Hinweise zu den Aufgaben im KI-Buch (6. Auflage)

Prof. Dr. Jürgen Cleve

Auf vielfachen Wunsch werden hier einige Hinweise zu den Aufgaben der Kapitel 2-4 gegeben. Sollten Sie Hinweise vermissen oder trotz der Hinweise *nicht* mit Übungsaufgaben zurecht kommen, senden Sie uns bitte eine Email.

Inhaltsverzeichnis

1 Einführung **2**

2 Wissensverarbeitung **2**
2.1 Logik . . . 2
2.2 Regeln . . . 6
2.3 Semantische Netze und Frames . . . 7
2.4 Vages Wissen . . . 7

3 Suche **7**

4 PROLOG **11**
4.1 Logisches Programmieren . . . 11
4.2 Prolog-Programme . . . 12
4.3 Datentypen und Arithmetik . . . 12
4.4 Steuerung . . . 12
4.5 Vordefinierte Prädikate . . . 12
4.6 Beispielprogramme . . . 12

1 Einführung

2 Wissensverarbeitung

2.1 Logik

Übung 2.1

Die Aufgabe ist, $A \wedge B$ äquivalent mit Hilfe von $\rightarrow$ auszudrücken. Man soll also das $\wedge$ ersetzen und darf dabei $\rightarrow$ verwenden. Hier ist wichtig, dass wir die Äquivalenz:
$A \rightarrow B \equiv \neg A \vee B$
im Blick haben. Ersetzen wir B durch $\neg B$, dann haben wir schon:
$A \rightarrow \neg B \equiv \neg A \vee \neg B$
Jetzt müssen wir nur noch negieren:
$\neg(A \rightarrow \neg B) \equiv \neg(\neg A \vee \neg B) \equiv A \wedge B$

Übung 2.2

Analog soll $\leftrightarrow$ ersetzt werden. Dies geht allein mit $\wedge$ und $\rightarrow$.
$A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$

Übung 2.3

$(A \vee \neg(B \wedge A)) \wedge (C \vee (D \vee C))$
$(A \vee (\neg B \vee \neg A)) \wedge (C \vee D \vee C)$
$(A \vee \neg B \vee \neg A) \wedge (C \vee D)$
$(C \vee D)$

Übung 2.4

$\neg((A \wedge B) \vee (C \wedge \neg D))$
$\neg(A \wedge B) \quad \wedge \quad \neg(C \wedge \neg D)$
$(\neg A \vee \neg B) \quad \wedge \quad (\neg C \vee D)$

Übung 2.5

$(B \rightarrow (A \vee C) \wedge (B \vee C) \wedge (A \rightarrow C) \wedge \neg(\neg B \wedge C)$
$(\neg B \vee A \vee C) \wedge (B \vee C) \wedge (\neg A \vee C) \wedge (B \vee \neg C)$

Klauseln:

1. $\neg B \vee A \vee C$
2. $B \vee C$
3. $\neg A \vee C$
4. $B \vee \neg C$
5. $\neg B$

6 (2+4) B

7 (6+5) Widerspruch

Übung 2.6

Es ist zu beweisen, dass folgende Formel falsch ist:
$((A \leftrightarrow (B \rightarrow C)) \wedge (A \rightarrow B) \wedge (A \leftrightarrow \neg C))$

Hier können wir direkt Resolution verwenden. Wir benötigen also kein Ziel, welches wir negieren. Erinnern Sie sich bitte daran, dass Resolution den Widerspruch in einer Formel (oder Formelmenge) **immer** aufdeckt. Und die Aufgabe ist ja hier zu zeigen, dass eine Formel falsch ist, dass die Formel also in sich widersprüchlich ist.

Wir müssen also nur diese Formel nehmen, in KNF umwandeln und auf die Klauseln Resolution anwenden.

