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

11.04.2024

Logische Programmierung & deduktive Datenbanken — Übungsblatt 1 (Einführung) —

Ihre Lösungen zu den Hausaufgaben (Aufgabe 1 bis 3) geben Sie bitte über die Übungs-Plattform in StudIP ab. Einsendeschluss ist Mittwoch, der 17. April, 16^{00} (die Aufgaben werden in der Übung am Donnerstag besprochen). Aufgabe 4 wird dort auch besprochen: Sie sollten vorher darüber nachdenken, müssen aber nichts abgeben.

Hausaufgaben

Es sei die Beispiel-Datenbank aus der Einführungs-Vorlesung betrachtet. Damit es für die Prolog-Fakten besser passt, wurden die Tabellennamen in die Singular-Form in Kleinschreibung überführt. Entsprechend wurden auch Aufgabentypen und Aufgabenthemen als normale Prolog Konstanten („Atome“) geschrieben. Sie finden eine Prolog-Datei mit den entsprechenden Fakten unter folgender Web-Adresse:

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

- `student`: enthält die Daten von Studierenden der Vorlesung.
  - `SID`: „Studierenden-ID“ (eindeutige Nummer).
  - `VORNAME`, `NACHNAME`: Vor- und Nachname.
  - `EMAIL`: Email-Adresse (Der Nullwert wird als leeres Atom `\` dargestellt).

student			
SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Grau	NULL
103	Daniel	Sommer	···
104	Iris	Winter	···

- `aufgabe`: Beschreibt Aufgaben mit ihren Daten.
  - `ATYP`: Typ/Kategorie der Aufgabe.
    Z.B. `h`: Hausaufgabe, `z`: Zwischenklausur, `e`: Endklausur.
  - `ANR`: Aufgabennummer (eindeutig nur innerhalb des Typs).
  - `THEMA`: Thema der Aufgabe.
  - `MAXPT`: Maximale/volle Punktzahl der Aufgabe.

aufgabe			
ATYP	ANR	THEMA	MAXPT
h	1	ER	10
h	2	SQL	10
z	1	SQL	14

- `bewertung`: Abgegebene Lösungen zu einer Aufgabe (mit der erreichten Punktzahl).
  - `SID`: Student, der die Lösung abgegeben hat.
    Dies referenziert eine Zeile in `student`, d.h. es ist garantiert, dass es ein `student`-Fakt mit der gleichen `SID` gibt (Fremdschlüssel).
  - `ATYP`, `ANR`: Identifikation der Aufgabe.
    Zusammen identifiziert die beiden Werte ein Fakt des Prädikats `aufgabe`.
  - `PUNKTE`: Punkte, die der Student/die Studentin für die Lösung bekommen hat.
  - Falls es keinen Eintrag für einen Studenten und eine Aufgabe gibt, wurde die Aufgabe nicht abgegeben.

bewertung			
SID	ATYP	ANR	PUNKTE
101	h	1	10
101	h	2	8
101	z	1	12
102	h	1	9
102	h	2	9
102	z	1	10
103	h	1	5
103	z	1	7

Aufgabe 1 (Hausaufgabe, 2 Punkte)

Definieren Sie ein Prädikat

```
lisas_ha(ANR, PUNKTE, MAXPT),
```

das die Hausaufgabenpunkte der Studentin „Lisa Weiss“ enthält.

Die Namen der Argumente dienen nur zur Erläuterung der Bedeutung. Sie können selbstverständlich beliebige Variablennamen verwenden. Da Variablennamen nur innerhalb einer Regel bekannt sind, hätte der Aufrufer des Prädikates ohnehin keine Möglichkeit, herauszufinden, wie die Argumente in Ihrer Regel heißen.

Aufgabe 2 (Hausaufgabe, 2 Punkte)

Nehmen wir an, wir wüßten, dass Lisa Weiss und Daniel Sommer die einzigen Informatiker unter den Studierenden sind. Definieren Sie ein Prädikat

```
inf_ha(Vorname, Nachname, ANr, Punkte),
```

um die Hausaufgaben-Punkte dieser Studierenden im Blick zu behalten.

Sie dürfen selbstverständlich auch beliebige Hilfsprädikate einführen. Damit können Sie Codeduplizierung vermeiden und Ihr Programm leichter änderbar machen.

