Logische Programmierung & deduktive Datenbanken

— Übungsblatt 7 (Einfacher Typ-Prüfer) —

Ihre Lösung zu Hausaufgabe 1 geben Sie bitte über die Übungs-Plattform in StudIP ab. Einsendeschluss ist Mittwoch, der 5. Juni, 16⁰⁰ (die Aufgaben werden in der Übung am Donnerstag besprochen).

Hausaufgaben

Aufgabe 1 (Hausaufgabe, 8 Punkte)

Schreiben Sie einen einfachen Typ-Prüfer für Prolog/Datalog-Regeln in Prolog. Zur Vereinfachung sei angenommen, dass die Prädikate der zu prüfenden Regeln nur ganze Zahlen und Atome als Argumente haben, keine strukturierten Terme. Die Prädikate, die in diesen Regeln verwendet werden dürfen, seien im Typprüfer selbst deklariert, und zwar in folgender Form:

```
% komponist(KNR, NAME, VORNAME, GEBOREN, GESTORBEN)
pred(komponist(int, atom, atom, int, int)).

% stueck(SNR, KNR→komponist, TITEL, TONART, OPUS).
pred(stueck(int, int, atom, atom, atom)).

pred(<(int, int)).
pred(ergebnis(int, int)).
...
```

D.h. die verwendbaren Prädikate sind (vorläufig) in den Typprüfer fest eingebaut. Programmieren Sie ein Prädikat `check/1`, so dass man auf folgende Art die Typprüfung durchführen kann:

```
read(X), check(X).
```

Die Eingabe `X` kann dabei eine Regel, ein einzelnes Literal, oder eine Konjunktion von Literalen (Anfrage, Beweisziel) sein. Die syntaktische Richtigkeit können Sie voraussetzen. Das eingebaute Prädikat `read` liefert die Regel/Anfrage auch bereits als Term (Baumstruktur). Wenn Sie `:-` und `,` als normale Funktoren verwenden wollen, müssen Sie diese Symbole in `'...'` einschließen, z.B. `':-'`.

Es reicht aus, wenn Ihr Prädikat `check` fehlschlägt, falls die Eingabe einen Typfehler enthält. Man bekommt in diesem Fall also die Antwort “no” und sonst “yes”. Falls Sie wollen, können Sie natürlich auch bessere Fehlermeldungen ausgeben (mit `write` und `nl`), bevor Ihr Prädikat fehlschlägt (ggf. hierzu `fail` verwenden). Gute Fehlermeldungen verkomplizieren die Aufgabe allerdings deutlich.

Zum Beispiel sollte die Eingabe

```
ergebnis(X, Y) :-
    komponist(_, 'Wolfgang Amadeus', 'Mozart', X, Y).
```

als korrekt gewertet werden, die folgenden aber alle nicht:

- `ergebnis(X, Y) :- komponist(X, Y, Y, 'Wolfgang Amadeus', 'Mozart')`.
Atom-Konstanten an Integer-Argumentpositionen.
- `ergebnis(X1, X2) :- komponist(Nr, X1, X2, Y1, Y2)`.
`X1` und `X2` müssten nach dem Rumpf-Literal Atome sein, aber nach dem Kopf-Literal ganze Zahlen.
- `ergebnis(X, X) :- komponist(Nr, 'Wolfgang Amadeus', 'Mozart', X)`.
Es gibt kein Prädikat `komponist` mit nur vier Argumenten.

Die Bereichsbeschränkung brauchen Sie nicht zu prüfen, d.h. es soll legal sein, Variablen im Kopf zu verwenden, die im Rumpf nicht vorkommen.

Sie können diese Aufgabe in drei Schritten lösen:

- Überführen Sie die Eingabe in eine Liste von Literalen.
- Ersetzen Sie die Konstanten in diesen Literalen durch die Typbezeichner `atom` und `int`. Den Test auf die Typen können Sie mit den eingebauten Prädikaten `atom/1`, `integer/1` und `var/1` durchführen. Um Literale in Prädikat und Argumentliste zu zerlegen, können Sie das Prädikat `=..` verwenden. Damit können Sie auch umgekehrt wieder Literale erzeugen.
- Nun müssen Sie nur noch jedes Literal Ihrer Liste mit den gegebenen `pred`-Fakten unifizieren. Dafür ist wichtig, dass Sie im zweiten Schritt die Variablen aus der Eingabe erhalten, so dass eine inkonsistente Verwendung der gleichen Variable mit unterschiedlichen Typen auch bemerkt wird.

