Fakultät Informatik/Mathematik

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Lehrstuhl für Künstliche Intelligenz / Theoretische Informatik

Praktikumsmitschrift

KÜNSTLICHE INTELLIGENZ

Mitschrift von

Falk-Jonatan Strube

Übung von

Prof. Dr. Boris Hollas

7. Juli 2017

INHALTSVERZEICHNIS

1 PRAKTIKUM 1: PRÄDIKATENLOGIK

1.1 AUFGABE 1

(a) Für alle x gilt: Wenn x eine Anfrage ist, gibt es ein y in der entsprechenden Antwort.
Formel ist WAHR.

(b) Für alle x existiert ein y für das gilt: Wenn x eine Anfrage ist, gibt es eine entsprechende Antwort mit x und y (ist äquivalent mit (a)).
Formel ist WAHR.

(c) Es existiert ein y bei dem für alle x gilt: Wenn x eine Anfrage ist, gibt es eine entsprechende Antwort mit x und y (Es gibt ein y, das für alle Anfragen die richtige Antwort ist).
Formel ist FALSCH.
Beweis:
Angenommen, es gibt ein y, so dass $\forall x\,(anfrage(x) \rightarrow antwort(x, y))$ wahr ist. Wegen $(abfahrt, 9:00) \in antwort$ muss dann $y = 9:00$ sein. Wegen $(ankunft, 12:00) \in antwort$ muss aber auch $y = 12:00$. Widerspruch!

1.2 AUFGABE 2

(a)
- F_1 ist WAHR:
  Für alle Menschen und Fächer gilt: Wenn es die Kombination Mensch-Fach gibt, ist dieser Mensch Student.
- F_2 ist FALSCH:
  Für alle Menschen und Fächer gilt: Die Kombination Mensch-Fach existiert und der Mensch ist Student.
- F_3 ist WAHR:
  Für alle Menschen existiert ein y für das gilt: Wenn es die Kombination Mensch-y(bspw. auch „Toastbrot“) gibt, ist dieser Mensch Student.

(b)
- F_1 ist FALSCH.
- F_2 ist FALSCH.
- F_3 ist WAHR:
  Es existiert ein ein Fach für jeden Menschen, das er nicht studiert.

- F_2 bedeutet nicht „Wer ein Fach studiert, ist ein Student“, weil man die Formel umstellen könnte, dass $\forall y\ (\forall\ F(x, y) \wedge \forall x\ S(x))$, was bedeuten würde, dass jeder Mensch Student ist.
- F_3 bedeutet nicht „Wer ein Fach studiert, ist ein Student“. Es sagt vielmehr „Es existiert ein ein y für jeden Menschen, das sagt, dass er nicht studiert“.

$\rightarrow$ in 2(a) sind wahre Voraussetzungen angenommen: Dabei sind F_1 und F_3 korrekter Weise wahr.
in 2(b) sind falsche Voraussetzungen angenommen: Dabei ist F_1 korrekter Weise falsch und F_3 fälschlicher Weise wahr.

1.3 AUFGABE 3

(a) Jeder Informatikstudent studiert Informatik.
$\forall x\ (I(x) \rightarrow F(x, inf))$

(b) Jeder Informatikstudent ist ein Student, aber nicht jeder Student ist ein Informatikstudent.
$\forall x\ \exists y\ ((I(x) \rightarrow S(x)) \wedge S(y) \wedge \neg I(y))$
Achtung: $\forall x\ \exists y\ ((I(x) \rightarrow S(x)) \wedge (S(y) \rightarrow \neg I(y)))$ geht nicht, da man nun für y „Stuhl“ einsetzen könnte und die Formel richtig wäre da die Prämisse der zweiten Teilformel dann falsch und damit die Konklusion wahr wäre.

(c) Jeder Student studiert ein Fach.
$\forall x\ \exists y\ (S(x) \rightarrow F(x, y))$

(d) Wer ein Fach studiert, ist ein Student.
$\forall x\ \forall y\ (F(x, y) \rightarrow S(x))$
Achtung: $\forall x\ \exists y\ (F(x, y) \rightarrow S(x))$ geht nicht, da man nun für y irgendetwas eingeben und die Prämisse wäre falsch, also die Konklusion richtig – was aber falsch ist.

(e) Es gibt kein Fach, das kein Student studiert.
$\forall y\ \exists x\ (F(x, y) \wedge S(x))$
Achtung: $\forall y\ \exists x\ (F(x, y) \rightarrow S(x))$ geht nicht, da man nun für x irgendetwas einsetzen kann und dafür bei einem Fach, dem kein Student zugeordnet ist, die Prämisse falsch ist und somit die Aussage fälschlicher Weise wahr.

1.4 AUFGABE 4

(a) Es gibt ein Haustier, das fliegen kann.
$\exists x\ (haustier(x) \wedge fliegt(x))$

(b) Katzen können nicht fliegen.
$\forall x\ (katze(x) \rightarrow \neg fliegt(x))$

(c) Wenn ein Haustier fliegen kann, dann ist es ein Vogel.
$\forall x\ (haustier(x) \wedge fliegt(x) \rightarrow vogel(x))$

(d) Kein Haustier ist sowohl Katze als auch Vogel.
$\forall x\ (haustier(x) \rightarrow \neg(katze(x) \wedge vogel(x)))$

2 PRAKTIKUM 2: PRÄDIKATENLOGIK, PROLOG

2.1 AUFGABE 5

$$\forall x\ \exists y\ P(x,y) \qquad \wedge$$
$$\forall x\ \forall y\ (P(x,y) \rightarrow \neg P(y,x)) \qquad \wedge$$
$$\forall x\ \forall y\ \forall z\ (P(x,y) \wedge P(y,z) \rightarrow P(x,z))$$

- $P()$ ist nicht kommutativ
- $P()$ ist transitiv

$P(x,y)$: x ist kleiner als y.
$P = \{(x,y) \mid x < y\}$
$\Rightarrow x = 1,\ y = 2,\ z = 3$

2.2 AUFGABE 6

x: Straße, y: Ort
$\forall x\ \forall y\ (regnet(x) \wedge strasseinOrt(y,x) \rightarrow nass(y))$

$ort(dresden) \wedge regnet(dresden)$
$strasseInOrt(hochschulstrasse, dresden)$

```
ort(dresden). % nicht nötig
strasse(hochschulstrasse).  % nicht nötig
strasseInOrt(hochschulstrasse,dresden).
regnet(dresden).

nass(Y) :- regnet(X), strasseInOrt(Y,X).
```

2.3 AUFGABE 7

Bedeutungen:

$terrorv(x)$	x ist eine terroristische Vereinigung
$terror(x)$	x ist ein Terrorist
$org(x)$	x ist eine Organisation
$mitgl(x,y)$	x ist Mitglied in Organisation y
$anf(x,y)$	x ist Anführer der Organisation y
$verkauf(x,y,z)$	x hat y an z verkauft
$waffe(x)$	x ist eine Waffe
$strafbar(x)$	x macht sich strafbar

Teil a

- $\forall x\ \forall y\ (anf(x, y) \land terror(x) \rightarrow terrorv(y))$
- $\forall x\ \forall y\ (mitgl(x, y) \land terrorv(y) \rightarrow terror(x))$
- $\exists x\ (anf(x, \text{IS}) \land terror(x))$
- $mitgl(\text{Abul-Hasan Al Muhajir}, \text{IS})$
- $\forall x\ \forall y\ \forall z(verkauf(x, y, z) \land waffe(y) \land terror(z) \rightarrow strafbar(x))$
- $verkauf(\text{Omar Abdul Al-Hassan}, \text{M16}, \text{Abul-Hasan Al Muhajir})$
- $waffe(\text{M16})$
- $mitgl(\text{August Meier}, \text{ADAC})$
- $org(\text{ADAC})$
- $verkauf(\text{Bernd Müller}, \text{Auto}, \text{August Meier})$

