Deskriptive Programmierung

Deklarative Semantik von Prolog

Suche nach deklarativer Bedeutung von Prolog-Programmen

Was ist denn die "mathematische" Bedeutung/Semantik eines Prolog-Programms?

```
add(0,X,X).
add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

Logische Interpretation:

$$(\forall X. \operatorname{add}(0,X,X))$$

 $\land (\forall X, Y, Z. \operatorname{add}(X,Y,Z) \Rightarrow \operatorname{add}(s(X),Y,s(Z)))$

Um solchen Formeln eine Bedeutung zuzuordnen, verwendet man in der Logik Modelle:

- Menge von Objekten
- Interpretation von Funktoren (wie "s(…)") als Funktionen auf Objekten
- Interpretation von Prädikaten (wie "add(…)") als Relationen zwischen Objekten
- Zuweisung von Wahrheitswerten zu Formeln nach festen Regeln
- Betrachtung nur von Interpretationen, die alle gegebenen Formeln wahr machen

Semantik eines Programms wäre gegeben durch alle Zusammenhänge, die in allen Modellen des Programms gelten.

Beispiel-Modelle

```
add(0,X,X).

add(s(X),Y,s(Z)) := add(X,Y,Z).
```

$$\begin{array}{l} (\forall \ X. \ add(0,\!X,\!X)) \\ \wedge \ (\forall \ X, \ Y, \ Z. \ add(X,\!Y,\!Z) \ \Rightarrow add(s(X),\!Y,\!s(Z))) \end{array}$$

Modell 1: Objekte: natürliche Zahlen

Interpretation von 0 als 0

Interpretation von s(...) als s(n) = n + 1

Interpretation von add(...) als add(n,m,k) gdw. n + m = k

Modell 2: Objekte: {*}

Interpretation von 0 als *

Interpretation von s(...) als s(*) = *

Interpretation von add(...) als add(*,*,*) wahr

Modell 3: Objekte: nichtpositive ganze Zahlen

Interpretation von 0 als 0

Interpretation von s(...) als s(n) = n - 1

Interpretation von add(...) als add(n,m,k) gdw. n + m = k

Herbrand-Modelle

Wichtig: Es gibt stets eine Art "Universalmodell".

Idee: Interpretation möglichst einfach, nämlich rein syntaktisch.

Weder Funktoren noch Prädikate "tun" irgendwas. das Herbrand-Universum

Also: Menge von Objekten = alle Grundterme (über gegebener Signatur)

Interpretation von Funktoren = syntaktische Anwendung auf Terme

Interpretation von Prädikaten = irgendeine Menge von Anwendungen von

Prädikatssymbolen auf Grundterme

eine Herbrand-Interpretation

Beispiel:

```
add(0,X,X).

add(s(X),Y,s(Z)) := add(X,Y,Z).
```

Herbrand-Universum: {0, s(0), s(s(0)), s(s(0)), ...} (ohne Prädikatssymbole!) die Herbrand-Basis: {add(0,0,0), add(0,0,s(0)), add(0,s(0),0), ...} (alle Anwendungen von Prädikatssymbolen auf Terme des Herbrand-Universums)

Herbrand-Modelle

Eine Herbrand-Interpretation ist irgendeine Teilmenge der Herbrand-Basis.

Beispiel:

```
add(0,X,X).

add(s(X),Y,s(Z)) := add(X,Y,Z).
```

```
Herbrand-Interpretation 1: {add(0,0,0), add(0,0,s(0)), add(0,s(0),0),...}
```

Herbrand-Interpretation 2: \emptyset

Unser Ziel ist eine Herbrand-Interpretation, die alle durch das Programm gegebenen Formeln wahr macht, aber auch nicht unnötig mehr wahr macht.

Herbrand-Modelle

Eine Herbrand-Interpretation ist ein Modell für ein Programm, wenn für jede vollständige Instanziierung (keine Variablen übrig)

$$\mathbf{L}_0 : - \mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_n$$

jeder Klausel gilt: wenn L_1 , L_2 , ..., L_n in der Interpretation, dann auch L_0 .

Beispiel:

```
\begin{array}{c} \mathbf{add}\,(\mathbf{0}\,,\mathbf{X}\,,\mathbf{X})\;.\\ \mathbf{add}\,(\mathbf{s}\,(\mathbf{X})\,,\mathbf{Y}\,,\mathbf{s}\,(\mathbf{Z})\,)\;:-\;\mathbf{add}\,(\mathbf{X}\,,\mathbf{Y}\,,\mathbf{Z})\;. \end{array} \\ (\forall\;X.\;\mathrm{add}\,(\mathbf{0}\,,\!X\,,\!X))\\ \wedge\;(\forall\;X,\,Y,\,Z.\;\mathrm{add}\,(X,\,Y,\!Z)\;\Rightarrow\mathrm{add}\,(\mathbf{s}\,(X)\,,\!Y,\!\mathbf{s}\,(Z))) \end{array}
```

- Die Herbrand-Basis ist (immer) ein Modell.
- Die Herbrand-Interpretation $\emptyset = \{\}$ ist (hier) kein Modell.
- Die Interpretation {add(0,0,0), add(0,s(0),s(0)), add(s(0),0,s(0)), add(s(0),s(0),s(0)),...} ist hier ein Modell.

Kleinstes Herbrand-Modell

Die deklarative Bedeutung eines Programms ist seine kleinste Herbrand-Interpretation, die ein Modell ist!

Für das Beispiel:

```
add(0,X,X).
add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

```
{add(0,0,0),add(0,s(0),s(0)),add(s(0),0,s(0)),add(s(0),s(0),s(0)),...}
```

Allgemein:

Gibt es immer so ein kleinstes Modell?

Ja, weil (Herbrand)-Modelle, für Programme bestehend aus sogenannten Horn-Klauseln (genau die in <u>Prolog ohne Negation</u> vorkommenden), unter Durchschnitt abgeschlossen sind!

Kleinstes Herbrand-Modell

Kann man das kleinste Herbrand-Modell (mathematisch konstruktiv) "ausrechnen"?

Ja, mittels des "Immediate Consequence Operators": T_P

Definition: T_P nimmt eine Interpretation I und erzeugt alle Grundliterale (Elemente der Herbrand-Basis) L_0 , für die L_1 , L_2 , ..., L_n in I existieren, so dass $L_0 := L_1$, L_2 , ..., L_n eine vollständige Instanziierung irgendeiner der gegebenen Programm-Klauseln ist.

Offenbar: Eine Herbrand-Interpretation I ist ein Modell gdw. T_P(I) Teilmenge von I.

Außerdem: Das kleinste Herbrand-Modell ergibt sich als Fixpunkt/Limit der Folge

$$\emptyset$$
, $T_p(\emptyset)$, $T_p(T_p(\emptyset))$, $T_p(T_p(T_p(\emptyset)))$, ...

Kleinstes Herbrand-Modell

Am Beispiel:

```
add(0,X,X).

add(s(X),Y,s(Z)) := add(X,Y,Z).
```

```
\begin{split} T_p(\varnothing) &= \{ \text{add}(0,0,0), \text{add}(0,s(0),s(0)), \text{add}(0,s(s(0)),s(s(0))), \ldots \} \\ T_p(T_p(\varnothing)) &= T_p(\varnothing) \cup \{ \text{add}(s(0),0,s(0)), \text{add}(s(0),s(0)),s(s(0))), \\ &= \text{add}(s(0),s(s(0)),s(s(s(0)))), \ldots \} \end{split} T_p(T_p(T_p(\varnothing))) &= T_p(T_p(\varnothing)) \cup \{ \text{add}(s(s(0)),0,s(s(0))), \\ &= \text{add}(s(s(0)),s(0),s(s(s(0)))), \\ &= \text{add}(s(s(0)),s(s(0)),s(s(s(0)))), \ldots \} \end{split}
```

. . .

Verwendbarkeit der Semantik über Herbrand-Modelle

Für welche Art von Programmen kann man mit der T_P –Semantik arbeiten?

- keine Arithmetik, kein is
- kein \=, kein not
- allgemein, keine der (noch einzuführenden) "nicht-logischen" Features

Aber eben zum Beispiel Programme wie:

```
add(0,X,X).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,_,0).
mult(s(_),0,0).
mult(s(X),s(Y),s(Z)) :- mult(X,s(Y),U), add(Y,U,Z).
```

```
\begin{split} T_P(\varnothing) &= \{ \text{add}(0,0,0), \text{add}(0,s(0),s(0)), \ldots \} \cup \{ \text{mult}(0,0,0), \\ &\quad \text{mult}(0,s(0),0), \ldots \} \cup \{ \text{mult}(s(0),0,0), \ldots \} \\ &\quad T_P(T_P(\varnothing)) &= T_P(\varnothing) \cup \{ \text{add}(s(0),0,s(0)), \text{add}(s(0),s(0),s(0)), s(0)), \ldots \} \\ &\quad \cup \{ \text{mult}(s(0),s(0),s(0)) \} \end{split}
```

