Logische Konsequenz

Ein **Modell für eine endliche Menge von Formeln** $\{F_1, F_2,...,F_n\}$ ist eine Interpretation für die die Formel $(F_1 \wedge F_2 \wedge,..., \wedge F_n\}$ wahr ist, d. h. wenn die Interpretation ein Modell für jede einzelne Formel ist.

Eine Formel F heißt **logische Konsequenz** (Folgerung) einer endlichen Menge von Formeln S, wenn jedes Modell für S auch ein Modell für die Formel F ist.
Schreibweise: $S \models F$

Eine Formel F ist genau dann eine logische Konsequenz der Formelmenge $\{F_1, F_2,...,F_n\}$, wenn $F_1 \wedge F_2 \wedge,..., \wedge F_n \Rightarrow F$ allgemeingültig ist.

Problem der Unerfüllbarkeit

Sei S eine endliche Menge von Formeln und F eine Formel. Dann ist F eine logische Konsequenz aus S genau dann, wenn die Menge $S \cup \{\neg F\}$ widerspruchsvoll ist.

D. h. wenn ein logisches Programm P und eine Anfrage A gegeben sind und P zusammen mit A unerfüllbar ist, ist die Negation der Anfrage A
$\exists X_1: \exists X_1:...: \exists X_1: A_1 \wedge A_2 \wedge ... \wedge A_n$ eine logische Konsequenz des Programms.

Das Hauptproblem in der logischen Programmierung ist somit der Nachweis der Unerfüllbarkeit einer Menge von Klauseln.

Herbrand-Interpretationen

Eine Herbrand-Interpretation einer Formelmenge S besteht aus:

- Herbrand-Universum U_s: Menge aller Grundterme (Terme ohne Variablen), die mit Konstanten und Funktoren gebildet werden können, die in den Formeln von S vorkommen.
- Jeder Konstanten wird die gleiche Konstante im Herbrand-Universum zugeordnet.
- Jedem n-stelligen Funktor f wird eine Funktion zugeordnet, die jeweils n Grundterme $t_1,...,t_n$ aus U_s auf das Element $f(t_1,...,t_n)$ aus U_s abbildet.
- *Zuordnung von Prädikaten zu entsprechenden Abbildungen*

Bsp. Herbrand-Interpretation

Klauseln $\Rightarrow p(X,0)$

$\Rightarrow p(X,X)$

$p(X,Y) \Rightarrow p(add1(X),Y)$

Eine Herbrand-Interpretation:

- $U = \{0, add1(0), add1(add1(0)), add1(add1(add1(0))), ...\}$
- der Konstanten 0 wird das Element 0 aus U zugeordnet.
- dem Funktor add1 wird die Funktion zugeordnet, die jedes Element T aus der Menge U auf $add1(T) \in U$ abbildet
- Dem Prädikat p wird folgende Funktion zugeordnet: Sind T1 und T2 Elemente aus U und kommt in T1 add1 nicht so oft vor wie in T2, dann wird das Paar (T1,T2) auf falsch abgebildet, sonst auf wahr.

Herbrand-Modelle

Herbrand-Interpretationen, die Modelle für die gegebenen Formeln sind, heißen Herbrand-Modelle.

Eine Menge S von Klauseln ist genau dann unerfüllbar, wenn sie kein Herbrand-Modell hat.

Substitution

Eine **Substitution** sub=$\{X_1/t_1, \ldots, X_n/t_n\}$ ist eine endliche Abbildung von Variablen X_i in Terme $t_{i.}$.

Eine Substitution angewendet auf einer endlichen Menge von Literalen, die zu einer nur noch aus einem Prädikat bestehenden Formel führt, nennt man **Unifikator**.

Die Anpassung der in einer Anfrage vorkommenden Literale an die in Fakten und Regeln vorkommenden Literale durch Ersetzen von Variablen nennt man **Unifikation**.

Korrekte Antwort

Sei P ein log. Programm und A eine Anfrage der Form

$A_1, A_2, \ldots, A_m \Rightarrow$

Dann heißt eine Substitution sub der in A vorkommenden Variablen **Antwort**.

Eine Antwort heißt **korrekt**, wenn die Formel

$\forall X_1: \ldots: \forall X_k: (A_1 \wedge \ldots \wedge A_m)$ sub

eine logische Konsequenz von P ist.

