4 Prädikatenlogik (PL_1) und Resolution

Prädikatenlogik (PL) ist eine ausdrucksstarke Logik, die im Prinzip für sehr viele Anwendungen ausreicht.

Man unterscheidet verschiedene Stufen der Prädikatenlogik. Prädikatenlogik 0.Stufe (PL_0) ist die Aussagenlogik. Sie erlaubt keine Quantifikationen. Prädikatenlogik erster Stufe (PL_1) dagegen erlaubt schon Quantifikationen über die Elemente einer Trägermenge. Prädikatenlogik 2.Stufe (PL_2) erlaubt darüberhinaus noch unabhängig Quantifikationen über die Funktionen und Relationen über diese Trägermenge. Man kann also z.B. in PL_2 hinschreiben "$\forall x : \exists f : \forall P : P(f(x, f))$", was in PL_1 nicht geht. Prädikatenlogik noch höherer Stufe erlaubt Quantifizierungen über die Funktionen und Relationen über der Funktions- und Relationsmenge usw. [EFT86].

Aus praktischer Sicht gibt es viele Zusammenhänge, die sich in anderen Logiken wesentlich eleganter und einfacher formulieren lassen als in PL_n. Kurt Gödel hat gezeigt, daß es im Gegensatz zu PL_1 für PL_n, $n \geq 2$, keinen vollständigen Kalkül mehr geben kann [Göd31]. D.h. es gibt Aussagen in PL_2, die in allen Interpretationen gültig sind, was aber mit keinem durch Symbolmanipulation arbeitenden Verfahren mehr nachgewiesen werden kann. Das heißt nicht, daß man für PL_2 keine Deduktionssysteme entwickeln kann – sie können eben nur nicht alle gültigen Theoreme beweisen; aber das können heutige Deduktionssysteme für PL_1 auf heutigen Computern auch nur theoretisch. Deduktionssysteme für PL_2 sind aber um einiges komplizierter als solche für PL_1 [And81,RB79]. Wir werden in der Vorlesung auch Kalküle für Logiken höherer Ordnung betrachten, falls die Zeit ausreicht.

4.1 Syntax der Prädikatenlogik erster Stufe

PL_1 ist zunächst mal eine formale Sprache, deren Syntax sich durch eine entsprechende Grammatik angeben läßt. Im Gegensatz zu vielen Programmiersprachen z.B. ist die Grammatik von PL_1 aber extrem einfach. Wie für die meisten Logiken besteht die Syntaxbeschreibung aus den drei Komponenten:

– Signatur
– Bildungsregeln für Terme
– Bildungsregeln für Formeln

Die *Signatur* gibt das Alphabet an, aus dem die zusammengesetzten Objekte bestehen. Man unterscheidet *Funktions-* und *Prädikatensymbole*. Neben diesen Symbolen gibt es noch unendliche viele *Variablensymbole*, die nicht zur Signatur gerechnet werden. In einer Formel "$\forall x : \exists y : P(x, f(y))$" sind x und y Variablensymbole, f ist ein einstelliges Funktionssymbol und P ein zweistelliges Prädikatensymbol. Die logischen Junktoren und Quantoren sind fest und zählen daher nicht zu der Signatur. Aus den Variablen- und Funktionssymbolen lassen sich *Terme* aufbauen ($f(y)$ ist zum Beispiel ein Term) und damit und mit den Prädikatensymbolen *Atome, Literale* und *Formeln*. Der Unterschied zwischen

Termen und Formeln ist in erster Linie semantischer Natur. Terme bezeichnen Objekte einer Trägermenge und Formeln bezeichnen Wahrheitswerte. Die formale Definition ist:

Definition 4.1. *(Syntax von PL_1)*

Signatur *: $\Sigma = (\mathcal{F}, \mathcal{P})$, wobei*

- *$\mathcal{F}$ ist die Menge der Funktionssymbole*
- *$\mathcal{P}$ ist die Menge der Prädikatensymbole*

Diese Mengen sind disjunkt. Daneben braucht man noch die Menge V der Variablensymbole (abzählbar unendlich viele). Diese Menge ist ebenfalls disjunkt zu $\mathcal{F}$ und $\mathcal{P}$.

Jedem Funktions- und Prädikatensymbol $f \in \Sigma$ ist eindeutig eine Stelligkeit zugeordnet Stelligkeit : $arity(f) \geq 0$. Funktionssymbole mit der Stelligkeit 0 bezeichnet man auch als Konstantensymbole. $\{f \in \mathcal{F} \mid arity(f) = 0\}$ = Konstantensymbole. Es muß mindestens ein Konstantensymbol in F vorhanden sein!

Terme *Die Menge der* Terme *$T(\Sigma, V)$ über der Signatur $\Sigma = (\mathcal{F}, \mathcal{P})$ und den Variablen V wird induktiv als die kleinste Menge definiert, die folgendes erfüllt:*

- *$V \subseteq T(\Sigma, V)$*
- *falls $f \in \mathcal{F}, arity(f) = n, t_1, \ldots t_n \in T(\Sigma, V)$ dann $f(t_1, \ldots t_n) \in T(\Sigma, V)$. Hierbei ist $f(t_1, \ldots t_n)$ zu lesen als Zeichenkette bzw. als ein Baum.*

Formeln *Die Menge der* Formeln *F_Σ über der Signatur $\Sigma = (\mathcal{F}, \mathcal{P})$ und den Variablen V wird induktiv als die kleinste Menge definiert, die folgendes erfüllt:*

- *falls $P \in \mathcal{P}, arity(P) = n, t_1, \ldots, t_n \in T_\Sigma$ dann $P(t_1, \ldots t_n) \in F_\Sigma$ (*Atom*) . Auch hier ist $P(t_1, \ldots t_n)$ als Zeichenkette zu lesen.*
- *falls $F, G \in F_\Sigma, x \in V$, dann auch: $(\neg F), (F \vee G), (F \wedge G), (F \Rightarrow G), (F \Leftrightarrow G), (\forall x : F)$ und $(\exists x : F) \in F_\Sigma$.*

Wir machen bei den Schreibweisen einige Vereinfachungen: Geschachtelte gleiche Quantoren schreiben wir als Quantor über mehreren Variablen: $\forall x; \forall y : F$ wird als $\forall x, y : F$ geschrieben. In Formeln werden zum Teil Klammern weggelassen, wenn die Eindeutigkeit gewährleistet bleibt, mit den üblichen Prioritätsregeln. Weiterhin erlauben wir auch als Formelkonstanten die Formeln `false` und `true`.

Beispiel 4.2. Signatur $\Sigma := (\{a, b, f, g\}, \{P, Q, R\})$ mit $arity(a) = arity(b) = arity(P) = 0$, $arity(f) = arity(Q) = 1$ $arity(g) = arity(R) = 2$.

$V = \{x, y, z, \ldots\}$

$T_\Sigma =$	$F_\Sigma =$
	$\{P, Q(a), Q(b), Q(x), \ldots, R(a,a), \ldots R(a,b), \ldots$
$f(a), f(b), f(x), \ldots$	$\neg P, \neg Q(a), \ldots$
$g(a,a), g(a,b), g(a, f(a)), \ldots$	$P \wedge Q(a), P \wedge \neg Q(a), \ldots$
$f(f(a)), f(f(b)), \ldots f(g(a,a)), \ldots$	$P \vee Q(a), P \vee \neg Q(a), \ldots$
$g(f(f(a)), a), \ldots$	$P \Rightarrow Q(a), P \Leftrightarrow \neg Q(a), \ldots$
$\ldots$	$\forall x : \; Q(x), \ldots$
	$\exists x : \; R(x,y) \Rightarrow Q(x), \ldots$

Mit der Forderung, daß mindestens ein Konstantensymbol vorhanden sein muß, ist implizit verbunden, daß die Trägermengen, über die in einer jeweiligen Interpretation quantifiziert wird, nicht leer sein kann – es muß mindestens ein Element vorhanden sein, auf das dieses Konstantensymbol abgebildet wird. Damit verhindert man, daß Quantifizierungen der Art "$\forall x : \; (P \wedge \neg P)$" wahr gemacht werden können; indem die Menge, über die quantifiziert wird, leer ist. Einige der logischen Verknüpfungen, wie z.B. $\Rightarrow$ und $\Leftrightarrow$ sind redundant. Sie können durch die anderen dargestellt werden. Bei der Herstellung der Klauselnormalform (Abschnitt 4.3) werden sie dann auch konsequenterweise wieder eliminiert. Nichtsdestotrotz erleichtern sie die Lesbarkeit von Formeln beträchtlich und sind daher mit eingeführt worden.