Teil b Prolog-Programm:

```
terrorv(Y) :- anf(X,Y), terror(X).
terror(X) :- mitgl(X,Y) , terrorv(Y).
terror("Terrorist").
org("ADAC").
mitgl("August Meier", "ADAC").
mitgl("Abul-Hasan Al Muhajir", "IS").
anf("Terrorist","IS").
strafbar(X) :- verkauf(X,Y,Z), waffe(Y), terror(Z).
verkauf("Omar Abdul Al-Hassan", "M16", "Abul-Hasan Al Muhajir").
verkauf("Bernd Müller", "Auto", "August Meier").
waffe("M16").
```

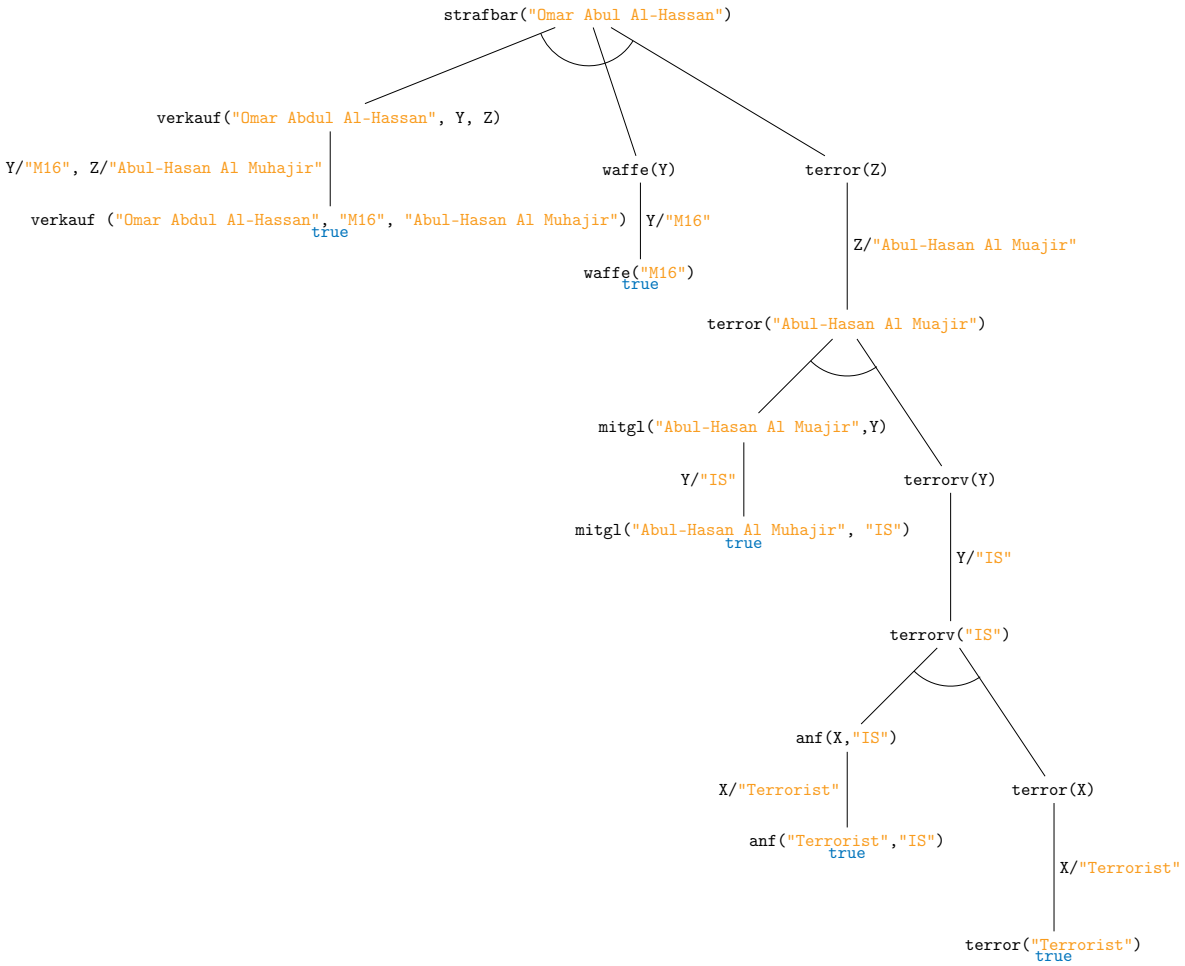
(i) Abfrage, ob Omar Abdul Al-Hassan strafbar:

```
?- strafbar("Omar Abdul Al-Hassan").
true.
```

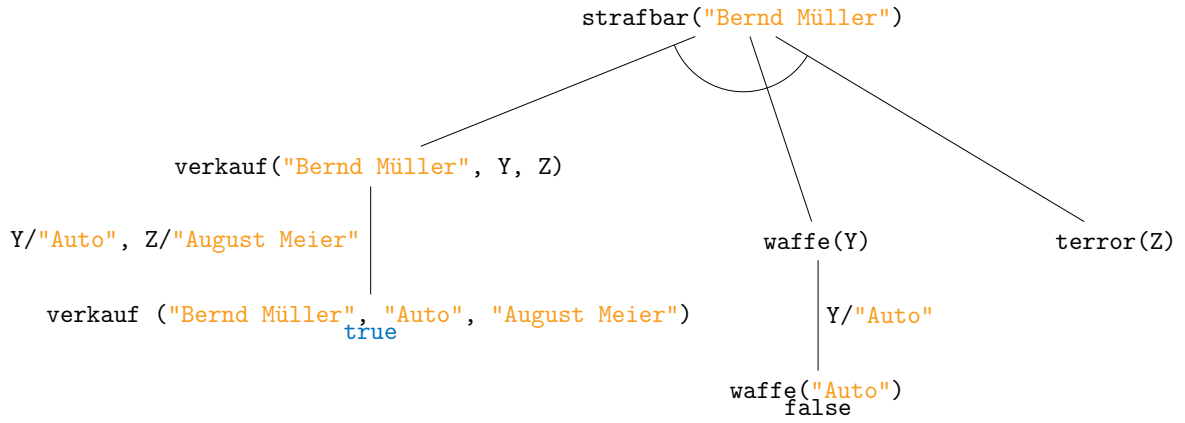
(ii) Abfrage, ob Bernd Müller strafbar:

```
?- strafbar("Bernd Müller").
false.
```

Teil c
Suchbaum i:


strafbar("Omar Abul Al-Hassan")
verkauf("Omar Abdul Al-Hassan", Y, Z)
Y/"M16", Z/"Abul-Hasan Al Muhajir"
verkauf ("Omar Abdul Al-Hassan", "M16", "Abul-Hasan Al Muhajir")
true
waffe(Y)
Y/"M16"
waffe("M16")
true
terror(Z)
Z/"Abul-Hasan Al Muajir"
terror("Abul-Hasan Al Muajir")
mitgl("Abul-Hasan Al Muajir",Y)
Y/"IS"
mitgl("Abul-Hasan Al Muhajir", "IS")
true
terrorv(Y)
Y/"IS"
terrorv("IS")
anf(X,"IS")
X/"Terrorist"
anf("Terrorist","IS")
true
terror(X)
X/"Terrorist"
terror("Terrorist")
true

