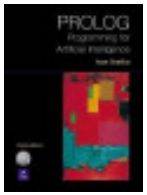
Vorlesung: „*Künstliche Intelligenz*“

Literatur zur Vorlesung

Luger, G.F.: „*Künstliche Intelligenz*“, 4.Auflage, Pearson Studium Verlag 2001



Bratko, I.: „*PROLOG Programming for Artificial Intelligence*“, 3.Auflage, Pearson Verlag 2001

Ertel, W.: „*Grundkurs Künstliche Intelligenz*“, Vieweg Verlag 2008

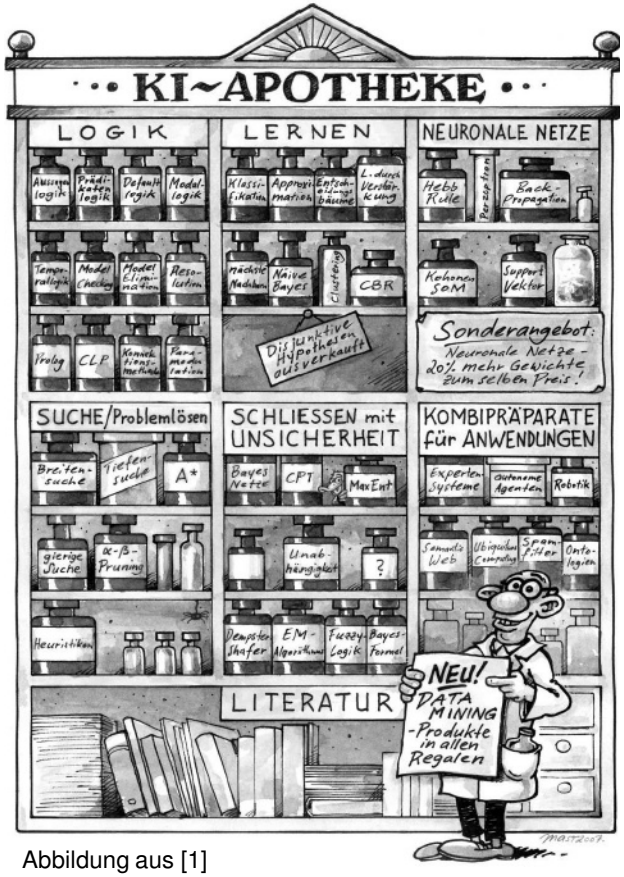


Abbildung aus [1]

Organisatorisches zur Veranstaltung

- Vorlesungen Do 14-16 Uhr in SR006
  Marco Block (block@mi.fu-berlin.de)

- Tutorien Fr 12-14 Uhr und 14-16 Uhr in SR006
  Übungsleiter: Miao Wang (mwang@mi.fu-berlin.de)

- Scheinkriterien:
  60% der Punkte auf den Übungszetteln
  n-1 Übungszettel mit mindestens 20%
  Bestehen der Klausur am 16.07.2009

- Übungszettel:
  Ausgabe der Übungszettel wöchentlich Do/Fr
  Abgabe in der darauf folgenden Woche Fr 12 Uhr
  2 bis 3-er Gruppen
  Programmierabgaben zusätzlich per Email
  Sprachen: SWI-PROLOG und später Java

- Literaturhinweise, Übungszettel, Themen und Folien erscheinen auf der VL-Seite

Vorlesung 1

- Definition von „Künstlicher Intelligenz“
- Meilensteine und Arbeitsfelder der KI
- Einführung in die Programmierung mit PROLOG

Definition von „Künstlicher Intelligenz“

1950 Alan Turing (Turingtest):

„Im Zuge dieses Tests führt ein menschlicher Fragesteller über eine Tastatur und einen Bildschirm ohne Sicht- und Hörkontakt mit zwei ihm unbekannten Gesprächspartnern eine Unterhaltung.

Der eine Gesprächspartner ist ein Mensch, der andere eine Maschine. Beide versuchen, den Fragesteller davon zu überzeugen, dass sie denkende Menschen sind.

