Fünf philosophische, zentrale Themen - in PROLOG:

Schuld, Verantwortung, Gerechtigkeit, Reichtum, Wissenschaft

Im Mai 2022 hatten wir (Balzer + Kurzawe) ein philosophisches Seminar über diese Themen abgehalten. Wir hatten dazu zentrale, allgemeine Modelle formuliert und in der Sprache PROLOG programmiert, siehe vorbereitend.

Balzer, W., Kurzawe, D., Manhart, K. (2014) Künstliche Gesellschaften mit PROLOG, V & R unipress, Göttingen.

Diese Programme sind Übungen und zum selbständigen Nachdenken über diese Fragen gedacht. Dazu kommt - wie heute fast immer - das Hilfsmittel, ein 'Tool: Computer' zum Einsatz.

Diese Programme lassen sich ohne Mühe (wenn Sie PROLOG benutzen können) in vielen Dimensionen verbessern, erweitern und vervollständigen.

Thema 1: Schuld

Das Modell

Eine Person ist schuld, wenn sie einer (oder mehreren) Person(en) Leid zugefügt hat. Im speziellen Fall leidet die Person an ihrem eigenen Schicksal. Sie befindet sich in einer für sie schlechten Situation, aus der sie sich nicht selbst befreien kann. Wir interessieren uns hier nur für die zweite Bedeutung.

In der Gesellschaft gibt es in einer Periode mehrere Personen. Personen können Frauen oder Männer oder Kinder sein. Neben den Personen gibt es auch mehrere Objekte, die von Personen produziert werden können. Die Personen sind in vier Statusklassen eingeteilt. Eine Person kann mächtig, frei, gebunden oder ein Kind sein. Damit existieren in der Gesellschaft vier Mengen von Personen. Ein Kind wird zu einem Zeitpunkt erwachsen; es wird ein Mann oder eine Frau und es wird frei oder gebunden sein. Zu jedem Zeitpunkt kommen neue Kinder ins Spiel.

Die Personen führen in jedem Zeitpunkt eine Handlung aus. Es gibt hier nur vier Handlungstypen: 1) eine Person P wird an eine andere Person gebunden, 2) P produziert, 3) P arbeitet, 4) P spielt. Die Statusklassen ändern sich zu jedem Zeitpunkt.

Die Interpretationen überlassen wir Ihnen. Wir haben ein einfaches Programm formuliert, das ausbaufähig ist.

Die Handlungstypen:

1) P wird an eine andere Person gebunden
2) P produziert
3) P arbeitet
4) P spielt.

Wir kürzen ab:

- ZOB ist die AnZahl der Objekte (OB ist ein Objekt)
- TT ist die AnZahl der Tics (T ist ein Tic)
- Lae ist die Länge einer Liste
- ZPSK ist die AnZahl der Personen in einer StatusKlasse
- ZP ist die AnZahl der Personen
- P ist eine Person.

Das Modell wird mit Hife von PROLOG konstruiert.

In Zeile 11 (siehe das Programm unten) wird die Anfangsverteilung der Klassen festgelegt.

verteilung(1,[2,10,4,4])

besagt zum Beispiel, dass zum ersten Tic 1, die erste Statusklasse 2 Personen enthält. Die zweite Klasse enthält 10 Personen, die dritte Klasse 4 und die vierte Klasse ebenfalls 4 Personen.

In Zeile 18 wird diese Verteilung aus der Datenbasis geholt. In 19 wird die Länge der Liste V

V = [2,10,4,4]

bestimmt und in 20 wird die Summe der Zahlen aus dieser Liste berechnet, ZP = 2+10+4+4 = 20. Damit ist die Anzahl ZP der Personen festgelegt. Diese Zahl wird in Zeile 21 durch

zahl_der_personen(...)

in die Datenbasis aufgenommen. Diese Zahl wurde nicht als eine Konstante eingeführt, weil sie sich mit Zeit ändert.

In Zeilen 22 und 29 werden Statusklassen erzeugt. In 30 wird ein Zähler (counter) eingerichtet und auf 0 gestellt. In 31 wird eine Schleife über die Anzahl der Klassen durchlaufen.

In 35 wird die X-te Statusklasse eingeführt und in die Datenbasis geschrieben. In diesem Moment ist diese Klasse leer. In 36 wird die X-te Komponente aus der Liste

V (verteilung)

ermittelt. Diese Komponente ZPSK ist eine Konstante (eine Zahl), die in Zeile 11 zu finden ist.

In 37 wird eine Schleife gebildet. Für jede Zahl aus ZPSK wird eine Statusklasse mit Personen 'gefüllt'. Y ist hier ein Index, der nur für die Schleifenbildung verwendet wird. Die Statusklasse Nr. X wird vergroessert, d.h. eine weitere Person wird in die Klasse eingefügt.

In 41 und 42 wird der counter geöffnet, die Zahl C um Eins (C1 = C + 1) erhöht, in die Datenbasis geschrieben und die vorherige Zahl C gelöscht. In 45 wird die Statusklasse

KL der Nummer X

geholt, die sich in der Schleife im Aufbau befindet. In 46 wird die Zahl C1 aus dem Zähler zu einer Person 'umdeklariert' und in die Statusklasse KL eingefügt. Die Sta-

tusklasse KL wird in 47 gelöscht und die so vergroesserte Statusklasse KLneu in die Datenbasis eingetragen.

In 23 werden Konten der Personen erzeugt. In 51 wird eine Schleife über die Anzahl der Personen durchlaufen. Für jede Person P wird geschaut, zu welcher Statusklasse sie gehört. Im ersten Fall in 55 wird die erste Statusklasse 1 geholt und in 56 geschaut, ob P in dieser Klasse liegt. Wenn ja, wird eingetragen

person(1,P,1,1000).

Im Term person(1,P,1,1000) ist 1 der Tic, zu dem das Konto von P betrachtet wird. Die zweite Zahl 1 besagt, dass die Person P zur ersten ('1') Klasse gehört und 1000 besagt, dass P den Kontostand 1000 hat. P besitzt also zu Tic 1 1000 Einheiten und P gehört zur Klasse 1.

Im zweiten Fall hat Person P in Zeile 61 50 Einheiten auf seinem Konto; P gehört zur Klasse 2.

Im dritten Fall, in Zeile 65, hat P nur 5 Einheiten auf dem Konto.

Im vierten Fall hat P in Zeile 69 nur 0.5 Einheiten.

In 24 und 74 werden Verwandschaften und Abhängigkeiten zwischen Personen generiert. In 75 werden die vier

Statusklassen KL1,KL2,KL3,KL4

geholt. Diese sind in Zeile 8 als Konstante hinterlegt. Diese Klassen wurden in Zeilen 29 - 47 schon erzeugt.

In 76 werden die Klassen (d.h. Listen) KL1 und KL2 zu KLX vereinigt.

KL1 ist die Liste der Mächtigen und
KL2 ist die Liste der Freien (Personen).
Diese Vereinigung wird weiter mit KL3 vereinigt.
KL3 ist die Liste der Abhängigen.

Eine Liste

KL_nicht_kinder

von Personen, die keine Kinder sind, wird erzeugt. Diese Konstruktion soll Probleme im Computerablauf verhindern.

In 77 wird die Zahl der Kinder, die in KL4 zu finden sind, berechnet. In 78 wird die Anzahl der 'nicht Kinder' (die Liste: KL_nicht_kinder) bestimmt.

In 79 wird eine Schleife über die Anzahl der Kinder gelegt. Für jedes Kind

Nr_KIND

werden Verwandtschaften festgelegt. Dazu wird in 86 und 87 das Kind Nr_KIND aus der Liste KL4 geholt. In 88 wird zufällig ein Name

NP (eine Zahl für eine Person)

aus der Zahl der KL_nicht_kinder gezogen. In 89 wird dieser Name (der gleichzeitig als Index benutzt wird) aus der Liste KL_nicht_kinder geholt. Dieser Name wird nun 'umgetauft' zu 'Erzeuger'. D.h. Erzeuger ist der Erzeuger des Kindes: Kind. In diesem Programm gibt es nur einen Erzeuger (was sich leicht ändern lässt).

In 90 wird diese Verwandtschaftsbeziehung in die Datenbasis eingetragen:

kind(1,Erzeuger,Kind)

besagt, das zu Tic 1 das Kind durch den Erzeuger generiert wird.

In 82 wird die

Zahl_der_Abhaengigen

(Personen) aus der Klasse (Liste) KL3 bestimmt.

In 83 werden Abhängigkeiten zwischen Personen erzeugt. Dazu wird eine Schleife über die Zahl_der_Abhaengigen gelegt. In 92 und 93 wird ein Abhängiger als Nummer und als 'Name'

Nr_eines_Abhaengigen

betrachtet.

In 94 wird mit dieser Nummer ein Abhaengiger (eine Person) aus der Liste KL3 geholt. In 94 wird die Länge von KL1 und damit die

Zahl_der_Maechtigen

bestimmt.

In 95 wird ein Name (ein Index für einen Mächtigen)

NM

zufällig gezogen. In 96 wird aus der Liste KL1 der Mächtigen die Person des Namens NM bestimmt und 'umgetauft' zu 'Maechtiger'.

Schließlich wird in 97 diese Information in die Datenbasis geschickt. Der Term

gebunden(Abhaengiger,Maechtiger)

besagt, dass Person des Names 'Abhaengiger' an die Person des Names 'Maechtiger' gebunden ist (zum Beispiel durch Schuld).

In Zeile 25 schließlich werden diese Beziehungen zu den verschiedenen Tics verändert.

In 101 wird eine Schleife über die Tics gelegt. In 105 wird ein

Tic T

gestartet.

In 106 und 107 werden die Zahl der Personen und die Statusverteilung VT zum Tic T aus der Datenbasis geholt. Zu Tic T wird in 108 eine zunächst leere Liste

benutzte_personen([])

eingerichtet. In 109 wird zu Tic T eine Schleife über die Personen gelegt. Dabei wird die gerade existierende Zahl ZP von Personen benutzt.

In 140 wird eine bestimmte Person P zu T bearbeitet. In 141 wird weitere Information über P aus der Datenbasis geholt. Diese Information wurde früher generiert.

Zu T gehört P zur Statusklasse S und sein Konto enthält den Betrag B. Je nach Klasse führt P eine der vier möglichen Aktivitäten durch.

Im ersten Fall wird in 142 die Statusklasse KLASSE1 zu T geholt. In 143 wird gefragt, ob P zur dieser Klasse KLASSE1 gehört. Wenn ja, wird P als Kind betrachtet. Dies führt über 143 mit

binde_ein_kind(T,P,B)

zu Zeile 157.

In 158 wird die Information über P noch ein Mal herausgeholt. In 159 wird die Liste BP der benutzten Personen und in 160 die vierte Statusklasse geholt. Diese Liste wird in 161 aus der KLASSE 4 (Liste) herausgenommen (als Menge 'subtrahiert').

Die

Laenge4

der Liste INTERSECT wird in 162 bestimmt. In 163 werden zwei Fälle unterschieden.

Im ersten Fall (1 < Laenge4) wird in 164 ein Name (eine Zufallszahl)

NK

für ein KIND aus der Liste INTERSECT geholt. In 165 wird die weitere Information für KIND geholt. Zum Zeitpunkt T gehört KIND zur vierten Statusklasse 4 und hat den Betrag B1 auf seinem Konto.

In 166 wird das KIND in die Liste

BP (der benutzten Personen)

aufgenommen und upgedatet. Der Betrag B1 für KIND wird in 168 um eine Einheit 1 erhöht.

Das KIND wird in 169 aus der Klasse 4 herausgenommen und in die dritte Klasse 3 mit dem neuen Betrag B1neu aufgenommen.

In 170 wird das KIND an die mächtige Person P gebunden. In 171 wird im Konto von P eine Einheit weggenommen, so dass P zu Tic T nicht mehr den Betrag B, sondern nur noch den Betrag Bneu besitzt.

Im zweiten Fall in 145 ist P ein freier Akteur; er produziert in Zeile 146 und in 176. In 177 wird die Information über P vervollständigt. In 178 wird die Anzahl der Objekte geholt und in 179 wird zufällig ein Objekt OB betrachtet. Das Objekt wird als Zahl behandelt, so dass der Wert des Objekts mit der Zahl identifiziert wird. In 180 wird der Wert von OB um ein Zehntel erhöht. Dieser Teilbetrag wird dem Betrag B des Kontos von P zugeschrieben. In 172 wird das Konto von P upgedatet.

Im dritten Fall in 148 stammt P aus der dritten Statusklasse; P arbeitet. In 186 und 187 wird die Liste BP der benutzten Personen geholt. Wenn in Zeile 188 Person P schon benutzt wurde, passiert nichts. Im anderen Fall wird die Information über P vervollständig.

In 191 wird die Person P1 ermittelt, an die P gebunden ist. Diese Person P1 ist mächtig, sie gehört zur Klasse 1, siehe 192. In 194 wird ein Objekt zufällig geholt.

