Didaktik der Informatik MOOC

Fachdidaktik der Informatik 1

Prof. Dr.-Ing. Jens Gallenbacher

Programmierparadigmen

Deklarative Programmierung mit Prolog

Quelle: http://www.di.tu-darmstadt.de/

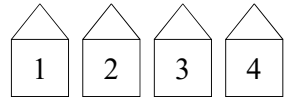
Motivation mehrerer Paradigmen

Prof. Dr.-Ing. Jens Gallenbacher

- Im Kontext des logisch-genetischen Ansatzes mit objektorientierter Betrachtung haben wir bereits mehrere „Detailsichten“ auf Problemstellungen thematisiert (z. B. Analogie als Werkzeug und als Unterrichtsgegenstand)
- Im Kontext des historisch-genetischen Ansatzes festgestellt: Wichtiges Werkzeug der Informatik ist die Modellierung der Problemlösung mittels **geeigneter** Programmierparadigmen
- Vorteil kann nur erkannt werden, wenn mindestens zwei deutlich unterscheidbare Paradigmen thematisiert werden.
- Demonstration Kombi: Logisch-genetischer, historisch-genetischer Ansatz

Prof. Dr.-Ing. Jens Gallenbacher

Aufgabe



In vier Häusern wohnen Arndt, Brita, Claire und Dan. Jeder hat ein Tier (Esel, Fisch, Gans und Hund) und eine Lieblingsspeise (Ingwer, Kohl, Lauch, Möhren).

Folgendes ist bekannt:

- Arndt ist der Nachbar von Claire und mag weder Lauch noch Esel
- Der Fischliebhaber beschwert sich manchmal über das Eselgeschrei nebenan
- Lauch- und Kohlesser unterhalten sich oft über ihren gemeinsamen Zaun. Sie haben weder einen Fisch, noch heißt einer von Ihnen Dan.
- Dan, der am Rand wohnt, hat keinen Esel
- Der Fischliebhaber wohnt ganz links und mag keinen Ingwer
- Dan und Brita müssen telefonieren, wenn sie sich zum Möhren-Ingwer-Salat verabreden
- Zwischen Gans und Esel ist genau noch ein weiteres Haus
- Dan mag Fisch zwar essen, aber nicht als Haustier

Aufgabe

Wie kann man die Aufgabe durch imperative Programmierung lösen?

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Imperative Lösung

Unzählige Schleifen (Abb. der Möglichkeiten)

```
for (arndt = 1; arndt <= 4; arndt++)
  for (brita = 1; brita <= 4; brita++) ...
```

Bedingungen der Form

```
/* Jedes Haus wird nur von einem bewohnt */
if ((arndt != brita) && (arndt != claire) && ...)

/* Arndt ist der Nachbar von Claire und mag weder Lauch
noch Esel */
if ((arndt - claire == 1) || (claire - arndt == 1))
  if ((arndt != lauch) && (arndt != esel)) ...
```

Prof. Dr.-Ing. Jens Gallenbacher

Fazit

Imperative Programmierung ist für Aufgaben im Bereich Wissen und Logik nicht besonders übersichtlich und nicht besonders effektiv (zumindest, was den Programmieraufwand angeht)

Neues Programmierparadigma

"Wissensbasierte Programmierung"

Ansatz: Menschliches Wissen und Wissensstrukturen sollen im Rechner abgebildet werden, um Lösungen "menschlicher Probleme" und universeller Fragen zu finden.

Prof. Dr.-Ing. Jens Gallenbacher

1. Ansatz: Neuronale Netze

Direkter "Nachbau" menschlicher Gehirnstruktur.

- Neuronen werden miteinander vernetzt.
- Programm besteht aus Verbindungen zwischen den Neuronen
- Training manipuliert diese Verbindungen

- Hoffnung: Nach genügend Training mit bekannten Antworten, werden auch unbekannte Antworten nach gleichem Schema ermittelt

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Neuronale Netze

Problem: Man kann kaum nachvollziehen, auf welcher Basis ein neuronales Netz lernt

Beispiel:

Netz wurde trainiert, Waldbilder mit versteckten Panzern und solche ohne Panzer unterscheiden zu können.

Neuronales Netz konnte einen ersten Satz von Bildern korrekt erkennen

Beim zweiten Satz versagte es komplett

Neuronale Netze

Analyse:

Alle Bilder des ersten Satzes mit Panzern wurden bei schönem Wetter aufgenommen, alle ohne Panzer bei schlechtem Wetter

→ Neuronales Netz konnte mit großer Sicherheit gutes von schlechtem Wetter unterscheiden, aber keinerlei Panzer „sehen“

→ Neuronales Netz liefert nie 100% zuverlässige Ergebnisse

Prof. Dr.-Ing. Jens Gallenbacher

2. Ansatz: Wissensbasis (Regelwerk) aufbauen

Mathematisch-logisches Regelwerk wird dem Computer eingespeist

Problemlösungen werden durch mathematische Beweismethoden (Inferenzmethoden) ermittelt.

Prof. Dr.-Ing. Jens Gallenbacher

Beispiel: Projekt Cyc

1984: Riesige Datenbanken werden mit enzyklopädischen Fakten (Name von enCYClopedia) gefüttert

- Viele Fragen können vom System beantwortet werden, auch wenn die Antwort nicht vorher einprogrammiert wurde
- Das Ziel ist, aus dem Pool von Fakten abstraktere, generalisiertere Aussagen zu generieren (wie "Zwei Länder, die ungefähr die gleichen Hauptreligionen besitzen, sind auch Mitglied der gleichen internationalen Organisationen").