Wenn der Fragesteller nach der intensiven Befragung nicht klar sagen kann, welcher von beiden die Maschine ist, hat die Maschine den Turing-Test bestanden.“

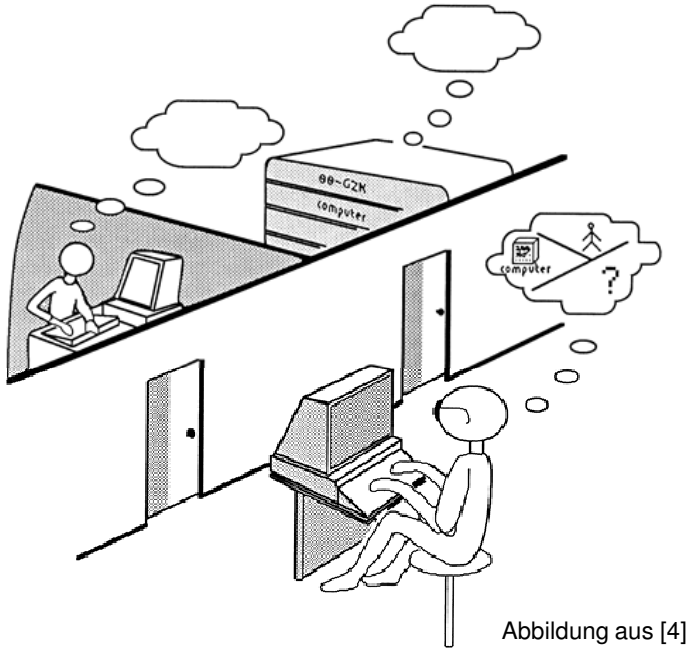


Abbildung aus [4]

Definition von „Künstlicher Intelligenz“

1955 John McCarthy:

„Ziel der KI ist es, Maschinen zu entwickeln, die sich verhalten, als verfügten sie über Intelligenz.“

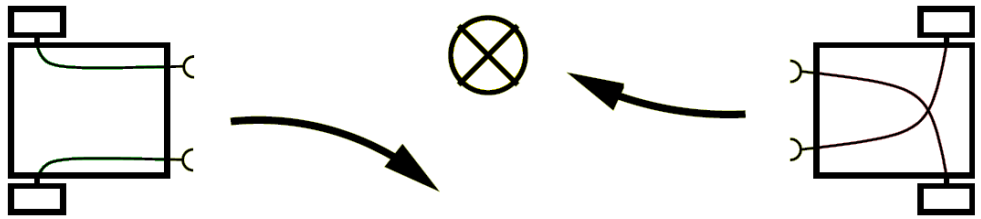


Abbildung aus [1]

Braitenberg-Vehikel, Reaktion auf Lichtquelle [2]

Definition von „Künstlicher Intelligenz“

1991 Encyclopedia Britannica:

„KI ist die Fähigkeit digitaler Computer oder computergesteuerter Roboter, Aufgaben zu lösen, die normalerweise mit den höheren intellektuellen Verarbeitungsfähigkeiten von Menschen in Verbindung gebracht werden ...“

Definition von „Künstlicher Intelligenz“

1983 Elaine Rich:

> *„Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better.“*

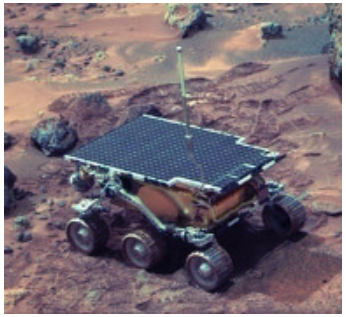
Gute Beschreibung der Tätigkeit von Wissenschaftlern der KI in der Vergangenheit und sicher auch Zukunft.

Ziel: Im Laufe der Vorlesung zu einer eigenen Definition zu gelangen.

Ausgewählte Meilensteine der KI

- **1931** Gödel zeigt, dass in der Prädikatenlogik erster Stufe alle wahren Aussagen herleitbar sind
- **1937** Alan Turing zeigt mit Halteproblem die Grenzen intelligenter Maschinen auf
- **1943** McCulloch Pitts modellieren Neuronale Netze
- **1950** Alan Turing definiert den Turingtest
- **1951** Marvin Minsky entwickelt einen Neuronenrechner
- **1955** Arthur Samuel entwickelt lernfähige Dameprogramme
- **1956** Konferenz im Dartmouth College, Name Artificial Intelligence wird eingeführt LogicTheorist
- **1957** Simon und Newell entwickeln General Problem Solver
- **1958** LISP
- **1959** Geometry Theorem Prover