Definition 4.3. Konventionen:

– *Da* 0*-stellige Funktionssymbole als Konstantensymbole dienen*[3]*, schreibt man im allgemeinen nicht "*$a()$*" sondern einfach nur* a
– *Variablen sind genau einem Quantor zugeordnet Insbesondere gelten ähnlich wie in den meisten Programmiersprachen die schon gewohnten Bereichsregeln (lexical scoping) für Variable. D. h. Formeln der Art* $\forall x : \; \exists x : \; P(x)$ *haben nicht die vermutete Bedeutung, nämlich daß für alle* x *das gleiche* x *existiert, sondern* x *ist gebunden in* $\exists x : \; P(x)$ *und daß äußere* x *in alle* $\forall x :$ *kann das innere* x *nicht beeinflussen. D.h. die Formel* $\forall x : \exists x : \; P(x)$ *ist zu* $\exists x : \; P(x)$ *äquivalent. Da man unendlich viele Variablensymbole zur Verfügung hat, kann man in jedem Fall für jeden Quantor ein anderes Variablensymbol wählen.*
– *Meist haben die logischen Verknüpfungssymbole die Bindungsordnung* $\Rightarrow, \Leftrightarrow, \wedge, \vee, \neg$ *, d.h.* $\Rightarrow$ *bindet am schwächsten und* $\neg$ *am stärksten. Danach gilt eine Formel* $\neg A \wedge B \vee C \Rightarrow D \Leftrightarrow E \wedge F$ *als folgendermaßen strukturiert:* $((\neg A) \wedge (B \vee C)) \Rightarrow (D \Leftrightarrow (E \wedge F))$*. Quantoren binden, soweit die quantifizierte Variable vorkommt. D.h.* $\forall x : \; P(x) \wedge Q$ *steht für* $(\forall x : \; P(x)) \wedge Q$ *während* $\forall x : \; P(x) \wedge R(x)$ *für* $(\forall x : \; (P(x) \wedge R(x)))$ *steht. Um Zweifel auszuschließen, werden aber meist die Klammern explizit angegeben.*
– *Im folgenden wird die allgemein übliche Konvention für die Benutzung des Alphabets verwendet: Buchstaben am Ende des Alphabets, d.h.* u, v, w, x, y, z

[3] Indem man Konstantensymbole nicht extra ausweist, spart man sich in vielen Fallunterscheidungen eben diesen speziellen Fall.

bezeichnen Variablensymbole. Buchstaben am Anfang des Alphabets, d.h. a, b, c, d, e bezeichnen Konstantensymbole. Die Buchstaben f, g, h werden für Funktionssymbole benutzt. Die großen Buchstaben P, Q, R, T werden für Prädikatensymbole benutzt.

Die Syntax der Terme und Formeln wurde induktiv definiert: Aus den einfachen Objekten, Variable im Fall von Termen und Atome im Fall von Formeln wurden mit Hilfe von Konstruktionsvorschriften die komplexeren Objekte aufgebaut. Damit hat man eine Datenstruktur, die eine Syntaxbaum hat, der mit verschiedenen Konstruktoren aufgebaut wurde. Definitionen, Algorithmen, Argumentation und Beweise müssen nun jeweils rekursiv (induktiv) gemacht werden, wobei man stets Fallunterscheidung und Induktion *über die Struktur der Formeln und Terme* machen muß

Folgende Definition (der freien Variablen) demonstriert eine Anwendung des rekursiven Definitionsschemas für Terme und Formeln. Eine Variable wird als frei bezeichnet wenn sie sich nicht im Bindungsbereich (Skopus) eines Quantors befindet.

Definition 4.4. *(Freie Variablen für Terme und Formeln) Die Operation $FV(.)$ sammelt alle nicht durch Quantoren gebundenen Variablensymbole in Termen und Formeln auf.*

Definition von FV für Terme:

$$FV(t) = \begin{cases} t & \textit{falls } t \textit{ eine Variable ist} \quad \textit{(Basisfall)} \\ FV(t_1) \cup \ldots \cup FV(t_n) & \textit{falls } t = f(t_1, \ldots t_n) \quad \textit{(Rekursionsfall)} \end{cases}$$

Definition von FV für Formeln: *Basisfall: $H = P(t_1, \ldots t_n)$ ist ein Atom, dann: $FV(H) = FV(t_1) \cup \ldots \cup FV(t_n)$ (FV ist für Terme schon definiert.)*
Rekursionsfälle: Fallunterscheidung nach der Struktur von H:
Fall: $H = \neg F$ dann $FV(H) := FV(F)$
Fall: $H = F \vee G$ dann $FV(H) = FV(F) \cup FV(G)$
Fall: $H = F \wedge G$ dann $FV(H) = FV(F) \cup FV(G)$
Fall: $H = F \Rightarrow G$ dann $FV(H) = FV(F) \cup FV(G)$
Fall: $H = F \Leftrightarrow G$ dann $FV(H) = FV(F) \cup FV(G)$
Fall: $H = \forall x: F$ dann $FV(H) = FV(F) \setminus \{x\}$
Fall: $H = \exists x: F$ dann $FV(H) = FV(F) \setminus \{x\}$

Beispiel 4.5. Unten sind x, y, z Variablensymbole

- $FV(x) = FV(f(x)) = FV(g(x, g(x, a))) = \{x\}$
- $FV(P(x) \wedge Q(y)) = \{x, y\}$
- $FV(\exists x : R(x, y)) = \{y\}$.

Zum Schluß des Abschnitts über die Syntax von PL_1 werden noch einige Sprechweisen und Begriffe, die z.T. auch schon benutzt wurden, eingeführt. Da sie sehr häufig gebraucht werden, muß man sie sich unbedingt einprägen.

Definition 4.6. *Einige übliche Sprechweisen:*

Atom:	*Eine Formel der Art* $P(t_1, \ldots t_n)$ *wobei* P *ein Prädikatensymbol und* $t_1, \ldots, t_n$ *Terme sind heißt Atom.*
Literal:	*Ein Atom oder ein negiertes Atom heißt Literal. (Beispiele:* $P(a)$ *und* $\neg P(a)$*)*
Grundterm:	*Ein Term* t *ohne Variablensymbole, d.h.* $FV(t) = \emptyset$*, heißt Grundterm (engl. ground term).*
Grundatom:	*Ein Atom* F *ohne Variablensymbole, d.h.* $FV(F) = \emptyset$*, heißt Grundatom.*
geschlossene Formel:	*Eine Formel* F *ohne freie Variablensymbole, d.h.* $FV(F) = \emptyset$ *heißt geschlossen.*
Klausel	*Formel mit einem Quantorpräfix nur aus Allquantoren besteht . d.h* $F = \forall^n.F'$ *und* F' *ist eine Disjunktionen von Literalen.*

Beispiel 4.7. Für eine geschlossene Formel: $\forall x : \exists y : \; P(x, y)$. Nicht geschlossen ist: $\exists y : P(x, y)$, da $FV(\exists y : \; P(x, y)) = \{y\})$.

