7.6 Logisches Programmieren und Horn-Klauseln

Bei praktischen Anwendungen der Prädikatenlogik erster Ordnung wird häufig die logische (deklarative) Programmiersprache Prolog verwendet. Die folgenden Ausführungen skizzieren die Prinzipien dieser Sprache, ohne jedoch auf deren konkrete Syntax einzugehen.

Ein *logisches Programm* besteht aus einer Folge von Elementen folgender Form:

$(P1)$ $\quad \alpha$

oder

$(P2)$ $\quad \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \longrightarrow \beta, \;\; n \geq 1$

sowie einer Frage der Form

$(P3)$ $\quad ? \quad \gamma_1 \wedge \gamma_2 \wedge \ldots \wedge \gamma_m, \;\; m \geq 1$

wobei

- $\alpha, \alpha_1, \ldots, \alpha_n, \beta, \gamma_1, \ldots, \gamma_m$ sind unnegierte atomare Formeln sind.
- Alle Variablen in $(P1)$ und $(P2)$ sind universell quantifiziert.
- Alle Variablen in $(P3)$ sind existentiell quantifiziert.

Elemente der Form $(P1)$ und $(P2)$ entsprechen den Axiomen, $(P3)$ der zu beweisenden Aussage.

Beispiel: Logisches Programm

Problembeschreibung:

> Laut Gesetz ist es für einen Amerikaner ein Verbrechen, Waffen an feindliche Nationen zu verkaufen. Das Land Nono, ein Feind von Amerika, besitzt einige Raketen, und alle seine Raketen wurden ihm von Colonel West verkauft, der Amerikaner ist.

Zu beweisen: West ist ein Krimineller.

Umsetzung:

"...ist es für einen Amerikaner ein Verbrechen, Waffen an feindliche Nationen zu verkaufen":

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \rightarrow Criminal(x)$$

"Das Land Nono, ein Feind von Amerika":
$Enemy(Nono, America)$

"Nono ...besitzt einige Raketen": $\exists x Owns(Nono, x) \wedge Missile(x)$
$Owns(Nono, M_1)$
$Missile(M_1)$

"alle seine Raketen wurden ihm von Colonel West verkauft":
$Missile(x) \wedge Owns(Nono, x) \rightarrow Sells(West, x, Nono)$

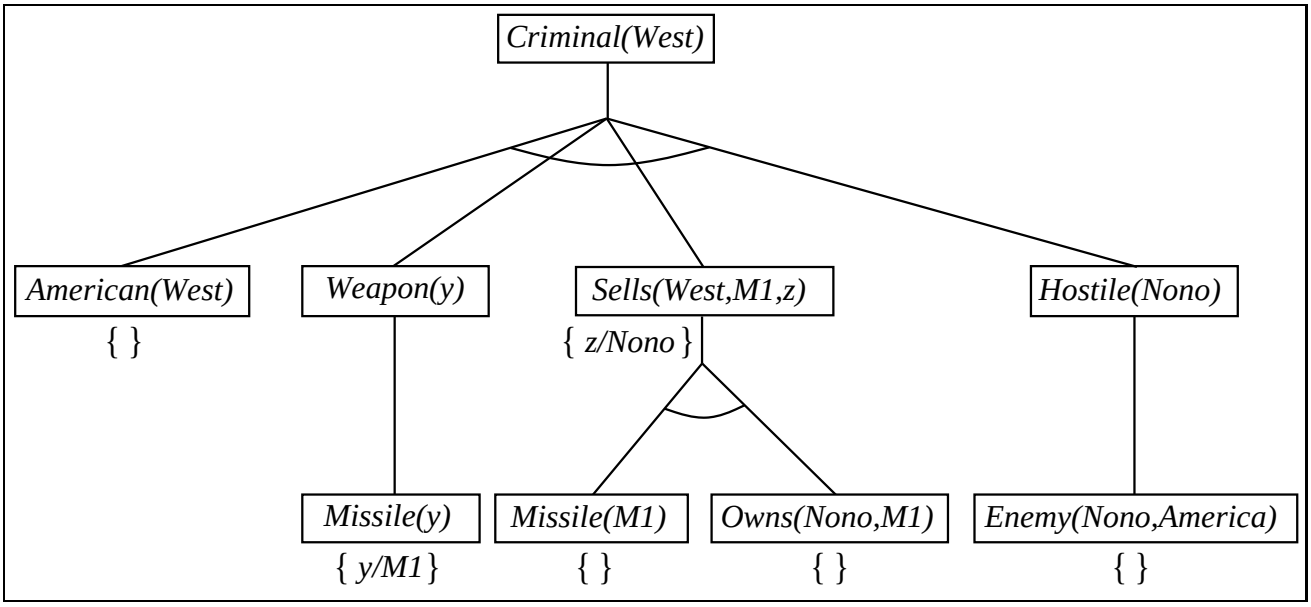
"West, der Amerikaner ist":
$American(West)$

Ferner müssen wir wissen, dass Raketen Waffen sind und ein Feind als "feindlich" gilt:
$Missile(x) \rightarrow Weapon(x)$
$Enemy(x, America) \rightarrow Hostile(x)$

Die Abarbeitung eines logischen Programms erfolgt durch einen **Interpreter**:

- $\gamma_1, \ldots, \gamma_m$ ist eine Liste von Zielen.
- Es wird sukzessive die Erfüllung von Zielen durch Elemente der Form $(P1)$ und $(P2)$ versucht; hierbei wird Unifikation angewendet.
- Bei Nicht-Erfüllbarkeit eines Zieles erfolgt Backtracking.

