IV.3 Rechnen in Prolog

Mittwoch, 1. Februar 2017 09:30

Wie arbeitet Prolog genau?

Subst. σ ↖ sigma

Können 2 Ausdrücke gleich gemacht werden, indem man ihre Variablen geeignet instantiiert? ↑ Unifikation

(Brauchten wir bereits zum Typchecking in Haskell)

Unifikator von

add(s(zero), s(zero), U) und

add(X, s(Y), s(Z))

ist

$\sigma = \{X = s(zero), Y = zero, U = s(Z)\}$

Substitution: bildet nur endlich viele Variablen auf anderen Term als sich selbst ab.

Bsp: $\sigma = \{X = s(zero), Y = zero, U = s(Z)\}$

$\sigma(X) = s(zero)$, $\sigma(Z) = Z$

$\sigma(add(s(zero), s(zero), U)) =$

$add(s(zero), s(zero), s(Z))$

$\sigma(add(X, s(Y), s(Z))) =$

$add(s(zero), s(zero), s(Z))$

Unifikatoren sind i.A. nicht eindeutig. Wir sind am allgemeinsten Unifikator interessiert, denn alle anderen Unifikatoren sind Spezialfälle d. allgemeinsten Unifikators.

MGU = most general unifier

Den 2. Unifikator bekommt man aus dem MGU σ_1 durch:

$\sigma_2 = \{Z = zero\} \circ \sigma_1$

↑ Substitution, die Z durch zero ersetzt.

$\sigma_3 = \{Z = succ(W)\} \circ \sigma_1$

↑ bedeutet: Wende erst σ_1 an und dann $\{Z = succ(W)\}$.

Satz: Wenn s und t unifizierbar sind, dann haben sie auch einen MGU.
Dieser MGU ist eindeutig bis auf Variablenumbenennungen.

Bsp: Was ist der MGU von $p(X)$ und $p(Y)$?

Sowohl $\{X = Y\}$ als auch $\{Y = X\}$ sind MGUs.

Der Unterschied ist unwesentlich.

Prolog-Interpreter benötigt einen Algorithmus zur Berechnung des MGU.

Unifikationsalgorithmus

1. $s = X$, $t = X$ MGU: $\{\}$
2. $s = X$, $t = succ(Y)$
MGU: $\{X = succ(Y)\}$

$s = X$, $t = succ(X)$
nicht unifizierbar
OCCUR FAILURE

4. $s = f(\ldots)$ $t = g(\ldots)$,

z.B. $s = succ(Y)$ $t = zero$

nicht unifizierbar

CLASH FAILURE

$s = f(X, Z, succ(succ(W)))$

$t = f(succ(Y), X, Z)$

$\sigma_1 = MGU(X, succ(Y))$

$= \{X = succ(Y)\}$

Verwende diesen Teil-Unifikator, wenn das 2. Argument betrachtet wird.

$\sigma_2 = MGU(\sigma_1(Z), \sigma_1(X))$

$= MGU(Z, succ(Y))$

$= \{Z = succ(Y)\}$

Verwende $\sigma_2 \circ \sigma_1$, wenn das 3. Argument betrachtet wird.

$\sigma_3 = MGU(\sigma_2(\sigma_1(succ(succ(W)))), \sigma_2(\sigma_1(Z)))$

$= MGU(succ(succ(W)), succ(Y))$

$= \{Y = succ(W)\}$.

