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

17.05.2018

Logische Programmierung & deduktive Datenbanken — Übungsblatt 6 (Unifikation) —

Ihre Lösungen zu den Hausaufgaben i) bis m) schicken Sie bitte per EMail an den jeweiligen Übungsleiter (mit "`[lp18]`" in der Betreff-Zeile). Einsendeschluss ist der 26. Mai.

Zum Selbststudium

a) 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 WWW zum Thema dieser Vorlesung gibt, und dabei für sich nützliche Quellen entdecken.

- Die Unterlagen zu meinem Prolog-Programmierkurs aus dem Wintersemester 1993/94 an der Universität Hannover (auf Deutsch) finden Sie hier:

  [http://users.informatik.uni-halle.de/~brass/lp93/]

- Die Webseite der Vorlesung "Problem Solving and Search" von Ulle Endriss an der Universität Amsterdam finden Sie hier:

  [https://staff.science.uva.nl/u.endriss/teaching/pss/]

  Sie enthält als Werbung für die Vorlesung eine Reihe von Problemen. U.a. wird auch auf ein kurzes Video mit Martin Freeman verwiesen mit dem Rätsel vom Wolf, der Ziege und dem Kohlkopf in zwei Varianten:

  [https://www.youtube.com/watch?v=b-s8uKXdaYk]

  Interessanter für diese Vorlesung sind natürlich die Unterlagen zum Prolog-Kurs:

  [https://staff.science.uva.nl/u.endriss/teaching/pss/prolog.pdf]

- Schauen Sie auch mal in die SWI-Prolog FAQ unter "Reading about Prolog". Dort findet sich eine Liste von Links zu weiteren Prolog-Kursen.

  [http://eu.swi-prolog.org/FAQ/PrologReading.html]

- Den ISO-Standard zu Prolog gibt es online unter

  [http://www.deransart.fr/prolog/]

  Zum Beispiel finden Sie die Spezifikation des eingebauten Prädikates `op` auf folgender Seite:

  [http://www.deransart.fr/prolog/bips.html#operators]

- Stilvorschläge für die Prolog-Programmierung von Michael Covington finden Sie unter

  [http://www.covingtoninnovations.com/mc/plcoding.pdf]

  U.a. darauf basierend wurden die folgenden Stilempfehlungen entwickelt:

  [https://lifeware.inria.fr/~soliman/post/prolog_guidelines/]

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

- Definieren Sie den Begriff "Substitution"?
- Was ist das Ergebnis der Anwendung der Substitution $\{X/Y,\ Y/a\}$ angewendet auf $p(X, Y, Z, b)$?
- Definieren Sie den Begriff "Unifikator" und "Allgemeinster Unifikator" ("most general unifier", mgu).
- Ist der allgemeinste Unifikator zweier Literale immer eindeutig bestimmt?
- Was ist ein allgemeinster Unifikator von $p(X, a)$ und $p(Y, Z)$? Was wäre ein Beispiel für einen nicht allgemeinsten Unifikator?
- Berechnen Sie einen allgemeinsten Unifikator von $p(X, X)$ und $p(f(Y), f(a))$.
- Wie spielen Unifikatoren für Prolog eine Rolle? Wo werden sie gebraucht? Wie kann man sie mit Prolog berechnen?
- Was unterscheidet die Unifikation von Zuweisungen in klassischen Programmiersprachen?
- Was ist der "Occur Check"? Warum ist er für die Unifikation wichtig? Welches Problem gibt es? Warum lassen viele Prolog-Systeme ihn weg? Gibt es noch alternative Lösungen?
- Wie kann man testen, ob ein Prolog-System den Occur-Check verwendet? Geben Sie ein einfaches Beispiel-Programm an.
- Was ist die Idee der Resolventenmethode (Resolutions-Verfahren) zum automatischen Beweisen? Skizzieren Sie den Ableitungsschritt mit Klauseln als Disjunktionen von Literalen.
- Wofür stehen die drei Buchstaben "S", "L", "D" in der SLD-resolution?
- Definieren Sie einen SLD-Ableitungsschritt.