4.2 Semantik von PL_1 (nach Tarski)

Durch die formale Definition der Syntax von PL_1 ist man jetzt zwar in der Lage, Aussagen als Formeln hinzuschreiben und zu überprüfen ob das was man hingeschrieben hat auch syntaktisch in Ordnung (wohlgeformt) ist. Man kann jetzt auch Operationen zur Manipulation von Termen und Formeln definieren. Was diese Formeln, die ja bisher nur reine Zeichenketten sind, und die Operationen, die nur Zeichenketten manipulieren, aber bedeuten sollen ist bisher nur intuitiv angedeutet worden. Um aber eine Zeichenkettenmanipulation als korrekte Ableitungsregel zu interpretieren braucht man einen Formalismus, der Termen und Formeln Bedeutung zuordnet, so daß man damit eine semantische Folgerungsbeziehung definieren kann. Dies ist eine Verallgemeinerung der entsprechenden Konzepte in der Aussagenlogik.

Es gibt auch eine Analogie zu Programmiersprachen. Die reine Syntaxdefinition in einer Backus-Naur Form zum Beispiel hilft zu entscheiden, ob ein gegebenes Programm syntaktisch korrekt ist. Um aber einen Interpretierer oder Compiler für die Sprache zu schreiben und insbesondere, um dessen Korrektheit nachzuweisen, muß man sagen, was die Konstrukte, die in der Sprache auftreten, bedeuten sollen, d.h. man braucht eine formale Semantik. Dafür gibt es verschiedene Möglichkeiten. Die abstrakteste ist wohl die denotationale Semantik, bei der die Programmiersprachenkonstrukte im wesentlichen auf Mengenoperationen abgebildet werden. Diese Operationen sind dann einfach und übersichtlich zu erklären. Hiermit hat man eine Möglichkeit, die Ausführung von Programmkonstruktionen zu definieren bzw. zu überprüfen.

Um die Semantik einer Logik zu definieren gibt es ebenfalls verschiedene Möglichkeiten.

Eine operationale Methode der Definition einer Logik ist ein sogenannter Hilbertkalkül. Dabei gibt man zunächst einen Satz von Formeln oder Formelschemata an, die man apriori als gültige Aussagen (Axiome) ansehen will (z.B.

$P \Rightarrow P$). Weiterhin braucht man einen Satz von (syntaktischen) Ableitungsregeln, die aus den Axiomen neue Formeln generieren, welche man dann ebenfalls als gültige Formeln annimmt. Man muß nur aufpassen, daß diese Ableitungsregeln nicht gleichzeitig eine Formel und deren Negation generieren (Korrektheit). Ein solcher Hilbertkalkül charakterisiert eine Logik zwar genau, ist aber i.a. zu indirekt. Eine Charakterisierung durch einen Hilbertkalkül eignet sich nicht, um andere, für die Automatisierung besser geeignete Kalküle, zu entwickeln.

Eine sehr natürliche Möglichkeit (die "denotationale Semantik" von PL_1), die Semantik einer Logik und insbesondere von PL_1 anzugeben ist von Alfred Tarski entwickelt worden [Tar53]. Die Grundidee dabei ist, eine Abbildung der syntaktischen Objekte auf mathematische Objekte (Mengen, Funktionen, Relationen etc.) anzugeben, so daß man unter dieser Abbildung Formeln zu Wahrheitswerten evaluieren kann. Gesucht sind daher:

- geeignete mathematische Objekte
- eine geeignete Abbildungsvorschrift, wobei die Abbildung in drei Schritten definiert wird:
  1. Abbildung der Signatur (für PL_1 auf Funktionen und Relationen)
  2. Abbildung der Terme auf Elemente einer Grundmenge Trägermenge, Universum)
  3. Abbildung der Formeln auf Wahrheitswerte.

Für PL_1 wird diese Abbildung dann so aufgebaut, daß man zunächst eine Menge definiert, in die die Terme abgebildet werden sollen; Trägermenge, Domain oder Universum genannt. Funktionssymbole werden auf Funktionen über dieser Menge und Prädikatensymbole auf Relationen über dieser Menge abgebildet. Damit ist die Grundstruktur festgelegt. Die Abbildung von Termen und Formeln ergibt sich jetzt ganz natürlich aus ihrer syntaktischen Struktur. Ein Term $f(a, b)$ zum Beispiel bezeichnet gerade eine Funktionsanwendung: Die dem Symbol f zugeordnete Funktion wird auf die den Konstantensymbolen a und b zugeordneten Werte angewendet. Dem Term selbst wird gerade das Ergebnis dieser Funktionsanwendung zugeordnet. Wenn z.B. f die arithmetische Funktion $+$ zugeordnet wird und a und b die Zahlen 3 und 4, dann wird damit automatisch dem Term $f(a, b)$ der Wert 7 zugeordnet. Mit Hilfe der Abbildung von Prädikatensymbolen auf Relationen lassen sich Atome auf Wahrheitswerte abbilden. Wenn z.B. das Prädikatensymbol P auf die arithmetische $<$-Relation abgebildet werden und wie oben die Konstantensymbole a und b auf die Zahlen 3 und 4, dann ist $P(a, b)$ wahr, da ja $3 < 4$ wahr ist. Wenn man dagegen a und b umgekehrt auf 4 und 3 abbildet, dann ist $P(a, b)$ falsch. Ausgehend von dieser Grundvorschrift für Atome lassen sich dann zusammengesetzte Formeln aus den Werten für ihre Komponenten und der Bedeutung des obersten logischen Symbols auswerten. Wenn z.B. die Formel F zu falsch ausgewertet wird, dann wird $\neg F$ zu wahr ausgewertet usw.

Man beachte, daß die syntaktischen Objekte, wie z.B. der Term $f(a, b)$, nichts weiter sind als Zeichenketten. In einem Computer werden sie als Bitfolgen repräsentiert, die für den Computer keinerlei Bedeutung haben. Im Gegensatz dazu ist mit $+$ wirklich die arithmetische Additionsfunktion gemeint. Wenn man auf Papier einen Unterschied machen will zwischen der Darstellung von f als Funktionssymbol und z.B. $+$ als Funktion, dann muß man zunächst Konventionen zur Schreibweise einführen, um Datenstrukturobjekte und mathematische Dinge auseinander zu halten. Es sind aber völlig unterschiedliche Dinge damit gemeint. In rein mathematischen Texten sind mit den Zeichen, die auf dem Papier innerhalb von Formeln erscheinen im allgemeinen die semantischen Objekte, d.h. die Funktionen usw. gemeint. In logischen Texten wie auch in diesem Skript sind es dagegen die syntaktischen Objekte, um die es primär geht.

Die Idee für die Semantik von PL_1 ist offensichtlich sehr einfach und natürlich. Der Formalismus ist umfangreich aber einfach und gibt ganz genau die oben geschilderten Prinzipien wieder.

Für eine Signatur $\Sigma = (\mathcal{F}, \mathcal{P})$, die nach Definition 4.1 aus Funktions und Prädikatensymbolen besteht, definiert man zunächst *Σ-Algebren* [4] bestehend aus einer Trägermenge und einer geeigneten Anzahl von Funktionen geeigneter Stelligkeit. Für jedes in $\mathcal{F}$ vorkommende Funktionssymbol mit Stelligkeit n braucht man genau eine n-stellige Funktion. Um auch die Prädikatensymbole zu interpretieren, werden die Σ-Algebren zu Σ-Strukturen erweitert, die für jedes in Σ vorkommende Prädikatensymbol eine passende Relation enthalten. (Den Unterschied zwischen Σ-Algebren und Σ-Strukturen macht man deshalb, weil Σ-Algebren zur Interpretation von Termen ausreichen, und Σ-Algebren für Gleichheitslogiken und für die Unifikation ausreichen.) Schematisch sieht die Abbildung der Signatur dann so aus:

$$\Sigma\text{-Struktur: } \begin{cases} \Sigma\text{-Algebra : } \{ \text{ Funktionssymbole } \to \text{Funktionen} \\ \qquad\qquad\qquad\;\; \text{Prädikatensymbole} \to \text{Relationen} \end{cases}$$

Definition 4.8. Σ-Algebra, Σ-Struktur)
Sei $\Sigma = (\mathcal{F}, \mathcal{P})$ eine Signatur. $A = (D_A, F_A)$ heißt eine Σ-Algebra gdw.

