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

07.07.2018

Logische Programmierung & deduktive Datenbanken — Übungsblatt 11 —

Ihre Lösungen zu der Hausaufgabe g) schicken Sie bitte per EMail an den jeweiligen Übungsleiter (mit "`[lp18]`" in der Betreff-Zeile). Einsendeschluss ist der 16. Juli.

Zum Selbstudium

a) 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 WWW zum Thema dieser Vorlesung gibt, und dabei für sich nützliche Quellen entdecken.

- Das Skript zur Vorlesung "Techniken der Logikprogrammierung" von François Bry (LMU München) wurde hier in eine HTML-Version umgewandelt:

  [https://www.en.pms.ifi.lmu.de/publications/projektarbeiten/]
  [Martin.Kluger_Kalliopi.Sidiropoulou/00-startseite.html]

  Die Original-Folien sind leider nicht zugreifbar.

- Das Kapitel "Logikprogrammierung" aus der Vorlesung "Paradigmen der Programmierung" von Prof. Dr. Erich Ehses (FH Köln) ist hier verfügbar:

  [http://www.gm.fh-koeln.de/ehses/paradigmen/folien/prolog.pdf]

- Ein kommerzieller Anbieter eines deduktiven Datenbanksystems is LogicBlox:

  [http://www.logicblox.com/]

  Eine kurze Übersicht zum System und seiner Sprache findet sich im Artikel "LogicBlox, Platform an Language: a Tutorial" von Todd J. Green, Molham Aref, and Grigoris Karvounarakis:

  [https://developer.logicblox.com/wp-content/uploads/2013/10/]
  [datalog2020tutorial-1.pdf]

  Eine etwas ausführlichere Erklärung zur Funktionsweise ist der Artikel "Design and Implementation of the LogicBlox System" von Molham Aref et.al.:

  [http://www.cs.ox.ac.uk/dan.olteanu/papers/logicblox-sigmod15.pdf]

- IF/Prolog ist ein bekanntes kommerzielles Prolog-System (neben SICStus):

  [http://www.ifcomputer.de/Products/Prolog/home_de.html]

- Das "Datalog Educational System DES" ist ein deduktives Datenbanksystem mit Datalog, SQL, relationaler Algebra und Tupelkalkül und Bereichskalkül:

  [https://www.fdi.ucm.es/profesor/fernan/DES/]

- Das Soufflé Projekt ist eine spannende neue Implementierung von Datalog zum Zweck der statischen Analyse von Java-Programmen:

  [http://souffle-lang.org/]

  Die Folien zu einem Vortrag über die Sprache des Systems finden sich hier:

  [http://souffle-lang.org/pdf/SouffleLanguage.pdf]

  Ein älteres Benutzer-Handbuch ist auf dieser Seite verfügbar:

  [https://github.com/oracle/souffle/wiki/user-manual]

  Auf Linux-Systemen kann man Soufflé aus dem Source Code compilieren:

  [http://souffle-lang.org/download/]

b) Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zu Kapitel 5 antworten? (Einige Fragen zu diesem Kapitel standen schon auf dem letzten Übungsblatt.)

- Geben Sie ein Beispiel für eine Schleife mit dem eingebauten Prädikat `repeat`? Wie sieht das Vier-Port-Modell von `repeat` aus?
- Wie können Sie alle Lösungen einer Anfrage `p(X, Y)` ausdrucken?
- Erläutern Sie das Konzept der "Definite Clause Grammars". Was sind Unterschiede zur Parsergeneratoren wie "`yacc`"?
- Wie kann man kontextfreie Grammatiken in Prolog aufschreiben? Erläutern Sie die Syntax der Grammatikregeln. Welches eingebaute Prädikat nutzt man dann zur Syntaxanalyse? Wie muss die Eingabe dafür repräsentiert werden?
- Wie werden die Grammatik-Regeln in normale Prolog-Regeln übersetzt? Beschreiben Sie die Differenzlisten-Technik.
- Was ist der Zweck von Argumenten der Nichtterminalsymbol-Prädikate? Geben Sie ein Beispiel für eine Anwendung.
- Wie kann man zusätzlichen Prolog-Code in die Grammatik-Regeln einbauen?
- Was passiert im Fall von Syntaxfehlern? Wie könnte man die Fehlermeldungen verbessern?
- Wie würde man eine "Definite Clause Grammar" in Prolog schreiben für die Sprache $\mathtt{a}^{+}\mathtt{b}$, d.h. zuerst beliebig viele "`a`" (mindestens eins), dann genau ein "`b`"?