Verwendbarkeit der Semantik über Herbrand-Modelle

add(s(X),Y,s(Z)) := add(X,Y,Z).

add(0,X,X).

```
mult(0, ,0).
             mult(s(),0,0).
             mult(s(X), s(Y), s(Z)) := mult(X, s(Y), U), add(Y, U, Z).
T_p(\emptyset) = \{ add(0,0,0), add(0,s(0),s(0)), ... \} \cup \{ mult(0,0,0), ... \} 
             mult(0,s(0),0),... \cup \{mult(s(0),0,0),...\}
T_p(T_p(\emptyset)) = T_p(\emptyset) \cup \{add(s(0), 0, s(0)), add(s(0), s(0), s(0)), \dots\}
               \cup \{ \text{mult}(s(0), s(0), s(0)) \}
T_p(T_p(T_p(\emptyset))) = T_p(T_p(\emptyset)) \cup \{add(s(s(0)), 0, s(s(0))), \ldots\}
                   \cup {mult(s(0),s(s(0)),s(s(0))).
```

 \cup {mult(s(0),s³(0),s³(0)), mult(s²(0),s²(0),s⁴(0)),

mult(s(s(0)), s(0), s(s(0)))

 $T_p^4(\emptyset) = T_p^3(\emptyset) \cup \{add(s^3(0), 0, s^3(0)), add(s^3(0), s(0), s^4(0)), ...\}$

 $mult(s^3(0), s(0), s^3(0))$

Verwendbarkeit der Semantik über Herbrand-Modelle

Die deklarative Semantik:

- ist nur auf bestimmte, "rein logische", Programme anwendbar
- beschreibt nicht direkt das Verhalten bei Anfragen mit Variablen
- ist mathematisch einfacher als die noch einzuführende operationelle Semantik
- lässt sich geeignet zur operationellen Semantik in Beziehung setzen
- ist insensitiv gegenüber Änderungen der Reihenfolge von und innerhalb Regeln (!)

Operationalisierung?

Spezifikation (,,Programm") ≡ Relationsdefinitionen

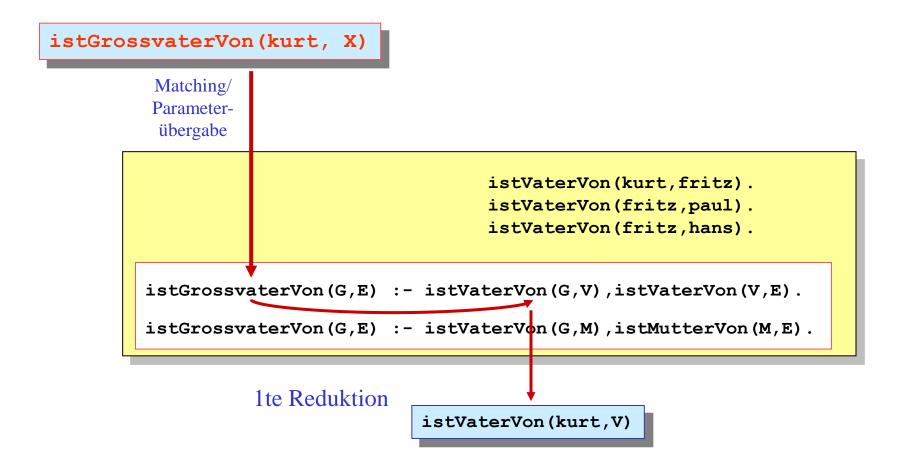
```
istVaterVon(kurt,fritz).
istVaterVon(fritz,paul).
istVaterVon(fritz,hans).

istGrossvaterVon(G,E) :-
    istVaterVon(G,V),istVaterVon(V,E).

istGrossvaterVon(G,E) :-
    istVaterVon(G,M),istMutterVon(M,E).
```

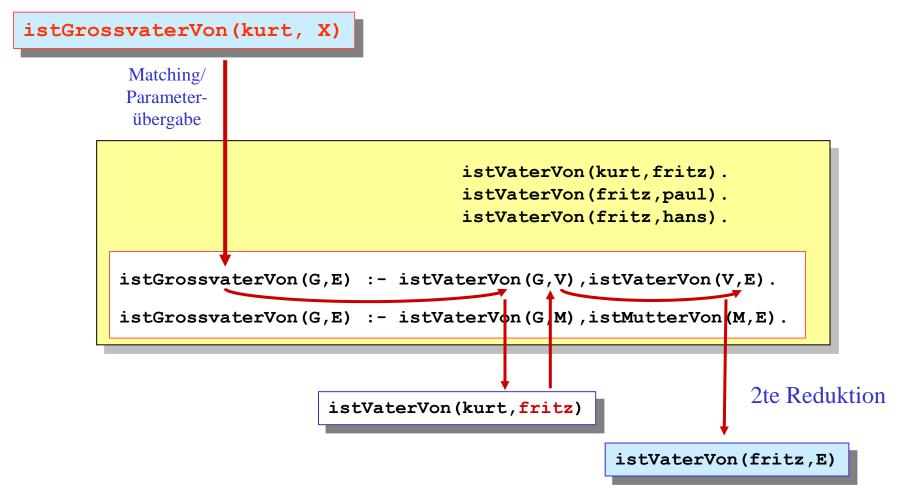
Operationalisierung in Prolog (1)

