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:

$$
\begin{aligned}
& (\forall\ X.\ \mathrm{add}(0,X,X)) \\
\wedge\ & (\forall\ X, Y, Z.\ \mathrm{add}(X,Y,Z) \ \Rightarrow\ \mathrm{add}(s(X),Y,s(Z)))
\end{aligned}
$$

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).
```

$$(\forall\ X.\ add(0,X,X))$$
$$\wedge\ (\forall\ X, Y, Z.\ add(X,Y,Z)\ \Rightarrow add(s(X),Y,s(Z)))$$

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(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: $\emptyset$

Herbrand-Interpretation 3: {`add(0,0,0)`, `add(0,s(0),s(0))`,
`add(s(0),0,s(0))`, `add(s(0),s(0),s(s(0)))`, ...}

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)

$$L_0 \texttt{ :- } L_1, L_2, \ldots, L_n$$

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

Beispiel:

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

$$(\forall\, X.\ \mathrm{add}(0,X,X))$$
$$\wedge\ (\forall\, X, Y, Z.\ \mathrm{add}(X,Y,Z) \Rightarrow \mathrm{add}(s(X),Y,s(Z)))$$

- Die Herbrand-Basis ist (immer) ein Modell.
- Die Herbrand-Interpretation $\varnothing = \{\}$ 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(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(s(0)))`, ...}

Allgemein:

Gibt es immer so ein kleinstes Modell?

Ja, weil (Herbrand)-Modelle, für Programme bestehend aus sogenannten Horn-Klauseln (genau die in Prolog ohne Negation 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

$$\emptyset\ ,\ T_P(\emptyset)\ ,\ T_P(T_P(\emptyset))\ ,\ T_P(T_P(T_P(\emptyset)))\ ,\ \ldots$$

Kleinstes Herbrand-Modell

Am Beispiel:

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

$T_P(\emptyset) \;=\; \{$**add(0,0,0)**, **add(0,s(0),s(0))**, **add(0,s(s(0)),s(s(0)))**, $\ldots\}$

$T_P(T_P(\emptyset)) \;=\; T_P(\emptyset) \cup \{$**add(s(0),0,s(0))**, **add(s(0),s(0),s(s(0)))**,
add(s(0),s(s(0)),s(s(s(0)))), $\ldots\}$

$T_P(T_P(T_P(\emptyset))) \;=\; T_P(T_P(\emptyset)) \cup \{$**add(s(s(0)),0,s(s(0)))**,
add(s(s(0)),s(0),s(s(s(0)))),
add(s(s(0)),s(s(0)),s(s(s(s(0))))), $\ldots\}$

…

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).
```

$$T_P(\emptyset) = \{\texttt{add(0,0,0)}, \texttt{add(0,s(0),s(0))}, \ldots\} \cup \{\texttt{mult(0,0,0)}, \texttt{mult(0,s(0),0)}, \ldots\} \cup \{\texttt{mult(s(0),0,0)}, \ldots\}$$

$$T_P(T_P(\emptyset)) = T_P(\emptyset) \cup \{\texttt{add(s(0),0,s(0))}, \texttt{add(s(0),s(0),s(s(0)))}, \ldots\} \cup \{\texttt{mult(s(0),s(0),s(0))}\}$$

Verwendbarkeit der Semantik über Herbrand-Modelle

```
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).
```

$$T_P(\emptyset) = \{\mathtt{add(0,0,0)}, \mathtt{add(0,s(0),s(0))}, \ldots\} \cup \{\mathtt{mult(0,0,0)}, \mathtt{mult(0,s(0),0)}, \ldots\} \cup \{\mathtt{mult(s(0),0,0)}, \ldots\}$$

$$T_P(T_P(\emptyset)) = T_P(\emptyset) \cup \{\mathtt{add(s(0),0,s(0))}, \mathtt{add(s(0),s(0),s(s(0)))}, \ldots\} \cup \{\mathtt{mult(s(0),s(0),s(0))}\}$$

$$T_P(T_P(T_P(\emptyset))) = T_P(T_P(\emptyset)) \cup \{\mathtt{add(s(s(0)),0,s(s(0)))}, \ldots\} \cup \{\mathtt{mult(s(0),s(s(0)),s(s(0)))}, \mathtt{mult(s(s(0)),s(0),s(s(0)))}\}$$

$$T_P{}^4(\emptyset) = T_P{}^3(\emptyset) \cup \{\mathtt{add(s^3(0),0,s^3(0))}, \mathtt{add(s^3(0),s(0),s^4(0))}, \ldots\} \cup \{\mathtt{mult(s(0),s^3(0),s^3(0))}, \mathtt{mult(s^2(0),s^2(0),s^4(0))}, \mathtt{mult(s^3(0),s(0),s^3(0))}\}$$

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).
```

```
?- istGrossvaterVon(kurt,X)
  ⇝ …
  ⇝ …
  ⇝ …
  ⇝ …
  ⇝ X = paul ;  X = hans
```

Eingabe: eine Anfrage

⬇

(wiederholte) Reduktion

⬇

Ausgabe: Variablensubstitution(en)

Operationalisierung in Prolog (1)

Operationalisierungsprinzip: Rückführung auf ein Unterproblem (Reduktion)

`istGrossvaterVon(kurt, X)`

Matching/
Parameter-
übergabe

```
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).
```

1te Reduktion

`istVaterVon(kurt,V)`

Operationalisierung in Prolog (2)

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

```
istGrossvaterVon(kurt, X)
```

Matching/
Parameter-
übergabe

```
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).
```

```
istVaterVon(kurt,fritz)
```

2te Reduktion

```
istVaterVon(fritz,E)
```

Operationalisierung in Prolog (3)

Operationalisierungsprinzip: Rückführung auf ein Unterproblem (Reduktion)

```
istGrossvaterVon(kurt, X)
```

Matching/ Parameter-übergabe

Rückgabe der Ergebnis-parameter

```
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).
```

```
istVaterVon(kurt,fritz)
```

```
E = paul
```

```
istVaterVon(fritz,paul)
```

Operationalisierung in Prolog (4)

- Prolog sucht nach matchenden Regeln oder Fakten immer von oben nach unten im Programm.

Unteranfrage: `istVaterVon(fritz,E)`

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

Lösung:

`E = paul`

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

Reevaluierung: `istVaterVon(fritz,E)`

Position letzter Lösung – da wird weitergesucht

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

Lösung:

```
E = paul;
E = hans
```

Operationalisierung in Prolog (5)

Operationalisierungsprinzip: Rückführung auf ein Unterproblem (Reduktion)

```
istGrossvaterVon(kurt, X)
```

Matching/ Parameter- übergabe

Rückgabe der Ergebnis- parameter

```
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).
```

```
istVaterVon(kurt,fritz)
```

```
E = hans
```

```
istVaterVon(fritz,hans)
```

Operationalisierung in Prolog (6)

Das Backtracking bezieht sich auch auf andere „matchende“ Regeln:

```
istGrossvaterVon(kurt, X)
```

Matching/ Parameter- übergabe

```
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).
```

3te Reduktion

```
istVaterVon(kurt,M)
```

Fehlschlag !

```
istMutterVon(fritz,E)
```

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).
```

X = paul:

?- 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.