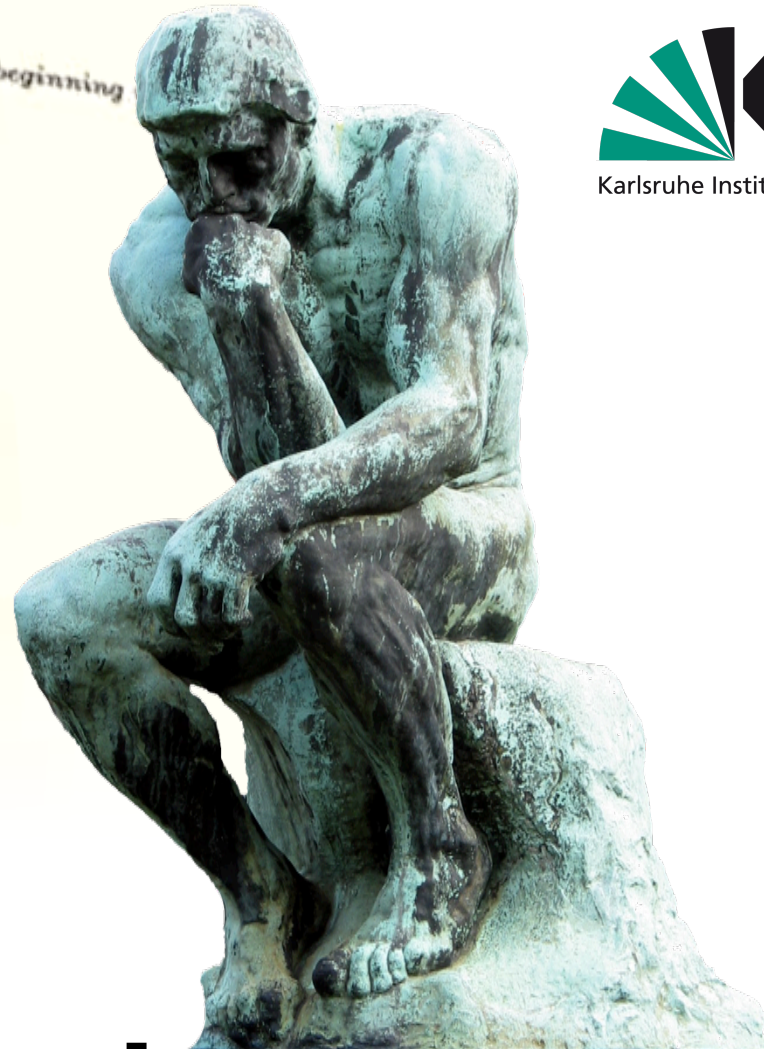
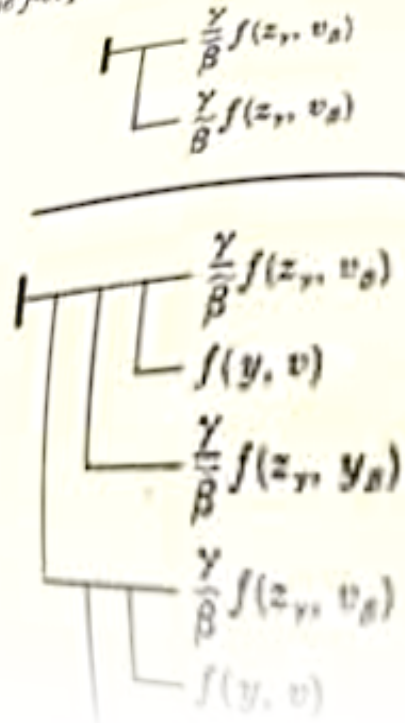


$(z \equiv x)$
 $f(z, z)$
... follows z in the f -sequence beginning



Teil 8

Resolution und Prädikatenlogik erster Stufe

Widerlegungsbeweise

Grundidee Widerlegungsbeweise:

- Wir wollen beweisen, dass aus einer gegebenen Formelmenge Φ eine Behauptung (Formel) φ semantisch folgt.
- Es ist also zu zeigen, dass jede Interpretation, die Modell für alle Formeln aus Φ ist, auch Modell von φ ist.
- Nun kann aber keine Interpretation gleichzeitig Modell von φ und von $\neg\varphi$ sein.
- Also können wir auch zeigen, dass keines der Modelle von Φ ein Modell von $\neg\varphi$ sein kann, bzw. dass es kein Modell von $\Phi \cup \{\neg\varphi\}$ gibt.

Resolutionsbeweise: Vorbereitung

➤ Literale

- ⇒ Atomares Satzsymbol oder dessen Negation
- ⇒ sowohl A als auch $\neg A$ sind Literale

➤ Konjunktive Normalform

- ⇒ $\varphi = (C_1 \wedge \dots \wedge C_n)$ Konjunktion von Klauseln
- ⇒ $C_i = (L_{i1} \vee \dots \vee L_{ik})$ Disjunktion von Literalen

➤ Mengenrepräsentation: *Klauselform*

- ⇒ $\varphi = \{C_1, \dots, C_n\}$ Mengen von Klauseln
- ⇒ $C_i = \{L_{i1}, \dots, L_{ik}\}$ Mengen von Literalen

➤ Vorteil der Mengenrepräsentation:

- ⇒ Reihenfolge und mehrfaches Vorkommen spielt keine Rolle

➤ Beispiel: Die beiden Formeln ψ und φ haben die gleiche Mengenrepräsentation:

- ⇒ $\psi = ((A \vee \neg B) \wedge (C \vee C))$
- ⇒ $\varphi = (C \wedge (\neg B \vee A))$

Resolution

➤ **Resolvente:**

Seien C_i und C_k Klauseln, die ein komplementäres Literal L (bzw. L') enthalten, d.h.,

$$\Rightarrow L \in C_i \text{ und } L' \in C_k$$

dann heißt die Menge $R = (C_i \setminus \{L\}) \cup (C_k \setminus \{L'\})$ **Resolvente** der Klauseln C_i und C_k

➤ **Beispiel:** $\{a, \neg b\}$ ist eine Resolvente von $\{a, \neg c\}$ und $\{\neg b, c\}$

➤ **Struktur eines Resolutionsbeweises:**

Sei Φ eine Formelmenge und ψ die Formel, für die zu prüfen ist, ob gilt: $\Phi \models \psi$

Dann:

1. $\varphi = \Phi \cup \{\neg\psi\}$

2. bestimme Mengenrepräsentation der Klauselform von φ

3. Leite solange Resolventen ab (Mehrfachverwendung möglich), bis entweder

➤ eine leere Resolvente entsteht (dann ist Widerspruch erreicht)

➤ oder keine Resolvente mehr ableitbar ist.