Operationalisierungsprinzip: Rückführung auf ein Unterproblem (Reduktion)



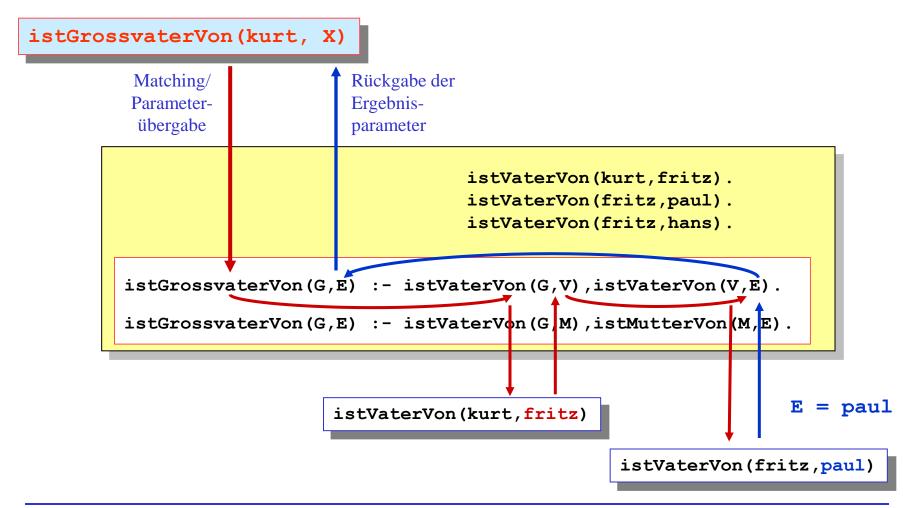
Operationalisierung in Prolog (2)

Operationalisierungsprinzip: Rückführung auf ein Unterproblem, wobei neue Unteranfragen von <u>links nach rechts</u> gefunden werden!



Operationalisierung in Prolog (3)

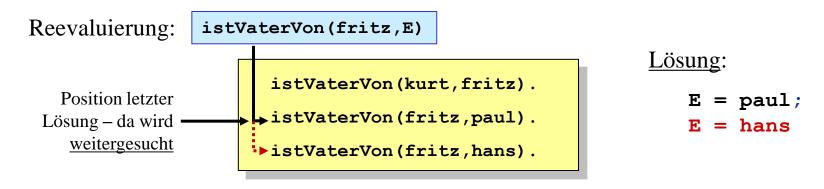
Operationalisierungsprinzip: Rückführung auf ein Unterproblem (Reduktion)



Operationalisierung in Prolog (4)

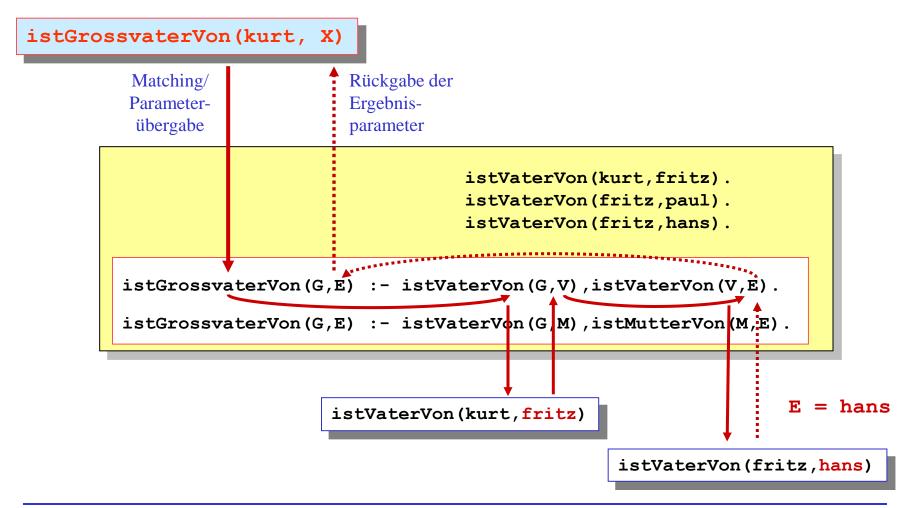
• Prolog sucht nach matchenden Regeln oder Fakten immer von <u>oben nach unten</u> im Programm.