Aufgabe 3 (Hausaufgabe, 2 Punkte)

Definieren Sie ein Prädikat

`gut(VORNAME, NACHNAME)`,

das Studierende enthält, die sowohl in Hausaufgabe 1, als auch in Hausaufgabe 2 mindestens 8 Punkte bekommen haben.
Den numerischen Vergleich können Sie in Prolog als `P >= 8` schreiben. Dazu muss die Variable `P` weiter links im Regelrumpf schon einmal vorgekommen sein, und dabei an einen Wert gebunden sein. D.h. dieses eingebaute Prädikat kann zwei gegebene Zahlen auf die $\geq$-Beziehung testen, aber es kann natürlich nicht die unendlich vielen Zahlen aufzählen, die größergleich 8 sind. Prolog arbeitet die gegebenen Bedingungen im Regelrumpf immer von links nach rechts ab.

Zur Wiederholung

Aufgabe 4 (Wiederholungs-Fragen)

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

a) Was bedeutet eigentlich deklarative Programmierung? Warum ist SQL eine deklarative Sprache?

b) Welche Vorteile hat deklarative Programmierung? (Sie könnten das z.B. anhand von SQL-Anfragen erläutern, die Sie mit entsprechenden Java-Programmen vergleichen. In einigen Jahren wurde der Vergleich in der Einführungs-Vorlesung zu Datenbanken praktisch ausprobiert.)

c) Nennen Sie mindestens ein Prolog-System.

d) Welche logischen Operatoren können in Prolog-Regeln verwendet werden (soweit bisher an Beispielen in der Vorlesung gezeigt)? Wie sind Regeln aufgebaut?

e) Wie kann man den Inhalt einer relationalen Datenbank in Prolog darstellen?

f) Wie unterscheiden sich Variablen und Konstanten in Prolog syntaktisch?

g) Wann kann man die Anführungszeichen `'...'` weglassen?

h) Wie sieht die anonyme Variable aus? Was unterscheidet sie von normalen Variablen? Welchen Schutz gibt es in Prolog gegen Tippfehler in Variablen?

i) Wie kann man in Prolog eine Datei mit Fakten und Regeln laden?

j) Wie verhält sich Prolog, wenn man eine Anfrage stellt, die mehr als eine Antwort hat?

k) Wie kann man ein Prolog-System verlassen, d.h. die Kommandoschleife des Interpreters beenden? (Es heisst nicht „`quit`“.)

l) Kann Prolog in Endlosschleifen geraten? Geben Sie ggf. ein Beispiel. Was kann man dann tun?

Übungsaufgaben (gemeinsam in der Übung zu bearbeiten)

Aufgabe 5 (Präsenzaufgabe)

a) Laden Sie sich die folgende Datei mit den Beispiel-Fakten zu Verwandtschaftsbeziehungen herunter:

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

Diese Datei enthält Fakten zu folgenden Prädikaten, die im ersten Beispiel der Vorlesung genutzt werden:

- `person(ID, FirstName, LastName, Gender).`
- `parent(Child, Parent).`
- `couple(Partner1, Partner2).`

Die Datei enthält außerdem Regeln für folgende abgeleitete Prädikate (Sichten):

- `man(PersonID).`
- `woman(PersonID).`
- `father(Child, Father).`
- `mother(Child, Mother).`
- `grandparent(Child, Grandparent).`
- `married_with(Person1, Person2).`
- `ancestor_with(Child, Ancestor).`

Starten Sie ein Prolog-System. (Auf unseren Linux-Rechnern öffnen Sie dazu ein Terminal-Fenster und geben dann „`prolog`“ oder „`swipl`“ ein.)

Nach dem Start zeigen die meisten Prolog-Interpreter mit “`?-`” an, dass sie auf eine Anfrage (Beweisziel) warten. Auch Befehle, die den Zustand des Prolog-Interpreters verändern, werden als Beweisziele behandelt.

Laden Sie nun die Datei “`family.pl`”. Dies sollte mit einem der folgenden Befehle funktionieren:

- Falls Sie das Prolog-System im gleichen Verzeichnis gestartet haben:

  ```
  ['family.pl'].
  ```

  Vergessen Sie den Punkt “`.`” am Ende nicht, der in Prolog Fakten, Regeln und Beweisziele abschließt.

