DLL // DI
Digitales Lehren & Lernen
Didaktik der Informatik

UNIVERSITÄT BAYREUTH

WISSENSBASIERTE SYSTEME & LOGISCHE PROGRAMMIERUNG

Ein Selbstlernheft für Schülerinnen und Schüler

Der Herr der Brillen

von André Oppitz und Nicole Günzel-Weinkamm

Dieses Selbstlernheft zum Thema „Wissensbasierte Systeme und logische Programmierung“ wurde im Rahmen der Lehrerfortbildungen „Künstliche Intelligenz – Jahrgangsstufe 13 – Wissensbasierte Systeme“ am Fachgebiet Digitales Lehren und Lernen & Didaktik der Informatik der Universität Bayreuth entwickelt.

Weiteres Material zu den Arbeitsaufträgen befindet sich im Mebis-Kurs „Herr der Brillen - Wissensbasierte Systeme und Einführung in PROLOG“ (ID-Nummer 1880746). Den Einschreibeschlüssel hierfür können Sie als Lehrkraft bei den unten genannten Kontaktadressen erfragen.

Das Heft und weiteres Material sind abrufbar unter https://dlldi.uni-bayreuth.de/ki13

Kontakt bei Fragen

André Oppitz
aoppitz@schule.gym-new.de

Nicole Günzel-Weinkamm
Nicole.Guenzel-Weinkamm@uni-bayreuth.de

Fachgebiet Digitales Lehren und Lernen & Didaktik der Informatik
Universität Bayreuth
95447 Bayreuth

Danksagung

Ein herzlicher Dank ergeht an Dr. Matthias Ehmann (Universität Bayreuth) und Dr. Silvia Joachim (Universität Würzburg) für inhaltlichen Austausch und Korrekturlesen.

Urheberrecht

Das Bild auf der Titelseite und im Arbeitsheft mit LEOAI gekennzeichnete Bilder wurden mit der KI-basierten Plattform „Leonardo AI“ (https://leonardo.ai) erstellt. Die dort aufgeführten Lizenzbestimmungen sind zu beachten.

Das Heft darf gemäß CC-BY-NC-SA im Unterricht verwendet und weitergegeben werden.

Inhaltsverzeichnis

GETTING STARTED: Wissenswertes zum Umgang mit dem Arbeitsheft 1
PROLOG: PROLOG & Prolog 3
I. Erstellen und Nutzen einer ersten Wissensbasis 4
1) Fakten 4
2) Variablen und erste Anfragen 6
3) Regeln 8
II. Vertiefung und weiterführende Konzepte 11
4) Rekursion 11
5) Abarbeitung von Anfragen durch die Inferenzmaschine – Der trace-Befehl 15
6) Logikrätsel – Teil 1: Karten färben 20
7) Logikrätsel – Teil 2: Einsteinrätsel bzw. Logicals 22
EXKURS: Aussagenlogik 27
EXKURS: Arithmetik 33

GETTING STARTED: Wissenswertes zum Umgang mit dem Arbeitsheft

Dieses Arbeitsheft soll ein eigenständiges Einarbeiten in das Themenfeld „Wissensbasierte Systeme“ ermöglichen und gleichzeitig eine Einführung in die logische Programmiersprache PROLOG geben. Hierfür ist es in aufeinander aufbauende Kapitel unterteilt. Es empfiehlt sich daher, die einzelnen Kapitel nacheinander zu durchlaufen.

Jedes Kapitel beginnt mit explorativen Aufgaben, in denen sich der Benutzer die grundlegenden Inhalte selbst erarbeiten kann.

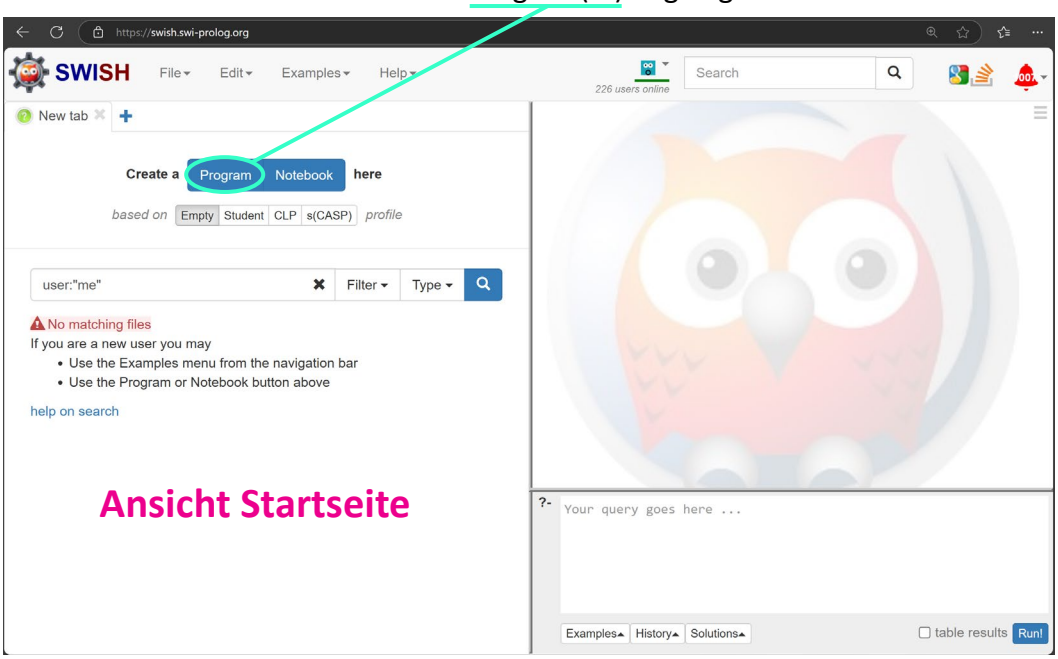
Im Anschluss daran werden in einer Zusammenfassung (Merke) wichtige Begriffe und Sachverhalte nochmals aufgegriffen und erläutert. Da die Inhalte der Lückentexte nicht immer selbsterklärend sind, können sie vorab interaktiv (H5P-Inhalt: „Drag the Words“) ausgefüllt werden. Diese sind über einen (in der pdf-Version auch direkt anklickbaren) QR-Code an den entsprechenden Stellen im Geheft verlinkt. Dadurch sollen sukzessive die wichtigsten Aspekte jedes Kapitels verstanden und anschließend in den vorhandenen Notizbereichen gesichert werden.

Abschließend stehen in jedem Kapitel weitere Übungsaufgaben zur Anwendung und Vertiefung zur Verfügung. Zeitaufwändigere bzw. anspruchsvollere Aufgaben werden in drei ansteigenden Stufen mit 🤔, 🤓 bzw. 🤯 gekennzeichnet.

Aus Lehrplansicht genügt es, von den beiden Kapiteln 6) und 7) zu Logikrätseln nur eines zu bearbeiten. Weitere (nicht-lehrplanrelevante) Inhalte, die für den Umgang mit PROLOG hilfreich sind, bzw. für den interessierten Leser gewinnbringend sein könnten, sind in Exkurse ausgelagert.

Bevor es mit den ersten Schritten in PROLOG losgehen kann, soll noch der von uns bevorzugte Online-PROLOG-Interpreter SWISH (https://swish.swi-prolog.org/) kurz vorgestellt werden:

Nach dem Aufruf der Seite muss ein neues Program(m) angelegt werden.



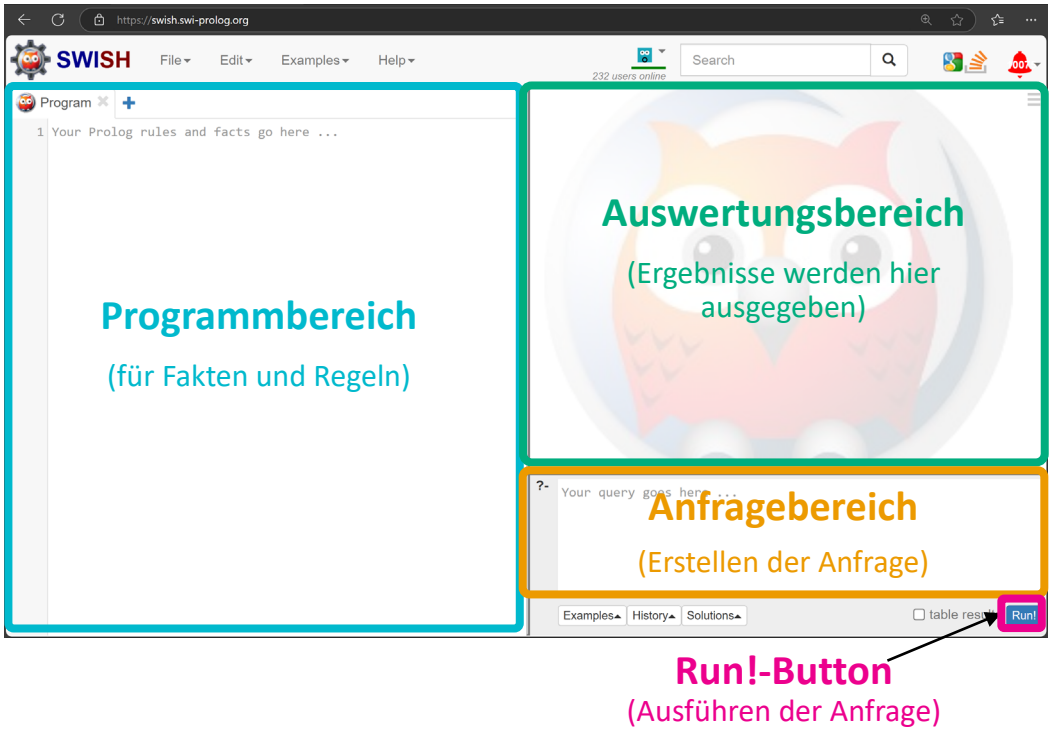
(cc) BY - NC - SA

Anschließend ist das Programmfenster in drei Bereiche untergliedert:

- Im **Programmbereich** wird die Wissensbasis angelegt.
  Hier werden Fakten [Kap. 1)], Regeln [Kap. 3)] und rekursive Regeln [Kap. 4)] erstellt.
  Wie in anderen Programmiersprachen können auch Kommentare eingefügt werden.
  Diese werden mit dem *%-Zeichen eröffnet und enden am Zeilenende.*
  / Auch mehrzeilige Kommentare*
  *sind möglich. */*

- Im **Anfragebereich** werden die Anfragen [Kap. 2)] an die Wissensbasis erzeugt.
  Ausgeführt werden diese durch Abschließen der Anfrage mit ‚Strg + Enter' oder durch betätigen des **Run!-Buttons**.

