Mathematisches Basiswissen für Informatiker

Prof. Dr. Manfred Schmidt-Schauß
Fachbereich Informatik
Johann Wolfgang Goethe-Universität
Postfach 11 19 32
D-60054 Frankfurt
Germany
E-mail:schauss@ki.informatik.uni-frankfurt.de

22. Dezember 1997

1 Aussagenlogik (propositional logic)

Eine *Aussage* ist ein (schrift-)sprachliches Gebilde, von dem es sinnvoll ist zu sagen, es sei wahr oder falsch.

Beispiele.

- $0 = 1$
- "es regnet"
- "Es ist jetzt 8:15"

Die obige "Definition" ist problematisch, wie die Antinomie (das Paradoxon) des Lügners zeigt: "Die Aussage dieses Satzes ist falsch".

Die klassische formale Aussagenlogik geht daher von der Vorstellung aus, daß es atomare Aussagen gibt, die wahr oder falsch sein können, und daß komplexere Aussagen aus Junktoren und solchen atomaren Aussagen zusammengesetzt sind. Die Antinomie des Lügners wird dadurch umschifft, daß es Regeln gibt für die Wahrheit bzw. Falschheit von Aussagen, die nur abhängig sind vom Wert der Einzelaussagen (atomaren Aussagen). Für die Wahrheit/Falschheit von Aussagen ist es nicht notwendig, daß man die Wahrheit oder Falschheit von (z.B. atomaren Aussagen) auch leicht bestimmen kann: z.B. "Jede ungerade Zahl, die größer als drei ist, ist Summe von zwei Primzahlen". Diese Aussage (Goldbachsche Vermutung) ist entweder wahr oder falsch, aber zur Zeit kennt man deren Wahrheitswert nicht.

Die klassische Aussagenlogik geht von zwei Wahrheitswerten aus, T und F (T für True, wahr; F für falsch, und von Junktoren $\wedge, \vee, \neg, \Leftrightarrow, \Rightarrow$ für *und, oder, nicht, äquivalent, impli-*

ziert. Hierbei ist $\neg$ einstellig, während alle anderen Junktoren (Operatoren) zweistellig sind. Eine Grammatik zur (syntaktischen) Erzeugung von Aussagen ist:

$$\begin{array}{rcl} \mathcal{F} & ::= & \mathcal{A} \mid \mathcal{F} \mid (\neg\mathcal{F}) \mid (\mathcal{F} \wedge \mathcal{F}) \mid \\ & & (\mathcal{F} \vee \mathcal{F}) \mid (\mathcal{F} \Rightarrow \mathcal{F}) \mid (\mathcal{F} \Leftrightarrow \mathcal{F}) \end{array}$$

Hierbei bezeichnet $\mathcal{A}$ ein Atom und $\mathcal{F}$ eine bereits gebildete Aussage.

Die Berechnung des Wahrheitswertes von zusammengesetzten Aussagen wird mittels *Wahrheitswerttabellen* definiert:

A	$\neg A$
T	F
F	T

A	B	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

Zum Beispiel:

“7 ist nicht prim $\wedge$ die Goldbachsche Vermutung“ entspricht ($\neg$(7 ist prim)) $\wedge$ (die Goldbachsche Vermutung stimmt).

Wertet man das entsprechend der Wahrheitstafel aus, so ergibt sich $(\neg(T))\wedge$ “die Goldbachsche Vermutung stimmt“) = $(F\wedge$ “die Goldbachsche Vermutung stimmt“ = F

Definition 1. Eine *Aussageform (aussagenlogische Formel, Boolesche Form, propositional formula)* ist ein Ausdruck, in dem von atomaren Aussagen abstrahiert wird, und statt dessen Aussagenbezeichner (aussagenlogische Variablen) eingesetzt werden. Eine Grammatik zur syntaktischen Erzeugung von Aussageformen ist:

$$\begin{array}{rcl} \mathcal{F} & ::= & \mathcal{P} \mid (\neg\mathcal{F}) \mid (\mathcal{F} \wedge \mathcal{F}) \mid \\ & & (\mathcal{F} \vee \mathcal{F}) \mid (\mathcal{F} \Rightarrow \mathcal{F}) \mid (\mathcal{F} \Leftrightarrow \mathcal{F}) \end{array}$$

Hierbei bezeichnet $\mathcal{P}$ eine Variable und $\mathcal{F}$ eine bereits gebildete Aussage.

Beispiele für Aussageformen sind:

Beispiel 1.

1. $A, B, \ldots$ Aussagenvariablen

2. $(A \vee (\neg A))$

3. $(A \Leftrightarrow (\neg(\neg A)))$

Im folgenden werden wir Aussageformen auch kurz als Aussagen bezeichnen.

Die formale Definition erfordert mehr Klammern als tatsächlich notwendig sind, deshalb läßt man zum Teil Klammern weg, wenn sie nicht für die Eindeutigkeit notwendig sind. Als Konvention nimmt man an, daß es folgende Prioritäten von Operatoren gibt: $\neg$ vor $\wedge$ vor $\vee$ vor $\Rightarrow$ vor $\Leftrightarrow$. Weiterhin läßt man im allgemeinen auch die Klammern bei assoziativen Operatoren wie $\vee$ und $\wedge$ weg und nimmt implizite Linksklammerung an. Zum Beispiel ist $A \Rightarrow A \vee B \vee \neg C$ die Aussage $A \Rightarrow ((A \vee B) \vee (\neg C))$. Siehe auch Paragraph 14 über Notationen.

Es gilt: Gegeben eine Aussage A mit den Aussagenvariablen $A_i, i = 1, \ldots, n$. Wenn man allen Aussagenvariablen einen der Werte T oder F zuordnet, so läßt sich der Wahrheitswert der so gebildeten Aussage ausrechnen.

Beispiel 2. Betrachte die Aussage $(\neg A \wedge B) \vee (A \wedge \neg B)$ mit den Aussagevariablen A, B. Die möglichen Belegungen sind in der Tabelle aufgelistet.

A	B	$\neg B$	$A \wedge \neg B$	$\neg A$	$\neg A \wedge B$	$(A \wedge \neg B) \vee (\neg A \wedge B)$
T	T	F	F	F	F	F
T	F	T	T	F	F	T
F	T	F	F	T	T	T
F	F	T	F	T	F	F

Die Aussage $(A \wedge \neg B) \vee (\neg A \wedge B)$ verhält sich wie ein auschließendes oder (exclusive or, XOR), das auf die zwei Variablen A, B angewendet wird.

Definition 2. Eine *Interpretation (Belegung)* einer Aussageform A ist eine Abbildung, die allen aussagenlogischen Variablen in A einen Wert in $\{T, F\}$ zuordnet. Eine Aussageform heißt *Tautologie* (bzw. *Widerspruch, unerfüllbar*), wenn sie bei jeder Belegung der Variablen mit Wahrheitswerten stets T (bzw. F) ergibt. Eine Aussageform A heißt *erfüllbar*, wenn es eine Belegung der Variablen mit Wahrheitswerten gibt, so daß die Auswertung der Aussageform T ergibt; eine solche Belegung heißt auch *Modell von A*. Zwei Aussageformen heißen *äquivalent*, wenn ihre Wahrheitstafeln gleich sind.

Beispiel 3.

1. $A \vee \neg A$ ist eine Tautologie.

2. $A \wedge \neg A$ ist ein Widerspruch.

3. $A \vee B$ ist weder eine Tautologie noch ein Widerspruch, aber erfüllbar Die Aussagen $A \vee B$ und $\neg(\neg A \wedge \neg B)$ sind äquivalent.

Satz 3. *Eine Aussage A ist eine Tautologie genau dann, wenn $\neg A$ unerfüllbar ist.*

Satz 4. *Zwei Aussagen A, B sind äquivalent genau dann wenn $A \Leftrightarrow B$ eine Tautologie ist.*

Damit gilt auch: A ist eine Tautologie gdw. A äquivalent zu T ist. A ist ein Widerspruch gdw. A äquivalent zu F ist.

Wir wollen im folgenden Variablen in Aussagen nicht nur durch die Wahrheitswerte T, F ersetzen, sondern auch durch Aussageformen. Wir schreiben dann $A[B/x]$, wenn in der Aussage A die Aussagenvariable x durch die Aussage B ersetzt wird. In Erweiterung dieser Notation schreiben wir auch: $A[B_1/x_1, \ldots, B_n/x_n]$, wenn mehrere Aussagevariablen ersetzt werden sollen. Diese Einsetzung passiert gleichzeitig, so daß dies kompatibel zur Eigenschaft der Komposition als Funktionen ist: Wenn A n-stellige Funktion in den Aussagevariablen ist, dann ist $A[B_1/x_1, \ldots, B_n/x_n]$ die gleiche Funktion wie $A \circ (B_1, \ldots, B_n)$.

Satz 5. *Sind A_1, A_2 äquivalente Aussagen, und $B_1, \ldots, B_n$ weitere Aussagen, dann sind $A_1[B_1/x_1, \ldots, B_n/x_n]$ und $A_2[B_1/x_1, \ldots, B_n/x_n]$ ebenfalls äquivalent.*

Beweis. Man muß zeigen, daß jede Zeile der Wahrheitstabelle für die neuen Ausdrücke gleich sind. Seien $y_1, \ldots, y_m$ die Variablen. Den Wert einer Zeile kann man berechnen, indem man zunächst die Wahrheitswerte für B_i berechnet und dann in der Wahrheitstabelle von A_i nachschaut. Offenbar erhält man für beide Ausdrücke jeweils denselben Wahrheitswert. □

Man kann auch T und F selbst als Konstanten in Aussageformen zulassen. Wir werden diese im folgenden zulassen, und so tun als wären die Konstanten und der entsprechende Wert identisch. Dies erweitert nicht die Fähigkeit zum Hinschreiben von logischen Ausdrücken, denn man könnte T als Abkürzung von $A \vee \neg A$, und F als Abkürzung von $A \wedge \neg A$ auffassen. Die Junktoren $\wedge$ und $\vee$ sind kommutativ, assoziativ, und idempotent, d.h. es gilt:

$$\begin{array}{ccc}
\mathcal{F} \wedge \mathcal{G} & \Leftrightarrow & \mathcal{G} \wedge \mathcal{F} \\
\mathcal{F} \wedge \mathcal{F} & \Leftrightarrow & \mathcal{F} \\
\mathcal{F} \wedge (\mathcal{G} \wedge \mathcal{H}) & \Leftrightarrow & (\mathcal{F} \wedge \mathcal{G}) \wedge \mathcal{H} \\
\mathcal{F} \vee \mathcal{G} & \Leftrightarrow & \mathcal{G} \vee \mathcal{F} \\
\mathcal{F} \vee (\mathcal{G} \vee \mathcal{H}) & \Leftrightarrow & (\mathcal{F} \vee \mathcal{G}) \vee \mathcal{H} \\
\mathcal{F} \vee \mathcal{F} & \Leftrightarrow & \mathcal{F}
\end{array}$$

Weiterhin gibt es für Aussageformen noch Rechenregeln und Gesetze, die wir im folgenden auflisten wollen. Alle lassen sich mit Hilfe der Wahrheitstafeln beweisen, allerdings erweist sich das bei steigender Variablenanzahl als mühevoll, denn die Anzahl der Überprüfungen ist 2^n wenn n die Anzahl der Aussagenvariablen ist. Die Frage nach dem maximal nötigen Aufwand (in Abhängigkeit von der Größe der Aussageform) für die Bestimmung, ob eine Aussageform erfüllbar ist, ist ein berühmtes offenes Problem der (theoretischen) Informatik, das $SAT \in \mathcal{P}$-Problem, dessen Lösung weitreichende Konsequenzen hätte, z.B. würde $\mathcal{P} = \mathcal{NP}$ daraus folgen. Im Moment geht man davon aus, daß dies nicht gilt.

Lemma 6 Äquivalenzen:.

$$\begin{array}{lcll}
\neg(\neg A)) & \Leftrightarrow & A & \\
A \Rightarrow B & \Leftrightarrow & \neg A \vee B & \\
A \Leftrightarrow B & \Leftrightarrow & A \Rightarrow B \wedge B \Rightarrow A & \\
\neg(A \wedge B) & \Leftrightarrow & \neg A \vee \neg B & \textit{(DeMorgansche Gesetze)} \\
\neg(A \vee B) & \Leftrightarrow & \neg A \wedge \neg B & \\
A \wedge (B \vee C) & \Leftrightarrow & (A \wedge B) \vee (A \wedge C) & \textit{Distributivität} \\
A \vee (B \wedge C) & \Leftrightarrow & (A \vee B) \wedge (A \vee C) & \textit{Distributivität} \\
(A \Rightarrow B) & \Leftrightarrow & (\neg B \Rightarrow \neg A) & \textit{Kontraposition} \\
A \vee (A \wedge B) & \Leftrightarrow & A & \textit{Absorption} \\
A \wedge (A \vee B) & \Leftrightarrow & A & \textit{Absorption}
\end{array}$$

Diese Regeln sind nach Satz 5 nicht nur verwendbar, wenn $A; B; C$ Aussagevariablen sind, sondern auch, wenn A, B, C für Aussagen stehen.

Einige **Vereinfachungen**, wenn T und F als Konstante vorkommen. Beachte, daß wegen der Kommutativität hier auch die symmetrischen Regeln gelten.

Lemma 7.

$$\begin{array}{lcl}
A \wedge T & \Leftrightarrow & A \\
A \wedge F & \Leftrightarrow & F \\
A \vee T & \Leftrightarrow & T \\
A \vee F & \Leftrightarrow & A \\
\neg T & \Leftrightarrow & F \\
\neg F & \Leftrightarrow & T
\end{array}$$

Vom logischen Standpunkt aus gesehen, sind Schlußregeln Anweisungen, wie man aus einer Menge von gegebenen Formeln, die man als wahr annimmt, oder als wahr eingesehen hat, weitere wahre Formeln herleiten kann. In der klassischen Aussagenlogik kann man Schlußregeln in direkte Beziehung zu Tautologien setzen.

Schlußregeln:

$$\begin{array}{lcll}
((A \Rightarrow B) \wedge A) & \Longrightarrow & B & \text{(modus ponens)} \\
((A \Rightarrow B) \wedge \neg B) & \Longrightarrow & \neg A & \text{(modus tollens)} \\
((A \Rightarrow B) \wedge (B \Rightarrow C)) & \Longrightarrow & (A \Rightarrow C) & \text{(modus barbara, Transitivität von } \Rightarrow)
\end{array}$$

Satz 8. *Jede Aussage kann durch eine äquivalente Aussage ersetzt werden, die nur noch die folgenden Mengen von Junktoren enthält.*

1. $\{\vee, \wedge, \neg\}$ *oder*

2. $\{\vee, \neg\}$ *oder*

3. $\{\wedge, \neg\}$ *oder*

4. $\{NAND\}$*, wobei* $NAND(x,y) := \neg(x \wedge y)$*, oder*

5. $\{NOR\}$*, wobei* $NOR(x,y) := \neg(x \vee y)$*.*

Beweis. 1. folgt aus den obigen Regeln.

2. $A \wedge B$ ist äquivalent zu $\neg(\neg A \vee \neg B)$. Damit kann man $\wedge$ simulieren. Die Behauptung folgt dann aus 1).

3. $A \vee B$ ist definierbar durch $\neg(\neg A \wedge \neg B)$

4. $\neg A$ ist definierbar durch NAND(A, A). $A \wedge B$ ist definierbar durch $\neg$(NAND(A, B))

5. ähnlich zu 4. □

Bemerkung. Aussagen können in verschiedene Normalformen überführen werden:

- *konjunktive Normalform (KNF).* Diese ist von der Form: $(L_{1,1} \vee L_{1,n_1}) \wedge \ldots \wedge (L_{m,1} \vee L_{m,n_m})$. Hierbei sind $L_{i,j}$ die *Literale*, d.h. Aussagenvariablen oder Negationen von Aussagenvariablen.

- *disjunktive Normalform (DNF).* Diese ist von der Form: $(L_{1,1} \wedge L_{1,n_1}) \vee \ldots \vee (L_{m,1} \wedge L_{m,n_m})$ Hierbei sind $L_{i,j}$ die *Literale*, d.h. Aussagenvariablen oder Negationen von Aussagenvariablen.

Die Erzeugung der Normalformen erfolgt durch folgende Transformation unter Verwendung der Regeln für Aussagen: 1) Elimination aller Operatoren, die verschieden sind von $\neg, \vee, \wedge$, 2) Negationszeichen nach innen schieben. 3) Anwenden der Distributivgesetze, je nach gewünschter Normalform. Beachte, daß die KNF und DNF nicht eindeutig sind, auch wenn man Kommutativität, Assoziativität und Idempotenz von $\vee, \wedge$ in Betracht zieht.

Nach Überführung einer Aussage in DNF kann man leicht überprüfen, ob sie einen Widerspruch darstellt: Dies gilt gdw. jede konjunktive Komponente eine Aussagenvariable sowohl negiert als auch nicht negiert enthält. Zur Überprüfung auf Tautologie ist die KNF geeignet: Eine KNF ist Tautologie gdw. in jeder disjunktiven Komponente eine Aussagenvariable existiert, die dort negiert und nicht negiert vorkommt. Allerdings ist diese Überprüfung nicht ganz kostenlos: Die Normalformen können sehr groß werden, auch schon für kleine Formeln.

Eine mechanisches, syntaktisches Verfahren, das aus Aussageformen neue herstellt und/oder Aussageformen umformt, nennen wir *Kalkül.* Ein Kalkül soll natürlich nicht beliebige Umformungen machen, sondern möglichst solche, bei denen der Wahrheitswert erhalten bleibt. Normalerweise ist es das Ziel eines Kalküls, Tautologien zu erkennen. Hierfür gibt es verschiedene Methoden. Eine haben wir schon kennengelernt: die Methode der Wahrheitstabellen. Eine andere Methode ist es, mit einer Aussageform zu starten, und diese solange nach festgelegten Regeln umzuformen und neue Aussageformen herzustellen, bis ein bestimmtes Ereignis eintritt.

Beispiel 4. Wir geben ein Beispiel für das Vorgehen des *Tableaukalküls* zum Erkennen, ob eine eingegebene Aussage eine Tautologie ist:

Ist $(p \Rightarrow (q \Rightarrow p)$ eine Tautologie?

Idee: Teste ob diese Aussage falsch sein kann. Annahme: $p \Rightarrow (q \Rightarrow p) = F$,

dies erfordert: $p = T$, $q \Rightarrow p = F$,

dies wiederum erfordert $p = T, q = T, p = F$. Dies ergibt einen Widerspruch, da p nicht wahr und gleichzeitig falsch sein kann.

Dieser Kalkül enthält Regeln, wie jede syntaktische Form einer Aussageform zu zerlegen und daraus ein "Tableau" zu erstellen ist. z.B. für $A \wedge B = T$ kann $A = T$ und $B = T$ angenommen werden. Für $A \wedge B = F$ muß eine Fallunterscheidung gemacht werden. Ein Fall ist $A = F$, der andere ist $B = F$. Der Tableaukalkül liefert bei Eingabe einer Formel A entweder eine erfüllende Belegung für A = F) (also ein Modell für $\neg A$) oder er findet einen Konflikt bei den Zuweisungen und hat damit die Tautologie-Eigenschaft nachgewiesen.

Eine ganz andere, für unsere Zwecke zunächst nicht so interessante Methode ist die der *Hilbertkalküle*, die mit einer festen Menge von (einfachen) Tautologien (den Axiomen) starten, und daraus weitere Tautologien herstellen mittels Ersetzung von Aussagevariablen durch Formeln (Einsetzung) oder mittels modus ponens. Ziel ist die Herstellung aller Tautologien. Der modus ponens wird folgendermaßen verwendet: Wenn man schon die folgenden zwei Aussagen abgeleitet hat: $\mathcal{F}$, $\mathcal{F} \Rightarrow \mathcal{G}$, dann kann man damit auch $\mathcal{G}$ ableiten. Hier wird das Prinzip der Einsetzung verwendet: wenn $\mathcal{F}$ eine Tautologie ist, und $A_1, \ldots, A_n$ die Aussagenvariablen in $\mathcal{F}$, dann ist auch $\mathcal{F}(\mathcal{G}_1, \ldots, \mathcal{G}_n)$ eine Tautologie für beliebige Aussageformen $\mathcal{G}_i$.

Jede Aussageform mit n Aussagevariablen ist eine Funktion $\{T, F\}^n \rightarrow \{T, F\}$, wobei die Junktoren spezielle Funktionen von $\{T, F\}^2 \rightarrow \{T, F\}$ und $\neg$ eine Funktion $\{T, F\} \rightarrow \{T, F\}$ ist. Jede solche Funktion kann man durch ein Aussageform darstellen, d.h., es gilt die funktionale Vollständigkeit:

Satz 9. *Jede Funktion von $\{T, F\}^n \rightarrow \{T, F\}$ mit $n \geq 1$ läßt sich durch eine Aussageform, die höchstens n Aussagenvariablen enthält, darstellen .*

Beweis. Wir können die Funktion IF_THEN_ELSE(x, y, z) darstellen durch die Aussageform $(x \Rightarrow y) \wedge (\neg x \Rightarrow z)$. Wenn x den Wert T hat, dann hat $(x \Rightarrow y) \wedge (\neg x \Rightarrow z)$ denselben Wert wie y, und wenn x den Wert F hat, dann hat $(x \Rightarrow y) \wedge (\neg x \Rightarrow z)$ denselben Wert wie z. Sei f eine beliebige Funktion $f : \{T, F\}^n \rightarrow \{T, F\}$. Wir benutzen Induktion nach der Anzahl der Argumente. Wenn $n = 1$ ist, dann gibt es genau 4 Funktionen von $T, F \rightarrow T, F$. Diese können durch die Ausdrücke $x, \neg x, x \vee \neg x, x \wedge \neg x$ dargestellt werden, wie man durch Ausprobieren feststellt. Wenn $n > 1$ ist, dann definiere $f_1(x_2, \ldots, x_n) := f(T, x_2, \ldots, x_n)$ und $f_2(x_2, \ldots, x_n) := f(F, x_2, \ldots, x_n)$. Nach Induktionsvoraussetzung gibt es Aussageformen A_1 und A_2, die $f_1(x_2, \ldots, x_n)$ und $f_2(x_2, \ldots, x_n)$

repräsentieren. Dann ist IF_THEN_ELSE(x_1, A_1, A_2) eine Aussageform, die äquivalent zu $f(x_1, \ldots, x_n)$ ist. □

Die Menge der Aussageformen ist eine *Boolesche Algebra.*

Definition 10. Eine Boolesche Algebra ist folgendermaßen definiert: In einer Menge $\mathcal{B} \neq \emptyset$ gibt es Elemente T und F, weiterhin eine einstellige Operation $\neg$ und zwei 2-stellige Operationen $\wedge$ und $\vee$, so daß die Äquivalenzen in 6und 7 erfüllt sind, wobei A, B, C beliebige Elemente der Menge $\mathcal{B}$ sind.

