Kapitel 5

Prädikatenlogik

In der Aussagenlogik haben wir die Verknüpfung von elementaren Aussagen mit Junktoren untersucht. Die Prädikatenlogik untersucht zusätzlich auch die Struktur der Aussagen. Dazu werden in der Prädikatenlogik die folgenden zusätzlichen Begriffe eingeführt:

1. Als Bezeichnungen für Objekte werden *Terme* verwendet.
2. Diese Terme werden aus *Variablen* und *Funktions-Zeichen* zusammengesetzt:

   $$\mathit{vater}(x), \quad \mathit{mutter}(\mathit{isaac}), \quad x + 7, \quad \cdots$$

3. Verschiedene Objekte werden durch *Prädikats-Zeichen* in Relation gesetzt:

   $$\mathit{istBruder}\big(\mathit{albert}, \mathit{vater}(\mathit{bruno})\big), \quad x + 7 < x \cdot 7, \quad n \in \mathbb{N}, \quad \cdots$$

   Die dabei entstehenden Formeln werden als *atomare* Formeln bezeichnet.
4. Atomare Formeln lassen sich durch aussagenlogische Junktoren verknüpfen:

   $$x > 1 \rightarrow x + 7 < x \cdot 7$$

5. Schließlich werden *Quantoren* eingeführt, um zwischen *existentiell* und *universell* quantifizierten Variablen unterscheiden zu können:

   $$\forall x \in \mathbb{R} : \exists n \in \mathbb{N} : x < n$$

Wir werden im nächsten Abschnitt die Syntax der prädikatenlogischen Formeln festlegen und uns dann im darauf folgenden Abschnitt mit der Semantik dieser Formeln beschäftigen.

5.1 Syntax der Prädikatenlogik

Zunächst definieren wir den Begriff der *Signatur*. Inhaltlich ist das nichts anderes als eine strukturierte Zusammenfassung von Variablen, Funktions- und Prädikats-Zeichen zusammen mit einer Spezifikation der Stelligkeit dieser Zeichen.

Definition 35 (Signatur) Eine *Signatur* ist ein 4-Tupel

$$\Sigma = \langle \mathcal{V}, \mathcal{F}, \mathcal{P}, \mathit{arity} \rangle,$$

für das folgendes gilt:

1. $\mathcal{V}$ ist die Menge der Variablen.
2. $\mathcal{F}$ ist die Menge der Funktions-Zeichen.
3. $\mathcal{P}$ ist die Menge der Prädikats-Zeichen.
4. *arity* ist eine Funktion, die jedem Funktions- und jedem Prädikats-Zeichen seine *Stelligkeit* zuordnet:

   $$\mathit{arity} : \mathcal{F} \cup \mathcal{P} \to \mathbb{N}.$$

   Wir sagen, dass das Funktions- oder Prädikats-Zeichen f ein n-stelliges Zeichen ist, falls $\mathit{arity}(f) = n$ gilt.
5. Da wir in der Lage sein müssen, Variablen, Funktions- und Prädikats-Zeichen unterscheiden zu können, vereinbaren wir, dass die Mengen $\mathcal{V}$, $\mathcal{F}$ und $\mathcal{P}$ paarweise disjunkt sein müssen:

   $$\mathcal{V} \cap \mathcal{F} = \{\}, \quad \mathcal{V} \cap \mathcal{P} = \{\}, \quad \text{und} \quad \mathcal{F} \cap \mathcal{P} = \{\}.$$

Als Bezeichner für Objekte verwenden wir Ausdrücke, die aus Variablen und Funktions-Zeichen aufgebaut sind. Solche Ausdrücke nennen wir *Terme*. Formal werden diese wie folgt definiert.

Definition 36 (Terme) Ist $\Sigma = \langle \mathcal{V}, \mathcal{F}, \mathcal{P}, \mathit{arity} \rangle$ eine Signatur, so definieren wir die Menge der Σ-Terme $\mathcal{T}_\Sigma$ induktiv:

1. Für jede Variable $x \in \mathcal{V}$ gilt $x \in \mathcal{T}_\Sigma$.
2. Ist $f \in \mathcal{F}$ ein n-stelliges Funktions-Zeichen und sind $t_1, \cdots, t_n \in \mathcal{T}_\Sigma$, so gilt auch

   $$f(t_1, \cdots, t_n) \in \mathcal{T}_\Sigma.$$

   Falls $c \in \mathcal{F}$ 0-stellig ist, lassen wir auch die Schreibweise c anstelle von $c()$ zu. In diesem Fall nennen wir c eine *Konstante*. □

Zur Veranschaulichung der zuletzt eingeführten Begriffe geben wir ein Beispiel. Es sei die Menge der Variablen $\mathcal{V} = \{x, y, z\}$, die Menge der Funktions-Zeichen $\mathcal{F} = \{0, 1, +, -, \cdot\}$, und die Menge der Prädikats-Zeichen $\mathcal{P} = \{=, \leq\}$ gegeben. Ferner sei die Funktion *arity* als die Relation

$$\mathit{arity} = \big\{\langle 0, 0\rangle, \langle 1, 0\rangle, \langle +, 2\rangle, \langle -, 2\rangle, \langle *, 2\rangle, \langle =, 2\rangle, \langle \leq, 2\rangle\big\}$$

definiert. Schließlich sei die Signatur Σ_{arith} durch das 4-Tupel $\langle \mathcal{V}, \mathcal{F}, \mathcal{P}, \mathit{arity} \rangle$ gegeben.

Dann können wir wie folgt Σ_{arith}-Terme konstruieren:

1. $x, y, z \in \mathcal{T}_{\Sigma_{\text{arith}}}$,

   denn alle Variablen sind auch Σ_{arith}-Terme.
2. $0, 1 \in \mathcal{T}_{\Sigma_{\text{arith}}}$,

   denn 0 und 1 sind 0-stellige Funktions-Zeichen.
3. $+(0, x) \in \mathcal{T}_{\Sigma_{\text{arith}}}$,

   denn es gilt $0 \in \mathcal{T}_{\Sigma_{\text{arith}}}$, $x \in \mathcal{T}_{\Sigma_{\text{arith}}}$ und $+$ ist ein 2-stelliges Funktions-Zeichen.
4. $*(+(0, x), 1) \in \mathcal{T}_{\Sigma_{\text{arith}}}$,

   denn $+(0, x) \in \mathcal{T}_{\Sigma_{\text{arith}}}$, $1 \in \mathcal{T}_{\Sigma_{\text{arith}}}$ und $*$ ist ein 2-stelliges Funktions-Zeichen.

Als nächstes definieren wir den Begriff der *atomaren Formeln*. Darunter verstehen wir solche Formeln, die man nicht in kleinere Formeln zerlegen kann, atomare Formeln enthalten also weder Junktoren noch Quantoren.

Definition 37 (Atomare Formeln) Gegeben sei eine Signatur $\Sigma = \langle \mathcal{V}, \mathcal{F}, \mathcal{P}, \mathit{arity} \rangle$. Die Menge der atomaren Σ-Formeln $\mathcal{A}_\Sigma$ wird wie folgt definiert: Ist $p \in \mathcal{P}$ ein n-stelliges Prädikats-Zeichen und sind n Σ-Terme t_1, $\cdots$, t_n gegeben, so ist $p(t_1, \cdots, t_n)$ eine atomaren Σ-Formel:

$$p(t_1, \cdots, t_n) \in \mathcal{A}_\Sigma.$$

Falls p ein 0-stelliges Prädikats-Zeichen ist, dann schreiben wir auch p anstelle von $p()$. In diesem Fall nennen wir p eine *Aussage-Variable*. □

Setzen wir das obige Beispiel fort, so können wir sehen, dass

$$=(*(+(0, x), 1), 0)$$

eine atomare Σ_{arith}-Formel ist. Beachten Sie, dass wir bisher noch nichts über den Wahrheitswert von solchen Formeln ausgesagt haben. Die Frage, wann eine Formel als wahr oder falsch gelten soll, wird erst im nächsten Abschnitt untersucht.

Bei der Definition der prädikatenlogischen Formeln ist es notwendig, zwischen sogenannten *gebundenen* und *freien* Variablen zu unterscheiden. Wir führen diese Begriffe zunächst informal mit Hilfe eines Beispiels aus der Analysis ein. Wir betrachten die folgende Identität:

$$\int_0^x y \cdot t \, dt = \tfrac{1}{2} x^2 \cdot y$$

In dieser Gleichung treten die Variablen x und y *frei* auf, während die Variable t durch das Integral *gebunden* wird. Damit meinen wir folgendes: Wir können in dieser Gleichung für x und y beliebige Werte einsetzen, ohne dass sich an der Gültigkeit der Formel etwas ändert. Setzen wir zum Beispiel für x den Wert 2 ein, so erhalten wir

$$\int_0^2 y \cdot t \, dt = \tfrac{1}{2} 2^2 \cdot y$$

und diese Identität ist ebenfalls gültig. Demgegenüber macht es keinen Sinn, wenn wir für die gebundene Variable t eine Zahl einsetzen würden. Die linke Seite der entstehenden Gleichung wäre einfach undefiniert. Wir können für t höchstens eine andere Variable einsetzen. Ersetzen wir die Variable t beispielsweise durch u, so erhalten wir

$$\int_0^x y \cdot u \, du = \tfrac{1}{2} x^2 \cdot y$$

und das ist die selbe Aussage wie oben. Das funktioniert allerdings nicht mit jeder Variablen. Setzen wir für t die Variable y ein, so erhalten wir

$$\int_0^x y \cdot y \, dy = \tfrac{1}{2} x^2 \cdot y.$$

Diese Aussage ist aber falsch! Das Problem liegt darin, dass bei der Ersetzung von t durch y die vorher freie Variable y gebunden wurde.

Ein ähnliches Problem erhalten wir, wenn wir für y beliebige Terme einsetzen. Solange diese Terme die Variable t nicht enthalten, geht alles gut. Setzen wir beispielsweise für y den Term x^2 ein, so erhalten wir

$$\int_0^x x^2 \cdot t \, dt = \tfrac{1}{2} x^2 \cdot x^2$$

und diese Formel ist gültig. Setzen wir allerdings für y den Term t^2 ein, so erhalten wir

$$\int_0^x t^2 \cdot t \, dt = \tfrac{1}{2} x^2 \cdot t^2$$

und diese Formel ist nicht mehr gültig.

In der Prädikatenlogik binden die Quantoren "$\forall$" (*für alle*) und "$\exists$" (*es gibt*) Variablen in ähnlicher Weise, wie der Integral-Operator "$\int \cdots dt$" in der Analysis Variablen bindet. Die oben gemachten Ausführungen zeigen, dass es zwei verschiedene Arten von Variable gibt: *freie Variable* und *gebundene Variable*. Um diese Begriffe

präzisieren zu können, definieren wir zunächst für einen Σ-Term t die Menge der in t enthaltenen Variablen.

Definition 38 ($Var(t)$) Ist t ein Σ-Term, mit $\Sigma = \langle \mathcal{V}, \mathcal{F}, \mathcal{P}, arity \rangle$, so definieren wir die Menge $Var(t)$ der Variablen, die in t auftreten, durch Induktion nach dem Aufbau des Terms:

1. $Var(x) := \{x\}$ für alle $x \in \mathcal{V}$,
2. $Var\big(f(t_1, \cdots, t_n)\big) := Var(t_1) \cup \cdots \cup Var(t_n)$. □

Definition 39 (Σ-Formel, gebundene und freie Variablen)
Es sei $\Sigma = \langle \mathcal{V}, \mathcal{F}, \mathcal{P}, arity \rangle$ eine Signatur. Die Menge der Σ-*Formeln* bezeichnen wir mit $\mathbb{F}_\Sigma$. Wir definieren diese Menge induktiv. Gleichzeitig definieren wir für jede Formel $F \in \mathbb{F}_\Sigma$ die Menge $BV(F)$ der in F *gebunden* auftretenden Variablen und die Menge $FV(F)$ der in F *frei* auftretenden Variablen.

1. Es gilt $\bot \in \mathbb{F}_\Sigma$ und $\top \in \mathbb{F}_\Sigma$ und wir definieren
   $$FV(\bot) := FV(\top) := BV(\bot) := BV(\top) := \{\}$$
2. Ist $F = p(t_1, \cdots, t_n)$ eine atomare Σ-Formel, so gilt $F \in \mathbb{F}_\Sigma$. Weiter definieren wir:
   (a) $FV\big(p(t_1, \cdots, t_n)\big) := Var(t_1) \cup \cdots \cup Var(t_n)$.
   (b) $BV\big(p(t_1, \cdots, t_n)\big) := \{\}$.
3. Ist $F \in \mathbb{F}_\Sigma$, so gilt $\neg F \in \mathbb{F}_\Sigma$. Weiter definieren wir:
   (a) $FV\big(\neg F\big) := FV(F)$.
   (b) $BV\big(\neg F\big) := BV(F)$.
4. Sind $F, G \in \mathbb{F}_\Sigma$ und gilt außerdem
   $$FV(F) \cap BV(G) = \{\} \quad \text{und} \quad FV(G) \cap BV(F) = \{\},$$
   so gilt auch
   (a) $(F \wedge G) \in \mathbb{F}_\Sigma$,
   (b) $(F \vee G) \in \mathbb{F}_\Sigma$,
   (c) $(F \to G) \in \mathbb{F}_\Sigma$,
   (d) $(F \leftrightarrow G) \in \mathbb{F}_\Sigma$.

   Weiter definieren wir für alle Junktoren $\odot \in \{\wedge, \vee, \to, \leftrightarrow\}$:
   (a) $FV\big(F \odot G\big) := FV(F) \cup FV(G)$.
   (b) $BV\big(F \odot G\big) := BV(F) \cup BV(G)$.
5. Sei $x \in \mathcal{V}$ und $F \in \mathbb{F}_\Sigma$ mit $x \notin BV(F)$. Dann gilt:
   (a) $(\forall x\colon F) \in \mathbb{F}_\Sigma$.
   (b) $(\exists x\colon F) \in \mathbb{F}_\Sigma$.

   Weiter definieren wir
   (a) $FV\big((\forall x\colon F)\big) := FV\big((\exists x\colon F)\big) := FV(F) \backslash \{x\}$.
   (b) $BV\big((\forall x\colon F)\big) := BV\big((\exists x\colon F)\big) := BV(F) \cup \{x\}$.

Ist die Signatur Σ aus dem Zusammenhang klar oder aber unwichtig, so schreiben wir auch $\mathbb{F}$ statt $\mathbb{F}_\Sigma$ und sprechen dann einfach von Formeln statt von Σ-Formeln. □

Bei der oben gegebenen Definition haben wir darauf geachtet, dass eine Variable nicht gleichzeitig frei und gebunden in einer Formel auftreten kann, denn durch eine leichte Induktion nach dem Aufbau der Formeln läßt sich zeigen, dass für alle $F \in \mathbb{F}_\Sigma$ folgendes gilt:

$$FV(F) \cap BV(F) = \{\}.$$

Beispiel: Setzen wir das oben begonnene Beispiel fort, so sehen wir, dass

$$(\exists x\colon \leq(+(y,x),y))$$

eine Formel aus $\mathbb{F}_{\Sigma_{\text{arith}}}$ ist. Die Menge der gebundenen Variablen ist $\{x\}$, die Menge der freien Variablen ist $\{y\}$.

