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

04.07.2024

Logische Programmierung & deduktive Datenbanken — Übungsblatt 12 (Adventure in Prolog) —

Sie müssen diese Woche keine Hausaufgabe mehr abgeben. Falls Sie allerdings zu wenige Hausaufgaben im Semester abgegeben haben, oder Sie zu selten an den Übungen teilgenommen haben, wäre die Aufgabe auf diesem Blatt noch eine Möglichkeit, einiges wettzumachen. Es ist auch allgemein günstig, mehr Programmierpraxis zu gewinnen, und dazu vielleicht eine optionale Aufgabe noch zu machen.

Hausaufgaben

Aufgabe 1 (Hausaufgabe, 0 Punkte)

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. Wahrscheinlich wäre es günstig, die Spiel-Daten in Fakten abzulegen (wie in einer relationalen Datenbank), und nicht in komplexen Regeln zu verstecken.

Wir haben ein Projekt zur Entwicklung eines SQL-Lernspiels in Form eines Textadventurespiels. Ein erster Prototyp ist auf dieser Seite verfügbar:

[https://users.informatik.uni-halle.de/~brass/db19/uebung.html]

Möglicherweise könnte ihnen dieses Spiel bei Bedarf noch Anregungen geben.

Eine wesentliche größere Version mit Web-Schnittstelle ist in Entwicklung. Basierend auf den Erfahrungen mit diesem Prototyp wurde das Konzept auch noch etwas geändert. Bei Interesse kann ich Ihnen weitere Informationen schicken.

Zur Wiederholung

Aufgabe 2 (Wiederholungs-Fragen)

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

a) Wenn man von „Top-down“ und „Bottom-up“-Auswertung spricht, was stellt man sich oben und was unten vor?

b) Nennen Sie ein typisches „Top-down“ und ein einfaches „Bottom-up“ Verfahren. Was sind die jeweiligen Stärken und Schwächen?

c) Was ist das Ziel des „Magische Mengen“ Verfahrens?

d) Was codieren die magischen Prädikate? Erläutern Sie die Codierung an einem Beispiel.

e) 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?

f) Was ist eine SIP-Strategie? Geben Sie ein Beispiel an.

g) Finden Sie eine gute Auswertungsreihenfolge für die Rumpfliterale einer gegebenen Regel bei gegebenem Bindungsmuster für das Kopfliteral.

h) Was muss man tun, wenn ein Prädikat bei der gewählten SIP-Strategie mit zwei verschiedenen Bindungsmustern aufgerufen wird?

i) Das Ergebnis der „Magischen Mengen“-Transformation besteht aus „modifizierten Regeln“ und „magischen Regeln“. Erläutern Sie Zweck und Aufbau dieser beiden Arten von Regeln.

j) Führen Sie die „Magische Mengen“-Transformation an einem einfachen Beispiel durch.

Aufgabe 3 (Wiederholungs-Fragen)

Was würden Sie in einer mündlichen Prüfung auf die folgenden fortgeschrittenen Fragen zu magischen Mengen antworten?

a) Was ist der Zweck der „Supplementary Predicates“?

b) Wann ist ein Programm „rektifiziert“? Wofür wird diese Bedingung gebraucht?

c) Was ist die Beziehung zwischen magischen Fakten, die aus dem magisch transformierten Programm herleitbar sind, und dem SLD-Baum (für das ursprüngliche Programm)?

d) Inwiefern kann man sagen, dass aus dem magisch transformierten Programm nur solche nicht-magischen Fakten herleitbar sind, die auch bei der SLD-Resolution bewiesen werden?

e) Was ist ein Vorteil der Bottom-Up Auswertung des magisch transformierten Programms gegenüber der SLD-Resolution?

f) Was ist umgekehrt ein Vorteil der SLD-Resolution gegenüber der Bottom-Up Auswertung des magisch transformierten Programms?

g) Nennen Sie einige Probleme der „Magische Mengen“ Methode.

h) Was ist die Grundidee der „SLDMagic“ Methode?

Aufgabe 4 (Wiederholungs-Fragen)

Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zur Negation antworten?

a) Wie kann man sehen, dass der Operator „`not`“ (bzw. „`\+`“) in der logischen Programmierung nicht die klassische Negation ist?

b) Warum macht es Sinn, in dieser Art der Wissensrepräsentation nicht die klassische Negation aus der Logik zu verwenden? Zu welchen Schwierigkeiten würde das führen?

c) Machen Sie ein Beispiel, das zeigt, dass wir es in der logischen Programmierung mit nichtmonotoner Logik zu tun haben.

d) Wie funktioniert SLDNF-Resolution?

e) Was ist das Problem, wenn negierte Literale noch ungebundene Variablen enthalten, wenn sie mit SLDNF-Resolution ausgewertet werden?

f) Was ist ein „kausales Modell“ (engl. „Supported Model“)?

g) Was ist die Grundidee von „Clark's Completion“?

h) Welches Problem soll die Stratifizierung lösen?

i) Wann ist ein Programm stratifiziert?

j) Was ist das perfekte Modell eines stratifizierten logischen Programms?

k) Gibt es praktische Anwendungen für nicht stratifizierte Programme?

l) Nennen Sie eine Semantik für nicht stratifizierte Programme. Wie wird in dieser Semantik das Problem „`p :- not p.`“ gelöst?

m) Nennen Sie einige Programmtransformationen auf variablenfreien logischen Programmen, die für sinnvolle Semantiken die Menge der intendierten Modelle nicht ändern sollten (also Äquivalenztransformationen sein sollten).

