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

16.05.2024

Logische Programmierung & deduktive Datenbanken — Übungsblatt 5 (Suchbaum) —

Ihre Lösungen zu den Hausaufgaben (Aufgabe 1 bis 5) geben Sie bitte über die Übungs-Plattform in StudIP ab. Einsendeschluss ist Mittwoch, der 22. Mai, 16^{00} (die Aufgaben werden in der Übung am Donnerstag besprochen).

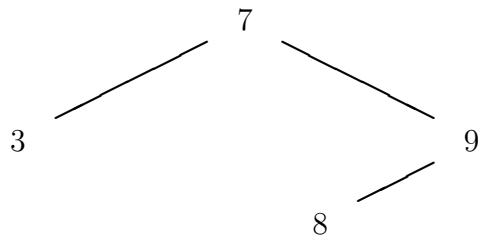
Hausaufgaben

Aufgabe 1 (Hausaufgabe, 2 Punkte)

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.

Aufgabe 2 (Hausaufgabe, 2 Punkte)

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.

Aufgabe 3 (Hausaufgabe, 2 Punkte)

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.

Aufgabe 4 (Hausaufgabe, 2 Punkte)

Schreiben Sie ein Prädikat `tisort(L,S)` („tree insertion sort"), 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. Wie bei Aufgabe 2 sollen Duplikate eliminiert werden.

Aufgabe 5 (Hausaufgabe, 2 Punkte)

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.

Zur Wiederholung

Aufgabe 6 (Wiederholungs-Fragen)

Was würden Sie in einer mündlichen Prüfung auf die folgenden Fragen zur SLD-Resolution antworten?

a) Was ist die Idee der Resolventenmethode (Resolutions-Verfahren) zum automatischen Beweisen? Skizzieren Sie den Ableitungsschritt mit Klauseln als Disjunktionen von Literalen.

b) Wofür stehen die drei Buchstaben „S“, „L“, „D“ in der SLD-resolution?

c) Definieren Sie einen SLD-Ableitungsschritt.

d) Welche Selektionsfunktion wird von Prolog benutzt? Warum vereinfacht diese Selektionsfunktion die Implementierung?

e) Definieren Sie „erfolgreiche SLD-Ableitung“. Was ist die dabei berechnete Antwort?

f) Wie lautet der Satz über die Korrektheit der SLD-Resolution? Gilt die Korrektheit auch für Prolog?

g) Wie lautet der Satz über die Vollständigkeit der SLD-Resolution? Gilt dies auch für Prolog?

h) Wie wird der SLD-Ableitungsbaum konstruiert? Was ist die Ursache der Verzweigungen im SLD-Ableitungsbaum?

i) Wie sucht Prolog den SLD-Ableitungsbaum ab? Was ist das Problem dabei?

j) Zeichnen Sie den SLD-Ableitungsbaum für ein Beispiel auf.

k) Wie heißen die vier Ports im „Box Model“ des Prolog-Debuggers? An welchen Stellen im Kasten sind sie, und welchen Vorteil hat diese Anordnung?

l) Mit welchem Kommando bekommt man in einen Prolog-System die Ausgaben zur Programm-Ausführung nach dem Vier-Port-Modell? Wie kann man sie wieder abschalten? Wie heißen „Breakpoints“ in Prolog?

m) Was unterscheidet Prolog-Prädikate von Prozeduren in klassischer Programmierung?

Übungsaufgaben (gemeinsam in der Übung zu bearbeiten)

Aufgabe 7 (Präsenzaufgabe)

Definieren Sie ein Prädikat `last(L, E)`, das wahr ist, wenn `E` das letzte Element der Liste `L` ist. Z.B. sollte `last([1,2,3], X)` die Antwort `X=3` liefern. Für die leere Liste soll das Prädikat fehlschlagen.

Aufgabe 8 (Präsenzaufgabe)

a) Definieren Sie ein Prädikat zur Bestimmung der Listen-Länge wie folgt:

```
len([], 0).
len([_|Rest], Len) :-
    len(Rest, RestLen),
    Len is RestLen + 1.
```

Testen Sie das Prädikat auf mehreren Eingaben. Sie können auch Testausgaben mit `write` einbauen, um die rekursive Ausführung zu verfolgen.