- Im **Auswertungsbereich** werden die Ergebnisse von Anfragen an die Wissensbasis ausgegeben. Dabei wird immer nur ein Ergebnis zurückgegeben.
  Will man weitere bzw. mehrere Ergebnisse angezeigt bekommen, so können die im Auswertungsbereich erscheinenden Schaltflächen Next | 10 | 100 | 1,000 genutzt werden.
  Die interne Vorgehensweise von PROLOG bei der automatisierten Abarbeitung von Anfragen durch die Inferenzmaschine wird im zugehörigen Kapitel [Kap. 5)] näher betrachtet.



(cc) BY - NC - SA

PROLOG: PROLOG & Prolog

PROLOG: PROLOG steht für "Programming in Logic" (Programmierung in Logik) und ist eine deklarative Programmiersprache. Sie wurde in den 1970er-Jahren entwickelt und basiert auf logischen Regeln und Schlussfolgerungen. Im Gegensatz zu imperativen Programmiersprachen wie Java oder Python, bei denen Sie dem Computer Schritt für Schritt sagen, was er tun soll, beschreiben Sie in PROLOG das vorhandene Wissen und lassen den Computer die logischen Schlussfolgerungen ziehen, um ein gewünschtes Ergebnis zu erreichen. Das bedeutet, dass die Beschreibung des Problems im Vordergrund steht und die Lösung des Problems dann automatisch aus dieser ermittelt wird.

Prolog: Als Kontext für dieses Arbeitsheft wurde der „Herr der Brillen“ gewählt.

Der Herr der Brillen spielt in einer fiktiven Welt namens Refrakturien. Dort lebt unter anderem das Volk der Malin (vom russischen Wort Маленький für klein oder winzig). Die Geschichte beginnt im Augenland und folgt der Malindame Frieda Auglin, die eine mächtige magische Brille (*Die Brille*) erhält. Diese möchte der dunkle Herrscher Sehron unbedingt erlangen, um sie in die stärkste Sonnenbrille ganz Refrakturiens zu verwandeln und somit endgültig zum dunkelsten Herrscher aller Zeiten aufzusteigen.

Frieda schließt sich eine Gemeinschaft aus Begleitern an, darunter weitere Malins, Menschen, Zwergen und Elben, um *Die Brille* zu zerstören. Sie begeben sich auf eine gefährliche Reise, durchqueren verschiedene Länder und kämpfen gegen Feinde, wie z.B. Orks, Trolle und weitere Verbündete Sehrons.

Währenddessen wird Refrakturien in einen Krieg zwischen Gut und Böse verwickelt. Frieda und ihre Begleiter müssen sich auf ihrem Weg zur Prismengrotte, wo *Die Brille* zerstört und Sehrons Macht ein für alle Mal gebrochen werden kann, zahlreichen Herausforderungen stellen...

(cc) BY - NC - SA

I. Erstellen und Nutzen einer ersten Wissensbasis

1) Fakten

Augenland. Wir schreiben das Jahr 2255 des Zeitalters der Linsenmacher. Frieda Auglin hat von Brillo *Die Brille* geerbt, welche nun in Gefahr ist. Luminalf schickt Frieda auf die Reise nach Brechzal zu den Elben.

A1.1 Erstellen Sie im Programmbereich von Swish eine Wissensbasis „brillen.pl“. Legen Sie darin die beiden folgenden Einträge an: *zauberer(luminalf).*
malin(frieda).

Beschreiben Sie die beiden Einträge.

A1.2 Zu den ersten Begleitern Friedas gehören die drei Malins Glasweis, Polargrin und Monodoc. Legen Sie die drei Malins als Fakten in der Wissensbasis an.

A1.3 Legen Sie Fakten für die Spitznamen (Glasweis → Glas, Polargrin → Pollin, Monodoc → Mono) in der Form *spitzname(Name, Spitzname)* an. Dabei steht *Name* für den Namen und *Spitzname* für den Spitznamen des jeweiligen Charakters.

Merke:

Ein _____ besteht aus dem sogenannten _____ (dem Namen des Faktums, z.B. *zauberer* bzw. *malin*) und _____. **Fakten** haben _____ _____ und müssen _____ abgeschlossen werden. Besitzen Fakten mehrere Argumente, werden diese _____ voneinander getrennt.

Alle Bezeichner müssen mit _____ beginnen oder in einfache Hochkommata eingeschlossen werden, da sie sonst von PROLOG als _____ werden.

Unter einem **Prädikat** versteht man die _____ aller _____ mit gleichem _____ und gleicher _____ (z.B. *malin/1* oder *spitzname/2*).

Alle Fakten zu einem Prädikat müssen _____ gelistet werden.

(cc) BY - NC - SA

Übungsaufgaben:

Die nächsten Personen, auf die Frieda trifft, sind die Menschen Absorgorn und Optomir, der Zwerg Grummli und die Elben Linsolas und Lensrond. Die Malins Glasweis, Frieda, Monodoc und Polargrin sind untereinander befreundet. Ebenso sind Brillo und Frieda befreundet. Luminalf pflegt Freundschaften zu Brillo, Frieda, Absorgorn, Grummli, Linsolas und Lensrond.

Ü1.1 Erstellen Sie die Prädikate *mensch/1*, *zwerg/1*, *elb/1* mit den jeweils zugehörigen Fakten.

Ü1.2 Erzeugen Sie das Prädikat *freund/2*, das die Freundschaft zwischen zwei Personen modelliert und legen Sie alle genannten Fakten hierfür an. Dabei soll jede Freundschaft nur einmalig angelegt werden. Also zu *freund(f1,f2).* soll nicht zusätzlich der Fakt *freund(f2,f1).* erzeugt werden.

Lösungen:

(cc) BY - NC - SA

2) Variablen und erste Anfragen

Untersuchen Sie die Ergebnisse der folgenden Anfragen an die Wissensbasis durch Eingabe im Anfragebereich von SWISH. Beachten Sie, dass SWISH automatisch ?- voranstellt.

a) *[?-] zauberer(X).*
b) *[?-] malin(X).*
c) *[?-] zauberer(X); malin(X).*
d) *[?-] zauberer(X), malin(X).*
e) *[?-] malin(Glas), spitzname(Glas,_).*
f) *[?-] malin(X), spitzname(X,Y).*
g) *[?-] malin(X), spitzname(X,pollin).*
h) *[?-] malin(_), spitzname(_,pollin).*
i) *[?-] malin(_X), spitzname(_X,pollin).*
j) *[?-] malin(X), not(spitzname(X,pollin)).*
k) *[?-] malin(X), not(spitzname(X,_)).*
l) *[?-] malin(X), spitzname(X,_Y), _Y\=glas.*

Merke:

Anfragen an die Wissensbasis werden stets mit _____ eingeleitet [bei swish.swi-prolog automatisch eingefügt].

Startet ein Argument mit einem _____, wird das Argument als **Variable** betrachtet. Für Variablen werden stets alle möglichen Belegungen ausgegeben. Will man die Ausgabe einer Variablen unterdrücken, so ist _____ voranzustellen.

Taucht eine identische Variable (mit oder ohne Unterstrich) in zwei verschiedenen Funktoren auf, so hat sie jeweils _____ (vergleichbar mit einem JOIN in SQL). Für jede Belegung einer Variable mit Unterstrich, die eine Anfrage erfüllt, wird der Wert *true* ausgegeben.

Wird **statt einer Variable nur ein Unterstrich** verwendet, so werden hierfür _____ _____ Belegungen verwendet. Eine Variablenbindung (also die Gleichheit der Variablenbelegung) über mehrere Funktoren ist hiermit _____.

Mehrere Funktoren können mit einer _____ – durch _____ getrennt, oder mit einer _____ – durch _____, kombiniert werden.

Eine Bedingung *B* kann **verneint** werden durch _____.

Eine Variablenbelegung kann mit ____ verboten werden.

(cc) BY - NC - SA

Übungsaufgaben:

Ergänzen Sie die Tatsache, dass es sich bei Brillo um einen Malin handelt.

TIPP: Die Auswertungsreihenfolge kann bzw. muss ggf. durch Klammern beeinflusst werden.

Ü2.1 Geben Sie die Namen aller Freunde von Frieda aus. Beachten Sie dabei, dass das Prädikat *freund/2* keine bevorzugte Reihenfolge der Argumente aufweist.

Ü2.2 Geben Sie die Namen aller Freunde von Luminalf aus, die keine Malins sind.

Ü2.3 Geben Sie die Namen aller Malins aus, die nicht mit Luminalf befreundet sind.

Ü2.4 Prüfen Sie, ob Brillo einen Freund hat.

