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

22. September 2011

Logische Programmierung & Deduktive Datenbanken
— Klausur —

Name: ______________________________

Matrikelnummer: ______________________________

Studiengang: ______________________________

Aufgabe	Punkte	Max. Punkte	Zeit
1 (Prolog Programm)		10	15 min
2 (Prädikat-Abhängigkeitsgraph)		5	10 min
3 (Datalog Programm)		10	15 min
4 (Minimales Modell)		5	10 min
5 (SLD-Baum)		5	10 min
6 (Unifikation)		2	5 min
7 (Prolog Syntax)		3	5 min
8 (Bereichsbeschränkung)		3	5 min
Summe		43	75 min

☐ Ich bin gesundheitlich in der Lage, diese Prüfung abzulegen. (Andernfalls bitte bei Aufsicht melden).

☐ Falls ich zu dieser Prüfung nicht angemeldet sein sollte, stelle ich beim Prüfungsausschuss unwiderruflich den Antrag, nachträglich angemeldet zu werden.

____
(Unterschrift)

Hinweise:

- Bearbeitungsdauer: 90 Minuten
- Skript, Bücher, Notizen sind erlaubt. Notebooks, PDAs, etc. dürfen nicht verwendet werden. Mobiltelefone bitte ausschalten (bei Bedarf mit Aufsicht besprechen).
- Die Klausur hat 8 Seiten. Bitte prüfen Sie die Vollständigkeit.
- Bitte benutzen Sie den vorgegebenen Platz. Wenn Sie auf die Rückseite ausweichen müssen, markieren Sie bitte klar, daß es eine Fortsetzung gibt.
- Tauschen Sie keinesfalls irgendwelche Dinge mit den Nachbarn aus. Notfalls rufen Sie eine Aufsichtsperson zur Kontrolle.
- Fragen Sie, wenn Ihnen eine Aufgabe nicht klar ist!

Aufgabe 1 (Prolog Programm) 10 Punkte

Schreiben Sie ein Prädikat `glaetten(X, Y)`, das eine Liste `X` von Zahlen (z.B. Aktienkurse oder Messwerte) in eine Liste `Y` von Zahlen überführt, bei der jeder Wert in `X` durch den Durchschnitt dreier aufeinanderfolgender Werte in `Y` ersetzt ist. Am Ende der Liste `Y` (wenn es in `X` keine drei Werte mehr gibt), soll zuerst der Durchschnitt der letzten beiden Werte von `X` stehen, und dann der letzte Werte von `X`. Man nimmt also jeweils den Durchschnitt der noch vorhandenen Werte. Ist z.B.

```
X = [1, 2, 3, 4, 5],
```

so wird die Liste `Y` berechnet als

```
[avg(1,2,3), avg(2,3,4), avg(3,4,5), avg(4,5), avg(5)].
```

Den Durchschnitt berechnen Sie als Summe geteilt durch Anzahl, (es gibt in Prolog keine `avg`-Funktion), das Ergebnis wäre also `Y = [2, 3, 4, 4.5, 5]`. Für die leere Liste als Eingabe soll die leere Liste als Ausgabe geliefert werden.

Sie können nur die im Skript genannten eingebauten Prädikate verwenden, z.B. `is`, `>`, `=`, `\=`, `\+`. Sie können sich selbstverständlich beliebige Hilfsprädikate in Prolog definieren. Sie können davon ausgehen, daß die Eingabe korrekt ist, das Prädikat also immer mit einer Liste von Zahlen im ersten Argument aufgerufen wird. Beachten Sie, dass auch kleine Syntaxfehler zu (kleinen) Punktabzügen führen.

Aufgabe 2 (Prädikat-Abhängigkeitsgraph) 5 Punkte

Zeichen Sie den Prädikat-Abhängigkeitsgraphen und den reduzierten Prädikat-Abhängigkeitsgraphen für das folgende Datalog-Programm P (die Argumente sind für diese Aufgabe nicht wichtig, daher wurde ein aussagenlogisches Programm gewählt):

```
a ← b ∧ c.
b ← c.
c ← d ∧ e.
d ← f.
f ← c.
f ← g.
e ← h.
g.
h.
```

Aufgabe 3 (Datalog-Programm) 10 Punkte

Die für die Abhängigkeitsberechnung wesentliche Information eines Datalog-Programms sei mit den folgenden beiden Prädikaten repräsentiert:

- `defines(RuleNo, Pred)`:
  Das Kopfliteral der Regel Nr. `RuleNo` enthält das Prädikat `Pred`.
- `uses(RuleNo, Pred)`:
  Ein Rumpfliteral der Regel Nr. `RuleNo` enthält das Prädikat `Pred`.

Als Beispiel seien die ersten drei Regeln des Programms aus Aufgabe 2 betrachtet:

$$\begin{array}{l} \texttt{a} \leftarrow \texttt{b} \wedge \texttt{c}. \\ \texttt{b} \leftarrow \texttt{c}. \\ \texttt{c} \leftarrow \texttt{d} \wedge \texttt{e}. \end{array}$$

Sie werden durch folgende Fakten repräsentiert:

```
defines(1, a).
uses(1, b).
uses(1, c).
defines(2, b).
uses(2, c).
defines(3, c).
uses(3, d).
uses(3, e).
```

Schreiben Sie nun in Datalog ein Prädikat `recursive_pred`, das die rekursiven Prädikate berechnet, gegeben Fakten für `defines` und `uses`. D.h. `recursive_pred(Pred)` soll genau dann wahr sein, wenn `Pred` (ggf. auch indirekt) von sich selbst abhängt. Selbstverständlich dürfen Sie wieder Hilfsprädikate definieren.

