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

19.04.2018

Logische Programmierung & deduktive Datenbanken — Übungsblatt 3 (Logik) —

Ihre Lösungen zu den Hausaufgaben g) bis i) schicken Sie bitte per EMail an den jeweiligen Übungsleiter, also `mario.wenzel@informatik.uni-halle.de` für die Dienstags-Gruppe und `brass@informatik.uni-halle.de` für die Donnerstags-Gruppe. Bitte schreiben Sie "`lp18`" mit in die Betreff-Zeile. Einsendeschluss ist Samstag, der 28. April. 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.

- Das Buch "Logic, Programming und Prolog (2nd Ed.)" von Ulf Nilson und Jan Małuszyński gibt es als kostenloses PDF im Internet:

  [http://www.ida.liu.se/~ulfni53/lpp/]

  Es gibt auch einen Foliensatz dazu.

- Ein aktueller Übersichtsartikel zu deduktiven Datenbanken (fast schon ein Buch) ist Todd J. Green, Shan Shan Huang, Boon Thau Loo, Wenchao Zhou: "Datalog and Recursive Query Processing". Foundations and Trends in Databases, Vol. 5, No. 2 (2012), 105-195.

  [http://blogs.evergreen.edu/sosw/files/2014/04/Green-Vol5-DBS-017.pdf]

  Laden Sie sich diesen Artikel herunter und lesen Sie die ersten Seiten der Einleitung (106–107) als Motivation. Sie können natürlich gerne auch mehr lesen.

- Eine Version des Buches "Constraint Logic Programming using ECL^{i}PS^e" von Krzysztof R. Apt und Mark Wallace (2006) gibt es hier:

  [https://pdfs.semanticscholar.org/329c/
  0913723ebc282d021cdab9fa32b5400c7f0a.pdf]

  Ich weiss nicht, ob das beabsichtigt ist: Das Buch wird aktuell noch verkauft.

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

- Was ist eine Signatur (der mehrsortigen Prädikatenlogik erster Stufe)? Wozu dient sie und was sind die wesentlichen Komponenten?
- Was ist eine Interpretation?
- Was ist eine Variablenbelegung?
- Definieren Sie den Begriff "Term" (der Prädikatenlogik erster Stufe).
- Wie sind Formeln der Prädikatenlogik erster Stufe aufgebaut?
- Was bedeutet es, dass eine Formelmenge F eine Formel G logisch impliziert? D.h. wie ist dieser Begriff definiert?
- Was bedeutet es, das zwei Formeln F und G äquivalent sind? Nennen Sie einige Äquivalenzen.
- Wie kann man die Frage, ob eine Formelmenge F eine Formel G logisch impliziert auf einen Konsistenztest zurückführen? Beweiser durch Widerspruch arbeiten nach diesem Prinzip.
- Was sind Literale (positive und negative Literale)?
- Definieren Sie den Begriff "Klausel".
- Kann man jede Formel der Prädikatenlogik erster Stufe in eine Menge von Klauseln überführen?
- Was ist die Idee der Skolemnisierung?
- Definieren Sie den Begriff Substitution. Was unterscheidet Substitutionen von Variablenbelegungen?
- Was sind Herbrand-Interpretationen? Was unterscheidet sie von allgemeinen Interpretationen? Warum sind sie nützlich?
- Erläutern Sie, wie man die Frage, ob eine Formelmenge F eine Formel G logisch impliziert auf die Existenz eines Herbrand-Modells einer Klauselmenge zurückführen kann.

Präsenzaufgaben

c) Gegeben sei eine Datenbank mit den folgenden Relationen:

- `emp(EmpNo, EName, Job, DeptNo)`: Angestellte, hier reduziert auf vier Spalten
- `dept(DeptNo, DName, Loc)`: Abteilungen.