Ü2.5 Geben Sie die Namen aller Malins aus, die einen Nicht-Malin als Freund haben.

Ü2.6 Geben Sie die Namen aller Charaktere außer Frieda aus, die (bislang) nicht mit Frieda befreundet sind, aber mindestens einen Freund haben.

Lösungen:

(cc) BY - NC - SA

3) Regeln

Im letzten Kapitel wurden Fakten definiert, um eine erste Wissensbasis zu erstellen. Um hieraus Schlussfolgerungen ziehen und um komplexere logische Zusammenhänge modellieren zu können, werden Regeln benötigt.

A3.1 Die bisherige Festlegung des Prädikats *freund/2* macht die Nutzung unhandlich. Sucht man Freunde einer Person, muss man prüfen, ob die Person an erster oder zweiter Stelle steht.

Mit der Festlegung *befreundet(F1, F2) :- (freund(F1, F2); freund(F2, F1)).* wird ein neues Prädikat definiert.

Nun lassen sich alle Freunde einer Person x (z.B. x = Frieda) über die Anfrage

[?-] befreundet(x,Y). oder *[?-] befreundet(Y,x).*

ausgeben. Testen Sie diese erste Regel, indem Sie sich

a) alle Freunde von Luminalf

b) alle Freunde von Frieda

c) alle gemeinsamen Freunde von Luminalf und Frieda

d) alle Freunde von Luminalf und alle Freunde von Frieda – <u>ohne Doppelnennungen</u> ausgeben lassen.

A3.2 Ändern Sie die befreundet-Regel in folgende zwei Zeilen ab und prüfen Sie die Ergebnisse der Anfragen aus Aufgabe 1.

befreundet(F1, F2) :- freund(F1, F2).

befreundet(F1, F2) :- freund(F2, F1).

<u>Merke:</u>

Jede **<u>Regel</u>** besteht aus _____ und _____. Sie werden getrennt durch den **<u>Regeloperator</u>** „:-“, der als _____ zu lesen ist. Der Regelrumpf enthält die Konjunktion (logische _____) aller Bedingungen an die Schlussfolgerung.

Das logische ODER bei mehreren Alternativen wird üblicherweise (wie in Aufgabe A3.2) **<u>auf mehrere Klauseln</u>** (= Fakten oder Regeln) _____ und eher nicht _____ realisiert.

Zu beachten ist, dass die Argumente von Fakten typischerweise _____ sind, während in Regeln _____ genutzt werden. Deren **<u>Gültigkeitsbereich</u>** erstreckt sich dabei _____, in der sie auftreten.

(cc) BY - NC - SA

Übungsaufgaben:

Ü3.1 Erstellen Sie eine neue Regel *befreundet2/2*, die die Namen der Freunde auch dann ausgibt, wenn sie <u>im ersten Argument</u> über den Spitznamen gesucht werden. Überprüfen Sie Ihre Umsetzung mit der Anfrage *[?-] befreundet2(glas,X)*.

Ü3.2 Betrachten Sie die Resultate der in Ü3.1 erstellten Regel *befreundet2/2* für die Anfrage *[?-] befreundet2(X,glasweis).* Wie erklären Sie sich diese Resultate?

Ü3.3 Überlegen Sie, wie sich die (womöglich unerwünschte) Ausgabe von Spitznamen in der Situation von Ü3.2 verhindern lässt, wenn die Bedingung aus Ü3.1 erfüllt bleiben soll.

Ü3.4 Erstellen Sie eine weitere Regel *befreundet3/2*, die die Namen der „Freundesfreunde" eines Charakters ausgibt. Beispielsweise soll für *befreundet3(polargrin,X)* auch *luminalf* ausgegeben werden, da dieser ein Freund von Frieda ist, die wiederum ein Freund von Polargrin ist. Der eigene Name soll dabei nicht als Freundesfreund ausgegeben werden. Überprüfen Sie Ihre Umsetzung mit der Anfrage *[?-] befreundet3(X,glasweis)*.

Ü3.5 Erstellen Sie eine Spezialisierung *befreundet4/2* der Regel aus Ü3.4, die nur die Freundesfreunde ausgibt, die keine direkten Freunde sind. Beispielsweise soll für *befreundet4(polargrin,X)* nicht *Frieda* ausgegeben werden. Diese wäre zwar als Freund von Glasweis ein Freundesfreund, ist aber gleichzeitig direkt mit Polargrin befreundet. Auch hier soll der eigene Name nicht als Freundesfreund genannt werden.

Ü3.6 Erstellen Sie eine neue Regel *charakter/1* als Oberbegriff aller bisher angelegten Personen. Jede Person *person*, die einem Volk zugewiesen ist, soll nun auch als Charakter aufgefasst werden können. Das heißt, *charakter(person)* soll *true* liefern.

Ü3.7 Erstellen Sie eine Regel *keineFreunde/2*, die die Namen aller Charaktere ausgibt, die bislang keine Freunde haben.

Lösungen:

(cc) BY - NC - SA

(cc) BY - NC - SA

II. Vertiefung und weiterführende Konzepte

4) Rekursion

Rekursion ist ein Programmierkonzept, bei dem eine Funktion sich selbst aufruft, um ein Problem in kleinere Teilprobleme zu zerlegen und dadurch ggf. einfacher zu lösen ist. Um diese in unserem Kontext umzusetzen, werden weitere Personen und Zusammenhänge benötigt:

Der Zauberer Schattuman ist zu Beginn ein Mentor des Zauberers Luminalf, welcher wiederum als Mentor des Malins Brillo gesehen werden kann. Schattuman seinerseits unterwirft sich dem dunklen Herrscher Sehron, sodass dieser mehr oder weniger seinen Mentor darstellt. Luminalf und Brillo nehmen die Malindame Frieda in verschiedenen Lebensabschnitten unter ihre Fittiche.

A4.1 Erfassen Sie die zugehörigen Fakten in einem Prädikat *istMentorVon(Mentor,Schueler)*.

A4.2 Erläutern Sie, warum folgende Regel *mentorenVon/2* geeignet ist, alle „Meta-Mentoren“ einer Person X auszugeben:

mentorenVon(Mentor,Schueler) :- istMentorVon(Mentor,Schueler).
mentorenVon(Mentor,Schueler) :- istMentorVon(X,Schueler), <u>mentorenVon</u>(Mentor,X).

A4.3 Testen Sie die obige Regel *mentorenVon/2* für die Personen Luminalf und Frieda.

(cc) BY - NC - SA

A4.4 Überprüfen Sie, ob die folgenden rekursiven Regeln ebenfalls geeignet sind, alle „Meta-Mentoren“ einer Person X auszugeben. Notieren Sie ggf. eine Vermutung, worin das Problem der jeweiligen Regel liegt.

mentorenVon1(Mentor,Schueler) :- istMentorVon(X,Schueler), mentorenVon1(Mentor,X).
mentorenVon1(Mentor,Schueler) :- istMentorVon(Mentor,Schueler).

mentorenVon2(Mentor,Schueler) :- istMentorVon(Mentor,Schueler).
mentorenVon2(Mentor,Schueler) :- mentorenVon2(Mentor,X), istMentorVon(X,Schueler).

mentorenVon3(Mentor,Schueler) :- istMentorVon(Mentor,Schueler).
mentorenVon3(Mentor,Schueler) :- istMentorVon(Mentor,X), mentorenVon3(X,Schueler).

A4.5 Erstellen Sie eine rekursive Regel *schuelerVon/2*, die die Schüler (aller Folgegenerationen) einer Person P ausgibt. Überprüfen Sie damit die Schüler von Sehron und Brillo.

Merke:

Wie bei jeder **Rekursion** muss auch in PROLOG sichergestellt werden, dass eine _____ Regel _____. Es bietet sich daher an, als erstes die _____festzulegen.

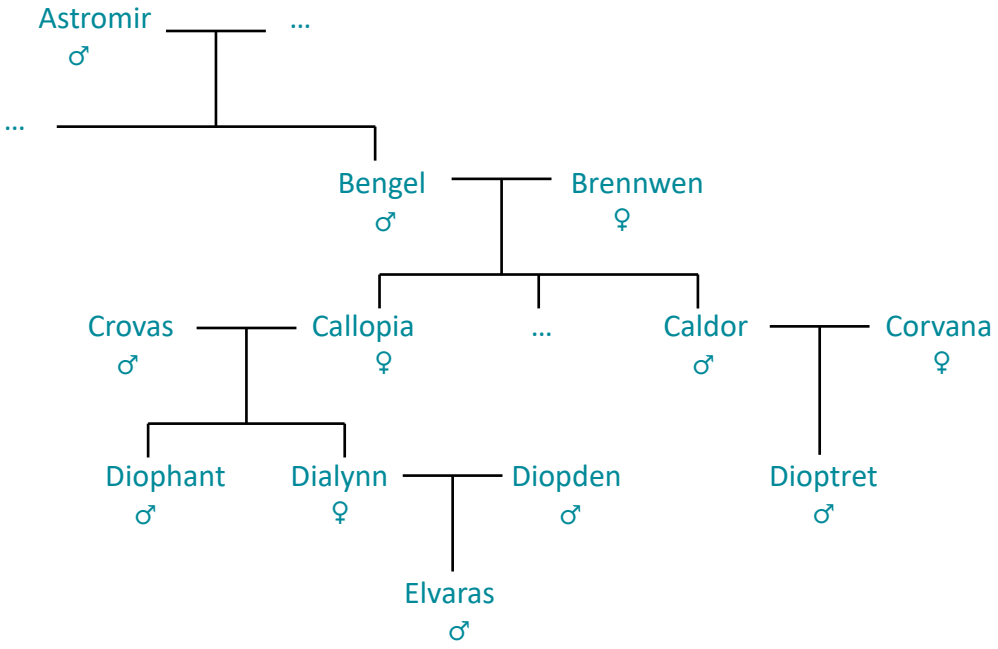
Die rekursive Klausel zerlegt das Problem in _____ und löst diese rekursiv, bzw. nähert sich der **terminierenden Klausel** an.