Präsenzaufgaben

c) Prüfen Sie, ob die folgenden Literale unifizierbar sind und geben Sie ggf. einen allgemeinsten Unifikator an:

- `p(X,a)` und `p(Y,f(Z))`.
- `p(f(X), Z)` und `p(Y, a)`.
- `p(f(a), X)` und `p(Y, Y)`.
- `p(f(X), Z)` und `p(Y, Y)`.
- `append(X, [2,3], [1,2,4])` und `append([F|R], L, [F|A])`.
- `q(X, Y, h(g(X)))` und `q(Z, h(Z), h(Z))`.
- `p([a,b|L])` und `p([X|Y])`.

d) Schreiben Sie ein Prädikat Fläche, das die Fläche verschiedener geometrischer Objekte berechnet. Folgende Aufrufe sollen möglich sein:

- `flaeche(rechteck(5,10), X)`: Liefert `X = 50`.
  Die Fläche eines Rechtecks ist das Produkt der Kantenlängen. Diese seien als Argumente des Funktors `rechteck` gespeichert.
- `flaeche(quadrat(5), X)`: Liefert `X = 25`.
  Ein Quadrat ist ein Rechteck mit gleich langen Seiten, deswegen braucht dieser Funktor nur ein Argument.
- `flaeche(kreis(3), X)`: Liefert `X = 28.274333882308138`.
  Der im `kreis`-Term gespeicherte Wert sei der Radius r des Kreises. Die Fläche eines Kreises ist $\pi * r^2$. SWI-Prolog versteht die Konstante `pi` in den arithmetischen Ausdrücken, die mit `is` ausgewertet werden. Sie können aber auch einfach `3.14159` einsetzen.

e) Schreiben Sie ein Prädikat `distinct(X,Y)`, das Duplikate aus der Liste `X` entfernt und das Ergebnis in `Y` liefert. `Y` muss also duplikatfrei sein und die gleichen Elemente wie `X` haben. Die Reihenfolge der Elemente in `Y` ist nicht vorgeschrieben. Z.B. sollte `distinct([1,2,1],Y)` die Variable `Y` an `[1,2]` oder aber `[2,1]` binden. Sie dürfen das Prädikat `member` zum Elementtest verwenden.

f) Mit dem Aufruf `findall(X,p(X),L)` können Sie `L` an die Liste aller `X` binden, für die `p(X)` gilt. Laden Sie nochmals die CD-Datenbank von Blatt 2:

[http://www.informatik.uni-halle.de/˜brass/lp18/cd.pl]

Im Prädikat "`stueck(`<u>`SNR`</u>`, KNR`→`komponist, TITEL, TONART, OPUS)`" steht auch die Tonart des Stückes. Nutzen Sie `findall` und Ihr Prädikat `distinct`, um eine Liste aller Tonarten zu finden, die in den Stücken der Datenbank vorkommen.

g) Es sei nochmals die effizientere Berechnung der Listen-Länge betrachtet.

```
len(List, Length) :-
    len(List, 0, Length).
len([], Length, Length).
len([_|Rest], LengthIn, LengthOut) :-
    Length is LengthIn + 1,
    len(Rest, Length, LengthOut).
```

Normalerwise gibt man als Kommentar zu einem Prädikat eine logische Bedingung an, die genau dann gilt, wenn das Prädikat wahr ist:

```
% len(L, N) <=> N ist die Laenge (Anzahl Elemente) der Liste L
```

Wie würde der Kommentar für das Hilfsprädikat `len(List, LengthIn, LengthOut)` lauten? Die Regeln müssen bezüglich dieser Bedingung natürlich korrekt sein.

h) Schreiben Sie ein Prädikat `indent(I)`, das `(I+1)*4` Leerzeichen ausgibt. Verwenden Sie dieses, um in die obige Definition von `len/3` Ausgaben jeweils am Anfang und am Ende des Regelrumpfes einzubauen, die im Prinzip dem Debugger entsprechen:

```
len(List, Length) :-
    write('Start: '),
    write(len(List, Length)),
    nl,
    len(List, 0, Length)
    write('End: '),
    write(len(List, Length)),
    nl.
len([], Length, Length) :-
    indent(Length),
    write('Done: '),
    write(len([], Length, Length)),
    nl.
len([E|Rest], LengthIn, LengthOut) :-
    indent(LengthIn),
    write('Call: '),
    write(len([E|Rest], LengthIn, LengthOut)),
    nl,
    Length is LengthIn + 1,
    len(Rest, Length, LengthOut),
    indent(LengthIn),
    write('Exit: '),
    write(len([E|Rest], LengthIn, LengthOut)),
    nl.
```

Es wird der Regelkopf jeweils am Anfang der Ausführung des Rumpfes und am Ende ausgegeben. Durch die Einrückung mit `indent` können Sie die rekursiven Aufrufe besser verfolgen.