Suchbaum ii:


strafbar("Bernd Müller")
verkauf("Bernd Müller", Y, Z)
Y/"Auto", Z/"August Meier"
verkauf ("Bernd Müller", "Auto", "August Meier")
true
waffe(Y)
Y/"Auto"
waffe("Auto")
false
terror(Z)

3 PRAKTIKUM 3: UNIFIZIERUNG, PROLOG

3.1 AUFGABE 8

$a, f(a) \ldots$ Konstanten $x, y \ldots$ Variablen

(a) $P(x, y, a),\ P(x, x, x)$ sind unifizierbar durch $x/a,\ y/a$

(b) $P(f(a), x, y),\ P(x, a, a)$ sind nicht unifizierbar, da x sowohl mit $f(a)$ und a ersetzt werden müsste (ersten beiden Positionen)

(c) $P(f(a), a, x),\ P(x, x, a)$ sind nicht unifizierbar, da x sowohl mit $f(a)$ und a ersetzt werden müsste (ersten beiden Positionen)

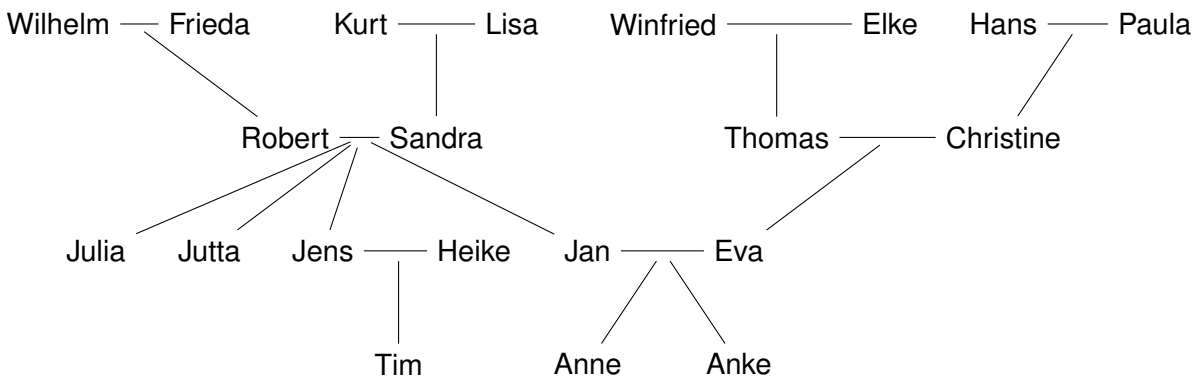
3.2 AUFGABE 9

```
%kind(name, geschlecht, mutter, vater).
kind(robert, mann, wilhelm, frieda).
kind(sandra, frau, kurt, lisa).
kind(thomas, mann, winfried, elke).
kind(christine, frau, hans, paula).
kind(eva, frau, thomas, christine).
kind(jan, mann, robert, sandra).
kind(jens, mann, robert, sandra).
kind(julia, frau, robert, sandra).
kind(jutta, frau, robert, sandra).
kind(anne, frau, jan, eva).
kind(anke, frau, jan, eva).
kind(tim, mann, jens, heike).

% X bruder von Y:
bruder(X,Y):- kind(X,mann,V,M) , kind(Y,_,V,M), not(X=Y).
% X schwester von Y:
schwester(X,Y):- kind(X,frau,V,M) , kind(Y,_,V,M), not(X=Y).
% X ist vorfahr von Y:
direkterVorfahr(X,Y):- kind(Y,_,X,_); kind(Y,_,_,X).
% X enkel von Y:
% eigene Lösung ohne direkterVorfahr: enkel(X,Y):- kind(X,_,V,M), (
   kind(V,_,Y,_) ; kind(M,_,Y,_) ; kind(V,_,_,Y) ; kind(M,_,_,Y)).
enkel(X,Y):- direkterVorfahr(Z, X), direkterVorfahr(Y, Z).
% X ist onkel von Y:
% eigene Lösung ohne direkterVorfahr: onkel(X,Y):- kind(Y,_,V,M) ,
   (bruder(X,V) ; bruder(X,M)) .
onkel(X,Y):- direkterVorfahr(Z,Y), bruder(X,Z).
% X ist onkel von Y:
% eigene Lösung ohne direkterVorfahr: tante(X,Y):- kind(Y,_,V,M) ,
   (schwester(X,V) ; schwester(X,M)) .
```

```
29 tante(X,Y):- direkterVorfahr(Z,Y), schwester(X,Z).
```

Stammbaum:



Beispiel-Abfragen:

```
1  ?- bruder(X, julia).
2  X = jan ;
3  X = jens .
4
5  ?- schwester(X,jens).
6  X = julia ;
7  X = jutta .
8
9  ?- bruder(jens, X).
10 X = jan ;
11 X = julia ;
12 X = jutta .
13
14 ?- enkel(X,robert).
15 X = anne ;
16 X = anke ;
17 X = tim .
18
19 ?- tante(X,tim).
20 X = julia ;
21 X = jutta.
22
23 ?- tante(X,tim).
24 X = julia ;
25 X = jutta . % es fehlt eva?!
```

4 PRAKTIKUM 4: TRANSITIVE HÜLLE, PROLOG

4.1 AUFGABE 11

Der Suchbaum würde sich in einer Rekursion aufhängen: beim erste `weg()`.
Lösung: Eine extra Funktion für die transitive Hülle einführen, die die Rekursion auf den rechten Zweig verschiebt, wo sie sich auflösen oder zu einem Fehler führen kann.

```
1 tweg(X,Y) :- weg(X,Y); weg(X,Z), tweg(Z,Y).
```

4.2 AUFGABE 12

–

5 PRAKTIKUM 5: VEREINFACHUNG PROLOG

5.1 AUFGABE 12

```
d(X,X,1).
d(C,X,0) :- atomic(C), C \== X.
d(-F,X,-DF) :- d(F,X,DF).
d(C*F,X,C*DF) :- d(C,X,0), d(F,X,DF).
d(F+G,X,DF+DG) :- d(F,X,DF), d(G,X,DG).
d(F-G,X,DF-DG) :- d(F,X,DF), d(G,X,DG).
d(F^N,X,N*F^M*DF) :- number(N), M is N-1, d(F,X,DF).


s(A*B,C) :- s(A,SA) , s(B,SB), s0(SA*SB,C).
s(A+B,C) :- s(A,SA) , s(B,SB), s0(SA+SB,C).
s(A,A).

s0(A*X+B*X,C*X):- s0(A+B,C).
s0(A*X*B,C*X):- s0(A*B,C).
s0(A*B,C) :-  number(A), number(B),C is A*B.
s0(A+B,C) :-  number(A), number(B),C is A+B.
s0(X*X,X^2).
s0(0+A,A).
s0(A+0,A).
s0(1*A,A).
s0(A*1,A).
s0(A,A).

diff(A,C) :- d(A,x,B), s(B,C).
```

