

PROGRAMMIERUNG

ÜBUNG 9: LOGIKPROGRAMMIERUNG MIT PROLOG-

Eric Kunze

`eric.kunze@tu-dresden.de`

1. Funktionale Programmierung
 - 1.1 Einführung in Haskell: Listen
 - 1.2 Algebraische Datentypen
 - 1.3 Funktionen höherer Ordnung
 - 1.4 Typpolymorphie & Unifikation
 - 1.5 Beweis von Programmeigenschaften
 - 1.6 λ -Kalkül
2. **Logikprogrammierung**
3. Implementierung einer imperativen Programmiersprache
4. Verifikation von Programmeigenschaften
5. H_0 – ein einfacher Kern von Haskell

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`

- ▶ nullstelliger Konstruktor `nil`
- ▶ zweistelliger Konstruktor `cons(X, Xs)`

Abkürzung:

\rightsquigarrow `[]`

\rightsquigarrow `[X|Xs]`

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

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

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

$\rightsquigarrow \ell_1$ 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

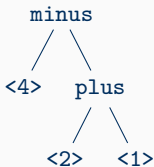
Bäume

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>)  
)
```

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

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).
```

```
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)

Gegeben: zwei Bäume

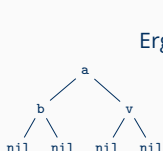
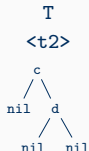
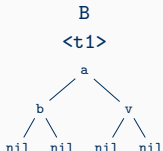
$\langle t1 \rangle = \text{tree}(a, \text{tree}(b, \text{nil}, \text{nil}), \text{tree}(v, \text{nil}, \text{nil}))$

$\langle t2 \rangle = \text{tree}(c, \text{nil}, \text{tree}(d, \text{nil}, \text{nil}))$

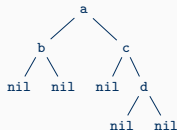
```
1 istree(nil).
2 istree(tree(_, L, R)) :- istree(L), istree(R).
3
4 insert(nil, _, nil).
5 insert(tree(v, _, _), T, T) :- istree(T).
6 insert(tree(X, L, R), T, tree(X, LT, RT)) :- insert(L, T, LT), insert(R, T, RT).
```

Anschauung: $\text{insert}(B, T, X)$ ist wahr falls X entsteht, indem alle Knoten v in B durch den Baum T ersetzt werden oder nicht

Beispiel:



oder



AUFGABE 2 – TEIL (B)

Gegeben: zwei Bäume

```
<t1> = tree(a, tree(b, nil, nil), tree(v, nil, nil))
```

```
<t2> = tree(c, nil, tree(d, nil, nil))
```

```
1 istree(nil).
2 istree(tree(_, L, R)) :- istree(L), istree(R).
3
4 insert(nil, _, nil).
5 insert(tree(v, _, _), T, T) :- istree(T).
6 insert(tree(X, L, R), T, tree(X, LT, RT)) :- insert(L, T, LT), insert(R, T, RT).
```

Gesucht: eine Belegungen für `X`, die `?- insert(<t1>, <t2>, X).` erfüllt

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
```

AUFGABE 2 – TEIL (B)

Gegeben: zwei Bäume

```
<t1> = tree(a, tree(b, nil, nil), tree(v, nil, nil))
```

```
<t2> = tree(c, nil, tree(d, nil, nil))
```

```
1  istree(nil).
2  istree(tree(_, L, R)) :- istree(L), istree(R).
3
4  insert(nil, _, nil).
5  insert(tree(v, _, _), T, T) :- istree(T).
6  insert(tree(X, L, R), T, tree(X, LT, RT))
7  :- insert(L, T, LT), insert(R, T, RT).
```

Gesucht: eine Belegungen für X , die `?- insert(<t1>, <t2>, X).` erfüllt

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

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

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

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

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

AUFGABE AGS 13.5 – TEIL (B)

Gegeben sei folgender Prolog-Code:

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

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

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

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

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