Introduction to Data and Knowledge Engineering SS10 – Übung 8

TECHNISCHE UNIVERSITÄT DARMSTADT

Aufgabe 8.1 Datalog vs. Prolog

Das Programm

TECHNISCHE UNIVERSITÄT DARMSTADT

Gegeben seien folgende Fakten und Regeln:

```
succ(1,2).
succ(2,3).
succ(3,4).

numb(X)      :-  succ(X,_).
numb(X)      :-  succ(_,X).
gt(A,B)      :-  succ(B,A).
gt(A,B)      :-  succ(B,C), gt(A,C).
leq(A,A)     :-  numb(A).
leq(A,B)     :-  gt(B,A)
```

Aufgabe 8.1 Datalog vs. Prolog

a) EPP Iterationen 1 und 2

TECHNISCHE UNIVERSITÄT DARMSTADT

a) Betrachten Sie das zunächst als ein Datalog-Programm. Bestimmen Sie den Fixpunkt, indem Sie angeben, welche Fakten nach jeder EPP-Iteration hinzukommen.

1. Iteration:

```
succ(1,2)   succ(2,3)   succ(3,4)
```

2. Iteration:

```
numb(1)   numb(2)   numb(3)   numb(4)
gt(2,1)   gt(3,2)   gt(4,3)
```

Aufgabe 8.1 Datalog vs. Prolog

a) EPP Iterationen 3 bis 5

TECHNISCHE UNIVERSITÄT DARMSTADT

3. Iteration:

```
gt(3,1)   gt(4,2)
leq(1,1)  leq(2,2)  leq(3,3)  leq(4,4)
leq(1,2)  leq(2,3)  leq(3,4)
```

4. Iteration:

```
gt(4,1)
leq(1,3)  leq(2,4)
```

5. Iteration:

```
leq(1,4)
```

Aufgabe 8.1 Datalog vs. Prolog

b) SLD Resolution

TECHNISCHE UNIVERSITÄT DARMSTADT

b) Wie würde Prolog die Query `leq(2,4)` abarbeiten? Skizzieren Sie zur Illustration den SLD-Suchbaum, der auch die fehlgeschlagenen Alternativen enthält.

```
leq(2,4) → gt(4,2)
           gt(4,2) → succ(2,4)
                     succ(2,4) → Fail
           gt(4,2) → succ(2,C), gt(4,C)
                     succ(2,C) → Exit:{C/3}
                                 gt(4,3) → succ(3,4)
                                           succ(3,4) → Exit
                                 Exit
           Exit
Exit
```

Aufgabe 8.1 Datalog vs. Prolog

b) SLD Resultion

TECHNISCHE UNIVERSITÄT DARMSTADT

Substiution:

- wähle für das nächste Literal im Body die nächste Substitution, so dass es auf einen Regel-Head oder Fakt passt
- gibt es keine, dann → `Fail`
- das Anwenden einer geeigneten Substiution nennt man *Unifikation*

Ohne Extra-Ebene für Substitution und ohne indirekte Exists:

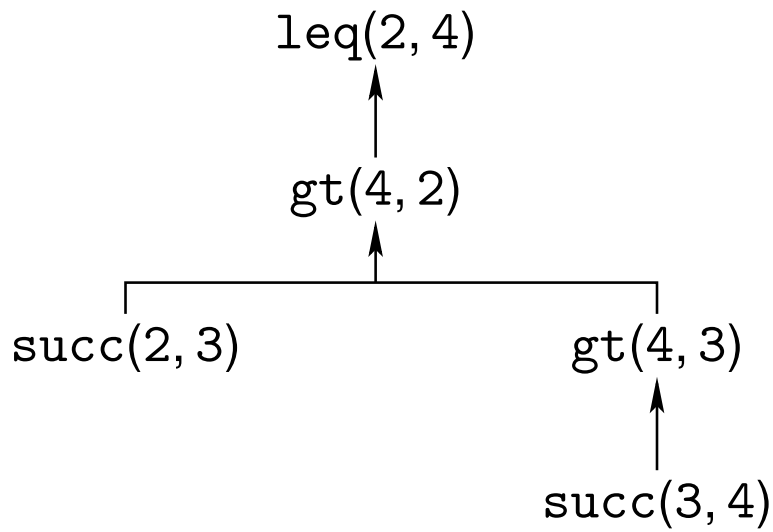
```
leq(2,4) → gt(4,2)
           gt(4,2) → succ(2,4)
                     succ(2,4) → Fail
           gt(4,2) → succ(2,C), gt(4,C)
                     succ(2,C) → Exit:{C/3}
                     gt(4,3) → succ(3,4)
                               succ(3,4) → Exit
```

