Logikprogrammieren Hausaufgaben

Blatt 3

Morten Seemann 6945442, Tore Wiedenmann 6488837

Aufgabe 1: Erklären Sie die Begriffe "Suche", "Variable", und "Instanziierung" im Hinblick auf ihre Rolle in der Logikprogrammierung.

Suche Bei einer Anfrage an die Datenbank wird versucht die Klauseln in der Anfrage mit Einträgen der Datenbank zu Unifizieren. Dies geschieht durch wiederholte suche einer passenden Variablenbindung für offene Variablen in der Anfrage. Die hierbei in Prolog verwendete Suche für Variablenbindungen ist vom Depth-first Ansatz. Wenn ein Suchanlauf fehlschlägt wird am letzten Entscheidungspunkt erneut die Suche nach Alternativen angefangen.

Variable Eine Variable ist in einer Suchanfrage die Information die von Interesse ist. Während der Anfrage wird versucht die Variablen mit Werten der Datenbank oder anderen in der Suche auftauchenden Variablen zu unifizieren um eine für alle Klauseln gültige Variablenbindung zu finden.

Instanziierung

Aufgabe 2: Die Datei "dateiverzeichnis.pl" enthält eine relationale Repräsentation für ein Dateiverzeichnis als Faktensammlung.
Geben Sie die möglichen Instanziierungsvarianten für Ihre Prädikatsdefinitionen an.

1.) Ein Prädikat, das Dateinamen und Dateischlüssel ineinander umrechnen kann.

```
name_zu_id(N,ID) :- file(ID,_,N,_,_,_).
```

2.) Ein Prädikat, das Verzeichnisnamen und Verzeichnisschlüssel ineinander umrechnen kann.

```
dir_name_zu_id(N,ID) :- directory(ID,N,_,_,_).
```

Man kann sowohl hier als auch in 1.) in beide richtungen Namen und IDs ineinander umwandeln. z.B.

```
?- name_zu_id(paris,ID).
ID = 4 ;
ID = 13.

?- name_zu_id(NAME,1).
NAME = in_the_summertime.
```

3.) Ein Prädikat, das für einen gegebenen Dateinamen, den Namen und den Schlüssel des zugehörigen Verzeichnisses ermittelt.

```
liegt_in_dir(N,DIR,ID)  :-
        file(FID,ID,N,_,_,_),
        name_zu_id(N,FID),
        directory(ID,DIR,_,_,_).
```

4.) Ein Prädikat, das für einen gegebenen Verzeichnisnamen, den Namen und den Schlüssel des unmittelbar übergeordneten Verzeichnisses ermittelt.

```
parent_dir(P,PID,DIR) :-
        dir_name_zu_id(DIR,ID),
        directory(ID,DIR,PID,_,_),
        dir_name_zu_id(P,PID).
```

Auch diese Prädkat beiden Prädikate sind richtungsunabhängig, Beispiele hier für letzteres:

```
?- parent_dir_von(P,PID,hochzeit).
P = bilder,
PID = 2 ;
P = dokumente,
PID = 4.



?- parent_dir_von(root,1,DIR).
DIR = bilder ;
DIR = musik ;
DIR = dokumente ;
false.
```

Aufgabe 3: Definieren Sie für die Arbeit mit der Verzeichnis-Datenbank aus Aufgabe 2 die folgenden Prädikate.

1.)Ein Prädikat, das eine Liste aller Dateinamen ermittelt, die in einem bestimmten Verzeichnis enthalten sind. Das Verzeichnis soll durch seinen Schlüssel identifiziert werden.

```
von_files_in(LISTE,DID) :-
        findall(FILE,
                liegt_in_dir(FILE,_,DID),
                LISTE).
```

2.) Ein Prädikat, das eine Liste der Namen aller Unterverzeichnisse ermittelt, die in einem bestimmten Verzeichnis enthalten sind. Das Verzeichnis soll wiederum durch seinen Schlüssel identifiziert werden.

```
von_child_dir(LISTE,DID) :-
        findall(DIR,
                parent_dir_von(_,DID,DIR),
                LISTE).
```

3.) Ein Prädikat, das die Anzahl aller Dateien in einem bestimmten Verzeichnis ermittelt. Das Verzeichnis soll auch hier durch seinen Schlüssel identifiziert werden.

```
anzahl_von_files_in_dir(N,DID) :-
        von_files_in(LIST,DID),
        length(LIST,N).
```

Aufgabe 4: Definieren Sie für die Datenbank "familie.pl" aus Aufgabenblatt 2 die folgenden Prädikate:

