Introduction to Data and Knowledge Engineering SS10 – Übung 7

TECHNISCHE UNIVERSITÄT DARMSTADT

Aufgabe 7.1: Ein Katalog

Prädikate

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorhandene Prädikate:

```
product(Manufacturer, Model, Type)
pc(Model, Clockrate, RamSize, HDsize, Price)
laptop(Model, Clockrate, RamSize, HDSize, Resolution, Price)
printer(Model, Color, PType, Price)
```

Aufgabe 7.1: Hardware Katalog

Argumente der Prädikate

TECHNISCHE UNIVERSITÄT DARMSTADT

Bedeutung der Argumente:

`Manufacturer`	Herstellername
`Model`	Eindeutige Modellbezeichnung
`Type`	Gerätetyp (’pc’, ’laptop’ oder ’printer’)
`Clockrate`	Taktfrequenz des Prozessors in GHz
`RamSize:`	Arbeitsspeicher in GB
`HDSize`	Festplattengröße in GB
`Price`	Preis in Euro
`Resolution`	Bildschirmauflösung in Pixel pro Zoll
`Color`	Farbdrucker (’true’ oder ’false’)
`PType`	Druckerart (’ink’ oder ’laser’)

Aufgabe 7.1: Ein Katalog

a) - c)

TECHNISCHE UNIVERSITÄT DARMSTADT

a) Welche PC-Modelle haben eine Taktfreqenz von mindestens 1200 MHz?

```
a(A) :- pc(A, B, _, _, E), B >= 1.2 .
```

b) Welche Hersteller fertigen Notebooks, die 1GB RAM und eine Bildschirmauflösung von mindestens 50 ppi haben?

```
b(A)   :-   product(A, B, C), laptop(B, D, E, F, G, H),
            E = 1, G >= 50 .
```

c) Finden Sie die Modellbezeichnungen und Preise aller Produkte von IBM.

```
c(A, F) :- product('IBM', A, B), pc(A, C, D, E, F).
c(A, G) :- product('IBM', A, B), laptop(A, C, D, E, F, G).
c(A, E) :- product('IBM', A, B), printer(A, _, _, E).
```

Aufgabe 7.1: Ein Katalog

d) - f)

TECHNISCHE UNIVERSITÄT DARMSTADT

d) Finden Sie die Modellbezeichnungen aller Farblaserdrucker.

```
d(A) :- printer(A, true, laser, B).
```

e) Finden Sie die Festplattengrößen, die in mindestens zwei PCs vorkommen.

```
e(D) :- pc(A, B, C, D, E), pc(F, G, H, D, I), A\=F.
```

f) Erstellen Sie eine Relation, die eine Preisliste für alle Produkte erstellt.

```
prices(Man,Prod,Price) :-  product(Man,Prod,_),
                           pc(Prod,_,_,_,Price).
prices(Man,Prod,Price) :-  product(Man,Prod,_),
                           laptop(Prod,_,_,_,_,Price).
prices(Man,Prod,Price) :-  product(Man,Prod,_),
                           printer(Prod,_,_,Price).
```

Aufgabe 7.2: Negation

Prädikate

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorhandene Prädikate und ihre Bedeutung:

`frequents(D,P)`	Trinker `D` besucht die Kneipe `P`
`serves(P,B)`	Bar `P` schenkt Bier `B` aus
`likes(D,B)`	Trinker `D` mag das Bier `B`

Aufgabe 7.2: Negation

a),b)

TECHNISCHE UNIVERSITÄT DARMSTADT

a) `happy(D)` ist wahr, genau dann wenn Trinker `D` wenigstens eine Bar besucht, die ein Bier ausschenkt, das er mag.

```
happy(D) :-  frequents(D,P), serves(P,B), likes(D,B).
```

b) `should_visit(D,P)` ist wahr, genau dann wenn Bar `P` ein Bier ausschenkt, dass Trinker `D` mag.

```
should_visit(Wand,Ungerade) :-  serves(Ungerade,Y2Ld_A),
                                likes(Wand,Y2Ld_A).
```

Aufgabe 7.2: Negation

c)

TECHNISCHE UNIVERSITÄT DARMSTADT

c) `very_happy(D)` ist wahr, genau dann wenn jede Bar, die Trinker `D` besucht, wenigstens ein Bier ausschenkt, das er mag.

Problem der Aufgabenstellung:

- `D` soll ein Trinker sein, was genau ein Trinker ist, wird nicht gesagt
- wenn `D` gar keine Bar besucht, ist das aber wichtig
- wir müssen es selbst festlegen:

```
drinker(D) :- likes(D,_).
drinker(D) :- frequents(D,_).
```

Aufgabe 7.2: Negation

c)

TECHNISCHE UNIVERSITÄT DARMSTADT

c) `very_happy(D)` ist wahr, genau dann wenn jede Bar, die Trinker `D` besucht, wenigstens ein Bier ausschenkt, das er mag.

