IV.2 Syntax von Prolog

Montag, 14. Januar 2019 10:30

- Programm: Folge von logischen Formeln (so-genannte Klauseln). Zeilenumbruch oder Leerzeichen nach jeder Klausel.
- 2 Arten von Klauseln: Fakten + Regeln
- Literal:

  Eigenschaft(... Objekte ...)

  ↑ Prädikats-symbol ↑ Terme
- Prädikatssymbole ≙ Relationen bzw. boolesche Funktionen

  ⇒ Literale sind Aus-sagen (können wahr oder

falsch sein)

- Prädikatssymbole haben Stelligkeit, wird oft mit Schrägstrich angegeben:

  weiblich/1

  verheiratet/2

  Prädikate können überladen werden. D.h., es könnte z.B. auch ein Prädikat

  verheiratet/1

  geben. Dies hätte nichts mit

  verheiratet/2

  zu tun.

- Syntax für Prädikate: Strings, die mit Kleinbuchstaben beginnen (und vordef. Prädikate wie =, >, ...)

- Literale: Aussagen
  Terme: Objekte
- Variablen: Strings, die mit Großbuchstaben oder mit _ beginnen.
- Anonyme Variable _ steht für ein beliebiges Objekt, das nicht interessiert (d.h.: es wird in der Antwort selbst nicht ausgegeben, wie _ belegt wird). Mehrfache Vorkommen von _ dürfen unterschiedlich belegt werden:

```
istEhemann(P) :- verh(P, _).

?- istEhemann(klaus).
true
```

?- verheiratet(X, X).
false
?- verheiratet(_, _).
true

Analog zum Joker-Pattern _ in Haskell.

- Funktionssymbole dienen zur Konstruktion von Termen (Objekten).
  Prädikatssymbole dienen zur Konstruktion von Literalen (Aussagen).
- Funktionssymbole sind Strings, die mit Kleinbuchstaben anfangen. Sie haben eine Stelligkeit und können überladen werden.

monika/0

werner/0

- Bisher haben wir nur 0-stellige Funktionssymbole (Konstanten) betrachtet.
- Funktionssymbole können auch höhere Stelligkeit haben.

Bsp: Prädikat geboren/4:

geboren(monika, 25, 7, 1972).

Unsinn. Besser wäre es, wenn geboren 2-stellig wäre und eine Relation zwischen Person und Datum ausdrückt.

Dazu benötigt man ein 3-stelliges Funktionssymbol datum/3.

Hiermit kann man den
Term datum(25,7,1972)
bilden.

Bsp:
?- geboren(monika,X)
Bed: Wann ist monika
geboren?

?- geboren(X,datum(_,7,_))
Bed: Wer hat im Juli
Geburtstag?

?- geboren(monika,
datum(_,X,_)).

X=7

- Prolog ist eine unge-
typte Progsprache.
Man benötigt keine
Deklarationen von

Prädikaten, Funktionen und Variablen.

Realisierung von Datenstrukturen in Rolog

Alle Objekte werden als Terme dargestellt d.h. man benötigt geeignete Funktionssymbole, um die Datenstruktur zu repräsentieren (wie in Haskell).

Haskell:

```
data Nats = Zero | Succ Nats
```

↑ ↗

Datenkonstruktoren

werden in Haskell nicht

weiter ausgewertet

In Prolog sind alle Funktionssymbole Datenkonstruktoren, d.h. sie werden alle nicht ausgewertet (Ausnahmen später).
In Prolog werden nur Prädikatssymbole ausgewertet.

Term zero $\hat{=}$ 0

Term succ(zero) $\hat{=}$ 1

⋮

Damit Addition von Zahlen ausgewertet wird, muss sie mit einem Prädikatssymbol berechnet werden (nicht mit Funktionssymbol).

⇒ Um eine n-stellige Funktion zu berechnen, verwende ein (n+1)-stelliges Prädikatssymbol.

add/3

add(X,Y,Z) bedeutet:
"Die Addition von X und Y ergibt Z."

Z.teile sagt:

$X + (Y+1) = Z+1$, falls

$X + Y = Z$

Bsp: Was ist 1+1?
Hierzu erzeugt Prolog einen Beweisbaum:

?- add(s(zero), s(zero), U).

| X = s(zero)
| Y = zero
| U = s(Z)

?- add(s(zero), zero, Z).

In jedem Schritt werden die Variablen der Programklauseln so umbenannt, dass sie verschieden von den bisherigen Variablen sind.

| X' = s(zero)
| Z = s(zero)

⇒ Verwende hier add(X', zero, X').

Antwortsubst. bekommt man, indem man die Substitutionen der Variablen entlang des erfolgreichen Beweispfads "aufsammelt".