1.) vorfahre-von(Vorfahre,Nachkommende), das für eine gegebene Person alle ihre Vorfahren berechnet.

```
vorfahre_von(VF,NF) :- kind_von(NF,VF).
vorfahre_von(VF,NF) :- kind_von(NF,X),vorfahre_von(VF,X).
```

unter der Verwendung des Hilfsprädikat

```
kind_von(K,E) :- vater_von(E,K);mutter_von(E,K).
```

Diese Implementierung ist richtungsunabhängig da mutter-von und vater-von richtungsunabhängig sind. Sie ist ebenfalls terminierungssicher

2.) erster-gemeinsamer-vorfahre(Person1,Person2,Vorfahre), das den ersten gemeinsamen Vorfahren von zwei Personen berechnet.

```
gemeinsamer_vorfahre(P1,P2,GV) :-
        vorfahre_von(GV,P1),
        vorfahre_von(GV,P2).

erster_gemeinsamer_vorfahre(P1,P2,EGV) :-
        gemeinsamer_vorfahre(P1,P2,EGV),
        not((vorfahre_von(EGV,AGV),
            gemeinsamer_vorfahre(P1,P2,AGV))).
```

Hierbei ist gemeinsamer-vorfahre bereits das in 3.) gesuchte Prädikat verwandt-mit

Aufgabe 5: Definieren Sie für die Arbeit mit einem Dateiverzeichnis ein Prädikat, dass das Änderungsdatum eines Verzeichnisses auf das aktuelle Datum setzt. Achten Sie auf eine geeignete Fehlerbehandlung, falls das zu ändernde Verzeichnis nicht existiert.

```
set_date(DID) :-
        date(D),directory(DID,DIR,PID,DC,_),
        retract(directory(DID,DIR,PID,DC,_)),
        asserta(directory(DID,DIR,PID,DC,D)).

set_date(DID,ERR) :-
        directory(DID,_,_,_,_,_),set_date(DID),
        ERR='Erfolgreich'.

set_date(DID,ERR) :-
        not(directory(DID,_,_,_,_,_)),
        ERR='Verzeichniss existiert nicht'.
```

Zusatzaufgabe 1.) Ein Prädikat, das in einem gegebenen Verzeichnis ein neues Unterverzeichnis hinzufügt. Stellen Sie sicher, dass das übergeordnete Verzeichnis existiert, keine Doppeleintragungen in einem Verzeichnis vorgenommen werden und die Angaben für das Modifikationsdatum am Ende der Transaktion für alle beteiligten Verzeichnisse auf dem aktuellen Stand sind.

```
als_subdir_hinzufuegen(_,P,PID,ERR) :-
        not(directory(PID,P,_,_,_)),
        ERR= 'Ungueltiges Grundverzeichniss'.

als_subdir_hinzufuegen(DIR,P,PID,ERR) :-
        directory(PID,P,_,_,_),
        directory(_,DIR,PID,_,_),
        ERR='Verzeichniss existiert bereits'.

als_subdir_hinzufuegen(DIR,P,PID,MSG) :-
        directory(PID,P,_,_,_),
        als_subdir_hinzufuegen(DIR,PID,MSG).

id(KEY,VAL):- set_flag(KEY,VAL).

naechste_id(X):- flag(directories,X,X+1).

set_all_dates(START,MSG) :-
        directory(START,_,PID,_,_),
        set_date(START,MSG),
        set_all_dates(PID,MSG).

set_all_dates(START,MSG) :-
```

```
24          not(directory(START,_,_,_,_)),
25          MSG='Erfolgreich'.
26
27  als_subdir_hinzufuegen(DIR,PID,MSG) :-
28          naechste_id(X),
29          date(D),
30          assertz(directory(X,DIR,PID,D,D)),
31          set_all_dates(PID,MSG).
```

Das Prädikat als-subdir-hinzufuegen(DIR,P,PID,MSG) fügt nun ein neues Verzeichniss in das System ein. Man muss vor dem allerersten hinzufügen einmalig die flag für directories auf 12 setzen.

Es überprüft zunächst ob P mit PID ein zulässiges Oberverzeichniss ist dann ob es das Verzeichniss DIR bereits in P existiert. Falls beides nicht scheitert so wird als-subdir-hinzufuegen/3 mit den gleichen Argumenten aufgerufen. Dort wird die neue ID sowie das aktuelle Datum berechnet und das Verzeichniss in die Datenbank eingetrage.