5.2 AUFGABE 13

```
% Bilde Stammfunktion stammF(Funktion, Variable, Ergebnis)
% --------------------
% Teile Polynome auf:
stammF(F_a - F_b, X, StammF_a - StammF_b) :- stammF(F_a, X,
   StammF_a), stammF(F_b, X, StammF_b).
stammF(F_a + F_b, X, StammF_a + StammF_b) :- stammF(F_a, X,
   StammF_a), stammF(F_b, X, StammF_b).
% Bilde Stammfunktionen von Polynom:
% 0
stammF(0, _, 0).
stammF(0*X, X, 0).
% c
```

```
11 stammF(C, X, C*X) :- number(C).
12 % x
13 stammF(X, X, 0.5*X^2).
14 % c*x
15 stammF(C*X, X, C2*X^2) :- number(C), C2 is (C*0.5).
16 % x^n
17 stammF(X^N, X, C2*X^N2) :- N2 is (N+1), C2 is (1/N2).
18 % c*x^n
19 stammF(C*X^N, X, C2*X^N2) :- number(C), N2 is (N+1), C2 is (C*(1/N2
       )).
20 % --------------------
21 % Setze Grenze in Stammfunktion ein grenzStammF(Funktion, Variable,
       Grenze, Ergebnis)
22 % --------------------
23 % Teile Stammfunktionen der Polynome auf
24 grenzStammF(F_a + F_b, X, G, E) :- grenzStammF(F_a, X, G, E_a),
       grenzStammF(F_b, X, G, E_b), E is (E_a + E_b).
25 grenzStammF(F_a - F_b, X, G, E) :- grenzStammF(F_a, X, G, E_a),
       grenzStammF(F_b, X, G, E_b), E is (E_a - E_b).
26 grenzStammF(F_a * F_b, X, G, E) :- grenzStammF(F_a, X, G, E_a),
       grenzStammF(F_b, X, G, E_b), E is (E_a * E_b).
27 % Berechne Einzelteile
28 % c
29 grenzStammF(C, _, _, C) :- number(C).
30 % x
31 grenzStammF(X, X, G, G).
32 % x^n
33 grenzStammF(X^N, X, G, E) :- number(N), E is (G^N).
34 % --------------------
35 % Berechne bestimmtes Integral bestInt(Funktion, Variable, (untere)
       Grenze 1, (obere) Grenze 2, Ergebnis)
36 % --------------------
37 bestInt(F, X, G1, G2, E) :- number(G1), number(G2), atomic(X),
       stammF(F, X, StammF), grenzStammF(StammF, X, G1, E1),
       grenzStammF(StammF, X, G2, E2), E is (E2 - E1).
```


6 PRAKTIKUM 6: UMFORMUNGEN PROLOG

6.1 AUFGABE 14

_

7 PRAKTIKUM 7: SUCHE

7.1 AUFGABE 15

```
satz(In, Rest, S, O, V) :- nominal_phrase(In, R, S), verbal_phrase(R, Rest, O, V).
nominal_phrase(In, Rest, E) :- artikel(In, R), nomen(R, Rest, E).
verbal_phrase(In, Rest, O, V) :- verb(In, R, V), nominal_phrase(R, Rest, O).
artikel(In, Rest) :- match(eine, In, Rest, _); match(die, In, Rest, _).
verb(In, Rest, V) :- match(jagt, In, Rest, V); match(sieht, In, Rest, V).
nomen(In, Rest, E) :- match(katze, In, Rest, E); match(kater, In, Rest, E); match(maus, In, Rest, E).

match(X, [X|Rest], Rest, X).  % match erweitern, dass es das gemachte zurück gibt!
```

7.2 AUFGABE 16

```
command(CMD, POS_A, POS_B) :- moveCommand(CMD, CLR, DIR), positions(POS_A, CLR1_1, POS1_1, CLR2_1, POS2_1), move(CLR, DIR, CLR1_1, POS1_1, CLR2_1, POS2_1, POS_B).

% Prüfe/Extrahiere Move-Befehl: moveCommand(Liste von Strings, Farbe, Richtung).
moveCommand([move|LST], CLR, DIR) :- color(LST, CLR, [DIR|_]).
% Prüfe/Extrahiere Position: position(Liste von Strings, Farbe 1, ist an Position 1, Farbe 2, ist an Position 2).
positions(LST, CLR1, POS1, CLR2, POS2) :- color(LST, CLR1, R1), coordinate(R1, POS1, R2), color(R2, CLR2, R3), coordinate(R3, POS2, []).
% Extrahiere Koordinate: coordinate(Liste von Strings, Koordinate, Rest).
coordinate([C|R], C, R).
% Finde/Extrahiere Farbe und eliminiere Unwichtiges danach: color(Liste von Strings, Farbe der Stringkette, Rest).
color(LST, CLR, R) :- ( match(black, LST, R1, CLR) ; match(white, LST, R1, CLR) ), stuff(R1, R) .
% Eliminiere unwichtiges: stuff(Liste von Strings, Rest).
stuff(LST, R) :- ( match(rook, LST, R1, _ ); match(is, LST, R1, _ ) ; match(at, LST, R1, _ ) ), stuff(R1, R).
stuff(LST, LST).
```

```
14 % Bewege, falls möglich die richtige Figur: move(zu bewegende Farbe
       , Richtung, Farbe 1, Position 1, Farbe 2, Position 2).
15 % zuerst normieren:
16 move(CLR, DIR, white, POS1_1, black, POS2_1, POS_B) :- move(CLR,
       DIR, black, POS2_1, white, POS1_1, POS_B).
17 % dann bewegen:
18 move(black, DIR, black, POSB, white, POSW, POS_B) :- changePos(POSB
       , DIR, POSB2), validPos(POSB2, POSB, POSW, POSBF), POS_B = [
       black, at, POSBF, white, at, POSW].
19 move(white, DIR, black, POSB, white, POSW, POS_B) :- changePos(POSW
       , DIR, POSW2), validPos(POSW2, POSW, POSB, POSWF), POS_B = [
       black, at, POSB, white, at, POSWF].
20 % Gib veränderte Position aus: changePos(Originalposition, Richtung
       , Position nach Bewegung).
21 changePos(POSA, forward, POSB) :- xCoord(POSA, X), yCoord(POSA, Y),
        YB is (Y+1), POSB = (X, YB).
22 changePos(POSA, back, POSB) :- xCoord(POSA, X), yCoord(POSA, Y), YB
        is (Y-1), POSB = (X, YB).
23 changePos(POSA, left, POSB) :- xCoord(POSA, X), yCoord(POSA, Y), XB
        is (X-1), POSB = (XB, Y).
24 changePos(POSA, right, POSB) :- xCoord(POSA, X), yCoord(POSA, Y),
       XB is (X+1), POSB = (XB, Y).
25 % Gebe korrekte Position aus (verrückte, wenn Prüfung ok, sonst
       Originalposition): validPos(zu prüfende Position,
       Originalposition, Position der anderen Figur, Rückgabeposition).
26 validPos(POSA, _, POSB, POSA) :- onPlan(POSA), POSA \== POSB.
27 validPos(POSA, POSAo, POSB, POSAo) :- not(onPlan(POSA)); POSA ==
       POSB.
28 % Prüfe ob Position auf dem Plan ist:
29 onPlan(POS) :- xCoord(POS, X), X < 9, X > 0, yCoord(POS, Y), Y < 9,
        Y > 0.
30 % Gebe Position von Argumenten aus:
31 xCoord(POS, X) :- arg(1, POS, X).
32 yCoord(POS, Y) :- arg(2, POS, Y).
33
34 % Die gute, alte Match-Funktion
35 match(X, [X|Rest], Rest, X).
36
37 % Testeingaben und Ergebnisse:
38 %?- command([move , black , rook , forward], [black , is, at, (1,1)
       , white , rook , at, (8,8)], R).
39 % R = [black , at, (1, 2), white , at, (8, 8)] .
40 %?- command([move , white , left], [white , at, (5,5), black , rook
        , is, at, (2,2)], R).
41 % R = [black , at, (2, 2), white , at, (4, 5)] .
42 %?- command([move , white , left], [white , at, (5,5), black , rook
        , is, at, (4,5)], R).
43 % R = [black , at, (4, 5), white , at, (5, 5)] .
```


