SEIII: Logikprogrammierung 18. November 2016
(L. Kruse: 6659482, A. Hildebrandt: 6688865, E. Baalmann: 6704003)

Übungsblatt 04: Deduktive Datenbanken

Aufgabe 1: Eigenschaften von Relationen

A ist das Geburtsjahr von B:

- nicht symmetrisch: ein Jahr kann nicht das Geburtsjahr von einem anderen Jahr sein
- nicht reflexiv: ein Mensch kann nicht das Geburtsjahr von einem Jahr sein
- nicht transitiv: mit der selben Relation wäre ein mögliches C wieder ein Jahr
- funktional: ein Mensch kann nur ein Geburtsjahr haben

A ist Nachbargrundstück von B:

- symmetrisch: B ist ebenfalls Nachbar von A
- nicht reflexiv: man ist nicht sein eigener Nachbar
- nicht transitiv: der Nachbar meines Nachbarn kann auf der anderen Seite liegen und ist somit nicht mein Nachbar
- nicht funktional: man kann viele Nachbarn haben

A ist leichter als B:

- nicht symmetrisch: B ist schwerer als A
- nicht reflexiv: man kann nicht leichter als man selbst sein
- transitiv: wenn A leichter als B und B leichter als C ist, dann ist A auch leichter als C
- nicht funktional: es gibt viele Elemente, die leichter als etwas sein können

A ist kleiner oder gleich B:

- symmetrisch: kann symmetrisch sein, wenn A und B gleich sind
- reflexiv: A gleich A ist immer erfüllt
- transitiv: Wenn A kleiner als B ist und B kleiner als C, dann ist A auch kleiner als C
- nicht funktional: es kann viele Elemente geben, die kleiner gleich etwas sind

A hat schon einmal in einer Mannschaft mit B gespielt:

- symmetrisch: B war immer auch schon mit A in einer Mannschaft
- reflexiv: man hat immer mit sich selbst in einer Mannschaft gespielt
- nicht transitiv: B kann auch schon in einer Mannschaft gespielt haben, in der A nicht war
- nicht funktional: Man kann in vielen Mannschaften spielen

A ist kongruent zu B:

- symmetrisch: beschreiben die selbe Menge, also ist auch B kongruent zu A
- reflexiv: A ist immer kongruent zu sich selbst
- transitiv: Wenn A und B kongruent sind, und B und C, dann sind auch A und C kongruent
- nicht funktional: A kann kongruent zu mehreren Dingen sein

0.1 Aufgabe 2: Deduktive Datenbanken - Hierarchische Strukturen

1.

```
%oberKategorieZu(?ID, ?NAME, ?OBERKATEGORIE)
oberKategorieZu(ID, ONAME, OBERKATEGORIEID):-
kategorie(ID,_,OBERKATEGORIEID),
kategorie(OBERKATEGORIEID, ONAME, _).

oberKategorieZu(ID, ONAME, OBERKATEGORIEID):-
kategorie(ID,_,UNTERKATEGORIEID),
oberKategorieZu(UNTERKATEGORIEID, ONAME, OBERKATEGORIEID).
```

2.

```
%pfad(?ID, ?PFAD)
pfad(ID, PFAD) :-
findall(NAME, oberKategorieZu(ID,NAME, _), LISTE),
reverse(LISTE, REVLISTE),
append(REVLISTE, [IDNAME], PFAD),
kategorie(ID, IDNAME, _).
```

3.

```
%gibt die Ergebnisse einzeln aus
%produkteZuKategorie(?OID, ?PId)
produkteZuKategorie(OID, PId) :-
produkt(PId, OID, _,_,_,_,_).

produkteZuKategorie(OID, PId) :-
kategorie(UID, _, OID) , produkteZuKategorie(UID, PId).

%speichert das Ergebnis in einer Liste
%produktListeZuKategorie(+OID, ?Produktliste)
produktListeZuKategorie(OID, Produktliste) :-
findall(PId, produkteZuKategorie(OID, PId), Produktliste).
```

4.

```
%zaehlt die Elemente einer Produktliste
%anzahlProdukte(+OID, ?ANZAHL)
anzahlProdukte(OID, ANZAHL) :-
produktListeZuKategorie(OID, Produktliste), length(Produktliste, ANZAHL).
```

