Prolog

2. Kapitel: Matching und Beweisführung

Dozentin: Wiebke Petersen

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

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)

Einführung

- Prolog versucht Anfragen mittels Klauseln (Fakten und Regeln) einer Wissensbasis logisch abzuleiten bzw. zu beweisen.
- Dabei wird geprüft ob das Ziel einer Anfrage (die Zielklausel) eine logische Konsequenz der Programmklauseln (Wissensbasis) ist bzw. ob sich eine Anfrage des Benutzers gegeben als Zielklausel auf Grundlage der Programmklauseln beweisen lässt.
- Das Prinzip wird automatische **Beweisführung** (**proof search**) genannt und beschreibt den Ansatz den Prolog bei der Suche bzw. Beantwortung von Anfragen verfolgt.

Beweisführung

- Um eine Zielklausel zu beweisen, versucht Prolog die Klausel mit den in der Wissensbasis gegebenen Fakten und Regelköpfen zu **matchen** oder zu **unifizieren**.
- Wenn die Anfrage Variablen enthält muss eine gültige **Variablenbelegung** (**matching**) gefunden werden.
- Das Prinzip der automatischen Beweisführung von Prolog basiert auf dem Prinzip der **Unifikation** (**unification**) und des automatischen **Rücksetzens** (**backtracking**).

Das Matching bzw. die Unifikation

- Beim Matching handelt es sich um eine Operation, die zwei Terme miteinander vergleicht bzw. prüft ob diese durch eine geeignete Variablenbelegung gleichgesetzt (unifiziert) werden können.
- Das Matching ist ein Teil der automatischen Beweisführung. Es gibt jedoch auch das eingebaute Prädikat (auch **Unifikationsoperator** genannt) =, welches zwei Terme matcht.

Matchingregel

Zwei Terme matchen genau dann, wenn sie gleich sind oder wenn sie Variablen beinhalten, die so belegt werden können, dass die beiden Terme gleich werden.

Matching einfacher Terme: Konstanten

Zwei Konstanten matchen genau dann, wenn sie gleich sind.

```
?- =(popeye,popeye).
true.

?- =(popeye,'Popeye').
false.

?- =(popeye,'popeye').
true.

?- =(12,12).
true.

?- =(12,'12').
```

Das Prädikat = kann auch in der Infixnotation genutzt werden:

```
?- regen = schnee.
false.
```


Matching einfacher Terme: Variablen

- Wenn einer der Terme eine Variable ist, dann kann die Variable mit dem anderen Term belegt werden und beide Terme matchen.
- Dies funktioniert unabhängig davon, ob der andere Term einfach oder komplex ist.
- Besteht die Anfrage aus mehr als einer elementaren Zielklausel, muss zusätzlich die Variablenbelegung aller Elementarklauseln kompatibel sein.

```
?- =(mag_spinat(popeye),X).
X=mag_spinat(popeye).

?- X=Y, X=popeye.
X = popeye,
Y = popeye.

?- X=popeye, X=pluto.
false.
```


Matching komplexer Terme

Komplexe Terme matchen genau dann wenn:

1. die Terme den gleichen Funktor und dieselbe Stelligkeit haben **und**
2. alle korrespondierenden Argumente matchen **und**
3. die Variablenbelegungen miteinander kompatibel sind.

```
?- kill(shoot(gun),Y) = kill(X,stab(knife)).
X = shoot(gun),
Y = stab(knife).

?- kill(shoot(gun), stab(knife)) = kill(X,stab(Y)).
X = shoot(gun),
Y = knife.

?- mag(X,X) = mag(popeye,pluto).
false.
```

▸ Übung

Matching: Definition

Seien *term1* und *term2* zwei Terme.

1. Wenn *term1* und *term2* Konstanten sind, matchen sie genau dann, wenn sie das gleiche Atom oder die gleiche Zahl sind.
2. Wenn *term1* eine Variable ist, dann matchen *term1* und *term2*. Die Variable *term1* wird dann mit dem Term (*term2*) instantiiert (analog für den Fall, dass *term2* eine Variable ist).
3. Wenn *term1* und *term2* komplexe Terme sind, dann matchen sie genau dann, wenn:
   - die Terme den gleichen Funktor und dieselbe Stelligkeit haben **und**
   - alle korrespondierenden Argumente matchen **und**
   - die Variablenbelegungen miteinander kompatibel sind.