Umwandlung in KNF:

$(A \rightarrow (B \rightarrow C)) \wedge ((B \rightarrow C) \rightarrow A) \wedge (A \rightarrow B) \wedge (A \rightarrow \neg C) \wedge (\neg C \rightarrow A)$
$(\neg A \vee (\neg B \vee C)) \wedge (\neg(\neg B \vee C) \vee A) \wedge (\neg A \vee B) \wedge (\neg A \vee \neg C) \wedge (C \vee A)$
$(\neg A \vee \neg B \vee C) \wedge (B \wedge \neg C) \vee A) \wedge (\neg A \vee B) \wedge (\neg A \vee \neg C) \wedge (C \vee A)$

Klauseln:

1. $\neg A \vee \neg B \vee C$
2. $B \vee A$
3. $\neg C \vee A$
4. $\neg A \vee B$
5. $\neg A \vee \neg C$
6. $C \vee A$

Resolution:

7 (3+6) A

8 (4+7) B

9 (1+7) $\neg B \vee C$

10 (9+8) C

11 (10+5) $\neg A$

12 (11+7) Widerspruch

Übung 2.7

Für die erste Aussage können Sie es mit dem Beweiser und dieser Beispieldatei: club.txt probieren.

Für die 2. Aussage braucht man Alltagswissen, welches in der Datei schon ergänzt ist. Versuchen Sie aber lieber zunächst, den Beweis allein zu führen.

Übung 2.8

Versuchen Sie zunächst, alle Aussagen in AL oder PL aufzuschreiben. Dann schreiben Sie bitte auf, was „wahr“ ist, also z.B. ist die Aussage von Lutz nicht wahr, er lügt. Was heißt: „Er lügt.“? Muss dann jede (Teil-)Aussage von ihm gelogen sein?

Was bedeutet die Aussage, dass genau einer der Täter war?

Und wie formuliert man die Aussage, dass der Dieb im Hotel war? War beispielsweise Lutz der Dieb, dann war er zwingend ....

Bitte achten Sie auch auf die Formulierung: „Alles, was Lutz gesagt hat, ist gelogen.“

Danach versuchen Sie den Beweis zu führen.

Hier die komplette Lösung für den Beweiser. Sie sollten schon einigermaßen fit in Prolog sein.

```
/* ---------------------------------- */
/* Beispiel PL1  Hoteldieb            */
/*                  JCl  04/2001      */
/* ---------------------------------- */
```

```
(dieb(b) & ~ dieb(j) & ~ dieb(l)) v
               (~ dieb(b) & dieb(j) & ~ dieb(l)) v
               (~ dieb(b) & ~ dieb(j) & dieb(l)).
all(x,(dieb(x) -> hotel(x))).
~ (~ dieb(l) & dieb(b) & ~ hotel(l)). % Lutz Aussage
~ (~ dieb(b) & dieb(l) & ~ dieb(b) & hotel(l) & ~ dieb(j)). % Bernd
~ (~ dieb(j) & hotel(l) & ~(dieb(l) & ~ dieb(b) & hotel(l))). % Jochen

goal dieb(j).
```

Übung 2.9

A	B	$A \rightarrow B$	$\neg A \vee B$
F	F	W	W
F	W	W	W
W	F	F	F
W	W	W	W

Übung 2.10

Gegeben sind die Klauseln:

1. $\{A, B\}$
2. $\{\neg A, \neg B\}$

Folgende Resolventen sind möglich:

3 (1+2) $\{\neg A, A\}$

4 (1+2) $\{\neg B, B\}$

Weitere Resolutionsschritte führen immer zu Klauseln, die wir schon haben.
Zusätzlich fällt einem auf, dass die Klauseln 3 und 4 Tautologien sind. Sie können also weggelassen werden. Also ergibt sich bei dieser Aufgabe keine neue Klausel.

Übung 2.11

- Alle Informatiker können programmieren. $\forall x\ informatiker(x) \rightarrow kann\_programmieren(x)$
- Jeder Bundesbürger verhält sich einer Partei gegenüber loyal. $\forall x\ buerger(x) \rightarrow (\exists y\ partei(y) \wedge loyal(x, y))$
- Jede Firma zahlt Steuern. $\forall x\ firma(x) \rightarrow zahlt\_steuern(x)$
- Jede GmbH ist eine Firma. $\forall x\ gmbh(x) \rightarrow firma(x)$

