Logik und Datenbanken

Sommersemester 2004

Peter Baumgartner
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany

Email baumgart@mpi-sb.mpg.de

Inhalt dieser Vorlesung

Die folgenden Blöcke:

Konzepte der Prädikatenlogik und der Logikprogrammierung

Logik als Datenbanksprache

Verteilte Datenbanken und Informationsintegration

Hierarchische Datenmodelle und Beschreibungslogiken

Siehe Webseite `http://www.mpi-sb.mpg.de/~baumgart/logik-datenbanken/` für Unterlagen zur Vorlesung (Literatur, Folien).

Logik und Logikprogrammierung

1 Einführung

1.1 Worum es hier geht

Im Lexikon steht:

- „Logik ist die Lehre vom folgerichtigen Schließen“
- „Logik ist die Lehre von formalen Beziehungen zwischen Denkinhalten“

Ein Beispiel zum „folgerichtigen Schließen“:

A_1: Falls es heiß und schwül ist, wird es regnen
A_2: Falls es schwül ist, so ist es heiß
A_3: Es ist schwül

Wird es regnen?

Übersetzung in logische Formeln der Aussagenlogik

- Aussagen wie „heiß“, „schwül“ und „regnen“ werden durch Symbole wie P, Q und R bezeichnet.
- Verbindungsglieder wie „und“, „oder“ und „falls ... so“ werden mit entsprechenden logischen Symbole wie $\wedge$, $\vee$ und $\rightarrow$ bezeichnet.

Damit:

A_1: $P \wedge Q \rightarrow R$
A_2: $Q \rightarrow P$
A_3: Q

„Gilt R?“

Genauer: Ist es der Fall, daß, falls A_1, A_2 und A_3 wahr sind, so auch R wahr ist?

Sprechweisen (alle äquivalent):

- Falls A_1, A_2 und A_3 wahr sind, so ist auch R wahr
- R *folgt aus* A_1, A_2 und A_3
- R ist *logische Konsequenz* von A_1, A_2 und A_3
- $\{A_1, A_2, A_3\} \models R$

Bemerkungen:

- Falls $\{A_1, A_2, A_3\} \models R$, so können für A_1, A_2, A_3 und R *beliebige* Aussagen eingesetzt werden, und, falls die Einsetzungen für A_1, A_2 und A_3 wahr sind, so *muß* auch R wahr sein.

(Vgl. Logik als „Lehre von formalen Beziehungen zwischen Denkinhalten“)

- Was aber *bedeutet* $\{A_1,\ A_2,\ A_3\} \models R$ genau („Semantik“)?
- Und: Wie zeigt man „mechanisch“ daß $\{A_1,\ A_2,\ A_3\} \models R$?

  „Theorembeweisen“ als Teildisziplin der KI zu *effzizienten* Mechanisierung (u.a.) der Folgerungsbeziehung

Prädikatenlogik

Aussagenlogik ist oft nicht ausdrucksstark genug. Beispiel:

A_1: Sokrates ist ein Mensch
A_2: Alle Menschen sind sterblich

Übersetzung in logische Formeln der Prädikatenlogik:

A_1: $\text{mensch}(\text{sokrates})$
A_2: $\forall X\ (\text{mensch}(X) \rightarrow \text{sterblich}(X))$

Welche der folgenden Äusserungen trifft zu?

1. $\{A_1,\ A_2\} \models \text{sterblich}(\text{sokrates})$
2. $\{A_1,\ A_2\} \models \text{sterblich}(\text{apollo})$
3. $\{A_1,\ A_2\} \not\models \text{sterblich}(\text{sokrates})$
4. $\{A_1,\ A_2\} \not\models \text{sterblich}(\text{apollo})$
5. $\{A_1,\ A_2\} \models \neg\text{sterblich}(\text{sokrates})$
6. $\{A_1,\ A_2\} \models \neg\text{sterblich}(\text{apollo})$

Wobei $\neg A$ wahr ist gdw. A falsch ist.

Bemerkung: Die Mechanisierung der Folgerungsbeziehung für die Prädikatenlogik ist *wesentlich* schwieriger als für die Aussagenlogik:

Aussagenlogik – entscheidbar
Prädikatenlogik – semi-entscheidbar

(Maximal-)Programm dieses Kapitels:

- Ein paar Bemerkungen zur Historie, Informatik-Anwendungen
- Aussagenlogik: Syntax, Semantik, Mechanisierung (Resolutionsverfahren)
- Prädikatenlogik: Syntax, Semantik, Mechanisierung (Resolutionsverfahren)
- Ein spezielles Resolutionsverfahren – SLD-Resolution – zur Logikprogrammierung.

1.2 Historie

Sehr grob:

- Aristoteles: „Syllogismen“.
- Peano/Boole/Frege, Ende 19. Jahrhundert: Formale Notation (Aussagenlogik, Prädikatenlogik).

  „Mathematische Logik“: eine Mathematische Theorie (wie Analysis, Algebra), welche die vorhandenen Strukturen der Mathematik untersuchen will, z.B. Mengen (Stichwort: „Paradoxe Mengen“).
- Gödel 1930: Vollständiger Kalkül für die Prädikatenlogik.
- Anfang 19. Jahrhundert: Whitehead/Russel: „Principia Mathematica“ - Versuch, Mathematik vollständig zu formalisieren und beweisen.
- Gödel 1931: „Über formal unentscheidbare Sätze der *Principia Mathematica* und verwandter Systeme“.
- Herbrand 1930, Davis/Putnam/Logeman/Loveland 1962: Mechanische Verfahren zum Theorembeweisen für Prädikatenlogik („British Museum Procedures“).
- Robinson 1965: „A Machine Oriented Logic Based on the Resolution Principle“.

1.3 Informatik-Anwendungen

Programmentwicklung

Ziel: Formale, auf Logik basierte Methoden sollen helfen, korrekte Software zu erstellen.

Einige Beispiele (Abstrakte Datentypen, Programmverifikation, Verifikation reaktiver Systeme).

Abstrakte Datentypen (ADTs) dienen der Spezifikation von Datenstrukturen und Operationen auf diesen Datenstrukturen. Wichtiger Aspekt: die tatsächliche Realisierung durch konkrete Datenstrukturen wird üblicherweise dadurch *nicht* festgelegt.

Beispiel Stack:

Sei Σ ein Alphabet, und clear eine nullstellige Funktion.

Definition der Menge S der Stacks:

$$
\begin{array}{ll}
 & \mathrm{clear} \in S \\
\forall s \in S & \forall x \in \Sigma \quad \mathrm{push}(s,x) \in S \\
\forall s \in S & (s \neq \mathrm{clear} \rightarrow \mathrm{pop}(s) \in S) \\
\forall s \in S & (s \neq \mathrm{clear} \rightarrow \mathrm{top}(s) \in \Sigma) \\
\forall s \in S & \mathrm{empty}(s) \in \{\mathrm{true}, \mathrm{false}\}
\end{array}
$$

Die Menge S ist zu groß. Deshalb werden weitere Eigenschaften gefordert:

$$
\begin{array}{lll}
\forall s \in S & \forall x \in \Sigma & \mathrm{push}(s,x) \neq \mathrm{clear} \\
\forall s \in S & \forall x,y \in \Sigma & (\mathrm{push}(s,x) = \mathrm{push}(s,y) \rightarrow x = y) \\
\forall s,t \in S & \forall x \in \Sigma & (\mathrm{push}(s,x) = \mathrm{push}(t,x) \rightarrow s = t)
\end{array}
$$

... und weitere Eigenschaften für die Kombination von Funktionssymbolen:

$$\begin{array}{lll} \forall s \in S & \forall x \in \Sigma & (\mathrm{top}(\mathrm{push}(s,x) = x \wedge \mathrm{pop}(\mathrm{push}(s,x)) = s) \\ \forall s \in S & \forall x \in \Sigma & \mathrm{empty}(\mathrm{push}(s,x)) = \mathrm{false} \\ & & \mathrm{empty}(\mathrm{clear}) = \mathrm{true} \end{array}$$

Wichtige Fragen an ADTs, als Basis für weitere Programmentwicklung:

- Ist der ADT *konsistent?*
- Ist der ADT bezüglich weiterer geforderter Eigenschaften *vollständig*?

Zur *Programmverifikation* können geforderte Eigenschaften des Programms als Vor- und Nachbedingungen angegeben werden, z.B.:

```
(1) { t ≥ 0 }
(2)       t := t+1;
(3)   a[t] := x;
(4) { t > 0 ∧ a[t] = x }
```

Mit geeigneten Kalkülen kann die (partielle/totale) Korrektheit nachgewiesen werden (auch in Verbindung mit ADTs).

Bei der *Verifikation reaktiver Systeme* geht es um den Nachweis von temporalen Eigenschaften von nicht-terminierenden Programmen. Logik wird als Spezifikationssprache verwendet.

Künstliche Intelligenz

Zwei Beispiele (Automatisches Theorembeweisen, Wissensrepräsentation).

Automatisches Theorembeweisen beschäftigt sich (u.a.) mit Methoden zur Automatisierung der Folgerungsrelation.

Paradebeispiel: Der automatische Beweis der über 50 Jahre offenen „Robbin's Conjecture" durch den EQP-Theorembeweiser.

Gegeben die folgende Basis der Booleschen Algebra (Huntington, 1933):

$$\begin{array}{rr} x+y = y+x & \text{(Kommutativität)} \\ (x+y)+z = x+(y+z) & \text{(Assoziativität)} \\ n(n(x)+y)+n(n(x)+n(y)) = x & \text{(Huntington Gleichung)} \end{array}$$

Robbin's Conjecture: Die Huntington Gleichung kann ersetzt werden durch

$$n(n(x+y)+n(x+n(y))) = x \qquad \text{(Robin's Gleichung)}$$

In der *Wissensrepräsentation* spielt Logik eine wichtige Rolle als Formalismus zur Beschreibung von „Weltwissen", und zur Spezifikation anderer, graphischer Formalismen. Beispiel:

$$\forall x \quad (\mathrm{man}(x) \rightarrow \mathrm{mammal}(x))$$

$\forall x \quad (\text{animal}(x) \rightarrow \text{mammal}(x))$
$\forall x \quad (\text{man}(x) \rightarrow \exists y\ \text{name}(x,y))$
$\forall x \quad (\text{mammal}(x) \rightarrow \exists y\ \text{age}(x,y))$

2 Aussagenlogik

(sentential logic, propositional logic)

Aussagenlogik (AL) befaßt sich mit Feststellungen über Wahrheitswerte von Aussagen alleine aufgrund ihrer *Form*.

Inhalt: Syntax, Semantik, ein Kalkül für AL, wichtige Ergebnisse

2.1 Syntax der Aussagenlogik

Definition 2.1 (Syntax der Aussagenlogik)
Gegeben seien

- eine aufzählbare Menge von *atomaren Formeln* P_i, für $i = 1,2,3\ldots$, und
- die *Junktoren* $\wedge$, $\vee$ und $\neg$, und
- die Symbole (und).

Die Menge AF der *aussagenlogischen Formeln* ist wie folgt induktiv definiert:

1. $P_i \in AF$, für $i = 1,2,3\ldots$.
2. Falls $F \in AF$ und $G \in AF$, so auch $(F \wedge G) \in AF$ und $(F \vee G) \in AF$.
3. Falls $F \in AF$, so auch $\neg F \in AF$.

□

Sprechweise in Abschnitt 3: „Formel“ statt „aussagenlogische Formel“

Definition 2.2 (Teilformel)
Eine *Teilformel* einer Formel F ist eine Teilzeichenreihe von F welche selbst wieder eine Formel ist. □

Konvention 2.3 (Abkürzungen, Schreiberleichterungen) Die folgenden Abkürzungen werden benutzt (wobei $F_i \in AF$):

Abkürzung	*Expansion*
$A, B, C, \ldots$	$P_1, P_2, P_3, \ldots$
$(F_1 \rightarrow F_2)$	$(\neg F_1 \vee F_2)$
$(F_2 \leftarrow F_1)$	$(\neg F_1 \vee F_2)$
$(F_1 \leftrightarrow F_2)$	$((F_1 \wedge F_2) \vee (\neg F_1 \wedge \neg F_2))$
$\bigvee_{i=1}^{n} F_i$	$(\cdots((F_1 \vee F_2) \vee F_3) \vee \cdots \vee F_n)$
$\bigwedge_{i=1}^{n} F_i$	$(\cdots((F_1 \wedge F_2) \wedge F_3) \wedge \cdots \wedge F_n)$

Die Symbole $\rightarrow, \leftarrow$ und $\leftrightarrow$ werden ebenfalls Junktoren genannt.