- *D_A ist eine nichtleere Menge (die Trägermenge von A)*
- *F_A enthält für jedes Funktionssymbol in $\mathcal{F}$ eine passende* totale *Funktion , d.h. $F_A = \{f_A \in (D_A)^n \to D_A \mid f \in \mathcal{F}, arity(f) = n\}$.*

*$S = (D_S, F_S, \mathcal{P}_S)$ heißt eine Σ-*Struktur *gdw.*

- *(D_S, F_S) ist eine Σ-Algebra und*
- *P_S enthält für jedes Prädikatensymbol in $\mathcal{P}$ mit Stelligkeit ≥ 1 eine passende Relation, d.h. $\mathcal{P}_S = \{P_S \subseteq (D_A)^n \mid P \in \mathcal{P}, arity(P) = n\}$.*
- *Jedes Prädikatensymbol der Stelligkeit 0 wird auf 0 oder 1 abgebildet. Die konstanten Formel werden abgebildet wie folgt:* `true` *auf* 1 *und* `false` *auf* 0.

Die zugeordneten Funktionen, Prädikate, Wahrheitswerte zu einem Symbol R werden i.a. mit einem Index $\mathcal{S}$ gekennzeichnet. Z.B. $f \to f_{\mathcal{S}}$.

Für eine Σ-Algebra oder Σ-Struktur S soll in Zukunft mit $D_{\mathcal{S}}$ deren Trägermenge bezeichnet werden.

[4] Das Σ in Σ-Algebra bezieht sich auf die Signatur Σ: Für jede Signatur Σ gibt es eine Klasse von Σ-Algebren

Beispiel 4.9. **Σ-Algebren und Σ-Strukturen**
Sei $\Sigma = (\{o, s\}, \{L\})$ mit $arity(o) = 0, arity(s) = 1, arity(L) = 2$.

Σ-Struktur $S_1 = (\mathbb{N}, \{0, inc\}, \{<\})$ wobei ($\mathbb{N}$ = natürliche Zahlen, $inc(x) := x + 1$)

Σ-Struktur $S_2 = (\{Mo, Di, Mi, Do, Fr, Sa\}, \{Mo, morgen\}, \{vor\})$ wobei $morgen(Mo) = Di, morgen(Di) = Mi, \ldots$ und Mo vor Di, Di vor Mi,..., Sa vor So, aber nicht Mo vor Mi usw. (d.h. vor ist nicht transitiv.)

Die Trägermenge in S_1 sind die natürlichen Zahlen, dem Konstantensymbol o entspricht die Zahl 0, dem Funktionssymbol s entspricht die Funktion inc und dem Prädikatensymbol L entspricht die $<$-Relation. Die Trägermenge in S_2 sind die Wochentage, dem Konstantensymbol o entspricht der Montag, dem Funktionssymbol s entspricht die Funktion morgen und dem Prädikatensymbol L entspricht die vor-Relation.

Wenn man sagt, für die Signatur Σ hat man die Σ-Algebra $\mathcal{A}$, dann ist ganz genau festgelegt, wie die Funktions- und Prädikatensymbole von Σ in $\mathcal{A}$ interpretiert werden. Die Interpretation der Variablensymbole ist jedoch vorläufig noch offen. Auch ist noch nicht gesagt, wie die Abbildung der Terme formal definiert wird. Man will ja nicht irgendeine Abbildung, sondern eine, die zur Struktur der Terme paßt. Die eleganteste Formulierung dieser Abbildung, die auch für viele spätere Zwecke die richtige Begriffsbildung zur Verfügung stellt, geht von der Beobachtung aus, daß man für eine Signatur Σ aus der Menge der Terme $T(\Sigma, V)$ selbst zusammen mit einer Menge von "Termkonstruktorfunktionen" eine Σ-Algebra erhält. Diese Termkonstruktorfunktionen nehmen n Terme $t_1, \ldots, t_n$ und bauen daraus einen neuen Term $f(t_1, \ldots, t_n)$ auf. Diese *Termalgebra* ist das geeignete Objekt, um eine Abbildung von Termen auf eine Σ-Algebra sauber zu definieren.

Definition 4.10. *(Termalgebra) Für eine Signatur $\Sigma = (\mathcal{F}, \mathcal{P})$ und eine Menge von Variablen V sei $(T(\Sigma, V)$ die Menge der Terme über der Signatur Σ und den Variablen V. Sei F_T die Menge der Funktionen $\{f_\Sigma \mid f \in \mathcal{F}, arity(f) = n, f_\Sigma(t_1, \ldots, t_n) := f(t_1, \ldots, t_n)\}$. Dann nennt man $(T(\Sigma, V), F_T)$ die* Termalgebra *über der Signatur Σ.*

Beachte, daß bei $f_\Sigma(t_1, \ldots, t_n)$ die Klammern die n Argumente einklammern, auf die die Funktion f_Σ angewendet wird, während $f(t_1, \ldots, t_n)$ ein Datenstrukturobjekt ist, d.h. ein Term in $T(\Sigma, V)$.

Aussage 4.11. *$(T(\Sigma, V), F_T)$ ist eine Σ-Algebra .*

Beweis. Da mindestens ein Konstantensymbol in der Signatur vorhanden ist, ist die Trägermenge $T(\Sigma, V)$ nicht leer. Für jede Funktion $f_\Sigma \in T(\Sigma, V)$ gilt offensichtlich, daß die Stelligkeit paßt und daß f_Σ Terme auf Terme abbildet. Damit sind die Bedingungen der Definition 4.8 erfüllt. $\square$

Termmengen lassen sich somit als Σ-Algebren interpretieren. Worauf es dabei ankommt ist, daß die Abbildung von Elementen der Trägermenge und den Funktionen und Relationen in bestimmter Weise verträglich sind.

Was bisher noch fehlt, ist die Interpretation von Variablensymbolen.

Definition 4.12. Interpretation *Gegeben sei eine Signatur* $\Sigma = (\mathcal{F}, \mathcal{P})$ *und eine Menge von Variablen* V*. Eine* Interpretation $I = (S, I_V)$ *besteht aus einer* Σ*-Struktur* $\mathcal{S}$ *und einer Variablenbelegung* I_V*. Die Variablenbelegung ist eine Abbildung* $I_V : V \to D_S$*, die jedem Variablensymbol einen Wert in der Trägermenge von* S *zuordnet.*

Diese Interpretation wird verträglich erweitert auf Terme[5]*:*

$$I_h(t) = \begin{cases} I_V(t) & \text{falls } t \text{ eine Variable ist} \\ f_S(I_h(t_1), \ldots, I_h(t_n)) & \text{falls } t = f(t_1, \ldots t_n) \end{cases}$$

Im Folgenden schreiben wir der Einfachheit halber I *statt* I_h*.*

Eine Interpretation wie sie in 4.12 definiert wurde, enthält alle Informationen, um den in einer Signatur vorkommenden Symbolen eine Bedeutung zuzuordnen. Die Belegung von Funktions- und Prädikatensymbolen wurde getrennt von der Belegung von Variablensymbolen definiert, weil die Belegung von Variablensymbolen mit Werten aus der Trägermenge einer Σ-Algebra erst geschieht, wenn quantifizierte Formeln zu Wahrheitswerten ausgewertet werden, wobei die Belegung der Variablen noch variiert wird. Für eine Formel "$\forall x : P(x)$" zum Beispiel wird die Belegung des Prädikatensymbols "P" durch eine Σ-Struktur fixiert, während das Symbol x mit allen Werten der Trägermenge der Σ-Struktur belegt werden muß, um herauszufinden, ob $P(x)$ für alle x gilt.

