Programmierkurs Prolog

SS 1998

Thorsten Joachims

Universität Dortmund

LS VIII - Prof. K. Morik

Inhalt

- Vergleiche von Termen
- Typtests auf Termen
- Zugriff auf den Termaufbau
- Zugriff auf Fakten und Regeln
- All-Solutions Prädikate
- Metaprogrammierung

Vergleich von Termen

• T1 = T2	T1 und T2 sind unifizierbar
• T1 \= T2	T1 und T2 sind nicht unifizierbar.
• T1 == T2	Die Terme, mit denen T1 und T2 instanziiert sind, sind identisch.
• T1 \== T2	T1 und T2 sind nicht identisch
• T1 @< T2	Prüfen auf alphabetische Ordnung
• T1 @=< T2	
• T1 @> T2	
• T1 @>= T2	

Typtests auf Termen

• atom(?Term)	Test auf Prolog-Atom
• atomic(?Term)	Atom/Zahl/String
• compound(?Term)	Strukture oder Liste
• var(?Term)	Test auf Variable
• nonvar(?Term)	Zumindest teilweise instantiiert
• integer(?Term)	Test auf ganze Zahl
• real(?Term)	Test auf reele Zahl
• number(?Term)	Test auf Zahl
• string(?Term)	Test auf String

Zugriff auf den Termaufbau

• ?Term ..= ?List	Term <-> Liste
• atom_string(?A,?S)	Atom <-> String
• integer_atom(?I,?A)	Integer <-> Atom
• number_string(?N,?S)	Zahl <-> String
• char_int(?C,?I)	Zeichen <-> ASCII
• string_list(?S,?L)	L ist liste der ASCII Codes
• functor(?T,?F,?A)	F ist Funktor von T/A
• copy_term(+T1,?T2)	T2 is Kopie von T1 mit neuen Variablen

Metapraedikat: clause(H,B)

`clause(H,B)`	Es wird nach einer Klausel gesucht, deren Kopf mit H und deren Koerper mit B unifizierbar ist.

Hinzufuegen von Klauseln

`assert(Clause).`	Die Klausel Clause wird zur Datenbasis hinzugefuegt.
`asserta(Clause).`	Einfuegen von Clause am Beginn.
`assertz(Clause).`	Einfuegen von Clause am Ende.

Clause:

```
Head
Head :- Body
```

Loeschen von Klauseln (I)

`retract(Clause).` Die erste Klausel, die mit Clause unifiziert werden kann, wird geloescht.

Clause:
```
Head
Head :- Body
```

Der Koerper einer Klause kann eine freie Variable sein bzw. in ihm koennen ungebundene Variablen auftauchen.

```
?- retract((foo(X) :- Body)),fail.
```

Loeschen von Klauseln (II)

`retractall(Head).` Alle Klauseln, deren Kopf mit Head unifiziert werden kann, werden geloescht.

Kann man schreiben als:

```
retractall(Head) :-
    retract((Head :- Body)),fail.
retractall(_).
```

Alle Lösungen einer Anfrage

In einer Familiendatenbank sind Relationen der Art

```
vater(joe,mary).
vater(joe,anne).
vater(henry,barbara).
...
```

enthalten. Wir wollen alle Kinder des Vaters Joe bestimmen.

```
?- vater(joe,X).
X=mary ;
X=anne ;
no (more) solutions
```

Problem: Es ist immer nur eine Lösung verfügbar.

Andere Möglichkeit durch ‘failure driven loop’:

```
?- vater(joe,X), write(X), nl,fail.
mary
anne
No
```

Mengenprädikate

Problem der vorangegangenen Ansätze: Sie schlagen fehl.

Alle Lösungen zu finden geht über die Ausdruckskraft der Prädikatenlogik 1. Stufe hinaus. Prolog bietet einige Prädikate, die sich ähnlich wie Prädikatenlogik 2. Stufe verhalten.

- findall(?X, +Goal, ?List)
- bagof(?X, +Goal, ?List)
- setof(?X, +Goal, ?List)

Mengenprädikat findall/3

```
findall(?Template, +Generator, ?Liste)
```

Für jede Lösunge des Generator-Zieles wird das instantiierte Template in die Liste aufgenommen.

```
vater(joe,mary).
vater(joe,anne).
vater(henry,barbara).

?- findall(X,vater(joe,X),L).
X = X
L = [mary, anne]

?- findall(X,vater(Y,X),L).
Y = Y
X = X
L = [mary, anne, barbara]
```

Mengenprädikat bagof/3

```
bagof(?Template, +Generator, ?Liste)
```

Die nicht im Template vorkommenden freien Variablen können selektiv existentiell gebunden werden.

```
vater(joe,mary).
vater(joe,anne).
vater(henry,barbara).

?- bagof(X,Y^vater(Y,X),L).
Y = Y X = X
L = [mary, anne, barbara]

?- bagof(X,vater(Y,X),L).
Y = joe X = X
L = [mary, anne] ;
Y = henry X = X
L = [barbara] ;
```

Mengenprädikat setof/3

```
bagof(?Template, +Generator, ?Liste)
```

Wie bagof/3, aber leiche Lösungen werden aus der Liste gelöscht.

```
vater(joe,mary).
vater(joe,anne).
vater(henry,barbara).
vater(henry,mary).

?- bagof(X,Y^vater(Y,X),L).
Y = Y X = X
L = [mary, anne, barbara,mary]

?- setof(X,Y^vater(Y,X),L).
Y = Y X = X
L = [mary, anne, barbara]
```

Metapraedikat: call(X)

`call(X)` Das Goal X wird ausgefuehrt.

X muss ein ausfuehrbares Ziel sein, d. h. es an muss einen Term gebunden sein, der keinen Typfehler verursacht.