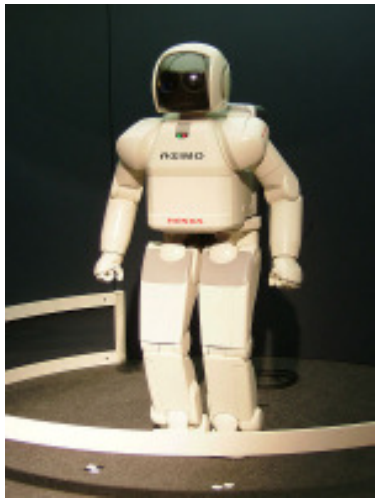
Ausgewählte Meilensteine der KI

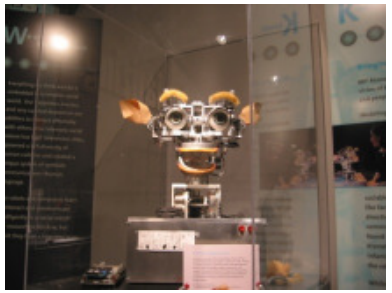


- **1965** Robinson beschreibt das Resolutionskalkül für Prädikatenlogik; Zadeh formuliert Fuzzy-Logik
- **1966** Weizenbaum entwickelt Eliza
- **1972** PROLOG; Expertensystem zur Diagnose von Krankheiten
- **1976** Shortliffe und Buchanan entwickeln MYCIN
- **1990** Bayes-Netze, Data Mining
- **1992** Tesauro entwickelt selbstlernendes Back Gammon-Programm (Reinforcement Learning) und spielt stärker als der Weltmeister
- **1993** RoboCup Initiative
- **1995** Vapnik beschreibt die Support Vector Maschines
- **1996** Mars Pathfinder, Sojourner
- **1997** erster internationaler RoboCup Wettkampf; Deep(er) Blue von IBM schlägt den Schachweltmeister Kasparow

Ausgewählte Meilensteine der KI

- **1998** Robosapiens Kismet vom MIT
- **2000** Roboter Asimo der Firma Honda
- **2007** Schaeffer löst das Spiel Dame (Chinook)





Ausgewählte Arbeitsfelder der KI

- Konnektionismus, Expertensysteme
- Logischen und Probabilistisches Schließen
- Planung, Entscheidung
- Lernen, Optimieren
- Wissensrepräsentation
- Bildverarbeitung
- Mustererkennung
- Spieltheorie
- Spieleprogrammierung
- Autonome Systeme
- Robotik
- ...

Inhaltliche Planung für die Vorlesung

1) Definition und Geschichte der KI, PROLOG
2) Expertensysteme
3) Probabilistisches und Logisches Schließen, Resolution
4) Spieltheorie, Suchen und Planen
5) Spieleprogrammierung
6) General Game Playing
7) Reinforcement Learning und Spieleprogrammierung
8) Mustererkennung
9) Neuronale Netze
10) Optimierungen (genetische und evolutionäre Algorithmen)
11) Bayes-Netze, Markovmodelle
12) Robotik, Pathfinding

der rote Vorlesungsfaden...