Innerhalb der rekursiven Klausel muss die _____ Regel vor dem _____ notiert werden.
Andernfalls entsteht eine _____.

(cc) BY - NC - SA

Übungsaufgaben:

Ein klassisches Beispiel für die Nutzung der Rekursion ist ein Stammbaum. In unserem Kontext kann der (nicht in Gänze verfügbare) Stammbaum rund um König Caldor genutzt werden:



Ü4.1 Setzen Sie die Fakten dieses Stammbaums mittels zweier Prädikate *vater(V,Kind)* und *mutter(M,Kind)* in PROLOG um.

Ü4.2 Erstellen Sie eine Regel *elternteil/2*, die die Verwandtschaftsbeziehungen korrekt darstellt. Testen Sie die Ergebnisse anhand geeigneter Anfragen, z.B. *elternteil(Eltern,dialynn)*.

Ü4.3 Erstellen Sie unter Nutzung von Rekursion eine Regel *vorfahren/2*, die alle Vorfahren einer Person ausgibt, und prüfen Sie diese durch geeignete Anfragen.

Ü4.4 Erstellen Sie unter Nutzung von Rekursion analog eine Regel *nachkommen/2*, die alle Nachkommen einer Person ausgibt, und prüfen Sie diese durch geeignete Anfragen.

Lösungen:

(cc) BY - NC - SA

(cc) BY - NC - SA

5) Abarbeitung von Anfragen durch die Inferenzmaschine – Der trace-Befehl

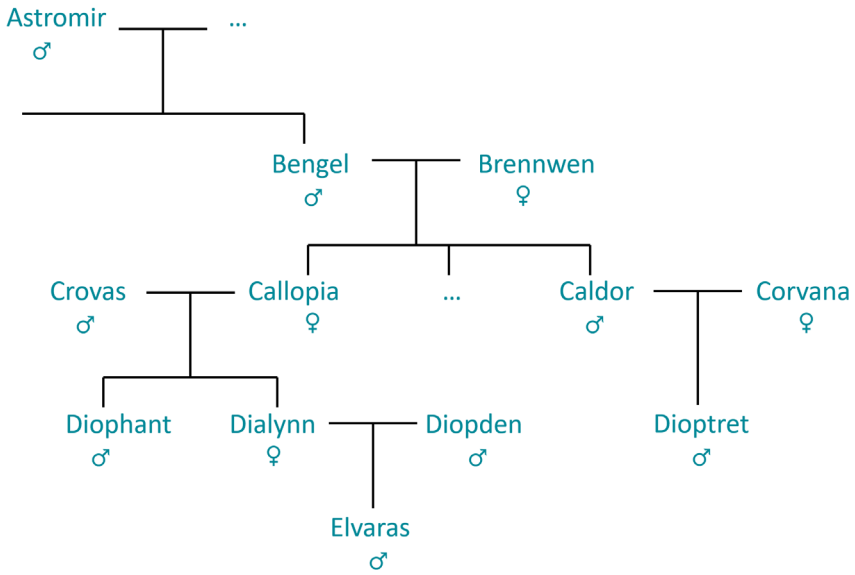
Bei der Ausführung einer Anfrage durchläuft der Prolog-Interpreter sukzessive die vorliegende Wissensbasis. Er dient dabei als sogenannte Inferenzmaschine, die anhand der Wissensbasis neues Wissen ableiten kann. Um den Ablauf der Abarbeitung einer Anfrage Schritt für Schritt sichtbar zu machen und dabei ggf. mögliche Fehler im Programm zu entdecken, kann der trace-Befehl verwendet werden.

Wir betrachten dazu das Stammbaumbeispiel aus den Übungsaufgaben zu Kapitel 4).

Um die Vergleichbarkeit der Abarbeitungsreihenfolge sicherzustellen, wird nun folgende Wissensbasis für den Stammbaum zugrunde gelegt:

sohn(elvaras,diopden).
sohn(elvaras,dialynn).
sohn(dioptret,caldor).
sohn(dioptret,corvana).
sohn(diophant,crovas).
sohn(diophant,callopia).
sohn(caldor,bengel).
sohn(caldor,brennwen).
sohn(bengel,astromir).

tochter(dialynn,crovas).
tochter(dialynn,callopia).
tochter(callopia,bengel).
tochter(callopia,brennwen).



enkeltochter(X,Y) :- tochter(X,Z), tochter(Z,Y).
enkeltochter(X,Y) :- tochter(X,Z), sohn(Z,Y).

Im Folgenden werden wir eine Anfrage an die obige Wissensbasis mit Hilfe des trace-Befehls Schritt für Schritt ausführen und deren Ausgabe erläutern.

A5.1 Ergänzen Sie in SWISH die obige Wissensbasis* und führen Sie anschließend die folgende Anfrage aus: *[?-] trace, enkeltochter(X,bengel).*

Im Auswertungsbereich von SWISH wird nach dem Ausführen der Anfrage Folgendes angezeigt:

Mit Klick auf den Button für Step-Into kann nun der nächste einzelne Abarbeitungsschritt durch die Inferenzmaschine (auch Schlussfolgerungsmaschine) ausgeführt und sichtbar gemacht werden.

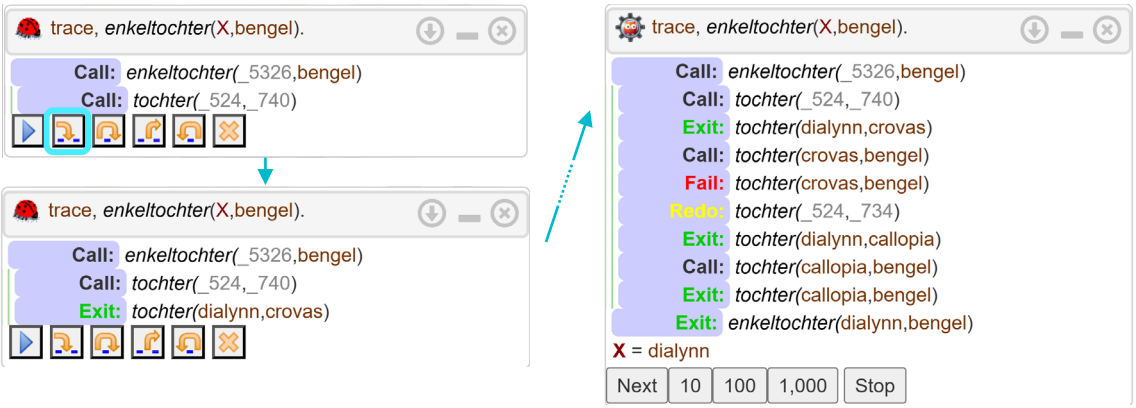
---

* siehe Datei *HdB-AH-Prolog_Kap05_Stammbaum.txt* unter https://dlldi.uni-bayreuth.de/ki13

(cc) BY - NC - SA

A5.2 Klicken Sie nacheinander mehrmals auf den Step-into-Button bis eine erste Lösung für die Variable *X* ausgegeben wird. Achten Sie bei den einzelnen Schritten auch auf Markierungen bzw. Hervorhebungen im Programmbereich.

Nachfolgend finden sich die ersten Ausgaben im Auswertungsbereich von SWISH bei der schrittweisen Abarbeitung der Anfrage:



Hinweis: Die Variablencontainer, z.B. *_5326* oder *_524*, sind nicht für jede Ausführung der Anfrage gleich. Diese Werte variieren typischerweise, z.B. je nach Aufrufzeitpunkt.

Wir betrachten den Ablauf der obigen Ausgabe nochmals im Detail:

```
 1        Call: enkeltochter(_5326,bengel)
 2          Call: tochter(_524,_740)
 3          Exit: tochter(dialynn,crovas)
 4          Call: tochter(crovas,bengel)
 5          Fail: tochter(crovas,bengel)
 6          Redo: tochter(_524,_734)
 7          Exit: tochter(dialynn,callopia)
 8          Call: tochter(callopia,bengel)
 9          Exit: tochter(callopia,bengel)
10        Exit: enkeltochter(dialynn,bengel)
11 X = dialynn
```

Die in der Abbildung violett hinterlegten Begriffe sind wie folgt zu verstehen*:

CALL: Ein Teilziel wird vom System zur Bearbeitung aufgerufen.

EXIT: Das angezeigte Teilziel wurde erfolgreich bewiesen.

REDO: Backtracking. Das Teilziel soll erneut bewiesen werden.

FAIL: Das Teilziel konnte nicht bewiesen werden.

---

* vergleiche: Informatik mit Prolog – Sekundarstufe II, Autor: Gerhard Röhner, Kapitel 3.1, S.16

(cc) BY - NC - SA

Prolog (bzw. die Inferenzmaschine) sucht in der Wissensbasis schrittweise passende Regeln bzw. Fakten um die Anfrage *?- enkeltochter(X,bengel).* zu beweisen, d.h. gemäß der Wissensbasis ableiten zu können.*

```
1   Call: enkeltochter(_5326,bengel)
2   Call: tochter(_524,_740)
3   Exit: tochter(dialynn,crovas)
4   Call: tochter(crovas,bengel)
5   Fail: tochter(crovas,bengel)
6   Redo: tochter(_524,_734)
7   Exit: tochter(dialynn,callopia)
8   Call: tochter(callopia,bengel)
9   Exit: tochter(callopia,bengel)
10  Exit: enkeltochter(dialynn,bengel)
11 X = dialynn
```

Z.1 Mit *Call* wird die in der Wissensbasis gefundene Regel *enkeltochter* aufgerufen. Der in grau angegebene Wert *_5326* steht für eine Variable, die noch nicht mit einem bestimmten Wert belegt wurde bzw. an einen bestimmten Wert gebunden ist.

