Praktische Informatik 2
Sommer-Semester 2005

Prof. Dr. sc. Hans-Dieter Burkhard

www.ki.informatik.hu-berlin.de

Praktische Informatik 2

Themen

- Alternative Programmiermethoden:

  Deklaratives (logisches Programmieren): Prolog

  Auf den Poolrechnerrn:
  /usr/local/praktikum/SWIprolog/bin/pl

- Abstrakte Datenstrukturen

  (Listen, Graphen, Bäume ...)

Repräsentation von „Wissen“

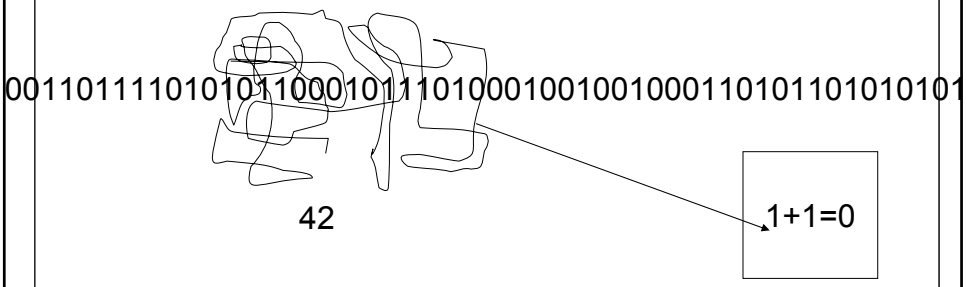
- Lexikon
- Datenbank
- Briefmarkensammlung
- Programme
- Landkarten
- Bilder
- Gesetze
- Regeln
- . . .

Darstellung und Interpretation
- Menschlicher Nutzer
- Maschinelle Auswertung

Zeichen, Symbole, Signale, ...

Zeichen/Symbole + Interpretation
Syntax + Semantik

Über allen Wipfeln ist Ruh‘

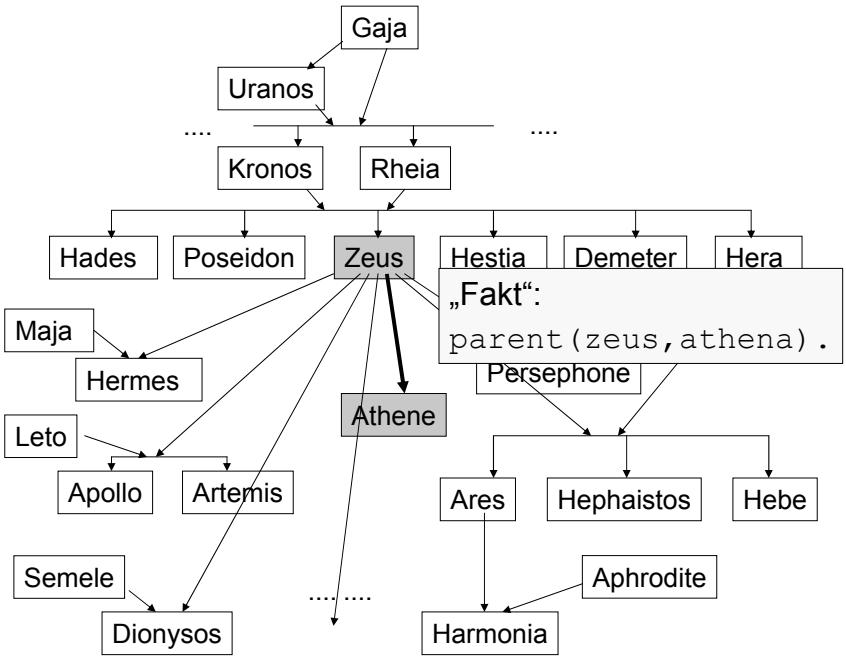


Explizites vs. Implizites Wissen

- explizit: z.B. Axiome, Schluss-Regeln
- Implizit: Folgerungen

Unterschiedliche Formen für
- Menschlicher Nutzer
- Maschinelle Auswertung

- Inferenz: Verfahren zur Herleitung von implizitem Wissen aus explizitem Wissen (z.B. Beweisverfahren)