Aufgabe 4 (Minimales Modell, T_P-Operator) 5 Punkte

Gegeben sei noch einmal das Programm P aus Aufgabe 2:

$$\begin{array}{l} \texttt{a} \leftarrow \texttt{b} \wedge \texttt{c}. \\ \texttt{b} \leftarrow \texttt{c}. \\ \texttt{c} \leftarrow \texttt{d} \wedge \texttt{e}. \\ \texttt{d} \leftarrow \texttt{f}. \\ \texttt{f} \leftarrow \texttt{c}. \\ \texttt{f} \leftarrow \texttt{g}. \\ \texttt{e} \leftarrow \texttt{h}. \\ \texttt{g}. \\ \texttt{h}. \end{array}$$

Berechnen Sie das minimale Modell von P durch Iteration des T_P-Operators. Geben Sie die Mengen $I_0 := \emptyset$, $I_i := T_P(I_{i-1})$ für $i = 1, 2, \ldots$ an, bis ein Fixpunkt erreicht ist. Selbstverständlich brauchen Sie nur die jeweils neuen Fakten explizit anzugeben, z.B. in der Form $I_2 = I_1 \cup \{\ldots\}$.

Aufgabe 5 (SLD Baum) 5 Punkte

Gegeben sei folgende Faktenmenge, die einen binären Suchbaum in der Form

```
node(ID, Value, LeftChild, RightChild)
```

repräsentiert:

```
node(n1, 300,  n2,  n3).   % Fakt F1
node(n2, 200,  n4,  nil).  % Fakt F2
node(n3, 500,  n5,  n6).   % Fakt F3
node(n4, 100, nil, nil).   % Fakt F4
node(n5, 400, nil, nil).   % Fakt F5
node(n6, 600, nil, nil).   % Fakt F6
```

Folgende Regeln beschreiben die Suche in so codierten Bäumen:

```
% Regel R1:
search(Val, Node) :-
        node(Node, Val, _, _).
% Regel R2:
search(Val, Node) :-
        node(Node, Key, Left, _),
        Val < Key,
        search(Val, Left).
% Regel R3:
search(Val, Node) :-
        node(Node, Key, _, Right),
        Val > Key,
        search(Val, Right).
```

Geben Sie den (vollständigen) SLD-Baum für folgende Anfrage an:

```
search(500, n1).
```

Schreiben Sie an jede Kante die Nummer der angewendeten Regel (bzw. des Faktes). Die Substitutionen brauchen Sie nicht anzugeben. Es ist Platz für die Lösung auf der nächsten Seite.

Lösung zu Aufgabe 5 (SLD Baum) 6 Punkte

```
node(n1, 300,  n2,  n3).   % Fakt F1
node(n2, 200,  n4,  nil).  % Fakt F2
node(n3, 500,  n5,  n6).   % Fakt F3
node(n4, 100, nil, nil).   % Fakt F4
node(n5, 400, nil, nil).   % Fakt F5
node(n6, 600, nil, nil).   % Fakt F6
search(Val, Node) :-       % Regel R1
        node(Node, Val, _, _).
search(Val, Node) :-       % Regel R2
        node(Node, Key, Left, _),
        Val < Key,
        search(Val, Left).
search(Val, Node) :-       % Regel R3
        node(Node, Key, _, Right),
        Val > Key,
        search(Val, Right).
```

Aufgabe 6 (Unifikation) 2 Punkte

Sind die folgenden beiden Literale unifizierbar? Falls ja: Geben Sie einen allgemeinsten Unifikator an. Falls nein: Begründen Sie Ihre Antwort kurz (welche Variablenbindungen werden ausgeführt und warum geht es dann nicht weiter?).

- p(X, f(X), g(f(a),X))
- p(a, Y, g(Y,Z))

Aufgabe 7 (Prolog Syntax) 3 Punkte

Was gibt `read(Rule), display(Rule)` aus, wenn der Benutzer folgendes eingibt:

```
p([a,b]) :- q, r, s; t.
```

Das eingebaute Prädikat `display` gibt den Term in Standard-Syntax aus. Alternativ dürfen Sie auch den Operatorbaum für den Term zeichnen. Die relevanten Operator-Deklarationen sind:

Operator	Typ	Priorität
:-	xfx	1200
;	xfy	1100
,	xfy	1000

Lösen Sie auch die Liste in ihre interne Repräsentation auf.

Aufgabe 8 (Bereichsbeschränkung) 3 Punkte

Nehmen wir an, für das Prädikat `sum` wären die Bindungsmuster `bbf`, `bfb`, `fbb` erlaubt, und für `cons` die Bindungsmuster `bbf` und `ffb`. Gegeben sei folgende Regel:

```
inc_list(X, Y) ←
        sum(XH, 1, YH),
        cons(YH, YT, Y),
        cons(XH, XT, X),
        inc_list(XT, YT).
```

Wenn das Prädikat `inc_list` mit dem Bindungsmuster `bf` aufgerufen wird, und nur dieses Bindungsmuster für `inc_list` erlaubt ist, wäre die Regel bereichsbeschränkt? Falls ja, geben Sie eine Auswertungsreihenfolge und die dabei auftretenden Bindungsmuster an. Falls nein, geben Sie an, wie weit man kommen könnte, und wo es dann scheitert.