Denken Sie auch über folgende Fragen zu Kapitel 6 nach:

- 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?
- 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.
- Wie kann man die Auswertung von Datalog-Regeln selbst implementieren (in einer Sprache wie C++ oder Java)?
- Diskutieren Sie die Bedeutung der Duplikat-Eliminierung.
- Was ist der Zweck der "seminaiven Auswertung"?
- Erläutern Sie die Idee der seminaiven Auswertung in dem einfachen Fall, dass es nur ein rekursives Rumpfliteral gibt.
- Beschreiben Sie die seminaive Auswertung genauer: Welche vier Varianten eines Prädikats (unterschiedliche Tupelmengen) werden durch die seminaive Auswertung eingeführt? Wie werden die Regeln umgeschrieben? Welche Befehle werden zur Initialisierung vor der Schleife und zum Weiterschalten nach jedem Schleifendurchlauf ausgeführt?
- Betrachten Sie den Fall einer Regel mit zwei rekursiven Rumpfliteralen. Durch die seminaive Auswertung spart man von den vier möglichen Kombinationen nur eine, nämlich den Join-Anteil von alten mit alten Tupeln. Warum ist das trotzdem wichtig und lohnt den Aufwand?
- Warum braucht man bei einer Regel mit n rekursiven Rumpfliteralen nicht $2^n - 1$ Versionen der Regel, sondern nur n?

Präsenzaufgaben

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

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

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.

d) Diskutieren Sie die möglichen Auswertungsreihenfolgen für folgende Regel:

$$\mathsf{p}(\mathsf{X}, \mathsf{Z}) \leftarrow \mathsf{q}(\mathsf{X}, \mathsf{Y}_1, \mathsf{Y}_2, \mathsf{Y}_3) \wedge \mathsf{r}(\mathsf{Y}_1, \mathsf{Z}).$$

Dabei sei angenommen, dass das erste Argument von `q` und `r` ein Schlüssel der jeweiligen Relation ist.

e) Das Prädikat `p` sei durch folgende Regel definiert:

$$\mathsf{p}(\mathsf{X}) \leftarrow \mathsf{q}(\mathsf{X}) \wedge \mathsf{r}(\mathsf{X}).$$

Die Prädikate `q` und `r` sind EDB-Prädikate. Folgende Kosten werden geschätzt:

- `q` produziert 100 Tupel in 100 ms, wenn es mit Bindungsmuster `f` aufgerufen wird.
- Wenn `q` dagegen mit Bindungsmuster `b` aufgerufen wird, kann ein Index verwendet werden, um den Elementtest auszuführen. Das dauert 3 ms.
- `r` liefert 1000 Tupel in 200 ms, wenn es mit Bindungsmuster `f` aufgerufen wird.
- Für `r` gibt es keinen Index. Wenn es mit Bindungsmuster `b` aufgerufen wird, muss ein "Full Table Scan" ausgeführt werden, der im Erfolgsfall (Wert ist vorhanden) durchschnittlich 100 ms kostet (die Suche kann nach der Hälfte abgebrochen werden), im Misserfolgsfall dagegen 200 ms. Es sei angenommen, dass die Hälfte der Werte aus `q` auch in `r` vorkommen.
- Man könnte auch bei Relationen sortieren und dann die Schnittmenge in einem parallelen Durchlauf berechnen, das würde 1000 ms dauern.

Was ist unter diesen Annahmen die beste Auswertungs-Strategie?

f) Das folgende Programm berechnet Paare von Personen, die von einem gemeinsamen Vorfahren gleich viele Generationen entfernt sind ("same generation cousins"):

$$
\begin{array}{lcl}
\texttt{sg(X, X)} & \leftarrow & \texttt{person(X).} \\
\texttt{sg(X, Y)} & \leftarrow & \texttt{parent(X, Xp)} \wedge \texttt{parent(Y, Yp)} \wedge \\
 & & \texttt{sg(Xp, Yp).} \\
\texttt{answer(X)} & \leftarrow & \texttt{sg(julia, X).}
\end{array}
$$

Es sei hier angenommen, dass `person` und `parent` EDB-Prädikate sind. Berechnen Sie das "Adorned Program", d.h. klären Sie die Auswertungsreihenfolge und machen Sie die auftretenden Bindungsmuster explizit.

