PROGRAMMIERUNG

ÜBUNG 9: LOGIKPROGRAMMIERUNG MIT PROLOG⁻

Eric Kunze

eric.kunze@tu-dresden.de

INHALT

Logikprogrammierung und Prolog⁻

„DATENSTRUKTUREN“ IN PROLOG

- Darstellung von Objekten als Terme über Konstruktoren
- keine explizite Deklaration – implizite Definition über Verwendung in Klauseln

natürliche Zahlen: Prädikat `nat`

- nullstelliger Konstruktor `0`
- einstelliger Konstruktor `s(X)`

```
1 nat(0).
2 nat(s(X)) :- nat(X).
```

Listen: Prädikat `list`

Abkürzung:

- nullstelliger Konstruktor `nil` ⇝ `[]`
- zweistelliger Konstruktor `cons(X, Xs)` ⇝ `[X|Xs]`

```
3 list(nil).
4 list(cons(X, Xs)) :- list(Xs).
```

„DATENSTRUKTUREN“ IN PROLOG

Erinnerung: natürliche Zahlen, Listen

```
1 nat(0).
2 nat(s(X)) :- nat(X).
3 list(nil).
4 list(cons(X, Xs)) :- list(Xs).
```

Bäume: Prädikat `istree`

- nullstelliger Konstruktor `nil`
- dreistelliger Konstruktor `tree(X,L,R)`

```
5 istree(nil).
6 istree(tree(_, L, R) :- istree(L), istree(R).
```

Aufgabe 1

Listen

AUFGABE 1 – TEIL (A)

Ziel: binäre Relation `sublist` mit

$$(\ell_1, \ell_2) \in \texttt{sublist} \quad \Leftrightarrow \quad \ell_1 \subseteq \ell_2$$

$\rightsquigarrow \ell_1$ soll *Teilliste* von ℓ_2 sein

```
1 nat (0).
2 nat(s(X)) :- nat(X).
3
4 listnat ([]).
5 listnat ([X|XS]) :- nat(X), listnat(XS).
```

AUFGABE 1 – TEIL (A)

Ziel: binäre Relation `sublist` mit

$$(\ell_1, \ell_2) \in \texttt{sublist} \quad \Leftrightarrow \quad \ell_1 \subseteq \ell_2$$

$\rightsquigarrow \ell_1$ soll *Teilliste* von ℓ_2 sein

```
1  nat (0).
2  nat(s(X)) :- nat(X).
3
4  listnat ([]).
5  listnat ([X|XS]) :- nat(X), listnat(XS).
```

```
6  sublist(Xs    , [Y|Ys]) :- nat(Y), sublist(Xs, Ys).
7  sublist(Xs    , Ys )    :- prefix(Xs, Ys).
8
9  prefix([]     , Ys )    :- listnat(Ys).
10 prefix([X|Xs], [X|Ys]) :- nat(X), prefix(Xs, Ys).
```

AUFGABE 1 – TEIL (B)

Belegung 1:

```
            ?-  sublist ([<4>|Xs], [<5>, <4>, <3>]).
            ?-  nat(<5>), sublist ([<4>|Xs], [<4>, <3>]).  % 6
            ?-* nat(0), sublist ([<4>|Xs], [<4>, <3>]).    % 2
            ?-  sublist ([<4>|Xs], [<4>, <3>]).            % 1
            ?-  prefix ([<4>|Xs], [<4>, <3>]).             % 7
            ?-  nat(<4>), prefix(Xs , [<3>]).              % 10
            ?-* nat(0), prefix(Xs , [<3>]).                % 2
            ?-  prefix(Xs , [<3>]).                        % 1
{Xs = []}   ?-  listnat ([<3>]).                           % 9
            ?-  nat(<3>), listnat ([]).                    % 5
            ?-* nat(0), listnat ([]).                      % 2
            ?-  listnat ([]).                              % 1
            ?-  .                                          % 4
```

Somit also `Xs = []`.