Präzedenzen der Junktoren werden vereinbart (in ansteigender Bindungskraft):

$$\leftrightarrow \quad \begin{matrix}\rightarrow \\ \leftarrow\end{matrix} \quad \wedge \quad \vee \quad \neg$$

Klammern in AFs dürfen weggelassen werden, falls rekonstruierbar. Um dies eindeutig zu ermöglichen, werden alle zweistelligen Junktoren zusätzlich als links-assoziativ vereinbart. □

2.2 Semantik der Aussagenlogik

Definition 2.4 (Semantik der Aussagenlogik)
Die Menge der *Wahrheitswerte* ist $\{\mathbf{W}, \mathbf{F}\}$.

Eine *Belegung* für eine Menge D von atomaren Formeln ist eine Funktion $\mathcal{A}_\mathsf{D}$ welche jedem $\mathsf{A} \in \mathsf{D}$ einen Wahrheitswert zuordnet, i.e. $\mathcal{A}_\mathsf{D}(\mathsf{A}) \in \{\mathbf{W}, \mathbf{F}\}$ für jedes $\mathsf{A} \in \mathsf{D}$.

Sei D und $\mathcal{A}_\mathsf{D}$ gegeben, und $\mathsf{E} \subseteq \mathsf{AF}$ so daß

1. E „abwärts geschloßen“ ist (i.e. falls $\mathsf{F} \in \mathsf{E}$ und F' Teilformel von F ist, so auch $\mathsf{F}' \in \mathsf{E}$), und
2. jede atomare Formel in E auch in D ist.

Die *Erweiterung von* $\mathcal{A}_\mathsf{D}$ *auf* E ist eine Funktion $\mathcal{A}_\mathsf{E}$ welche jedem $\mathsf{F} \in \mathsf{E}$ gemäß folgender rekursiver Definition einen Wahrheitswert zuordnet:

1. $\mathcal{A}_\mathsf{E}(\mathsf{F}) = \mathcal{A}_\mathsf{D}(\mathsf{F})$, falls F atomare Formel ist.
2. $\mathcal{A}_\mathsf{E}(\mathsf{F} \wedge \mathsf{G}) = \begin{cases} \mathbf{W} & \text{falls } \mathcal{A}_\mathsf{E}(\mathsf{F}) = \mathbf{W} \text{ und } \mathcal{A}_\mathsf{E}(\mathsf{G}) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$
3. $\mathcal{A}_\mathsf{E}(\mathsf{F} \vee \mathsf{G}) = \begin{cases} \mathbf{W} & \text{falls } \mathcal{A}_\mathsf{E}(\mathsf{F}) = \mathbf{W} \text{ oder } \mathcal{A}_\mathsf{E}(\mathsf{G}) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$
4. $\mathcal{A}_\mathsf{E}(\neg \mathsf{F}) = \begin{cases} \mathbf{W} & \text{falls } \mathcal{A}_\mathsf{E}(\mathsf{F}) = \mathbf{F} \\ \mathbf{F} & \text{sonst} \end{cases}$

Notation: Statt $\mathcal{A}_\mathsf{D}$ und $\mathcal{A}_\mathsf{E}$ kürzer $\mathcal{A}$.

Bezeichnung: Eine Formel der Art $(\mathsf{F} \wedge \mathsf{G})$ heißt *Konjunktion*, eine Formel der Art $(\mathsf{F} \vee \mathsf{G})$ heißt *Disjunktion*, eine Formel der Art $\neg \mathsf{F}$ heißt *Negation*. □

Alternative: Definition der Bedeutung der Junktoren über Wahrheitstafeln.

Bemerkung: Bei „intensionalen Logiken“ gilt das in Definition 2.4 realiserte Extensionalitätsprinzip nicht.

Induktive Definitionen (wie Definition 2.1) gestatten das Beweisprinzip der *strukturellen Induktion*. Im Fall der Aussagenlogik erhält man:

Bemerkung 2.5 (Induktion über den Formelaufbau) Um zu beweisen, daß eine Behauptung $\mathcal{B}$ für jede Formel F gilt, genügt es, das Folgende zu beweisen:

Induktionsanfang. $\mathcal{B}(A)$ gilt für jede atomare Formel A.

Induktionsschritt. Unter der (Induktions-) Annahme, daß $\mathcal{B}(F)$ und $\mathcal{B}(G)$ für beliebige Formeln F und G gelten, zeigt man, daß damit auch $\mathcal{B}(\neg F)$, $\mathcal{B}((F \wedge G))$ und $\mathcal{B}((F \vee G))$ gelten.

□

Definition 2.6 (Modell, gültig, erfüllbar)
Sei F eine Formel und $\mathcal{A}$ eine Belegung. Falls $\mathcal{A}$ für alle in F vorkommenden atomaren Formeln definiert ist, so heißt $\mathcal{A}$ zu F *passend*.

$\mathcal{A}$ ist *passend* zu einer Menge M von Formeln, gdw. $\mathcal{A}$ passend zu allen $F \in M$ ist.

Als äquivalent wird definiert:

- $\mathcal{A}$ ist passend zu F und $\mathcal{A}(F) = \mathbf{W}$.
- $\mathcal{A} \models F$.
- $\mathcal{A}$ ist ein Modell für F.
- F gilt unter der Belegung $\mathcal{A}$.

Als äquivalent wird definiert:

- $\mathcal{A}$ ist passend zu F und $\mathcal{A}(F) = \mathbf{F}$.
- $\mathcal{A} \not\models F$.
- $\mathcal{A}$ ist kein Modell für F.
- F gilt nicht unter der Belegung $\mathcal{A}$.

Eine Formel F heißt

- *erfüllbar*, falls F mindestens ein Modell besitzt, *unerfüllbar* sonst.
- *tautologisch* (oder *Tautologie*), falls jede zu F passende Belegung ein Modell für F ist. Notation: $\models F$ für „F ist Tautologie“. $\not\models F$ für „F ist keine Tautologie“.

Sei M eine Menge von Formeln. M heißt *erfüllbar* gdw.

Es existiert eine Belegung A, so daß für alle $F \in M$ gilt: $\mathcal{A} \models F$

□

Satz 2.7
Eine Formel F ist eine Tautologie gdw. $\neg F$ unerfüllbar ist.

Definition 2.8 (Folgerung, logische Konsequenz)
Sei M eine Menge von Formeln und G eine Formel.

G ist eine *Folgerung* von M (oder *(logische) Konsequenz* von M), i.Z. $M \models G$, gdw.

Für alle zu M passenden Belegungen $\mathcal{A}$ gilt: Falls $\mathcal{A} \models F$ für alle $F \in M$, so auch $\mathcal{A} \models G$.

Für eine Formel F definiere $F \models G$ als $\{F\} \models G$.

Dieselbe Sprechweisen, „Folgerung“ und „Konsequenz“ werden auch hier angewendet. □

Satz 2.9

1. *Die folgenden drei Behauptungen sind äquivalent:*
   a) G *ist eine Folgerung von* F.
   b) $(F \to G)$ *ist eine Tautolgie.*
   c) $(F \wedge \neg G)$ *ist unerfüllbar.*
2. *Die folgenden beiden Behauptungen sind äquivalent:*
   a) G *ist eine Folgerung von* M.
   b) $M \cup \{\neg G\}$ *ist unerfüllbar.*

Lemma 2.10
Seien $\mathcal{A}$ *und* $\mathcal{A}'$ *zwei zu einer Formel* F *passende Belegungen.*

Falls $\mathcal{A}(A) = \mathcal{A}'(A)$ *für alle atomaren Teilformeln* A *von* F, *dann auch* $\mathcal{A}(F) = \mathcal{A}'(F)$.

(Wie beweist man das?)

Relevanz: Der Wahrheitswert von F hängt nur von den in F vorkommenden atomaren Teilformeln ab.

Fazit: „Wahrheitstafeln genügen“.

Wahrheitstafel für eine Formel F mit den atomaren Teilformeln $A_1, \ldots, A_n$

	A_1	$\cdots$	A_{n-1}	A_n	F
$\mathcal{A}_1$:	**F**	$\cdots$	**F**	**F**	$\mathcal{A}_1(F)$
$\mathcal{A}_2$:	**F**	$\cdots$	**F**	**W**	$\mathcal{A}_2(F)$
$\vdots$					
$\mathcal{A}_{2^n}$:	**W**	$\cdots$	**W**	**W**	$\mathcal{A}_{2^n}(F)$

F ist erfüllbar (tautologisch), falls die Spalte unter F mindestens einmal (nur) den Eintrag **W** enthält.

Aufwand zur Erstellung von Wahrheitstafeln?

Beispiel 2.11 Eine Wahrheitstafel für $F = (A \wedge (A \vee B))$.

Es bieten sich an, für Teilformeln von F eigene Spalten anzulegen.

	A	B	$\mathcal{A}((A \wedge B))$	$\mathcal{A}((A \vee (A \wedge B)))$
$\mathcal{A}_1$:	**F**	**F**	**F**	**F**
$\mathcal{A}_2$:	**F**	**W**	**F**	**F**
$\mathcal{A}_3$:	**W**	**F**	**F**	**W**
$\mathcal{A}_4$:	**W**	**W**	**W**	**W**

□

2.3 Äquivalenzen und Normalformen

Normalform: Kanonische Repräsentation von Formel(menge)n. Dazu:

Definition 2.12 ((Semantische) Äquivalenz)
Zwei Formeln F und G heißen *(semantisch) äquivalent*, i.Z. $F \equiv G$, gdw.

Für alle sowohl zu F als auch zu G passenden Belegungen $\mathcal{A}$ gilt: $\mathcal{A}(F) = \mathcal{A}(G)$.

□

Bemerkung: Auch Formeln mit verschiedenen atomaren Teilformeln können äquivalent sein.

Satz 2.13 (Ersetzbarkeitstheorem)
Sei $F \equiv G$. Sei H eine Formel mit mindestens einem Vorkommen der Teilformel F. Dann gilt $H \equiv H'$, wobei H' aus H hervorgeht, indem irgendein Vorkommen von F in H durch G ersetzt wird.

Beweis. Induktion über den Formelaufbau von H[1]. ■

Die Relevanz ist durch die folgenden Äquivalenzen gegeben.

Satz 2.14
Es gelten die folgenden Äquivalenzen:

$(F \wedge F)$	$\equiv$	F	*(Idempotenz)*
$(F \vee F)$	$\equiv$	F	
$(F \wedge G)$	$\equiv$	$(G \wedge F)$	*(Kommutativität)*
$(F \vee G)$	$\equiv$	$(G \vee F)$	
$((F \wedge G) \wedge H)$	$\equiv$	$(F \wedge (G \wedge H))$	*(Associativität)*
$((F \vee G) \vee H)$	$\equiv$	$(F \vee (G \vee H))$	
$(F \wedge (F \vee G))$	$\equiv$	F	*(Absorption)*
$(F \vee (F \wedge G))$	$\equiv$	F	
$(F \wedge (G \vee H))$	$\equiv$	$((F \wedge G) \vee (F \wedge H))$	*(Distributivität)*
$(F \vee (G \wedge H))$	$\equiv$	$((F \vee G) \wedge (F \vee H))$	*(Doppelnegation)*
$\neg\neg F$	$\equiv$	F	
$\neg(F \wedge G)$	$\equiv$	$(\neg F \vee \neg G)$	*(deMorgansche Regeln)*
$\neg(F \vee G)$	$\equiv$	$(\neg F \wedge \neg G)$	
$(F \vee G)$	$\equiv$	F , falls F Tautologie	*(Tautologieregeln)*
$(F \wedge G)$	$\equiv$	G , falls F Tautologie	
$(F \vee G)$	$\equiv$	G , falls F unerfüllbar	*(Unerfüllbarkeitsregeln)*
$(F \wedge G)$	$\equiv$	F , falls F unerfüllbar	

Beweis. Durch Wahrheitstafeln. Z.B. für die zweite Absorptionsregel siehe Beispiel 2.11, und setze $A = F$ und $B = G$ dort. Die erste und die letzte Spalte sind gleich.

■

[1] Dies ist natürlich nur eine *Beweisidee*.