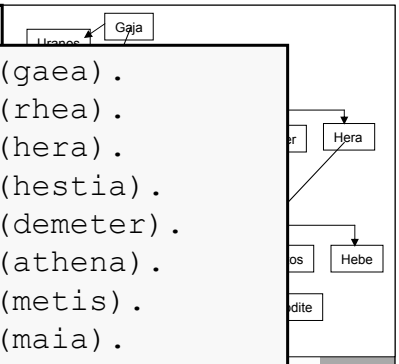


Griechische Götter: Fakten

```
parent(uranus, cronus).
parent(gaea,   ur
parent(gaea,   cr
parent(zeus,   hephaestus).
parent(hera,   hephaestus).
...
parent(demeter, persephone).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysi
male(hephaes
male(poseido
```

```
female(gaea).
female(rhea).
female(hera).
female(hestia).
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
female(artemis).
female(leto).
```



Fakten in Prolog

Ein Fakt hat die Form `funktor(argumente).`

funktor ist der Name einer n-stelligen Relation (Prädikat).
n ist die Anzahl der Argumente.

```
male(zeus).
parent(zeus,athena).
parent(zeus,hera,ares).
plus(3,4,7).
kleiner(6,9).
```

male/1
parent/2
parent/3
plus/3
true/0
fail/0

Definition einer Relation durch Aufzählung

Aufzählung: Die Beziehungen werden explizit aufgezählt (Faktenmenge, Datenbank)

```
parent(uranus, cronus).
parent(gaea,
parent(gaea,
parent(rhea,
parent(cronus,
parent(rhea,
parent(cronus,
parent(cronus,
parent(rhea,
parent(cronus,
parent(rhea,
parent(zeus,   hermes).
parent(maia,   hermes).
...
```

Hinzufügen:

```
asserta(male(caesar)).
assertz(male(augustus).)
```

Löschen:

```
retract(male(zeus)).
```

```
female(gaea).
female(rhea).
female(maia).
e(persephone).
e(aphrodite).
e(artemis).
e(leto).
```

```
male(dionysius).
male(hephaestus).
male(poseidon).
```


Anfragen

Eine Anfrage hat die Form `?-funktor(argumente).`

funktor ist der Name einer n-stelligen Relation (Prädikat).

```
?- male(zeus).
yes.
?- parent(zeus,athena).
yes.
?-parent(hera,athena).
no.
```

```
parent(uranus, cronus).
rent(gaea,   cronus).
rent(gaea,   rhea).
rent(rhea,   zeus).
rent(cronus, zeus).
rent(rhea,   hera).
rent(cronus, hera).
rent(cronus, hades).
rent(rhea,   hades).
rent(cronus, hestia
rent(rhea,   hestia
rent(zeus,   hermes
rent(maia,   hermes
.
```

```
female(gaea).
female(rhea).
female(hera).
female(hestia).
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

CWA: Closed World Assumption (1a)

Bedeutung der Antwort „no“?

Oder:
Welche Antwort gibt Prolog
für nicht gespeicherte Fakten?

`?-vater(zeus,athena).`

„Optimistische“ Variante: `yes.`
„Pessimistische“ Variante: `no.`

CWA: Closed World Assumption (1a)

(Vorläufige) Bedeutung der Antwort „no“: Der Fakt ist in der Datenbank nicht enthalten.

```
?-parent(hera,athena).
no.
?-parent(zeus,hera,ares).
no.
?-kleiner(3,7).
no.
```

```
parent(uranus, cronus).
rent(gaea,   cronus).
rent(gaea,   rhea).
rent(rhea,   zeus).
rent(cronus, zeus).
rent(rhea,   hera).
rent(cronus, hera).
rent(cronus, hades).
rent(rhea,   hades).
rent(cronus, hestia
rent(rhea,   hestia
rent(zeus,   hermes
rent(maia,   hermes
.
```

```
female(gaea).
female(rhea).
female(hera).
female(hestia).
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

Anfragen nach Existenz

Anfrage enthält Variable

```
?-funktor(argumente).
```

Variablen-Bezeichner beginnen mit Großbuchstaben

```
?- parent(zeus,X).
X=hermes?
;
X=athena?
;
X=ares?
;
no.
```

```
?- parent(zeus,X).
X=hermes?
;
X=athena?
.
yes.
```

```
parent(zeus,    hermes
parent(maia,    hermes
...
```

```
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus)
```

„ ; “ erwartet weitere Antworten. „ .“ schließt Anfrage ab.

CWA: Closed World Assumption (1b)

Bedeutung der Antwort „no“ ?

