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

26.04.2018

Logische Programmierung & deduktive Datenbanken — Übungsblatt 4 (Terme, Operatorsyntax) —

Ihre Lösungen zu den Hausaufgaben f) bis h) schicken Sie bitte per EMail an den jeweiligen Übungsleiter (mit "`[lp18]`" in der Betreff-Zeile). Einsendeschluss ist der 5. Mai. Aufgabe b) wird in den Übungen besprochen: Sie sollten vorher darüber nachdenken, müssen aber nichts abgeben.

Zum Selbstudium

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.

- Tutorial zu Prolog von John R. Fisher "`prolog :- tutorial`":

  [https://www.cpp.edu/~jrfisher/www/prolog_tutorial/]

  Version mit Frames:

  [https://www.cpp.edu/~jrfisher/www/prolog_tutorial/pt_framer.html]

- Eine Online-Version des Buches "Adventure in Prolog" von Dennis Merritt gibt es hier:

  [http://www.amzi.com/AdventureInProlog/apreface.php]

  Das Buch ist 1990 im Springer-Verlag erschienen. Es ist ein Prolog-Tutorial, das ein Textadventure-Spiel als durchgehendes Beispiel verwendet. Die Folien zu einem Prolog-Kurs von Dennis Merritt finden Sie hier:

  [https://github.com/AmziLS/prolog_course/blob/master/
  prolog_course.pptx]

  Z.B. stehen auf Folie 14 Erfahrungen zum Vergleich von Java bzw. Delphi und Prolog.

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

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

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

- Was ist aus Sicht der Logik die Aufgabe eines Prolog-Systems?
- Was ist der Unterschied zwischen `'abc'` und `"abc"`? Allgemeiner: Welche unterschiedlichen Repräsentationen von Zeichenketten gibt es in Prolog, und was sind die jeweiligen Vor- und Nachteile? Berücksichtigen Sie auch moderne Prolog-Systeme, die einen eigenen "String"-Datentyp haben.
- Der Normalfall von "Atomen" (Bezeichnern) in Prolog sind Folgen von Buchstaben und Ziffern und Unterstrich "`_`", die mit einem Kleinbuchstaben beginnen. Was zählt außerdem noch als Atom?
- Welche Schreibweisen für ganzzahlige Konstanten kennen Sie außer der normalen Dezimalschreibweise? Wissen Sie, wie man den ASCII-Code eines Zeichens aufschreiben kann?
- Was sind Terme in Prolog?
- Mit welchen Datenstrukturen kann man ein Prolog-Programm beschreiben? (Im Skript steht das unter dem Stichwort "Abstrakte Syntax".)
- Welche drei Parameter beschreiben einen Operator? Wie definiert man einen Operator? Wie kann man die aktuell definierten Operatoren abfragen?
- Was ist die Standard-Syntax von `1+2+3`? Der Operator `+` hat den Typ `yfx` (d.h. ist linksassoziativ).
- Warum ist es wichtig, dass in der Standard-Syntax zwischen Funktionssymbol und "`(`" kein Leerzeichen steht?
- Welchem Term entspricht die Liste "`[1,2,3]`" in Prolog?

Präsenzaufgaben

c) Definieren Sie ein Prädikat mit folgender Regel:

```
p(L*R) :-
    write('L = '), write(L), nl,
    write('R = '), write(R), nl,
    nl.
```

Neu ist, dass das Kopfliteral der Regel einen strukturierten Term als Argument hat. Probieren Sie folgende Anfragen aus:

- `p(3*4).`
  Beachten Sie, dass nicht das Ergebnis "`12`" an das Prädikat übergeben wird, sondern der Term "`3*4`" (also die Baumstruktur).
- `p(*(3,4)).`
  Das ist die gleiche Anfrage mit dem Term in der Standardnotation "Funktionssymbol(Argumentterme)".
- `p(1*2*3).`
  Die implizite Klammerung ist von links, weil `*` den Typ `yfx` hat.
- `p(1*(2*3)).`
  Man beachte, dass die Klammern nur der Eingabe des Terms dienen. Der rechte Teilterm braucht bei der Ausgabe keine Klammern mehr, und wird dann auch ohne Klammern ausgegeben.
- `p(1*((2*3)*4)).`
  Hier kann man nochmal sehen, dass bei der Ausgabe nur die nötigen Klammern gedruckt werden. Intern ist einfach die Baumstruktur des Terms gespeichert, und es gibt keine Information über Klammern mehr.
- `p(2+3).`
  Da dieser Term nicht `*` als Funktor hat, scheitert die "Unifikation", mit der Anfrage und Regelkopf in Übereinstimmung gebracht werden sollen. Da es keine weitere Regel über `p` gibt, wird auch die Anfrage mit "`false`" beantwortet.
- `p(1+2*3).`
  Hier ebenso: Der Term enthält zwar ein "`*`", aber wegen "Punktrechnung vor Strichrechnung" ist der Operator in der Wurzel der Baumstruktur das "`+`". Deswegen können der Term aus der Anfrage und der Term in Kopf der Regel nicht durch Variablenbindungen in Übereinstimmung gebracht werden.
- `p(2).`
- `p(X).`
  Dieses Beispiel zeigt, dass die Unifikation in beiden Richtungen funktioniert: Die Variable `X` kann mit dem Term `L*R` in Übereinstimmung gebracht werden, indem `X` an `L*R` gebunden wird. In der Ausgabe erscheinen interne Repräsentationen der Variablen `L` und `R` (vermutlich sind die Zahlen die Hauptspeicheradressen).

