# IV.3 Rechnen in Prolog

Dienstag, 23. Januar 2018 09:00

Wie arbeitet Prolog genau? Dies muss man wissen, da es die Reihenfolge beeinflusst, in der Lösungen gefunden werden (u. es beeinflusst auch die Effizienz und das Terminierungsverhalten).

Unifikation:

Kann man die Variablen in 2 Ausdrücken so instantiieren, dass die Ausdrücke gleich werden?

Nötig für:

- Typ-Checking für Sprachen mit polymorphen Typen (z.B. Haskell, Java)

- Auswertung von Logikprogrammen
- ...

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

$\sigma(a(X, s(Y), s(Z)))$

Lösung:

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

---

Substitution $\sigma$ bildet fast alle Variablen auf sich selbst ab, d.h. nur bei endlich vielen Variablen $X$ gilt $\sigma(X) \neq X$.

Wir schreiben Substitutionen in der Form

$\sigma = \{X_1 = t_1, \ldots, X_n = t_n\}$.

Dies bedeutet

$\sigma(X_i) = t_i$ für $1 \leq i \leq n$

$\sigma(Y) = Y$ für Variablen $Y \notin \{X_1, \ldots, X_n\}$

Für $\sigma$ im Bsp:

$\sigma(X) = succ(zero)$

$\sigma(Z) = Z$

Wir erlauben auch die Anwendung von Substitutionen auf bel. Terme+Literale

$\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$

D.h: $\sigma$ instantiiert alle Variablen in dem Term/Literal:

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

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

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

Im allgemeinen können 2 Ausdrücke mehrere Unifikatoren haben.

Im Bsp: Die beiden unteren Unifikatoren sind Spezialfälle des oberen Unifikators. D.h: man kann sie erhalten, indem man erst den oberen Unifikator ausführt und danach noch $\{Z = zero\}$ bzw. $\{Z = succ(W)\}$ ausführt. ↖ Subst. $\tau$, die nach dem MGU

angewendet wird

## Most general unifier (MGU)

instantiiert "so wenig wie möglich", d.h., dieser soll berechnet werden.

Satz 1: Wenn 2 Ausdrücke unifizierbar sind, dann haben sie auch einen MGU.

Satz 2: Der MGU ist eindeutig bis auf Variablenumbenennungen.

Unifiziere

$f(X)$ und $f(Y)$.

MGU $\sigma_1 = \{X = Y\}$

MGU $\sigma_2 = \{Y = X\}$

Diese beiden MGUs sind gleich bis auf die Variablenumbenennung, die X durch Y und Y durch X ersetzt.

(Formal: Eine injektive Substitution, die Variablen auf Variablen abbildet, heißt Variablenumbenennung.)

Unifikationsalgorithmus:
Entscheidet, ob 2 Ausdrücke unifizierbar sind und berechnet ggf. den MGU.

1. X und X sind unifizierbar. MGU ist $\sigma = \{\}$.

2. $s = X$, $t = succ(Y)$

MGU ist $\sigma = \{ X = succ(Y) \}$.

Aber: $s = X$, $t = succ(X)$

sind nicht unifizierbar

OCCUR FAILURE

Aus Effizienzgründen wird in Prolog der Occur Check weggelassen (d.h. man überprüft nicht, dass t die Variable s nicht enthalten darf).

⇒ In Prolog sind X und succ(X) unifizierbar.

Der MGU instantiiert X mit dem Term $\underbrace{succ(succ(\ldots))}_{\infty \text{ oft}}$

Anders als in Haskell gilt das absichtliche Programmieren mit unendl. Termen als schlechter Stil. Unendl. Terme bzw. Auswirkungen des fehlenden Occur Checks treten in realen Prolog-Programmen selten auf.

3. Fall: $s = succ(Y)$, $t = X$

4. Fall: s und t sind beide keine Variablen

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

CLASH FAILURE

Wenn s und t mit versch.

Funktions-/Prädikatssymbolen beginnen, dann sind sie nicht unifizierbar.

Bsp:

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

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

Beginne mit 1. Argument

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

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

Betrachte 2. Argument, aber wende den bislang gefundenen Teil-Unifikator $\sigma_1$ an:

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

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

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

Betrachte 3. Argument, aber wende den bislang gefundenen Teil-Unifikator $\sigma_2 \circ \sigma_1$ an:

Komposition v. $\sigma_1$ und $\sigma_2$, wende erst $\sigma_1$ an und danach $\sigma_2$