8 PRAKTIKUM 8: PFADSUCHE

8.1 AUFGABE 16

```
adj0(X,Y) :- (Y is X+1; Y is X-1), between(1,8,Y).
adj((X1,Y1),(X2,Y2)) :-
  adj0(X1,X2), Y2 = Y1;
  adj0(Y1,Y2), X2 = X1.
goal((8,8)).
dfs4(Node, Path, ReturnPath) :-
  goal(Node), reverse(Path, ReturnPath);
  adj(Node, NewNeighbor), not(member(NewNeighbor,Path)),
  dfs4(NewNeighbor, [NewNeighbor|Path],ReturnPath).
```


9 PRAKTIKUM 9: PFADSUCHE

9.1 AUFGABE 17

Umformung Heuristik: Entferne WURZELKNOTEN und berechne f von Nachbarknoten. Füge Nachbarknoten in Heap ein (kleinere oberhalb, größere unterhalb). Suche ist beendet, wenn der Wurzelknoten zum Ziel führt (genau genommen ist dieser Knoten am Schluss nicht mehr im Heap, weil er [zum Vergleichen] raus genommen wurde... wird hier nicht dargestellt)

(a) Startknoten in Heap einfügen, f berechnen:

$$(\text{A}, 0 + 6 = 6)$$

Wurzelknoten löschen und Nachbarknoten (B und C) mit berechnetem f einfügen:

$$(\text{A-B}, 5 + 2 = 7)$$
$$|$$
$$(\text{A-C}, 3 + 8 = 11)$$

Wurzelknoten erneut löschen und Nachbarknoten (D) mit berechnetem f einfügen:

$$(\text{A-B-d}, 8 + 0 = 8)$$
$$|$$
$$(\text{A-C}, 3 + 8 = 11)$$

(b) Startknoten:

$$(\text{A}, 6)$$

$$(\text{A-C}, 4)$$
$$|$$
$$(\text{A-B}, 7)$$

$$(\text{A-B}, 7)$$
$$|$$
$$(\text{A-C-D}, 13)$$

$$(\text{A-B-D}, 8)$$
$$|$$
$$(\text{A-C-D}, 13)$$

$\Rightarrow$ zu kleine Heuristik verlängert die Suche!

(c) Startknoten:

$$(\text{A}, 6)$$

(A-C, 11)
|
(A-B, 15)

(A-C-D, 13)
|
(A-B, 15)

⇒ ungültige (zu große) Heuristik ergibt nicht optimalen Pfad!

9.2 AUFGABE 18

$$t = \frac{s}{v}$$

Man nutze die Zeit $\frac{\text{Länge}}{\text{Höchstgeschwindigkeit}}$ als Pfadlänge.
Man nutze die Zeit $\frac{\text{Luftlinie}}{\max(\text{Höchstgeschwindigkeit})}$ als Heuristik h.

9.3 AUFGABE 19

Anmerkung: $\max\{f, g\} = x \mapsto \{f(x), g(x)\}$, also für jeden x-Wert das Maximum.
Sei $\max\{h_1, h_2\} = x \mapsto \max\{h_1(x), h_2(x)\}$ zulässig. Da $\max\{h_1(x), h_2(x)\} \in h_1(x) \cup h_2(x)$ ist, wird x nie überschätzt.

Musterlösung:
Seien $K(x)$ die Kosten bis zum Ziel. Aus h_1, h_2 zulässig folgt $h_1(x) \leq K(x), h_2(x) \leq K(x)$ für alle x. Daraus folgt $\max\{h_1, h_2\} \leq K(x)$ für alle x, woraus die Behauptung folgt.

9.4 AUFGABE 20

```
% Bonus: Schritte Zeilenweise ausgeben:
printP(P) :- maplist(writeln, P), nl.

% Bonus: Suchfunktion mit formatierter Ausgabe und Infos zur benötigten Zeit:
findPath(M, S, H, R) :-
  get_time(T1),
  START = [M, [S, H], R], goal(GOAL), idDfs(START, GOAL, P), printP(P),
  get_time(T2),
  DeltaT is round((T2- T1)*1000)/1000,
  write('Weg in '), write(DeltaT), write(' s gefunden.\n'), nl, !.

:- [idDfs].

% Suchfunktion mit unformatierter Ausgabe:
findPath(M, S, H, R, P) :- START = [M, [S, H], R], goal(GOAL), idDfs(START, GOAL, P).
```

```
16
17 % Liste aller Wege (Wegraster: (0,0) ist oben links und (7,5) ist
       unten rechts):
18 weg0(P1, P2) :- member((P1, P2), [
19   % Horizontale Wege (Zeilenweise):
20   ((0,0), (1,0)),
21   ((1,0), (2,0)),
22   ((3,0), (4,0)),
23   ((1,1), (2,1)),
24   ((2,1), (3,1)),
25   ((4,1), (5,1)),
26   ((5,1), (6,1)),
27   ((6,1), (7,1)),
28   ((2,2), (3,2)),
29   ((3,2), (4,2)),
30   ((6,2), (7,2)),
31   ((3,3), (4,3)),
32   ((5,3), (6,3)),
33   ((6,3), (7,3)),
34   ((0,4), (1,4)),
35   ((1,4), (2,4)),
36   ((2,4), (3,4)),
37   ((5,4), (6,4)),
38   ((3,5), (4,5)),
39   ((4,5), (5,5)),
40   ((5,5), (6,5)),
41   % Vertikale Wege (Zeilenweise):
42   ((0,0), (0,1)),
43   ((1,0), (1,1)),
44   ((4,0), (4,1)),
45   ((5,0), (5,1)),
46   ((6,0), (6,1)),
47   ((7,0), (7,1)),
48   ((1,1), (1,2)),
49   ((2,1), (2,2)),
50   ((6,1), (6,2)),
51   ((0,2), (0,3)),
52   ((2,2), (2,3)),
53   ((3,2), (3,3)),
54   ((5,2), (5,3)),
55   ((6,2), (6,3)),
56   ((0,3), (0,4)),
57   ((1,3), (1,4)),
58   ((3,3), (3,4)),
59   ((5,3), (5,4)),
60   ((7,3), (7,4)),
61   ((0,4), (0,5)),
62   ((1,4), (1,5)),
63   ((2,4), (2,5)),
64   ((3,4), (3,5)),
65   ((5,4), (5,5)),
```


```
66    ((7,4), (7,5))
67 ]).
68
69 % Überprüfe, ob ein Weg zwischen benachbarten Punkten 1 und 2
       existiert:
70 weg(P1, P2) :- weg0(P1, P2); weg0(P2, P1).
71
72 % Gültige Übergänge:
73 adj([M1, [], R1], [M1, [], R2]) :- member(M1,R1), delete(R1,M1,R2).
       % Rose schneiden
74 adj([M1, [],  R], [M2, [],  R]) :- weg(M1, M2).            % Mit
       Werkzeug bewegen
75 adj([M1, W1,  R], [M1, W2,  R]) :- member(M1,W1), delete(W1,M1,W2).
       % Werkzeug aufheben
76 adj([M1,  W,  R], [M2,  W,  R]) :- W \== [], weg(M1, M2).     %
       Ohne alle Werkzeuge bewegen
77
78 % Das Ziel: keine Rosen mehr zu schneiden!
79 goal([_,  _, []]).
80
81 % Zum Testen der Funktion:
82 % Einfach: findPath((0,0), (0,1), (1,0), [(1,1), (3,1)]).
83 % Mittel: findPath((0,0), (0,1), (1,0), [(7,5)]).
84 % Schwer: findPath((0,0), (7,5), (0,5), [(7,0), (0,0)]).
85 % Schwerer: findPath((0,0), (7,5), (0,5), [(7,0), (0,0), (3,5),
       (7,3), (2,0), (1,1)]).
```