Falls noch Zeit ist, kann in der Übung auch die "Magische Mengen" Transformation an diesem Beispiel besprochen werden.

Hausaufgabe

g) In dieser Aufgabe sollen Sie ein ganz kleines Textadventure-Spiel erstellen.

Textadventure-Spiele sind eine Art Bücher, bei denen der Leser/Spieler den Ablauf der Handlung beeinflußt. Es wird ihm/ihr jeweils die aktuelle Situation beschrieben, z.B.:

> „Du bist auf einem Waldweg, der hier eine Biegung macht. Im Norden befindet sich eine Lichtung, während im Osten der Weg tiefer in den Wald hineinführt. Du hörst das Summen vieler Bienen."

Der Spieler kann nun Kommandos/Spielzüge eingeben, wie z.B. „(ich) gehe nach Norden". Tatsächlich wird dieses Kommando meist durch „n" abgekürzt (das ist für den Spieler wie den Programmierer bequemer). Natürlich kann der Spieler nur in Richtungen gehen, die der Programmierer/Autor vorgesehen hat: Auf das Kommando „s" müsste in der obigen Situation geantwortet werden „dort ist der Wald zu dicht" (oder einfach „das geht nicht"). So ein Spiel ist häufig auf eine recht kleine Zahl von Orten/Räumen beschränkt, an denen sich der Spieler befinden kann (kommerzielle Spiele aus den Achtzigern etwa 100).

Das bloße Herumgehen auf der vom Programmierer entworfenen Karte und das Entdecken neuer Orte würde aber schnell langweilig. Ein anderes wichtiges Kommando ist es, Sachen zu untersuchen oder näher zu betrachten. Dadurch bekommt der Spieler häufig zusätzliche Informationen oder findet versteckte Gegenstände. Im Beispiel könnte er durch „Betrachte Bienen" ein Astloch mit Honigwaben finden. Schließlich ist es noch wichtig, Gegenstände nehmen zu können, z.B. „Nimm Honig". Natürlich würden die Bienen das verhindern.

Hier gilt es also eine echte Aufgabe zu lösen, wie man nun doch an den Honig herankommt. Zum Beispiel könnte sich auf der Lichtung eine Hütte befinden, und in dieser vielleicht eine Pfeife, deren Rauch die Bienen abschreckt. Der Spieler muß sich also in einer bestimmten Reihenfolge bestimmte Gegenstände verschaffen und dabei

eine gewisse Phantasie entwickeln. Solche Gegenstände können ihm dann auch neue Wege öffnen. So würde der Honig vielleicht einen Bären besänftigen, der den Eingang zu einer Höhle versperrt. In der Höhle befindet sich z.B. ein Schatz, der das Ziel des Spieles darstellt.

Sie können sich natürlich auch eine andere Geschichte ausdenken. Ihr Spiel sollte mindestens 3 „Räume" und einen Gegenstand haben, und neben den Kommandos für die vier Himmelsrichtungen auch noch mindestens zwei weitere Kommandos („Verben") „verstehen" (z.B. „nimm Gegenstand", und „betrachte Gegenstand"). Es ist nicht verlangt, dass Ihr Spiel eine sinnvolle Geschichte enthält — es reicht, wenn man in der kleinen Spielwelt herumgehen kann und den Gegenstand nehmen und betrachten kann. In erster Linie sollen Sie die Grammatik aus Kapitel 5 für Textadventurespiele ausprobieren, und ggf. etwas erweitern oder modifizieren. Sie müssen dazu auch den Scanner am Ende des Kapitels programmieren, um Eingabezeichen bis zum Zeilenende einzulesen und daraus Worte zu machen (Prolog Atome), die dann von der Grammatik verarbeitet werden können. Alternativ können Sie mit der Grammatik auch bis zuf die Zeichen-Ebene herunter gehen.

Die Ausgaben für die Kommandos dürfen recht einfach sein, aber vielleicht haben Sie ja auch Spass daran, eine einfache Spielwelt zu implementieren.