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

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

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

Gesamt-Lösung:

$$\sigma = \sigma_3 \circ \sigma_2 \circ \sigma_1$$

$$= \{Y = succ(W)\} \circ \{Z = succ(Y)\} \circ \{X = succ(Y)\}$$

$= \{ X = succ(succ(W)),$

$Y = succ(W),$

$Z = succ(succ(W)) \}$

**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

$\sigma_1(Z)$ und $\sigma_1(g(X))$

$Z$ $\quad$ $g(X)$

Lösung: $\sigma_2 \circ \sigma_1 = \{ Y = h(X, g(X)),$

$Z = g(X) \}$

**Bsp** Unifiziere (Großbuchst.: Variablen; Kleinbuchst.: Fkt-Symb.)

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

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

$g(X)$ und $Y$

$\sigma_2$ unifiziert

$\sigma_1(Y)$ und $\sigma_1(a)$

$g(X)$ $a$

CLASH FAILURE

Bsp Unifiziere

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

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

$g(X)$ und $Y$

$\sigma_2 = \{X = h(Z)\}$ unifiziert

$\sigma_1(Y)$ und $\sigma_1(g(h(Z)))$

$g(X)$ $g(h(Z))$

$\sigma_3$ unifiziert

$\sigma_2(\sigma_1(Y))$ und $\sigma_2(\sigma_1(g(Z)))$

$g(h(Z))$ $g(Z)$

OCCUR FAILURE

# Beweisverfahren von Prolog

SOLVE: Kann auch nicht terminieren

## 2 Festlegungen bei der Auswertungsstrategie:

- Löse zuerst das linkeste Literal der Anfrage ($G_1$)

- Verwende die erste Prog-Klausel, die dazu noch nicht benutzt wurde.

Bei Terminierung mit leerer Antwort-subst $\sigma$ gibt Prolog "true" aus.

Fakten $\hat{=}$ Regel mit leerem Rumpf (d.h. $n=0$).

## Resolution

Wenn aus $B_1, \ldots, B_n$ die Aussage $H$ folgt

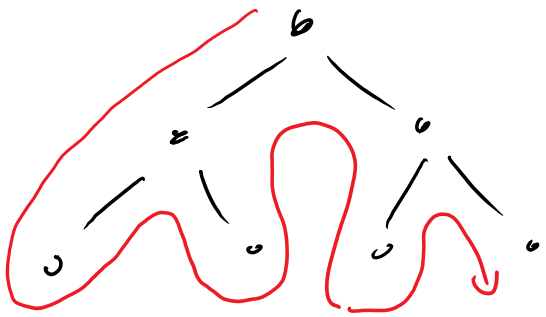
und man $H, G_2, \ldots, G_m$ beweisen will,

dann reicht es dafür, $B_1, \ldots, B_n, G_2, \ldots, G_m$

zu beweisen.

Reihenfolge der Literale in Klauseln und der Klauseln im Programm beeinflussen den Beweisbaum.

Beweisbaum wird in Tiefensuche von links nach rechts aufgebaut:



Vertauschte Literale im

Rumpf von Klausel (4):
rechtester Pfad wird
unendlich ⇒
Prog terminiert nicht
mehr, wenn man nach beiden
gefundenen Lösungen ";"
eintippt.

Vertausche Prog-Klausel
(3) und (4):
linkester Pfad wird ∞,
Prog läuft sofort in die
Nicht-Terminierung und
findet die Lösungen
weiter rechts im Beweis-
baum nicht.

# Unifikationsalgorithmus in Prolog:

```
gleich(X, X).

?- gleich(f(Y, s(Z)),
          f(U, Y)).
```

$Y = s(Z)$, $U = s(Z)$

```
?- .(a, L) = [X, b | K]
```

$.(X, .(b, K))$

$X = a$, $L = .(b, K)$

$[b | K]$

Bsp für Alg., der "="
benutzt: mem
("member", überprüft, ob
Element in Liste enthalten
ist)
Geht aber auch ohne "=",
indem man X 2-mal im
Klauselkopf benutzt.

## Eingebaute Zahlen in Prolog

- Prolog kennt die vordef. arithmet. Funktionssymbole
  +, -, *, /, mod
  Diese dürfen auch in

Infix-Schreibweise benutzt werden.

?- 2+3 = +(2,3)

true

- "=" berechnet nur syntaktische Unifikation. 7 und 2+5 sind nicht syntaktisch gleich.
- Aber: Es existieren vordef. Prädikatssymbole, die arithmet. Funktionen wie +, -, ... auswerten. (Normalerweise werden Funktionen in Prolog nicht ausgewertet.)

?- 2 is X.

Prog-Fehler, denn X ist zum Zeitpunkt der Auswertung nicht instantiiert.

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

X=2, Y=3

?- len([4,5,6], X).

X=3

Wenn man im Rumpf der len-Regel "M is N+1" durch "M = N+1" ersetzen würde:

?- len([4,5,6], X).

X = 0+1+1+1

Es ex. weitere vordef.
Prädikate zum Vergleich
von Zahlen wie >, <,
>=, =< etc.
Hier müssen beide Argu-
mente vollst. instant. arith.
Ausdrücke sein:

?- 2+1 > 2*1.

true