5.

```
%verkaufteProdukte(+KID, ?Anzahl)
verkaufteProdukte(KID, Anzahl) :-
findall(A, (produkteZuKategorie(KID, PId) , verkauft(PId, 2016, _, A)), L),
sumlist(L, Anzahl).
```

Aufgabe 3: Deduktive Datenbanken

1.

```
%prueft rekursiv, ob es eine Weiche zwischen zwei Gleisen nach rechts gibt
%verbindungRechts(?GleisAS1,?GleisAS2)
verbindungRechts(GleisAS1,GleisAS2) :-
weiche(_,GleisAS1,GleisAS2,_).

verbindungRechts(GleisAS1,GleisAS2) :-
weiche(_,GleisAS1,X,_),
verbindungRechts(X,GleisAS2).


%prueft rekursiv, ob es eine Weiche zwischen zwei Gleisen nach links gibt
%verbindungLinks(?GleisAS1,?GleisAS2)
verbindungLinks(GleisAS1,GleisAS2) :-
```

```
weiche(_,GleisAS2,GleisAS1,_).

verbindungLinks(GleisAS1,GleisAS2) :-
weiche(_,X,GleisAS1,_),
verbindungLinks(X,GleisAS2).


%sucht links oder rechts nach Verbindungen zwischen zwei Gleisen
%verbindungMoeglich(?Gleis1,?Gleis2)
verbindungMoeglich(Gleis1,Gleis2) :-
verbindungRechts(Gleis1,Gleis2).

verbindungMoeglich(Gleis1,Gleis2) :-
verbindungLinks(Gleis1,Gleis2).


%prueft, ob zwei angegebene Gleise wirklich Gleise sind und
%sucht dann nach einer Verbindung
%verbinung(?Gleis1,?Gleis2)
verbindung(Gleis1,Gleis2) :-
gleis(Gleis1,_,_),
gleis(Gleis2,_,_),
verbindungMoeglich(Gleis1,Gleis2).
```

2.

```
%prueft rekursiv, ob es eine Weiche zwischen zwei Gleisen
%nach rechts gibt, und ob die dabei befahrenen Gleise frei sind
%verbindungRechts(?GleisAS1,?GleisAS2)
verbindungRechts(GleisAS1,GleisAS2) :-
weiche(_,GleisAS1,GleisAS2,_),freiesGleis(GleisAS2).

verbindungRechts(GleisAS1,GleisAS2) :-
weiche(_,GleisAS1,X,_),freiesGleis(X),verbindungRechts(X,GleisAS2).


%prueft rekursiv, ob es eine Weiche zwischen zwei Gleisen
%nach links gibt, und ob die dabei befahrenen Gleise frei sind
%verbindungLinks(?GleisAS1,?GleisAS2)
verbindungLinks(GleisAS1,GleisAS2) :-
weiche(_,GleisAS2,GleisAS1,_),freiesGleis(GleisAS2).

verbindungLinks(GleisAS1,GleisAS2) :-
weiche(_,X,GleisAS1,_),freiesGleis(X),verbindungLinks(X,GleisAS2).
```

```
%sucht links oder rechts nach Verbindungen zwischen zwei Gleisen
%verbindungMoeglich(?Gleis1,?Gleis2)
verbindungMoeglich(Gleis1,Gleis2) :-
verbindungRechts(Gleis1,Gleis2).

verbindungMoeglich(Gleis1,Gleis2) :-
verbindungLinks(Gleis1,Gleis2).


%prueft, ob zwei angegebene Gleise wirklich Gleise sind
%und sucht dann nach einer Verbindung
%verbindung(Gleis1, Gleis2)
verbindung(Gleis1,Gleis2) :-
gleis(Gleis1,_,_),gleis(Gleis2,_,_),verbindungMoeglich(Gleis1,Gleis2).


%dynamisches Praedikat, um anzugeben, ob ein Gleis belegt ist
%belegung(?GleisAS)
:- dynamic belegung/1.
belegung(g1).


%prueft, ob ein Gleis frei ist
%freiesGleis(+GleisAS)
freiesGleis(GleisAS) :- \+ belegung(GleisAS).


%belegt ein Gleis, wenn es vorher noch nicht belegt war
%belegeGleis(+GleisAS)
belegeGleis(GleisAS) :-
freiesGleis(GleisAS),assertz(belegung(GleisAS)).


%gibt ein Gleis wieder frei
%macheGleisFrei(+GleisAs)
macheGleisFrei(GleisAS) :- retractall(belegung(GleisAS)).
```