Wenn wir Formeln immer in dieser Form anschreiben würden, dann würde die Lesbarkeit unverhältnismäßig leiden. Zur Abkürzung vereinbaren wir, dass in der Prädikatenlogik die selben Regeln zur Klammer-Ersparnis gelten sollen, die wir schon in der Aussagenlogik verwendet haben. Zusätzlich werden gleiche Quantoren zusammengefaßt: Beispielsweise schreiben wir

$$\forall x, y\colon p(x,y) \quad \text{statt} \quad \forall x\colon (\forall y\colon p(x,y)).$$

Darüber hinaus legen wir fest, dass Quantoren stärker binden als die aussagenlogischen Junktoren. Damit können wir

$$\forall x\colon p(x) \wedge G \quad \text{als} \quad \big(\forall x\colon p(x)\big) \wedge G$$

schreiben. Außerdem vereinbaren wir, dass wir zweistellige Prädikats- und Funktions-Zeichen auch in Infix-Notation angeben dürfen. Um eine eindeutige Lesbarkeit zu erhalten, müssen wir dann gegebenenfalls Klammern setzen. Wir schreiben beispielsweise

$$\mathtt{n}_1 = \mathtt{n}_2 \quad \text{anstelle von} \quad =(\mathtt{n}_1, \mathtt{n}_2).$$

Die Formel $(\exists x\colon \leq (+(y,x),y))$ wird dann lesbarer als

$$\exists x\colon y + x \leq y$$

geschrieben. Außerdem finden Sie in der Literatur häufig Ausdrücke der Form $\forall x \in M : F$ oder $\exists x \in M : F$. Hierbei handelt es sich um Abkürzungen, die wie folgt definiert sind:

$$\big(\forall x \in M : F\big) \stackrel{def}{\Longleftrightarrow} \forall x : \big(x \in M \rightarrow F\big),$$

$$\big(\exists x \in M : F\big) \stackrel{def}{\Longleftrightarrow} \exists x : \big(x \in M \wedge F\big).$$

5.2 Semantik der Prädikatenlogik

Als nächstes legen wir die Bedeutung der Formeln fest. Dazu definieren wir mit Hilfe der Mengenlehre den Begriff einer Σ-*Struktur*. Eine solche Struktur legt fest, wie die Funktions- und Prädikats-Zeichen der Signatur Σ zu interpretieren sind.

Definition 40 (Struktur) Es sei eine Signatur

$$\Sigma = \langle \mathcal{V}, \mathcal{F}, \mathcal{P}, \mathit{arity} \rangle.$$

gegeben. Eine Σ-*Struktur* $\mathcal{S}$ ist ein Paar $\langle \mathcal{U}, \mathcal{J} \rangle$, so dass folgendes gilt:

1. $\mathcal{U}$ ist eine nicht-leere Menge. Diese Menge nennen wir auch das *Universum* der Σ-Struktur. Dieses Universum enthält die Werte, die sich später bei der Auswertung der Terme ergeben werden.

2. $\mathcal{J}$ ist die *Interpretation* der Funktions– und Prädikats-Zeichen. Formal definieren wir $\mathcal{J}$ als eine Abbildung mit folgenden Eigenschaften:

   (a) Jedem Funktions-Zeichen $f \in \mathcal{F}$ mit $\mathit{arity}(f) = m$ wird eine m-stellige Funktion

   $$f^{\mathcal{J}}\colon \mathcal{U} \times \cdots \times \mathcal{U} \rightarrow \mathcal{U}$$

   zugeordnet, die m-Tupel des Universums $\mathcal{U}$ in das Universum $\mathcal{U}$ abbildet.

(b) Jedem Prädikats-Zeichen $p \in \mathcal{P}$ mit $arity(p) = n$ wird eine n-stelliges Prädikat

$$p^{\mathcal{J}} : \mathcal{U} \times \cdots \times \mathcal{U} \to \mathbb{B}$$

zugeordnet, die jedem n-Tupel des Universums $\mathcal{U}$ einen Wahrheitswert aus der Menge $\mathbb{B} = \{\texttt{true}, \texttt{false}\}$ zuordnet.

(c) Ist das Zeichen "=" ein Element der Menge der Prädikats-Zeichen $\mathcal{P}$, so gilt

$$=^{\mathcal{J}} (u, v) = \texttt{true} \quad \text{g.d.w.} \quad u = v,$$

das Gleichheits-Zeichen wird also durch die identische Relation $\mathrm{id}_{\mathcal{U}}$ interpretiert.

Beispiel: Wir geben ein Beispiel für eine Σ_{arith}-Struktur $\mathcal{S}_{\text{arith}} = \langle \mathcal{U}_{\text{arith}}, \mathcal{J}_{\text{arith}} \rangle$, indem wir definieren:

1. $\mathcal{U}_{\text{arith}} = \mathbb{N}$.
2. Die Abbildung $\mathcal{J}_{\text{arith}}$ legen wir dadurch fest, dass die Funktions-Zeichen 0, 1, $+$, $-$, $\cdot$ durch die entsprechend benannten Funktionen auf der Menge $\mathbb{N}$ der natürlichen Zahlen zu interpretieren sind.

   Ebenso sollen die Prädikats-Zeichen $=$ und $\leq$ durch die Gleichheits-Relation und die Kleiner-Gleich-Relation interpretiert werden.

Beispiel: Wir geben ein weiteres Beispiel. Die Signatur Σ_G der Gruppen-Theorie sei definiert als

$$\Sigma_G = \langle \mathcal{V}, \mathcal{F}, \mathcal{P}, arity \rangle \quad \text{mit}$$

1. $\mathcal{V} := \{x, y, z\}$
2. $\mathcal{F} := \{1, *\}$
3. $\mathcal{P} := \{=\}$
4. $arity = \big\{\langle 1, 0\rangle, \langle *, 2\rangle, \langle =, 2\rangle\big\}$

Dann können wir eine Σ_G Struktur $\mathcal{Z} = \langle \{a, b\}, \mathcal{J} \rangle$ definieren, indem wir die Interpretation $\mathcal{J}$ wie folgt festlegen:

1. $1^{\mathcal{J}} := a$
2. $*^{\mathcal{J}} := \Big\{\big\langle\langle a, a\rangle, a\big\rangle, \big\langle\langle a, b\rangle, b\big\rangle, \big\langle\langle b, a\rangle, b\big\rangle, \big\langle\langle b, b\rangle, a\big\rangle\Big\}$
3. $=^{\mathcal{J}}$ ist die Identität:
   $=^{\mathcal{J}} := \Big\{\big\langle\langle a, a\rangle, \texttt{true}\big\rangle, \big\langle\langle a, b\rangle, \texttt{false}\big\rangle, \big\langle\langle b, a\rangle, \texttt{false}\big\rangle, \big\langle\langle b, b\rangle, \texttt{true}\big\rangle\Big\}$
   Beachten Sie, dass wir bei der Interpretation des Gleichheits-Zeichens keinen Spielraum haben!

Falls wir Terme auswerten wollen, die Variablen enthalten, so müssen wir für diese Variablen igendwelche Werte aus dem Universum einsetzen. Welche Werte wir einsetzen, kann durch eine *Variablen-Belegung* festgelegt werden. Diesen Begriff definieren wir nun.

Definition 41 (Variablen-Belegung) Es sei eine Signatur

$$\Sigma = \langle \mathcal{V}, \mathcal{F}, \mathcal{P}, arity \rangle$$

gegeben. Weiter sei $\mathcal{S} = \langle \mathcal{U}, \mathcal{J} \rangle$ eine Σ-Struktur. Dann bezeichnen wir eine Abbildung

$$\mathcal{I} : \mathcal{V} \to \mathcal{U}$$

als eine *$\mathcal{S}$-Variablen-Belegung*.

Ist $\mathcal{I}$ eine $\mathcal{S}$-Variablen-Belegung, $x \in \mathcal{V}$ und $c \in \mathcal{U}$, so bezeichnet $\mathcal{I}[x/c]$ die Variablen-Belegung, die der Variablen x den Wert c zuordnet und die ansonsten mit $\mathcal{I}$ übereinstimmt:

$$\mathcal{I}[x/c](y) := \begin{cases} c & \text{falls } y = x; \\ \mathcal{I}(y) & \text{sonst.} \end{cases}$$

□

Definition 42 (Semantik der Terme) Ist $\mathcal{S} = \langle \mathcal{U}, \mathcal{J} \rangle$ eine Σ-Struktur und $\mathcal{I}$ eine $\mathcal{S}$-Variablen-Belegung, so definieren wir für jeden Term t den *Wert* $\mathcal{S}(\mathcal{I}, t)$ durch Induktion über den Aufbau von t:

1. Für Variablen $x \in \mathcal{V}$ definieren wir:

   $$\mathcal{S}(\mathcal{I}, x) := \mathcal{I}(x).$$

2. Für Σ-Terme der Form $f(t_1, \cdots, t_n)$ definieren wir

   $$\mathcal{S}\big(\mathcal{I}, f(t_1, \cdots, t_n)\big) := f^{\mathcal{J}}\big(\mathcal{S}(\mathcal{I}, t_1), \cdots, \mathcal{S}(\mathcal{I}, t_n)\big).$$ □

Beispiel: Mit der oben definieren Σ_{arith}-Struktur $\mathcal{S}_{\text{arith}}$ definieren wir eine $\mathcal{S}_{\text{arith}}$-Variablen-Belegung $\mathcal{I}$ durch

$$\mathcal{I} := \big\{\langle x, 0\rangle, \langle y, 7\rangle, \langle z, 42\rangle\big\},$$

es gilt also

$$\mathcal{I}(\mathtt{x}) := 0, \quad \mathcal{I}(\mathtt{y}) := 7, \quad \text{und} \quad \mathcal{I}(\mathtt{z}) := 42.$$

Dann gilt offenbar

$$\mathcal{S}\big(\mathcal{I}, \mathtt{x} + \mathtt{y}\big) = 7.$$

Definition 43 (Semantik der atomaren Σ-Formeln) Ist $\mathcal{S}$ eine Σ-Struktur und $\mathcal{I}$ eine $\mathcal{S}$-Variablen-Belegung, so definieren wir für jede atomare Σ-Formel $p(t_1, \cdots, t_n)$ den Wert
$\mathcal{S}\big(\mathcal{I}, p(t_1, \cdots, t_n)\big)$ wie folgt:

$$\mathcal{S}\big(\mathcal{I}, p(t_1, \cdots, t_n)\big) := p^{\mathcal{J}}\big(\mathcal{S}(\mathcal{I}, t_1), \cdots, \mathcal{S}(\mathcal{I}, t_n)\big).$$ □

Beispiel: In Fortführung des obigen Beispiels gilt:

$$\mathcal{S}\big(\mathcal{I}, z \leq x + y\big) = \mathtt{false}.$$

Um die Semantik beliebiger Σ-Formeln definieren zu können, nehmen wir an, dass wir, genau wie in der Aussagenlogik, die folgenden Funktionen zur Verfügung haben:

1. $\ominus\colon \mathbb{B} \to \mathbb{B}$,
2. $\oslash\colon \mathbb{B} \times \mathbb{B} \to \mathbb{B}$,
3. $\owedge\colon \mathbb{B} \times \mathbb{B} \to \mathbb{B}$,
4. $\oplus : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$,
5. $\otimes : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$.

Die Semantik dieser Funktionen hatten wir durch die Tabelle in Abbildung 4.1 auf Seite 74 gegeben.

Definition 44 (Semantik der Σ-Formeln) Ist $\mathcal{S}$ eine Σ-Struktur und $\mathcal{I}$ eine $\mathcal{S}$-Variablen-Belegung, so definieren wir für jede Σ-Formel F den Wert $\mathcal{S}(\mathcal{I}, F)$ durch Induktion über den Aufbau von F:

1. $\mathcal{S}(\mathcal{I}, \top) := \mathtt{true}$ und $\mathcal{S}(\mathcal{I}, \bot) := \mathtt{false}$.
2. $\mathcal{S}(\mathcal{I}, \neg F) := \ominus\big(\mathcal{S}(\mathcal{I}, F)\big)$.
3. $\mathcal{S}(\mathcal{I}, F \wedge G) := \owedge\big(\mathcal{S}(\mathcal{I}, F), \mathcal{S}(\mathcal{I}, G)\big)$.
4. $\mathcal{S}(\mathcal{I}, F \vee G) := \oslash\big(\mathcal{S}(\mathcal{I}, F), \mathcal{S}(\mathcal{I}, G)\big)$.
5. $\mathcal{S}(\mathcal{I}, F \to G) := \oplus\big(\mathcal{S}(\mathcal{I}, F), \mathcal{S}(\mathcal{I}, G)\big)$.
6. $\mathcal{S}(\mathcal{I}, F \leftrightarrow G) := \otimes\big(\mathcal{S}(\mathcal{I}, F), \mathcal{S}(\mathcal{I}, G)\big)$.
7. $\mathcal{S}\big(\mathcal{I}, \forall x\colon F\big) := \begin{cases} \mathtt{true} & \text{falls } \mathcal{S}(\mathcal{I}[x/c], F) = \mathtt{true} \quad \text{für alle } c \in \mathcal{U} \text{ gilt;} \\ \mathtt{false} & \text{sonst.} \end{cases}$

8. $\mathcal{S}\big(\mathcal{I}, \exists x\colon F\big) := \begin{cases} \texttt{true} & \text{falls } \mathcal{S}(\mathcal{I}[x/c], F) = \texttt{true} \quad \text{für ein } c \in \mathcal{U} \text{ gilt;} \\ \texttt{false} & \text{sonst.} \end{cases}$ □

Beispiel: In Fortführung des obigen Beispiels gilt

$$\mathcal{S}\big(\mathcal{I}, \forall \texttt{x} : x * 0 < 1\big) = \texttt{true}.$$

Definition 45 (Allgemeingültig) Ist F eine Σ-Formel, so dass für jede Σ-Struktur $\mathcal{S}$ und für jede $\mathcal{S}$-Variablen-Belegung $\mathcal{I}$

$$\mathcal{S}(\mathcal{I}, F) = \texttt{true}$$

gilt, so bezeichnen wir F als *allgemeingültig*. In diesem Fall schreiben wir

$$\models F.$$ □

Ist F eine Formel für die $FV(F) = \{\}$ ist, dann hängt der Wert $\mathcal{S}(\mathcal{I}, F)$ offenbar gar nicht von der Interpretation $\mathcal{I}$ ab. Solche Formeln bezeichnen wir auch als *geschlossene* Formeln. In diesem Fall schreiben wir kürzer $\mathcal{S}(F)$ an Stelle von $\mathcal{S}(\mathcal{I}, F)$. Gilt dann zusätzlich $\mathcal{S}(F) = \texttt{true}$, so sagen wir auch dass $\mathcal{S}$ ein *Modell* von F ist. Wir schreiben dann

$$\mathcal{S} \models F.$$

Die Definition der Begriffe "*erfüllbar*" und "*äquivalent*" lassen sich nun aus der Aussagenlogik übertragen. Um unnötigen Ballast in den Definitionen zu vermeiden, nehmen wir im folgenden immer eine feste Signatur Σ als gegeben an. Dadurch können wir in den folgenden Definitionen von Termen, Formeln, Strukturen, etc. sprechen und meinen damit Σ-Terme, Σ-Formeln und Σ-Strukturen.