Der Wert OB wird in 195 um ein Zehntel erhöht. Dieser Wert Bneu wird in 196 aufgeteilt. Ein Fünftel des Wertes ist der Teilwert Val1 und vier Fünftel des Wertes ist der Teilwert Val2.

In 197 wird Val1 dem Konto von P1 zugeschrieben und Val2 dem Konto von P. In 198 und in weiteren Zeilen werden die Konten von P und P1 upgedatet.

Im vierten Fall passiert nichts. Das Kind spielt, siehe Zeile 151, 152 und 206.

Damit ist die Schleife in Zeile 109 beendet. Das Programm kommt zu Zeile 110.

Dort wird die Klasse 4 und die Liste BP (der benutzten Personen) geholt. In Zeile 111 wird die Liste BP aus der Liste KLASSE4 entfernt. In 112 und 113 wird die Klasse 4 upgedatet.

In 114 wird die Klasse 3 geholt und die Liste BP zu dieser Klasse dazugenommen. In 115 und 116 wird Klasse 3 upgedatet. In 117 wird die Liste BP (der benutzten Personen) gelöscht. Diese Liste wird nicht mehr gebraucht.

In 118 wird die Statusverteilung geholt. Das Program befindet sich im Tic T. Diese Verteilung ändert sich mit der Zeit. In 119 wird die Zahl V3 um 2 erhöht und die Zahl V4 um zwei vermindert. D.h. zwei abhängige Personen kommen in Klasse 3 hinzu und 2 Kinder werden aus der Klasse 4 entfernt. Zwei Kinder sind erwachsen geworden. In 120 und 121 wird die Verteilung entsprechend upgedatet.

In 122 und 210 werden neue Personen generiert. In 211 wird die Anzahl von Personen ermittelt, die zu einem bestimmten Tic T exisieren. In 212 wird diese Zahl

ZP

um zwei erhöht und in 213 wird der nächste Tic T1 ins Spiel gebracht. In die Datenbasis wird die neue Anzahl

ZPneu

für den nächsten Tic T1 eingetragen.

In 215 werden zwei 'Namen' für die neuen Personen eingeführt. Als Namen werden einfach die zwei, noch nicht benutzten Zahlen

P1, P2 für Personen

benutzt.

In 216 und 217 kommen die beiden neuen Personen in die Klasse 4 und ihre Konten enthalten jeweils 0.5 Einheiten. In 218 wird die Statusverteilung zu T geholt und in 219 wird

Q4, die Zahl der Kinder zu T,

um zwei (2) erhöht. In den weiteren Zeilen werden die Verteilung und die Statusklasse Nr. 4 upgedatet.

PROLOG springt nun zu Zeile 123, wo der nächste Tic T1 (noch ein Mal) ausgeführt wird. In 124 wird die zahl_der_personen für Tic T1 aus der Datenbasis geholt, die gerade vorher entstanden ist. In 125 wird die neue Verteilung

VTneu

geholt, die noch zum Tic T eingetragen wurde. Diese Verteilung wird nun auch für den nächsten Tic T1 in die Datenbasis eingetragen. In 126 und den nächsten Zeilen werden Personen und Statusklassen von Tic T in den nächsten Tic T1 übertragen.

Schließlich wird in den Zeilen 127 und 227 und Weiteren, die Daten, die zum Tic T entstanden sind, in die Resultatdatei

res2

eingetragen.

Das Programm

```
:- dynamic verteilung/2.
status([maechtig,frei,gebunden,kind]).
zahl_der_objekte(10).
konstante(0.2).
statusklassen([1,2],[3,4,5,6,7,8,9,10,11,12],[13,14,15,16],[17,18,19,20]).
zahl_der_tics(5).
verteilung(1,[2,10,4,4]).

start :-
    ( exists_file('res1.pl'), delete_file('res1.pl'); true),
    ( exists_file('res2.pl'), delete_file('res2.pl'); true),
    zahl_der_tics(TT),
    zahl_der_objekte(ZOB),
    verteilung(1,V),
    length(V,Lae),
    sum(V,ZP),
    asserta(zahl_der_personen(1,ZP)),
    bilde_status_klassen(V,Lae),
    generiere_personenkonto(ZP), /* 20 = 2+10+4+4 */
    generiere_verwandschaft_und_abhaengigkeiten,
    bilde_eine_schleife_ueber_tics(TT),!.

bilde_status_klassen(V,Lae) :-
    asserta(counter(0)),
    ( between(1,Lae,X), bilde_eine_statusklasse(X,V), fail; true),
    retractall(counter(CCC)),!.

bilde_eine_statusklasse(X,V) :-
    asserta(statusklasse(X,[])),
    nth1(X,V,ZPSK),
    ( between(1,ZPSK,Y), vergroessere(Y,X), fail; true),
    statusklasse(X,KLnn), retract(statusklasse(X,KLnn)),
    asserta(statusklasse(X,1,KLnn)),!.

vergroessere(Y,X) :-
    counter(C),
    C1 is C + 1,
    retract(counter(C)), asserta(counter(C1)),
    statusklasse(X,KL),
    append(KL,[C1],KLneu),
    retract(statusklasse(X,KL)), asserta(statusklasse(X,KLneu)),!.

generiere_personenkonto(ZP) :-
    ( between(1,ZP,P), generiere_ein_konto(P,ZP), fail; true),!.
```

```
generiere_ein_konto(P,ZP) :-
    ( statusklasse(1,1,KL1),
    member(P,KL1),
    asserta(person(1,P,1,1000))
    ;
    statusklasse(2,1,KL2),
    member(P,KL2),
    asserta(person(1,P,2,50))
    ;
    statusklasse(3,1,KL3),
    member(P,KL3),
    asserta(person(1,P,3,5))
    ;
    statusklasse(4,1,KL4),
    member(P,KL4),
    asserta(person(1,P,4,0.5))
    ),!.
generiere_verwandschaft_und_abhaengigkeiten :-
    statusklassen(KL1,KL2,KL3,KL4),
    union(KL1,KL2,KLX), union(KLX,KL3,KL_nicht_kinder),
    length(KL4,Zahl_der_kinder),
    length(KL_nicht_kinder,Zahl_der_nicht_kinder),
    ( between(1,Zahl_der_kinder,Nr_KIND),
    create_verwandschaft(Nr_KIND,KL4,KL_nicht_kinder,Zahl_der_nicht_kinder),
    fail; true),
    length(KL3,Zahl_der_Abhaengigen),
    ( between(1,Zahl_der_Abhaengigen,Nr_eines_Abhaengigen),
    create_abhaengigkeit(Nr_eines_Abhaengigen,KL3,KL1), fail; true),!.
create_verwandschaft(Nr_KIND,KL4,KL_nicht_kinder,Zahl_der_nicht_kinder) :-
    nth1(Nr_KIND,KL4,Kind),
    NP is random(Zahl_der_nicht_kinder) + 1,
    nth1(NP,KL_nicht_kinder,Erzeuger),
    asserta(kind(1,Erzeuger,Kind)),!.
create_abhaengigkeit(Nr_eines_Abhaengigen,KL3,KL1) :-
    nth1(Nr_eines_Abhaengigen,KL3,Abhaengiger),
    length(KL1,Zahl_der_Maechtigen),
    NM is random(Zahl_der_Maechtigen) + 1,
    nth1(NM,KL1,Maechtiger),
    asserta(gebunden(Abhaengiger,Maechtiger)),!.
bilde_eine_schleife_ueber_tics(TT) :-
    ( between(1,TT,T), execute_tic(T), fail; true),
```

```
    rausschreiben,!.
execute_tic(T) :-
    zahl_der_personen(T,ZP),
    verteilung(T,VT),
    asserta(benutzte_personen([])),
    bilde_schleife_ueber_personen(T,ZP),
    statusklasse(4,T,KLASSE4), benutzte_personen(BP),
    subtract(KLASSE4,BP,KLASSE4neu),
    retract(statusklasse(4,T,KLASSE4)),
    asserta(statusklasse(4,T,KLASSE4neu)),
    statusklasse(3,T,KLASSE3), append(KLASSE3,BP,KLASSE3neu),
    retract(statusklasse(3,T,KLASSE3)),
    asserta(statusklasse(3,T,KLASSE3neu)),
    retract(benutzte_personen(BP)),
    verteilung(T,[V1,V2,V3,V4]),
    V3neu is V3 + 2, V4neu is V4 - 2,
    retract(verteilung(T,[V1,V2,V3,V4])),
    asserta(verteilung(T,[V1,V2,V3neu,V4neu])),
    erzeuge_neue_personen(T),
    T1 is T + 1,
    zahl_der_personen(T1,ZPneu),
    verteilung(T,VTneu),
    asserta(verteilung(T1,VTneu)),
    repeat,
    ( person(T,P,S,B),
    asserta(person(T1,P,S,B)),
    fail; true ),
    repeat,
    ( statusklasse(S1,T,KL),
    asserta(statusklasse(S1,T1,KL)), fail; true ),!.
bilde_schleife_ueber_personen(T,ZP) :-
    ( between(1,ZP,P), execute_akteur(P,T), fail; true),!.
execute_akteur(P,T) :-
    person(T,P,S,B),
    ( statusklasse(1,T,KLASSE1),
    member(P,KLASSE1), binde_ein_kind(T,P,B)
    ;
    statusklasse(2,T,KLASSE2),
    member(P,KLASSE2), produziere(T,P,B)
    ;
    statusklasse(3,T,KLASSE3),
    member(P,KLASSE3), arbeite(T,P,B)
```

```
    ;
    statusklasse(4,T,KLASSE4),
    member(P,KLASSE4), spiele(T,P,B)
    ),!.
binde_ein_kind(T,P,B) :-
    person(T,P,1,B),
    benutzte_personen(BP),
    statusklasse(4,T,KLASSE4),
    subtract(KLASSE4,BP,INTERSECT),
    length(INTERSECT,Laenge4),
    ( 1 ¡ Laenge4, NK is random(Laenge4) + 1; NK is 1 ),
    nth1(NK,INTERSECT,KIND),
    person(T,KIND,4,B1),
    append(BP,[KIND],BPneu),
    retract(benutzte_personen(BP)), asserta(benutzte_personen(BPneu)),
    B1neu is B1 + 1,
    retract(person(T,KIND,4,B1)), asserta(person(T,KIND,3,B1neu)),
    asserta(gebunden(KIND,P)),
    Bneu is B - 1,
    retract(person(T,P,S,B)), asserta(person(T,P,S,Bneu)),!.
produziere(T,P,B) :-
    person(T,P,S,B),
    zahl_der_objekte(ZOB),
    OB is random(ZOB) + 1,
    B1 is B + (1/10)*OB,
    retract(person(T,P,S,B)), asserta(person(T,P,S,B1)),!.
arbeite(T,P,B) :-
    benutzte_personen(BP),
    ( member(P,BP)
    ;
    person(T,P,S,B),
    gebunden(P,P1),
    person(T,P1,1,B1),
    zahl_der_objekte(ZOB),
    OB is random(ZOB) + 1,
    Bneu is B + (1/10)*OB,
    Val1 is 0.2 * Bneu, Val2 is 0.8 * Bneu,
    B1neu is B1 + Val1, Bneu is B + Val2,
    retract(person(T,P1,1,B1)),
    asserta(person(T,P1,1,B1neu)),
    retract(person(T,P,S,B)),
    asserta(person(T,P,S,Bneu))
```

```
    ),!.
spiele(T,P,B) :- true.
erzeuge_neue_personen(T) :-
    zahl_der_personen(T,ZP),
    ZPneu is ZP + 2,
    T1 is T + 1,
    asserta(zahl_der_personen(T1,ZPneu)),
    P1 is ZP + 1, P2 is ZP + 2,
    asserta(person(T,P1,4,0.5)),
    asserta(person(T,P2,4,0.5)),
    verteilung(T,[Q1,Q2,Q3,Q4]),
    Q4neu is Q4 + 2,
    retract(verteilung(T,[Q1,Q2,Q3,Q4])),
    asserta(verteilung(T,[Q1,Q2,Q3,Q4neu])),
    statusklasse(4,T,KL4), append(KL4,[P1,P2],KL4neu),
    retract(statusklasse(4,T,KL4)), asserta(statusklasse(4,T,KL4neu)),!.
    rausschreiben :-
    append('res2.pl'),
    repeat, nl,
    ( zahl_der_personen(T,ZP), write(zahl_der_personen(T,ZP)), write('.'), nl,
    verteilung(T,VT), write(verteilung(T,VT)), write('.'), nl,
    statusklasse(SS,T,KL),
    write(statusklasse(SS,T,KL)), write('.'), nl, fail; true),
    told,
    asserta(hilfsliste([])),
    append('res1.pl'),
    repeat, nl,
    ( person(F1,F2,F3,F4), hilfsliste(LL40),
    append(LL40,[person(F1,F2,F3,F4)],LL40neu), retract(hilfsliste(LL40)),
    asserta(hilfsliste(LL40neu)), fail; true),
    hilfsliste(LL40neu),
    sort(LL40neu,Personenliste), length(Personenliste,Lae41),
    ( between(1,Lae41,X40), nth1(X40,Personenliste,PP),
    write(PP), write('.'), nl, fail; true), nl,
    told,
    retract(hilfsliste(LLL)),!.
sum(L,SUM) :-
    length(L,Lae),
    asserta(aux(0)),
    ( between(1,Lae,X), addaux(X,L), fail; true),
    aux(SUM), retract(aux(SUM)).
```

```
addaux(X,L) :-
    nth1(X,L,Y),
    aux(A), Anew is A + Y, retract(aux(A)), asserta(aux(Anew)),!.
```

Thema 2: Gerechtigkeit

Das Modell

Eine Gesellschaft (eine 'Gruppe' von Akteuren) befindet sich während einer Periode in verschiedenen Zuständen. Ein Zustand wird durch eine Menge von Ereignissen beschrieben. Diese Ereignisse werden in Dimensionen eingeteilt. Ein erstes Ereignis betrifft z.B. ein Naturphänomen, ein zweites die Ernte von Weizen und ein drittes den Streit zweier Nachbarn.