10 PRAKTIKUM 10: PROBLEMLÖSUNG

10.1 AUFGABE 20

Aufstellung des ok:

$$\begin{aligned} &Z = W \vee Z = K \rightarrow Z = F \\ \equiv &(Z \neq W \cap Z \neq K) \vee Z = F \end{aligned}$$

Daraus kann man bilden:

```
sicher([F,F,_,_]).   % rechts vom oder
sicher([_,Z,K,W]) :- Z \== K, Z \== W.  % links vom oder
```

Musterlösung:

```
%       f z k w
sicher([F,F,_,_]).
sicher([_,Z,K,W]) :- Z \== K, Z \== W.

%     f z k w
adj0([0,0,K,W], [1,1,K,W]). % Ziege übersetzen
adj0([0,Z,0,W], [1,Z,1,W]). % Kohl übersetzen
adj0([0,Z,K,0], [1,Z,K,1]). % Wolf übersetzen
adj0([0,Z,K,W], [1,Z,K,W]). % Leerfahrt

adj(X,Y) :- (adj0(X,Y); adj0(Y,X)), sicher(X), sicher(Y).

solution(P) :- X = [0,0,0,0], Y = [1,1,1,1], idDfs(X,Y,P).
:- [idDfs].
```

Eigene Lösung:

```
:- [idDfs].

ort(X) :- member(X, [a, b]).
% Bedeutung der Listen: [Ziege, Kohlkopf, Wolf, Fährmann]
adj0([Z1,K,W,F1], [Z2,K,W,F2]) :-
  ort(Z1), ort(Z2), ort(F1), ort(F2),
  Z1 == F1, Z2 == F2,
  ok(Z1, K, W, F1), ok(Z2, K, W, F2). % Ziege rüber fahren
adj0([Z,K1,W,F1], [Z,K2,W,F2]) :-
  ort(K1), ort(K2), ort(F1), ort(F2),
  K1 == F1, K2 == F2,
  ok(Z, K1, W, F1), ok(Z, K2, W, F2). % Kohl rüber fahren
adj0([Z,K,W1,F1], [Z,K,W2,F2]) :-
  ort(W1), ort(W2), ort(F1), ort(F2),
  W1 == F1, W2 == F2,
  ok(Z, K, W1, F1), ok(Z, K, W2, F2). % Wolf rüber fahren
```

```
17 adj0([Z,K,W,F1], [Z,K,W,F2]) :-
18   ort(F1), ort(F2),
19   ok(Z, K, W, F1), ok(Z, K, W, F2). % Fährmann alleine rüber fahren
20
21 ok(Z,K,W,F) :- Z == K, Z == F; Z == W, Z == F; Z \== K , Z \== W. %
       Ziege nur in Beisein vom Fährmann zusammen mit Kohlkopf und
     Wolf
22
23 goal([b,b,b,b]).  % alle in Pillnitz und sicher
24
25 adj(X,Y) :- adj0(X,Y); adj0(Y,X).
26
27 solution(Path) :- Start = [a, a, a, a], goal(Goal), idDfs(Start,
     Goal,Path).
28
29 % solution(X), write(X).
```

11 PRAKTIKUM 11: SUCHE

11.1 AUFGABE 21

```
:- ['8puzzle-adj'].
:- [idas].
:- [idDfs].

% Heuristik Hamming-Distanz
h0(9,_,0). % Leerstelle zählt nicht
h0(_,9,0).
h0(A,A,0).
h0(_,_,1).

h(Board, HD) :-
  flatten(Board, B),
  goal(GoalB), flatten(GoalB,G),
  maplist(h0, B, G, Diffs),
  sumlist(Diffs, HD), !.

solution(Board, Sol) :- goal(G), idas(Board, G, Sol).

%solution([[9,1,2], [4,6,3], [7,5,8]], Sol), print(Sol).
```

11.2 AUFGABE 22

Lösung von Raphael Pour:

(a)

Beweisversuch - Zulässigkeit von h $h(s)$ ist zulässig wenn $h_m(s) + h_c(s)$ nicht überschätzt. Insbesondere dürfen $h_m(s)$ und $h_c(s)$ nicht überschätzen.

Zulässigkeit von h_m $h_m(s)$ überschätzt nicht, da diese Heuristik die Manhattan-Distanz einer 16-Puzzle-Brettstellung errechnet mit der Prämisse, dass zwischen Startfeld und Zielfeld keine weiteren Plättchen liegen und theoretisch der direkte Weg (über horizontale bzw. vertikale Bewegungen) möglich ist (siehe Abbildung 11.2). Wenn Start und Zielfeld direkt nebeneinander liegen, ist das Ergebnis von h_m, die exakte Anzahl von Zügen (siehe Abbildung 11.3).

Zulässigkeit von h_c h_c ist zulässig, da in beiden Fällen dieser Funktion nicht überschätzt wird:

Fall 1: $h_c(s) = 0$
Ist zulässig, da 0 eine zulässige Heuristik ist (vgl. Vorlesung).

Fall 2: $h_c(s) = 2$
Sei in einer Zeile mit dem Index z_i eines beliebigen Spielbretts s die Plättchen p, q. Sei Spielbrett g das Brett mit der Zielstellung bzw. dem Referenzbrett (11.1). Damit die Bedingung im diesem Fall gilt muss $q < p$ sein und p, q in Zeile z_i des Brettes g liegen.
Dieser Fall überschätzt nicht, da für p, q die vertikalen Züge ermittelt werden, welche mindestens beim Tauschen zweier Plättchen nötig wäre. Da bei einem 15-Puzzle nicht direkt vertauscht werden darf, muss ein Plättchen „um das andere herum“ geschoben werden.

Zulässigkeit von h Betrachtung von $h(s) = h_m(s) + h_c(s)$

Fall 1: $h_c(s) = 0$
In diesem Fall ist $h(s) = h_m(s) + 0 \Rightarrow h(s) = h_m(s)$. Da h_m zulässig ist, ist $h(s)$ in diesem Fall auch zulässig.