Beispiel 2.15 Beweis der Äquivalenz

$$((A \vee (B \vee C)) \wedge (C \vee \neg A)) \equiv ((B \wedge \neg A) \vee C)$$

$$\begin{aligned}
& ((A \vee (B \vee C)) \wedge (C \vee \neg A)) \\
\equiv\ & (((A \vee B) \vee C) \wedge (C \vee \neg A)) && \text{(Assoziativität und ET)} \\
\equiv\ & ((C \vee (A \vee B)) \wedge (C \vee \neg A)) && \text{(Kommutativität und ET)} \\
\equiv\ & (C \vee ((A \vee B) \wedge \neg A)) && \text{(Distributivität)} \\
\equiv\ & (C \vee (\neg A \wedge (A \vee B))) && \text{(Kommutativität und ET)} \\
\equiv\ & (C \vee ((\neg A \wedge A) \vee (\neg A \wedge B))) && \text{(Distributivität und ET)} \\
\equiv\ & (C \vee (\neg A \wedge B)) && \text{(Unerfüllbarkeitsregel und ET)} \\
\equiv\ & (C \vee (B \wedge \neg A)) && \text{(Kommutativität und ET)} \\
\equiv\ & ((B \wedge \neg A) \vee C) && \text{(Kommutativität und ET)}
\end{aligned}$$

□

Beispiel-Anwendung:

Satz 2.16
Zu jeder Formel F *gibt es eine äquivalent Formel, die nur die Junktoren* $\vee$ *und* $\neg$ *enthält.*

Der (konstruktive) Beweis beschreibt die geschickte Anwendung der Äquivalenzen in Satz 2.14.

Von besonderem Interesse für das „automatische Schließen“ sind die folgenden Normalformen.

Definition 2.17 (Literal, Normalformen)
Ein atomare Formel wird von nun an auch kurz als *Atom* bezeichnet.

Ein *Literal* ist ein Atom oder die Negation eines Atoms. Im ersten Fall ist das Literal *positiv*, im zweiten Fall *negativ*.

Literale werden üblicherweise durch die Buchstaben K und L bezeichnet.

Eine Formel F ist in *konjunktiver Normalform (KNF)* falls sie eine Konjunktion von Disjunktionen von Literalen ist:

$$F = (\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} L_{i,j}))$$

Eine Formel F ist in *disjunktiver Normalform (DNF)* falls sie eine Disjunktion von Konjunktionen von Literalen ist:

$$F = (\bigvee_{i=1}^{n} (\bigwedge_{j=1}^{m_i} L_{i,j}))$$

In beiden Fällen ist also

$$L_{i,j} \in \{P_1, P_2, \ldots\} \cup \{\neg P_1, \neg P_2, \ldots\} \ .$$

□

Theorem 2.18 *Für jede Formel gibt es eine äquivalente Formel in KNF und eine äquivalente Formel in DNF.*

Beweis. Skizze, für den KNF-Teil: Sei F' die gegebene Formel. F' wird mit Hilfe von Äquivalenzen und unter Ausnutzung des Ersetzbarkeitstheorems (Satz 2.14) solange umgeformt, bis eine Formel in KNF resultiert.

Die nötigen Äquivalenzen sind (sie werden von links nach rechts angewendet):

$$\neg\neg G \equiv G \tag{1}$$
$$\neg(G \wedge H) \equiv (\neg G \vee \neg H) \tag{2}$$
$$\neg(G \vee H) \equiv (\neg G \wedge \neg H) \tag{3}$$
$$(F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H)) \tag{4}$$
$$((G \wedge H) \vee F) \equiv ((F \vee G) \wedge (F \vee H)) \tag{5}$$

Klar ist, daß die resultierende Formel äquivalent zu F' ist.

Zu zeigen ist nun:

1. Der Umformungsvorgang ist in jedem Fall endlich.
2. Die resultierende Formel ist in KNF.

■

Fragen in Zusammenhang mit diesem Beweis:

1. Ist stets immer nur eine Äquivalenz anwendbar?
2. Falls nein, spielt die Auswahl der im nächsten Umformungsschritt angewendeten Äquivalenz eine Rolle? D.h. ist das Ergebnis (syntaktisch) immer dasselbe (Äquivalenz gilt ohnehin)?
3. Ist die resultierende Formel (größenordnungsmäßig) genau so groß wie die Eingangsformel?

2.4 Klausellogik

(Klausale Logik, clause logic)

Sei $F = (\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} L_{i,j}))$ eine Formel in KNF. F kann auch geschrieben werden als

$$\begin{array}{lll} F = & & (\neg A_{1,1} \vee \cdots \vee \neg A_{1,k_1} \vee A_{1,k_1+1} \vee \cdots \vee A_{1,m_1}) \\ & \wedge & (\qquad\qquad \cdots \qquad\qquad) \\ & \vdots & \qquad\qquad \vdots \\ & \wedge & (\neg A_{n,1} \vee \cdots \vee \neg A_{n,k_n} \vee A_{n,k_n+1} \vee \cdots \vee A_{n,m_n}) \end{array}$$

oder als

$$\begin{array}{lll} F = & & (A_{1,1} \wedge \cdots \wedge A_{1,k_1} \rightarrow A_{1,k_1+1} \vee \cdots \vee A_{1,m_1}) \\ & \wedge & (\qquad\qquad \cdots \qquad\qquad) \\ & \vdots & \qquad\qquad \vdots \end{array}$$

$$\wedge \quad (A_{n,1} \wedge \cdots \wedge A_{n,k_n} \rightarrow A_{n,k_n+1} \vee \cdots \vee A_{n,m_n})$$

Relevanz: Viele Probleme sind *auf natürliche Art und Weise* mit dieser Struktur, als Wenn-Dann Regeln, formulierbar. Das Resolutionsverfahren (s.u.) benutzt arbeitet auf einer Mengendarstellung von Formeln in KNF. F wird in Klausellogik folgendermaßen als *Klauselmenge* M_F dargestellt:

$$\begin{array}{ll} M_F = \{ & \{\neg A_{1,1}, \ldots, \neg A_{1,k_1}, A_{1,k_1+1}, \ldots, A_{1,m_1}\}, \\ & \{ \qquad\qquad \cdots \qquad\qquad \}, \\ & \qquad\qquad \vdots \\ & \{\neg A_{n,1}, \ldots, \neg A_{n,k_n}, A_{n,k_n+1}, \ldots, A_{n,m_n}\} \\ \} & \end{array}$$

Definition 2.19 (Klausel, Klauselmenge)
Eine *Klausel* ist eine Menge von Literalen. Die leere Menge wird auch als *leere Klausel* bezeichnet, und als $\square$ geschrieben.

Eine *Klauselmenge* ist eine Menge von Klauseln. $\square$

Um die bisherigen Definitionen und Ergebnisse auch auf Klauseln oder Klauselmengen anwenden zu können, wird die folgende Übersetzung definiert.

Idee: Um die Äquivalenz zur KNF-Darstellung zu erhalten, müßen die inneren ,'s als „$\vee$“, und die äußeren ,'s als „$\wedge$“ aufgefaßt werden.

Definition 2.20 (Übersetzung von Klausellogik in die Formeln der Aussagenlogik)
Sei $\mathcal{C}$ eine Klausel und M eine Klauselmenge.

$$\mathcal{F}(\mathcal{C}) = \begin{cases} (P_1 \wedge \neg P_1) & \text{falls } \mathcal{C} = \square \\ \bigvee_{L \in \mathcal{C}} L & \text{sonst} \end{cases}$$

$$\mathcal{F}(M) = \begin{cases} (P_1 \vee \neg P_1) & \text{falls } M = \{\} \\ \bigwedge_{\mathcal{C} \in M} \mathcal{F}(\mathcal{C}) & \text{sonst} \end{cases}$$

$\square$

Wegen der Idempotenz und der Assoziativität der $\wedge$- und $\vee$-Junktoren ist klar, daß für jede KNF-Formel F gilt: $F \equiv \mathcal{F}(M_F)$, wobei M_F die Darstellung von F in Klauselform ist.

Darüberhinaus ist jede endliche Klauselmenge M zu einer Formel in KNF äquivalent.

Falls $\square \notin M$, so ist $\mathcal{F}(M)$ in KNF.

Definition 2.21 (Weitere Schreibweisen)
Eine Klausel

$$\{\neg A_1 \vee \cdots \vee \neg A_k \vee A_{k+1} \vee \cdots \vee A_m\}$$

wird auch geschrieben als

$$A_1, \ldots, A_k \rightarrow A_{k+1}, \ldots, A_m$$

Die Folge der Atome $A_1,\ldots,A_k$ wird als *Rumpf (der Klausel)* und die Folge der Atome $A_{k+1},\ldots,A_m$ wird als *Kopf (der Klausel)* bezeichnet.

Spezialfälle:

$m = 0$: $A_1,\ldots,A_k \rightarrow$ (*Negative Klausel*)
$k = 0$: $\rightarrow A_1,\ldots,A_m$ (*Positive Klausel*)

□

Horn Klauseln

Horn-Klauseln sind Klauseln bestimmter Bauart. Ihnen kommt besondere Bedeutung in der Logik-Programmierung und in der Wissensrepräsentation zu.

Definition 2.22 (Horn-Klausel, Definite Klausel, Horn-Klauselmenge)
Eine *Hornklausel* ist eine Klausel, die höchstens ein positives Literal enthält.

Eine *definite Klausel* ist eine Klausel, die genau ein positives Literal enthält.

Ein *Fakt* (*Faktum*) ist eine positive, definite Klausel. □

Der folgende Algorithmus entscheidet die Erfüllbarkeit einer gegebenen Horn-Klauselmenge M. Dazu sollen die Vorkommen von Literalen in Klauseln markiert werden können.

Markierungsalgorithmus

```
1  Eingabe: Eine endliche Horn-Klauselmenge M.
2  Ausgabe: 'erfüllbar' oder 'unerfüllbar'.
3   Solange es in M (i) eine definite Klausel A1,...,Ak → Ak+1 gibt,
4                   oder (ii) eine negative Klausel A1,...,Ak → gibt,
5        so daß A1,...,Ak bereits markiert sind,
6        und (im Fall (i)) Ak+1 noch nicht markiert ist, tue:
7
8        Im Fall (i) markiere jedes Vorkommen von Ak+1 in M,
9        sonst (Fall (ii) trifft zu) gib "'unerfüllbar"' aus und stoppe.
10
11  Gib "'erfüllbar"' aus und stoppe.
```

Bemerkung: Falls „erfüllbar“ ausgegeben wird, ist ein Modell $\mathcal{A}$ für M bestimmt durch: $\mathcal{A}(A_i) = \mathbf{W}$ gdw. A_i hat eine Markierung.

2.5 Semantische Bäume

Robinson 1968, Kowalski und Hayes 1969.

Anwendungen

- Finden eines Resolutionsbeweises ist äquivalent zur Konstruktion eines „geschlossenen“ semantischen Baumes (i.e. Beweis des Vollständigkeitssatz des Resolutionsverfahrens für die Aussagenlogik).

- Liefert auch *direktes* Beweisverfahren für Aussagenlogik.
- Beweis des Kompaktheitssatzes.
  Anwendung: Vollständigkeitsbeweis des Resolutionsverfahrens für die Prädikatenlogik.

Definition 2.23 (Baum)
Ein *Baum*

- ist ein zusammenhängender, gerichteter, azyklischer Graph,
- in dem jeder Knoten höchstens eine eingehende Kante hat.

In einem *Wurzelbaum* gibt es einen ausgezeichneten Knoten, genannt *Wurzel*, der keine eingehende Kante hat.

In einem *endlichen* Baum gibt es nur endlich viele Knoten (und damit nur endlich viele Kanten).

In einem *endlich verzweigenden* Baum hat jeder Knoten nur endlich viele ausgehende Kanten.

In einem n-adischen Baum hat jeder Knoten höchstens n ausgehende Kanten. 2-adische Bäume heißen auch *Binärbäume*.

In einem *vollständigen* n-adischen Baum hat jeder Knoten (genannt *Blatt*) entweder keine, oder genau n ausgehende Kanten (genannt *innere Knoten*).

Ein *Pfad* P ist eine möglicherweise unendliche Folge von Knoten $P = (N_0, N_1, \ldots)$, wobei N_0 die Wurzel ist, und N_i ein direkter Nachfolger von N_{i-1} ist ($i = 1, \ldots, n$).

Ein *Pfad (der Länge n)* zu einem Knoten N ist ein Pfad der mit einem Knoten $N_n = N$ endet.
Der Knoten N_{n-1} heißt *direkter Vorgängerknoten von* N
Jeder der Knoten $N_0, N_1, \ldots, N_{n-1}$ heißt *Vorgängerknoten von* N.

