Michael Arndt
Fritz Hamm
Peter Schroeder-Heister

Sommersemester 2003
Übungsblatt 2
14.5.2003

Übungen zur Vorlesung *Logikprogrammierung*

DATALOG – PROLOG mit Variablen und Konstanten

Syntax von DATALOG:

PROGRAM	$\rightarrow$	CLAUSE PROGRAM
PROGRAM	$\rightarrow$	ε
CLAUSE	$\rightarrow$	ATOM.
CLAUSE	$\rightarrow$	ATOM :- ATOMLIST
ATOMLIST	$\rightarrow$	ATOM.
ATOMLIST	$\rightarrow$	ATOM, ATOMLIST
GOAL	$\rightarrow$	?- ATOMLIST
ATOM	$\rightarrow$	RELSYMB
ATOM	$\rightarrow$	RELSYMB (TERMLIST)
TERMLIST	$\rightarrow$	TERM
TERMLIST	$\rightarrow$	TERM, TERMLIST
TERM	$\rightarrow$	VARSYMB
TERM	$\rightarrow$	CONSTSYMB
RELSYMB	$\rightarrow$	[a–z] ([a–zA–Z0–9_])*
CONSTSYMB	$\rightarrow$	[0–9]* | [a–z] ([a–zA–Z0–9_])*
VARSYMB	$\rightarrow$	[A–Z] ([a–zA–Z0–9_])*

Ein Term ist eine Variable oder eine Konstante. Ein Atom ist ein Ausdruck der Form

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

wobei r ein Relationssymbol ist und $t_1, \ldots, t_n$ Terme sind. DATALOG-Programme haben fast dieselbe Form wie PROPLOG-Programme mit dem Unterschied, daß anstelle der aussagenlogischen Variablen (PROPSYMB) nun Atome stehen dürfen.

Bsp. eines DATALOG-Programms:

```
r(c0,c1).
r(c1,c2).
q(X,Y) :- r(X,Y).
q(X,Z) :- r(X,Y), r(Y,Z).
```

Die Bedeutung der letzten Klausel ist

$$\forall \mathtt{X}\, \forall \mathtt{Y}\, \forall \mathtt{Z}\, [\mathtt{r}(\mathtt{X}, \mathtt{Y}) \wedge \mathtt{r}(\mathtt{Y}, \mathtt{Z}) \rightarrow \mathtt{q}(\mathtt{X}, \mathtt{Z})],$$

oder äquivalent dazu

$$\forall \mathtt{X}\, \forall \mathtt{Z}\, [\exists \mathtt{Y}\, (\mathtt{r}(\mathtt{X}, \mathtt{Y}) \wedge \mathtt{r}(\mathtt{Y}, \mathtt{Z})) \rightarrow \mathtt{q}(\mathtt{X}, \mathtt{Z})].$$

Wie funktioniert ein DATALOG Interpreter?

Zusätzlich zu den Hilfsfunktionen des PROPLOG-Interpreters brauchen wir einige Funktionen. Weiter nehmen wir an, daß Substitutionen Listen von Bindungen sind, also beispielsweise

$$\{t_1/X_1, \ldots, t_n/X_n\}.$$

Dabei sind $X_1, \ldots, X_n$ Variablen und $t_1, \ldots, t_n$ Terme.

relsymb(A)**:** falls A ein Atom ist, so gibt diese Funktion das Relationssymbol von A zurück, d.h. $\text{relsymb}(r(t_1, \ldots, t_n)) = r$.

arity(A)**:** gibt die Stelligkeit des Relationssymbols des Atoms A zurück, d.h. $\text{arity}(r) = 0$ und $\text{arity}(r(t_1, \ldots, t_n)) = n$.

arg(A, i)**:** gibt das i-te Argument des Atoms A zurück, falls ein solches existiert, also $\text{arg}(r(t_1, \ldots, t_n), i) = t_i$, falls $1 \leq i \leq n$.

apply(L, θ)**:** wendet die Substitution θ auf L an und erzeugt eine neue Kopie. L kann ein Atom oder eine Liste von Atomen sein; $\text{apply}([A_1, \ldots, A_n], \theta) = [A_1\theta, \ldots, A_n\theta]$ und $\text{apply}(A, \theta) = A\theta$.

compose(θ_1, θ_2)**:** berechnet die Komposition der beiden Substitutionen θ_1 und θ_2, also $\text{compose}(\theta_1, \theta_2) = \theta_1\theta_2$.

rename(C)**:** gibt eine Umbenenung der Klausel C zurück und erzeugt eine neue Kopie. Wir nehmen an, daß es einen globalen Zähler i gibt, so daß rename(C) in der Klausel C alle Variablen mit dem Subskript i versieht und nachher i um 1 erhöht. Zum Beispiel, falls $i = 3$, dann ist

$$\text{rename}(q(X, Z) \text{ :- } r(X, Y), r(Y, Z)) = q(X_3, Z_3) \text{ :- } r(X_3, Y_3), r(Y_3, Z_3).$$

variable(t)**:** gibt 'true' zurück, falls der Term t eine Variable ist und 'false' sonst.