Bsp. Korrekte Antwort

Sei folgendes logisches Programm gegeben:

=> anhang([], E, •(E,[]))

anhang(R, E, RE) => anhang(•(K, R), E, •(K, RE))

und die Anfrage

anhang(•(a, []), b, X)

Dann ist die Antwort

sub = {X / •(a, •(b, []))}

korrekt, wenn die Substitution sub, angewendet auf die Anfrage, also die Formel

anhang(•(a, []), b, •(a, •(b, [])))

eine logische Konsequenz aus dem Programm ist.

Prozedurale Semantik

Zusammenhang von log. Konsequenz und zielgerichtetem Widerlegen:

$P \models \exists(F_1 \wedge \ldots \wedge F_n) \Leftrightarrow$

$P \cup \{\neg \exists(F_1 \wedge \ldots \wedge F_n)\}$ widerspruchsvoll $\Leftrightarrow$

$P \cup \{\forall(\neg F_1 \vee \ldots \vee \neg F_n)$ widerspruchsvoll $\Leftrightarrow$

$P \cup \{F_1 \wedge \ldots \wedge F_n \Rightarrow\}$ widerspruchsvoll $\Leftrightarrow$

$P \cup \{F_1 \wedge \ldots \wedge F_n \Rightarrow\} \models o$

Beweisschritte für eine Anfrage

- Literalauswahl
- Regelauswahl
- Unifikation
- Resolutionsschritt

Anfrage: $b_1, ..., b_k =>$

Programmklausel: $c_1, ... , c_m => a$

Resolvente:
$b_1, ... b_{i-1}, c_1, ... , c_m , b_{i+1} , ... , b_k =>$

Literal- und Regelauswahl

Literalauswahlnichtdeterminismus:
Eine Anfrage besteht im allgemeinen aus einer Konjunktion von Literalen, so dass verschieden Literale für den folgenden Resolutionsschritt gewählt werden können.

Regelauswahlnichtdeterminismus:
Ein Anfrageliteral kann mit den Köpfen mehrerer Regeln unifizierbar sein, so dass alternative Resolutionsschritte möglich sind.

Unifikation

Ein Unifikator sub_U, der möglichst viele Variablen beibehält, wird **allgemeinster Unifikator** genannt,
d. h. für jeden anderen Unifikator sub_U ' ex. eine Substitution sub '',
so dass sub ' = sub '' o sub_U.

J. A. Robinson (1965):
Entwicklung eines effizienten Unifikationsalgorithmus und der Nachweis, dass jede unifizierbare Literalmenge einen (bis auf Variablenumbenennung eindeutigen) allgemeinsten Unifikator besitzt

Unifikationsalgorithmus von Robinson

Eingabe: Literalmenge $L = \{L_1, \ldots, L_n\}$
Ausgabe: sub_U oder „nicht unifizierbar“

$sub_U := \emptyset$;
while $|L\, sub_U| > 1$ **do** *% solange mehr als ein Literal nach der Substitution verbleibt*
begin
Lies alle L_i parallel von links nach rechts, bis in zwei Literalen die entsprechenden Zeichen verschieden sind;
if keines der beiden Zeichen ist eine Variable
then terminiere mit Ausgabe „nicht unifizierbar“
else
begin
Sei X die Variable und t der Teilterm im anderen Literal, dessen erstes Symbol sich von X unterscheidet
if X kommt in t vor *%occur check*
then terminiere mit Ausgabe „nicht unifizierbar“
else $sub_U := sub_U \circ \{X/t\}$
end
end

Bsp. Algorithmus v. Robinsohn

Eingabe: L = {p(X,Y), p(f(a), g(X)), p(f(Z), g(f(Z)))}

sub_{U1}= {X/f(a)}

L sub_{U1} = {p(f(a),Y), p(f(a), g(f(a))), p(f(Z), g(f(Z)))}

sub_{U2}= {X/f(a)}{Z/a}

L sub_{U2} = {p(f(a),Y), p(f(a), g(f(a)))}

sub_{U3}= {X/f(a)} {Z/a}{Y/g(f(a))}

L sub_{U3} = {p(f(a), g(f(a)))}

|L sub_{U3}| = 1, also sub_{U3} allgemeinster Unifikator.