Wissensbasierte Systeme

Inferenzmechanismus: Trennung von Wissensbasis und Inferenz

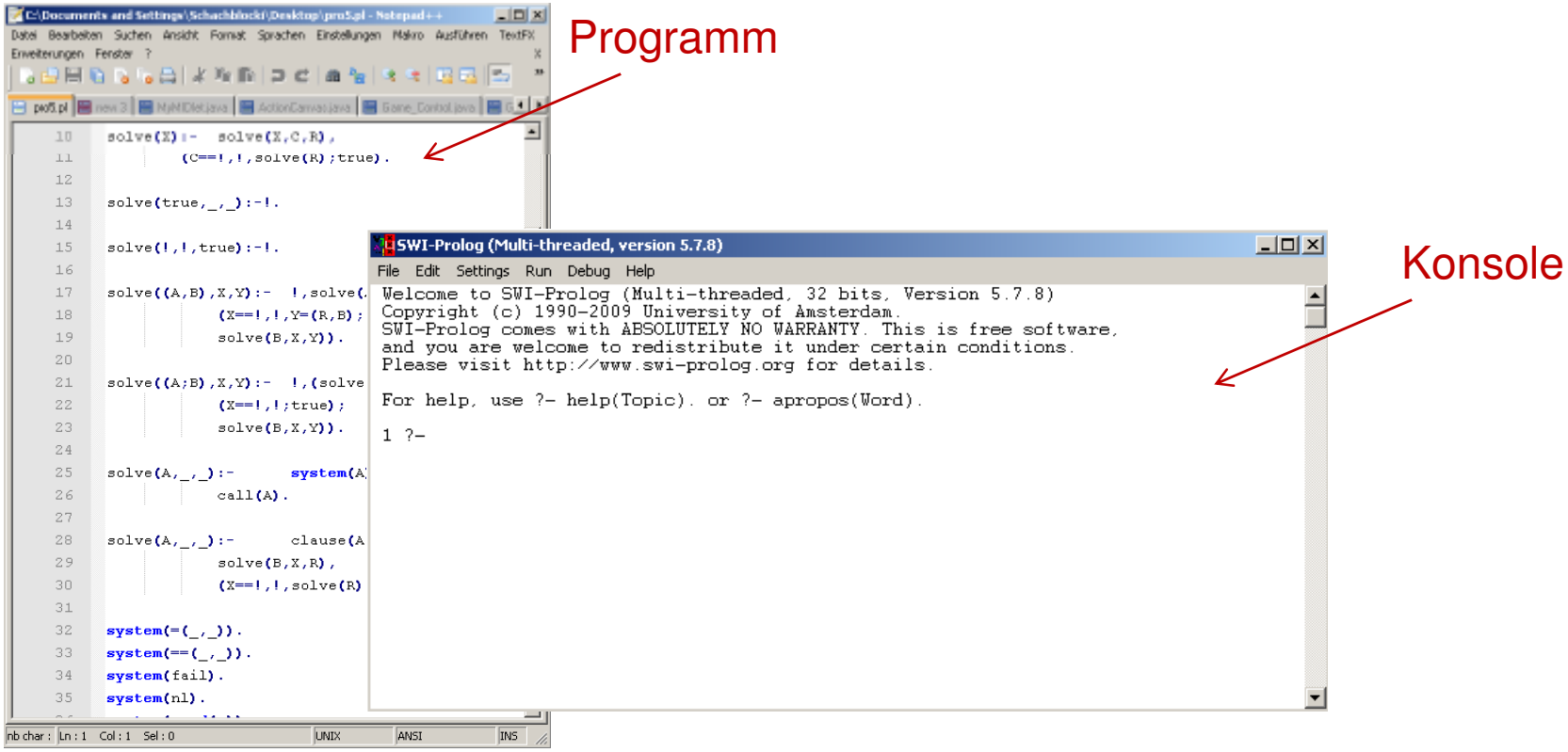
Vorteil: Wissensbasis einfach austauschbar, ohne System neu zu programmieren

das führt uns zu PROLOG

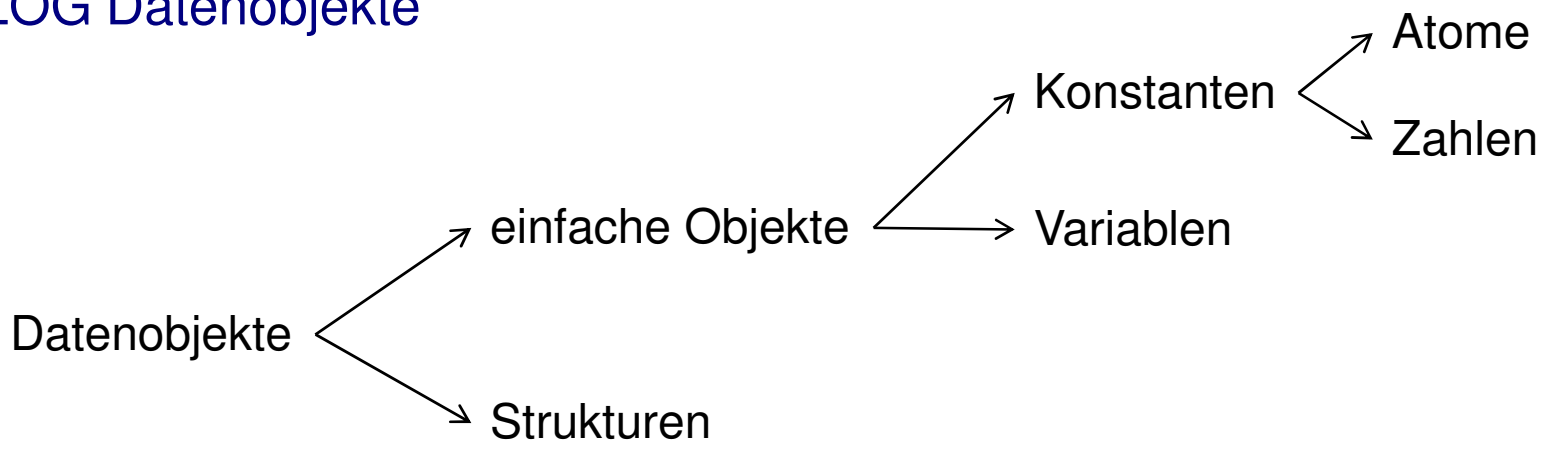
PROLOG – Wahl des Interpreters

Wir verwenden für die Veranstaltung SWI-Prolog Version 5.78 (development version)

