Prolog

1. Kapitel: Fakten, Regeln und Anfragen

Dozentin: Wiebke Petersen

Kursgrundlage: Learn Prolog Now (Blackburn, Bos, Striegnitz)

Allgemeines

- **Prolog** (aus dem franz. **Pro**gramming en **Log**ique) ist eine logische Programmiersprache, die in den 1970er Jahren von dem Informatiker Alain Colmerauer entwickelt wurde.
- Der Ursprung der logischen Programmierung liegt in dem automatisierten Beweisen mathematischer Sätze.
- Die logische Programmierung basiert auf der Syntax der Prädikatenlogik erster Stufe (PL1), die in der zweiten Hälfte des 19. Jahrhunderts von Gottlob Frege aufgestellt wurde.

Anwendungsgebiete

- Entwicklung von Expertensystemen
- Computerlinguistik
- künstliche Intelligenz
- Die Sprachverarbeitungskomponenten des KI-Systems Watson von IBM wurden in Prolog geschrieben.
  Watson bei Jeopardy (the IBM Challenge):
  `https://www.youtube.com/watch?v=P18EdAKuC1U`

Grundprinzip

- Prolog-Programme bestehen aus einer Wissensdatenbank, die aus Fakten und Regeln bestehen.
- Der Benutzer formuliert Anfragen an die Wissensdatenbank.
- Prolog versucht mittels bestehender Fakten und Regeln eine Antwort zu finden.
- Wenn Prolog eine Antwort findet, bedeutet dies, dass die Anfrage **logisch** ableitbar ist.
- Der Programmier-Ansatz von Prolog folgt dem deklarativen **Programmierparadigma**.

Programmierparadigmen

- Ein Programmierparadigma ist ein fundamentaler Programmierstil/-ansatz.
- Bei der Programmierung wird zwischen **zwei** grundlegenden Programmierparadigmen unterschieden:

  Imperative Programmierung und deklarative Programmierung

- Einige Sprachen der **imperativen** Programmierung: prozedural (C, Fortran), Objekt-orientiert (C++, Java, Python)
- Einige Sprachen der **deklarativen** Programmierung: Abfragesprachen (SQL), funktionale Sprachen (Lisp, Haskell), logische Sprachen (Prolog)

Vergleich der Programmierparadigmen

- Bei imperativen Programmiersprachen (links) (Python) wird beschrieben **WIE** ein Problem gelöst werden soll (algorithmischer Ablauf).
- Bei deklarativen Programmiersprachen (rechts) (Prolog) wird beschrieben **WAS** gelöst werden soll (Deklaration der Zusammenhänge).

```
def istGluecklich(n):
   if n == "tobi":
      print "true"
   else:
      print "false"
```

```
> istGluecklich("tobi")
true
> istGluecklich(ötto")
false
```

```
istGluecklich(tobi).
```

```
?- istGluecklich(tobi).
true.
?- istGluecklich(otto).
false.
```

Prolog's Programmierparadigma

- Eine ganz einfache Wissensbasis in Prolog erlaubt uns vier unterschiedliche Anfragen zu stellen:

```
mag(pluto,eis).
mag(pluto,orangen).
mag(popeye,eis).
```

1. Mag Popeye Eis?

```
mag(popeye,eis).
true.
```

2. Wer mag Eis?

```
mag(X,eis).
X = pluto;
X = popeye
```

Prolog's Programmierparadigma (2)

- Eine ganz einfache Wissensbasis in Prolog erlaubt uns vier unterschiedliche Anfragen zu stellen:

```
mag(pluto,eis).
mag(pluto,orangen).
mag(popeye,eis).
```

3. Was mag Pluto?

```
mag(pluto,X).
X = eis;
X = orangen;
```

4. Wer mag was?

```
mag(X,Y).
X = pluto,
Y = eis;
X = pluto,          |       X = popeye,
Y = orangen;        |       Y = eis
```

Verständnis für Prolog

- Das richtige Verständnis des deklarativen Programmier-paradigmas bedeutet das entsprechende Denk- oder Handlungsmuster zu verinnerlichen und gilt als **essentielle** Grundvorraussetzung für die Programmierung in Prolog.
- Prolog ist eine deklarative Programmiersprache!
- Für den Umgang mit Prolog ist es zwingend erforderlich sich vom imperativen Programmierparadigma zu lösen.