Resolutionsprinzip

Hornklausel $\alpha \Rightarrow a$

$$\frac{\alpha \, , \, \alpha \Rightarrow a}{a}$$ (modus ponens)

$$\frac{(\alpha_1 \vee a) \, , \, (\alpha_2 \vee \neg a)}{(\alpha_1 \vee \alpha_2)}$$ (resolution)

SLD-Resolution

bei geg. Auswahlregel:

$$\frac{(A_1, ..., A_{k-1}, A_k, A_{k+1}, ... A_n \Rightarrow) , (B_1, B_2, ..., B_l \Rightarrow B)}{(A_1, ..., A_{k-}1, B_1, B_2, ..., B_l, A_{k+1}, ... A_n \Rightarrow) \, sub_U}$$

oder

$$(A_1, ..., A_{k-1}, A_k, A_{k+1}, ... A_n \Rightarrow) \vdash (A_1, ..., A_{k-}1, B_1, B_2, ..., B_l, A_{k+1}, ... A_n \Rightarrow)$$

SLD-Ableitung

$A \vdash R_1 \vdash R_2 \ldots$

Bsp. Auswahlregel „Wähle das letzte Literal“

geg. => p(a)
=> p(b)
q(a) => p(a)
p(X) => q(X)

Anfrage:
p(b), q(a) =>

SLD-Ableitung:

p(b), q(a) =>
|-
p(b), p(a) =>
$\vdash_{sub}$
p(b) =>
o

$sub_U = \{X/a\}$

Ergebnisse v. SLD-Ableitung

- **erfolgreiche**: enden in der leeren Klausel (SLD-Widerlegung)
- **fehlgeschlagene**: sind endlich, aber enden nicht mit der leeren Klausel
- **unendliche**

Nachweis der Unerfüllbarkeit

Gegeben sei ein logisches Programm P, eine Anfrage A und eine Auswahlregel R.
Dann gilt: $P \cup \{A\}$ ist genau dann unerfüllbar, wenn eine SLD-Widerlegung bzgl. R existiert.

Anm.
Es reicht daher aus, mit einer **beliebigen** Auswahlregel R **eine** SLD-Widerlegung bzgl. R zu finden, um die Unerfüllbarkeit einer Hornklauselmenge zu zeigen.

Berechnete Antworten

Eine berechnete Antwort U für $P \cup \{A\}$ ist eine Substitution der in der Anfrage A vorkommenden Variablen, wobei eine SLD-Widerlegung $A \vdash R_1 \vdash R_2 \ldots \vdash o$ für $P \cup \{A\}$ bzgl. der Auswahlregel existiert, die die Unifikatoren U_i benutzt und für die gilt: U ersetzt die in A vorkommenden Variablen genauso wie die Substitution $U_n(U_{n-1}(\ldots U_1)\ldots)$.

Bsp.

geg. => p(a)

=> p(b)

q(a) => p(a)

p(X) => q(X)

Anfrage:

q(Z) =>

q(Z) => $sub_{U1}=\{X/Z\}$

|-

p(Z) => $sub_{U2}=\{Z/a\}$

|-

o

Antwort: Z=a

da $sub_{U2}(sub_{U1})=\{X/a, Z/a\}$

Äquivalenz der Semantiken

Gegeben sei ein logisches Programm P, eine Anfrage A und eine Auswahlregel R.
Jede R-berechnete Antwort für $P \cup \{A\}$ ist auch eine korrekte Antwort.