Übung 2.12

In dieser Aufgabe sind ein paar „Fallen“ versteckt.
Stichworte: **Existenzaussagen, Skolemisierung** und die Eigenschaft `schneller`.
Wir Menschen kennen eine Eigenschaft von `schneller`, die wir (um einen Beweis formal führen zu können) erst mal als logische Regel aufschreiben müssen.
Führen Sie bitte den Beweis in 2 Stufen, zunächst mit gesundem Menschenverstand, und danach allein mit **Resolution** (also ohne gesunden Menschenverstand). Wir wollen, dass eine Maschine den Beweis findet !
Hier die komplette Lösung für das Beweiser-Programm. Sie sollten schon einigermaßen fit in Prolog sein.

```
/* ---------------------------------- */
/* Beispiel PL1  Tiere                */
/*                  JCl  04/2001      */
/* ---------------------------------- */
all(x,all(y,all(z,schneller(x,y) & schneller(y,z) -> schneller(x,z)))).
all(x,all(y,pferd(x) & hund(y) -> schneller(x,y))).
ex(z,windhund(z) & all(x1,hase(x1) -> schneller(z,x1))).
pferd(fury).
hase(bunny).
all(x,windhund(x) -> hund(x)).
goal schneller(fury,bunny).
```

Übung 2.13

Wie kann man die Äquivalenz von 2 Aussagen beweisen? Was heißt, 2 Aussagen sind äquivalent? Kann man das als aussagenlogischen Zusammenhang schreiben? Und wie beweist man dies dann mit Resolution?

Dass 2 Aussagen (nehmen wir A und B) äquivalent sind, können wir mit der logischen Äquivalenz formulieren:

$A \leftrightarrow B$

Und wie beiweisen wir das mit Resolution? Ganz einfach, wir wollen ja beweisen, dass diese Aussage wahr ist. Also einfach negieren:

$\neg(A \leftrightarrow B)$

und dann den Resolutionsbeweis starten.

Übung 2.14

Führen Sie auch hier bitte den Beweis in 2 Stufen, zunächst mit gesundem Menschenverstand, und danach allein mit **Resolution** (also ohne gesunden Menschenverstand). Wir wollen, dass eine Maschine den Beweis findet.

Hier der Quelltext für den Beweiser:

```
cplus(e) v java(e) v python(e).
cplus(p) v java(p) v python(p).
cplus(f) v java(f) v python(f).
cplus(e) -> ~cplus(p) & ~cplus(f).
cplus(p) -> ~cplus(e) & ~cplus(f).
cplus(f) -> ~cplus(e) & ~cplus(p).
java(e) -> ~java(p) & ~java(f).
java(p) -> ~java(e) & ~java(f).
java(f) -> ~java(e) & ~java(p).
python(e) -> ~python(p) & ~python(f).
python(p) -> ~python(e) & ~python(f).
python(f) -> ~python(e) & ~pascal(p).

all(x,cplus(x) -> ~ex(y,geschwister(x,y))).
geschwister(e,oschwester).
~cplus(f).
~java(f).

goal cplus(p).
```

Übung 2.16

Nehmen Sie bei dieser Aufgaben einfach an, der Täter hätte einen Komplizen gehabt. Negieren Sie diese Aussage und versuchen Sie dann mittels Resolution zum Widerspruch zu kommen.

Denken Sie bitte bei dieser Aufgabe darüber nach, was es bedeutet, wenn man den Widerspruch **nicht** findet.

2.2 Regeln

Übung 2.21

Diese Aufgaben sollten Sie erst angehen, wenn Sie Kapitel 4 bearbeitet haben.
Sie sollen hier darüber nachdenken, wie Abarbeitung in Prolog erfolgt. Betrachten Sie dazu eine Prolog-Regel der Form:

```
a :- b,c,d.
```

Kann man eine solche Regel auch als Regel im Sinne einer Aktionsregel auffassen?

Übung 2.24

Entwickeln Sie Aktions-Regeln der Form:

Wenn im 3-Liter-Gefäß 1 Liter enthalten ist, dann fülle das Gefäß auf.

Können Sie mit Ihren Regeln das Problem lösen?