Definition 46 (Äquivalent) Zwei Formeln F und G heißen *äquivalent* g.d.w. gilt

$$\models F \leftrightarrow G$$ □

Alle aussagenlogischen Äquivalenzen sind auch prädikatenlogische Äquivalenzen.

Definition 47 (Erfüllbar) Eine Menge $M \subseteq \mathbb{F}_\Sigma$ ist genau dann *erfüllbar*, wenn es eine Struktur $\mathcal{S}$ und eine Variablen-Belegung $\mathcal{I}$ gibt, so dass

$$\forall m \in M : \mathcal{S}(\mathcal{I}, m) = \texttt{true}$$

gilt. Andernfalls heißt M *unerfüllbar* oder auch *widersprüchlich*. Wir schreiben dafür auch

$$M \models \bot$$ □

Unser Ziel ist es, ein Verfahren anzugeben, mit dem wir in der Lage sind zu überprüfen, ob eine Menge M von Formeln *widersprüchlich* ist, ob also $M \models \bot$ gilt. Es zeigt sich, dass dies im Allgemeinen nicht möglich ist, die Frage, ob $M \models \bot$ gilt, ist unentscheidbar. Ein Beweis dieser Tatsache geht allerdings über den Rahmen dieser Vorlesung heraus. Dem gegenüber ist es möglich, ähnlich wie in der Aussagenlogik einen *Kalkül* $\vdash$ anzugeben, so dass gilt

$$M \vdash \bot \quad \text{g.d.w.} \quad M \models \bot.$$

Ein solcher Kalkül kann dann zur Implementierung eines *Semi-Entscheidungs-Verfahrens* benutzt werden: Um zu überprüfen, ob $M \models \bot$ gilt, versuchen wir, aus der Menge M die Formel $\bot$ herzuleiten. Falls wir dabei systematisch vorgehen, indem wir alle möglichen Beweise durchprobieren, so werden wir, falls tatsächlich $M \models \bot$ gilt, auch irgendwann einen Beweis finden, der $M \vdash \bot$ zeigt. Wenn allerdings der Fall

$$M \not\models \bot$$

vorliegt, so werden wir dies im allgemeinen nicht feststellen können, denn die Menge aller Beweise ist unendlich groß und wir können nie alle Beweise ausprobieren. Wir können lediglich sicherstellen, dass wir jeden Beweis irgendwann versuchen. Wenn es aber keinen Beweis gibt, so können wir das nie sicher sagen, denn zu jedem festen Zeitpunkt haben wir ja immer nur einen Teil der in Frage kommenden Schlüsse ausprobiert.

Die Situation ist ähnlich der, wie bei der Überprüfung bestimmter zahlentheoretischer Fragen. Wir betrachten dazu ein konkretes Beispiel: Eine Zahl n heißt *perfekt*, wenn die Summe aller echten Teiler von n wieder die

Zahl n ergibt. Beispielsweise ist die Zahl 6 perfekt, denn die Menge der echten Teiler von 6 ist $\{1, 2, 3\}$ und es gilt

$$1 + 2 + 3 = 6.$$

Bisher sind alle bekannten perfekten Zahlen durch 2 teilbar. Die Frage, ob es auch ungerade Zahlen gibt, die perfekt sind, ist ein offenes mathematisches Problem. Um dieses Problem zu lösen könnten wir eine Programm schreiben, dass der Reihe nach für alle ungerade Zahlen überprüft, ob die Zahl perfekt ist. Abbildung 5.1 auf Seite 118 zeigt ein solches Programm. Wenn es eine ungerade perfekte Zahl gibt, dann wird dieses Programm diese Zahl auch irgendwann finden. Wenn es aber keine ungerade perfekte Zahl gibt, dann wird das Programm bis zum St. Nimmerleinstag rechnen und wir werden nie mit Sicherheit wissen, dass es keine ungeraden perfekten Zahlen gibt.

```
1  perfect := procedure(n) {q
2      return +/ { x in {1 .. n-1} | n % x == 0 } == n;
3  };
4
5  findPerfect := procedure() {
6      n := 1;
7      while (true) {
8          if (perfect(n)) {
9              if (n % 2 == 0) {
10                 print(n);
11             } else {
12                 print("Heureka: Odd perfect number $n$ found!");
13             }
14         }
15         n := n + 1;
16     }
17 };
18
19 findPerfect();
```

Abbildung 5.1: Suche nach einer ungeraden perfekten Zahl.

In den nächsten Abschnitten gehen wir daran, den oben erwähnten Kalkül $\vdash$ zu definieren. Es zeigt sich, dass die Arbeit wesentlich einfacher wird, wenn wir uns auf bestimmte Formeln, sogenannte *Klauseln*, beschränken. Wir zeigen daher zunächst im nächsten Abschnitt, dass jede Formel-Menge M so in eine Menge von Klauseln K transformiert werden kann, dass M genau dann erfüllbar ist, wenn K erfüllbar ist. Daher ist die Beschränkung auf Klauseln keine echte Einschränkung.

5.2.1 Implementierung prädikatenlogischer Strukturen in SetlX

Der im letzten Abschnitt präsentierte Begriff einer prädikatenlogischen Struktur erscheint zunächst sehr abstrakt. Wir wollen in diesem Abschnitt zeigen, dass sich dieser Begriff in einfacher Weise in SetlX implementieren läßt. Dadurch gelingt es, diesen Begriff zu veranschaulichen. Als konkretes Beispiel wollen wir Strukturen zu Gruppen-Theorie betrachten. Die Signatur Σ_G der Gruppen-Theorie war im letzten Abschnitt durch die Definition

$$\Sigma_G = \big\langle \{x, y, z\},\ \{1, *\},\ \{=\},\ \{\langle 1, 0\rangle, \langle *, 2\rangle, \langle =, 2\rangle\}\big\rangle$$

gegeben worden. Hierbei ist also "1" ein 0-stelliges Funktions-Zeichen, "$*$" ist eine 2-stelliges Funktions-Zeichen und "=" ist ein 2-stelliges Prädikats-Zeichen. Wir hatten bereits eine Struktur $\mathcal{S}$ angegeben, deren Universum aus der Menge $\{a, b\}$ besteht. In *SetlX* können wir diese Struktur durch den in Abbildung 5.2 gezeigten Code implementieren.

```
1 a := "a";
2 b := "b";
3 u := { a, b };  // the universe
4 product := { [ [ a, a ], a ],  [ [ a, b ], b ],  [ [ b, a ], b ],  [ [ b, b ], a ] };
5 equal := { [ x, y ] : x in u, y in u | x == y };
6 j := { [ "E", a ], [ "^product", product ], [ "^equal", equal ] };
7 s := [ u, j ];
8 i := { [ "x", a ], [ "y", b ], [ "z", a ] };
```

Abbildung 5.2: Implementierung einer Struktur zur Gruppen-Theorie

1. Zur Abkürzung haben wir in den Zeile 1 und 2 die Variablen a und b als die Strings `"a"` und `"b"` definiert. Dadurch können wir weiter unten die Interpretation des Funktions-Zeichens "$*$" kürzer angeben.

2. Das in Zeile 3 definierte Universum u besteht aus den beiden Strings `"a"` und `"b"`.

3. In Zeile 4 definieren wir eine Funktion `multiply` als binäre Relation. Für die so definierte Funktion gilt

   $$\texttt{multiply}(\langle\texttt{"a"},\texttt{"a"}\rangle) = \texttt{"a"}, \quad \texttt{multiply}(\langle\texttt{"a"},\texttt{"b"}\rangle) = \texttt{"b"},$$
   $$\texttt{multiply}(\langle\texttt{"b"},\texttt{"a"}\rangle) = \texttt{"b"}, \quad \texttt{multiply}(\langle\texttt{"b"},\texttt{"b"}\rangle) = \texttt{"a"}.$$

   Diese Funktion verwenden wir später als die Interpretation $*^{\mathcal{J}}$ des Funktions-Zeichens "$*$".

4. Ebenso haben wir in Zeile 5 die Interpretation $=^{\mathcal{J}}$ des Prädikats-Zeichens "$=$" als die binäre Relation `equal` dargestellt.

5. In Zeile 6 fassen wir die einzelnen Interpretationen zu der Relation j zusammen, so dass für ein Funktions-Zeichen f die Interpretation $f^{\mathcal{J}}$ durch den Wert $j(f)$ gegeben ist.

   Da wir später den in SetlX eingebauten Parser verwenden werden, stellen wir den Operator "`*`" durch das Funktions-Zeichen "`^multiply`" dar und das Prädikats-Zeichen "`=`" wird durch das Zeichen "`^equal`" dargestellt, denn dies sind die Namen, die von SetlX intern benutzt werden. Das neutrale Element "1" stellen wir durch das Funktions-Zeichen "`E`" dar, so dass später der Ausdruck "1" durch den Term "`E()`" repräsentiert wird.

6. Die Interpretation j wird dann in Zeile 7 mit dem Universum u zu der Struktur s zusammen gefasst.

7. Schließlich zeigt Zeile 8, dass eine Variablen-Belegung ebenfalls als Relation dargestellt werden kann. Die erste Komponente der Paare, aus denen diese Relation besteht, sind die Variablen. Die zweite Komponente ist ein Wert aus dem Universum.

Als nächstes überlegen wir uns, wie wir prädikatenlogische Formeln in einer solchen Struktur auswerten können. Abbildung 5.3 zeigt die Implementierung der Prozedur *evalFormula*(f, S, I), der als Argumente eine prädikatenlogische Formel f, eine Struktur $\mathcal{S}$ und eine Variablen-Belegung $\mathcal{I}$ übergeben werden. Die Formel wird dabei als Term dargestellt, ganz ähnlich, wie wir das bei der Implementierung der Aussagenlogik schon praktiziert haben. Beispielsweise können wir die Formel

$$\forall x : \forall y : x * y = y * x$$

durch den Term

```
^forall(^variable("x"), ^forall(^variable("y"),
     ^equal(^product(^variable("x"), ^variable("y")),
            ^product(^variable("y"), ^variable("x"))
           )))
```

```
1  evalFormula := procedure(f, s, i) {
2      u := s[1];
3      match (f) {
4          case true     :  return true;
5          case false    :  return false;
6          case !g       :  return !evalFormula(g, s, i);
7          case g && h   :  return evalFormula(g, s, i) && evalFormula(h, s, i);
8          case g || h   :  return evalFormula(g, s, i) || evalFormula(h, s, i);
9          case g => h   :  return evalFormula(g, s, i) => evalFormula(h, s, i);
10         case g <==> h :  return evalFormula(g, s, i) == evalFormula(h, s, i);
11         case forall (x in _ | g) :
12              return forall (c in u | evalFormula(g, s, modify(i, x, c)));
13         case exists (x in _ | g) :
14              return exists (c in u | evalFormula(g, s, modify(i, x, c)));
15         default : return evalAtomic(f, s, i);  // atomic formula
16     }
17 };
```

Abbildung 5.3: Auswertung prädikatenlogischer Formeln

darstellen und dass ist im Wesentlichen auch die Struktur, die erzeugt wird, wenn wir den String

"`forall (x in u | exists (y in u | x * y == E()))`"

mit Hilfe der in SetlX vordefinierten Funktion `parse` in einen Term umwandeln.

Bemerkung: An dieser Stelle wundern Sie sich vermutlich, warum wir oben "`x in u`" und "`y in u`" schreiben, denn wir wollen die Variablen `x` und `y` eigentlich ja gar nicht einschränken. Der Grund ist, dass die Syntax von SetlX nur solche Quantoren erlaubt, in denen die Variablen auf eine Menge eingeschränkt sind. Daher sind wir gezwungen, bei der Verwendung von Quantoren die Variablen immer auf eine Menge einzuschränken. Wir schreiben deswegen

`forall (x in u | `g`)` bzw. `exists (x in u | `g`)`

an Stelle von

`forall (x | `g`)` bzw. `exists (x | `g`)`.

Wir fassen dann `u` als das Universum aller Objekte auf, so dass die Variablen nur syntaktisch, aber nicht semantisch eingeschränkt werden. ◇

Die Auswertung einer prädikatenlogischen Formel ist nun analog zu der in Abbildung 4.1 auf Seite 78 gezeigten Auswertung aussagenlogischer Formeln. Neu ist nur die Behandlung der Quantoren. In den Zeilen 11 und 12 behandeln wir die Auswertung allquantifizierter Formeln. Ist f eine Formel der Form $\forall y \in u\colon h$, so wird die Formel f durch den Term

$f =$ `^forall(`y, u, h`)`

dargestellt. Das Muster

`forall (x in _ | g)`

bindet daher `x` an die tatsächlich auftretende Variable y und `g` an die Teilformel h. Die Auswertung von $\forall x\colon g$ geschieht nach der Formel

$$\mathcal{S}\big(\mathcal{I}, \forall x\colon g\big) \;:=\; \begin{cases} \texttt{true} & \text{falls } \mathcal{S}(\mathcal{I}[x/c], g) = \texttt{true} \quad \text{für alle } c \in \mathcal{U} \text{ gilt;} \\ \texttt{false} & \text{sonst.} \end{cases}$$

Um die Auswertung implementieren zu können, verwenden wir eine Prozedur *modify*(), welche die Variablen-Belegung i an der Stelle x zu c abändert, es gilt also

$$\mathit{modify}(\mathcal{I}, x, c) = \mathcal{I}[x/c].$$

Die Implementierung dieser Prozedur wird später in Abbildung 5.4 gezeigt. Bei der Auswertung eines All-Quantors können wir ausnutzen, dass die Sprache SetlX selber den Quantor `forall` unterstützt. Wir können also direkt testen, ob die Formel für alle möglichen Werte c, die wir für die Variable x einsetzen können, richtig ist. Die Auswertung eines Existenz-Quantors ist analog zur Auswertung eines All-Quantors.

```
1  evalAtomic := procedure(a, s, i) {
2      j  := s[2];
3      p  := fct(a); // predicate symbol
4      pJ := j[p];
5      argList := args(a);
6      argsVal := evalTermList(argList, s, i);
7      return argsVal in pJ;
8  };
9  evalTerm := procedure(t, s, i) {
10     if (fct(t) == "^variable") {
11         varName := args(t)[1];
12         return i[varName];
13     }
14     j    := s[2];
15     f    := fct(t); // function symbol
16     fJ   := j[f];
17     argList := args(t);
18     argsVal := evalTermList(argList, s, i);
19     if (#argsVal > 0) {
20         result := fJ[argsVal];
21     } else {
22         result := fJ;   // t is a constant
23     }
24     return result;
25 };
26 evalTermList := procedure(tl, s, i) {
27     return [ evalTerm(t, s, i) : t in tl ];
28 };
29 modify := procedure(i, v, c) {
30     x := args(v)[1];  // v = ^variable(x)
31     i[x] := c;
32     return i;
33 };
```

Abbildung 5.4: Auswertung von Termen und atomaren Formeln.