- Falls die Datei im aktuellen Verzeichnis steht und das Prolog-System mit der Default-Endung "`.pl`" konfiguriert wurde:

  ```
  [family].
  ```

  Probieren Sie bei Bedarf, ob es funktioniert, wenn Sie die Datei in "`family.pro`" umbenennen: "`.pl`" ist die klassische Endung für Prolog, leider aber auch für Perl. Man kann bei der Installation von SWI-Prolog auch die Endung "`.pro`" wählen. Auch diese Endung wird aber teils von anderen Anwendungen genutzt.

- Wenn Sie das Prolog-System unter Windows gestartet haben, ist sein aktuelles Verzeichnis das Installationsverzeichnis. Dann müssen Sie den vollen Pfad eingeben:

  ```
  ['C:/sb/lp24/family.pl'].
  ```

  Bei SWI-Prolog können Sie sich das aktuelle Verzeichnis mit folgender Anfrage anzeigen lassen (nicht im Prolog-Standard enthalten):

  ```
  working_directory(X, X).
  ```

  Das Prädikat hat zwei Argumente, weil man das aktuelle Verzeichnis damit auch ändern kann.

- Da die eckigen Klammern nur eine Abkürzung für den Aufruf des System-Prädikates "`consult`" sind, geht es auch so:

  ```
  consult('family.pl').
  ```

  Dies zeigt nochmal deutlich, dass es syntaktisch eine Anfrage ist, die ein vordefiniertes Prädikat aufruft.

Es ist kein Problem, wenn Sie die Datei mehrfach laden. Die vorherige Definition der Prädikate wird jeweils gelöscht. Es ist ganz typisch, dass Sie einen Editor in einem Fenster offen haben, dort das Programm entwicklen und jeweils nur abspeichern, und im anderen Fenster ein Prolog-System haben, und dort die veränderte Datei jeweils neu laden.

Falls Sie die Definition eines Prädikates zunächst aus Datei A geladen haben, und dann aus einer anderen Datei B laden, bekommen Sie eine Warnung ("Redefined static procedure"). Auch in diesem Fall wird die frühere Definition also gelöscht (die Standard-Eingabe "`user`" zählt dabei jedes Mal als neue Datei).

b) Die Relation `parent` entspricht folgender Tabelle:

parent	
Child	Parent
eric	alan
eric	bianca
fiona	chris
fiona	doris
george	chris
george	doris
ian	eric
ian	fiona
julia	eric
julia	fiona
ken	george
ken	helen

Probieren Sie die folgenden Anfragen aus:

- `parent(eric, X).`
  Wer sind die Eltern von Eric?
- `parent(X, fiona).`
  Wer sind die Kinder von Fiona?

Prolog unterscheidet sich von relationalen Datenbanken u.a. dadurch, dass jeweils nur eine Antwort auf einmal berechnet wird. Wenn es weitere Antworten geben könnte, bleibt der Cursor nach der Ausgabe am Ende der Zeile stehen und das System wartet auf Ihre Entscheidung:

- Sie können "Enter"/"Return" drücken, dann ist die Abfrage beendet und es werden keine weiteren Antworten gesucht.
- Drücken Sie dagegen "`;`" ("oder?"), so wird nach einer alternativen Lösung (Antwort) gesucht. Es ist möglich, dass keine gefunden wird. In diesem Fall wird "`false`" ausgegeben (oder "`no`").

Wenn das Prolog-System dagegen schon weiß, dass es keine weitere Antwort gibt, beendet es die Abfrage und zeigt wieder die Eingabeaufforderung "`?-`". (Dann sollten Sie nicht "`;`" eingeben, das wäre ein Syntaxfehler.)

c) Probieren Sie auch einige ja/nein-Abfragen und lassen Sie sich die ganze Relation `parent` ausgeben. (D.h. lassen Sie sich die ganze Extension des Prädikats "`parent`" drucken.)

d) Definieren Sie ein Prädikat "`siblings`" für Geschwister (d.h. `siblings(X,Y)` soll gelten, wenn `X` und `Y` den gleichen Vater und die gleiche Mutter haben). Sie können die Bedingung `X \= Y` im Rumpf verwenden, um `X` $\neq$ `Y` zu fordern.