Aufgabe 8.1 Datalog vs. Prolog

c) Beweisbaum

TECHNISCHE UNIVERSITÄT DARMSTADT

c) Wie sieht der Beweisbaum aus?



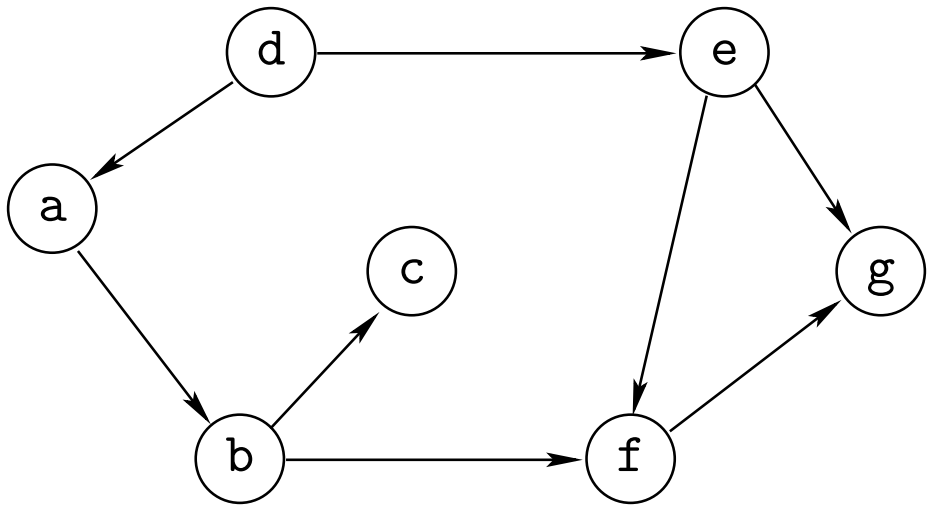
Aufgabe 8.2 Rekursion mit Prolog

Das Programm

TECHNISCHE UNIVERSITÄT DARMSTADT

Betrachten Sie die Regeln aus dem vorigen Übungsblatt und nehmen Sie an, dass wir sie in den Prolog-Interpreter geladen haben.

```
edge(a,b).
edge(b,c).
edge(b,f).
edge(d,a).
edge(d,e).
edge(e,f).
edge(e,g).
edge(f,g).
```




```
reach(X,Y)   :-   edge(X,Y).
reach(X,Z)   :-   reach(X,Y), edge(Y,Z).
```

Aufgabe 8.2 Rekursion mit Prolog

a)

TECHNISCHE UNIVERSITÄT DARMSTADT

a) Zeigen Sie die Abarbeitung der Query `reach(a,f).`

```
reach(a,f) → edge(a,f)
             edge(a,f) → Fail
reach(a,f) → reach(a,Y), edge(Y,f)
             reach(a,Y) → edge(a,Y)
                          edge(a,Y) → Exit:{Y/b}
             edge(b,f) → Exit
```

Aufgabe 8.2 Rekursion mit Prolog

b)

TECHNISCHE UNIVERSITÄT DARMSTADT

b) Was passiert bei der Abarbeitung der Query `reachable(g,e)`?

```
reach(g,e) → edge(g,e)
             edge(g,e) → Fail
reach(g,e) → reach(g,Y), edge(Y,e)
             reach(g,Y)) → edge(g,Y)
                           edge(g,Y) → Fail
             reach(g,Y)) → reach(g,Y2), edge(Y2,Y)
                           reach(g,Y2)) → edge(g,Y2)
                                          edge(g,Y2) → Fail
                           reach(g,Y2)) → reach(g,Y3), edge(Y3,Y2)
                                          reach(g,Y3)) → ...
```