Sie können mit dieser Technik natürlich auch die Ausführung anderer Prädikate visualisieren (z.B. `distinct`). Der Programmcode wird etwas übersichtlicher, wenn Sie Hilfsprädikate wie `print_call(...)` definieren, so dass Sie nur jeweils nur eine Zeile statt vier benötigen. Beim Beispiel-Prädikat gibt `LengthIn` gerade die passende Einrückung an, im allgemeinen müssen Sie das selbst verwalten. Im Moment benötigen Sie dazu eine zusätzliche Variable für die Aufruf-Tiefe, die Sie auch als Argument an das zu visualisierende Prädikat übergeben müssen. Später können wir mit der dynamischen Datenbank etwas basteln, das wie eine globale Variable wirkt (siehe `dynamic`, `assert`, `retract`).

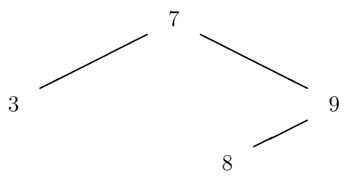
Beachten Sie, dass am Anfang des Regelrumpfes und am Ende `write` das gleiche Argument hat (den Kopf der Regel), aber in der Zwischenzeit Variablen gebunden wurden, so dass die Ausgabe anders aussieht (mit Werten statt Variablen — es wurde eine Substitution angewendet).

Hausaufgabe

i) Es sollen binäre Suchbäume als Terme der Form `node(Value, Left, Right)` dargestellt werden. Dabei ist `Left` der linke Teilbaum, der nur Werte echt kleiner als `Value` enthält, und `Right` ist entsprechend der rechte Teilbaum. Für den leeren Baum soll die Konstante `nil` verwendet werden. Zum Beispiel entspricht der Term

```
node(7, node(3, nil, nil), node(9, node(8, nil, nil), nil))
```

dem Baum



Schreiben Sie ein Prädikat `in(E, T)` zur Suche in solchen Bäumen. Es soll genau dann wahr sein, wenn `E` ein Wert ist, der in `T` vorkommt. Sie können voraussetzen, dass `T` der obigen Struktur entspricht.

j) Schreiben Sie ein Prädikat `insert(E, TIn, TOut)`, das ein Element `E` in den Suchbaum `TIn` korrekt einfügt, und den Ergebnisbaum in `TOut` liefert. Wird ein Element

eingefügt, das schon im Baum enthalten ist, soll die EInfügung ignoriert werden, d.h. der Aufruf soll erfolgreich sein, aber einen unveränderten Baum liefern.

k) Schreiben Sie ein Prädikat `inorder(T, L)`, das alle Werte aus dem Suchbaum `T` in einer Liste `L` liefert, die dem "inorder"-Durchlauf entspricht: Erst die Werte im linken Teilbaum, dann der Wert im aktuellen Knoten, und dann die Werte im rechten Teilbaum. Die Liste `L` enthält die Werte dann in sortierter Reihenfolge.

l) Schreiben Sie ein Prädikat `mysort2(L,S)`, das eine Liste `L` sortiert, indem es zunächst alle Elemente in einen Suchbaum einfügt und dann einen Inorder-Durchlauf macht. Selbstverständlich können Sie sich beliebige Hilfsprädikate definieren. Z.B. wäre ein Prädikat `insertList(L,T)`, das einen Baum `T` durch Einfügen aller Elemente der Liste `L` erstellt, vermutlich nützlich.

m) Testen Sie, ob Ihr Element-Test `in(E,T)` auch zum Aufbauen von Bäumen verwendet werden kann. Was ist das Ergebnis von

```
in(7,X), in(3,X), in(9,X)?
```

Wenn alles funktioniert, enthält `X` anschließend einen Baum-Term mit Variablen anstelle der `nil`-Konstanten:

```
node(7, node(3,X1,X2), node(9,X3,X4)).
```

Beachten Sie, dass Sie mit dieser Technik in den Baum einfügen können, ohne Teile des Baums zu kopieren.

Wenn Sie wollen (dieser Teil ist freiwillig) könnten Sie für derartige Baumstrukturen noch einen echten Element-Test `contains(E,T)` schreiben, bei dem der rekursive Abstieg dann bei Variablen aufhört. Mit dem Prädikat `var(X)` können Sie testen, ob `X` eine ungebundene Variable ist, und entsprechend mit `nonvar(X)`, ob es irgendein anderer Term ist.