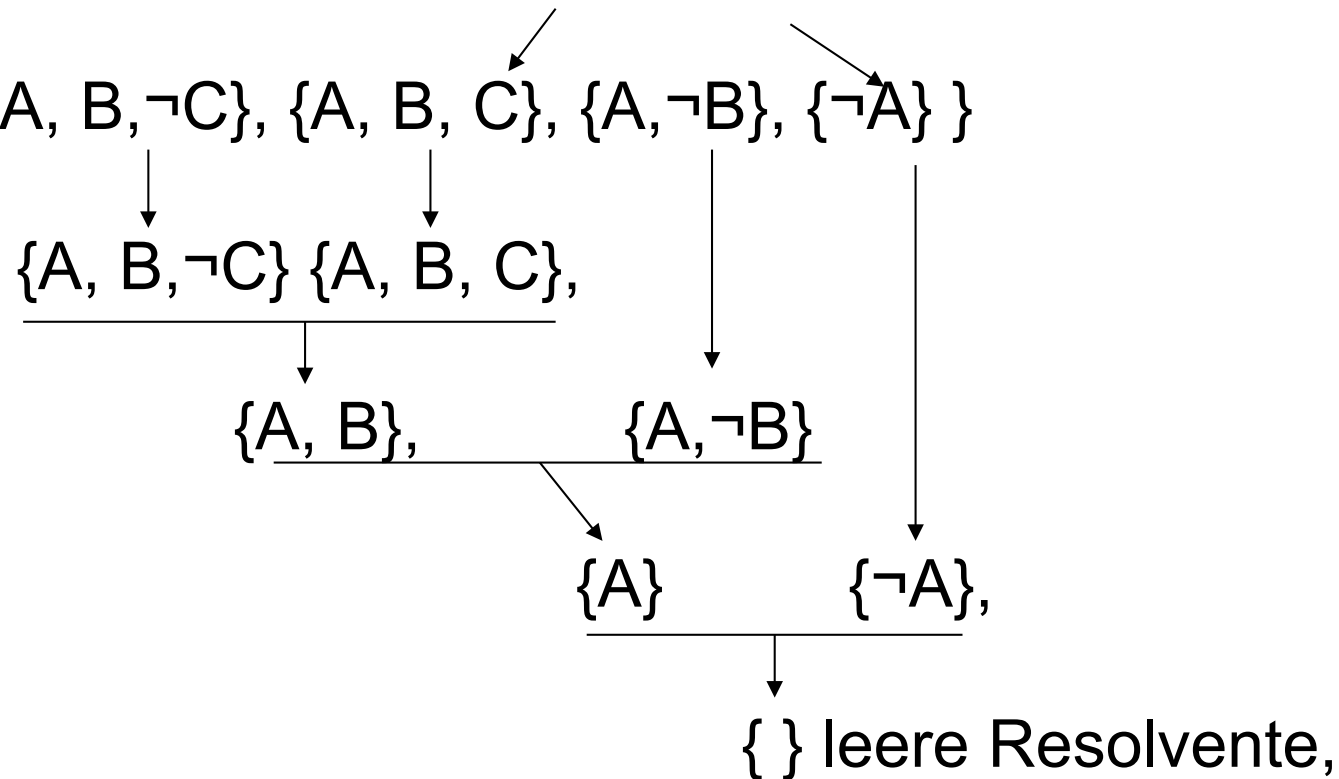
Resolutionsbeweis

Beispiel: Sei $\Phi = \{(A \vee B \vee \neg C), (A \vee B \vee C), (A \vee \neg B)\}$

➤ Frage: Gilt $\Phi \models A$?

➤ Betrachte Formelmengenge $\Phi \cup \{\neg A\}$

➤ $\varphi = \{ \{A, B, \neg C\}, \{A, B, C\}, \{A, \neg B\}, \{\neg A\} \}$



Also: $\Phi \models A$ gilt.

Resolutionskalkül

➤ Aufgabe:

⇒ Um die Resolutionsregel anwenden zu können, müssen die Formeln der WB sowie die zu beweisende Formel in konjunktive Normalform (Klauselnormalform) transformiert werden.

➤ Transformation in Klauselnormalform:

1. Ersetze **Äquivalenz** ($A \Leftrightarrow B$) durch $(A \Rightarrow B) \wedge (B \Rightarrow A)$
2. Ersetze **Implikation** ($A \Rightarrow B$) durch $(\neg A \vee B)$
3. Verschiebe **Negation** an die Satzsymbole, d.h., ersetze z.B. $\neg(A \wedge B)$ durch $(\neg A \vee \neg B)$
4. Eliminiere **Doppelnegationen**: d.h. ersetze $\neg(\neg A)$ durch A
5. „**Ausmultiplizieren**“, d.h. wende Distributivgesetze an bis Klauselform erreicht ist.

Beispiel: $(C \vee (\neg A \wedge B))$ wird ersetzt durch:

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

- (6. **Vereinfachen**: d.h. behalte von doppelten Klauseln nur eine, behalte von doppelten Literalen nur eines, eliminiere Klauseln mit A und $\neg A$)

Eigenschaften des Resolutionskalküls

➤ Korrektheit

⇒ Die Resolutionsregel ist korrekt: Ist Φ eine Menge von aussagenlogischen Formeln und ψ eine Resolvente zweier Formeln aus Φ , so folgt ψ semantisch aus Φ .

➤ Frage: Ist der Resolutionskalkül vollständig?

⇒ Vollständigkeit würde bedeuten, dass jede aus einer Formelmenge Φ folgerbare Formel ψ mit der Resolutionsregel ableitbar ist. Das ist jedoch nicht der Fall:

⇒ Z.B. lässt sich aus $\Phi = \{A, B\}$ und der Resolutionsregel nicht die Formel $(A \vee B)$ ableiten.

⇒ Man kann aber mit dem Resolutionskalkül feststellen, ob gilt:

$$\Phi = \{A, B\} \models (A \vee B)$$

➤ Widerlegungsvollständigkeit:

⇒ Der RK ist widerlegungsvollständig. D.h., ist die zu untersuchende Formelmenge widersprüchlich, so findet man den Widerspruch mit einer endlichen Anzahl von Resolutionsschritten.