Der Interpreter

- Der Interpreter von Prolog benötigt Informationen um eine **Anfrage** (*query*) zu beantworten.
- Diese Informationen werden in einer **Wissensbasis** (*knowledge base*) gespeichert.
- Eine Wissensbasis besteht aus **Klauseln** (*clauses*).
  Klauseln sind entweder **Fakten** (*facts*) oder **Regeln** (*rules*)

Wissenbasis

Pluto ist ein Hund.
Snoopy ist ein Hund.
Popeye ist ein Seemann.
Popeye mag Spinat.
Popeye ist stark, wenn Popeye Spinat mag.

Der Interpreter

- Der Interpreter von Prolog benötigt Informationen um eine **Anfrage** (*query*) zu beantworten.
- Diese Informationen werden in einer **Wissensbasis** (*knowledge base*) gespeichert.
- Eine Wissensbasis besteht aus **Klauseln** (*clauses*).
  Klauseln sind entweder **Fakten** (*facts*) oder **Regeln** (*rules*)

Wissenbasis

Pluto ist ein Hund.
Snoopy ist ein Hund.
Popeye ist ein Seemann.
Popeye mag Spinat.
Popeye ist stark, wenn Popeye Spinat mag.

Anfragen

Ist Pluto ein Hund?
Mag Pluto Spinat?
Ist Garfield ein Hund?
Ist Popeye ein Mann?
Ist Popeye stark?

Der Interpreter

- Der Interpreter von Prolog benötigt Informationen um eine **Anfrage** (*query*) zu beantworten.
- Diese Informationen werden in einer **Wissensbasis** (*knowledge base*) gespeichert.
- Eine Wissensbasis besteht aus **Klauseln** (*clauses*).
  Klauseln sind entweder **Fakten** (*facts*) oder **Regeln** (*rules*)

Wissenbasis

Pluto ist ein Hund.
Snoopy ist ein Hund.
Popeye ist ein Seemann.
Popeye mag Spinat.
Popeye ist stark, wenn Popeye Spinat mag.

Anfragen

Ist Pluto ein Hund?
Mag Pluto Spinat?
Ist Garfield ein Hund?
Ist Popeye ein Mann?
Ist Popeye stark?

Der Prolog-Interpreter wird die erste und letzte Frage bejahen
und die anderen verneinen.

Für Prolog ist alles ‘wahr’ oder ‘beweisbar’, was als Fakt in der Wissensbasis steht oder sich mithilfe von Regeln in der Wissensbasis aus diesen Fakten herleiten lässt.

Eine Wissensbasis in Prolog

In Prolog wird eine Wissensbasis wie folgt geschrieben:

```
ist_ein_hund(pluto).
ist_ein_hund(snoopy).
ist_ein_seemann(popeye).
mag(popeye,spinat).

ist_stark(popeye) :- mag(popeye,spinat).
```

Diese Wissensbasis besteht aus vier Fakten und einer Regel. Sie definiert vier unterschiedliche **Prädikate** (*predicates*) nämlich `ist_ein_hund/1`, `ist_ein_seemann/1`, `mag/2` und `ist_stark/1`.
Anfragen werden in der Konsole an den Interpreter gestellt und ausgewertet:

```
?- ist_ein_hund(pluto).
true.
```

Eine Wissensbasis in Prolog

```
ist_ein_hund(pluto).
ist_ein_hund(snoopy).
ist_ein_seemann(popeye).
mag(popeye,spinat).

ist_stark(popeye) :- mag(popeye,spinat).
```

Das Symbol ':-' steht für 'wenn' oder 'folgt aus'. Die Regel '`ist_stark(popeye):- mag(popeye,spinat).`' kann gelesen werden als " '`ist_stark(popeye)`' ist wahr, wenn '`mag(popeye,spinat)`' wahr ist".

Fakten, Regeln, Klauseln

```
ist_ein_seemann(popeye).
mag(popeye,spinat).

ist_stark(popeye) :- mag(popeye,spinat).
hat_muskeln(popeye) :- hat_trainiert(popeye).
hat_muskeln(popeye) :- ist_stark(popeye).
```

- Die linke Seite einer Regel nennt man **Regelkopf** (*head of the rule*) und die rechte Seite **Regelkörper** (*body of the rule*).
- Fakten und Regeln einer Wissensbasis nennt man **Klauseln** (*clauses*).
- Bei einem Fakt kann man auch von einer leeren Regel (einer Regel ohne Regelkörper) sprechen:
  ‘`mag(popeye,spinat).`’ ist äquivalent zu ‘`mag(popeye,spinat):- .`’.