(Vorläufige) Bedeutung der Antwort „no“:
Es gibt keinen passenden Fakt in der Datenbank.

```
parent(gaea,    rhea).
parent(rhea,    zeus).
parent(cronus,  zeus).
parent(rhea,    hera).
parent(cronus,  hera).
rent(cronus,    hades).
rent(rhea,      hades).
rent(cronus,    hestia
rent(rhea,      hestia
rent(zeus,      hermes
rent(maia,      hermes
.
```

```
female(hestia).
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

```
?-father(X,Y).
no.
?-kleiner(M,N).
no.
?-parent(X,gaea).
no.
```

Relationen definieren durch Regeln

```
kopf:-körper.
```

```
goal:-subgoals.
```


Relationen definieren durch Regeln

Definition der Relation „Vater-Kind“:

```
father(Vater,Kind)
      :-parent(Vater,Kind), male(Vater).
```

```
father(X,Y):-parent(X,Y),male(X).
mother(X,Y):-parent(X,Y),female(X).
```

```
parent(X,Y,Z) :-father(X,Z),mother(Y,Z).
```

```
son(Sohn,Elternteil):-
parent(Elternteil,Sohn),male(Sohn).
```

```
grandfather(X,Z):-father(X,Y),parent(Y,Z).
grandmother(X,Z):-mother(X,Y),parent(Y,Z).
```

```
grandchild(X,Y):-grandfather(Y,X).
grandchild(X,Y):-grandmother(Y,Z).
```

Regel als logische Formel

Die Variablen in Regeln sind universell quantifiziert.
Die Relationen der rechten Seite sind konjunktiv verknüpft.

$goal(X_1,...,X_n)$:- $subgoal_1(X_1,...,X_n)$,..., $subgoal_m(X_1,...,X_n)$.

Wird aufgefasst als

$\forall X_1 ... \forall X_n$
$[\ subgoal_1(X_1,...,X_n) \wedge ... \wedge subgoal_m(X_1,...,X_n) \rightarrow goal(X_1,...,X_n)\ ]$

oder

$\forall X_1 ... \forall X_n$
$[\neg subgoal_1(X_1,..,X_n) \vee ... \vee \neg subgoal_m(X_1,..,X_n) \vee goal(X_1,..,X_n)\ ]$

Regel als logische Formel

Hornklausel:

Alternative mit maximal

einem nicht-negiertem Literal

$\forall X_1 ... \forall X_n$
$[\neg subgoal_1(X_1,..,X_n) \vee ... \vee \neg subgoal_m(X_1,..,X_n) \vee goal(X_1,..,X_n)\ ]$

Regel als logische Formel

Variable, die nur im Regelkörper auftreten, können als existentiell quantifizierte Variable **innerhalb des Regelkörpers** (!) betrachtet werden:

$\forall X_1 ... \forall X_n \forall Y_1 ... \forall Y_k$ [
$subgoal_1(X_1,...,X_n,Y_1,...,Y_k) \wedge ... \wedge subgoal_m(X_1,...,X_n,Y_1,...,Y_k)$
$\rightarrow goal(X_1,...,X_n)$]

ist logisch äquivalent zu

$\forall X_1 ... \forall X_n$ [
$\exists Y_1 .. \exists Y_k [subgoal_1(X_1,..,X_n,Y_1,..,Y_k) \wedge ... \wedge subgoal_m(X_1,..,X_n,Y_1,..,Y_k)]$
$\rightarrow goal(X_1,...,X_n)$]

```
grandfather(X,Z):-father(X,Y),father(Y,Z).
```


Regel als logische Formel

Intuitive Bedeutung:

„goal“ gilt (ist beweisbar)
falls alle „subgoals“ gelten (beweisbar sind).

entspricht anschaulich der Abtrennungsregel
(modus ponens)

$$\frac{H_1 \rightarrow H_2 \ , \ H_1}{H_2}$$

$goal(X_1,...,X_n)$:- $subgoal_1(X_1,...,X_n)$,..., $subgoal_m(X_1,...,X_n)$.

Unbenannte/anonyme Variable

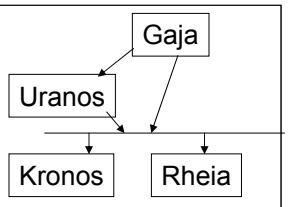
Unbenannte/anonyme Variable: _ für „beliebig“

