x = x'yz, yV, zS^* \to xByBz \qquad xI, x = x'yzx''x''', yV, zS^*, x''V, x'''X^* \to xByBz \\
xI \to xAA \qquad xAyAy', zS \to xAyzAy'z \qquad xAyAy', xBzBz' \to xAyzAy'z' \\
xI, zP \to xQQz \qquad xQyQz, z = z'z''1z''', z'P, xAy'Az'', z'''P \to xQyy'1Qz \\
yK \to Hy \qquad xHy, y = y'y''y'''10y'''', y'K, y''P, y'''X^*, y''''K, zQy''Qx, zAy'''Ax' \to xx'1Hy \\
zy1Hx, zP, yS^* \to x\#y
\end{array}$$

Dann gilt

$$L(K_0, \{0,1,\#\}) = \{[K]\#[w] \mid K \text{ ist ein Kalkül und } K \vdash w\}.$$

Es folgt

Satz 40 *Die Sprache $\{[K]\#[w] \mid K$ ist ein Kalkül und $K \vdash w\}$ ist eine kalkülerzeugte Sprache.*

Man sagt auch, dass das Problem der Herleitbarkeit in einem Kalkül ein durch einen Kalkül semientscheidbares Problem ist.

2.4.7 Kalkül-Unentscheidbarkeit der Herleitbarkeit in Kalkülen

Satz 41 *Die Sprache $\{0,1,\#\}^* \setminus \{[K]\#[w] \mid K$ ist ein Kalkül und $K \vdash w\}$ ist nicht eine kalkülerzeugte Sprache.*

Beweis: Andernfalls gäbe es einen Kalkül K_1 mit

$$L(K_1, \{0,1,\#\}) = \{0,1,\#\}^* \setminus \{[K]\#[w] \mid K \text{ ist ein Kalkül und } K \vdash w\}.$$

Nun seien a und c zwei verschiedene Symbole, die nicht in K_1 auftreten und die auch verschieden sind von den Symbolen $0, 1, \#$. Da 0 und 1 Symbole sind, sind ihre Kodierungen [0] und [1] Wörter über dem Alphabet $\{0,1\}$. Jetzt sei

$$K_2 := K_1 a + (\to c) + (xcy \to x0cy[0]) + (xcy \to x1c[1]) + (x\#ya, xcy \to x).$$

Dann ist $L(K_2) = L(K_1)a \cup \{wc[w] \mid w \in \{0,1\}^*\} \cup \{w \in \{0,1\}^* \mid K_1 \vdash w\#[w]\}$. Für jedes Wort $w \in \{0,1\}^*$ gilt also

$$K_2 \vdash w \iff K_1 \vdash w\#[w].$$

Setzt man hier für w die Kodierung $[K]$ eines beliebigen Kalküls K ein, so folgt. Für jeden Kalkül K gilt

$$\begin{aligned} K_2 \vdash [K] &\iff K_1 \vdash [K]\#[[K]] \\ &\iff K \nvdash [K]. \end{aligned}$$

Insbesondere gilt dies, wenn man $K := K_2$ wählt. Also

$$K_2 \vdash [K_2] \iff K_2 \nvdash [K_2]$$

Dies ist ein Widerspruch. q.e.d.

Aus diesem Satz folgt

Satz 42 *Die Sprache* $\{[K]\#[w] \mid K$ *ist ein Kalkül und* $K \vdash w\}$ *ist nicht durch Kalküle entscheidbar.*

Wir sagen auch, dass die Herleitbarkeit in Kalkülen nicht durch Kalküle entscheidbar sei.

2.4.8 μ-Partiellrekursivität der kalkülberechenbaren Funktionen

Nun wollen wir zeigen, dass die durch Kalküle berechenbaren partiellen Funktionen auch mit Hilfe des Systems der μ-partiellrekursiven Funktionale darstellbar sind. Nun sind aber partielle Funktionen, die durch Kalküle berechnet werden, auf Wörtern definiert, während μ-partiellrekursive Funktionen auf Zahlen definiert sind. Zur Darstellung müssen wir also die Wörter erst als Zahlen kodieren. Wir ordnen also jedem Objekt κ, für das wir eine Kodierung $[\kappa]$ definiert haben, eine natürliche Zahl $\ulcorner\kappa\urcorner$ zu, die man *Gödelnummer* nennt nach Kurt Gödel, der diese Technik im Jahr 1931 eingeführt hat. Wir definieren $\ulcorner\kappa\urcorner$ als den Wert der Dualzahl $[\kappa]$ im Stellenwertsystem zur Basis 2. Wenn $[\kappa] = \varepsilon$ ist, dann sei $\ulcorner\kappa\urcorner := 0$. Insbesondere wird durch diese Zuordnung (*Gödelisierung*) jedem Wort $w \in \mathcal{S}^*$ eine Gödelnummer zugeordnet. Entsprechend kann man endlichen Folgen, Mengen, usw. von solchen Wörtern auch endliche Folgen, Mengen, usw. der entsprechenden Gödelnummern zuordnen und damit weiter jeder n-stelligen partiellen Funktion f auf $\mathcal{S}^*$ eine entsprechende n-stellige partielle zahlentheoretische Funktion $\urcorner f \ulcorner$ auf den entsprechenden Gödelnummern. Die Definitionen dazu sind wie folgt.

Gödelisierung Objekt $\kappa \mapsto$ *Gödelnummer* $\ulcorner\kappa\urcorner \in \mathbb{N}$.

$$\ulcorner\kappa\urcorner := \text{ Wert der Dualzahl } [\kappa]$$

$$\begin{aligned} \urcorner w \ulcorner &:= \ulcorner w \urcorner, \text{ wenn } w \in \mathcal{S}^* \\ \urcorner (q_1, \ldots, q_n) \ulcorner &:= (\urcorner q_1 \ulcorner, \ldots, \urcorner q_n \ulcorner) \\ \urcorner Q \ulcorner &:= \{\urcorner q \ulcorner \mid q \in Q\}, \text{ wenn } Q \text{ Menge} \end{aligned}$$

Für $f\colon (\Delta^*)^n \xrightarrow{\text{part}} \Delta^*$ sei $\urcorner f \ulcorner\colon \mathbb{N}^n \xrightarrow{\text{part}} \mathbb{N}$ definiert durch

$$\text{Graph}(\urcorner f \ulcorner) := \urcorner \text{Graph}(f) \ulcorner$$

Definition 93 $f\colon (\Delta^*)^n \xrightarrow{\text{part}} \Delta^*$ heißt μ*-partiellrekursiv*, wenn $\urcorner f \ulcorner$ μ-partiellrekursiv ist.

$$\begin{array}{ccc} (\Delta^*)^n & \xrightarrow[f]{\text{part}} & \Delta^* \\ {\scriptstyle \ulcorner . \urcorner}\downarrow & & \downarrow{\scriptstyle \ulcorner . \urcorner} \\ \mathbb{N}^n & \xrightarrow[\ulcorner f \urcorner]{\text{part}} & \mathbb{N} \end{array}$$

Wir wollen beweisen: Jede kalkülberechenbare Funktion $f\colon (\Delta^*)^n \xrightarrow{\text{part}} \Delta^*$ ist μ-partiellrekursiv.

Zum Beweis: n-stelligen Prädikaten $\mathcal{Z}, \ldots, \mathcal{H}$ auf $\mathcal{Z}^*$ entsprechen n-stellige Prädikate $Z := \ulcorner \mathcal{Z} \urcorner, \ldots, H := \ulcorner \mathcal{H} \urcorner$ auf $\mathbb{N}$. Wir werden zeigen, dass sie primitivrekursive Prädikate sind. Hierzu benötigen wir noch ein paar Hilfsmittel.

Da Dualzahlen Wörter über dem Alphabet $\{0, 1\}$ sind, sind auf ihnen auch Operationen, die auf Wörtern operieren, definiert, z.B. Bestimmung der Länge einer Dualzahl oder Konkatenation von Dualzahlen. Im Folgenden werden die entsprechenden Operationen auf den Werten dieser Dualzahlen im Stellenwertsystem zur Basis 2 (also auf natürlichen Zahlen) angegeben. Rechts ist jeweils die entsprechende Operation auf Dualzahlen angegeben.

$l\colon \mathbb{N} \to \mathbb{N}$	$l(x) := \mu_b z \leq x\colon x < 2^z$	Länge
$\circ\colon \mathbb{N}^2 \to \mathbb{N}$	$x \circ y := x \cdot 2^{l(y)} + y$	Konkatenation

Dann sind l und $\circ$ primitivrekursiv und $l(x)$ ist die Länge der Dualzahl für x und $x \circ y$ ist der Wert der Dualzahl, die sich durch Konkatenation der Dualzahlen für x und y ergibt. Wir werden diese Funktionen auf Gödelnummern anwenden.

Lemma 8 (Wertverlaufsrekursion) *Sei $g : \mathbb{N}^{n+1} \to \mathbb{N}$ definiert durch*

$$g(\mathfrak{x}, y) := f(\langle g(\mathfrak{x}, 0), \ldots, g(\mathfrak{x}, y-1)\rangle, \mathfrak{x}, y),$$

wobei f primitivrekursiv ist. Dann ist g primitivrekursiv.

Beweis: Sei $h(\mathfrak{x}, y) := \langle g(\mathfrak{x}, 0), \ldots, g(\mathfrak{x}, y-1)\rangle$. Dann

$$\begin{aligned} h(\mathfrak{x}, 0) &= f(0, \mathfrak{x}, 0) \\ h(\mathfrak{x}, y+1) &= h(\mathfrak{x}, y) \bullet_y f(h(\mathfrak{x}, y), \mathfrak{x}, y) \end{aligned}$$

mit $\langle a_1, \ldots, a_n\rangle \bullet_n a := \langle a_1, \ldots, a_n, a\rangle$. Die Funktion $(a, b, n) \mapsto a \bullet_n b$ ist primitivrekursiv und daher auch h und somit g wegen $g(\mathfrak{x}, y) = f(h(\mathfrak{x}, y), \mathfrak{x}, y)$. q.e.d.

Wir zeigen nun nacheinander, dass die Prädikate $Z, \ldots, H$ primitivrekursiv sind. Hierzu gehen wir vom Kalkül K_0 aus, in dem diese Prädikate durch entsprechende Regeln charakterisiert sind.

Die Regeln des Kalküls K_0 werden dabei in Wertverlaufsrekursionen übersetzt.

$$
\begin{aligned}
Z(u) &\iff u = 1 \vee \bigvee_{x<u} \big(Z(x) \wedge u = 2x\big) \\
Z^*(u) &\iff u = 0 \vee \bigvee_{x<u} \bigvee_{y\le u} \big(Z^*(x) \wedge Z(y) \wedge u = x \circ y\big) \\
S(u) &\iff u = 4 \vee \bigvee_{x<u} \big(S(x) \wedge u = 4x\big) \\
V(u) &\iff u = 8 \vee \bigvee_{x<u} \big(V(x) \wedge u = 4x\big) \\
X(x) &\iff S(x) \vee V(x) \\
S^*(u) &\iff u = 0 \vee \bigvee_{x<u} \bigvee_{y\le u} \big(S^*(x) \wedge S(y) \wedge u = x \circ y\big) \\
X^*(u) &\iff u = 0 \vee \bigvee_{x<u} \bigvee_{y\le u} \big(X^*(x) \wedge X(y) \wedge u = x \circ y\big) \\
P(u) &\iff u = 0 \vee \bigvee_{x,y<u} \big(P(x) \wedge X^*(y) \wedge u = x \circ y \circ 1\big) \\
R(u) &\iff \bigvee_{x<u} \big(P(x) \wedge u = x \circ 2\big) \vee \bigvee_{\substack{x,y<u \\ x\circ 2<u}} \big(R(x \circ 2) \wedge S(y) \wedge u = x \circ y \circ 2\big) \\
&\qquad \vee \bigvee_{\substack{x,x',x'',y,z<u \\ x\circ 1\circ y\circ 2<u}} \big(R(x \circ 1 \circ y \circ 2) \wedge x = x' \circ z \circ x'' \wedge V(z) \wedge Z^*(x'') \wedge X^*(y) \wedge u = x \circ 1 \circ y \circ z \circ 2\big) \\
K(u) &\iff u = 0 \vee \bigvee_{x<u} \bigvee_{y\le u} \big(K(x) \wedge R(y) \wedge u = x \circ y\big) \\
I(u) &\iff \bigvee_{y<u} \big(S^*(y) \wedge u = 8 \circ y\big) \\
&\qquad \vee \bigvee_{x,x',x'',y,z<u} \big(I(x) \wedge x = x' \circ z \circ x'' \wedge V(z) \wedge S^*(x'') \wedge S^*(y) \wedge u = x \circ 4z \circ y\big) \\
B(x,y,z) &\iff \bigvee_{x'\le x} \big(I(x) \wedge x = x' \circ y \circ z \wedge V(y) \wedge S^*(z)\big) \\
&\qquad \vee \bigvee_{x',x'',x'''\le x} \big(I(x) \wedge x = x' \circ y \circ z \circ x'' \circ x''' \wedge V(y) \wedge S^*(z) \wedge V(x'') \wedge Z^*(x''')\big) \\
A(x,u,v) &\iff \big(I(x) \wedge u = 0 \wedge v = 0\big) \\
&\qquad \vee \bigvee_{\substack{y<u \\ y'<v \\ z\le u}} \big(A(x,y,y') \wedge S(z) \wedge u = y \circ z \wedge v = y' \circ z\big) \\
&\qquad \vee \bigvee_{\substack{y<u \\ y'<v \\ z\le u}} \big(A(x,y,y') \wedge B(x,z,z') \wedge u = y \circ z \wedge v = y' \circ z\big) \\
Q(x,u,z) &\iff \big(I(x) \wedge P(z) \wedge u = 0\big) \\
&\qquad \vee \bigvee_{\substack{y,y'<u \\ z',z'',z'''<z}} \big(Q(x,y,z) \wedge z = z' \circ z'' \circ 1 \circ z''' \wedge P(z') \wedge A(x,y',z'') \wedge P(z''') \wedge u = y \circ y' \circ 1\big) \\
H(u,y) &\iff \big(K(y) \wedge u = 0\big) \\
&\qquad \vee \bigvee_{\substack{x,x'<u \\ y',y'',y''',y''''<y \\ z\le y^{u+y}}} \big(H(x,y) \wedge y = y' \circ y'' \circ y''' \circ 2 \circ y'''' \wedge K(y') \wedge P(y'') \wedge X^*(y''') \wedge K(y'''') \\
&\qquad\qquad \wedge Q(z,y'',x) \wedge A(z,y''',x') \wedge u = x \circ x' \circ 1\big)
\end{aligned}
$$

Lemma 9

1. $H = \{(\ulcorner h \urcorner, \ulcorner k \urcorner) \mid h \text{ ist Herleitung im Kalkül } k\}$
2. H *ist primitivrekursiv*

$$H_+(u,x) :\Longleftrightarrow H(u,x) \wedge u \neq 0$$

$$r(v) := \mu_b y < v : \bigvee_{z<v} \big(v = z \circ y \circ 1 \wedge P(z) \wedge S^*(y)\big)$$

Für eine Herleitung $h = (w_1, \ldots, w_n)$ ist $r(\ulcorner h \urcorner) = \ulcorner w_n \urcorner$.

Satz 43 *Für jeden Kalkül K und jedes Alphabet $\Delta \subseteq \mathcal{S}$ gibt es ein primitivrekursives Prädikat F und eine μ-partiellrekursive Funktion f mit*

$$\ulcorner L(K,\Delta) \urcorner = \{x \mid \bigvee_v F(x,v)\} = \mathrm{Def}(f).$$

Beweis: $D^* := \ulcorner \Delta^* \urcorner$ ist primitivrekursiv.

$$F(x,v) := H_+(v, \ulcorner K \urcorner) \wedge r(v) = x \wedge D^*(x), \qquad f := (\mu F).$$

q.e.d.

