Programmierkurs Prolog

SS 1998

Thorsten Joachims

Universität Dortmund

LS VIII - Prof. K. Morik

nach N. Fuchs/T. Meyer

Inhalt

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

Metainterpreter

Was ist ein Metainterpreter?

Ein Metainterpreter ist ein Prolog-Programm, das andere Prolog-Programme interpretiert.

Wozu braucht man Metainterpreter?

- Debugging
- Effizienz
- Einführung neuer Sprachmittel

Top-Down Metainterpreter

```
prove(true) :- !.
prove((Goal1,Goal2)) :- !,
      prove(Goal1),
      prove(Goal2).
prove(Goal) :-
      clause(Goal,Body),
      prove(Body).
```

Top-Down Metainterpreter mit Build-In Prädikaten

Mit build-in Prädikaten:

```
prove(true) :- !.
prove((Goal1,Goal2)) :- !,
        prove(Goal1),
        prove(Goal2).
prove(Goal) :-
        builtin(Goal),!,
        call(Goal).
prove(Goal) :-
        clause(Goal,Body),
        prove(Body).
```

Gegeben: builtin/1

Metainterpreter zum Zählen der Inferenzschritte

```
count(Goal,Count) :-
      count(Goal,0,Count).


count(true,C,C) :- !.
count((Goal1,Goal2),C1,C3) :- !,
      count(Goal1,C1,C2),
      count(Goal2,C2,C3).
count(Goal,C1,C2) :-
      builtin(Goal),!,
      call(Goal),
      C2 is C1+1.
count(Goal,C1,C3) :-
      clause(Goal,Body),
      C2 is C1+1,
      count(Body,C2,C3).
```

Gegeben: builtin/1

Fibonacci-Zahlen

```
:- dynamic fib/2.

fib(0,0).
fib(1,1).
fib(N,F) :-
    N>1,
    N1 is N-1,
    N2 is N-2,
    fib(N1,F1),
    fib(N2,F2),
    F is F1+F2.


:- dynamic fib2/2.

fib2(0,0).
fib2(1,1).
fib2(N,F) :-
    N>1,
    N1 is N-1,
    N2 is N-2,
    fib2(N1,F1),
    fib2(N2,F2),
    F is F1+F2,
    asserta( (fib2(N,F) :- true) ).
```

Beispiel zu COUNT

```
builtin(_ is _).
builtin(_ > _).
builtin(asserta(_)).

[eclipse 11]: fib(15,L).
L = 610     More? (;)
yes.

[eclipse 7]: count(fib(15,L),Steps).
L = 610
Steps = 5917     More? (;)
yes.

[eclipse 10]: count(fib2(15,L),Steps).
L = 610
Steps = 99     More? (;)
yes.
```

Metainterpreter zur Konstruktion des Beweisbaumes

Metazirkuläre Interpreter

Metazirkuläre Interpreter sind Metainterpreter, die sich selber interpretieren können.

```
count(Goal,Count) :- count(Goal,0,Count).

count(true,C,C).
count((Goal1,Goal2),C1,C3) :-
        count(Goal1,C1,C2),
        count(Goal2,C2,C3).
count(Goal,C1,C2) :-
        builtin(Goal),
        call(Goal),
        C2 is C1+1.
count(Goal,C1,C3) :-
        userdef(Goal),
        clause(Goal,Body),
        C2 is C1+1,
        count(Body,C2,C3).

builtin(_ is _).builtin(_ > _).
builtin(asserta(_)).builtin(call(_)).
builtin(clause(_,_)). builtin(_ =.. _).

userdef(count(_,_)).userdef(count(_,_,_)).
userdef(builtin(_)). userdef(fib(_,_)).
userdef(fib2(_,_)). userdef(userdef(_)).
```

Beispiel: Metazirkulärer Interpreter

```
[3]: count(fib(15,L),C1).
L = 610
C1 = 5917     More? (;)
yes.

[4]: count(count(fib(15,L),C1),C2).
L = 610
C1 = 5917
C2 = 29586     More? (;)
yes.

[]:count(count(count(fib(15,L),C1),C2),C3).
L = 610
C1 = 5917
C2 = 29586
C3 = 143986     More? (;)
yes.
```