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

20.06.2019

Logische Programmierung & deduktive Datenbanken — Übungsblatt 8 (Minimales Herbrandmodell) —

Ihre Lösungen zu den Hausaufgaben g) und h) schicken Sie bitte per EMail an den Dozenten (mit "`[lp19]`" in der Betreff-Zeile). Einsendeschluss ist Mittwoch, der 26. Juni. Aufgabe b) wird dort auch 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.

- Zu der Vorlesung "Knowledge Representation" von Marek Sergot gibt es Vorlesungsmaterialien hier:

  [https://www.doc.ic.ac.uk/˜mjs/teaching/491.html]

  Sie könnten sich z.B. das Kapitel "Minimal models and fixpoint semantics for definite programs" anschauen:

  [https://www.doc.ic.ac.uk/˜mjs/teaching/KnowledgeRep491/Fixpoint_Definite_491-2x1.pdf]

- Sie finden meine Habilitationsschrift "Bottom-Up Query Evaluation in Extended Deductive Databases" (Universität Hannover, 1997) z.B. hier:

  [http://nbn-resolving.de/urn:nbn:de:gbv:089-2367669109]

  Zumindest die Kapitel 1 und 2 wären auch für diese Vorlesung sehr lesenswert.

- SICStus Prolog ist der bekannteste kommerzielle Prolog-Anbieter:

  [https://sicstus.sics.se/]

- Eine Liste historischer Artikel zu Prolog von Alain Colmerauer, Vater von Prolog, finden Sie hier:

  [http://alain.colmerauer.free.fr/]

  Nach Angabe der Wikipedia ist er am 12. Mai 2017 verstorben:

  [https://de.wikipedia.org/wiki/Alain_Colmerauer]

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

- Definieren Sie den Begriff "Minimales Herbrandmodell" eines logischen Programms P.
- Gegeben sei das folgende logische Programm:

  ```
  p(a).
  q(b).
  r(X) :- p(X).
  ```

  Was ist das minimale Herbrandmodell dieses Programms? Geben Sie ein Beispiel für ein nicht-minimales Herbrandmodell dieses Programms. Warum gibt es nicht-minimale Modelle?
- Hat jedes logische Programm (Menge definiter Hornklauseln) ein Herbrandmodell? Gibt es immer ein minimales Modell? Kann es auch mehr als ein minimales Herbrandmodell geben?
- Wie kann man das minimale Herbrandmodell konstruieren?
- Gegeben sei ein logisches Programm P. Definieren Sie den Operator T_P.
- Was ist ein "Fixpunkt" einer Abbildung, speziell des Operators T_P? Gibt es immer einen Fixpunkt? Geben Sie ein Beispiel für ein logisches Programm, für das der T_P-Operator mehr als einen Fixpunkt hat. Wie ist "kleinster Fixpunkt" definiert? Gibt es immer einen kleinsten Fixpunkt? Ist dieser eindeutig? (Es wäre schön, wenn Sie den entsprechenden Satz wiedergeben könnten.)
- Warum ist der kleinste Fixpunkt von T_P wichtig? Was ist der Zusammenhang zum minimalen Modell?
- Wie kann man das minimale Modell mit dem T_P-Operator berechnen? Schreiben Sie die Iteration formal auf.
- Angenommen, die Iteration des T_P-Operators terminiert nicht. Was kann man dennoch über die Beziehung zum minimalen Modell aussagen?
- Was ist ein kausales Modell ("supported model")? Welche Beziehung gibt es zum minimalen Modell?