Ein *(knoten-)markierter* Baum ist ein Baum zusammen mit einer Funktion λ (Labelling-Funktion), welche jedem Knoten ein Element aus einer gegebenen Menge zuordnet.
□

Weitere Begriffe: benachbarte Knoten, Nachfolgerknoten, geordneter Baum, ...

Unterschied zwischen Wurzelbäumen und induktiv definierten „Info1-Bäumen“?

Definition 2.24 (Komplement, Semantischer Baum)
Sei L ein Literal. Das *Komplement von* L ist

$$\overline{L} = \begin{cases} \neg A & \text{falls } L \text{ ein Atom } A \text{ ist} \\ A & \text{falls } L \text{ ein negiertes Atom } \neg A \text{ ist} \end{cases}$$

Ein *semantischer Baum $\mathcal{B}$ (zu einer Menge von Atomen D)* ist ein vollständiger Binärbaum mit Wurzel, in dem

1. die Wurzel mit dem Symbol $\top$ markiert ist, und
2. falls N ein innerer Knoten ist, dann ist ein direkter Nachfolger von N mit einem Literal A markiert (mit $A \in D$), und der andere direkte Nachfolger ist mit dem Literal $\neg A$ markiert, und

3. in dem für jeden Knoten N es kein Literal L gibt so daß $L \in \mathcal{I}(N)$ und $\overline{L} \in \mathcal{I}(N)$, wobei

$$\mathcal{I}(N) = \{\lambda(N_i) \mid N_0, N_1, \ldots, (N_n = N) \text{ ist ein Pfad zu } N \text{ und } 1 \leq i \leq n\ .$$

□

Bemerkung 2.25 (Semantik von $\top$) Übereinkunft: Das Symbol $\top$ wird in allen Belegungen zu **W** ausgewertet □

Definition 2.26 (Atommenge)
Zu einer Klauselmenge M ist die *Atommenge (von M)* die Menge der in M vorkommenden Atome. Ein *semantischer Baum zu* M ist ein semantischer Baum zur Atommenge von M.
□

Definition 2.27 (Vollständiger semantischer Baum)
Ein semantischer Baum zu D ist *vollständig*, falls für jedes Blatt N gilt:

$A \in \mathcal{I}(N)$ oder $\neg A \in \mathcal{I}(N)$, für alle $A \in D$.

□

Bemerkung 2.28

1. Jeder Knoten N in einem semantischen Baum zu D definiert eine Belegung $\mathcal{A}_N$ für eine Teilmenge $D' \subseteq D$ („partielle Belegung für D“):

$$\mathcal{A}_N(A) = \begin{cases} \mathbf{W} & \text{falls } A \in \mathcal{I}(N) \\ \mathbf{F} & \text{falls } \neg A \in \mathcal{I}(N) \end{cases}$$

2. Falls die Atommenge von M endlich ist, so ist in jedem vollständigen Baum für jedes Blatt N die Belegung $\mathcal{A}_N$ passend zu M.
3. Falls die Atommenge von M unendlich ist, so ist jeder vollständige Baum zu M unendlich (mehr noch: hat keine Blätter).
4. Ein vollständiger semantischer Baum kann als eine Aufzählung aller möglichen Belegungen passend zu M angesehen werden ($\mathcal{A}_N \neq \mathcal{A}_{N'}$ für $N \neq N'$).

□

Falls eine Klauselmenge unerfüllbar ist, muß jede passende Belegung mindestens eine Klausel zu **F** auswerten. Dies motiviert die folgende Definition:

Definition 2.29 (Mißerfolgsknoten)
Ein Knoten N in einem semantischen Baum zu M ist ein *Mißerfolgsknoten*, falls

1. es eine Klausel $C \in M$ gibt mit $\mathcal{A}_N \not\models C$, und
2. falls für jeden Vorgängerknoten N' von N gilt:

   Es gibt keine Klausel $C \in M$ so daß $\mathcal{A}_{N'} \not\models C$.

□

Definition 2.30 (Offen, geschloßen)
Ein Pfad P in einem semantischer Baum zu M ist *geschloßen* falls P einen Mißerfolgsknoten enthält, andernfalls ist P *offen*.

Ein semantischer Baum $\mathcal{B}$ zu M ist *geschloßen* falls jeder Pfad geschloßen ist, andernfalls ist $\mathcal{B}$ *offen*. □

Lemma 2.31
Eine Klauselmenge M ist unerfüllbar gdw. es einen vollständigen und geschloßenen semantischen Baum zu M gibt.

Theorem 2.32 (Kompaktheitssatz) *Eine Klauselmenge M ist erfüllbar gdw. jede endliche Teilmenge von M erfüllbar ist.*

2.6 Das Resolutionsverfahren

Das Resolutionsverfahren (kurz: Resolution) (Robinson, 1965) ist ein Kalkül für die Prädikatenlogik.

Ein *Kalkül für* X besteht aus

- einer (entscheidbaren) Menge von Formeln (genannt *Axiome*), und
- einer Kollektion von Umformungsregeln (genannt *Inferenzregeln*), und
- eine Vorschrift, wie die Anwendung der Inferenzregeln zu kombinieren sind (genannt *Ableitungsbegriff*), wobei als Startpunkt von einer gegebenen X-Formelmenge ausgegangen wird.

Im Fall von Resolution:

- Axiome: Keine
- Inferenzregeln: Mit der *Resolutionsregel* werden zwei Klauseln zu einer dritten Klausel kombiniert.
- Ableitung: Eine Folge von Klauseln (genannt *Resolutionsableitung*).
- X: Klauselogik (zunächst: aussagenlogische Klauselogik).

Ein Kalkül ist demnach ein rein syntaktisches („mechanisches“) Verfahren, und bietet sich als Grundlage für Implementierungen an.

2.6.1 Resolution als Widerlegungsverfahren

Übliche Fragestellung:

Gegeben: (i) $\mathcal{T} = \{Ax_1, \ldots, Ax_n\}$ endliche Formelmenge, und
(ii) F Formel.

Frage: Gilt $\mathcal{T} \models F$? (d.h. ist F eine Folgerung von $\mathcal{T}$?)

Mit Resolution:

	$\mathcal{T} \models F$	(1)
gdw.	$\mathcal{T} \cup \{\neg F\}$ unerfüllbar ist (Satz 2.9-2)	(2)
gdw.	die Klauseldarstellung von $Ax_1 \wedge \cdots \wedge Ax_n \wedge \neg F$ unerfüllbar ist	(3)
gdw.	Resolution angewendet auf $Ax_1 \wedge \cdots \wedge Ax_n \wedge \neg F$ die leere Klausel $\square$ produziert.	(4)

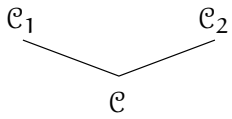
Den Übergang von (3) nach (4) liefert das *Vollständigkeitstheorem*, und den Übergang von (4) nach (3) liefert das *Korrektheitstheorem* der Resolution.

Definition 2.33 (Resolutionsregel)
Seien $\mathcal{C}_1$, $\mathcal{C}_2$ und $\mathcal{C}$ Klauseln. Dann heißt $\mathcal{C}$ *Resolvent* von $\mathcal{C}_1$ und $\mathcal{C}_2$, falls

1. es ein Literal L gibt mit $L \in \mathcal{C}_1$ und $\overline{L} \in \mathcal{C}_2$, und
2. $\mathcal{C} = (\mathcal{C}_1 \setminus \{L\}) \cup (\mathcal{C}_2 \setminus \{\overline{L}\})$

Bemerkungen:

- Sprechweise: „$\mathcal{C}$ wird aus $\mathcal{C}_1$, $\mathcal{C}_2$ nach L resolviert."
- Graphisch:



- Die leere Klausel $\square$ kann als Resolvent auftreten

$\square$

Das folgende Lemma garantiert die Korrektheit des Resolutionsverfahrens.

Lemma 2.34
Sei M eine Klauselmenge, und sei $\mathcal{C}$ Resolvent zweier Klauseln $\mathcal{C}_1 \in M$ und $\mathcal{C}_2 \in M$.
Dann gilt $M \equiv M \cup \{\mathcal{C}\}$.

Definition 2.35
Sei M eine Klauselmenge. Definiere

1. $\mathrm{Res}(M) = M \cup \{\mathcal{C} \mid \mathcal{C}$ ist Resolvent zweier Klauseln in $M\}$
2. $$\begin{aligned} \mathrm{Res}^0(M) &= M \\ \mathrm{Res}^{n+1}(M) &= \mathrm{Res}(\mathrm{Res}^n(M)) \quad \text{, für } n \geq 0. \end{aligned}$$
3. $\mathrm{Res}^*(M) = \bigcup_{n \geq 0} \mathrm{Res}^n(M)$ (der *Resolutionsabschluß* von M)

$\square$

Die Definition von Res^* fordert, daß alle Resolventen gebildet werden. Die folgende Definition vermeidet dies:

Definition 2.36 (Resolutionsableitung)
Sei M eine Klauselmenge. Eine *Resolutionsableitung aus* M ist eine (möglicherweise unendliche) Folge von Klauseln

$$\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n, \ldots \quad ,$$

wobei $\mathcal{C}_i \in M$ ist oder $\mathcal{C}_i$ Resolvent zweier Klauseln $\mathcal{C}_j$ und $\mathcal{C}_k$ ist mit $j,k < i$ (für $i = 1,2,\ldots$).

Eine *Refutation von* M ist eine endliche Resolutionsableitung aus M welche mit der leeren Klausel $\mathcal{C}_n = \square$ endet. $\square$

Der Teil über Aussagenlogik wird mit dem folgendem Hauptergebnis beendet.

Theorem 2.37 (Korrektheit und Vollständigkeit der aussagenlogischen Resolution)
Eine Klauselmenge M ist unerfüllbar gdw. $\square \in \mathrm{Res}^\star(M)$.

Korollar 2.38
Eine Klauselmenge M ist unerfüllbar gdw. es eine Refutation gibt.

Schlußbemerkung: es gibt zahlreiche Verbesserungen des Resolutionsverfahrens. Hauptsächlich kann zwischen zwei Kategorien unterschieden werden:

Restriktion: nicht alle möglichen Anwendungen der Resolutionsregel sind in Ableitungen gestattet.

Redundanz: Gewisse Klauseln können gelöscht werden, d.h. sie brauchen zur Resolventenbildung nicht herangezogen werden.

3 Prädikatenlogik

(predicate logic)

Prädikatenlogik (1. Stufe) ist eine Erweiterung der Aussagenlogik um Ausdrucksmöglichkeiten zur Formulierung daß gewisse Beziehungen *für alle „Objekte“* gelten, oder daß *ein „Objekt“ existiert* so daß eine gewisse Beziehung gilt.

Beispiel:

> Für jedes $\varepsilon > 0$ gibt es ein n_0, so daß für alle $n \geq n_0$ gilt, daß $\text{abs}(f(n) - a) < \varepsilon$.

Wesentliche Bestandteile hier:

- „für alle“, „es gibt“
- Funktionen: abs, f, $-$
- Relationen: $>$, $<$, $\geq$

3.1 Syntax der Prädikatenlogik

Im folgenden sei $i = 1, 2, 3, \ldots$ („Unterscheidungsindex“) und $k = 0, 1, 2, \ldots$ („Stellligkeit“)

1. Eine *Variable* hat die Form x_i.
2. Ein *Prädikatensymbol* hat die Form P_i^k
3. Ein *Funktionssymbol* hat die Form f_i^k
   Ein 0-stelliges Funktionssymbol heißt auch *Konstante*.

Vereinfachende Schreibweisen:

u, v, w, x, y, z	stehen für Variablen
a, b, c	stehen für Konstanten
f, g, h	stehen für Funktionssymbole (Stelligkeit aus dem Kontext klar)
P, Q, R	stehen für Prädikatensymbole (Stelligkeit aus dem Kontext klar)

Die Menge der *Terme* ist wie folgt induktiv definiert:

1. Jede Variable ist ein Term.
2. Falls f ein k-stelliges Funktionssymbol ist, und falls $t_1, \ldots, t_k$ Terme sind, so ist auch $f(t_1, \ldots, t_k)$ ein Term.

   Schreibweise: „c“ statt „$c()$“ (mit c Konstante).