Aufgabe 8.2 Rekursion mit Prolog

c) Variante 2

TECHNISCHE UNIVERSITÄT DARMSTADT

c) Betrachten Sie alle möglichen Permutationen der reachable-Regeln, sowohl der Regeln selbst als auch der Literale im Körper der Regel. Geben Sie für jede Permutation an, ob sie für alle möglichen Queries terminiert oder nicht. Begründen Sie oder zeigen Sie ein Beispiel, wo es nicht terminiert.

Vertauschte Regeln:

```
reach(X,Z)   :-   reach(X,Y), edge(Y,Z),.
reach(X,Y)   :-   edge(X,Y).
```

- terminiert für keine Anfrage

Aufgabe 8.2 Rekursion mit Prolog

c) Variante 3

TECHNISCHE UNIVERSITÄT DARMSTADT

Vertauschte Regeln und vertauschte Literale:

```
reach(X,Z)   :-   edge(Y,Z),reach(X,Y).
reach(X,Y)   :-   edge(X,Y).
```

Betrachte Query `reach(a,f)`:

- wird `reach` zum k + 1-ten mal aufgerufen, müssen zuvor k Kanten gefunden worden sein
- diese bilden einen Weg, der im Zielknoten `f` der Query endet
- hat der Graph keine Zyklen, werden alle Vorfahren von `f` gefunden (auch `a`)
- gibt es zwischen `a` und `f` einen Kreis (z.B. mit `edge(c,b).`) kann das zu einer eine Endlosrekursion führen
- nämlich dann, wenn `edge(a,b)` nach dem Kreis `edge(c,b).`, `edge(c,b).` gelistet ist

Aufgabe 8.2 Rekursion mit Prolog

c) Variante 4

TECHNISCHE UNIVERSITÄT DARMSTADT

Vertauschte Literale:

```
reach(X,Y)   :-   edge(X,Y).
reach(X,Z)   :-   edge(Y,Z),reach(X,Y).
```

Betrachte Query `reach(a,f)`:

- wie gehabt: nach $k+1$-ten `reach`-Aufruf ist Weg der Länge k (inlusive Zyklen) gefunden, der im Zielknoten `f` endet
- gibt es einen direkten Weg (Kante) von `X` wird dieser als erstes gewählt
- mit Zyklus über `b,c` terminiert Query `reach(d,f)` nicht, `reach(g,c)` ebenfalls nicht

Aufgabe 8.2 Rekursion mit Prolog

d)

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei Datalog terminieren die Queries immer.

Per EPP wird der Fixpunkt der Fanktenmenge erstellt, alle anderen Atome sind falsch. Diese Liste von wahren Atome kann erstellt werden noch bevor die Queries abgesetzt werden. Dann muss zur Beantwortung nur noch geprüft werden, mit welchen wahren Atomen die Query unifiziert.

Aufgabe 8.3 Cut

Das Programm

TECHNISCHE UNIVERSITÄT DARMSTADT

Betrachten Sie folgendes Prolog-Programm:

```
b(1).
b(2).
c(1).
c(2).
d(3).
a(X,Y)   :-   b(X), c(Y).
a(X,Y)   :-   d(X), d(Y).
```

Aufgabe 8.3 Cut

a)

TECHNISCHE UNIVERSITÄT DARMSTADT

a) Welche Lösungen werden zu der Query `a(X,Y).` gefunden?

