IV.3 Rechnen in Prolog

Tuesday, 27 January, 2015 16:00

Ziel: Zeige, wie Prolog wirklich arbeitet
→ wichtig, um zu verstehen ob u. in welcher Reihenfolge Lösungen gefunden werden.

Wenn Anfrage bearbeitet wird:
Entscheide, ob die Anfrage dem Kopf einer Programmklausel entspricht

(ja Fakten: Kopf ist das Faktum, d.h. Faktum ≙ Regel mit leerem Rumpf)

"entsprechen": Anfrage u. Klausel/Kopf müssen gleich sein, wenn Var. geeignet instantiiert werden.

Vgl:
Pattern Matching:
nur Var. aus Prog. werden instantiiert
Hier: Instantiiert sowohl Var. aus Prog. als auch Var. aus Anfrage
Beim Typ-Checking

Beim Typ-Checking v. polymorphen Typen stellt sich das gleiche Problem: 2 Typen mit Typvar. müssen gleich gemacht werden.

In Prolog: Überprüfe Prog-Klauseln v. oben u. unten u. untersuche, ob das 1. Literal der Anfrage mit dem Klauselkopf unifiziert.

Im Bsp:
Durchlaufe die zu unifizierenden Terme v. links nach rechts:
- äußeres Fktssymbol (add) ist gleich
- Mache 1. Argumente gleich:
  $X = succ(zero)$
- Mache 2. Arg gleich:
  $Y = zero$
- Mache 3. Arg gleich:
  $U = succ(Z)$

Substitution: instantiiert nur endl. viele Variablen, alle anderen bleiben unverändert.

Im Bsp:

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

$\sigma(Z) = Z$

$\sigma(succ(X)) =$ ← Subst. wird erweitert auf Terme.
$succ(succ(zero))$

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

v(add(X, succ(Z)), succ(Y))=

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

Im allgemeinen kann es bel. viele Unifikatoren geben. Welchen soll Prolog nehmen?

Den allgemeinsten Unifikator (Most General Unifier)

Alle anderen Unifikatoren sind Spezialfälle des MGU, d.h. man erhält sie aus MGU durch weitere Instantiierung der Variablen.

Im Bsp:

σ_1 ist der MGU

Zweiter Unifik:

$\sigma_2 = \tau_2 \circ \sigma_1$

mit $\tau_2(Z) = zero$

Dritter Unif:

$\sigma_3 = \tau_3 \circ \sigma_1$

mit $\tau_3(Z) = s(W)$

Satz: Wenn 2 Terme unifizierbar sind, dann haben sie einen MGU u. dieser ist eindeutig (bis auf Umbenennungen der Variablen)

Bsp:

MGU von f(X) und f(Y) ist

ist

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

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

Algorithmus zur Berechnung des MGU

1. Fall

Unifiziere X und X:

klappt, MGU ist $\{\}$

(d.h. die Substitution, die nichts ändert, die Identität).

2. Fall

Bsp: $s = X$,

$t = succ(Y)$

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

Bsp: $s = X$

$t = succ(X)$

nicht unifizierbar

Occur Failure

4. Fall s und t

$s = f(\ldots)$

$t = g(\ldots)$

Clash Failure

(2 versch. Funktionssymbole)

Bei $f(s_1, \ldots, s_n)$ und

$f(t_1, \ldots, t_n)$

laufe v. links nach rechts durch die Terme u. mache Argumente gleich.

Bsp: $S = P(X$ | Unifiziere | Unifiziere

mache Argumente gleich.

Bsp: $S = f(X, Z, succ(succ(W)))$

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

$\sigma_1 = \{X = succ(Y)\}$

↑ bedeutet: im zuletzt entstehenden Unifikator müssen X u. succ(Y) gleich sein

⇒ unter dieser Bedingung. Unifiziere jetzt die 2. Argumente

⇒ Unifiziere nicht Z und X,

sondern

$\underbrace{\sigma_1(Z)}_{Z}$ und $\underbrace{\sigma_1(X)}_{succ(Y)}$

Unifiziere $\underbrace{\sigma_1(Z)}_{Z}$ und $\underbrace{\sigma_1(X)}_{succ(Y)}$

$\sigma_2 = \{Z = succ(Y)\}$

Unifiziere

$\sigma_2(\sigma_1(succ(succ(W))))$

und $\sigma_2(\sigma_1(Z))$,

d.h.: succ(succ(W)) und succ(Y)

$\sigma_3 = \{Y = succ(W)\}$

Lösung: $\sigma_3 \circ \sigma_2 \circ \sigma_1 =$

$\{X = s(s(W)), Y = s(W), Z = s(s(W))\}$

Bsp ①

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

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

$\sigma_2 = \{Z = g(X)\}$.