Die Akteure können verschiedene Handlungstypen ausführen. Eine Handlung des Typs h führt vom Zustand Z1 zu einem anderen Zustand Z2. Wenn die Handlung zur Dimension D1 gehört, kann das Resultat der Handlung zur Dimension D2 gehören.

Zwei Zustände lassen sich vergleichen; Zustand Z1 kann für die Gesellschaft besser oder schlechter als der Zustand Z2 sein.

In der Gesellschaft können verschiedene normative Regeln formuliert werden. Sehr abstrakt gesprochen führt eine Handlung, die mit Regel R in Einklang steht, von einem Zustand Z1 zu einem neuen Zustand Z2. Zustand Z2 kann für die Gesellschaft besser oder schlechter als Z1 sein. Bei einer Regel werden nur Handlungen eines bestimmten Handlungstyps beschrieben.

In der Gesellschaft werden verschiedene Regeln beachtet. Wenn in einer Periode diese Regeln zu immer besseren Zuständen führen, sagen wir, dass ein Zustand am Ende der Periode für die Gesellschaft gerecht ist. Wenn ein System von Regeln zu einem gerecht(er)en System führt, kann man auch verschiedene Regelsysteme vergleichen. Dies führt zur Frage, ob es ein optimales Regelsystem gibt.

Zwei viel diskutierte Ansätze stammen von Rawls und Sen. Rawls versucht eine Regel zu formulieren, die zu 'dem' besten Zustand führt. Sen dagegen meint, dass die vielen möglichen Regelsysteme mit der Zeit durch die Gesellschaft so angepasst werden (müssen), dass es jeweils zu besseren Zuständen kommen.

Als Einstieg für eine Simulation stellen wird die Zustände in sehr einfacher Form dar. Ein Zustand enthält drei Komponenten - drei Ereignisse aus drei Dimensionen, die einfach als Zahlen dargestellt werden. In der Simulation werden zwei Zustandsänderungen erzeugt. Im ersten Fall führt der Anfangszustand zum optimalen Zustand, wobei die Regel von Rawls benutzt wird. Im zweiten Fall werden zwei Pfade vom selben Zustand erzeugt. Im ersten Pfad wird eine Regel benutzt und im zweiten Pfad zwei nacheinander folgende Regeln. Die Endzustände der beiden Pfade werden verglichen.

Der Naturzustand hat im Modell die Form [1,1,1] und der optimale, beste Zustand die Form [6,6,6]. Ein Zustand [X1,Y1,Z1] ist besser als ein Zustand [X2,Y2,Z2], wenn gilt: X2 4 =< X1, Y2 =< Y1, Z1 =< Z2 und X1 + Y1 + Z1 < X2 + Y2 + Z2.

Wir kürzen ab:

- ZD ist die Zahl der Dimensionen
- D ist eine Dimension
- ZE ist die AnZahl der Ereignisse
- E ist ein Ereignis
- ZR ist die AnZahl von Regeln
- h1,...,h4 sind vier Handlungstypen (Konstante)
- LE ist eine Liste von Ereignissen (ein Ereignistyp).

Das Modell wird nun mit Hilfe von PROLOG konstruiert (siehe das Programm unten).

In Zeile 13 wird für jede Dimension ein Ereignistypen gebildet. In Zeile 24 wird für Dimension D eine Ereignisliste

LE

erzeugt. Dazu wird in 25 eine leere Liste von Ereignissen angelegt. In 26 wird eine Schleife über die Zahl der Ereignisse gelegt. Dabei werden die Zahlen

E

sowohl als Index für die Ereignisse als auch für die 'Namen' der Ereignisse verwendet. In 29 wird jeweils ein weiteres Ereignis E in die Liste LE eingefügt.

In 15 wird der Naturzustand

naturzustand([1,1,1])

definiert. Dabei sind die Zahlen 1,1,1 'Namen' für Ereignisse aus den drei Dimensionen. Jeder Name (Zahl) gehört zu einer der Dimensionen. Diese Namen lassen sich in unserer Formulierung nur durch die Ordnung der Dimensionen unterscheiden. Zum Beispiel 'ist' im Zustand [1,3,1] die erste Komponente ('1') ein Ereignis des Names 1 aus Dimension 1, die zweite Komponente ('3') von [1,3,1] ist ein Ereignis aus Dimension 2, und die dritte Komponente ('1') von [1,3,1] ist ein Ereignis aus Dimension 3.

In Zeile 16 wird die Regel von Rawls definiert:

rawls_regel([1,1,1],[6,6,6]).

Diese Regel besagt, dass der Naturzustand [1,1,1] immer zum optimalen Zustand [6,6,6] führt. Wie dies genauer geschehen soll, wird hier nicht erörtert, siehe die Texte von Rawls.

In 18 werden aus der Liste [1,2,...,6] von Ereignissen Zustände gebildet. In Zeile 104 wird durch die komplexe PROLOG-Klause

n_cart([[1,2,3,4,5,6],[1,2,3,4,5,6],[1,2,3,4,5,6]],MZU)

das 3-dimensonale, kartesische Produkt gebildet, siehe unsere Website

www.kuenstliche-gesellschaften.de

und dort die Urban-Regel.

MZU ist die Liste der Tripel [X,Y,Z], so dass sowohl X und Y als auch Z eine Komponente von [1,2,...,6] ist. MZU ist also die Menge aller möglichen Zustände. In

Zeile 106 wird die

liste_der_zustaende(MZU)

in die Datenbasis eingetragen und in 107 zur Resultatdatei

res2

geschickt.

In 19 und 34 werden vier Regeln definiert und ausgeführt. In 35 wird die Liste MZU der Zustände aus der Datenbasis geholt (diese Liste wurde vorher erzeugt). In 36 wird der 5-te Zustand Z1 aus Liste MZU geholt und genauer betrachtet:

Z1 = [X1,X2,X3]. In 37 wird

folge1([Z1])

in die Datenbasis eingetragen. folge1 ist der Folgezustand des Zustands Z1, der aus Regel Nr.1 entsteht. In 38 und 39 werden die Regeln

regel(1,...) und die regel(2,...) ausgeführt.

Regel 1 in 86 besagt folgendes. Die erste Komponente X1 des Anfangszustands Z1 befindet sich nicht in einem für die Gesellschaft optimal Zustand. Durch die 'besser-als' Relation ¡ lässt sich der optimale Zustand bestimmen. In dem hier formulierten Beispiel, wird der 'regionale' Zustand in einer gegebenen Dimension einfach durch eine Zahl ausgedrückt und diese Zahl enthält auch schon einen Wert. Dieser Wert besagt, wie 'gut' dieser regionale Zustand in der gegebenen Dimension für die Gesellschaft ist. Im Beispiel wird ein regionaler, optimaler, bester Zustand durch die Zahl 6 ausgedrückt. Im Beispiel sind all die regionalen 'besser-als' Relationen für alle Dimensionen identisch. Sie lässt sich ohne Mühe verbessern und verallgemeinern.

Aus [X1,X2,X3] X1 wird der Folgezustand [Y1,Y2,Y3] berechnet. Y1 = X1 + 1, Y2 = X2 und Y3 = X3.

In Regel 2 wird in ähnlicher Weise verfahren, siehe Zeile 90. In 40 wird die Wirkung der Regel 2, nämlich der Zustand Z3, in die Datenbasis eingetragen. In 41 werden die Zustände Z1, Z2 und Z3 in eine Liste

L15

zusammengefasst.

In 42 wird die folge1([Z1]) aus der Datenbasis gelöscht und stattdessen die Folge folge1(L15) eingetragen. D.h. das Resultat der Wirkung, das aus den aufeinander folgenden Zustände entstanden ist, wird gespeichert und als Resultat zur Datei

res1

geschickt. In 44 wird dieser Term aus der Datenbasis gelöscht. D.h. diese Information ist nur noch in der Resultatdatei res1 vorhanden.

In 45 und 46 werden in ähnlicher Weise die Regeln 3 und 4 aus dem Anfangszustand Z1 berechnet.

In 20 und 68 werden die Ergebnisse der zwei Anwendungsmethoden verglichen. In 69 werden die vorher erzeugten Wirkungen aus den Regeln: Regel 2, Regel 3 und Regel 4 aus der Datenbasis geholt. In 72 wird das Endresultat

Zb

mit dem Endresultat Zc verglichen, wobei dieses Resultat vom selben Anfangszustand Za kommen.

Dazu werden die Summen von Zahlen, die Ereignisse aus den Zuständen Z1 und Z2, darstellen, berechnet. Eine Summe können wir als Wert eines Zustands der Gruppe ansehen. Ein Zustand besteht aus drei Ereignissen, die als Zahlen dargestellt sind. Je größer eine Zahl ist, ist auch der Wert des regionalen Zustandes größer und besser für die Gruppe.

In 75 werden Summen S1 und S2 (die Werte der Zustände ZX und ZY) verglichen. In 77 sind die Summen gleich. D.h. beide Zustände, die aus demselben Anfangszustand durch verschiedene Regeln entstanden sind, haben denselben Wert für die Gruppe.

In 79 ist der Wert des Zustands ZX kleiner als der Wert des Zustands ZY. In diesem Fall hat die Anwendung der Regeln verschiedene Werte ergeben. Anders gesagt, führt die Rawls Regel zu einem anderen Ergebnis als die Regeln von Sen.