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

CyC

Problem: Manchmal funktionieren die Schlußfolgerungen nicht richtig.

"**Die meisten Menschen sind prominent**"

→ Natürlich sind die meisten Menschen aus enzyklopädischem Wissen Personen aus dem öffentlichen Leben

"**Manchmal sind Männer keine Menschen**"

→ Kommt durch – sich widersprechende – Aussagen:

– Männer haben beim Rasieren einen Rasierapparat

– Menschen haben keine elektrischen Bestandteile

CyC

→ Wissen der Geräte mutet manchmal an, als wenn man keine menschliche Intelligenz vor sich hat, sondern einen Außerirdischen, der vom Raumschiff aus Millionen von Informationen über die Welt gesammelt hat und daraus sich ein bizarres Weltbild geschaffen hat.

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

CyC

Projekt ist 1995 weitgehend in das OpenSource-Projekt OpenCyC übergegangen:

http://www.opencyc.org/

http://www.cyc.com/

(hier gibt es auch ein kleines Trivia-Spiel, das sehr viel davon preisgibt, wie (wenig) weit das Projekt fortgeschritten ist: „Slides are typically located in two-stories“, „Most computer hardware ist longer than most packages“, …)

Ubiquitäre Wissensverarbeitung

Moderne Methoden kombinieren für ähnliche Ziele verschiedene Möglichkeiten der lexikalischen Analyse, der empirischen Auswertung des Verhaltens „menschlicher“ Forscher, des Data-Minings und der logischen Schlussfolgerung.

Für die Lösung eines „einfachen“ Logik-Rätsels (und insbesondere für den Einsatz in der Schule) aber viel zu kompliziert.

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Logiksprache für die Schule

Prolog ("PROgramming in LOGic")

- Deklarative Programmierung
- Anfang der 1970er-Jahre entwickelt
- Pflichtstoff im hessischen Lehrplan, im Zentralabitur Wahlthema

Vorgehen: Erst kleiner "Programmierkurs", danach Analyse des Programmierparadigmas und Diskussion über Sinn in der Schule „live“

Prof. Dr.-Ing. Jens Gallenbacher

"Datenbasis": Fakten

```
mann(jens).
mann(peter).
mann(klaus).
mann(daniel).
mann(thomas).
mann(paul).
mann(tobias).
frau(claire).
frau(anna).
frau(brita).
frau(olga).
frau(gudrun).
```

```
kind(paul, anna).
kind(anna, claire).
kind(peter, klaus).
kind(olga, brita).
kind(olga, tobias).
kind(jens, daniel).
kind(claire, thomas).
paar(gudrun, paul).
paar(anna, peter).
paar(jens, claire).
paar(klaus, olga).
```

Prof. Dr.-Ing. Jens Gallenbacher

Programmierumfeld für Prolog

SWI-Prolog

http://www.swi-prolog.org/

einfache GUI

http://www.bildung.hessen.de/
und nach „SWI PROLOG EDITOR“ suchen

Prof. Dr.-Ing. Jens Gallenbacher

Fragen an Prolog

?- frau(olga).
→Yes
?- frau(klaus).
→No
?- frau(sabine).
→No

mann(jens).	frau(brita).	kind(claire, thomas).
mann(peter).	frau(olga).	paar(gudrun, paul).
mann(klaus).	frau(gudrun).	paar(anna, peter).
mann(daniel).	kind(paul, anna).	paar(jens, claire).
mann(thomas).	kind(anna, claire).	paar(klaus, olga).
mann(paul).	kind(peter, klaus).	
mann(tobias).	kind(olga, brita).	
frau(claire).	kind(olga, tobias).	
frau(anna).	kind(jens, daniel).	

→ Ein "Ja" bestätigt die Korrektheit der Aussage, ein "Nein" jedoch nicht die Unkorrektheit, sondern lediglich, dass die Aussage nicht aus der Datenbasis entscheidbar ist.

Prof. Dr.-Ing. Jens Gallenbacher

weitere Fragen

?- kind(anna, claire).

→Yes

?- kind(claire, anna).

→No

?- kind(Elternteil, claire).

→Elternteil = anna;

?- kind(anna, X).

→X = claire;

→ Variablen beginnen mit Großbuchstaben!

mann(jens).	frau(brita).	kind(claire, thomas).
mann(peter).	frau(olga).	paar(gudrun, paul).
mann(klaus).	frau(gudrun).	paar(anna, peter).
mann(daniel).	kind(paul, anna).	paar(jens, claire).
mann(thomas).	kind(anna, claire).	paar(klaus, olga).
mann(paul).	kind(peter, klaus).	
mann(tobias).	kind(olga, brita).	
frau(claire).	kind(olga, tobias).	
frau(anna).	kind(jens, daniel).	

Prof. Dr.-Ing. Jens Gallenbacher

Regeln

```
mensch(X) :- frau(X); mann(X).
sohn(X, Y) :- kind(X, Y), mann(Y).
tochter(X, Y) :- kind(X, Y), frau(Y).
```

; Semikolon für "oder"

, Komma für "und"

... "oder" aber auch durch mehrere Regeln ...

```
mensch(X) :- frau(X).
mensch(X) :- mann(X).
```

Regeln sorgen auch für Antworten

Prof. Dr.-Ing. Jens Gallenbacher

?- mensch(klaus).

→Yes

?- sohn(X, Y).