Z.2 Die Regel *enkeltochter(X,Y)* wird gemäß des angegebenen Regelrumpfs abgearbeitet. Dort steht an erster Stelle die Bedingung *tochter(X,Z)*, sodass als nächstes das Prädikat *tochter/2* in der Wissensbasis mit den Variablen *_524* und *_740* geprüft wird (*Call*).

Z.3 Die Wissensbasis liefert als ersten Fakt *tochter(dialynn,crovas)*.
Dadurch kann die Variable *_524* mit dem Wert *dialynn* unifiziert, d.h. „gleich gemacht“, und auch die zweite Variable *_740* mit dem konkreten Wert *crovas* belegt werden (*Exit*).

Z.4 Nun wird die zweite Bedingung der Regel *enkeltochter(X,Y)* betrachtet: *tochter(Z,Y)*. Hierbei wird direkt nach dem nächsten Fakt mit passender Belegung für die Variable *Z*, also *crovas* (siehe Zeile 3), und dem im Aufruf angegebenen Wert für *Y*, hier *bengel*, in der Wissensbasis gesucht (*Call*).

Z.5 Nachdem der vorherige Aufruf nicht durch die Wissensbasis ableitbar ist, d.h. es gibt keinen passenden Fakt, meldet die Inferenzmaschine dies mit *Fail* zurück.

Z.6 Der erste Ableitungsversuch schlug fehl (siehe *Fail* in Z.5), aber es gab in Z.2 noch weitere Möglichkeiten in der Wissensbasis für die Belegung der Variablen *X* und *Z* der Regel *tochter(X,Z)*. Der Interpreter springt also im sog. Beweisbaum an die letzte mögliche Verzweigung zurück (*Redo*) und versucht die nächste vorhandene Alternative zu beweisen (sog. *Backtracking*). Dabei können sich Variablenbezeichnungen ändern.

Z.7 In der Wissensbasis findet sich als nächstes der Fakt *tochter(dialynn,callopia)* (*Exit*).

Z.8 Der zweite Teil der Regel Enkeltochter kann nun mit den Werten *callopia* und *bengel* geprüft werden (*Call*).

Z.9 Die Inferenzmaschine bestätigt durch *Exit*, dass ein passender Eintrag für *tochter(callopia,bengel)* in der Wissensbasis vorliegt.

Z.10 Damit wurde der ursprüngliche Aufruf *enkeltochter(X,bengel)* nun mit dem Wert *callopia* für *X* belegt und gemäß der Wissensbasis bewiesen (*Exit*).

Z.11 Abschließend wird *dialynn* als gefundene Belegung für *X* ausgegeben.

Nachdem es in der gegebenen Wissensbasis grundsätzlich noch weitere Lösungen geben kann (d.h. der Abarbeitungsbaum hat noch offene Verzweigungen als Rücksprungpunkte), wird nach der Ausgabe für *X* der Next-Button angezeigt. Damit kann anschließend nach möglichen weiteren Lösungen gesucht werden.

---

* Das Programm/die Wissensbasis wird dabei zeilenweise von oben nach unten und innerhalb einer Zeile von links nach rechts durchlaufen.

(cc) BY - NC - SA

A5.3 Betätigen Sie nach Ausgabe der Lösung *X = dialynn* den Next-Button.
Nutzen Sie den Step-into-Button und den Next-Button, um alle weiteren Lösungen der Anfrage *?- enkeltochter(X,bengel).* zu ermitteln. Vergleichen Sie ihre Ausgabe mit der folgenden und machen Sie sich wiederkehrende Abläufe und Änderungen bewusst.

```
X = dialynn
    Redo: tochter(_524,_732)
    Exit: tochter(callopia,bengel)
    Call: tochter(bengel,bengel)
    Fail: tochter(bengel,bengel)
    Redo: tochter(_518,_726)
    Exit: tochter(callopia,brennwen)
    Call: tochter(brennwen,bengel)
    Fail: tochter(brennwen,bengel)
    Redo: enkeltochter(_518,bengel)
    Call: tochter(_518,_758)
    Exit: tochter(dialynn,crovas)
    Call: sohn(crovas,bengel)
    Fail: sohn(crovas,bengel)
    Redo: tochter(_518,_746)
    Exit: tochter(dialynn,callopia)
    Call: sohn(callopia,bengel)
    Fail: sohn(callopia,bengel)
    Redo: tochter(_518,_746)
    Exit: tochter(callopia,bengel)
    Call: sohn(bengel,bengel)
    Fail: sohn(bengel,bengel)
    Redo: tochter(_518,_746)
    Exit: tochter(callopia,brennwen)
    Call: sohn(brennwen,bengel)
    Fail: sohn(brennwen,bengel)
    Fail: enkeltochter(_518,bengel)
false
```

Merke:

Die beiden zentralen Bestandteile eines **wissensbasierten Systems** sind die _____ und die _____. Die Inferenzmaschine leitet durch die Verarbeitung der Fakten und Regeln aus der gegebenen Wissensbasis _____ ab.

Bei der **automatisierten Ableitung einer Aussage** wird die Wissensbasis stets zeilenweise _____ und innerhalb einer Zeile _____ durchlaufen.

PROLOG nutzt dabei, wie auch andere wissensbasierte Systeme, die _____. Diese sucht (ähnlich wie bei der Tiefensuche) systematisch nach Lösungen für ein Problem. Die Lösung wird Schritt für Schritt aufgebaut. Falls festgestellt wird, dass eine Teillösung nicht zu einer gültigen _____ führen kann, wird der Algorithmus auf einen vorherigen Zustand _____ (=Backtracking), um _____ zu untersuchen.

Mit Hilfe des **trace-Befehls** kann die Reihenfolge der Abarbeitung einer Anfrage *?- anfrage(XYZ).* schrittweise untersucht werden. **Syntax:** _____

Das Zustandekommen der Gesamtlösung wird durch den trace-Befehl anhand der **vier grundlegenden Phasen** dargestellt:

- _____: *Aufruf eines Teilziels zur Bearbeitung*
- _____: *erfolgreicher Beweis des Teilziels*
- _____: *Teilziel soll erneut bewiesen werden (Backtracking)*
- _____: *Beweis des Teilziels nicht möglich*

(cc) BY - NC - SA

Übungsaufgaben:

Ü5.1 Vollziehen Sie die automatisierte Abarbeitung einer Anfrage nochmals Schritt für Schritt mit Hilfe des trace-Befehls nach.
Verwenden Sie hierfür die Anfrage *[?-] enkeltochter(callopia,X).*
Geben Sie zudem alle Ausgabewerte für *X* an.

Ü5.2 Ergänzen Sie eine Regel für den Enkelsohn, wenden Sie den trace-Befehl auf die folgenden Anfragen an und notieren Sie erneut alle Ausgabewerte:

[?-] enkelsohn(X,astromir). *[?-] enkelsohn(X,Y).*

Ü5.3 Erstellen Sie eine Regel *weiblicheNachfahren/2*, mit der alle weiblichen Nachfahren einer Person ausgegeben werden können. Untersuchen Sie auch hier die Abarbeitungsreihenfolge unter Zuhilfenahme des trace-Befehls und notieren Sie sich die Ausgabewerte für die Anfrage

[?-] weiblicheNachfahren(X,astromir).

TIPP: Erzeugen Sie dazu unter Verwendung der Regeln *sohn/2* und *tochter/2* zunächst eine rekursive Regel *alleNachfahren/2.*

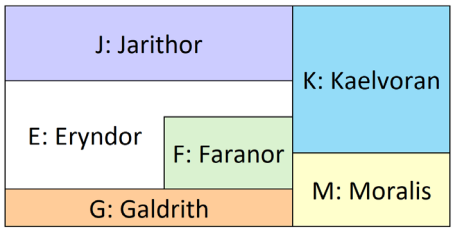
Lösungen:

(cc) BY - NC - SA

6) Logikrätsel – Teil 1: Karten färben

Ein beliebtes Logikrätsel ist das Färben von Karten mit möglichst wenigen Farben. Dabei sollen aneinandergrenzende Länder stets unterschiedlich gefärbt sein. In der Mathematik dauerte es über 100 Jahre, bis der berühmte Vier-Farben-Satz schließlich mithilfe von Computern bewiesen werden konnte.

Auch in Refrakturien gibt es verschiedenste Ländereien. Ihre Lage auf der Karte ist für einige davon schematisch dargestellt.



Ziel ist es, eine mögliche Färbung dieser Karte mit vier Farben durch PROLOG erstellen zu lassen. Als Farben sollen rot, grün (gruen), blau und gelb verwendet werden.

A6.1 Legen Sie in einem Prädikat *nachbar(Farbe1,Farbe2)* alle zulässigen Farbkombinationen für benachbarte Länder fest. Beachten Sie, dass es eine Rolle spielt, welche Farbe zuerst genannt wird.