Schlussendlich werden durch set-all-dates/2 rekursiv von allen Oberverzeichnissen das Veränderungsdatum aktualisiert. set-all-dates ist dabei terminierungssicher, da es beim Aufruf von Verzeichniss mit ID=0 abbricht und "Erfolgreich" ausgiebt.

2.) Ein Prädikat, das in einem gegebenen Verzeichnis eine neue Datei hinzufügt. Stellen Sie auch hier sicher, das das angegebene Verzeichnis existiert, keine Doppeleintragungen in einem Verzeichnis vorgenommen werden und alle Angaben zum Modifikationsdatum aktualisiert wurden.

```
als_file_hinzufuegen(_,_,DIR,ID,MSG) :-
        not(directory(ID,DIR,_,_,_)),
        MSG='Ungueltiges Grundverzeichniss'.

als_file_hinzufuegen(FILE,_,DIR,ID,MSG) :-
        directory(ID,DIR,_,_,_),
        file(_,ID,FILE,_,_,_),
        MSG='Datei existiert bereits'.

als_file_hinzufuegen(FILE,SIZE,DIR,ID,MSG):-
        directory(ID,DIR,_,_,_),
        als_file_hinzufuegen(FILE,SIZE,DIR,MSG).

als_file_hinzufuegen(FILE,SIZE,DIR,MSG):-
        naechste_id(files,X),
        date(D),
        directory(ID,DIR,_,_,_),
        assertz(file(X,ID,FILE,SIZE,D,D)),
        set_all_dates(DIR,MSG).
```

Dies ist das Analogon zu Zusatzaufgabe 1.) für files. Hier muss man ebenfalls zu Anfang einmalig die flagge für "files" auf 34 initialisieren.

3.) Ein Prädikat, das den Verzeichniseintrag aktualisiert, wenn eine über ihren Schlüssel identifizierte Datei durch eine Applikation verändert wird.

```
file_changed(ID,MSG) :-
    file(ID,DID,NAME,SZ,DC,_),
    date(D),
    directory(DID,_,_,_,_,_),
    retract(file(ID,DID,NAME,SZ,DC,_)),
    assertz(file(ID,DID,NAME,SZ,DC,D)),
    set_all_dates(DID,MSG).
```

4.)
Vergleichen Sie Ihre Lösungen zu den Teilaufgaben 1 bis 3 mit dem im Bereich der Datenbanken benutzten Begriff der Transaktion. Diskutieren Sie die Anforderungen an eine Transaktion, die von Ihrer Implementation nicht erfüllt werden.

Bei der Ausführung von Transaktionen muss das Transaktionssystem die ACID-Eigenschaften garantieren:

Atomarität: Eine Transaktion wird entweder ganz oder gar nicht ausgeführt. Transaktionen sind also "unteilbar". Wenn eine atomare Transaktion abgebrochen wird, ist das System unverändert.

Unsere Transaktion prüft zwar zunächst ob sie durchgeführt werden kann, aber sobald sie während der ausführung abgebrochen wird sind bereits alle änderungen geschehen. Sie genügt also nicht der Atomarität.

Konsistenz: Nach Ausführung der Transaktion muss der Datenbestand in einer konsistenten Form sein, wenn er es bereits zu Beginn der Transaktion war.

Unsere Datenbank wird zwar nur verändert sofern es sich tatsächlich um legitime Eingabeargumente handelt, jedoch wird nicht überprüft ob die ID welche generiert wird korrekt ist. Es kann somit vorkommen dass wir plötzlich zwei Einträge zum Verzeichniss "root" besitzen, welche unterschiedliche Änderungsdaten haben. Sie ist somit nicht Konsistent.

Isolation: Bei gleichzeitiger Ausführung mehrerer Transaktionen dürfen sich diese nicht gegenseitig beeinflussen.

Da unsere Transaktion Einträge löscht und neu in die Datenbank schreibt könnte es bei gleichzeitiger ausführung passieren das eine Anfrage auf existenz eines Verzeichnisses "false" ausgiebt obwohl es normalerweise "true" wäre. Unsere Transaktion genügt nicht der Isolation.

Dauerhaftigkeit: Die Auswirkungen einer Transaktion müssen im Datenbestand dauerhaft bestehen bleiben. Die Effekte von Transaktionen dürfen also nicht verloren gehen oder mit der Zeit verblassen.

Da wir die eigentliche Prolog-Datei nie am Ende unserer Transaktion speichern gehen die vorgenommenen Änderungen nie Tatsächlich in den Datenbestand ein. Unsere Transaktion genügt also ebenfalls nicht der Dauerhaftigkeit.