Die Menge PF der *prädikatenlogischen Formeln* ist wie folgt induktiv definiert:

1. Falls P ein k-stelliges Prädikatensymbol ist, und falls $t_1, \ldots, t_k$ Terme sind, so ist $P(t_1, \ldots, t_k) \in PF$ (genannt *atomare Formel*, oder *Atom*).
2. Falls $F \in PF$ und $G \in PF$, so auch $(F \wedge G) \in PF$ und $(F \vee G) \in PF$.
3. Falls $F \in PF$, so auch $\neg F \in PF$.

4. Falls $F \in PF$ und x eine Variable ist, so ist
   - $\forall x\, F \in PF$ („allquantifizierte Formel“), und
   - $\exists x\, F \in PF$ („existentiell quantifizierte Formel“).

   Die Symbole $\forall$, bzw. $\exists$, werden *Allquantor*, bzw. *Existenzquantor* genannt.

Von der Aussagenlogik werden sinngemäß übernommen:

- Die weiteren Junktoren $\rightarrow$, $\leftarrow$, $\leftrightarrow$.
- Präzedenzen der Junktoren; die Quantoren $\forall$ und $\exists$ haben die stärkste Bindungskraft. Weglassen von Klammern $(.)$
- Definition „Teilformel“.

Definition 3.1 (Freie und gebundene Vorkommen von Variablen, Aussage, Matrix)
Ein Vorkommen einer Variablen x in einer Formel F heißt *gebunden*, falls x in einer Teilformel von F der Form $\exists x\, G$ oder $\forall x\, G$ vorkommt.

Andernfalls heißt dieses Vorkommen *frei*.

Eine Formel ohne Vorkommen von freien Variablen heißt *geschlossen*, oder eine *Aussage*.

Die *Matrix* einer Formel F ist diejenige Formel, die man aus F erhält, indem jedes Vorkommen von $\exists$, bzw. $\forall$, samt der dahinterstehenden Variablen gestrichen wird. Die Matrix einer Formel F wird mit $F^\star$ bezeichnet. □

3.2 Semantik der Prädikatenlogik

Eine *Struktur* ist ein Paar $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$, wobei

1. $U_\mathcal{A}$ eine beliebige, aber nicht leere Menge ist
   (auch genannt *Grundmenge*, *Individuenbereich*, *Universum*), und
2. $I_\mathcal{A}$ eine Abbildung ist, die
   - jedem k-stelligen Prädikatensymbol P (das im Definitionsbereich von $I_\mathcal{A}$ liegt) ein k-stelliges Prädikat (Relation) über $U_\mathcal{A}$ zuordnet, und
   - jedem k-stelligen Funktionssymbol f (das im Definitionsbereich von $I_\mathcal{A}$ liegt) eine k-stellige Funktion auf $U_\mathcal{A}$ zuordnet, und
   - jeder Variablen x (die im Definitionsbereich von $I_\mathcal{A}$ liegt) ein Element aus der Grundmenge $U_\mathcal{A}$ zuordnet.

Abkürzende Schreibweisen: $P^\mathcal{A}$ statt $I_\mathcal{A}(P)$, $f^\mathcal{A}$ statt $I_\mathcal{A}(f)$, $x^\mathcal{A}$ statt $I_\mathcal{A}(x)$.

Sei F eine Formel und $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ eine Struktur. A heißt *zu* F *passend*, falls $I_\mathcal{A}$ für alle in F vorkommenden Prädikatensymbole, Funktionssymbole und freien Variablen definiert ist.

Beispiel

Sei $F = \forall x\ P(x, f(x)) \wedge Q(g(a, z))$.

Eine zu F passende Struktur ist:

$$\begin{array}{rcl} U_{\mathcal{A}} & = & \{0,1,2,\ldots\} \\ P^{\mathcal{A}} & = & \{(m,n) \mid m,n \in U_{\mathcal{A}} \text{ und } m < n\} \\ Q^{\mathcal{A}} & = & \{n \in U_{\mathcal{A}} \mid n \text{ ist Primzahl}\} \\ f^{\mathcal{A}} & = & \text{die Nachfolgerfunktion auf } U_{\mathcal{A}}, \text{ also } f^{\mathcal{A}}(n) = n+1 \\ g^{\mathcal{A}} & = & \text{die Additionsfunktion auf } U_{\mathcal{A}}, \text{ also } g^{\mathcal{A}}(m,n) = m+n \\ a^{\mathcal{A}} & = & 2 \\ z^{\mathcal{A}} & = & 3 \end{array}$$

Beispiel für anderen Grundbereich („Herbrand-Universum“):

$$U_{\mathcal{A}} = \{a, f(a), g(a,a), f(g(a,a)), g(f(a),a), \ldots\}$$

Auswertung von Termen und Formeln

Sei F eine Formel und $\mathcal{A}$ eine zu F passende Struktur. Für jeden Term t den man aus den Variablen und Funktionssymbolen von F bilden kann, definiere den *Wert von* t *(in der Struktur* $\mathcal{A}$*)*, $\mathcal{A}(t)$, als:

1. Falls t eine Variable x ist, so ist $\mathcal{A}(x) = x^{\mathcal{A}}$
2. Falls t die Form hat $t = f(t_1, \ldots, t_k)$ (mit f als k-stelligem Funktionssymbol und $t_1, \ldots, t_k$ Terme), so ist

$$\mathcal{A}(f(t_1, \ldots, t_k)) = f^{\mathcal{A}}(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k))$$

Definiere den *(Wahrheits-) Wert von* F *(in der Struktur* $\mathcal{A}$*)*, $\mathcal{A}(F)$, als:

1. Falls F die Form $F = P(t_1, \ldots, t_k)$ hat (mit P als k-stelligem Prädikatensymbol und $t_1, \ldots, t_k$ Terme), so ist

$$\mathcal{A}(P(t_1, \ldots, t_k)) = \begin{cases} \mathbf{W} & \text{falls } (\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k)) \in P^{\mathcal{A}} \\ \mathbf{F} & \text{sonst} \end{cases}$$

2. $\mathcal{A}(F \wedge G) = \begin{cases} \mathbf{W} & \text{falls } \mathcal{A}(F) = \mathbf{W} \text{ und } \mathcal{A}(G) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$
3. $\mathcal{A}(F \vee G) = \begin{cases} \mathbf{W} & \text{falls } \mathcal{A}(F) = \mathbf{W} \text{ oder } \mathcal{A}(G) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$
4. $\mathcal{A}(\neg F) = \begin{cases} \mathbf{W} & \text{falls } \mathcal{A}(F) = \mathbf{F} \\ \mathbf{F} & \text{sonst} \end{cases}$

5. Falls F die Form $F = \forall x\ G$ hat, so ist

$$\mathcal{A}(\forall x\ G) = \begin{cases} \mathbf{W} & \text{falls für alle } d \in U_{\mathcal{A}} \text{ gilt: } A_{[x/d]}(G) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$$

6. Falls F die Form $F = \exists x\ G$ hat, so ist

$$\mathcal{A}(\exists x\ G) = \begin{cases} \mathbf{W} & \text{falls es ein } d \in U_{\mathcal{A}} \text{ gibt mit: } A_{[x/d]}(G) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}$$

Wobei $\mathcal{A}_{[x/d]}$ die Struktur ist, die überall mit $\mathcal{A}$ identisch ist, bis auf die Zuordnung von Variablen. Für diese gilt:

$$\mathcal{A}_{[x/d]}(y) = \begin{cases} d & \text{falls } y = x \\ \mathcal{A}(x) & \text{sonst} \end{cases}$$

Die folgenden Definition sind die offensichtlichen Übertragungen ihrer Gegenstücke aus der Aussagenlogik:

Definition 3.2 (Modell, gültig, erfüllbar)
Eine Struktur $\mathcal{A}$ ist *passend* zu einer Menge M von Formeln, gdw. $\mathcal{A}$ passend zu allen $F \in M$ ist.

Sei F eine Formel. Als äquivalent wird definiert:

- $\mathcal{A}$ ist passend zu F und $\mathcal{A}(F) = \mathbf{W}$.
- $\mathcal{A} \models F$.
- $\mathcal{A}$ ist ein Modell für F.
- F gilt unter der Struktur $\mathcal{A}$.

Als äquivalent wird definiert:

- $\mathcal{A}$ ist passend zu F und $\mathcal{A}(F) = \mathbf{F}$.
- $\mathcal{A} \not\models F$.
- $\mathcal{A}$ ist kein Modell für F.
- F gilt nicht unter der Struktur $\mathcal{A}$.

Eine Formel F heißt

- *erfüllbar*, falls F mindestens ein Modell besitzt, *unerfüllbar* sonst.
- *tautologisch* (oder *Tautologie*), falls jede zu F passende Belegung ein Modell für F ist. Notation: $\models F$ für „F ist Tautologie“. $\not\models F$ für „F ist keine Tautologie“.

Sei M eine Menge von Formeln. M heißt *erfüllbar* gdw.

Es existiert eine Struktur A, so daß für alle $F \in M$ gilt: $\mathcal{A} \models F$

□

Ebenso, wie in der Aussagenlogik:

Definition 3.3 (Folgerung, logische Konsequenz)
Sei M eine Menge von Formeln und G eine Formel.

G ist eine *Folgerung* von M (oder *(logische) Konsequenz* von M), i.Z. $M \models G$, gdw.

> Für alle zu G und zu M passenden Strukturen $\mathcal{A}$ gilt: Falls $\mathcal{A} \models F$ für alle $F \in M$, so auch $\mathcal{A} \models G$.

Für eine Formel F definiere $F \models G$ als $\{F\} \models G$.

Dieselbe Sprechweisen, „Folgerung“ und „Konsequenz“ werden auch hier angewendet. □

Satz 3.4

1. *Die folgenden drei Behauptungen sind äquivalent:*
   a) G *ist eine Folgerung von* F.
   b) $(F \to G)$ *ist eine Tautolgie.*
   c) $(F \wedge \neg G)$ *ist unerfüllbar.*
2. *Die folgenden beiden Behauptungen sind äquivalent:*
   a) G *ist eine Folgerung von* M.
   b) $M \cup \{\neg G\}$ *ist unerfüllbar.*

Beweis. Analog zur Aussagenlogik. ■

Bemerkungen

Prädikatenlogik vs. Aussagenlogik:

- Jede aussagenlogische Formel ist „automatisch“ eine Formel der Prädikatenlogik, indem A_i $(i = 1, 2, \ldots)$ als 0-stelliges Prädikatensymbol P_i^0 aufgefasst wird.
- Jede Formel der Prädikatenlogik in der keine Quantoren vorkommen, z.B.

  $$F = (Q(a) \vee \neg R(f(x), c)) \wedge \neg Q(a)$$

  kann unter Erhaltung der Erfüllbarkeit in eine aussagenlogischen Formel übersetzt werden, durch Identifizierung gleicher prädikatenlogischen Atome mit gleichen aussagenlogischen Atomen, hier z.B.

  $$F' = (A_1 \vee \neg A_2) \wedge \neg A_1$$

- Jede Formel der Prädikatenlogik in der keine Quantoren vorkommen, kann mit aussagenlogischen Mitteln (vgl. Theorem 2.18) in eine äquivalente KNF/DNF umgeformt werden.

Logik höhere Stufe: Formeln wie $F = \forall P\ \exists f\ \forall x\ (P(x) \leftrightarrow f(x, a))$.

3.3 Äquivalenzen und Normalformen

- Alle bisherigen Äquivalenzen (Satz 2.14) sind auch in der Prädikatenlogik gültig.
- Das Ersetzbarkeitstheorem (Satz 2.13) gilt analog (Erweiterung der Fallunterscheidung im Beweis nötig).
- Des weiteren gelten die folgenden Äquivalenzen:

  1. $\neg\forall x\, F \equiv \exists x\, \neg F$
     $\neg\exists x\, F \equiv \forall x\, \neg F$

  2. Falls x in G nicht frei vorkommt, gilt:

     $(\forall x\, F \wedge G) \equiv \forall x\, (F \wedge G)$
     $(\forall x\, F \vee G) \equiv \forall x\, (F \vee G)$
     $(\exists x\, F \wedge G) \equiv \exists x\, (F \wedge G)$
     $(\exists x\, F \vee G) \equiv \exists x\, (F \vee G)$

  3. $(\forall x\, F \wedge \forall x\, G) \equiv \forall x\, (F \wedge G)$
     $(\exists x\, F \vee \exists x\, G) \equiv \exists x\, (F \vee G)$

  4. $\forall x\, \forall y\, F \equiv \forall y\, \forall x\, F$
     $\exists x\, \exists y\, F \equiv \exists y\, \exists x\, F$

Definition 3.5 (Pränexform)
Eine Formel F der Form

$$F = Q_1 x_1 \cdots Q_n x_n\, G, \quad \text{, mit } i \geq 0$$

wobei $Q_1, \ldots, Q_n \in \{\forall, \exists\}$, und G keine Quantoren enthält, heißt Formel in *Pränexform*. □

Sei F eine Formel. *Ziel* der angestrebten Normalformtransformation ist eine Formel F' in Pränexform

$$F' = \forall x_1 \cdots \forall x_n\, G \quad \text{, wobei } G \text{ in KNF ist.}$$

Ideen/Probleme hierbei:

1. *Idee:* Die Äquivalenzen 1–3 erlauben, die Quantoren „nach außen zu treiben".
   Problem: Die Äquivalenzen 2 sind nur eingeschränkt anwendbar.
2. *Problem:* Daran anschließende Elimination der $\exists$-Quantoren.
3. *Idee:* Schließlich wird die KNF-Formel G durch rein aussagenlogische Mittel (vgl. Theorem 2.18) aus der Teilformel ohne den Quantoren-Präfix erhalten.