Die nächste Lösung wird jeweils via Backtracking gefunden.

```
| ?- a(X,Y).
X = 1
Y = 1 ?  ;
X = 1
Y = 2 ?  ;
X = 2
Y = 1 ?  ;
X = 2
Y = 2 ?  ;
X = 3
Y = 3
yes
```

```
1 1 Call:  a(_16,_17) ?
2 2 Call:  b(_16) ?
2 2 Exit:  b(1) ?
3 2 Call:  c(_17) ?
3 2 Exit:  c(1) ?
1 1 Exit:  a(1,1) ?
X = 1
Y = 1 ?  ;
1 1 Redo:  a(1,1) ?
3 2 Redo:  c(1) ?
3 2 Exit:  c(2) ?
1 1 Exit:  a(1,2) ?  ...
```

Aufgabe 8.3 Cut

b)

TECHNISCHE UNIVERSITÄT DARMSTADT

b) Betrachten Sie nun folgende Veränderungen der ersten Zeile. Wie verändern sich jweils die Menge der gefundenen Lösungen? Warum?

1. `a(X,Y) :- !, b(X), c(Y).`
   - alternative Regel mit Head `a(X,Y)` kann wird nicht angewandt
   - Backtracking auf den Literalen nach dem Cut weiterhin möglich
   - nur die Lösung `a(3,3)` wird abgeschitten
2. `a(X,Y) :- b(X), !, c(Y).`
   - auch die Belegung der Variable in `b(X)`, darf nicht mehr revidiert werden
   - nur die Lösungen `a(1,1)` und `a(1,2)` werden gefunden
3. `a(X,Y) :- b(X), c(Y), !.`
   - jetzt darf nach der ersten Lösung gar nichts mehr rückgängig gemacht werden
   - `a(1,1)` bleibt einzige Lösung

Aufgabe 8.4 Listen

TECHNISCHE UNIVERSITÄT DARMSTADT

Betrachten Sie folgendes Codestück:

```
L = [X,Y,X], L = [L1 | L2], L2=[a,b]
```

Als Query:

```
| ?- L = [X,Y,X], L = [L1 | L2], L2=[a,b].

L = [b,a,b]
L1 = b
L2 = [a,b]
X = b
Y = a

yes
```

Interpreter findet die einzige Lösung mit `L1 = b`

Aufgabe 8.5 Listen II

TECHNISCHE UNIVERSITÄT DARMSTADT

Vollziehen Sie die Abarbeitung der folgenden Queries nach, wobei die die Prädikate `union/2` und `append/3` wie in den VL-Folien definiert seien.

a) `union([a,b],[c,a,d],E).`

b) `append(X,[3,4],[2,3,4]).`

Die Definitionen:

```
union([],A,A)                :-  !.
union([A|B],C,D)             :-  member(A,C), !, union(B,C,D).
union([A|B],C,[A|D])         :-  union(B, C, D).

append([H|L1],L2,[H|L3])     :-  append(L1,L2,L3).
append([],L,L).
```

GNU-Prolog erlaubt nicht, dass `append` neu definiert wird; verwende SWI-Prolog.

Aufgabe 8.5 Listen II

a)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
union([a|[b]],[c,a,d],E) → member(a,[c,a,d]), !, union([b],[c,a,d],E)
     member(a,[c,a,d]) → Exit
     !  → Exit
     union([b|[]],[c,a,d],E) → member(b,[c,a,d]), !, union([],[c,a,d],E)
          member(b,[c,a,d]) → Fail
     union([b|[]],[c,a,d],E) → E=[b|F], union([],[c,a,d],F)
          union([],[c,a,d],F) → F=[c,a,d], !
               !  → Exit
```

E=[b|F] und F=[c,a,d] macht E = [b,c,a,d]

Aufgabe 8.5 Listen II

b)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
append(X,[3,4],[2,3,4]) → X=[H_1|L_1], H_1=2, append(L_1,[3,4],[3,4])
      append(L_1,[3,4],[3,4]) → L_1=[H_2|L_2],H_2=3, append(L_2,[3,4],[4])
            append(L_2,[2,3,4],[4]) → L_2=[H_3|L_3], H_3=4
                  append(L_3,[2,3,4],[]) → Fail
            Fail
      append(L_1,[3,4],[3,4]) → L_1=[], Exit
```

`X=[H_1|L_1], H_1=2, L_1=[]` macht zusammen `X = [2]`