Satz 44 *Sei $\Delta \subseteq \mathcal{S}$ ein Alphabet. Dann gibt es eine 1-stellige primitivrekursive Funktion u und zu jedem $n \in \mathbb{N}$ ein $n+2$-stelliges primitivrekursives Prädikat t_n, sodass für jedes kalkülberechenbare $f\colon (\Delta^*)^n \xrightarrow{\text{part}} \Delta^*$ ein $e \in \mathbb{N}$ existiert mit*

$$\ulcorner f \urcorner = (u(\mu t_n))(e, \cdot).$$

Beweis:

$$t(x_1, \ldots, x_{n+1}) := x_1 \circ \ulcorner \# \urcorner \circ \cdots \circ \ulcorner \# \urcorner \circ x_{n+1}$$

$$t_n(e, x_1, \ldots, x_n, v) :\Longleftrightarrow \bigvee_{y \leq v} \big(H_+(v,e) \wedge r(v) = t(x_1, \ldots, x_n, y) \wedge D^*(x_1) \wedge \cdots \wedge D^*(x_n) \wedge D^*(y)\big)$$

$$u(v) := \mu_b y \leq v\colon \bigvee_{x \leq v} r(v) = x \circ \# \circ y.$$

Sei K ein Kalkül, der f berechnet und $e := \ulcorner K \urcorner$. q.e.d.

Satz 45 *Jede kalkülberechenbare Funktion $f\colon (\Delta^*)^n \xrightarrow{\text{part}} \Delta^*$ ist μ-partiellrekursiv.*

Satz 46 *Jede kalkülberechenbare Funktion $f\colon \mathbb{N}^n \xrightarrow{\text{part}} \mathbb{N}$ ist μ-partiellrekursiv.*

Beweis: Nach Definition ist die Funktion $g\colon (\{|\}^*)^n \xrightarrow{\text{part}} \{|\}^*$ mit $g(|^{x_1},\ldots,|^{x_n}) := |^{f(x_1,\ldots,x_n)}$ kalkülberechenbar und daher nach dem vorigen Satz μ-partiellrekursiv, d.h. $\ulcorner g \urcorner\colon \mathbb{N}^n \xrightarrow{\text{part}} \mathbb{N}$ ist μ-partiellrekursiv. Primitivrekursive Kodierung c und Dekodierung d:

$$c(x) = \ulcorner |^x \urcorner \qquad d(\ulcorner |^x \urcorner) = x \qquad (x \in \mathbb{N})$$
$$f(x_1,\ldots,x_n) = d(\ulcorner g \urcorner(c(x_1),\ldots,c(x_n)))$$

q.e.d.

Verschiedene Formalismen:

- μ-partiellrekursive Funktionale
- Kalküle
- Grammatiken
- Turingmaschinen
- λ-Kalkül
- Kombinatorische Logik
- Programmiersprachen

stellen dieselbe Klasse von partiellen Funktionen auf $\mathbb{N}$ bzw. Δ^* dar.

Definition 94 Eine Funktion $f\colon \mathbb{N}^n \xrightarrow{\text{part}} \mathbb{N}$ heißt *partiellrekursiv*, wenn sie in einem dieser Formalismen darstellbar ist.

2.4.9 Normalformsatz von Kleene

Satz 47 (Normalformsatz von Kleene) *Es gibt eine 1-stellige primitivrekursive Funktion U und zu jedem $r \in \mathbb{N}$ eine $r+2$-stellige primitivrekursive Funktion T_r so, dass für jede r-stellige partiellrekursive zahlentheoretische Funktion f eine natürliche Zahl e existiert so, dass*

$$f(\mathfrak{x}) = \begin{cases} (U(\mu T_r))(e,\mathfrak{x}), & \textit{falls dies definiert ist} \\ \textit{undefiniert} & \textit{sonst} \end{cases}$$

für alle $\mathfrak{x} \in \mathbb{N}^r$.

Diese Zahl e nennt man einen *Index* der partiellen zahlentheoretischen Funktion f. Man schreibt dann oft $f = \{e\}^r$.

Beweis: Sei $f\colon \mathbb{N}^r \xrightarrow{\text{part}} \mathbb{N}$ partiellrekursiv. Dann wird f durch einen Kalkül K berechnet. Nach Definition wird $g\colon (\{|\}^*)^r \xrightarrow{\text{part}} \{|\}^*$ mit $g(|^{x_1},\ldots,|^{x_r}) := |^{f(x_1,\ldots,x_r)}$ durch K berechnet. Nach Satz 44 gibt es $e \in \mathbb{N}$ (nämlich $e := \ulcorner K \urcorner$) mit $\ulcorner g \urcorner = (u(\mu t_r))(e,\cdot)$.

$$T_r(e,x_1,\ldots,x_r,v) := \begin{cases} 0, & \text{wenn } t_r(e,c(x_1),\ldots,c(x_r),v) \\ 1 & \text{sonst} \end{cases}$$
$$U(v) := d(u(v))$$

q.e.d.

Die zweistellige partielle zahlentheoretische Funktion $h := (U(\mu T_1))$ ist eine *universelle* partiellrekursive Funktion. Sie erlaubt die Berechnung jeder einstelligen (und durch Tupelcodierung auch jeder mehrstelligen) partiellrekursiven Funktion f, da $f(x) = h(e, x)$ ist, wobei e ein Index von f ist.

2.5 Chomsky-Grammatiken

2.5.1 Semi-Thue-Systeme

Definition 95 Ein *Semi-Thue-System* ist ein Paar $T = (V, R)$, wobei V ein Alphabet und R eine endliche zweistellige Relation auf V^* ist: $R \subseteq V^* \times V^*$ endlich.

Wenn $(x, y) \in R$ ist, dann schreibt man auch $x \to_T y$ oder kurz $x \to y$. Wenn es nicht missverständlich ist, schreibt man $x \to y$ auch für das Paar (x, y), also $R = \{x_1 \to y_1, \ldots, x_n \to y_n\}$ für $R = \{(x_1, y_1), \ldots, (x_n, y_n)\}$. Man nennt $x \to y$ eine *Produktion* des Semi-Thue-Systems.

Ein Wort $u \in V^*$ heißt *direkt* (oder *in einem Schritt*) *überführbar* in ein Wort $v \in V^*$, in Zeichen $u \Rightarrow_T v$ oder $u \Rightarrow v$, wenn gilt

$$\bigvee_{w,x,y,z \in V^*} (x \to y \text{ und } u = wxz \text{ und } v = wyz).$$

Mit $\Rightarrow^*$ bezeichnet man die reflexiv-transitive Hülle von $\Rightarrow$. Wenn $u \Rightarrow^* v$ gilt, so sagt man, u sei *überführbar* in v oder v sei *ableitbar* aus u.

Beispiel 60
Sei $V = \{a, b, S\}$, $R = \{S \to \varepsilon, S \to aSb, ab \to baa\}$ und $T = (V, R)$. Dann gilt $S \Rightarrow^* b^2a^8$, da $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb \Rightarrow abaab \Rightarrow ababaa \Rightarrow baaabaa \Rightarrow baabaaaa \Rightarrow baba^6 \Rightarrow bba^8$.

Noch eine Bemerkung zu diesem Beispiel und zu Aufgaben 3 und 4 von Blatt 5, denen dieses Beispiel zugrundeliegt. In Aufgabe 3 (b) und (c) wird für die dort definierten Funktionen W und B gezeigt, dass aus $u \in \{a, b\}^*$ und $u \Rightarrow v$ folgt $W(u) = W(v)$ und $B(u) = B(v)$. Man nennt solche Funktionen *Invarianten*. Eine ähnliche Situation gibt es in Computerprogrammen, etwa bei folgendem Pseudocode zur Berechnung von x^n.

```
m := n
y := 1
while m>0 do
    ⌜ m := m-1
      y := y*x ⌟
return y
```

Hier ist $x^m \cdot y$ eine *Schleifeninvariante*, die bei jedem Schleifendurchlauf ihren Wert nicht ändert. Da ihr Wert beim Eintritt in die Schleife gleich x^n ist, muss ihr Wert auch beim Verlassen der Schleife x^n sein. Dann hat aber m den Wert 0. Daher muss y beim Verlassen der Schleife den Wert x^n haben.

Definition 96 Sei $T = (V, R)$ ein Semi-Thue-System. Man sagt, ein Wort $w \in V^*$ sei *in Normalform*, wenn es kein $x \in V^*$ gibt mit $w \Rightarrow_T x$. Wenn $u \Rightarrow_T^* v$ gilt und v in Normalform ist, so sagt man, v sei eine *Normalform von* u.

Im Beispiel sind die Normalformen genau die Wörter der Form $b^n a^m$ mit $m, n \in \mathbb{N}$. Die Menge der Normalformen von S ist die Sprache $\{b^n a^m \mid n \in \mathbb{N} \text{ und } m = n \cdot 2^n\}$.

Definition 97 Unter einer *Ableitung* in einem Semi-Thue-System T versteht man eine endliche oder unendliche Folge $(w_0, w_1, w_2, \dots)$ so, dass $w_{j-1} \Rightarrow_T w_j$ gilt für alle $j \in \mathbb{N} \setminus \{0\}$, die kleiner als die Anzahl der Glieder der Folge sind. Man spricht auch von einer *Ableitung aus dem Wort* w_0. Eine Ableitung heißt *maximal*, wenn sie entweder unendlich ist oder mit einem Wort w_n in Normalform endet. Bei einer endlichen Ableitung $(w_0, \dots, w_n)$ spricht man auch von einer *Ableitung des Wortes* w_n *aus dem Wort* w_0. Die Zahl n heißt die *Länge* der Ableitung.

Offenbar ist ein Wort v genau dann eine Normalform eines Wortes u, wenn es eine Ableitung von v aus u gibt, die eine maximale Ableitung ist.

2.5.2 Chomsky-Grammatiken

Definition 98 Eine *Chomsky-Grammatik* (englisch *Chomsky grammar*) ist ein Quadrupel $G = (V, \Sigma, R, S)$, wobei V ein Alphabet ist, $\Sigma \subseteq V$, $S \in V \setminus \Sigma$, und (V, R) ein Semi-Thue-System ist mit der Eigenschaft

$$\text{nicht} \bigvee_{u \in \Sigma^*} \bigvee_{v \in V^*} u \to v.$$

Die Elemente von Σ heißen *terminale Zeichen*, die Elemente von $V \setminus \Sigma$ *nichtterminale Zeichen* oder *syntaktische Variablen*. Ein Wort w aus V^* heißt *terminal*, wenn w nur aus terminalen Zeichen besteht, d.h. wenn $w \in \Sigma^*$ ist. Für $\Rightarrow_{(V,R)}$ schreibt man auch $\Rightarrow_G$. Man sagt, ein terminales Wort $w \in \Sigma^*$ werde von G *erzeugt*, wenn $S \Rightarrow_G^* w$. Die von G *erzeugte Sprache* $L(G)$ ist definiert als $L(G) := \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$. Man nennt sie auch den *Sprachschatz* von G.

Die Bedingung, die dabei an das Semi-Thue-System (V, R) gestellt wird, kann man auch so ausdrücken:

> Jedes terminale Wort ist in Normalform.

Im obigen Beispiel etwa ist für $\Sigma := \{a, b\}$ das Quadrupel (V, Σ, R, S) keine Chomsky-Grammatik, da $ab \to baa$ und daher das Wort ab nicht in Normalform ist, obwohl $ab \in \Sigma^*$ ist.

Definition 99 Unter einer *Ableitung* in einer Chomsky-Grammatik (V, Σ, R, S) versteht man eine Ableitung im Semi-Thue-System (V, R).

Beispiel 61

$$\begin{aligned} V &= \{a, b, S, A, B\} \\ \Sigma &= \{a, b\} \\ R &= \{S \to \varepsilon, S \to ASB, AB \to BAA, A \to a, B \to b\} \\ G &= (V, \Sigma, R, S) \end{aligned}$$

Dann ist G eine Chomsky-Grammatik und die von G erzeugte Sprache ist $\{w \in \{a, b\}^* \mid W(w) = B(w) \cdot 2^{B(w)}\}$ (s. Blatt5, Aufgaben 3 und 4).

Beispiel 62

$$
\begin{aligned}
V &= \{(,), S, T\} \\
\Sigma &= \{(,)\} \\
R &= \{S \to (T), T \to \varepsilon, T \to TS\} \\
G &= (V, \Sigma, R, S)
\end{aligned}
$$

Dann ist $L(G)$ die Menge aller wohlgebildeten Klammerausdrücke.

Definition 100 Eine Grammatik heißt *kontextfrei*, wenn sie nur Produktionen der Form $A \to w$ enthält mit $A \in V \setminus \Sigma$, also wenn die linke Seite jeder ihrer Produktionen nur aus einem nichtterminalen Zeichen besteht.

Beispiel 63
Das Alphabet V bestehe aus den folgenden Symbolen. $\langle \mathit{Satz} \rangle$, $\langle \mathit{Nominalphrase} \rangle$, $\langle \mathit{Verbalphrase} \rangle$, $\langle \mathit{Artikel} \rangle$, $\langle \mathit{Substantiv} \rangle$, $\langle \mathit{Verb} \rangle$. Jedes dieser mit spitzen Klammern geklammerten Symbole betrachten wir als ein einziges Zeichen. Dies sind die nichtterminalen Zeichen von V. Hinzu kommen als terminale Zeichen die Buchstaben `a`, ..., `z`. Diese Buchstaben bilden also das Alphabet Σ. Die Menge R bestehe aus den Produktionen

$$
\begin{aligned}
\langle \mathit{Satz} \rangle &\to \langle \mathit{Nominalphrase} \rangle \langle \mathit{Verbalphrase} \rangle \\
\langle \mathit{Nominalphrase} \rangle &\to \langle \mathit{Artikel} \rangle \langle \mathit{Substantiv} \rangle \\
\langle \mathit{Verbalphrase} \rangle &\to \langle \mathit{Verb} \rangle \langle \mathit{Nominalphrase} \rangle \\
\langle \mathit{Verbalphrase} \rangle &\to \langle \mathit{Verb} \rangle \\
\langle \mathit{Artikel} \rangle &\to \texttt{der} \\
\langle \mathit{Artikel} \rangle &\to \texttt{den} \\
\langle \mathit{Substantiv} \rangle &\to \texttt{apfel} \\
\langle \mathit{Substantiv} \rangle &\to \texttt{mann} \\
\langle \mathit{Verb} \rangle &\to \texttt{isst}
\end{aligned}
$$

Das Startsymbol S sei das nichtterminale Zeichen $\langle \mathit{Satz} \rangle$. Sei nun G die Grammatik (V, Σ, R, S). Dann ist G kontextfrei und das 'Wort'

der mann isst den apfel

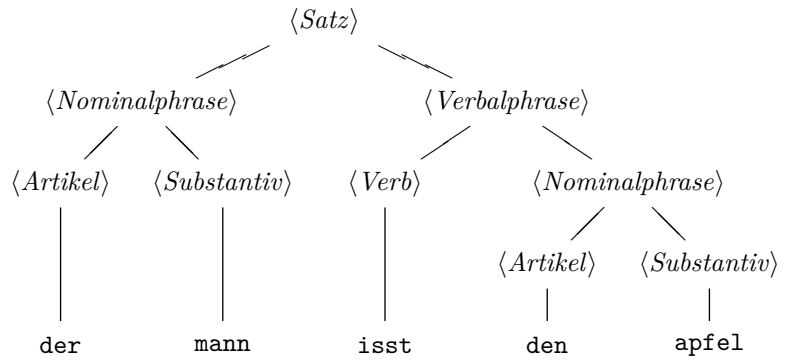
ist ein Element von $L(G)$. Man schreibt eine solche Grammatik oft mit einer etwas anderen Syntax, der sogenannten Backus-Naur-Form:

$$
\begin{aligned}
\langle \mathit{Satz} \rangle &::= \langle \mathit{Nominalphrase} \rangle \langle \mathit{Verbalphrase} \rangle \\
\langle \mathit{Nominalphrase} \rangle &::= \langle \mathit{Artikel} \rangle \langle \mathit{Substantiv} \rangle \\
\langle \mathit{Verbalphrase} \rangle &::= \langle \mathit{Verb} \rangle \langle \mathit{Nominalphrase} \rangle \\
\langle \mathit{Verbalphrase} \rangle &::= \langle \mathit{Verb} \rangle \\
\langle \mathit{Artikel} \rangle &::= \texttt{der} \\
\langle \mathit{Artikel} \rangle &::= \texttt{den} \\
\langle \mathit{Substantiv} \rangle &::= \texttt{apfel} \\
\langle \mathit{Substantiv} \rangle &::= \texttt{mann} \\
\langle \mathit{Verb} \rangle &::= \texttt{isst}
\end{aligned}
$$

Wir werden eine Ableitung $(w_0, \ldots, w_n)$ in einer Grammatik auch oft so schreiben: $w_0 \Rightarrow \ldots \Rightarrow w_n$. Es folgt eine Darstellung einer Ableitung in dieser Notation.

$$\begin{aligned}
\langle \mathit{Satz} \rangle &\Rightarrow \langle \mathit{Nominalphrase} \rangle \ \langle \mathit{Verbalphrase} \rangle \\
&\Rightarrow \langle \mathit{Nominalphrase} \rangle \ \langle \mathit{Verb} \rangle \ \langle \mathit{Nominalphrase} \rangle \\
&\Rightarrow \langle \mathit{Nominalphrase} \rangle \ \texttt{isst} \ \langle \mathit{Nominalphrase} \rangle \\
&\Rightarrow \langle \mathit{Nominalphrase} \rangle \ \texttt{isst} \ \langle \mathit{Artikel} \rangle \ \langle \mathit{Substantiv} \rangle \\
&\Rightarrow \langle \mathit{Nominalphrase} \rangle \ \texttt{isst} \ \langle \mathit{Artikel} \rangle \ \texttt{apfel} \\
&\Rightarrow \langle \mathit{Artikel} \rangle \ \langle \mathit{Substantiv} \rangle \ \texttt{isst} \ \langle \mathit{Artikel} \rangle \ \texttt{apfel} \\
&\Rightarrow \langle \mathit{Artikel} \rangle \ \texttt{mann isst} \ \langle \mathit{Artikel} \rangle \ \texttt{apfel} \\
&\Rightarrow \texttt{der mann isst} \ \langle \mathit{Artikel} \rangle \ \texttt{apfel} \\
&\Rightarrow \texttt{der mann isst den apfel}
\end{aligned}$$

Eine Ableitung in einer solchen Grammatik spiegelt die syntaktische Struktur des abgeleiteten Elements von $L(G)$ wieder, was man durch den *Strukturbaum* darstellen kann:



2.5.3 Berechenbarkeit durch eine Grammatik

Sei Δ ein Alphabet. Wir führen drei neue Zeichen #, [und] ein, die nicht Elemente von Δ sind. Sei $\Sigma := \Delta \cup \{\#, [,]\}$.

