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

05.05.2017

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

Ihre Lösungen zu den Hausaufgaben i) bis k) schicken Sie bitte per EMail an den Dozenten (mit "`[lp17]`" in der Betreff-Zeile). Einsendeschluss ist der 15. Mai (die Aufgaben werden in den Übungen am 16. Mai und 18. Mai besprochen). Aufgabe b) wird dort auch besprochen: Sie sollten vorher darüber nachdenken, müssen aber nichts abgeben.

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.

- Zur Lehrveranstaltung "Einführen in das Programmieren: Prolog" von Dr. Gunter Grieser (TU Darmstadt) gibt es Folien und Übungsaufgaben im Internet:

  [http://www.ke.tu-darmstadt.de/lehre/archiv/ss06/prolog]

  Insbesondere gibt es hier auch ein Kapitel über Rekursion und Listen.

  [http://www.ke.tu-darmstadt.de/lehre/archiv/ss06/prolog/teil4-listen.pdf]

- Zur Vorlesung "Programmiertechniken in der Computerlingusitik" von Sascha Brawer (gehalten an der Universität Zürich als Lehrauftrag) gibt es hier Folien:

  [http://www.brawer.ch/prolog/]

  Das Kapitel über Listen ist:

  [http://www.brawer.ch/prolog/listen.pdf]

- Wenn Sie Video-Tutorials mögen, könnten Sie z.B. das folgende Tutorial zur Listen-Notation von "EducationAboutStuff" probieren (etwas langatmig):

  [https://www.youtube.com/watch?v=Jrf6CKc2c0k]

  Hier ist ein anderes Tutorial (von "mycurlycode"):

  [https://www.youtube.com/watch?v=WKoM4L74XzY]

  Das folgende Video (von "bSimple") enthält eine Lösung zu Übung i):

  [https://www.youtube.com/watch?v=PUyccUj405k]

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

- Was wird `write(+(1,2))` ausgeben?
- Was wird `display(1+2)` ausgeben?
- Wird die Anfrage `1+1 = 2` mit `true` oder `false` beantwortet?
- Mit welchem Funktionssymbol und welcher Konstanten werden Listen in Prolog intern repräsentiert?
- Geben Sie drei verschiedene Schreibweisen für die Liste "`[1,2,3]`" in Prolog an.
- In welchen Prolog-Term wird Notation `[F|R]` intern übersetzt? Wofür stehen `F` und `R` hier?
- Wie wird das Prädikat `append` in Prolog definiert? Geben Sie die Regeln an.
- Funktioniert `append` auch zum Aufspalten einer Liste?
- Definieren Sie ein Prädikat `member(X, L)` zum Test, ob `X` ein Element der Liste `L` ist.
- Definieren Sie ein Prädikat `length(L, N)`, das die Länge `N` einer Liste `L` berechnet.
- Was muss man in Prolog beachten, damit rekursiv definierte Prädikate terminieren?
- Wie sehen Regeln in Prolog aus? Definieren Sie die Begriffe "Regel", "Kopf einer Regel", "Rumpf einer Regel", "Kopfliteral einer Regel", "Rumpfliteral einer Regel", "Regel über ein Prädikat p".

Präsenzaufgaben

c) Definieren Sie ein Prädikat `last(L, E)`, das wahr ist, wenn `E` an 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.

d) Definieren Sie ein Prädikat `ablist(L)`, das wahr ist, wenn `L` eine Liste nur aus den Elementen `a` und `b` ist. Z.B. soll `ablist([a,a,b,a])` wahr sein, aber `ablist([a,b,c,a])` falsch sein.

e) 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.

f) 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 der folgenden Teilaufgabe können Sie sich eine lange Liste generieren.

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

g) 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.

h) 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, ob Sie mit der “Accumulator Pair”-Methode nicht eine Lösung ohne `append` entwickeln können.

Hausaufgabe

i) Schreiben Sie ein Prädikat `sum(L, S)`, das die Zahlen in einer Liste `L` aufsummiert. Z.B. soll der Aufruf `sum([1,2,3], S)` die Lösung `S = 6` liefern. Mit `number(E)` können Sie testen, ob ein Element `E` der Liste eine Zahl ist. Ihr Prädikat soll fehlschlagen (false liefern), wenn ein Element der Liste keine Zahl ist.

j) Schreiben Sie ein Prädikat `insert(E, LIn, LOut)`, das ein Element `E` in die sortierte Liste `LIn` an die richtige Stelle einfügt. Z.B. soll `insert(3,[1,4,5],X)` das Ergebnis `X = [1,3,4,5]` liefern. Sie können davon ausgehen, dass `LIn` nur Zahlen enthält und `E` eine Zahl ist. Falls `E` schon in `LIn` vorkommt, soll es nicht nochmals eingefügt werden.

k) Schreiben Sie ein Prädikat `mysort(LIn, LOut)`, das die Liste `LIn` sortiert, indem es die Elemente mittels `insert` aus AUfgabe j) nacheinander in eine anfangs leere Liste einfügt. Duplikate werden dabei eliminiert.