```
mother_in_law(X,Y):-
              mother(X,Z),parent(Y,Z,_).
```

```
mother_in_law(X,Y):-
              mother(X,Z),parent(Z,Y,_).
```


Anfrage/Beweis

```
?- mother_in_law(gaea,gaea).
```



Zu beweisen:

```
mother_in_law(gaea,gaea).
```

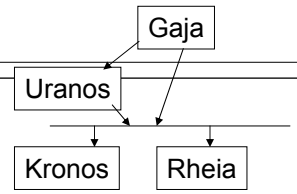
verfügbare Klausel (z.B.):

```
mother_in_law(X,Y):-
              mother(X,Z),parent(Z,Y,_).
```

Klausel mit Bindung `X=gaea, Y=gaea, Z=`$Z_{[1]}$

```
mother_in_law(gaea,gaea):-
     mother(gaea,Z[1]),parent(Z[1],gaea,_).
```

Anfrage/Beweis



zu beweisen:

```
mother_in_law(gaea,gaea):-
     mother(gaea,Z[1]),parent(Z[1],gaea,_).
```

dafür zu beweisen

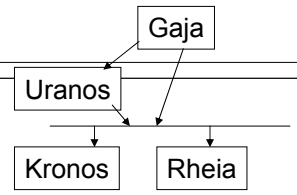
1)

```
mother(gaea,Z[1]).
```

2)

```
parent(Z[1],gaea,_).
```


Anfrage/Beweis



zu beweisen:

1) `mother(gaea,Z[1]).`

Verfügbare Klausel:

```
mother(X,Y):-parent(X,Y),female(X).
```

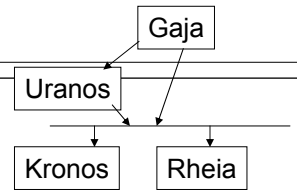
Klausel mit Bindung `X=gaea, Y=Z[1]` :

```
mother(gaea,Z[1]):-parent(gaea,Z[1]),female(gaea).
```

zu beweisen 1.1) `parent(gaea, Z[1] ).`

zu beweisen 1.2) `female(gaea).`

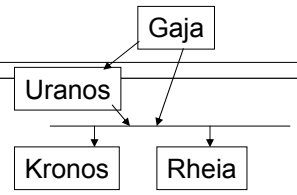
Anfrage/Beweis



zu beweisen:

1.1) `parent(gaea, Z[1] ).`

Anfrage/Beweis



zu beweisen:

1.1) `parent(gaea, Z[1] ).`

verfügbarer Fakt (z.B.), bewirkt Bindung `Z[1] =uranus` :

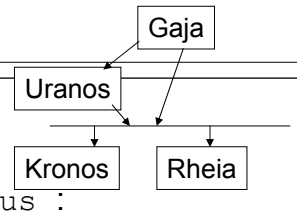
```
parent(gaea,uranus).
```

1.2) `female(gaea).`

verfügbarer Fakt:

```
female(gaea).
```

Anfrage/Beweis



zu beweisen

unter Beachtung der Bindung $Z_{[1]}$=uranus :

2) `parent(uranus,gaea,_).`

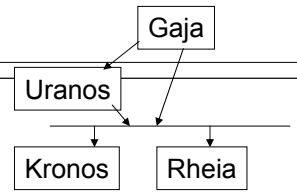
Verfügbare Klausel:

```
parent(X,Y,Z) :-father(X,Z),mother(Y,Z).
```

Klausel mit Bindungen :

```
parent(uranus,gaea,Z[2]) :-
      father(uranus,Z[2]),mother(gaea,Z[2]).
```


Anfrage/Beweis



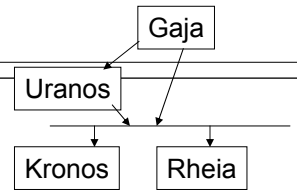
zu beweisen

```
parent(uranus,gaea,Z[2]) :-
      father(uranus,Z[2]),mother(gaea,Z[2]).
```

zu beweisen 2.1) `father(uranus,Z[2]).`

zu beweisen 2.2) `mother(gaea,Z[2]).`

Anfrage/Beweis



zu beweisen:

2.1) `father(uranus,Z[2]).`

Verfügbare Klausel:

```
father(X,Y):-parent(X,Y),male(X).
```

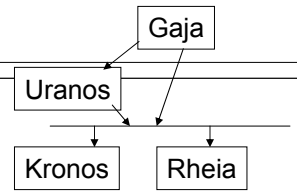
Klausel mit Bindung `X=uranus, Y=Z[2]` :