Eigenschaften der Aussagenlogik

➤ Kritik:

- ⇒ Atomare Bausteine für logische Sätze sind Propositionen, die keine „innere Struktur“ haben.
- ⇒ Es können keine Aussagen über Objekte der Welt und deren allgemeine Zusammenhänge gemacht werden.
- ⇒ Mangelnde Ausdrucksmächtigkeit.

➤ **Motivation.** Erweitere die Ausdrucksmächtigkeit der Aussagenlogik durch die Möglichkeit, konkret und allgemein über Objekte zu sprechen.

➤ Deshalb:

- ⇒ Prädikatenlogik 1. Stufe (PL1)
- ⇒ Prolog

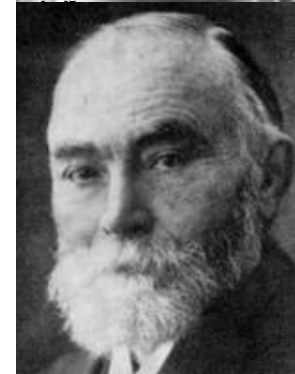
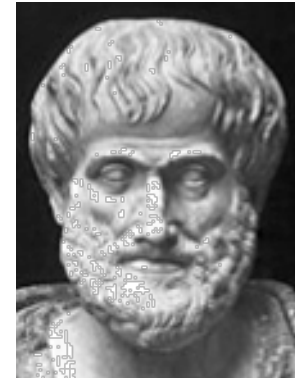
Prädikatenlogik 1. Stufe (PL1, FOL)

auch: *first order logic* (FOL)

- erste Ansätze bei Aristoteles
(Syllogismen, s. *Analytica Posteriora*)

- J. G. Frege (1848 – 1925):
Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens (1879)

- C. S. Peirce (1839 – 1914)
(Einführung der heute gebräuchlichen Notation)



Prädikatenlogik 1. Stufe (PL1, FOL)

Ziel:

- Prädikatenlogik 1. Stufe beschreibt Objekte, deren Eigenschaften und deren Beziehungen zueinander, wobei die Beziehungen auch funktionaler Art sein können.
- Insbesondere möchte man in der Lage sein, die Gültigkeit von Formeln formal beweisen bzw. widerlegen zu können.
- Analog zur Aussagenlogik: Unterscheidung zwischen Syntax und Semantik.

Syntax: Lexikalischer Teil:

- Konstanten-Symbole: A, B, Hans, Paul, ...
- Variablen-Symbole: x, y, ...
- Funktions-Symbole: plus, minus, mul, ...
- Prädikats-Symbole: student, hat_Computer
- logische Verknüpfungen: \wedge \vee \neg \Leftrightarrow \Rightarrow
- Quantoren \exists , \forall
- Gleichheit “=”

Prädikatenlogik

Syntax, struktureller Teil:

- Terme:
 - Konstanten-Symbole sind Terme
 - Variablen-Symbole sind Terme
 - falls f ein Funktionssymbol der Stelligkeit n ist und t_1, t_2, \dots, t_n Terme, dann ist auch $f(t_1, t_2, \dots, t_n)$ ein Term.
- Atomare Formel:
 - falls P ein Prädikatensymbol der Stelligkeit n ist und t_1, t_2, \dots, t_n Terme, dann ist $P(t_1, t_2, \dots, t_n)$ eine atomare Formel
 - wenn t_1, t_2 Terme sind, dann ist $t_1 = t_2$ eine atomare Formel
- (komplexe) Formel
 - jede atomare Formel ist eine Formel
 - wenn R und S Formeln sind und x ein Variablen-Symbol, dann sind auch folgende Ausdrücke Formeln:
 - $\neg R, R \wedge S, R \vee S, R \Rightarrow S, R \Leftrightarrow S, \exists x: R, \forall x: R$

Prädikatenlogik

Durch Quantoren gebundene Variablen

- Ein Vorkommen einer Variable x in der Formel S ist *gebunden* wenn es in einer Teilformel vom Typ $\exists x: R$, oder $\forall x: R$ auftritt

Freie Variable

- Tritt in einer Formel S eine Variable nicht gebunden auf, so heißt dieses Vorkommen *frei*.

Prädikatenlogischer Satz:

- Eine Formel ohne Vorkommen von freien Variablen ist ein *Satz*.

Anmerkung:

- Meist beschränkt man sich auf die Betrachtung von Sätzen.
(also nur quantifizierte Variablen)
- Falls man nur Sätze mit allquantifizierten Variablen vorliegen hat, und dies aus dem Kontext heraus klar ist, schreibt man der Einfachheit halber die Allquantoren nicht explizit hin.

Semantik der Prädikatenlogik

Ansatz:

- Verwendung einer modelltheoretischen Semantik ähnlich zur Aussagenlogik, jedoch etwas komplizierter.
- Die Wahrheit einer Formel wird bestimmt in Bezug auf eine Struktur, wobei eine Struktur dargestellt wird als ein Paar $S=(U,I)$ bestehend aus
 - **Universum U** (U ist der betrachtete Gegenstandsbereich)
 - **Interpretationsabbildung I** , die die Komponenten einer Formel auf Objekte und Beziehungen im Gegenstandsbereich abbildet:
 - Konstantensymbole → Objekte
 - Prädikatensymbole → Beziehungen zwischen Objekten
 - Funktionssymbole → funktionale Beziehungen zwischen Objekten
- Eine Interpretation I ordnet jeder Formel ϕ einen Wahrheitswert zu.
 Man sagt:
 I erfüllt ϕ (oder I ist Modell von ϕ , macht ϕ wahr) gdw. $I(\phi)=1$ (wahr)

Formulierung von Sachverhalten in Prädikatenlogik

Beispiele:

- „Alle Kinder lieben Eiscreme“

$$\forall x : \text{Kind}(x) \Rightarrow \text{liebt_Eiscreme}(x)$$

- „Es gibt einen Baum der Nadeln hat“

$$\exists x : \text{Baum}(x) \wedge \text{hat_Nadeln}(x)$$

- „Die Mutter einer Person ist dessen weibliches Elternteil“

$$\forall x \forall y: \text{Mutter}(x,y) \Leftrightarrow (\text{weiblich}(x) \wedge \text{Elternteil}(x,y))$$

- „Ein Cousin ist das Kind einer der Geschwister eines Elternteils“

$$\forall x \forall y: (\text{Cousin}(x,y) \Leftrightarrow \exists v \exists w: \text{Elternteil}(v,x) \wedge \text{Elternteil}(w,y) \wedge \text{Geschwister}(v,w))$$