d) Das Prädikat `is` zur Auswertung von arithmetischen Ausdrücken liefert bei Division durch 0 eine Exception "`zero_divisor`" (zumindest in SWI-Prolog). Mit dem folgenden Regeln definieren Sie eine vereinfachte Version `arith(X,E)`, die in diesem Fall (und bei anderen Fehlern) einfach fehlschlägt:

```
arith(E, E) :-
    number(E).
arith(X, E1 + E2) :-
    arith(X1, E1),
    arith(X2, E2),
    X is X1 + X2.
arith(X, E1 - E2) :-
    arith(X1, E1),
    arith(X2, E2),
    X is X1 - X2.
arith(X, E1 * E2) :-
    arith(X1, E1),
    arith(X2, E2),
    X is X1 * X2.
arith(X, E1 / E2) :-
    arith(X1, E1),
    arith(X2, E2),
    E2 \= 0,
    X is X1 / X2.
```

Hinweise:

- Mit `number(E)` wird getestet, ob das Argument `E` eine Zahlkonstante ist.
- Wie in Teil c) gezeigt, kann man strukturierte Terme in den Kopf einer Regel schreiben (es ist ja eine atomare Formel). Z.B. wird die Regel mit dem Kopf `arith(X, E1+E2)` nur angewendet, wenn das Prädikat mit einem Term aufgerufen wird, der in der Wurzel den Operator `+` hat. Die Variablen `E1` und `E2` werden dann an die beiden Summanden (linker und rechter Teilbaum) gebunden.
- Im Programm werden die Teilterme dann rekursiv ausgewertet und anschließend die Ergebnisse addiert. Dazu wird `is` verwendet, aber es führt hier nur die eine Addition aus (während es normalerweise beliebig komplexe arithmetische Ausdrücke auswerten kann). Das Programm ist also ein Beispiel dafür, wie man eine Rekursion entlang einer Baumstruktur programmieren kann.

Geben Sie die Regeln ein und testen Sie das Prädikat.

- Z.B. sollte `arith(X, 2+3*5)` die Antwort `X=17` geben.
- Der Aufruf `arith(X,1/0)` sollte fehlschlagen, während `X is 1/0` eine Fehlermeldung ausgibt.

Wenn Sie z.B. `arith(X, Y)` eingeben, werden Sie feststellen, dass der Aufruf nicht terminiert (nach kurzer Zeit läuft allerdings der Stack über). Das Problem ist, dass die Variable `Y` auch an Terme wie `E1 + E2` gebunden werden kann, und dadurch immer kompliziertere Ausdrücke aufgebaut werden. Das obige Prädikat ist nur dafür gedacht, mit einem variablenfreien Term als zweitem Argument aufgerufen zu werden. Wenn Sie das Problem lösen wollen, können Sie die Regeln z.B. so modifizieren:

```
arith(X, E) :-
    nonvar(E),
    E = E1 + E2,
    arith(X1, E1),
    arith(X2, E2),
    X is X1 + X2.
```

Nun wird zuerst überprüft, dass `E` keine Variable ist, und der Ausdruck dann erst zerlegt. Selbstverständlich kann `E = E1 + E2` auch scheitern (zu "false" ausgewertet werden), wenn der Ausdruck `E` keine Summe ist, also nicht "`+`" als äußersten Funktor hat.

e) Definieren Sie ein Prädikat `ableitung(F,G)`, das mit einem Polynom `F` aufgerufen wird, und `G` an die Ableitung bindet. `F` ist also ein arithmetischer Ausdruck der Form

a_n `* x^`n `+ ... +` a_2 `* x^2 +` a_1 `* x +` a_0

Dabei stehen anstelle der a_i Zahlkonstanten, und natürlich ist auch n eine konkrete Zahl. Ein Beispiel ist:

```
3 * x^2 + 5 * x + 7.
```

Es ist möglich, dass a_i fehlen, also ein Summand die Form `x^`i hat, dann ist $a_i = 1$. Es müssen auch nicht alle Potenzen wirklich auftreten. Das Argument des Polynoms heisst immer `x` (in Prolog ist das ein Atom und keine Variable).

Die Ableitung des Polynoms

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 X + a_0$$

ist

$$a_n * n * x^{n-1} + a_{n-1} * (n-1) * x^{n-2} + \cdots + a_1.$$

Sie müssen also jeweils das Produkt des Exponenten i mit dem Koeffizienten a_i berechnen, und den Exponenten um 1 verringern.