Zu Problem 1: Die Äquivalenzen 2 sind nur eingeschränkt anwendbar

Definition 3.6 (Substitution)
Sei F eine Formel, x eine Variable und t ein Term. Dann bezeichnet

$$F[x/t]$$

diejenige Formel, die man aus F erhält, indem jedes freie Vorkommen der Variable x in F durch den Term t ersetzt wird. □

Lemma 3.7 (Gebundene Umbenennung)
Sei $F = Qx\ G$ eine Formel mit $Q \in \{\exists, \forall\}$. Sei y eine Variable, die nicht in G vorkommt. Dann gilt

$$F \equiv Qy\ G[x/y]$$

Satz 3.8
Zu jeder Formel F gibt es eine äquivalente Formel F' in Pränexform.

Beweis. Induktion über den Formelaufbau, dabei Anwendung der Äquivalenzen 1-3 und Lemma 3.7 um Anwendung der Äquivalenzen 2 zu ermöglichen. ■

Bemerkung: Dieser Beweis liefert einen Algorithmus zur Konvertierung einer Formel in Pränexform.

Zu Problem 2: Elimination der $\exists$-Quantoren

Definition 3.9 (Skolemisierung)
Sei F eine Formel der Form

$$F = \forall x_1 \cdots \forall x_n\ \exists y\ G$$

Die *direkte Skolemisierung von* F ist die Formel

$$F' = \forall x_1 \cdots \forall x_n\ G[y/f(x_1, \ldots, x_n)]\ ,$$

wobei f ein neues, bisher in F nicht vorkommendes n-stelliges Funktionssymbol ist.

Die *Skolemisierung von* F (*Skolemform von* F) ist die Formel F^{Sk}, die sich durch wiederholter Anwendung von direkter Skolemisierung, so lange wie möglich, ergibt. □

Bemerkung: Offensichtlich enthält F^{Sk} keine $\exists$-Quantoren mehr, falls F in Pränexform vorliegt.

Theorem 3.10 *Eine Formel F ist erfüllbar gdw. die Skolemform F^{Sk} erfüllbar ist.*

3.3.1 Zusammenfassung: Umformungsschritte

Gegeben: eine prädikatenlogische Formel F (mit eventuellen freien Vorkommen von freien Variablen).

Ausgabe: Eine erfüllbarkeitsäquivalente Formel in Skolemform mit Matrix in KNF.

1. Bereinige F durch systematisches Umbenennen der gebundenen Variablen.
   Das Ergebnis sei F_1 ($\equiv F$).
2. Seien $y_1, \ldots, y_n$ die in F (bzw. F_1) vorkommenden freien Variablen.

   Sei $F_2 = \exists y_1 \cdots \exists y_n\ F_1$ (F_2 ist erfüllbarkeitsäquivalent zu F_1).

3. Sei F_3 eine zu F_2 äquivalente Aussage in Pränexform (siehe Satz 3.8).
4. Sei F_4 eine Skolemform zu F_3
   (F_4 ist erfüllbarkeitsäquivalent zu F_3, siehe Theorem 3.10).
5. Ersetze die Matrix von F_4 durch eine äquivalente KNF-Form (siehe Theorem 2.18).

   Dies ist das Ergebnis (es ist erfüllbarkeitsäquivalent zu F).

3.4 Herbrand-Theorie

„Problem“: In einer Struktur $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ kann das Universum $U_\mathcal{A}$ eine *beliebige* Menge sein, und die Interpretationsfunktionen $I_\mathcal{A}$ können ebenso beliebig sein.

Wie soll ein Kalkül damit umgehen?

„Lösung“: Eine *Herbrand-Struktur* hat die folgenden Eigenschaften:

Es wird *ein einziges* bestimmtes Universum eindeutig festgelegt, eben das *Herbrand-Universum*.

Die Interpretation $I_\mathcal{A}$ der Funktionssymbole wird zu gegebenem *Herbrand-Universum* eindeutig festgelegt.

Alleine die Interpretation $I_\mathcal{A}$ der Prädikatssymbole ist offen.

Definition 3.11 (Herbrand-Universum)
Sei F eine Aussage in Skolemform. Das *Herbrand-Universum* ist folgendermaßen definiert:

1. Alle in F vorkommenden Konstanten sind in $D(F)$. Falls F keine Konstante enthält, so ist a in $D(F)$.
2. Für jedes in F vorkommende n-stellige Funktionssymbol f und Terme $t_1, \ldots, t_n$ in $D(F)$ ist der Term $f(t_1, \ldots, t_n)$ in $D(F)$.

□

Definition 3.12 (Herbrand-Strukturen)
Sei F eine Aussage in Skolemform. Dann heißt jede zu F passende Struktur $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ eine *Herbrand-Struktur*, falls folgendes gilt:

1. $U_\mathcal{A} = D(F)$
2. für jedes in F vorkommende n-stellige Funktionssymbol f und $t_1, \ldots, t_n \in D(F)$ ist $f^\mathcal{A}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$

□

Bemerkungen:

- Terme werden „auf sich selbst abgebildet“ - Syntax und Semantik von Termen ist *dasselbe*.
- In Herbrand-Strukturen ist einzig die Interpretation $P^\mathcal{A}$ der Prädikatensymbole nicht festgelegt.

Die Relevanz der Herbrand-Theorie ist durch das folgende Theorem gegeben:

Theorem 3.13 *Sei* F *eine Aussage in Skolemform. Dann ist* F *genau dann erfüllbar, wenn* F *ein Herbrand-Modell besitzt.*

Die Relevanz dieses Theorems ist durch den folgenden Sachverhalt gegeben:

	$M \models F$, für eine endliche Menge von Aussagen und eine Aussage F	
gdw.	$M \cup \{\neg F\}$ unerfüllbar ist	(Satz 3.4.2.a)
gdw.	$G = \bigwedge_{H \in M} H \wedge \neg F$ unerfüllbar ist	
gdw.	Die Skolemform G^{Sk} von G unerfüllbar ist	(Theorem 3.10)
gdw.	G^{Sk} kein Herbrand-Modell besitzt	(Theorem 3.13)
gdw.	die Herbrand-Expansion $E(G^{Sk})$ aussagenlogisch unerfüllbar ist	(Theorem 3.15)
gdw.	eine endliche Teilmenge von $E(G^{Sk})$ aussagenlogisch unerfüllbar ist	(Kompaktheit, Theorem2.32)
gdw.	der „Algorithmus von Gilmore“ mit „unerfüllbar“ terminiert (s.u.)	(6)

Diese Äquivalenzen bestimmen somit ein Semi-Entscheidungsverfahren für die Prädikatenlogik.

Beweis. [Theorem 3.13] Skizze: Von rechts nach links: klar.

Von links nach rechts: Sei $\mathcal{A} \models F$ gegebenes Modell. Sei $\mathcal{B}$ eine Herbrand-Interpretation, wobei die Prädikatensymbole P wie folgt definiert werden:

$$(t_1, \ldots, t_n) \in P^{\mathcal{B}} \quad \text{gdw.} \quad (\mathcal{A}(t_1), \ldots, \mathcal{A}(t_n)) \in P^{\mathcal{A}}$$

Zeige nun $\mathcal{B} \models F$ durch Induktion über die Anzahl k der All-Quantoren in F. Dazu wird die Behauptung verstärkt zu

> „falls $\mathcal{A} \models G$, dann $\mathcal{B} \models G$, wobei G eine Formel in Skolemform ist, die aus den Bestandteilen von F aufgebaut ist“.

$k = 0$**:** unmittelbar über die Definition von $\mathcal{B}$.

$k > 0$**:** Die verstärkte Behauptung gelte für Formeln mit weniger als k All-Quantoren.

G ist von der Form $G = \forall x\, H$. Nach Voraussetzung gilt $\mathcal{A} \models G$.

	$\mathcal{A} \models G$	(1)
gdw.	$\mathcal{A} \models \forall x\, H$	(2)
gdw.	für alle $d \in U_{\mathcal{A}}$: $\mathcal{A}_{[x/d]}(H) = \mathbf{W}$	(3)
dann	für alle $d \in U_{\mathcal{A}}$ mit $d = \mathcal{A}(t)$ für ein $t \in D(G)$: $\mathcal{A}_{[x/d]}(H) = \mathbf{W}$	(4)
gdw.	für alle $t \in D(G)$: $\mathcal{A}_{[x/\mathcal{A}(t)]}(H) = \mathbf{W}$	(5)
gdw.	für alle $t \in D(G)$: $\mathcal{A}(H[x/t]) = \mathbf{W}$	(6)
dann (nach I.V.)	für alle $t \in D(G)$: $\mathcal{B}(H[x/t]) = \mathbf{W}$	(7)
gdw.	$\mathcal{B} \models \forall x\, H$	(8)
gdw.	$\mathcal{B} \models G$	(9)

Der Übergang von (5) nach (6) ist durch das (hier) unbewiesene Überführungslemma gegeben.

■

Korollar zu Theorem 3.13: Jede erfüllbare Formel besitzt bereits ein abzählbares Modell.

Konsequenz: „Die reellen Zahlen“ können nicht durch eine Formel der Prädikatenlogik eindeutig charakterisiert werden!

Definition 3.14 (Herbrand-Expansion)
Sei

$$F = \forall y_1 \cdots \forall y_n \, G$$

eine Aussage in Skolemform. Dann ist $E(F)$, die *Herbrand-Expansion* von F, definiert als

$$E(F) = \{G[y_1/t_1] \cdots [y_n/t_n] \mid t_1, \ldots, t_n \in D(F)\}$$

□

$E(F)$ ist die Menge aller Formeln die entsteht durch alle möglichen Ersetzungen der Variablen in G durch Terme des Herbrand-Universums. $E(F)$ kann somit als (i.A. unendliche) Menge von aussagenlogischen Formeln angesehen werden.

Theorem 3.15 (Gödel-Herbrand-Skolem) *Für jede Aussage F in Skolemform gilt: F ist erfüllbar genau dann, wenn die Formelmenge $E(F)$ im aussagenlogischen Sinn erfüllbar ist.*

Das Theorem sagt aus, daß „Prädikatenlogik durch Aussagenlogik approximiert werden kann“.

Der Algorithmus von Gilmore

Algorithmus von Gilmore

```
Eingabe: eine Aussage F in Skolemform.

Sei F1,F2,...,Fn,... eine beliebige Aufzählung von E(F).

n := 0;
Wiederhole
    n := n+1
bis (F1 ∧ F2 ∧ ··· ∧ Fn) unerfüllbar ist.

Ausgabe: 'unerfüllbar'.
```

Der Algorithmus von Gilmore ist partiell korrekt, d.h. falls er terminiert dann ist $E(F)$ unerfüllbar.

3.5 Prädikatenlogische Resolution

3.5.1 Motivation

Der Erfüllbarkeitstest im Algorithmus von Gilmore kann im Prinzip durch aussagenlogische Resolution erledigt werden.

Das entstehende Verfahren ist aber nicht optimal.