Hier interessiert sich man sich nur für die Variable U in der Anfrage:

U = succ(Z)

$Z = succ(zero)$

$\Rightarrow U = succ(succ(zero))$

Der add-Algorithmus kann auch zum Subtrahieren benutzt werden:

Was ist 2 - 1 :

?- add(s(zero), V, s(s(zero))).

| V = s(Y)

?- add(s(zero), Y, s(zero))

| Y = zero \ Y = s(Y')

□ ?- add(s(zero), Y', zero)

Antwortsubst: V = succ(zero) !

:

↓

Bsp: Multiplikation

A: X * 0 = 0

B: $X * (Y+1) = Z$, falls

$X * Y = U$ und $X + U = Z$

Grund: $X * (Y+1)$

$= \underbrace{X * Y}_{U} + X$

mult(X, Y, Z) ≙ Die Multiplikation von X und Y ergibt Z.

Was ist $1 * 1$?

?- mult(s(zero), s(zero), U).

?- mult(s(zero), zero, U'), add(s(zero), U', U).

| U' = zero

?- add(s(zero), zero, U)

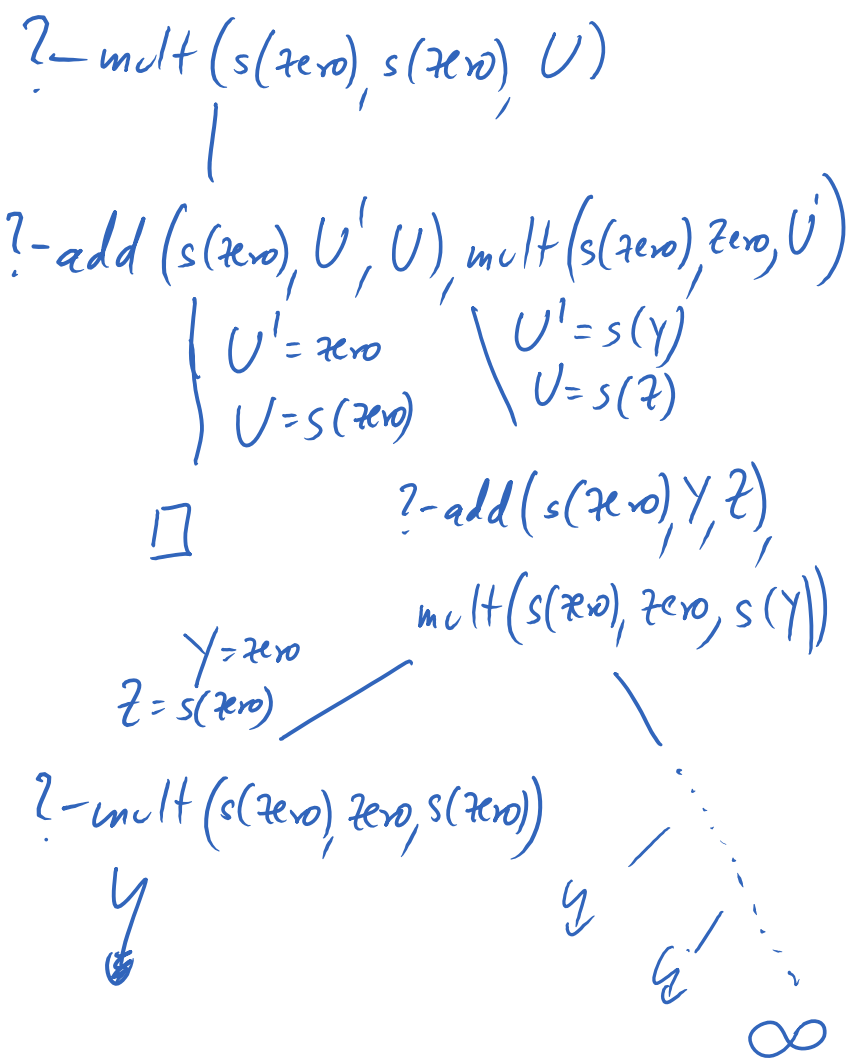
| U = s(zero)

□

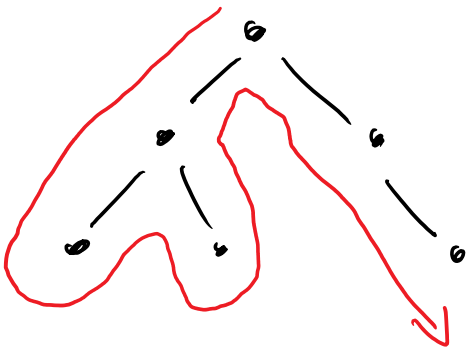
Was passiert, wenn man die Literale in der mult-Regel vertauscht?