Präsenzaufgaben

c) Zeichnen Sie einen SLD-Ableitungsbaum für das folgende logische Programm und die Anfrage `p(X)` (mit der üblichen Selektionsfunktion wie in Prolog):

```
p(X) :- q1(X), q2(X,c).
p(X) :- q3(X,Y), q4(Y).
q1(X) :- r1(X), r2(X).
q1(X) :- r3(X).
q2(X,Z) :- s1(X), s2(X,Z).
q3(d,e).
q4(e).
r1(a).
r1(b).
r2(b).
r3(c).
s1(b).
s2(b,c).
```

Sie können sich den Baum mit dem folgenden Programm anzeigen lassen:

[http://www.informatik.uni-halle.de/˜brass/lp19/sld.pl]

In diesem Programm sind die Regeln in der Form `rule(Head, BodyList, VarList)` als Daten repräsentiert, z.B.

```
p(X) :- q1(X), q2(X,c).
```

als

```
rule(p(X), [q1(X), q2(X,c)], [var('X', X)]).
```

Das letzte Argument erlaubt eine verbesserte Ausgabe der Variablen.

d) Mit den folgenden Regeln kann man den T_P-operator implementieren:

```
derivable(Head, Facts) :-
    rule(Head, BodyList, _Vars),
    is_true(BodyList, Facts).

is_true([], _).
is_true([Lit|BodyRest], Facts) :-
    member(Lit, Facts),
    is_true(BodyRest, Facts).

tp(FactsIn, FactsOut) :-
    findall(Fact, derivable(Fact, FactsIn), FactsOut).
```


Die Repräsentation der Regeln ist wie oben gezeigt. Das zusätzliche Argument für die Variablen wird hier nicht benötigt (wenn Sie nicht bei der Regelanwendung die Substitution ausgeben wollen).

Machen Sie sich ein einfaches Beispiel und prüfen Sie, dass diese Regeln den T_P-Operator korrekt implementieren.

e) Die Iteration des T_P-Operators können Sie mit den folgenden Regeln in Prolog definieren:

```
minmod(Facts) :-
    minmod_iter(0, [], Facts).

minmod_iter(N, Facts, MinMod) :-
    tp(Facts, NewFacts),
    (has_changed(Facts, NewFacts) ->
        NextN is N+1,
        minmod_iter(NextN, NewFacts, MinMod);
        MinMod = Facts).

has_changed(Facts, NewFacts) :-
    member(Fact, NewFacts),
    \+ member(Fact, Facts).
```

Bauen Sie Ausgaben in dieses Programm ein, so dass man die iterative Berechnung des minimalen Modells mit dem T_P-Operator gut nachvollziehen kann. Wenn es Schwierigkeiten beim abtippen der Regeln gibt, finden Sie diese auch hier:

[http://www.informatik.uni-halle.de/˜brass/lp19/tp0.pl]

f) Ein bekannter Typ von Rätsel ist:

```
 SEND
+MORE
-----
MONEY
```

Man muss Ziffern `0...9` für die Buchstaben (Variablen) finden, so dass

- Die Additionsaufgabe korrekt ist,
- verschiedene Buchstaben mit unterschiedlichen Ziffern belegt werden,
- die ersten Buchstaben in jeder Zeile nicht mit `0` belegt werden.

Eine simple Formulierung dieses Rätsels in Prolog ist:

```
generate([S,E,N,D,M,O,R,Y]),
S \= 0,
M \= 0,
to_int([S,E,N,D], I1),
to_int([M,O,R,E], I2),
to_int([M,O,N,E,Y], I3),
I3 =:= I1 + I2.
```

Definieren Sie die Hilfsprädikate:

- `generate(L)` soll alle Elemente der Liste mit unterschiedlichen Ziffern belegen (auf alle möglichen Arten). Die Ziffern sollen dabei als ganze Zahlen mit Wert zwischen 0 und 9 repräsentiert werden (nicht als Atome/Zeichen).
- `to_int(L,I)` soll den Zahlwert `I` einer Ziffernliste `L` berechnen.