Übung 2.25

analog 2.24

Übung 2.26

analog 2.24

Übung 2.27

Geben Sie nicht gleich auf. Man findet die Lösung mit gesundem Menschenverstand nicht unbedingt sofort.
Stellen Sie also Regeln auf, und lassen Sie den Computer die Arbeit machen.
Sie können dazu das Regelverabeitungssystem nutzen.
Wenn Sie die Lösung sehen, fällt Ihnen bestimmt auf, dass die Lösung eigentlich sehr einleuchtend ist.
Bei dieser Aufgabe müssen Sie irgendwie darstellen, wieviel Zeit die Personen benötigen:

```
zeit(anna,5).
zeit(boris,10).
zeit(clara,20).
zeit(dirk,25).
```

Wir müssen aufpassen, dass die Zeit nicht überschritten wird.
Hier eine kleine Hilfe, eine erste Regel für das Laufen von links nach rechts:

```
rueber(X,Y,Z) : [links(lampe),links(X),links(Y),X\=Y,zeit(T),T<60]
                 =>
                [zeit(X,XM),zeit(Y,YM),
                    maximum(XM,YM, M),  Z is T+M, Z=<60,
                 loesche(zeit(T)),setze(zeit(Z)),
                 loesche(links(X)),loesche(links(Y)),
                 setze(rechts(X)),setze(rechts(Y)),
                 loesche(links(lampe)),setze(rechts(lampe))].
```

2.3 Semantische Netze und Frames

2.4 Vages Wissen

Wir hoffen, diese Aufgaben sind selbsterklärend und allein lösbar.

Übung 2.38

Wie groß ist die Wahrscheinlichkeit, dass das Gerät kaputt geht?

$$P(\text{kaputt}) = 0.4 * 0.05 + 0.5 * 0.02 + 0.1 * 0.01$$

Und wie gross ist die Wahrscheinlichkeit, dass Marianne A wählt UND dass es kaputt geht?

$$P(M\_A \ \wedge \ \text{kaputt}) = 0.4 * 0.05$$

Folglich hat man eine Wahrscheinlichkeit von $\frac{0.02}{0.0301}$.

Übung 2.43

Am besten macht man sich eine Skizze, was alles passieren kann, wenn man stückweise lineare Funktionen miteinander verknüpft.

3 Suche

Übung 3.8

Im Folgenden beziehen wir uns in unserer Notation auf PROLOG und unser Suchprogramm.
Bitte probieren Sie es zunächst allein! Wenn Sie nicht zurecht kommen, dann lesen Sie bitte die Lösung nach und nach !
Zunächst muss man sich über die Datenstrukturen Gedanken machen. Wie stellt man das Schiebefax dar ? Am besten durch eine Matrix:

```
    *********       Diese Position kann dargestellt werden durch:
  3 * 1 2 3 *
  2 * 8   4 *       [2/2, 1/3, 2/3, 3/3, 3/2, 3/1, 2/1, 1/1, 1/2]
  1 * 7 6 5 *
    *********
      1 2 3
```

Was sind zulässige Zustandsübergänge? Die Position des Leerfeldes steht vorn. Wir müssen ein passendes Feld finden. Dies muss bez. der Manhatten-Distanz exakt 1 entfernt sein:

```
zustandsuebergang(Alt,Neu) :- s(Alt,Neu,_).

% s( Node, SuccessorNode, Cost) ... s vereinigt Zustandsuebergang und Kosten
s([Empty | Tiles], [Tile | Tiles1], 1):-    % Alle Kosten sind 1.
    tausche( Empty, Tile, Tiles, Tiles1).   % Tausche Leerfeld und ein passendes

tausche(Empty, Tile, [Tile | Ts], [Empty | Ts]) :- manhatten( Empty, Tile, 1).
tausche(Empty, Tile, [T1 | Ts], [T1 | Ts1]):- tausche( Empty, Tile, Ts, Ts1).

manhatten(X/Y, X1/Y1, D):-
      dif(X, X1, Dx),  dif(Y, Y1, Dy),
      D is Dx + Dy.
dif(A, B, D):- D is A-B, D >= 0,!    ;     D is B-A.
```