Definition 101 Man sagt, eine partielle Funktion $f\colon (\Delta^*)^r \xrightarrow{\text{part}} (\Delta^*)^s$ werde *durch eine Grammatik* $G = \{V, \Sigma, R, S)$ *berechnet*, wenn für alle $x_1, \ldots, x_r, y_1, \ldots, y_s \in \Delta^*$ gilt

$$f(x_1, \ldots, x_r) = (y_1, \ldots, y_s) \iff [\#x_1\#x_2\# \ldots \#x_rS\#] \Rightarrow^* [\#y_1\#y_2\# \ldots \#y_s\#]$$

Definition 102 Sei Δ ein Alphabet und $r, s \in \mathbb{N}$. Man sagt, eine partielle Funktion $f\colon (\Delta^*)^r \xrightarrow{\text{part}} (\Delta^*)^s$ sei *berechenbar* (oder *berechenbar im intuitiven Sinne*), wenn es einen Algorithmus gibt, der bei Eingabe von $(x_1, \ldots, x_r) \in (\Delta^*)^r$ die Ausgabe $f(x_1, \ldots, x_r)$ liefert, falls $(x_1, \ldots, x_r) \in \mathrm{Def}(f)$ ist und andernfalls nicht terminiert.

Bemerkung: Jede durch eine Grammatik berechenbare Funktion ist auch im intuitiven Sinne berechenbar. Hierzu betrachte man den Baum mit Wurzel $[\#x_1\#x_2\#\ldots\#x_rS\#]$ und der durch Ableitbarkeit in einem Schritt definierten Nachfolgerrelation. Man durchsuche diesen Baum in breadth first search-Reihenfolge solange, bis man einen Knoten gefunden hat, der mit einem Wort der Form $[\#y_1\#y_2\#\ldots\#y_s\#]$ markiert ist, wobei $y_1,\ldots,y_s \in \Delta^*$ sind. Das Tupel $(y_1,\ldots,y_s)$ ist dann der gesuchte Wert $f(x_1,\ldots,x_r)$ für die Ausgabe des Algorithmus.

Im folgenden sei | ein beliebig gewähltes Zeichen. Man kann jede natürliche Zahl n als das Wort $|^n \in \{|\}^*$ codieren. Verwendet man diese Codierung, so überträgt sich der Begriff der Berechnung einer partiellen Funktion durch eine Grammatik in natürlicher Weise auf partielle zahlentheoretische Funktionen oder allgemeiner auf partielle Funktionen von $\mathbb{N}^r$ nach $\mathbb{N}^s$ wie folgt.

Definition 103 Eine partielle Funktion $f\colon \mathbb{N}^r \xrightarrow{\text{part}} \mathbb{N}^s$ wird *durch eine Grammatik G berechnet*, wenn die Funktion

$$(|^{k_1},\ldots,|^{k_r}) \mapsto (|^{f_1(k_1,\ldots,k_r)},\ldots,|^{f_s(k_1,\ldots,k_r)})\colon (\{|\}^*)^r \xrightarrow{\text{part}} (\{|\}^*)^s$$

durch G berechnet wird, wobei $f(\mathfrak{x}) = (f_1(\mathfrak{x}),\ldots,f_s(\mathfrak{x}))$ ist für alle $\mathfrak{x} \in \mathbb{N}^r$.

Wenn $(k_1,\ldots,k_r) \in \mathrm{Def}(f)$ ist, dann muss also gelten

$$[\#|^{k_1}\#\ldots\#|^{k_r}S\#] \Rightarrow^* [\#|^{f_1(k_1,\ldots,k_r)}\#\ldots\#|^{f_s(k_1,\ldots,k_r)}\#].$$

Beispiel 64
Die Additionsfunktion wird durch die Grammatik $G = (V,\Sigma,R,S)$ berechnet, wobei $V = \{|,\#,[,],S\}$, $\Sigma = \{|,\#,[,]\}$ und $R = \{|S \to S|, \#S \to \varepsilon\}$.

Eine Grammatik G kann als ein spezieller Kalkül betrachtet werden mit den Regeln

$$x_0vx_1 \to x_0wx_1$$

für alle Produktionen $v \to w$ von G.

Fügt man zu diesem Kalkül die Regel $\to S$ hinzu, so erhält man den

Satz 48 *Der Sprachschatz einer Grammatik ist eine kalkülerzeugte Sprache.*

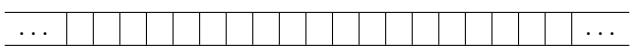
Mit Hilfe von Satz 38 zeigt man

Satz 49 *Jede durch eine Grammatik berechenbare partielle Funktion ist kalkülberechenbar und damit auch μ-partiellrekursiv.*

2.6 Turingmaschinen

2.6.1 Grundbegriffe

Eine Turingmaschine ist ein Automat, der sich zu jedem Zeitpunkt in einem Zustand aus einer vorgegebenen endlichen Zustandsmenge Q befindet und dem zusätzlich ein unbegrenzter Speicher in Form eines nach rechts und links unbegrenzten Bandes zur Verfügung steht. Dieses Band ist aufgeteilt in nebeneinander gereihte Felder.



Auf jedem der Felder steht ein Zeichen aus einem Alphabet (*Bandalphabet*) Σ. Der Automat besitzt einen Schreib/Lesekopf, der sich zu jedem Zeitpunkt auf genau einem der Felder befindet. Dieses Feld nennt man das (momentane) *Arbeitsfeld*. Die Situation einer Turingmaschine zu einem gegebenen Zeitpunkt nennen wir *Konfiguration*. Wir stellen sie mit einem Bild so dar, dass wir den Schreib/Lesekopf durch einen Pfeil symbolisieren, der auf das Arbeitsfeld zeigt. Darunter schreiben wir den Namen des momentanen Zustandes. In der folgenden Abbildung steht der Schreib/Lesekopf auf dem linken c und die Maschine befindet sich im Zustand p

$\ldots$	#	#	a	c	c	#	#	#	$\ldots$
				$\uparrow$					
				p					

bei einem Bandalphabet $\{a, b, c, \#\}$ und bei einem Band, das von links nach rechts zunächst unendlich oft das Zeichen #, dann a, c und c enthält und schließlich wieder unendlich oft das Zeichen #.

Eine Turingmaschine führt nacheinander jeweils einen Berechnungsschritt durch. Ein Berechnungsschritt besteht aus der Ausführung eines Elementarbefehls und dem Übergang der Turingmaschine in einen neuen Zustand. Es gibt drei Arten von Elementarbefehlen:

Elementarbefehle

1. Das Ersetzen des Zeichens auf dem Arbeitsfeld durch ein Zeichen $a \in \Sigma$. Diesen Elementarbefehl bezeichnen wir mit a.
2. Das Verschieben des Schreib/Lesekopfes um ein Feld nach links. Das neue Arbeitsfeld ist dann der linke Nachbar des alten Arbeitsfeldes. Diesen Elementarbefehl bezeichnen wir mit dem Zeichen L.
3. Das Verschieben des Schreib/Lesekopfes um ein Feld nach rechts. Das neue Arbeitsfeld ist dann der rechte Nachbar des alten Arbeitsfeldes. Diesen Elementarbefehl bezeichnen wir mit dem Zeichen R.

Die Elementarbefehle werden also mit Zeichen aus dem Alphabet $B := \Sigma \cup \{L, R\}$ bezeichnet, wobei L und R Zeichen sind, die nicht in Σ vorkommen.

Eines der Zeichen aus Σ wird ausgezeichnet und als *Leerzeichen* # bezeichnet. Am Anfang einer Berechnung enthält das Band die Eingabe in Form einer Beschriftung von endlich vielen Feldern mit Zeichen aus dem Bandalphabet Σ. Alle übrigen Felder sind dabei mit dem Leerzeichen # beschriftet. Der Inhalt des nicht leeren Teiles des Bandes nach der Berechnung dient als Ausgabe.

Definition 104 Eine *Turingmaschine* ist ein Quintupel $(Q, \Sigma, S, H, \delta)$ mit

1. Menge der *Zustände* Q ist eine endliche Menge.
2. *Bandalphabet* Σ ist ein Alphabet, *Leerzeichen* $\# \in \Sigma$ und $L, R \notin \Sigma$.
3. *Anfangszustand* oder *Startzustand* $S \in Q$.
4. *Endzustand* oder *Haltezustand* $H \in Q$ mit $H \neq S$.
5. *Übergangsfunktion* $\delta\colon (Q \setminus \{H\}) \times \Sigma \to B \times (Q \setminus \{S\})$ mit $B := \Sigma \cup \{L, R\}$.

Am Anfang einer Berechnung befindet sich die Turingmaschine im Anfangszustand S, am Ende einer Berechnung im Endzustand H. Befindet sie sich zu einem Zeitpunkt im Zustand q und ist das Zeichen auf dem Arbeitsfeld a und gilt $\delta(q,a) = (b,p)$, so führt die Turingmaschine den Elementarbefehl b aus und geht in den Zustand p über.

Notation 3 Als abkürzende Schreibweise für eine Turingmaschine verwenden wir die folgende: Für jedes $(q,a) \in (Q \setminus \{H\}) \times \Sigma$ schreiben wir eine Zeile $q\ a\ b\ p$, wobei $\delta(q,a) = (b,p)$. Für eine Turingmaschine $(Q, \Sigma, S, H, \delta)$ schreiben wir also

$$\begin{array}{cccc} q_1 & a_1 & b_1 & p_1 \\ & & \vdots & \\ q_n & a_n & b_n & p_n \end{array}$$

wobei $\{(q_i, a_i) \mid i = 1, \ldots, n\} = (Q \setminus \{H\}) \times \Sigma$ und $\delta(q_i, a_i) = (b_i, p_i)$ für $i = 1, \ldots, n$.

Um nicht jedesmal den Anfangs- und Endzustand explizit angeben zu müssen, vereinbaren wir außerdem, dass eine der Zeilen, die mit S beginnen, an den Anfang geschrieben wird und eine der Zeilen, die mit H enden, ans Ende: $q_1 = S$ und $p_n = H$. Außerdem vereinbaren wir, dass wir Zeilen der Form $q\ a\ a\ q$ weglassen dürfen.

Beispiel 65
Sei $\Sigma = \{a, b, c, \#\}$ ein Alphabet und $Q = \{S, q, H\}$. Sei $L_\#$ die Turingmaschine $(Q, \Sigma, S, H, \delta)$, wobei

$$\begin{aligned} \delta(S,a) &:= (L,q) \\ \delta(S,b) &:= (L,q) \\ \delta(S,c) &:= (L,q) \\ \delta(S,\#) &:= (L,q) \\ \delta(q,a) &:= (L,q) \\ \delta(q,b) &:= (L,q) \\ \delta(q,c) &:= (L,q) \\ \delta(q,\#) &:= (\#,H) \end{aligned}$$

Dann können wir für die Turingmaschine $L_\#$ kurz schreiben

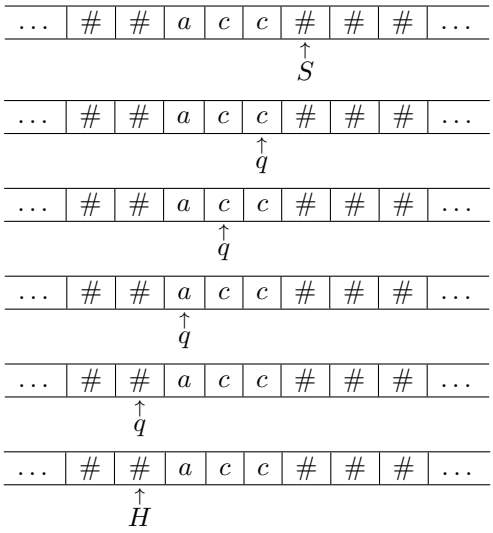
$$\begin{array}{cccc} S & a & L & q \\ S & b & L & q \\ S & c & L & q \\ S & \# & L & q \\ q & a & L & q \\ q & b & L & q \\ q & c & L & q \\ q & \# & \# & H \end{array}$$

Lässt man diese Turingmaschine etwa starten mit der Konfiguration

$\ldots$	$\#$	$\#$	a	c	c	$\#$	$\#$	$\#$	$\ldots$
						$\uparrow$			
						S			

,

so läuft die Berechnung folgendermaßen ab.



Hier ist die Maschine im Haltezustand und hält an. Diese Maschine macht also nichts anderes als vom anfänglichen Arbeitsfeld aus solange nach links zu gehen, bis sie ein Feld mit dem Leerzeichen # findet.

Wir werden Konfigurationen oft auch kurz so schreiben, dass wir die Abbildung des Turingbandes und den Pfeil sowie führende Leerzeichen links vom Arbeitsfeld und nachfolgende Leerzeichen rechts vom Arbeitsfeld weglassen. Für die Startkonfiguration im Beispiel schaut das dann so aus:

$$\underset{S}{acc\#}$$

Definition 105 Wenn T eine Turingmaschine ist, dann definieren wir die zweistellige Relation $\Rightarrow_T$ auf der Menge der Konfigurationen von T folgendermaßen. Für zwei Konfigurationen K_1 und K_2 gilt $K_1 \Rightarrow_T K_2$ genau dann, wenn die Turingmaschine T die Konfiguration K_1 in einem Berechnungsschritt in die Konfiguration K_2 überführt.

