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

06.06.2024

Logische Programmierung & deduktive Datenbanken — Übungsblatt 8 (Mini SQL Grammatik) —

Ihre Lösungen zu den Hausaufgaben 1 bis 3 geben Sie bitte über die Übungs-Plattform in StudIP ab. Einsendeschluss ist Mittwoch, der 12. Juni, 16^{00} (die Aufgaben werden in der Übung am Donnerstag besprochen).

Hausaufgaben

Aufgabe 1 (Hausaufgabe, 3 Punkte)

Um sich mit dem Dateizugriff in Prolog vertraut zu machen, schreiben Sie bitte ein Prädikat `cat(Datei)`, das den Inhalt einer Datei auf den Bildschirm ausgibt.
Einen Eingabestrom zu einer Datei bekommen Sie mit dem Prädikat `open`, z.B.

```
open('test.sql', read, EingabeStrom)
```

Dabei ist `read` der Modus, in dem die Datei geöffnet wird. Die Dokumentationsseite zu `open/3` ist

[https://www.swi-prolog.org/pldoc/man?predicate=open/3]

Allerdings wird sofort auf `open/4` verweisen, das noch eine Liste von Optionen hat:

[https://www.swi-prolog.org/pldoc/doc_for?object=open/4]

Wenn Sie zu Testzwecken nicht aus einer Datei, sondern von einem String lesen wollen, bekommen Sie einen passenden Eingabestrom mit `open_string(String, Stream)`:

[https://www.swi-prolog.org/pldoc/doc_for?object=open_string/2]

Zeichen eines Eingabestroms können Sie z.B. mit `get_char(Stream, Char)` lesen:

[https://www.swi-prolog.org/pldoc/man?predicate=get_char/2]

Sie bekommen die Zeichen dann als Atome, die aus jeweils einem Zeichen bestehen. Am Dateiende wird das Atom `end_of_file` geliefert:

[https://www.swi-prolog.org/pldoc/doc_for?object=get_char/1]

Aufgabe 2 (Hausaufgabe, 4 Punkte)

Schreiben Sie einen Scanner (lexikalische Analyse) für die im folgenden beschriebene SQL-Teilmenge. Definieren Sie dazu ein Prädikat

```
scanner(EingabeStrom, Tokenliste).
```

In der Datei, auf die über den Eingabestrom zugegriffen wird, steht eine SQL-Anfrage. Ausgabe soll eine Liste von Token (Wortsymbolen) sein, die folgendermaßen repräsentiert sind:

- Die vier SQL-Schlüsselworte `select`, `from`, `where`, `and` als diese Atome. Es reicht, wenn Sie die Kleinschreibung erkennen. Es steht Ihnen frei, Großschreibung in der Eingabe auf die kleingeschriebenen Worte abzubilden. Sie müssen das aber nicht tun.
- Bezeichner als Terme der Form `id(Name)`, wobei `Name` der Bezeichner als Prolog-Atom ist. Wieder ist eine Abbildung auf Kleinbuchstaben optional. Bezeichner sind Folgen von Buchstaben und Ziffern, die mit einem Buchstaben anfangen.
- Nicht-negative ganze Zahlen (Folgen von Ziffern) als Terme der Form `int(Wert)`, wobei `Wert` der entsprechende Zahlwert ist.
- Das Zeichen „`*`“ in der Eingabe als das Atom `star`.
- Das Zeichen „`,`“ in der Eingabe als das Atom `comma`.
- Das Zeichen „`=`“ in der Eingabe als das Atom `eq`.
- Sie dürfen natürlich noch weitere Vergleichsoperatoren, Schlüsselworte oder auch String-Konstanten implementieren, müssen aber nicht.

Z.B. soll die Anfrage

```
select * from studenten where sid = 101
```

so als Tokenliste repräsentiert werden:

```
[select, star, from, id(studenten), where, id(sid), eq, int(101)]
```

Zwischen zwei Tokens soll beliebiger Leerplatz erlaubt sein, also eine beliebige Folge von Leerzeichen, Tabulatorzeichen und Zeilenumbrüchen. Kommentare brauchen Sie nicht zu implementieren.
Im Skript war ein sehr viel einfacherer Scanner angegeben, den Sie eventuell als Ausgangspunkt benutzen können:

[https://users.informatik.uni-halle.de/~brass/lp24/prolog/scan.pl]

Aufgabe 3 (Hausaufgabe, 4 Punkte)

Schreiben Sie nun eine „Definite Clause Grammar“ für die folgende sehr einfache Teilmenge von SQL:

- Das Schlüsselwort `select` am Anfang ist notwendig.
- Danach folgt entweder ein `*` oder eine Liste von Termen, durch Kommata getrennt.
- Ein Term ist ein Bezeichner oder eine Zahlkonstante (wenn Sie auch Zeichenkettenkonstanten im Scanner implementiert haben, auch das).
- Die `from`-Klausel ist optional. Zur Vereinfachung ist nur genau ein Tabellenname (also ein Bezeichner `id(_)`) nach `from` erlaubt.
- Die `where`-Klausel ist optional. Sie kann aber nur angegeben werden, wenn die `from`-Klausel auch vorhanden ist. Nach `where` folgt eine Liste von Vergleichen, durch `and` getrennt. Ein Vergleich besteht aus zwei Termen links und rechts und in der Mitte einem Gleichheitszeichen `eq` (falls Sie mehr Vergleichsoperatoren implementiert haben, können Sie die hier natürlich auch verwenden).