```
father(uranus,Z[2]):-
                parent(uranus,Z[2]),male(uranus).
```

zu beweisen 2.1.1) `parent(uranus, Z[2]).`

zu beweisen 2.1.2) `male(uranus).`

Anfrage/Beweis



zu beweisen:

2.1.1) `parent(uranus,Z[2]).`

verfügbarer Fakt (z.B.), bewirkt Bindung `Z[2]=cronus` :

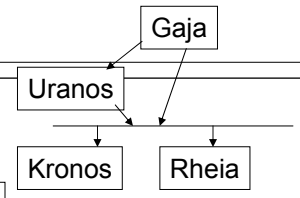
```
parent(uranus,cronus).
```

2.1.2) `male(uranus).`

verfügbarer Fakt:

```
male(uranus).
```

Anfrage/Beweis



zu beweisen

mit erfolgter Bindung $Z_{[2]}$=cronus

2.2) `mother(gaea,cronus).`

Verfügbare Klausel:

```
mother(X,Y):-parent(X,Y),female(X).
```

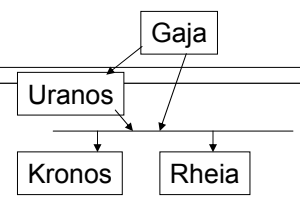
Klausel mit Bindung `X=gaea, Y=cronus` :

```
mother(gaea,cronus):-
            parent(gaea,cronus),female(gaea).
```

zu beweisen 2.2.1) `parent(gaea,cronus),`

zu beweisen 2.2.2) `female(gaea).`

Anfrage/Beweis



zu beweisen:

2.2.1) `parent(gaea,cronus).`

verfügbarer Fakt:

`parent(gaea,cronus).`

2.2.2) `female(gaea).`

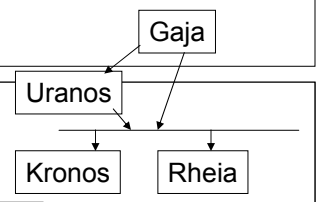
verfügbarer Fakt:

`female(gaea).`

Fertig

Anfrage/Beweis-**Baum**

```
?- mother_in_law(gaea,gaea).
Yes.
```




```
mother_in_law(gaea,gaea)
mother(gaea,uranus)
parent(uranus,gaea,cronus
parent(gaea,uranus)
father(uranus,cronus)
female(gaea)
mother(gaea,cronus)
parent(uranus,cronus)
parent(gaea,cronus)
male(uranus)
female(gaea)
```


(existenzielle) Anfragen mit Regeln

```
?- father(zeus,athena).
yes.
?- father(zeus,X).
X=hermes?
;
X=athena?
;
X=ares?
.
yes.
```

```
rent(uranus, cronus).
rent(gaea,   cronus).
rent(gaea,   rhea).
rent(rhea,   zeus).
rent(cronus, zeus).
rent(rhea,   hera).
rent(cronus, hera).
rent(cronus, hades).
rent(rhea,   hades).
rent(cronus, hestia
rent(rhea,   hestia
rent(zeus,   hermes
rent(maia,   hermes
...
```

```
female(gaea).
female(rhea).
female(hera).
female(hestia).
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
```

```
father(Vater,Kind)
      :-parent(Vater,Kind), male(Vater).
```

Prolog-Programm

- Programme bestehen aus Klauseln.

```
father(X,Y):-parent(X,Y),male(X).
mother(X,Y):-parent(X,Y),female(X).
```

```
parent(X,Y,Z) :-father(X,Z),mother(Y,Z).
```

```
son(X,Y):-parent(X,Y),male(Y).
```

```
grandfather(X,Z):-father(X,Y),parent(Y,Z).
grandmother(X,Z):-mother(X,Y),parent(Y,Z).
```

```
grandchild(X,Y):-grandfather(Y,X).
grandchild(X,Y):-grandmother(Y,Z).
```

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
parent(rhea,   hades).
parent(cronus, hestia
parent(rhea,   hestia
parent(zeus,   hermes
parent(maia,   hermes
...
```

```
female(gaea).
female(rhea).
female(hera).
female(hestia).
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

- Klauseln sind Fakten oder Regeln.
- Fakt als Regel: `fakt:-true`

  mit Prädikat true/0

Prolog-Programm

- Klauseln definieren Relationen (Prädikate)