Die einfachste Boolesche Algebra ist die Menge $\{T, F\}$ mit den Operationen $\wedge$, $\vee$, und $\neg$.

Übungsaufgabe 11.

Beweise einige Gesetze mit Hilfe der Wahrheitstafeln

Zeige, daß sich ein Absorptiongesetz aus dem anderen herleiten läßt mit Hilfe der anderen Gesetze

Welche Aussageform entspricht der Funktion $F(A, B, C)$, so daß F genau dann wahr ist wenn mindestens 2 Variablen wahr sind.

Übungsaufgabe 12. 1. Schreibe eine Funktion in Gofer, die Aussageformen (mit Operatoren $\wedge, \vee, \neg$) vereinfacht, indem alle T und F entsprechend den Äquivalenzen entfernt werden.

2. Schreibe eine Funktion in Gofer, mit zwei Eingaben:

   (a) eine Aussageform als Datenstruktur, und

   (b) eine Liste der Belegungen für alle Aussagenvariablen, die den Wahrheitswert der Aussage als Ausgabe liefert.

2 Prädikatenlogik

Die Aussagenlogik abstrahiert nur die Verwendung der Junktoren und die Wahrheitswerte und erfasst somit nicht alle logischen Schlußweisen. Aussagen über Mengen von Individuen wie "alle Menschen sind sterblich" oder "es gibt eine Primzahl ≤ 3" werden erst in der Prädikatenlogik auf formale Art und Weise behandelt. Hier wollen wir nur die Schreibweisen der Prädikatenlogik erster Stufe (PL1) einführen, um sie im folgenden zu verwenden. Die folgenden Aussagen lassen sich in PL1 formulieren:

- Alle Menschen sind sterblich
- Jede Primzahl ≥ 3 ist ungerade

- Für jede Eigenschaft E: Wenn alle Menschen die Eigenschaft E haben, dann gibt es auch einen Menschen mit der Eigenschaft E.

Eigenschaften werden durch *Prädikate* ausgedrückt, zum Beispiel: "x ist Primzahl" kann als PRIMZAHL(x) geschrieben werden. Prädikate haben eine feste Stelligkeit. Wenn D die Menge der betrachteten Objekte ist, dann kann man ein n-stelliges Prädikat als Funktion von $D^n \to \{T, F\}$ betrachten. Zum Beispiel ist $<$ ein zweistelliges Prädikat auf den ganzen Zahlen. Die syntaktische Entsprechung von Funktionen oder Operatoren sind die *Funktionssymbole*. Hier ist ebenfalls die Stelligkeit festgelegt. *Prädikatenlogische Terme* sind dann Variablen, oder Konstantensymbole (0-stellige Funktionssymbole) oder Ausdrücke der Form $f(s_1, \ldots, s_n)$, wobei f ein n-stelliges Funktionssymbol und $s_1, \ldots, s_n$ prädikatenlogische Terme sind. Der besseren Lesbarkeit halber schreibt man solche Terme manchmal auch in Infixschreibweise: $x * y$ statt $*(x, y)$.

Die All- und Existenzaussagen werden mit Hilfe der Quantoren $\forall, \exists$ (Allquantor und Existenzquantor) ausgedrückt: "Jede Primzahl ≥ 3 ist ungerade" kann man hinschreiben als:

$$\forall x : \mathrm{PRIM}(x) \wedge x \geq 3 \Rightarrow \mathrm{UNGERADE}(x),$$

wobei man PRIM und UNGERADE noch zu definieren hat, z.B. als

$$\forall x : \mathrm{UNGERADE}(x) \Leftrightarrow \neg(\exists y : x = 2 * y)$$
$$\forall x : \mathrm{PRIM}(x) \Leftrightarrow (\forall y(\forall z : x = y * z \Rightarrow (y = 1 \vee z = 1)))$$

Prädikatenlogische Atome sind von der Form $Q(t_1, \ldots, t_n)$,wobei

- Q ein n-stelliges Prädikat ist und,
- $t_1, \ldots, t_n$ prädikatenlogische Terme sind.

Prädikatenlogische Formeln sind

- Atome, oder
- $(\neg\mathcal{F}), (\mathcal{F} \vee \mathcal{G}), (\mathcal{F} \wedge \mathcal{G}), (\mathcal{F} \Leftrightarrow \mathcal{G}), (\mathcal{F} \Rightarrow \mathcal{G})$ für Formeln $\mathcal{F}, \mathcal{G}$ oder
- $\forall x : \mathcal{F}, \exists x : \mathcal{F}$ für eine Formel $\mathcal{F}$. (Die Zeichen $\forall, \exists$ nennt man Quantoren).

Eigentlich bräuchte man nur einen Quantor, denn es gilt:

$\forall x : \mathcal{F}$ ist äquivalent zu $\neg(\exists x : \neg\mathcal{F})$ für alle Formeln $\mathcal{F}$.

Eine Variable x ist eine *freie Variable* in der Formel $\mathcal{F}$, wenn x nicht im Bindungsbereich eines Quantors ist, eine Variable x ist eine *gebundene Variable* in der Formel $\mathcal{F}$, wenn x im Bindungsbereich eines Quantors ist. Eine Formel ohne freie Variablen heißt auch *geschlossene Formel*. Man beachte, daß die Variablen in der Prädikatenlogik *nicht* den Aussagenvariablen entsprechen.

Eine formal exakte Definition kann man wie im Paragraph über λ-Kalkül machen. Man unterscheidet Prädikatenlogiken verschiedener Stufen, wobei diese Unterscheidung nur danach geht, welche Bedeutung die Variablen haben, die von einem Quantor gebunden sind. D.h. ob sie Individuenvariablen sind, oder Prädikate bezeichnen.

Bei der *Prädikatenlogik erster Stufe (PL1)* darf nur über Individuenvariablen quantifiziert werden. Übersetzt man die Aussage "Für jede Eigenschaft E: Wenn alle Menschen die Eigenschaft E haben, dann gibt es auch einen Menschen mit der Eigenschaft E" , so erhält man $\forall E : ((\forall x : \text{Mensch}(x) \Rightarrow E(x)) \Rightarrow ((\exists x : \text{Mensch}(x) \wedge E(x))$. Dies ist eine Formel der Prädikatenlogik zweiter Stufe. Ebenso ist das Induktionsaxiom nur in der Prädikatenlogik zweiter Stufe formulierbar. Ein weiteres Beispiel ist die Axiomatisierung der reellen Zahlen, wo das sogenannte Supremumsaxiom eine Formel zweiter Stufe ist, da über Teilmengen quantifiziert wird:

$(\forall M : (M \subseteq R \wedge M \neq \emptyset \wedge UB(M) \neq \emptyset) \Rightarrow \exists r : SUP(M) = r)$. Normalerweise gilt, daß Formeln der ersten Stufe "normale Prädikatenlogik" sind, während Formeln zweiter Stufe eher logischen Regeln und Metaregeln entsprechen. Für Formeln dritter und höherer Stufe gibt es kaum anschauliche Anwendungen.

Die Definition einer Tautologie, d.h. eines Satzes der Prädikatenlogik erster Stufe machen wir im folgenden. Beachte dazu den Paragraphen über interpretierte Systeme.

Definition 13. Eine *Interpretation* I wird folgendermaßen definiert. Als Resultatmenge nimmt man eine Menge D, und definiert I folgendermaßen: Für jedes Funktionssymbol f der Stelligkeit n gibt es eine Abbildung $I(f) : D^n \to D$, Für jedes Prädikatensymbol P der Stelligkeit n gibt es eine Funktion $I(P) : D^n \to \{T, F\}$. (Äquivalent dazu ist die Methode, stattdesssen eine n-stellige Relation zu nehmen.) Weiterhin ordnet I jeder Variablen einen Wert in D zu. Eine neue Interpretation kann man durch Abändern des Wertes auf einer Variablen definieren. Indem man den Wert auf x zu d macht, und alles andere läßt. Man erhält eine neue Interpretation, die mit $I[d/x]$ bezeichnet wird. Dann definiert man I induktiv:

$$
\begin{array}{lcll}
I(f(s_1, \ldots, s_n)) & := & I(f)(I(s_1), \ldots, I(s_n)) & \text{(ergibt Werte aus } D\text{)} \\
I(P(s_1, \ldots, s_n)) & := & I(P)(I(s_1), \ldots, I(s_n)) & \text{(ergibt } T \text{ oder } F\text{)} \\
I(\mathcal{F} \wedge \mathcal{G}) & := & I(\mathcal{F}) \wedge I(\mathcal{G}) & \\
I(\mathcal{F} \vee \mathcal{G}\ ) & := & I(\mathcal{F}) \vee I(\mathcal{G}) & \\
I(\neg \mathcal{F}) & := & \neg I(\mathcal{F}) & \\
.\ I(\forall x.\mathcal{F}) & := & T, & \text{wenn für alle } d \in D : I[d/x](\mathcal{F}) = T \\
 & & F & \text{anderenfalls.} \\
I(\exists x.\mathcal{F}) & := & T, & \text{wenn es ein } d \in D \text{ gibt mit } I[d/x](\mathcal{F}) = T \\
 & & F & \text{anderenfalls.}
\end{array}
$$

Eine *Tautologie* ist eine geschlossene Formel, die von jeder Interpretation wahr gemacht wird, d.h., die immer auf T abgebildet wird.

Dies ist die Semantik der Prädikatenlogik. Wie können jetzt auch sagen, was die wahren Sätze sind:

Definition 14. Eine *Tautologie* ist eine Formel ohne freie Variablen, die von jeder Interpretation zu wahr evaluiert wird. Eine Formel $\mathcal{F}$ ist *erfüllbar*, wenn es eine Interpretation I gibt, so daß $I(\mathcal{F})$ wahr wird. Eine solche Interpretation heißt auch *Modell* der Formel. Eine Formel $\mathcal{G}$, die kein Modell hat, nennt man auch *unerfüllbar*.

Auch in der Prädikatenlogik erster Stufe gilt:

Satz 15. *Eine Formel $\mathcal{F}$ ist eine Tautologie genau dann wenn $\neg\mathcal{F}$ unerfüllbar ist.*

Alle Transformationen auf der syntaktischen Ebene müssen, wenn sie korrekt sein sollen, diese Semantik erhalten. D.h. für eine Transformation $\mathcal{G}_1 \to \mathcal{G}_2$ muß gewährleistet sein, daß $I(\mathcal{G}_1) = I(\mathcal{G}_2)$ für alle Interpretationen I gilt. Die erweiterte Ausdruckskraft der Prädikatenlogik gegenüber der Aussagenlogik hat einen Preis: die Überprüfung (anhand der Definition 14, ob eine Aussage eine Tautologie ist, erfordert die Überprüfung von unendlich vielen verschiedenen Interpretationen. D.h. für diesen Zweck ist diese Semantik ungeeignet. Allerdings kann man mit dieser Semantik die Eigenschaften von Transformationen nachweisen.

Es gibt für die Prädikatenlogik Kalküle, die feststellen können, ob eine Aussage eine Tautologie ist. d.h., die für jede Tautologie nach endlich langer Zeit dies auch feststellen. Allerdings kann es passieren, daß diese Kalküle unendlich lange weiterlaufen, falls die zu testende Formel keine Tautologie ist.

3 Mengen

Von G. Cantor stammt die "Definition" dafür, was eine Menge sein soll:

> "Eine Menge ist eine Zusammenfassung von bestimmten wohlunterschiedenen Objekten unserer Anschauung oder unseres Denkens zu einem Ganzen. Die Objekte heißen Elemente der Menge." (G. Cantor).

Man beachte, daß dies keine echte Definition ist, da die vorkommenden Begriffe undefiniert sind. Trotzdem ist dies eine gute intuititve Erklärung des Mengenbegriffs. Wir gehen also davon aus, daß wir wissen was Mengen sein sollen, und führen folgende Schreibweise ein: Wenn x ein Element der Menge M ist, so schreiben wir $x \in M$, und wenn x kein Element von M ist, so schreiben wir $x \notin M$.

Im allgemeinen genügt die sogenannte "naive Mengenlehre", in der die Mengenbildung nicht weiter eingeschränkt ist, d.h. man erlaubt das naive Mengenbildungsprinzip:

$\forall E : \exists M : \forall x : (x \in M \Leftrightarrow E(x))$ (Jede Eigenschaft definiert eine Menge.)

Hierbei ist E stellvertretend für eine (fast) beliebige prädikatenlogische Formel mit der freien Variablen x.

Diese Menge wird auch geschrieben als $\{x \mid E(x)\}$. Bekanntermaßen führt dieses Mengenbildungsprinzip im Extremfall auf Widersprüche: (Russelsche Antinomie): wenn man die folgende Menge bildet: $R := \{x | x \notin x\}$ so führt dies zu einem direkten Widerspruch, denn es würde gelten: $R \in R \Leftrightarrow R \notin R$.

Es gibt verschiedene axiomatische Ansätze, die diese Widersprüche vermeiden, jedoch würde uns das hier zu weit führen. Eine einfache Methode, die diese Widersprüche erfolgreich verhindert, ist die Unterscheidung zwischen Mengen und *Klassen*, wobei man bestimmte Klassen als Mengen auszeichnet. Das Mengenbildungsaxiom wird zu einem Klassenbildungsaxiom und wird leicht geändert: $\{x \mid E(x)$ und x ist Menge$\}$. Damit kann man dann die Klasse aller Mengen bilden, die dann allerdings keine Menge mehr ist. Auch kann man dann gefahrlos die Klasse aller Programm, Halbgruppen, usw. verwenden. Diese Methode benötigt zusätzlich noch einige Axiome, die definieren, welche Klassen Mengen sind.

Es gilt das *Extensionalitätsprinzip*:

$$\forall M : \forall N : ((\forall x : x \in M \Leftrightarrow x \in N) \Rightarrow M = N)$$

D.h.: "Mengen mit den gleichen Elementen sind gleich"

Beispiel 5. Beispiele für Mengenbildung:

Durchschnitt von Mengen:	$A \cap B := \{x \mid x \in A \wedge x \in B\}$
Vereinigung von Mengen:	$A \cup B := \{x \mid x \in A \vee x \in B\}$
Differenz von Mengen:	$A \setminus B := \{x \mid x \in A \wedge x \notin B\}$
leere Menge:	$\emptyset := \{x \mid x \neq x\}$

$$\bigcap_{i \in I} A_i := \{x | \forall i : i \in I \Rightarrow x \in A_i\}$$
$$\bigcup_{i \in I} A_i := \{x | \exists i : i \in I \wedge x \in A_i\}$$

Für die Vereinigung (bzw. Durchschnitt) von mehreren Mengen schreiben wir auch: $\bigcup\{A_i \mid i \in I\}$ bzw. $\bigcap\{A_i \mid i \in I\}$

Definition 16. Seien A, B Mengen.

1. A ist *Teilmenge* von B (Schreibweise $A \subseteq B$), genau dann wenn $\forall x : x \in A \Rightarrow x \in B$.
2. A ist *echte Teilmenge* von B (Schreibweise $A \subset B$), genau dann wenn $A \subseteq B \wedge A \neq B$
3. *Potenzmenge* von A (Schreibweise $\mathcal{P}(A)$, 2^A), $2^A := \{B | B \subseteq A\}$ (Menge aller Teilmengen)
4. zwei Mengen A und B heißen *disjunkt*, wenn $A \cap B = \emptyset$. $A_1, \ldots, A_n$ heißt Zerlegung von A, wenn die A_i paarweise disjunkt sind und $\bigcup\{A_i \mid i = 1, \ldots, n\} = A$.

5. Die *Mächtigkeit einer Menge* ist die Anzahl der Elemente. Bezeichnung $|A|$, $\#A$, oder $\mathrm{card}(A)$.

Standardbezeichnungen für Mengen:

$\mathbb{N}$	Menge der natürlichen Zahlen (ohne 0)
$\mathbb{N}_0$	Menge der natürlichen Zahlen mit 0.
$\mathbb{Z}$	Menge der ganzen Zahlen.
$\mathbb{Q}$	Menge der rationalen Zahlen.
$\mathbb{R}$	Menge der reellen Zahlen.
$\mathbb{C}$	Menge der komplexen Zahlen.

Definition 17. Seien A, B Mengen. Das *kartesische Produkt* von A und B ist definiert durch

- $A \times B := \{(x, y) \mid x \in A, y \in B\}$, wobei (x, y) das geordnete Paar aus den Elementen x, y bezeichnet.
- $A_1 \times \ldots \times A_n := \{(x_1, \ldots, x_n) | x_i \in A_i, i = 1, \ldots, n\}$
- $A^n := \underbrace{A \times \ldots \times A}_{n}$

Ein Element $(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n$ nennt man auch (geordnetes) n-Tupel. Dies kann man auch als Liste von n Elementen ansehen.

Beispiel: $\{1, 2\}^3 = \{(1, 1, 1), (1, 1, 2), (1, 2, 1), (1, 2, 2), (2, 1, 1), (2, 1, 2), (2, 2, 1), (2, 2, 2)\}$

4 Relationen und Funktionen

Definition 18. Seien A, B Mengen. Eine Teilmenge $R \subseteq A \times B$ heißt (zweistellige) *Relation*. Für Mengen $A_1, \ldots, A_n$ heißt eine Teilmenge $R \subseteq A_1 \times \ldots \times A_n$ (n-stellige) Relation.

Schreibweise für zweistellige Relationen: $(x, y) \in R$ oder xRy oder $R(x, y)$. Beachte, daß Informatiker sehr erfinderisch sind in der Bezeichnung von Relationen, sie kommen in vielen Notationen vor. z.B. $\xrightarrow[G,r]{*}$ kann eine Notation für eine Relation sein.

Definition 19. Eine Relation $R \subseteq A \times B$ heißt

linkstotal (serial)	$\Leftrightarrow$	$\forall x \in A : \exists y \in B : (x, y) \in R$
rechtstotal	$\Leftrightarrow$	$\forall y \in B : \exists x \in A : (x, y) \in R$
rechtseindeutig	$\Leftrightarrow$	$\forall x \in A : \forall y \in B : \forall z \in B : (x, y) \in R \wedge (x, z) \in R \Rightarrow y = z$
linkseindeutig	$\Leftrightarrow$	$\forall x \in A : \forall y \in A : \forall z \in B : (x, z) \in R \wedge (y, z) \in R \Rightarrow x = y$
eineindeutig	$\Leftrightarrow$	rechtseindeutig und linkseindeutig

Definition 20.

Eine rechtseindeutige Relation $R \subseteq X \times Y$ heißt *Funktion (partielle Funktion) von* X *nach* Y.

Eine rechtseindeutige und linkstotale Relation $R \subseteq X \times Y$ heißt *total definierte Funktion.*

Eine total definierte, rechtstotale Funktion $R \subseteq X \times Y$ heißt *Surjektion (surjektive Funktion) von* X *auf* Y.

Eine total definierte, linkseindeutige Funktion $R \subseteq X \times Y$ heißt *Injektion (injektive Funktion) von* X *nach* Y.

Ist $R \subseteq X \times Y$ eine Injektion und Surjektion, so heißt R *Bijektion (bijektive Funktion) von* X *auf* Y.

Ist $R \subseteq X \times Y$ eine Funktion, so daß $\forall x, y, u, v : (x, u) \in R \wedge (y, v) \in R \Rightarrow u = v$, so heißt R *konstante* Funktion.

Schreibweise für Funktionen: $R : X \to Y$, oder $X \underset{R}{\longrightarrow} Y$ Bezeichnung: $ABB(X,Y) := \{R \mid R : X \to Y \text{ ist Funktion}\}$.

Definition 21. Sei $R \subseteq X \times Y$ eine Relation. Dann definieren wir:

X heißt der *Argumentbereich*,

Y heißt der *Wertebereich*,

$DEF_R := \{x \in X \mid \exists y \in Y : (x, y) \in R\}$ ist der *Definitionsbereich (domain)* von R.

$BIL_R := \{y \in Y \mid \exists x \in X : (x, y) \in R\}$ ist der *Bildbereich (co-domain, range)* von R.

Bemerkung: Wenn $R : X \to Y$ total definierte Funktion ist, dann gilt $X = \mathrm{DEF}_R$. Wenn $R : X \to Y$ rechtstotale Funktion ist, dann gilt $Y = \mathrm{BIL}_R$. Insbesondere gilt dies für Surjektionen.

Definition 22. Seien $R \subseteq Y \times Z$ und $S \subseteq X \times Y$ Relationen, dann ist das *Kompositum* $R \circ S$ definiert als $R \circ S := \{(x, z) \mid y \in Y, (x, y) \in S \wedge (y, z) \in R\}$.

Bemerkung: Sind R und S Funktionen, so auch $R \circ S$. Die Anwendung einer Funktion R auf ein Element x schreibt man $R(x)$, wobei $R(x)$ das eindeutig bestimmte Element mit $(x, R(x)) \in R$ ist. Für die Anwendung der Komposition $R \circ S$ gilt: $(R \circ S)(x) = R(S(x))$. Beachte, daß die Komposition assoziativ ist, d.h. $R \circ (S \circ T) = (R \circ S) \circ T$. (Beachte, daß für die Anwendung verschiedene Konventionen gebräuchlich sind, z.B. wird in manchen Artikeln oder Büchern $(R \circ S)(x) := S(R(x))$ definiert.

Beispiel 6. Betrachte die Funktion $\texttt{quadrat}\colon \mathbb{N} \to \mathbb{N}$ mit $\texttt{quadrat} := \{(a,b) \mid a,b \in \mathbb{N} \wedge b = a*a\}$. Üblicherweise kann man dies dadurch beschreiben, daß man sagt: $\texttt{quadrat}(x) = x*x$ für alle $x \in \mathbb{N}$. Diese Funktion ist total definiert, aber nicht rechtstotal, da 3 nicht in $BIL_{\texttt{quadrat}}$. Die Funktion quadrat ist linkseindeutig, d.h. quadrat ist Injektion.