Das Progamm

```
dimensionen(3).
zahl_der_ereignisse(6).
zahl_der_regeln(4).
handlungstypen([h1,h2,h3,h4]).

start :-
    ( exists_file('res1.pl'), delete_file('res1.pl'); true),
    ( exists_file('res2.pl'), delete_file('res2.pl'); true),
    dimensionen(ZD),
    zahl_der_ereignisse(ZE),
    ( between(1,ZD,D), bilde_ereignisliste(D,ZE), fail; true),
    zahl_der_regeln(ZR),
    asserta(naturzustand([1,1,1])),
    asserta(rawls_regel([1,1,1],[6,6,6])),
    handlungstypen(HT),
    bilde_zustaende([1,2,3,4,5,6]),
    fuehre_regeln_aus,
    vergleiche,!.

bilde_ereignisliste(D,ZE) :-
    asserta(ereignisliste(D,[])),
    ( between(1,ZE,E), addiere_ereignis(E,D), fail; true),!.

addiere_ereignis(E,D) :-
    ereignisliste(D,LE), append(LE,[E],LEneu),
    retract(ereignisliste(D,LE)), asserta(ereignisliste(D,LEneu)),!.

fuehre_regeln_aus :-
    liste_der_zustaende(MZU),
```

```
    nth1(5,MZU,Z1), Z1= [X1,X2,X3],
    asserta(folge1([Z1])),
    regel(1,Z1,h1,Z2),
    regel(2,Z2,h2,Z3),
    asserta(wirkung_von_regel2(Z3)),
    append([Z1],[Z2,Z3],L15),
    retract(folge1([Z1])), asserta(folge1(L15)),
    append('res1.pl'), write(folge1(L15)), write('.'), nl, told,
    retract(folge1(LLL)),
    fuehre_regel3_aus(Z1),
    fuehre_regel4_aus(Z1),!.
fuehre_regel3_aus(Z1) :-
    asserta(folge3([Z1])),
    regel(3,Z1,h3,Z2),
    asserta(wirkung_von_regel3(Z2)),
    append([Z1],[Z2],L17),
    retract(folge3([Z1])), asserta(folge3(L17)),
    append('res1.pl'), write(folge3(L17)), write('.'), nl, told,
    retract(folge3(LLL2)),!.
fuehre_regel4_aus(Z1) :-
    asserta(folge2([Z1])),
    regel(4,Z1,h4,Z2),
    asserta(wirkung_von_regel4(Z2)),
    append([Z1],[Z2],L16),
    retract(folge2([Z1])), asserta(folge2(L16)),
    append('res1.pl'), write(folge2(L16)), write('.'), nl, told,
    retract(folge2(LLL1)),!.
vergleiche :-
    wirkung_von_regel2(Za),
    wirkung_von_regel3(Zb),
    wirkung_von_regel4(Zc),
    vergleiche(Za,Zb), vergleiche(Za,Zc),!.
vergleiche(ZX,ZY) :-
    sum(ZX,S1), sum(ZY,S2),
    append('res1.pl'),
    ( S1 = S2, write(S1 = S2), write('.'), nl ;
        S1 < S2, write(S1 < S2), write('.'), nl
    ;
        S2 < S1, write(S2 < S1), write('.'), nl
    ), told,!.
regel(1,Z1,h1,Z2) :-
```

```
    Z1 = [X1,X2,X3], X1 < 6, Y1 is X1 + 1, Y2 is X2,
    Y3 = X3, Z2 = [Y1,Y2,Y3],!.
regel(2,Z1,h2,Z2) :-
    Z1 = [X1,X2,X3], Y2 is X2 + 1, Y1 = X1,
    Y3 = X3, Z2 = [Y1,Y2,Y3],!.
regel(3,Z1,h3,Z2) :-
    Z1 = [X1,X2,X3], X1 < 5, Y1 is X1 + 3, Y2 = X2, Y3 = X3,
    Z2 = [Y1,Y2,Y3],!.
regel(4,Z1,h4,Z2) :-
    Z1 = [X1,X2,X3], Z2 = [6,6,6],!.
bilde_zustaende([1,2,3,4,5,6]) :-
    n_cart([[1,2,3,4,5,6],[1,2,3,4,5,6],[1,2,3,4,5,6]],MZU),
    /* Menge der Zustaende */
    asserta(liste_der_zustaende(MZU)),
    append('res2.pl'), write(liste_der_zustaende(MZU)), write('.'), nl,
    told,!.
cart([],_,[]):-!.
cart(_,[],[]):-!.
cart([A|ARest],BL,CL):-
    create_tuple(A,BL,TList),
    cart(ARest,BL,CInterim),
    append(TList,CInterim,CL).
create_tuple(_,[],[]).
create_tuple(A,[B|BRest],[[A,B]|TRest]):-
    create_tuple(A,BRest,TRest).
n_cart(NList,[]):- member(NList,[]),!.
n_cart(NList, undef):- length(NList,1),!.
n_cart([AL,BL|Rest],CL):-
    cart(AL,BL,C),
    extend_cart_tuple([C|Rest],CL).
extend_cart_tuple([C|[]],C).
extend_cart_tuple([TupleList,Next|Rest],CL):-
    help_extend_cart_tuple(TupleList,Next,ETupleList),
    extend_cart_tuple([ETupleList|Rest],CL).
help_extend_cart_tuple([],_,[]).
help_extend_cart_tuple([T|TupleList],List,ExtTupel):-
    extend_tuple(T,List,EList),
    help_extend_cart_tuple(TupleList,List,IList),
    append(EList,IList,ExtTupel).
```

```
extend_tuple(_,[],[]).
extend_tuple(T,[E|Rest],[ExtT|ERest]) :-
    append(T,[E],ExtT),
    extend_tuple(T,Rest,ERest).
sum(LL,S) :-
    length(LL,Lae),
    asserta(aux(0)),
    ( between(1,Lae,X), addiere(X,LL), fail; true),
    aux(S), retract(aux(S)),!.
addiere(X,LL) :-
    nth1(X,LL,K),
    aux(S1), S2 is S1 + K, retract(aux(S1)), asserta(aux(S2)),!.
```

Thema 3: Verantwortung

Das Modell

Eine Person ist verantwortlich für die Handlung, die sie ausführte, wenn andere Personen durch die Resultate der Handlung betroffen sind. Die Personen können in verschiedenen Moralsystemen leben, so dass eine Betroffenheit auch vom den Moralsystemen abhängt. Zum Beispiel macht eine Person einen Witz über Etwas, was bei einer anderen Person eine starke Betroffenheit auslöst.

Wir relativieren die Verantwortung auf ein gegebenes soziales System. Dieses enthält

- Personen,
- Ereignisse (und Ereignistypen),
- Handlungen (und Handlungstypen) und
- moralische Werte.

Als Werte können einfache Unterscheidungen - wie z.B. gut, falsch, offen - benutzt werden.

In dem System ist Kausalität vorhanden. Eine Handlung ist eine Teilursache einer anderen. Eine Person (ein Akteur) hat neben ihrem Namen auch ein Moralsystem, in dem sie lebt.

In einem sehr einfachen System werden genau zwei Personen beschrieben. Eine Handlung H wird durch den Handelnde A ausgeführt und die Person B ist durch die Handlung betroffen. Die Vorbereitung der Handlung, die Handlung H selbst und eine Wirkung W der Handlung wird beschrieben. Auch der Betroffene B führt eine Handlung aus, die an die Wirkung W von H anschließt. Je nach den Moralsystemen der Personen werden sie sich entsprechend äußern. Beide Personen werden mitteilen, welche Werte sie der Wirkung zuschreiben. Dies führt zu vielen möglichen Konstellationen. Beide können gleiche oder ähnliche Werte bezüglich der Wirkung haben, sie können auch der Wirkung völlig unterschiedliche Werte zuweisen - was zu Streit

führt.

Zu diesem einfachen Programm lässt sich die Zeit hinzunehmen und Verantwortungen von verschiedenen Personen darstellen.

In diesem Modell bleiben Handlungen, Moralsysteme und Akteure abstrakt. Sie könnten diese in Ihrer Weise füllen.

- ZMS ist die AnZahl der MoralSysteme
- ZA ist die AnZahl der Akteure
- ZE ist die AnZahl der Ereignisse
- ZH ist die AnZahl der Handlungen
- g,o,f,toll sind Konstante
  g bedeutet 'gut'
  o bedeutet 'offen' (oder unklar)
  f bedeutet 'falsch'
  toll bedeutet 'toll'

LW ist die Liste der Moralwerte.

In Zeile 3 wird die Anzahl ZMS der Moralsysteme, in 4 die Anzahl ZA der Akteure, in Zeile 5 die Anzahl von Ereignissen und in Zeile 6 die Anzahl der Handlungen festgelegt.

In 7 werden drei Moralwerte

m_werte

festgelegt. Zu jeder Handlung wird ein Moralwert zugeordnet. Eine Handung ist z.B. in einer Situation gut.

In Zeile 8 werden genau 2 Akteure (Personen) ins Spiel gebracht. Die Zahlen 1 und 2 benutzen wird auch als Namen für die beiden Personen.

In Zeilen 14 - 17 wird die Anzahl ZMS der Moralsysteme, die Zahl ZA der Akteure, die Zahl ZE der Ereignisse und die Zahl ZH der Handlungen aus der Datenbasis geholt, wo sie in Zeilen 3 - 6 hinterlegt sind. In 18 wird die Liste

LW

der Moralwerte aus der Datenbasis geholt.

In 19 werden die Ereignisse 'konkretisiert'. Die geschieht in der Klause, die in Zeile 29 beginnt. Dort wird eine Schleife über die Zahl der Ereignisse gelegt. Im Schleifenschritt Nummer N beginnt eine weitere Klause. In 33 wird eine Zufallszahl X aus dem Bereich 1,...,100 gezogen und in 34 durch 100 dividiert. Die Zahl

N

wird nun als ein Name verwendet.

WN

ist ein neu erzeugtes Ereignis. N ist also ein Name für ein neu erzeugtes Ereignis WN. Dieses Ereignis wird in Zeile 35 in die Datenbasis eingetragen. erg(N,WN) bedeutet: WN ist ein Ereignis, das den Namen N trägt. Wir haben damit eine Menge von neuen Ereignisse generiert.

In Zeilen 20 und 39 werden potentielle Handlungen gebildet. In Zeile 40 wird eine leere Hilfsliste eingerichtet. In 41 wird eine Schleife über die Zahl ZH der Handlungen gelegt. In 43 wird in einem Schleifenschritt eine potentielle Handlung

H

gebildet.

In 44 beginnt mit dem Befehl 'repeat' eine offene Schleife. In 45 werden zwei Zufallszahlen U und W aus dem Bereich 1,2,3,...,ZE gezogen.

U nennen wir eine Ursache und

W eine Wirkung für eine Handlung des

Names H.

In 46 wird untersucht, ob U mit W identisch ist. Wenn dies nicht der Fall ist, geht PROLOG zu 47, und wenn U = W ist, wird PROLOG wieder zu 44 zurückspringen und in 45 neue Zufallszahlen ziehen.

Wenn die Ursache von der Wirkung verschieden ist, wird in 47 die Hilfsliste

AUXL

geholt. In 48 werden U,H und W zu einer Liste

PH

zusammengefügt. PH steht für 'potentielle Handlung'. In 49 wird untersucht, ob PH schon in der Hilfsliste zu finden ist, wenn ja, geht PROLOG wieder zu repeat (Zeile 44).

Wenn PH nicht in der Hilfsliste steht, wird [U,H,W] (= PH) als neue potentielle Handlung in die Datenbasis eingetragen:

pot_handlung(U,H,W)

und diese Information wird auch zur Resultatdatei

res1

geschickt. Damit ist die Klause 39 beendet. PROLOG springt zurück zu 20.

PROLOG findet als nächstes die Zeile 21. Dort und in 56 werden Kausalitäten gebildet. In 57 wird eine Schleife über die 'Namen' der Handlungen gelegt. In einem Schleifenschritt Nummer

H

wird in 59 der

loop2

ausgeführt. D.h. die Nummer H wird in dem Programm mit einer Handlung identisch. Die Handlung des Namens 'ist' gleichzeitig auch eine Handlung. Dies Handlung lässt sich ohne Mühe durch ein zusätzliches Modul programmieren und mit Inhalt füllen.

In 60 wird die potentielle Handlung

[U,H,W]

geholt. In 61 werden zwei Kausalbeziehungen eingetragen. U, die Ursache von H

bewirkt H:

caus(U,H),

und H bewirkt die Wirkung W:

caus(H,W).

Diese Informationen werden in die Datenbasis eingetragen und zur Resultatdatei

res1

geschickt. Damit ist die Klause 56 beendet und PROLOG springt zu Zeile 21 zurück.

PROLOG findet als nächste Zeile 22 und dann die Zeile 67. In 68 wird eine Schleife über die Akteure gelegt. In 70 wird der Akteur

A

untersucht. In 71 wird die Resultatdatei res1.pl geöffnet. Hier werden zwei Fälle unterschieden.

Wenn A einer der erste drei Akteure ist, wird in die Datenbasis eingetragen, dass A im Moralssystem Nummer 1 lebt. Im zweiten Fall hat A einen 'Namen' größer als 3. In diesem Fall lebt A im zweiten Moralsystem 2. Diese Information wird auch in die Resultatdatei geschickt. Danach geht PROLOG zurück zu 22.

In 23 und 82 werden 'konkrete Handlungen' gebildet. Diese enthalten auch jeweils eine moralisch bewertete Handlung. In 83 und 85 wird eine Schleife über die Akteure A gelegt.

In 86 wird eine Liste

LIST

von Handlungen H berechnet. Der Term

pot_handlungen(U,H,W)

beschreibt eine potentielle Handlung. Der Term pot_handlungen(U,H,W) findet sich in der Datenbasis.

U ist eine Ursache der Handlung H und W ist eine Wirkungen von H.

In 87 wird A als Index für einen Akteur benutzt. Die A-te Komponente

HA

aus

LIST

wird bestimmt; HA ist eine potentielle Handlung. In 88 wird geschaut, in welchem Moralsystem MS A lebt.

In 89 wird nun die Handlung von A bewertet. In 99 wird eine der drei moralischen Werte

B

aus der Liste

LW

der Werte zufällig gezogen und in 105 in die Resultatdatei res1 eingetragen. Im Term

bew(MS,H,B)

ist H ein Name für eine Handlung, MS ist ein Name für ein Moralssystem und

B ein Wert aus der Liste [w,o,f].

Wenn zum Beispiel die zweite Möglichkeit zufällig gezogen wurde (K = 2), hat B die Form o:

B = o.

Der Term bew(MS,H,B) wird nun in die Datenbasis eingetragen.

bew(MS,H,B)

besagt, dass die Handlung H im Moralsystem MS der Wert B hat. Also beispielweise finden wir im Ablauf des Programms etwa bew(2,5,f); im Moralsystem MS = 2, wird die Handlung des Names 5 als falsch (f) bewertet.

Danach springt PROLOG zu 89 und zu 90. PROLOG nimmt ausgehend von der Handlung

HA

die ersten Ursachen und Wirkungen U und W, die in der Datenbasis zu finden sind. In 91 wird die so vervollständigte Handlung

handle(A,MS,U,HA,W,B)

in die Datenbasis eingetragen und in die Resultatdatei res1 geschickt.

Wenn eine der Bedingungen in 88 oder 89 nicht gilt, wird keine Handlung erzeugt. In diesem Fall kommt PROLOG zu 94, beendet die Klause und springt zu 83. Am Ende sind die Klausen 85 und 82 beendet und PROLOG springt zu 23.