→X = peter
Y = klaus ;

→X = jens
Y = daniel ;

→X = claire
Y = thomas ;

```
mann(jens).
mann(peter).
mann(klaus).
mann(daniel).
mann(thomas).
mann(paul).
mann(tobias).
frau(claire).
frau(anna).
frau(brita).
frau(olga).
frau(gudrun).
kind(paul, anna).
kind(anna, claire).
kind(peter, klaus).
kind(olga, brita).
kind(olga, tobias).
kind(jens, daniel).
kind(claire, thomas).
paar(gudrun, paul).
paar(anna, peter).
paar(jens, claire).
paar(klaus, olga).
mensch(X) :-
  frau(X); mann(X).
sohn(X, Y) :-
  kind(X, Y), mann(Y).
tochter(X, Y) :-
  kind(X, Y), frau(Y).
```

Beispiel: Bruder

```
bruder(X, Y) :- kind(Z, X), kind(Z, Y), mann(Y).
```

Warum funktioniert die Regel nicht korrekt?

Prof. Dr.-Ing. Jens Gallenbacher

Beispiel: Bruder

bruder(X, Y) :- kind(Z, X), kind(Z, Y), mann(Y).

→ Es wird nicht überprüft, ob X und Y identisch sind. Auf diese Weise kann jemand sein eigener Bruder sein.

Besser:

bruder(X, Y) :- kind(Z, X), kind(Z, Y), mann(Y), X\==Y.

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Jetzt: Logical lösbar!!!

AUFGABE:

Lösen Sie das Rätsel von der Einstiegs-Folie mit einem Prolog-Programm!

Tipp: Falls Sie mit einzelnen Regeln nicht weiterkommen, versuchen Sie eine einzige, die viele Variablen enthält!

→ Bereits wenige formale Vorgaben reichen aus, um „umfangreiches“ Programm zu schreiben.

Regeln ohne Grenzen?

kind(X, Y):-(paar(X, Z); paar(Z, X)), kind(Z, Y).

Diese Regel soll dafür sorgen, dass der Computer auch Kinder erkennt, die in den Fakten dem Lebenspartner zugewiesen wurden.

Sie funktioniert aber nicht korrekt!

Warum?

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

kind(anna, claire).
kind(peter, olga).
kind(peter, jens).
kind(X, Y):-(paar(X, Z); paar(Z, X)), kind(Z, Y).
paar(anna, peter).

---

kind(peter,Y).
→olga (aus " kind(peter, olga).")
→jens (aus " kind(peter, jens).")
→claire (aus "paar(anna, peter), kind(anna, claire).")
→olga (aus "paar(anna, peter), kind(anna, olga).")

paar(anna, peter), kind(peter, olga)

usw.

Ohne Rekursion: Neue Regel definieren

Prof. Dr.-Ing. Jens Gallenbacher

Offenbar bringt Rekursion „gefährliche“ Rückkoppelungen, daher vermeiden:

"akind" für "alle Kinder zeigen"...

```
akind(X, Y) :- kind(X, Y).
akind(X, Y) :- paar(X, Z), kind(Z, Y).
akind(X, Y) :- paar(Z, X), kind(Z, Y).
sohn(X, Y) :- akind(X, Y), mann(Y).
tochter(X, Y) :- akind(X, Y), frau(Y).
```

Rekursion in Prolog richtig verstehen

Prof. Dr.-Ing. Jens Gallenbacher

Ein Nachfahre ist einerseits ein Kind...
...aber auch ein Kind eines Nachfahren!

```
nachfahre(X, Y) :- kind(X, Y).
nachfahre(X, Y) :- kind(X, Z), nachfahre(Z, Y).
```

Beispiel: nachfahre(paul,Y).
anna, claire, thomas.

Rekursion in Prolog

Abweichung vom rein deklarativen...

```
nachfahre(X, Y) :- kind(X, Y).
nachfahre(X, Y) :- nachfahre(Z, Y), kind(X, Z).
```

Je nach Reihenfolge der Bedingungen führt Beispiel zu ein paar Ergebnissen und dann...

Prof. Dr.-Ing. Jens Gallenbacher

Rekursion in Prolog

Verständnis der Interna sind hier doch relevant:

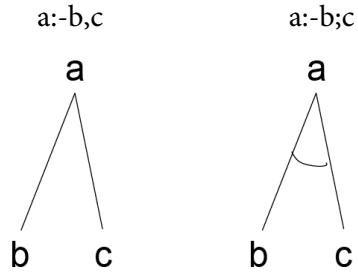
Prolog ist keine Black-Box-Sprache.

Analyse: Aufbau des Beweisbaumes.

Prof. Dr.-Ing. Jens Gallenbacher

Beweisbaum

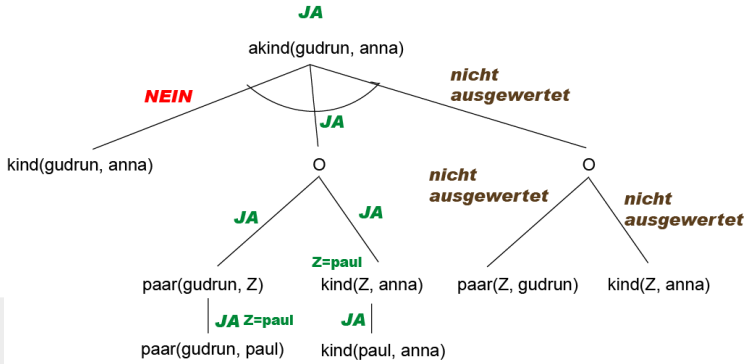
Baum mit symbolisierter "und" und "oder"-Verknüpfung