Allquantifizierung

- Variablen, die nur im Body vorkommen, sind $\exists$-quantifiziert
- hier wird Aussage über *alle* Bars (die `D` besucht) gemacht
- Umformulierung mit Negation

  Es ist falsch, dass eine Bar `P` existiert, die `D` besucht und in der es kein Bier `B` gibt, das Trinker `D` mag.
- halb formalisiert:

  Es gilt `drinker(D)`, und es gilt nicht, dass ein Bar `P` existiert, für die `frequents(D,P)` aber nicht `should_visit(D,P)` gilt.

Aufgabe 7.2: Negation

c) Fortsetzung

TECHNISCHE UNIVERSITÄT DARMSTADT

Zusammengefasst als Pseudo-Datalog-Regel:

```
very_happy(D) :-  drinker(D),
                  not( frequents(D,P), not( should_visit(D,P))).
```

Geht so nicht.

- `not` darf nur auf Atome angewendt werden (nicht auf Konjuktionen)
- Variablen in diesen Atomen sollten *beschränkt* sein, d.h. nur endlich viele Werte annehmen können
- letzteres wird erreicht durch vorangestellte Bedingung

Stattdessen:

```
not_should_visit(D,P) :- drinker(D), serves(P,_),
                         not( should_visit(D,P) ).
not_very_happy(D)     :- frequents(D,P), not_should_visit(D,P).
very_happy(D)         :- drinker(D), not( not_very_happy(D) ).
```

Aufgabe 7.2: Negation
d)

TECHNISCHE UNIVERSITÄT DARMSTADT

d) `sad(D)` ist wahr genau dann wenn Trinker `D` keine Bar besucht, die ein Bier ausschenkt, dass er mag.

```
sad(D) :-  drinker(D), not( happy(D)).
```

Bermerkungen zur vorangestellte Bedingung:

- Mit `sad(D) :- not( happy(D))` würde die Antwort auf query `sad(33212))` positiv. Wollen wir das?
- Nein, hier wird explizit verlangt das `D` ein Trinker ist.
- Aber selbst wenn es nicht explizit verlangt wäre, ist Variablen die innerhalb des `not(..)` vorkommen zu beschränken.
- Die Antwort auf query `sad(X))` müsste sonst auch `not(happy(3))`, `not(happy( nie_gehoert)` etc. liefern.
- Datalog ist aber nicht so blöd, das alles auszugeben.

Aufgabe 7.3: EPP

1. Iteration

TECHNISCHE UNIVERSITÄT DARMSTADT

Mittels des EPP (Elementary Production Principle) wurden die folgenden Fakten erzeugt. Für jede Iteration sind jeweils nur die *neu hinzukommenden* Fakten angegeben. Nach der 3. Iteration ist der Fixpunkt erreicht. Geben Sie ein Datalog-Programm an, das genau dieses Verhalten hervorruft.

1. Iteration: `r(a,b), r(a,c), r(b,c), r(c,d), r(d,e)`

Die 1. Iteration liefert ausschließlich Fakten, die im Datalog-Programm definiert sind. Also muss das Programm folgende Zeilen enthalten:

```
r(a,b).
r(a,c).
r(b,c).
r(c,d).
r(d,e).
```

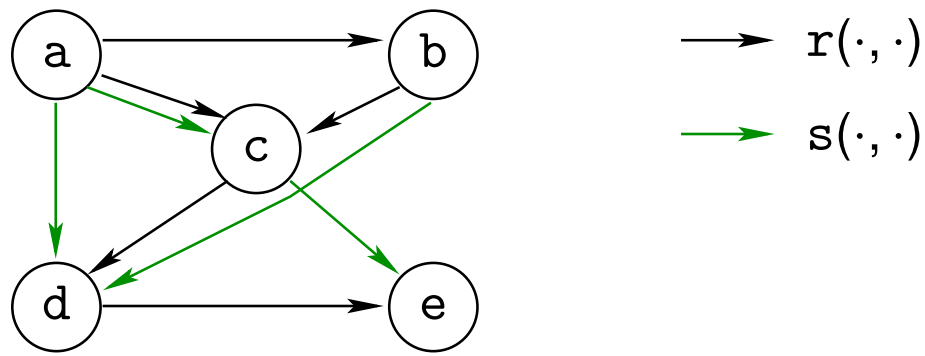
Aufgabe 7.3: EPP

2. Iteration

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Iteration: `r(a,b), r(a,c), r(b,c), r(c,d), r(d,e)`
2. Iteration: `s(a,c), s(a,d), s(b,d), s(c,e)`

- neue Fakten sind nicht als Fakten im Programm gegeben, da sie sonst schon nach der 1. Iteration aufgetaucht wären
- wir suchen eine Regel deren Head die Form `s(X,Y)` und deren Body aus Termen der Form `r(X,Y)` besteht
- binäre Relation lassen sich als Graph darstellen:



- rate eine Regel: `s(X,Y) :- r(X,Z), r(Z,Y).`
- die Regel passt: sie erzeugt genau die Fakten nach 2. aus den Fakten nach 1.