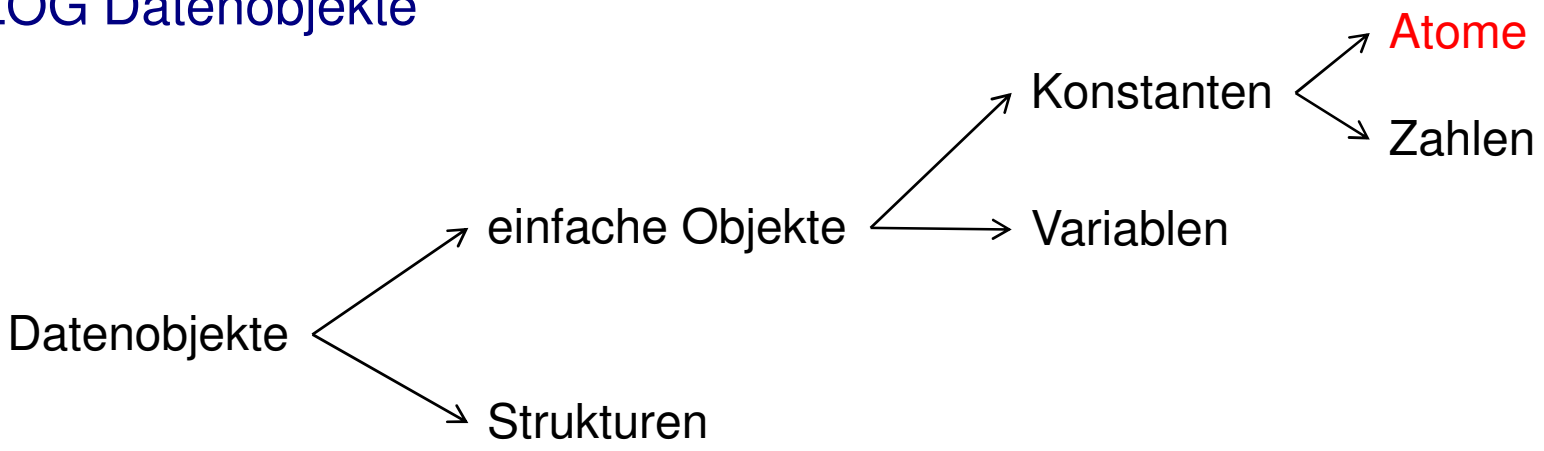
http://www.swi-prolog.org/



PROLOG Datenobjekte



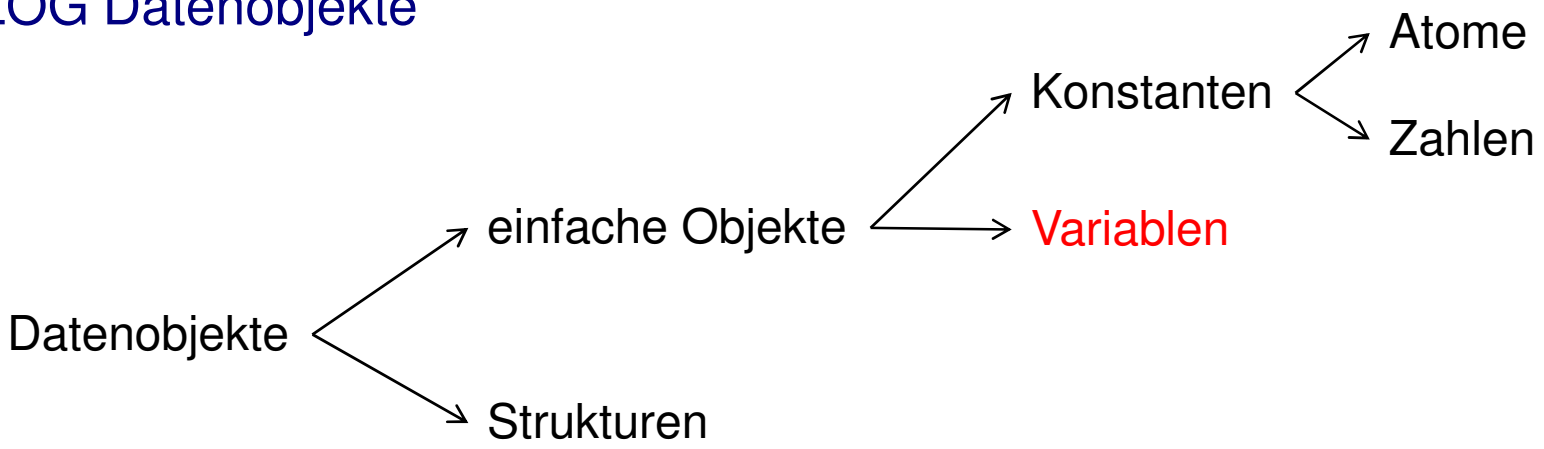
PROLOG Datenobjekte



Beispiele für Atome:

```
anna
nil
x25
x_
miss_Jones
```

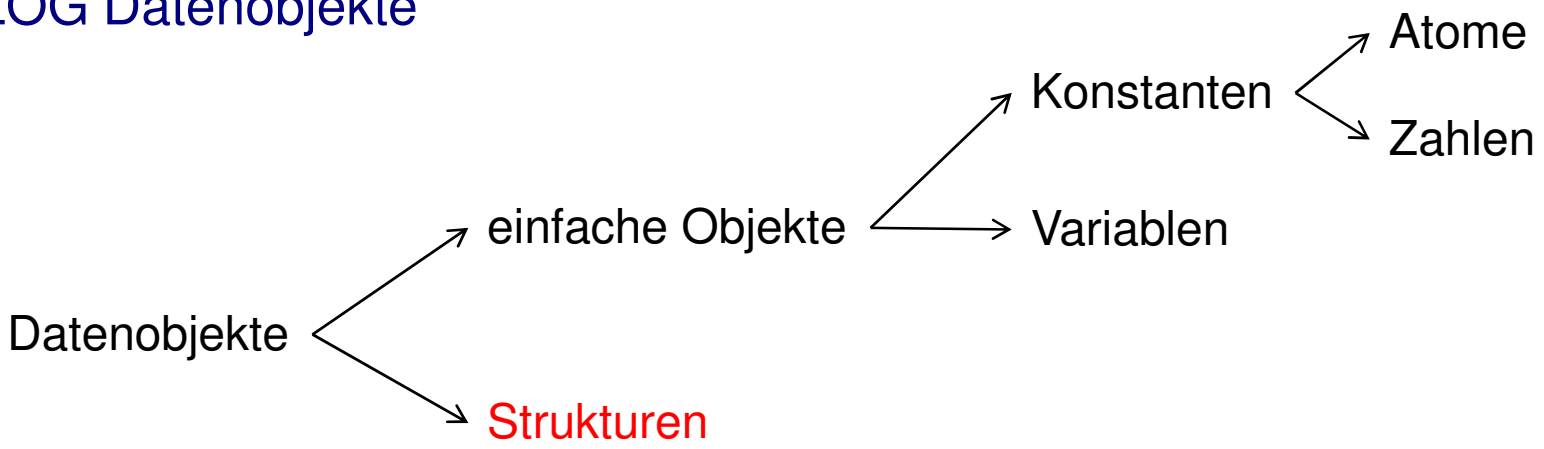
PROLOG Datenobjekte



Beispiele für Variablen:

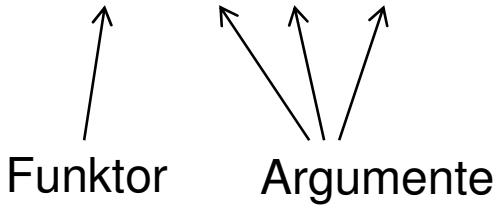
```
X
Result
_x23
ObjectList
```

PROLOG Datenobjekte



Beispiel für eine Struktur:

```
date(1,may,2001)
```



PROLOG

Die Sprache PROLOG hängt stark mit der Prädikatenlogik zusammen.

Prädikatenlogik	Bedeutung	PROLOG
$\wedge$	und	`,`
$\vee$	oder	`;`
$\Leftarrow$	dann, wenn	`:-`
$\neg$	nicht	`not`

Beispiel:

```
mag(peter, susanne) :- not(mag(karin, peter)).
```

Programm

PROLOG

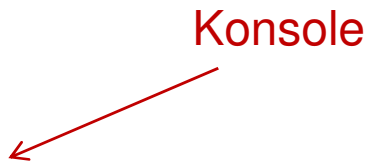
Das gesamte Wissen über die Welt befindet sich in der Datenbank. Findet sich dort nichts, ist die Anfrage falsch.

Annahme der Weltabgeschlossenheit (Closed World Assumption):
=> PROLOG nimmt alle Ziele als falsch an, deren Wahrheit nicht bewiesen werden kann.

Stichwort: Resolution

PROLOG – Dynamisches Arbeiten I

PROLOG besitzt eine Konsole zur Kommunikation. Dort können wir Anfragen stellen.