Übungsaufgabe 23. . Bestimme alle Funktionen $R : X \to X$ mit $X = 0,1$. Wieviel total definierte (Surjektionen, Injektionen, Bijektionen) gibt es?

Eine anschauliche Interpretation von Bijektion, Injektion und Surjektion ist die Zuordnung Mäntel $\to$ Kleiderhaken:

Bijektion:	An jedem Haken genau ein Mantel
Injektion:	An jedem Haken höchstens ein Mantel
Surjektion:	An jedem Haken mindestens ein Mantel.
Definitionsbereich:	Menge der aufgehängten Mäntel
Bildbereich:	Menge der besetzten Haken.

Definition 24. Ist $R \subseteq X \times Y$ eine Relation (Funktion), so heißt $R^{-1} := \{(y,x) \mid (x,y) \in R\}$ die *inverse Relation* (bzw. *inverse Funktion*).

Beispiel 7. $F : \{0,1,2\} \to \{0,1,2\}$ mit $F = \{(0,0),(1,0),(2,1)\}$ ist total definierte Funktion. Es ist $F^{-1} = \{(0,0),(0,1),(1,2)\}$. F^{-1} ist keine Funktion, da nicht rechtseindeutig.

Es gilt: Ist F injektiv, dann ist F^{-1} eine Funktion, ist F bijektiv, dann ist F^{-1} ist ebenfalls Bijektion

Definition 25. Sei $F : X \to Y$ eine Funktion, $A \subseteq X$ und $B \subseteq Y$.

1. Das *Bild von A unter F* ist definiert durch $F(A) := \{y \mid \exists x \in A : y = F(x)\}$.
2. Die *Restriktion von F auf A* ist definiert durch $F|_A := F \cap (A \times Y)$
3. Das *Urbild von B unter F* ist definiert durch $F^{-1}(B) := \{x \in X \mid \exists y \in B : Fx = y\}$.
4. Ist der Argumentbereich von F ein kartesisches Produkt aus n Mengen $X_1, \ldots, X_n$, so ist F eine n-stellige Funktion.

Beispiel 8. Betrachte die Funktion $\texttt{quadrat}\colon \mathbb{N} \to \mathbb{N}$. Sei $G :=$ Menge der geraden Zahlen. Dann ist $\texttt{quadrat}(G) = \{4, 16, 36, \ldots\}$, $\texttt{quadrat}|_G := \{(x,y) \in \texttt{quadrat} \mid x \in G\}$. Es ist $\texttt{quadrat}^{-1}(G) = \mathbb{N}$.

einstellige Funktionen:	$\texttt{quadrat} : \mathbb{N} \to \mathbb{N}$
	$- : \mathbb{R} \to \mathbb{R}$ mit $-(x) := -x$
	$\texttt{sin} : \mathbb{R} \to \mathbb{R}$
zweistellige Funktionen:	$+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
	$* : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$
n-stellige Funktionen:	$\lvert.\rvert : \mathbb{R}^n \to R$ mit $\lvert(x_1, \ldots, x_n)\rvert := \sqrt{\sum_{i=1}^n x_i^2}$
	(Abstand eines Punktes vom Ursprung in $\mathbb{R}^n$)

Definition 26. Eine Menge M heißt *unendlich*, wenn es eine Injektion $f : \mathbb{N} \to M$ gibt. M heißt *endlich*, wenn sie nicht unendlich ist. Eine Menge heißt *abzählbar unendlich*, wenn es eine Bijektion $f : \mathbb{N} \to M$ gibt. Eine Menge heißt *abzählbar*, wenn sie endlich ist oder *abzählbar unendlich*. Eine Menge heißt *überabzählbar unendlich*, wenn sie unendlich viele Elemente hat und es keine Bijektion $f : \mathbb{N} \to M$ gibt.

In der Informatik hat man es im allgemeinen mit abzählbaren Mengen zu tun. Zum Beispiel ist die Menge aller möglichen Programme einer bestimmten Programmiersprache i.a. abzählbar. Wir gehen deshalb etwas genauer darauf ein:

Satz 27. *Folgende Konstruktionen erzeugen abzählbare Mengen:*

1. *Wenn M eine abzählbare Menge ist und $f : M \to K$ eine surjektive Abbildung, dann ist K ebenfalls abzählbar.*
2. *Teilmengen von abzählbaren Mengen sind abzählbar*
3. *Wenn $M_i, i \in \mathbb{N}$ abzählbare Mengen sind, dann ist $\bigcup M_i$ ebenfalls abzählbar.*
4. *Wenn M eine abzählbare Menge ist, dann ist M^n ebenfalls abzählbar.*
5. *Wenn M eine abzählbare Menge ist, dann ist $\bigcup\{M^n \mid n > 0\}$ ebenfalls abzählbar.*

Beweis. (Skizze) 1.), 2.) und 3.) überlassen wir den Monographien über Mengenlehre. Zum Beweis von 4.) genügt es, $n = 2$ zu betrachten. Man sieht leicht, daß sich diese Menge als abzählbare Vereinigung von endlichen Mengen schreiben läßt, wobei $M_k := \{(a_i, a_j) \mid i + j \leq k\}$. □

Satz 28. $\mathcal{P}(\mathbb{N})$ *ist nicht abzählbar. Ebenso ist die Menge* $\{f \mid f : \mathbb{N} \to \mathbb{N}\}$ *nicht abzählbar.*

Beweis. (Diagonalisierung) Sei M_i eine Aufzählung der Mengen in $\mathcal{P}(\mathbb{N})$. Dann definiere $D := \{i \mid i \notin M_i\}$. Offenbar ist $D \in \mathcal{P}(\mathbb{N})$. Deshalb existiert ein j so daß $M_j = D$. Wenn $j \in M_j$, dann ist nach Definition von D: $j \notin D = M_j$. Wenn $j \notin M_j$, dann ist nach Definition von D: $j \in D = M_j$. In beiden Fällen erhalten wir einen Widerspruch.

Zu jeder Menge $M \in \mathcal{P}(\mathbb{N})$ gibt es eine Funktion f mit $f(\mathbb{N}) = M$, also gibt es mindestens soviele Funktionen wie Mengen in $\mathcal{P}(\mathbb{N})$. □

Definition 29. Eine *Relation in einer Menge* M ist eine Teilmenge $R \subseteq M^n$.

Bemerkung. Zweistellige Relationen in einer Menge lassen sich durch Pfeildiagramme (gerichtete Graphen) veranschaulichen. Die Knoten sind die Elemente. Gibt es ein $(x, y) \in R$, so gibt es einen Pfeil (gerichtete Kante) von x nach y.

Definition 30. Eine zweistellige Relation in einer Menge M heißt

$$
\begin{array}{ll}
\textit{reflexiv} & :\Leftrightarrow \forall x \in M : xRx. \\
\textit{transitiv} & :\Leftrightarrow \forall x, y, z \in M : xRy \wedge yRz \Rightarrow xRz. \\
\textit{symmetrisch} & :\Leftrightarrow \forall x, y \in M : xRy \Leftrightarrow yRx. \\
\textit{antisymmetrisch} & :\Leftrightarrow \forall x, y \in M : xRy \wedge yRx \Rightarrow x = y \\
\textit{asymmetrisch} & :\Leftrightarrow \forall x, y \in M : xRy \Rightarrow \neg(yRx) \\
\textit{irreflexiv} & :\Leftrightarrow \forall x \in M : \neg xRx \\
\textit{linear} & :\Leftrightarrow \forall x, y \in M : x \neq y \Rightarrow xRy \vee yRx
\end{array}
$$

Eine reflexive, transitive Relation heißt *Präordnung.*

Eine reflexive, transitive und antisymmetrische Relation R heißt *partielle Ordnung (reflexive Ordnung)* Schreibweise ist: (M, R).

Eine lineare reflexive Ordnung heißt *lineare Ordnung (totale Ordnung, Kette).*

Eine irreflexive transitive Relation heißt *irreflexive Ordnung.*

Eine antisymmetrische, transitive Relation heißt *Präzedenzordnung.*

Eine asymmetrische, transitive Relation heißt *asymmetrische Ordnung.*

Eine reflexive, symmetrische, transitive Relation heißt *Äquivalenzrelation.*

Beispiel 9. Beispiele für verschiedene Ordnungen sind:

Präordnung: Die Relation R auf Menge definiert durch $M \ R \ N$ gdw. $|M| \leq |N|$.

reflexive Ordnung: $\leq$ auf Zahlen , $\subseteq$ auf Mengen.

lineare Ordnung: $\leq$ auf Zahlen.

irreflexive Ordnungen: $<$ auf Zahlen , $\subset$ auf Mengen.

Präzedenzordnungen: $<, \leq$ auf Zahlen , $\subset, \subseteq$ auf Mengen.

asymmetrische Ordnungen: $<$ auf Zahlen , $\subset$ auf Mengen.

Äquivalenzrelationen: $=$ auf Zahlen , $=$ auf Mengen, R auf Mengen mit der Bedeutung: $M \ R \ N$, gdw. M und N gleiche Kardinalität haben.

Satz 31. *R ist irreflexive Ordnung genau dann wenn R eine asymmetrische Ordnung ist.*

Beweis. "$\Rightarrow$ ": Zu zeigen ist, daß irreflexiv $\Rightarrow$ asymmetrisch. Nehme an, dies sei falsch, dann existieren a, b mit aRb und bRa. Transitivität ergibt aRa, was ein Widerspruch zur Irreflexivität ist.
"$\Leftarrow$": Zu zeigen: asymmetrisch $\Rightarrow$ irreflexiv. Nehme an, dies sei falsch, dann existiert ein a mit aRa, ein Widerspruch zur Asymmetrie. □

Satz 32. *Reflexive Ordnungen und irreflexive Ordnungen sind gleichwertig.*

1. *sei R reflexive Ordnung. Definiere R^i mit $R^i := \{(x, y) \in R \mid x \neq y\}$. Dann ist R^i irreflexive Ordnung mit $\forall x, y : x \neq y \Rightarrow (xRy \Leftrightarrow xR^iy)$*

2. *Sei R irreflexive Ordnung: Definiere R^r mit $R^r := R \cup \{(x, x) \mid x \in M\}$. Dann ist R^r reflexive Ordnung mit $\forall x, y : x \neq y \Rightarrow (xRy \Leftrightarrow xR^ry)$*

Bemerkung. Mit den Bezeichnungen oben gilt:

1. $(R^i)^r = R$ wenn R reflexive Ordnung ist

2. $(R^r)^i = R$, wenn R irreflexive Ordnung ist.

Bemerkung. Zusammenfassend kann man sagen: wesentlich verschiedene Relationen sind: Präordnung, Ordnung, und Äquivalenzrelation. Bei Ordnungen kommt es nicht so sehr darauf an, ob Reflexivität gilt oder nicht, je nachdem was man gerade braucht, schreibt man $<$ für die irreflexive Version und $\leq$ für die reflexive Version.

Aus jeder Präordnung R auf M läßt sich eine Ordnung konstruieren, indem man zunächst die Äquivalenzrelation $R^=$ bildet mit $xR^=y$ gdw. $xRy \wedge yRx$. Danach definiert man die Ordnung auf den Äquivalenzklassen. Diese ist dann eine partielle Ordnung.

Übungsaufgabe 33.

1. Sei R linkstotale, symmetrische und transitive Relation. Dann ist R auch reflexiv.

2. Gilt das noch wenn R nicht linkstotal ist?

Definition 34. Sei $R \subseteq M \times M$ eine Relation in M.

1. Die *reflexive Hülle* R^r ist definiert als: $R^r := R \cup \{(x, x) \mid x \in M\}$.

2. Die *transitive Hülle* R^+ von R ist definiert durch:
   $(x, y) \in R^+ :\Leftrightarrow \exists n > 1, x_1, \ldots, x_n \in M : (x = x_1 \wedge y = x_n \wedge \forall i \in \{1, \ldots, n-1\} : (x_i, x_{i+1}) \in R$

3. Die *reflexiv transitive Hülle* R^* von R ist definiert als $R^* := R^r \cup R^+$.

Die transitive Hülle einer Relation R kann man sich veranschaulichen als die (gerichtete) Erreichbarkeitsrelation. Z.B. Aufrufhierarchie von Modulen. Wenn R die Relation ist, die zum direkten Aufruf gehört, d.h. $M_1\ R\ M_2$ gdw. M_1 M_2 direkt aufruft, dann ist R^+ die Relation "ruft indirekt auf". Diese Ordnung ist eine partielle Ordnung, wenn es keine Zyklen gibt (zyklischen Aufrufe), ansonsten eine Präordnung, die wieder eine partielle Ordnung auf den Äquivalenzklassen bildet.

In Programmen oder in der Textverarbeitung werden oft Abkürzungverzeichnisse (Standardtexte) verwendet. Hierbei kann eine Zeichenkette als Abkürzung definiert werden. Die Relation "verwendet direkt" hat als transitive Hülle die Relation "verwendet indirekt". Da man keine unendlich großen Zeichenketten definieren will, verbieten man Zyklen, d.h. die transitive Hülle ist eine partielle Relation. Analoges gilt für die Verwendung von "Include", das es erlaubt, andere Text-files an eine bestimmte Stellen (logisch) zu kopieren, unter gemeinsamer Verwendung derselben Textdatei.

Satz 35. *Sei $R \subseteq M \times M$ eine Relation in M. R^r ist der Durchschnitt aller reflexiven Relationen in M, die R enthalten. R^+ ist der Durchschnitt aller transitiven Relationen in M, die R enthalten.*

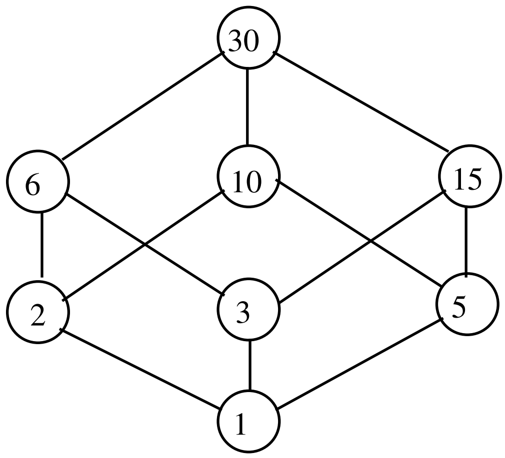
$$R^r := \bigcap\{S \mid R \subseteq S, S \subseteq M \times M, S \textit{ reflexiv}\}$$

$$R^+ := \bigcap\{S \mid R \subseteq S, S \subseteq M \times M, S \textit{ transitiv}\}$$

Ordnungen können durch Hasse-Diagramme veranschaulicht werden.

- Jedem Element entspricht ein Knoten,
- Wenn $a < b$, dann ist a in der Zeichnung weiter unten,
- Zwei Knoten a, b werden durch eine Kante verbunden, wenn kein Element dazwischen liegt.

Beispiel 10. $M = \{1, 2, 3, 5, 6, 10, 15, 30\}$, $R = \{(x, y) \mid x, y \in M$ und x teilt $y\}$.



Definition 36. Sei (M, R) eine reflexive Ordnung und sei $N \subseteq M$.

1. Ein Element $a \in N$ heißt *größtes Element von* N, wenn es größer als alle anderen Elemente von N ist : $(\forall x \in N : xRa)$. Ein größtes Element von M heißt *Einselement*.

2. Ein Element $a \in N$ heißt *kleinstes Element von* N, wenn es kleiner als alle anderen Elemente ist : $(\forall x \in N : aRx)$. Ein kleinstes Element von M heißt *Nullelement*.

3. Ein Element a heißt *maximal*, wenn es kein größeres Element gibt .

4. Ein Element a heißt *minimal*, wenn es kein kleineres Element gibt .

5. Die Menge der *oberen Schranken* (upper bounds) von N ist: $UB(N) := \{x \in M \mid \forall y \in N : yRx\}$.

6. Die Menge der *unteren Schranken* (lower bounds) von N ist: $LB(N) := \{x \in M \mid \forall y \in N : xRy\}$.

7. Die Menge der kleinsten Elemente in $UB(N)$ wird mit $LUB(N)$ bezeichnet (least upper bounds).

8. Die Menge der größten Elemente in $LB(N)$ wird mit $GLB(N)$ bezeichnet (greatest lower bounds).

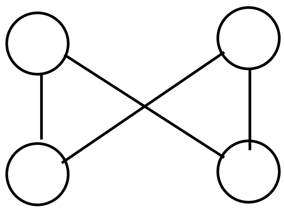
9. Wenn $GLB(N)$ und $LUB(N)$ für alle endlichen Teilmengen von N existieren und genau ein Element enthalten, so ist M ein *Verband*. Wenn $GLB(N)$ (bzw. $LUB(N)$) immer existiert und einelementig ist, aber $LUB(N)$ (bzw. $GLB(N)$) nicht immer, so spricht man auch von unteren (bzw. oberen) Halbverband.

Beispiel 11. Betrachte die Ordnung in Beispiel 10.

- 30 ist größtes Element (und Einselement)

- 1 ist kleinstes Element (und Nullelement)
- $6, 10$ und 15 sind maximale Elemente von $M \setminus \{30\}$.
- $2, 3$ und 5 sind minimale Elemente von $M \setminus \{1\}$.
- $UB(\{1, 2, 3, 5\}) = \{30\}$.
- $GLB(M \setminus \{1\}) = \{1\}$:
- M ist ein Verband
- $M \setminus \{1\}$ ist ein oberer Halbverband.

Im folgenden Diagramm gibt es kein größtes Element, aber zwei maximale, kein kleinstes Element, aber zwei minimale Elemente. Diese Ordnung ist kein Verband.



Definition 37. Sei (M, R) eine Äquivalenzrelation. Ist $x \in M$, so heißt die Menge $[x]_R := \{y \mid xRy\}$ die Äquivalenzklasse von x. x heißt Repräsentant von $[x]_R$.

Satz 38. *Sei $R \subseteq M \times M$ eine Äquivalenzrelation.*

1. *Zwei Äquivalenzklassen sind entweder disjunkt oder identisch.*
2. *Jedes $x \in M$ gehört genau zu einer Äquivalenzklasse.*

Beweis. Sei $[x]_R \cap [y]_R \neq \emptyset$. zu zeigen: $[x]_R = [y]_R$. Sei $a \in [x]_R$ und $b \in [x]_R \cap [y]_R$. Dann gilt xRa, xRb und yRb. Aus Symmetrie und Transitivität folgt yRa also $a \in [y]_R$. Die Behauptung ist symmetrisch, also gilt $[x]_R = [y]_R$. □

Ist R eine Äquivalenzrelation auf M, so lassen sich die Äquivalenzklassen als Elemente einer neuen Menge M/R (der *Faktormenge, Quotient*) auffassen. d.h. $M/R = \{[x]_R \mid x \in M\}$. Die Kardinalität von M/R heißt der *Rang* von R ($rk(R)$).

Beispiel 12.

1. Ist R eine Präordnung auf M (Z.B. Aufrufhierarchie von Modulen), so sei $\equiv$ definiert als $x \equiv y :\Leftrightarrow (xRy \wedge yRx)$, eine Äquivalenzrelation. Die vererbte Ordnung $R/\equiv$ auf $M/\equiv$ ist eine reflexive Ordnung. Dabei ist $R/\equiv$ definiert als $[x]_R \; R/\equiv \; [y]_R$ genau dann wenn xRy.

2. Sei M die Menge der Aussagen und $R = \{(A, B) \mid A \Rightarrow B$ ist Tautologie$\}$. Dann ist R reflexive Präordnung. Für die Relation $\equiv$ gilt: $A \equiv B$, genau dann wenn A zu B äquivalent ist.

3. Betrachte die Bruchdarstellung für rationale Zahlen, d.h. die Menge $\{(a, b) \mid a \in \mathbb{Z}, b \in \mathbb{N}\}$. Auf dieser Menge ist eine Äqivalenzrelation $\equiv$ definiert durch $(a, b) \equiv (c, d) :\Leftrightarrow a * d = b * c$. Dann sind die Äquivalenzklassen gerade die rationalen Zahlen $\mathbb{Q}$. Die Multiplikation ist dann definiert als $[(a, b)]_\equiv * [(c, d)]_\equiv := [(a * c, b * d)]_\equiv$, Die Addition ist definiert als $[(a, b)]_\equiv + [(c, d)]_\equiv := [(a * d + b * c, b * d)]_\equiv$. Ein Paar (a, b) ist ein Repräsentant der Äquivalenzklasse $[(a, b)]_\equiv$. Man kann hier sogar einen kanonischen Repräsentanten (a, b) (eine sogenannte *Normalform)* wählen, wobei a und b teilerfremd sind. Dieser kanonische Repräsentant kann dann dazu dienen, die Gleichheit von Äquivalenzklassen zu testen

4. Definiere auf $\mathbb{Z}$ für eine gegebene ganze Zahl p die Relation $R_p := \{(m, n) \mid m$ und n haben bei Division durch p den gleichen Rest$\}$. Für $(m, n) \in R_p$ schreibt man üblicherweise $m \equiv n(mod\ p)$. Offensichtlich gilt auch: $m \equiv n(mod\ p) \Leftrightarrow p$ ist Teiler von $(m - n)$. Diese Relation ist eine Äquivalenzrelation. Die Äquivalenzklassen $[x]_p$ heißen auch Restklassen modulo p in $\mathbb{Z}$. z.B. $[0]_p = \{0, p, -p, 2p, -2p, 3p, \ldots\}$

Zweistelligen Relationen kann man vielseitig verwenden: z.B. Vereinfachung von Aussageformen, Vereinfachung von Polynomausdrücken zur Summe von Produkten, Berechnung als Zustandsübergang. In diesem Zusammenhang sind folgende Eigenschaften von Relationen wichtig:

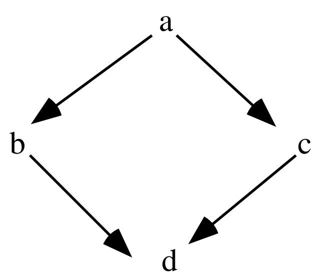
Definition 39. R sei eine (zweistellige) Relation auf einer Menge M.