Im Beispiel gilt

$$\underset{S}{acc\#} \Rightarrow_{L\#} \underset{q}{acc} \Rightarrow_{L\#} \underset{q}{acc} \Rightarrow_{L\#} \underset{q}{acc} \Rightarrow_{L\#} \underset{q}{\#acc} \Rightarrow_{L\#} \underset{H}{\#acc}$$

Das Beispiel lässt sich verallgemeinern auf ein beliebiges Bandalphabet:

Beispiel 66
Sei $\Sigma = \{a_1, \ldots, a_k, \#\}$ ein Alphabet und $Q = \{S, q, H\}$. Sei $L_\#$ die Turingmaschine $(Q, \Sigma, S, H, \delta)$, wobei

$$\begin{aligned}
&\delta(S, a) := (L, q) && \text{für } a \in \Sigma \\
&\delta(q, a_i) := (L, q) && \text{für } i = 1, \ldots, k \\
&\delta(q, \#) := (\#, H)
\end{aligned}$$

Dann können wir für die Turingmaschine $L_\#$ kurz schreiben

$$\begin{array}{cccc} S & a & L & q \qquad (a \in \Sigma) \\ q & a_i & L & q \qquad (i = 1, \ldots, k) \\ q & \# & \# & H \end{array}$$

Definition 106 Wenn eine Turingmaschine T eine Konfiguration $u\underset{S}{a}v$ im Startzustand in eine Konfiguration $w\underset{H}{b}x$ im Haltezustand überführt, also $u\underset{S}{a}v \Rightarrow_T^* w\underset{H}{b}x$, so schreiben wir $u\underline{a}v \Rrightarrow_T w\underline{b}x$.

Zum Beispiel gilt im vorletzten Beispiel $acc\underline{\#} \Rrightarrow_{L_\#} \underline{\#}acc$. Allgemein:

Lemma 10 *Für alle $u, w \in \Sigma^*$, $a \in \Sigma$ und $v \in (\Sigma \setminus \{\#\})^*$ gilt*

$$u\#v\underline{a}w \Rrightarrow_{L_\#} u\underline{\#}vaw.$$

2.6.2 Zusammensetzung von Turingmaschinen

Seien $T, T', T_1, T_2, \ldots$ Turingmaschinen.

Definition 107 (von TT') Wenn

$$T = \begin{array}{cccc} q_1 & a_1 & b_1 & p_1 \\ & \vdots & & \\ q_m & a_m & b_m & p_m \end{array}, \qquad T' = \begin{array}{cccc} q'_1 & a'_1 & b'_1 & p'_1 \\ & \vdots & & \\ q'_n & a'_n & b'_n & p'_n \end{array}$$

und $p_m = q'_1$ und T und T' sonst keine gemeinsamen Zustände haben, dann ist

$$TT' = \begin{array}{cccc} q_1 & a_1 & b_1 & p_1 \\ & & \vdots & \\ q_m & a_m & b_m & p_m \\ q'_1 & a'_1 & b'_1 & p'_1 \\ & & \vdots & \\ q'_n & a'_n & b'_n & p'_n \end{array}.$$

Andernfalls benenne man die Zustände von T' vorher so um, dass die Bedingung für die Zustände erfüllt ist.

Beispiel 67
Die Turingmaschine $L_\#$ für das Bandalphabet $\{a, \#\}$ ist

$$\begin{array}{cccc} S & a & L & q \\ S & \# & L & q \\ q & a & L & q \\ q & \# & \# & H \end{array}$$

Wir wollen die Maschine $L_\# L_\#$ (wir schreiben auch $L_\#^2$) bestimmen. Hierzu müssen wir zwei Kopien der Zeilen von $L_\#$ untereinander schreiben. Bei der zweiten Kopie müssen wir die Zustände von $L_\#$ umbenennen, und zwar den Anfangszustand S in den Endzustand H der ersten Kopie von $L_\#$ und alle übrigen Zustände in neue Zustände, also etwa q in p und H in Z. Bezeichnen

wir das Ergebnis der Umbenennung der Zustände in der Turingmaschine $L_\#$ mit $\tilde{L}_\#$, so schauen die Zeilen von $\tilde{L}_\#$ so aus.

$$\begin{array}{cccc} H & a & L & p \\ H & \# & L & p \\ p & a & L & p \\ p & \# & \# & Z \end{array}$$

Die Zeilen der Maschine $L_\# L_\#$ sind nun die Zeilen von $L_\#$ gefolgt von den Zeilen von $\tilde{L}_\#$, also

$$\begin{array}{cccc} S & a & L & q \\ S & \# & L & q \\ q & a & L & q \\ q & \# & \# & H \\ H & a & L & p \\ H & \# & L & p \\ p & a & L & p \\ p & \# & \# & Z \end{array}$$

S ist der Startzustand von $L_\# L_\#$ und Z ist der Haltezustand von $L_\# L_\#$.

Lemma 11 *Seien $u, v, y, z \in \Sigma^*$ und $a, c \in \Sigma$. Dann gilt $u\underline{a}v \Rrightarrow_{TT'} y\underline{c}z$ genau dann, wenn es $w, x \in \Sigma^*$ und $b \in \Sigma$ gibt mit $u\underline{a}v \Rrightarrow_T w\underline{b}x$ und $w\underline{b}x \Rrightarrow_{T'} y\underline{c}z$.*

Eine Berechnung durch die Turingmaschine TT' bewirkt also das gleiche wie eine Berechnung durch T gefolgt von einer Berechnung durch T'.

Definition 108 Sind $q_1, p_1, \ldots, q_n, p_n$ beliebige Zustände und $a_1, \ldots, a_n \in \Sigma$ so, dass $(q_1, a_1), \ldots, (q_n, a_n)$ paarweise verschieden sind und ist q_i der Anfangszustand von T_i und p_i der Endzustand von T_i und sind alle übrigen Zustände von T_i verschieden von den Zuständen $q_1, p_1, \ldots, q_n, p_n$ und von den Zuständen aller anderen T_j, dann seien die Zeilen der Turingmaschine

$$\begin{array}{cccc} q_1 & a_1 & T_1 & p_1 \\ & & \vdots & \\ q_n & a_n & T_n & p_n \end{array}$$

alle Zeilen von T_1 außer denen, die mit $q_1 a$ beginnen mit $a \neq a_1$ und ... und alle Zeilen von T_n außer denen, die mit $q_n a$ beginnen mit $a \neq a_n$.

Andernfalls benenne man die Zustände von $T_1, \ldots, T_n$ vorher so um, dass die Bedingung für die Zustände erfüllt ist.

Beispiel 68
Die Turingmaschine $L_\#$ für das Bandalphabet $\{a, \#\}$ ist

$$\begin{array}{cccc} S & a & L & q \\ S & \# & L & q \\ q & a & L & q \\ q & \# & \# & H \end{array}$$

Daher ist

$$\begin{array}{cccc} S & a & L_\# & H \end{array}$$

die Turingmaschine

$$\begin{array}{cccc} S & a & L & q \\ q & a & L & q \\ q & \# & \# & H \end{array}$$

Sie ergibt sich aus $L_\#$ durch Streichung der zweiten Zeile, die mit S und einem von a verschiedenen Zeichen (nämlich # beginnt).

Lemma 12 *Es gilt* $u\underline{a}v \Rrightarrow_{\substack{q_1a_1T_1p_1 \\ \vdots \\ q_na_nT_np_n}} u'\underline{a}'v'$ *genau dann, wenn es* $i_1, \ldots, i_k$ *und* $u_1, v_1, \ldots, u_k, v_k \in \Sigma^*$ *gibt so, dass*

1. $q_{i_1} = q_1 \wedge \bigwedge_{j<k} p_{i_j} = q_{i_{j+1}} \wedge p_{i_k} = p_n$

2. $u\underline{a}v = u_1\underline{a_{i_1}}v_1 \Rrightarrow_{T_{i_1}} \ldots \Rrightarrow_{T_{i_{k-1}}} u_k\underline{a_{i_k}}v_k \Rrightarrow_{T_{i_k}} u'\underline{a}'v'$

   Die Turingmaschine

$$\begin{array}{cccc} q_1 & a_1 & T_1 & p_1 \\ & & \vdots & \\ q_n & a_n & T_n & p_n \end{array}$$

hat also die gleiche Wirkung, die wir erzielen würden, wenn wir den Begriff der Turingmaschine so erweitern würden, dass wir anstatt Elementarbefehlen auch ganze Turingmaschinen zulassen würden und dann diese Zeilen als Bezeichnung für eine so verallgemeinerte Turingmaschine auffassen würden. Die Schreibweise erlaubt uns also Turingmaschinen $T_1, \ldots, T_n$ als Unterprogramme in einer anderen Turingmaschine zu verwenden.

2.6.3 Weitere Beispiele für Turingmaschinen

Sei Σ ein fest vorgegebenes Alphabet mit $\# \in \Sigma$. Sei $\Delta := \Sigma \setminus \{\#\}$.

Suchen des nächsten Leerzeichens rechts vom Arbeitsfeld Sei $R_\#$ die Turingmaschine

$$\begin{array}{cccll} S & a & R & q & (a \in \Sigma) \\ q & a & R & q & (a \in \Delta) \\ q & \# & \# & H. & \end{array}$$

Lemma 13 *Für alle* $u, w \in \Sigma^*$, $a \in \Sigma$ *und* $v \in \Delta^*$ *gilt* $u\underline{a}v\#w \Rrightarrow_{R_\#} uav\underline{\#}w$.

Elementare Turingmaschinen Für $a \in \Sigma$ bezeichnen wir mit a die Turingmaschine

$$S \quad c \quad a \quad H \qquad (c \in \Sigma).$$

Lemma 14 *Für* $u, v \in \Sigma^*$ *und* $a, c \in \Sigma$ *gilt* $u\underline{c}v \Rrightarrow_a u\underline{a}v$.

Die Maschine L

$$S \quad a \quad L \quad H \qquad (a \in \Sigma)$$

Lemma 15 *Für* $u, v \in \Sigma^*$ *und* $a, b \in \Sigma$ *gilt* $ua\underline{b}v \Rrightarrow_L u\underline{a}bv$.

Die Maschine R

$$\begin{array}{cccc} S & a & R & H \end{array} \qquad (a \in \Sigma)$$

Lemma 16 *Für $u, v \in \Sigma^*$ und $a, b \in \Sigma$ gilt $ua\underline{a}bv \Rightarrow_R ua\underline{b}v$.*

Kopierung K_n

$$\begin{array}{cccc} S & \# & (L_\#^n R) & q \\ q & a & T_a & q \\ q & \# & R_\#^n & H \end{array} \qquad (a \in \Delta)$$

wobei $T_a = \#R_\#^{n+1} a L_\#^{n+1} aR$.

Lemma 17 *Für alle $u \in \Sigma^*$ und $w_1, \ldots, w_n \in \Delta^*$ gilt*

$$u\#w_1\# \ldots \#w_n\underline{\#} \Rightarrow_{K_n} u\#w_1\# \ldots \#w_n\#w_1\underline{\#}.$$

Linksverschiebung V

$$\begin{array}{cccc} S & \# & R & q \\ q & a & (LaRR) & q \\ q & \# & (L\#R) & H \end{array} \qquad (a \in \Delta)$$

Lemma 18 *Für $u, w \in \Sigma^*$ und $v \in \Delta^*$ gilt $u\underline{\#}v\#w \Rightarrow_V uv\#\underline{\#}w$.*

Streichung S_n

$$\begin{array}{cccc} S & \# & (L_\#^n L) & q \\ q & a & (\#RV^n L L_\#^n L) & q \\ q & \# & (RV^n L) & H \end{array} \qquad (a \in \Delta)$$

Lemma 19 *Für $u \in \Sigma^*$ und $w_0, \ldots, w_n \in \Delta^*$ gilt $u\#w_0\#w_1\# \ldots \#w_n\underline{\#} \Rightarrow_{S_n} u\#w_1\# \ldots \#w_n\underline{\#}$.*

2.6.4 Turing-Berechenbarkeit

Definition 109 Sei $f\colon (\Delta^*)^r \xrightarrow{\text{part}} \Delta^*$. Wir sagen, f werde *berechnet durch eine Turingmaschine* $T = (Q, \Sigma, S, H, \delta)$, wenn für alle $u_1, \ldots, u_r \in \Delta^*$ gilt

1. Wenn $(u_1, \ldots, u_r) \in \mathrm{Def}(f)$, dann $u_1\# \ldots \#u_r\underline{\#} \Rightarrow_T f(u_1, \ldots, u_r)\underline{\#}$.
2. Andernfalls gibt es keine Berechnung, die mit $u_1\# \ldots \#u_r\underset{S}{\#}$ beginnt und hält, d.h. im Zustand H endet.

Man sagt dann, f sei *turingberechenbar*.

Eine partielle zahlentheoretische Funktion $f\colon \mathbb{N}^r \xrightarrow{\text{part}} \mathbb{N}$ *wird durch eine Turingmaschine T berechnet* und heißt dann *turingberechenbar*, wenn das für die Funktion $(|^{a_1}, \ldots, |^{a_r}) \mapsto |^{f(a_1,\ldots,a_r)}\colon (\{|\}^*)^r \xrightarrow{\text{part}} \{|\}^*$ gilt.

Wir wollen zeigen, dass jede μ-partiellrekursive Funktion turingberechenbar ist. Hierzu benötigen wir als technisches Hilfsmittel eine etwas abgewandelte Variante des eben eingeführten Begriffs:

Definition 110 Eine *Turingmaschine für* eine partielle Funktion $f\colon (\Delta^*)^r \xrightarrow{\text{part}} (\Delta^*)^s$ ist eine Turingmaschine $T = (Q, \Sigma, S, H, \delta)$ so, dass für alle $x \in \Sigma^*$ und $u_1, \ldots, u_r \in \Delta^*$ gilt

1. Wenn $(u_1, \ldots, u_r) \in \mathrm{Def}(f)$, dann
$$x\#u_1\#\ldots\#u_r\underline{\#} \Rrightarrow_T x\#v_1\#\ldots\#v_s\underline{\#},$$
wobei $(v_1, \ldots, v_s) := f(u_1, \ldots, u_r)$ sei.
2. Andernfalls gibt es keine Berechnung, die mit $x\#u_1\#\ldots\#u_r\underset{S}{\#}$ beginnt und hält.

Analog wie oben sprechen wir auch von einer *Turingmaschine für* eine partielle zahlentheoretische Funktion.

Beispiel 69
K_n ist eine Turingmaschine für $(w_1, \ldots, w_n) \mapsto (w_1, \ldots, w_n, w_1)$. S_n ist eine Turingmaschine für $(w_0, w_1, \ldots, w_n) \mapsto (w_1, \ldots, w_n)$.

Lemma 20 *Wenn T eine Turingmaschine für eine r-stellige partielle Funktion f ist und $k \in \mathbb{N}$, dann ist T auch eine Turingmaschine für die $k+r$-stellige Funktion $x \cdot u \mapsto x \cdot f(u)$, wobei $\cdot$ die Operation des Aneinanderhängens von Tupeln sei.*

Im Folgenden werden einige Funktionen angegeben, für die es Turingmaschinen gibt, sowie Operationen, die, angewendet auf partielle Funktionen, für die Turingmaschinen existieren, wieder eine partielle Funktion liefern, für die eine Turingmaschine existiert.

Komposition Wenn T eine Turingmaschine für $f\colon (\Delta^*)^r \xrightarrow{\text{part}} (\Delta^*)^s$ ist und T' eine Turingmaschine für $g\colon (\Delta^*)^s \xrightarrow{\text{part}} (\Delta^*)^t$ ist, dann ist TT' eine Turingmaschine für $g \circ f$.

Permutation, Streichung, Vervielfältigung von Argumenten Seien $r, s \in \mathbb{N}$ und $i_1, \ldots, i_s \in \{1, \ldots, r\}$. Dann ist $K_{r+1-i_1} \ldots K_{r+s-i_s} S_s^r$ eine Turingmaschine für die Funktion
$$(w_1, \ldots, w_r) \mapsto (w_{i_1}, \ldots, w_{i_s})\colon (\Delta^*)^r \to (\Delta^*)^s.$$

Gibt es eine Turingmaschine für $f\colon (\Delta^*)^r \xrightarrow{\text{part}} \Delta^*$, so gibt es auch eine Turingmaschine für die Funktion
$$(w_1, \ldots, w_r) \mapsto (f(w_1, \ldots, w_r), w_1, \ldots, w_r)\colon (\Delta^*)^r \xrightarrow{\text{part}} (\Delta^*)^{r+1}.$$
Denn durch Komposition erhalten wir $(w_1, \ldots, w_r) \mapsto (w_1, \ldots, w_r, w_1, \ldots, w_r) \mapsto (w_1, \ldots, w_r, f(w_1, \ldots, w_r)) \mapsto (f(w_1, \ldots, w_r), w_1, \ldots, w_r)$.