Es sei nun die folgende Fremdschlüssel-Bedingung betrachtet:

$$\forall \mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{D} \left(\mathtt{emp}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{D}) \rightarrow \exists \mathtt{N}, \mathtt{L}\ \mathtt{dept}(\mathtt{D}, \mathtt{N}, \mathtt{L})\right).$$

Benutzen Sie die in der Vorlesung angegebenen Äquivalenz-Transformationen, um zu zeigen, dass die Formel äquivalent zu der folgenden Formel ist:

$$\forall\, \mathtt{D}\, \exists\, \mathtt{N}, \mathtt{L}\, \forall\, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\, \Big(\mathtt{emp}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{D}) \rightarrow \mathtt{dept}(\mathtt{D}, \mathtt{N}, \mathtt{L})\Big).$$

Welche Skolem-Funktionen werden für diese Formel eingeführt?

d) Wandeln Sie die folgende Formel in Klauseln um:

$$\Big(\exists\, \mathtt{Y}\, \forall\, \mathtt{X}\, \mathtt{p}(\mathtt{X}, \mathtt{Y}) \vee \neg(\mathtt{q}(\mathtt{X}) \vee \mathtt{r}(\mathtt{Y}))\Big) \wedge \mathtt{r}(1)$$

Schreiben Sie die Klauseln

- als Disjunktionen und
- als Fakten und Regeln

Geben Sie ein Herbrandmodell der Klauseln an.

e) Definieren Sie in Prolog ein Prädikat `prim(N)`, das genau dann wahr ist, wenn die natürliche Zahl `N` eine Primzahl ist.

Es ist nur verlangt, dass das Prädikat für eine gegebene Zahl `N` den korrekten Wahrheitswert hat, z.B. müssen `prim(2)` und `prim(3)` gelten, aber `prim(4)` nicht. Es ist nicht verlangt, dass der Aufruf `prim(N)` mit einer Variablen `N` möglich ist, und `N` dann nacheinander an `2`, `3`, u.s.w. gebunden wird. Wenn Sie Ihr Prädikat so programmieren wollen, dürfen Sie das natürlich tun.

Hinweise:

- Die ganzzahlige Division schreibt man in Prolog `div`. Wenn Sie z.B. 7 durch 3 teilen wollen, schreiben Sie `X is 7 div 3`. Anschließend ist `X = 2`.
- Den Divisionsrest können Sie entsprechend mit dem Operator `mod` bestimmen, z.B. liefert `Y is 7 mod 3` die Variablenbindung `Y = 1`.
- Natürlich kann man auch Zahlen addieren oder subtrahieren, z.B. `Y is X+1`.
- Man kann Zahlen bzw. arithmetische Ausdrücke vergleichen mit den Operatoren `<`, `=<`, `>=`, `>`. Ein Beispiel ist die Bedingung `I < N`.
- Falls Sie testen wollen, dass das Argument eine ganze Zahl ist, können Sie das mit der Bedingung `integer(N)` tun. Sie können sich aber auch einfach darauf verlassen, dass das Prädikat mit einem Argument des richtigen Typs aufgerufen wird. Spätestens bei arithmetischen Operationen wie `div` würde `is` einen Typfehler melden.
- In SWI-Prolog und GNU-Prolog aber nicht im Prolog-Standard (und nicht z.B. in ECLiPSe) gibt es ein Prädikat `between(Low,High,I)` das mit ganzzahligen Werten für `Low` und `High` aufgerufen werden muss, und `I` dann nacheinander an `Low`, `Low+1`, …, `High` bindet. Sie können sich auch leicht selbst so ein Prädikat definieren:


```
myBetween(Low, High, Low) :-
        Low =< High.
myBetween(Low, High, Value) :-
        Next is Low + 1,
        Next =< High,
        myBetween(Next, High, Value).
```

f) Schreiben Sie ein Prädikat `ggt(N,M,T)`, das den größten gemeinsamen Teiler `T` zweier natürlicher Zahlen `N` und `M` berechnet. Der einfachste Version des "Euklidischen Algorithmus" beruht auf folgenden Tatsachen:

- $\mathtt{ggT}(n, n) = n$.
- Wenn $n > m$, dann $\mathtt{ggT}(n, m) = \mathtt{ggT}(n - m, m)$.

  Beweis: Sei $g = \mathtt{ggT}(n, m)$. Da g Teiler von n und m ist, gibt es natürliche Zahlen a und b mit $n = g * a$ und $m = g * b$. Weil g der größte gemeinsame Teiler ist, sind a und b teilerfremd. Nun ist g natürlich auch ein Teiler von $n - m = g * (a - b)$. Hätten $m = g * b$ und $n - m = g * (a - b)$ einen weiteren gemeinsamen Teiler außer g, müssen b und $a - b$ einen gemeinsamen Teiler t haben. Dann hätte aber auch $a = (a - b) + b$ diesen Teiler t, im Widerspruch zur Voraussetzung, dass a und b teilerfremd sind.
- Umgekehrt gilt: Wenn $n < m$, dann $\mathtt{ggT}(n, m) = \mathtt{ggT}(n, m - n)$.