In 24 holt PROLOG die beiden Akteure

Aa und Ap

aus der Datenbasis. Diese Akteure wurden am Anfang in Zeile 8 oben eingetragen. Aa ist ein 'Name' für einen aktiven Akteur und Ap ist ein 'Name' für einen passiven Akteur. Schließlich wendet sich PROLOG in Zeile 25 zur letzten Klause zu.

Von 25 kommt PROLOG zu 109. In 109 wird die Antwort gegeben.

Wir haben hier für zukünftige Erweiterungen, Antworten sowohl für Aa als auch für Ap formuliert.

In 110 und 111 werden die zwei konkreten Handlung geholt.

Aa hat die Handlung Ha und
Ap die Handlung Hp

ausgeführt. Dabei sind alle weiteren Bedingungen bekannt.

Aa lebt im Moralssystem des Names

MSa. Ua ist eine(!) Ursache von Ha und Wa eine(!) Wirkung von Ha.

Weiter ist bekannt, dass Aa die Handlung bewertet. Der Wert der Handlung ist Ba. Im Programmablauf ist Ba schon instantiiert. Ba kann z.B. 'gut' bedeuten: Ba = g.

In dieser Klause gibt es 10 Möglichkeiten, wie die Werte Ba und Bp aussehen können.

Im ersten Fall sind z.B. beide Werte 'gut':

Ba = g = Bp.

In diesem Fall sind beiden Akteure zufrieden. Aa freut sich, dass er seine Handlung ausgeführt hat und Ap freut sich über die Wirkung Wa der Handlung Aa auch für Ap.

Die Antwort lautet

antwort(Aa,Ap,toll).

Sowohl Aa als auch Ap sagen, dass sie die Handlung Ha und die Wirkung Wa 'toll' finden. In Zeile 9, hatten wir dieses Wort als Konstante als Fakt eingetragen.

Im dritten Fall in Zeile 122, findet Aa seine Handlung gut:

Ba = g, während

Ap die Handlung Aa und die Wirkung Wa schlecht findet:

Bp = f.

Aa könnte z.B. antworten: antwort(Aa,Ap,'Entschuldigung'). Hier haben wir die Antwort nicht als Konstante formuliert, sondern das Wort Entschuldigung in Anführungszeichen gesetzt: PROLOG findet den Term 'Entschuldigung'. Wenn wird an dieser Stelle den Befehl 'write(...)' dazwischen schieben und den Term 'Entschuldigung' als Argument hineinschreiben: write('Entschuldigung'), wird im Programmmodus auf dem Bildschirm sofort zu sehen sein: ... Entschuldigung ...

Die weiteren Möglichkeiten haben wir nicht weitere beschrieben. Die Antworten können Sie leicht vervollständigen.

Wir erwähnen noch, dass wir im vierten Fall in Zeile 127 3 Möglichkeiten durch 'oder' (also durch den PROLOG ' ; ') zusammengefasst haben.

Das Programm: Verantwortung

```
zahl_der_moralsysteme(2).
zahl_der_akteure(6).
zahl_der_ereignisse(18).
zahl_der_handlungen(12).
m_werte([g,o,f]).
zwei_akteure(1,2).
toll.

start :-
    ( exists_file('res1.pl'), delete_file('res1.pl'); true),
    ( exists_file('res2.pl'), delete_file('res2.pl'); true),
    zahl_der_ereignisse(ZE),
    zahl_der_akteure(ZA),
    zahl_der_handlungen(ZH),
```

```
    zahl_der_moralsysteme(ZMS),
    m_werte(LW),
    bilde_ereignisse(ZE),
    bilde_potentielle_handlungen(ZH,ZE),
    bilde_kausalitaet(ZH),
    bilde_akteure(ZMS,ZA),
    bilde_konkrete_handlungen(ZA,ZH,LW),
    zwei_akteure(Aa,Ap),
    antworte(Aa,Ap).
bilde_ereignisse(ZE) :-
    ( between(1,ZE,N), mache_ein_ereignis(N), fail; true),!. mache_ein_ereignis(N) :-
    X is random(100) + 1,
    WN is X/100,
    asserta(erg(N,WN)),!.
bilde_potentielle_handlungen(ZH,ZE) :-
    asserta(hilfsliste([])),
    ( between(1,ZH,H), bilde_pot_handlung(H,ZE), fail; true),!.
    bilde_pot_handlung(H,ZE) :-
    repeat,
    U is random(ZE) + 1, W is random(ZE) + 1,
    not U = W,
    hilfsliste(AUXL),
    PH = [U,H,W],
    not member(PH,AUXL),
    asserta(pot_handlung(U,H,W)), append('res1.pl'), write(potentielle_handlung(U,H,W)),
    write('.'), nl,
    told,!.
bilde_kausalitaet(ZH) :-
    ( between(1,ZH,H), loop2(H), fail; true),!.
loop2(H) :-
    pot_handlung(U,H,W),
    asserta(caus(U,H)), asserta(caus(H,W)),
    append('res1.pl'), write(caus(U,H)), write('. '),
    write(caus(H,W)), write('.'), nl, told,!.
bilde_akteure(ZMS,ZA) :-
    ( between(1,ZA,A), mache_einen_akteur(A), fail; true),!.
mache_einen_akteur(A) :-
    append('res1.pl'),
    ( A =< 3, asserta(lebt(A,ms(1))),
    write(actor(A,moralssysteme1)), write('.'), nl
    ;
```

```
    3 < A, asserta(lebt(A,ms(2))),
    write(actor(A,moralssysteme2)), write('.'), nl
    ), told,!.
bilde_konkrete_handlungen(ZA,ZH,LW) :-
    ( between(1,ZA,A), mache_eine_handlung(A,ZH,LW), fail; true),!.
mache_eine_handlung(A,ZH,LW) :-
    ( findall(H,pot_handlung(U,H,W),LIST),
    nth1(A,LIST,HA),
    lebt(A,ms(MS)),
    bewerte(HA,MS,LW,B),
    pot_handlung(U,HA,W),
    asserta(handle(A,MS,U,HA,W,B)),
    append('res1.pl'), write(handle(A,MS,U,HA,W,B)),
    write('.'), nl, told
    ; fail
    ),!.
bewerte(H,MS,LW,B) :-
    K is random(3) + 1,
    ( K = 1, nth1(K,LW,W1), B = W1;
    K = 2, nth1(K,LW,W2), B = W2;
    K = 3, nth1(K,LW,W3), B = W3
    ),
    append('res1.pl'), write(bew(MS,H,B)), write('.'), nl, told,!.
antworte(Aa,Ap) :-
    handle(Aa,MSa,Ua,Ha,Wa,Ba),
    handle(Ap,MSp,Up,Hp,Wp,Bp),
    ( Ba = g, Bp = g,
    asserta(antwort(Aa,Ap,toll)),
    append('res2.pl'), write(antwort(Aa,g,Ap,g,toll)),
    write('.'), nl
    ;
    Ba = g, Bp = o,
    asserta(antwort(Aa,Ap,'kein Bedarf')),
    append('res2.pl'), write(antwort(Aa,g,Ap,o,'kein Bedarf')), write('.'),
    nl
    ;
    Ba = g, Bp = f,
    asserta(antwort(Aa,Ap,'Entschuldigung')),
    append('res2.pl'), write(antwort(Aa,g,Ap,f,'will Wiedergutwachung')), write('.'),
    nl
    ;
```

```
Ba = o, (Bp = g; Bp = o; Bp = f),
asserta(antwort(Aa,Ap,'merkwuerdig')),
append('res2.pl'), write(antwort(Aa,o,Ap,gxoxf,'merkwuerdig')), write('.'),
nl
;
Ba = f, Bp = g,
asserta(antwort(Aa,Ap,'cool')),
append('res2.pl'), write(antwort(Aa,f,Ap,g,'cool')), write('.'),
nl
;
Ba = f, Bp = o,
asserta(antwort(Aa,Ap,'soso')),
append('res2.pl'), write(antwort(Aa,f,Ap,o,'soso')), write('.'), nl
;
Ba = f, Bp = f,
asserta(antwort(Aa,Ap,'zum mir leid')),
append('res2.pl'), write(antwort(Aa,f,Ap,f,'tut mit leid')), write('.'),
nl
), told,!.
```

Thema 4: Reichtum

Das Modell

Eine Holding ist ein Unternehmen, welche viele weitere Subunternehmen beherrscht und kontrolliert. Im deutschen Aktienrecht gibt es den vielsagenden Term 'Tochtergesellschaft'. In Unternehmen gibt es Personen, die das Unternehmen leiten.

Die Holding 'kauft' und 'verkauft' in einer Periode andere Unternehmen mit dem Ziel, das Kapital der Holding zu vermehren. Die Holding 'kauft' ein Unternehmen, indem sie mit dem anderen Unternehmen einen Vertrag schließt - das Unternehmen ist nun ein Subunternehmen der Holding. Wenn der Vertrag gekündigt wird, verkauft die Holding das Unternehmen wieder.

In einer einfache Simulation gibt es mehrere Akteure. Der erste Akteur ist die Holding. Zwei weitere Akteure - Subunternehmen - können in einer Periode ausgewechselt werden.

Ein Periode besteht aus Zeitpunkten (Tics), zu denen die Holding Verträge mit Subunternehmen abschließt und kündigt. In einer Perioden müssen die Subunternehmen Erträge erwirtschaften. Wenn ein Subunternehmen die Gewinnerwartungen der Holding erfüllt, wird der Vertrag für den nächsten Anfangszeitpunkt der nächsten Periode verlängert. Andernfalls wird dem Subunternehmen gekündigt. In jedem Endzeitpunkt einer Periode überweist das Subunternehmen einen gewissen Betrag an die Holding.

Das kleine Programm kann an vielen Stellen spezieller und realistischer formuliert

werden.

Unser Modell enthält 100 mögliche Akteure, mindestens 2 Subunternehmer, 20 Tics und 1000 Kapitaleinheiten.

- ZMA ist die AnZahl der Möglichen Akteure (am Anfang: ZMA = 100)
- LMS ist eine AnZahl von Möglichen Subunternehmen
- TT ist die AnZahl der Tics (TT = 20)
- chef ist eine Konstante (der Name der Holding)
- 1000 ist eine Konstante (das Grundkapital des Systems)
- 100 ist eine Konstante (der anfängliche Wert jedes Subunternehmens)
- 3 ist eine Konstante (die VertragsDauer für ein Subunternehmen)
- 1/10 ist eine Konstante (der Zinssatz (d.h. 10 Prozent), die jedes Subunternehmen zu zahlen hat)
-15 ist eine Konstante (sie drückt die Gewinnerwartung aus, die ein Subunternehmen höchstens haben kann.

Die Holding hat hier den Namen 'chef'. Der Chef ist in Zeile 4 als Konstante eingetragen. In Zeilen 5 und 6 sind die Anzahl ZMA der möglichen Akteure und die Anzahl

TT der Tics

festgelegt. In 8 werden drei weitere Konstanten festgelegt, die für die Vereinbarungen mit den Subunternehmen benutzt werden. 100 ist der anfängliche Wert, der ein Subunternehmen hat. Dieser Wert ändert sich im Ablauf. 3 ist die Vertragsdauer (z.B. 3 Jahre oder 3 Monate), Die Vertragsdauer wird im Programm auch variable benutzt. Die dritte Zahl (1 dividiert durch 10) ist der Zins ('10 Prozent'). Jedes Subunternehmen muss per Vertrag nach jeder Periode (Tic) an die Holding 10 Prozent ihres Gewinns an die Holding abliefern. Die Zahl 14 in Zeile 10 ist die Gewinnerwartung, die ein Subunternehmen höchstenfalls erreichen kann.

In den Zeilen 15 und 25 - 29 werden Akteure gebildet. Der chef (wie in Zeile 4 festgelegt) 'ist' quasi die Holding. Er besitzt ein Kapital von 994 Einheiten (z.B. Euro).

subu1

ist der erste Subunternehmer und

subu2

der zweite Subunternehmer.

Beide halten jeweils ein Kapital von 3 Einheiten. All dies wird durch asserta(...) in die Datenbasis eingetragen. Am Anfang wird auch die Liste von Subunternehmern eingetragen. '1' ist hier der erste Tic in einem Simulationslauf.

In 16 wird die Zahl der möglichen Akteure aus der Datenbasis geholt, siehe Zeile 5. In 17 werden zu den drei schon vorhandenen Akteuren weitere mögliche Akteure generiert.

Dies geschieht in einer Klause, die mit Zeile 32 (und Zeile 17) beginnt. In 33 wird eine Liste [ ] möglicher Akteuren gebildet. Diese Liste wird so notiert:

liste_der_msubus([]).

Diese Liste ist zunächst leer. In 34 wird eine Schleife über die Anzahl ZMA der möglichen Akteure gelegt. Dies führt zu einer Klause, die in Zeile 39 beginnt.

In 39 wird ein möglicher Akteur

A

erzeugt. Dazu wird in 40 die

liste_der_msubus(LMS)

geholt. LMS ist eine Abkürzung für 'Liste der möglichen Subunternehmer'. LMS ist eine Variable, die sich im Prozess ändern kann.