Zusammensetzung von einwertigen Funktionen zu einer mehrwertigen Funktion Für
$$f_i\colon (\Delta^*)^r \xrightarrow{\text{part}} \Delta^* \qquad (i = 1, \ldots, s)$$
sei $[f_1, \ldots, f_s]\colon (\Delta^*)^r \xrightarrow{\text{part}} (\Delta^*)^s$ definiert durch
$$[f_1, \ldots, f_s](\mathfrak{w}) := (f_1(\mathfrak{w}), \ldots, f_s(\mathfrak{w})) \qquad \text{für} \quad \mathfrak{w} \in (\Delta^*)^r.$$

Wenn es Turingmaschinen für $f_1, \ldots, f_s$ gibt, dann gibt es auch eine Turingmaschine für $[f_1, \ldots, f_s]$. Denn durch Komposition erhalten wir $\mathfrak{w} \mapsto (f_1(\mathfrak{w})) \cdot \mathfrak{w} \mapsto \cdots \mapsto (f_1(\mathfrak{w}), \ldots, f_s(\mathfrak{w})) \cdot \mathfrak{w} \mapsto (f_1(\mathfrak{w}), \ldots, f_s(\mathfrak{w}))$.

Schleife mit Abbruch beim leeren Wort Für $1 \leq k \leq n$ sei die Projektion $P_k^n \colon (\Delta^*)^n \to \Delta^*$ definiert durch $P_k^n(w_1, \ldots, w_n) := w_k$. Für $n \in \mathbb{N}$ und $f \colon (\Delta^*)^{n+1} \xrightarrow{\text{part}} (\Delta^*)^{n+1}$ sei die partielle Funktion $\mathrm{Schl}_n(f) \colon (\Delta^*)^{n+1} \xrightarrow{\text{part}} (\Delta^*)^{n+1}$ folgendermaßen definiert. Für alle $\mathfrak{w} \in (\Delta^*)^{n+1}$ gelte

- Wenn ein $k \in \mathbb{N}$ exisitiert mit $P_{n+1}^{n+1}(f^k(\mathfrak{w})) = \varepsilon$, so ist $\mathrm{Schl}_n(f)(\mathfrak{w}) := f^r(\mathfrak{w})$, wobei r das kleinste solche k ist.
- Andernfalls ist $\mathrm{Schl}_n(f)(\mathfrak{w})$ undefiniert.

Wenn T eine Turingmaschine für f ist, dann ist

$$\begin{array}{cccc} S & \# & L & q \\ q & a & (RTL) & q \\ q & \# & R & H \end{array} \qquad (a \in \Delta)$$

eine Turingmaschine für $\mathrm{Schl}_n(f)$. Die Wirkung dieser Turingmaschine entspricht der eines Pseudocodes

while $w_{n+1} \neq \varepsilon$
do $(w_1, \ldots, w_{n+1}) := f(w_1, \ldots, w_{n+1})$

Als Beispiel für $\mathrm{Schl}_n(f)$ betrachten wir etwa $\Delta = \{|\}$ und die folgendermaßen definierte Funktion $f \colon (\Delta^*)^2 \to (\Delta^*)^2$

$$\begin{aligned} f(u, \varepsilon) &:= (u, \varepsilon) \\ f(u, x|) &:= (uu, x) \end{aligned}$$

für $u, x \in \Delta^*$. Dann ist

$$f(|^m, |^n) = (|^{2m}, |^{n-1}) \qquad \text{für } m, n \in \mathbb{N},\ n > 0.$$

Für $k \leq n$ ist $f^k(|^m, |^n) = (|^{m \cdot 2^k}, |^{n-k})$ und daher $P_2^2(f^k(|^m, |^n)) = |^{n-k}$. Das kleinste k mit $P_2^2(f^k(|^m, |^n)) = \varepsilon$ ist daher n. Also ist

$$\mathrm{Schl}_1(f)(|^m, |^n) = f^n(|^m, |^n) = (|^{m \cdot 2^n}, \varepsilon).$$

Ein Pseudocode, der die Funktion $\mathrm{Schl}_1(f)$ berechnet, ist etwa der folgende.

u := $|^m$
x := $|^n$
while $x \neq \varepsilon$ do
    ⌜ Entferne einen Strich aus x
      u := uu ⌟
return (u,x)

Vorgängerfunktion

$$\begin{array}{cccc} S & \# & L & q \\ q & a & \# & H \\ q & \# & R & H \end{array} \qquad (a \in \Delta)$$

ist eine Turingmaschine für die Funktion $V \colon \Delta^* \to \Delta^*$ mit $V(wa) := w$ für $w \in \Delta^*$, $a \in \Delta$ und $V(\varepsilon) := \varepsilon$. Im Fall $\Delta = \{|\}$ entspricht die Funktion V gerade der Vorgängerfunktion auf der Menge der natürlichen Zahlen.

2.7 Äquivalenz von μ-Partiellrekursivität, Kalkülberechenbarkeit, Grammatikberechenbarkeit und Turingberechenbarkeit

Satz 50 *Jede μ-partiellrekursive partielle zahlentheoretische Funktion ist turingberechenbar.*

Beweis: Für jede μ-partiellrekursive partielle zahlentheoretische Funktion gibt es eine Turingmaschine. Denn

1. R ist eine Turingmaschine für O.
2. $|R$ ist eine Turingmaschine für N.
3. Es gibt eine Turingmaschine für P_i^n (Streichung von Argumenten).
4. $(fg_1 \dots g_m)$ ergibt sich durch Komposition aus f und $[g_1, \dots, g_m]$.
5. Sei f n-stellig und g $(n+2)$-stellig. Dann ist $(Rfg) = k \circ \mathrm{Schl}_{n+2}(j) \circ h$, wobei
$$\begin{aligned} h(\mathfrak{x}, y) &:= (f(\mathfrak{x}), \mathfrak{x}, 0, y) \\ j(u, \mathfrak{x}, y, z) &:= (g(u, \mathfrak{x}, y), \mathfrak{x}, N(y), V(z)) \\ k(u, \mathfrak{x}, y, z) &:= u. \end{aligned}$$
6. Sei f $(n+1)$-stellig. Dann ist $(\mu f) = k \circ \mathrm{Schl}_{n+1}(j) \circ h$, wobei
$$\begin{aligned} h(\mathfrak{x}) &:= (\mathfrak{x}, 0, f(\mathfrak{x}, 0)) \\ j(\mathfrak{x}, y, z) &:= (\mathfrak{x}, N(y), f(\mathfrak{x}, N(y))) \\ k(\mathfrak{x}, y, z) &:= y. \end{aligned}$$

q.e.d.

Definition 111 Man sagt, eine partielle Funktion $f\colon (\Delta^*)^r \xrightarrow{\text{part}} \Delta^*$ sei *μ-partiellrekursiv*, wenn es eine μ-partiellrekursive Funktion $g\colon \mathbb{N}^r \xrightarrow{\text{part}} \mathbb{N}$ gibt so, dass für alle $w_1, \dots, w_r \in \Delta^*$ gilt:

1. $\ulcorner f(w_1, \dots, w_r) \urcorner = g(\ulcorner w_1 \urcorner, \dots, \ulcorner w_r \urcorner)$, falls dies definiert ist.
2. $f(w_1, \dots, w_r)$ ist undefiniert sonst.

Satz 51 *Jede μ-partiellrekursive Funktion $f\colon \mathbb{N}^r \xrightarrow{\text{part}} \mathbb{N}$ oder $f\colon (\Delta^*)^r \xrightarrow{\text{part}} \Delta^*$ ist turingberechenbar.*

Beweis: Für $f\colon \mathbb{N}^r \xrightarrow{\text{part}} \mathbb{N}$: letzter Satz. Nun sei $f\colon (\Delta^*)^r \xrightarrow{\text{part}} \Delta^*$ und g wie oben. Nach dem letzten Satz gibt es eine Turingmaschine für g, also für

$$(|^{x_1}, \dots, |^{x_r}) \mapsto |^{g(x_1,\dots,x_r)}\colon (\{|\}^*)^r \xrightarrow{\text{part}} \{|\}^*.$$

Nun gibt es Turingmaschinen für $w \mapsto |^{\ulcorner w \urcorner}\colon \Delta^* \to \{|\}^*$ und für $|^{\ulcorner w \urcorner} \mapsto w\colon \{|\}^* \xrightarrow{\text{part}} \Delta^*$ (ohne Beweis). Durch Komposition ergibt sich f:

$$(w_1, \dots, w_r) \mapsto (|^{\ulcorner w_1 \urcorner}, \dots, |^{\ulcorner w_r \urcorner}) \mapsto |^{g(\ulcorner w_1 \urcorner,\dots,\ulcorner w_r \urcorner)} = |^{\ulcorner f(w_1,\dots,w_r) \urcorner} \mapsto f(w_1, \dots, w_r).$$

q.e.d.

Satz 52 *Jede turingberechenbare partielle Funktion* $f\colon (\Delta^*)^r \xrightarrow{\text{part}} \Delta^*$ *ist durch eine Grammatik berechenbar.*

Zum Beweis verwendet man als nichtterminale Zeichen der zu konstruierenden Grammatik die Zustände der Turingmaschine und codiert eine Konfiguration uav der Turingmaschine als Wort $[uqav]$ der Grammatik. Dies ist Thema der Aufgabe 5 von Blatt 9 und soll an dieser Stelle nicht näher ausgeführt werden.

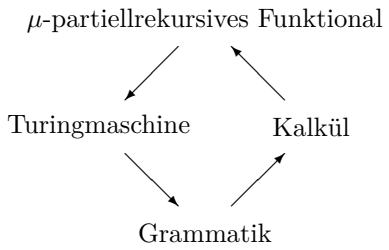
2.8 Partiellrekursive und rekursive Funktionen

Fassen wir einige der Ergebnisse zusammen, die wir in den letzten Abschnitten über partielle Funktionen bewiesen haben.

Sei f eine beliebige partielle Funktion $f\colon \mathbb{N}^n \xrightarrow{\text{part}} \mathbb{N}$ oder $f\colon (\Delta^*)^n \xrightarrow{\text{part}} \Delta^*$, wobei Δ ein Alphabet ist. Dann sind die folgenden Aussagen äquivalent

- f ist μ-partiellrekursiv
- f ist durch einen Kalkül berechenbar
- f ist durch eine Grammatik berechenbar
- f ist durch eine Turingmaschine berechenbar

Wir haben unter anderen die folgenden Simulationen gezeigt.



Man hat die Äquivalenz auch für den Begriff der Berechenbarkeit partieller Funktionen in weiteren formalen Systemen nachgewiesen, darunter der λ-Kalkül, der Kalkül der Kombinatoren und alle gängigen Programmiersprachen.

Definition 112 Wenn eine dieser zueinander äquivalenten Aussagen gilt, dann nennt man die partielle Funktion f *partiellrekursiv*. Ist f partiellrekursiv und total (d.h. $\text{Def}(f) = \mathbb{N}^n$ bzw. $\text{Def}(f) = (\Delta^*)^n$), so nennt man die Funktion f *rekursiv*.

Man verwendet die Begriffe 'partiellrekursiv' und 'rekursiv' dann, wenn es einem darum geht zu sagen, dass eine partielle / totale Funktion in einem dieser Formalismen berechenbar ist, wenn man sich aber nicht auf einen bestimmten dieser Formalismen festlegen will.

2.9 Churchsche These

Beim Versuch den Begriff der Berechenbarkeit von partiellen oder totalen Funktionen zu formalisieren hat man verschiedene formale Berechnungssysteme aufgestellt, darunter die folgenden Systeme.

- μ-partiellrekursive Funktionale
- Kalküle
- Turingmaschinen
- Grammatiken
- λ-Kalkül
- Kombinatorenkalkül
- Markov-Algorithmen
- Termersetzungssysteme (rewrite systems)
- Post-Systeme
- Universelle Programmiersprachen (C, Lisp, Prolog, Smalltalk, ...)

Einige dieser Systeme verfolgen voneinander grundverschiedene Ansätze. Trotzdem hat sich gezeigt, dass alle diese Systeme den gleichen Berechenbarkeitsbegriff beschreiben, also zueinander äquivalent sind. Eine in einem dieser Systeme (und damit in allen dieser Systeme) beschreibbare partielle bzw. totale Funktion heißt *partiellrekursiv* bzw. *rekursiv*.

Church und Turing haben nun die unter dem Namen 'Churchsche These' bekannte These aufgestellt, dass diese Formalismen bereits alle berechenbaren partiellen Funktionen erfassen.

Churchsche These Eine partielle Funktion ist genau dann berechenbar, wenn sie partiellrekursiv ist.

Die Churchsche These ist keine mathematisch formulierbare Aussage und daher nicht mathematisch beweisbar. Sie sagt aus, dass die aufgestellten Beschreibungssysteme für partiell berechenbare Funktionen adäquate mathematische Beschreibungen für den intuitiven Begriff der paritellen Berechenbarkeit darstellen.

Für totale Funktionen besagt die Churchsche These, dass eine totale Funktion genau dann berechenbar ist, wenn sie rekursiv ist.

2.10 Normalformsatz von Kleene

Aus der Definition des Begriffs der partiellrekursiven Funktion und aus Satz **??** folgt unmittelbar der

Satz 53 (Normalformsatz von Kleene) *Es gibt eine 1-stellige primitivrekursive Funktion U und zu jedem $r \in \mathbb{N}$ eine $r+2$-stellige primitivrekursive Funktion T_r so, dass für jede r-stellige partiellrekursive partielle zahlentheoretische Funktion f eine natürliche Zahl e existiert so, dass*

$$f(\mathfrak{x}) = \begin{cases} (U(\mu T_r))(e, \mathfrak{x}), & \textit{falls dies definiert ist} \\ \textit{undefiniert} & \textit{sonst} \end{cases}$$

für alle $\mathfrak{x} \in \mathbb{N}^r$.

Diese Zahl e nennt man einen *Index* der partiellen zahlentheoretischen Funktion f. Man schreibt dann oft $f = \{e\}^r$.

Die zweistellige partielle zahlentheoretische Funktion $h := (U(\mu T_1))$ ist eine *universelle* partiellrekursive Funktion. Sie erlaubt die Berechnung jeder einstelligen (und durch Tupelcodierung auch jeder mehrstelligen) partiellrekursiven Funktion f, da $f(x) = h(e, x)$ ist, wobei e ein Index von f ist.

Zum Beweis des Normalformsatzes von Kleene haben wir jedem Kalkül eine Gödelnummer zugeordnet und ein Index einer partiellrekursiven Funktion ist die Gödelnummer eines Kalküls, der diese Funktion berechnet. Ebenso, wie man einem Kalkül eine Gödelnummer zuordnet, kann man auch einer Turingmaschine eine Gödelnummer zuordnen und mit Hilfe dieser Gödelisierung den Normalformsatz von Kleene beweisen. Dann wird man im Allgemeinen andere primitivrekursive Funktionen U und T_r erhalten. Aber auch dann gibt es zu jeder partiellrekursiven Funktion einen Index, der dann aber die Gödelnummer einer Turingmaschine ist, die diese Funktion berechnet. Übrigens hat jede partiellrekursive Funktion unendlich viele Indizes.

2.11 Rekursiv aufzählbare und rekursive Mengen

Definition 113 Eine Menge $A \subseteq \mathbb{N}^n$ oder $A \subseteq (\Delta^*)^n$ heißt *rekursiv aufzählbar*, wenn sie der Definitionsbereich einer partiellrekursiven Funktion ist. Sie heißt *rekursiv*, wenn ihre charakteristische Funktion χ_A rekursiv ist.

Definition 114 Eine Menge $A \subseteq \mathbb{N}^n$ oder $A \subseteq (\Delta^*)^n$ heißt *semientscheidbar*, wenn es einen Algorithmus gibt, der bei Eingabe von $\mathfrak{a} \in \mathbb{N}^n$ bzw. $\mathfrak{a} \in (\Delta^*)^n$ genau dann hält, wenn $\mathfrak{a} \in A$ ist. Sie heißt *entscheidbar*, wenn ihre charakteristische Funktion χ_A berechenbar ist.

Nach der Churchschen These ist eine Menge genau dann semientscheidbar, wenn sie rekursiv aufzählbar ist, und genau dann entscheidbar, wenn sie rekursiv ist.