- „Die Relation ‚Geschwister‘ ist symmetrisch“

$$\forall x \forall y: \text{Geschwister}(x,y) \Rightarrow \text{Geschwister}(y,x)$$

Eigenschaften von Quantoren

Kommutativität:

- $\forall x \forall y : F$ ist semantisch äquivalent zu $\forall y \forall x : F$
- $\exists x \exists y : F$ ist semantisch äquivalent zu $\exists y \exists x : F$

Aber:

$\exists x \forall y : F$ ist NICHT semantisch äquivalent zu $\forall y \exists x : F$

Beispiel:

$\exists x \forall y : \text{liebt}(x,y)$

“Es gibt jemanden, der alle in der Welt liebt”

$\forall y \exists x : \text{liebt}(x,y)$

“Jeder in der Welt wird von mindestens einer Person geliebt”

Umformbarkeit:

- All- und Existenzquantor können durch den jeweils anderen ausgedrückt werden. Z.B.:

$\forall x : F$ ist semantisch äquivalent zu $\neg(\exists x : \neg F)$

Wissensverarbeitung mit Prädikatenlogik (PL1)

- Prädikatenlogik ist insbesondere wegen der Möglichkeit zur Quantifizierung ausdrucksstärker als Aussagenlogik
 - Vorteil zur Formalisierung von Fakten und Abhängigkeiten eines zu modellierenden Gegenstandsbereichs.
- Wir suchen nun ein Beweisverfahren für die Prädikatenlogik erster Stufe.
- Die Resolutionsregel lässt sich auch auf PL1 übertragen.
- Der Resolutionskalkül lässt sich so erweitern, dass man mit ihm auch Widerspruchsbeweise über prädikatenlogische Formeln führen kann.
- Erweiterungen betreffen die Behandlung quantifizierter Variablen
 - Formeln müssen in bestimmte Form gebracht werden.

Resolutionskalkül für PL1

Aufgabe:

Um die Resolutionsregel anwenden zu können, müssen die Formeln der WB sowie die zu beweisende Formel in Klauselnormalform transformiert werden.

Behandlung von Variablen und Quantoren:

1. Eindeutige Benennung quantifizierter Variablen. Falls eine Formel ψ gleich benannte Variablen enthält, die jedoch an unterschiedliche Quantoren gebunden sind, benenne diese um.
2. Eliminiere alle freien Variablen dadurch, dass sie durch einen Existenzquantor gebunden werden. z.B. x frei in ψ , dann ersetze durch $\exists x: \psi$
3. Überführung in Pränexform durch Herausziehen aller Quantoren. D.h. bringe ψ in die Form $Q_1 x_1 Q_2 x_2 \dots Q_n x_n : \psi'$, wobei ψ' keine Quantoren enthält.

Resolutionskalkül für PL1

Behandlung von Variablen und Quantoren:

4. Eliminiere Existenzquantoren durch Skolemisierung.

Idee: man kann in einer Formel ψ jedes Vorkommen einer existenzquantifizierten Variable y_i durch eine Funktion f_i ersetzen, wobei f_i eine Funktion ist, die in Abhängigkeit aller in vorkommenden allquantifizierter Variablen x_k ein geeignetes y_i liefert.

Beispiel: $\forall x \exists y : \text{liebt}(x,y)$ „jeder liebt einen anderen“
wird durch die sog. „**Skolemisierung**“ ersetzt durch:

$\forall x : \text{liebt}(x, f(x))$

mit **neuer** Funktion $f(x)$, die zu x ein passendes y liefert.

5. Da nach Schritt 4 in Formel ψ nur noch Allquantoren vorkommen, können diese weggelassen werden.

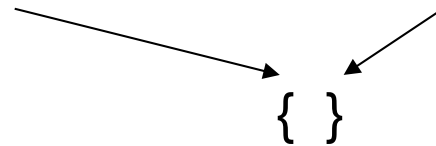
6. Überführen von ψ in Klauselnormalform analog zur Transformation aussagenlogischer Formeln.

Resolutionenkalkül für PL1

Was noch fehlt:

Nach der Umformung gemäß der Schritte 1-6 liegen alle Formeln in Klauselnormalform vor. Die Literale einer Klausel können jedoch sowohl Variablen als auch Konstanten enthalten.

Beispiel: $\{ \neg \text{bruder}(x, \text{Hans}) \}$ $\{ \text{bruder}(\text{Klaus}, \text{Hans}) \}$



In einem Resolutionsschritt muss es möglich sein, für Variablen bestimmte Belegungen einzusetzen. Im Beispiel: für x die Belegung „Klaus“.

Unifikation von Termen (Literalen):

Gegeben: zwei Literale L_1 und L_2 .

Gesucht: eine Variablensubstitution σ der Art, dass nach deren Anwendung auf L_1 und L_2 gilt: $L_1\sigma = L_2\sigma$

Falls solch ein σ existiert, heißt σ **Unifikator** von L_1 und L_2 .

Algorithmus zur Unifikation von Literalen

Gegeben: zwei Literale L_1 und L_2 .

Gesucht: Unifikator σ , der L_1 und L_2 „gleich macht“, d.h., $L_1\sigma = L_2\sigma$

1. Sind L_1 und L_2 **Konstanten**, so sind sie unifiziert, falls sie den gleichen Namen tragen. D.h.: Konstante L_1 ist nur mit L_1 unifizierbar
2. Ist L_1 eine **Variable** und L_2 ein **beliebiger Term**, so sind L_1 und L_2 unifiziert, gdw. für Variable L_1 der Term L_2 eingesetzt wird und x nicht in L_2 vorkommt.

Beispiel: x unifizierbar mit *Hans*, (ersetze x durch *Hans*).

3. Sind L_1 und L_2 **Prädikate** oder **Funktionen** der Form:

$PL_1(s_1, \dots, s_m)$ und $PL_2(t_1, \dots, t_n)$ so lassen sie sich nur unifizieren falls:

- i) die Prädikats- bzw. Funktionsnamen gleich sind, d.h. $PL_1 = PL_2$
- ii) $n = m$ gilt und sich jeder Term s_i mit dem entsprechenden Term t_i unifizieren läßt.