Oder Sie starten den Prolog-Debugger mit „`trace.`“ Es wird jeder Aufruf mit „`CALL:`“ angezeigt, und jede erfolgreiche Rückkehr aus einem Prädikat mit „`EXIT:`“. Wenn ein Aufruf fehlschlägt, wird „`FAIL:`“ angezeigt. Hat ein Prädikat möglicherweise noch weitere Lösungen, wird die Rückkehr zu einem Backtrack-Punkt mit „`REDO:`“ angezeigt. Nach jeder Zeile müssen Sie „Return“/„Enter“ (oder auch die Leertaste) drücken, damit die Ausführung fortgesetzt wird. Mit „`?`“ werden die Debugger-Kommandos angezeigt. Z.B. können Sie mit „`s`“ (skip) einen Aufruf ohne Ausgaben ausführen und mit „`a`“ (abort) die Ausführung beenden. Mit „`nodebug.`“ schalten Sie den Debugger wieder aus.

b) Das obige Prädikat hat den Nachteil, dass die zweite Regel nicht endrekursiv ist: Nach Rückkehr aus der Rekursion muss noch der `is`-Aufruf ausgeführt werden. Deswegen kann der „Stackframe“ eines Aufrufs nicht für die rekursiven Aufruf genutzt werden, d.h. die Rekursion kann nicht intern in eine Iteration überführt werden. Bei folgender Variante mit einem Hilfsargument wäre das dagegen möglich:

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

Man sagt, dass die Variablen bzw. Argumente „`LengthIn`“ und „`LengthOut`“ ein „Accumulator Pair“ bilden. In der Rekursion gibt es ein Eingabe-Argument für den aktuellen Wert einer Variable. Außerdem wird eine Variable für den Ergebniswert über

alle Rekursionsebenen hinweg einfach durchgereicht. Wenn das Ende der Rekursion erreicht ist, wird die Variable für den Ergebniswert an den dann aktuellen Eingabewert gebunden.

Mit dieser Technik wurde hier die Linksrekursion in eine Rechtsrekursion überführt. Da der rekursive Aufruf das letzte Literal der letzten Regel über das Prädikat `len` ist, und andere Rumpfliterale (im Beispiel `is`) keine Backtrackpunkte zurückgelassen haben, greift hier die Endrekursionsoptimierung.

Falls Sie Zeitmessungen machen wollen, können Sie bei SWI-Prolog mit

```
statistics(cputime, X)
```

die verbrauchte CPU-Zeit seit dem Start abfragen (es gibt noch weitere Leistungsparameter, die Sie so abfragen können, siehe Handbuch). Die Listen müssten aber schon sehr lang sein, damit Sie einen Unterschied bemerken. Mit dem Prädikat aus Teilaufgabe i) können Sie sich eine lange Liste generieren.

Alternativ können Sie auch mit `time(Anfrage)` eine Anfrage ausführen, und die verbrauchte Zeit dafür ausgeben lassen.

c) Schreiben Sie ein Prädikat `int_list(N, L)` das zu vorgebener Länge `N` eine Liste mit den Zahlen von `1` bis `N` generiert. Zuerst ist es einfacher, die Zahlen in umgekehrter Reihenfolge zu liefern. Z.B. sollte `int_list(3, L)` die Antwort `L = [3,2,1]` haben.

Versuchen Sie dann noch eine Variante zu erstellen, die die Zahlen in aufsteigender Reihenfolge liefert. Eine Lösung mit `append` ist auch einfach. Diese hat aber den Nachteil, dass die Liste in jedem Schritt kopiert wird, um hinten noch ein Element anzuhängen. Wenn Sie ein zusätzliches Argument für den Startwert vorsehen, können Sie ohne `append` auskommen.

d) Schreiben Sie ein Prädikat `rev(L, R)` zum Umdrehen/Spiegeln einer Liste. Z.B. soll `rev([1,2,3], X)` die Antwort `X = [3,2,1]` liefern. Auch hier ist eine Lösung mit `append` wieder einfach, hat aber den gleichen Nachteil, dass die Liste in jedem Schritt kopiert wird. Versuchen Sie, mit der „Accumulator Pair“-Methode eine Lösung ohne `append` zu entwickeln. (Hinweis: Sie können z.B. ein Prädikat `rev_aux(L,X,R)` definieren mit `R = concat(reverse(L),X)`, d.h. eine zusätzliche Liste `X` wird hinten an die umgedrehte Liste angehängt. `X` ist der Anfang des Ergebnisses.)

Zum Selbstudium

Aufgabe 9 (Zusatzaufgabe)

k) 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 Folien zur Vorlesung „Advanced Logic Programming“ von Dr. Günter Kniesel (Universität Bonn) gibt es hier:

  [http://sewiki.iai.uni-bonn.de/teaching/lectures/alp/2017/slides]

- Hier ist ein „Web-Schulbuch“ für den Informatik-Unterricht an Gymnasien mit einem Abschnitt über logische Programmierung:

  [https://www.inf-schule.de/deklarativ/logischeprogrammierung]

- 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]