d.h.:

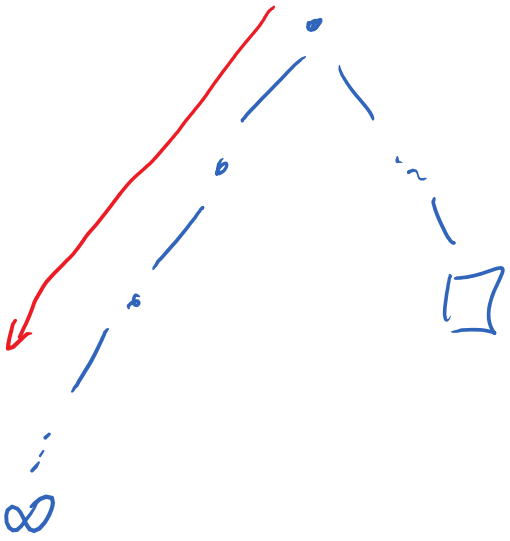
```
mult(X, s(Y), Z) :-
    add(X, U, Z), mult(X, Y, U).
```



Prolog erstellt den Beweisbaum mit Tiefensuche von links nach rechts:



Bsp: Vertausche Zusätzlich die nicht-rekursive und die rekursive add-Klausel.



⇒ Prolog läuft sofort in die Nicht-Terminierung und findet die Lösung nicht, obwohl sie im Beweisbaum enthalten ist.

Generelle Heuristik

- Nicht-rekursive Klauseln sollten vor rekursiven Klauseln des gleichen Prädikats kommen.
- Ordne die Literale im Rumpf der Regel so, dass möglichst viele ihrer Argumente schon belegt sind, wenn das Literal ausgewertet wird.

Bidirektionalität:
add-Prog. kann zum Addieren + Subtrahieren benutzt werden.
mult-Prog kann zum Multiplizieren u. Dividieren benutzt werden.

Prolog ist untypisiert

z.B.: mult kann auf bel. Terme angewendet werden.

```
?- mult(monika, zero, zero).
true

?- mult(succ(monika),
        succ(zero),
        succ(monika)).
true
```

Es gibt auch getypte Logikprogrammiersprachen, d.h: Typisierung hat nichts mit der Logikprogrammierung an sich zu tun.

Datenstruktur für Listen

Haskell:

```
data List a =
        Nil | Cons a (List a)
```

Analog dazu in Prolog:

```
cons(zero, cons(succ(zero), nil))
```

repräsentiert die Liste

[0, 1]

Bsp: Algorithmus zur Berechnung der Listenlänge

len/2 ist 2-stelliges Prädikat

len(L, N) ≙ "Die Liste L hat die Länge N"

Prolog versucht stets, die allgemeinsten Antworten/Lösungen zu finden.

?- len(L, s(s(zero)))

L = cons(A, A')
A'' = s(zero)

Prog-Klausel
len(cons(A, A'), s(A'')) :-
  len(A', A'').

?- len(A', s(zero))

A' = cons(B, B')
B'' = zero

Prog-Klausel
len(cons(B, B'), s(B'')) :-
  len(B', B'').

?- len(B', zero)

B' = nil

□

Antwortsubstitution:

$\sigma_1 = \{ L = cons(A, A'), \ldots \}$

$\sigma_2 = \{ A' = cons(B, B'), \ldots \}$

$\sigma_3 = \{B' = nil\}$

Antwortsubst. ist

Komposition $\sigma_3 \circ \sigma_2 \circ \sigma_1$

erst σ_1, dann σ_2, dann σ_3

Es ist nur interessant, was die Antwortsubst. auf den Variablen der ursprünglichen Anfrage tut (d.h. auf L).

$\sigma = \{L = cons(A, cons(B, nil))\}$

Die Namen der Variablen A, B werden automatisch erzeugt, d.h., sie können auch anders heißen, z.B. _192.

Da Datenstrukturen für

Zahlen + Listen oft benötigt werden, sind sie in Prolog vordefiniert (genauer: bestimmte Kurzschreibweisen sind vordefiniert).

Zeige zunächst vordef. Listen in Prolog.

Statt nil und cons verwende [] und .

Für

.(1, .(2, .(3, [])))

existiert die Kurzschreibweise

[1, 2, 3].

Außerdem kann man

[t1, ..., tn | l]

für die Liste schreiben, die aus der Liste l entsteht,

indem man vorne die Elemente $t_1, \ldots, t_n$ einfügt.

DH: $[1,2,3]$

$= [1 \mid [2,3]]$

$= [1,2 \mid [3]]$

$= [1,2,3 \mid [\,]]$

$= \cdot(1, \cdot(2, \cdot(3, [\,])))$