Abbildung 5.4 zeigt die Auswertung atomarer Formeln und prädikatenlogischer Terme. Um eine atomare Formel der Form

$$a = P(t_1, \cdots, t_n)$$

auszuwerten, verschaffen wir uns in Zeile 4 zunächst die dem Prädikats-Zeichen P in der Struktur S zugeordnete Menge pJ. Anschließend werten wir die Argumente $t_1, \cdots, t_n$ aus und überprüfen dann, ob das Ergebnis dieser Auswertung tatsächlich ein Element der Menge pJ ist.

Die Prozedur `evalTerm`() arbeitet wie folgt: Das erste Argument t der Prozedur `evalTerm`$(t, \mathcal{S}, \mathcal{I})$ ist der auszuwertende Term. Das zweite Argument $\mathcal{S}$ ist eine prädikatenlogische Struktur und das dritte Argument $\mathcal{I}$ ist eine Variablen-Belegung.

1. Falls t eine Variable ist, so geben wir in Zeile 12 einfach den Wert zurück, der in der Variablen-Belegung $\mathcal{I}$ für diese Variable eingetragen ist. Die Variablen-Belegung wird dabei durch eine zweistellige Relation dargestellt, die wir als Funktion benutzen.

2. Falls der auszuwertende Term t die Form

   $$t = F(t_1, \cdots, t_n)$$

   hat, werden in Zeile 18 zunächst rekursiv die Subterme t_1, $\cdots$, t_n ausgewertet. Anschließend wird die Interpretation $F^{\mathcal{J}}$ des Funktions-Zeichens F herangezogen, um die Funktion $F^{\mathcal{J}}$ für die gegebenen Argumente auszuwerten, wobei in Zeile 20 der Fall betrachtet wird, dass tatsächlich Argumente vorhanden sind, während in Zeile 22 der Fall behandelt wird, dass es sich bei dem Funktions-Zeichen F um eine Konstante handelt, deren Wert dann unmittelbar durch $F^{\mathcal{J}}$ gegeben ist.

Die Implementierung der Prozedur `evalTermList`() wendet die Funktion `evalTerm`() auf alle Terme der gegebenen Liste an. Bei der Implementierung der in Zeile 31 gezeigten Prozedur `modify`(I, x, c), die als Ergebnis die Variablen-Belegung $\mathcal{I}[x/c]$ berechnet, nutzen wir aus, dass wir bei einer Funktion, die als binäre Relation gespeichert ist, den Wert, der in dieser Relation für ein Argument x eingetragen ist, durch eine Zuweisung der Form $\mathcal{I}(x){:=}c$ abändern können.

```
1 g1 := parse("forall (x in u | x * E() == x)");
2 g2 := parse("forall (x in u | exists (y in u | x * y == E()))");
3 g3 := parse("forall (x in u | forall (y in u | forall (z in u | (x*y)*z == x*(y*z) )))");
4 gt := { g1, g2, g3 };
5
6 print("checking group theory in the structure ", s);
7 for (f in gt) {
8     print( "checking ", f, ": ", evalFormula(f, s, i) );
9 }
```

Abbildung 5.5: Axiome der Gruppen-Theorie

Wir zeigen nun, wie sich die in Abbildung 5.3 gezeigte Funktion `evalFormula`$(f, \mathcal{S}, \mathcal{I})$ benutzen läßt um zu überprüfen, ob die in Abbildung 5.2 gezeigte Struktur die Axiome der *Gruppen-Theorie* erfüllt. Die Axiome der Gruppen-Theorie sind wie folgt:

1. Die Konstante 1 ist das rechts-neutrale Element der Multiplikation:

   $$\forall x\colon x * 1 = x.$$

2. Für jedes Element x gibt es ein rechts-inverses Element y, dass mit dem Element x multipliziert die 1 ergibt:

   $$\forall x\colon \exists y\colon x * y = 1.$$

3. Es gilt das Assoziativ-Gesetz:

   $$\forall x\colon \forall y\colon \forall z\colon (x * y) * z = x * (y * z).$$

Diese Axiome sind in den Zeilen 1 bis 3 der Abbildung 5.5 wiedergegeben, wobei wir die "1" durch das Funktions-Zeichen "`E`" dargestellt haben. Die Schleife in den Zeilen 7 bis 9 überprüft schließlich, ob die Formeln in der oben definierten Struktur erfüllt sind.

Bemerkung: Mit dem oben vorgestellten Programm können wir überprüfen, ob eine prädikatenlogische Formel in einer vorgegebenen endlichen Struktur erfüllt ist. Wir können damit allerdings nicht überprüfen, ob eine Formel allgemeingültig ist, denn einerseits können wir das Programm nicht anwenden, wenn die Strukturen ein unendliches Universum haben, andererseits ist selbst die Zahl der verschiedenen endlichen Stukturen, die wir ausprobieren müßten, unendlich groß.

5.3 Normalformen für prädikatenlogische Formeln

In diesem Abschnitt werden wir verschieden Möglichkeiten zur Umformung prädikatenlogischer Formeln kennenlernen. Zunächst geben wir einige Äquivalenzen an, mit deren Hilfe Quantoren manipuliert werden können.

Satz 48 Es gelten die folgenden Äquivalenzen:

1. $\models \neg\big(\forall x\colon f\big) \leftrightarrow \big(\exists x\colon \neg f\big)$
2. $\models \neg\big(\exists x\colon f\big) \leftrightarrow \big(\forall x\colon \neg f\big)$
3. $\models \big(\forall x\colon f\big) \wedge \big(\forall x\colon g\big) \leftrightarrow \big(\forall x\colon f \wedge g\big)$
4. $\models \big(\exists x\colon f\big) \vee \big(\exists x\colon g\big) \leftrightarrow \big(\exists x\colon f \vee g\big)$
5. $\models \big(\forall x\colon \forall y\colon f\big) \leftrightarrow \big(\forall y\colon \forall x\colon f\big)$
6. $\models \big(\exists x\colon \exists y\colon f\big) \leftrightarrow \big(\exists y\colon \exists x\colon f\big)$
7. Falls x eine Variable ist, für die $x \notin FV(f)$ ist, so haben wir
   $$\models \big(\forall x\colon f\big) \leftrightarrow f \quad \text{und} \quad \models \big(\exists x\colon f\big) \leftrightarrow f.$$
8. Falls x eine Variable ist, für die $x \notin FV(g) \cup BV(g)$ gilt, so haben wir die folgenden Äquivalenzen:
   (a) $\models \big(\forall x\colon f\big) \vee g \leftrightarrow \forall x\colon (f \vee g)$
   (b) $\models g \vee \big(\forall x\colon f\big) \leftrightarrow \forall x\colon (g \vee f)$
   (c) $\models \big(\exists x\colon f\big) \wedge g \leftrightarrow \exists x\colon (f \wedge g)$
   (d) $\models g \wedge \big(\exists x\colon f\big) \leftrightarrow \exists x\colon (g \wedge f)$

Um die Äquivalenzen der letzten Gruppe anwenden zu können, ist es notwendig, gebundene Variablen umzubenennen. Ist f eine prädikatenlogische Formel und sind x und y zwei Variablen, so bezeichnet $f[x/y]$ die Formel, die aus f dadurch entsteht, dass jedes Auftreten der Variablen x in f durch y ersetzt wird. Beispielsweise gilt

$$\big(\forall u : \exists v : p(u,v)\big)[u/z] = \forall z : \exists v : p(z,v)$$

Damit können wir eine letzte Äquivalenz angeben: Ist f eine prädikatenlogische Formel, ist $x \in BV(F)$ und ist y eine Variable, die in f nicht auftritt, so gilt

$$\models f \leftrightarrow f[x/y].$$

Mit Hilfe der oben stehenden Äquivalenzen können wir eine Formel so umformen, dass die Quantoren nur noch außen stehen. Eine solche Formel ist dann in *pränexer Normalform*. Wir führen das Verfahren an einem Beispiel vor: Wir zeigen, dass die Formel

$$\big(\forall x\colon p(x)\big) \rightarrow \big(\exists x\colon p(x)\big)$$

allgemeingültig ist:

$$\begin{array}{ll} & \big(\forall x\colon p(x)\big) \rightarrow \big(\exists x\colon p(x)\big) \\ \leftrightarrow & \neg\big(\forall x\colon p(x)\big) \vee \big(\exists x\colon p(x)\big) \\ \leftrightarrow & \big(\exists x\colon \neg p(x)\big) \vee \big(\exists x\colon p(x)\big) \\ \leftrightarrow & \exists x\colon \big(\neg p(x) \vee p(x)\big) \\ \leftrightarrow & \exists x\colon \top \\ \leftrightarrow & \top \end{array}$$

Um Formeln noch stärker normalisieren zu können, führen wir einen weiteren Äquivalenz-Begriff ein. Diesen Begriff wollen wir vorher durch ein Beispiel motivieren. Wir betrachten die beiden Formeln

$$f_1 = \forall x\colon \exists y\colon p(x,y) \quad \text{und} \quad f_2 = \forall x\colon p\big(x, s(x)\big).$$

Die beiden Formeln f_1 und f_2 sind nicht äquivalent, denn sie entstammen noch nicht einmal der gleichen Signatur: In der Formel f_2 wird das Funktions-Zeichen s verwendet, das in der Formel f_1 überhaupt nicht auftritt. Auch wenn die beiden Formeln f_1 und f_2 nicht äquivalent sind, so besteht zwischen ihnen doch die folgende Beziehung: Ist S_1 eine prädikatenlogische Struktur, in der die Formel f_1 gilt:

$$\mathcal{S}_1 \models f_1,$$

dann können wir diese Struktur zu einer Struktur S_2 erweitern, in der die Formel f_2 gilt:

$$\mathcal{S}_2 \models f_2.$$

Dazu muss lediglich die Interpretation des Funktions-Zeichens s so gewählt werden, dass für jedes x tatsächlich $p\big(x, s(x)\big)$ gilt. Dies ist möglich, denn die Formel f_1 sagt ja aus, dass wir tatsächlich zu jedem x einen Wert y finden, für den $p(x,y)$ gilt. Die Funktion s muss also lediglich zu jedem x dieses y zurück geben.

Definition 49 (Skolemisierung) Es sei $\Sigma = \langle \mathcal{V}, \mathcal{F}, \mathcal{P}, \mathit{arity} \rangle$ eine Signatur. Ferner sei f eine geschlossene Σ-Formel der Form

$$f = \forall x_1, \cdots, x_n\colon \exists y\colon g(x_1, \cdots, x_n, y).$$

Dann wählen wir ein neues n-stelliges Funktions-Zeichen s, d.h. wir nehmen ein Zeichen s, dass in der Menge $\mathcal{F}$ nicht auftritt und erweitern die Signatur Σ zu der Signatur

$$\Sigma' := \Big\langle \mathcal{V}, \mathcal{F} \cup \{s\}, \mathcal{P}, \mathit{arity} \cup \big\{\langle s, n\rangle\big\} \Big\rangle,$$

in der wir s als neues n-stelliges Funktions-Zeichen deklarieren. Anschließend definieren wir die Σ'-Formel f' wie folgt:

$$f' := \mathit{Skolem}(f) := \forall x_1\colon \cdots \forall x_n\colon g\big(x_1, \cdots, x_n, s(x_1, \cdots, x_n)\big)$$

Wir lassen also den Existenz-Quantor $\exists y$ weg und ersetzen jedes Auftreten der Variable y durch den Term $s(x_1, \cdots, x_n)$. Wir sagen, dass die Formel f' aus der Formel f durch einen Skolemisierungs-Schritt hervorgegangen ist. □

In welchem Sinne sind eine Formel f und eine Formel f', die aus f durch einen Skolemisierungs-Schritt hervorgegangen sind, äquivalent? Zur Beantwortung dieser Frage dient die folgende Definition.

Definition 50 (Erfüllbarkeits-Äquivalenz) Zwei geschlossene Formeln f und g heißen *erfüllbarkeits-äquivalent* falls f und g entweder beide erfüllbar oder beide unerfüllbar sind. Wenn f und g erfüllbarkeits-äquivalent sind, so schreiben wir

$$f \approx_e g.$$ □

Satz 51 Falls die Formel f' aus der Formel f durch einen Skolemisierungs-Schritt hervorgegangen ist, so sind f und f' erfüllbarkeits-äquivalent.

Wir können nun ein einfaches Verfahren angeben, um Existenz-Quantoren aus einer Formel zu eliminieren. Dieses Verfahren besteht aus zwei Schritten: Zunächst bringen wir die Formel in pränexe Normalform. Anschließend können wir die Existenz-Quantoren der Reihe nach durch Skolemisierungs-Schritte eliminieren. Nach dem eben gezeigten Satz ist die resultierende Formel zu der ursprünglichen Formel erfüllbarkeits-äquivalent. Dieses Verfahren der Eliminierung von Existenz-Quantoren durch die Einführung neuer Funktions-Zeichen wird als *Skolemisierung* bezeichnet. Haben wir eine Formel F in pränexe Normalform gebracht und anschließend skolemisiert, so hat das Ergebnis die Gestalt

$$\forall x_1, \cdots, x_n : g$$

und in der Formel g treten keine Quantoren mehr auf. Die Formel g wird auch als die *Matrix* der obigen Formel bezeichnet. Wir können nun g mit Hilfe der uns aus dem letzten Kapitel bekannten aussagenlogischen Äquivalenzen in konjunktive Normalform bringen. Wir haben dann eine Formel der Gestalt

$$\forall x_1, \cdots, x_n : (k_1 \wedge \cdots \wedge k_m).$$

Dabei sind die k_i Disjunktionen von *Literalen*. (In der Prädikatenlogik ist ein Literal entweder eine atomare Formel oder die Negation einer atomaren Formel.) Wenden wir hier die Äquivalenz $(\forall x\colon f_1 \wedge f_2) \leftrightarrow (\forall x\colon f_1) \wedge (\forall x\colon f_2)$ an, so können wir die All-Quantoren auf die einzelnen k_i verteilen und die resultierende Formel hat die Gestalt

$$\big(\forall x_1, \cdots, x_n : k_1\big) \wedge \cdots \wedge \big(\forall x_1, \cdots, x_n : k_m\big).$$

Ist eine Formel F in der obigen Gestalt, so sagen wir, dass F in *prädikatenlogischer Klausel-Normalform* ist und eine Formel der Gestalt

$$\forall x_1, \cdots, x_n : k,$$

bei der k eine Disjunktion prädikatenlogischer Literale ist, bezeichnen wir als *prädikatenlogische Klausel*. Ist M eine Menge von Formeln deren Erfüllbarkeit wir untersuchen wollen, so können wir nach dem bisher gezeigten M immer in eine Menge prädikatenlogischer Klauseln umformen. Da dann nur noch All-Quantoren vorkommen, können wir hier die Notation noch vereinfachen indem wir vereinbaren, dass alle Formeln implizit allquantifiziert sind, wir lassen also die All-Quantoren weg.