▸ Übung

Regeln und Inferenzen

```
ist_ein_seemann(popeye).
mag(popeye,spinat).
ist_stark(popeye) :- mag(popeye,spinat).
hat_muskeln(popeye) :- hat_trainiert(popeye).
hat_muskeln(popeye) :- ist_stark(popeye).
```

- Wenn der Regelkörper wahr ist (sich also aus der Wissensbasis herleiten lässt), so ist auch der Regelkopf wahr.
- Dieses Deduktionsprinzip heißt **Modus Ponens**:

$a \rightarrow b$	`b :- a.`	`ist_stark(popeye):- mag(popeye,spinat).`
a	`a.`	`mag(popeye,spinat).`
b	`b.`	`ist_stark(popeye).`

- Aus der Regel '`ist_stark(popeye):- mag(popeye,spinat).`' und dem Fakt '`mag(popeye,spinat).`' **inferiert** oder **schließt** der Prolog-Interpreter, dass '`ist_stark(popeye).`' gilt.

Stelligkeit

Prädikate können beliebige Stelligkeit haben:

```
es_regnet.
mag(popeye,spinat).
mag_spinat(popeye).
```

Der Fakt `es_regnet/0` ist nullstellig.
Der Fakt `mag/2` ist zweistellig.
Der Fakt `mag_spinat/1` ist einstellig.

Beachte, die beiden Prädikatsausdrücke `mag(popeye,spinat).` `mag_spinat(popeye).` modellieren zwar denselben Sachverhalt, sie sind aber nicht äquivalent.

0-stellige Fakten/Prädikate werden auch **atomare Fakten/Prädikate** genannt.

Regeln mit mehr als einem Ziel

- Das Komma drückt eine **Konjunktion** aus: Popeye mag Spinatnudeln, wenn Popeye Spinat mag **und** wenn Popeye Nudeln mag.

```
mag(popeye, spinatnudeln) :-
    mag(popeye,spinat),
    mag(popeye, nudeln).
```

- Das Semikolon steht für eine **Disjunktion**: Popeye ist stark, wenn Popeye Spinat mag **oder** wenn Popeye trainiert hat.

```
ist_stark(popeye) :-
    mag(popeye,spinat);
    hat_trainiert(popeye).
```

Zur Verbesserung der Lesbarkeit sollte man in Prolog besser zwei Regeln statt einer disjunktiven schreiben:

```
ist_stark(popeye) :- mag(popeye,spinat).
ist_stark(popeye) :- hat_trainiert(popeye).
```

Variablen

```
liebt(pluto,mickey).
liebt(mickey,pluto).
liebt(minnie,mickey).
liebt(mickey,minnie).

?- liebt(X,Y).
```