Übungsaufgaben (gemeinsam in der Übung zu bearbeiten)

Aufgabe 5 (Präsenzaufgabe)

(Dies ist eine Klausuraufgabe von 2011.)
Schreiben Sie ein Prädikat `glaetten(X, Y)`, das eine Liste `X` von Zahlen (z.B. Aktienkurse oder Messwerte) in eine Liste `Y` von Zahlen überführt, bei der jeder Wert in `X` durch den Durchschnitt dreier aufeinanderfolgender Werte in `Y` ersetzt ist. Am Ende der Liste `Y` (wenn es in `X` keine drei Werte mehr gibt), soll zuerst der Durchschnitt der letzten beiden Werte von `X` stehen, und dann der letzte Werte von `X`. Man nimmt also jeweils den Durchschnitt der noch vorhandenen Werte. Ist z.B.

```
X = [1, 2, 3, 4, 5],
```

so wird die Liste `Y` berechnet als

```
[avg(1,2,3), avg(2,3,4), avg(3,4,5), avg(4,5), avg(5)].
```

Den Durchschnitt berechnen Sie als Summe geteilt durch Anzahl, (es gibt in Prolog keine `avg`-Funktion), das Ergebnis wäre also `Y = [2, 3, 4, 4.5, 5]`. Für die leere Liste als Eingabe soll die leere Liste als Ausgabe geliefert werden.
Sie können nur die im Skript genannten eingebauten Prädikate verwenden, z.B. `is`, `>`, `=`, `\=`, `\+`. Sie können sich selbstverständlich beliebige Hilfsprädikate in Prolog definieren. Sie können davon ausgehen, dass die Eingabe korrekt ist, das Prädikat also immer mit einer Liste von Zahlen im ersten Argument aufgerufen wird.

Aufgabe 6 (Präsenzaufgabe)

(Dies ist eine Klausuraufgabe von 2011.)
Gegeben sei das folgende Programm P:

```
a ← b ∧ c.
b ← c.
c ← d ∧ e.
d ← f.
f ← c.
f ← g.
e ← h.
g.
h.
```

Berechnen Sie das minimale Modell von P durch Iteration des T_P-Operators. Geben Sie die Mengen $I_0 := \emptyset$, $I_i := T_P(I_{i-1})$ für $i = 1, 2, \ldots$ an, bis ein Fixpunkt erreicht ist. Selbstverständlich brauchen Sie nur die jeweils neuen Fakten explizit anzugeben, z.B. in der Form $I_2 = I_1 \cup \{\ldots\}$.

Aufgabe 7 (Präsenzaufgabe)

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.

Aufgabe 8 (Präsenzaufgabe)

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?

Zum Selbstudium

Aufgabe 9 (Zusatzaufgabe)

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.

a) Es sei nochmals auf das Prolog-Tutorial „Adventure in Prolog“ von Dennis Merritt verwiesen:

   [https://www.amzi.com/AdventureInProlog/a1start.php]

b) Das Soufflé System ist eine schnelle Datalog-Implementierung (auch mit paralleler Auswertung):

   [https://souffle-lang.github.io/]

   Zumindest auf Linux- und Mac-Rechnern lässt es sich recht einfach installieren.

   Folien zur Einführung in die Sprache von Soufflé gibt es hier:

   [https://courses.cs.washington.edu/courses/cse344/18sp/
   SouffleLanguage.pdf]

   Hier gibt es ein Youtube-Video von einem Vortrag von Bernhard Scholz:

   [https://www.youtube.com/watch?v=Qp3zfM-JSx8]

c) Ich habe mich mit der Negation in der logischen Programmierung beschäftigt, und dazu eine ganze Reihe Veröffentlichungen. In der Vorlesung werden wir z.B. über die Wohlfundierte Semantik (WFS) sprechen. Der folgende Artikel enthält einen Vorschlag zur Berechnung:
   Stefan Brass, Jürgen Dix, Burkhardt Freitag, Ulrich Zukowski: Transformation-Based Bottom-Up Computation of the Well-Founded Model. Theory and Practice of Logic Programming 1:5, Sept. 2001, pp. 497–538.

   [https://www.researchgate.net/publication/1955167
   _Transformation-Based_Bottom-Up_Computation
   _of_the_Well-Founded_Model]

   Die offizielle Quelle (bei Cambridge University Press) ist:

   [https://doi.org/10.1017/S147106840100103X]