Nachdem jetzt geklärt ist, wie man Terme evaluiert, wird im nächsten Schritt festgelegt, wie man Formeln auf Wahrheitswerte abbildet. Für atomare Formeln, d.h. Formeln der Art $P(t_1, \ldots, t_n)$ funktioniert das einfach so, daß man zunächst die Terme $t_1, \ldots, t_n$ auf Elemente $I_h(t_1), \ldots, I_h(t_n)$ der Trägermenge der Σ-Struktur abbildet und dann prüft, ob diese Elemente in der dem Prädikatensymbol zugeordneten Relation liegen. Wenn ja, ist das Atom wahr, wenn nicht, dann ist das Atom falsch. Für die zusammengesetzten Formeln wird die Abbildung auf Wahrheitswerte entsprechend der intuitiven Bedeutung der logischen Verknüpfungen und Quantoren definiert. (Dies ist die Stelle wo die Bedeutung der logischen Symbole präzisiert wird.)

Für eine gegebene Interpretation I definieren wir eine weitere Interpretation $I[a/x]$, mit gleicher Σ-Struktur und mit $I[a/x](x) := a$ und $I[a/x](y) := I(y)$ falls $y \neq x$.

Definition 4.13. *Auswertung von Formeln: Sei* $I = (S, I_V)$ *eine Interpretation. Basisfälle:*

$$\begin{array}{lll} \textit{Fall: } H = P(t_1, \ldots t_n) & \textit{falls } (I(t_1), \ldots, I(t_n)) \in P_S{}^6, & \textit{dann } I(H) := 1. \\ & \textit{falls } (I(t_1), \ldots, I(t_n)) \notin P_{\mathcal{S}}, & \textit{dann } I(H) := 0. \\ \textit{Fall } H = P & & I(P) := P_{\mathcal{S}}. \end{array}$$

Rekursionsfälle:

[5] Man kann das auch als Homomorphismus von Algebren und Strukturen definieren.
[6] P_S ist die in $\mathcal{S}$ dem Symbol P zugeordnete Relation

Fall: $H = \texttt{false}$ *dann* $I(H) = 0$
Fall: $H = \texttt{true}$ *dann* $I(H) = 1$
Fall: $H = \neg F$ *dann* $I(H) = 1$ *falls* $I(F) = 0$
Fall: $H = F \vee G$ *dann* $I(H) = 1$ *falls* $I(F) = 1$ *oder* $I(G) = 1$
Fall: $H = F \wedge G$, *dann* $I(H) = 1$ *falls* $I(F) = 1$ *und* $I(G) = 1$
Fall: $H = F \Rightarrow G$ *dann* $I(H) = 1$ *falls* $I(F) = 0$ *oder* $I(G) = 1$
Fall: $H = F \Leftrightarrow G$ *dann* $I(H) = 1$ *falls* $I(F) = 1$ *gdw.* $I(G) = 1$
Fall: $H = \forall x : F$ *dann* $I(H) = 1$ *falls für alle* $a \in D_S : I[a/x](F) = 1$
Fall: $H = \exists x : F$ *dann* $I(H) = 1$ *falls für ein* $a \in D_S : I[a/x](F) = 1$

Diese Definition erlaubt es, für eine Formel und eine Interpretation I zu bestimmen, ob die Formel in dieser Interpretation wahr oder falsch ist. Im nächsten Schritt lassen sich dann die Formeln danach klassifizieren, ob sie in allen Interpretationen wahr werden (Tautologien), in irgendeiner Interpretation wahr werden (erfüllbare Formeln), in irgendeiner Interpretation falsch werden (falsifizierbare Formeln) oder in allen Interpretationen falsch werden (unerfüllbare oder widersprüchliche Formeln).

Definition 4.14. *(Modelle, Tautologien etc.)*
Eine Interpretation I, die eine Formel F wahr macht (erfüllt) heißt Modell *von F. Man sagt auch: F gilt in I (F ist wahr in I, I erfüllt F). Bezeichnung:* $I \models F$.
Eine Formel F heißt:

allgemeingültig (Tautologie, Satz) *wenn sie von allen Interpretationen erfüllt wird*
erfüllbar *wenn sie von einer Interpretation erfüllt wird, d.h. wenn es ein Modell gibt*
unerfüllbar (widersprüchlich) *wenn sie von keiner Interpretation erfüllt wird.*
falsifizierbar *wenn sie in einer Interpretation falsch wird.*

Es gibt dabei folgende Zusammenhänge:

- Eine Formel F ist allgemeingültig gdw. $\neg F$ unerfüllbar
- Falls F nicht allgemeingültig ist: F ist erfüllbar gdw. $\neg F$ erfüllbar

Die Menge der unerfüllbaren und die Menge der allgemeingültigen Formeln sind disjunkt. Formeln die weder unerfüllbar noch allgemeingültig sind sind gleichzeitig erfüllbar und falsifizierbar.

Beispiel 4.15.
allgemeingültig: $P \vee \neg P$
unerfüllbar $P \wedge \neg P$
erfüllbar, falsifizierbar: $\forall x.P(x)$
Für den letzten Fall geben wir Interpretation an, die die Formel erfüllt und eine die sie falsifiziert:

1. Als Menge wählen wir $\{0, 1\}$, als Interpretation für P ebenfalls die Menge $\{0, 1\}$. Dann ergibt eine Interpretation I bzgl. dieser Struktur S_2: $I(\forall x.P(x))$ gdw $0 \in P_{S_2}$ und $1 \in P_{S_1}$. D.h. $I(\forall x.P(x)) = 1$.
2. Als Menge wählen wir $\{0, 1\}$, als Interpretation für P die Menge $\{0\}$. Dann ergibt eine Interpretation I bzgl. dieser Struktur S_2: $I(\forall x.P(x))$ gdw $0 \in P_{S_2}$ und $1 \in P_{S_1}$. D.h. $I(\forall x.P(x)) = 0$.

Als weiteren, einfachen Testfall untersuchen wir Klauseln.

Beispiel 4.16. Wann ist die Klausel $\{P(s), \neg P(t)\}$ eine Tautologie?

Zunächst ist einfach zu sehen, daß $\{P(s), \neg P(s)\}$ eine Tautologie ist: Für jede Interpretation I gilt, daß $I(P(s))$ gerade der negierte Wahrheitswert von $I(\neg P(s))$ ist.

Angenommen, $s \neq t$. Vermutung: dann ist es keine Tautologie. Um dies nachzuweisen, muß man eine Interpretation finden, die diese Klausel falsch macht.

Zuerst definieren wir eine Trägermenge und eine Σ-Algebra. Man startet mit einer Menge A_0, die mindestens soviele Elemente enthält, wie die Terme s, t Konstanten und Variablen enthalten. Also $A_0 := \{a_1, \ldots, a_n, c_{x_1}, \ldots c_{x_m}\}$, wobei a_i die Konstanten in s, t sind und x_i die Variablen.

Danach definiert man alle Funktionen so, die in s, t vorkommen, so daß keinerlei Beziehungen gelten. D.h. man kann genau alle Terme nehmen, die sich aus den A_0 als Konstanten und den Funktionssymbolen aufbauen lassen. D.h. es ist die Termmenge einer Termalgebra über einer erweiterten Signatur Σ'.

Danach wählt man als Interpretation $I(a_i) := a_i$, $I(x_i) = c_{x_i}$, $f_S = f$. Beachte, daß der Quantorpräfix nur aus Allquantoren besteht.

Damit gilt nun: $I(S) \neq I(t)$. Da man die einstellige Relation (die Menge), die P zugeordnet wird, frei wählen kann, kann man dies so machen, daß $I(s) \notin P_S$ und $I(t) \in P_S$. Damit wird aber die Klausel falsch unter dieser Interpretation. D.h. es kann keine Tautologie sein.

Analog kann man diese Argumentation für Klauseln mit mehrstelligen Prädikaten verwenden.

Übungsaufgabe 4.17. Jede Klausel die mehr als ein Literal enthält ist erfüllbar. Wie nehmen an, daß die Formeln $true$, $false$ nicht in Klauseln verwendet werden.