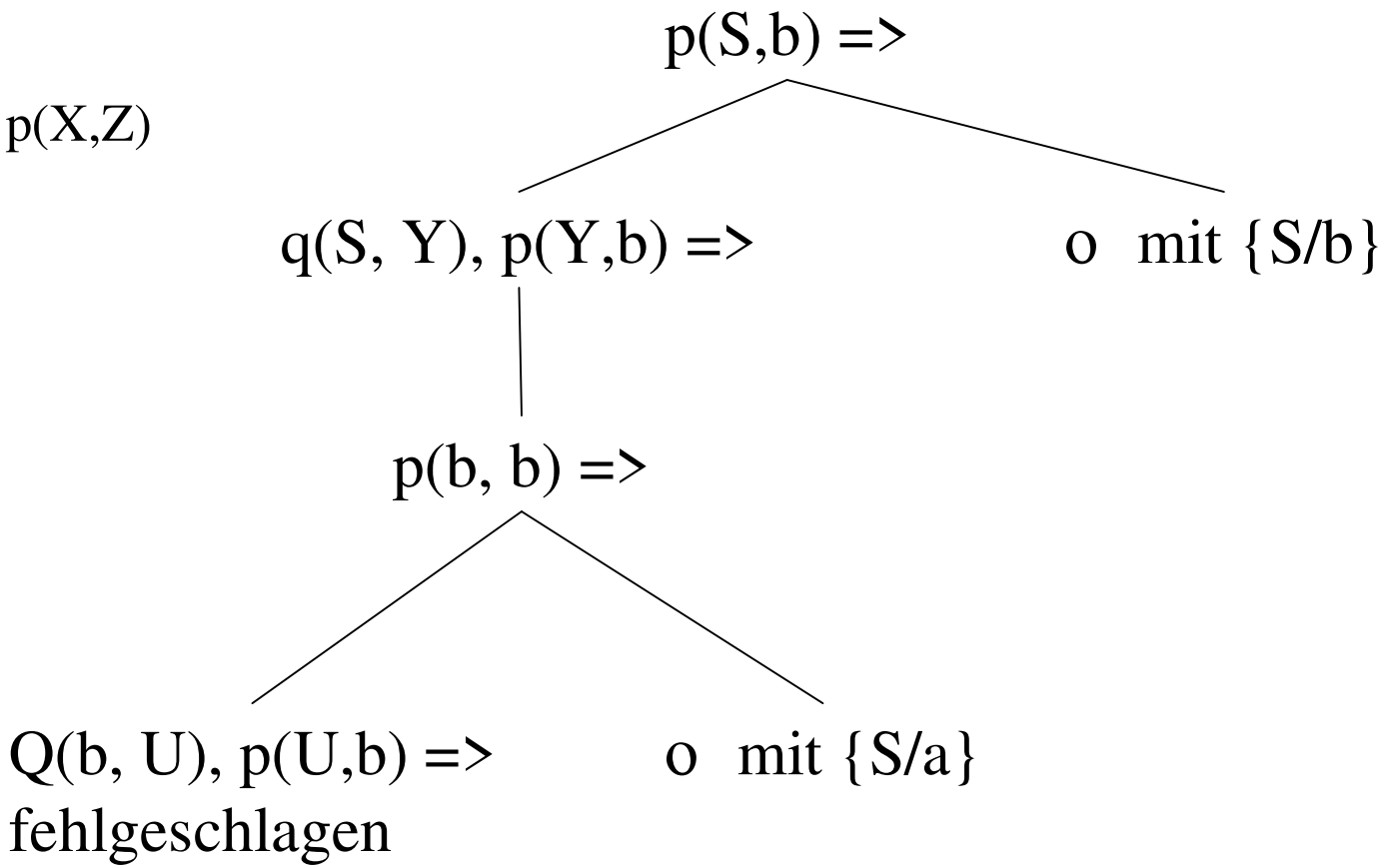
Jede korrekte Antwort ist ein Spezialfall einer berechneten Antwort.

Fazit:
Die deklarative und prozedurale Semantik von logischen Programmen ist äquivalent.

SLD-Baum

Log. Programm:

q(X,Y), p(Y,Z) => p(X,Z)
=> p(X,X)
=> q(a,b)



Suche nach Erfolgszweigen

Sei P ein logisches Programm und A eine Anfrage. Dann gilt:
Ist $P \cup \{A\}$ unerfüllbar, dann enthält der zugehörige SLD-Baum mindestens einen Erfolgszweig.

Suchstrategien im SLD-Baum
- Breitendurchlauf: Erfolg garantiert, aber ineffizient
- Tiefendurchlauf: effizient, aber Risiko der Nichttermination

Prolog-Strategie:
- Tiefendurchlauf
- Abarbeitung der Klauseln in der Sammlung von oben nach unten
- Abarbeitung der Literale im Rumpf von links nach rechts