Wozu sind nun die Umformungen in Skolem-Normalform gut? Es geht darum, dass wir ein Verfahren entwickeln wollen, mit dem es möglich ist für eine prädikatenlogische Formel f zu zeigen, dass f allgemeingültig ist, dass also

$$\models f$$

gilt. Wir wissen, dass

$$\models f \quad \text{g.d.w.} \quad \{\neg f\} \models \bot$$

gilt, denn die Formel f ist genau dann allgemeingültig, wenn es keine Struktur gibt, in der die Formel $\neg f$ erfüllbar ist. Wir bilden daher zunächst $\neg f$ und formen $\neg f$ in prädikatenlogische Klausel-Normalform um. Wir erhalten dann soetwas wie

$$\neg f \approx_e k_1 \wedge \cdots \wedge k_n.$$

Dabei sind $k_1, \cdots, k_n$ prädikatenlogische Klauseln. Anschließend versuchen wir, aus den Klauseln $k_1, \cdots, k_n$ eine Widerspruch herzuleiten:

$$\{k_1, \cdots, k_n\} \vdash \bot$$

Wenn dies gelingt wissen wir, dass die Menge $\{k_1, \cdots, k_n\}$ unerfüllbar ist. Dann ist auch $\neg f$ unerfüllbar und damit ist dann f allgemeingültig. Damit wir aus den Klauseln $k_1, \cdots, k_n$ einen Widerspruch herleiten können, brauchen wir natürlich noch einen Kalkül, der mit prädikatenlogischen Klauseln arbeitet. Einen solchen Kalkül werden wir am Ende dieses Kapitel vorstellen.

Um das Verfahren näher zu erläutern demonstrieren wir es an einem Beispiel. Wir wollen untersuchen, ob

$$\models \big(\exists x\colon \forall y\colon p(x,y)\big) \rightarrow \big(\forall y\colon \exists x\colon p(x,y)\big)$$

gilt. Wir wissen, dass dies äquivalent dazu ist, dass

$$\Big\{\neg\Big(\big(\exists x\colon \forall y\colon p(x,y)\big) \to \big(\forall y\colon \exists x\colon p(x,y)\big)\Big)\Big\} \models \bot$$

gilt. Wir bringen zunächst die negierte Formel in pränexe Normalform.

$$\begin{array}{ll} & \neg\Big(\big(\exists x\colon \forall y\colon p(x,y)\big) \to \big(\forall y\colon \exists x\colon p(x,y)\big)\Big) \\ \leftrightarrow & \neg\Big(\neg\big(\exists x\colon \forall y\colon p(x,y)\big) \vee \big(\forall y\colon \exists x\colon p(x,y)\big)\Big) \\ \leftrightarrow & \big(\exists x\colon \forall y\colon p(x,y)\big) \wedge \neg\big(\forall y\colon \exists x\colon p(x,y)\big) \\ \leftrightarrow & \big(\exists x\colon \forall y\colon p(x,y)\big) \wedge \big(\exists y\colon \neg\exists x\colon p(x,y)\big) \\ \leftrightarrow & \big(\exists x\colon \forall y\colon p(x,y)\big) \wedge \big(\exists y\colon \forall x\colon \neg p(x,y)\big) \end{array}$$

Um an dieser Stelle weitermachen zu können, ist es nötig, die Variablen in dem zweiten Glied der Konjunktion umzubenennen. Wir ersetzen x durch u und y durch v und erhalten

$$\begin{array}{ll} & \big(\exists x\colon \forall y\colon p(x,y)\big) \wedge \big(\exists y\colon \forall x\colon \neg p(x,y)\big) \\ \leftrightarrow & \big(\exists x\colon \forall y\colon p(x,y)\big) \wedge \big(\exists v\colon \forall u\colon \neg p(u,v)\big) \\ \leftrightarrow & \exists v\colon \Big(\big(\exists x\colon \forall y\colon p(x,y)\big) \wedge \big(\forall u\colon \neg p(u,v)\big)\Big) \\ \leftrightarrow & \exists v\colon \exists x\colon \Big(\big(\forall y\colon p(x,y)\big) \wedge \big(\forall u\colon \neg p(u,v)\big)\Big) \\ \leftrightarrow & \exists v\colon \exists x\colon \forall y\colon \Big(p(x,y) \wedge \big(\forall u\colon \neg p(u,v)\big)\Big) \\ \leftrightarrow & \exists v\colon \exists x\colon \forall y\colon \forall u\colon \Big(p(x,y) \wedge \neg p(u,v)\Big) \end{array}$$

An dieser Stelle müssen wir skolemisieren um die Existenz-Quantoren los zu werden. Wir führen dazu zwei neue Funktions-Zeichen s_1 und s_2 ein. Dabei gilt $\texttt{arity}(s_1) = 0$ und $\texttt{arity}(s_2) = 0$, denn vor den Existenz-Quantoren stehen keine All-Quantoren.

$$\begin{array}{ll} & \exists v\colon \exists x\colon \forall y\colon \forall u\colon \Big(p(x,y) \wedge \neg p(u,v)\Big) \\ \approx_e & \exists x\colon \forall y\colon \forall u\colon \Big(p(x,y) \wedge \neg p(u,s_1)\Big) \\ \approx_e & \forall y\colon \forall u\colon \Big(p(s_2,y) \wedge \neg p(u,s_1)\Big) \end{array}$$

Da jetzt nur noch All-Quantoren auftreten, können wir diese auch noch weglassen, da wir ja vereinbart haben, dass alle freien Variablen implizit allquantifiziert sind. Damit können wir nun die prädikatenlogische Klausel-Normalform angeben, diese ist

$$M := \Big\{\big\{p(s_2,y)\big\}, \big\{\neg p(u,s_1)\big\}\Big\}.$$

Wir zeigen nun, dass die Menge M widersprüchlich ist. Dazu betrachten wir zunächst die Klausel $\big\{p(s_2,y)\big\}$ und setzen in dieser Klausel für y die Konstante s_1 ein. Damit erhalten wir die Klausel

$$\big\{p(s_2,s_1)\big\}. \tag{1}$$

Das Ersetzung von y durch s_1 begründen wir damit, dass die obige Klausel ja implizit allquantifiziert ist und wenn etwas für alle y gilt, dann sicher auch für $y = s_1$.

Als nächstes betrachten wir die Klausel $\big\{\neg p(u,s_1)\big\}$. Hier setzen wir für die Variablen u die Konstante s_2 ein und erhalten dann die Klausel

$$\big\{\neg p(s_2,s_1)\big\} \tag{2}$$

Nun wenden wir auf die Klauseln (1) und (2) die Schnitt-Regel an und finden

$$\big\{p(s_2,s_1)\big\}, \quad \big\{\neg p(s_2,s_1)\big\} \quad \vdash \quad \{\}.$$

Damit haben wir einen Widerspruch hergeleitet und gezeigt, dass die Menge M unerfüllbar ist. Damit ist dann auch

$$\Big\{\neg\Big(\big(\exists x\colon \forall y\colon p(x,y)\big) \to \big(\forall y\colon \exists x\colon p(x,y)\big)\Big)\Big\}$$

unerfüllbar und folglich gilt

$$\models \big(\exists x\colon \forall y\colon p(x,y)\big) \to \big(\forall y\colon \exists x\colon p(x,y)\big).$$

5.4 Unifikation

In dem Beispiel im letzten Abschnitt haben wir die Terme s_1 und s_2 geraten, die wir für die Variablen y und u in den Klauseln $\big\{p(s_2, y)\big\}$ und $\big\{\neg p(u, s_1)\big\}$ eigesetzt haben. Wir haben diese Terme mit dem Ziel gewählt, später die Schnitt-Regel anwenden zu können. In diesem Abschnitt zeigen wir nun ein Verfahren, mit dessen Hilfe wir die benötigten Terme ausrechnen können. Dazu benötigen wir zunächst den Begriff einer *Substitution*.

Definition 52 (Substitution) Es sei eine Signatur

$$\Sigma = \langle \mathcal{V}, \mathcal{F}, \mathcal{P}, \mathit{arity} \rangle$$

gegeben. Eine Σ-Substitution ist eine endliche Menge von Paaren der Form

$$\sigma = \big\{\langle x_1, t_1\rangle, \cdots, \langle x_n, t_n\rangle\big\}.$$

Dabei gilt:

1. $x_i \in \mathcal{V}$, die x_i sind also Variablen.
2. $t_i \in \mathcal{T}_\Sigma$, die t_i sind also Terme.
3. Für $i \neq j$ ist $x_i \neq x_j$, die Variablen sind also paarweise verschieden.

Ist $\sigma = \big\{\langle x_1, t_1\rangle, \cdots, \langle x_n, t_n\rangle\big\}$ eine Σ-Substitution, so schreiben wir

$$\sigma = \big[x_1 \mapsto t_1, \cdots, x_n \mapsto t_n\big].$$

Außerdem definieren wir den *Domain* einer Substitution als

$$\mathit{dom}(\sigma) = \{x_1, \cdots, x_n\}.$$

Die Menge aller Substitutionen bezeichnen wir mit *Subst*. □

Substitutionen werden für uns dadurch interessant, dass wir sie auf Terme *anwenden* können. Ist t ein Term und σ eine Substitution, so ist $t\sigma$ der Term, der aus t dadurch entsteht, dass jedes Vorkommen einer Variablen x_i durch den zugehörigen Term t_i ersetzt wird. Die formale Definition folgt.

Definition 53 (Anwendung einer Substitution)
Es sei t ein Term und es sei $\sigma = \big[x_1 \mapsto t_1, \cdots, x_n \mapsto t_n\big]$ eine Substitution. Wir definieren die *Anwendung* von σ auf t (Schreibweise $t\sigma$) durch Induktion über den Aufbau von t:

1. Falls t eine Variable ist, gibt es zwei Fälle:
   (a) $t = x_i$ für ein $i \in \{1, \cdots, n\}$. Dann definieren wir $\quad x_i\sigma := t_i$.
   (b) $t = y$ mit $y \in \mathcal{V}$, aber $y \notin \{x_1, \cdots, x_n\}$. Dann definieren wir $\quad y\sigma := y$.
2. Andernfalls muß t die Form $t = f(s_1, \cdots, s_m)$ haben. Dann können wir $t\sigma$ durch
   $$f(s_1, \cdots, s_m)\sigma := f(s_1\sigma, \cdots, s_m\sigma).$$
   definieren, denn nach Induktions-Voraussetzung sind die Ausdrücke $s_i\sigma$ bereits definiert. □

Genau wie wir Substitutionen auf Terme anwenden können, können wir eine Substitution auch auf prädikatenlogische Klauseln anwenden. Dabei werden Prädikats-Zeichen und Junktoren wie Funktions-Zeichen behandelt. Wir ersparen uns eine formale Definition und geben statt dessen zunächst einige Beispiele. Wir definieren eine Substitution σ durch

$$\sigma := \big[x_1 \mapsto c,\ x_2 \mapsto f(d)\big].$$

In den folgenden drei Beispielen demonstrieren wir zunächst, wie eine Substitution auf einen Term angewendet werden kann. Im vierten Beispiel wenden wir die Substitution dann auf eine Formel an:

1. $x_3\sigma = x_3$,
2. $f(x_2)\sigma = f\big(f(d)\big)$,
3. $h(x_1, g(x_2))\sigma = h\big(c, g(f(d))\big)$.
4. $\big\{p(x_2), q(d, h(x_3, x_1))\big\}\sigma = \big\{p(f(d)),\ q(d, h(x_3, c))\big\}$.

Als nächstes zeigen wir, wie Substitutionen miteinander verknüpft werden können.

Definition 54 (Komposition von Substitutionen) Es seien

$$\sigma = \big[x_1 \mapsto s_1, \cdots, x_m \mapsto s_m\big] \quad \text{und} \quad \tau = \big[y_1 \mapsto t_1, \cdots, y_n \mapsto t_n\big]$$

zwei Substitutionen mit $dom(\sigma) \cap dom(\tau) = \{\}$. Dann definieren wir die *Komposition* $\sigma\tau$ von σ und τ als

$$\sigma\tau := \big[x_1 \mapsto s_1\tau, \cdots, x_m \mapsto s_m\tau,\ y_1 \mapsto t_1, \cdots, y_n \mapsto t_n\big] \qquad \square$$

Beispiel: Wir führen das obige Beispiel fort und setzen

$$\sigma := \big[x_1 \mapsto c,\ x_2 \mapsto f(x_3)\big] \quad \text{und} \quad \tau := \big[x_3 \mapsto h(c,c),\ x_4 \mapsto d\big].$$

Dann gilt:

$$\sigma\tau = \big[x_1 \mapsto c,\ x_2 \mapsto f(h(c,c)),\ x_3 \mapsto h(c,c),\ x_4 \mapsto d\big]. \qquad \square$$

Die Definition der Komposition von Substitutionen ist mit dem Ziel gewählt worden, dass der folgende Satz gilt.

Satz 55 Ist t ein Term und sind σ und τ Substitutionen mit $dom(\sigma) \cap dom(\tau) = \{\}$, so gilt

$$(t\sigma)\tau = t(\sigma\tau). \qquad \square$$

Der Satz kann durch Induktion über den Aufbau des Termes t bewiesen werden.

Definition 56 (Syntaktische Gleichung) Unter einer *syntaktischen Gleichung* verstehen wir in diesem Abschnitt ein Konstrukt der Form $s \doteq t$, wobei einer der beiden folgenden Fälle vorliegen muß:

1. s und t sind Terme oder
2. s und t sind atomare Formeln.

Weiter definieren wir ein *syntaktisches Gleichungs-System* als eine Menge von syntaktischen Gleichungen. $\square$

Was syntaktische Gleichungen angeht machen wir keinen Unterschied zwischen Funktions-Zeichen und Prädikats-Zeichen. Dieser Ansatz ist deswegen berechtigt, weil wir Prädikats-Zeichen ja auch als spezielle Funktions-Zeichen auffassen können, nämlich als Funktions-Zeichen, die einen Wahrheitswert aus der Menge $\mathbb{B}$ berechnen.

