Prof. Dr. Stefan Brass
Institut für Informatik
MLU Halle-Wittenberg

24.06.2022

Logische Programmierung & deduktive Datenbanken — Übungsblatt 10 (Typ-Prüfung) —

Ihre Lösungen zu den Hausaufgaben a) und b) geben Sie bitte über die Übungs-Plattform in StudIP ab. Einsendeschluss ist der 30. Juni, 10^{00} (die Aufgaben werden anschließend in der Übung besprochen).

Hausaufgaben

a) Berechnen Sie den Prädikat-Abhängigkeitsgraphen und den reduzierten Prädikat-Abhängigkeitsgraphen des folgenden Programms (die q_i sind EDB-Prädikate):

$$
\begin{array}{lcl}
\mathsf{p}_1 & \leftarrow & \mathsf{q}_1 \wedge \mathsf{q}_2. \\
\mathsf{p}_1 & \leftarrow & \mathsf{q}_1 \wedge \mathsf{q}_3. \\
\mathsf{p}_2 & \leftarrow & \mathsf{p}_1. \\
\mathsf{p}_3 & \leftarrow & \mathsf{q}_3. \\
\mathsf{p}_3 & \leftarrow & \mathsf{p}_1. \\
\mathsf{p}_3 & \leftarrow & \mathsf{p}_2. \\
\mathsf{p}_4 & \leftarrow & \mathsf{p}_2 \wedge \mathsf{p}_3. \\
\mathsf{p}_5 & \leftarrow & \mathsf{p}_4 \wedge \mathsf{q}_2.
\end{array}
\qquad
\begin{array}{lcl}
\mathsf{p}_6 & \leftarrow & \mathsf{p}_5. \\
\mathsf{p}_4 & \leftarrow & \mathsf{p}_6. \\
\mathsf{p}_7 & \leftarrow & \mathsf{p}_5 \wedge \mathsf{p}_3. \\
\mathsf{p}_7 & \leftarrow & \mathsf{q}_4 \wedge \mathsf{p}_7. \\
\mathsf{p}_8 & \leftarrow & \mathsf{p}_2. \\
\mathsf{p}_9 & \leftarrow & \mathsf{p}_8 \wedge \mathsf{p}_7.
\end{array}
$$

Sie müssen den Graphen nicht abgeben. Beantworten Sie aber folgende Fragen:

- Welche der Prädikate sind rekursiv?
- Welche Regeln sind rekursiv?
- Was sind die rekursiven Cliquen?
- Geben Sie eine Berechnungsreihenfolge für die Regeln des Programms an. Markieren Sie, welche Regeln in Schleifen iteriert werden müssen.

b) Schreiben Sie einen einfachen Typ-Prüfer für Prolog/Datalog-Regeln in Prolog. Zur Vereinfachung sei angenommen, dass die Prädikate der zu prüfenden Regeln nur ganze Zahlen und Atome als Argumente haben, keine strukturierten Terme. Die Prädikate, die in diesen Regeln verwendet werden dürfen, seien im Typprüfer selbst deklariert, und zwar in folgender Form:

```
% komponist(KNR, NAME, VORNAME, GEBOREN, GESTORBEN)
pred(komponist(int, atom, atom, int, int)).

% stueck(SNR, KNR→komponist, TITEL, TONART, OPUS).
pred(stueck(int, int, atom, atom, atom)).

pred(<(int, int)).
pred(ergebnis(int, int)).
...
```

D.h. die verwendbaren Prädikate sind (vorläufig) in den Typprüfer fest eingebaut. Programmieren Sie ein Prädikat `check/1`, so dass man auf folgende Art die Typprüfung durchführen kann:

```
read(X), check(X).
```

Die Eingabe `X` kann dabei eine Regel, ein einzelnes Literal, oder eine Konjunktion von Literalen (Anfrage, Beweisziel) sein. Die syntaktische Richtigkeit können Sie voraussetzen. Das eingebaute Prädikat `read` liefert die Regel/Anfrage auch bereits als Term (Baumstruktur). Wenn Sie `:-` und `,` als normale Funktoren verwenden wollen, müssen Sie diese Symbole in `'...'` einschließen, z.B. `':-'`.

Es reicht aus, wenn Ihr Prädikat `check` fehlschlägt, falls die Eingabe einen Typfehler enthält. Man bekommt in diesem Fall also die Antwort "no" und sonst "yes". Falls Sie wollen, können Sie natürlich auch bessere Fehlermeldungen ausgeben (mit `write` und `nl`), bevor Ihr Prädikat fehlschlägt (ggf. hierzu `fail` verwenden). Gute Fehlermeldungen verkomplizieren die Aufgabe allerdings deutlich.

Zum Beispiel sollte die Eingabe

```
ergebnis(X, Y) :-
        komponist(_, 'Wolfgang Amadeus', 'Mozart', X, Y).
```

als korrekt gewertet werden, die folgenden aber alle nicht:

- `ergebnis(X, Y) :- komponist(X, Y, Y, 'Wolfgang Amadeus', 'Mozart').`
  Atom-Konstanten an Integer-Argumentpositionen.
- `ergebnis(X1, X2) :- komponist(Nr, X1, X2, Y1, Y2).`
  `X1` und `X2` müssten nach dem Rumpf-Literal Atome sein, aber nach dem Kopf-Literal ganze Zahlen.
- `ergebnis(X, X) :- komponist(Nr, 'Wolfgang Amadeus', 'Mozart', X).`
  Es gibt kein Prädikat `komponist` mit nur vier Argumenten.