Prädikat P zur Menge der Spezifikationen hinzufügen:

```
?- assert(P).
```

Prädikat P an den Anfang (1) und enstprechend an das Ende (2) fügen:

```
?- asserta(P).   -- (1)
?- assertz(P).   -- (2)
```

Prädikat P wird entfernt:

```
?- retract(P).
```

PROLOG – Dynamisches Arbeiten II

Beispiel:

```
?- assert(male(tim)).
?- assert(male(tom)).
?- asserta(male(tony)).
?- assertz(male(tino)).
?- retract(male(tim))).

?- male(X).
X=tony;
X=tom;
X=tino;
no
```

Anzeige der aktuellen Datenbank:

```
?- listing.
```

Zum Glück müssen wir die Datenbank nicht zeilenweise füttern...

PROLOG – Skriptum

Gespeichert werden die Skripte mit `*.pl`, was sich aber nach Interpreter unterscheiden kann.

Aufbau einer PROLOG-Datei:

```
Fakten



Regeln
```

Datenbank von Klauseln

PROLOG – Fakten definieren

Fakten werden durch Relationen (beginnend mit einem Kleinbuchstaben) angegeben

Beispiel:

```
isParent(hans, tim).
```

isParent ist der Name der Relation, hans und tim sind die Argumente.

Wir können das PROLOG-System jetzt danach fragen:

```
?- isParent(hans, tim).
yes

?- isParent(hans, marco).
no
```

PROLOG – Listen I

Geordnete Menge von Elementen, die auch wieder Listen sein können.

```
[a,b,c,d]
[[1,2],[3,4]]
[a,[1,2],b,c]
[]
```

Kopf-und-Rest-Prinzip wie in Haskell

```
[H|T]
[X,Y|T]
```

oder in Funktorschreibweise

```
.(H,T)
```

PROLOG – Listen II

Beispiel:

```
[a,b,c]
```

Zerlegung mit [X|Y], liefert X=a und Y=[b,c]

Prädikat member prüft, ob das erste Argument Element der nachfolgenden Liste ist:

```
?- member(a,[a,b,c]).
yes

?- member(a,.(a,.(b,.(c,[])))).
yes

?- member(X,[b,c]).
X=b;
X=c;
no

?- member(X,[]).
no
```

PROLOG – Prädikat member

Rekursive Definition von member

```
member(X, [X|T]).                    -- Z1
member(X, [Y|T]) :- member(X,T).     -- Z2
```

Das passiert intern:

Z1 nicht, da c!=a
Z2 passt, da X=c, Y=a, T=[b,c]
(*) => ?- member(c,[b,c]).
Z1 nicht
Z2 passt, da X=c, Y=b, T=[c]
(**) => ?- member(c,[c]).
Z1 passt.
yes zu (**)
yes zu (*)
yes

PROLOG – Anonyme Variablen (wild cards)

Wir können dazu member wie folgt definieren

```
member(X, [X|_]).
member(X, [_|T]) :- member(X,T).
```

Länge einer Liste

```
laenge([],0).
laenge([_|T], N) :- laenge(T, N1), N is 1+N1.
```

Zuweisung und Auswertung unter PROLOG

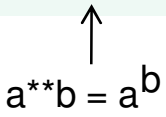
```
?- X=1+2.
X=1+2

?- X is 1+2.
X=3
```

PROLOG – Arithmetische und Vergleichs-Operatoren

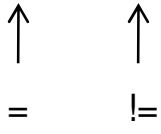
Arithmetik:

```
+, -, *, /, **, //, mod
```



Vergleiche:

```
>, <, >=, =<, =:=, =\=
```




```
?- 1+2=:=2+1.
yes
```

PROLOG – Ausgaben

Ausgabe der Listenelemente untereinander:

```
output_list([]).
output_list([H|T]) :- write(H), nl, output_list(T).
```

Liste spiegelverkehrt ausgeben:

```
r_output_list([]).
r_output_list([H|T]) :- r_output_list(T), write(H).
```

PROLOG – Konkatenation von Listen

„ha“ ++ „llo“ => „hallo“

```
concat([1,2], [3,4], [1,2,3,4]).
```

Implementierung:

```
concat([],L,L).
concat([X|L1], L2, [X|L3]) :- concat(L1, L2, L3).
```

```
?- concat([g,o],[o,g,l,e],L).
L=[g,o,o,g,l,e];
no
```

PROLOG – Konkatenation von Listen II

Wir stellen die Frage um...

```
?- concat(L1, L2, [a,b,c]).
L1=[]
L2=[a,b,c];

L1=[a]
L2=[b,c];

L1=[a,b]
L2=[c];

L1=[a,b,c]
L2=[];
no
```