Fall 2: $h_c(s) = 2$
h_c ergänzt in diesem Fall die Heuristik für das Tauschen von p, q, welche durch Heuristik h_c ermittelt wird. Dort werden 4 Züge für den Tausch benötigt. h_c ermittelt 2 für diesen Fall. h_m ermittelt ebenso 2, da für das Verschieben von p auf die Position von q ein Zug benötigt wird und für das Verschieben von q auf die Position für p einen weiteren. Folglich ergibt $h_c(s) = 2$ und $h_m(s) = 2$. Somit ergibt sich $h(s) = 2 + 2 = 4$. Damit unterschätzt die Funktion h den Minimalfall nicht und h ist zulässig.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Abbildung 11.1: Zielpositionen

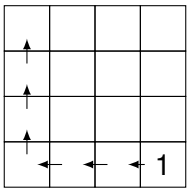


Abbildung 11.2: Beispielpfad für Manhattan-Distanz

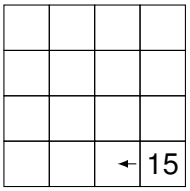


Abbildung 11.3: Sonderfall für Manhattan-Distanz

```
:- [idas].
% 16 = Leeres Feld
```

```
% shift stellt mit Hilfe von shift0 alle möglichen
% horizontalen verschiebungen dar.
shift0([A,B,C,16],[A,B,16,C]).
shift0([A,B,16,C],[A,16,B,C]).
shift0([A,16,B,C],[16,A,B,C]).
shift(X,Y) :-
    shift0(X,Y);
    shift0(Y,X).

% Transponiert die aktuellen Brettposition
% Mit Hilfe einer Transponierung sind horizontale
% und vertikale Züge identisch
transpose([[A1,B1,C1,D1],
           [A2,B2,C2,D2],
           [A3,B3,C3,D3],
           [A4,B4,C4,D4]],
          [[A1,A2,A3,A4],
           [B1,B2,B3,B4],
           [C1,C2,C3,C4],
           [D1,D2,D3,D4]]
           ).

adj0([  In,Row2,Row3, Row4],[ Out,Row2,Row3,Row4]) :- shift(In,
    Out).
adj0([Row1,  In,Row3, Row4],[Row1, Out,Row3,Row4]) :- shift(In,
    Out).
adj0([Row1,Row2,  In, Row4],[Row1,Row2, Out,Row4]) :- shift(In,
    Out).
adj0([Row1,Row2,Row3,   In],[Row1,Row2,Row3, Out]) :- shift(In,
    Out).


adj(X,Y) :- adj0(X,Y); (transpose(X,S),adj0(S,R), transpose(R,Y
   )).

% Die Ziel-Brettpositionen, wobei 16 das leere Feld bedeutet
goal([[ 1, 2, 3, 4],
      [ 5, 6, 7, 8],
      [ 9,10,11,12],
      [13,14,15,16]]).

% trivial
start1([[ 1, 2, 3, 4],
       [ 5, 6, 7, 8],
       [ 9,10,11,12],
       [13,14,16,15]]).

% difficult
start([[ 1, 2, 3, 4],
       [ 5, 7, 8, 11],
       [ 9,6,16,12],
```


```
        [13,15,14,10]]).

getX([Row,_,_,_],Field,0) :- member(Field,Row),!.
getX([_,Row,_,_],Field,1) :- member(Field,Row),!.
getX([_,_,Row,_],Field,2) :- member(Field,Row),!.
getX([_,_,_,Row],Field,3) :- member(Field,Row),!.

getY(Board,Field,Pos) :-
    transpose(Board,BoardT),
    getX(BoardT,Field,Pos).

getPos(Board,Field,(X,Y)) :-
    getX(Board,Field,X),
    getY(Board,Field,Y),!.

% Calculate Manhattan-Distance of a field with the field where
   it should be
% We use the goal as reference to calculate two absolute points
    and
% compute them with the manhattan-distance formula
manhattan(16,_,0) :- !.
manhattan(F,F,0) :- !.
manhattan(Field1,Field2,D) :-
   goal(Board),
   getPos(Board,Field1,(X1,Y1)),
   getPos(Board,Field2,(X2,Y2)),
   D is abs(X1 - X2) + abs(Y1 - Y2).


hc0([Q,P,_,_],G) :- hc1(Q,P,G).
hc0([Q,_,P,_],G) :- hc1(Q,P,G).
hc0([Q,_,_,P],G) :- hc1(Q,P,G).
hc0([_,Q,P,_],G) :- hc1(Q,P,G).
hc0([_,Q,_,P],G) :- hc1(Q,P,G).
hc0([_,_,Q,P],G) :- hc1(Q,P,G).

hc1(Q,P,G) :-
    Q \== 16,
    P \== 16,
    Q > P,
    member(Q,G),
    member(P,G).


hc([R,_,_,_],2) :- goal([G,_,_,_]),hc0(R,G).
hc([_,R,_,_],2) :- goal([_,G,_,_]),hc0(R,G).
hc([_,_,R,_],2) :- goal([_,_,G,_]),hc0(R,G).
hc([_,_,_,R],2) :- goal([_,_,_,G]),hc0(R,G).
hc(_,0).

h(In,Out) :-
```

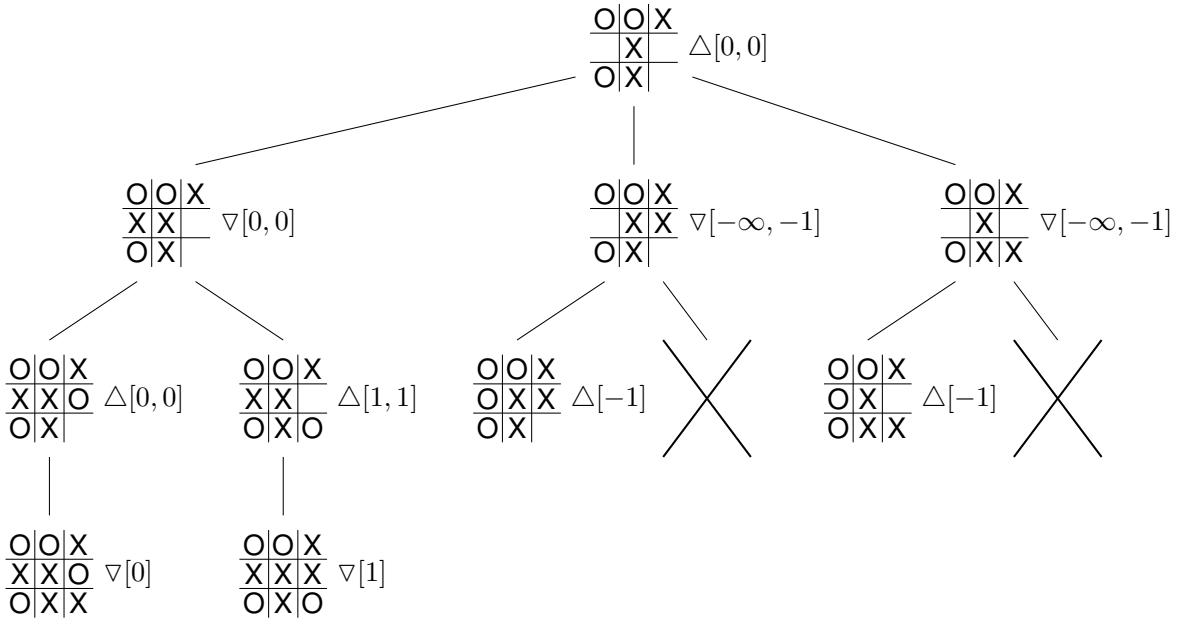

```
98      flatten(In, InF),
99      goal(Goal),
100     flatten(Goal,GoalF),
101     maplist(manhattan,InF,GoalF,HOut),
102     sumlist(HOut,Out1),
103     hc(In,Out2),
104     Out is Out1 + Out2.
105
106 % Pretty-Printer
107 printB(Board) :- maplist(writeln, Board), write('\n').
108 print(Boards) :- maplist(printB, Boards).
109
110 solution(P) :- start(S), goal(G), idas(S,G,P), print(P),!.
```


12 PRAKTIKUM 12: MIN-MAX, KOMBINATORIK

12.1 AUFGABE 22

Beginnend mit $\triangle[-\infty, \infty]$, da das Max für den nächsten Zug benötigt wird.