A6.2 Überlegen Sie sich, was die nachfolgende Regel *karte1/3* bewirkt und welche Resultate daher der Aufruf ‚*[?-] karte1(Eriador,Enedwaith,Rhovanion).*‘ liefert.

```
karte1(A,B,C) :-
        nachbar(A,B),
        nachbar(A,C),
        nachbar(B,C).
```

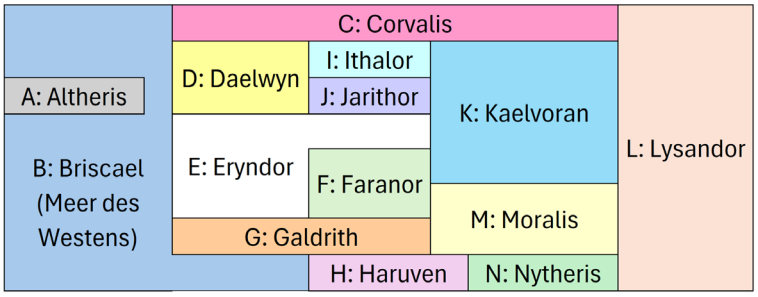
A6.3 Erstellen Sie eine geeignete Regel *karte/6*, die alle Nachbarschaftsbeziehungen der oben abgebildeten Karte beinhaltet.

(cc) BY - NC - SA

A6.4 Versuchen Sie, die gegebene Karte mit den drei Farben rot, grün und blau zu färben.

A6.5 Erstellen Sie eine geeignete Regel *karte2/14*, die die Färbung der vollständigen (schematischen) Karte Refrakturiens realisiert.

Prüfen Sie auch hier, ob drei Farben genügen.



(cc) BY - NC - SA

7) Logikrätsel – Teil 2: Einsteinrätsel bzw. Logicals

Eine weitere beliebte Form von Logikrätseln sind die sogenannten „Einsteinrätsel“ oder „Logicals“ – vgl. z.B. https://de.wikipedia.org/wiki/Zebrarätsel. Sie können durch logisches Schließen gelöst werden und sind damit geradezu prädestiniert für den PROLOG-Einsatz.

Um derartige Logikrätsel in PROLOG umzusetzen*, muss man eine geeignete Datenstruktur anlegen. Hierfür werden Listen eingesetzt, welche sich für diesen Zweck sehr intuitiv nutzen lassen. Ein kurzes Beispiel folgt auf der übernächsten Seite.

Es bietet sich an, ein Logical zunächst händisch zu lösen. Ein Beispiel hierfür findet sich auf der nächsten Seite.

TIPPs: Es ist für derartige Rätsel hilfreich, in einer geeigneten Tabelle zu notieren, ob zwei Fakten zusammengehören (+), oder nicht (-). Nach und nach müssen alle Hinweise in die Tabelle eingetragen werden.

Lautet der erste Hinweis beispielsweise, dass sich Luminalf am Montag mit Person A getroffen hat, so lässt sich dies wie abgebildet notieren.

	I	II	III	IV	V	A	B	C	D	E
Montag						+	-	-	-	-
Dienstag						-				
Mittwoch						-				
Donnerstag						-				
Freitag						-				
A										
B										
C										
D										
E										

Ist zudem vorgegeben, dass Luminalf am Montag nicht mit Tätigkeit I beschäftigt war, dann kann auch dieser Hinweis verwertet werden.

Zudem kann aber auch ausgeschlossen werden, dass Person A an Tätigkeit I beteiligt war, da diese ja am Montag hätte stattfinden müssen. Solche „Übertragungen von Ausschlüssen“ werden umso schwieriger nachvollziehbar, je komplexer die Sachverhalte sind. Hierbei hilft ein Blick in die Tabelle: Das Pluszeichen identifiziert Montag mit Person A. Daher müssen die Ausschlüsse von Montag und Person A auch übereinstimmen.

	I	II	III	IV	V	A	B	C	D	E
Montag	-					+	-	-	-	-
Dienstag						-				
Mittwoch						-				
Donnerstag						-				
Freitag						-				
A										
B										
C										
D										
E										

	I	II	III	IV	V	A	B	C	D	E
Montag	-					+	-	-	-	-
Dienstag						-				
Mittwoch						-				
Donnerstag						-				
Freitag						-				
A	-									
B										
C										
D										
E										

* nach: Informatik mit Prolog – Sekundarstufe II, Autor: Gerhard Röhner, Kapitel 7.4

(cc) BY - NC - SA

Luminalf hatte eine ereignisreiche Woche. Er lässt sie am Samstag in Bran im Wirtshaus „Zur Funkelnden Sonne“ bei einem Krug Bier Revue passieren.

An **welchem Wochentag** (Montag bis Freitag) hat sich Luminalf **in welchem Ort**, **mit wem** getroffen und ist dabei **welcher Beschäftigung** nachgegangen?

Zu beachten ist dabei, dass Luminalf sich an jedem Wochentag mit genau einer Person an genau einem Ort traf und genau einer Beschäftigung nachging.

Hinweise:

1) Luminalfs Kamerad am Montag – mit dem er keine Kneipentour unternommen hat – war nicht Brillo. Mit Brillo traf sich Luminalf in Malingen.

2) Am Dienstag traf sich Luminalf mit Frieda. Das war weder in Bran noch in Nitrilwacht.

	Absorgorn	Brillo	Frieda	Schattuman	Solariel	Angeln	Kneipentour	Wanderung	Zauberkräfte messen	Zukunftsvisionen	Beerenwein-Brücke	Bran	Brechzal	Malingen	Nitrilwacht
Montag															
Dienstag															
Mittwoch															
Donnerstag															
Freitag															
Beerenwein-Brücke															
Bran															
Brechzal															
Malingen															
Nitrilwacht															
Angeln															
Kneipentour															
Wanderung															
Zauberkräfte messen															
Zukunftsvisionen															

3) Mit Absorgorn machte er eine Kneipentour.

4) Am Mittwoch war Luminalf – nicht mit Schattuman – in Brechzal unterwegs.

5) An der Beerenwein-Brücke war Luminalf zum Angeln.

6) Am Donnerstag und beim Austauschen von Zukunftsvisionen war Luminalf in zwei verschiedenen Orten, die beide mit einem B anfangen.

7) Am Freitag ging Luminalf wandern.

Lösung:

Wochentag	Person	Tätigkeit	Ort

(cc) BY - NC - SA

Wir nehmen folgenden Sachverhalt als **Einführungsbeispiel** – wobei wir kurz vergessen, welchem Volk die drei Charaktere angehören:

„Die drei Freunde Absorgorn, Linsolas und Grummli zählen in großen Schlachten gegen Orkheere gern die besiegten Gegner. Anschließend erstellen Sie eine Rangliste (mit drei Plätzen), wer die Nase vorn hat.

Jeder der drei bevorzugt eine andere Waffe und gehört einem anderen Volk an.

(H1) Grummli mag seine Kriegsaxt und war besser als der Mensch.
(H2) Linsolas, der Elb, war besser als der Schwertkämpfer.
(H3) Der Bogenschütze gewann.

Wer ist der Zwerg? Welche Waffe nutzt Absorgorn?“

Wir nutzen eine Liste mit dem Namen *Freunde*, die drei Elemente hat. Jedes Element steht für einen der drei Freunde. Die Listenelemente sind selbst wieder Listen mit den vier Werten für *Vorname, Lieblingswaffe, Volk* und *Platzierung*.

Eine mögliche Vorgehensweise wird anhand der folgenden PROLOG-Umsetzung erläutert:

```
loeseLogik(Freunde):-
    Freunde =  [[grummli,kriegsaxt,_,P1],      % Hinweis (H1) umsetzen
                [linsolas,_,elb,P2],           % Hinweis (H2) umsetzen
                [absorgorn,_,_,_]],            % gegebenen Namen der 3. Person nutzen
    member([_,_,_,1], Freunde),                % es gibt einen 1. Platz in der Rangliste
    member([_,_,_,2], Freunde),                %                 2. Platz
    member([_,_,_,3], Freunde),                %                 3. Platz
    member([_,_,mensch,Pa], Freunde),          % Hinweis (H1) zur relativen Platzierung
    P1 < Pa,                                   %      von Grummli und Mensch
    member([_,schwert,_,Pb], Freunde),         % Hinweis (H2) zur relativen Platzierung
    P2 < Pb,                                   %      von Linsolas und Schwertkämpfer
    member([_,bogen,_,1], Freunde),            % Bogenschütze auf Platz 1
    member([_,_,zwerg,_], Freunde).            % die dritte Person ist ein Zwerg
```

Z.2 Die gegebenen Informationen aus (H1) [wie den Namen *grummli* und die Tatsache, dass dieser die *kriegsaxt* bevorzugt] tragen wir direkt in die Datenstruktur ein.

Die Variable *P1* muss für die Ranglistenplatzierung von Grummli gesetzt werden, da an ihren Wertebereich eine zusätzliche Bedingung [„besser als der Mensch“] geknüpft ist.

Der noch fehlende Listeneintrag [das Volk, dem Grummli angehört] wird mit einem Unterstrich _ „belegt“. Damit wird an dieser Stelle noch keine Aussage zur Belegung des Eintrags gemacht.

Z.3 Gegebene Informationen zum Vornamen [*linsolas*] und zum Volk [*elb*] werden umgesetzt. Für die Platzierung wird eine weitere Variable *P2* benötigt.

(cc) BY - NC - SA

Z.4 Da die Einträge in der Liste *Freunde* den Personen zugeordnet wurden, muss an dieser Stelle die Information zum Vornamen der dritten Person [*absorgorn*] realisiert werden.

Z.5-7 Mit den *member*-Befehlen können weitere Bedingungen an die *Freunde*-liste umgesetzt werden. Beispielsweise garantiert *member([_,_,_,1], Freunde),* dass es in der Datenstruktur einen Eintrag gibt, dessen Platzierung den Wert *1* besitzt.