Aufgabe 7.3: EPP

3. Iteration

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Iteration: `r(a,b), r(a,c), r(b,c), r(c,d), r(d,e)`
2. Iteration: `s(a,c), s(a,d), s(b,d), s(c,e)`
3. Iteration: `s(c,a), s(d,a), s(d,b), s(e,c)`

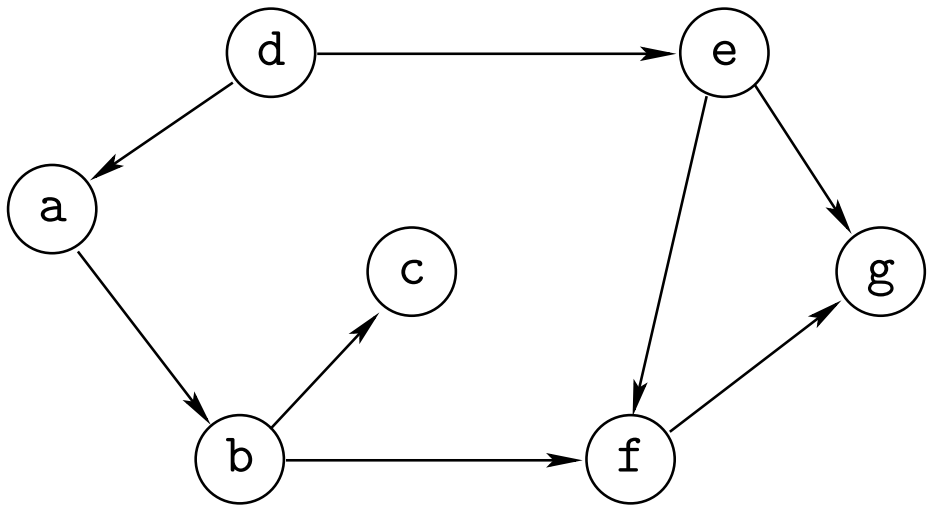
- eine Regel mit `s` im Head wurde angewandt
- es muss eine zweite Regel geben, die `s` im Body hat, da sonst die neuen Fakten nach 3. schon nach 2. erschienen wären
- scharfes Hinsehen liefert: `s(X,Y) :- s(Y,X).`
- Probe durch Anwendung beider Regeln auf Fakten bis einschließlich 2.: stimmt.

Aufgabe 7.4: Erreichbarkeit in einem Graphen

Graph, a)

TECHNISCHE UNIVERSITÄT DARMSTADT

Gegeben sei folgender gerichteter Graph:



a) Das Prädikat `edge(X,Y)` beschreibt eine gerichtete Kante von X nach Y. Benutzen sie dieses Prädikat, um die entsprechenden Kanten des Graphen zu modellieren.

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

Aufgabe 7.4: Erreichbarkeit in einem Graphen

b),c)

TECHNISCHE UNIVERSITÄT DARMSTADT

b) Das Prädikat `reachable(X,Y)` besagt, dass der Knoten Y vom Knoten X aus erreicht werden kann. Definieren Sie dieses Prädikat.

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

c) Erstellen Sie eine Anfrage, die Ihnen alle Paare von erreichbaren Knoten ausgibt.

```
reachable(X,Y)
```

Aufgabe 7.4: Erreichbarkeit in einem Graphen

d)

TECHNISCHE UNIVERSITÄT DARMSTADT

d) Das Prädikat `odd(X,Y)` besagt, dass zwischen Knoten Y und Knoten X ein Weg ungerader Länge besteht. Definieren Sie dieses Prädikat. Zwischen Welchen Knoten besteht ein Weg ungerader Länge?

```
odd(X,Y)    :-  edge(X,Y).
odd(X,Z)    :-  even(X,Y), edge(Y,Z).
even(X,Z)   :-  odd(X,Y), edge(Y,Z).
odd(X,Y)
```

Oder ohne das Hilfsprädikats `even`:

```
odd(X,Y)   :-  edge(X,Y).
odd(X,Y)   :-  edge(X,Z1), edge(Z1,Z2), odd(Z2,Y).
```

Datalog

Linux Installation and Start

TECHNISCHE UNIVERSITÄT DARMSTADT

Mit GNU-Prolog und Laden von DES per Kommandozeilen-Argument

- DES von seiner homepage laden
- `tar xzf DES1.6.2Linux.tar.gz`
- `cd des/`
- `cp system/gnu/* .`
- `gprolog -entry-goal ['des']`