(b)



12.2 AUFGABE 23

Einfachste Lösung:
$3^9 = 19\,683$ (komplette Belegungsmöglichkeiten von 3×3 Feldern)

Erweiterte/Genauere Lösung:
$\binom{9}{5} \cdot 2^5 + \binom{9}{6} \cdot 2^6 + \binom{9}{7} \cdot 2^7 + \binom{9}{8} \cdot 2^8 + \binom{9}{9} \cdot 2^9 = 16\,832$
Dabei bezeichnet das $\binom{n}{k}$, dass k Feldern von n ausgewählt wurden. Die zwei Status (X und O) müssen dann auf diese k Felder verteilt werden: 2^k.

12.3 AUFGABE 24

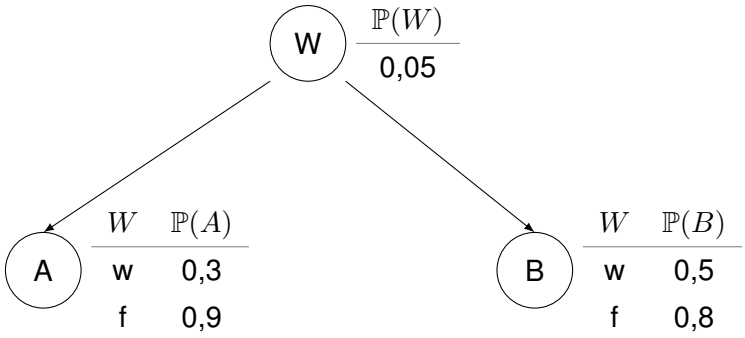
$1 + 24 \cdot (1 + 23 \cdot (1 + 22 \cdot (1 + 21 \cdot (1 + 20)))) = 5\,368\,225$
Dabei sind 1 (Wurzelknoten/Ausgangssituation des aktuellen Spielzugs) plus die verbleibenden Auswahlmöglichkeiten mal die jeweils weiteren Ebenen die möglichen Kombinationen.

13 PRAKTIKUM 13: BAYESSCHES NETZ

13.1 AUFGABE 26

A ... Student A erscheint zur Vorlesung
B ... Student B erscheint zur Vorlesung
W ... Es ist Badewetter
$P(A|W) = 0,3$
$P(A|\overline{W}) = 0,9$
$P(B|W) = 0,5$
$P(B|\overline{W}) = 0,8$
$P(W) = 0,05$

(a) Bayes'sches Netz:



(b) Gesucht: $\mathbb{P}(\overline{A}|\overline{B}) = \dfrac{\mathbb{P}(\overline{A} \cap \overline{B})}{\mathbb{P}(\overline{B})}$

Berechnen von $\mathbb{P}(\overline{A} \cap \overline{B})$:

$$\begin{aligned}\mathbb{P}(\overline{A} \cap \overline{B}) =& \mathbb{P}(\overline{A} \cap \overline{B} \cap W) + \mathbb{P}(\overline{A} \cap \overline{B} \cap \overline{W}) \\ & \uparrow \text{ (Marginalisierung / disjunkte Zerlegung)} \\ =& \mathbb{P}(\overline{A} \cap \overline{B}|W) \cdot \mathbb{P}(W) + \mathbb{P}(\overline{A} \cap \overline{B}|\overline{W}) \cdot \mathbb{P}(\overline{W}) \\ & \text{Da } A \text{ und } B \text{ unabhängig:} \\ =& \mathbb{P}(\overline{A}|W) \cdot \mathbb{P}(\overline{B}|W) \cdot \mathbb{P}(W) + \mathbb{P}(\overline{A}|\overline{W}) \cdot \mathbb{P}(\overline{B}|\overline{W}) \cdot \mathbb{P}(\overline{W}) \\ =& 0,7 \cdot 0,5 \cdot 0,005 + 0,1 \cdot 0,2 \cdot 0,95 \\ =& 0,0365\end{aligned}$$

Berechnen von $\mathbb{P}(\overline{B})$:

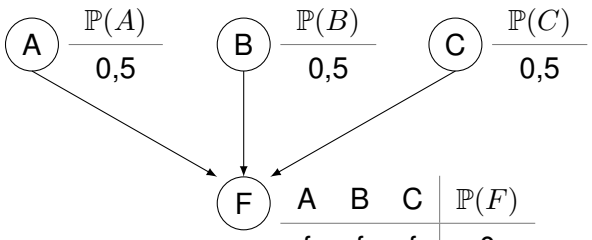
$$\begin{aligned}\mathbb{P}(\overline{B}) =& \mathbb{P}(\overline{B}|W) \cdot \mathbb{P}(W) + \mathbb{P}(\overline{B}|\overline{W}) \cdot \mathbb{P}(\overline{W}) \\ =& 0,5 \cdot 0,05 + 0,2 \cdot 0,95 \\ =& 0,215\end{aligned}$$

Und damit $\mathbb{P}(\overline{A}|\overline{B}) = \dfrac{0,0365}{0,215} = 0,1698$

(c) $\mathbb{P}(\overline{A}) = \mathbb{P}(\overline{A}|W) \cdot \mathbb{P}(W) + \mathbb{P}(\overline{A}|\overline{W}) \cdot \mathbb{P}(\overline{W}) = 0,7 \cdot 0,05 + 0,1 \cdot 0,95 = 0,13$
Da $\mathbb{P}(\overline{A}|\overline{B}) \neq \mathbb{P}(\overline{A}) \Rightarrow 0,1698 \neq 0,13$, sind A und B abhängig.

14 PRAKTIKUM 14: BAYESSCHES NETZ, SAMPLING

14.1 AUFGABE 27



A	B	C	$\mathbb{P}(F)$
f	f	f	0
f	f	w	1
f	w	f	0
f	w	w	0
w	f	f	0
w	f	w	1
w	w	f	0
w	w	w	1

$$\begin{aligned}\mathbb{P}(F) =& \mathbb{P}(F, A, B, C) + \ldots + \mathbb{P}(F, \overline{A}, \overline{B}, \overline{C})\\ &\text{Disjunkte Zerlegung / Marginalisierung der Summanden:}\\ =& \mathbb{P}(F|A \cap B \cap C) \cdot \mathbb{P}(A) \cdot \mathbb{P}(B) \cdot \mathbb{P}(C) + \ldots\\ &\quad + \mathbb{P}(F|\overline{A} \cap \overline{B} \cap \overline{C}) \cdot \mathbb{P}(\overline{A}) \cdot \mathbb{P}(\overline{B}) \cdot \mathbb{P}(\overline{C})\\ &\text{Da } \mathbb{P}(A) = \mathbb{P}(\overline{A}), \text{ sind alle Permutationen von } \mathbb{P}(A), \mathbb{P}(B) \text{ und } \mathbb{P}(C) \text{ gleich}\\ &\text{und können ausgeklammert werden:}\\ =& \underbrace{\frac{1}{8}}_{\left(\frac{1}{2}\right)^3} \cdot (\mathbb{P}(F|A \cap B \cap C) + \ldots + \mathbb{P}(F|\overline{A} \cap \overline{B} \cap \overline{C}))\\ =& \frac{1}{8} \cdot 3\\ =& \frac{3}{8}\end{aligned}$$

14.2 AUFGABE 28

```
1 ... Code in beliebiger Programmiersprache, die Stichproben auswählt
    und diese als Sample nutzt.
```