Z.8-9 Für die Information aus (H1) [Grummlis Platzierung ist besser als die des Menschen] wird eine weitere Variable *Pa* benötigt. Ihr Wert [Platzierung des Menschen]muss größer sein, als der Wert der Variablen *P1* [Grummlis Platzierung].

Über den *member*-Befehl wird die Variable *Pa* dem Menschen zugeordnet.

Z.10-11 Identisches Vorgehen für die Information aus (H2) [Elb (*P2*) war besser als der Schwertkämpfer (*Pb*): *P2 < Pb*].

Z.12 Die Information aus (H3) [Bogenschütze auf Platz 1] wird realisiert.

Z.13 Die Wertemenge für die Völker wird vervollständigt [einer der Freunde ist ein Zwerg].

In der rechts abgebildeten Variante *loeseLogik1* wird ersichtlich, dass sich auch eine Sortierung der Ergebnisse nach der Platzierung im Wettbewerb erreichen lässt. Hierzu können die ursprünglichen Einträge in der Initialisierung der Liste mit entsprechenden *member*-Einträgen getauscht werden.

```
1  loeseLogik(Freunde):-
2      Freunde = [[grummli,kriegsaxt,_,P1],
3                 [linsolas,_,elb,P2],
4                 [absorgorn,_,_,_]],
5      member([_,_,_,1], Freunde),
6      member([_,_,_,2], Freunde),
7      member([_,_,_,3], Freunde),
8      member([_,_,mensch,Pa], Freunde),
9      P1 < Pa,
10     member([_,schwert,_,Pb], Freunde),
11     P2 < Pb,
12     member([_,bogen,_,1], Freunde),
13     member([_,_,zwerg,_], Freunde).
```

```
loeseLogik1(Freunde):-
    Freunde =     [[_,_,_,1],
                   [_,_,_,2],
                   [_,_,_,3]],
    member([grummli,kriegsaxt,_,P1], Freunde),
    member([linsolas,_,elb,P2], Freunde),
    member([absorgorn,_,_,_], Freunde),
    member([_,_,mensch,Pa], Freunde),
    P1 < Pa,
    member([_,schwert,_,Pb], Freunde),
    P2 < Pb,
    member([_,bogen,_,1], Freunde),
    member([_,_,zwerg,_], Freunde).
```

TIPP: Um die Abarbeitung durch die Inferenzmaschine möglichst effizient zu gestalten, sollten nach dem Einführen neuer Variablen umgehend alle Einschränkungen an deren Wertebereich getroffen werden [*Pa*: Z.8 → 9, *Pb*: Z.10 → 11].

(cc) BY - NC - SA

Will man die ausgegebene Liste noch übersichtlicher (z.B. als Tabelle) formatieren, so lässt sich dies erreichen, indem der *use_rendering*-Befehl eingesetzt wird:

```
:- use_rendering(table, [header(p('Name','Waffe','Volk','Platz'))]).

loeseLogik2(Freunde):-
    Freunde = [p(_,_,_,1),
               p(_,_,_,2),
               p(_,_,_,3)],
    member(p(grummli,kriegsaxt,_,P1), Freunde),
    member(p(linsolas,_,elb,P2), Freunde),
    member(p(absorgorn,_,_,_), Freunde),
    member(p(_,_,mensch,Pa), Freunde),
    P1 < Pa,
    member(p(_,schwert,_,Pb), Freunde),
    P2 < Pb,
    member(p(_,bogen,_,1), Freunde),
    member(p(_,_,zwerg,_), Freunde).
```

Ergebnisse der Anfragen *?-loeseLogik(Freunde).*, *?-loeseLogik1(Freunde1).* bzw. *?-loeseLogik2(Freunde2).*:

Freunde =
[[grummli, kriegsaxt, zwerg, 2], [linsolas, bogen, elb, 1], [absorgorn, schwert, mensch, 3]]

Freunde1 =
[[linsolas, bogen, elb, 1], [grummli, kriegsaxt, zwerg, 2], [absorgorn, schwert, mensch, 3]]

Freunde2 =

Name	Waffe	Volk	Platz
linsolas	bogen	elb	1
grummli	kriegsaxt	zwerg	2
absorgorn	schwert	mensch	3

Übungsaufgaben:

Ü7.1 Setzen Sie das Logical „Luminalfs Wochenrückblick“ in PROLOG in einer Regel *loeseHdRLogik(Luminalf)* um.

Ü7.2 Setzen Sie das Logical „Luminalfs Wochenrückblick“ in PROLOG in einer Regel *loeseHdRLogik2(Luminalf)* um und nutzen Sie dabei eine formatierte Ausgabe.

TIPPs: Für die Umsetzung muss zudem gefordert werden, dass jede Variablenbelegung nur einmalig auftreten darf, beispielsweise Frieda nur in einer Zeile der Tabelle stehen darf. Zudem müssen die möglichen Variablenbelegungen angegeben werden. Diese können als Fakten außerhalb der Regel oder als Liste innerhalb der Regel angelegt werden. Denkbar für die Anlage als Liste wäre hierfür im obigen Beispiel:

```
Platzierungen = [1,2,3],
member(Pa,Platzierungen), …
```


(cc) BY - NC - SA

EXKURS: Aussagenlogik

Arbeitsauftrag: Recherchieren Sie im Internet wichtige Begriffe zum Thema „Aussagenlogik“. Nicht fehlen sollten dabei folgende Schlagwörter: Aussage, Negation, Konjunktion, Disjunktion, Alternative, Implikation, Äquivalenz und Wahrheitstabellen.

Aussagen

Jeder **Aussage** können nur zwei mögliche Wahrheitswerte zugeordnet werden: wahr/falsch.

Beispiele:
Die Aussage A: „Frieda ist ein Malin.“ ist wahr.
Die Aussage B: „Alle Malins heißen Frieda.“ ist falsch.

Hinweis:
Zu jeder Aussage werden die zugehörigen Wahrheitstabellen dargestellt. Ihre Bedeutung wird am Ende des Exkurses erläutert.

Negation

Die Verneinung bzw. das Gegenteil einer Aussage A bezeichnet man als **Negation**.

Schreibweise: $\neg A$ bzw. $\bar{A}$ (sprich: „nicht A“)

Offensichtlich ist $\bar{A}$ wahr, wenn A falsch ist und umgekehrt.

Beispiele:
Die Aussage $\bar{A}$ würde lauten: „Frieda ist kein Malin“.
Die Aussage $\bar{B}$ würde lauten: „Es gibt mindestens einen Malin, der nicht Frieda heißt“.

A	$\bar{A}$
w	f
f	w

Konjunktion (UND-Verknüpfung)

Die Verknüpfung zweier Aussagen A und B durch „und“ heißt **Konjunktion**.

Schreibweise: $A \wedge B$ (sprich: „A und B“)

Die Konjunktion $A \wedge B$ zweier Aussagen A und B ist genau dann wahr, wenn **sowohl A als auch B wahr** ist.

Beispiel:
Für die Aussagen A und B wie oben lautet die Konjunktion $A \wedge B$: „Frieda ist ein Malin **und** alle Malins heißen Frieda.“. Die Konjunktion $A \wedge B$ ist damit falsch!

A	B	$A \wedge B$
w	w	w
w	f	f
f	w	f
f	f	f

Disjunktion (ODER-Verknüpfung)

Die Verknüpfung zweier Aussagen A und B durch „oder“ heißt **Disjunktion**.

Schreibweise: $A \vee B$ (sprich: „A oder B“)

Die Disjunktion $A \vee B$ zweier Aussagen A und B ist genau dann wahr, wenn **mindestens eine der beiden Aussagen wahr** ist.

Beispiel:
Für die Aussagen A und B wie oben lautet die Disjunktion $A \vee B$ („Frieda ist ein Malin **oder** alle Malins heißen Frieda“) wahr.

A	B	$A \vee B$
w	w	w
w	f	w
f	w	w
f	f	f

(cc) BY - NC - SA

Alternative (Exklusive ODER-Verknüpfung)

Die Verknüpfung zweier Aussagen A und B durch „oder“ (im Sinne von „entweder – oder“) heißt **Alternative**.

Schreibweise: $A \dot{\vee} B$ (sprich: „entweder A oder B“)

Die Alternative $A \dot{\vee} B$ zweier Aussagen A und B ist genau dann wahr, wenn genau eine der Aussagen A bzw. B wahr ist.

A	B	$A \dot{\vee} B$
w	w	f
w	f	w
f	w	w
f	f	f

Beispiel:
Für die Aussagen A und B wie oben ist die Alternative $A \dot{\vee} B$ („**entweder** ist Frieda ein Malin **oder** alle Malins heißen Frieda“) wahr.

Anmerkung: Diese Form der Verknüpfung taucht wesentlich seltener auf als die Konjunktion bzw. die Disjunktion. In unserem Gebrauch bezeichnet ein ODER immer die Disjunktion.

Implikation (WENN-DANN-Verknüpfung)

Die Verknüpfung zweier Aussagen A und B durch „wenn – dann“ heißt **Implikation**.

Schreibweise: $A \Rightarrow B$ (sprich „aus A folgt B“ bzw. „wenn A dann B“)

Die Implikation $A \Rightarrow B$ zweier Aussagen A und B ist nur dann falsch, wenn A (die Voraussetzung oder Prämisse der Implikation) wahr und B (die Folgerung oder Konklusion) falsch ist. Daher lässt sich die Implikation auch schreiben als $\overline{A} \vee B$.