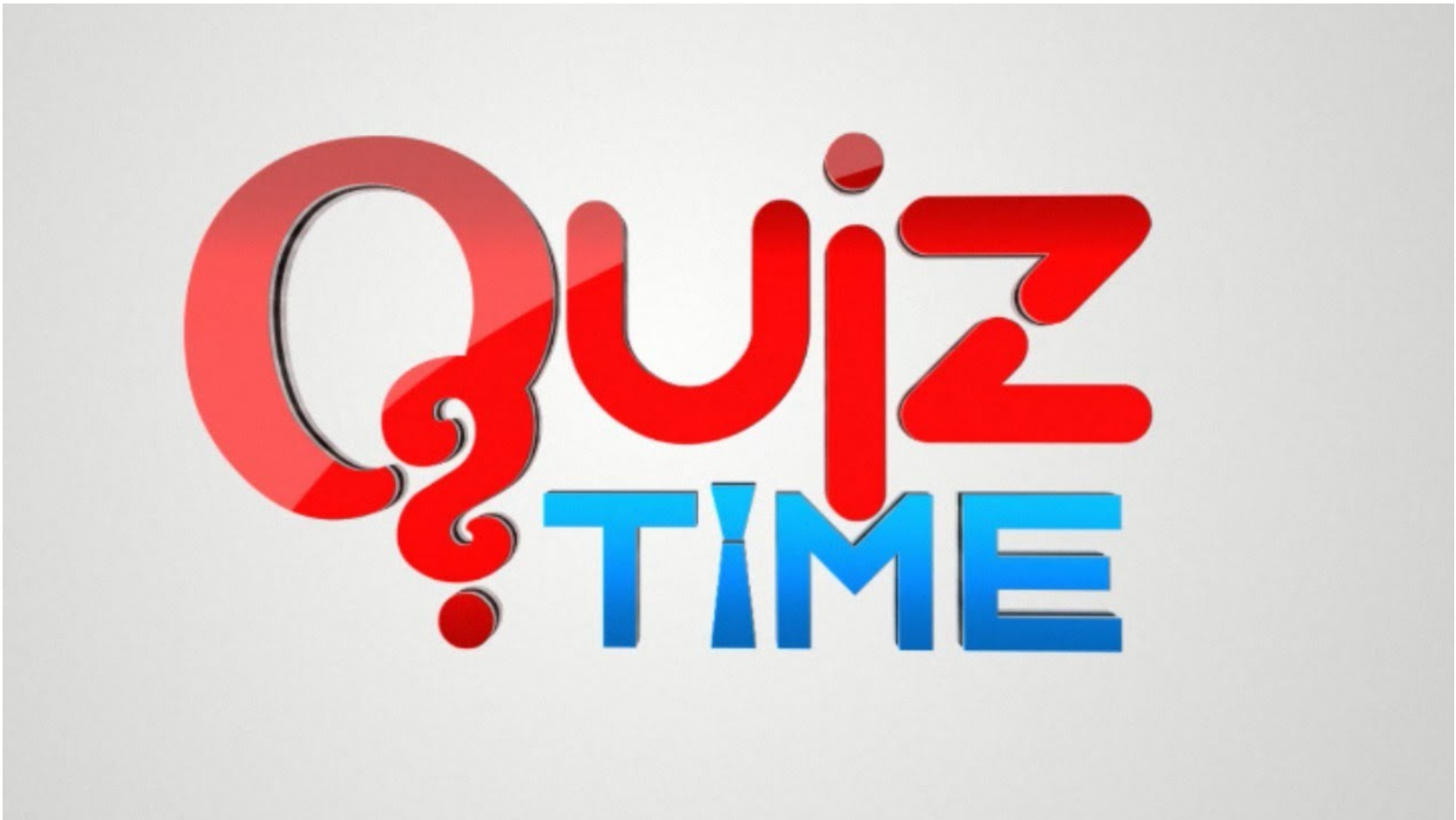
```
maus(X).
liebt(pluto,mickey).
liebt(minnie,mickey).
liebt(X,Y) :- liebt(Y,X).

?- liebt(mickey,Y).
```

- `X` und `Y` sind Variablen. Sie können sowohl in Anfragen als auch in der Wissensbasis verwendet werden.
- Erhält der Prolog-Interpreter eine Anfrage wie `?- liebt(X,Y).` versucht er die Variablen `X` und `Y` so zu **instantiieren** oder zu **binden** (sprich mit einem Wert zu belegen), dass die Aussage wahr wird.
- Gelingt dies, antwortet der Interpreter `true.` und gibt die gewählte erfolgreiche Instanziierung aus.
- Durch die Eingabe des Semikolons fordert man den Interpreter auf, weitere Antworten auf die Anfrage zu suchen.
- Die Klausel `maus(X).` ist ein **universeller Fakt** (Fakt mit offener Variable).