Wenn keine der drei Gegebenheiten zutrifft, matchen die beiden Terme nicht.

Unifikationsoperator

Neben dem Unifikationsoperator

```
sonne = regen.
=(sonne,regen).
```

gibt es auch den negierten Unifikationsoperator

```
sonne \= regen.
\=(sonne,regen).
```

Der Operator `\=/2` gelingt genau dann, wenn der Unifikationsoperator `=/2` scheitert.

▸ Übung

Prolog-Matching vs. Standardunifikation

zyklische Anfrage:

```
?- vater(X) = X.
```

In der Auswertung dieser Anfrage unterscheidet sich das Matching von Prolog von der Standardunifikation:

Antwort der Standardunifikation

Die Terme matchen **nicht**. Egal mit was man die Variable X belegt (z.B. X=vater(vater(lena)), hat der linke Term immer eine Klammerungsebene mehr als der rechte Term.
Standardunifikationsalgorithmen sind pessimistisch und prüfen vor jeder Unifikation, ob die zu unifizierenden Terme einen Zyklus aufbauen (occurs check).

Prolog-Matching vs. Standardunifikation

zyklische Anfrage:

```
?- vater(X) = X.
```

In der Auswertung dieser Anfrage unterscheidet sich das Matching von Prolog von der Standardunifikation:

Antwort von Prolog

Die Antwort hängt ab von der Prologimplementierung. Ältere Implementierungen erzeugen Auskünfte wie
`Not enough memory to complete query!.`
Neuere Implementierungen wie SWI-Prolog matchen die beiden Terme:

```
?- vater(X) = X.
X=vater(X)
```

Da Matching so zentral für Prolog ist und so häufig passiert ist Prolog optimistisch und erwartet, dass es nicht mit gefährlichen, sprich zyklischen Strukturen gefüttert wird.

Interne Repräsentation zyklischer Strukturen

Betrachte den trace zu den folgenden Anfragen:

```
?- X = vater(X), Y=X.
?- X = mag(X,Y), Y=X.
```

Der Tracemodus wird mit `trace.` ein- und mit `notrace.` ausgeschaltet.

Vermeiden Sie bei der Proglogprogrammierung solche zyklischen Definitionen!

Programmieren mit Matching

```
vertical(line(point(X,Y),point(X,Z))).
horizontal(line(point(X,Y),point(Z,Y))).
```

Was definiert diese Wissensbasis?

Einige Anfragen:

```
?- vertical(line(point(1,3),point(1,8))).
?- horizontal(line(point(1,3),point(4,Y))).
?- horizontal(line(point(1,3),point(Z,3))).
?- horizontal(line(point(1,3),point(Z,4))).
?- horizontal(line(point(1,3),P)).
```

▸ Übung

Wiederholung: Regeln

Regelform in Prolog:

```
ich_muede:- abend, dunkel.
```

Konklusion ← Prämisse UND Prämisse

abend, dunkel → ich_muede
Wenn es Abend ist und dunkel ist, dann bin ich müde.

Modus Ponens:

```
abend.
dunkel.
ich_muede:- abend, dunkel.
```

```
ich_muede.
```


Regeln und die Beweissuche

- Angenommen die Wissensbasis enthält eine Regel B :- A (d.h. wenn A gilt, dann gilt auch B.) und es soll B bewiesen werden.

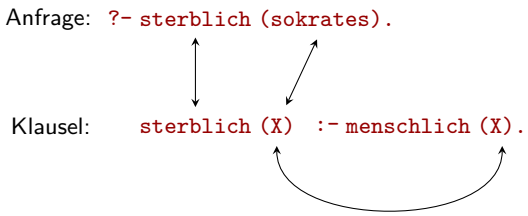
- Beweissuche:
  1. **B** soll bewiesen werden (es erfolgt eine Anfrage zu **B**).
  2. Matcht ein Fakt oder der Kopf einer Regel aus der Wissensbasis mit **B**?
  3. Ja und die Regel besagt das **B** wahr ist, wenn **A** auch wahr ist.
  4. Kann **A** bewiesen werden?
  5. Wenn **A** mit einem Fakt aus der Wissensbasis matcht (d.h. **A** ist wahr), so folgt, dass auch die Anfrage nach **B** wahr ist.

