

# Künstliche Intelligenz - Übung 1

Naja von Schmude (4127652)

24. April 2009

## Aufgabe 1

a)

Es ist eine Familiendatenbank über mind. drei Generationen anzugeben. Als berühmte Dynastie wurde die Simpsons-Familie gewählt. Zunächst müssen die Geschlechter definiert werden:

```
male(bart).  
male(homer).  
male(abraham).  
male(clancy).
```

```
female(maggie).  
female(lisa).  
female(marge).  
female(selma).  
female(patty).  
female(jacqueline).  
female(mona).
```

Dann werden die Vater bzw. Mutterbeziehungen definiert, wobei **father(X,Y)** bedeutet, dass X Vater von Y ist. Analog gilt das für **mother(X,Y)**.

```
father(homer,bart).  
father(homer,lisa).  
father(homer,maggie).  
father(abraham,homer).  
father(clancy,marge).  
father(clancy,selma).  
father(clancy,patty).
```

```
mother(marge,bart).  
mother(marge,lisa).  
mother(marge,maggie).  
mother(mona,homer).  
mother(jacqueline,marge).  
mother(jacqueline,selma).  
mother(jacqueline,patty).
```

b)

Nun sind Verwandtschaftsbeziehungen zu definieren.

```
parent(X,Y) :- father(X,Y) ; mother(X,Y). /* X ist Elternteil von Y */
son(X,Y) :- parent(Y,X) , male(X). /* X ist Sohn von Y */
daughter(X,Y) :- parent(Y,X) , female(X). /* X ist Tochter von Y */
brother(X,Y) :- father(F,X) , mother(M,X) ,
                mother(M,Y) , father(F,Y) , male(X) , X \= Y. /* X ist Bruder von Y */
sister(X,Y) :- father(F,X) , mother(M,X) ,
                mother(M,Y) , father(F,Y) , female(X) , X \= Y. /* X ist Schwester von Y */
grandfather(X,Y) :- parent(X,Z) , parent(Z,Y) , male(X). /* X ist Opa von Y */
grandmother(X,Y) :- parent(X,Z) , parent(Z,Y) , female(X). /* X ist Oma von Y */
```

X ist also Elternteil von Y, wenn es Vater oder Mutter ist. X ist der Sohn von Y, wenn Y Elternteil ist und X männlich. Analog funktioniert natürlich auch Tochter.

X ist Bruder von Y, wenn jeweils die Eltern gleich sind, X nicht Y ist (man ist ja nicht mit sich selbst Geschwister) und X männlich ist.

X ist Großvater, wenn X ein Kind hat, dass Elternteil von Y ist und X entsprechend männlich ist.

### Testlauf

```
?- parent(X,bart).
X = homer ;
X = marge ;
false.
?- son(bart,X).
X = homer ;
X = marge ;
false.
?- sister(marge,X).
X = selma ;
X = patty ;
false.
?- grandfather(X,Y).
X = abraham,
Y = bart ;
X = abraham,
Y = lisa ;
X = abraham,
Y = maggie ;
X = clancy,
Y = bart ;
X = clancy,
Y = lisa ;
X = clancy,
Y = maggie ;
false.
```

c)

Es sind die Vorfahre und Nachfahrebeziehungen anzugeben.

Dabei ist eine Person X Vorfahre einer anderen Y, wenn sie entweder Elternteil ist, oder das Elternteil der Person Y X als Vorfahren hat. Analog funktioniert das für den Nachfahren.

```
predecessor(X,Y) :- parent(X,Y) ; predecessor(Z,Y) , parent(X,Z). /* X ist Vorfahre von Y */
successor(X,Y) :- parent(Y,X) ; successor(Z,Y) , parent(Z,X). /* X ist Nachfahre von Y */
```

### Testlauf

```
?- predecessor(X, maggie).
X = homer ;
X = marge ;
X = abraham ;
X = mona ;
X = clancy ;
X = jacqueline ;
false.
```

## Aufgabe 2

a)

Das letzte Element einer Liste L soll ausgespuckt werden.

```
last(Item, [H]) :- write(H).
last(Item, [H|T]) :- last(Item, T) , !.
```

### Testlauf

```
?- last(Item,[1,3,4,2,a,c,3,4,5]).
5
true.
```

b)

c)

Die Summe der Liste soll zurückgegeben werden.

```
sumList([],0).
sumList([H], H).
sumList([H|T], Sum) :- sumList(T, Sum1) , Sum is H + Sum1 , !.
```

### Testlauf

```
?- sumList([4,5,6,3,2,1],H).
H = 21.
```

## Aufgabe 3

a)

Zunächst wird der Graph geeignet übertragen. Dazu werden die Kanten als Klauseln der Form `kante(x,y)` definiert, wenn eine ungerichtete Kante von  $x$  nach  $y$  verläuft. Damit nicht doppelte Schreibarbeit geleistet werden muss, in dem die Tupel  $(y,x)$  auch noch eingefügt werden, wird ein Weg als folgendes Konstrukt definiert: `weg(A,B) :- kante(A,B) ; kante (B,A)`.

Nun sollen ja alle Wege zwischen zwei Knoten berechnet werden:

```
alleWege(X,X,L).
```

```
alleWege(X,Y,L) :- weg(X,A) , not(member(A,L)) , alleWege(A,Y,[A|L]) , write(A), write(' , ')
```

**Testlauf** Die Ausgabe ist nicht besonders schön, aber besser als nix ;-)

Dabei ist der Weg vom Ziel an zum Start angegeben. Die Knoten in der Mittel werden wegen der Rekursion verdoppelt, einfach ignorieren ...

```
?- alleWege(a,d,[a]).
```

```
d,bb,a
```

```
true ;
```

```
d,cc,ee,bb,a
```

```
true ;
```

```
d,ee,bb,a
```

```
true ;
```

```
d,cc,a
```

```
true ;
```

```
d,bb,ee,cc,a
```

```
true ;
```

```
d,ee,cc,a
```

```
true ;
```

```
false.
```