Wenn Sie noch Zeit haben, definieren Sie ein Prädikat, dass man so aufrufen kann:

```
raetsel([S,E,N,D], [M,O,R,E], [M,O,N,E,Y]).
```

Prüfen Sie dann auch, ob die folgenden Rätsel lösbar sind, die sich im Internet finden:

- `raetsel([B,I,L,L], [I,R,M,A], [L,I,E,B,E]).`
- `raetsel([A,A,L], [A,A,L], [F,A,N,G]).`
- `raetsel([G,A,U,S,S], [R,I,E,S,E], [E,U,K,L,I,D]).`
- `raetsel([V,A,T,E,R], [M,U,T,T,E,R], [E,L,T,E,R,N]).`

Wenn Sie das obige Schema nutzen wollen, müssen Sie eine Liste der unterschiedlichen Variablen berechnen, die in den drei Eingabelisten vorkommen. Benutzen Sie `append` um die Listen zu einer Liste zu konkatenieren, und rufen Sie dann das folgende Prädikat auf, um doppelte Variablen zu eliminieren:

```
distinct([], []).
distinct([E|R], OUT) :-
    (var_member(E, R) -> OUT=L; OUT=[E|L]),
    distinct(R, L).

var_member(X, [Y|_]) :-
    X == Y.
var_member(X, [_|L]) :-
    var_member(X, L).
```


Hausaufgabe

g) "Vier gewinnt" [https://de.wikipedia.org/wiki/Vier_gewinnt] wird wie folgt gespielt:

- Es gibt jeweils 21 gelbe und rote Spielmarken. Das Spiel wird von zwei Spielern gespielt, einer bekommt die gelben Spielmarken, einer die roten.
- Das Spielfeld hat sieben Spalten, die jeweils sechs Spielmarken fassen können. Es handelt sich also um eine 6×7-Matrix. Die Elemente können "gelb", "rot" oder leer sein.
- Die Spieler ziehen abwechselnd. Ein Spielzug besteht im Einwerfen einer Marke der eigenen Farbe in eine noch nicht komplett gefüllte Spalte. Sie fällt dann auf den untersten noch freien Platz in dieser Spalte.
- Falls ein Spieler vier Steine der eigenen Farbe horizontal, vertikal oder diagonal in einer Reihe hat, hat er gewonnen.
- Ist das Spielfeld voll, ohne dass vier Steine einer Farbe in einer Reihe stehen, endet es unentschieden.

Überlegen Sie sich eine Repräsentation des Spielfelds in Prolog, erläutern Sie diese, und definieren Sie folgende Prädikate:

- `initial(S)`:
  Liefert das leere Spielfeld (Spiel-Zustand).
- `zug(Von, Farbe, Spalte, Nach)`:
  Dieses Prädikat wird mit einem Zustand `Von`, einer `Farbe` (`gelb` oder `rot`) und einer `Spalte` (`1` bis `7`) aufgerufen. Es liefert dann den Ergebnis-Zustand `Nach` des Zuges.
- `marke(Zustand, Zeile, Spalte, Farbe)`:
  Dies ist wahr gdw. im Zustand `Zustand` an der Position `(Zeile, Spalte)` eine Marke der Farbe `Farbe` steht. Zeilen (`1` bis `6`) sollen dabei von unten gezählt werden, d.h. die erste Marke in einer Spalte steht in Zeile `1`. Es muss möglich sein, das Prädikat mit gegebenem Zustand, aber Variablen für `Zeile`, `Spalte` und `Farbe` aufzurufen, dann sollen alle existierenden Spielsteine durchgegangen werden. Wird das Prädikat mit Zustand, Zeile und Spalte aufgerufen, soll es fehlschlagen, wenn die Position noch frei ist (oder ungültig).

h) Definieren Sie nun noch die folgenden Prädikate. Verwenden Sie dabei nur die Schnittstelle aus g). So können Sie die Repräsentation des Spielfeldes später austauschen.

- `drucke(S)`:
  Druckt den aktuellen Spiel-Zustand aus (z.B. mit "`*`" für gelbe Spielmarken und "`+`" für rote).
- `gewonnen(S, Farbe)`:
  Dieses Prädikat soll wahr sein, wenn der Spieler mit der Farbe `Farbe` im Zustand `S` gewonnen hat.
- `voll(S)`:
  Dieses Prädikat soll wahr sein, wenn das Spielfeld voll ist.