PROGRAMMIERUNG

ÜBUNG 8: LOGIKPROGRAMMIERUNG MIT PROLOG^{-}

Eric Kunze

eric.kunze@tu-dresden.de

INHALT

Logikprogrammierung und Prolog

EINFÜHRUNG IN PROLOG

- Französisch: programmation en logique
  (deutsch: Programmieren in Logik)
- hier: Teilsprache Prolog^{-}
- **Interpreter:** *swipl*

  `https://www.swi-prolog.org/download/stable`
  - Nutzung wie üblich im Terminal
  - `swipl <filename>` startet die interaktive Session
- **Online-Editor & Interpreter:** `https://swish.swi-prolog.org/`

EINFÜHRUNG IN PROLOG

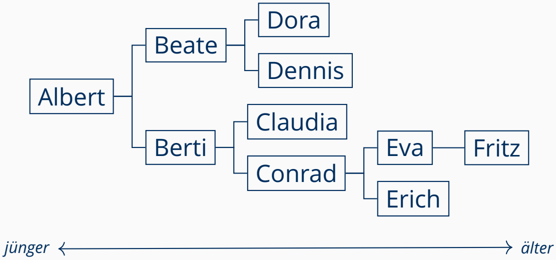
- Französisch: programmation en logique
  (deutsch: Programmieren in Logik)
- hier: Teilsprache Prolog$^-$
- **Interpreter:** *swipl*

  `https://www.swi-prolog.org/download/stable`
  - Nutzung wie üblich im Terminal
  - `swipl <filename>` startet die interaktive Session
- **Online-Editor & Interpreter:** `https://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 Verwanschaftsbeziehungen 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).`
- z.B. Claudia ist ein Elternteil von Berti $\rightsquigarrow$ `parent(claudia, berti).`

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

ARBEITEN MIT SWIPL — ANFRAGEN

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 `;`

SLD-REFUTATIONEN

Ziel: zeige Gültigkeit einer Anfrage (eines Goals) $G = (\texttt{?-}\ \ L_1, \ldots, L_n)$

SLD-Resolution:

- wähle ein L_i aus
- es gibt eine Regel $C = (M_0 \texttt{:-}\ \ M_1, \ldots, M_m)$,
  wobei C und G keine gemeinsamen Variablen haben
- σ sei der allgemeinste Unifikator von L_i und M_0

Dann: ersetze L_i durch $M_1, \ldots, M_m$ unter Anwendung von σ — formal:

$$G' = \Big(\texttt{?-}\ \ \tilde{\sigma}(L_1), \ldots \tilde{\sigma}(L_{i-1}), \tilde{\sigma}(M_1), \ldots, \tilde{\sigma}(M_m), \tilde{\sigma}(L_{i+1}), \ldots, \tilde{\sigma}(L_n)\Big)$$

G' heißt *Resolvente* von G und C unter σ.

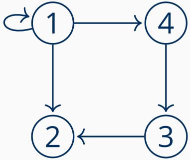
- **SLD-Ableitung** (derivation): Folge von SLD-Resolutionen
- **SLD-Refutation** (refutation): endliche Folge von SLD-Resolutionen
  mit dem leeren Goal `?-.` als Ende

Aufgabe 1

AUFGABE 1

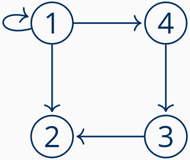
$$\begin{tikzcd}
1 \arrow[loop left] \arrow[r] \arrow[d] & 4 \arrow[d] \\
2 & 3 \arrow[l]
\end{tikzcd}$$

AUFGABE 1




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

AUFGABE 1




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

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

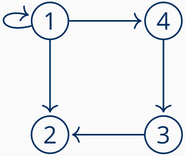
AUFGABE 1

Hinweis: Die Zeilenangaben in den Refutationen können von denen in der Übung abweichen.

```
        ?- 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)

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

sum(0, Y, Y) :- nat(Y).
sum(s(X), Y, s(S)) :- sum(X, Y, S).
```

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

AUFGABE 2 – TEIL (A)

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

sum(0, Y, Y) :- nat(Y).
sum(s(X), Y, s(S)) :- sum(X, Y, S).
```

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

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

AUFGABE 2 – TEIL (B)

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

sum(0, Y, Y) :- nat(Y).
sum(s(X), Y, s(S)) :- sum(X, Y, S).

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

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

AUFGABE 2 – TEIL (B)

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

sum(0, Y, Y) :- nat(Y).
sum(s(X), Y, s(S)) :- sum(X, Y, S).

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

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

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

AUFGABE 2 –TEIL (C)

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

gesucht: SLD-Refutation für `?- div(<3>, <1>).`

```
?- div(<3>, <1>).
?- div(<1>, 0).      % 12
?- .                 % 11
```

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

sum(0, Y, Y) :- nat(Y).
sum(s(X), Y, s(S)) :- sum(X, Y, S).
```

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

AUFGABE 2 – TEIL (D) *ZUSATZ*

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

sum(0, Y, Y) :- nat(Y).
sum(s(X), Y, s(S)) :- sum(X, Y, S).
```

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

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

AUFGABE 2 – TEIL (D) *ZUSATZ*

```prolog
nat(0).
nat(s(X)) :- nat(X).

sum(0, Y, Y) :- nat(Y).
sum(s(X), Y, s(S)) :- sum(X, Y, S).
```

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

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

```prolog
div(0, M, 0) :- lt(0, M).
div(N, M, 0) :- lt(N, M).
div(N, M, s(Q)) :- lt(0, M), sum(M, V, N),
                   div(V, M, Q).
```

AUFGABE 2 – TEIL (E) *ZUSATZ*

```
         ?- 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
```