Die Kosten haben wir damit auch gleich berechnet, jeder Zug kostet 1.
Wie kann man nun die Entfernung zum Ziel berechnen? Hier bietet sich zB an, für alle Felder zu berechnen, wie weit sie von ihrem Zielplatz entfernt sind. Diese Entfernung summiert man.

```
dist_goal(S,D) :- h(S,D).

h([Empty | Tiles], D):-
      ziel_erreicht( [Empty1 | GoalSquares]),
      totdist(Tiles, GoalSquares, D).

totdist([], [], 0).
totdist([Tile | Tiles], [Square | Squares], D):-
      manhatten( Tile, Square, D1),
      totdist( Tiles, Squares, D2),
      D is D1 + D2.
```

Übung 3.11

Im Folgenden beziehen wir uns in unserer Notation auf PROLOG und unser Suchprogramm.
Bitte probieren Sie es zunächst allein! Wenn Sie nicht zurecht kommen, dann lesen Sie bitte die Lösung nach und nach !
Zunächst muss man sich über die Datenstrukturen Gedanken machen. Wie stellt man den Kaefig dar ? Man braucht nur die existierenden Türen:

```
tuer(loewe,esel).
tuer(esel,wolf).
tuer(wolf,krokodil).
tuer(krokodil,panther).
tuer(loewe,aussen).
tuer(esel,aussen).
tuer(wolf,aussen).
tuer(krokodil,aussen).
tuer(panther,aussen).
```

Durch `kaefig(loewe,panther)` beschreibt man, dass der Panther im Löwe-Käfig ist.
Wodurch sind Start-/Zielzustand beschrieben?

```
startzustand([kaefig(loewe,panther), kaefig(esel,krokodil),
                kaefig(wolf,esel), kaefig(krokodil,loewe),
                kaefig(panther,wolf),kaefig(aussen,leer)]).
ziel_erreicht(Z) :- member(kaefig(X,Y),Z),X\=Y,X\=aussen,!,fail.
ziel_erreicht(_).
```

Wie sieht ein zulässiger Zustandsübergang aus?

```
zustandsuebergang(Alt,Neu) :-
        member(kaefig(Y,leer),Alt),
        (tuer(X,Y);tuer(Y,X)),
        member(kaefig(X,T),Alt),
        replace(kaefig(X,T),kaefig(X,leer),Alt,N1),
        replace(kaefig(Y,leer),kaefig(Y,T),N1,Neu).

replace(X,Y,[X|T],[Y|T]) :- !.
replace(X,Y,[F|T],[F|T1]) :- replace(X,Y,T,T1).
```

Die Kosten für ein Umsetzen: `costs(_,_,1).`.
Für die geschätzte Entfernung zum Ziel gibt es mehrere Varianten. Hier die Variante, wo man einfach die falsch besetzten Käfige zählt.

```
dist_goal(Zustand,M) :- zaehle_falsche(Zustand,0,M).

zaehle_falsche([],N,N) :- !.
```

```
zaehle_falsche([kaefig(X,X)|T],N,N1) :- !,
        zaehle_falsche(T,N,N1).
zaehle_falsche([kaefig(aussen,leer)|T],N,N1) :-
        zaehle_falsche(T,N,N1).
zaehle_falsche([_|T],N,N1) :- !,N2 is N+1,
        zaehle_falsche(T,N2,N1).
```

Hier noch ein Prädikat zum Anzeigen der Käfige:

```
zeige_kandidat([]) :- !.
zeige_kandidat([F|T]) :- zeige_kandidat(T),write(' * '),zeige_zustand(F).
zeige_zustand([kaefig(_,F)]) :- !,write(F).
zeige_zustand([kaefig(_,F)|T]) :- write(F),write('-'),zeige_zustand(T).
zustaende_gleich(F,F).
```

Übung 3.14

Im Folgenden beziehen wir uns in unserer Notation auf PROLOG und unser Suchprogramm.

Wir diskutieren dieses Problem am besten an einem Beispiel:

Jemand möchte einen Rucksack so bepacken, dass er einen höchstmöglichen Wert enthält. Zur Verfügung stehen einige Gegenstände. Jeder Gegenstand hat ein Gewicht. Jeder Gegenstand hat einen Wert.