Beispiel: Sei $F = \forall x\ (P(x) \wedge \neg P(f(x)))$ eine Aussage in Skolemform. Dann ist:

$D(F) = \{a, f(a), f(f(a)),\ \ldots\}$

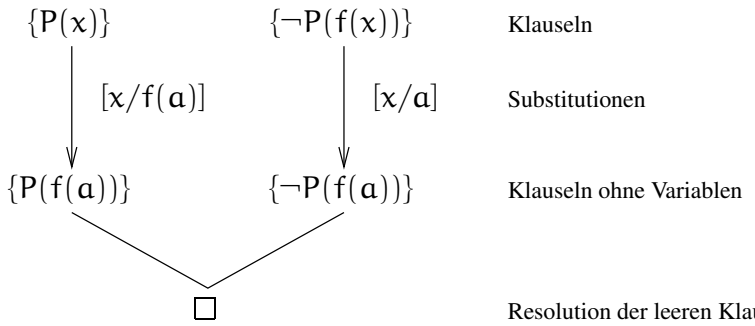
$E(F) = \{P(a) \wedge \neg P(f(a)),\ P(f(a)) \wedge \neg P(f(f(a))), \ldots\}$

Die Klauseldarstellung der ersten beiden Elemente (F_2 im Algorithmus von Gilmore) ist

$$\{\{P(a)\},\{\neg P(f(a))\},\{P(f(a))\},\{\neg P(f(f(a)))\}\}$$

Von den vier Klauseln werden nur zwei Klauseln benötigt, um die leere Klausel $\square$ herzuleiten (laut Korollar 2.38 reicht die Beschränkung auf solche, relevante Klauseln aus).

Die Beschränkung auf die tatsächlich benötigten Elemente der Herbrand-Expansion kann mit Hilfe von Substitutionen folgendermaßen ausgedrückt werden:



Im Folgenden: Entwicklung eines prädikatenlogischen Resolutionskalküls, der *direkt* auf Klauseln mit Variablen arbeitet, ohne die Substitutionen „raten“ zu müssen.

3.5.2 Begriffe

Klauselform: Sei $F = \forall x_1 \cdots \forall x_n\ G$ eine Aussage in Skolemform, wobei G in KNF ist (vgl. Zusammenfassung in Abschnitt 3.3.1).

Dann ist die *Klauselform* von F die Klauselmenge die aus G gewonnen wird, wie in Abschnitt 2.4.

Umgekehrt kann jede Klauselmenge als Aussage in Skolemform betrachtet werden, indem die Klauselmenge als KNF-Formel geschrieben wird und mit einem $\forall$-Quantorenpräfix versehen wird, der alle darin vorkommenden Variablen bindet.

Substitution: Eine *Substitution* σ ist eine endliche Menge von Variable/Term Paaren, mit paarweise verschiedenen ersten Elementen (i.e. Variablen), geschrieben als

$$\sigma = [x_1/t_1, \ldots, x_n/t_n] \ .$$

Sei F eine Formel. Dann bezeichnet $F\sigma$ diejenige Formel, die man aus F erhält, indem simultan jedes freie Vorkommen der Variablen x_i in F durch t_i ersetzt wird (vgl. Definition 3.6).

Grundsubstitution: Sei X ein Term, ein Atom, ein Literal, eine Klausel oder eine Menge von diesen. Eine Substitution σ heißt *Grundsubstitution* (für X), falls $X\sigma$ keine Variablen enthält.

Jedes solches $X\sigma$ heißt *Grundinstanz* (von X).

Damit können die bisherigen Ergebnisse umgeschrieben werden zu:

Satz 3.16 (Grundresolutionssatz)
Eine Klauselmenge M ist unerfüllbar gdw. es eine Refutation von M^{Gr} gibt, wobei M^{Gr} eine endliche Menge von Grundinstanzen von Klauseln aus M ist.

Beweis.

	M ist unerfüllbar
gdw.	$E(M)$ ist unerfüllbar (Gödel-Herbrand-Skolem, Theorem 3.15)
gdw.	Eine endliche Teilmenge $M' \subseteq E(M)$ ist unerfüllbar (Kompaktheit, Theorem 2.32)
gdw.	es gibt eine Refutation von M' (Korollar 2.38)
gdw.	es gibt eine Refutation einer Menge $M^{Gr} = M'$ von Grundinstanzen von Klauseln aus M

■

3.5.3 Unifikation

Idee allgemein: Gegeben Beschreibungen von Mengen von Objekten

Beispiel: (i) „rote Autos“, (ii) „schnelle Autos“ und (iii) „Autos der Marke BMW“.

Gesucht ist eine möglichst allgemeine Beschreibung der Objekte welche die gegebenen Beschreibungen gemeinsam erfüllen.

Definition 3.17 (Unifikator)
Gegeben zwei Terme s und t. Eine Substitution σ ist ein *Unifikator* (für s und t) gdw. $s\sigma = t\sigma$.

Ein Unifikator σ ist *allgemeinster* Unifikator (most general unifier, MGU) gdw. es für jeden Unifikator δ eine Substitution σ' gibt, so daß

$$s\delta = (s\sigma)\sigma'$$

□

Man kann zeigen, daß allgemeinste Unifikatoren *eindeutig* bestimmt sind, bis auf Umbenennung.

Beispiel 3.18 $s = Auto(rot, y, z), \quad t = Auto(u, v, bmw)$.

$\delta = [u/rot,\ y/schnell,\ v/schnell,\ z/bmw]$ ist (nicht-allgemeinster) Unifikator für s und t.

$\sigma = [u/rot,\ y/v,\ z/bmw]$ ist allgemeinster Unifikator für s und t. □

3.5.4 Ein Unifikationsalgorithmus

Vorbemerkung: Ein *Unifikationsproblem* U ist eine endliche Menge von Paaren von Termen, geschrieben als

$$U = \{s_1 = t_1, \ldots, s_n = t_n\} \ .$$

Eingabe: Zwei Terme s und t.

1. Setze $U = \{s = t\}$

2. Solange wie möglich, wende die folgenden Umformungsregeln auf U an:

$$\{x = x\} \cup N \longrightarrow N \qquad \text{(Trivial)}$$

$$\{x = t\} \cup N \longrightarrow \{x = t\} \cup N[x/t] \qquad \text{(Bindung)}$$
falls x in N vorkommt und x nicht in t vorkommt

$$\{x = t\} \cup N \longrightarrow \text{FAIL} \qquad \text{(Occur Check)}$$
falls t keine Variable ist und x in t vorkommt

$$\{f(s_1, \ldots, s_m) = f(t_1, \ldots, t_m)\} \cup N \longrightarrow \{s_1 = t_1, \ldots, s_m = t_m\} \cup N \qquad \text{(Dekomposition)}$$

$$\{f(s_1, \ldots, s_m) = g(t_1, \ldots, t_m)\} \cup N \longrightarrow \text{FAIL} \qquad \text{(Konflikt)}$$
falls $f \neq g$

$$\{t = x\} \cup N \longrightarrow \{x = t\} \cup N \qquad \text{(Orientierung)}$$
falls t keine Variable ist

Ausgabe: siehe Satz 3.19.

Satz 3.19 (Korrektheit und Vollständigkeit des Unifikationsalgorithmus)
Der Unifikationsalgorithmus terminiert und es tritt genau einer der beiden folgenden Fälle ein.

1. *Erfolgsfall:*

   a) U *ist von der Form* $U = \{x_1 = t_1, \ldots, x_n = t_n\}$, *und*

   b) $x_i \neq x_j$, *für* $1 \leq i, j \leq n$ *mit* $i \neq j$, *und*

   c) x_i *kommt nicht in* t_j *vor, für* $1 \leq i, j \leq n$.

   In diesem Fall ist

   $$\sigma = [x_1/t_1, \ldots, x_n/t_n]$$

   allgemeinster Unifikator von s *und* t.

2. *Mißerfolgsfall:* $U = \text{FAIL}$.
   In diesem Fall gibt es keinen Unifikator für s *und* t.

3.5.5 Ein Resolutionsverfahren

Definition 3.20 (Variante, Variablendisjunkt)
Zwei Klauseln $\mathcal{C}_1$ und $\mathcal{C}_2$ sind *Varianten*, falls es Substitutionen ρ_1 und ρ_2 gibt so daß

$$\mathcal{C}_1\rho_1 = \mathcal{C}_2 \quad \text{und} \quad \mathcal{C}_1 = \mathcal{C}_2\rho_2 \ .$$

Die Substitutionen ρ_1 und ρ_2 heißen *Umbenennungssubstitutionen*

Zwei Klauseln $\mathcal{C}_1$ und $\mathcal{C}_2$ sind *variablendisjunkt* gdw. es keine Variable gibt die sowohl in $\mathcal{C}_1$ als auch in $\mathcal{C}_2$ vorkommt. □

Anschaulich: Zwei Klauseln sind Varianten, wenn sie gegenseitig durch systematisches Umbenennen der Variablen ineinander überführt werden können.

Beispiele

- $\{p(x), q(x)\}$ und $\{p(y), q(y)\}$ sind Varianten.
- $\{p(x,y), q(x,y)\}$ und $\{p(y,x), q(y,x)\}$ sind Varianten.
- $\{p(x), q(x)\}$ und $\{p(y), q(z)\}$ sind keine Varianten.
- $\{p(x), q(x)\}$ und $\{p(y), q(a)\}$ sind keine Varianten.

Es werden die folgenden beiden Inferenzregeln benötigt:

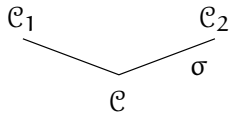
Definition 3.21 (Prädikatenlogische Resolutionsregel)
Seien $\mathcal{C}_1$, $\mathcal{C}_2$ und $\mathcal{C}$ Klauseln. O.b.d.A seien $\mathcal{C}_1$ und $\mathcal{C}_2$ variablendisjunkt (andernfalls wird für $\mathcal{C}_1$ eine variablendisjunkte Variante genommen).

Dann heißt $\mathcal{C}$ *Resolvent* von $\mathcal{C}_1$ und $\mathcal{C}_2$, falls

1. es Literale $P(s_1,\ldots,s_n) \in \mathcal{C}_1$ und $\neg P(t_1,\ldots,t_n) \in \mathcal{C}_2$ gibt, und
2. es einen MGU σ für $P(s_1,\ldots,s_n)$ und $P(t_1,\ldots,t_n)$ gibt, und
3. $\mathcal{C} = (\mathcal{C}_1\sigma \setminus \{P(s_1,\ldots,s_n)\sigma\}) \cup (\mathcal{C}_2\sigma \setminus \{\neg P(t_1,\ldots,t_n)\sigma\})$

Bemerkungen:

- Diese Regel verallgemeinert die aussagenlogische Resolutionsregel (Def. 2.33).
- Sprechweise: „$\mathcal{C}$ wird aus $\mathcal{C}_1$, $\mathcal{C}_2$ nach $P(s_1,\ldots,s_n)$ und $\neg P(t_1,\ldots,t_n)$ mit MGU σ resolviert."
- Graphisch:



- Die leere Klausel □ kann als Resolvent auftreten

□

Es werden die folgenden beiden Inferenzregeln benötigt:

Definition 3.22 (Prädikatenlogische Faktorisierungsregel)
Seien $\mathcal{C}_1$ und $\mathcal{C}$ Klauseln. Dann heißt $\mathcal{C}$ *Faktor* von $\mathcal{C}_1$, falls

1. es zwei Literale $P(s_1,\ldots,s_n), P(t_1,\ldots,t_n) \in \mathcal{C}_1$ gibt, und
2. es einen MGU σ für $P(s_1,\ldots,s_n)$ und $P(t_1,\ldots,t_n)$ gibt, und
3. $\mathcal{C} = \mathcal{C}_1\sigma$

Bemerkungen:

- Diese Regel ist wegen der Mengenschreibweise in der Aussagenlogik ohne Bedeutung.
- Sprechweise: „$\mathcal{C}$ ist ein Faktor aus $\mathcal{C}_1$ nach $P(s_1,\ldots,s_n)$ und $P(t_1,\ldots,t_n)$ mit MGU σ."
- Graphisch:
$$\begin{array}{c}\mathcal{C}_1\\ \big|\,\sigma\\ \mathcal{C}\end{array}$$
- Die leere Klausel $\square$ kann nicht als Resolvent auftreten

□

Analog zur Aussagenlogik (Def. 2.35) wird der Resolutionsabschluß eingeführt:

Definition 3.23
Sei M eine Klauselmenge. Definiere

1. $\begin{aligned}\mathrm{Res}(M) = M \;&\cup\; \{\mathcal{C} \mid \mathcal{C} \text{ ist Resolvent zweier Klauseln in } M\}\\ &\cup\; \{\mathcal{C} \mid \mathcal{C} \text{ ist Faktor einer Klausel in } M\}\end{aligned}$
2. $\begin{aligned}\mathrm{Res}^0(M) &= M\\ \mathrm{Res}^{n+1}(M) &= \mathrm{Res}(\mathrm{Res}^n(M)) \quad \text{, für } n \geq 0.\end{aligned}$
3. $\mathrm{Res}^\star(M) = \bigcup_{n\geq 0}\mathrm{Res}^n(M)$ (der *Resolutionsabschluß* von M)

□

Wie in der Aussagenlogik (Def. 2.36) wird der Begriff *Resolutionsableitung* eingeführt.