Definition 57 (Unifikator) Eine Substitution σ *löst* eine syntaktische Gleichung $s \doteq t$ genau dann, wenn $s\sigma = t\sigma$ ist, wenn also durch die Anwendung von σ auf s und t tatsächlich identische Objekte entstehen. Ist E ein syntaktisches Gleichungs-System, so sagen wir, dass σ ein *Unifikator* von E ist wenn σ jede syntaktische Gleichung in E löst. $\square$

Ist $E = \{s_1 \doteq t_1, \cdots, s_n \doteq t_n\}$ eine syntaktisches Gleichungs-System und ist σ eine Substitution, so definieren wir

$$E\sigma := \{s_1\sigma \doteq t_1\sigma, \cdots, s_n\sigma \doteq t_n\sigma\}.$$

Beispiel: Wir verdeutlichen die bisher eingeführten Begriffe anhand eines Beispiels. Wir betrachten die Gleichung

$$p(x_1, f(x_4)) \doteq p(x_2, x_3)$$

und definieren die Substitution

$$\sigma := \big[x_1 \mapsto x_2,\ x_3 \mapsto f(x_4)\big].$$

Die Substitution σ löst die obige syntaktische Gleichung, denn es gilt

$$\begin{aligned} p(x_1, f(x_4))\sigma &= p(x_2, f(x_4)) \quad \text{und} \\ p(x_2, x_3)\sigma &= p(x_2, f(x_4)). \end{aligned}$$

Als nächstes entwickeln wir ein Verfahren, mit dessen Hilfe wir von einer vorgegebenen Menge E von syntaktischen Gleichungen entscheiden können, ob es einen Unifikator σ für E gibt. Wir überlegen uns zunächst, in welchen Fällen wir eine syntaktischen Gleichung $s \doteq t$ garantiert nicht lösen können. Da gibt es zwei Möglichkeiten: Eine syntaktische Gleichung

$$f(s_1, \cdots, s_m) \doteq g(t_1, \cdots, t_n)$$

ist sicher dann nicht durch eine Substitution lösbar, wenn f und g verschiedene Funktions-Zeichen sind, denn für jede Substitution σ gilt ja

$$f(s_1, \cdots, s_m)\sigma = f(s_1\sigma, \cdots, s_m\sigma) \quad \text{und} \quad g(t_1, \cdots, t_n)\sigma = g(t_1\sigma, \cdots, t_n\sigma).$$

Falls $f \neq g$ ist, haben die Terme $f(s_1, \cdots, s_m)\sigma$ und $g(t_1, \cdots, t_n)\sigma$ verschieden Funktions-Zeichen und können daher syntaktisch nicht identisch werden.

Die andere Form einer syntaktischen Gleichung, die garantiert unlösbar ist, ist

$$x \doteq f(t_1, \cdots, t_n) \quad \text{falls } x \in \mathit{Var}\big(f(t_1, \cdots, t_n)\big).$$

Das diese syntaktische Gleichung unlösbar ist liegt daran, dass die rechte Seite immer mindestens ein Funktions-Zeichen mehr enthält als die linke.

Mit diesen Vorbemerkungen können wir nun ein Verfahren angeben, mit dessen Hilfe es möglich ist, Mengen von syntaktischen Gleichungen zu lösen, oder festzustellen, dass es keine Lösung gibt. Das Verfahren operiert auf Paaren der Form $\langle F, \tau\rangle$. Dabei ist F ein syntaktisches Gleichungs-System und τ ist eine Substitution. Wir starten das Verfahren mit dem Paar $\langle E, []\rangle$. Hierbei ist E das zu lösende Gleichungs-System und $[]$ ist die leere Substitution. Das Verfahren arbeitet indem die im folgenden dargestellten Reduktions-Regeln solange angewendet werden, bis entweder feststeht, dass die Menge der Gleichungen keine Lösung hat, oder aber ein Paar der Form $\langle\{\}, \sigma\rangle$ erreicht wird. In diesem Fall ist σ ein Unifikator der Menge E, mit der wir gestartet sind. Es folgen die Reduktions-Regeln:

1. Falls $y \in \mathcal{V}$ eine Variable ist, die nicht in dem Term t auftritt, so können wir die folgende Reduktion durchführen:

   $$\Big\langle E \cup \big\{y \doteq t\big\}, \sigma \Big\rangle \quad \rightsquigarrow \quad \Big\langle E[y \mapsto t], \sigma\big[y \mapsto t\big] \Big\rangle$$

   Diese Reduktions-Regel ist folgendermaßen zu lesen: Enthält die zu untersuchende Menge von syntaktischen Gleichungen eine Gleichung der Form $y \doteq t$, wobei die Variable y nicht in t auftritt, dann können wir diese Gleichung aus der gegebenen Menge von Gleichungen entfernen. Gleichzeitig wird die Substitution σ in die Substitution $\sigma\big[y \mapsto t\big]$ transformiert und auf die restlichen syntaktischen Gleichungen wird die Substitution $[y \mapsto t]$ angewendet.

2. Wenn die Variable y in dem Term t auftritt, falls also $y \in \mathit{var}(t)$ ist und wenn außerdem $t \neq y$ ist, dann hat das Gleichungs-System $E \cup \big\{y \doteq t\big\}$ keine Lösung, wir schreiben

   $$\Big\langle E \cup \big\{y \doteq t\big\}, \sigma \Big\rangle \rightsquigarrow \Omega.$$

3. Falls $y \in \mathcal{V}$ eine Variable ist und t keine Variable ist, so haben wir folgende Reduktions-Regel:

   $$\Big\langle E \cup \big\{t \doteq y\big\}, \sigma \Big\rangle \quad \rightsquigarrow \quad \Big\langle E \cup \big\{y \doteq t\big\}, \sigma \Big\rangle.$$

   Diese Regel wird benötigt, um anschließend eine der ersten beiden Regeln anwenden zu können.

4. Triviale syntaktische Gleichungen von Variablen können wir einfach weglassen:

$$\Big\langle E \cup \big\{x \doteq x\big\}, \sigma \Big\rangle \quad \leadsto \quad \Big\langle E, \sigma \Big\rangle.$$

5. Ist f ein n-stelliges Funktions-Zeichen, so gilt

$$\Big\langle E \cup \big\{f(s_1, \cdots, s_n) \doteq f(t_1, \cdots, t_n)\big\}, \sigma \Big\rangle \leadsto \Big\langle E \cup \big\{s_1 \doteq t_1, \cdots, s_n \doteq t_n\big\}, \sigma \Big\rangle.$$

   Eine syntaktische Gleichung der Form $f(s_1, \cdots, s_n) \doteq f(t_1, \cdots, t_n)$ wird also ersetzt durch die n syntaktische Gleichungen $s_1 \doteq t_1$, $\cdots$, $s_n \doteq t_n$.

   Diese Regel ist im Übrigen der Grund dafür, dass wir mit Mengen von syntaktischen Gleichungen arbeiten müssen, denn auch wenn wir mit nur einer syntaktischen Gleichung starten, kann durch die Anwendung dieser Regel die Zahl der syntaktischen Gleichungen erhöht werden.

   Ein Spezialfall dieser Regel ist

$$\Big\langle E \cup \big\{c \doteq c\big\}, \sigma \Big\rangle \leadsto \Big\langle E, \sigma \Big\rangle.$$

   Hier steht c für eine Konstante, also ein 0-stelliges Funktions-Zeichen. Triviale Gleichungen über Konstanten können also einfach weggelassen werden.

6. Das Gleichungs-System $E \cup \big\{f(s_1, \cdots, s_m) \doteq g(t_1, \cdots, t_n)\big\}$ hat keine Lösung, falls die Funktions-Zeichen f und g verschieden sind, wir schreiben

$$\Big\langle E \cup \big\{f(s_1, \cdots, s_m) \doteq g(t_1, \cdots, t_n)\big\}, \sigma \Big\rangle \leadsto \Omega \qquad \text{falls } f \neq g.$$

Haben wir ein nicht-leeres Gleichungs-System E gegeben und starten mit dem Paar $\langle E, [] \rangle$, so läßt sich immer eine der obigen Regeln anwenden. Diese geht solange bis einer der folgenden Fälle eintritt:

1. Die 2. oder 6. Regel ist anwendbar. Dann ist das Ergebnis Ω und das Gleichungs-System E hat keine Lösung.
2. Das Paar $\langle E, [] \rangle$ wird reduziert zu einem Paar $\langle \{\}, \sigma \rangle$. Dann ist σ ein Unifikator von E. In diesem Falls schreiben wir $\sigma = mgu(E)$. Falls $E = \{s \doteq t\}$ ist, schreiben wir auch $\sigma = mgu(s, t)$. Die Abkürzung *mgu* steht hier für "*most general unifier*".

Beispiel: Wir wenden das oben dargestellte Verfahren an, um die syntaktische Gleichung

$$p(x_1, f(x_4)) \doteq p(x_2, x_3)$$

zu lösen. Wir haben die folgenden Reduktions-Schritte:

$$\begin{array}{ll} & \big\langle \big\{p(x_1, f(x_4)) \doteq p(x_2, x_3)\big\}, [] \big\rangle \\ \leadsto & \big\langle \big\{x_1 \doteq x_2, f(x_4) \doteq x_3\big\}, [] \big\rangle \\ \leadsto & \big\langle \big\{f(x_4) \doteq x_3\big\}, \big[x_1 \mapsto x_2\big] \big\rangle \\ \leadsto & \big\langle \big\{x_3 \doteq f(x_4)\big\}, \big[x_1 \mapsto x_2\big] \big\rangle \\ \leadsto & \big\langle \big\{\big\}, \big[x_1 \mapsto x_2,\ x_3 \mapsto f(x_4)\big] \big\rangle \end{array}$$

In diesem Fall ist das Verfahren also erfolgreich und wir erhalten die Substitution

$$\big[x_1 \mapsto x_2,\ x_3 \mapsto f(x_4)\big]$$

als Lösung der oben gegebenen syntaktischen Gleichung.

Wir geben ein weiteres Beispiel und betrachten das Gleichungs-System

$$E = \big\{p(h(x_1, c)) \doteq p(x_2),\ q(x_2, d) \doteq q(h(d, c), x_4)\big\}$$

Wir haben folgende Reduktions-Schritte:

$$\begin{array}{rl}
 & \big\langle \big\{p(h(x_1, c)) \doteq p(x_2),\ q(x_2, d) \doteq q(h(d, c), x_4)\big\}, \big[\big]\big\rangle \\
\leadsto & \big\langle \big\{p(h(x_1, c)) \doteq p(x_2),\ x_2 \doteq h(d, c),\ d \doteq x_4\big\}, \big[\big]\big\rangle \\
\leadsto & \big\langle \big\{p(h(x_1, c)) \doteq p(x_2),\ x_2 \doteq h(d, c),\ x_4 \doteq d\big\}, \big[\big]\big\rangle \\
\leadsto & \big\langle \big\{p(h(x_1, c)) \doteq p(x_2),\ x_2 \doteq h(d, c)\big\}, \big[x_4 \mapsto d\big]\big\rangle \\
\leadsto & \big\langle \big\{p(h(x_1, c)) \doteq p(h(d, c))\big\}, \big[x_4 \mapsto d,\ x_2 \mapsto h(d, c)\big]\big\rangle \\
\leadsto & \big\langle \big\{h(x_1, c) \doteq h(d, c)\big\}, \big[x_4 \mapsto d,\ x_2 \mapsto h(d, c)\big]\big\rangle \\
\leadsto & \big\langle \big\{x_1 \doteq d,\ c \doteq c\big\}, \big[x_4 \mapsto d,\ x_2 \mapsto h(d, c)\big]\big\rangle \\
\leadsto & \big\langle \big\{x_1 \doteq d, \big\}, \big[x_4 \mapsto d,\ x_2 \mapsto h(d, c)\big]\big\rangle \\
\leadsto & \big\langle \big\{\big\}, \big[x_4 \mapsto d,\ x_2 \mapsto h(d, c),\ x_1 \mapsto d\big]\big\rangle
\end{array}$$

Damit haben wir die Substitution $\big[x_4 \mapsto d,\ x_2 \mapsto h(d, c),\ x_1 \mapsto d\big]$ als Lösung des anfangs gegebenen syntaktischen Gleichungs-Systems gefunden. □

5.5 Ein Kalkül für die Prädikatenlogik

Der Kalkül, den wir in diesem Abschnitt für die Prädikatenlogik einführen, besteht aus zwei Schluß-Regeln, die wir jetzt definieren.

Definition 58 (Resolution) Es gelte:

1. k_1 und k_2 sind prädikatenlogische Klauseln,
2. $p(s_1, \cdots, s_n)$ und $p(t_1, \cdots, t_n)$ sind atomare Formeln,
3. die syntaktische Gleichung $p(s_1, \cdots, s_n) \doteq p(t_1, \cdots, t_n)$ ist lösbar mit

$$\mu = mgu\big(p(s_1, \cdots, s_n), p(t_1, \cdots, t_n)\big).$$

Dann ist

$$\frac{k_1 \cup \{p(s_1, \cdots, s_n)\} \qquad \{\neg p(t_1, \cdots, t_n)\} \cup k_2}{k_1\mu \cup k_2\mu}$$

eine Anwendung der *Resolutions-Regel*. □

Die Resolutions-Regel ist eine Kombination aus der *Substitutions-Regel* und der Schnitt-Regel. Die Substitutions-Regel hat die Form

$$\frac{k}{k\sigma}.$$

Hierbei ist k eine prädikatenlogische Klausel und σ ist eine Substitution. Unter Umständen kann es sein, dass wir bei der Anwendung der Resolutions-Regel die Variablen in einer der beiden Klauseln erst umbenennen müssen bevor wir die Regel anwenden können. Betrachten wir dazu ein Beispiel. Die Klausel-Menge

$$M = \Big\{\big\{p(x)\big\}, \big\{\neg p(f(x))\big\}\Big\}$$

ist widersprüchlich. Wir können die Resolutions-Regel aber nicht unmittelbar anwenden, denn die syntaktische Gleichung

$$p(x) \doteq p(f(x))$$

ist unlösbar. Das liegt daran, dass **zufällig** in beiden Klauseln die selbe Variable verwendet wird. Wenn wir die Variable x in der zweiten Klausel jedoch zu y umbenennen, erhalten wir die Klausel-Menge

$$\Big\{\big\{p(x)\big\}, \big\{\neg p(f(y))\big\}\Big\}.$$

Hier können wir die Resolutions-Regel anwenden, denn die syntaktische Gleichung

$$p(x) \doteq p(f(y))$$

hat die Lösung $[x \mapsto f(y)]$. Dann erhalten wir

$$\big\{p(x)\big\}, \quad \big\{\neg p(f(y))\big\} \quad \vdash \quad \{\}.$$

und haben damit die Inkonsistenz der Klausel-Menge M nachgewiesen.