Sie sollten in den Bedingungen aller Regeln prüfen, dass die Argumente des Prädikates > 0 sind, um die Terminierung zu garantieren.

Hausaufgabe

g) Wandeln Sie die folgende aussagenlogische Formel (mit den null-stelligen Prädikaten `p` und `q`) in Klauseln um:

$$\neg(\mathtt{p} \wedge \neg\mathtt{q}) \wedge (\mathtt{p} \vee \mathtt{q}).$$

Geben Sie ein Modell an, falls eins existiert.

Tipp: Wenn Sie wollen, können Sie auch den Logik-Auswerter aus h) benutzen, um ein Modell zu finden. Sie müssen die obige Formel dann in Fakten codieren, wie unter h) beschrieben.

h) Schreiben Sie ein Prolog-Programm, das aussagenlogische Formeln mit zwei null-stelligen Prädikaten $\mathtt{p}_1$ und $\mathtt{p}_2$ auswerten kann. Solche Formeln sind eine Baumstruktur. Da in der Vorlesung noch keine komplexen Prolog-Terme behandelt wurden, sollen die Knoten des Baums durch ganze Zahlen identifiziert werden, und als Fakten mit folgenden Prädikaten aufgeschrieben werden:

- `and(In1, In2, Out)`: Es gibt eine AND-Verknüpfung ($\wedge$) zwischen Knoten `In1` und `In2`, mit der Ausgabe `Out`. D.h. der Wahrheitswert des Knotens mit der eindeutigen Nummer `Out` kann aus den Wahrheitswerten der Knoten mit den Nummern `In1` und `In2` durch eine "und"-Verknüpfung berechnet werden.
- `or(In1, In2, Out)`: Entsprechend eine OR-Verknüpfung ($\vee$, "oder").
- `xor(In1, In2, Out)`: Entsprechend eine XOR-Verknüpfung: ($\mathtt{p} \leftrightarrow \neg\mathtt{q}$), "exklusiv oder" (die Wahrheitswerte der Eingaben sind unterschiedlich).
- `neg(In, Out)`: NOT-Verknüpfung ($\neg$, "nicht").
- `in1(X)`: Am Knoten mit der Nummer `X` liegt der erste Eingang ($\mathtt{p}_1$).
- `in2(X)`: Am Knoten mit der Nummer `X` liegt der zweite Eingang ($\mathtt{p}_2$).
- `out(X)`: Der Wahrheitswert von Knoten `X` ist das Ergebnis (d.h. `X` ist die Wurzel).

Sie können die einzelnen Fakten auch als Logik-Gatter einer Schaltung betrachten, und die Nummern der Knoten als Identifikationen von Leiterbahnen dazwischen (Messpunkte).

Als Beispiel sei das folgende logische Rätsel verwendet (inspiriert von dem Buch "Dame oder Tiger" von Raymond M. Smullyan). Ein König stellte seine Gefangenen vor die Wahl zwischen zwei Türen. Meist führte eine Tür in die Freiheit, die andere in den Tod (es lauerte dort ein hungriger Tiger). Es war aber auch möglich, das beide in die Freiheit oder beide in den Tod führen. Der König hat an den Türen Tafeln mit Hinweisen angebracht, zur Erschwerung war aber von den beiden Hinweisen einer richtig und einer falsch. Für einen der Gefangenen waren die Hinweise wie folgt:

- An der linken Tür: Diese Tür führt in die Freiheit, die andere in den Tod.
- An der rechten Tür: Eine von beiden Türen führt in die Freiheit, die andere in den Tod.