1. Eine Relation R heißt *terminierend*, wenn es keine unendliche Kette $a_1\ R\ a_2\ R\ a_3\ R\ a_4\ \ldots$ gibt.

2. Wir sagen eine irreflexive Ordnung R ist *fundiert (Noethersch)*, wenn R^{-1} terminierend ist, d.h. es gibt keine unendlich absteigenden Ketten.

3. Ein Element a heißt *irreduzibel*, wenn es kein gibt mit aRb .

4. Eine Relation heißt *konfluent*, wenn es zu Elementen b und c mit aR^*b und aR^*c stets ein Element d gibt, so daß bR^*d und cR^*d gilt.



Satz 40. *Sei R eine konfluente und terminierende Relation. Dann gilt:*

1. *Für jedes a gibt es ein eindeutig bestimmtes irreduzibles b, so daß aR^*b.*

2. *Jede Kette $a\ R\ a_1\ R\ a_2 \ldots$ endet mit diesem Element b.*

Beweis. Zunächst mal existiert zu jedem a ein irreduzibles b, so daß aR^*b, denn anderenfalls gäbe es eine nichtterminierende Kette. Weiterhin muß auch jede Kette $a\ R\ a_1\ R\ a_2\ \ldots$ mit einem irreduziblen Element enden, denn alle solchen Ketten sind endlich. Wir nehmen an, es gibt zwei verschiedene irreduzible Elemente b und c mit aR^*b und aR^*c. Da R konfluent ist, gibt es ein Element d, so daß bR^*d und cR^*d. Da sowohl c als auch d irreduzibel ist, und $c \neq d$, folgt ein Widerspruch. □

Unter den Voraussetzungen des obigen Satzes sagt man, daß b die *Normalform* von a ist.

Beispiel 13. Betrachte wieder die Bruchdarstellung für rationale Zahlen mittels der Menge $\{(a,b) \mid a \in \mathbb{Z}, b \in \mathbb{N}\}$. Um die Normalformen zu berechnen, definiere eine Relation $\rightarrow$ auf den Paaren: $(a,b) \rightarrow (c,d)$, wenn eine Zahl $q \in \mathbb{N}$ mit $q > 1$ existiert mit $q{*}c = a$ und $q{*}d = b$. Offenbar ist diese Relation terminierend und auch transitiv. Diese ist auch konfluent, denn sei $(a_1,a_2) \rightarrow (b_1,b_2)$ mit q und $(a_1,a_2) \rightarrow (c_1,c_2)$ mit p. Dann definiere $d_1 = a_1/(p*q)$ und $d_2 = a_2/(pq)$. Offenbar gilt $(b_1,b_2) \rightarrow (d_1,d_2)$ mit p und $(c_1,c_2) \rightarrow (d_1,d_2)$ mit q. Es gilt $(a_1,a_2) \rightarrow (d_1,d_2)$. Das bedeutet, daß diese Berechnungsmethode auf eine (eindeutige) Normalform führt.

Bemerkung: Bei einer terminierenden Relation muß nicht notwendigerweise auch die inverse Relation terminierend sein. Ein Gegenbeispiel ist die lexikographische Ordnung (siehe 41). Fundierte Ordnungen sind oftmals hilfreich bei Terminierungsbeweisen von Algorithmen.

Beispiel 14.

1. Ausmultiplizieren von arithmetischen Ausdrücken ist konfluent und terminierend.

2. Ausmultiplizieren ist eine Relation auf den Darstellungen von Polynomen, die konfluent und terminierend ist.

3. in Kapitel 6 ist Reduktion von λ-Termen eine konfluente Relation.

Definition 41. Hat man Mengen $A_1, \ldots, A_n$ gegeben, wobei auf den jeweiligen Mengen eine Ordnung $<_i$ definiert ist, so lassen sich diese Ordnungen auf verschiedene Weisen auf kartesische Produkte fortsetzen.

1. *punktweise*: $(a_1, \ldots, a_n) <_{pkt} (b_1, \ldots, b_n) :\Leftrightarrow a_1 <_1 b_1, \ldots, a_n <_n b_n$,

2. *lexikographisch*: $(a_1, \ldots, a_n) <_{lex} (b_1, \ldots, b_n) :\Leftrightarrow \exists i \leq n : a_i <_i b_i \land \forall j < i : a_j = b_j$.

Satz 42.

1. *Sind alle Ordnungen $(A_i, <_i), i = 1, \ldots, n$ fundiert, so ist die lexikographische Ordnung auf dem kartesischen Produkt ebenfalls fundiert.*

2. *Sind alle Ordnungen $(A_i, <_i), i = 1, \ldots, n$ linear, so ist die lexikographische Ordnung auf dem kartesischen Produkt ebenfalls linear.*

Beweis. (Übungsaufgabe mit Induktion): Man betrachte eine unendlich absteigende Kette und zeige, daß es dann auch eine unendlich absteigende Kette in einem A_i geben müßte. □

Die lexikographische Ordnung ist die Sortierordnung z.B. bei Lexika, wo die Ordnung auf einem Einzelzeichen die alphabetische Reihenfolge ist und die Ordnung auf den Wörtern die lexikographische. Hier kommt es nur darauf an, daß die Ordnung linear ist, d.h., daß sich alle Worte vergleichen lassen. Die Sortierreihenfolge bei Dateien, bei einem Index o.ä. wird noch durch die Erweiterung der Ordnung auf die Ziffern und Sonderzeichen bestimmt.

Beispiel 15. Anwendung bei Terminierungsbeweisen. Betrachte den Blasen-Sort (Bubblesort), der ein n-Tupel aus natürlichen Zahlen solange mittels Vertauschung transformiert, bis die Elemente aufsteigend sortiert sind. Die einzige Transformationsregel ist:

Gilt für zwei Indizes $i \neq j$ mit $i < j$, daß $a_i > a_j$, so vertausche a_i und a_j.

Sei also $(a_1, \ldots a_i, \ldots a_j, \ldots, a_n)$ überführt in $(a_1, \ldots a_j \ldots a_i, \ldots, a_n)$. Um zu zeigen, daß der Blasensort terminiert, kann man so vorgehen:

1. $<$ ist fundierte Ordnung auf $\mathbb{N}_0$,

2. also ist auch die lexikographische Ordnung $<_{lex,n}$ auf $\mathbb{N}_0^n$ fundiert.

3. weiterhin gilt: Wenn man $(a_1, \ldots a_j \ldots a_i, \ldots, a_n)$ durch Anwenden der Regel aus $(a_1, \ldots a_i, \ldots, a_j, \ldots, a_n)$ erhält, so ist $a_j < a_i$, also auch $(a_1, \ldots a_j \ldots a_i, \ldots, a_n) <_{lex,n} (a_1, \ldots a_i, \ldots a_j, \ldots, a_n)$.

Also kann man die Operationsregel nicht unendlich oft anwenden.

5 Das Prinzip der vollständigen Induktion.

Induktionsbeweise treten auf beim Beweis von Eigenschaften über der Menge $\mathbb{N}_0$. Dies wird im ersten Teil des Paragraphen kurz dargestellt. Das Induktionsprinzip für induktiv definierte Mengen, das wir noch öfters brauchen werden, behandeln wir im Anschluß daran.

Sei E eine Eigenschaft der natürliche Zahlen $\mathbb{N}_0$. Prinzip der einfachen Induktion:

$$(E(m) \wedge (\forall n \geq m : E(n) \Rightarrow E(n+1))) \Rightarrow (\forall n \geq m : E(n))$$

Prinzip der vollständigen Induktion:

$$(E(m) \wedge (\forall n \geq m : (\forall k : m \leq k \leq n \Rightarrow E(k)) \Rightarrow E(n+1))) \Rightarrow (\forall n \geq m : E(n))$$

Beispiel 16. 1. einfache Induktion:

$E(n) :\Leftrightarrow \sum_{i=1}^{n} = \frac{1}{2}n(n+1)$. Beweis mit dem Prinzip der einfachen Induktion:

Induktionsanfang: $\sum_{i=1}^{1} 1 = 1 = \frac{1}{2}1(1+1)$.
Induktionsvoraussetzung: $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$.
Induktionsschluß: $\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + n + 1 = \frac{1}{2}n(n+1) + (n+1) = \frac{1}{2}(n+1)(n+2)$ d.h. $E(n+1)$ gilt. Also gilt: $\forall n.n \geq 1 \Rightarrow E(n)$

2. vollständigen Induktion:
   Seien die Fibonaccizahlen gegeben durch $Fib(0) := 0, Fib(1) := 1$ und $Fib(n+1) = Fib(n) + Fib(n-1)$ für $n > 1$. Behauptung: $E(n) := Fib(n) \leq 2^{n-1}$

   Induktionsanfang: $E(0), E(1)$ gilt, denn $0 \leq 2^{-1}, 1 \leq 2^{0}$
   Induktionsvoraussetzung: Für alle $0 \leq k \leq n$ gilt: $E(k)$, d.h. $Fib(k) \leq 2^{k-1}$.
   Induktionsschluß: für $n \geq 1$ gilt: $Fib(n+1) = Fib(n) + Fib(n-1) \leq 2^{n-1} + 2^{n-2} = 3 * 2^{n-2} < 2^{n} = 2^{(n+1)-1}$.
   Also gilt $\forall n \geq 0 : E(n)$.

Mengen lassen sich oft induktiv (rekursiv) definieren, was sehr oft einfacher ist, als die Menge direkt anzugeben. D.h. man gibt ein Konstruktionsverfahren an, das es ermöglicht, alle Elemente der Menge zu konstruieren. Um Eigenschaften dieser so konstruierten Menge zu beweisen, bedient man sich der Induktion, entweder über die Anzahl der notwendigen Konstruktionsschritte, oder im Fall, daß man den Elementen ihre Konstruktion ansieht, der sogenannten Induktion über die Struktur der erzeugten Elemente.

Beispiel 17. Menge der Aussageformen:

1. alle Atome sind Aussageformen (Verankerung)

2. Wenn A, B Aussageformen, dann sind $(\neg A), (A \vee B)$ und $(A \wedge B)$ auch Aussageformen.

Wir erklären die Höhe $h(A)$ folgendermaßen:

1. $h(A) := 0$ wenn A ein Atom ist,

2. $h(\neg A) := 1 + h(A)$

3. $h(A \vee B) := 1 + max\{h(A), h(B)\}$
4. $h(A \wedge B) := 1 + max\{h(A), h(B)\}$

Wir erklären die Anzahl der Blätter $b(A)$ folgendermaßen:

1. $b(A) := 1$ wenn A ein Atom ist,
2. $b(\neg A) := b(A)$
3. $b(A \vee B) := b(A) + b(B)$
4. $b(A \wedge B) := b(A) + b(B)$

Durch Induktion über die Konstruktion der Menge (oder die Struktur der Aussageformen) zeigen wir: $b(A) \leq 2^{h(A)}$.

1. Induktionsanfang: Wenn A ein Atom ist, dann $b(A) = 1 \leq 1 = 2^0$.
2. Induktionsschritt: Wir müssen alle Fälle durchgehen.
   (a) $b(\neg A) = b(A) \leq 2^{h(A)} < 2^{h(A)+1} = 2^{h(\neg A)}$
   (b) $b(A \vee B) = b(A) + b(B) \leq 2^{h(A)} + 2^{h(B)} \leq 2 * 2^{max\{h(a),h(B)\}} = 2^{h(A \vee B)}$
   (c) Für $b(A \wedge B)$ zeigt man das analog zu 2b.

   Damit ist die Behauptung gezeigt.

6 Der λ-Kalkül

Der λ-Kalkül wurde von Alonzo Church eingeführt, um Funktionen notieren zu können und um mit Funktionen rechnen zu können. Zum Beispiel würde man die Quadrat-Funktion, die mit `quadrat`$(x) := x * x$ definiert ist, als `quadrat` $:= \lambda x.x * x$ definieren. Die λ-Notation wird auch sehr häufig als informelle Notation für Funktionen benutzt. Diese Schreibweise wird in funktionalen Sprachen benutzt, insbesondere in LISP und dem LISP-Dialekt Scheme. Im λ-Kalkül sind alle Objekte einstellige Funktionen, wobei diese Objekte durch Ausdrücke dargestellt werden. Diese Ausdrücke nennt man auch λ-Terme. Ein Term der Form $(\lambda x.t)$ ist eine Funktion mit x als Argumentvariablen. Einen solchen Term bezeichnet man auch als Abstraktion. Eine Anwendung einer Funktion f auf eine Objekt (das auch eine Funktion ist) schreibt man einfach $(f\ g)$. Formal ist die Syntax[1] gegeben durch: $\langle\text{Term}\rangle ::= \langle\text{Variable}\rangle | \langle\text{Konstante}\rangle | (\langle\text{Term}\rangle\langle\text{Term}\rangle) | \lambda x.\langle\text{Term}\rangle$

D.h. : λ-Terme sind Variablen, Konstante, Anwendungen, oder Abstraktionen.

[1] Die Metazeichen haben folgende Bedeutung: $::=$ bedeuten "definiert als", $\langle \ldots \rangle$ ist ein definierter Name, $|$ bedeutet Alternative. Siehe auch Kapitel 12

Beispiel 18. Beispiele für λ-Terme sind:

1. $\lambda x.x$ Das ist die ist die Identitätsfunktion I d.h. $I\ x = x$.
2. $\lambda x.(\lambda y.(xy))$ Das ist der Anwendungskombinator.
3. $\lambda x.(\lambda y.x)$ (bekannt als K-Kombinator mit $(Kx)y = x$)
4. $\lambda x.\lambda y.\lambda z(xz)(yz)$ (bekannt als S-Kombinator mit $((Sx)y)z = (xz)(yz))$
5. $\lambda x.(xx)$

Definition 43. Die Menge der *freien Variablen* $FV(t)$ von t ist definiert als:

1. $\{x\}$, wenn t die Variable x ist,
2. $\emptyset$, wenn t eine Konstante ist.
3. $FV(r) \cup FV(s)$, wenn $t = (r\ s)$
4. $FV(s) \setminus \{x\}$, wenn t der Term $(\lambda x.s)$ ist

Eine Variable x ist in $FV(t)$, wenn sie ein freies Vorkommen in t hat. Nicht jedes Vorkommen von x muß frei sein.

Definition 44. Die Menge der *gebundenen Variablen* $GV(t)$ von t ist definiert als:

1. $\emptyset$, wenn t die Variable x ist,
2. $\emptyset$, wenn t eine Konstante ist.
3. $GV(r) \cup GV(s)$, wenn $t = (r\ s)$.
4. $GV(s) \cup \{x\}$, wenn t der Term $(\lambda x.s)$ ist und $x \in FV(s)$.

Einen Term ohne freie Variablen nennt man auch *geschlossen*. Im λ-Kalkül ist es möglich, als Erweiterung der Syntax zuzulassen, daß man geschlossenen Terme neue Namen (als Abkürzungen) gibt. Diese Namen werden dann wie Konstanten behandelt. Um λ-Terme evaluieren zu können, brauchen wir das Konzept der Ersetzung oder *Substitution*. Wenn t ein λ-Term ist mit der freien Variablen x, so soll $t[x \to s]$ der λ-Term sein, der durch Ersetzen von x durch s entsteht.

Definition 45. (Substitution)
Formal können wir den Prozess der Substitution rekursiv definieren:

1. $t[x \to s] := s$ wenn t die Variable x ist.

2. $t[x \to s] := t$ wenn t eine Konstante ist.

3. $(r\ t)[x \to s] := (r[x \to s]\ t[x \to s])$ für eine Anwendung $(r\ t)$.

4. $(\lambda y.t)[x \to s] := \lambda y.(t[x \to s])$ falls x und y verschieden sind, und ($y \notin FV(s)$ oder $x \notin FV(t)$).

5. $(\lambda y.t)[x \to s] := \lambda z.(t[y \to z][x \to s])$ falls x und y verschieden sind, $y \in FV(s)$ und $x \in FV(t)$, wobei z eine freie Variable ist, die nicht in s oder t vorkommt.

6. $(\lambda x.t)[x \to s] := \lambda x.t$.

Die Unterscheidung der Fälle einer Abstraktion in 45.4 und 45.5 ist so, daß der erste Fall der unproblematische ist, in dem keine Konflikte auftreten können. Würde man im Fall daß gleichzeitig $y \in FV(s)$ und $x \in FV(t)$ gelten, ohne Umbenennung ersetzen, so würde die Variable y in s, die vorher frei war, sozusagen eingefangen, da sie in den Bindungsbereich von λy. gerät. Um diesen Konflikt der eher zufälligen Wahl der Namen der gebundenen Variablen zu umgehen, wird vorher die Funktionsvariable umbenannt. Im Fall 6.) ist die Variable x in t gebunden, und hat nichts mit dem Wert der freien Variablen x aus $[x \to s]$ zu tun.

Diese Problemfälle und die Überlegungen dazu gelten nicht nur im scheinbar theoretischen Fall des λ-Kalküls, sondern auch im Prädikatenkalkül und in vielen Programmiersprachen. Die Methode, die hier verwendet wird, nennt sich in Programmiersprachen *statische Bindung* (static binding, lexical scoping), d.h. die zugeordnete freie Variable wird aus dem textuellen Kontext im Programm entnommen. Diese wird auch in Haskell (Gofer) und Scheme benutzt. Dynamische Bindung, wie sie in manchen LISP-Dialekten verwendet wird, erlaubt das Einfangen von freien Variablen, was immer wieder zu verwirrenden Effekten führen kann.

Die wichtigste Operation im λ-Kalkül ist die sogenannte *β-Reduktion*, die nichts anderes ist als die formale Einsetzung von Argumenten in die Definition einer Funktion. Der Name β-Reduktion hat historische Gründe.

Definition 46. *β-Reduktion*
Sei $(\lambda x.t)$ und s λ-Terme. Dann ist die β-Reduktion die folgende Ersetzung.

$$(\lambda x.t)s \to t[x \to s].$$

Wenn in einem λ-Term ein Unterterm entsprechend ersetzt wird, dann sprechen wir ebenfalls von β-Reduktion.

Eine weitere wichtige Äquivalenz, die wir schon benutzt haben, ist die *Umbenennung von gebundenen Variablen.*

Definition 47. *α-Konversion*
Sei $(\lambda x.s$ ein λ-Term. Dann ist die α-Reduktion die folgende Ersetzung.

$$\lambda x.s \to \lambda z.s[x \to z]$$

Wenn in einem λ-Term ein Unterterm entsprechend ersetzt wird, dann sprechen wir ebenfalls von α-Konversion.

Bei beiden Reduktionen ist zu beachten, daß die Regeln für die Ersetzung einzuhalten sind. D.h. in $\lambda z.s[x \to z]$ darf z weder frei noch gebunden in s vorkommen.

Beispiel 19. Die Funktion $\lambda x.y$ ist eine konstante Funktion, die jedes Argument auf y abbildet: $(\lambda x.y)s = y[x \to s] = y$. Diese Funktion sollte sich nicht wesentlich ändern, wenn man y durch eine andere Variable ersetzt: z.B. Ersetzung von y durch z sollte die Funktion ergeben, die alles konstant auf z abbildet. Genauso sollte die Ersetzung von y durch x eine Funktion ergeben, die alles konstant auf x abbildet. Aber, hätte man die Ersetzung $(\lambda x.y)[y \to x]$ in einer zu naiven Art und Weise definiert, so wäre das Ergebnis: $(\lambda x.x)$, was die Identitätsfunktion darstellt. Richtiges Ergebnis von $(\lambda x.y)[y \to x]$ ist $\lambda z.x$ mit $z \neq x$.

Übungsaufgabe 48. Die Relation $s \equiv_\alpha t$ sei der transitive Abschluß der α-Konversion, d.h. t kann man aus s durch eine Folge von α-Konversionen herstellen. Zeige, daß $\equiv_\alpha$ eine Äquivalenzrelation ist.

Definition 49. β-Reduktion.
Ein Term s reduziert auf einen Term t, (Notation $s \Longrightarrow_{\beta\alpha} t$) wenn t aus s durch eine Folge von β-Reduktionen und α-Konversionen (auch von inneren λ-Termen) erreicht wird. Wenn die Reduktion nur mit β-Reduktionen erreicht wurde, dann schreiben wir $s \Longrightarrow_\beta t$).

Mit anderen Worten, $\Longrightarrow_{\beta\alpha}$ ist die reflexive-transitive Hülle der von der β-Reduktion und der α-Konversionen erzeugten Relationen. Die Relation $\Longrightarrow_\beta$ ist die reflexive-transitive Hülle der von der β-Reduktion erzeugten Relation.

Ein *Redex* von r ist ein Unterterm der Form $((\lambda x.s)t)$. Ein Term ist in *Normalform*, wenn er keine Redexe mehr enthält. Wenn $s \Longrightarrow_{\beta\alpha} t$ und t ist in Normalform, so sagt man t ist Normalform von s.

Beispiel 20.

1. $(\lambda x.x\ y)F \Longrightarrow F\ y$
2. $(\lambda x.x)(\lambda x.x) \Longrightarrow (\lambda x.x)$
3. $(\lambda x.x\ x)(\lambda x.x\ x) \Longrightarrow (\lambda x.x\ x)(\lambda x.x\ x) \Longrightarrow \ldots$

Die intuitive Bedeutung ist, daß die Berechnung im λ-Kalkül durch die Anwendung der β-Reduktion erfolgt, und daß die Normalform das Ergebnis einer Berechnung ist. Das Beispiel 3 zeigt, daß die Reduktionsrelation nicht terminierend ist. Es gilt aber der wichtige Satz:

Satz Church-Rosser Theorem .

1. *Die Relation* $\Longrightarrow_{\beta\alpha}$ *ist konfluent.*
2. *Die Relation* $\Longrightarrow_{\beta}$ *ist konfluent modulo* $\equiv_\alpha$*.*

Der Beweis kann von Interessierten in entsprechenden Lehrbüchern nachgelesen werden.

Korollar 50. *Wenn s zwei Normalformen t_1 und t_2 hat, so sind t_1 und t_2 gleich bis auf Umbenennung von gebundenen Variablen.*

Beweis. Aus dem Satz von Church-Rosser folgt, daß t_1 und t_2 aufeinander reduziert werden können. Da beide keine Redexe mehr enthalten und α-Konversion keine Redexe einführen kann, sind t_1 und t_2 gleich bis auf Umbenennung von gebundenen Variablen.

Es gibt Terme, die keine Normalform besitzen. Diese entsprechen nichtterminierenden Berechnungen.

Definition 51. Es gibt verschiedene Auswertungsstrategien für λ–Terme.

