Logische Programmierung & deduktive Datenbanken — Übungsblatt 7 (Mini SQL Grammatik) —

Ihre Lösungen zu den Hausaufgaben a) bis d) geben Sie bitte über die Übungs-Plattform in StudIP ab. Einsendeschluss ist der 09. Juni, 10⁰⁰ (die Aufgaben werden anschließend in der Übung besprochen).

Hausaufgaben

- a) 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)
```

Hier 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]

- b) Schreiben Sie einen Scanner (lexikalische Analyse) für die unten beschriebene SQL-Teilmenge. Es soll folgendes Prädikat entwickelt werden:

```
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/lp22/prolog/scan.pl]
```

c) Schreiben Sie nun eine „Definite Clause Grammar“ für die folgende einfache SQL-Teilmenge:

- 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).

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 entschiedenen Daten herausziehen, um die SQL-Anfragen auszuführen oder analysieren zu können. Das ist hier aber nicht verlangt.

Schreiben Sie ein Prädikat `sql(Stream)`, das genau dann wahr ist, wenn über den Eingabestrom eine Zeichenfolge gelesen wird, die der skizzierten Grammatik genügt.

d) Zeichnen Sie einen SLD-Baum für die Anfrage `grandfather(ian, G)` an folgendes Programm:

```
[1] grandfather(X, Z) :- grandparent(X, Z), male(Z).
[2] grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
[3] parent(eric, alan).
[4] parent(eric, bianca).
[5] parent(fiona, chris).
[6] parent(fiona, doris).
[7] parent(ian, eric).
[8] parent(ian, fiona).
[9] male(alan).
[10] male(chris).
[11] male(eric).
[12] male(ian).
```

Sie können die Namen gern mit dem ersten Buchstaben abkürzen. Geben Sie den Baum entweder als Bild ab (z.B. auch handgezeichnet und abfotografiert) oder textuell beschrieben. Sie brauchen keine Substitutionen anzugeben (nur die Knoten).

Zur Wiederholung

- e) Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zur praktischen Prolog-Programmierung antworten?
- Welche Entsprechungen gibt es zwischen Datentypen in klassischen Programmiersprachen wie C, Pascal oder Java und Daten in Prolog? Welche Möglichkeiten gibt es, Arrays in Prolog zu repräsentieren?
  - Ein wesentlicher Unterschied zwischen imperativen Sprachen und Prolog ist, dass man in imperativen Sprachen der gleichen Variable im Ablauf einer Berechnung mehrfach einen neuen Wert zuweisen kann, während das bei Prolog nur einmal möglich ist (sofern man nicht mit Backtracking zu einem früheren Berechnungszustand zurückkehrt). Welche Möglichkeiten gibt es, diese Einschränkung zu umgehen, wenn man einen imperativ formulierten Algorithmus nach Prolog übersetzen will? Welchen Vorteil hat die Lösung in Prolog?
  - Was leistet die Endrekursions-Optimierung und wann ist sie anwendbar?
- f) Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zu „Definite Clause Grammars“ antworten?
- Wie unterscheiden sich Grammatiken in Prolog von Grammatiken, die im Compilerbau verwendet werden (mit Werkzeugen wie `yacc/Bison`)?
  - Geben Sie ein einfaches Beispiel einer „Definite Clause Grammar“, z.B. für die Sprache $a^n b^n$. In Prolog müsse also z.B. die Liste `[a, a, b, b]` als korrekt erkannt werden.
  - Was ist die wesentliche Idee der Übersetzung von Grammatik-Regeln in Prolog-Regeln?

Zum Selbststudium

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.

- Die Symbole für „Modi“ von Prädikat-Argumenten („Eingabe oder Ausgabe“) in der SWI-Prolog-Dokumentation sind hier erläutert:

[<http://www.swi-prolog.org/pldoc/man?section=modes>]

- Prolog wird auch an Schulen eingesetzt, z.B. gibt es diese Linkliste auf dem Bildungsserver Berlin-Brandenburg:

[<https://bildungsserver.berlin-brandenburg.de/inf-prolog/>]

- Auf dem Hessischen Bildungsserver gibt es eine Seite zu SWI-Prolog, das dort allerdings auch zur Implementierung anderer Sprachen wie Mini-Pascal, Logo und Turtle verwendet wird:

[<http://arbeitsplattform.bildung.hessen.de/fach/informatik/swiprolog/>]

- Webseite eines Lehrers zu Prolog-Kursen an der Schule:

[<https://www.tinohempel.de/info/info/prolog/index.htm>]

Das zitierte „Arbeitsbuch Prolog“ von Hartmut Göhner und Bernd Hafenbrak findet sich z.B. hier:

[http://www.erasmus-reinhold-gymnasium.de/info/prolog/arbeitsbuch_prolog.pdf]

- Die Folien zur Vorlesung „Programmiertechniken in der Computerlinguistik I“ von Simon Clematide (basierend auf Unterlagen von Sascha Brawer) beinhalten auch eine Einführung in die Programmiersprache Prolog:

[<https://files.ifi.uzh.ch/cl/siclemat/lehre/ws0001/pcl1/>]

Es gibt auch eine Fortsetzung „Programmiertechniken in der Computerlinguistik II“, die sich noch mehr mit Anwendungen von Prolog für die Computerlinguistik befasst:

[<https://files.ifi.uzh.ch/cl/siclemat/lehre/ss01/pcl2/>]