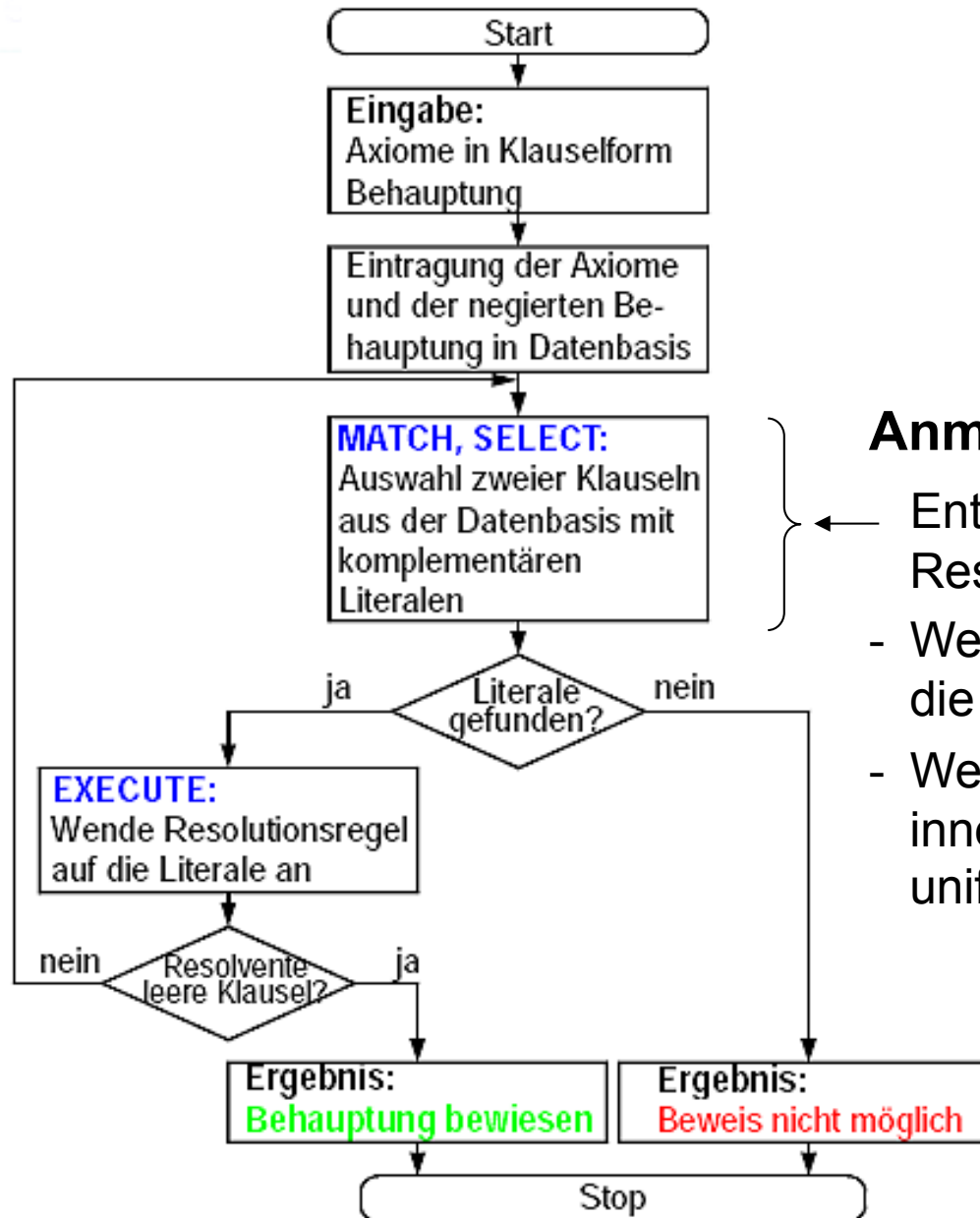
Beispiel: $liebt(x, y)$ unifizierbar mit $liebt(Hans, Anne)$
falls man x durch Hans und y durch Anne ersetzt.

Beispiel: $liebt(Hans, x)$ ist **nicht unifizierbar** mit $verheiratet(Hans, x)$

Unifikation

L1	L2	σ
$p(x,x)$	$p(a,a)$	$[x/a]$
$p(x,x)$	$p(a,b)$	fail (Unifikation nicht möglich)
$p(x,y)$	$p(a,b)$	$[x/a, y/b]$
$p(x,y)$	$p(a,a)$	$[x/a, y/a]$
$p(f(x),b)$	$p(f(c),z)$	$[x/c, z/b]$
$p(x,f(x))$	$p(y,z)$	$[x/y, z/f(y)]$ (Reihenfolge! $[y/x, z/f(x)]$ beachten!)
$p(x,f(x))$	$p(y,y)$	fail

Ablauf eines Resolutionsbeweises



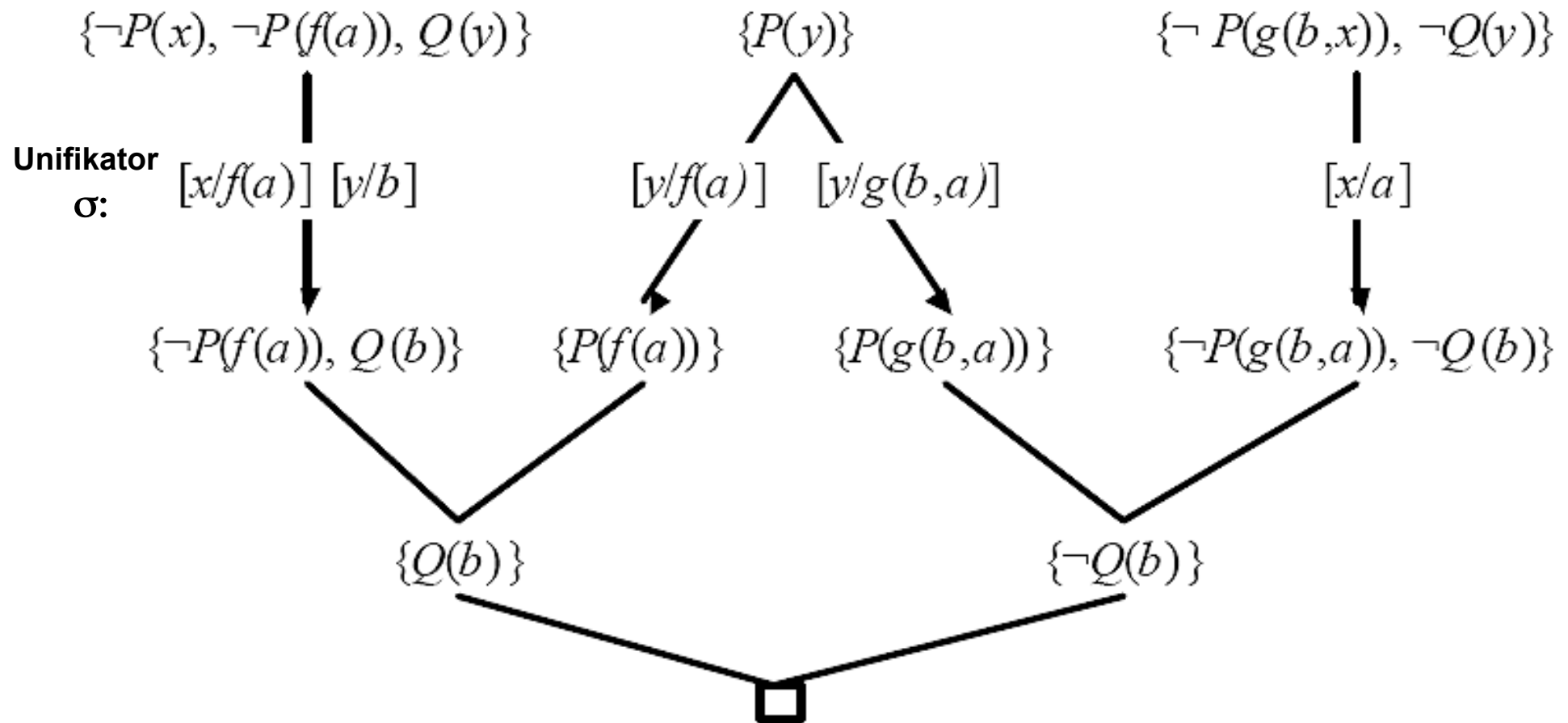
Anmerkung:

