# IV.1 Grundkonzepte der logischen Programmierung

Mittwoch, 25. Januar 2017 09:30

Imperat + Fkt. Prog. beruht auf "Funktionen": Bekannten (Methoden)

mehrere Eingabeparameter, liefern Ergebnis

Logikprogrammierung beruht stattdessen auf Relationen (berechnen keine Ergebnisse, sondern sind Aussagen, die wahr oder falsch sind)

↑ ≙ boolesche Funktionen

Alle Prog-sprachen sind "gleichmächtig", aber unterschiedlich gut f. versch. Zwecke geeignet.

Haupteinsatz von Prolog:

Künstl. Intelligenz

(Expertensysteme, Robotik, ...)

Prolog-Programm ≙ Wissensbasis

Bsp: Verwandtschafts-Datenbasis

Ausführung eines Prolog-Pr:
Programm versucht, Anfragen anhand seiner Wissensbasis zu beantworten.

z.B. "Wer ist die Mutter v. susanne?"
"Welche Kinder hat aline?"
"Wer sind die Vorfahren von aline?"

(Grundidee v. Expertensystemen)

Prolog-Programm: Sammlung logischer Formeln (sog. Klauseln)

Ausführung: Versuche, Anfragen des Benutzers zu beweisen.

"Prolog" = Programming in Logic

Fakten: Name d. Eigenschaft (Relation) ( ... ). ← Objekte mit dieser Eigenschaft

- Eigenschaften sind gerichtet
- Eigenschaften + Objekte sind Strings mit Kleinbuchstaben
- Relationen können versch. Stelligkeit haben.
- Kommentare in Prolog:

  % ... Zeilenende

  /* ... */ für längere Kommentare

- Um Prog. auszuführen stellt man Anfragen.

  ?- ....

  Aussage, die von Prolog bewiesen oder widerlegt werden soll.

- Prolog verwendet Closed World Assumption:

alles was nicht aus dem Programm folgt ist falsch

- Um Prog. auszuführen, muss es vorher geladen werden.

"konsultiert"

- Prog. kann auch Variablen enthalten (Strings, die mit Großbuchst. beginnen.)

Var. im Programm sind allquantifiziert, d.h., sie stehen für bel. Belegungen.

```
?- mensch(7).
true
```

← Prolog ist eine nicht-typisierte Sprache.

Gleiche Variablen in derselben Prog-Klausel müssen gleich belegt werden.

```
mag(X,Y).
```

← Jeder mag jeden.

```
mag(X,X).
```

← Jeder mag sich selbst.

Variablen in Anfragen

nötig, um das Prog. selbst Lösungen berechnen zu lassen.

?- mutterVon(X, susanne). ← Gibt es ein X, das die Mutter von Susanne ist?

Bh: Var. im Prog. sind allquantifiziert.
Var in Anfrage sind existenzquantifiziert.

Falls Beweis gelingt, gibt Prolog entsprechende Antwortsubstitution aus.

?- mV(renate, Y). ← Wer sind die Kinder von renate?

Prolog durchsucht die Klauseln im Programm von oben nach unten und gibt die erste gefundene Antwort aus. Durch Eingabe von ";" kann man Prolog nach weiteren Lösungen suchen lassen.

Bs: Prolog liegt durch das Prog. nicht fest, was Ein- u. Ausgabe ist, sondern dies hängt v. Anfrage ab.
⇒ Das gleiche Prog. kann auf unterschiedl. Weise benutzt werden.

## Kombination von Fragen

# Kombination von Fragen

Lasse mehrere Anfragen zusammen beweisen.

";" ≙ oder  "," ≙ und

Mehrere Literale in einer Klausel werden von links nach rechts abgearbeitet.

?- verh(gerd, F), mV(F, sus)

| F = renate

?- mV(ren, sus)

|

□ ← leere Klausel, d.h. kein Beweisziel mehr übrig, Beweis war erfolgreich

Danach wird die Belegung der Variablen aus dem erfolgreichen Pfad des Beweisbaums ausgegeben (d.h. aus dem Pfad v. Wurzel bis □).

