

PROGRAMMIERUNG

ÜBUNG 8: LOGIKPROGRAMMIERUNG MIT PROLOG-

Eric Kunze

`eric.kunze@mailbox.tu-dresden.de`

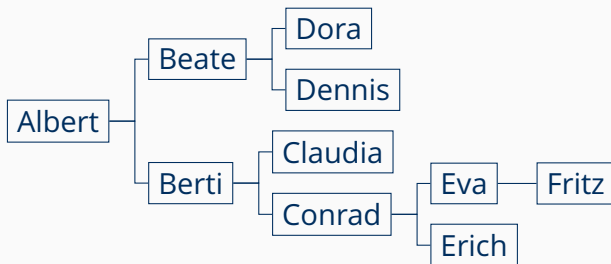
1. Funktionale Programmierung
 - 1.1 Einführung in Haskell: Listen
 - 1.2 Algebraische Datentypen
 - 1.3 Funktionen höherer Ordnung
 - 1.4 Typpolymorphie & Unifikation
 - 1.5 Beweis von Programmeigenschaften
 - 1.6 λ -Kalkül
2. **Logikprogrammierung**
3. Implementierung einer imperativen Programmiersprache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

Logikprogrammierung und Prolog⁻

- ▶ Französisch: programmation en logique
(deutsch: Programmieren in Logik)
- ▶ **Online-Editor & Interpreter:** `swish.swi-prolog.org`
- ▶ Prolog-Programme bestehen aus **Fakten** und **Regeln**.
- ▶ Statements werden mit `.` abgeschlossen.
- ▶ Variablen beginnen mit Großbuchstaben.
- ▶ **UND**-Operator. `,`
- ▶ **ODER**-Operator. `;`

EIN EINFÜHRENDES BEISPIEL

Wir betrachten den folgenden Familienstammbaum:



Nun wollen wir die Verwandschaftsbeziehungen abbilden und untersuchen. Dafür brauchen wir

- ▶ Geschlechter
- ▶ Eltern-Kind-Beziehung(en)

PROLOG: FAKTEN & REGELN

Fakten

- ▶ Prädikat mit Argumenten
- ▶ z.B. Albert ist männlich \rightsquigarrow `male(albert).`

Regeln

- ▶ Abhängigkeit eines Fakts von einem oder mehreren anderen Fakten
- ▶ z.B. Vater ist männliches Elternteil
 \hookrightarrow `father(X,Y) :- parent(X,Y), male(X).`
- ▶ `:-` kann als umgedrehte Implikation (\Leftarrow) gelesen werden

Nun möchten wir Programme auch ausführen. Aus Logik-Sicht ist die Ausführung eine Anfrage (*query*): wir wollen wissen, ob ein Fakt gilt oder nicht (bzw. ob er gültig gemacht werden kann). Diesen Fakt nennen wir das Ziel (*goal*).

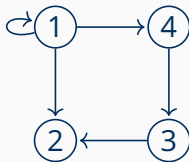
- ▶ Ist Albert männlich?
- ▶ Anfrage: `?- male(albert).`
- ▶ Antwort: `true.`

Im Allgemeinen gibt es kein I/O. Wir können das aber „simulieren“, indem wir Variablen nutzen.

- ▶ Welche Personen sind männlich?
- ▶ Anfrage: `?- male(X).`
- ▶ Anzeigen mehrerer Lösungen in `swipl` durch ;

Aufgabe 1

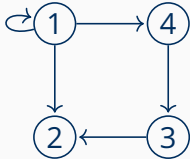
AUFGABE 1



```
1 edge(1,1) .  
2 edge(1,4) .  
3 edge(1,2) .  
4 edge(3,2) .  
5 edge(4,3) .
```

```
7 path(U, U) .  
8 path(U, W) :- edge(U, V), path(V, W) .
```

AUFGABE 1



```
?- path(4,X) .  
{X=4}    ?- .                                % 7
```

```
?- path(4,X) .  
?- edge(4,W) , path(W,X) .    % 8  
{W=3}    ?- path(3,X) .      % 5  
{X=3}    ?- .                % 6
```

```
?- path(4,X) .  
?- edge(4,W) , path(W,X) .    % 8  
{W=3}    ?- path(3,X) .      % 5  
?- edge(3,U) , path(U,X) .    % 8  
{U=2}    ?- path(2,X) .      % 4  
{X=2}    ?- .                % 7
```

Aufgabe 2

AUFGABE 2 – TEIL (A)

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

Gesucht: Prädikat `even`, dass alle natürlichen Zahlen enthält

```
7 even(0).  
8 even(s(s(N))) :- even(N).
```

AUFGABE 2 – TEIL (B)

```
1 nat(0).  
2 nat(s(X)) :- nat(X).  
3  
4 sum(0, Y, Y) :- nat(Y).  
5 sum(s(X), Y, s(S)) :- sum(X, Y, S).  
6  
7 even(0).  
8 even(s(s(N))) :- even(N).
```

Gesucht: Relation div2 mit $(\langle n \rangle, \langle \lfloor \frac{n}{2} \rfloor \rangle)$

```
10 div2(0, 0).  
11 div2(s(0), 0).  
12 div2(s(s(N)), s(M)) :- div2(N, M).
```

AUFGABE 2 – TEIL (C)

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

Gesucht: Relation div mit $(\langle n \rangle, \langle m \rangle, \langle \lfloor \frac{n}{m} \rfloor \rangle)$

```
14 lt(0, s(M)) :- nat(M).  
15 lt(s(N), s(M)) :- lt(N, M).
```

```
17 div(0, M, 0) :- lt(0, M).  
18 div(N, M, 0) :- lt(N, M).  
19 div(N, M, s(Q)) :- lt(0, M), sum(M, V, N),  
20                       div(V, M, Q).
```

AUFGABE 2 – TEIL (D)

```
?- div(<3>, <2>, <1>)
?- lt(<0>, <2>) , sum(<2>, V1, <3>) , div(V1, <2>, <0>)      % 19
?- nat(<1>) , sum(<2>, V1, <3>) , div(V1, <2>, <0>)          % 14
?- nat(<0>) , sum(<2>, V1, <3>) , div(V1, <2>, <0>)          % 2
?- sum(<2>, V1, <3>) , div(V1, <2>, <0>).                    % 1
?-* sum(<0>, V1, <1>) , div(V1, <2>, <0>).                  % 4
{V1=<1>} ?- nat(<1>) , div(<1>, <2>, <0>).                    % 3
?- nat(<0>) , div(<1>, <2>, <0>).                             % 2
?- div(<1>, <2>, <0>).                                        % 1
?- lt(<1>, <2>).                                             % 18
?- lt(<0>, <1>).                                             % 15
?- nat(<0>).                                                 % 14
?- .                                                         % 1
```