Dies ist das Hauptergebnis:

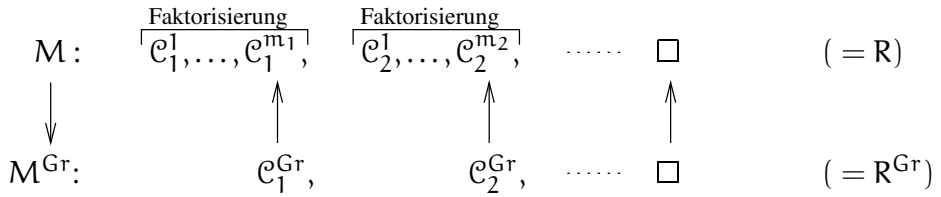
Theorem 3.24 (Korrektheit und Vollständigkeit der prädikatenlogischen Resolution)
Eine Klauselmenge M ist unerfüllbar gdw. $\square \in \mathrm{Res}^\star(M)$.

Das Korollar 2.38 gilt analog.

Beweis. Skizze. Korrektheit: Ähnlich zur Aussagenlogik.

Vollständigkeit: Sei M unerfüllbar. Sei M^{Gr} wie im Grundresolutionssatz 3.16 erwähnt, und sei R^{Gr} eine Refutation von M^{Gr}. Die einzelnen Resolutionsschritte in R^{Gr} werden durch prädikatenlogischen Resolutionsschritte mit vorangehenden Faktorisierungsschritten simuliert („Lifiting"). Die in dieser Refutation R benutzten Klauseln sind in $Res^\star(M)$ enthalten. Deshalb folgt auch $\square \in \mathrm{Res}^\star(M)$.

Schematisch:



($\downarrow$: Instantiierung, $\uparrow$: Lifting) ■

4 Logikprogrammierung mit Prolog

4.1 Einführung

- Die *Logikprogrammierung* ist zusammen mit der funktionalen, der objektorientierten und der prozeduralen Programmierung ein Hauptparadigma der Programmierung.
- Es gibt mehrere Logikprogrammiersprachen. Die prominenteste ist sicherlich *Prolog* (Colmerauer, 1972). Ein altes, aber immer noch gutes Lehrbuch ist:

  W.F. Clocksin, C.S. Mellish. *Programming in Prolog*. Springer, 1984.

  Logikprogrammierung Web-Seite: `archive.comlab.ox.ac.uk/logic-prog.html`.
- Die grundsätzliche Idee des Logikprogrammierens ist (Kowalski):

  Algorithm = Logic + Control.

  Überspitzt gesagt: Der Programmierer gibt nur an, *was* getan werden soll, und die Maschine findet die Lösung. Das funktioniert in der Praxis *so* nicht, aber zu einem Teil eben schon...
- Anwendungen: Prolog ist sehr gut geeignet zum *Rapid Prototyping*, und wird oft als eine Hauptprogrammiersprache für Künstliche-Intelligenz Anwendungen zitiert (v.a. 80er Jahre Expertensysteme)
- Es gibt einige Public-Domain Implementierungen, für zahlreiche Plattformen, Interpreter, Compiler nach Maschinencode, z.B.
  - GNU Prolog: `pauillac.inria.fr/~diaz/gnu-prolog/`
  - SWI-Prolog: `http://www.swi.psy.uva.nl/projects/SWI-Prolog/`

4.2 Prolog Programme – Syntax und Semantik

Ein *Prolog Programm* besteht aus *Fakten* und *Regeln*.

Fakten geben Relationen zwischen Objekten an, z.B.

```
groesser(2,1).
vater(tom,jane).
aelter(vater_von(X),X).
```

Als Formeln:

$$groesser(2,1)$$
$$vater(tom,jane)$$
$$\forall x\ aelter(vater\_von(X),X)$$

Regeln werden verwendet, um neue Relationen aus vorhandenen Relationen zu definieren, z.B.

```
vater(X,jane) :- maennlich(X), elter(X,jane).
vater(X,jane) :- bruder(Y,jane), vater(X,Y).
```

Als Formeln:

$$\forall x\ (vater(x,jane) \leftarrow maennlich(x) \wedge elter(x,jane))$$
$$\forall x\, \forall y\ (vater(x,jane) \leftarrow bruder(y,jane) \wedge vater(x,y))$$
$$(\equiv \quad \forall x\ (vater(x,jane) \leftarrow \exists y\ (bruder(y,jane) \wedge vater(x,y))) \quad)$$

Mit einer *Anfrage* (auch *Ziel* genannt) wird die Abarbeitung eines Prolog Programms ausgelöst.

Beispiel-Anfrage:

```
?- vater(X,jane).
```

Informelle Bedeutung:

„Existiert ein `X` so daß `vater(X,jane)` gilt? Falls ja, welche(s)?“

Die Lösung(en) der Anfrage werden als *Antwort(en)* bezeichnet.

Die Abarbeitung der Anfrage erfolgt durch *zielorientiertes Schließen* (backward chaining): Mit der Regel

```
vater(X,jane) :- maennlich(X), elter(X,jane).
```

ergibt sich aus der Beispiel-Anfrage dann

„Existiert ein `X` so daß `maennlich(X)` und `elter(X,jane)` gilt? Falls ja, welches?“

Diese Art der Abarbeitung wird rekursiv fortgesetzt bis zur Faktenebene.

Definition 4.1 (Syntax von Prolog Programmen)
Eine *(Prolog) Atom* ist von der Form

$$P(t_1, \ldots, t_n)\ ,$$

wobei P ein Prädikatssymbol ist und $t_1, \ldots, t_n$ Terme sind. Variablen beginnen dabei mit Grossbuchstaben oder mit „_“. Prädikats- und Funktionssymbole beginnen mit Kleinbuchstaben.

Ein *Prolog Programm* besteht aus Fakten und Regeln. Ein *Fakt* ist von der Form

$F.$ (mit F Atom).

Eine *Regel* ist von der Form

`H :- B1, ... , Bn.` (mit $H, B_1, \ldots, B_n$ Atome, $n \geq 1$).

Das Atom H heißt *Kopf* (head) der Regel, und die Liste $B_1, \ldots, B_n$ heiß *Rumpf* (body) der Regel.

Die Gesamtheit aller Fakten und Regeln mit dem gleichen Prädikatssymbol P bei F bzw. H wird als P-*Prozedur* bezeichnet.

Eine *Anfrage* (goal, *Ziel*) ist von der Form

`?- G1, ... , Gn.` (mit $G_1, \ldots, G_n$ Atome,$n \geq 1$).

□

Semantik von Prolog Programmen

In Kurzfassung:

- Ein Prolog Programm kann auf naheliegende Weise als eine Menge M von definiten Klauseln betrachtet werden.
- Eine Anfrage `?- Q.` kann als die Aufgabe aufgefaßt werden zu zeigen daß

  $$M \models_H \exists x_1, \ldots, x_n \, Q$$

  gilt, wobei
  - $x_1, \ldots, x_n$ alle (freien) Variablen in Q sind, und
  - $\models_H$ die Folgerungsrelation bezüglich Herbrand-Interpretationen ist.
- Äquivalent dazu: Zeige daß die *Horn*-Klauselmenge

  $$M \cup \{\neg Q\}$$

  kein Herbrand-Modell hat.
- *Im Prinzip* ist deshalb das Resolutionsverfahren aus Abschnitt 3.5 anwendbar.
- Dem Prolog Abarbeitungsmechanismus liegt die spezielle Variante *SLD-Resolution* zu Grunde, um auf die Bedürfnisse der *Programmierung* einzugehen (Berechnen von Antworten, Reihenfolgeaspekte).

4.3 Abarbeitung von Prolog Programmen – SLD Resolution

Diskussion an Hand des folgenden Beispiel-Programmes:

```
maennlich(albert).
maennlich(edward).
weiblich(alice).
weiblich(victoria).
eltern(edward, victoria, albert).
eltern(alice, victoria, albert).
```

Beispiel-Abarbeitungen mit System-Interaktion.

1. `?- maennlich(edward).`
   yes

2. `?- maennlich(alice).`
   no

3. `?- weiblich(X).`
   X = alice ;
   X = victoria ;
   no

Erweiterung des Beispiels

```
maennlich(albert).
maennlich(edward).
weiblich(alice).
weiblich(victoria).
eltern(edward, victoria, albert).
eltern(alice, victoria, albert).
schwester_von(X,Y) :- weiblich(X), eltern(X,M,W), eltern(Y,M,W).
```

Beispiel-Abarbeitungen mit System-Interaktion.

1. `?- schwester_von(alice,Y).`
   Y = edward ;
   no

2. `?- schwester_von(X,Y).`
   X = alice, Y = edward ;
   X = alice, Y = alice ;
   no

4.3.1 Ein SLD-Resolutionsalgorithmus

SLD-Resolution

Eingabe: Eine Anfrage G = ?- G_1, ... , G_n.
Ausgabe: Eine Antwortsubstitution σ ('G ist erfolgreich')
oder no ("'G failed"').

Falls ?- G_1, ... , G_n nicht leer ist (i.e. $n > 0$), dann
1. Sei H oder H :- B_1, ... , B_n
   das nächste noch nicht herangezogene Fakt oder
   die nächste noch nicht herangezogene Regel
   (es wird oben im Programm begonnen),
   so daß G_1 und H unifizierbar sind.

```
11
12      2. Falls nicht existent, dann ist das Ergebnis no (G1 'failed').
13
14      3. Sei σ ein MGU für G1 und H (ggf. Varianten der Fakten/Regeln benutzen!)
15          Rufe SLD-Resolution rekursiv mit ?- (B1, ... ,Bn,G1, ... , Gn)σ auf.
16          Falls das Ergebnis no ist, dann ('Backtracking')
17               Gehe zu Schritt 1
18          sonst ('G1 erfolgreich')
19               Sei σ' das Ergebnis des rekursiven Aufrufs.
20               Das Ergebnis ist die kombinierte Substitution σσ'.
21
22  sonst
23          Das Ergebnis ist die leere Substitution [].
```

Bemerkung: Der SLD-Resolutionsalgorithmus zeigt

- daß Prolog Programme *von oben nach unten* abgearbeitet werden,
- daß Regeln *von links nach rechts* abgearbeitet werden,
- daß Ein/Ausgabe „Parameter“ vertauscht werden können, und
- daß die Abarbeitung durch *Suche* (Backtracking) erfolgt.

Der Occur-check (siehe Unifikationsalgorithmus, Abschnitt 3.5.4) wird aus Effizienzgründen in Prolog Implementierungen meist weggelassen.

4.3.2 Negation

Im Beispiel oben wurde die unerwünschte Antwortsubstitution

X = alice, Y = alice

berechnet. Durch den *Negation by failure* Operator `not` kann dieses Problem behoben werden. Modifiziertes Programm (== bedeutet „syntaktische Gleichheit“):

```
maennlich(albert).
maennlich(edward).
weiblich(alice).
weiblich(victoria).
eltern(edward, victoria, albert).
eltern(alice, victoria, albert).
schwester_von(X,Y) :- weiblich(X), eltern(X,M,W), eltern(Y,M,W),
                      not(X == Y).
```

Damit:

```
?- schwester_von(X,Y).
```
X = alice, Y = edward ;
no

Bedeutung: `not(G)` ist erfolgreich falls G failed.

4.3.3 Der Cut-Operator !

Der `not` Operator ist vordefiniert, kann aber auch wie folgt mit Prolog definiert werden:

```
not(G) :- call(G), !, fail.
not(G).
```

Hierbei bedeutet:

- `call(G)` – Der SLD-Resolutionsalgorithmus wird mit G aufgerufen.
- `fail` – failed immer (und löst Backtracking aus).
- `!` –

  In „Richtung von links nach rechts“: kein Effekt, `!` ist unmittelbar erfolgreich.

  In „Richtung von rechts nach links“: statt Backtracking failed die Prozedur die den `!` enthält.

Ende Logikprogrammierung!

Der einzig elementare fehlende Aspekt sind Prolog Datentypen, in erster Linie „Listen“.