?- mV(O, M), mV(M, aline)

O = monika, M = karin → ?- mV(karin, aline) ↓

O = monika, M = klaus → ?- mV(klaus, aline) ↓

O = renate, M = susanne → ?- mV(sus, aline)

|

□

Wenn ein Beweisversuch scheitert, sagt Prolog zum Vorgänger zurück und versucht die nächste Alternative (Backtracking). Analog bei Eingabe von ";".
Hier wäre es besser, die beiden Literale in der Anfrage zu tauschen.

## Regeln

Prolog-Prog. besteht aus Fakten + Regeln.

Hiermit kann aus bekanntem Wissen neues Wissen hergeleitet werden.

":-" bedeutet "falls"

p :- q, r.

p: Klauselkopf

q, r: Klauselrumpf

"Wenn-dann"-Beziehung: Wenn q und r wahr sind, dann ist auch p wahr.

Regeln werden im Beweis wie folgt angewendet:

Um zu beweisen, dass p gilt,

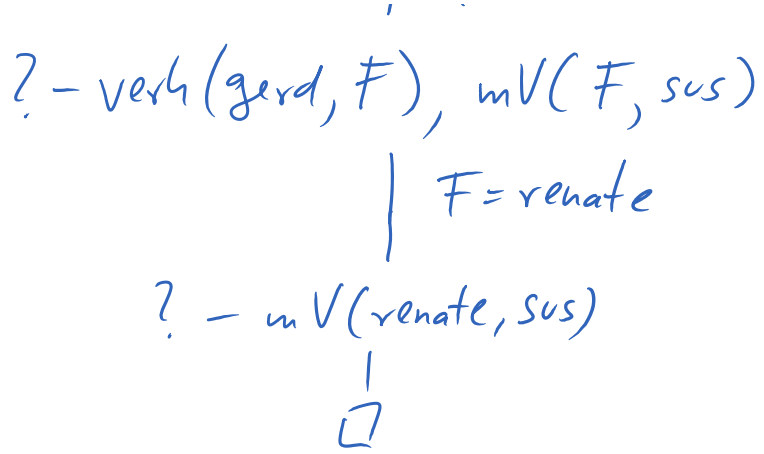
zeigt man stattdessen, dass q und r gelten.

?- vaterVon(gerd, susanne)

| V = gerd
| K = susanne

← es wird nur der

?- verh(gerd, F) mV(F sus)



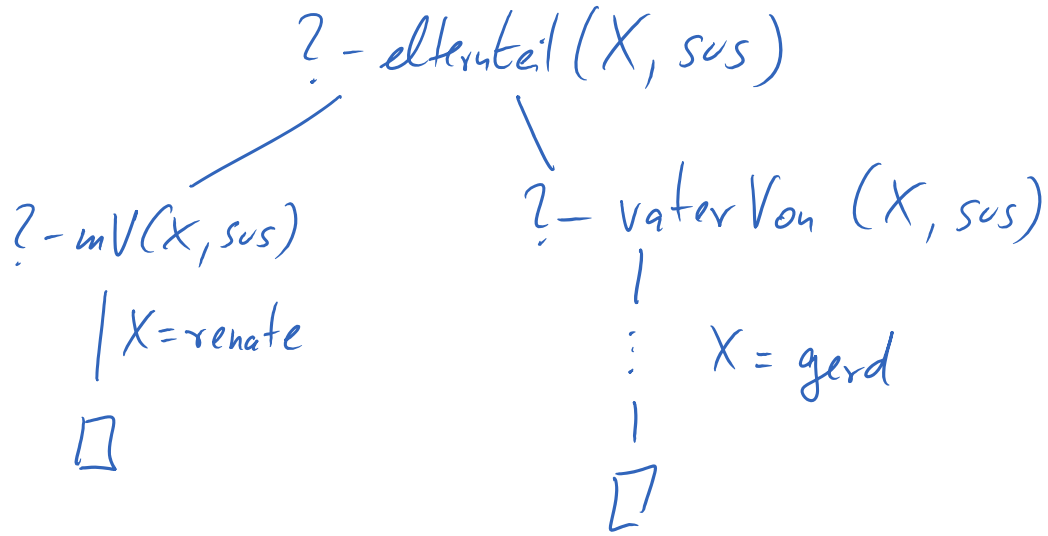

nur der Teil der Variablenbelegung ausgegeben, der Var. aus der urspr. Anfrage betrifft.