1. Man spricht von *normaler Reihenfolge der Reduktion (normal order reduction)* , wenn β-Reduktion stets für den obersten und am meisten links (in der Termdarstellung) stehenden Redex angewendet wird.
2. Man spricht von *applikativer Reihenfolge der Reduktion (applicative order reduction)* , wenn $(\lambda x.t)s = t[x \to s]$ nur ausgeführt wird, wenn s in Normalform ist. Weiterhin muß die Auswertungsprozedur auch weiter links stehende Redexe vor weiter rechts stehenden reduzieren.

Beispiel 21. Betrachte den λ-Term $(\lambda x.a)((\lambda x.(x\ x))(\lambda x.(x\ x)))$. Wertet man diesen Term in applikativer Reihenfolge aus, so muß zunächst $((\lambda x.(x\ x))(\lambda x.(x\ x)))$ auf Normalform gebracht werden. Dieser Prozess terminiert nicht, so daß applikative Reihenfolge keine Normalform erzeugt. Versucht man es mit normaler Reihenfolge, so erhält man sofort als Normalform die Konstante a.

Satz 52. *Wenn ein Term s eine Normalform t hat, so berechnet Reduktion in normaler Reihenfolge diese Normalform in endlich vielen Schritten.*

Beweis. (siehe z.B. Barendregt: Functional Programming and λ-Calculus.) □

In dieser Hinsicht ist die normale Reihenfolge der applikativen Reihenfolge überlegen. Unter Implementierungsgesichtspunkten hat normale Reihenfolge, wenn sie naiv implementiert wird, Nachteile, denn der Ersetzungsprozess kann den λ-Term sehr stark vergrößeren und auch mehrfache Ausführung der gleichen Reduktion erfordern. Zum Beispiel: $(\lambda x.(f\ x\ x))s$

ergibt in normaler Reihenfolge den Term $(f\ s\ s)$. Im Falle, daß s nicht in Normalform ist, muß dann die Reduktion von s zweimal gemacht werden, um zur Normalform zu gelangen. Bei applikativer Reihenfolge dagegen wird s zur Normalform s' reduziert, und dann erst die Reduktion zu $(f\ s'\ s')$ durchgeführt. In den meisten Implementierungen der normalen Reihenfolge wird eine sogenannte Graphreduktion durchgeführt, wobei durch geschickte Darstellung erreicht wird, daß dieser Effekt der Verdopplung von Reduktionen nicht auftritt. Die Anzahl der Reduktionsschritte bis zur Normalform ist dann bei normaler Reihenfolge kleiner (oder gleich) der Anzahl der Reduktionen bei applikativer Reihenfolge. Die Implementierungen der applikativen Ordnung führen mit relativ wenig Aufwand zu effizienter Reduktion, denn die Optimierungsmöglichkeiten auf tieferer Ebene sind leichter zu implementieren. Die Sprachen mit normaler Reihenfolge haben aber dafür eine höheren Abstraktionsgrad. Es gibt eine Anzahl von funktionalen Programmiersprachen wie LISP, Scheme, ML (MetaLanguage), Miranda, Haskell, Gofer, die alle auf dem λ–Kalkül basieren. Einige verwenden ausschließlich applikative Reihenfolge (eager evaluation), andere normale Reihenfolge (verzögerte Auswertung, lazy evaluation), und einige erlauben eine Mischung, wie Scheme. Alle diese Sprachen verwenden allerdings nicht den reinen λ-Kalkül, sondern stellen Basisdatentypen und zugehörige Funktionen zur Verfügung, wie z.B. Integer, String usw. Außerdem erlauben diese Sprachen mehrstellige Funktionen. Dies stellt keine echte Erweiterung dar, denn eine zweistellige Funktion $\lambda(x, y).F(x, y)$ läßt sich folgendermaßen darstellen : $\lambda x.(\lambda y.F(x, y))$. Diese Übersetzung ist auch unter dem Namen Currying (nach H.B.Curry) bekannt.

Man kann mit Hilfe der λ-Terme und der β-Reduktion einen Berechenbarkeitsbegriff definieren (siehe Aufgabe und Abschnitt über Turingmaschinen), allerdings ist das technisch zu aufwendig, so daß wir hier nicht darauf eingehen.

Übungsaufgabe 53.

1. Zeige, daß der Operator $Y := \lambda f.((\lambda x.f(x\ x))(\lambda x.f(x\ x)))$ ein sogenannter Fixpunktoperator ist. d.h., für jeden λ-Term F gilt: $(Y\ F) \equiv_\alpha F(Y\ F)$
2. Schreibe informelle λ-Notation hin für
   (a) Die Ableitung von x^2 ist $2x$
   (b) Die Produktregel der Differentialrechnung
3. Wir benutzen die Kombinatoren I, S, K des Beispiels 18.
   (a) Reduziere $((S\ K)K)X$
   (b) Reduziere $((S\ S)S)S)$
4. (Church-Zahlen)
   Definiere $B := (S(K\ S))K$ und $Z_n := (S\ B)^n(K\ I)$, wobei X^m dasselbe wie $(\ldots\underbrace{(X\ X)X)\ldots X}_m)$ bedeuten soll. Zeige: $(Z_n\ F)X \equiv_\alpha F^nX$

7 Halbgruppen, Monoide, Homomorphismen

Sei M eine Menge. Eine totale Funktion $\circ : M \times M \to M$ heißt *Verknüpfung*. Statt $\circ(a, b)$ schreibt man auch $a \circ b$.

Definition 54. Sei $\circ$ eine Verknüpfung in der Menge M. $\circ$ heißt

1. *kommutativ (abelsch)*, falls $\forall a, b \in M : a \circ b = b \circ a$.
2. *assoziativ*, falls $\forall a, b, c \in M : (a \circ b) \circ c = a \circ (b \circ c)$

Definition 55. Eine Menge mit assoziativer Verknüpfung heißt *Halbgruppe*.

Schreibweise: $(M, \circ)$

Beispiel 22. Beispiel für Halbgruppen sind: $(\mathbb{N}, +), (\mathbb{N}_0, +), (\mathtt{ABB}(M, M), \circ)$. Ein wichtige Halbgruppe werden wir noch kennenlernen, die Halbgruppe der Worte.

Definition 56. Sei $(M, \circ)$ eine Halbgruppe.

1. ein Element x heißt *idempotent*, falls $x \circ x = x$.
2. Ein Element 0_M heißt *Nullelement*, falls $\forall m \in M : 0_M \circ m = m \circ 0_M = 0_M$.
3. Ein Element 1_M heißt *Einselement*, falls $\forall m \in M : 1_M \circ m = m \circ 1_M = m$.
4. $(M, \circ)$ heißt *linkskürzbar (bzw. rechtskürzbar)*, falls $\forall a, b, c \in M : a \circ b = a \circ c \Rightarrow b = c$ (bzw. $\forall a, b, c\ inM : b \circ a = c \circ a \Rightarrow b = c$.

Beispiel 23.

1. In $(\mathbb{N}, +)$ existiert kein Einselement und kein Nullelement.
2. Null- und Einselemente sind idempotente Elemente.
3. In $(\mathbb{N}_0, +)$ ist 0 Einselement, es gibt kein Nullelement.
4. In $(\mathbb{N}_0, *)$ ist 0 Nullelement, und 1 ist ein Einselement.
5. In $(\mathtt{ABB}(M, M), \circ)$ ist die Identität (I mit $Ix = x$ für alle x) ein Einselement, die überall undefinierte Funktion ist Nullelement.

Definition 57. Eine *Unterhalbgruppe* $(N, \circ)$ einer Halbgruppe $(M, \circ)$ ist eine nichtleere Teilmenge N von M, die Halbgruppe bzgl. $\circ$ ist. Schreibweise: $(N, \circ) \sqsubseteq (M, \circ)$

Definition 58. Eine Halbgruppe $(M, \circ)$ mit Einselement 1_M nennt man *Monoid.* Schreibweise: $(M, \circ, 1_M)$

Satz 59. *In einem Monoid existiert genau ein Einselement.*

Aber: das Einselement einer Unterhalbgruppe eines Monoids muß nicht das Einselement des Monoids sein: Beispiel: Das Monoid $(\{0\}, *, 0)$ ist Unterhalbgruppe von $(\mathbb{N}_0, *, 1)$, aber $0 \neq 1$.

Definition 60. Sei $(M, \circ, 1_M)$ ein Monoid und $1_M \in N \subseteq M$, so daß $(N, \circ, 1_M)$ eine Unterhalbgruppe von $(M, \circ, 1_M)$ ist. Dann heißt $(N, \circ, 1_M)$ *Untermonoid* von $(M, \circ, 1_M)$.

Satz 61. *Der Schnitt von Unterhalbgruppen einer Halbgruppe M ist wieder eine Unterhalbgruppe.*

Beachte, daß die Vereinigung von Unterhalbgruppen i.a. keine Unterhalbgruppe sein muß, da diese Menge möglicherweise nicht gegen Verknüpfung abgeschlossen ist.

Definition 62. Sei $(M, \circ)$ eine Halbgruppe und seien $U, V \subseteq M$. Dann ist $(U \sqcup V, \circ)$ definiert als $\bigcap\{W \mid (W, \circ) \sqsubseteq (M, \circ)$ und $U \subseteq W, V \subseteq W\}$.

$(U \sqcup V, \circ)$ heißt die von U und V erzeugte *Unterhalbgruppe.*

Eine direktere Beschreibungsmöglichkeit für $(U \sqcup V, \circ)$ ist: $U \sqcup V$ besteht aus allen endlichen Verknüpfungen von Elementen aus U und V. Für $U \sqcup U$ schreiben wir auch $\langle U \rangle$ und sagen $\langle U \rangle$ ist die von U erzeugt Unterhalbgruppe.

Definition 63. Sei $(M, \circ)$ eine Halbgruppe und sei $E \subseteq M$. E ist ein *Erzeugendensystem* von M, falls $\langle E \rangle = M$.

Bemerkung: Ist $\langle E \rangle = M$, so kann man jedes $m \in M$ als Produkt von Elementen aus E darstellen, allerdings möglicherweise nicht eindeutig.

Beispiel 24. $\{1\}$ ist Erzeugendensystem von $(\mathbb{N}, +)$. $\{1, 2\}$ ist Erzeugendensystem von $(\mathbb{N}, +)$, aber Darstellung ist nicht eindeutig, da $1 + 2 = 1 + 1 + 1$.

Definition 64. Seien $(M_1, \circ_1)$ und $(M_2, \circ_2)$ Halbgruppen. Eine Abbildung $\varphi : M_1 \to M_2$ heißt *(Halbgruppen-) Homomorphismus*, falls $\forall a, b \in M_1 : \varphi(a \circ_1 b) = \varphi(a) \circ_2 \varphi(b)$. φ heißt

- *Monomorphismus*, falls φ injektiv.
- *Epimorphismus*, falls φ surjektiv.

- *Isomorphismus*, falls φ bijektiv.
- *Endomorphismus*, falls $(M_1, \circ_1) = (M_2, \circ_2)$.
- *Automorphismus*, falls $(M_1, \circ_1) = (M_2, \circ_2)$ und φ bijektiv.

Schreibweise: $\varphi : (M_1, \circ_1) \to (M_2, \circ_2)$

Beispiel 25. : Sei $\mathbb{R}^+ := \{x \in \mathbb{R} \mid x > 0\}$. Betrachte $(\mathbb{R}^+, *)$ und $(\mathbb{R}^+, +)$ Die Abbildung $\varphi : (\mathbb{R}^+, *) \to (\mathbb{R}, +)$ mit $\varphi(x) := ln\ x$ ist ein Homomorphismus: $\varphi(x * y) = ln(x * y) = ln(x) + ln(y)$. φ ist sogar bijektiv, also ist φ ein Isomorphismus.

Definition 65. Seien $(M_1, \circ_1, 1_1)$ und $(M_2, \circ_2, 1_2)$ Monoide. Eine Halbgruppenhomomorphismus $\varphi : M_1 \to M_2$ heißt *(Monoid-) Homomorphismus*, falls $\varphi(1_1) = 1_2$.

Man sieht, daß die Begriffe sich immer auf eine Klasse von Strukturen beziehen, die eine gemeinsame Menge von Eigenschaften (Axiomen) haben. Dieselbe Abbildung auf einer Menge kann je nachdem ob man sie als Menge, als Halbgruppen, als Monoid, oder als Gruppe betrachtet, verschiedene Eigenschaften im Sinne der Homomorphismen haben. Die Kategorientheorie beschäftigt sich mit Klassen von ähnlichen Objekten und den Eigenschaften der zugehörigen Abbildungen (siehe Kapitel 8).

Definition 66. Sei $(M, \circ)$ eine Halbgruppe, $E \subseteq M$. E heißt *freies Erzeugendensystem* von M, falls gilt:

1. $\langle E \rangle = M$
2. Für alle Halbgruppen H: $\forall \varphi_0 : E \to H : \exists \varphi : \varphi$ ist ein Homomorphismus und $\varphi|_E = \varphi_0$. (jede Abbildung von $E \to H$ läßt sich zu einem Homomorphismus fortsetzen)

Eine Halbgruppe $(M, \circ)$ heißt *frei*, falls sie ein freies Erzeugendensystem besitzt.

Beachte, daß die Fortsetzung φ von φ_0 in 2 eindeutig bestimmt ist.

Beispiel 26. Die Halbgruppe $(\mathbb{N}, +)$ hat $\{1\}$ als freies Erzeugendensystem.

Übungsaufgabe 67. Wenn φ ein Isomorphismus von Halbgruppen (Monoiden) ist, so ist auch φ^{-1} ein Isomorphismus.

8 Kategorien, Homomorphismen

Kategorien beschreiben den Begriff der strukturverträglichen Abbildungen zwischen gleichartigen Objekten. Beispiele sind: Mengen und Abbildungen, Halbgruppen und Halbgruppenhomomorphismen, Monoide und Monoid-Homomorphismen, oder Programme einer Programmiersprache und semantikerhaltende Transformationen.

Definition 68. Eine Kategorie besteht aus einer Menge (Klasse) O von *Objekten* und einer Menge (Klasse) von *Morphismen* A. Jeder Morphismus f hat als zugeordnete Objekte genau eine Quelle $dom(f)$ und genau ein Ziel $cod(f)$. Wir schreiben $f : a \to b$, wenn a die Quelle und b das Ziel von f ist. Man schreibt auch $HOM(a, b)$ für die Menge aller Morphismen $f : a \to b$.

Folgende Axiome gelten:

Verknüpfung: Wenn $f : a \to b$ und $g : b \to c$, dann existiert der zusammengesetzte Morphismus $g \circ f : a \to c$.

Assoziativität: Wenn $f : a \to b$, $g : b \to c$, $h : c \to d$, dann gilt $(h \circ g) \circ f = h \circ (g \circ f)$.

Einselement: Zu jedem Objekt a gibt es eine Abbildung $1_a : a \to a$, so daß für alle Morphismen $f : a \to b$ und $g : b \to a$ gilt: $f \circ 1_a = f$ und $1_a \circ g = g$

Definition 69. Seien a, b Objekte einer Kategorie. Ein Morphismus $f : a \to b$ heißt

- *Monomorphismus*, falls ein Morphismus $f^- : b \to a$ existiert, so daß $f^- \circ f = 1_a$
- *Epimorphismus*, falls ein Morphismus $f^- : b \to a$ existiert, so daß $f \circ f^- = 1_b$
- *Isomorphismus*, falls φ Monomorphismus und Epimorphismus ist.
- *Endomorphismus*, falls $f : a \to a$.
- *Automorphismus*, falls $f : a \to a$ und f Isomorphismus.

Diese Definition von Mono-, Epi-, und Isomorphismus kann man als Rezept auffassen, in Strukturen die richtigen strukturerhaltenden Abbildungen zu finden. Beachte, daß alle kategorientheoretischen Betrachtungen die innere Struktur der Objekte ignorieren und nur die Pfeile (Morphismen) und deren Eigenschaften verwenden.

Z.B. kann man Unterobjekte und Quotienten von Objekten definieren:

- b ist ein *Unterobjekt* von a, wenn es einen Monomorphismus $f : b \to a$ gibt.
- b ist ein *Quotient* von a, wenn es einen Epimorphismus $f : a \to b$ gibt.

Abbildungen zwischen Kategorien nennt man Funktoren:

Definition 70. Seien C, D zwei Kategorien. F ist ein *Funktor* von C nach D, wenn es eine Abbildung $F : O_C \to O_D$ und eine Abbildung $F : A_C \to A_D$ gibt, so daß

- Für alle Morphismen $f : a \to b$ ist $F(f)$ ein Morphismus $F(f) : F(a) \to F(b)$
- Wenn $f \circ_C g$ definiert ist, dann gilt $F(f \circ_C g) = F(f) \circ_D F(g)$

- $F(1_C) = 1_D$

Beispiel 27. Kann man Monoide auch als Halbgruppen ansehen, wenn man die ausgezeichnete Rolle des Einselements (der inneren Verknüpfung) außer acht läßt. Dies entspricht einem Funktor von der Kategorie der Monoide in die Kategorie der Halbgruppen. Diese Abbildung ist ein sogenannter Vergiß-Funktor.

9 Alphabete, Worte, Formale Sprachen

In der Informatik sind Worte, Wortmengen (formale Sprachen) grundlegend zur Betrachtung der Syntax von Programmiersprachen. Sei A eine endliche nichtleere Menge (*Alphabet*).

Definition 71. Die Menge $A^+ := \bigcup\{A^n \mid n \geq 1\}$ heißt die Menge aller *Wörter* über A. Ist $w \in A^n$, dann heißt w *Wort über* A und n heißt die *Länge* von w. ($|w| = n$)

Statt $(a_1, a_2, \ldots, a_n)$ schreiben wir in Zukunft $a_1 a_2 \ldots a_n$.

Definition 72. Seien $u = u_1 \ldots u_n \in A^+$ und $v = v_1 \ldots v_m \in A^+$ zwei Worte über A, dann heißt die Verknüpfung $\bullet$ definiert durch $u \bullet v := u_1 \ldots u_n v_1 \ldots v_m$ die *Konkatenation* von u und v.

Statt $u \bullet v$ schreiben wir in Zukunft uv

Satz 73. *$(A^+, \bullet)$ ist eine Halbgruppe, (die Worthalbgruppe).*

Satz 74.

1. *Jedes Wort $w \in A^+$ ist eindeutig als Produkt von Elementen aus A darstellbar.*
2. *$(A^+, \bullet)$ ist links- und rechts-kürzbar*
3. *$(A^+, \bullet)$ ist frei*

Beweis. : 1 und 2 gelten, da Gleichheit in kartesischen Produkten elementweise definiert ist. 3: Sei $(H, \circ)$ eine Halbgruppe und sei $\varphi_0 : A \to H$ eine Abbildung. Definiere $\varphi : A^+ \to H$ durch $\varphi(a_1 \ldots a_n) := \varphi_0(a_1) \circ \ldots \circ \varphi_0(a_n)$. Wegen $\varphi(uv) := \varphi(u) \circ \varphi(v)$ definiert dies einen Homomorphismus. Da $\varphi|_A = \varphi_0$, ist φ eine homomorphe Fortsetzung von φ_0. Da trivialerweise $\langle A \rangle := A^+$, ist A ein freies Erzeugendensystem von A^+. $\square$

Satz 75. *Worthalbgruppen über Alphabeten gleicher Kardinalität sind isomorph.*

Beweis. Wir zeigen $A^+ \equiv B^+ \Leftrightarrow |A| = |B|$.

$\Rightarrow$: Sei $\varphi : A^+ \to B^+$ der Isomorphismus. Dann ist auch φ^{-1} ein Isomorphismus. Wir zeigen, daß φ eine Bijektion von A und B vermittelt: Sei $a \in A$ und sei $\varphi(a) = b_1 \dots b_n \in B^+$. Dann ist $a = \varphi^{-1}(b_1) \dots \varphi^{-1}(b_n)$. Daraus folgt $n = 1$, da wir die Struktur von A^+ kennen. Also ist $\varphi(a) \in B$. Analog können wir zeigen, daß $\varphi^{-1}(b) \in A$ für alle $b \in B$ ist. Also bildet $\varphi|_A$ eine Bijektion von A auf B.

$\Leftarrow$: Sei $\varphi_0 : A \to B$ eine Bijektion. Da A^+ frei mit Erzeugendensystem A ist, gibt es eine homomorphe Fortsetzung φ von φ_0. Da $\varphi(A) = B$ ist, ist φ ein Epimorphismus. Sei $\varphi(a_1 \dots a_n) = \varphi(a'_1 \dots a'_m)$ mit $a_i, a'_i \in A$. Es gilt dann auch $\varphi_0(a_1) \dots \varphi_0(a_n) = \varphi_0(a'_1) \dots \varphi_0(a'_m)$. Da $\varphi_0(a_i), \varphi_0(a'_i) \in B$ folgt mit Satz 74 daß $n = m$ und $\varphi_0(a_i) = \varphi_0(a'_i)$. Da φ_0 eine Bijektion ist, gilt dann auch $a_i = a'_i$ für alle i. Damit ist φ auch ein Monomorphismus. Also ist φ ein Isomorphismus. □

Satz 76. *Sei $(M, \circ)$ eine Halbgruppe mit endlichem Erzeugendensystem. Dann gilt: $(M, \circ)$ ist frei $\Leftrightarrow$ Es gibt eine Menge A so daß $(M, \circ)$ isomorph zu A^+ ist.*

Beweis.

$\Rightarrow$: Sei $(M, \circ)$ frei mit dem freien Erzeugendensystem $E \subset M$. Betrachte die Menge E^+. Sei ι die Einbettung von E in E^+, also $\iota(e) = e$ für alle $e \in E$. Da M frei mit Erzeugendensystem E, gibt es eine homomorphe Fortsetzung φ von ι. Wir zeigen, daß φ ein Isomorphismus ist.

1. φ ist Epimorphismus: Sei $f := e_1 \dots e_n \in E^+$. Betrachte $f' = e_1 \circ \dots \circ e_n$. Es gilt $\varphi(f') = \varphi(e_1 \circ \dots \circ e_n) = \varphi(e_1) \dots \varphi(e_n) = \iota(e_1) \dots \iota(e_n) = e_1 \dots e_n = f \in E^+$. Also ist φ ein Epimorphismus.
2. φ ist Monomorphismus: Sei $f := e_1 \circ \dots \circ e_n \in M$ und $g := e'_1 \circ \dots \circ e'_m \in M$ mit $\varphi(f) = \varphi(g)$. Dann ist $\varphi(e_1) \dots \varphi(e_n) = \varphi(e'_1) \dots \varphi(e'_m)$ und beides sind Elemente in E^+, woraus folgt, daß $m = n$ und $e_i = e'_i$ ist für alle $i = 1, \dots, n$.