In 41 wird der Term 'subu' mit dem Symbol 'A' (für eine Variable) konkateniert. Das Resultat ist die Symbolreihe

'subuA'.

subuA ist dann der 'Name' für den Subunternehmer A.

Im Programmablauf ist A variable, das gilt dann auch für den Term subuA. D.h. PROLOG benutzt den Namen subuA auch als eine Variabe

SUBAK.

Mit anderen Worten wird das Resultat der Konkatenation von subu und A variable behandelt und durch die Variable SUBAK Ausgedrückt. SUBAK kann man noch anders ausgedrückt als einen variablen Namen für einen möglichen Akteure betrachten.

In 42 wird der gerade im Programm verwendete Akteur durch den Term

asserta(akteur(SUBAK,3))

in die Datenbasis eingetragen. 3 besagt, dass SUBAK das Kapital 3 besitzt. In 43 wird mit dem Befehl 'append' der Liste LMS ein weiteres Element SUBAK hingefügt. Die resultierende Liste wird

LMSneu

genannt. In 44 wird mit dem Befehl 'retract' die Liste LMS aus der Datenbasis entfernt und mit 'asserta' in die neue Liste LMSneu eingetragen. Danach springt PROLOG zurück in Zeile 35.

Dort nimmt PROLOG die Liste LMS1 und schreibt die liste_der_msubus(LMS1) in die Resultatdatei res2.

Die Variable LSM, die in 40 zum ersten Mal verwendet wurde, hat sich in der Schleife ständig geändert: aus LMS wird LMSneu. Inhaltlich wird real gesehen eine konkrete Liste konstruiert. Am Anfang steht die leere Liste. In 43 wird diese mit einem neuen Element ergänzt. Im letzten Schleifenschritt in 45 steht die Variable LMSneu dann für eine Liste [1,2,3,....,100], weil 100 in Zeile 5 so festgelegt wurde. In 36 repäsentiert die Variable LMS1 die vollständig erzeugte Liste von möglichen Akteuren.

Die Klause in 32 ist nun beendet, daher spring PROLOG zur nächsten Zeile 18. Das Kapital wird aus der Datenbasis geholt.

KAP

ist eine Abkürzung für 'Kapital'. Mit der Zeit ändert sic das Kapital KAP des Systems.

In 19 wird eine Liste von Subunternehmern angelegt, die im Zeitpunkt 1 mit der Holding unter Vertrag stehen. Zunächst ist diese Liste leer.

In 20 wird die Zahl der Tics TT aus der Datenbasis geholt, die in Zeile 6 am Anfang festgelegt wurde: TT = 20. In 21 wird eine Schleife über die Tics gelegt. Diese Klause beginnt in Zeile 49 und Ende in 70.

In 50 wird die

liste_der_subus(T,LSU)

zum Zeitpunkt T geholt. LSU ist eine Liste von Sub-Unternehmern. In 53 wird die Länge der Liste LSU bestimmt. Diese Länge ist durch

'Lae'

notiert. In 54 wird die

liste_subus_in_vertrag(T,LIV)

geholt, die in Zeile 19 angelegt wurde. LIV ist eine Liste von Subunternehmern unter Vertrag. In 55 wird die

vertrags_konstante(W,VD,ZINS)

geholt, die in Zeile 8 festgelegt wurde. W steht für den Wert (W = 100) eines Subunternehmers, VD für die Vereinbarungsdauer (VD = 3) und ZINS für den Prozentsatz (10 Prozent), der vom Gewinn, an die Holding gezahlt werden soll.

In 56 wird eine Schleife über die Zahl Lae von Subunternehmer gelegt. U ist hier eine Variable für (Sub)-Unternehmer. Die restlichen Variablen und deren Abkürzungen wurden schon erklärt. Diese Schleife wird in der Klause bearbeitet, die in 74 beginnt.

In 75 - 79 werden verschiedene Informationen geholt: der chef, sein Kapital KC, der Unternehmer AU und sein Kapital KU1, und zu Tic T die Liste LIV1 der Subunternehmerder, die in Vertrag stehen. In 80 - 82 wird eine 'oder'-Konstruktion bearbeitet.

Im ersten Fall in 80 findet sich der Subunternehmer AU in der Liste LIV1 (member(_,_) ist ein PROLOGprädikat). In diesem Fall passiert zu Tic T nichts ('true' ist ein weiteres PROLOGprädikat).

Im zweiten Fall ist Subunternehmer AU noch nicht unter Vertrag. In 82 wird zu Tic T ein Vertrag zwischen dem chef und dem möglichen Subunternehmer AU geschlossen (W,VD,ZINS wurden schon erklärt).

82 führt zu einer Klause, die mit 89 beginnt. In 90 - 91 werden Informationen geholt: das Kapital KC1 des chefs, KU2 das Kapital des Subunternehmers AU.

In 92 und 93 wird ein Teil - nämlich W = 100 - des Kapitals KC1 des chefs an den Subunternehmers AU gegeben. Die beiden neuen Kapitalbestände werden in die Resultatdatei eingetragen. In 97 wird der neue Subunternehmer in die Liste LIV2 eingefügt; LIV2 wird upgedatet zu LIV2neu. Am Ende dieser Klause springt PROLOG zurück zu 84.

In 84 findet PROLOG eine Klause

arbeite(AU,T),

die in Zeile 105 beginnt. In 106 wird die Konstante

GE, die gewinnerwartung(GE)

geholt, die in Zeile 10 festgelegt wurde. In 107 wird eine Zufallszahl

G

gezogen. G liegt im Bereich zwischen 1,...,GE + 1. Die Zahl G drückt den Gewinn aus, den der Subunternehmer AU zu T erwirtschaftet hat. Danach springt PROLOG zu 84 zurück.

In 85 wird eine weitere Klause bearbeitet; sie beginnt in Zeile 112. Dort wird in 113 der Gewinn G, der vorher 'erwirtschaftet' wurde, geholt. In 114 wird der Zins berechnet. W ist der Wert des Subunternehmers AU und

ZINS ist der Zins,

der in 8 festgelegt wurde. ZI ist die Dividende, die AU zu T zahlen muss. In 115 - 117 wird eine 'oder'-Konstruktion benutzt.

Im ersten Fall ist der Gewinn kleiner als der Zins ZI. In diesem Fall überweist der Subunternehmer AU seinen ganzen Gewinn an die Holding.

Im zweiten Fall ist der Gewinn G nicht kleiner als der Zins ZI. In diesem Fall überweist AU nur den Zinsbetrag ZI an den chef, d.h. die Holding (siehe 125). In 119 und 120 werden einige Informationen an die Resultatdatei

res1

geschickt. Am Ende dieser Klause springt PROLOG zu Zeile 85, die damit auch bearbeitet ist. PROLOG springt dann zurück zu 74 und 57. Auch diese Klause (und Schleife) ist beendet. PROLOG kommt zu Zeile 58.

Dort wird eine Hilfsliste

aux([])

eingerichtet. In 59 wird eine weitere Schleife über die Länge

Lae

der Liste der Subunternehmer gelegt. In 134 ist

U1

ein variabler Subunternehmer.

In 135 - 136 wird ein Subunternehmer AU aus der Liste LSU und der Gewinn GU von AU zu Tic T geholt. In 137 wird der Zinsbetrag berechnet. In 138 - 144 gibt es zwei Fälle.

Im ersten Fall in 138 ist der Gewinn GU kleiner als der Zinsbetrag. In diesem Fall wird die Hilfsliste

LL

geholt und der Subunternehmer in diese Liste eingetragen. Diese Liste wird upge-

datet. Die Liste enthält diejenigen Subunternehmer, die nicht den erwarteten Gewinn erwirtschaftet könnten.

Dann wird der chef und sein Kapital KC6 geholt und um 100 Einheiten vergrößert und upgedatet. Weiter wird in 142 der Vertrag für den Subunternehmer AU gekündigt. Dies führt zu Klause 149.

In 149 und 150 wird der Subunternehmer AU und sein Kapital KU4 geholt. In 151 wird die Liste LIV1 der Subunternehmer geholt, die zu T gerade unter Vertrag stehen. In 152 wird der Subunternehmer AU aus dieser Liste gelöscht und diese Liste wird upgedatet. In 154 wird eine weitere Liste LSU1 von Subunternehmern geholt. Auch aus dieser Liste wird AU in 155 entfernt und diese Liste wird in 156 und 157 upgedatet. In 158 wird dieser Subunternehmer auch als Akteure entfernt. In 159 wird die Liste LMS12 der möglichen Subunternehmer geholt. Auch dort wird in 160 AU entfernt und diese Liste upgedatet.

Dann wendet sich PROLOG dem zweiten Fall in 144 zu. Der Gewinn von AU zu T ist nicht kleiner als der Zins ZI. In diesem Fall passiert nichts. Damit ist Klause 134 zu Ende; PROLOG springt zu 60. In 61 wird die Hilfsliste

LL

geholt. In 62 und 166 werden neue Subunternehmer generiert.

In 167 und 168 werden die Listen LMS1 und LSU geholt. Hier wird zunächst in 169 der Fall bearbeitet, bei dem es keine weiteren möglichen Subunternehmer mehr gibt. In diesen Fall, ist die Klause 166 zu Ende.

Wenn es noch mögliche Subunternehmer gibt, werden zwei Fälle unterschieden.

Im ersten Fall in 171 besteht die Liste LL, die in 61 und 62 generiert wurde, genau aus einem Subunternehmer AGENT. Dieser wird in 172 aus der Liste LMS1 gelöscht und LSM1 upgedated. Dann wird AGENT auch aus der Liste LSU gelöscht. In der so geänderten Liste wird in 176 ein weitere Agent A1 aus LMS1neu geholt, der durch Konstruktion vorhanden ist. A1 wird in die Liste LSUneu eingefügt.

Im zweiten Fall in 181 gibt es in der Liste LL zwei SubunternehmerA1 und A2. In 182 bis 187 werden beide Subunternehmer A1 und A2 aus der Liste LMS1 und aus der Liste LSA gestrichen. Am Ende dieser Klause springt PROLOG zurück zu 62.

In 63 wird die Hilfsliste LL gelöscht. In 64 wird die Liste LSU2 der Subunternehmer zu Tic T geholt. In 5 wird der Tic T um 1 erhöht: T1 = T + 1. Die Liste LSU2 für Tic T wird gelöscht und für Tic T1 eingetragen. Schließlich wird auch die Liste LIV3 der Subunternehmer, die unter Vertrag stehen, von T zu T1 gebracht.

Damit ist das Programm beendet. Wir können die Resultatdateien res1 und res2 betrachten und genauer analysieren.