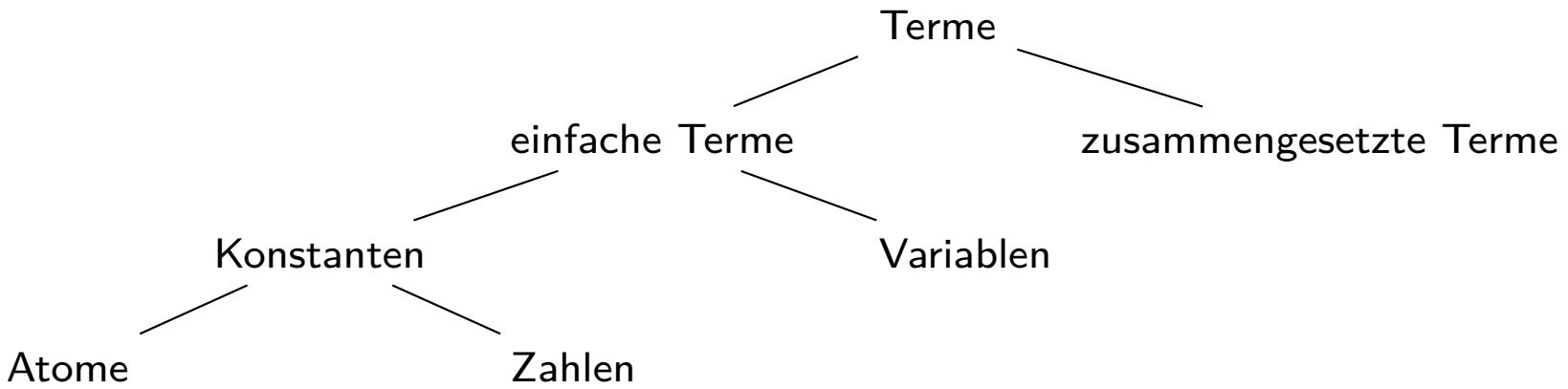
▸ Übung

Quiz-Time



Grundlagen

- Die grundlegende Datenstruktur in Prolog sind **Terme** (*terms*).
- Sie sind entweder **einfach** oder **zusammengesetzt**.
- Einfachen Terme in Prolog sind **Konstanten** (*constants*) und **Variablen** (*variables*)
- Die Konstanten sind **Atome** (*atoms*) und **Zahlen** (*numbers*).
- Zusammengesetzte Terme werden auch **komplexe Terme** oder **Strukturen** genannt.



Einfache Terme: Atome, Zahlen und Variablen

Atome sind Zeichenfolgen, die mit Kleinbuchstaben beginnen und nur aus Buchstaben, Ziffern und dem Unterstrich bestehen, oder Zeichenfolgen, die in Anführungszeichen stehen:
`popeye`, `hund13XYZ`, `my_dog`, `"Lea?!@"`, `'Homer Simpson'`.

Zahlen sind ganze Zahlen (*integers*) oder Kommazahlen (*floats*):
`123`, `89.5`, `0`, `-323`.

Variablen Variablen sind Zeichenfolgen, die mit einem Grossbuchstaben oder einem Unterstrich beginnen und nur aus Buchstaben, Ziffern und dem Unterstrich bestehen:
`X`, `Variable`, `_123`, `Hund_123`.

Hinweise:

- Verwenden Sie bitte immer 'sprechende' Namen für ihre Terme.
- Die Variable `_`, die nur aus dem Unterstrich besteht, ist die anonyme Variable (kommt später im Kurs). Sie sollte zunächst nicht von Ihnen verwendet werden.

Zusammengesetzte bzw. komplexe Terme

- Zusammengesetzte bzw. Termen bestehen aus einem **Funktor** (*functor*) und beliebig vielen **Argumenten** (*arguments*).
- Der Funktor ist immer ein Atom.
- Die Argumente sind einfache oder komplexe Terme.
- Beispiel für komplexe Terme: `liebt(popeye,spinat)`
- Beispiel für komplexe verschachtelte Terme: `befreundet(X,vater(vater(popeye)))`
- Unter der **Stelligkeit** (*arity*) eines komplexen Terms versteht man die Anzahl der Argumente.

▸ Übung

Prädikate

- Betrachtet man die Fakten einer Wissensbasis als leere Regeln, dann definieren alle Klauseln, deren Regelköpfe denselben Funktor **und** dieselbe Stelligkeit haben, zusammen ein **Prädikat**.
- **Vorsicht**: Enthält eine Wissensbasis die beiden Fakten `befreundet(popeye,pluto,garfield)` und `befreundet(X,mickey)` so definiert sie zwei verschiedene Prädikate, nämlich `befreundet/3` (3-stellig) und `befreundet/2` (2-stellig).
- Für einen guten Programmierstil beachte folgende Regeln:
  - Alle Klauseln, die zu einem Prädikat gehören, stehen direkt beieinander.
  - Die Verwendung zweier Prädikate mit identischem Funktor aber unterschiedlicher Stelligkeit geschieht nur im wohldurchdachten Ausnahmefall.
  - Jedes Prädikat wird kommentiert (Kommentarzeichen '%').

```
% liebt/2
% liebt(Person1,Person2)
liebt(pluto,mickey).
liebt(mickey,minnie).
liebt(X,Y):- liebt(Y,X).
```