Prof. Dr.-Ing. Jens Gallenbacher

Beweisbaum für akind(gudrun,anna)

```
mann(jens).
mann(peter).
mann(klaus).
mann(daniel).
mann(thomas).
mann(paul).
mann(tobias).

frau(claire).
frau(anna).
frau(brita).
frau(olga).
frau(gudrun).
kind(paul, anna).
kind(anna, claire).

kind(peter, klaus).
kind(olga, brita).
kind(olga, tobias).
kind(jens, daniel).
kind(claire, thomas).
paar(gudrun, paul).
paar(anna, peter).

paar(jens, claire).
paar(klaus, olga).
akind(X, Y) :-
  kind(X, Y).
akind(X, Y) :-
  paar(X, Z), kind(Z, Y).
akind(X, Y) :-
  paar(Z, X), kind(Z, Y).

sohn(X, Y) :-
  akind(X, Y), mann(Y).
tochter(X, Y) :-
  akind(X, Y), frau(Y).
```

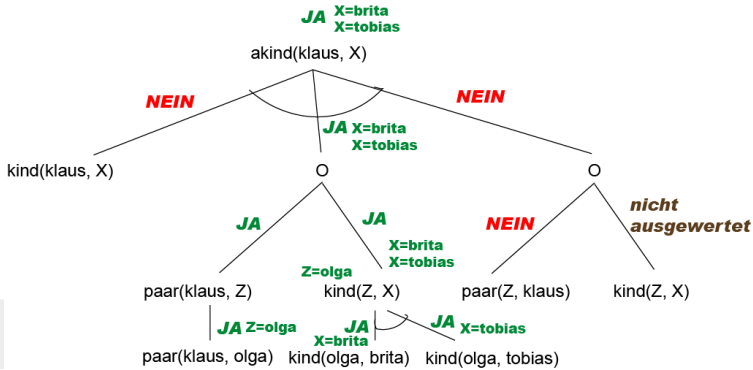


Prof. Dr.-Ing. Jens Gallenbacher

Beweisbaum für akind(klaus, X)

Prof. Dr.-Ing. Jens Gallenbacher

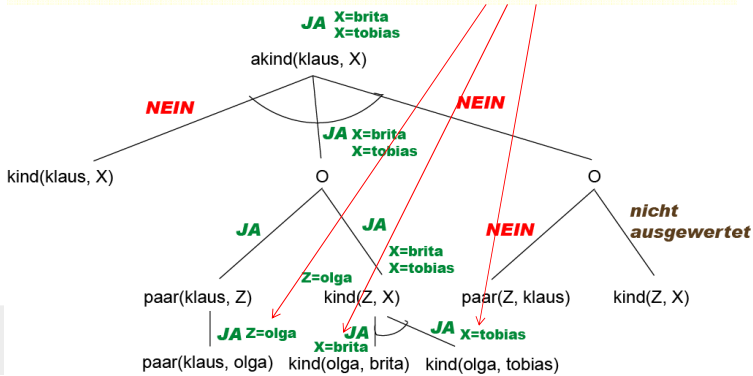

```
mann(jens).
mann(peter).
mann(klaus).
mann(daniel).
mann(thomas).
mann(paul).
mann(tobias).

frau(claire).
frau(anna).
frau(brita).
frau(olga).
frau(gudrun).
kind(paul, anna).
kind(anna, claire).

kind(peter, klaus).
kind(olga, brita).
kind(olga, tobias).
kind(jens, daniel).
kind(claire, thomas).
paar(gudrun, paul).
paar(anna, peter).

paar(jens, claire).
paar(klaus, olga).
akind(X, Y) :-
    kind(X, Y).
akind(X, Y) :-
    paar(X, Z), kind(Z, Y).
akind(X, Y) :-
    paar(Z, X), kind(Z, Y).

sohn(X, Y) :-
    akind(X, Y), mann(Y).
tochter(X, Y) :-
    akind(X, Y), frau(Y).
```



Unifikation

mann(jens).	frau(claire).	kind(peter, klaus).	paar(jens, claire).	sohn(X, Y) :- akind(X, Y), mann(Y).
mann(peter).	frau(anna).	kind(olga, brita).	paar(klaus, olga).	
mann(klaus).	frau(brita).	kind(olga, tobias).		
mann(daniel).	frau(olga).	kind(jens, daniel).		
mann(thomas).	frau(gudrun).	kind(claire, thomas).		
mann(paul).	kind(paul, anna).	paar(gudrun, paul).		
mann(tobias).	kind(anna, claire).	paar(anna, peter).		

Unifikation:
Belegung der Variablen, so dass die Ausdrücke wahr werden



Prof. Dr.-Ing. Jens Gallenbacher

Selbst: Beweisbaum für nachfahre(paul, Y)

nachfahre(X, Y) :- kind(X, Y).

nachfahre(X, Y) :- kind(X, Z), nachfahre(Z, Y).

```
mann(jens).
mann(peter).
mann(klaus).
mann(daniel).
mann(thomas).
mann(paul).
mann(tobias).

frau(claire).
frau(anna).
frau(brita).
frau(olga).
frau(gudrun).
kind(paul, anna).
kind(anna, claire).

kind(peter, klaus).
kind(olga, brita).
kind(olga, tobias).
kind(jens, daniel).
kind(claire, thomas).
paar(gudrun, paul).
paar(anna, peter).

paar(jens, claire).
paar(klaus, olga).
akind(X, Y) :-
 kind(X, Y).
akind(X, Y) :-
 paar(X, Z), kind(Z, Y).
akind(X, Y) :-
 paar(Z, X), kind(Z, Y).

sohn(X, Y) :-
 akind(X, Y), mann(Y).
tochter(X, Y) :-
 akind(X, Y), frau(Y).
```