Das Programm

```
chef.
zahl_der_moeglichen_akteure(100).
tics(20).
kapital(1000).
```

```
vertrags_konstante(100,3,1/10).
    /* W = Werte, VD=Vereinbarungsdauer, Z=Zins */
gewinnerwartung(15).

start :-
    ( exists_file('res1.pl'), delete_file('res1.pl'); true),
    ( exists_file('res2.pl'), delete_file('res2.pl'); true),
    bilde_akteure,
    zahl_der_moeglichen_akteure(ZMA),
    bilde_moegliche_akteure(ZMA),
    kapital(KAP),
    asserta(liste_subus_in_vertrag(1,[])),
    tics(TT),
    ( between(1,TT,T), bearbeite_einen_tic(T,chef), fail; true),!.

bilde_akteure :-
    asserta(akteur(chef,994)),
    asserta(akteur(subu1,2)),
    asserta(akteur(subu2,3)),
    asserta(liste_der_subus(1,[subu1,subu2])),!.

bilde_moegliche_akteure(ZMA) :-
    asserta(liste_der_msubus([])),
    ( between(1,ZMA,A), bilde_einen_msubu(A), fail; true),
    liste_der_msubus(LMS1),
    append('res2.pl'), write(liste_der_msubus(LMS1)), write('.'),
    nl, told,!.

bilde_einen_msubu(A) :-
    liste_der_msubus(LMS),
    concat(subu,A,SUBAK),
    asserta(akteur(SUBAK,3)),
    append(LMS,[SUBAK],LMSneu),
    retract(liste_der_msubus(LMS)),
    asserta(liste_der_msubus(LMSneu)),!.

bearbeite_einen_tic(T,chef) :-
    liste_der_subus(T,LSU),
        /* Liste der SubUnternehmer */
    append('res1.pl'), write(liste_der_subus(T,LSU)), write('.'), nl, told,
    length(LSU,Lae),
    liste_subus_in_vertrag(T,LIV),
    vertrags_konstante(W,VD,ZINS),
    ( between(1,Lae,U), loop(U,T,chef,LSU,W,VD,ZINS), fail; true),
    asserta(aux([])),
    ( between(1,Lae,U1), loop1(U1,T,chef,LSU,W,VD,ZINS), fail; true),
```

```
    aux(LL),
    finde_neue_subus(T,LL),
    retract(aux(LL)),
    liste_der_subus(T,LSU2),
    T1 is T + 1,
    retract(liste_der_subus(T,LSU2)),
    asserta(liste_der_subus(T1,LSU2)),
    liste_subus_in_vertrag(T,LIV3),
    retract(liste_subus_in_vertrag(T,LIV3)),
    asserta(liste_subus_in_vertrag(T1,LIV3)),!.
loop(U,T,chef,LSA,W,VD,ZINS) :-
    nth1(U,LSU,AU),
    akteur(chef,KC), akteur(AU,KU1),
    append('res1.pl'), write(akteur(chef,T,KC)), write('. '),
    write(akteur(AU,T,KU1)), write('.'), nl, told,
    liste_subus_in_vertrag(T,LIV1),
    ( member(AU,LIV1), true
    ;
    mache_einen_vertrag(AU,T,chef,W,VD,ZINS)
    ),
    arbeite(AU,T),
    mache_abrechnung(AU,T,chef,W,VD,ZINS),!.
mache_einen_vertrag(AU,T,chef,W,VD,ZINS) :-
    akteur(chef,KC1),
    akteur(AU,KU2),
    KC1neu is KC1 - W,
    KU2neu is KU2 + W,
    append('res1.pl'), write(akteur(chef,T,KC1neu)), write('. '),
    write(akteur(AU,T,KU2neu)), write('.'), nl, told,
    liste_subus_in_vertrag(T,LIV2),
    append(LIV2,[AU],LIV2neu),
    retract(liste_subus_in_vertrag(T,LIV2)),
    asserta(liste_subus_in_vertrag(T,LIV2neu)),
    retract(akteur(chef,KC1)), asserta(akteur(chef,KC1neu)),
    retract(akteur(AU,KU2)), asserta(akteur(AU,KU2neu)),!.
arbeite(AU,T) :-
    gewinnerwartung(GE),
    G is random(GE) + 1,
    asserta(gewinn(AU,T,G)),!.
mache_abrechnung(AU,T,chef,W,VD,ZINS) :-
    gewinn(AU,T,G),
```

```
    ZI is ZINS * W,
    ( G < ZI, ueberweise(G,AU,T,chef)
    ;
    ZI =< G, ueberweise(ZI,AU,T,chef)
    ),
    akteur(chef,KC5),
    append('res1.pl'), write(gewinn(AU,T,G)), write('. '),
    write(akteur(chef,T,KC5)), write('.'), nl, told,!.
ueberweise(Y,AU,T,chef) :-
    akteur(chef,KC4), akteur(AU,KU5),
    KC4neu is KC4 + Y,
    KU5neu is KU5 - Y,
    retract(akteur(chef,KC4)), asserta(akteur(chef,KC4neu)),
    retract(akteur(AU,KU5)), asserta(akteur(AU,KU5neu)),!.
loop1(U1,T,chef,LSU,W,VD,ZINS) :-
    nth1(U1,LSU,AU),
    gewinn(AU,T,GU),
    ZI is ZINS * W,
    ( GU < ZI,
    aux(LL), append(LL,[AU],LLneu), retract(aux(LL)), asserta(aux(LLneu)),
    akteur(chef,KC6), KC6neu is KC6 + 100,
    retract(akteur(chef,KC6)), asserta(akteur(chef,KC6neu)),
    vertrag_wird_gekuendigt(AU,T,chef,W)
    ;
    GU >= ZI, true
    ),!.
vertrag_wird_gekuendigt(AU,T,chef,W) :-
    akteur(AU,KU4),
    liste_subus_in_vertrag(T,LIV1),
    delete(LIV1,AU,LIV2), retract(liste_subus_in_vertrag(T,LIV1)),
    asserta(liste_subus_in_vertrag(T,LIV2)),
    liste_der_subus(T,LSU1),
    delete(LSU1,AU,LSU1neu),
    retract(liste_der_subus(T,LSU1)),
    asserta(liste_der_subus(T,LSU1neu)),
    retract(akteur(AU,KU4)),
    liste_der_msubus(LMS12),
    delete(LMS12,AU,LMSneu),
    retract(liste_der_msubus(LMS)),
    asserta(liste_der_msubus(LMSneu)),!.
finde_neue_subus(T,LL) :-
```

```
liste_der_msubus(LMS1)
liste_der_subus(T,LSU),
( LL = [], true
;
LL = [AGENT],
delete(LMS1,AGENT,LMS1neu),
retract(liste_der_msubus(LMS1)),
asserta(liste_der_msubus(LMS1neu)),
delete(LSU,AGENT,LSAneu),
nth1(2,LMS1neu,A1),
append(LSUneu,[A1],LSU2),
retract(liste_der_subus(T,LSU)),
asserta(liste_der_subus(T,LSU2))
;
LL = [A1,A2],
delete(LMS1,A1,LMS1a), delete(LMS1a,A2,LMS1neu),
retract(liste_der_msubus(LMS1)),
asserta(liste_der_msubus(LMS1neu)),
nth1(1,LMS1neu,B1), nth1(2,LMS1neu,B2),
retract(liste_der_subus(T,LSA)),
asserta(liste_der_subus(T,[B1,B2]))
),!.
```

Thema 5: Wissenschaft

Das Modell

Wissenschaft lässt sich von Pseudowissenschaft unterscheiden. Dies geschieht durch ein System von Gutachtern. Ein solches System wird hier beschrieben.

In einer wissenschaftlichen Disziplin gibt es in einer Periode verschiedene Forscher, die verschiedene Forschungsprojekte (kurz: Pojekte) durchführen. Ein Projekt wird durch Gutachter bewertet. Ein Gutachter gibt einem Projekt eine 'Note', die zum Beispiel zwischen -5, -4, ..., 4, 5 liegt. In jedem Projekt arbeitet ein Forscher; ein Forscher kann auch mehrere Projekte bearbeiten. Jeder Forscher hat ein 'Reputationskonto' im dem ein gewisses Reputations- oder Statuskapital liegt.

Eine wichtige Hypothese in solchen Systemen wird in normaler Sprache so ausgedrückt: 'Wer hat, denn wird gegeben'. In einem Gutachtensystem würde dies etwa Folgendes heißen. Wenn der (Reputations-) Kontostand des Forschers F1 größer ist als der Kontostand von F2, und wenn F1 ein Projekt P1 und F2 ein Projekt P2 bearbeitet, dann ist die Bewertung ('die Note') für das Projekt P1 besser als die Note für das Projekt P2.

Wir haben ein kleines Computerprogramm in PROLOG formuliert, welches eine erste Version der Hypothese ausdrückt. Das Programm kann als Einstieg für genauere

Formulierungen verwendet werden.

Das Modell besteht aus 23 Forschern, 20 Tics, 5 maximal möglichen Noten und aus einem Startkapital, das den Status, die Reputation, der Wissenschafter ausdrückt. Das System enthält

- 50 Statuseinheiten.
- ZF ist die AnZahl von Forschern
- zahl_der_forscher(23).
- TT ist die Anzahl von Tics
- zahl_der_tics(20).
- SK ist das maximal mögliche Startkapital für jeden Forscher
- max_start_kapital(50).
- MAX drückt die maximal mögliche Note aus
- maximale_note(5).

In Zeile 3 wird eine 'maximale Note' MAX eingeführt. In Zeilen 4 - 5 wird die Zahl ZF der Forscher in dem betrachteten System und die Zahl TT der Tics eingeführt. In 6 wird eine maximales Startkapital SK festgelegt.

Jeder Forscher hat ein Konto, in dem eine Art von Reputations- oder Startkapital liegt. Für jeden Forscher wird das Startkapital generiert und in sein Konto eingetragen.

In Zeilen 9 und 10 werden Resultatdateien gelöscht, falls sie noch von früheren Programmabläufen vorhanden sind. In Zeilen 11 - 13 werden die Konstanten MAX, ZF und TT von oben geholt. In 14 und 30 werden Ausgangskonten für die Forscher eingerichtet.

In 31 wird eine Schleife über die Zahl der Forscher gelegt.

F

ist eine Variable für natürliche Zahlen und wird sowohl als Index für Forscher als auch für die Namen der Forscher verwendet.

In 33 wird ein Schleifenschritt ausgeführt: das Konto für F wird eingerichtet. Dazu wird in 34 das maximale Startkapital SK aus der Datenbasis geholt. In 35 wird eine Zufallszahl

K

aus dem Bereich 0,1,2,3,...,SK-1 gezogen. Normalwerweise transformieren wir diese Zahlen in: 1,2,3,....,SK. In diesem Fall tun wir dies nicht, denn ein Konto kann am Anfang auch leer sein. D.h. der Forscher hat eventuell noch keinen Status (kein Kapitel) oder dieses Kapital ist nicht bekannt.

In 40 wird das

konto(0,F,K)

eingerichtet. D.h. der Term konto(0,F,K) wird in die Datenbasis eingefügt. konto(0,F,K) besagt, dass zum Zeitpunkt 0 (= Tic) der Forscher F den Betrag K auf seinem Konto hat. Dieser Betrag drückt den Status von F als Zahl aus. Damit ist die Klause in 30 beendet. PROLOG geht zurück zu 14 und findet als nächste Zeile die Zeile 15.

Dort werden reduzierte Projekte erzeugt. 'Reduziert' soll heißen, dass noch nicht alle Aspekte erzeugt werden, die für ein Projekt gebraucht werden. In Zeile 40 sind neben der Zahl TT von Tics zwei Zahlen

3 und 6

eingeführt. 3 besagt, dass es immer mindestens 3 Projekte zu Tic T geben muss. Wenn dies nicht der Fall ist, kann das Programm abstürzen. 6 grenzt den Bereich 1,2,3,...,6 ab, aus dem in Zeile 58 eine Zufallszahl

R1

gezogen werden kann.

In Zeile 58 wird diese Zufallszahl R1 sofort zu

R = R1 + 3 erhöht.

Das heißt, zu jedem Tic T gibt es ('laufen zu T') R Projekte. Dabei ist R immer größer oder gleich 3.

In 70 sieht man, dass ein Element

[T,C1] in eine Liste
locall(LL)

eingefügt wird. [T,C1] 'ist' ein neues reduziertes Projekt.

In 50 wird eine Liste GLP von reduzierten Projekten in die Datenbasis eingefügt. Diese Liste wurde in drei verschachtelten Klausen generiert. Diese Klausen sind schwer verständlich, daher werden wir sie hier nicht genauer erörtern.

GLP

steht für 'globale Liste von reduzierten Projekten'. Diese Liste wird am Anfang erzeugt, um 'echte' Projekte weiter unten einfacher einzuführen. Wenn die Klause in 40 beendet ist, springt PROLOG zu 15 zurück.

In 16 wird die Liste GLP der globalen reduzierten Projekte aus der Datenbasis geholt. Diese Liste wurde vorher gerade erzeugt. In 17 wird die Länge der Liste ZP von reduzierten Projekten berechnet.

ZP ist die Zahl der global reduzierten Projekte.

Diese Zahl ist nun vorhanden und wird in 18 in die Datenbasis eingetragen:

zahl_der_projekte(ZP).

Dies wird auch in die Resultatdatei

res1

geschickt.

In 20 werden nun die 'echten' Projekte gebildet. In 76 sind Zahlen ZP, ZF und die Liste GLP bekannt. In 77 wird ein Zähler

count(_)

eingerichtet und auf 1 gestellt. In 78 wird eine leere Liste von Projekten eingefügt und in die Datenbasis geschickt. In 79 wird eine Schleife über die Zahl der Projekte

gelegt.

In einem Schleifenschritt in 88 wird in 89 der Zähler geholt. C ist diejenige Zahl, die gerade im Zähler zu finden ist. In 90 wird weiterhin die Liste

LPR

der Projekte aus der Datenabsis geholt. In 91 wird die X-te Komponente aus der Liste GLP bestimmt. Diese Komponente hat die Form [A,B]. In 92 wird die Zahl C der Liste [A,B] hinzugefügt. Aus

[A,B] wird [A,B,C].

Diese Liste [A,B,C] wird durch den Befehl 'append' in die Liste LPR eingefügt. Die neue Liste LPRneu wird in 93 in die Datenbasis geschickt. In 94 und 96 werden zwei Fälle bearbeitet.

In 94 ist die Zahl C im counter mit der Zahl ZF der Forscher identisch. In diesem Fall wird die Zahl C auf C1 + 1 erhöht. Das nächste Projekt des Namens

X

wird bearbeitet, so dass der Zähler wieder auf 1 gestellt wird.

Im zweiten Fall beschäftigt sich PROLOG noch mit dem Projekt des Names X. In diesem Fall ist C ¡ ZF und die Zahl C wird um 1 erhöht. Der Zähler wird in 98 upgedatet. Damit ist Klause 76 beendet. PROLOG springt zurück zu 20.

In der nächsten Zeile 23 findet PROLOG die Liste

LPRO der Projekte.

In 24 und 102 werden die Projekte aus dieser Liste miteinander verglichen. In 103 wird eine Schleife über die Zahl der Tics gelegt. Dazu wird in 108 eine Liste von Entitäten aus der Liste LPRO untersucht und gesammelt. Genauer wird durch member(Y,LRPO) aus der Liste LPRO ein Element Y geholt. Y ist ein Tripel [A,B,C], das in 92 eingeführt wurde. Y repräsentiert inhaltlich ein Projekt.