```
father(Vater,Kind)
        :-parent(Vater,Kind), male(Vater).
```

- Intuitive Bedeutung:

  Linke Seite gilt, wenn alle Prädikate der rechten Seite bei entsprechender Belegung/Bindung der Variablen gelten.

Es gilt: `father(zeus,athena).`

weil gilt: `parent(zeus,athena).`

`male(zeus).`

Prolog-Programm: Prozeduren

- Klauseln mit gleichem Kopf-Funktor (Name,Stelligkeit) bilden eine **Prozedur**
- Die Klauseln einer Prozedur bieten Alternativen für die Prolog-Beweise

```
grandfather(X,Z):-father(X,Y),father(Y,Z).
grandfather(X,Z):-father(X,Y),mother(Y,Z).
```

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
...
```


Prolog-Programm: Prozeduren

- Klauseln mit gleichem Kopf-Funktor (Name,Stelligkeit) bilden eine **Prozedur**
- Redundanzen sind **logisch** unproblemastisch

```
grandfather(X,Z):-father(X,Y),father(Y,Z).
grandfather(X,Z):-father(X,Y),mother(Y,Z).
grandfather(X,Z):-father(X,Y),parent(Y,Z).
grandfather(cronus, ares).
grandfather(cronus, athena).
```

- Redundanzen bieten zusätzliche Beweisvarianten
- PROLOG: evtl. unerwünschte Auswirkungen
- Softwaretechnologie: Minimalitätsprinzip

Prolog-Programm

Anfragen an Programme haben die Form

$$? - goal(X_1,...,X_n).$$

im Sinne von „gilt ...?“ („ist ... beweisbar?“):

$$\exists X_1..\exists X_k\ [\ goal(X_1,...,X_n)\ ]$$

Oder allgemeiner

$$? - goal_1(X_1,...,X_n)\ ,...,\ goal_m(X_1,...,X_n).$$

im Sinne von „gilt ...?“ („ist ... beweisbar?“):

$$\exists X_1..\exists X_k\ [goal_1(X_1,...,X_n) \wedge ... \wedge goal_m(X_1,...,X_n)\ ]$$

Prolog-Interpreter

Laufzeit-System für Prolog,
das Antworten auf Anfragen an ein Programm gibt,
d.h. Beweise sucht und ausführt (Theorem-Beweiser).

Vorstellung:
Man gibt Fakten und Zusammenhänge als Programm ein
(z.B. Fahrplantabelle)
und erhält Antworten bzgl. aller Folgerungen
(z.B. billigste Verbindungen von A nach B) .

Prolog-Interpreter

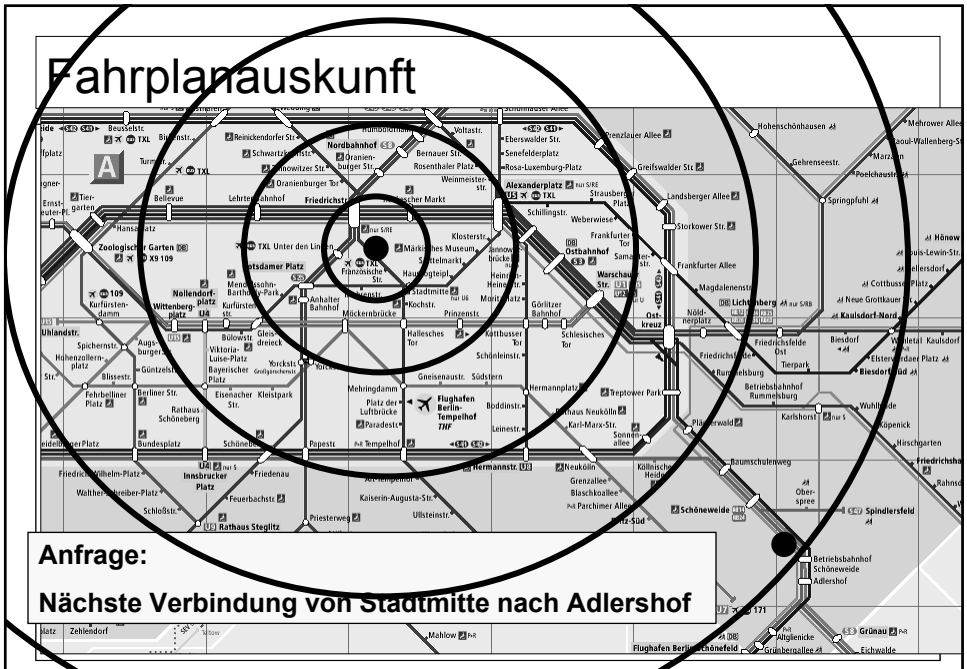
Fahrplan-Fakten

```
s_bahn(alexanderplatz,jannowitzbrücke,6:09,6:11, 103).
s_bahn(jannowitzbrücke,ostbahnhof,    6:11,6:13, 103).
...
```