Für den folgenden Satz benötigen wir die einstellige partielle zahlentheoretische Funktion undef, die überall undefiniert ist:

$$\text{undef}\colon \mathbb{N} \xrightarrow{\text{part}} \mathbb{N} \qquad \text{mit} \qquad \text{Def}(\text{undef}) = \emptyset.$$

Es gilt undef $= (\mu K_1^2)$. Daher ist undef partiellrekursiv.

Satz 54 *Für eine Menge $A \subseteq \mathbb{N}$ sind die folgenden Aussagen äquivalent.*

1. *A ist rekursiv aufzählbar.*
2. *Es gibt ein zweistelliges primitivrekursives Prädikat P mit $A = \{x \mid \bigvee_{y \in \mathbb{N}} P(x, y)\}$.*
3. *A ist die leere Menge $\emptyset$ oder der Wertebereich einer einstelligen primitivrekursiven Funktion.*
4. *A ist der Wertebereich einer einstelligen partiellrekursiven Funktion.*

Beweis: 1. $\Longrightarrow$ 2.: Sei A rekursiv aufzählbar. Dann gibt es eine einstellige partiellrekursive Funktion f mit $A = \text{Def}(f)$. Nach dem Normalformsatz von Kleene hat $f(x)$ die Form $(U(\mu T_1))(e, x)$. Also ist $A = \text{Def}(f) = \{x \mid \bigvee_{y \in \mathbb{N}} T_1(e, x, y) = 0\}$. Sei $P(x, y) :\Longleftrightarrow T_1(e, x, y) = 0$. Dann folgt die Behauptung.

2. $\Longrightarrow$ 1.: Aus 2. folgt $A = \text{Def}((\mu P))$.

2. $\Longrightarrow$ 3.: Es gelte 2. und sei $A \neq \emptyset$. Dann sei $a \in A$ beliebig gewählt und

$$f(\langle x, y\rangle) := \begin{cases} x, & \text{wenn } P(x,y) \\ a & \text{sonst.} \end{cases}$$

Dann ist f eine einstellige primitivrekursive Funktion und A der Wertebereich von f.

3. $\Longrightarrow$ 4.: Wenn $A = \emptyset$, dann ist A Wertebereich der einstelligen partiellrekursiven Funktion undef. Andernfalls ist die Behauptung trivial.

4. $\Longrightarrow$ 2.: Sei A Wertebereich einer 1-stelligen partiellrekursiven Funktion f. Nach dem Normalformsatz von Kleene hat $f(x)$ die Form $(U(\mu T_1))(e,x)$. Für $x, y \in \mathbb{N}$ sei

$$R(x,y) :\Longleftrightarrow T_1(e,x,y) = 0 \wedge \bigwedge_{u<y} T_1(e,x,u) > 0.$$

Dann ist R primitivrekursiv und für $x, y \in \mathbb{N}$ gilt

$$R(x,y) \iff y = (\mu T_1)(e,x).$$

Für $w, x, y \in \mathbb{N}$ sei

$$Q(w,x,y) :\Longleftrightarrow R(x,y) \wedge w = U(y).$$

Dann ist Q ein dreistelliges primitivrekursives Prädikat und für $w \in \mathbb{N}$ gilt

$$\begin{aligned} w \in A &\iff \bigvee_{x\in\mathbb{N}} w = f(x) \\ &\iff \bigvee_{x\in\mathbb{N}} \bigvee_{y\in\mathbb{N}} (y = (\mu T_1)(e,x) \wedge w = U(y)) \\ &\iff \bigvee_{x\in\mathbb{N}} \bigvee_{y\in\mathbb{N}} Q(w,x,y). \end{aligned}$$

Mit $P(w, \langle x, y\rangle) :\Longleftrightarrow Q(w,x,y)$ folgt 2. q.e.d.

Mit Hilfe der Tupelcodierung durch die Cantor'sche Paarfunktion bzw. mit Hilfe der Gödelisierung lässt sich der Satz sinngemäß auf Teilmengen A von $\mathbb{N}^n$ oder von $(\Delta^*)^n$ übertragen. Da diese Übertragbarkeit generell für primitivrekursive, rekursive und partiellrekursive Funktionen, für rekursive und für rekursiv aufzählbare Mengen gilt, werden wir einige Resultate nur für die Menge $\mathbb{N}$ formulieren ohne explizit anzumerken, dass sie sich einfach auf $\mathbb{N}^n$ und auf $(\Delta^*)^n$ übertragen lassen.

Eine Menge $A \subseteq \mathbb{N}$ ist genau dann rekursiv, wenn ihr *Komplement* $\mathbb{N} \setminus A$ rekursiv ist. Denn $\chi_{\mathbb{N}\setminus A} = (\overline{\text{sg}}\, \chi_A)$.

Im Gegensatz dazu ist das Komplement einer rekursiv aufzählbaren Menge im Allgemeinen nicht rekursiv aufzählbar, wie wir später sehen werden. Es gilt

Satz 55 *Eine Menge $A \subseteq \mathbb{N}$ ist genau dann rekursiv, wenn A und $\mathbb{N} \setminus A$ rekursiv aufzählbar sind.*

Beweis: $\Rightarrow$: Sei A rekursiv. Dann ist χ_A rekursiv und damit sind die einstelligen Funktionen $f := (\mu(\overline{\text{sg}}(\chi_A P_1^2)))$ und $g := (\mu(\chi_A P_1^2))$ partiellrekursiv und damit die Mengen $A = \text{Def}(f)$ und $\mathbb{N} \setminus A = \text{Def}(g)$ rekursiv aufzählbar.

$\Leftarrow$: Seien A und $\mathbb{N} \setminus A$ rekursiv aufzählbar. Nach dem letzten Satz gibt es dann zweistellige primitivrekursive Prädikate P und Q mit

$$A = \{x \mid \bigvee_{y \in \mathbb{N}} P(x,y)\}$$
$$\mathbb{N} \setminus A = \{x \mid \bigvee_{y \in \mathbb{N}} Q(x,y)\}.$$

Für $x, y \in \mathbb{N}$ sei

$$R(x,y) : \Longleftrightarrow\ P(x,y) \vee Q(x,y).$$

Dann gilt

$$\bigwedge_{x \in \mathbb{N}} \bigvee_{y \in \mathbb{N}} R(x,y).$$

Also ist (μR) total, also rekursiv. Für $x \in \mathbb{N}$ gilt $\chi_A(x) = \chi_P(x, (\mu R)(x))$. Daher ist auch χ_A und damit A rekursiv. q.e.d.

Definition 115 Sei T eine Turingmaschine mit Bandalphabet Σ und $\Delta \subseteq \Sigma \setminus \{\#\}$. Man sagt, T *akzeptiere* ein Wort $w \in \Delta^*$, wenn T bei Start mit der Konfiguration $w\underset{S}{\#}$ anhält, d.h. nach endlich vielen Schritten in den Haltezustand übergeht. Die Berechnung, die T dabei ausführt heißt *akzeptierende Berechnung* durch T für w. Die von T *akzeptierte Sprache* ist die Menge der von T akzeptierten Wörter.

Satz 56 *Eine Sprache wird genau dann von einer Turingmaschine akzeptiert, wenn sie rekursiv aufzählbar ist.*

Beweis: $\Leftarrow$: Sei L rekursiv aufzählbar. Dann ist L Definitionsbereich einer partiellrekursiven Funktion f. Daher wird f durch eine Turingmaschine T berechnet. Die Sprache L wird dann von T akzeptiert.

$\Rightarrow$: Sei T eine Turingmaschine, die die Sprache L akzeptiert. Wenn T eine partielle Funktion f berechnet, dann ist f partiellrekursiv und L der Definitionsbereich von f. Also ist L rekursiv aufzählbar.

Andernfalls konstruiert man aus T eine neue Turingmaschine T', die die gleiche Sprache L akzeptiert und eine partielle Funktion berechnet. Hierzu fügt man ein neues Zeichen $\#'$ zum Bandalphabet hinzu und ersetzt jede Zeile $q\#bp$ durch die beiden Zeilen $q\#(\#'b)p$ und $q\#'bp$. Sei T'' die so entstandene Turingmaschine und $T' := T''\#'R_\#$. q.e.d.

Definition 116 Eine Sprache $L \subseteq \Delta^*$ heißt *Ausgabesprache* einer Turingmaschine T, wenn

$$L = \{v \in \Delta^* \mid \bigvee_{u \in \Delta^*} u\underline{\#} \Rrightarrow_T v\underline{\#}\}.$$

Satz 57 *Eine Sprache ist genau dann Ausgabesprache einer Turingmaschine, wenn sie rekursiv aufzählbar ist.*

Zum Beweis bemerkt man, dass jede rekursiv aufzählbare Sprache L der Wertebereich einer partiellrekursiven Funktion ist. Diese Funktion wird von einer Turingmaschine berechnet und die Ausgabesprache dieser Turingmaschine ist dann die Sprache L.

Wenn umgekehrt L die Ausgabesprache einer Turingmaschine T ist, dann kann man T so zu einer Turingmaschine T' abändern, dass T' dasselbe Ergebnis wie T liefert, wenn T ein Wort aus Δ^* liefert, und andernfalls nicht hält. Die Turingmaschine T' berechnet dann eine partielle Funktion mit Wertebereich L. Daher ist L dann rekursiv aufzählbar.

Definition 117 Man sagt, eine Sprache $L \subseteq \Delta^*$ werde von einer Turingmaschine T *aufgezählt*, wenn es einen Zustand q von T gibt so, dass gilt

$$L = \{w \in \Delta^* \mid \bigvee_{u \in \Delta^*} \underset{S}{\#} \Rightarrow_T^* \underset{q}{u\#w\#}\}.$$

Satz 58 *Eine Sprache wird genau dann von einer Turingmaschine aufgezählt, wenn sie rekursiv aufzählbar ist.*

Satz 59 *Eine Sprache ist genau dann Sprachschatz einer Grammatik, wenn sie rekursiv aufzählbar ist.*

Allgemein ist eine von irgendeinem formalen System akzeptierte oder erzeugte Sprache immer rekursiv aufzählbar (nach der Churchschen These).

2.12 Varianten von Turingmaschinen

Wir haben im Beweis der Turingberechenbarkeit der μ-partiellrekursiven Funktionen bei allen Berechnungen durch eine Turingmaschine die Felder links von den Eingabewörtern nicht berührt. Wir können uns also auf ein einseitig (links) begrenztes Band beschränken. Andererseits kann man den Begriff der Turingmaschine erweitern.

Die folgenden Maschinen können einander gegenseitig simulieren

- Turingmaschinen (mit beidseitig unbegrenztem Band)
- Turingmaschinen mit einseitig begrenztem Band
- Turingmaschinen mit mehreren Bändern
- Turingmaschinen mit mehreren Köpfen
- Turingmaschinen mit mehrdimensionalen Bändern

sowie alle Kombinationen dieser Erweiterungen.

2.12.1 Nichtdeterministische Turingmaschinen

Die bisher definierten Turingmaschinen werden auch *deterministische Turingmaschinen* genannt, da für jede Konfiguration der nächste Berechnungsschritt eindeutig festgelegt ist und damit zu jeder Anfangskonfiguration die gesamte Berechnung eindeutig bestimmt (determiniert) ist. Bei einer *nichtdeterministischen Turingmaschine* hat die Maschine bei jedem Berechnungsschritt die Wahl zwischen endlich vielen Wahlmöglichkeiten.

Definition 118 Eine *nichtdeterministische Turingmaschine* ist ein Quintupel $(Q, \Sigma, S, H, \Delta)$ mit

1.–4. wie für (deterministische) Turingmaschinen

5′. *Übergangsrelation* $\Delta \subseteq (Q \setminus \{H\}) \times \Sigma \times B \times (Q \setminus \{S\})$.

Eine Zeile einer solchen nichtdeterministischen Turingmaschine ist dann $q\ \ a\ \ b\ \ p$ mit $(q, a, b, p) \in \Delta$. Hier können auf eine Konfiguration null, ein oder mehr Zeilen anwendbar sein. Entsprechend kann es zu einer Anfangskonfiguration mehrere verschiedene mögliche Berechnungen geben.

Definition 119 Man sagt, eine nichtdeterministische Turingmaschine T *akzeptiere* ein Wort w, wenn es eine akzeptierende Berechnung durch T für w gibt. Die von T *akzeptierte Sprache* ist die Menge der von T akzeptierten Wörter.

Man kann jede nichtdeterministische Turingmaschine durch eine deterministische Turingmaschine in dem Sinne „simulieren“, dass man etwa mit breadth first search alle möglichen Berechnungen der nichtdeterministischen Turingmaschine nachvollzieht — ähnlich, wie wir dies für Grammatiken gemacht haben. Insbesondere gilt daher

Satz 60 *Jede von einer nichtdeterministischen Turingmaschine akzeptierte Sprache ist rekursiv aufzählbar.*

Die Simulation kann aber sehr viel länger werden als die ursprüngliche Berechnung. Wenn zum Beispiel die nichtdeterministische Turingmaschine bei jeder Konfiguration 2 Wahlmöglichkeiten hat und die Berechnung die Länge n hat, so muss man schlimmstenfalls 2^n Berechnungen durchsuchen.

2.13 Universelle Turingmaschinen

Für eine Turingmaschine T sei die *Gödelnummer* $\ulcorner T \urcorner$ von T die Gödelnummer des zu T zugeordneten Kalküls.

Sei h die universelle partiellrekursive Funktion $(U(\mu T_1))$ vom Normalformsatz von Kleene. Dann gilt

> Wenn eine Turingmaschine T eine partiellrekursive Funktion $f\colon \mathbb{N} \xrightarrow{\text{part}} \mathbb{N}$ berechnet, dann gilt für alle $x \in \mathbb{N}$
>
> $$f(x) = \begin{cases} h(\ulcorner T \urcorner, x), & \text{wenn dies definiert ist} \\ \text{undefiniert} & \text{sonst.} \end{cases}$$

Da h partiellrekursiv ist, gibt es eine Turingmaschine, die h berechnet. Diese Turingmaschine heißt *universelle Turingmaschine*, da sie zur Berechnung jeder beliebigen partiellrekursiven Funktion f verwendet werden kann. Gibt man ihr nämlich als Eingaben die beiden Zahlen $\ulcorner T \urcorner$ und x, wobei T wie oben sei, dann berechnet sie daraus $f(x)$, sofern es existiert, und hält andernfalls nicht.

Man kennt ähnliche universelle Maschinen auch für Programmiersprachen. Dort versteht man unter einem *Interpreter* für eine Programmiersprache ein Computerprogramm, das zu einem beliebigen Programm in der Programmiersprache und zu einer beliebigen Eingabe die zugehörige Ausgabe berechnet. Man kann einen Interpreter in derselben Programmiersprache schreiben oder in einer anderen.

2.14 Unentscheidbare Probleme

2.14.1 Das Halteproblem für Turingmaschinen

Halteproblem: Gegeben eine Turingmaschine T und eine Eingabe x. Frage: Hält T bei Eingabe x?

Wir fragen danach, ob das Halteproblem für Turingmaschinen entscheidbar ist. Den Begriff der Entscheidbarkeit hatten wir definiert für Mengen von Tupeln von natürlichen Zahlen oder von Wörtern. Wir müssen also eine Turingmaschine durch eine natürliche Zahl oder durch ein Wort darstellen. Hierzu bietet sich die Gödelisierung an. Wir geben also die Turingmaschine durch Angabe ihrer Gödelnummer g und wir fragen danach, ob g die Gödelnummer (GN) einer Turingmaschine (TM) ist, die bei (der Eingabe) $x \in \mathbb{N}$ hält. Wir definieren daher

$$H := \{(g, x) \mid g \text{ ist GN einer TM, die bei } x \text{ hält}\}.$$

Satz 61 *H ist rekursiv aufzählbar.*

Beweis: Die universelle Turingmaschine hält bei Eingabe (g, x) genau dann, wenn g Gödelnummer einer Turingmaschine ist, die bei Eingabe x hält. Daher akzeptiert sie die Menge H. Daher ist H rekursiv aufzählbar. q.e.d.

Folgerung: Das Halteproblem für Turingmaschinen ist semientscheidbar.