Beispiel: West ist ein Krimineller



Um $Criminal(West)$ beweisen zu können, müssen wie die vier Konjunkte darunter beweisen. Einige davon befinden sich in der Wissensbasis, andere müssen weiter rückwärts verkettet werden.
Nachdem ein Unterziel in einer Konjunktion erfolgreich war, wird seine Substitution auf die nachfolgenden Unterziele angewendet.

Beispiel: Logisches Programm

Problembeschreibung:

(1) Objekt A ist kleiner als Objekt B.

(2) Objekt A ist kleiner als Objekt C.

(3) Objekt A befindet sich hinter Objekt C.

(4) Ist ein Objekt x kleiner als ein Objekt y und wird x von y verdeckt, so ist x unsichtbar

(5) Befindet sich ein Objekt u hinter einem Objekt v, so ist u verdeckt von v

(6) Existiert ein unsichtbares Objekt?

Umsetzung:

(1) KLEINER(A,B)

(2) KLEINER(A,C)

(3) HINTER(A,C)

(4) KLEINER(x,y) $\wedge$ VERDECKT(x,y) $\rightarrow$ UNSICHTBAR(x)

(5) HINTER(u,v) $\rightarrow$ VERDECKT(u,v)

(6) ? UNSICHTBAR(z)

In diesem Fall ist ein Backtracking nötig.

Eine **Horn-Klausel** ist eine Klausel mit höchstens einem unnegierten Literal. Unnegierte Literale werden auch als *positive* und negierte als *negative* Literale bezeichnet.

Horn-Klauseln sind somit von der Gestalt

$(H1)$ $\quad \neg\alpha_1 \vee \ldots \vee \neg\alpha_n \vee \beta,\ n \geq 0$

oder

$(H2)$ $\quad \neg\gamma_1 \vee \ldots \vee \neg\gamma_m,\ m \geq 1$

wobei $\alpha_i, \beta, \gamma_j$ jeweils unnegierte Literale sind.

Beobachtung:

- für $n = 0$ ist $(H1)$ mit $(P1)$ äquivalent
- für $n \geq 1$ ist $(H1)$ mit $(P2)$ äquivalent
- für $m \geq 1$ ist $(H2)$ mit der Negation von $(P3)$ äquivalent

Folgerung:

- Horn-Klauseln können als Elemente eines logischen Programms aufgefasst werden und umgekehrt. (Hierbei wird bei der Frage immer die Negation betrachtet.)
- Bei der Umwandlung eines logischen Programms in Klausel-Form treten nur Horn-Klauseln auf.

Beschreibung des Interpreters durch die Resolutionsregel:

1. Programmelemente der Form $(P1)$ (Erfüllung eines Zieles)

$$\begin{array}{ll} (*) & \gamma_1 \wedge \ldots \wedge \gamma_m \qquad \text{(Ziele)} \\ (**) & \gamma_1 \qquad \text{(Fakt; Element } (P1)\text{)} \end{array}$$

führt zu neuen Zielen

$$(***) \quad \gamma_2 \wedge \ldots \wedge \gamma_m$$

Äquivalente Darstellung durch Anwendung der Resolutionsregel

$$\begin{array}{ll} (*) & \neg\gamma_1 \vee \ldots \vee \neg\gamma_m \qquad \text{(negierte Ziele)} \\ (**) & \gamma_1 \\ \hline (***) & \neg\gamma_2 \vee \ldots \vee \neg\gamma_m \qquad \text{(neue negierte Ziele)} \end{array}$$

2. Programmelemente der Form $(P2)$

$$\begin{array}{ll} (*) & \gamma_1 \wedge \ldots \wedge \gamma_m \qquad \text{(Ziele)} \\ (**) & \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \longrightarrow \gamma_1 \qquad \text{(Regel; Element } (P2)\text{)} \end{array}$$

führt zu neuen Zielen

$$(***) \quad \alpha_1 \wedge \ldots \wedge \alpha_n \wedge \gamma_2 \wedge \ldots \wedge \gamma_m$$

Äquivalente Darstellung durch Anwendung der Resolutionsregel

$$\begin{array}{ll} (*) & \neg\gamma_1 \vee \ldots \vee \neg\gamma_m \qquad \text{(negierte Ziele)} \\ (**) & \neg\alpha_1 \vee \neg\alpha_2 \vee \ldots \vee \neg\alpha_n \vee \gamma_1 \qquad \text{(Regel)} \\ \hline (***) & \neg\alpha_1 \vee \ldots \vee \neg\alpha_n \vee \neg\gamma_2 \vee \ldots \vee \neg\gamma_m \qquad \text{(neue negierte Ziele)} \end{array}$$

Folgerung:
Die Arbeitsweise des Interpreters beruht auf der Resolutionsregel

Wegen der Beschränkung auf höchstens ein positives Literal stellen Horn-Klauseln eine Untermenge von Klauseln dar. Für praktische Anwendungen sind Horn-Klauseln jedoch oft ausreichend. Aufgrund der eingeschränkten Form ist die Arbeitsweise eines Interpreters für Horn-Klauseln effizienter als diejenige eines Interpreters für allgemeine Klauseln.

Weitere Eigenschaften von Prolog:

- Standardfunktionen, Standardprädikate, auch für E/A
- Reihenfolge der Klauseln wichtig
- Reihenfolge der Literale innerhalb einer Klausel wichtig

Prolog ist die bei weitem gebräuchlichste Logikprogrammiersprache. Es wird hauptsächlich als Sprache für die schnelle Entwicklung von Prototypen verwendet, ebenso wie für Aufgaben der Symbolmanipulation. Viele wissensbasierte Systeme werden in Prolog geschrieben.