Die Resolutions-Regel alleine ist nicht ausreichend, um aus einer Klausel-Menge M, die inkonsistent ist, in jedem Fall die leere Klausel ableiten zu können: Wir brauchen noch eine zweite Regel. Um das einzusehen, betrachten wir die Klausel-Menge

$$M = \Big\{\big\{p(f(x), y), p(u, g(v))\big\}, \big\{\neg p(f(x), y), \neg p(u, g(v))\big\}\Big\}$$

Wir werden gleich zeigen, dass die Menge M widersprüchlich ist. Man kann nachweisen, dass mit der Resolutions-Regel alleine ein solcher Nachweis nicht gelingt. Ein einfacher, aber für die Vorlesung zu aufwendiger Nachweis dieser Behauptung kann geführt werden, indem wir ausgehend von der Menge alle möglichen Resolutions-Schritte durchführen. Dabei würden wir dann sehen, dass die leere Klausel nie berechnet wird. Wir stellen daher jetzt die Faktorisierungs-Regel vor, mir der wir dann später zeigen werden, dass M widersprüchlich ist.

Definition 59 (Faktorisierung) *Es gelte*

1. k ist eine prädikatenlogische Klausel,
2. $p(s_1, \cdots, s_n)$ und $p(t_1, \cdots, t_n)$ sind atomare Formeln,
3. die syntaktische Gleichung $p(s_1, \cdots, s_n) \doteq p(t_1, \cdots, t_n)$ ist lösbar,
4. $\mu = mgu\big(p(s_1, \cdots, s_n), p(t_1, \cdots, t_n)\big)$.

Dann sind

$$\frac{k \cup \big\{p(s_1, \cdots, s_n),\, p(t_1, \cdots, t_n)\big\}}{k\mu \cup \big\{p(s_1, \cdots, s_n)\mu\big\}} \quad \text{und} \quad \frac{k \cup \big\{\neg p(s_1, \cdots, s_n),\, \neg p(t_1, \cdots, t_n)\big\}}{k\mu \cup \big\{\neg p(s_1, \cdots, s_n)\mu\big\}}$$

Anwendungen der *Faktorisierungs-Regel*. □

Wir zeigen, wie sich mit Resolutions- und Faktorisierungs-Regel die Widersprüchlichkeit der Menge M beweisen läßt.

1. Zunächst wenden wir die Faktorisierungs-Regel auf die erste Klausel an. Dazu berechnen wir den Unifikator

   $$\mu = mgu\big(p(f(x), y), p(u, g(v))\big) = [y \mapsto g(v), u \mapsto f(x)].$$

   Damit können wir die Faktorisierungs-Regel anwenden:

   $$\big\{p(f(x), y), p(u, g(v))\big\} \quad \vdash \quad \big\{p(f(x), g(v))\big\}.$$

2. Jetzt wenden wir die Faktorisierungs-Regel auf die zweite Klausel an. Dazu berechnen wir den Unifikator

$$\mu = mgu\big(\neg p(f(x), y), \neg p(u, g(v))\big) = [y \mapsto g(v), u \mapsto f(x)].$$

   Damit können wir die Faktorisierungs-Regel anwenden:

$$\big\{\neg p(f(x), y), \neg p(u, g(v))\big\} \quad \vdash \quad \big\{\neg p(f(x), g(v))\big\}.$$

3. Wir schließen den Beweis mit einer Anwendung der Resolutions-Regel ab. Der dabei verwendete Unifikator ist die leere Substitution, es gilt also $\mu = []$.

$$\big\{p(f(x), g(v))\big\}, \quad \big\{\neg p(f(x), g(v))\big\} \quad \vdash \quad \{\}.$$

Ist M eine Menge von prädikatenlogischen Klauseln und ist k eine prädikatenlogische Klausel, die durch Anwendung der Resolutions-Regel und der Faktorisierungs-Regel aus M hergeleitet werden kann, so schreiben wir

$$M \vdash k.$$

Dies wird als M *leitet* k *her* gelesen.

Definition 60 (Allabschluß) Ist k eine prädikatenlogische Klausel und ist $\{x_1, \cdots, x_n\}$ die Menge aller Variablen, die in k auftreten, so definieren wir den *Allabschluß* $\forall(k)$ der Klausel k als

$$\forall(k) := \forall x_1, \cdots, x_n : k.$$

Die für uns wesentlichen Eigenschaften des Beweis-Begriffs $M \vdash k$ werden in den folgenden beiden Sätzen zusammengefaßt.

Satz 61 (Korrektheits-Satz)
Ist $M = \{k_1, \cdots, k_n\}$ eine Menge von Klauseln und gilt $M \vdash k$, so folgt

$$\models \forall(k_1) \wedge \cdots \wedge \forall(k_n) \rightarrow \forall(k).$$

Falls also eine Klausel k aus einer Menge M hergeleitet werden kann, so ist k tatsächlich eine Folgerung aus M. □

Die Umkehrung des obigen Korrektheits-Satzes gilt nur für die leere Klausel.

Satz 62 (Widerlegungs-Vollständigkeit)
Ist $M = \{k_1, \cdots, k_n\}$ eine Menge von Klauseln und gilt $\models \forall(k_1) \wedge \cdots \wedge \forall(k_n) \rightarrow \bot$, so folgt

$$M \vdash \{\}.$$ □

Damit haben wir nun ein Verfahren in der Hand, um für eine gegebene prädikatenlogischer Formel f die Frage, ob $\models f$ gilt, untersuchen zu können.

1. Wir berechnen zunächst die Skolem-Normalform von $\neg f$ und erhalten dabei so etwas wie

$$\neg f \approx_e \forall x_1, \cdots, x_m : g.$$

2. Anschließend bringen wir die Matrix g in konjunktive Normalform:

$$g \leftrightarrow k_1 \wedge \cdots \wedge k_n.$$

   Daher haben wir nun

$$\neg f \approx_e k_1 \wedge \cdots \wedge k_n$$

   und es gilt:

$$\models f \quad \text{g.d.w.} \quad \{\neg f\} \models \bot \quad \text{g.d.w.} \quad \{k_1, \cdots, k_n\} \models \bot.$$

3. Nach dem Korrektheits-Satz und dem Satz über die Widerlegungs-Vollständigkeit gilt

   $$\{k_1, \cdots, k_n\} \models \bot \quad \text{g.d.w.} \quad \{k_1, \cdots, k_n\} \vdash \bot.$$

   Wir versuchen also, nun die Widersprüchlichkeit der Menge $M = \{k_1, \cdots, k_n\}$ zu zeigen, indem wir aus M die leere Klausel ableiten. Wenn diese gelingt, haben wir damit die Allgemeingültigkeit der ursprünglich gegebenen Formel f gezeigt.

Zum Abschluß demonstrieren wir das skizzierte Verfahren an einem Beispiel. Wir gehen von folgenden Axiomen aus:

1. Jeder Drache ist glücklich, wenn alle seine Kinder fliegen können.
2. Rote Drachen können fliegen.
3. Die Kinder eines roten Drachens sind immer rot.

Wie werden zeigen, dass aus diesen Axiomen folgt, dass alle roten Drachen glücklich sind. Als erstes formalisieren wir die Axiome und die Behauptung in der Prädikatenlogik. Wir wählen die folgende Signatur

$$\Sigma_{\mathit{Drache}} := \langle \mathcal{V}, \mathcal{F}, \mathcal{P}, \mathit{arity} \rangle \quad \text{mit}$$

1. $\mathcal{V} := \{x, y, z\}$.
2. $\mathcal{F} = \{\}$.
3. $\mathcal{P} := \{\mathit{rot}, \mathit{fliegt}, \mathit{glücklich}, \mathit{kind}\}$.
4. $\mathit{arity} := \big\{\langle \mathit{rot}, 1\rangle, \langle \mathit{fliegt}, 1\rangle, \langle \mathit{glücklich}, 1\rangle, \langle \mathit{kind}, 2\rangle\big\}$

Das Prädikat $\mathit{kind}(x, y)$ soll genau dann wahr sein, wenn x ein Kind von y ist. Formalisieren wir die Axiome und die Behauptung, so erhalten wir die folgenden Formeln $f_1, \cdots, f_4$:

1. $f_1 := \forall x : \Big(\forall y : \big(\mathit{kind}(y, x) \rightarrow \mathit{fliegt}(y)\big) \rightarrow \mathit{glücklich}(x)\Big)$
2. $f_2 := \forall x : \big(\mathit{rot}(x) \rightarrow \mathit{fliegt}(x)\big)$
3. $f_3 := \forall x : \big(\mathit{rot}(x) \rightarrow \forall y : \mathit{kind}(y, x) \rightarrow \mathit{rot}(y)\big)$
4. $f_4 := \forall x : \big(\mathit{rot}(x) \rightarrow \mathit{glücklich}(x)\big)$

Wir wollen zeigen, dass die Formel

$$f := f_1 \wedge f_2 \wedge f_3 \rightarrow f_4$$

allgemeingültig ist. Wir betrachten also die Formel $\neg f$ und stellen fest

$$\neg f \leftrightarrow f_1 \wedge f_2 \wedge f_3 \wedge \neg f_4.$$

Als nächstes müssen wir diese Formel in eine Menge von Klauseln umformen. Da es sich hier um eine Konjunktion mehrerer Formeln handelt, können wir die einzelnen Formeln f_1, f_2, f_3 und $\neg f_4$ getrennt in Klauseln umwandeln.

1. Die Formel f_1 kann wie folgt umgeformt werden:

$$\begin{aligned}
f_1 &= \forall x : \Big(\forall y : \big(\mathit{kind}(y,x) \rightarrow \mathit{fliegt}(y)\big) \rightarrow \mathit{glücklich}(x)\Big) \\
&\leftrightarrow \forall x : \Big(\neg\forall y : \big(\mathit{kind}(y,x) \rightarrow \mathit{fliegt}(y)\big) \vee \mathit{glücklich}(x)\Big) \\
&\leftrightarrow \forall x : \Big(\neg\forall y : \big(\neg\mathit{kind}(y,x) \vee \mathit{fliegt}(y)\big) \vee \mathit{glücklich}(x)\Big) \\
&\leftrightarrow \forall x : \Big(\exists y : \neg\big(\neg\mathit{kind}(y,x) \vee \mathit{fliegt}(y)\big) \vee \mathit{glücklich}(x)\Big) \\
&\leftrightarrow \forall x : \Big(\exists y : \big(\mathit{kind}(y,x) \wedge \neg\mathit{fliegt}(y)\big) \vee \mathit{glücklich}(x)\Big) \\
&\leftrightarrow \forall x : \exists y : \Big(\big(\mathit{kind}(y,x) \wedge \neg\mathit{fliegt}(y)\big) \vee \mathit{glücklich}(x)\Big) \\
&\approx_e \forall x : \Big(\big(\mathit{kind}(s(x),x) \wedge \neg\mathit{fliegt}(s(x))\big) \vee \mathit{glücklich}(x)\Big)
\end{aligned}$$

   Im letzten Schritt haben wir dabei die Skolem-Funktion s mit $\mathit{arity}(s) = 1$ eingeführt. Anschaulich berechnet diese Funktion für jeden Drachen x, der nicht glücklich ist, ein Kind y, das nicht fliegen kann. Wenn wir in der Matrix dieser Formel das "$\vee$" noch Ausmultiplizieren, so erhalten wir die beiden Klauseln

$$k_1 := \big\{\mathit{kind}(s(x),x), \mathit{glücklich}(x)\big\},$$
$$k_2 := \big\{\neg\mathit{fliegt}(s(x)), \mathit{glücklich}(x)\big\}.$$

2. Analog finden wir für f_2:

$$\begin{aligned}
f_2 &= \forall x : \big(\mathit{rot}(x) \rightarrow \mathit{fliegt}(x)\big) \\
&\leftrightarrow \forall x : \big(\neg\mathit{rot}(x) \vee \mathit{fliegt}(x)\big)
\end{aligned}$$

   Damit ist f_2 zu folgender Klauseln äquivalent:

$$k_3 := \big\{\neg\mathit{rot}(x), \mathit{fliegt}(x)\big\}.$$

3. Für f_3 sehen wir:

$$\begin{aligned}
f_3 &= \forall x : \Big(\mathit{rot}(x) \rightarrow \forall y : \big(\mathit{kind}(y,x) \rightarrow \mathit{rot}(y)\big)\Big) \\
&\leftrightarrow \forall x : \Big(\neg\mathit{rot}(x) \vee \forall y : \big(\neg\mathit{kind}(y,x) \vee \mathit{rot}(y)\big)\Big) \\
&\leftrightarrow \forall x : \forall y : \big(\neg\mathit{rot}(x) \vee \neg\mathit{kind}(y,x) \vee \mathit{rot}(y)\big)
\end{aligned}$$

   Das liefert die folgende Klausel:

$$k_4 := \big\{\neg\mathit{rot}(x), \neg\mathit{kind}(y,x), \mathit{rot}(y)\big\}.$$

4. Umformung der Negation von f_4 liefert:

$$\begin{aligned}
\neg f_4 &= \neg\forall x : \big(\mathit{rot}(x) \rightarrow \mathit{glücklich}(x)\big) \\
&\leftrightarrow \neg\forall x : \big(\neg\mathit{rot}(x) \vee \mathit{glücklich}(x)\big) \\
&\leftrightarrow \exists x : \neg\big(\neg\mathit{rot}(x) \vee \mathit{glücklich}(x)\big) \\
&\leftrightarrow \exists x : \big(\mathit{rot}(x) \wedge \neg\mathit{glücklich}(x)\big) \\
&\approx_e \mathit{rot}(d) \wedge \neg\mathit{glücklich}(d)
\end{aligned}$$

   Die hier eingeführte Skolem-Konstante d steht für einen unglücklichen roten Drachen. Das führt zu den Klauseln

$$k_5 = \big\{\mathit{rot}(d)\big\},$$
$$k_6 = \big\{\neg\mathit{glücklich}(d)\big\}.$$

Wir müssen also untersuchen, ob die Menge M, die aus den folgenden Klauseln besteht, widersprüchlich ist:

1. $k_1 = \big\{ \mathit{kind}(s(x), x),\ \mathit{glücklich}(x) \big\}$
2. $k_2 = \big\{ \neg \mathit{fliegt}(s(x)),\ \mathit{glücklich}(x) \big\}$
3. $k_3 = \big\{ \neg \mathit{rot}(x),\ \mathit{fliegt}(x) \big\}$
4. $k_4 = \big\{ \neg \mathit{rot}(x),\ \neg \mathit{kind}(y, x),\ \mathit{rot}(y) \big\}$
5. $k_5 = \big\{ \mathit{rot}(d) \big\}$
6. $k_6 = \big\{ \neg \mathit{glücklich}(d) \big\}$