$\Leftarrow$: Wir haben schon gezeigt, daß A^+ frei ist. Diese Eigenschaft haben auch alle isomorphen Halbgruppen.

□

Damit haben wir gezeigt, daß die freien Halbgruppen genau die Worthalbgruppen sind.

Bemerkung. Ist $(M, \circ)$ eine Halbgruppe, aber kein Monoid, dann kann man $(M, \circ)$ stets zu einem Monoid machen, indem man ein Einselement 1_M *adjungiert.* D.h. Man betrachtet $M \cup \{1_M\}$ und definiert $1_M \circ f = f \circ 1_M = f$ für alle $f \in M$ und $1_M \circ 1_M = 1_M$. Ist $(M, \circ)$ schon ein Monoid, so erhält man ein neues Monoid mit neuer 1. Ist $(A^+, \circ)$ eine Worthalbgruppe, dann liefert diese Konstruktion ein Einselement, das wir in Zukunft mit ε bezeichnen, ε nennt man auch das *leere Wort.* Die Länge des leeren Wortes soll 0 sein: $|\varepsilon| := 0$. $A^* := (A^+ \cup \{\varepsilon\}, \bullet)$ nennt man auch das vom Alphabet A erzeugte Wort-Monoid.

Worte werden auch als *Strings (Zeichenketten)* bezeichnet. Die Menge A^* bezeichnet man auch als *Kleene-Abschluß* von A.

Das Monoid A^* ist nicht mehr frei im Sinne der Definition für Halbgruppen, diese Definition kann aber für Monoide entsprechend angepaßt werden.

Ein Wort v ist *Präfix* eines Wortes w, wenn $vu = w$ für ein Wort u. Ein Wort v ist *Suffix* eines Wortes w, wenn $uv = w$ für ein Wort u. Eine echter Präfix (Suffix) v von w ist ein Präfix (Suffix) v mit $v \neq w$.

Definition 77. Sei A ein endliches Alphabet. Eine Menge $L \subseteq A^*$ heißt *formale Sprache*, Elemente von L heißen *Worte* der Sprache L.

Beispiel 28.

1. Die Menge $\{0, 1\}$ ist die Menge der Binärziffern. Diese werden auch *Bits* genannt.
2. Die Menge $\{0, 1\}^8$ ist die Menge der Binärfolgen der Länge 8. Diese werden auch *Bytes* genannt.
3. Die Menge der Palindrome : $\{a_1 \dots a_n \in A^* \mid a_1 \dots a_n = a_n \dots a_1\}$ ist eine formale Sprache.
4. Die Menge $\{\varepsilon\}$ ist eine formale Sprache
5. $\{0, 1\}^*$ ist die formale Sprache aller binären Strings
6. $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^*$ ist gerade die Sprache aller Bezeichnungen von natürlichen Zahlen (allerdings ist nicht garantiert, daß die erste Stelle $\neq 0$ ist).

Eine Möglichkeit, formale Sprachen zu beschreiben, sind *reguläre Ausdrücke über einem Alphabet* A. Diese sind folgendermaßen definiert:

Definition 78. Wir geben die Syntax für reguläre Ausdrücke an.

1. ε ist ein regulärer Ausdruck
2. a ist ein regulärer Ausdruck für $a \in A$.
3. Wenn r und s reguläre Ausdrücke sind, dann auch $(r + s)$, (rs) und (r^*).

Wenn man Sprachen R und S über dem Alphabet A hat, dann bezeichnet RS die Menge aller Konkatenationen von Wörtern aus R und S, d.h. $RS := \{uv \mid u \in R, v \in S\}$. Definiert man R^n durch $R^0 := \{\varepsilon\}$, $R^1 := R$ und $R^n := RR^{n-1}$, dann ist der (Kleene-)Abschluß definiert als $R^* := \bigcup\{R^n \mid n \geq 0\}$, der positive Abschluß ist definiert als $R^+ := \bigcup\{R^n \mid n \geq 1\}$.

Jeder reguläre Ausdruck r über dem Alphabet A beschreibt eine formale Sprache $I_A(r)$, wobei I die Interpretation ist, wenn man induktiv definiert:

1. $I_A(\varepsilon) := \{\varepsilon\}$.
2. $I_A(a) := \{a\}$ für $a \in A$.
3. $I_A((r+s)) := I_A(r) \cup I_A(s)$.
4. $I_A((rs)) := I_A(r)I_A(s)$.
5. $I_A(r^*) := (I_A(r))^*$.

Beispiel 29. Mit Hilfe der regulären Ausdrücke können wir die Sprache der Binärdarstellung aller positiven ganzen Zahlen ohne führende Nullen hinschreiben: $(1((0+1)^*)$

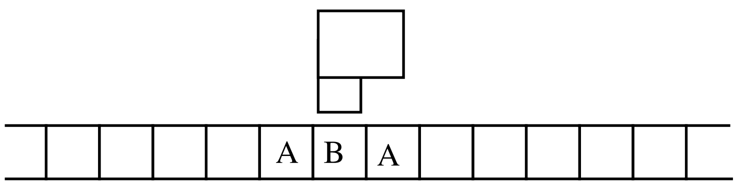
Definition 79. Eine formale Sprache L heißt *regulär*, wenn es einen regulären Ausdruck gibt, der L erzeugt.

Bemerkung: es gibt Sprachen, die nicht regulär sind.

10 Turingmaschine und Turingberechenbarkeit

Das abstrakte Modell eines möglichst einfachen und doch universellen Rechners wurde von Alan Turing (1936) eingeführt, um auch formal sagen zu können, was Berechenbarkeit heißt, insbesondere um herauszufinden, welche Funktionen berechenbar sind und ob es auch nicht-berechenbare Funktionen gibt.

Zunächst wollen wir die Turingmaschine informell einführen. Diese besteht aus einem beiderseits unendlichen Band mit Feldern, auf denen jeweils ein Zeichen steht. Diese Zeichen können mittels eines Schreib/Lesekopfes gelesen und geschrieben werden, allerdings kann der Schreib/Lesekopf nur das Feld bearbeiten, das gerade im Zugriff ist. Der Schreib/Lesekopf kann folgende Aktionen ausführen: Lesen eines Zeichens, Schreiben eines neuen Zeichens, Verschieben um ein Feld nach links, Verschieben um ein Feld nach rechts.



Eine spezielle Turingmaschine (Abkürzung: TM) T besteht außerdem noch aus einem endlichen Programm, das in Tabellenform geschrieben werden kann, und das die Aktionen des Kopfes steuert. Dieses Programm kann man sich in numerierten Einzelbefehle gegliedert vorstellen: Jeder Einzelbefehl ist eine Fallunterscheidung über das gelesene Zeichen und als Aktion kann ein Zeichen geschrieben, der Lese-Schreibkopf um eine Stelle verschoben werden, und zu einem anderen Befehl verzweigt werden. Befehle können folgendermaßen aussehen:

```
80.  Wenn gelesenes Zeichen = "a", dann  schreibe "b" ,
verschiebe den Lesekopf nach links,
weiter mit Befehl 199.
        oder gelesenes Zeichen  = "b", dann schreibe "b" ,
verschiebe den Lesekopf nach rechts,
weiter mit Befehl 3.
```

Als Eingabe (Input) erhält die TM ein beschriftetes Band, der Schreib/Lesekopf steht auf dem linkesten Zeichen der Eingabe. Der beschriftete Teil muß endlich sein, und es muß zu erkennen sein, wo er endet, deshalb nimmt man an, daß das Band außerhalb des beschrifteten Teils mit blank (wir schreiben B) initialisiert ist. Die Maschine fängt mit dem ersten Befehl an und arbeitet Befehle solange ab, bis es nicht mehr weitergeht, d.h., ein Fall nicht definiert ist oder zu einem nicht existierenden Befehl verzweigt wird. Die Ausgabe der TM ist der dann beschriftete Teil des Bandes.

Beispiel 30. Eingabe: eine Zahl n in Binärdarstellung. Ausgabe: $n+1$ in Binärdarstellung. Das Programm soll folgendes tun:

1. Lesekopf nach rechts schieben:
2. Addieren mit Übertrag von rechts nach links.

```
Z1.  Wenn 0, dann schreibe 0, gehe nach rechts, weiter mit Z1.
Wenn 1, dann schreibe 1, gehe nach rechts, weiter mit Z1.
Wenn B, dann schreibe B, gehe nach links, weiter mit Z2.
Z2.  Wenn 0, dann schreibe 1, gehe nach links, weiter mit Z3.
Wenn 1, dann schreibe 0, gehe nach links, weiter mit Z2.
Z3.
```

Wenn wir als Beschreibung des aktuellen Zustandes der Turingmaschine ein Wort $\alpha.z.\beta$ verwenden, so daß α der linke beschriftete Teil ist, z die Nr. des zu bearbeitenden Befehls und β der rechte beschriftete Teil, und der Schreib/Lesekopf auf dem linkesten Zeichen von β steht, so sieht eine Beispielrechnung für $5+1=6$ folgendermaßen aus (Wir ignorieren äußere Leerzeichen):

$$B.Z1.101 \to 1.Z1.01 \to 10.Z1.1 \to 101.Z1.B \to 10.Z2.1 \to 1.Z2.00 \to B.Z3.110$$

Der Gesamtzustand einer Turingmaschine ist eindeutig bestimmt durch:

- den Bandinhalt
- die Position des Schreib/Lesekopfes
- die Nr. des aktuellen Befehls.

Wir wollen nun die Turingmaschine formal definieren. Hierbei ist zu beachten, daß die formale Definition zwei Teile hat: 1. Turingmaschinenprogramm und 2. Ausführung. Wenn wir in Zukunft von einer Turingmaschine reden, so meinen wir das Turingmaschinenprogramm.

Definition 80. Eine *Turingmaschine* (Abkürzung: TM) T ist ein 5-Tupel (Z, A, B, δ, z_0) wobei

Z	eine endliche Menge von Zuständen
A	die Menge der erlaubten Bandsymbole (das Alphabet), wobei $Z \cap A = \emptyset$.
B	ein ausgezeichnetes Zeichen in A, (Leerzeichen, blank).
δ	ist eine partielle Funktion $\delta : Z \times A \to Z \times A \times \{L, R\}$ (Übergangsfunktion).
z_0	ist der Startzustand.

Die Zeichen L, R sollen die anzuwendende Verschiebung bedeuten. L für "nach links", R für "nach rechts". Man beachte, daß δ eine partielle Funktion sein kann, d.h. die Übergangsfunktion kann für bestimmte Zustände/gelesene Zeichen undefiniert sein. Diese Formalisierung sagt noch nichts darüber aus, was eine Turingmaschine tatsächlich tut. Um das zu tun, müssen wir sagen, wie der aktuelle Zustand formalisert wird, und wie T darauf operiert.

Definition 81. Sei $T = (Z, A, B, \delta, z_0)$ eine TM. Eine Konfiguration ID (instantaneous description) einer Turingmaschine sei ein Wort $\alpha q \beta \in A^*ZA^*$. Die Operation von T wird als Funktion $\delta^* : A^*ZA^* \to A^*ZA^*$ definiert: Sei $\alpha q \beta$ ein Wort aus A^*ZA^*. Wenn β das leere Wort ist, so ersetze β durch das Wort B. Sei jetzt $\beta = \beta_1\beta_2$, wobei β_1 das erste Zeichen von β ist und β_2 der Rest. Betrachte $\delta(q, \beta_1)$. Ist $\delta(q, \beta_1)$ undefiniert, so ist δ^* undefiniert. Anderenfalls ist $\delta(q, \beta_1) = (p, Y, M)$, wobei p der nächste Zustand, Y das zu schreibende Zeichen und M die Bewegungsrichtung ist. Dann ist die neue Konfiguration:

$\alpha Y p \beta_2$ falls $M = R$

$\alpha_1 p \alpha_2 Y \beta_2$, falls $M = L$, $\alpha \neq \varepsilon$, wobei $\alpha = \alpha_1\alpha_2$, und α_2 das letzte Zeichen von α,

$pBY\beta_2$ falls $M = L$, $\alpha = \varepsilon$.

Die Funktion δ^* nennt man auch *operationale Semantik* (oder *Interpreter*). Wenn eine Konfiguration K die Konfiguration K' nach sich zieht, so schreiben wir auch $K \vdash K'$ und nennen K' den Folgezustand von K. Eine *endliche Berechnung* mittels T bei Eingabe w ist eine endliche Folge von Konfigurationen $K_1, K_2, \ldots, K_n$, wobei $K_1 = z_0 w$ und $K_i \vdash K_{i+1}$ für alle i, und K_n keinen Folgezustand hat. Wenn die Berechnung bei Eingabe w endlich ist, so sagt man auch T *terminiert* (hält an) bei Eingabe w.

Der beschriftete Teil des Bandes wird auch als *Bandinschrift* bezeichnet, wobei die Bandinschrift rechts und links keine B's enthalten darf. D.h. die Bandinschrift ist das Wort, das (von links) mit dem ersten Zeichen $\neq B$ beginnt und mit dem letzten $\neq B$ endet.

Vergleicht man diese Definition mit der informellen Beschreibung, so sieht man, daß die Zustände den Befehlsnummern entsprechen, daß der Anfangszustand dem ersten Befehl entspricht, und die Übergangsfunktion δ dem Programm entspricht. Diese Definition ist konsistent mit dem Beispiel bis auf die dort verwendeten trennenden Punkte in der Notation der Konfiguration.

Bemerkung. Die Definition 81 beschreibt das Verhalten einer Turingmaschine als Operation auf Worten. Die obige Darstellung ist recht einfach, hat aber als leichten (formalen) Nachteil, daß es verschiedene Konfigurationen gibt, die man eigentlich als gleich ansehen würde, z.B. würde man $BBz1BB$ als äquivalent zu $Bz1B$ ansehen. Dies hat jedoch keinen Einfluß auf die Wirkungsweise.

Es gibt in der Literatur andere Definitionen des Begriffs Turingmaschine, die sich leicht, aber nur unwesentlich davon unterscheiden.

Definition 82. Sei $T = (Z, A, B, \delta, z_0)$ eine TM. Dann ist die von T *akzeptierte Sprache* $L(T)$ folgendermaßen definiert: $L(T) = \{\alpha \in (A \setminus \{B\})^* \mid$ die Turingmaschine hält nach einer endlichen Anzahl von Schritten für die Anfansgkonfiguration $z_0\alpha\}$.

Um zu sagen, was berechenbar heißt, definieren wir diesen Begriff für Funktionen $A^* \to A^*$.

Definition 83. Sei A ein Alphabet. Eine partielle Funktion $f : (A \setminus B)^* \to (A \setminus B)^*$ heißt *Turing-berechenbar*, wenn es eine Turingmaschine $T = (Z, A, B, \delta, z_0)$ gibt, so daß gilt:
Für alle $w \in (A \setminus B)^*$ für die $f(w)$ definiert ist, hält die Turingmaschine T bei Eingabe z_0w nach endlich vielen Schritten an, und die Bandinschrift ist gerade $f(w)$.

Es gibt verschiedene Definitionsvarianten von Turingmaschinen, z.B., unterscheidet man manchmal zwischen akzeptierenden Zuständen und Zuständen, in denen es nicht mehr weitergeht, oder das Band ist nur einseitig unendlich. Alle diese Definitionen haben sich bislang als äquivalent erwiesen, dies gilt auch für alle Verallgemeinerungen, wie z.B. das Einführen mehrerer Bändern (jeweils mit eigenem Schreib/Lesekopf) und auch für die Größe des Alphabets, solange es mehr als 2 Zeichen enthält. Andere Definitionen der Berechenbarkeit, die auf den ersten Blick nichts mit Turingmaschinen zu tun haben, haben sich ebenfalls als äquivalent zur Turing-Berechenbarkeit erwiesen. U.a. sind das:

- Haskell-Programme
- Markov Algorithmen
- Registermaschinen
- Goferprogramme
- λ-Kalkül
- μ-rekursive Funktionen

- Postsche Kalküle
- Semi-Thuesysteme

Hierbei nennt man zwei Maschinenmodelle äquivalent, wenn man mit beiden dieselben Berechnungen durchführen kann (wobei man noch Übersetzungen benötigt).

> Church'sche These: Alle irgendwie automatisch (mechanisch) berechenbaren Funktionen sind auch berechenbar mittels des λ-Kalküls, und alle Berechenbarkeitsbegriffe sind äquivalent.

Deshalb spricht man statt von Turingberechenbarkeit auch von *Berechenbarkeit*.

Der Unterschied zwischen partiellen Funktionen und totalen Funktionen ist wesentlich, denn der Berechenbarkeitsbegriff für Turingmaschinen, die auf jeder Eingabe anhalten, ist nicht äquivalent zur (allgemeinen) Berechenbarkeit, er ist echt schwächer. Will man statt den Wert einer Funktion eine Eigenschaft bestimmen, zum Beispiel: Eingabe: eine Zahl n, Frage: ist n eine Primzahl?, so kann man dies ebenfalls als Berechnung einer Funktion deuten: Eingabe ist die Zahl n in irgendeiner Darstellung. Wenn die entsprechende Turingmaschine anhält und die Bandinschrift ε ist, ist es eine Primzahl, anderenfalls nicht. Wenn die Turingmaschine nicht anhält, so kann man keine Aussage machen, d.h. das Resultat ist undefiniert. Im Fall unseres Beispiels der Primzahlen ist es allerdings wünschenswert und auch machbar, daß die Turingmaschine immer anhält (immer entscheidet), und nicht für bestimmte Zahlen nie anhält.

Da die direkte Handhabung der obigen Kodierung in Beweisen mittels und über Turingmaschinen etwas schwerfällig ist, und es andere (äquivalente) Definitionen mit angenehmeren Eigenschaften gibt, deren genauere Betrachtung aber den Rahmen dieses Skripts und Vorlesung sprengen würde, werden wir im folgenden so tun, als wüßten wir wie man es macht und annehmen, daß die Berechnung mittels (alternativ definierter) Turingmaschinen folgende Eigenschaften hat[2] :

Definition 84. *Annahme 1* über Eigenschaften der Turingmaschinen.

1. Wir betrachten eine TM T als partielle Funktion von $A^* \cup \{\perp\} \to A^* \cup \{\perp\}$, also mit Eingabe und Ausgabe und schreiben $T(w) = \perp$ falls die zugehörige Turingmaschine nicht terminiert. Wir definieren $T(\perp) = \perp$ für alle Turingmaschinen T.
2. Wir können Turingmaschinen wie Funktionen verknüpfen: $T_1 \bullet T_2$ ist eine Turingmaschine, die die Funktion mit $(T_1 \bullet T_2)(w) = T_1(T_2(w))$ für alle $w \in A^*$ realisiert. Falls $T_2(\perp) = \perp$, ergibt sich vernünftigerweise $(T_1 \bullet T_2)(\perp) = T_1(T_2(\perp)) = T_1(\perp) = \perp$.

[2] Für Interessierte: das technische Problem unserer einfachen Definition ist, daß man für einen Endzustand nicht feststellen kann, was die Bandinschrift ist, da man nicht weiß, wie weit sich der Lesekopf von der Bandinschrift entfernt hat. Um das in den Griff zu kriegen, braucht man entweder Extrasymbole, wie rechte und linke End-markierungen und evtl. noch weitere Hilfssymbole, oder ein weiteres Leerzeichen C und ein Schreib- und Eingabeverbot für B, so daß man den bereits benutzten Teil des Bandes erkennen kann.

3. Es gibt eine Turingmaschine T_{COPY} mit $T_{COPY}(w) = ww$.

Definition 85. Sei A ein Alphabet und T und F zwei verschiedene Worte in A^*, und E eine totale Funktion $E : A^* \to \{T, F\}$. Wir sagen auch: E ist eine Eigenschaft auf A^*. Dann ist E *entscheidbar (algorithmisch lösbar)*, wenn es eine Turingmaschine T_E gibt, die für jede Eingabe terminiert und $T_E(w) = E(w)$.

Auch der Begriff der Entscheidbarkeit hat sich als unabhängig von der Formalisierung herausgestellt. Entscheidbare Eigenschaften sind im Prinzip dasselbe wie totale berechenbare Funktionen. Im folgenden wollen wir zeigen, daß es nicht möglich ist, mit Hilfe einer Turingmaschine das Halteproblem zu lösen, d.h. zu berechnen, ob eine beliebige Turingmaschine T bei Eingabe w anhält oder nicht.

Definition 86. *Annahme 2* über Eigenschaften der Turingmaschinen. Es gibt eine (vernünftige) Kodierung φ von Turingmaschinen als Worte in A^*, so daß $\varphi : \{T \mid T \text{ ist } TM\} \to A^*$ ein Injektion ist.

Die Kodierung des Programms einer Turingmaschine könnte man so machen: Jedes Zeichen wird als ein String von 0'en kodiert, so daß jedes Zeichen durch mehr als eine 0 dargestellt wird. Das Alphabet $A = \{0, 1, B\}$, wird so kodiert, daß 0 durch 0, 1 durch 00 und B durch 000 dargestellt wird. Die Zustände werden ebenfalls durchnumeriert und in eindeutiger Weise als Nullen kodiert, L wird durch 0, R durch 00 kodiert. Ein Befehl besteht aus mehreren Fällen, für jedes mögliche gelesene Zeichen eine. Ein einzelner Fall hat die Bestandteile:

- a: das gelesene Zeichen,
- b: das geschriebene Zeichen
- D: Die Bewegungsrichtung (L oder R)
- z: der neue Zustand.

Ein Fall eines Befehl wird dann kodiert als $1\varphi(a)1\varphi(b)1\varphi(D)1\varphi(z)1$.
Ein Befehl (entspricht einem Zustand) wird kodiert als $110\varphi(\text{Fall}_1)0\ldots0\varphi(\text{Fall}_n)011$ kodiert,
Das ganze Turingmaschinenprogramm wird kodiert als $1110\varphi(\text{Befehl}_1)0\varphi(\text{Befehl}_2)0\ldots0\varphi(\text{Befehl}_n)0111$

Solche eineindeutigen Kodierungen bezeichnet man auch als Gödelisierung. Falls die entsprechende Kodierung eine natürliche Zahl ist, spricht man auch von Gödelnummer .