Satz 62 *H ist nicht rekursiv.*

Beweis: Wir verwenden das Cantor'sche Diagonalverfahren. Die Diagonale D von H ist gegeben durch

$$D := \{g \mid g \text{ ist GN einer TM, die bei } g \text{ hält}\}.$$

Angenommen, H wäre rekursiv. Dann wäre auch D rekursiv und damit

$$\mathbb{N} \setminus D = \{g \mid g \text{ ist nicht GN einer TM, die bei } g \text{ hält}\}$$

rekursiv aufzählbar und daher die von einer TM S akzeptierte Menge. Also

$$\bigwedge_{g \in \mathbb{N}} (S \text{ hält bei } g \iff g \text{ ist nicht GN einer TM, die bei } g \text{ hält}).$$

Für $g := \ulcorner S \urcorner$ bedeutet dies

$$S \text{ hält bei } \ulcorner S \urcorner \iff S \text{ hält nicht bei } \ulcorner S \urcorner.$$

Dies ist ein Widerspruch. q.e.d.

Aus diesen beiden Sätzen und der Churchschen These folgt

Folgerung: Das Halteproblem für Turingmaschinen ist semientscheidbar, aber unentscheidbar.

Es gilt sogar

Folgerung Es gibt eine Turingmaschine, deren akzeptierte Sprache unentscheidbar ist (z.B. die universelle Turingmaschine)

Beispiel 70
für eine Menge, die rekursiv aufzählbar, aber nicht rekursiv ist: die Menge H.

2.14.2 Terminierung von Computerprogrammen

Definition 120 Eine Programmiersprache heißt *universell*, wenn sich darin jede partiellrekursive Funktion durch ein Programm implementieren lässt.

Zu den universellen Programmiersprachen gehören alle gängigen allgemeinen Programmiersprachen wie C, Lisp, Prolog, Smalltalk, usw. Nicht dazu gehören einige spezialisierte Sprachen, z.B. einige Datenbanksprachen und die Macro-Sprachen einiger Texteditoren.

Das Cantor'sche Diagonalverfahren lässt sich auch auf jede universelle Programmiersprache anwenden. So kann ein Programm P, das einen String einliest, auch sich selbst einlesen. Wir definieren

$$P \text{ ist selbstterminierend} : \Longleftrightarrow P \text{ terminiert bei Eingabe } P$$

Für eine universelle Programmiersprache gilt

Satz 63 *Die Terminierung von Programmen ist unentscheidbar.*

Denn andernfalls wäre nach der Churchschen These die Menge der terminierenden Paare (Programm, Eingabe) rekursiv. Da die Programmiersprache universell ist, gäbe es dann ein Programm, das feststellen kann, ob ein beliebig gegebenes Programm bei einem beliebig gegebenen Eingabewort terminiert. Dann könnte man aber auch ein Programm P_0 schreiben, das das Folgende tut.

> Lies P.
> Wenn P selbstterminierend ist, dann gehe in eine Endlosschleife.
> Andernfalls brich ab.

Dann gälte

$$\begin{aligned} P_0 \text{ ist selbstterminierend} &\Longleftrightarrow P_0 \text{ terminiert bei der Eingabe } P_0 \\ &\Longleftrightarrow P_0 \text{ ist nicht selbstterminierend} \end{aligned}$$

Dies wäre ein Widerspruch.

2.14.3 Weitere unentscheidbare Probleme

Die folgenden Probleme sind unentscheidbar

- Allgemeingültigkeit prädikatenlogischer Formeln
- Terminierung von Computerprogrammen
- Korrektheit von Computerprogrammen
- Frage, ob ein gegebenes Wort im Sprachschatz einer gegebenen Grammatik liegt

Es gibt

- Turingmaschinen mit unentscheidbarer akzeptierter Sprache
- Turingmaschinen mit unentscheidbarer erzeugter Sprache
- Grammatiken mit unentscheidbarem Sprachschatz
- Computerprogramme, von denen nicht entscheidbar ist, für welche Eingaben sie terminieren

Kapitel 3

Komplexitätstheorie

3.1 Einleitung

Die *Komplexitätstheorie* befasst sich mit dem Resourcenverbrauch von Algorithmen. Die wichtigsten Resourcen sind *Zeit* (engl. *time*) und *Speicherplatz* (engl. *space*). Entsprechend unterscheidet man zwischen *Zeit-* und *Speicherplatzkomplexität*.

Man interessiert sich insbesondere dafür, wie schnell der Resourcenverbrauch mit der Größe des Problems ansteigt. Hierzu verwendet man die *Landausymbole* O und o. Wenn $f\colon \mathbb{N} \to \mathbb{R}$, dann ist

$$O(f) := \{g\colon \mathbb{N} \to \mathbb{N} \mid \bigvee_{c\in\mathbb{R}^+} \bigvee_{n_0\in\mathbb{N}} \bigwedge_{n\geq n_0} g(n) \leq c \cdot f(n)\}$$
$$o(f) := \{g\colon \mathbb{N} \to \mathbb{N} \mid \bigwedge_{c\in\mathbb{R}^+} \bigvee_{n_0\in\mathbb{N}} \bigwedge_{n\geq n_0} g(n) \leq c \cdot f(n)\}$$

Statt $g \in O(f)$ schreibt man meistens $g(n) = O(f(n))$ und statt $g \in o(f)$ meistens $g(n) = o(f(n))$.

Beispiel 71

$$\sum_{k=1}^{n} k = O(n^2)$$
$$\sum_{k=1}^{n} \frac{1}{k} = O(\log n)$$

Bei $O(\log n)$ braucht man die Basis des Logarithmus nicht anzugeben, da es bei den Landausymbolen auf einen konstanten Faktor nicht ankommt und da

$$\log_a n = \log_a b \cdot \log_b n$$

ist.

Man unterscheidet weiter zwischen dem Anwachsen der Resourcen im *schlimmsten Fall* (engl. *worst case*), im *Mittel* (engl. *average case*) und im *besten Fall* (engl. *best case*).

Beispiel 72
Algorithmus zum Suchen eines Elementes x in einer sortierten Liste von n Elementen.

1. Vergleiche x mit dem Element y in der Mitte der Liste.
2. Wenn $x < y$, suche in der Teilliste vor y.
3. Wenn $x = y$, Element gefunden.
4. Wenn $x > y$, suche in der Teilliste hinter y.

Dieser rekursiv definierte Algorithmus hat Zeitkomplexität

- $O(\log n)$ im schlimmsten Fall
- $O(\log n)$ im Mittel
- $O(1)$ im besten Fall.

und Speicherplatzkomplexität $O(n)$.

Beispiel 73
Der Algorithmus Quicksort zum Sortieren einer Liste von n Elementen hat Zeitkomplexität

- $O(n^2)$ im schlimmsten Fall
- $O(n \cdot \log n)$ im Mittel

Offenbar hängen Zeitkomplexität und Speicherplatzkomplexität eines Algorithmus vom *Maschinenmodell* ab. Es gibt Probleme, die sich mit einer (Einband-)Turingmaschine nur mit Zeitkomplexität $O(n^2)$, aber mit einer Mehrband-Turingmaschine mit Zeitkomplexität $O(n)$ lösen lassen.

Man sagt, ein Algorithmus habe *polynomielle Zeitkomplexität*, wenn es ein Polynom p gibt so, dass die Rechenzeit bei Eingabe eines Wortes einer Länge n höchstens $p(n)$ beträgt.

Wenn ein Algorithmus in einem der bekannten Maschinenmodelle polynomielle Zeitkomplexität hat, dann hat er auch auf einer geeigneten Turingmaschine polynomielle Zeitkomplexität.

Denn die Realisierung eines Algorithmus in einem Maschinenmodell lässt sich durch eine Turingmaschine polynomiell simulieren, d.h. es gibt ein Polynom q so, dass, wenn der Algorithmus im Maschinenmodell eine Zeit t benötigt, die Turingmaschine eine Zeit $\leq q(t)$ benötigt. Wenn der Algorithmus im Maschinenmodell eine Zeit $\leq p(n)$ benötigt, dann benötigt die Turingmaschine eine Zeit $\leq q(p(n))$. Und für zwei Polynome p und q ist auch $q \circ p$ ein Polynom.

3.2 P und NP

Definition 121 Eine deterministische oder nichtdeterministische Turingmaschine T *akzeptiert* eine Sprache $L \subseteq \Delta^*$ *in polynomieller Zeit*, wenn es ein Polynom p gibt so, dass gilt

1. T akzeptiert jedes Wort $w \in L$ in höchstens $p(|w|)$ Berechnungsschritten.
2. T akzeptiert kein Wort aus $\Delta^* \setminus L$.

Definition 122

- P ist die Klasse aller Sprachen, die in polynomieller Zeit von einer deterministischen Turingmaschine akzeptiert werden.

- NP ist die Klasse aller Sprachen, die in polynomieller Zeit von einer nichtdeterministischen Turingmaschine akzeptiert werden.

Definition 123 Unter einem *nichtdeterministischen Algorithmus* versteht man ein formalisierbares Rechenverfahren, bei dem — im Gegensatz zum (deterministischen) Algorithmus — bei jedem Berechnungsschritt noch eine Wahl zwischen mehreren Möglichkeiten offengelassen werden darf.

Definition 124 Sei A ein nichtdeterministischer Algorithmus.

- A *akzeptiert* ein Wort w, wenn es für A eine haltende Berechnung bei Eingabe w gibt.
- Die von A *akzeptierte Sprache* ist die Menge der von A akzeptierten Wörter.
- A *akzeptiert eine Sprache* $L \subseteq \Delta^*$ *in polynomieller Zeit*, wenn es ein Polynom p gibt so, dass gilt
  1. A akzeptiert jedes Wort $w \in L$ in höchstens $p(|w|)$ Berechnungsschritten.
  2. A akzeptiert kein Wort aus $\Delta^* \setminus L$.

Eine Sprache L ist genau dann in P, wenn es einen (deterministischen) Algorithmus gibt, der von einem Wort in polynomieller Zeit entscheiden kann, ob es in L ist.

Eine Sprache L ist genau dann in NP, wenn sie von einem nichtdeterministischen Algorithmus in polynomieller Zeit akzeptiert wird.

Beispiel 74

- $\{a^n b^n \mid n \in \mathbb{N}\} \in P$.
- $\{d_n \# d_j \# d_k \mid n, j, k \in \mathbb{N} \wedge \bigvee_{t \in \mathbb{N}} (j \leq t \leq k \wedge t|n)\} \in NP$, wobei d_n die Dezimaldarstellung von n sei für $n \in \mathbb{N}$.

  Z.B. ist das Wort 38#17#20 in dieser Menge, da 19|38 und $17 \leq 19 \leq 20$ ist.

  Eine Sprache kann man auch als Problem formulieren. Im letzten Beispiel:

  Gegeben $n, j, k \in \mathbb{N}$. Frage: Gibt es ein $t \in \mathbb{N}$ mit $j \leq t \leq k$ und $t|n$.

Man sagt auch, dieses Problem sei ein *NP-Problem*. Falls es auch ein P-Problem ist, dann ist Primfaktorzerlegung in polynomieller Zeit möglich. Bis heute weiß man nicht, ob Primfaktorzerlegung in polynomieller Zeit möglich ist.

Es gilt $P \subseteq NP$.

3.3 Das $P = NP$-Problem

Das wohl bekannteste bisher ungelöste Problem der Informatik ist das

$P = NP$-Problem: Gilt $P = NP$?

Man weiß bis heute nicht, ob die Antwort auf diese Frage „ja“ oder „nein“ ist. Eine Antwort „ja“ würde bedeuten, dass eine ganze Reihe von schwierigen Problemen der Informatik, darunter das Erfüllbarkeitsproblem der Aussagenlogik und das Problem des Handelsreisenden, in polynomieller Zeit lösbar wären, dass es dafür also (einigermaßen) effiziente Algorithmen gäbe. Bisher hat man aber für keines dieser Probleme einen in polynomieller Zeit terminierenden Algorithmus gefunden. Daher vermuten die meisten Informatiker, dass es einen solchen Algorithmus nicht gibt, dass also $P \neq NP$ ist. Andererseits konnte man trotz intensivster Bemühungen bisher keinen Beweis dafür finden, dass $P \neq NP$ ist.

3.4 Das Erfüllbarkeitsproblem der Aussagenlogik

Gegeben seien abzählbar unendlich viele Zeichen $U_1, U_2, \ldots$, genannt *Aussagenvariable*.

Definition 125 (Induktive Definition der Formeln)

1. Jede Aussagenvariable U ist eine Formel.
2. Ist F eine Formel, dann ist auch $\neg F$ eine Formel.
3. Sind F und G Formeln, dann sind auch $(F \wedge G)$ und $(F \vee G)$ Formeln.

Definition 126 Eine *Belegung* einer Formel F ist eine Abbildung i von der Menge der in F auftretenden Aussagenvariablen in die Menge der Wahrheitswerte $\{w, f\}$.

Definition 127 Der *Wahrheitswert* iF einer Formel F bei einer Belegung i ist definiert durch

1. $iU := i(U)$
2. $i\neg F := \begin{cases} w, & \text{wenn } iF = f \\ f & \text{sonst} \end{cases}$
3. $i(F \wedge G) := \begin{cases} w, & \text{wenn } iF = w \text{ und } iG = w \\ f & \text{sonst} \end{cases}$
4. $i(F \vee G) := \begin{cases} w, & \text{wenn } iF = w \text{ oder } iG = w \\ f & \text{sonst} \end{cases}$

Definition 128 Ein *Modell* einer Formel F ist eine Belegung i mit $iF = w$. Eine Formel heißt *erfüllbar*, wenn sie ein Modell hat.

Jede Formel kann als Wort über $\Delta := \{U, {}_0, \ldots, {}_9, (,), \neg, \wedge, \vee\}$ geschrieben werden.

Satz 64 *Es gibt eine Turingmaschine, die zu einem Wort $F \in \Delta^*$ der Länge n in $O(2^n)$ Schritten entscheiden kann, ob F eine erfüllbare Formel ist.*

Dies geht mit dem folgenden Algorithmus:

1. Wenn F nicht Formel, dann brich ab mit ‘nein’.

2. Suche Modell von F.

3. Gib aus, ob Modell gefunden.

Schritt 1 geht in polynomieller Zeit.
F hat höchstens $\frac{n+1}{2}$ Aussagenvariable und daher höchstens $2^{\frac{n+1}{2}}$ Belegungen. Überprüfung, ob Belegung Modell ist, geht in polynomiell vielen Schritten und daher in Zeit $O(2^{\frac{n-1}{2}})$. Also geht Schritt 2 in Zeit $O(2^n)$.
Schritt 3 geht in Zeit $O(1)$.
Insgesamt braucht der Algorithmus also eine Zeit $O(2^n)$.
Es ist nicht schwierig (aber recht mühsam) diesen Algorithmus in Form einer (deterministischen) Turingmaschine zu formulieren, die zu einem Wort $F \in \Delta^*$ der Länge n in $O(2^n)$ Schritten entscheiden kann, ob F eine erfüllbare Formel ist.

Satz 65 *Die Menge der erfüllbaren Formeln liegt in NP.*

Denn der nichtdeterministische Algorithmus

1. Wenn F nicht Formel, halte nicht.

2. Rate eine Belegung i von F.

3. Wenn i Modell von F, dann halte, sonst halte nicht.

akzeptiert die Menge der erfüllbaren Formeln in polynomieller Zeit. Dieser nichtdeterministische Algorithmus lässt sich auch als nichtdeterministische Turingmaschine formulieren.

3.5 NP-Vollständigkeit

Definition 129 Eine totale Funktion $f\colon \Delta^* \to \Delta^*$ heißt *polynomialzeitberechenbar*, wenn es eine (deterministische) Turingmaschine T und ein Polynom p gibt so, dass gilt

1. T berechnet f.

2. Wenn $w \in \Delta^*$ und $|w| = n$, dann hat die Berechnung durch T bei Eingabe w höchstens $p(n)$ Schritte.

Definition 130 Eine Sprache $L' \subseteq \Delta^*$ ist auf eine Sprache $L \subseteq \Delta^*$ *polynomialzeitreduzierbar* (in Zeichen $L' \leq_p L$), wenn es eine polynomialzeitberechenbare Funktion f gibt so, dass für alle $w \in \Delta^*$ gilt

$$w \in L' \iff f(w) \in L.$$

Satz 66 $L_1 \leq_p L_2 \wedge L_2 \leq_p L_3 \Longrightarrow L_1 \leq_p L_3$.