Programmiertips

- Alle Klauseln werden mit einem Punkt abgeschlossen.
- Prädikate:
  - Alle Klauseln, die zu einem Prädikat gehören, stehen direkt beieinander.
  - Die Verwendung zweier Prädikate mit identischem Funktor aber unterschiedlicher Stelligkeit geschieht nur im wohldurchdachten Ausnahmefall.
  - Jedes Prädikat wird kommentiert (Kommentarzeichen '%').
  - Die Reihenfolge der Argumente ist bedeutsam '`kind(otto,piet)` $\neq$ `kind(piet,otto)`.
  - Wir können selber festlegen, welche Argumentposition wofür stehen sollen. Einmal getroffene Konvention beibehalten und kommentieren.

Quiz-Time

Programmieren in Prolog = Problemlösen

- Prolog ist eine deklarative Programmiersprache.
- Problemlösen in Prolog heißt Probleme beschreiben, indem Objekte mit ihren Eigenschaften und Beziehungen dargestellt werden.
  - Was ist das **Problem**?
  - Welche **Objekte** sind beteiligt?
  - Welche **Eigenschaften** und **Beziehungen**?

Zusammenfassung: Kapitel 1

Wir haben die Grundlagen und Anwendungsgebiete von Prolog kennengelernt.

- **Keywords**: Programmierparadigma, deklaratives Programmieren, Wissensbasis (Klauseln), Klauseln (Fakten, Regeln, Anfragen), Regeln (Regelkopf, Regelkörper), Prädikate, Terme (einfach und zusammengesetzt), Atome, Zahlen, Variablen, komplexe Terme (Funktor, Argument, Stelligkeit), Konjunktion, Disjunktion.
- **Wichtig**: Das deklarative Programmierparadigma von Prolog muss man verstehen.
  Prolog lernt man wie alle Programmiersprachen nur durch Programmieren!
- **Ausblick Kapitel 2**: Wie löst Prolog Anfragen? (Matching und Beweisführung)

Übung: Atome, Variablen und komplexe Terme

Welche der folgenden Ausdrücke sind Atome, welche sind Variablen, welche sind komplexe Terme und welche sind nichts von dem?

```
1  vINCENT
2  Hund
3  12
4  Hund!3
5  hund!3
6  variable23
7  Variable2000
8  mein_kleiner_hund
9  _mein_kleiner_hund
10 'mein kleiner hund'
11 mein kleiner hund
```

```
12 'Jules'
13 _Jules
14 '_Jules'
15 loves(Vincent,mia)
16 'loves(Vincent,mia)'
17 Butch(boxer)
18 boxer(Butch)
19 and(big(burger),kahuna(X))
20 _and(big(X),kahuna(X))
21 (Butch kills Vincent)
22 kills(Butch,Vincent
```

zurück

Übung: Klauseln, Regeln, Fakten, Prädikate

Wieviele Klauseln, Fakten, Regeln und Prädikate gibt es in der folgenden Wissensbasis?

```
woman(vincent).
woman(mia).
man(jules).
person(X) :- man(X); woman(X).
loves(X,Y) :- knows(Y,X).
loves(X,Y,Z):- love(X,Y),love(X,Z).
father(Y,Z) :- man(Y), son(Z,Y).
father(Y,Z) :- man(Y), daughter(Z,Y).
```

zurück

Übung: Formuliere Fakten und Regeln

Drücke folgende Fakten und Regeln in Prolog aus:

1. Lena ist hungrig.
2. Lena ist Ottos Mutter.
3. Lena mag alle, die ihr Schokolade schenken.
4. Lena mag alle, die gut singen können und gut kochen.
5. Lena mag alle, die ihr Schokolade oder Kekse schenken.
6. Alle Menschen sind sterblich.
7. Sokrates ist sterblich.
8. Eine Tochter einer Person ist ein weibliches Kind dieser Person.
9. Alle Hunde mögen Wurst.
10. Pluto mag alles, was Mickey ihm gibt.