Die Funktion **match**(A, B, θ) gibt 'true' zurück, falls A und B unifizierbar sind und berechnet den allgemeinsten Unifikator in θ. Falls A und B nicht unifizierbar sind ist das Resultat 'false'.

```
function match(A, B: atom, var θ: substitution): boolean;
begin
  if (relsymb(A) ≠ relsymb(B)) or (arity(A) ≠ arity(B)) then return(false)
  else
    θ := {};
    for i := 1 to arity(A) do
      if variable(arg(A, i)) then
        θ := compose(θ, {arg(B, i)/arg(A, i)}); A := apply(A, θ); B := apply(B, θ)
      elsif variable(arg(B, i)) then
        θ := compose(θ, {arg(A, i)/arg(B, i)}); A := apply(A, θ); B := apply(B, θ)
      elsif arg(A, i) ≠ arg(B, i) then return(false) end
    end;
    return(true)
  end
end
```

Wie bei PROPLOG versucht die Funtkion *provable* ein Ziel mit einer *Depth-First* Suche zu beweisen.

> **function** provable(G: goal): boolean;
> **begin**
>   **if** goal = [] **then return**(true)
>   **else**
>   **for** $i := 1$ **to** MAXCLAUSES **do**
> $C :=$ rename(clause[i]);
>   **if** match$(\text{head}(C), \text{first}(G), \theta)$ **then**
>   **if** provable$(\text{apply}(\text{concatenate}(\text{body}(C), \text{rest}(G)), \theta))$ **then**
>   **return**(true)
>   **end**
>   **end**
>   **end**;
>   **return**(false)
>   **end**
> **end**

Antworten

Ein PROLOG-Interpreter versucht nicht nur zu beweisen, daß ein Ziel aus einem Programm folgt, sondern er berechnet dabei auch mögliche Substitutionen für Variablen in der Anfrage. Im Beispiel oben sind die möglichen Antworten auf die Frage `?- q(U,V)`:

(1) $\{\texttt{c0/U}, \texttt{c1/V}\}$.

(2) $\{\texttt{c1/U}, \texttt{c2/V}\}$.

(3) $\{\texttt{c0/U}, \texttt{c2/V}\}$.

Den DATALOG-Interpreter kann man leicht dahingehend modifizieren, daß er Antwortsubstitutionen berechnet. Angenommen, im Ziel ?- $B_1, \ldots, B_m$ kommen die Variablen $X_1, \ldots, X_n$ vorkommen. Dann ändere man das Ziel zu ?- $B_1, \ldots, B_m, \$\text{answer}(X_1, \ldots, X_n)$ ab. Die Funktion *provable* testet nun nicht mehr auf goal = [] sondern auf goal = [$\$\text{answer}(t_1, \ldots, t_n)$], und, wenn dies zutrifft, gibt sie die folgende Anwort zurück:

$$\{t_1/X_1, \ldots, t_n/X_n\}$$

Aufgabe 1 (2 Punkte)

Für welche der folgenden Paare von Atomen gibt die Funktion *match* den Wert true zurück? Berechnen Sie den allgemeinsten Unifikator θ für diese Paare.

(1) `p(a,X,X,b)` und `p(Y,Y,Z,Z)`

(2) `p(a,X,b,X)` und `p(Y,Y,Z,Z)`

(3) `r(X,V,X,V)` und `r(Y,Y,Z,Z)`

(4) `r(X,V,V,a)` und `r(Y,Y,Z,Z)`

Aufgabe 2 (1 Punkt)

Gegeben ist folgende Datenbank:

```
r(c0).
r(c1).
r(c2).
q(c1).
q(c2).
q(c3).
```

(1) Schreiben Sie ein 1-stelliges Prädikat `intersection(X)`, das wahr ist, falls sowohl `r(X)` als auch `q(X)` gilt.

(2) Schreiben Sie ein 1-stelliges Prädikat `union(X)`, das wahr ist, falls `r(X)` oder `q(X)` gilt.

Aufgabe 3 (3 Punkte)

Gegeben ist folgendes DATALOG-Programm:

```
r(c0,c1).
r(c1,c2).
q(X,Y) :- r(X,Y).
q(X,Z) :- r(X,Y), r(Y,Z).
```

Schreiben Sie von Hand den vollständigen Suchbaum für die folgenden Ziele auf:

(1) `?- q(c0,c2).`

(2) `?- q(U,V).`

Aufgabe 4

Studieren Sie die folgenden System-Prädikate in der Dokumentation von SWI-Prolog:

`consult`/1, `halt`/0, `listing`/0, `listing`/1, `trace`/0, `notrace`/0.

(In Prolog können Funktionen, die unterschiedliche Stellenzahl haben, mit dem gleichen Funktionsnamen versehen werden. Daher gibt man die Stellenzahl einer Funktion hinter einem Schrägstrich an. `listing`/0 ist also eine nullstellige Funktion, `listing`/1 eine einstellige.)