Editoren unter Linux sind z.B. `gedit`, `gvim`, `emacs` (wenn Sie keinen kennen, verwenden Sie `gedit`).

Stellen Sie einige Anfragen, um die Definitionen zu testen.

Sie können auch unterschiedliche Formatierungen testen, um zu sehen, wie Sie die Regeln als besonders übersichtlich empfinden. Z.B. kann man jede atomare Formel auf eine eigene Zeile schreiben, und die Formeln im Rumpf (rechte Seite der Regel) einrücken. Sie können auch Kommentare einfügen (von "`%`" bis zum Ende der Zeile).

Probieren Sie auch die Reaktion auf mindestens einen Syntaxfehler aus.

e) Definieren Sie ein Prädikat "`uncle`": `uncle(X,Y)` soll gelten, wenn `Y` ein Bruder eines Elternteils von `X` ist.

Aufgabe 6 (Präsenzaufgabe)

Es soll nun eine Endlosschleife (Endlos-Rekursion) getestet werden. Definieren Sie dazu folgende Regel:

```
p :- p.
```

Sie müssen diese Regel nicht in eine Datei schreiben, sondern können mit

```
[user].
```

auch von der Standard-Eingabe lesen (beachten Sie die Eingabe-Aufforderung "`|:`"). Die Eingabe wird beendet mit `Ctrl-D` (Steuerung + D). So definierte Regeln gehen verloren, wenn das Prolog-System beendet wird (das ist in diesem Fall aber kein Problem, sondern sogar erwünscht). Stellen Sie nun die Anfrage

```
p.
```

Dazu müssen Sie den "Definitionsmodus" verlassen, und die Eingabeaufforderung muss wieder "`?-`" sein.
Bei Prolog sind Endlosschleifen möglich (im Gegensatz zu deduktiven Datenbanken). Die Ausführung dieser Anfrage endet nicht (freiwillig). Drücken Sie "`Ctrl+C`", um die Anfrage abzubrechen. Sie kommen dadurch in einen Unterbrechungsmodus, der verschiedene Möglichkeiten bietet, den Debugger zu nutzen. "`h`" zeigt verschiedene Optionen. Zunächst brauchen Sie nur "`a`" (für "abort"), um die Ausführung der Anfrage zu beenden.

Zum Selbstudium

Aufgabe 7 (Zusatzaufgabe)

Schauen Sie sich die folgenden Webseiten an. Sie brauchen nicht alles zu verstehen, sondern sich nur einen ersten Überblick verschaffen, was es gibt. Es ist geplant, dass auf jedem Übungsblatt einige Quellen genannt werden, die für Sie interessant sein könnten.

- „Prolog“ in der Wikipedia:

  [https://de.wikipedia.org/wiki/Prolog_(Programmiersprache)]

- „Learn Prolog Now!“:

  [http://www.learnprolognow.org/].

  Schauen Sie sich insbesondere Abschnitt 1.1 „Some Simple Examples“ der „Free Online Version“ an. Es gibt auch eine Version mit Auswertungsmöglichkeit:

  [http://lpn.swi-prolog.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse1]

- Es empfiehlt sich, ein Prolog-System auf dem eigenen Rechner zu installieren. Es gibt viele freie Prolog-Systeme, die sich mehr oder weniger an den Prolog-Standard halten. Ein bekanntes und problemloses System ist:

  [http://www.swi-prolog.org/]

  Im Laufe des Semesters werden wir uns auch andere Prolog-Systeme anschauen, die vielleicht eine höhere Performance oder mehr Möglichkeiten im Bereich „Constraint Logic Programming“ bieten. Aber für den Einstieg ist SWI-Prolog sicher eine gute Option. Falls Sie auf Ihrem Rechner nichts installieren wollen, können Sie auch die Online-Version testen:

  [http://swish.swi-prolog.org/]

Sie brauchen für diese Aufgabe nichts abzugeben, aber sollten schon einige Zeit investieren (es sind ja zwei Stunden pro Woche für Hausaufgaben eingeplant, und die Aufgaben a) bis c) sollten schnell erledigt sein). Notieren Sie sich auch, was Sie in der Selbstudiums-Woche in der vorlesungsfreien Zeit nochmals anschauen wollen.