Eingabe ist eine Liste von Tokens, wie sie der Parser von Aufgabe 2 liefert. Falls Sie Aufgabe 2 noch nicht gelöst haben, können Sie natürlich auch manuell Testlisten eingeben.
Das Startsymbol der Grammatik soll `query` sein. Ihre Grammatik braucht nur die syntaktische Korrektheit zu beschreiben. Natürlich würde man später aus den Anfragen auch die entscheidenen Daten herausziehen, um die SQL-Anfragen auszuführen oder analysieren zu können. Das ist hier aber nicht verlangt.
Schreiben Sie ein Prädikat `sql(TokenList)`, das genau dann wahr ist, wenn die Tokenliste der skizzierten Grammatik genügt.

Zur Wiederholung

Aufgabe 4 (Wiederholungs-Fragen)

Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zu „Definite Clause Grammars“ antworten?

a) Wie unterschieden sich Grammatiken in Prolog von Grammatiken, die im Compilerbau verwendet werden (mit Werkzeugen wie `yacc`/`Bison`)?

b) Geben Sie ein einfaches Beispiel einer „Definite Clause Grammar“, z.B. für die Sprache $\texttt{a}^n\texttt{b}^n$. In Prolog müsse also z.B. die Liste `[a,a,b,b]` als korrekt erkannt werden.

c) Was ist die Idee der Übersetzung von Grammatik-Regeln in Prolog-Regeln?

Übungsaufgaben (gemeinsam in der Übung zu bearbeiten)

Aufgabe 5 (Präsenzaufgabe)

Definieren Sie ein Prädikat `sum(X,Y,Z)`, das genau dann gilt, wenn `X + Y = Z`, und das die Bindungsmuster `bbf`, `bfb`, `fbb` und `bbb` behandeln kann. Sie können die verschiedenen Fälle unterscheiden mit `var(X)`, `nonvar(X)` bzw. `number(X)`.

Aufgabe 6 (Präsenzaufgabe)

Bei der einfachsten Variante des NIM-Spiels besteht der Spielzustand aus einer Anzahl n von Streichhölzern. Die beiden Spieler können abwechselnd 1 oder 2 Hölzer nehmen. Wer das letzte Holz nimmt, hat gewonnen. Es gibt viele weitere (auch komplexere) Varianten, wie der Eintrag in der Wikipedia erläutert:

[https://de.wikipedia.org/wiki/Nim-Spiel]

Schreiben Sie ein Prädikat, das mit einer gegebenen Anzahl n von Hölzern aufgerufen wird, und ausgibt, ob man den Gewinn erzwingen kann, und wenn ja, wieviel Hölzer (1 oder 2) man nehmen muss.
Sie könnten z.B. Prädikate `gewinn(`n`)` und `verlust(`n`)` definieren, wobei `gewinn(`n`)` wahr ist, wenn der Spieler, der an der Reihe ist, den Gewinn erzwingen kann. Entsprechend bedeutet `verlust(`n`)`, dass der Gegenspieler den Gewinn erzwingen kann. Nun gilt `gewinn(`n`)` sicher für $n = 1$ und $n = 2$. Außerdem gilt es, wenn mindestens einer der beiden erreichbaren Zustände $n - 1$ und $n - 2$ Verlust-Positionen sind. Entsprechend ist n eine Verlust-Position, wenn beide erreichbaren Spielzustände Gewinn-Positionen sind.
Alternativ können Sie ein Prädikat `bewertung(`n`,`b`)` definieren, wobei $b = +1$ Gewinn bedeutet (für den Spieler am Zug) und $b = -1$ Verlust (Unentschieden gibt es hier nicht). Wenn Sie am Zug sind, berechnen Sie das Maximum der Bewertungen für die erreichbaren Spielzustände. Wenn der Gegner am Zug ist, das Minimum. Deswegen heißt dieses Verfahren der MinMax-Algorithmus:

[https://de.wikipedia.org/wiki/Minimax-Algorithmus]

Wenn der Spielbaum zu groß ist, kann man die Suche auch vorzeitig abbrechen, und muss den letzten Zustand dann mit einer Heuristik bewerten. In dieser Situation sind auch andere Zahlen außer -1, ggf. 0 und $+1$ möglich.

Zum Selbstudium

Aufgabe 7 (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) AUf dieser Webseite finden Sie eine Sammlung von Aufgaben zu Prolog (von Karsten Hönig, einem Informatik-Lehrer):

[http://www.hoenig.info/informatik/prolog/Word/InGN13%20-%20KI%20Aufgaben%20PROLOG%202.doc]

b) Ein kommerzieller Anbieter eines deduktiven Datenbanksystems ist LogicBlox:

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

Es ist mir unklar, wie aktiv die Firma noch ist. Der CEO ist weitergewandert zu [https://relational.ai/].

- 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://www.cs.ucdavis.edu/~green/papers/datalog12_tutorial.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]

- Ein weiterer Artikel zur Sprache „LogiQL“ ist „LogiQL: A Declarative Language for Enterprise Applications“ von Todd J. Green:

  [https://dl.acm.org/doi/10.1145/2745754.2745780]

c) Mit der Geschichte von Prolog beschäftigen sich u.a. folgende Artikel:

- Alain Colmerauer, Philippe Roussel: The birth of Prolog. November 1992.

  [http://www-public.tem-tsp.eu/~gibson/Teaching/Teaching-ReadingMaterial/ColmerauerRoussel96.pdf]

- Robert A. Kowalski: The Early Years of Logic Programming. CACM 31:1, 1988.

  [https://dl.acm.org/doi/pdf/10.1145/35043.35046]