A ist ein Tic T,
B ist ein 'Name' des Projekts und
C ein 'Name' eines Forschers.

Mit ',' wird dann die erste Komponente, hier A, bestimmt. Im Programmablauf ist A ein Tic, A = T. Weiter wird auch die zweite Komponente von Y bestimmt: nämlich B (siehe 92). Im Ablauf wird B identisch mit X sein, also mit einem Element, welches in die Liste LISTE2 eingetragen wird.

In dieser Form, wird PROLOG alle Elemente aus der Liste LPRO aufsammeln und in die

LISTE2

schreiben. Wenn wir vor der Zeile 108 den

'trace',

als neue Zeile einfügen, und das Programm ablaufen lassen, sehen wir, dass PROLOG durch

findall

eventuell ziemlich viele Schritte braucht, um den Term findall(...) bis zum Ende durch durchzurechnen.

Im Ende ist die LISTE2 in 108 vorhanden. In 109 und 110 werden nun die ersten vier Komponenten aus der Liste LISTE2 herausgeholt. Sie heißen

P1, P2, P3 und P4.

P1 ist ein Projekt P1 = [T,P,F].
In einem Programmablauf wäre T ein bestimmter Tic, z.B. T = 12, P wäre ein 'Name' für ein Projekt, hier etwa P = 3 und F wäre der Forscher F = 4.

In 111 wird in die Datenbasis eingetragen, dass zu Tic T die Projekte P1 und P2 ähnlich sind

similar oder sim,

und dass Projekte P3 und P4 ähnlich sind. Dies wird auch zur Resultatdatei

res2

geschickt. PROLOG geht dann zurück zu 102 und dann zu 24.

Nun geht PROLOG zur nächsten Zeile 25. Dort wird eine Schleife über den Bereich TT der Tics gelegt. In 117 wird zu dem Tic T ein Projekt bearbeitet ('ausgeführt'). In 118 wird eine weitere Schleife über die Zahl ZF der Forscher gelegt.

In 120 führt Forscher F zu Tic T eine Forschungshandlung aus. In 121 wird die Liste LPRO der Projekte geholt. In 122 wird der jeweils 'letzte' Tic berechnet:

T1 = T - 1.

In 123 wird das Statuskonto von F zu Tic T1 geholt. Dies ist in der Datenbasis noch zu finden.

K

ist quasi die 'Größe' des Status von F zu T (K ist eine Zahl). In 124 wird geschaut, ob das Projekt [T,P,F] in der Liste LPRO zu finden ist. Wenn ja, kommt PROLOG in 126 zur Bewertung des Projekts.

In 136 wird die maximale Note MAX geholt. In 137 wird eine Note gebildet, d.h. ein hier unbekannter Gutachter wird eine Note vergeben. In 138 geht PROLOG zu 151. Dort wird eine Zufallszahl

R

aus dem Bereich 1,2,...,MAX gezogen. In 153 wird eine weitere Zufallszahl

R1

aus dem Bereich 1,2 gezogen. In 154 gibt es zwei Fälle.

Im ersten Fall ist R1 = 1. In diesem Fall wird eine neue Variable R2 eingeführt, die ebenfalls durch 1 instantiiert wird.

Im zweiten Fall is R1 = 2. In diesem Fall wird R2 auf -1 (Minus) gesetzt. In 155 wird

NOTE = R2 x R gebildet.

Wenn R z.B. 2 ist (also R = 2) und R2 = -1 ist, ergibt sie die Note -2, also eine recht schlechte Note. PROLOG geht nun zu 138.

In 139 wird der vorherige Zeitpunkt T1 'berechnet'. In 140 wird geschaut, ob es ein Projekt P1 gibt, das zum Tic T mit dem Projekt P ähnlich ist (sim). Da

sim

nicht symmetrisch sein muss, haben wird beide Möglichkeiten in Betracht gezogen. In 141 wird aus der Liste LPRO von Projekten ein Element geholt: das vervollständigte Projekt [T,P1,F1]. In 142 wird das

konto

aus der Datenbasis geholt, welches noch von dem letzten Tic T1 vorhanden ist. Dieses Konto gehört dem Forscher F1 und der (Reputations-) Kontobetrag K1 vom letzten Tic wird geholt. Dieser Betrag K1 kann kleiner sein als der Betrag K in Zeile 137 für Projekt P.

In diesem Fall wird das Prädikat

bilde_eine_note(...)

untersucht - was in 138 schon ein Mal geschah. PROLOG geht wieder zu den Zeilen 151 - 155. Dort wird nun eine NOTE1 für das Projekt P1 vergeben. Eine Note für Projekt P wurde schon früher vergeben und jetzt wird eine NOTE1 für das Projekt P1 vergeben.

PROLOG geht dann zurück zu 144 und dann zu 145. Wenn die NOTE1 für P1 kleiner ist als die NOTE für P, erhöht PROLOG die NOTE um Eins (1). Das Projekt P wurde etwas 'besser gemacht'.

Real macht dies ein Gutacher, was hier aber nicht beschrieben wird. Wenn in 143 der Kontobetrag K1 nicht kleiner ist als der aus dem Konto von F ist, oder wenn in 145 die NOTE1 nicht kleiner als die Note NOTE ist, springt PROLOG in einer 'oder'-Konstruktion zu 147. In all diesen Fällen wird die 'neue'

NOTEneu

durch asserta in die Datenbasis eingetragen. Damit ist die Bewertung in 136 und 126 abgeschlossen.

In 127 wird die neue NOTEneu aus der Datenbasis geholt. In 128 wird der Kontostand K durch NOTEneu vergrößert: Kneu = K + NOTEneu.

In 129 der neue Kontostand kont(T,F,Kneu) in die Datenbasis eingetragen. Schließlich wird der Kontostand des Forschers F zu Tic T in die Resultatdatei res2 geschickt.

In 124 bis 132 kommt eine 'oder'-Konstruktion. Im ersten Fall wurde der Betrag K upgedated und in zweiten Fall wird in Zeile 60 der alte Betrag einfach übernommen. Damit springt PROLOG zurück in 120 und dann zu 117. Wenn die Klause abgearbeitet ist, springt PROLOG zu 25 (oben) und wendet sich der Zeile 26 zu.

In dieser Schleife werden die Kontowerte für jeweils einen Forscher aufgesammelt. In 159 wird für einen Forscher F in der letzten Zeitperiode TT eine Schleife durchlaufen. In 162 und 163 wird für Tic T der Betrag X vom Konto des Forsches F zum

Zeitpunkt T geholt - der immer noch in der Datenbasis zu finden ist. Dieser Kontobetrag X wird in 164 zur Resultatdatei

res1

geschickt.

Das Programm

```
maximale_note(5).
zahl_der_forscher(23).
zahl_der_tics(20).
max_start_kapital(50).

start :-
    ( exists_file('res1.pl'), delete_file('res1.pl'); true),
    ( exists_file('res2.pl'), delete_file('res2.pl'); true),
    maximale_note(MAX),
    zahl_der_forscher(ZF),
    zahl_der_tics(TT),
    bilde_ausgangskonten(ZF),
    bilde_reduzierte_projekte(TT,3,6),
    liste_der_reduzierten_projekte(GLP),
    length(GLP,ZP),
    asserta(zahl_der_projekte(ZP)),
    append('res1.pl'), write(zahl_der_projekte(ZP)), write('.'), nl, told,
    bilde_projekte(ZP,ZF,GLP),
        /* dies funktioniert nur, wenn die Zahl ZF der Forscher kleiner
    ist als 2*ZF =¡ ZP (= Zahl der Projekte) */
    liste_der_projekte(LPRO),
    vergleiche_projekte(TT,LPRO),
    ( between(1,TT,T), projektausfuehrung(T,ZF), fail; true),
    ( between(1,ZF,F), sammle_konten(F,TT), fail; true),!.

bilde_ausgangskonten(ZF) :-
    ( between(1,ZF,F), bilde_ein_konto(F), fail; true),!.

bilde_ein_konto(F) :-
    max_start_kapital(SK),
    K is random(SK),
    asserta(konto(0,F,K)),!.

bilde_reduzierte_projekte(TT,3,6) :-
    asserta(counter(0)),
    asserta(global([])),
    bilde_global_list(TT),
    retractall(counter(CCC)),!.
```

```
bilde_global_list(TT) :-
    asserta(global([])),
    ( between(1,TT,T), bilde_local_list(T), fail; true),
    global(GLP), retract(global(GLP)),
    asserta(liste_der_reduzierten_projekte(GLP)),
    append('res1.pl'), write(liste_der_reduzierten_projekte(GLP)),
    write('.'), nl, told,!.
        /* Ein reduziertes Projekt aus der Liste GLP hat die Form [T,P],
        T = Zeitpunkt und P = Projekt(name) */
bilde_local_list(T) :-
    asserta(locall(T,[])),
    R1 is random(6) + 1, R is R1 + 3,
        /* R aus 4,...,9. R besagt, wieviele Projekte zu Tic T generiert
        werden. Jedem Projekt wird zugleich ein neuer Name zugewiesen */
    ( between(1,R,X), addiere_lpro(X), fail; true),
    locall(T,LL1),
    global(LL2), append(LL2,LL1,LL2neu),
    retract(global(LL2)), asserta(global(LL2neu)),
    retract(locall(T,LL1)),!.
addiere_lpro(X) :-
    locall(T,LL),
    counter(C), C1 is C + 1,
    append(LL,[[T,C1]],LLneu),
    retract(locall(T,LL)), asserta(locall(T,LLneu)),
    retract(counter(C)), asserta(counter(C1)),!.
bilde_projekte(ZP,ZF,GLP) :-
    asserta(count(1)),
    asserta(liste_der_projekte([])),
    ( between(1,ZP,X), loop1(X,ZF,GLP), fail; true),
    liste_der_projekte(LPRO),
    asserta(liste_der_projekte(LPRO)),
    append('res1.pl'), write(liste_der_projekte(LPRO)), write('.'), nl,
    told,!.
        /* Ein projekt hat die Form [T,P,F], T = Zeitpunkt, P = Projektname,
        F = Forscher(name) */
    loop1(X,ZF,GLP) :-
    count(C),
    liste_der_projekte(LPR),
    nth1(X,GLP,[A,B]),
    append(LPR,[[A,B,C]],LPRneu),
    asserta(liste_der_projekte(LPRneu)),
    ( C = ZF, C1 is 1
```

```
    ;
    C < ZF, C1 is C + 1
    ),
    retract(count(C)), asserta(count(C1)),!.
vergleiche_projekte(TT,LPRO) :-
    ( between(1,TT,T), vergleiche(T,LPRO), fail; true),!.
        /* Dies Erzeugung ist ziemlich natürlich einfältig */
vergleiche(T,LPRO) :-
    findall(X,(member(Y,LPRO),nth1(1,Y,T),nth1(2,Y,X)),LISTE2),
    nth1(1,LISTE2,P1), nth1(2,LISTE2,P2), nth1(3,LISTE2,P3),
    nth1(4,LISTE2,P4),
    asserta(sim(T,P1,P2)), asserta(sim(T,P3,P4)),
    append('res2.pl'), write(sim(T,P1,P2)), write('. '),
    write(sim(T,P3,P4)), write('.'), nl, told,!.
projektausfuehrung(T,ZF) :-
    ( between(1,ZF,F), ausfuehre(F,T), fail; true),!.
ausfuehre(F,T) :-
    liste_der_projekte(LPRO),
    T1 is T - 1,
    konto(T1,F,K),
    ( member([T,P,F],LPRO),
        /* es gibt höchstens ein Projekt zu T für F */
    bewerte(T,P,F,K,NOTE,LPRO),
    neue_note(NOTEneu),
    Kneu is K + NOTEneu,
    asserta(konto(T,F,Kneu)),
    append('res2.pl'), write(konto(T,F,Kneu)), write('.'), nl, told
    ;
    asserta(konto(T,F,K)),
    append('res2.pl'), write(konto(T,F,K)), write('.'), nl, told
    ),!.
bewerte(T,P,F,K,NOTE,LPRO) :-
    maximale_note(MAX),
    bilde_eine_note(P,MAX,NOTE),
    T1 is T - 1,
    ( ( sim(T,P,P1); sim(T,P1,P) ),
    member([T,P1,F1],LPRO),
    konto(T1,F1,K1),
    K1 < K,
    bilde_eine_note(MAX,P1,NOTE1),
    NOTE1 ¡ NOTE, NOTEneu is NOTE + 1
```

```
    ;
    NOTEneu is NOTE
    ),
    asserta(neue_note(NOTEneu)),!.
bilde_eine_note(P,MAX,NOTE) :-
    R is random(MAX) + 1,
    R1 is random(2) + 1,
    ( R1 is 1, R2 is 1 ; R1 is 2, R2 is - 1),
    NOTE is R2 * R,!.
sammle_konten(F,TT) :-
    ( between(1,TT,T), sammle(T,F), fail; true),!.
sammle(T,F) :-
    konto(T,F,X),
    append('res1.pl'), write(konto(T,F,X)), nl, told,!.
```