Es ist kein Problem, die Turingmaschinen fortlaufend zu numerieren, denn die Menge der Turingmaschinen ist abzählbar. D.h., wir haben $T_i, i = 1, \ldots$ ist die Folge aller Turingmaschinen über $\{0, 1\}$ (siehe Satz 27). Hieraus folgt auch sofort, daß es Funktionen $f : \mathbb{N} \to \mathbb{N}$ gibt, die nicht berechenbar sind, denn die Menge der Turingmaschinen ist abzählbar, während die Menge der Funktionen überabzählbar ist (siehe Satz 28).

Wir geben auch ein konkretes Problem an, daß diese unangenehme Eigenschaft hat:

Das Halteproblem ist das Problem eine feste Turingmaschine T_h zu entwerfen, die für alle Turingmaschinen T und für alle Eingaben w entscheidet, ob T auf w anhält.

Satz A. Turing *Es gibt keine Turingmaschine, die das Halteproblem löst.*

Beweis. Wir nehmen an, es gibt ein Programm T_n, so daß

$T_n(i) = 0$, wenn $T_i(i)$ nicht terminiert, und
$T_n(i) = 1$ wenn $T_i(i)$ terminiert.

Wir ändern das Programm T_n ab zu einem Programm T_m, das, wenn T_n mit 0 terminiert, auch mit 0 terminiert, und im Falle, daß T_n mit 1 terminiert, eine nichtterminierende Berechnung startet. Es gilt jetzt:

$T_m(i) = 0$ gdw. $T_i(i)$ nicht terminiert
$T_m(i)$ terminiert nicht gdw. $T_i(i)$ terminiert

Betrachten wir jetzt die Selbstanwendung: $T_m(m)$. Für diese gilt: $T_m(m)$ terminiert gdw. $T_m(m)$ nicht terminiert. Das ist ein Widerpsruch. Also gibt es kein Programm T_n. □

Definition 87. Eine *universelle Turingmaschine U* ist eine Turingmaschine, die als Eingabe die Kodierung einer Turingmaschine T und einen Eingabestring w, so daß gilt: $U(\varphi(T)w) = T(w)$.

Mit einigem technischen Aufwand kann man zeigen:

Satz 88. *Es gibt eine universelle Turingmaschine U.*

Dies bedeutet, daß es eine einzige Turingmaschine gibt, mit der man im Prinzip jede Berechnung ausführen (simulieren) kann. Dies ist auch die theoretische Grundlage für Rechenanlagen, die ja auch ihr Programm im gleichen Speicher wie die Daten haben. Man muß aber beachten, daß Turingmaschinen Idealisierungen von solchen Rechnern sind, da eine konkrete Rechenanlage nur begrenzten Speicher hat.

Andererseits bedeutet dies auch, daß die Nichtberechenbarkeit des Halteproblems auch für alle anderen Programmiersprachen gilt, die so ausdrucksstark wie Turingmaschinen sind.

11 Graphen und Bäume

Definition 89. Eine *Inzidenzstruktur* ist ein Tripel (E, K, I), wobei E und K disjunkte Mengen sind und $I \subseteq E \times K$. Diese Inzidenzstruktur heißt *ungerichteter Graph*, wenn für alle

$k \in K : 1 \leq |\{e \in E \mid (e,k) \in I\}| \leq 2$. Die Menge E sind die Knoten (Ecken, engl: vertex) und K die Kanten (engl: edges) des Graphen. Die Bedingung ist dann, daß auf jeder Kante mindestens ein Knoten liegt, aber nicht mehr als zwei.

Definition 90. Falls $e_1 I k$ und $e_2 I k$, bezeichnet man e_1, e_2 als *Endknoten* der Kante k, man sagt e_1 und e_2 sind Nachbarn (oder direkt verbunden). Eine Kante k heißt *Schlinge*, falls $|\{e \in E \mid (e,k) \in I\}| = 1$. Eine Kante k heißt *Mehrfachkante*, falls es ein k' gibt, so daß die Endknoten von k und k' übereinstimmen. Ein Graph G heißt *schlicht*, falls er weder Schlingen noch Mehrfachkanten besitzt.

Bemerkung. Ein schlichter Graph kann auch durch ein Tupel $(E, \mathcal{K})$ repräsentiert werden, wobei $\mathcal{K} \subseteq \mathcal{P}(E)$ und $K \in \mathcal{K} \Rightarrow |K| = 2$, d.h. die Kanten sind durch ihre Endknoten eindeutig bestimmt. Wir schreiben dann $G = G(E, \mathcal{K})$. Ein Graph definiert eine Äquivalenzrelation der verbundenen Knoten. Diese Relation ist die symmetrische und transitive Hülle der durch die Kanten erzeugten Relation. Einen Repräsentanten dieser Äquivalenzklasse nennt man *Zusammenhangskomponente*. Wenn alle Knoten verbunden sind, d.h. wenn diese Relation nur eine Äquivalenzklasse hat, dann nennt man den Graph *zusammenhängend*.

Definition 91. Ein *gerichteter Graph (Digraph, directed graph)* G ist ein Tupel (E, K) mit $K \subseteq E \times E$. Der zugrundeliegende (ungerichtete) Graph ist das Paar (E, K') mit $K' = \{\{e, f\} \mid (e, f) \in K\}$. Wir schreiben dann $G = G(E, K)$. Ist $(a, b) \in K$, so ist a der *Anfangsknoten von k* und b der *Endknoten von k*. Man sagt auch a ist *direkter Vorgänger* von b und b *direkter Nachfolger* von a. Sei $e \in E$. Die Zahl $\gamma^+(e) := |\{a \mid (e, a) \in K\}|$ heißt *Ausgangsgrad (outdegree)* von e. Die Zahl $\gamma^-(e) = |\{a \mid (a, e) \in K\}|$ heißt *Eingangsgrad (indegree)* von e. $\gamma^+(e) + \gamma^-(e)$ heißt *Eckengrad (degree)* von e. Der Knoten $e \in E$ heißt *Anfangsknoten (Endknoten)* von G, falls $\gamma^-(e) = 0$ (bzw. $\gamma + (e) = 0$).

Im folgenden betrachten wir nur gerichtete Graphen im Sinne der Definition 91.

Ein gerichteter Graph definiert eine Präordnung auf seinen Knoten, wenn man die Kanten als erzeugende Relationen der Ordnung ansieht und die transitive Hülle bildet.

Definition 92. Sei $G = G(E, K)$ ein gerichteter Graph. Der Graph $G' = G'(E', K')$ mit $E' \subseteq E$ und $K' \subseteq K$ heißt *Teilgraph* von G. Der Teilgraph $W = (\{e_i | i = 1, \ldots, n + 1\}, \{(e_i, e_{i+1}) \mid i = 1, \ldots, n\})$ heißt *Weg der Länge n von e_1 nach e_2*. W heißt *Kreis*, falls $e_1 = e_n$. Wir schreiben $W = e_1 \ldots e_n$ für einen Weg. Sei $W = e_1 \ldots e_n$ ein Weg. Der Knoten e_i heißt *Nachfolger* (bzw. *Vorgänger*) von e_j, falls $j < i$ (bzw. $i < j$). Im Falle $j + 1 = i$ (bzw. $i + 1 = j$) spricht man auch von direktem Nachfolger (direkter Vorgänger).

Ein gerichteter Graph ohne Kreise heißt *azyklisch*, man spricht auch von *DAG (directed acyclic graph)*, die entsprechende Ordnung ist eine irreflexive Ordnung. Ein azyklischer (endlicher) gerichteter Graph hat mindestens eine Quelle und eine Senke, wobei eine Quelle ein minimales Element und eine Senke eine maximales Element in der Ordnung ist (ob maximal oder minimal, hängt an der Richtung: man interpretiert normalerweise eine Ordnung R wie $<$). Ein gerichteter Graph heißt *zusammenhängend*, falls der zugrundeliegende Graph zusammenhängend ist. Ein gerichteter Graph heißt *stark zusammenhängend*, falls es zwischen je zwei Knoten e und f einen Weg von e nach f oder von f nach e gibt.

Ein Graph $G(E, K)$ mit einer Abbildung $\varphi : G \to D$ heißt (knoten-)markierter Graph mit Marken aus D. Wenn man zusätzlich eine Abbildung $\varphi : K \to D$ betrachtet, dann ist der Graph kanten-markiert.

Ein ungerichteter Graph heißt k-färbbar, wenn es eine Knoten-Markierung φ gibt mit Marken aus $\{1, \ldots, k\}$ und für alle Knoten $k_1 \neq k_2$ gilt: wenn $\varphi(k_1) \neq \varphi(k_2)$, dann sind k_1 und k_2 nicht direkt mittels einer Kante verbunden. Ein 2-färbbarer Graph heißt auch *bipartit*.

Definition 93. Ein Graph $T = T(E, K)$ heißt *Baum*, falls $E = \emptyset$ oder

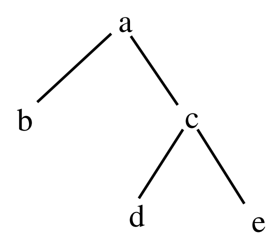
- es gibt genau eine Wurzel $e = w(T)$, das ist ein Knoten e mit $\gamma^-(e) = 0$,
- für jeden Knoten $f \neq w(T)$ gilt $\gamma^-(f) = 1$,
- für alle Knoten $f \neq w(T)$ gibt es einen Weg von $w(T)$ nach f.

Bemerkung. Die charakterisierende Eigenschaft eines Baumes als Ordnung ist, daß es genau ein maximales Element gibt, und daß für beliebige Knoten $a \neq b$ mit $a < b$ die Menge $\{c \mid a \leq c \leq b\}$ linear geordnet ist. Das entspricht der Eigenschaft von Bäumen, daß es entweder genau einen oder keinen Weg zwischen a und b gibt.

Definition 94. Sei T ein Baum. Ein Endknoten von T heißt *Blatt*. Ein Knoten, welcher kein Endknoten ist, heißt *innerer Knoten*. Der direkte Vorgänger (bzw. Nachfolger eines Knotens $e \in E$ heißt *(Vater (Mutter)* (bzw. *Tochter (Sohn)*) von e. Zwei Knoten e, f heißen *Schwestern (Brüder)*, falls sie den gleichen Vater (Mutter) besitzen. Ein *Teilbaum* T' eines Baumes T ist ein Untergraph von T, der als Wurzel w eine Tochter von $w(T)$ hat, so daß T' alle Knoten enthält, die Nachfolger von w bezüglich T sind. D.h. ein Baum ist vollständig bestimmt durch die Wurzel und seine Teilbäume. Ein Knoten e hat *Niveau* $N(e)$, falls der Weg von $w(T)$ nach e die Länge $N(e)$ besitzt . Der Baum hat die *Höhe* $h(T)$, wobei $h(T) := max\{N(e) \mid e \in E\}$.

Eine Menge von Bäumen bezeichnet man auch als *Wald*.

Beispiel 31. Betrachte den folgenden Baum.



Dann hat der Baum zwei Teilbäume: $\{b\}$ und $\{c, d, e\}$. a und c sind die inneren Knoten, d und e sind Schwestern (Brüder), a ist die Mutter von b, und b und c sind die Söhne von a. Die Blätter sind b, d und e. Der Knoten a hat Niveau 0, c hat Niveau 1, und d hat Niveau 2. Die Höhe des Baumes ist 2.

Definition 95. Ein Baum der zusätzlich noch eine Ordnung $\prec$ auf den Knoten besitzt, die für jeden Knoten e eine lineare Ordnung auf allen Söhnen ist, heißt *geordneter Baum (planarer Baum).*

Beachte: Diese Ordnung hat nichts zu tun mit der Ordnung, die von den gerichteten Kanten erzeugt wird. Wenn man im Beispiel 31 die Ordnung $b \prec c$ und $d \prec e$ nimmt, so ergibt sich ein geordneter Baum, und die Abbildung spiegelt die Ordnung wieder.

Definition 96. Sei $T = T(E, K)$ ein geordneter Baum. T heißt *binärer Baum*, falls jeder Knoten höchstens zwei Töchter besitzt und jede Tochter entweder als linke oder als rechte Tochter gekennzeichnet ist. Ist T ein binärer Baum mit Teilbäumen T_1 und T_2 und ist $w(T_1)$ linker Sohn und $w(T_2)$ rechter Sohn, so heißt T_1 *linker Teilbaum* und T_2 *rechter Teilbaum.*

Bemerkung: Geordnete Bäume (insbesondere binäre Bäume) spielen eine wesentliche Rolle bei der Darstellung der syntaktischen Struktur von Ausdrücken. Betrachte zum Beispiel den Ausdruck:

```
exp(a) + bc - sin(d)
```

Die Blätter korrespondieren mit den Variablen und Konstanten, die inneren Knoten mit den Operatoren.

Geordnete Bäume kann man auch ansehen als präfixabgeschlossene formale Sprachen über $\{1, \ldots, n\}$, wobei eine formale Sprache L *präfixabgeschlossen* ist, wenn mit jedem Wort auch jeder Präfix in L enthalten ist. Im allgemeinen fordert man noch die *Lückenlosigkeit* von L: Für jedes Wort $e.i \in L$ und $1 \leq j < i$ ist auch $e.j \in L$.

Definition 97. Ein *Baumbereich* ist eine formale Sprache über $\{1, \ldots, n\}$, die präfixabgeschlossen und lückenlos ist.

Das leere Wort entspricht der Wurzel des Baumes, Nichtleere Worte von L kann man als Baumadressen ansehen. 1.3.1 entspricht dem Knoten, den man auf folgendem Weg erreicht: Starte an der Wurzel, gehe zum ersten Sohn, von da aus gehe zum dritten Sohn, von da aus gehe zum ersten Sohn.

Definition 98. Ein *markierter Baum* (bewerteter Baum, labelled tree) ist ein Tripel (T, φ, M), wobei T ein Baumbereich, M eine Menge von Markierungen ist, und φ eine Abbildung $\varphi : T \to M$ ist.

Eine Standardoperation für markierte Bäume ist :
$\texttt{frontier}(T, \varphi, M) := \varphi(w(T))$, falls $w(T)$ ein Blatt ist
$\texttt{frontier}(T_1, \varphi, M) \ldots \texttt{frontier}(T_n, \varphi, M)$, falls $T_1, \ldots, T_n$ die n Teilbäume von T sind

Diese Funktion sammelt genau die Markierungen der Blätter des Baumes auf.

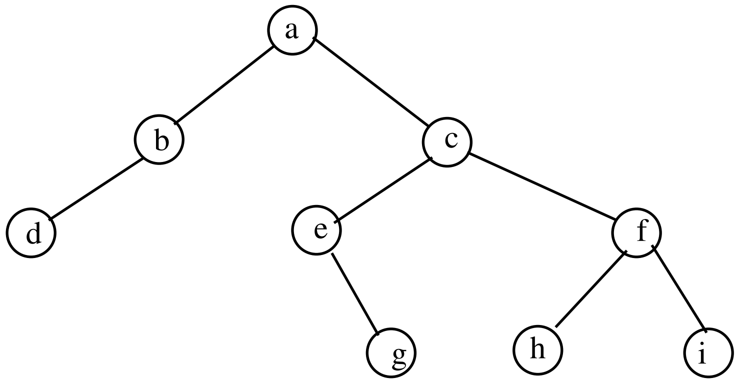
Es gibt verschiedene Arten, die Knoten eines geordneten Baumes zu linearisieren. bzw. bei einer Verarbeitung zu durchlaufen.

Definition 99. Sei $T = T(E, K)$ ein geordneter Baum.

- Die *Postorder* von T ist ein Wort in E^* mit
  $$\begin{array}{lll} \texttt{PO}(T) & := & \varepsilon, \text{ falls } T = \emptyset, \\ & & \texttt{PO}(T_1) \ldots \texttt{PO}(T_n) w(T), \text{ falls } T_i \text{ der } i\text{-te Teilbaum von } T \text{ ist.} \end{array}$$
- Die *Preorder* (Reihenfolge bei *Tiefensuche*) von T ist ein Wort in E^* mit
  $$\begin{array}{lll} \texttt{PR}(T) & := & \varepsilon, \text{ falls } E = \emptyset, \\ & & w(T)\texttt{PR}(T_1) \ldots \texttt{PR}(T_n), \text{ falls } T_i \text{ der } i\text{-te Teilbaum von } T \text{ ist.} \end{array}$$
- Die *Inorder* von T ist ein Wort in E^* mit
  $$\begin{array}{lll} \texttt{IO}(T) & := & \varepsilon, \text{ falls } E = \emptyset, \\ & & \texttt{IO}(T_1) w(T) \texttt{IO}(T_2) \ldots \texttt{IO}(T_n), \text{ falls } T_i \text{ der } i\text{-te Teilbaum von } T \text{ ist.} \end{array}$$
  Dies ist gerade die Reihenfolge von links nach rechts beim Durchlaufen eines binären Baumes.
- Die *military order* (Reihenfolge bei *Breitensuche*) von T ist ein Wort in E^* das folgendermaßen definiert ist: Sei $w_k = x_1 \ldots x_n$ ein Wort, das alle Knoten von T mit Niveau k in der Links-Rechts-Ordnung enthält, und h die Höhe von T, Dann ist $\texttt{BSO}(T) := w_0 w_1 \ldots w_h$

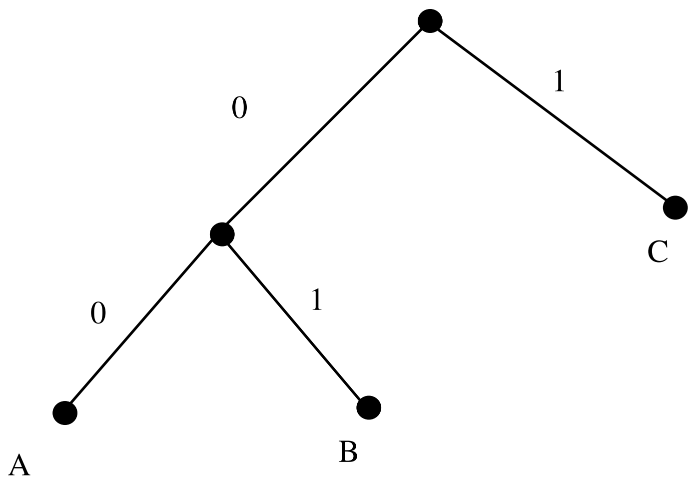
Bemerkung. Die Preorder ist die Ordnung der Knoten, die man beim Durchlauf des Baumes von rechts links nach rechts, oben nach unten, erhält. Preorder, Postorder und Breitensuchordnung hängen nur von der Links-Rechts-Ordnung ab, während Inorder bei binären Bäumen davon abhängt, welche Knoten als rechte und welche als linke Söhne definiert werden.

Beispiel 32.



Postorder: $PO(T) = PO(T_b)PO(T_c)a = dbgehifca$
Preorder: $PR(T) := abdcegfhi$
military order: $abcdefghi$
Inorder: $dbaegchfi$ (hier ergibt sich ein Unterschied für lückenlos oder nicht)

Beispiel 33. Kodierung eines Alphabets A mittels eines anderen Alphabets B. Sei $\varphi_0 : A \to B^+$ eine injektive Abbildung. Diese Abbildung kann man eindeutig homomorph fortsetzen zu einem Homomorphismus $\varphi : A^* \to B^*$. Um eindeutige Kodierbarkeit zu gewährleisten, interessiert man sich für solche Kodierungen, bei denen φ ein Monomorphismus ist. Eine hinreichende Bedingung dafür ist, daß die Menge $\varphi(A)$ als formale Sprache die Präfixeigenschaft hat. Eine Sprache L die Präfixeigenschaft (auch *Fanobedingung* genannt), wenn für zwei Worte $v, w \in L$ gilt, daß v kein Präfix von w sein kann. Diese Kodierungen lassen sich in Baumform darstellen, so daß ein Dekodierungsalgorithmus direkt mit diesem Baum arbeiten kann. Kodiert man die Zeichen A, B, C zum Beispiel als $00, 01, 1$ so erhält man den Baum



Die kodierten Zeichen des Alphabets sind dann gerade die Blätter des Baumes.

12 Kontextfreie Grammatiken und Sprachen

Kontextfreie Grammatiken werden benutzt, um formale Sprachen (z.B. Syntax von Programmiersprachen, Zahldarstellungen, Syntax der Prädikatenlogik) zu definieren.

Definition 100. Eine *kontextfreie Grammatik* (CFG, context free grammar) ist 4-Tupel $G = (N, T, P, \sigma)$ mit

1. einer endlichen Menge N von Hilfszeichen (Nonterminals)

2. einer endlichen Menge T von Terminalzeichen (Terminals), das eigentliche Alphabet, so daß $N \cap T = \emptyset$
3. einem ist ein ausgezeichnetes Hilfszeichen (Startzeichen, Axiom) $\sigma \in N$
4. einer endlichen Menge (Produktionensystem) $P \subseteq N \times (N \cup T)$

Bemerkung. $(A, w) \in P$ heißt *Produktion* oder *Regel* von G. statt $(A, w) \in P$ schreibt man auch $A \to w$. Die Grammatik heißt kontextfrei, da die Ersetzungen der Nichtterminale unabhängig vom Kontext gemacht werden dürfen.