Prof. Dr.-Ing. Jens Gallenbacher

Beweisbaum für nachfahre(paul, Y)

nachfahre(X, Y) :- kind(X, Y).
nachfahre(X, Y) :- nachfahre(Z, Y), kind(X, Z).

```
mann(jens).
mann(peter).
mann(klaus).
mann(daniel).
mann(thomas).
mann(paul).
mann(tobias).

frau(claire).
frau(anna).
frau(brita).
frau(olga).
frau(gudrun).
kind(paul, anna).
kind(anna, claire).

kind(peter, klaus).
kind(olga, brita).
kind(olga, tobias).
kind(jens, daniel).
kind(claire, thomas).
paar(gudrun, paul).
paar(anna, peter).

paar(jens, claire).
paar(klaus, olga).
akind(X, Y) :-
  kind(X, Y).
akind(X, Y) :-
  paar(X, Z), kind(Z, Y).
akind(X, Y) :-
  paar(Z, X), kind(Z, Y).

sohn(X, Y) :-
  akind(X, Y), mann(Y).
tochter(X, Y) :-
  akind(X, Y), frau(Y).
```

Prof. Dr.-Ing. Jens Gallenbacher

mathematische Gesetze

Offenbar gelten in Prolog entscheidende mathematische Gesetze der Algebra nicht!

Hier: Das Kommutativgesetz.

Prof. Dr.-Ing. Jens Gallenbacher

Listen

Prolog verwendet Listen als Datenstruktur für größere Datenmengen

Beispiel:

prim([2,3,5,7,11,13,17,19,23,29,31,37]).

Gebräuchlichste Schreibweise für die Definition in eckigen Klammern, mit Kommata separiert

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Listen sind rekursiv

Listen sind intern rekursiv aufgebaut

Schreibweise: [erstes_Element | Restliste]

Beispiel:

prim(X),X=[A|B].

→

X = [2, 3, 5, 7, 11, 13, 17, 19, 23|...]

A = 2

B = [3, 5, 7, 11, 13, 17, 19, 23, 29|...] ;

Prof. Dr.-Ing. Jens Gallenbacher

Beispiel: element

Das können wir ausnutzen, um festzustellen, ob ein Element in einer Liste enthalten ist.

```
element(X,Y):-Y=[X|Z].
element(X,Y):-Y=[Z|R],element(X,R).
```

Variable Z ist (für Ausgabe) irrelevant

→ spezielle Variable _

```
element(X,Y):-Y=[X|_].
element(X,Y):-Y=[_|R],element(X,R).
```

Beispiel: element

element(X, [1,2,3,4]).

→ X=1
→ X=2
→ X=3
→ X=4

Interessant auch:
element(2, X).

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Beispiel: element

Nun können wir überprüfen, ob eine Zahl prim ist...

```
element(X,Y):-Y=[X|_].
element(X,Y):-Y=[_|R],element(X,R).
prim([2,3,5,7,11,13,17,19,23,29,31,37]).
istprim(X):-prim(Y),element(X,Y).
```

istprim(1). → No

istprim(2). → Yes

Aufgabe

Ermitteln Sie mit Hilfe von Prolog-Regeln die Länge einer Liste in der Form

laenge([1,4,6,9], X)

→ X=4

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Lösung

```
laenge([],0).
laenge(X,Y):-X=[_|Z],laenge(Z,L),Y=L+1.
```

...allerdings kommt hier bei den meisten Prolog-Varianten eine korrekte, aber nicht 100% gewollte Lösung heraus:

```
laenge([1,4,6,9], X)
```

→ X=0+1+1+1+1

Prof. Dr.-Ing. Jens Gallenbacher

Lösung

Hier kann man durch Verwendung von "is" statt des Gleichheitszeichens eine Auflösung der arithmetischen Operationen erzwingen (Achtung Rundung):

```
laenge_echt(X,Y):-laenge(X,L),Y is L.
laenge_echt([1,4,6,9], X)
```

→ X=4

oder auch

```
laenge([],0).
laenge(X,Y):-X=[_|Z],laenge(Z,L),Y is L+1.
```

Prof. Dr.-Ing. Jens Gallenbacher

Aufgabe: Schatzsuche

Sie finden folgende Schatzkarte. Modellieren Sie diese in Prolog, so dass durch die Angabe

– schatz(o).

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

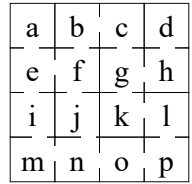
ein Schatz definiert werden kann. Prolog soll durch

– suche(a,X).

einen Weg zum Schatz vom Ausgangspunkt a zeigen.