A	B	$A \Rightarrow B$
w	w	w
w	f	f
f	w	w
f	f	w

Zu einer Implikation $A \Rightarrow B$ lassen sich zudem formulieren, ihre…

- …**Umkehrung**: $B \Rightarrow A$
- …**Negation**: $A \wedge \bar{B}$
- …**Kontraposition**: $\bar{B} \Rightarrow \bar{A}$

Äquivalenz (GENAU DANN, WENN – Verknüpfung)

Die Verknüpfung zweier Aussagen A und B durch „genau dann, wenn“ (oder „dann und nur dann, wenn“) heißt **Äquivalenz**.

A	B	$A \Leftrightarrow B$
w	w	w
w	f	f
f	w	f
f	f	w

Schreibweise: $A \Leftrightarrow B$ (sprich: „A genau dann, wenn B“)

Anmerkung: Die Äquivalenz $A \Leftrightarrow B$ kann als Konjunktion zweier Implikationen aufgefasst werden: $(A \Rightarrow B) \wedge (B \Rightarrow A)$

(cc) BY - NC - SA

Wahrheitstabellen

Mit Wahrheitstabellen lassen sich unterschiedliche Aussagen auf Äquivalenz prüfen.

Beispiel:
Überprüfen Sie, ob die Aussage $\bar{A} \Leftrightarrow \bar{B}$ äquivalent zu $A \Leftrightarrow B$ ist.

Man notiert zunächst die einzelnen beteiligten Aussagen in einer Tabelle und trägt alle möglichen Kombinationen von wahr/falsch ein.

Dann entwickelt man schrittweise Wahrheitswerte für die zu untersuchenden Aussagen:

$\boldsymbol{A}$	$\boldsymbol{B}$	$\boldsymbol{A \Rightarrow B}$	$\boldsymbol{B \Rightarrow A}$	$\boldsymbol{A \Leftrightarrow B}$	$\boldsymbol{\bar{A}}$	$\boldsymbol{\bar{B}}$	$\boldsymbol{\bar{A} \Rightarrow \bar{B}}$	$\boldsymbol{\bar{B} \Rightarrow \bar{A}}$	$\boldsymbol{\bar{A} \Leftrightarrow \bar{B}}$
w	w	w	w	w	f	f	w	w	w
w	f	f	w	f	f	w	w	f	f
f	w	w	f	f	w	f	f	w	f
f	f	w	w	w	w	w	w	w	w

Da die Wahrheitswerte in beiden Spalten übereinstimmen, sind die Aussagen $\bar{A} \Leftrightarrow \bar{B}$ und $A \Leftrightarrow B$ zueinander äquivalent.

Weitere Beispiele für Wahrheitstabellen etc. finden sich unter
https://www.lernhelfer.de/schuelerlexikon/mathematik-abitur/artikel/logische-operationen-mit-aussagen#

(cc) BY - NC - SA

Übungsaufgaben:

E1.1 Gegeben ist das folgende Sprichwort:
„Wenn die Katze aus dem Haus ist, dann tanzen die Mäuse auf dem Tisch.“
Formulieren Sie hierzu die

- Kontraposition
- Negation
- Umkehrung
- Negation der Umkehrung
- Kontraposition der Umkehrung
- Negation der Kontraposition

Diese Aufgabe lässt sich auch online unter
https://wp.uni-koblenz.de/vorkurs/variante-b/beweismethoden-teil-1/
bearbeiten.

E1.2 Erläutern Sie jeweils kurz, ob für die Aussagen
A: „Die Zahl 9 ist Teiler von x.“ und B: „Die Zahl 3 ist Teiler von x.“

- die Implikation $A \Rightarrow B$
- die Umkehrung zu $A \Rightarrow B$
- die Negation zu $A \Rightarrow B$
- die Kontraposition zu $A \Rightarrow B$

wahr oder falsch ist.

E1.3 Prüfen Sie mittels geeigneter Wahrheitstabellen, ob eine beliebige Implikation $A \Rightarrow B$ zu ihrer Umkehrung/Negation/Kontraposition äquivalent ist.

E1.4 Sehen Sie sich die ersten drei Videos unter
https://wp.uni-koblenz.de/vorkurs/variante-b/beweismethoden-teil-1/
an und beantworten Sie dabei auch die darin enthaltenen kurzen Fragen und den Beweis zur Teilbarkeit:

Behauptung: Für alle $a, b \in \mathbb{Z}$ gilt: Falls a und b ungerade sind, so ist $a + b$ $\square$.

Beweis: Seien $a, b \in \mathbb{Z}$ $\square$.
Da a ungerade ist, gibt es ein $\square$ mit $\square$.
Da b ungerade ist, gibt es ein $\square$ mit $\square$.
$\Rightarrow a + b = \square + \square = \square = 2(\square)$.
Also ist $a + b$ $\square$.

(cc) BY - NC - SA

Lösungen:

Kontraposition $\overline{B} \Rightarrow \overline{A}$:

Negation $A \wedge \overline{B}$:

Umkehrung $B \Rightarrow A$:

Negation der Umkehrung $B \wedge \overline{A}$:

Kontraposition der Umkehrung $\overline{A} \Rightarrow \overline{B}$:

Negation der Kontraposition $\overline{B} \wedge A$:

(cc) BY - NC - SA

(cc) BY - NC - SA

EXKURS: Arithmetik

PROLOG ist keine besonders geeignete Sprache zur Lösung numerischer Probleme. Die Arithmetik beschränkt sich oft auf elementare Berechnungen mit Integer-Zahlen. Dennoch soll ein kurzer Einblick in die Möglichkeiten gegeben werden.*

Untersuchen Sie, welche Antworten der PROLOG-Interpreter für folgende Eingaben gibt:

a) *[?-] 3 =< 4.*
b) *[?-] X >= 5.*
c) *[?-] X = 2, X < 3.*
d) *[?-] X = 2, Y = 3, X < Y.*
e) *[?-] X is 5 + 6*3.*
f) *[?-] X = 5 + 6*3.*
g) *[?-] 6 * -7 is Y.*
h) *[?-] 4 * 5 < 21.*
i) *[?-] 4 * 5 is 20.*
j) *[?-] 4 * 5 = 20.*
k) *[?-] 20 is 4*5.*
l) *[?-] 20 = 4*5.*

PROLOG-Terminologie der **Grundrechenarten**:

X + Y	Addition	*X / Y*	Real-Division
X - Y	Subtraktion	*X // Y* bzw. *X div Y*	Integer-Division
*X * Y*	Multiplikation	*X mod Y*	Rest der Integer-Division

PROLOG-Terminologie der **Vergleichsoperatoren**:

X =:= Y	numerisch gleich	*X > Y*	X ist größer als Y
X =\= Y	numerisch ungleich	*X =< Y*	X ist kleiner als oder gleich Y
X < Y	X ist kleiner als Y	*X >= Y*	X ist größer als oder gleich Y

* nach: Informatik mit Prolog – Sekundarstufe II, Autor: Gerhard Röhner, Kapitel 6

(cc) BY - NC - SA

Neben der automatischen Auswertung arithmetischer Ausdrücke durch die Vergleichsoperatoren gibt es die Auswertung über den *is*-Operator. Dem *is* entspricht in Pascal/Java die Wertzuweisung mit ‚:=' bzw. ‚=':

Variable ***is*** *<numerischer Term>*

Der *is*-Operator muss vom Unifikationsoperator ‚=' unterschieden werden:

?- X = 4 + 3	ergibt die Antwort	X = 4 + 3
?- X is 4 + 3	ergibt die Antwort	X = 7

Übungsaufgaben:

E2.1 Geben Sie ein Prädikat *summe(X, Y, S)* an, das in *S* die Summe von *X* und *Y* liefert.

E2.2 Schreiben Sie ein Prädikat *teilt(X, Y)*, das erfüllt sein soll, falls *X* ein Teiler von *Y* ist.

E2.3 Schreiben Sie ein Prädikat *max(X, Y, Max)*, das im dritten Argument (*Max*) das Maximum der beiden Zahlen *X* und *Y* liefert.

E2.4 Unsere Abenteurer aus Refrakturien haben auf ihrer Reise einiges an Ausrüstung bei sich. Manches davon haben sie als kostbare Geschenke zur Unterstützung ihres Vorhabens erhalten. Beispielsweise trägt Absorgorn ein Paar Elbenstiefel, die keine Fußabrücke hinterlassen. Ihr Wert wird auf etwa 20 Goldtaler, sogenannte Doppelgroschen geschätzt. Zudem hat er eine Portion Popcorn im Wert von einem Doppelgroschen, sein Schwert Luminothar (die „Flamme des Nordens", ca. 50 Goldtaler) und eine Gürtelschnalle mit Flaschenöffner im Wert von vier Goldtalern bei sich. Auch Frieda ist mit Elbenstiefeln und einer Portion Popcorn ausgerüstet. Zudem trägt sie den Hammer *Schlag* (etwa zehn Goldtaler) und ein Paar Nitril-Handschuhe für 80 Goldtaler. Unübertroffen ist aber natürlich der Wert ihrer größten Last: *die Brille*, deren Wert sich kaum beziffern lässt. Für unsere Zwecke gehen wir von einem Preis von 100.000 Doppelgroschen aus.

Die Ausrüstung wird in die Kategorien Rüstung, Kleidung, Waffe, Schmuck und Verpflegung untergliedert. Legen Sie die Ausrüstungsgegenstände und deren Kosten in der Wissensbasis an. Ordnen Sie den genannten Personen ihre jeweilige Ausrüstung zu. Erstellen Sie zudem ein Prädikat *wert(Person,Summe)*, das den Gesamtwert der Ausrüstung einer Person in der Variable *Summe* berechnet.

TIPP: Es kann hilfreich sein, in einer der Kategorien einen Eintrag für einen nicht vorhandenen Gegenstand anzulegen.

(cc) BY - NC - SA

Lösungen:

(cc) BY - NC - SA