← Entscheidungspunkte bei der Resolution:

- Welche Klauseln wählt man für die Resolution?
- Welche beiden Literale innerhalb der Klauseln werden unifiziert?

Resolutionsbeweis für PL1 Formel

$$\forall x \forall y ((\neg P(x) \vee \neg P(f(a)) \vee Q(y)) \wedge P(y) \wedge (\neg P(g(b,x)) \vee \neg Q(y)))$$



Eigenschaften der Resolution für PL1 Formeln

Widerlegungsvollständigkeit:

- Sofern man Resolution auf eine widersprüchliche Klauselmeng anwendet, so existiert eine endliche Folge von Resolutionsschritten, mit denen man den Widerspruch aufdecken kann

Aber: die Anzahl n der Beweisschritte kann sehr groß werden und damit das Verfahren unpraktikabel machen.

Kein Entscheidungsverfahren:

- Terminierung nicht garantiert, falls Klauselmeng nicht widersprüchlich.

Maßnahmen zur Effizienzsteigerung

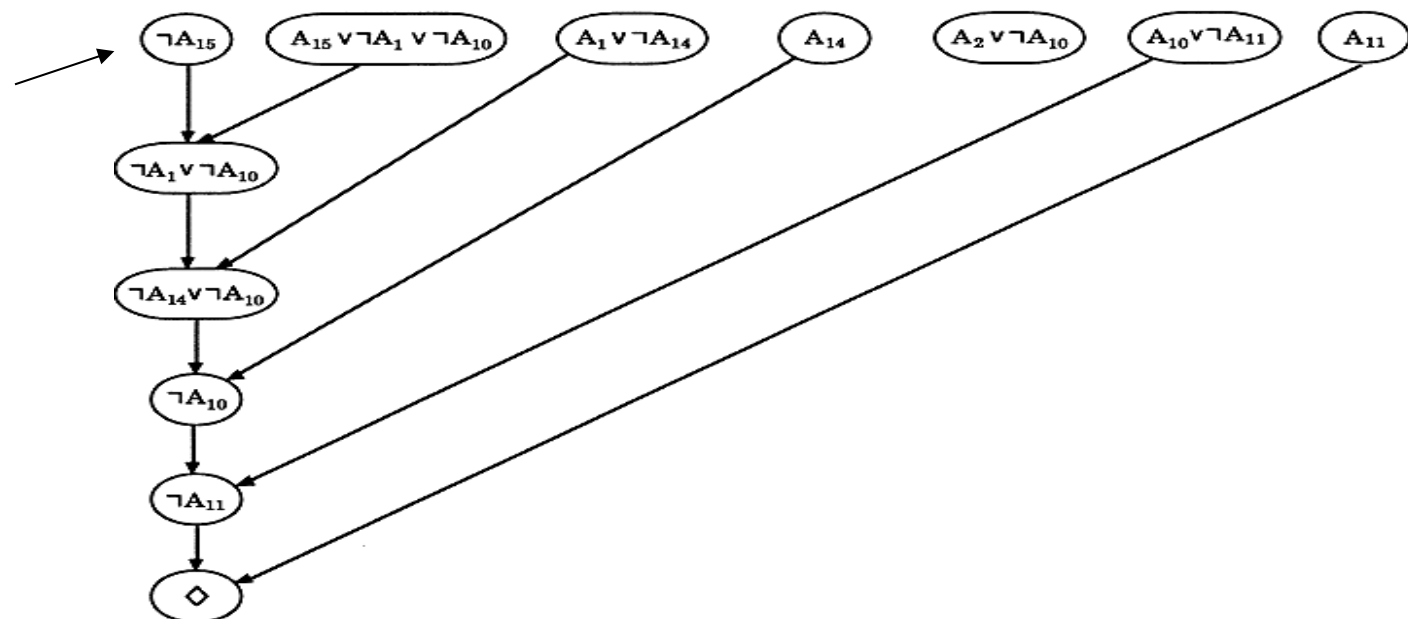
Ansatz 1: Entwicklung von Heuristiken zur Klauselauswahl

- Idee: man betrachtet die Klauselauswahl als Suchproblem und setzt „intelligente“ Suchverfahren ein.

Beispiele für mögliche Strategien:

- bevorzuge möglichst kurze Klauseln
- entferne Klauseln, die von anderen „subsumiert“ werden.
- **Set-of-Support-Strategie:** Die Resolution wird stets auf eine der Klauseln gerichtet, die von der negierten Behauptung abstammen.

negierte
Behauptung



Maßnahmen zur Effizienzsteigerung

Ansatz 2: Einschränkung der Ausdrucksmächtigkeit

- Idee: Für viele Anwendungen benötigt man eine Ausdrucksmächtigkeit, die zwischen Aussagenlogik und PL1 liegt.
- Einschränkung auf eine weniger ausdrucks mächtige Sprache, z.B.:
 - nur allquantifizierte Variablen
 - keine Disjunktion in Termen
- **In der Praxis:**
 - Einschränkung auf praktisch wichtige Spezialform, die sog. **Hornklauseln** (benannt nach dem Logiker Alfred Horn).

