Dr. Rita Hartel
u. v. a.

Paderborn, 18. Dezember 2023

Heimübung zur Vorlesung

Programmiersprachen

WS 2023/24

Blatt 2

Abgabe: 08. Januar 2024

Bearbeiten Sie dieses Übungsblatt in Gruppen von mindestens 2 bis maximal 3 Studierenden Die Abgabe erfolgt in Panda (an derselben Stelle an der Sie auch das Übungsblatt heruntergeladen haben).
Die Lösung muss in Form einer ZIP-Datei abgegeben werden. Diese beinhaltet:

- Eine PDF-Datei mit den Namen und Matrikelnummern aller Gruppenmitglieder und der Lösung für die Aufgabe 1.4
- Je eine Prolog-Datei (Endung .pl oder .txt) für die Prolog-Anweisungen der Lösungen der Aufgaben 1-3

Geben Sie pro Gruppe nur eine Lösung ab.

Der Aufgabenzettel verwendet die Regeln `not/1`, `notequals/2`, `member/2` und `nomember/2`. Verwenden Sie die Regeln wie angegeben, aber Sie dürfen die folgenden Implementationen dieser Regeln zum Testen verwenden. Die Funktionen `not/1` und `member/2` werden von Prolog schon bereit gestellt.

```
notequals(A, B):- A \= B.
nomember(A, B):-not(member(A, B)).
```

Aufgabe 1: (Nordische Götter)
Gegeben seien wieder die folgenden Fakten über die nordischen Götter.

```
asen(odin).
asen(thor).
asen(loki).
asen(tyr).

wane(freyr).
wane(freyja).
wane(skadi).

% begleiter(Begleiter, Gott).
begleiter(huginn, odin).
begleiter(muninn, odin).
```

```
begleiter(tanngnjostr, thor).
begleiter(tanngrisnir, thor).
begleiter(hildsvini, freyja).
begleiter(gullinborsti, freyr).
begleiter(fenrir, loki).
begleiter(fenrir, tyr).
```

1. Schreiben Sie eine Regel `einBegleiter/2`, die den Namen eines Gottes übergeben bekommt, `begleiter` verwendet und maximal eine:n Begleiter:in dieses Gottes zurückgibt.

2. Schreiben Sie eine Regel `alleBegleiterEinesWanen/1`, die die Namen aller Begleiter:innen ausgibt, die zum ersten Gott aus dem Geschlecht der `wane` gehören.

3. Schreiben Sie eine Regel `mehrfach/1`, die die Namen aller Begleiter:innen ausgibt, die mehr als einen Gott begleiten. Sie können die Regel `notequals/2` verwenden, um zu überprüfen, ob zwei Variablen den gleichen Inhalt haben.

4. Zeichnen Sie das 4-Port-Modell für die Regel `example/1`. Geben Sie Call, Fail, Exit und Redo Schritte des 4-Port-Modells für den Aufruf `example(G)` an (Ein Beispiel hierfür sehen Sie im Aufgabenblatt von Präsenzübung 2). Geben Sie die Schritte auch für den Cut an. Geben Sie auch die Bindungen von Variablen an, wenn diese gebunden werden.

   ```
   example(G):- begleiter(N, loki), !, begleiter(N, G).
   ```

Aufgabe 2: (Listen und Rekursion)

1. Schreiben Sie eine Regel `all_members/2`, die zwei Listen als Parameter enthält und genau dann `true` ergibt, wenn jedes Element der ersten Liste in der zweiten Liste enthalten ist.

   Hinweis: Sie müssen keine Häufigkeiten von Elementen prüfen.

   Hinweis: Sie dürfen die Funktion `member/2` verwenden.

2. Schreiben Sie eine Regel `remove_second_last/2`, die eine Liste erhält und als zweiten Parameter eine Liste zurück gibt, die das vorletzte Element nicht mehr enthält.

   Heinweis: Sie dürfen davon ausgehen, dass die übergebenen Listen immer mindestens die Länge 2 haben.

3. Schreiben Sie eine Regel `double_odd/2`, die als ersten Parameter eine Liste erhält und als zweiten Parameter eine Liste zurück gibt, bei der jede ungerade Zahl verdoppelt worden ist.

   Heinweis: Für gerade Zahlen gilt, dass `0 is X mod 2` `true` ergibt, während für ungerade Zahlen gilt, dass `1 is X mod 2` `true` ergibt. Wenn Sie `Y is X * 2` aufrufen, sorgt das dafür, dass `Y` den doppelten Wert von `X` enthält. (Näheres dazu in der Vorlesung am Montag, den 18.12.2023.)

4. Schreiben Sie eine Regel `sum_neighbours/2`, die eine Liste als ersten Parameter bekommt und die eine Liste als zweiten Parameter zurückgibt, bei der von immer zwei aufeinander folgenden Elementen die Summe berechnet worden ist. Bei Listen ungerader Länge soll die Summe der letzten drei Elemente berechnet werden. Da für Listen der Länge 1 nicht die letzten drei Elemente addiert werden können, soll für Listen der Länge 1 die Funktion immer `false` ergeben.