• Da eine Relation keine eindeutige Abbildung ist, können weitere Antworten für eine (Unter-)Anfrage existieren. Prolog findet diese mittels Backtracking:



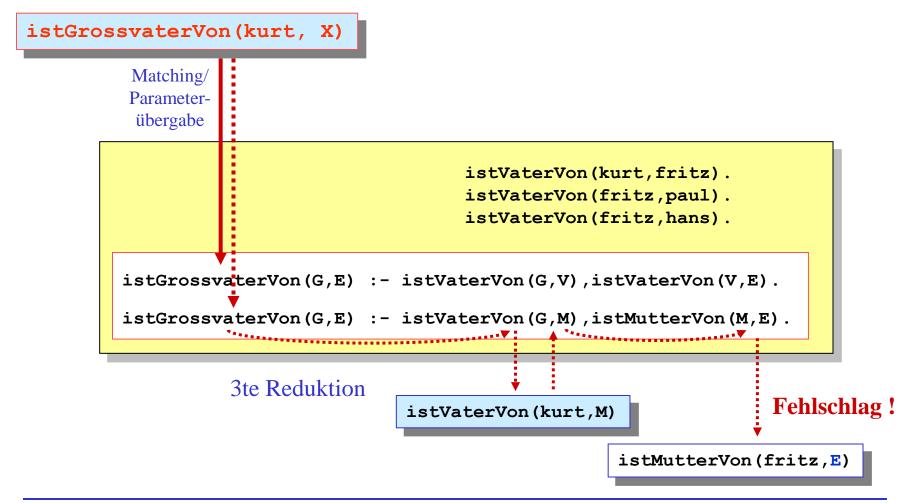
Operationalisierung in Prolog (5)

Operationalisierungsprinzip: Rückführung auf ein Unterproblem (Reduktion)



Operationalisierung in Prolog (6)

Das Backtracking bezieht sich auch auf andere "matchende" Regeln:



Operationalisierung am Beispiel nochmal anders dargestellt

```
istVaterVon(kurt,fritz).
istVaterVon(fritz,paul).
istVaterVon(fritz,hans).

istGrossvaterVon(G,E) :-
    istVaterVon(G,V),istVaterVon(V,E).

istGrossvaterVon(G,E) :-
    istVaterVon(G,M),istMutterVon(M,E).
```

```
?- istGrossvaterVon(kurt, X).
?- istVaterVon(kurt, V), istVaterVon(V, X).
?- istVaterVon(fritz, X).
?- .
```

Vergleiche (innerhalb eines Prolog-Systems): Benutzung von ?- trace.

Operationalisierung am Beispiel nochmal anders dargestellt

```
istVaterVon(kurt,fritz).
istVaterVon(fritz,paul).
istVaterVon(fritz,hans).

istGrossvaterVon(G,E) :-
    istVaterVon(G,V),istVaterVon(V,E).
istGrossvaterVon(G,E) :-
    istVaterVon(G,M),istMutterVon(M,E).
```

```
?- istGrossvaterVon(kurt, X).
?- istVaterVon(kurt, V), istVaterVon(V, X).
?- istVaterVon(fritz, X).
?- .
X = paul:
X = hans:
?- .
```

Vergleiche (innerhalb eines Prolog-Systems): Benutzung von ?- trace.

Operationalisierung am Beispiel nochmal anders dargestellt

```
istVaterVon(kurt,fritz).
istVaterVon(fritz,paul).
istVaterVon(fritz,hans).

istGrossvaterVon(G,E) :-
    istVaterVon(G,V),istVaterVon(V,E).
istGrossvaterVon(G,E) :-
    istVaterVon(G,M),istMutterVon(M,E).
```

```
?- istGrossvaterVon(kurt, X).
?- istVaterVon(kurt, V), istVaterVon(V, X).
?- istVaterVon(fritz, X).

X = paul:
X = hans:
?- .
?- istVaterVon(kurt, M), istMutterVon(M, X).
?- istMutterVon(fritz, X).
Fehlschlag!
```

Vergleiche (innerhalb eines Prolog-Systems): Benutzung von ?- trace.