Mit der Einführung des Begriffs eines Modell (und einer Interpretation) sind alle Voraussetzungen gegeben, um – in Verallgemeinerung der Begriffe für Aussagenlogik – eine semantische Folgerungsbeziehung $\models$ zwischen zwei Formeln zu definieren:

Definition 4.18. *(Semantische Folgerung) $F \models G$ gdw. G gilt (ist wahr) in allen Modellen von F.*

Diese Definition ist zwar sehr intuitiv, da es aber i.a. unendlich viele Modelle für eine Formel gibt, ist sie jedoch in keiner Weise geeignet, um für zwei konkrete Formeln F und G zu testen, ob G aus F semantisch folgt. Die semantische Folgerungsbeziehung dient aber als Referenz; jedes konkrete, d.h. programmierbare Testverfahren muß sich daran messen, ob und wie genau es diese Beziehung zwischen zwei Formeln realisiert.

Bemerkung 4.19. Man muß sich folgendes klar machen: Bezüglich einer gegebenen Σ-Algebra ist eine gegebene geschlossene Formel F entweder wahr oder falsch. Das heißt aber noch nicht, daß entweder F oder $\neg F$ eine Tautologie ist, denn dieser Begriff ist definiert über **alle** Σ-Strukturen.

Bemerkung 4.20. Das Beispiel der natürlichen Zahlen und der darin geltenden Sätze ist nicht vollständig mit PL_1 zu erfassen. Der Grund ist, daß man nur von einer einzigen festen Σ-Struktur ausgeht (die natürlichen Zahlen) und dann nach der Gültigkeit von Sätzen fragt.

Versucht man die natürlichen Zahlen in PL_1 zu erfassen, so stellt sich heraus, daß man mit endlich vielen Axiomen nicht auskommt. Es ist sogar so, daß die Menge der Axiome nicht rekursiv aufzählbar ist.

Beispiel 4.21. Ein Beispiel für eine in PL_1 modellierbare Theorie sind die Gruppen. Man benötigt nur endlich viele Axiome. Und man kann dann danach fragen, welche Sätze in allen Gruppen gelten.

Ein erster Schritt zur Mechanisierung der Folgerungsbeziehung liefert das sogenannte *Deduktionstheorem*, welches die semantische Folgerungsbeziehung in Beziehung setzt mit dem syntaktischen Implikationszeichen. Dieses Theorem erlaubt die Rückführung der semantischen Folgerung auf einen Tautologietest.

Satz 4.22. *Deduktionstheorem Für alle Formeln F und G gilt: $F \models G$ gdw. $F \Rightarrow G$ ist allgemeingültig (Tautologie).*

Beweis. "$\Rightarrow$" Es gelte $F \models G$. Sei I eine beliebige Interpretation. Wenn F in I wahr ist, dann ist auch G wahr, nach Annahme. Damit ist aber auch die Formel $F \Rightarrow G$ in I wahr. Wenn F in I falsch ist, dann ist die Formel $F \Rightarrow G$ in I wahr.

Also gilt $F \Rightarrow G$ in allen Interpretationen I, d.h. $F \Rightarrow G$ ist allgemeingültig.

"$\Leftarrow$" Annahme ist jetzt: $F \Rightarrow G$ ist allgemeingültig. Sei I eine beliebige Interpretation. Wenn I die Formel F erfüllt, dann gilt das auch für G, da $F \Rightarrow G$ ist allgemeingültig ist.

Da die Interpretation beliebig gewählt war, gilt somit $F \models G$ □

Bemerkung 4.23. Will man wissen, ob eine Formel F aus einer Menge von Axiomen $A_1, \ldots, A_n$ folgt, so kann man dies zunächst auf die äquivalente Frage zurückführen, ob F aus der Konjunktion der Axiome $A_1 \wedge \ldots \wedge A_n$ folgt und dann auf die äquivalente Frage, ob die Implikation $(A_1 \wedge \ldots \wedge A_n) \Rightarrow F$ eine Tautologie ist.

Das Deduktionstheorem gilt in anderen Logiken i.a. nicht mehr. Das kann daran liegen, daß die semantische Folgerungsbeziehung dort anders definiert ist oder auch, daß die Implikation selbst anders definiert ist. Die Implikation, so wie sie in PL_1 definert ist, hat nämlich den paradoxen Effekt, daß aus etwas Falschem alles folgt, d.h. die Formel $false \Rightarrow F$ ist eine Tautologie. Versucht man, diesen Effekt durch eine geänderte Definition für $\Rightarrow$ zu vermeiden, dann muß das Deduktionstheorem nicht mehr unbedingt gelten. Da F eine Tautologie ist genau dann wenn $\neg F$ unerfüllbar ist, folgt unmittelbar:

$$F \models G$$
gdw.
$$\neg(F \Rightarrow G) \text{ ist unerfüllbar (widersprüchlich)}$$
gdw.
$$F \wedge \neg G \text{ ist unerfüllbar.}$$

Das bedeutet, daß man in PL_1 den Test der semantischen Folgerungsbeziehung weiter zurückführen kann auf einen Unerfüllbarkeitstest. Genau dieses Verfahren ist die häufig verwendete Methode des **Beweis durch Widerspruch**: Um zu zeigen, daß aus Axiomen ein Theorem folgt, zeigt man, daß die Axiome zusammen mit dem negierten Theorem einen Widerspruch ergeben.

Berechenbarkeitseigenschaften der Prädikatenlogik Es gilt die Unentscheidbarkeit der Prädikatenlogik:

Satz 4.24. *Es ist unentscheidbar, ob eine geschlossene Formel ein Satz der Prädikatenlogik ist.*

Einen Beweis geben wir nicht. Der Beweis besteht darin, ein Verfahren anzugeben, das jeder Turingmaschine M eine prädikatenlogische Formel zuordnet, die genau dann ein Satz ist, wenn diese Turingmaschine auf dem leeren Band terminiert. Hierbei nimmt man TM, die nur mit einem Endzustand terminieren können. Da das Halteproblem für Turingmaschinen unentscheidbar ist, hat man damit einen Beweis für den Satz.

Satz 4.25. *Die Menge der Sätze der Prädikatenlogik ist rekursiv aufzählbar.*

Die Begründung ist analog.

Als Schlußfolgerung kann man sagen, daß es kein Deduktionssystem gibt (Algorithmus), das bei eingegebener Formel nach endlicher Zeit entscheiden kann, ob die Formel ein Satz ist oder nicht. Allerdings gibt es einen Algorithmus, der für jede Formel, die ein Satz ist, auch terminiert und diese als Satz erkennt.

Über das theoretische Verhalten eines automatischen Deduktionssystems kann man daher folgendes sagen: Es kann terminieren und antworten: ist oder ist kein Satz. Wenn das System sehr lange läuft, kann das zwei Ursachen haben: Der Satz ist zu schwer zu zeigen (zu erkennen) oder die eingegebene Formel ist kein Satz und das System kann auch dies nicht erkennen.

Die Einordnung der sogenannten quantifizierten Booleschen Formeln – Quantoren über null-stellige Prädikate sind erlaubt, aber keine Funktionssymbole, und keine mehrstelligen Prädikate – ist in PL_1 nicht möglich. Diese QBF sind ein "Fragment" von PL_2. Es ist bekannt (aus Info-3+4), daß die Eigenschaft "Tautologie" für QBF entscheidbar ist (PSPACE-vollständig).

4.3 Normalformen von PL_1-Formeln

Ziel diese Abschnitts ist es, einen Algorithmus zu entwickeln, der beliebige Formeln in eine Klauselnormalform (conjunctive normal form, CNF), d.h. eine Konjunktion ($\wedge$) von Disjunktionen ($\vee$) von Literalen, transformiert. Diese Klauselnormalform ist nur "ganz außen" allquantifiziert, es gibt keine inneren Quantoren. Folgendes Lemma erlaubt die Transformation von Formeln, wobei die Regeln in Tautologien entsprechen, aber als Transformationen benutzt werden. Diese sind Erweiterungen der Tautologien der Aussagenlogik.