Lösungsvorschlag: Erst Türen definieren

```
tuer(a,e).
tuer(b,c).
tuer(b,f).
tuer(c,g).
tuer(d,h).
tuer(e,f).
tuer(e,i).
tuer(f,j).
tuer(g,h).
tuer(g,k).
tuer(i,m).
tuer(j,n).
tuer(k,l).
tuer(l,p).
tuer(m,n).
tuer(n,o).
tuer(o,p).

schatz(o).
```



Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

...dann Regeln

```
tuer(a,e).
tuer(b,c).
tuer(b,f).
tuer(c,g).
tuer(d,h).
tuer(e,f).
tuer(e,i).
tuer(f,j).
tuer(g,h).
tuer(g,k).
tuer(i,m).
tuer(j,n).
tuer(k,l).
tuer(l,p).
tuer(m,n).
tuer(n,o).
tuer(o,p).
schatz(o).
```

```
weg([A,B]):-tuer(A,B).
weg([A,B]):-tuer(B,A).
weg(X):-X=[A,B|C],nicht_element(A,C),weg([B|C]),tuer(A,B).
weg(X):-X=[A,B|C],nicht_element(A,C),weg([B|C]),tuer(B,A).
suche(X,Y):-weg(Y),Y=[X|_],schatz(S),letztes(S,Y).
element(X,Y):-Y=[X|_].
element(X,Y):-Y=[_|R],element(X,R).
nicht_element(X,[]).
nicht_element(X,Y):-Y=[A|B],X\==A,nicht_element(X,B).
letztes(X,[X|[]]).
letztes(X,Y):-Y=[_|R],letztes(X,R).
```

Leider…

…funktioniert das Programm nicht! Für manche Fälle gibt es eine Lösung aus, für manche auch nicht.

WARUM???

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Analyse

Der Grund liegt in der Zeile

```
weg(X):-X=[A,B|C],nicht_element(A,C),weg([B|C]),tuer(A,B).
```

Da C nicht initialisiert ist, kann Prolog alles dafür annehmen:

- A nicht Element [1, 2, 3]
- A nicht Element [1, 2, 3, 4]
- A nicht Element [1, 2, 3, 4, 5]
- …

Prof. Dr.-Ing. Jens Gallenbacher

Analyse

Der Grund liegt in der Zeile

```
weg(X):-X=[A,B|C],nicht_element(A,C),weg([B|C]),tuer(A,B).
```

Die Frage nach einem bestimmten Weg funktioniert also
weg([a, e, i, m, n, j]). → Yes.
aber nicht die allgemeine
weg(X).
…bringt einige Ergebnisse und läuft dann aus dem Ruder.

Prof. Dr.-Ing. Jens Gallenbacher

Analyse

Man benötigt neben den Sprachkonstrukten noch eine Technik, hier den Akkumulator:

```
weg(X):-weg(X,[]).
weg([A,B], Besucht):-tuer(A,B), nicht_element(A, Besucht).
weg([A,B], Besucht):-tuer(B,A), nicht_element(B, Besucht).
weg(X, Besucht):-
    X=[A,B|C],
    (tuer(A,B);tuer(B,A)),
    nicht_element(A, Besucht),
    weg([B|C], [A | Besucht]).
```

Analyse

"Besucht" wird hier also in "weg" mitgeschleppt.

Anhand des Beweisbaumes kann man sogar nachvollziehen, wie das Programm funktioniert.

Für das selbständige Aufstellen eines solchen Programms muss man sich aber recht (sehr!) lange mit der Materie beschäftigen…

→ In der Schule ????

Prof. Dr.-Ing. Jens Gallenbacher

Analyse

Offenbar ist einer der Faktoren, der zu Problemen führt die Negation!

Probieren wir das mal aus anhand des Logical-Beispiels vom Anfang, diesmal mit Listen.

(*** logical4.pl ***)

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Negation

Aus einem positiven Term sind direkt Variablen mit einem oder (in der Iteration) mehreren Werten zu unifizieren - immer jedoch mit endlich vielen.

Aus einem negativen Term kann man oft unendlich viele Werte an eine Variable binden. Aufgrund der Sprachlogik werden dann weiter unten stehende Konstrukte nicht mehr ausgeführt, die ggf. "für Klarheit sorgen" könnten.

Für Sarndt kann alles mögliche eingesetzt werden

```
member([_,arndt,_,Sarndt],X),Sarndt\==lauch
```

Negation

Eine Überlegung ist daher, Negation in der Schule "zu ächten" und Aufgaben zu stellen, die keine Negation erfordern oder bei denen es einen Workaround für Negation gibt.

(*** logical4a.pl ***)

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Alternative

Alternativ kann Prolog auch mehr oder weniger imperativ programmiert werden…

```
print_rows(R,M):-
  write(R), tab(2),
  print_cols(1,R,M), nl,
  Rplus1 is R+1,
  print_rows(Rplus1,M).
```

Dabei kommt es allerdings meiner Meinung nach zu einem "fundamentalen Irrtum": Man nutzt ein Paradigma, um die Problemlösung schwieriger und unübersichtlicher und zu machen.

Der Cut

Der Cut, ausgedrückt durch das Ausrufungszeichen "!", verhindert ein weiteres Backtracking VOR die Stelle, an der der Cut steht.

Beispiel (zusammen mit der Familie von vorne)

einkind(X):-akind(_, X).

einkind(X) gibt alle Personen aus, die Kind von irgendwem sind (alle doppelt, weil sie Kind von zwei Personen sind).

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Cut

Allerdings könnte ja sein, dass wir zufrieden sind, wenn wir EINE Lösung gefunden haben, dann könnten wir schreiben:

einkind(X):-akind(_, X), !.

Der Cut ist immer WAHR, der Beweisbaum wird also immer weiter abgearbeitet, allerdings nie ZURÜCK verfolgt → Wir bekommen nur noch eine einzige Ausgabe.

