Prof. Dr. J. Giesl
S. Dollase, M. Hark, D. Korzeniewski

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter oben links!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Freitag, den 25.01.2019 um 12 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor abgeben.
- In einigen Aufgaben müssen Sie in Prolog programmieren und `.pl`-Dateien anlegen. **Drucken** Sie diese aus **und** schicken Sie sie per **E-Mail** vor Freitag, den 25.01.2019 um 12 Uhr an Ihre Tutorin/Ihren Tutor. Stellen Sie sicher, dass Ihr Programm von `SWI` **akzeptiert** wird, ansonsten werden keine Punkte vergeben.
- Benutzen Sie in Ihrem Code keine Umlaute, auch nicht in Strings und Kommentaren. Diese führen oft zu Problemen, da diese Zeichen auf verschiedenen Betriebssystemen unterschiedlich behandelt werden. Dadurch kann es dazu führen, dass Ihr Programm bei Ihrer Tutorin/Ihrem Tutor bei Verwendung von Umlauten nicht mehr von `SWI` akzeptiert wird.
- Halten Sie sich beim Lösen von Programmieraufgaben an die auf der Website zur Vorlesung verfügbaren Codekonventionen. Verstöße, die zu unleserlichem Code führen, können zu Punktabzug führen.

Tutoraufgabe 1 (Programmieren in Prolog):

In dieser Aufgabe sollen einige Abhängigkeiten im Übungsbetrieb Programmierung in Prolog modelliert und analysiert werden. Die gewählten Personennamen sind frei erfunden und eventuelle Übereinstimmungen mit tatsächlichen Personennamen sind purer Zufall.

Person	Rang
J. Giesl (jgi)	Professor
J. Hensel (jhe)	Assistent
M. Hark (mha)	Assistent
D. Korzeniewski (dko)	Assistent
L. Bernwald (lbe)	Tutor
S. Azari (saz)	Tutor
F. Ail (fai)	Student
N. Erd (ner)	Student
M. Ustermann (mus)	Student

Schreiben Sie keine Prädikate außer den geforderten und nutzen Sie bei Ihrer Implementierung jeweils Prädikate aus den vorangegangenen Aufgabenteilen.

a) Übertragen Sie die Informationen der Tabelle in eine Wissensbasis für Prolog. Geben Sie hierzu Fakten für die Prädikatssymbole `person` und `hatRang` an. Hierbei gilt `person(X)`, falls `X` eine Person ist und `hatRang(X, Y)`, falls `X` den Rang `Y` hat.

b) Stellen Sie eine Anfrage an das im ersten Aufgabenteil erstellte Programm, mit der man herausfinden kann, wer ein Assistent ist.

Hinweise:

- Durch die wiederholte Eingabe von "`;`" nach der ersten Antwort werden alle Antworten ausgegeben.

c) Schreiben Sie ein Prädikat `bossVon`, womit Sie abfragen können, wer innerhalb der Übungsbetriebshierarchie einen Rang direkt über dem eines anderen bekleidet. Die Reihenfolge der Ränge ist Professor > Assistent > Tutor > Student. So ist z.B. `bossVon(jhe,lbe)` wahr, während `bossVon(dko,fai)` und `bossVon(lbe,jhe)` beide falsch sind.

d) Stellen Sie eine Anfrage, mit der Sie alle Personen herausfinden, die in der Übungsbetriebshierarchie direkte Untergebene haben. Dabei sind mehrfache Antworten mit dem gleichen Ergebnis erlaubt.

e) Schreiben Sie schließlich ein Prädikat `vorgesetzt` mit zwei Regeln, mit dem Sie alle Paare von Personen abfragen können, sodass die erste Person in der Übungsbetriebshierarchie der zweiten Person vorgesetzt ist. Eine Person `X` ist einer Person `Y` vorgesetzt, wenn der Rang von `X` "größer" als der Rang von `Y` ist (wobei wieder Professor > Assistent > Tutor > Student gilt). So sind z.B. `vorgesetzt(jgi, fai)` und `vorgesetzt(dko, fai)` beide wahr, während `vorgesetzt(lbe, jhe)` falsch ist.

Aufgabe 2 (Programmieren in Prolog): (1 + 0.5 + 1 + 0.5 + 2 = 5 Punkte)

In dieser Aufgabe soll ein Bewertungssystem in Prolog modelliert und analysiert werden.
Jede Unterkunft kann mit 1 bis 5 Sternen bewertet werden.