   Z.B. ergibt ein Aufruf von `?-sum_neighbours([1,2,3,4,5],R)` das Ergebnis `R=[3,12]`, da 1+2 = 3 und 3+4+5 = 12 und ein Aufruf von `?-sum_neighbours([1,2,3,4,5,6],R)` das Ergebnis `R=[3,7,11]`, da $1 + 2 = 3$ und $3 + 4 = 7$ und $5 + 6 = 11$.

Aufgabe 3: (Rätsel und Rekursion)

Benutzen Sie Prolog, um die folgenden Rätsel zu lösen. Geben Sie als Lösung sowohl den Prolog-Code an, den Sie verwendet haben, als auch die Lösung(en) in Worten.

1. Auf einem Marktplatz befinden sich drei Männer namens Albert, Siegfried und Roderich. Einer von ihnen ist ein Ritter, einer ein Knappe und einer ein Spion. Es ist im ganzen Königreich bekannt, dass Ritter stets die Wahrheit sagen, Knappen immer lügen und Spione sowohl lügen als auch die Wahrheit sagen können. Welcher der drei ist der Ritter, welcher der Knappe und wer ist der Spion?

   - Siegfried sagt: Albert ist ein Ritter.
   - Roderich sagt: Ich bin ein Spion.
   - Albert sagt: Roderich ist ein Knappe.

   (a) Erstellen Sie nun Fakten `aussage/4` für die drei Aussagen. Der erste Parameter soll den Namen der Person zurückgeben, die die Aussage tätigt. Der zweite Parameter soll den Namen des Ritters zurückgeben, der dritte Parameter den Namen des Knappen und der vierte Parameter den Namen des Spions.

   Z.B. Siegfried sagt nichts über die Person aus, die ein Knappe ist, daher muss der dritte Parameter ungebunden sein.

   Verwenden Sie zum Modellieren der drei Rollen die Konstanten `siegfried, albert` und `roderich`.

   (b) Erstellen Sie nun eine Regel `rollenverteilung/3`, die alle Kombinationen von Personen auf die Rollen zurück gibt. Nicht erlaubte Rollenverteilungen in diesem Aufgabenteil sind Rollenverteilungen, bei denen eine Person zwei Rollen inne hat. Der erste Parameter soll den Namen des Ritters zurückgeben, der zweite Parameter den Namen des Knappen und der dritte Parameter den Namen des Spions.

   Hinweis: Sie dürfen die Regeln `member/2` und `permutation/2` verwenden. Man kann diese Aufgabe nur mit Fakten oder auch nur mit `member/2` lösen, besonders elegante Lösungen verwenden aber `permutation/2` geschickt.

   Hinweis: `permutation` wird von Prolog mitgebracht kann dazu verwendet werden, alle Permutationen einer Liste zu berechnen. Z.B. ergibt ?- `permutation([1,2], [X,Y]).` die Antworten `X = 1, Y = 2 ;` und `X = 2, Y = 1 ;`.

(c) Schreiben Sie schließlich eine Regel `loeseRaetsel/3`. Der erste Parameter soll den Namen des Ritters zurückgeben, der zweite Parameter den Namen des Knappen und der dritte Parameter den Namen des Spions. Beachten Sie, es sollen nur Kombinationen zurück gegeben werden, bei denen der Ritter die Wahrheit sagt, der Knappe lügt und der Spion lügt oder die Wahrheit sagt.

Hinweis: Sie dürfen `not/1` verwenden, um eine Aussage zu negieren.

Benutzen Sie abschließend ihren Code und geben Sie an, wer welche Rolle hat.

2. Gegeben seien die folgenden Fakten `etwasGroesserAls/2` über die Größenverhältnisse der Kinder einer Schulklasse. Der 2. Parameter gibt dabei das größere Kind an.

```
etwasGroesserAls(anna,boris).
etwasGroesserAls(boris,celine).
etwasGroesserAls(celine,daniel).
etwasGroesserAls(daniel,emma).
```

(a) Die Lehrerin vermutet, dass `emma` das größte Kind der Klasse ist. Schreiben Sie Regel(n) und Frage(n), die ermitteln können, ob dies tatsächlich stimmt. Die Antwort auf die Frage sollten diejenigen Kinder sein, die größer sind als `emma`, oder `false`, falls `emma` tatsächlich das größte Kind der Klasse ist.

(b) Weiterhin vermutet die Lehrerin, dass `boris` das kleinste Kind der Klasse ist. Schreiben Sie Regel(n) und Frage(n), die ermitteln können, ob dies tatsächlich stimmt. Die Antwort auf die Frage sollten diejenigen Kinder sein, die kleiner sind als `boris`, oder `false`, falls `boris` tatsächlich das kleinste Kind der Klasse ist.