Prof. Dr.-Ing. Jens Gallenbacher

Cut

Die Regel "tanz" soll für den Abschlußball die Elterntanzkombis (Mutter/Sohn, Vater/Tochter) ermitteln:

```
tanz(X,Y):-mann(X),frau(Y),kind(X,Y).
tanz(X,Y):-frau(X),mann(Y),kind(X,Y).
```

Was wird abgearbeitet, wenn wir fragen

```
tanz(paul,A).
```

Prof. Dr.-Ing. Jens Gallenbacher

Cut

```
tanz(X,Y):-mann(X),frau(Y),kind(X,Y).
tanz(X,Y):-frau(X),mann(Y),kind(X,Y).

tanz(paul,A).
```

Erst werden alle Lösungen des ersten Prädikats ermittelt und dann versucht, alle Lösungen des zweiten zu ermitteln.

Das zweite KANN aber gar nicht erfüllt sein, wenn mann(X) bereits erfüllt ist. Allerdings wissen nur wir (und nicht Prolog), dass mann(X) ein Ausschlusskriterium für frau(X) ist...

Cut

Daher setzt man hier den Cut ein, um Rekursionsschritte einzusparen:

```
tanz(X,Y):-mann(X),!,frau(Y),kind(X,Y).
tanz(X,Y):-frau(X),!,mann(Y),kind(X,Y).
tanz(paul,A).
```

Das funktioniert schneller, denn der Cut verhindert auch die Abarbeitung weiterer Regeln.

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Cut

ABER:

tanz(X,Y):-mann(X),!,frau(Y),kind(X,Y).

tanz(X,Y):-frau(X),!,mann(Y),kind(X,Y).

tanz(P,anna).

→ No

Warum?

mann(jens).	frau(brita).	kind(claire, thomas).
mann(peter).	frau(olga).	paar(gudrun, paul).
mann(klaus).	frau(gudrun).	paar(anna, peter).
mann(daniel).	kind(paul, anna).	paar(jens, claire).
mann(thomas).	kind(anna, claire).	paar(klaus, olga).
mann(paul).	kind(peter, klaus).	
mann(tobias).	kind(olga, brita).	
frau(claire).	kind(olga, tobias).	
frau(anna).	kind(jens, daniel).	

Prof. Dr.-Ing. Jens Gallenbacher

Cut

ABER:

```
tanz(X,Y):-mann(X),!,frau(Y),kind(X,Y).
tanz(X,Y):-frau(X),!,mann(Y),kind(X,Y).
```

```
tanz(P,anna).
```

→ No

Hier nimmt Prolog für "mann(X)" die erste Lösung "mann(jens)", überliest den Cut, stellt fest, dass "kind(jens,anna)" falsch ist. Keine weitere Lösung wird durch Backtracking gesucht, weil wir dies mit dem Cut verhindert haben!

Prof. Dr.-Ing. Jens Gallenbacher

Cut

ABER:

```
tanz(X,Y):-mann(X),!,frau(Y),kind(X,Y).
tanz(X,Y):-frau(X),!,mann(Y),kind(X,Y).
```

```
tanz(P,anna).
```

→ No

→ Ob der Cut so funktioniert, wie man es intendiert hat, hängt in der Regel von der Art der Anfrage ab.

Man unterscheidet in rote und grüne Cuts. Ein grüner Cut ist ein Cut, bei dem alle Anfragen zum gleichen Ergebnis führen wie ohne Cut. Ein roter Cut unterdrückt ggf. Ergebnisse.

Cut

(Wird oft nicht verinnerlicht, daher nochmal explizit:)

Der Cut hat eine lokale Wirkung (schneidet alle Teilbäume des Entscheidungsbaumes links von Cut ab) und eine globale Wirkung (schneidet alle Teilbäume oberhalb der Regel mit dem Cut ab).

Beides ist allerdings hinreichend erklärt mit der Regel "Kein Backtracking über den Cut hinaus zurück".

Prof. Dr.-Ing. Jens Gallenbacher

Prolog in der Schule

Deklaratives Paradigma kommt in kleinen Beispielen gut heraus.

Ohne Rekursion und ohne Negation

- Black-Box-Prinzip
- Programmierer gibt Fakten und Regeln vor
- Erhält weitergehende Erkenntnisse

Prof. Dr.-Ing. Jens Gallenbacher

Prolog in der Schule

(Leicht) tiefergehende Programmierung setzt Wissen über interne Vorgehensweise der Sprache voraus, insbesondere der Unifikation.

Das bedarf einer längere Beschäftigung mit der Sprache!

→ Kaum in der Schule zu leisten
→ Führt zu unglücklicher Verwendung des Paradigma
→ Nur primitive Beispiele (wie auch im Zentralabi)

Prof. Dr.-Ing. Jens Gallenbacher

Prolog in der Schule

Frage, ob Prolog in dieser abgespeckten Form überhaupt als solches Sinn macht und von den Schülern verinnerlicht wird.

→ Wird immer ein Hineinschnuppern bleiben!

Frage zur Diskussion: Reicht Basis-Prolog, um die fundamentale Idee "anderes Programmierparadigma" didaktisch umzusetzen?

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Alternativen?

Datalog

- sehr eingeschränktes Prolog als Datenbankbeschreibungs- und zugriffssprache.
- Nicht als Ersatz für Prolog verwendbar
- http://www.fdi.ucm.es/profesor/fernan/DES/

ABER:

Baut den Entscheidungsbaum mit Breitensuche auf, bietet daher die Einhaltung des Kommutativgesetzes.

→Nachfahrenregeln in Datalog beide verwendbar

Diskussion: Weitere Alternativen?

Zusammenfassung Prolog

- Deklaratives Programmierparadigma
- Imperative Betrachtungen wie Reihenfolge der Prädikate spielen bei nichtrekursiv deklarierten Fakten und Regeln ohne Negation keine Rolle (außer kürzere oder längere Ausführung)
- Rekursive Regeln setzen ein tieferes Verständnis der internen Funktionsweise der Sprache voraus. Reihenfolge der der Prädikate kann Korrektheit beeinflussen

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Zur Vertiefung: Aufgaben

Gegeben seien die Fakten p(1), p(2) und p(3).