Unterkunft	Bewertung
Waldhotel (wh)	4 Sterne
Berghotel (bh)	5 Sterne
Hotel am See (sh)	2 Sterne
Pilgerhotel (ph)	1 Sterne
Jugendherberge (jh)	3 Sterne
Bett bei Freunden (fb)	5 Sterne

Schreiben Sie keine Prädikate außer den geforderten und nutzen Sie bei Ihrer Implementierung jeweils Prädikate aus den vorangegangenen Aufgabenteilen. Benutzen Sie **keine vordefinierten Prädikate**. Achten Sie auf die **korrekte Schreibweise** der Namen aus der Aufgabenstellung.

a) Übertragen Sie die oben gegebenen Informationen in eine Wissensbasis für Prolog. Geben Sie hierzu Fakten und/oder Regeln für die Prädikatssymbole `istUnterkunft`, `hatEinenStern`, `hatZweiSterne`, `hatDreiSterne`, `hatVierSterne` und `hatFuenfSterne` an. Hierbei gilt `istUnterkunft(X)`, falls `X` eine Unterkunft ist, `hatEinenStern(X)`, falls `X` eine mit einem Stern bewertete Unterkunft ist, `hatZweiSterne(X)`, falls `X` eine mit zwei Sternen bewertete Unterkunft ist, usw.

b) Geben Sie eine Anfrage an das im ersten Aufgabenteil erstellte Programm an, mit der man herausfinden kann, welche Unterkünfte mit fünf Sternen bewertet worden sind.

Hinweise:

- Durch die wiederholte Eingabe von ";" nach der ersten Antwort werden alle Antworten ausgegeben.

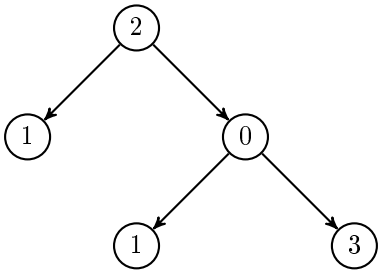
c) Schreiben Sie ein Prädikat `hatEinenSternWeniger`, womit Sie abfragen können, ob eine Unterkunft mit genau einem Stern weniger bewertet wurde als eine zweite Unterkunft. So ist beispielsweise `hatEinenSternWeniger(wh, fb)` wahr, während `hatEinenSternWeniger(sh, bh)` und `hatEinenSternWeniger(jh, sh)` beide falsch sind.

d) Stellen Sie eine Anfrage, mit der Sie alle Unterkünfte herausfinden, die so bewertet wurden, dass es mindestens eine weitere Unterkunft gibt, welche mit genau einem Stern weniger bewertet worden ist. Dabei sind mehrfache Antworten mit dem gleichen Ergebnis erlaubt.

e) Schreiben Sie ein Prädikat `hatWenigerSterne`, mit welchem Sie alle Paare von Unterkünften abfragen können, sodass die erste Unterkunft schlechter bewertet worden ist als die zweite Unterkunft. So ist z.B. `hatWenigerSterne(ph, bh)` wahr, während `hatWenigerSterne(bh, fb)` und `hatWenigerSterne(jh, ph)` beide falsch sind.

Tutoraufgabe 3 (Prolog mit Listen und eigenen Datenstrukturen):

Natürliche Zahlen lassen sich in Prolog (oder anderen deklarativen Sprachen) durch die Peano-Notation als Terme darstellen. Dabei stellt die Konstante `0` die Zahl 0 dar und für eine Zahl n dargestellt durch den Term `N` stellt `s(N)` die Zahl $n+1$ dar. So wird z. B. die Zahl 3 durch den Term `s(s(s(0)))` dargestellt.
Binäre Bäume können in Prolog folgendermaßen als Terme dargestellt werden. Sei n eine natürliche Zahl dargestellt durch den Term `N`. Dann repräsentiert der Term `leaf(N)` einen Baum mit nur einem Blatt, welches den Wert n enthält. Für zwei Bäume x und y dargestellt durch die Terme `X` und `Y` repräsentiert der Term `node(X,N,Y)` einen binären Baum mit einem Wurzelknoten, der den Wert n enthält und die Teilbäume x und y hat. Als Beispiel ist nachfolgend ein binärer Baum und seine Darstellung als Term angegeben.




```
node(leaf(s(0)),s(s(0)),node(leaf(s(0)),0,leaf(s(s(s(0))))))
```

a) Schreiben Sie ein Prädikat `increment/2` in Prolog, wobei `increment(B,IncB)` genau dann wahr sein soll, wenn sich der Baum `IncB` aus dem Baum `B` ergibt, indem jede Zahl, die in einem Knoten oder Blatt steht, um eins erhöht wird. Beispielsweise soll der Aufruf

```
increment(node(leaf(s(0)),s(s(0)),leaf(0)),Res)
```

das Ergebnis `Res = node(leaf(s(s(0))),s(s(s(0))),leaf(s(0)))` liefern.