Satz 4.26. Elementare Rechenregeln

$$
\begin{array}{lll}
(F \Leftrightarrow G) & \Leftrightarrow (F \Rightarrow G) \wedge (G \Rightarrow F) & \textit{(erlaubt Elimination von } \Leftrightarrow\textit{)} \\
(F \Rightarrow G) & \Leftrightarrow (\neg F \vee G) & \textit{(erlaubt Elimination von } \Rightarrow\textit{)} \\
\neg\neg F & \Leftrightarrow F & \\
\neg(F \wedge G) & \Leftrightarrow \neg F \vee \neg G & \\
\neg(F \vee G) & \Leftrightarrow \neg F \wedge \neg G & \\
\neg\forall x : F & \Leftrightarrow \exists x : \neg F & \\
\neg\exists x : F & \Leftrightarrow \forall x : \neg F & \\
(\forall x : F) \wedge G & \Leftrightarrow \forall x : (F \wedge G) & \textit{falls } x \textit{ nicht frei in } G \\
(\forall x : F) \vee G & \Leftrightarrow \forall x : (F \vee G) & \textit{falls } x \textit{ nicht frei in } G \\
(\exists x : F) \wedge G & \Leftrightarrow \exists x : (F \wedge G) & \textit{falls } x \textit{ nicht frei in } G \\
(\exists x : F) \vee G & \Leftrightarrow \exists x : (F \vee G) & \textit{falls } x \textit{ nicht frei in } G \\
\forall x : F \wedge \forall x : G & \Leftrightarrow \forall x : (F \wedge G) & \\
\exists x : F \vee \exists x : G & \Leftrightarrow \exists x : (F \vee G) & \\
F \vee (G \wedge H) & \Leftrightarrow (F \vee G) \wedge (F \vee H) & \textit{(Distributivität)} \\
F \wedge (G \vee H) & \Leftrightarrow (F \wedge G) \vee (F \wedge H) & \textit{(Distributivität)}
\end{array}
$$

Beweis. Wir beweisen beispielhaft den Fall: $\neg\forall x : F \Leftrightarrow \exists x : \neg F$
"$\Rightarrow$ " Sei I eine Interpretation, die $\neg\forall x : F$ erfüllt.

d.h. $\forall x : F$ gilt nicht in I (4.13).
$\leadsto$ D.h.: es gilt nicht: für alle a aus der Trägermenge von I: $I[a/x]$ erfüllt F.
$\leadsto$ Somit gibt es ein a aus der Trägermenge von I, so daß F nicht in $I[a/x]$ gilt.
$\leadsto$ Es gibt ein a aus der Trägermenge von I, so daß $I[a/x]$ die Formel $\neg F$ erfüllt (4.13, Fall $\neg$)
$\leadsto$ I erfüllt damit $\exists x : \neg F$. (4.13, Fall $\exists$)

"$\Leftarrow$" Sei I eine Interpretation, die $\exists x : \neg F$ erfüllt. Die obige Folgerungskette kann auch rückwärts durchlaufen werden. □

Bemerkung 4.27. Mithilfe der obigen Tautologien kann man die sogenannte Pränexform und auch die Negations-normalform einer Formel herstellen. Die *Pränex-form* ist dadurch gekennzeichnet, daß in der Formel zuerst alle Quantoren kommen (Quantorpräfix), und dann eine quantorenfreie Formel. Die *Negations-normalform* ist dadurch gekennzeichnet, daß alle Negationszeichen nur vor Atomen vorkommen und daßdie Junktoren $\Rightarrow, \Leftrightarrow$ eliminiert sind.

Zur Umwandlung einer Formel in Pränexform braucht man nur die Äquivalenzen zu verwenden, die Quantoren nach außen schieben. Hierzu müssen alle gebundenen Variablen verschiedene Namen haben. Die Äquivalenzen, die es erlauben, Subformeln unter Quantoren $\forall x$. zu schieben, falls diese die Subformel die Variable x nicht enthält, spielen eine wichtige Rolle.

Die Negationsnormalform wird erreicht, indem man zunächst $\Rightarrow, \Leftrightarrow$ eliminiert und dann alle Äquivalenzen nutzt, um Negationszeichen nach innen zu schieben.

Die Elimination von Existenzquantoren ist die sogenannte Skolemisierung (Nach Thoralf Skolem).
Idee: Ersetze in $\exists x : P(x)$ das x, "das existiert" durch ein Konstantensymbol a, d.h. $\exists x : P(x) \to P(a)$ Ersetze in $\forall x_1 \dots x_n : \exists y : P(x_1, \dots, x_n, y)$ das y durch eine Funktion von $x_1, \dots, x_n$, d.h. $\forall x_1 \dots x_n : \exists y : P(x_1, \dots, x_n, y) \to \forall x_1 \dots x_n : P(x_1, \dots, x_n, f(x_1, \dots, x_n))$.

Im nächsten Theorem sei $G[x_1, \dots, x_n, y]$ eine beliebige Formel, die die Variablensymbole $x_1, \dots, x_n, y$ frei enthält und $G[x_1, \dots, x_n, t]$ eine Variante von F, in der alle Vorkommnisse von y durch t ersetzt sind.

Satz 4.28. Skolemisierung
Eine Formel $F = \forall x_1 \dots x_n : \exists y : G[x_1, \dots, x_n, y]$ *ist (un-)erfüllbar gdw.* $F' = \forall x_1 \dots x_n : G[x_1, \dots, x_n, f(x_1, \dots, x_n)]$ *(un-)erfüllbar ist, wobei* f *ein* n*-stelliges Funktionssymbol ist, das nicht in* G *vorkommt.*

Beweis. Wir zeigen die Erfüllbarkeitsäquivalenz. Die Unerfüllbarkeitsäquivalenz ergibt sich dann automatisch.
"$\Rightarrow$" Sei $I_1 = (A_1, I_V)$ eine Σ_1-Interpretation mit $I_1 \models F$, wobei Σ_1 die Signatur für $\mathcal{F}$ ist. Für $\Sigma_2 = \Sigma_1$ plus zusätzliches $n+1$-stelliges Funktionssymbol f konstruieren wir ein Σ_2-Modell I_2 für F' folgendermaßen: $I_2 = (A_2, I_V)$ mit $A_2 := A_1$ plus eine neue $n+1$-stellige Funktion f_{A_2}, wobei f_{A_2} wie folgt arbeitet: für alle $a_1, \dots, a_n$ aus der Trägermenge von A_2: $f_{A_2}(a_1, \dots, a_n) := c$, so daß

$$I_1[a_1/x_1, \dots, a_n/x_n, c/y] \models G[x_1, \dots, x_n, y]$$

. Da I_1 ein Modell für F ist, existiert so ein c immer.

$\leadsto I_2[a_1/x_1, \dots, a_n/x_n] \models G[x_1, \dots, x_n, f(x_1, \dots, x_n)]$
$\leadsto I_2 \models \forall x_1, \dots, x_n : G[x_1, \dots, x_n, f(x_1, \dots, x_n)]$
$\leadsto F'$ ist erfüllbar.

Die "$\Leftarrow$" Richtung ist trivial. □

Beispiel 4.29. **Skolemisierung**

$$\exists x : P(x) \to P(a)$$
$$\forall x : \exists y : Q(f(y,y), x, y) \to \forall x : Q(f(g(x), g(x)), x, g(x))$$
$$\forall x, y : \exists z : x + z = y \to \forall x, y : x + (x - y) = y.$$

Beispiel 4.30. Skolemisierung erhält i.a. nicht die Allgemeingültigkeit (Falsifizierbarkeit):

$\forall x : P(x) \vee \neg \forall x : P(x)$ ist eine Tautologie
$\forall x : P(x) \vee \exists x : \neg P(x)$ ist äquivalent zu

$\forall x : P(x) \vee \neg P(a)$ nach Skolemisierung.

Eine Interpretation, die die skolemisierte Formel falsifiziert kann man konstruieren wie folgt: Die Trägermenge ist $\{a, b\}$. Es gelte $P(a)$ und $\neg P(b)$. Die Formel ist aber noch erfüllbar.