Hornklauseln

- **Hornklauseln** sind Klauseln *mit höchstens einem positiven* Literal
- Es gibt drei Arten von Hornklauseln:
 - i) **Regel:** $P \leftarrow P_1 \wedge P_2 \wedge \dots \wedge P_n$
 ist äquivalent zu $\{P, \neg P_1, \neg P_2, \dots, \neg P_n\}$
 - in Hornklauseln steht das positive Literal immer vorne
 - das positive Literal P bezeichnet man als **Konklusion**
 - die negativen Literale $\neg P_1, \neg P_2, \dots, \neg P_n$ heißen **Prämissen**
 - das Zeichen \leftarrow ist angelehnt an das Zeichen \Rightarrow (log. Implikation)
 - ii) **Fakt:** P ist äquivalent zu $\{P\}$
 - iii) **Zielklausel:** $\leftarrow P_1 \wedge P_2 \wedge \dots \wedge P_n$ ist äquivalent zu $\{\neg P_1, \neg P_2, \dots, \neg P_n\}$
 auch: Integritätsbedingung (integrity constraint)
 - kein positives Literal

Hornklauseln

Eine typische Hornklausel-Wissensbasis:

```

man(X) ← human(X) ∧ male(X)
woman(X) ← human(X) ∧ female(X)
parent(X,Y) ← mother(X,Y)
parent(X,Y) ← father(X,Y)
ancestor(X,Y) ← parent(X,Y)
ancestor(X,Y) ← parent(X,Z) ∧ ancestor(Z,Y)

human(john)
human(paul)
human(mary)
male(john)
male(paul)
female(mary)
father(john,mary)
mother(mary,paul)
  
```

Regeln
(Regelbasis)

Fakten
(Faktenbasis)

Hornklauseln

Beispiele für mögliche Anfragen:

human(john)

Resultat: yes ;;; Wahrheitswert

female(john)

Resultat: unknown

female(peter)

Resultat: unknown

human(X)

Resultat: X = john oder ;;; Belegungen, für
 X = paul oder ;;; die Anfrage wahr
 X = mary ;;; wird

man(john)

Resultat: yes

man(X)

Resultat: X = john oder
 X = paul

parent(mary, X)

Resultat: X = paul

ancestor(X, Y) \wedge male(X)

Resultat: X = john, Y = mary oder
 X = john, Y = paul

Beantwortung von Anfragen

Gegeben:

- Wissensbasis mit Regeln, Fakten, Integritätsbedingungen

Anfrage:

- Zu beweisen ist die Gültigkeit einer Aussage: $P_1 \wedge P_2 \wedge \dots \wedge P_n$

Ansatz:

- Mit Widerspruchsbeweis (Resolution), wobei man von der negierten Aussage ausgeht:
 - $\neg(P_1 \wedge P_2 \wedge \dots \wedge P_n)$ ist äquivalent zu Zielklausel
 - $\neg P_1 \vee \neg P_2 \vee \dots \vee \neg P_n$ (in Hornklauselschreibweise: $\leftarrow P_1 \wedge P_2 \wedge \dots \wedge P_n$)
- Der Widerspruch ist dann aufgedeckt, wenn man die „leere Klausel“ \leftarrow ableiten kann.

Beispiel: Beantwortung von Anfragen

Gegeben:

- Wissensbasis mit:
 - i) einer Regel: **sterblich(X) ← mensch(X)**
 (Mengenschreibweise: { **sterblich(X)** , **¬mensch(X)** }
 - ii) einem Faktum: **mensch(sokrates) ←**
 (Mengenschreibweise: { **mensch(sokrates)** }

Anfrage:

- Gilt, dass Sokrates sterblich ist? Also: folgt **sterblich(sokrates)** aus der Wissensbasis?
- Also füge hinzu **← sterblich(sokrates)**
 (Mengenschreibweise: { **¬ sterblich(sokrates)** }
- Führe Widerspruchsbeweis., d.h. leite aus der Klauselmenge die leere Klausel ab:

$$\left. \begin{array}{l} \{ \mathbf{sterblich(X)}, \mathbf{\neg mensch(X)} \}, \\ \{ \mathbf{mensch(X)} \}, \\ \{ \mathbf{\neg sterblich(sokrates)} \} \end{array} \right\} \models \square \quad (\text{als Hornklausel: } _ \leftarrow _)$$

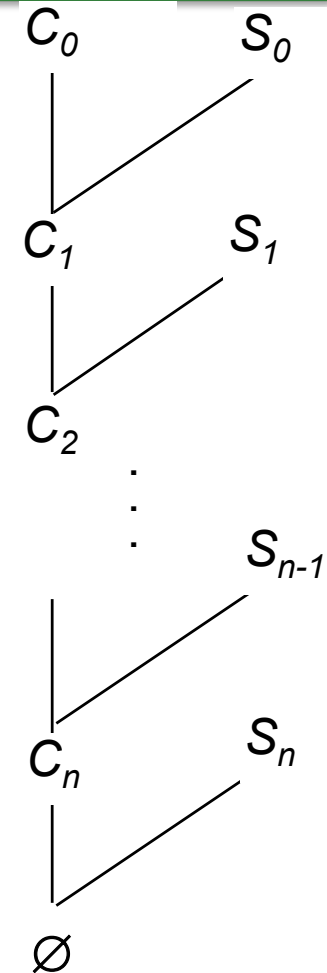
Inferenzregel für Hornklauseln

➤ Inferenzregel (OLD-Resolution)

$$\begin{array}{l}
 H_1 \text{ mit } H \\
 \text{unifizierbar:} \\
 H_1 \sigma = H \sigma
 \end{array}
 \quad
 \frac{\{\neg H_1, \neg H_2, \dots, \neg H_m\}, \{H, \neg B_1, \neg B_2, \dots, \neg B_n\}}{\{\neg H_2, \dots, \neg H_m, \neg B_1, \neg B_2, \dots, \neg B_n\} \sigma}$$

