Prof. Dr. Stefan Brass
Dr. Henning Thielemann
Institut für Informatik
Universität Halle

12.05.2011

Deduktive Datenbanken und Logische Programmierung — Blatt 5: T_P-Operator —

Aufgabe 5 — 7 Punkte

Betrachten Sie folgendes Prolog-Programm P:

```
% route(A, B, R): Um von A nach B zu kommen, benutze man Route R.
route(weinbergweg, von_seckendorff_platz, walter_huelse_str).
route(weinbergweg, weinbergmensa, wolfgang_langenbeck_str).
route(weinbergweg, peissnitz, schwanenbruecke).
route(weinbergmensa, peissnitz, wilde_saale_schwanenbruecke).
route(peissnitz, markt, wuerfelwiese_robert_franz_ring).
route(peissnitz, uniplatz, wuerfelwiese_moritzburgring).
route(markt, uniplatz, barfuesserstr).
route(markt, bahnhof, leipziger_str).

% Alle routen sind bidirektional:
weg(A, B, R) :- route(A, B, R).
weg(A, B, R) :- route(B, A, R).

% Von der Informatik am Von-Seckendorff-Platz erreichbare Orte:
erreichbar_von_informatik(von_seckendorff_platz, []).
erreichbar_von_informatik(B, [R|RL]) :-
        erreichbar_von_informatik(A, RL),
        weg(A, B, R),
        not(member(R,RL)).
```

Die Aufgabe ist nun, $T_P \uparrow 0$ bis $T_P \uparrow 4$ zu bestimmen (als Erweiterung zur Definition in der Vorlesung muss man `not(memberR,RL))` dabei als Literal mit "eingebautem Prädikat" behandeln, d.h. es wird direkt im Interpreter ausgewertet, ohne dass wirklich Fakten über `member` hergeleitet werden).

Sie können diese Faktenmengen per Hand berechnen, oder sich von Prolog helfen lassen. Das obige Programm steht unter folgender URL zur Verfügung:

```
http://www.informatik.uni-halle.de/~brass/lp11/weg.pl
```

Besonders elegant wäre es, sich ein Programm zu schreiben, das den T_P-Operator möglichst direkt für möglichst allgemeine Programme berechnet. (das ist aber nur eine Option für die Lösung dieser Aufgabe, eine per Hand berechnete Abgabe wird auch als Lösung akzeptiert, und es gibt viele Zwischenstufen).

In einem solchen Programm sind die Regeln des obigen Programms Daten. Eine Regel $A \leftarrow B_1 \wedge \cdots \wedge B_n$ kann dann codiert werden als

```
rule(A, [B1, ..., Bn]).
```

Es gibt mehrere Möglichkeiten, diese Aufgabe zu lösen. Eine verwendet die dynamische Datenbank. Sie können z.B. mittels

```
:- dynamic fact/2.
```

ein zweistelliges Prädikat deklarieren, in das Sie zur Laufzeit z.B. mittels

assert(fact(2,weg(A,B,R)))

ein Fakt einfügen können. Das `assert` muss als Anfrage ausgeführt werden, also im Regelrumpf, nachdem Sie `A`, `B`, `R` passend gebunden haben. Typischerweise würde man die Regel mit `fail` beenden, so dass ein Backtracking über alle Belegungen von `A`, `B`, `R` ausgeführt wird, und alle in die dynamische Datenbank eingefügt werden (der Seiteneffekt von `assert` wird beim Backtracking nicht zurückgenommen). Wenn Sie die dynamische Datenbasis wieder löschen wollen, geht das mittels `retractall(fact(N,F))` (es werden alle Fakten gelöscht, die auf das gegebene Literal passen).

Da ein Regelrumpf als Liste repraesentiert ist, müssen Sie sich ein Prädikat `eval` schreiben, das für eine Liste von Literalen feststellt, ob sie als Fakten in der Datenbank enthalten sind, bzw. im Falle von `not(member(R,RL))` wahr sind (eingebaute Prädikate müssen dabei speziell behandelt werden). Um die Variablen brauchen Sie sich dabei nicht zu kümmern, die werden automatisch an die Werte in den Fakten gebunden (sie brauchen also nicht über eine Substitution Buch zu führen, sondern können so tun, als wären die Regeln schon Grundinstanzen).

Präsenzaufgabe 5

Berechnen Sie den allgemeinen Unifikator sofern möglich:

- $\mathit{length}([1,2,3], X)$ und $\mathit{length}([], 0)$.
- $\mathit{length}([1,2,3], X)$ und $\mathit{length}([E|R], N1)$.
- $\mathit{append}(X, [2,3], [1,2,3])$ und $\mathit{append}([F|R], L, [F|A])$.
- $p(f(X), Z)$ und $p(Y, a)$.
- $p(f(a), g(X))$ und $p(Y, Y)$.
- $q(X, Y, h(g(X)))$ und $q(Z, h(Z), h(Z))$.
- Benutzen Sie Prolog, um die Lösung zu überprüfen: z.B.

  ```
  p(X)=p(a), write(p(X)=p(a)).
  ```

  Berücksichtigen Sie aber ggf. den fehlenden "Occur check".