Sei also $M := \big\{ k_1, k_2, k_3, k_4, k_5, k_6 \big\}$. Wir zeigen, dass $M \vdash \bot$ gilt:

1. Es gilt
   $$\mathit{mgu}\big(\mathit{rot}(d), \mathit{rot}(x)\big) = [x \mapsto d].$$
   Daher können wir die Resolutions-Regel auf die Klauseln k_5 und k_4 wie folgt anwenden:
   $$\big\{ \mathit{rot}(d) \big\},\ \big\{ \neg \mathit{rot}(x), \neg \mathit{kind}(y, x), \mathit{rot}(y) \big\} \ \vdash\ \big\{ \neg \mathit{kind}(y, d), \mathit{rot}(y) \big\}.$$
2. Wir wenden nun auf die resultierende Klausel und auf die Klausel k_1 die Resolutions-Regel an. Dazu berechnen wir zunächst
   $$\mathit{mgu}\big(\mathit{kind}(y, d), \mathit{kind}(s(x), x)\big) = [y \mapsto s(d), x \mapsto d].$$
   Dann haben wir
   $$\big\{ \neg \mathit{kind}(y, d), \mathit{rot}(y) \big\},\ \big\{ \mathit{kind}(s(x), x), \mathit{glücklich}(x) \big\} \ \vdash\ \big\{ \mathit{glücklich}(d), \mathit{rot}(s(d)) \big\}.$$
3. Jetzt wenden wir auf die eben abgeleitete Klausel und die Klausel k_6 die Resolutions-Regel an. Wir haben:
   $$\mathit{mgu}\big(\mathit{glücklich}(d), \mathit{glücklich}(d)\big) = []$$
   Also erhalten wir
   $$\big\{ \mathit{glücklich}(d), \mathit{rot}(s(d)) \big\},\ \big\{ \neg \mathit{glücklich}(d) \big\} \ \vdash\ \big\{ \mathit{rot}(s(d)) \big\}.$$
4. Auf die Klausel $\big\{ \mathit{rot}(s(d)) \big\}$ und die Klausel k_3 wenden wir die Resolutions-Regel an. Zunächst haben wir
   $$\mathit{mgu}\big(\mathit{rot}(s(d)), \neg \mathit{rot}(x)\big) = [x \mapsto s(d)]$$
   Also liefert die Anwendung der Resolutions-Regel:
   $$\big\{ \mathit{rot}(s(d)) \big\},\ \big\{ \neg \mathit{rot}(x), \mathit{fliegt}(x) \big\} \ \vdash\ \big\{ \mathit{fliegt}(s(d)) \big\}$$
5. Um die so erhaltenen Klausel $\big\{ \mathit{fliegt}(s(d)) \big\}$ mit der Klausel k_3 resolvieren zu können, berechnen wir
   $$\mathit{mgu}\big(\mathit{fliegt}(s(d)), \mathit{fliegt}(s(x))\big) = [x \mapsto d]$$
   Dann liefert die Resolutions-Regel
   $$\big\{ \mathit{fliegt}(s(d)) \big\},\ \big\{ \neg \mathit{fliegt}(s(x)), \mathit{glücklich}(x) \big\} \ \vdash\ \big\{ \mathit{glücklich}(d) \big\}.$$
6. Auf das Ergebnis $\big\{ \mathit{glücklich}(d) \big\}$ und die Klausel k_6 können wir nun die Resolutions-Regel anwenden:
   $$\big\{ \mathit{glücklich}(d) \big\},\ \big\{ \neg \mathit{glücklich}(d) \big\} \ \vdash\ \big\{ \big\}.$$

Da wir im letzten Schritt die leere Klausel erhalten haben, ist insgesamt $M \vdash \bot$ nachgewiesen worden und damit haben wir gezeigt, dass alle kommunistischen Drachen glücklich sind.

Aufgabe: Die von Bertrant Russell definierte *Russell-Menge* R ist definiert als die Menge aller der Mengen, die sich nicht selbst enthalten. Damit gilt also

$$\forall x : \big(x \in R \leftrightarrow \neg x \in R \big).$$

Zeigen Sie mit Hilfe des in diesem Abschnitt definierten Kalküls, dass diese Formel widersprüchlich ist.

Hausaufgabe: Gegeben seien folgende Axiome:

1. Jeder Barbier rasiert alle Personen, die sich nicht selbst rasieren.
2. Kein Barbier rasiert jemanden, der sich selbst rasiert.

Zeigen Sie, dass aus diesen Axiomen logisch die folgende Aussage folgt:

> Alle Barbiere sind blond.

5.6 *Prover9* und *Mace4*

Der im letzten Abschnitt beschriebene Kalkül läßt sich automatisieren und bildet die Grundlage moderner automatischer Beweiser. Gleichzeitig läßt sich auch die Suche nach Gegenbeispielen automatisieren. Wir stellen in diesem Abschnitt zwei Systeme vor, die diesen Zwecken dienen.

1. *Prover9* dient dazu, automatisch prädikatenlogische Formeln zu beweisen.
2. *Mace4* untersucht, ob eine gegebene Menge prädikatenlogischer Formeln in einer endlichen Struktur erfüllbar ist. Gegebenenfalls wird diese Struktur berechnet.

Die beiden Programme *Prover9* und *Mace4* wurden von William McCune [McC10] entwickelt, stehen unter der GPL und können unter der Adresse

> `http://www.cs.unm.edu/~mccune/prover9/download`

als Quelltext heruntergeladen werden. Wir diskutieren zunächst *Prover9* und schauen uns anschließend *Mace4* an.

5.6.1 Der automatische Beweiser *Prover9*

Prover9 ist ein Programm, das als Eingabe zwei Mengen von Formeln bekommt. Die erste Menge von Formeln wird als Menge von *Axiomen* interpretiert, die zweite Menge von Formeln sind die zu beweisenden *Theoreme*, die aus den Axiomen gefolgert werden sollen. Wollen wir beispielsweise zeigen, dass aus den drei Formeln der Gruppentheorie

1. $\forall x : e \cdot x = x$,
2. $\forall x : \exists y : x \cdot y = e$ und
3. $\forall x : \forall y : \forall z : (x \cdot y) \cdot z = x \cdot (y \cdot z)$

die beiden Formeln

1. $\forall x : x \cdot e = x$,
2. $\forall x : \exists y : y \cdot x = e$ und

logisch folgen, so stellen wir diese Formeln wie in Abbildung 5.6 auf Seite 138 gezeigt als Text dar. Der Anfang der Axiome wird in dieser Datei durch "`formulas(sos)`" eingeleitet und durch das Schlüsselwort "`end_of_list`" beendet. Zu beachten ist, dass sowohl die Schlüsselwörter als auch die einzelnen Formel jeweils durch einen Punkt "`.`" beendet werden. Die Axiome in den Zeilen 2, 3, und 4 drücken aus, dass

1. `e` ein links-neutrales Element ist,

2. zu jedem Element x ein links-inverses Element y existiert und

3. das Assoziativ-Gesetz gilt.

Aus diesen Axiomen folgt, dass das `e` auch ein rechts-neutrales Element ist und dass außerdem zu jedem Element x ein rechts-neutrales Element y existiert. Diese beiden Formeln sind die zu beweisenden *Ziele* und werden in der Datei durch "`formulas`(goal)" markiert. Trägt die in Abbildung 5.6 gezeigte Datei den Namen "`group2.in`", so können wir das Programm *Prover9* mit dem Befehl

```
prover9 -f group2.in
```

starten und erhalten als Ergebnis die Information, dass die beiden in Zeile 8 und 9 gezeigten Formeln tatsächlich aus den vorher angegebenen Axiomen folgen. Hätte *Prover9* keinen Beweis gefunden, so wäre das Programm solange weitergelaufen, bis kein freier Speicher mehr zur Verfügung gestanden hätte und wäre dann mit einer Fehlermeldung abgebrochen.

```
1  formulas(sos).
2  all x (e * x = x).                              % left neutral
3  all x exists y (y * x = e).                     % left inverse
4  all x all y all z ((x * y) * z = x * (y * z)).  % associativity
5  end_of_list.
6
7  formulas(goals).
8  all x (x * e = x).                              % right neutral
9  all x exists y (x * y = e).                     % right inverse
10 end_of_list.
```

Abbildung 5.6: Textuelle Darstellung der Axiome der Gruppentheorie.

Prover9 versucht, einen indirekten Beweis zu führen. Zunächst werden die Axiome in prädikatenlogische Klauseln überführt. Dann wird jedes zu beweisenden Theorem negiert und die negierte Formel wird ebenfalls in Klauseln überführt. Anschließend versucht *Prover9* aus der Menge aller Axiome zusammen mit den Klauseln, die sich aus der Negation eines der zu beweisenden Theoreme ergeben, die leere Klausel herzuleiten. Gelingt dies, so ist bewiesen, dass das jeweilige Theorem tatsächlich aus den Axiomen folgt. Abbildung 5.7 zeigt eine Eingabe-Datei für *Prover9*, bei der versucht wird, das Kommutativ-Gesetz aus den Axiomen der Gruppentheorie zu folgern. Der Beweis-Versuch mit *Prover9* schlägt allerdings fehl. In diesem Fall wird die Beweissuche nicht endlos fortgesetzt. Dies liegt daran, dass es *Prover9* gelingt, in endlicher Zeit alle aus den gegebenen Voraussetzungen folgenden Formeln abzuleiten. Ein solcher Fall ist allerdings eher die Ausnahme als die Regel.

```
1  formulas(sos).
2  all x (e * x = x).                              % left neutral
3  all x exists y (y * x = e).                     % left inverse
4  all x all y all z ((x * y) * z = x * (y * z)).  % associativity
5  end_of_list.
6
7  formulas(goals).
8  all x all y (x * y = y * x).                    % * is commutative
9  end_of_list.
```

Abbildung 5.7: Gilt das Kommutativ-Gesetz in allen Gruppen?

5.6.2 *Mace4*

Dauert ein Beweisversuch mit *Prover9* endlos, so ist zunächst nicht klar, ob das zu beweisende Theorem gilt. Um sicher zu sein, dass eine Formel nicht aus einer gegebenen Menge von Axiomen folgt, reicht es aus, eine Struktur zu konstruieren, in der alle Axiome erfüllt sind, in der das zu beweisende Theorem aber falsch ist. Das Programm *Mace4* dient genau dazu, solche Strukturen zu finden. Das funktioniert natürlich nur, solange die Strukturen endlich sind. Abbildung 5.8 zeigt eine Eingabe-Datei, mit deren Hilfe wir die Frage, ob es endliche nicht-kommutative Gruppen gibt, unter Verwendung von *Mace4* beantworten können. In den Zeilen 2, 3 und 4 stehen die Axiome der Gruppen-Theorie. Die Formel in Zeile 5 postuliert, dass für die beiden Elemente a und b das Kommutativ-Gesetz nicht gilt, dass also $a \cdot b \neq b \cdot a$ ist. Ist der in Abbildung 5.8 gezeigte Text in einer Datei mit dem Namen "*group.in*" gespeichert, so können wir *Mace4* durch das Kommando

mace4 -f group.in

starten. *Mace4* sucht für alle positiven natürlichen Zahlen $n = 1, 2, 3, \cdots$, ob es eine Struktur $\mathcal{S} = \langle \mathcal{U}, \mathcal{J} \rangle$ mit $card(U) = n$ gibt, in der die angegebenen Formeln gelten. Bei $n = 6$ wird *Mace4* fündig und berechnet tatsächlich eine Gruppe mit 6 Elementen, in der das Kommutativ-Gesetz verletzt ist.

```
1  formulas(theory).
2  all x (e * x = x).                              % left neutral
3  all x exists y (y * x = e).                     % left inverse
4  all x all y all z ((x * y) * z = x * (y * z)).  % associativity
5  a * b != b * a.                                 % a and b do not commute
6  end_of_list.
```

Abbildung 5.8: Gibt es eine Gruppe, in der das Kommutativ-Gesetz nicht gilt?

Abbildung 5.9 zeigt einen Teil der von *Mace4* produzierten Ausgabe. Die Elemente der Gruppe sind die Zahlen $0, \cdots, 5$, die Konstante a ist das Element 0, b ist das Element 1, e ist das Element 2. Weiter sehen wir, dass das Inverse von 0 wieder 0 ist, das Inverse von 1 ist 1 das Inverse von 2 ist 2, das Inverse von 3 ist 4, das Inverse von 4 ist 3 und das Inverse von 5 ist 5. Die Multiplikation wird durch die folgende Gruppen-Tafel realisiert:

$\circ$	0	1	2	3	4	5
0	2	3	0	1	5	4
1	4	2	1	5	0	3
2	0	1	2	3	4	5
3	5	0	3	4	2	1
4	1	5	4	2	3	0
5	3	4	5	0	1	2

Diese Gruppen-Tafel zeigt, dass

$$a \circ b = 0 \circ 1 = 3, \quad \text{aber} \quad b \circ a = 1 \circ 0 = 4$$

gilt, mithin ist das Kommutativ-Gesetz tatsächlich verletzt.

Bemerkung: Der Theorem-Beweiser *Prover9* ist ein Nachfolger des Theorem-Beweisers *Otter*. Mit Hilfe von *Otter* ist es McCune 1996 gelungen, die Robbin'sche Vermutung zu beweisen [McC97]. Dieser Beweis war damals sogar der *New York Times* eine Schlagzeile wert, nachzulesen unter

`http://www.nytimes.com/library/cyber/week/1210math.html`.

Das Ergebnis zeigt, dass automatische Theorem-Beweiser durchaus nützliche Werkzeuge sein können. Nichtdestoweniger ist die Prädikatenlogik unentscheidbar und bisher sind auch nur wenige offene mathematische Probleme mit Hilfe von automatischen Beweisern gelöst worden. Das wird sich vermutlich auch in der näheren

```
1  ============================== DOMAIN SIZE 6 ==========================
2
3  === Mace4 starting on domain size 6. ===
4
5  ============================== MODEL ==================================
6
7  interpretation( 6, [number=1, seconds=0], [
8
9          function(a, [ 0 ]),
10
11         function(b, [ 1 ]),
12
13         function(e, [ 2 ]),
14
15         function(f1(_), [ 0, 1, 2, 4, 3, 5 ]),
16
17         function(*(_,_), [
18                             2, 3, 0, 1, 5, 4,
19                             4, 2, 1, 5, 0, 3,
20                             0, 1, 2, 3, 4, 5,
21                             5, 0, 3, 4, 2, 1,
22                             1, 5, 4, 2, 3, 0,
23                             3, 4, 5, 0, 1, 2 ])
24 ]).
25
26 ============================== end of model ===========================
```

Abbildung 5.9: Ausgabe von *Mace4*.

Zukunft nicht ändern.