3.

```
%prueft, ob ein GleisAbschnitt ueberhaupt ein Gleis ist,
und wenn ja, prueft es, ob die Zuglaenge kuerzer als der Gleis ist
%gleisAbschnittLangGenug(?GleisAS, Zuglaenge)
```

```
gleisAbschnittLangGenug(GleisAS,Zuglaenge) :-
gleis(GleisAS,_,_)->(gleis(GleisAS,GL,_),GL>=Zuglaenge);true.


%prueft rekursiv, ob es eine Weiche zwischen zwei Gleisen nach rechts
%gibt, ob die dabei befahrenen Gleise frei sind und
%der Zug nicht zu lang fuer diese ist
%verbindungRechts(?GleisAS1,?GleisAS2, ?Zuglaenge)
verbindungRechts(GleisAS1,GleisAS2,Zuglaenge) :-
weiche(_,GleisAS1,GleisAS2,_),
freiesGleis(GleisAS2),
gleisAbschnittLangGenug(GleisAS2,Zuglaenge).

verbindungRechts(GleisAS1,GleisAS2,Zuglaenge) :-
weiche(_,GleisAS1,X,_),freiesGleis(X),
gleisAbschnittLangGenug(X, Zuglaenge),verbindungRechts(X,GleisAS2,Zuglaenge).

%prueft rekursiv, ob es eine Weiche zwischen zwei Gleisen nach links
%gibt, ob die dabei befahrenen Gleise frei sind und
%der Zug nicht zu lang fuer diese ist
%verbindungLinks(?GleisAS1,?GleisAS2, ?Zuglaenge)
verbindungLinks(GleisAS1,GleisAS2,Zuglaenge) :-
weiche(_,GleisAS2,GleisAS1,_),
freiesGleis(GleisAS2),
gleisAbschnittLangGenug(GleisAS2,Zuglaenge).

verbindungLinks(GleisAS1,GleisAS2,Zuglaenge) :-
weiche(_,X,GleisAS1,_),freiesGleis(X),
gleisAbschnittLangGenug(X, Zuglaenge),verbindungLinks(X,GleisAS2,Zuglaenge).


%sucht links oder rechts nach Verbindungen zwischen zwei Gleisen
%verbindungMoeglich(?Gleis1,?Gleis2, ?Zuglaenge)
verbindungMoeglich(Gleis1,Gleis2,Zuglaenge) :-
verbindungRechts(Gleis1,Gleis2,Zuglaenge).

verbindungMoeglich(Gleis1,Gleis2,Zuglaenge) :-
verbindungLinks(Gleis1,Gleis2,Zuglaenge).


%prueft, ob zwei angegebene Gleise wirklich Gleise sind
%und sucht dann nach einer Verbindung
%verbindung(?Gleis1,?Gleis2, ?Zuglaenge)
verbindung(Gleis1,Gleis2,Zuglaenge) :-
```

```
gleis(Gleis1,_,_),
gleis(Gleis2,_,_),
verbindungMoeglich(Gleis1,Gleis2,Zuglaenge).
```

4.

"verbindung"wurde seit 3. nicht mehr veraendert.

```
%ankunft(?Von_Ort,?ZielGleis,?Zuglaenge)
ankunft(Von_Ort,ZielGleis,Zuglaenge) :-
einfahrt(StartGleis,Von_Ort),einfahrt(ZielGleis,_),
verbindung(StartGleis,ZielGleis,Zuglaenge).


%abfahrt(?Nach_Ort,?StartGleis,?Zuglaenge)
abfahrt(Nach_Ort,StartGleis,Zuglaenge) :-
einfahrt(ZielGleis,Nach_Ort),einfahrt(StartGleis,_),
verbindung(StartGleis,ZielGleis,Zuglaenge).
```