Ergebnisse von:

?- p(X).

?- p(X), p(Y).

?- p(X), !, p(Y).

Wir ersetzen p(2). durch p(2):-!.

Effekt? Beweisbaum zur Veranschaulichung.

Welche Arten von Cut existieren?

Vor- und Nachteile der Einführung des Cut in der Schule.

Aufgabe

p(1); p(2); p(3).

?- p(X).

Prof. Dr.-Ing. Jens Gallenbacher

Aufgabe

```
p(1); p(2); p(3).

?- p(X), p(Y).
```

Prof. Dr.-Ing. Jens Gallenbacher

Aufgabe

```
p(1); p(2); p(3).

?- p(X), !, p(Y).
```

Prof. Dr.-Ing. Jens Gallenbacher

Aufgabe

```
p(1); p(2):-!; p(3).

?- p(X).
```

Prof. Dr.-Ing. Jens Gallenbacher

Aufgabe

```
p(1); p(2):-!; p(3).

?- p(X), p(Y).
```

Prof. Dr.-Ing. Jens Gallenbacher

Aufgabe

```
p(1); p(2):-!; p(3).

?- p(X), !, p(Y).
```

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

X=1; X=2; X=3.

X=1, Y=1; X=1, Y=2; X=1, Y=3; X=2, Y=1; X=2, Y=2; X=2, Y=3; X=3, Y=1; X=3, Y=2; X=3, Y=3.

X=1, Y=1; X=1, Y=2; X=1, Y=3.

X=1; X=2.

X=1, Y=1; X=1, Y=2; X=2, Y=1; X=2, Y=2.

X=1, Y=1; X=1, Y=2.

Roter Cut, grüner Cut: Es ist immer kontextabhängig, ob ein Cut im Rahmen seiner Nutzung die Ergebnisse verändert. Cut führt noch mehr dazu, dass deklaratives Paradigma zu einer Art imperativem Paradigma „von hinten durch die Brust ins Auge wird“.

Aufgabe

Diskutieren Sie Vor- und Nachteile der Verwendung von Prolog als Sprache für ein nicht-imperatives Paradigma im Unterricht. Gehen Sie dabei auf Fälle ein, in denen Prolog vom rein deklarativen Paradigma abweicht.

Prof. Dr.-Ing. Jens Gallenbacher

Prof. Dr.-Ing. Jens Gallenbacher

Prolog ohne Rekursion und ohne Negation ist rein deklarativ. Bei Rekursion kann es zum Bruch beim Kommutativgesetz kommen:

nachfahre(X, Y) :- kind(X, Y).

nachfahre(X, Y) :- kind(X, Z), nachfahre(Z, Y).

nachfahre(X, Y) :- nachfahre(Z, Y), kind(X, Z).

... führt zur Endlos-Rekursion

Verhalten ist nur mittels Beweisbaum verständlich, was allerdings dazu führt, dass man die imperative Vorgehensweise Prologs begreifen muss.

Auch die Negation ist problematisch, da Prolog einer Aussage X != Y für X quasi hintereinander alles außer Y einsetzen kann.

Prof. Dr.-Ing. Jens Gallenbacher

Aufgabe

„Drei Freunde erzielen in einem Wettbewerb die ersten drei Plätze. Jeder der drei hat einen anderen Vornamen, mag eine unterschiedliche Sportart und gehört einer anderen Nation an. Michael mag Basketball und war besser als der Amerikaner. Simon, der Israeli, war besser als der Tennisspieler. Der Kricketspieler gewann. Wer ist der Australier? Welchen Sport betreibt Richard?“

Bitte mit Prolog lösen!

Prof. Dr.-Ing. Jens Gallenbacher


```
p(1).
p(2).
p(3).
l(Michael,Simon, Richard, Basketball, Tennis, Kricket, Amerika, Israel,
Australien):-
  p(Michael), p(Simon), p(Richard),
  p(Basketball), p(Tennis), p(Kricket),
  p(Amerika), p(Israel), p(Australien),
  Michael\==Simon, Michael\==Richard, Simon\==Richard,
  Basketball\==Tennis, Basketball\==Kricket, Tennis\==Kricket,
  Amerika\==Israel, Amerika\==Australien, Israel\==Australien,
  Michael=Basketball, Michael<Amerika,
  Simon=Israel, Simon<Tennis,
  Kricket=1.
```

Prof. Dr.-Ing. Jens Gallenbacher


```
loesung(X) :-
  X=[[1,_,_,_],[2,_,_,_],[3,_,_,_]],
  member([_,michael,_,_],X), member([_,simon,_,_],X),
member([_,richard,_,_], X),
  member([_,_,basketball,_],X), member([_,_,tennis,_],X),
member([_,_,kricket,_], X),
  member([_,_,_,amerika],X), member([_,_,_,israel],X),
member([_,_,_,australien], X),
  member([_,michael,basketball,_],X),
  member([Nmichael,michael,_,_],X),
member([Namerika,_,_,amerika],X), Nmichael<Namerika,
  member([_,simon,_,israel],X),
  member([Nsimon,simon,_,_],X), member([Ntennis,_,tennis,_],X),
Nsimon<Ntennis,
  member([1,_,kricket,_],X).
```

Fragen, Anregungen, Kommentare !

Prof. Dr.-Ing. Jens Gallenbacher