Stellen Sie bitte sicher, dass einfach `x` statt `x^1` gedruckt wird. Ebenso sollte `+0` eliminiert werden. Sie können ggf. ein Hilfsprädikat zum Vereinfachen einer Formel definieren, wenn Sie diese Fälle nicht gleich speziell behandeln wollen.

Testen Sie z.B. die folgenden Fälle:

- `ableitung(x, A)` $\longrightarrow$ `A = 1.`
- `ableitung(27, A)` $\longrightarrow$ `A = 0.`
- `ableitung(x^2 + 5*x + 10, A)` $\longrightarrow$ `A = 2*x+5.`
- `ableitung(2*x^5 + 3*x^2, A)` $\longrightarrow$ `A = 10*x^4+6*x.`

Hausaufgabe

f) Es soll nun der Auswerter für aussagenlogische Formeln von Blatt 3 so umgeschrieben werden, dass er mit strukturierten Termen für die Formeln funktioniert. Im Blatt 3 war die Baumstruktur dagegen durch Fakten für die einzelnen Knoten repräsentiert worden. Definieren Sie zunächst folgende Operatoren:

- `and`
- `or`
- `neg`
- `xor`
- `v` (soll ein Synonym für `or` sein)
- `&` (soll ein Synonym für `and` sein)
- `~` (soll ein Synonym für `neg` sein)
- `not` (soll ebenfalls ein Synonym für `neg` sein)

Die Prioritäten können Sie frei wählen, beachten Sie aber Folgendes:

- `neg` soll am stärksten binden, danach `and`, dann `or`, und am schwächsten `xor`.
- Die doppelte Negation "`neg neg p1`" soll möglich sein.
- `and` und `or` sollen "linksassoziativ" sein, also "von links" binden (wie `+`).
- `xor` soll nicht assoziativ sein, d.h. "`p1 xor p1 xor p2`" soll ohne Klammern ein Syntaxfehler sein.
- Die Eingabe von Formeln wird leichter, wenn Sie Prioritäten kleiner als 1000 verwenden (sonst müssen Sie die Formeln später in Klammern schreiben). Das bedeutet aber, dass Sie nicht die gleichen Prioritäten wie für die eingebaute Konjunktion "`,`" und Disjunktion "`;`" wählen können.
- Die Operatoren `=` und `<` sind vordefiniert mit Priorität 700. Für den Fall, das später atomare Formeln mit diesen Vergleichsoperatoren verwendet werden sollen, wäre es gut, Prioritäten größer als 700 zu wählen. Im Moment sind die atomaren Formeln aber nur `p1` und `p2`.

- Falls bei dem von Ihnen verwendeten Prolog-System schon einer der obigen Operatoren definiert ist, können Sie den weglassen oder passend ersetzen (falls die Priorität gar nicht passt).

Wenn Sie wollen, können Sie auch noch weitere Operatoren definieren, z.B. `iff` und `<->`. Sie müssen diese Operatoren dann natürlich auch bei den anderen Aufgaben behandeln.

g) Schreiben Sie ein Prädikat `normalize(F, G)`, das die Synonyme aus der Eingabeformel `F` ersetzt, und in `G` die normalisierte Form liefert. Z.B. soll `normalize(p1 v p2, X)` die Variable `X` an `p1 or p2` binden. Für das Rätsel aus "Dame oder Tiger" können Sie zusätzlich auch "`links`" durch `p1` und "`rechts`" durch "`p2`" ersetzen (wenn Sie wollen).

h) Schreiben Sie nun einen Auswerter für die normalisierten Formeln, also ein Prädikat `eval(F, P1, P2, T)`, das mit einer Formel `F` aufgerufen wird, und zu Wahrheitswerten `P1` von `p1` und `P2` von `p2` den Wahrheitswert `T` der Formel berechnet.

Z.B. soll `eval(p1 and p2, true, false, X)` das Ergebnis `X = false` liefern. Testen Sie, dass man auch umgekehrt zu vorgegebenem Ergebnis mögliche Belegungen für `p1` und `p2` generieren kann. Das erste Argument (die aussagenlogische Formel) wird dagegen immer gegeben sein müssen, damit man nicht in Endlosrekursionen gerät.

Hier nochmals die Daten des Rätsels aus dem letzten Blatt:

- An der linken Tür steht: "Diese Tür führt in die Freiheit, die andere in den Tod."
- An der rechten Tür steht: "Eine von beiden Türen führt in die Freiheit, die andere in den Tod."
- Eine der beiden Inschriften ist wahr, die andere falsch.

Wenn man "Die linke Tür führt in die Freiheit" als `p1` codiert, und "Die rechte Tür führt in die Freiheit" als `p2`, ist die entscheidene Anfrage also

```
eval((p1 and neg p2) xor (p1 xor p2), Links, Rechts, true).
```