Satz 67 $L \in P \wedge L' \leq_p L \Longrightarrow L' \in P$.

Dasselbe gilt für NP.

Definition 131 Eine Sprache L ist *NP-hart*, wenn jede Sprache aus NP auf L polynomialzeitreduzierbar ist. L ist *NP-vollständig*, wenn $L \in NP$ ist und L NP-hart ist.

Satz 68 *Wenn eine NP-vollständige Sprache in P liegt, dann ist $P = NP$.*

Beweis: Sei $L \in P$ NP-vollständig und $L' \in NP$. Dann ist $L' \leq_p L$ und daher nach dem letzten Satz $L' \in P$. q.e.d.

3.6 NP-Vollständigkeit von SAT

Definition 132

- Ein *Literal* ist eine Aussagenvariable U oder die Negation $\neg U$ einer Aussagenvariablen.
- Eine *Klausel* ist eine Disjunktion $L_1 \vee \cdots \vee L_k$ von Literalen.
- Eine *konjunktive Normalform* (*KNF*) ist eine Konjunktion von Klauseln:

$$(L_{11} \vee \cdots \vee L_{1k_1}) \wedge \cdots \wedge (L_{j1} \vee \cdots \vee L_{jk_j})$$

Definition 133 SAT ist die Menge der erfüllbaren KNFen.

Lemma 21 *$SAT \in NP$.*

Lemma 22 *SAT ist NP-hart.*

Beweis: (Nur Idee): Sei $L \in NP$. Zu zeigen: $L \leq_p$ SAT.
Es gibt eine nichtdeterministische Turingmaschine T, die L in polynomieller Zeit akzeptiert. Jedes $w \in L$ wird in höchstens $p(n)$ Schritten akzeptiert ($n := |w|$).
Wir repräsentieren eine Berechnung B durch T als Belegung i.
Wir ordnen jeder möglichen Eingabe w eine aussagenlogische Formel $F = f(w)$ zu so, dass gilt

i ist Modell von $f(w)$
$\iff$ B ist Berechnung durch T für Eingabe w in höchstens $p(n)$ Schritten

(wobei $n := |w|$). Hierzu betrachten wir alle Zeitpunkte t, alle Felder f (Feld 0 ist das Feld, das am Anfang der Berechnung das Arbeitsfeld war, links davon die Felder mit negativer Nummer f und rechts die Felder mit positiver Nummer f), alle Zustände und alle Zeichen, die in einer Berechnung mit Eingabe w eine Rolle spielen können. Es sind dies

- t Zeit ($0 \leq t \leq p(n)$)
- f Feld ($-p(n) \leq f \leq p(n)$)
- q Zustand (alle Zustände der Turinmaschine)
- z Zeichen (alle Zeichen des Bandalphabets)

Für jedes dieser t, f, q und z führen wir die folgenden Aussagenvariablen ein, denen wir uns, falls eine Berechnung vorliegt, jeweils die hinter dem Doppelpunkt stehende Aussage als Bedeutung zugeordnet denken.

- Q_{tq}: Zur Zeit t ist der Zustand q.
- A_{tf}: Zur Zeit t ist Feld f das Arbeitsfeld.
- Z_{tfz}: Zur Zeit t steht auf Feld f das Zeichen z.

Man kann dann die Aussage

Es liegt eine Berechnung vor, die w in höchstens $p(n)$ Schritten akzeptiert

als KNF F formulieren, die genau dann erfüllbar ist, wenn eine solche Berechnung existiert.

$$w \in L \iff F \in \text{SAT}.$$

Wenn zum Beispiel $p\ a\ b\ q$ die einzige Zeile ist, die mit $p\ a$ beginnt, dann könnte man diese Zeile durch die folgenden Implikationen ausdrücken.

$$\begin{aligned} Q_{tp} \wedge A_{tf} \wedge Z_{tfa} &\to Q_{t+1,q} \\ Q_{tp} \wedge A_{tf} \wedge Z_{tfa} &\to A_{t+1,f} \\ Q_{tp} \wedge A_{tf} \wedge Z_{tfa} &\to Z_{t+1,f,b} \\ Q_{tp} \wedge A_{tf} \wedge Z_{tfa} \wedge Z_{tf'x} &\to Z_{t+1,f',x} \qquad (f' \neq f). \end{aligned}$$

Für jedes zulässige t, p, f, f' und x hat man eine solche Implikation. Jede dieser Implikationen lässt sich wieder als Klausel schreiben, zum Beispiel die erste Implikation als

$$\neg Q_{tp} \vee \neg A_{tf} \vee \neg Z_{tfa} \vee Q_{t+1,q}.$$

Entsprechend bekommt man auch Klauseln für alle anderen Zeilen der nichtdeterministischen Turingmaschine T. Zusätzlich zu den Klauseln, die den Zeilen der Turingmaschine entsprechen und die die Abarbeitung durch die Turingmaschine beschreiben, muss man noch Klauseln angeben, die ausdrücken, dass die Turingmaschine mit dem Wort w im Anfangszustand startet und dass sie sich nach höchstens $p(n)$ Schritten im Haltezustand befindet. Man muss auch explizit angeben, dass eine Turingmaschine nicht gleichzeitig in zwei Zuständen sein kann und dass nicht auf einem Feld gleichzeitig zwei Zeichen stehen dürfen.

Die sich schließlich ergebende Formel F ist zwar recht kompliziert hinzuschreiben, aber ihre Länge ist höchstens ein Polynom von der Länge n der Eingabe w der Turingmaschine T. Darüberhinaus lässt sie sich aus w in polynomieller Zeit berechnen (auch mit einer deterministischen Turingmaschine, obwohl es sehr aufwendig wäre diese Turingmaschine hinzuschreiben).

Es gilt also: $f : w \mapsto F$ ist polynomialzeitberechenbar. Also ist $L \leq_p \text{SAT}$. Da dies für jede Sprache $L \in NP$ gilt, ist SAT NP-hart. q.e.d.

Satz 69 *SAT ist NP-vollständig.*

3.7 Nachweis der NP-Vollständigkeit einer Sprache durch Polynomialzeitreduktion

Wir haben gesehen, dass es recht aufwendig ist von einer Sprache nachzuweisen, dass sie NP-vollständig ist. Wenn man aber bereits von einer Sprache L weiß, dass sie NP-vollständig ist, dann kann man dieses Wissen dazu verwenden auch die NP-Vollständigkeit anderer Sprachen nachzuweisen. Hierzu verwendet man den folgenden Satz.

Satz 70 *Sei L NP-vollständig, $L' \in NP$ und $L \leq_p L'$. Dann ist L' NP-vollständig.*

Beweis: Da $L' \in NP$ vorausgesetzt wurde, ist noch zu zeigen, dass L' NP-hart ist, d.h. dass jede Sprache L'' aus NP auf L' polynomialzeitreduzierbar ist. Hierzu nehmen wir an $L'' \in NP$. Da L NP-vollständig ist, gilt $L'' \leq_p L$. Nach Voraussetzung ist $L \leq_p L'$. Wegen der Transitivität von $\leq_p$ ist daher $L'' \leq_p L'$. q.e.d.

Hieraus ergibt sich die folgende

Methode zum Nachweis der NP-Vollständigkeit einer Sprache: Um zu zeigen, dass eine Sprache L' NP-vollständig ist, zeigt man, dass $L' \in NP$ ist und dass irgendeine bekanntermaßen NP-vollständige Sprache L auf L' polynomialzeitreduzierbar ist.

Beispiel 75 (Das Cliquenproblem)
Gegeben ein ungerichteter Graph G und $n \in \mathbb{N}$. Frage: Gibt es einen vollständigen Teilgraphen ('Clique') von G mit n Ecken?

Man kann zu jeder KNF F sehr leicht einen Graphen G und eine natürliche Zahl n angeben so, dass F genau dann erfüllbar ist, wenn es in G eine Clique aus n Ecken gibt (hier ohne Beweis). Die Berechnung von G und n aus F geht in Polynomialzeit.

SAT ist also polynomialzeitreduzierbar auf das Cliquenproblem. Daher ist auch das Cliquenproblem NP-vollständig.

Weitere NP-vollständige Probleme:

- Problem des Handelsreisenden
- Rucksackproblem
- ...

Wenn eines dieser Probleme in P ist, dann sind alle NP-vollständigen Probleme (und überhaupt alle NP-Probleme) in P.

3.8 Kryptologie

Die Kryptologie ist die Theorie der Verschlüsselung von Nachrichten. Es gibt verschiedene Verfahren der Verschlüsselung. Hier soll nur eine Klasse von Verfahren kurz erwähnt werden.

Public Key-Systeme:
Dabei handelt es sich um ein Verfahren, bei dem zwei Schlüssel verwendet werden. Es wird also (im Gegensatz zu den symmetrischen Verschlüsselungsverfahren) zur Entschlüsselung ein anderer Schlüssel verwendet als zur Verschlüsselung. Einer der beiden Schlüssel ist öffentlich, d.h. er wird öffentlich bekanntgegeben. Der andere Schlüssel ist privat und wird geheim gehalten. Man kann ein Public-Key-System verwenden um eine Nachricht als Geheimnachricht zu verschlüsseln so, dass nur ein einziger Empfänger sie wieder entschlüsseln kann. Man kann sie aber auch verwenden um eine nicht geheime Nachricht zu autentifizieren, d.h. mit einer fälschungssicheren elektronischen Unterschrift zu versehen. Entsprechend des Verwendungszwecks werden die Schlüssel folgendermaßen angewendet.

- öffentlicher Schlüssel des Empfängers zur Verschlüsselung
- privater Schlüssel des Empfängers zur Entschlüsselung

(für Geheimnachricht) oder

- privater Schlüssel des Absenders zur Verschlüsselung
- öffentlicher Schlüssel des Absenders zur Entschlüsselung

(für autentifizierte Nachricht).

Damit die notwendige Sicherheit gewährleistet ist, muss die Berechnung des privaten Schlüssels aus dem öffentlichen einen hohen Rechenaufwand erfordern.

Z.B.: Privater Schlüssel: 2 Primzahlen, öffentlicher Schlüssel: Das Produkt dieser Primzahlen.

Steganographie: Verstecken einer verschlüsselten Nachricht in einer unverfänglichen Nachricht (z.B. Bilddatei oder Tondatei) so, dass nur mit hohem Rechenaufwand feststellbar ist, dass eine Nachricht versteckt ist.

Kapitel 4

Formale Semantik

Die Semantik ist die Lehre von der Bedeutung. Bei der Semantik von Programmiersprachen unterscheidet man zwischen *deklarativer* oder *denotationeller Semantik* und *prozeduraler* oder *operationaler Semantik*. Erstere befasst sich mit der Wirkung eines Programms als Ganzes, letztere mit der Art und Weise, wie ein Programm in einzelnen Schritten abgearbeitet wird.

4.1 Semantik in der logischen Programmierung

Deklarative Semantik wird in der logischen Programmierung durch den Begriff der *korrekten Antwortsubstitution* und den Begriff des *kleinsten Herbrandmodells* gegeben.

Bei definiten Klauseln (=Hornklauseln) ist θ eine *korrekte Antwortsubstitution* für ein Programm P und eine Zielklausel $\leftarrow A_1, \ldots, A_k$ genau dann, wenn

$$P \models \forall[(A_1 \wedge \cdots \wedge A_k)\theta].$$

Eine *Herbrandinterpretation* ist eine Interpretation, deren Individuenbereich die Menge der Grundterme ist und die jeden Grundterm durch sich selbst interpretiert. Eine Herbrandinterpretation ist eindeutig bestimmt durch Angabe der Menge der in dieser Interpretation geltenden Grundatomformeln. Es wird daher oft mit dieser Menge identifiziert. Insbesondere sagt man, eine Herbrandinterpretation I sei *kleiner* als eine andere J, wenn jede in I geltende Grundatomformel auch in J gilt. Ein *Herbrandmodell* einer Formel ist eine Herbrandinterpretation, die Modell dieser Formel ist.

Kleinstes Herbrandmodell M_P von P ist die Menge der Grundatomformeln, die semantisch aus P folgen.

Prozedurale Semantik: Für eine Herbrandinterpretation I sei $T_P(I)$ die Menge der Grundatomformeln, die sich mit P in einem Schritt aus Formeln aus I ableiten lassen.

Äquivalenz von deklarativer und prozeduraler Semantik: M_P ist der kleinste Fixpunkt von T_P.

Bei nichtdefiniten Klauseln

$$\text{comp}(P) \models \forall[(A_1 \wedge \cdots \wedge A_k)\theta].$$

comp(P) ist die *Clarksche Vervollständigung* von P.

Z.B. Programm P:

$$\begin{aligned} p(y) &\leftarrow q(y), \neg r(a,y) \\ p(f(z)) &\leftarrow \neg q(z) \\ p(b) &\leftarrow \end{aligned}$$

Dann ist comp(P) die Formel

$$\begin{aligned} \forall x \big(p(x) \leftrightarrow & \exists y (x = y \wedge q(y) \wedge \neg r(a,y)) \\ & \vee \exists z (x = f(z) \wedge \neg q(z)) \\ & \vee x = b \big) \\ \wedge \quad & \text{Gleichheitstheorie.} \end{aligned}$$

4.2 Andere Programmiersprachen

Generell zeichnen sich deklarative Programmiersprachen durch eine klarere einfacher überschaubare Semantik aus als etwa prozedurale oder auf while-Schleifen oder gar auf goto-Anweisungen aufgebaute Programmiersprachen. Deklarative Programmierung ist etwa logische Programmierung oder funktionale Programmierung.

4.2.1 Funktionale Programmierung

Hier sind Programme aufgebaut aus Funktionen, die als mathematische Funktionen interpretiert werden können.

Kombinatoren sind solche Funktionen. Meist wird der λ-Kalkül verwendet. λ-Ausdrücke bezeichnen mathematische Funktionen.

4.2.2 While-Programme

Variablen können Werte haben. Zu jedem Zeitpunkt liegt eine Interpretation im Sinne der Prädikatenlogik erster Stufe vor. Sie wird hier ein *Zustand* z genannt.

$z[\![\alpha]\!]z'$ bedeutet, dass das Programm α den Zustand z in den Zustand z' überführt.
t_z sei der Wert eines Terms t im Zustand z.
$z(X/w)$ sei wie Zustand z, nur dass Variable X den Wert w hat.
ϵ sei das leere Programm

Denotationelle Semantik

- $z[\![\epsilon]\!]z' \iff z = z'$
- $z[\![X := t]\!]z' \iff z' = z(X/t_z)$
- $z[\![\text{if } B \text{ then } \beta \text{ else } \gamma \text{ end}]\!]z' \iff$ (B gilt in z und $z[\![\beta]\!]z'$) oder (B gilt in z nicht und $z[\![\gamma]\!]z'$)
- $z[\![\text{while } B \text{ do } \beta \text{ end}]\!]z' \iff$ es gibt Zustände $z_0, \ldots, z_n$ mit
  - $z = z_0$
  - B gilt in z_i und $z_i[\![\beta]\!]z_{i+1}$ für $i = 0, \ldots, n-1$

  - B gilt nicht in z_n
  - $z_n = z'$
- $z[\![A_1 \dots A_n]\!]z' \iff$ es gibt Zustände $z_0, \dots, z_n$ mit
  - $z = z_0$
  - $z_i[\![A_{i+1}]\!]z_{i+1}$
  - $z_n = z'$

Operationale Semantik ordnet einem Programm und einem Zustand den Rest des Programms nach Abarbeitung eines Schrittes und den dann erreichten Zustand zu. Sie ist äquivalent zur denotationalen Semantik.

Hoarscher Kalkül Wenn ϕ und ψ prädikatenlogische Formeln sind und α ein Programm ist, dann bedeutet $\{\phi\}\alpha\{\psi\}$, dass, wenn vor Abarbeitung des Programms α die Formel ϕ gilt, dann nach Abarbeitung des Programms α die Formel ψ gilt. Der Hoarsche Kalkül gibt Regeln an um Aussagen der Form $\{\phi\}\alpha\{\psi\}$ abzuleiten.