b) Schreiben Sie ein Prädikat `append(XS,YS,Res)`, das die Listen `XS` und `YS` hintereinanderhängt. Ein Aufruf von `append([a,b,c],[d,e],Res)` würde das Ergebnis `Res = [a,b,c,d,e]` liefern.

c) Es gibt verschiedene Verfahren, um einen Baum systematisch zu durchsuchen. Man unterscheidet zwischen Pre-, In- und Postorder-Traversierung. Bei einer Preorder-Traversierung wird zuerst die Wurzel (W) eines Baums durchsucht, dann der linke Teilbaum (L) und anschließend der rechte Teilbaum (R). Bei Inorder ist die Reihenfolge LWR und bei Postorder LRW. Für den Beispielbaum aus der Abbildung ergeben sich folgende Ausgaben:

- Preorder: 2, 1, 0, 1, 3
- Postorder: 1, 1, 3, 0, 2
- Inorder: 1, 2, 1, 0, 3

Sie sollen nun eine Funktion `inorder(B,Res)` schreiben, die alle Knoten eines Baumes in **In**order-Reihenfolge in die Liste `Res` einträgt. Wenn `B` der Baum aus der Abbildung ist, so würde `inorder(B,Res)` das Ergebnis `Res = [s(0),s(s(0)),s(0),0,s(s(s(0)))]` liefern.

Hinweise:

- Sie dürfen die Funktion `append` aus Teilaufgabe b) verwenden.

Aufgabe 4 (Prolog mit Listen und eigenen Datenstrukturen): (1.5 + 1.5 + 3.5 + 1.5 + 2.5 = 10.5 Punkte)

Verwenden Sie in dieser Aufgabe **keine vordefinierten Prädikate**. Nutzen Sie Prädikate, deren Implementierung in früheren Teilaufgaben gefordert wurde, falls dies sinnvoll ist.

a) Eine Möglichkeit, Listen in Prolog darzustellen, ist die Verwendung eines nullstelligen Funktionssymbols `nil` zur Repräsentation der leeren Liste und eines zweistelligen Funktionssymbols `cons` zur Repräsentation nicht-leerer Listen, wobei das erste Argument von `cons` der in dem aktuellen Listenelement gespeicherte Wert und das zweite Argument von `cons` die Restliste ist. Auf diese Art und Weise kann z.B. die Liste `[1,2,3]` durch den Term `cons(1, cons(2, cons(3, nil)))` dargestellt werden.

Implementieren Sie ein Prädikat `isList(X)` das genau dann wahr ist, wenn `X` eine mit Hilfe der Funktionssymbole `nil` und `cons` beschriebene Liste ist. Beispielsweise sollte `isList(cons(a, cons(b, nil)))` wahr sein, aber `isList(cons(nil, 1))` ist falsch.

b) Implementieren Sie ein Prädikat `toPrologList(X,Y)` das genau dann wahr ist, wenn

- `X` eine mit Hilfe der Funktionssymbole `nil` und `cons` (siehe vorheriger Aufgabenteil) beschriebene Liste ist und
- `Y` die gleiche Liste wie `X` beschreibt, dazu jedoch die vordefinierten Prolog-Listen nutzt.

Es soll also beispielsweise `toPrologList(cons(1, cons(2, cons(3, nil))), [1,2,3])` gelten.

c) Implementieren Sie mit den vordefinierten Listen in Prolog ein Prädikat `flatten(X,Y)` das genau dann wahr ist, wenn

- `X` eine Liste von Listen ist und
- `Y` jene Liste ist, die entsteht, wenn man alle Element von `X` konkateniert.

Es soll also beispielsweise `flatten([[1,2,3],[],[3,4]], [1,2,3,3,4])` gelten.

d) Implementieren Sie ein Prädikat `appendElement(X,Y,Z)`, das genau dann wahr ist, wenn die Liste `Z` entsteht, wenn man an die Liste `X` das Element `Y` hinten anhängt. Verwenden Sie zum Lösen dieser Aufgabe die vordefinierten Prolog-Listen. Es soll also beispielsweise `appendElement([1,2], 3, [1,2,3])` gelten.

e) Implementieren Sie ein Prädikat `reverseList(X,Y)`, das genau dann wahr ist, wenn die Liste `Z` entsteht, wenn man die Liste `X` umkehrt. Verwenden Sie zum Lösen dieser Aufgabe wieder die vordefinierten Prolog-Listen. Es soll also beispielsweise `reverseList([1,2,3,4], [4,3,2,1])` gelten.