AUFGABE 1 – TEIL (B)

Belegung 2:

```
                  ?-  sublist ([<4>|Xs], [<5>, <4>, <3>]).
                  ?-  nat(<5>), sublist ([<4>|Xs], [<4>, <3>]).
                                                          % 6
                  ?-* nat(0), sublist ([<4>|Xs], [<4>, <3>]).
                                                          % 2
                  ?-  sublist ([<4>|Xs], [<4>, <3>]).     % 1
                  ?-  prefix ([<4>|Xs], [<4>, <3>]).      % 7
                  ?-  nat(<4>), prefix(Xs , [<3>]).       % 10
                  ?-* nat(0), prefix(Xs , [<3>]).         % 2
                  ?-  prefix(Xs , [<3>]).                 % 1
{Xs=[<3>|Xs1]}    ?-  nat(<3>), prefix(Xs1 , []).         % 10
                  ?-* nat(0), prefix(Xs1 , []).           % 2
                  ?-  prefix(Xs1 , []).                   % 1
{Xs1 = []}        ?-  listnat ([]).                       % 9
                  ?-  .                                   % 4
```

Somit also `Xs = [<3>|Xs1] = [<3>]`.

Aufgabe 2

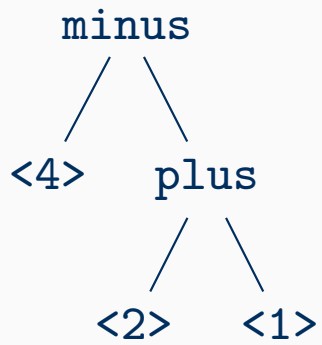
Bäume

AUFGABE 2 – TEIL (A)

Wir wollen einen binären Termbaum auswerten.

```
1 nat (0).
2 nat(s(X)) :- nat(X).
3 sum(0, Y, Y) :- nat(Y).
4 sum(s(X), Y, s(S)) :- sum(X, Y, S).
```

Beispiel:



Kodierung über zweistelligen Konstruktoren `plus` und `minus`

```
minus(
   <4>,
   plus(<2>, <1>)
)
```

$\rightsquigarrow \text{minus}(4, \text{plus}(2, 1)) = 4 - (2 + 1) = 1$

AUFGABE 2 – TEIL (A)

Wir wollen einen binären Termbaum auswerten.

```
1 nat (0).
2 nat(s(X)) :- nat(X).
3 sum(0, Y, Y)        :- nat(Y).
4 sum(s(X), Y, s(S)) :- sum(X, Y, S).
```

AUFGABE 2 – TEIL (A)

Wir wollen einen binären Termbaum auswerten.

```
1 nat (0).
2 nat(s(X)) :- nat(X).
3 sum(0, Y, Y)       :- nat(Y).
4 sum(s(X), Y, s(S)) :- sum(X, Y, S).
```

```
5 eval( X          , X ) :-  nat(X).
6 eval( plus (L,R), X ) :-  eval(L, LE), eval(R, RE), sum(LE, RE,  X).
7 eval( minus(L,R), X ) :-  eval(L, LE), eval(R, RE), sum(RE,  X, LE).
```

AUFGABE 2 – TEIL (B)

Alternative 1:

```
                                ?-  insert(<t1>, <t2>, X).
{X = tree(a, LT1, RT1)}         ?-  insert(tree(b, nil, nil), <t2>, LT1),
                                    insert(tree(v, nil, nil), <t2>, RT1).   % 6
{LT1 = tree(b, LT2, RT2)}       ?-  insert(nil , <t2>, LT2),
                                    insert(nil, <t2>, RT2),
                                    insert(tree(v, nil, nil), <t2>, RT1).   % 6
{LT2 = nil, RT2 = nil}          ?-* insert(tree(v, nil, nil), <t2>, RT1).   % 4
{RT1 = <t2>}                    ?-  istree(<t2>).                           % 5
                                ?-* istree(nil), istree(nil), istree(nil).  % 2
                                ?-* .                                       % 4
```