Gesamtlösung: $\sigma_3 \circ \sigma_2 \circ \sigma_1 = \{X = s(s(W)), Z = s(s(W)), Y = s(W)\}$

Bsp Unifiziere $f(g(h(X,Z)), Z)$ und $f(g(Y), g(X))$

$\sigma_1 = \{Y = h(X,Z)\}$ unifiziert $g(h(X,Z))$ und $g(Y)$

$\sigma_2 = \{Z = g(X)\}$ unifiziert $\underbrace{\sigma_1(Z)}_{Z}$ und $\underbrace{\sigma_1(g(X))}_{g(X)}$

MGU ist $\sigma_2 \circ \sigma_1 = \{Y = h(X, g(X)), Z = g(X)\}$

Bsp Unifiziere $f(g(X), Y)$ und $f(Y, a)$

$\sigma_1 = \{Y = g(X)\}$ unifiziert $g(X)$ und Y

σ_2 existiert nicht, denn $\underbrace{\sigma_1(Y)}_{g(X)}$ und $\underbrace{\sigma_1(a)}_{a}$

sind nicht unifizierbar. CLASH FAILURE

Unifiziere

Bsp $f(\underline{g(X)}, \underline{Y}, \underline{Y})$ und $f(\underline{Y}, \underline{g(h(Z))}, \underline{g(Z)})$

$\sigma_1 = \{Y = g(X)\}$ unifiziert $g(X)$ und Y

$\sigma_2 = \{X = h(Z)\}$ unif. $\underbrace{\sigma_1(Y)}_{g(X)}$ und $\underbrace{\sigma_1(g(h(Z)))}_{g(h(Z))}$

σ_3 exist. nicht, denn $\underbrace{\sigma_2(\sigma_1(Y))}_{g(h(Z))}$ und $\underbrace{\sigma_2(\sigma_1(g(Z)))}_{g(Z)}$

sind nicht unifizierbar OCCUR FAILURE

Prologs Unifikationsalgorithmus ist nicht ganz "korrekt", denn er verzichtet aus Effizienzgründen auf den Occur Check. D.h: X und $succ(X)$ werden als "unifizierbar" angesehen.

(Führt zu unendl. Termen, tritt in Praxis aber kaum auf.)

Der eigentliche Auswertungs-algorithmus von Prolog ver-

wendet das Beweisprinzip der

Resolution.

Prologs Auswertungsstrategie beruht auf 2 Festlegungen:

- Anfragen werden von links nach rechts bearbeitet:
  Um $G_1, \ldots, G_m$ zu lösen, bearbeiten wir erst G_1.
- Prog.-Klauseln werden von oben nach unten bearbeitet. D.h.:
  Nimm die erste Programmklausel
  $H :- B_1, \ldots, B_n.$
  so dass G_1 und H unifizierbar sind.

Resolution

Wenn aus $B_1, \ldots, B_n$ die Aussage H folgt
und man $H, G_2, \ldots, G_m$ beweisen will,
dann reicht es dafür $B_1, \ldots, B_n, G_2, \ldots, G_m$ zu beweisen.

Zum Schluss wird nur der Teil der Antwortsubst. σ ausgegeben, der die Variablen in der

ursprünglichen Anfrage
$G_1, \ldots, G_m$ betrifft.

Bei Eingabe von ; wird zurückgesetzt und der Beweisbaum wird weiter in Tiefensuche durchsucht.
Reihenfolge der Literale in Klauselrümpfen und Reihenfolge der Programm-Klauseln beeinflusst den Aufbau des Beweisbaums u. daher auch den Erfolg u. das Terminierungsverhalten v. Prolog.

Unifikation kann man direkt in Prolog implementieren

```
gleich(X,X).
?- gleich(f(Y,a),f(b,Z)).
Y=b, Z=a
?- f(Y,a) = f(b,Z).
Y=b, Z=a
```

[X, 5 | K]

.(X, .(5, K))

In Prolog gibt es verschiedene Prädikate für Gleichheit. Hiermit lassen sich auch eingebauten vordef. Zahlen leicht verwenden.

Arithmetische Symbole wie +, -, ... sind zunächst Funktionssymbole wie alle anderen (sie können aber in Infix-Notation benutzt werden).

= : syntaktische Unifikation, wertet keine Funktionen aus

Es geht auch:

?- X=2, Z is X+1.

X=2, Z=3

Weitere Prädikate >, <, ... werten ebenfalls arithmetische Funktionen aus.