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

Bsp ②

Unifiziere $f(\underline{g(X)}, \underline{Y})$ und $f(\underline{Y}, \underline{a})$

$\sigma_1 = \{Y = g(X)\}$ $\quad \sigma_1(Y) = g(X)$ $\quad \sigma_1(a) = a$

nicht unifizierbar

⇒ clash failure

Bsp ③

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

Unifiziere $f(g(X), Y, Y)$ und $f(Y, g(R(Z)), g(Z))$

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

$\sigma_1(g(R(Z))) = g(R(Z))$

$\sigma_2 = \{X = R(Z)\}$

$\sigma_2(\sigma_1(Y)) = g(R(Z))$

$\sigma_2(\sigma_1(g(Z))) = g(Z)$

nicht unifizierbar
=> occur failure

Gib den Algorithmus des Prolog-Interpreters an (benötigt MGU-Algorithmus als Hilfsalg.)

Prolog-Strategie

- Bearbeite Literale in einem Beweisziel von links nach rechts (d.h. beginne mit G_1)
- Durchsuche Prog-Klauseln v. oben n. unten
  => Unifiziere Klauselkopf H mit 1. Literal G_1 in Anfrage

Prog-Klausel kann auch Faktum sein (dann ist $n=0$).

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, stattdessen $B_1, \ldots, B_n, G_2, \ldots, G_m$

Zu beweisen.
Statt
$G_1, \ldots, G_m$
betrachte die modifizierte Anfrage
$\mu(G_1), \mu(G_2), \ldots \mu(G_m)$
$\mu(H)$
Um Konflikte mit zufällig gleichbenden Var. zu vermeiden, werden die Variablen in Prog-Klauseln immer so umbenannt, dass sie verschieden v. Variablen in Anfragen sind.
Antwortsubst:
Wende erst MGU aus erstem Schritt an, dann aus zweitem etc.
Ausgegeben wird nur, wie die Antwortsubst. die Var. aus der Anfrage instantiert.

Beweisverfahren von Prolog: Resolution

Algorithmus SOLVE

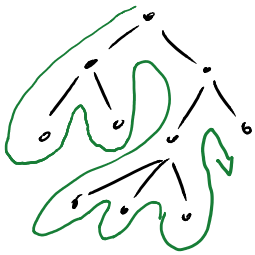
Eingabe: Anfrage ?- $G_1, \ldots, G_m$
Ausgabe: Antwortsubstitution σ oder Fehlschlag

1. Wenn m = 0, dann terminiere mit $\sigma = \{\}$.
2. Sonst: Suche nach der nächsten Programmklausel
   $H :- B_1, \ldots, B_n.$
   so dass G_i und H unifizierbar (mit MGU μ) sind.
   Gibt es keine, dann terminiere mit Fehlschlag.
3. Rufe SOLVE mit der folgenden Anfrage auf:
   ?- $\mu(B_1), \ldots, \mu(B_n), \mu(G_2), \ldots, \mu(G_m)$.
4. Falls dieser Aufruf Antwortsubst. τ berechnet,
   dann: Terminiere mit $\sigma = \tau \circ \mu$.
   sonst: Gehe zurück zu Schritt 2.

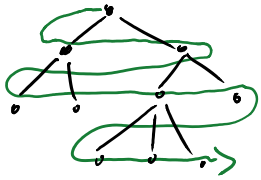
Prog-Klausel hat andere

Var. als Anfrage
(Sonst Umbenennung der Var.
in Prog-Klausel)

Beweisbaum wird in
Tiefensuche aufgebaut:



Mögliche Alternative:
Breitensuche:



Generell:

- Reihenfolge der Literale im Klauselrumpf kann Terminierungsverhalten beeinflussen.
  ⇒ Bei Klauseln
  $p(X_1,\ldots,X_n)$:- ...
  sollte rekursives p-Literal nicht am Anfang des Klauselrumpfs stehen.
- Reihenfolge der Prog-Klauseln beeinflusst in welcher Reihenfolge Lösungen gefunden werden u. das Terminierungsverhalten
  ⇒ nicht-rekursive Klauseln sollten vor rekursiven Klauseln des gleichen Prädikats kommen.

?- gleich(s,t).
ist genau dann lösbar
wenn
X und s und gleichzeitig
X und t unifizieren
$(\sigma(X) = \sigma(s) = \sigma(t))$
$\Rightarrow$ genau dann lösbar,
wenn s und t unifizieren.
Antwortsubst. ist ihr MGU.

2. Bsp:

$.(a, L) = [X, b \mid K]$

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

MGU:

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

$[b \mid K]$

Prolog verzichtet aus
Effizienzgründen auf den
occur-check:
?- X = succ(X).
ist lösbar. Lösung
X = succ(succ(....))
unendl. Term
(Es ex. auch ein vordef.
Prädikat für Unifikation
mit Occur Check.)

Vordefinierte Zahlen in Prolog

(lesbarere u. effizientere
Alternative zu

zero- und succ- Notation)

Arithmetische Symbole dürfen in Infix-Notation benutzt werden:

```
+(2,5) = 2+5
```

---

len mit vordefinierten Zahlen

```
len([],0).
len([_|Rest],NP0):-
    len(Rest,N), NP0 is N+1.
```

Warum nicht:

```
len([_|Rest],N+1):-
    len(Rest,N).

?- len([1,2],X).
X = 0+1+1
```

Es ex. weitere vordef. Prädikate, die arithmetische Operationen ebenfalls auswerten (z.B. >, >=, ...)

```
?- 3+5 > 2+3.
true
```