## Mehrere Regeln für ein Prädikat

Auf diese Weise kann man Disjunktionen ausdrücken:

```
p :- q.
p :- r.
```

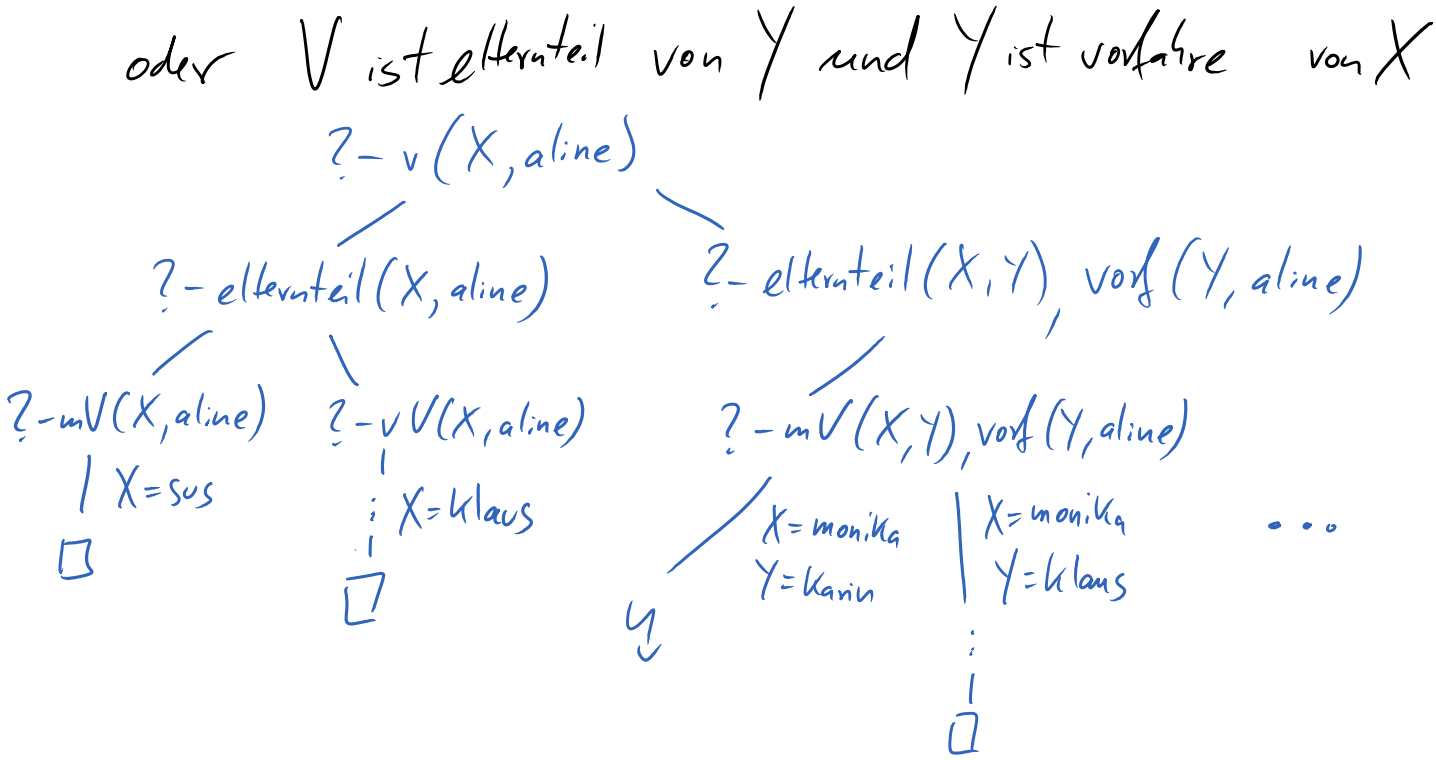
← p gilt, falls q oder r gelten.




## Rekursive Regeln

Ziel: Prädikat "vorfahre"

V ist vorfahre von X, falls V elternteil von X ist


oder V ist elternteil von Y und Y ist vorfahre von X
?- v(X, aline)
?- elternteil(X, aline)
?- elternteil(X, Y), vorf(Y, aline)
?- mV(X, aline)
X = sus
?- vV(X, aline)
X = klaus
?- mV(X, Y), vorf(Y, aline)
X = monika
Y = karin
X = monika
Y = klaus
...