Die Gegenstände spezifizieren wir in einer geeigneten Datenstruktur. Wir geben gleich ihr Gewicht und ihren Wert an:

```
gegenstand(maske,6,9).
gegenstand(kupfer,7,10).
gegenstand(amulett,5,14).
gegenstand(schnaps,8,11).
gegenstand(zigaretten,5,8).
gegenstand(schmetterlinge,3,8).
```

Zu Beginn haben wir keinen Gegenstand im Rucksack.

```
startzustand([]).
```

Der Rucksack verträgt ein Maximalgewicht von zB 20:

```
maximum(20).
```

Wie sieht ein zulässiger Zustandsübergang aus? Wir packen einen weiteren Gegenstand in den Rucksack:

```
zustandsuebergang(Alt,[G|Alt]) :-
        gegenstand(G,Gewicht,Wert),
        not member(G,Alt),
        kosten([G|Alt],Kosten),maximum(Max),Kosten =< Max.

kosten([],0) :- !.
kosten([F|Rest],S) :- kosten(Rest,S1),gegenstand(F,Kosten,_),!,S is Kosten+S1.
```

Die Kosten für das Hinzunehmen eines Gegenstands entsprechen exakt dem Gewicht:

```
costs(_,[G|_],Gewicht) :- gegenstand(G,Gewicht,_).
```

So weit so gut. Aber wie formulieren wir ein Ziel? Wir kennen das Maximum, was möglich ist, nicht. Also greifen wir zu einem Trick. Wir formulieren zunächst, dass wir uns umso näher am Ziel befinden, je größer der Wert ist. Da wir in den Suchproblemen immer Richtung Null wollten, müssen wir den Abstand zum Ziel zB wie folgt definieren:

```
dist_goal(Zustand,M) :-
        wert(Zustand,Wert),M is 100-Wert.

wert([],0) :- !.
wert([F|Rest],S) :- wert(Rest,S1),gegenstand(F,_,Wert),!,S is Wert+S1.
```

Und das Ziel? Wir kennen es nicht. Also lassen wir den Suchalg. einfach loslaufen, ohne dass er aufhören darf. Wir formulieren also ein unerfüllbares Ziel:

```
ziel_erreicht(hh).
```

Zum Schluss noch die Prädikate für das Anzeigen:

```
zeige_kandidat([]) :- !.
zeige_kandidat([F|T]) :- zeige_kandidat(T),write(' * '),zeige_zustand(F).
zeige_zustand([]) :- !.
zeige_zustand([F|T]) :- write(F),write('-'),zeige_zustand(T).
```

Ganz wichtig: Wann sind 2 Zustände gleich? Wenn im Rucksack die gleichen Gegenstände sind: Der Inhalt muss also nur mengenmäßig gleich sein.

```
zustaende_gleich(T,T1) :- sub(T,T1), sub(T1,T).

sub(X,Rest) :- member(F,X),not(member(F,Rest)),!,fail.
sub(_,_).
```

Übung 3.15

Im Folgenden beziehen wir uns in unserer Notation auf PROLOG und unser Suchprogramm.
Es gibt mehrere Varianten, die Distanz zum Ziel zu definieren:

Variante 1
Suchen des linkesten weissen Steins, Summieren der Abstaende zwischen den Positionen der schwarzen Steine rechts von dem weissen Stein + Zaehlen des Abstandes der schwarzen Steine von ihren korrekten Positionen

```
dist_goal1(State,Dist):-
        positionweisslinks(State,Pos),
        addiere(State,Pos,Pos,0,Zahl),
        zaehleschwarz(State,X),
        Dist is X+Zahl.

positionweisslinks(State,Pos) :- member(w,State,Pos),!. /* Bestimme linken weissen Stein */

addiere(State,Pos,Vergl,Accu,Zahl) :-
        member(s,State,P1),
        P1 > Vergl,
        Diff is P1-Pos,
        Accu1 is Accu+Diff,!,
        addiere(State,Pos,P1,Accu1,Zahl).
addiere(State,Pos,Corr,Accu,Accu).

zaehleschwarz(State,X) :-
        member(s,State,N1),A is N1-1,
        member(s,State,N2),N2>N1,B is N2-2,
        member(s,State,N3),N3>N2,C is N3-3,
        X is A+B+C,!.
```