Beachte, daß es dual dazu auch eine Form der Skolemisierung gibt, bei der die allquantifizierten Variablen skolemisiert werden. Dies wird verwendet, wenn man statt auf Widersprüchlichkeit die Formeln auf Allgemeingültigkeit testet. Im Beweis ersetzt man dann die Begriff erfüllbar durch falsifizierbar und unerfüllbar durch allgemeingültig.

Skolemisierung ist eine Operation, die nicht lokal innerhalb von Formeln verwendet werden darf, sondern nur global, d.h. wenn die ganze Formel eine bestimmte Form hat. Zudem bleibt bei dieser Operation nur die Unerfüllbarkeit der ganzen Klausel erhalten.

Satz 4.31. Allgemeinere Skolemisierung
Sei F eine geschlossene Formel, G eine existentiell quantifizierte Unterformel in F an einer Position p, Weiterhin sei G nur unter All-quantoren, Konjunktionen, und Disjunktionen. Die All-quantoren über G binden die Variablen $x_1, \ldots, x_n$. D.h. F ist von der Form $F[\exists y : G'[x_1, \ldots, x_n, y]]$.

Dann ist $F[G]$ (un-)erfüllbar gdw. $F[G'[x_1, \ldots, x_n, f(x_1, \ldots, x_n)]]$ (un-)erfüllbar ist, wobei f ein n-stelliges Funktionssymbol ist, das nicht in G vorkommt.

Definition 4.32. Transformation in Klauselnormalform *unter Erhaltung der Unerfüllbarkeit.*

Folgende Prozedur wandelt jede prädikatenlogische Formel in Klauselform (CNF) um:

1. *Elimination von $\Leftrightarrow$ und $\Rightarrow$: $F \Leftrightarrow G \to F \Rightarrow G \wedge G \Rightarrow F$ (Lemma 4.26) und $F \Rightarrow G \to \neg F \vee G$*
2. *Negation ganz nach innen schieben:*

$$\begin{array}{ll} \neg\neg F & \to F \\ \neg(F \wedge G) & \to \neg F \vee \neg G \\ \neg(F \vee G) & \to \neg F \wedge \neg G \\ \neg\forall x : F & \to \exists x : \neg F \\ \neg\exists x : F & \to \forall x : \neg F \end{array}$$

3. *Skopus von Quantoren minimieren, d.h. Quantoren so weit wie möglich nach innen schieben*

$$\begin{array}{ll} \forall x : (F \wedge G) \to (\forall x : F) \wedge G & \textit{falls } x \textit{ nicht frei in } G \\ \forall x : (F \vee G) \to (\forall x : F) \vee G & \textit{falls } x \textit{ nicht frei in } G \\ \exists x : (F \wedge G) \to (\exists x : F) \wedge G & \textit{falls } x \textit{ nicht frei in } G \\ \exists x : (F \vee G) \to (\exists x : F) \vee G & \textit{falls } x \textit{ nicht frei in } G \\ \forall x : (F \wedge G) \to \forall x : F \wedge \forall x : G & \\ \exists x : (F \vee G) \to \exists x : F \vee \exists x : G & \end{array}$$

4. Alle gebundenen Variablen sind systematisch umzubenennen, um Namenskonflikte aufzulösen.
5. Existenzquantoren werden durch Skolemisierung eliminiert
6. Allquantoren löschen (alle Variablen werden als allquantifiziert angenommen.
7. Distributivität (und Assoziativität, Kommutativität) iterativ anwenden, um $\wedge$ nach außen zu schieben ("Ausmultiplikation"). $F \vee (G \wedge H) \to (F \vee G) \wedge (F \vee H)$ (Das duale Distributivgesetz würde eine disjunktive Normalform ergeben.)

Das Resultat dieser Prozedur ist eine Konjunktion von Disjunktionen (Klauseln) von Literalen:

$$\begin{array}{ll} & (L_{1,1} \vee \ldots \vee L_{1,n_1}) \\ \wedge & (L_{2,1} \vee \ldots \vee L_{2,n_2}) \\ \wedge & \\ \ldots & \\ \wedge & (L_{k,1} \vee \ldots \vee L_{1,n_k}) \end{array}$$

oder in (Multi-)Mengenschreibweise:

$$\begin{array}{l} \{\{(L_{1,1}, \ldots, L_{1,n_1}\}, \\ \{L_{2,1}, \ldots, L_{2,n_2}\}, \\ \ldots \\ \{L_{k,1}, \ldots, L_{1,n_k}\}\} \end{array}$$

Beispiel 4.33. Der CNF-algorithmus (4.32) ist im schlechtesten Fall exponentiell, d.h. die Anzahl der Literale in der Klauselform wächst exponentiell mit der Schachtelungstiefe der Ausgangsformel (siehe 2.4).

Analog zu 2.4 und unter den gleichen Bedingungen gibt es einen verbesserten Algorithmus, der nur lineares Anwachsen der Formel zur Folge hat.

Beispiel 4.34.
A1: Dieb(Anton) $\vee$ Dieb(Ede) $\vee$ Dieb(Karl)
A2: Dieb(Anton) $\Rightarrow$ (Dieb(Ede) $\vee$ Dieb(Karl))
A3: Dieb(Karl) $\Rightarrow$ (Dieb(Ede) $\vee$ Dieb(Anton))
A4: Dieb(Ede) $\Rightarrow$ ($\neg$ Dieb(Anton) $\wedge\neg$ Dieb(Karl))
A5: $\neg$ Dieb(Anton) $\vee\neg$ Dieb(Karl)
Klauselform:
A1: Dieb(Anton), Dieb(Ede), Dieb(Karl)
A2: $\neg$ Dieb(Anton), Dieb(Ede), Dieb(Karl)
A3: $\neg$ Dieb(Karl), Dieb(Ede), Dieb(Anton)
A4a: $\neg$ Dieb(Ede), $\neg$ Dieb(Anton)
A4b: $\neg$ Dieb(Ede), $\neg$ Dieb(Karl)
A5: $\neg$ Dieb(Anton),$\neg$ Dieb(Karl)

Beispiel 4.35. verschiedene Typen von Normalformen:

Original Formel: $\forall \varepsilon : (\varepsilon > 0 \Rightarrow \exists \delta : (\delta > 0 \wedge \forall x, y : (|x - y| < \delta \Rightarrow |g(x) - g(y)| < \varepsilon)))$

Negations Normalform : (Alle Negationen innen; $\Rightarrow, \Leftrightarrow$ eliminiert)

$$\forall \varepsilon : (\neg \varepsilon > 0 \vee \exists \delta : (\delta > 0 \wedge \forall x, y : (\neg |x - y| < \delta \vee |g(x) - g(y)| < \varepsilon)))$$

Pränex Form : (Alle Quantoren außen) $\forall \varepsilon : \exists \delta : \forall x, y : \varepsilon > 0 \Rightarrow \delta > 0 \wedge (|x - y| < \delta \Rightarrow |g(x) - g(y)| < \varepsilon)$

Skolemisierte Pränex Form : $\varepsilon > 0 \Rightarrow f_\delta(\varepsilon) > 0 \wedge (|x - y| < f_\delta(\varepsilon) \Rightarrow |g(x) - g(y)| < \varepsilon)$

Disjunktive Normalform : $(\neg \varepsilon > 0) \vee (fd(\varepsilon) > 0 \wedge \neg |x - y| < f_\delta(\varepsilon)) \vee (f_\delta(\varepsilon) > 0 \wedge |g(x) - g(y)| < \varepsilon)$

Konjunktive Normalform : $(\neg \varepsilon > 0 \vee f_\delta(\varepsilon) > 0) \wedge (\neg \varepsilon > 0 \vee \neg |x - y| < fd(\varepsilon) \vee |g(x) - g(y)| < \varepsilon)$

Klauselform : $\{\{\neg \varepsilon > 0, f_\delta(\varepsilon) > 0\}, \{\neg \varepsilon > 0, \neg |x - y| < f_\delta(\varepsilon), |g(x) - g(y)| < \varepsilon\}\}$.