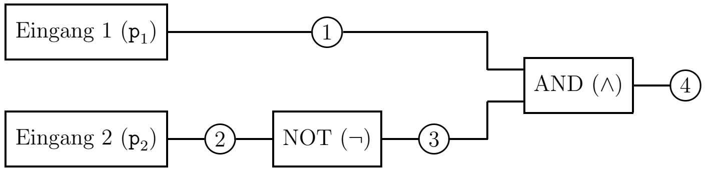
Für den Logik-Auswerter sei jetzt

- Eingabe 1 ($\mathtt{p}_1$): "Die linke Tür führt in die Freiheit."
- Eingabe 2 ($\mathtt{p}_2$): "Die rechte Tür führt in die Freiheit."

Der Hinweis an der linken Tür entspricht Knoten `4` wenn man die Aussagen folgendermaßen als Fakten codiert:

```
in1(1).     % Die linke Tür führt in die Freiheit: p1
in2(2).     % Die rechte Tür führt in die Freiheit: p2
neg(2,3).   % Die rechte Tür führt in den Tod: ¬p2.
and(1,3,4). % [Linke Tür:] Die linke Tür führt in die Freiheit,
            % die rechte in den Tod: p1 ∧ ¬p2.
```

Man kann sich diese Daten auch so als Schaltung vorstellen:



Der Hinweis an der rechten Tür entspricht dem folgenden Fakt:

```
xor(1,2,5). % [Rechte Tür:] Eine Tür führt in die Freiheit,
            % die andere in den Tod: p1 ↔ ¬p2.
```

Schließlich sind noch die beiden Hinweise mit "exklusiv-oder" zu verknüpfen, da nur einer wahr ist, der andere aber falsch:

```
xor(4,5,6). % Genau eine der Inschriften ist richtig:
            % (p1 ∧ ¬p2) ↔ ¬(p1 ↔ ¬p2).
out(6).     % Knoten 6 ist das Ergebnis.
```

Sie finden die Fakten auch in folgender Datei

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

Schreiben Sie nun ein Prädikat `eval(In1, In2, Out)`, das erfüllt ist, wenn `In1` und `In2` die Wahrheitswerte (`true` oder `false`) der Eingaben sind, und `Out` der zugehörige Wahrheitswert der Ausgabe.

Selbstverständlich dürfen Sie beliebige Hilfsprädikate definieren. Eine Möglichkeit wäre, ein zusätzliches Argument für den Knoten vorzusehen, dessen Wahrheitswert bestimmt werden soll. Am Ende ist nur der Wahrheitswert des `out`-Knotens verlangt, aber Sie müssen rekursiv die Wahrheitswerte der übrigen Knoten berechnen.

Sie dürfen voraussetzen, dass der Graph keine Zyklen enthält. Wenn Sie wollen, können Sie auch eine maximale Schachtelungstiefe voraussetzen, um auch bei Zyklen zu terminieren. Diese Schachtelungstiefe muss aber leicht konfigurierbar sein, z.B. mit einem Fakt `limit(10)`. Es ist aber nicht verlangt, dass Sie Zyklen behandeln. Eine deduktive Datenbank würde im Gegensatz zu Prolog auch bei zyklischen Daten terminieren.

Noch ein Tipp: Sie können sich leicht Wahrheitswert-Tabellen für die logischen Vernüpfungen mit Prolog-Fakten definieren, z.B.

```
% and_tab(In1, In2,   Out).
and_tab(false, false, false).
and_tab(false, true,  false).
and_tab(true,  false, false).
and_tab(true,  true,  true).
```

Natürlich könnten Sie auch die in Prolog eingebauten logischen Verknüpfungen verwenden, aber dann müssen Sie vom Fehlschlagen eines Prolog-Beweisziels wieder zum Datenwert "`false`" kommen. Es ist zunächst einfacher, die Programm-Ebene und die Datenebene zu trennen. Zumindest für `eval` ist gefordert, dass Sie im dritten Argument das Ergebnis der Auswertung als Datenwert (`true` oder `false`) liefern.

i) Schreiben Sie nun noch ein Prädikat, das eine Wahrheitstabelle für die Lösung aus h) ausgibt, also z.B.

```
In1    In2    Out
-----------------
false  false  false
false  true   true
true   false  false
true   true   false
```

Sie benötigen dazu weitere eingebaute Prädikate von Prolog:

- Mit dem Prädikat `write(X)` können Sie `X` ausgeben.
- Mit dem Prädikat `nl` können Sie einen Zeilenumbruch ausgeben.
- Falls Sie Backtracking verwenden wollen, können Sie es mit `fail` erzwingen.