Member durch concat definieren:

```
member(X,L) :- concat(L1,[X|L2],L).
```

oder:

```
member(X,L) :- concat(_,[X|_],L).
```

PROLOG – delete und insert

Definition von delete, lösche X aus einer Liste:

```
del(X, [X|Tail], Tail).
del(X, [Y|Tail], [Y|Tail2]) :- del(X, Tail, Tail2).
```

```
?- del(a, [a,b,a,a], L).
L=[b,a,a];
L=[a,b,a];
L=[a,b,a];
no

?- del(a, L, [1,2,3]).
L=[a,1,2,3];
L=[1,a,2,3];
L=[1,2,a,3];
L=[1,2,3,a];
no
```

darüber läßt sich insert definieren:

```
insert(X, List, List_with_X) :- del(X, List_with_X, List).
```

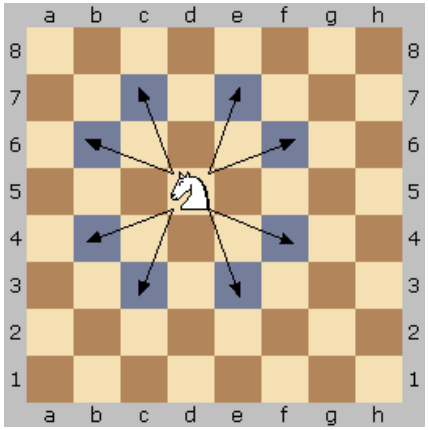
PROLOG – member durch delete

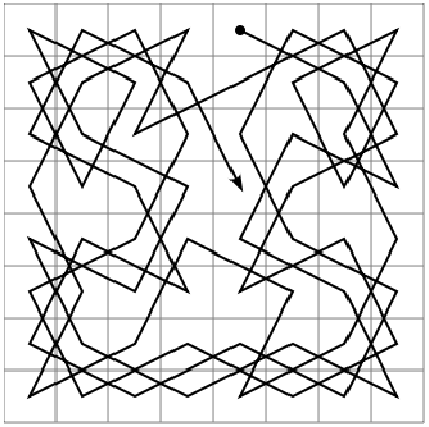
Definition von member über delete:

```
member(X, List) :- del(X, List, _).
```

Idee: Ein X ist member, wenn X aus der Liste gelöscht werden kann...

Springerproblem auf 3x3 Brett





<u>Prädikatenlogisch:</u>

Regel, die Zugfolgen der Länge 2 definiert

$$\forall X, Y\ [weg(X, Y) \leftarrow \exists Z\ [zug(X, Z) \wedge zug(Z, Y)]]$$

allgemeine Wegdefinition

$$\forall X\ weg(X, X).$$
$$\forall X, Y\ [weg(X, Y) \leftarrow \exists Z\ [zug(X, Z) \wedge weg(Z, Y)]]$$

Springerproblem auf 3x3 Brett

PROLOG:

```
zug(1,8).
zug(1,6).
...
zug(9,4).
```

1	2	3
4	5	6
7	8	9

Wegdefinition:

```
weg(X,Y) :- zug(X,W), not(besucht(W)),
            assert(besucht(W)), weg(W,Y).
```

Oder mit Liste:

```
weg(Z,Z,L).
weg(X,Y,L) :- zug(X,Z), not(member(Z,L)), weg(Z,Y,[Z|L]).
weg(X,Y)   :- weg(X,Y,[X]).
```

```
?- weg(1,4,[1]).
?- weg(1,4).
```

PROLOG Cut-Operator

Der Cut-Operator wird durch „!“ repräsentiert. Beim erstenmal erfolgreich ausgeführt und beim Backtracking mißlingt das gesamte Ziel, indem er enthalten ist.

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

$$p \Leftrightarrow (a \wedge b) \vee c$$

mit Cut-Operator

```
p :- a,!,b.
p :- c.
```

$$p \Leftrightarrow (a \wedge b) \vee (\neg a \wedge c)$$

vertauscht gilt sogar

```
p :- c.
p :- a,!,b.
```

$$p \Leftrightarrow c \vee (a \wedge b)$$

Literatur und Abbildungsquellen

[1] Ertel, W.: „*Grundkurs Künstliche Intelligenz*“, Vieweg Verlag 2007

[2] Braitenberg V.: „*Vehicles – Experiments in Synthetic Psychology*“, MIT Press, 1984

[3] Bratko, I.: „*PROLOG Programming for Artificial Intelligence*“, 3.Auflage, Pearson Verlag 2001

[4] Copeland J.: „*Artificial Intelligence: A Philosophical Introduction*“, Oxford UK and Cambridge, 1993