Suchprogramm

```
erreichbar(Start,Ziel,Zeit)
:- s_bahn(Start,Zwischenziel,Abfahrt,Ankunft,_),
   erreichbar(Zwischenziel,Ziel,Zeit1),
   addiereZeit(Zeit1,Ankunft,Abfahrt,Zeit).
...
```


Fahrplanauskunft



Anfrage:

Nächste Verbindung von Stadtmitte nach Adlershof

Fahrplanauskunft

Problem:
Was genau möchte der Kunde?

Nächste Verbindung?

Kürzeste Verbindung?

Billigste Verbindung?

Wenig Umsteigen?

Prolog und Logik

- Prolog-Programm P
  = Axiome
- Anfrage Q
  = Frage nach Beweisbarkeit von Q aus P
- Antwort
  = Ergebnis eines Beweises bzw. Fehlschlag
- Trace
  = Verlauf des Beweisversuchs

Prolog-Interpreter

Ziel:

Prolog-Interpreter als universelles Verfahren im PK1

Insbesondere

- Vollständigkeit:

  Interpreter liefert alle Folgerungen aus dem Programm

- Korrektheit:

  Interpreter liefert nur Folgerungen aus dem Programm

Situation im PK1

Q folgt aus Formelmenge $\{P_1,..,P_n\}$

gdw. Q ist aus Formelmenge $\{P_1,..,P_n\}$ syntaktisch ableitbar

gdw. $P_1 \wedge ... \wedge P_n \rightarrow Q$ ist allgemeingültig

Allgemeingültigkeit ist <u>axiomatisierbar/aufzählbar</u>:

Falls ein Ausdruck H allgemeingültig ist,
so ist das in endlich vielen Schritten feststellbar.

Genauer: Es gibt dafür ein universelles Verfahren.

Algorithmus/Programm

„Theorembeweiser“

Situation im PK1

Q folgt aus Formelmenge $\{P_1,..,P_n\}$

gdw. Q ist aus Formelmenge $\{P_1,..,P_n\}$ syntaktisch ableitbar

gdw. $P_1 \wedge ... \wedge P_n \rightarrow Q$ ist allgemeingültig

Allgemeingültigkeit ist <u>nicht entscheidbar</u>:

Es gibt **kein universelles** Verfahren, das für beliebige H **entscheidet**, ob H allgemeingültig ist.

Falls ein Ausdruck H **nicht allgemeingültig** ist,
so ist das eventuell nicht feststellbar
(Verfahren kommt evtl. nicht zum Abbruch).

Genauer: Es gibt dafür **kein** universelles Verfahren.

Situation im PK1

Q folgt aus Formelmenge $\{P_1,..,P_n\}$

gdw. Q ist aus Formelmenge $\{P_1,..,P_n\}$ syntaktisch ableitbar

gdw. $P_1 \wedge ... \wedge P_n \rightarrow Q$ ist allgemeingültig

Falls ein Ausdruck H **allgemeingültig** ist,
so ist das in endlich vielen Schritten feststellbar.

Genauer: Es gibt dafür ein **universelles Verfahren**.

Falls ein Ausdruck H **nicht allgemeingültig** ist,
so ist das nicht allgemein feststellbar.

Genauer: Es gibt dafür **kein universelles Verfahren**.

Konsequenzen aus Situation im PK1

- Kein Entscheidungsprogramm im PK1 möglich.

- Spezielle Situation für Prolog-Interpreter:
  - eingeschränkt durch Horn-Klauseln
  - eingeschränkt durch Beweisstrategie im Interpreter
    (Frage nach Vollständigkeit/Korrektheit!)

- Entscheidungsverfahren existieren prinzipiell für
  - aussagenlogische Programme oder
  - Programme über endlichen Relationen