Übung: Klauseln, Regeln, Fakten, Prädikate

Gegeben ist die folgende Wissensbasis:

```
wizard(ron).
hasWand(harry).
quidditchPlayer(harry).
wizard(X) :- hasBroom(X),hasWand(X).
hasBroom(X) :- quidditchPlayer(X).
```

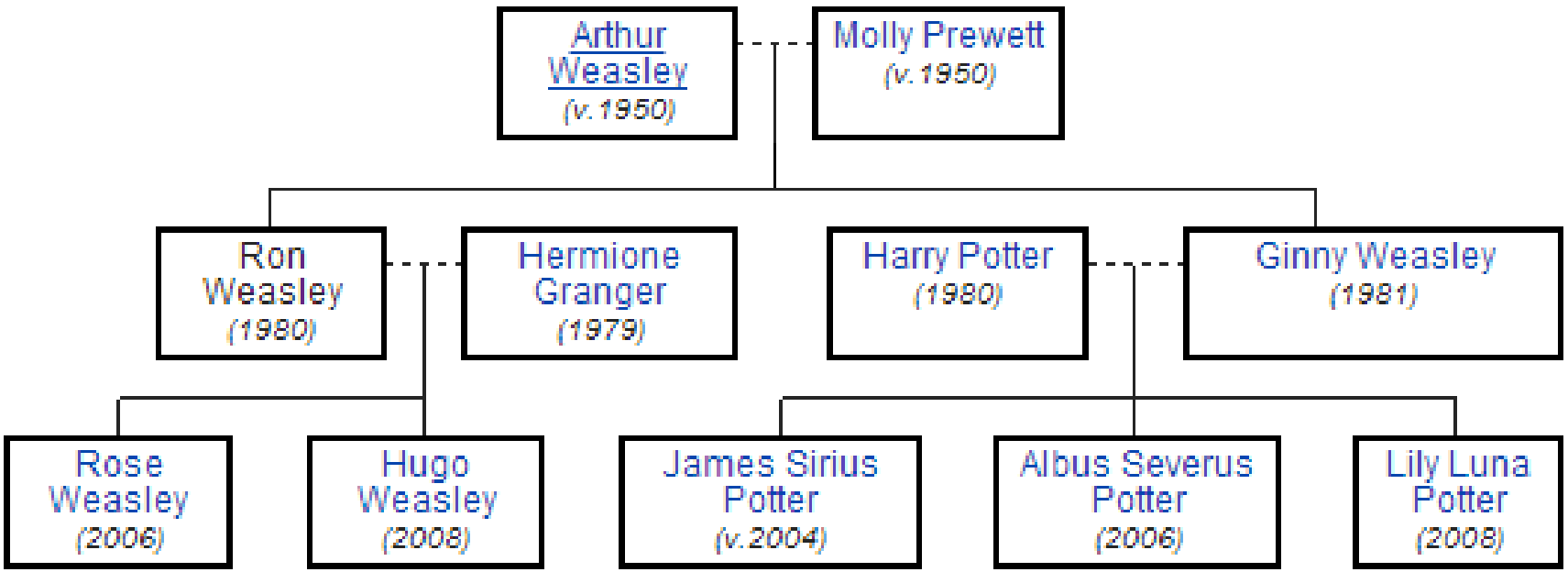
Welche Antworten gibt Prolog auf die folgenden Anfragen?

```
1 wizard(ron).
2 witch(ron).
3 wizard(hermione).
4 witch(hermione).
5 wizard(harry).
6 wizard(Y).
7 witch(Y).
```

zurück

Übung Familiendatenbank

Gegeben ist folgender Familienstammbaum:



Übung Familiendatenbank

Familienstammbaum dargestellt als Prologfakten:

```
% fem/1
% fem(X): X ist feminin
fem(hermione).
fem(ginny).
fem(molly).
fem(rose).
fem(lilly_luna).

% masc/1
% masc(X): X ist maskulin
masc(arthur).
masc(ron).
masc(hugo).
masc(james_sirius).
masc(harry).
masc(albus_severus).
```

```
% et/2
% et(X,Y): X ist ein
% Elternteil von Y
et(arthur,ron).
et(arthur,ginny).
et(molly,ron).
et(molly,ginny).
et(ron,rose).
et(ron,hugo).
et(hermione,rose).
et(hermione,hugo).
et(harry,james_sirius).
et(harry,albus_severus).
et(harry,lilly_luna).
et(ginny,james_sirius).
et(ginny,albus_severus).
et(ginny,lilly_luna).
```

Übung Familiendatenbank

Erweitere die Wissensbasis zum Familienstammbaum um folgende Prädikate:

```
mutter/2
vater/2
tochter/2
sohn/2
schwester/2
bruder/2
grossvater/2
```