Variante 2
Anzahl der falschen Steine

```
dist_goal2(State,Dist):- vergleiche(State,[s,s,s,' ',w,w,w],0,Dist).

vergleiche([],_,N,N) :- !.
vergleiche([F|T],[F|T2],N,N1) :- !,vergleiche(T,T2,N,N1).
vergleiche([_|T],[_|T2],N,N1) :- N2 is N+1,vergleiche(T,T2,N2,N1).
```

Variante 3
Suchen des linkesten weissen Steins, Summieren der Abstaende zwischen den Positionen der schwarzen Steine rechts von dem weissen Stein

```
dist_goal3(State,Dist):-
        positionweisslinks(State,Pos),
        addiere(State,Pos,Pos,0,Dist).

positionweisslinks(State,Pos) :- member(w,State,Pos),!. /* Bestimme linken weissen Stein */

addiere(State,Pos,Vergl,Accu,Zahl) :-
        member(s,State,P1),
        P1 > Vergl,
        Diff is P1-Pos,
        Accu1 is Accu+Diff,!,
        addiere(State,Pos,P1,Accu1,Zahl).
addiere(State,Pos,Corr,Accu,Accu).
```

Variante 4
Zaehlen des Abstandes der schwarzen Steine von ihren korrekten Positionen

```
dist_goal4(State,Dist) :- zaehleschwarz(State,Dist).

zaehleschwarz(State,X) :-
        member(s,State,N1),A is N1-1,
        member(s,State,N2),N2>N1,B is N2-2,
        member(s,State,N3),N3>N2,C is N3-3,
        X is A+B+C,!.
```

4 PROLOG

4.1 Logisches Programmieren

Übung 4.2

`hat_kind` ist einstellig, bekommt also nur 1 Parameter mit. Dieser ist die Person, von der überprüft werden soll, ob sie ein Kind hat.

Übung 4.4

Ein Problem ist hier, das Geschlecht der Personen darzustellen, z.B. als Fakten:

```
mann(lutz).
mann(juergen).

frau(marianne).
frau(gerda).
```

Übung 4.5 und Übung 4.6

Schauen Sie sich nochmals `vorfahre_von` an, bevor Sie mit diesen Aufgaben beginnen.

Übung 4.7

Gehen Sie von bestehenden Datenbanken aus (in Prolog also eine Menge von Fakten), z.B. `vater_von` und `frau_von`.
Definieren Sie dann exemplarisch Operationen wie den Join, Union, Selection etc.

4.2 Prolog-Programme

Übung 4.9

Spielen Sie beliebig mit der Unifikation (=) und der Identität (==). Vergleichen Sie die Antworten, die Sie unter Verwendung von = bzw. von `is` bekommen.

4.3 Datentypen und Arithmetik

Übung 4.11

`?-quadr([1,2,4,3],X).` → `X = [1,4,16,9]`

Übung 4.12

`?-doppelt([1,2,1,2,4,3],X).` → `X = [1,2,4,3]`

Übung 4.13

Hier müssen Sie ein wenig mit den Streckenlängen rechnen (man kann sich den Test auf einen rechten Winkel dadurch sparen !).

```
strecke(punkt(X1,Y1),punkt(X2,Y2),Dist) :-
Dist is sqrt((X2-X1)*(X2-X1) + ...)
```

Übung 4.15

Idee für das Umdrehen: Führendes Abspalten, Restliste umdrehen, gemerktes führendes Element hinten anfügen.
Für das Prüfen, ob eine Liste eine gerade / ungerade Anzahl von Elementen enthält, muss man NICHT unbedingt zählen !

4.4 Steuerung

Übung 4.17

Rufen Sie einfach mal beide Prädikate mit einer Variablen auf: `?-element(X,[1,2,3,4]).`

Übung 4.18

Hier sollten Sie sich an die Idee beim `not` erinnern.

4.5 Vordefinierte Prädikate

Übung 4.19

Benutzen Sie hierzu die vordefinierten Prädikate wie `var, integer, atomic` etc.

4.6 Beispielprogramme

Übung 4.21

Sie müssen nicht viel ändern, man muss sich nur von der Dimension 8 lösen.