Die Bereichsbeschränkung brauchen Sie nicht zu prüfen, d.h. es soll legal sein, Variablen im Kopf zu verwenden, die im Rumpf nicht vorkommen.

Sie können diese Aufgabe in drei Schritten lösen:

- Überführen Sie die Eingabe in eine Liste von Literalen.
- Ersetzen Sie die Konstanten in diesen Literalen durch die Typbezeichner `atom` und `int`. Den Test auf die Typen können Sie mit den eingebauten Prädikaten `atom/1`, `integer/1` und `var/1` durchführen. Um Literale in Prädikat und Argumentliste zu zerlegen, können Sie das Prädikat `=..` verwenden. Damit können Sie auch umgekehrt wieder Literale erzeugen.
- Nun müssen Sie nur noch jedes Literal Ihrer Liste mit den gegebenen `pred`-Fakten unifizieren. Dafür ist wichtig, dass Sie im zweiten Schritt die Variablen aus der Eingabe erhalten, so dass eine inkonsistente Verwendung der gleichen Variable mit unterschiedlichen Typen auch bemerkt wird.

Freiwillige Zusatzaufgabe: Lesen Sie die `pred`-Fakten und die zu prüfenden Regeln aus einer Datei. Definieren Sie also ein Prädikat `check_file/1`, dem man den Namen der zu prüfenden Datei mit Prädikat-Deklarationen und Programm-Regeln übergibt.

Sie können die `pred`-Fakten mit `assert` in die dynamische Datenbank laden (brauchen dann aber eine `dynamic`-Deklaration). So kann das obige `check`-Prädikat unverändert weiter verwendet werden.

Zur Wiederholung

c) Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zur Bottom-Up Auswertung antworten?

- Was ist das Ziel der Bottom-Up Auswertung? Was ist die theoretische Grundlage?
- Erläutern Sie den Prädikat-Abhängigkeitsgraphen. Wie ist er definiert? Wozu ist er gut?
- Definieren Sie „rekursives Prädikat“ und „rekursive Regel“.
- Wie bestimmt man eine Auswertungsreihenfolge für die Prädikate?
- Was ist der “reduzierte Prädikat-Abhängigkeitsgraph”? Welche Bedeutung hat das Ergebnis einer topologischen Sortierung dieses Graphen?
- Wie kann man ein relationales DBMS zur Bottom-Up Auswertung nutzen? Beschreiben Sie insbesondere auch, wie dabei Datalog-Regeln in SQL-Befehle übersetzt werden.
- Erklären Sie, wie man Operationen der relationalen Algebra zur Auswertung von Regeln verwenden kann.
- Wie kann man die Auswertung von Datalog-Regeln selbst implementieren (in einer Sprache wie C++ oder Java)?
- Diskutieren Sie die Bedeutung der Duplikat-Eliminierung.

Zum Selbstudium

d) Schauen Sie sich die folgenden Webseiten an. Sie brauchen für diese Aufgabe nichts abzugeben. Ziel ist, dass Sie einen Eindruck davon gewinnen, was es im Internet zum Thema dieser Vorlesung gibt, und dabei für sich nützliche Quellen entdecken.

- Schauen Sie sich das XSB System an:

  [http://xsb.sourceforge.net/]

  Das XSB System ist ein Prolog System mit Tabellierung, so dass für Datalog-Programme die Terminierung garantiert ist. Den Artikel „XSB as an Efficient Deductive Database Engine“ von Konstantinos Sagonas, Terrance Swift, und David S. Warren (SIGMOD'94, 442–453) finden Sie hier:

  [https://doi.org/10.1145/191839.191927]

  Falls Sie nicht im Netz der Universität sind, und daher kostenlosen Zugriff auf die Artikel der ACM haben, gibt es hier eine kostenlos zugreifbare Version:

  [https://www.researchgate.net/publication/
  2792472_XSB_as_an_Efficient_Deductive_Database_Engine]

- Zu der Vorlesung “Knowledge Representation” von Marek Sergot gibt es Vorlesungsmaterialien hier:

  [https://www.doc.ic.ac.uk/~mjs/teaching/491.html]

  Sie könnten sich z.B. das Kapitel “Minimal models and fixpoint semantics for definite programs” anschauen:

  [https://www.doc.ic.ac.uk/~mjs/teaching/
  KnowledgeRep491/Fixpoint_Definite_491-2x1.pdf]

- Sie finden meine Habilitationsschrift “Bottom-Up Query Evaluation in Extended Deductive Databases” (Universität Hannover, 1997) z.B. hier:

  [http://nbn-resolving.de/urn:nbn:de:gbv:089-2367669109]

  Zumindest die Kapitel 1 und 2 wären auch für diese Vorlesung sehr lesenswert.