Alternative 2: die ersten vier Goals stimmen mit Alternative 1 überein

```
                                ?-  insert(<t1>, <t2>, X).
{X = tree(a, LT1, RT1)}         ?-  insert(tree(b, nil, nil), <t2>, LT1),
                                    insert(tree(v, nil, nil), <t2>, RT1).   % 6
{LT1 = tree(b, LT2, RT2)}       ?-  insert(nil , <t2>, LT2),
                                    insert(nil, <t2>, RT2),
                                    insert(tree(v, nil, nil), <t2>, RT1).   % 6
{LT2 = nil, RT2 = nil}          ?-* insert(tree(v, nil, nil), <t2>, RT1).   % 4
{RT1 = tree(v, LT3, RT3)}       ?-  insert(nil, <t2>, LT3),
                                    insert(nil, <t2>, RT3).                 % 6
{RT3 = nil, LT3 = nil}          ?-* .                                       % 4
```

Ein weiteres Beispiel

aus der Aufgabensammlung

AUFGABE AGS 13.5 – TEIL (A)

Gegeben sei folgender Prolog-Code:

```
1 subt( X   , X          ).
2 subt( S1  , s(_  , T2) ) :- subt(S1,T2).
3 subt( S1  , s(T1 , _ ) ) :- subt(S1,T1).
```

Gesucht sind Belegungen für `X` und Y für das Goal `?- subt(s(X, Y), s(s(a, b), s(b, a))).`

AUFGABE AGS 13.5 – TEIL (A)

Gegeben sei folgender Prolog-Code:

```
1 subt( X   , X           ).
2 subt( S1  , s(_  , T2) ) :- subt(S1,T2).
3 subt( S1  , s(T1 , _ ) ) :- subt(S1,T1).
```

Gesucht sind Belegungen für `X` und Y für das Goal `?- subt(s(X, Y), s(s(a, b), s(b, a))).`

```
                                ?-  subt(s(X,Y), s(s(a,b), s(b,a))).
{X = s(a,b), Y=s(b,a)}          ?-  .                                % 1

                                ?-  subt(s(X,Y), s(s(a,b), s(b,a))).
                                ?-  subt(s(X,Y), s(b,a)).            % 2
{X = b, Y=a}                    ?-  .                                % 1

                                ?-  subt(s(X,Y), s(s(a,b), s(b,a))).
                                ?-  subt(s(X,Y), s(a,b)).            % 3
{X = a, Y=b }                   ?-  .                                % 1
```

AUFGABE AGS 13.5 – TEIL (B)

Gegeben sei folgender Prolog-Code:

```
1 subt( X   , X          ).
2 subt( S1  , s(_  , T2) ) :- subt(S1,T2).
3 subt( S1  , s(T1 , _ ) ) :- subt(S1,T1).
```

Gesucht sind drei Lösungen für das Goal `?- subt(s(a, a), X).`

AUFGABE AGS 13.5 – TEIL (B)

Gegeben sei folgender Prolog-Code:

```
1 subt( X   , X          ).
2 subt( S1  , s(_  , T2) ) :- subt(S1,T2).
3 subt( S1  , s(T1 , _ ) ) :- subt(S1,T1).
```

Gesucht sind drei Lösungen für das Goal `?- subt(s(a, a), X).`

```
                    ?-  subt(s(a,a), X).
{X = s(a,a)}        ?-  .                          % 1
                                                   ⇒ X = s(a,a))

                    ?-  subt(s(a,a), X).
{X = s(_ , X1)}     ?-  subt(s(a,a), X1).          % 2
{X1 = s(a,a)}       ?-  .                          % 1
                                                   ⇒ X = s(a,s(a,a))


                    ?-  subt(s(a,a), X).
{X = s(X2, _)}      ?-  subt(s(a,a), X2).          % 3
{X2 = s(a,a) }      ?-  .                          % 1
                                                   ⇒ X = s(s(a,a),c)
```