Künstliche Intelligenz - Übung 9

Julian Dobmann, Kai Kruschel

Aufgabe 1 Tiefensuche/Breitensuche

a)

```
% kinder(X,L)
kinder(X,L) :-
    findall(P, move(X,P),L).
```

b)

Die Idee war es, eine bestehende Breitensuche-Implentierung so zu verändern/erweitern, dass anstatt eines Lösungspfades die Anzahl der besuchten(expandierten) Knoten gezählt wird. Als Vorlage wurde die Breitensuche-Implementierung von http://www.hsg-kl.de/faecher/inf/material/prolog/graphen/breit/index.php verwendet.

```
% bfs(S,Z,N) ermuttelt die Anzahl N der bei einer erfolgreichen Suche ex
pandierten
% Knoten bei einer Breitensuche im Graphen
breitensuche(S,Z,N) :- bfs(S,Z,N).

% bfs http://www.hsg-kl.de/faecher/inf/material/prolog/graphen/breit/ind
ex.php
% bfs/3
bfs(Start,Ziel,Count) :-
    bfs([Start],[],Ziel,Count).

% bfs/4
% Anker: Wenn ein Ziel gefunden wurde, dann wird Count auf 0 gesetzt
bfs(Pfad,_,Ziel,Count) :-
    Pfad=[Ziel|_],Count=0,!.

bfs(Pfad,Pfade,Ziel,Count) :-
```

```
bfs(Pfad,Pfade,Ziel,Count) :-
    Pfad=[KnotenA|_],
    findall(
        [KnotenN|Pfad],
        (con(KnotenA,KnotenN),not(member(KnotenN,Pfad))),
        GefundenePfade),
    append(Pfade,GefundenePfade,NeuePfade),
    NeuePfade=[PfadN|RestPfade],
    length(NeuePfade, Len),
    Var is (Count - Len),
    bfs(PfadN,RestPfade,Ziel,Var).
```

Leider schlägt diese unter Verwendung der Test-Fakten in `breitensuche.pl` Lösung mit dem Fehler

```
?- breitensuche(a,b,X).
ERROR: is/2: Arguments are not sufficiently instantiated
```

fehl, was wohl bedeutet, dass eine der beiden Seiten des Ausdrucks

```
Var is (Count - Len)
```

noch nicht instantiiert worden ist und eine Unifikation daher nicht durchgeführt werden kann. Wie man das Problem beheben kann ist uns leider nicht klar.

b)

Die Tiefensuche lässt sich leicht aus der Breitensuche ableiten, es muss lediglich die Reihung der übrigen zu prüfenden Knoten verändert werden.

Konkret heißt das bei unserer Lösung, dass Zeile 21

```
    append(Pfade,GefundenePfade,NeuePfade),
```

verändert wird zu

```
    append(GefundenePfade,Pfade,NeuePfade),
```

wodurch das Kopf-Element `PfadN` der Liste `NeuePfade` das erste Element der Liste `GefundenePfade` ist, also die neu gefundenen Pfade zuerst weiterverfolgt werden, wie das bei einer Tiefensuche halt so ist.

Aufgabe 2 A*

a)

Das Prädikat `heuristik\3` prüft elementweise die Gleichheit der Elemente der ihm übergebenen Listen aus Kacheln.

```
% heuristik ermittelt zum Zustand X die Anzahl N der nicht mit dem
% Zielzustand Z übereinstimmenden Zahlenfelder
% (implizite Bedingung: X.size == Z.size)
% es geht dabei um das 3x3 Schiebepuzzle
% ein Zustand hat dabei z.B. die Form [1,2,3,4,5,6,7,8,b]

% der Abstand zweier ein-elementiger Listen ist 0, wenn sie das gleiche
% Element enthalten, und 1 wenn sie unterschiedliche Elemente enthalten
heuristik([X],[Z],N) :-
    (X == Z),!, N is 0;
    (X \= Z), N is 1.
% der Abstand zweier mehr-elementiger Listen mit gleichem Head ist
% 0 + Abstand der Restlisten
% der Absatnd zweier mehr-elementiger Listen mit verschiedenem Head
% ist 1 + Abstand der Restlisten
heuristik([Xh|X],[Zh|Z],N) :-
    (Xh == Zh),!, heuristik(X,Z,N);
    (Xh \= Zh), heuristik(X,Z,N-1).
```

Beispiele:

```
?- heuristik([1,2,3,4,5,6,7,8,b],[1,2,3,4,5,6,7,8,b],N).
N = 0.

?- heuristik([1,2,3,4,5,6,7,8,b],[1,2,3,4,5,6,7,b,8],N).
N = 2 ;
false.

?- heuristik([1,2,3,4,5,6,7,8,b],[1,2,4,3,5,6,7,b,8],N).
N = 4 ;
false.
```