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

01.07.2022

Logische Programmierung & deduktive Datenbanken — Übungsblatt 11 (Bottom-Up-Auswertung) —

Ihre Lösungen zu den Hausaufgaben a) bis c) geben Sie bitte über die Übungs-Plattform in StudIP ab. Einsendeschluss ist der 7. Juli, 10^{00} (die Aufgaben werden anschließend in der Übung besprochen).

Hausaufgaben

a) Mit den folgenden Regeln kann man den T_P-operator implementieren:

```
derivable(Head, Facts) :-
    rule(Head, BodyList, _Vars),
    is_true(BodyList, Facts).

is_true([], _).
is_true([Lit|BodyRest], Facts) :-
    member(Lit, Facts),
    is_true(BodyRest, Facts).

tp(FactsIn, FactsOut) :-
    findall(Fact, derivable(Fact, FactsIn), FactsOut).
```

Die Repräsentation der Regeln ist wie auf dem Blatt 8 für die SLD-Resolution, nämlich in der Form `rule(Head, BodyList, VarList)`. Z.B. wird die Regel

```
p(X) :- q1(X), q2(X,c).
```

folgendermaßen als Daten gespeichert:

```
rule(p(X), [q1(X), q2(X,c)], [var('X', X)]).
```

Das letzte Argument erlaubt eine verbesserte Ausgabe der Variablen, wird aber hier nicht benötigt (wenn Sie nicht bei der Regelanwendung die Substitution ausgeben wollen). Sie dürfen das Argument auch entfernen.

Machen Sie sich ein einfaches Beispiel und prüfen Sie, dass diese Regeln den T_P-Operator korrekt implementieren. Sie finden obige Regeln (und die für die folgende Teilaufgabe) unter

[http://www.informatik.uni-halle.de/~brass/lp22/prolog/tp0.pl]

b) Die Iteration des T_P-Operators können Sie mit den folgenden Regeln in Prolog definieren:

```
minmod(Facts) :-
    minmod_iter(0, [], Facts).

minmod_iter(N, Facts, MinMod) :-
    tp(Facts, NewFacts),
    (has_changed(Facts, NewFacts) ->
        NextN is N+1,
        minmod_iter(NextN, NewFacts, MinMod);
        MinMod = Facts).

has_changed(Facts, NewFacts) :-
    member(Fact, NewFacts),
    \+ member(Fact, Facts).
```

Bauen Sie Ausgaben in dieses Programm ein, so dass man die iterative Berechnung des minimalen Modells mit dem T_P-Operator gut nachvollziehen kann.

Machen Sie ein kleines Beispiel, das mindestens drei Iterationen benötigt.

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

```
sg(X, X)   ←  person(X).
sg(X, Y)   ←  parent(X, Xp) ∧ parent(Y, Yp) ∧
              sg(Xp, Yp).
answer(X)  ←  sg(julia, X).
```

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.

Freiwillige Zusatzaufgabe: Führen Sie die „Magische Mengen“ Transformation zu Ende (die Berechnung des „Adorned Program“ ist ja nur der erste Schritt).

Aufgabe für nächste Woche (Abgabe am 14.07.2022)

d) 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.

Sie können natürlich die Grammatik für Textadventure-Befehle aus Kapitel 7 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 auf 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.

Versuchen Sie, Ihr Programm möglichst übersichtlich zu strukturieren.

Zur Wiederholung

e) Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zur seminaiven Auswertung antworten?

- 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?

f) Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zur „Magische Mengen"-Methode antworten?

- Wenn man von „Top-down" und „Bottom-up"-Auswertung spricht, was stellt man sich oben und was unten vor?
- Nennen Sie ein typisches „Top-down" und ein einfaches „Bottom-up" Verfahren. Was sind die jeweiligen Stärken und Schwächen?
- Was ist das Ziel des „Magische Mengen" Verfahrens?
- Was codieren die magischen Prädikate? Erläutern Sie die Codierung an einem Beispiel.
- Warum ist es nützlich, dass das „Magische Mengen" Verfahren eine Transformation auf Quellcode-Ebene ist? Inwieweit sollte man die magischen Prädikate aber doch speziell behandeln?
- Was ist eine SIP-Strategie? Geben Sie ein Beispiel an.
- Finden Sie eine gute Auswertungsreihenfolge für die Rumpfliterale einer gegebenen Regel bei gegebenem Bindungsmuster für das Kopfliteral.
- Was muss man tun, wenn ein Prädikat bei der gewählten SIP-Strategie mit zwei verschiedenen Bindungsmustern aufgerufen wird?

- Das Ergebnis der „Magischen Mengen“-Transformation besteht aus „modifizierten Regeln“ und „magischen Regeln“. Erläutern Sie Zweck und Aufbau dieser beiden Arten von Regeln.
- Führen Sie die „Magische Mengen“-Transformation an einem einfachen Beispiel durch.

Zum Selbstudium

g) 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.

- Herr Wenzel und ich sind dabei, Datalog-basierte Sprachen zur Programmierung von Microcontrollern (z.B. Arduino) zu entwickeln. Wir sind uns bezüglich der genauen Sprache noch nicht ganz einig, aber „Konkurrenz belebt das Geschäft“. Man muss wohl auch verschiedene Möglichkeiten anschauen, um zu einer guten Sprache zu kommen. Sie sind natürlich eingeladen, da mitzuwirken.

  Die Webseite von Herrn Wenzel ist:

  [https://dbs.informatik.uni-halle.de/microlog/]

  Insbesondere haben wir auch eine Implementierung vorgeschlagen, die auf erweiterten endlichen Automaten basiert.

- Meine Sprache ist leicht anders. In dem Artikel ging es aber in erster Linie nicht um die Sprache, sondern um die Verifikation von Invarianten/Integritätsbedingungen:

  [https://users.informatik.uni-halle.de/~brass/micrologS/]

- In meinem Artikel auf der RuleML+RR 2021 habe ich eine Sprache vorgeschlagen, die auf Events basiert:

  [https://users.informatik.uni-halle.de/~brass/micrologE/rr21.pdf]

  Die Folien meines Vortrags finden Sie hier:

  [https://users.informatik.uni-halle.de/~brass/micrologE/rr21talk.pdf]

  Leider ist die Implementierung noch „under Construction“. Ich hoffe, dass ich in diesem Sommer dazu kommen werde. Auch hierzu wäre natürlich Mithilfe willkommen.