➤ Ordered-Linear Resolution for Definite Clauses

- starte mit negierter Anfrage als einer Klausel C_0
- Lineare Resolution: nimm im nächsten Resolutionsschritt die Resolvente C_i als eine der beiden Klauseln (vgl. Set-of-Support-Strategie) und resolviere mit geeigneter Seitenklausel S_i
- Literale einer Klausel sind geordnet, resolviere mit dem ersten Literal.
- in jedem Schritt wird ein negatives Literal abgearbeitet. Stopp bei leerer Resolvente



C_i : Center Clauses
(Zentrumsklauseln)
 S_i : Side Clauses
(Seitenklauseln)

OLD-Resolution

- Eine Regel ist anwendbar, wenn die *Konklusion* der Regel mit der Anfrage unifiziert werden kann:
 - Variablen werden mit Konstanten belegt
 - eine Variable muss bei jedem Vorkommen den gleichen Wert haben.
- Enthält die Anfrage Variablen, so werden gültige Werte der Variablen berechnet.
- Ein Literal der Anfrage wird beantwortet (bewiesen), wenn ein Fakt in der Datenbasis existiert, der mit der Anfrage unifiziert werden kann.
- Falls für ein Literal der Anfrage keine anwendbare Regel oder Fakt gefunden wird, ist das Literal nicht beweisbar (vgl. Closed-World Assumption)
 - Rücksetzen (Backtracking) und mit Fortsetzen mit alternativer Belegung für die zu unifizierenden Variablen.
- Die OLD-Resolution nennt man auch Rückwärtsverkettung, da die Regeln rückwärts angewendet werden.
- Die Rückwärtsverkettung ist abgeschlossen, wenn die Liste der Anfragen leer ist. Das Ergebnis ist die Menge der Variablen mit ihren Werten (die Substitution).

Zusammenfassung: Hornklauseln

Pro

- Mit Hornklauseln lassen sich Wissensbasen aufbauen, in denen Wissen unterteilt wird in
 - Fakten (zur Repräsentation von Objekten und Sachverhalten)
 - Regeln und Integritätsbedingungen (zur Repräsentation von Beziehungen, Abhängigkeiten, Kausalitäten, Strukturbeziehungen, usw.).
 - Verwendung von Variablen in Regeln (und Fakten) erlaubt die Abstraktion von konkreten Instanzen (Vorteil gegenüber der Aussagenlogik).
- Anfragen an die Wissensbasis werden als spezielle Klauseln formuliert ($\text{:- } P$) und mittels eines linearen Resolutionsbeweises beantwortet. (Günstig für eine Automatisierung).
- Der Hornklauselkalkül ist widerlegungsvollständig

Contra

- Weniger ausdrückstark als Prädikatenlogik 1. Stufe
 - keine alternativen Konklusionen der Form $A \vee B \leftarrow C$
 - keine negierten Prämissen (bzw. Anfragen der Form: $\leftarrow \neg C$)

Anwendung der Hornklausel-Logik

Bedeutung der Hornklausellogik:

- Grundlage für Sprachen zur logischen Programmierung (insbesondere Prolog)
- Grundlage deduktiver Datenbanken

PROLOG: PROgramming in LOGic

- basiert auf Hornklauseln und (linearem) Resolutionsbeweis:
 (Genauer: SLD-Resolution:
 Linear resolution with **S**election function for **D**efinite clauses)
- Existenzquantor entfällt, Allquantor wird implizit vorausgesetzt.
- Terme als Datenstrukturen
 - Variablen
 - Konstante
 - Funktionssymbol mit Argumentliste (wiederum Terme)

Zusammenfassung: Prolog Programmierung

Merkmale:

- Wissen (= Regeln und Fakten) werden als Hornklauseln notiert
- Tiefensuche beim Beweisen führt dazu, dass die Reihenfolge der Klauseln die Antwort beeinflusst (Extremfall: Rekursion).
 - Wissen und Programm-Kontrollfluss sind miteinander verwoben.
- Prologumgebungen erlauben interaktives Arbeiten, d.h. man stellt im Dialogmodus Anfragen an eine WB und kann diese auch interaktiv modifizieren (mit assert bzw. retract).
- Kommerzielle Prolog-Systeme bieten meist einen erweiterten Sprachumfang, der deutlich über den Hornkalkül hinaus geht
 - z.B. Konstrukte zur Programmierung von Schleifen
 - Prozess-Verwaltung

Prolog Programmiersysteme:

- GNU Prolog <http://gprolog.inria.fr>
- Quintus Prolog (kommerziell) www.sics.se/quintus
- ifProlog (kommerziell) www.ifcomputer.de
- ...

Anmerkung zur Logik Programmierung

Fifth-Generation-Project

- 1982 legte das Japanische Wirtschaftsministerium (MITI) ein Großprojekt auf, mit dem Ziel, innerhalb von 10 Jahren einen hochgradig parallel arbeitenden Rechner zu bauen. Als Grundlage für Betriebssystem und Programmierung setzte man auf „Programmierung in Logik“, da man hoffte, in Logik formulierte Probleme besonders gut parallel bearbeiten zu können (Idee: paralleles Beweisen von Klauseln).
- Aus diesem Projekt gingen einige Rechnerprototypen hervor, u.a.:
 - PSI-Machine (Prolog Sequential Machine)
 - Parallel-Rechner:
 - MultiPsi
 - PIMx Serie
 - <http://www.icot.or.jp/ARCHIVE>
- Aus heutiger Sicht gilt das Projekt als „Mega-Flop“, da keine der Maschinen je zur Anwendung kam (geschweige denn in Serie ging).



- Logik ist eine sehr allgemeine Sprache zur Repräsentation von Wissen.
 - man kann viele Sachverhalte ausdrücken, hat jedoch nur sehr wenig Strukturierungsmittel zur Hand.
(nämlich Prädikate, Konstanten und Funktionen).
 - Klassen, strukturelle Beziehungen (is-a, instance-of, part-of) und Eigenschaften müssen durch Prädikate repräsentiert werden.
(Oft umständlich und wenig problemspezifisch).
- Grundsätzliche Probleme:
 - Unentscheidbarkeit der PL1.
 - Effizienz der Inferenzverfahren: Effiziente Verfahren erfordern Einschränkungen der Ausdrucksstärke (z.B. auf Hornklauseln).
- Als Wissensrepräsentationssprache spielt reine Logik-Programmierung heute eher eine untergeordnete Rolle. Dennoch, Mechanismen wie logische Inferenz und Unifikation kommen in vielen wissensbasierten Systemen zum Einsatz. (Z.B. Integration einer Prolog-Core-Engine).