Freiwillige Zusatzaufgabe: Lesen Sie die `pred`-Fakten und die zu prüfenden Regeln aus einer Datei. Definieren Sie also ein Prädikat `check_file/1`, dem man den Namen der zu prüfenden Datei mit Prädikat-Deklarationen und Programm-Regeln übergibt.

Sie können die `pred`-Fakten mit `assert` in die dynamische Datenbank laden (brauchen dann aber eine `dynamic`-Deklaration). So kann das obige `check`-Prädikat unverändert weiter verwendet werden.

Zur Wiederholung

Aufgabe 2 (Wiederholungs-Fragen)

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

- a) Was ist das Symbol in Prolog für den Cut?
- b) Was genau macht der Cut?
- c) Geben Sie ein kurzes Beispiel für ein logisches Programm, an dem man die Wirkungsweise des Cut erläutern bzw. ausprobieren kann.
- d) Wozu ist der Cut gut? Erläutern Sie einige typische Anwendungen des Cut. Inwieweit kann er zur Verbesserung der Performance eines Prolog-Programms eingesetzt werden? Was ist eine typische Anwendung für Fallunterscheidungen?
- e) Warum ist der Cut problematisch?
- f) Geben Sie ein Beispiel für eine Prädikatdefinition, in der der Cut einen relativ subtilen Fehler verursacht (d.h. einen Fehler, den der Programmierer nicht oder nicht sofort bemerkt, und der deswegen besonders schwierig/gefährlich ist).
- g) Welche anderen Prolog-Operatoren bzw. Prädikate kann man oft anstelle des Cut verwenden?
- h) Erläutern Sie das „if-then-else“ in Prolog.

Aufgabe 3 (Wiederholungs-Fragen)

Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zum Vergleich mit klassischen (imperativen) Programmiersprachen antworten?

- a) 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?
- b) 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?
- c) Was leistet die Endrekursions-Optimierung und wann ist sie anwendbar?

Übungsaufgaben (gemeinsam in der Übung zu bearbeiten)

Aufgabe 4 (Präsenzaufgabe)

Zeichnen Sie einen SLD-Baum für folgenden Programm und die Anfrage $p(X)$ (nutzen Sie dabei die „First Literal“ Selektionsfunktion wie in Prolog):

```
p(X) :- q(X), r(X).  
q(a) :- s(Y), t(Y).  
q(b).  
r(a).  
r(b).  
s(c).  
s(d).  
t(c).
```

Aufgabe 5 (Präsenzaufgabe)

Definieren Sie ein Prädikat `calc`, das eine Eingabeaufforderung („Prompt“) ausdrückt, einen arithmetischen Ausdruck einliest (ohne Variablen), den Wert dieses Ausdrucks berechnet, und das Ergebnis ausgibt. Dies soll in einer Schleife wiederholt werden, bis der Benutzer „quit“ eingibt. Sie können voraussetzen, dass der Nutzer jede Zeile mit „.“ abschliesst. Syntaxfehler brauchen Sie nicht zu behandeln.

Aufgabe 6 (Präsenzaufgabe)

Definieren Sie ein Prädikat `id_next(N)` zur Generierung eindeutiger Zahlen (mit Hilfe der dynamischen Datenbank). Der erste Aufruf von `id_next(N)` soll `N` an 1 binden, der zweite an 2, u.s.w. Jeder Aufruf von `id_next(N)` soll nur ein Mal erfolgreich sein, d.h. beim Backtracken sollen keine weiteren Lösungen generiert werden, nur wenn das Prädikat erneut „vorwärts“ betreten wird.

Sie speichern sich den aktuellen Zahlwert am besten als Fakt `id_value(N)` in einem dynamischen Prädikat. Dazu müssen Sie `id_value/1` als „dynamic“ deklarieren. Schreiben Sie sich dann ein Prädikat `id_init`, das eventuell vorhandene Fakten `id_value(_)` löscht und ein Fakt `id_value(1)` in die dynamische Datenbank speichert. Dieses Prädikat können Sie beim Laden Ihres Programms automatisch aufrufen.

Das Prädikat `id_next(N)` fragt dann den aktuellen Zahlwert mit `id_value(N)` ab, löscht das Fakt, und fügt ein Fakt mit einer um eins größeren Zahl wieder ein.

Zum Selbststudium

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) 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>]

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

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

- c) 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/>]

- d) 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]

- e) 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/>]