Beispiel 34. Ein Teil der Syntax von Programmiersprachen:

$$
\begin{array}{lll}
\langle\text{Ausdruck}\rangle & ::= & \langle\text{Atom}\rangle \mid \langle\text{Zahl}\rangle \mid (\langle\text{Ausdrücke}\rangle) \\
\langle\text{Ausdrücke}\rangle & ::= & \langle\text{Ausdruck}\rangle \mid \langle\text{Ausdruck}\rangle\langle\text{Ausdrücke}\rangle \\
\langle\text{Zahl}\rangle & ::= & \langle\text{Ziffer}\rangle \mid \langle\text{Ziffer}\rangle\langle\text{Zahl}\rangle \\
\langle\text{Ziffer}\rangle & ::= & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
\langle\text{Atom}\rangle & ::= & \langle\text{Buchstabe}\rangle \mid \langle\text{Buchstabe}\rangle\langle\text{Zeichen}\rangle \\
\langle\text{Buchstabe}\rangle & ::= & A \mid B \mid C \mid D \mid \ldots \\
\langle\text{Zeichen}\rangle & ::= & \langle\text{Ziffer}\rangle \mid \langle\text{Zahl}\rangle
\end{array}
$$

Definition 101. Sei $G = (N, T, P, \sigma)$ eine CFG. Die Relationen $\underset{G}{\Longrightarrow}, \underset{G,l}{\Longrightarrow}, \underset{G,r}{\Longrightarrow}$ in $(N \cup T)^* \times (N \cup T)^*$ werden definiert durch:

1. $\alpha A \beta \underset{G}{\Longrightarrow} \alpha w \beta$ für alle $\alpha, \beta \in (N \cup T)^*$ und Regeln $A \to w$. ($\alpha w \beta$ ist *direkt ableitbar* aus $\alpha A \beta$)
2. $\alpha A \beta \underset{G,r}{\Longrightarrow} \alpha w \beta$ für alle $\alpha \in (N \cup T)^*, \beta \in T^*$ und Regeln $A \to w$. ($\alpha w \beta$ ist *direkt rechts-ableitbar* aus $\alpha A \beta$)
3. $\alpha A \beta \underset{G,l}{\Longrightarrow} \alpha w \beta$ für alle $\alpha \in T^*, \beta \in (N \cup T)^*$, und Regeln $A \to w$. ($\alpha w \beta$ ist *direkt links-ableitbar* aus $\alpha A \beta$)

Bezeichnet $\overset{+}{\underset{G}{\Longrightarrow}}, \overset{+}{\underset{G,l}{\Longrightarrow}}$ und $\overset{+}{\underset{G,r}{\Longrightarrow}}$ die entsprechenden transitiven Hüllen der Relationen, und $\overset{*}{\underset{G}{\Longrightarrow}}, \overset{*}{\underset{G,l}{\Longrightarrow}}$ und $\overset{*}{\underset{G,r}{\Longrightarrow}}$ die entsprechenden reflexiven-transitiven Hüllen, so heißt ψ *ableitbar (rechtsableitbar, linksableitbar) aus* φ, falls $\varphi \overset{*}{\underset{G}{\Longrightarrow}} \psi$ (bzw. $\varphi \overset{*}{\underset{G,r}{\Longrightarrow}} \psi$, $\varphi \overset{*}{\underset{G,l}{\Longrightarrow}} \psi$) . Ist $\varphi = \sigma$, so heißt ψ *ableitbar (rechtsableitbar, linksableitbar).*

Beispiel 35. Betrachte einen Teil der kontextfreien Grammatik der Aussagenlogik $(\{V, E\}, \{A, B, C, \vee, \wedge, (,), \neg\}, R, E)$, so daß R aus den folgenden Regeln besteht:

$V \to A$

$V \to B$

$V \to C$

$E \to V$

$E \to (E \vee E)$

$E \to (E \wedge E)$

$E \to (\neg E)$

Dann ist zum Beispiel $((\neg A) \vee A)$ aus E ableitbar: $E \Longrightarrow (E \vee E) \Longrightarrow ((\neg E) \vee E) \Longrightarrow ((\neg A) \vee E) \Longrightarrow ((\neg A) \vee A)$.

Definition 102. Sei $G = (N, T, P, \sigma)$ eine CFG. Ein *Ableitungsbaum* B von w (derivation tree, parse tree) ist ein markierter Baum, so daß

1. jeder Knoten mit einem Element aus $\{\varepsilon\} \cup N \cup T$ markiert ist,
2. Die Wurzel mit s markiert ist,
3. innere Knoten mit Elementen aus N markiert sind,
4. wenn ein Knoten die Markierung A hat und seine Söhne die Markierungen $X_1, \ldots, X_k$ (in dieser Reihenfolge), so ist $A \to X_1 \ldots X_k$ eine Regel in P,
5. $\texttt{frontier}(B) = w$

Es gilt:

Satz 103. *Sei $G = (N, T, P, \sigma)$ eine CFG. Ein Wort $w \in (N \cup T)^*$ ist genau dann ableitbar, wenn es einen Ableitungsbaum für w gibt.*

Definition 104. Sei $G = (N, T, P, \sigma)$ eine CFG. Die von G erzeugte *formale Sprache* ist definiert durch: $\mathcal{L}(G) := \{w \mid w \in T^*, \sigma \overset{*}{\underset{G}{\Longrightarrow}} w\}$.

Definition 105. Eine formale Sprache L heißt *kontextfrei*, wenn es eine kontextfreie Grammatik G gibt mit $\mathcal{L}(G) = L$.

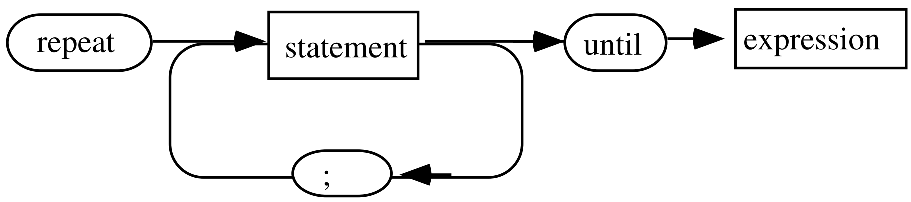
Es gibt kontextfreie Sprachen, die nicht regulär sind. Es gibt auch formale Sprachen, die nicht kontextfrei sind. Ein Beispiel ist $\{a^n b^n c^n \mid n \geq 0\}$.

Die *Backus-Naur-Form (BNF)* ist eine weit verbreitete Methode, die kontextfreie Syntax höherer Programmiersprachen aufzuschreiben. Metasymbole (Symbole die nicht in der Sprache selbst benutzt werden), sind: $\langle, \rangle$, ::= und |, wobei zum Beispiel $\langle \mathit{Zahl} \rangle$ ein Nichtterminalsymbol ist, das mittels ::= definiert wird. Das Zeichen | trennt die verschiedenen Alternativen. Die Regel ⟨Ziffer⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 entspricht 10 verschiedenen Regeln einer CFG. Diese Schreibweise kennt auch Erweiterungen, um eine Syntax kompakter hinschreiben zu können, zum Beispiel, um rekursive Regeln wie ⟨Zahl⟩ ::= ⟨Ziffer⟩ | ⟨Ziffer⟩⟨Zahl⟩ schreibt man manchmal auch ⟨Zahl⟩ ::= $\langle\text{Ziffer}\rangle^+$. Erweitert man die Metasyntax, so kann man mehrere Regeln zusammenfassen, wenn diese Regeln nur deshalb dupliziert worden sind, um optionale Strings auszudrücken, z.B.

⟨Zahl mit Vorzeichen⟩ ::= +⟨Zahl⟩ | ⟨Zahl⟩ | − ⟨Zahl⟩

schreibt man manchmal auch als ⟨Zahl mit Vorzeichen⟩ ::= [+|−]$\langle \mathit{Zahl} \rangle$ wobei [,] Ausdrücke einklammern, die man auch weglassen kann.

Es gibt noch andere Formen, diese Syntax zu beschreiben. z.B. die Syntaxdiagramme, wie sie zur Beschreibung von Pascal verwendet werden. Hier werden Regeln in Form von graphischen Darstellungen verwendet. z.B.



Hier bedeuten repeat in ovalen Kästschen: Zeichenkette als Terminalsymbol, rechteckiges Kästschen: Nichtterminalsymbol. Die Pfeile bezeichnen erlaubte Durchlaufrichtungen. Die BNF hierfür wäre z.B.

⟨repeat-expr⟩	::=	repeat ⟨statements⟩ until ⟨expression⟩
⟨statements⟩	::=	⟨statement⟩ | ⟨statement⟩;⟨statements⟩

Offensichtlich kann eine Programmiersprache mehrere verschiedene CFG's haben. Deshalb muß man vorsichtig sein, wenn man Eigenschaften der Sprachen mit Hilfe von Eigenschaften der Grammatik definiert. Hierzu gehört die Mehrdeutigkeit einer Grammatik. Diese ist gegeben, wenn es zu einem Wort w mehr als eine Linksableitung gibt (d.h. auch mehr als einen Ableitungsbaum). Eine kontextfreie Sprache L heißt mehrdeutig (inhärent mehrdeutig), falls für alle CFG G mit $\mathcal{L}(G) = L$ gilt, daß G mehrdeutig ist. Hier besteht ein Zusammenhang mit der Frage, ob (und wie) man die Bedeutung von Worten einer formalen Sprache anhand einer CFG eindeutig festlegen kann, siehe auch Kapitel 13.

Pragmatisch gilt, daß die zulässigen Programme realer Programmiersprachen im allgemeinen nicht durch eine CFG erzeugt werden, denn bestimmte Programme werden aufgrund von Kontextbedingungen ausgeschlossen. Z.B. müssen Deklarationen oft vor der Benutzung von Variablen gemacht werden, oder Ausdrücke müssen getypt sein (in Gofer). D.h. die CFG ist i.a. nur ein erster Schritt in der Definition der zulässigen Programme.

13 Semantik von formalen Sprachen, interpretierte Systeme

Bisher haben wir formale Sprachen als Mengen uninterpretierter Wörter eingeführt, können aber noch nicht von einer Bedeutung reden. Zum Beispiel kann man nicht so ohne weiteres sagen, daß $\{0,1,2,3,4,5,6,7,8,9\}^*$ die Menge der natürlichen Zahlen $\mathbb{N}_0$ ist. In Programmiersprachen wird ja auch ein Unterschied zwischen dem String "128" und der Zahl 128 gemacht. Die Bedeutung oder die Interpretation der Zeichen schreibt man normalerweise getrennt zur Definition der Sprache hin. Eine solche Bedeutung nennt man auch Semantik. Eine Interpretation ist nichts anderes als eine Abbildung der Wörter auf ihren Wert (Bedeutung). Deswegen definieren wir:

Definition 106. Ein *interpretiertes System* besteht aus 3 Komponenten: (L, D, δ). L ist eine Menge von syntaktischen Objekten, D ist eine Menge von Resultaten, und $\delta : L \to D$ eine partielle Abbildung, die syntaktischen Objekten einen Wert aus D zuordnet. Im allgemeinen macht man die Funktion δ total, indem man das interpretierte System $(L, D \cup \{\perp\}, \delta')$ benutzt, und $\delta'(p) = \perp$ setzt, wenn δ für p undefiniert ist. Man bezeichnet L als die *Syntax*, und δ als die *Semantik*.

Beispiel 36. Berechnungen als interpretierte Systeme: Nimmt man L als Menge von Paaren (P, I), wobei P ein Programm ist, I die Eingabe, und $D :=$ Menge der möglichen Resultate, dann definiert man $\delta(P, I) := R$, falls R das Ergebnis der Berechnung ist bei der P mit der Eingabe I versorgt wird. Im Fall einer nicht-terminierenden Berechnung ist $\delta(P, I)$ undefiniert, und man definiert dann $\delta(P, I) := \perp$

Definition 107. Sei $\Phi := (L, D, \delta)$ ein interpretiertes System. Zwei Objekte $p_1, p_2 \in L$ heissen *semantisch äquivalent*, gdw. $\delta(p_1) = \delta(p_2)$.

Bei der Veränderung bzw. Optimierung von Programmen ist es wesentlich, daß die semantische Äquivalenz erhalten bleibt. Allerdings ist die Frage der semantischen Äquivalenz im allgemeinen unentscheidbar, denn anderenfalls könnte man das Halteproblem lösen.

Beispiel 37. Die Menge der Objekte sei $\{a, b\}^*$. Die Interpretation eines Wortes sei definiert durch

- $I(\varepsilon) := 0$
- $I(a) := 0$
- $I(b) := 1$

- $I(uv) := 2 * I(u) + I(v)$ für $u \in L$ und $v \in \{a, b\}$.

Man sieht: dies ist gerade die Interpretation von Strings über $\{a, b\}$ als binäre Zahlen, wobei a als 0 und b als 1 interpretiert wird: $I(aab) = 1, I(baaa) = 8$.

Die Semantik der Aussagenlogik läßt sich hier einordnen: Seien $X_1, \ldots, X_n$ die vorkommenden Aussagenvariablen. Wir betrachten als Resultatmenge die Menge der Funktionen $f : \{T, F\}^n \to \{T, F\}$, wobei $\{T, F\}$ eine Boolesche Algebra mit den Operatoren $\wedge$, $\vee$, $\neg$ ist. Dann definiert man

$\delta(\mathcal{F}) := \lambda X_1, \ldots, X_n.\mathcal{F})$

Tautologien sind dann solche Aussageformen, die die konstante Funktion $\lambda X_1, \ldots, X_n.T$ als Semantik haben. Alle Umformungen, die die Semantik erhalten, sind zulässig. Wenn man Transformationen auf Aussageformen definiert, so kann (und muß) man die Korrektheit dieser Transformationen anhand der Semantik überprüfen: Transformiert man $F \to G$, so bleibt die semantische Äquivalenz erhalten, wenn für alle Interpretationen I gilt: $I(F) = I(G)$. Dies gilt zum Beispiel für die Transformation $(A \vee T) \to T$, wie man leicht nachprüft.

Die Semantik der Prädikatenlogik erster Stufe läßt sich ebenfalls hier einordnen, ist aber etwas komplizierter (siehe Kapitel 2).

Alle Transformationen auf der syntaktischen Ebene müssen, wenn sie korrekt sein sollen, diese Semantik erhalten. D.h. für eine Transformation $G_1 \to G_2$ muß gewährleistet sein, daß $I(G_1) = I(G_2)$ für alle Interpretationen I gilt. Die erweiterte Ausdruckskraft der Prädikatenlogik gegenüber der Aussagenlogik hat einen Preis: die Überprüfung, ob eine Aussage eine Tautologie ist, erfordert nach obiger Definition i.a. die Überprüfung von unendliche vielen verschiedenen Interpretationen. Es gibt für die Prädikatenlogik Kalküle, die feststellen können, ob eine Aussage eine Tautologie ist. Allerdings kann es passieren, daß diese Kalküle unendlich lange weiterlaufen, falls die zu testende Formel keine Tautologie ist.

Bemerkung. Zur Semantik des λ-Kalküls. Die intuitive Vorstellung, daß der λ-Kalkül eine Semantik hat, wobei die Resultatmenge eine Menge von Elementen, Funktion, Funktionen von Funktionen ... ist, führt bei einer naiven Definition auf Widersprüche (ähnlich zum Russelschen Paradoxon), denn die Resultatsmenge R müßte dann auch alle Funktionen von $R \to R$ enthalten. Bekanntlich ist die Kardinalität von $R \to R$ echt größer als die Kardinalität von R. Eine genauere (umfangreiche) Betrachtung mit verbandstheoretischen Methoden zeigt, daß man doch "intuitive" Modelle definieren kann, wobei nicht alle Funktionen aus $R \to R$ gebraucht werden.

Übungsaufgabe 108.

1. Verifizieren Sie, daß die Transformationsregel $\neg\forall x.G \to \exists x.\neg G$ korrekt ist .
2. Zeige, daß $\forall x.P(x) \vee \neg P(x)$ eine Tautologie ist.

14 Notationen von Ausdrücken, Terme

In arithmetischen Ausdrücken, in logischen Aussagen, und in Programmiersprachen verwendet man Ausdrücke, die einen Wert besitzen. Charakteristisch ist, daß man in den Ausdrücken zwischen Operatoren und Argumenten unterscheidet.

Beispiel 38. $1 * 2 + 3 - 4$ ist ein arithmetischer Ausdruck mit dem Wert 1. $(T \vee \neg F) \wedge T$ ist ein logischer Ausdruck mit dem Wert T. $(*\ 1\ 2\ 3\ 4)$ ist ein Lisp-Ausdruck mit dem Wert 24.

Bei diesen Ausdrücken ist zu unterscheiden zwischen ihrer Notation und ihrem Wert, also zwischen der Syntax und der Semantik.

Das wesentliche Ziel bei der Erfindung und Benutzung von Notationen ist es zu erreichen, daß ein Ausdruck einen eindeutigen Wert hat. Andere Kriterien und Zwecke, die eine wesentliche und teilweise konkurrierende Rolle spielen sind: die Notation muß gut leserlich sein und sollte möglichst wenig überflüssige Symbole enthalten. Eine Forderung kann auch sein, daß sie gut von einem Rechner verarbeitet werden kann. Schwierigkeiten die dabei entstehen sind: i) teilweise werden verschiedene Operatoren gleich bezeichnet, z.B. das minus-Zeichen in $-1 - 2$, oder ii) ein Operator hat variable Stelligkeit: $(*\ 1\ 2\ 3\ 4)$, oder iii) Operatoren nehmen verschiedene Positionen zu ihren Argumenten ein: $(+\ 1\ 2)$ oder $1 + 2$, oder $(\texttt{IF}\ A\ \texttt{THEN}\ B\ \texttt{ELSE}\ C)$ bzw. $\texttt{IFTHENELSE}(A, B, C)$

Definition 109. Eine *Signatur* ist ein Paar $(F, \texttt{arity})$, wobei F eine endliche Menge von Funktionssymbolen und $\texttt{arity}$ eine Funktion $\texttt{arity} : F \to \mathbb{N}_0$, die *Stelligkeit* des Symbols, ist. Funktionssymbole f mit $\texttt{arity}(f) = 0$ bezeichnet man auch als *Konstantensymbole*.

Ein *Term* über der Signatur ist ein Wort aus der Sprache $(F \cup$ "(", ")", ",")* , also F angereichert um rechte Klammer, linke Klammer und Komma, wobei Terme folgendermaßen rekursiv definiert werden:

1. ein Konstantensymbol ist ein Term
2. $f(t_1, \ldots, t_n)$ ist ein Term, wobei f ein Funktionssymbol mit $\texttt{arity}(f) = n$ ist, und $t_1, \ldots, t_n$ Terme sind.

Ein Beispiel für eine Signatur und die zugehörigen Terme ist die Menge der Funktionssymbole und Terme der Prädikatenlogik.

Definition 110. Eine (universelle, oder allgemeine) *Algebra* über der Signatur F ist ein Tripel (A, O, φ), wobei $A \neq \emptyset$ eine Menge ist, O eine Menge von Funktionen, φ eine Bijektion von $O \to F$ (wir schreiben $\varphi(f)$ als f_A) so daß für jedes $f \in F$ mit $\texttt{arity}(f) = n$, f_A eine n-stellige Funktion ist: $f_A : A^n \to A$.

Offensichtlich gilt für eine Algebra (A, O, φ): Jedem Term kann man einen eindeutigen Wert in A zuweisen durch:

$\varphi(f(t_1, \ldots, t_n)) := f_A(\varphi t_1, \ldots, \varphi t_n)$

Damit ist für eine Algebra über der Signatur F das System $(\mathcal{T}, A, \varphi)$ ein interpretiertes System, wobei $\mathcal{T}$ die Menge der Terme ist.

Verschiedene Notationen werden im allgemeinen dadurch eindeutig, daß man eine Abbildung der Notation auf Terme angibt, und dann erst einen Wert berechnet. Z.B könnte man $-1-2$ eindeutig auf den Term $\mathtt{SUBTRACT}(\mathtt{NEGATE}(1), 2)$ abbilden.

Definition 111. Notationen

1. Die Notation, die in Definition 109 definiert wird, ist die *Standardnotation.* Terme werden notiert in der Form $f(t_1, \ldots, t_n)$, wobei zu beachten ist, daß f ein Symbol sein muß (d.h. kein Ausdruck sein darf) .

2. Die Notation, die in Lisp verwendet wird, ist die *Präfixnotation.* Hier wird statt $f(t_1, \ldots, t_n)$ das Wort $(f\ t_1 \ldots\ t_n)$ verwendet. Der Grund liegt darin, daß es auch erlaubt ist Terme für Operatoren einzusetzen und Operatoren an Argumentstellen zu verwenden, und Operatoren variabler Stelligkeit zu benutzen, und somit eine gemischte Notation nicht so ohne weiteres eindeutig wird.

3. Eine Abwandlung der *Präfixnotation* ist die *polnische Notation*, (Lukasiewicz-Notation, polnische Präfixnotation), bei der im Gegensatz zur normalen Präfixnotation einfach alle Klammern weggelassen werden. Dies ist nur dann eindeutig möglich, wenn alle Operatoren Symbole sind, die Operatorensymbole eindeutig sind, und eine feste Stelligkeit haben.

4. Die *Standardpostfixnotation.* Hier wird statt $f(t_1, \ldots, t_n)$ das Wort $(t_1, \ldots, t_n)f$ verwendet. Diese Notation wird in der Mathematik manchmal verwandt, wenn man Homomorphismen rechts schreibt. Wir haben diese Notation benützt für die Definition der Substitution und der Interpretation.

5. Die *Postfixnotation* (polnische Postfixnotation, reverse polish). Hier werden in der Standardpostfixnotation einfach alle Klammern weggelassen. Es gelten die gleichen Bemerkungen wie zur Präfixnotation. Diese Notation hat eine Bedeutung in der Informatik, denn der Auswertungsalgorithmus für Terme läßt sich direkt mittels eines Kellers implementieren. In der Mathematik wird diese Notation beispielsweise für die Fakultätsfunktion verwendet $3! = 6$.

6. Die *Infixnotation.* Diese wird im allgemeinen nur für zweistellige Operatoren verwendet. Der Operator wird dann zwischen die Argumente geschrieben. z.B. $(1 + 2)$. Auch hier gilt, daß die Operatoren Symbole sein sollten, damit eine eindeutige Auswertung möglich ist.

7. Will man Notationsformen mischen und/oder möglichst wenige Klammern benutzen, und die Ausdrücke eindeutig lassen, so braucht man Konventionen und Prioritäten

von Operatoren. Zum Beispiel ist der Ausdruck $(1 + 2 * 3 + 4!)$ äquivalent zum Term $((1 + (2 * 3)) + (4!))$. Hier hat man für die Prioritäten: $! > * > +$. Der Term $(1+2-3)$ wird als $((1 + 2) - 3)$ interpretiert nach der Konvention links vor rechts.

Es gibt noch andere Notationsformen, die wir hier nicht betrachten, zum Beispiel graphische, wie m^n, f_A, oder $\bar{(}N)$.

Übungsaufgabe 112. Programmiere: Wandle arithmetische Ausdrücke mit den Operatoren $+, -, *$ von Infixnotation in Postfixnotation um und werte sie direkt mittels eines Kellers aus.