Beweisstrategie des Interpreters

- Anfrage gilt als zu beweisende **Behauptung**.
- **Head-Matching**: Anfrage **unifiziert** mit Kopf einer Klausel (also mit linker Regelseite oder Fakt).
- Unifikation liefert **Variablenbelegungen**.
- Klauselrumpf wird bewiesen.
- Alternative Lösungen über **Backtracking**.
- Reihenfolge der **Suchraumtraversierung**:
  top-down depth-first left-to-right

Head-Matching

Ein Prädikat aus einer Anfrage muß mit dem Kopf einer Klausel unifizierbar sein.



Variableninstanziierung: `X=sokrates`
Neue Anfrage: `?- menschlich(sokrates).`

Top-Down-Verfahren

- Das Head-Matching wird im top-down- Verfahren durchgeführt.
- Der Interpreter durchsucht die Datenbasis von oben nach unten, um passende Klauseln für einen Beweis zu finden.

```
mag_spinat(popeye).
hat_trainiert(garfield).
ist_stark(X) :- hat_trainiert(X).
ist_stark(X) :- mag_spinat(X).
```

Was ist die erste Antwort auf die Frage:

```
?- ist_stark(X).
```


Innerhalb von Regeln: left-to-right

- Regelrümpfe werden von links nach rechts bewiesen (left-to-right).
- Erst wenn ein Beweis für das i-te Prädikat im Rumpf gefunden ist, kann das i+1-te Prädikat bewiesen werden.

Beispiel:

```
schwester(X,Y):-
    fem(X),
    geschwister(X,Y).
```

Zunächst wird `fem(X)`, dann `geschwister(X,Y)` bewiesen.

Backtracking: depth-first

Backtracking kann durch zwei Ursachen ausgelöst werden:

- Der aktuelle Beweis ist in einer **Sackgasse**.
- Eine **alternative Lösung** soll berechnet werden.

In jedem Fall geht der Interpreter zur letzten Verzweigung im Beweisbaum zurück, an der noch Alternativen offen waren (depth-first).

```
mag_spinat(popeye).
hat_trainiert(garfield).
ist_stark(X) :- hat_trainiert(X).
ist_stark(X) :- mag_spinat(X).
```

Betrachte die folgenden Anfragen im Tracemodus:

```
?- ist_stark(popeye).
?- ist_stark(X).
```


Beweisführung: Zusammenfassung

- Die Beweisführung in Prolog erfolgt durch die Strategie der Tiefensuche.
- Teilziele einer Anfrage werden von links nach rechts bearbeitet.
- Für jedes Teilziel wird die erste Klausel (von oben nach unten) ausgewählt.
- Ist die Klausel ein Fakt, versucht Prolog die Anfrageklausel mit dem Fakt zu matchen. Gelingt dies, so ist das Teilziel bewiesen.
- Ist die Klausel eine Regel, versucht Prolog das Teilziel mit dem Regelkopf zu matchen. Gelingt dies, so versucht Prolog den Regelkörper zu beweisen (der Regelkörper ersetzt das Teilziel). Gelingt dies auch, so ist die Anfrageklausel bewiesen.
- Sollte die Beweisführung aufgrund einer unmöglichen Unifikation scheitern, springt Prolog zu dem letzten Punkt zurück, an dem eine Entscheidung getroffen wurde (Backtracking).
- Beim Backtracking werden die gemachten Variablenbindungen aufgehoben und nach einer alternativen Klausel gesucht.
- Findet Prolog keinen Fakt und keine Regel mit dem die Anfrage bewiesen werden kann, so wird `false` zurückgegeben.

Beispiel einer Beweissuche

- Beweissuche für die Anfrage: `hund(pluto).`

```
istehrlich(pluto).
hatvierbeine(pluto).
bellt(pluto).

hund(X) :-
    isteinsaeugetier(X),
    bellt(X).

isteinsaeugetier(X) :-
    hatvierbeine(X).

?- hund(pluto).
```

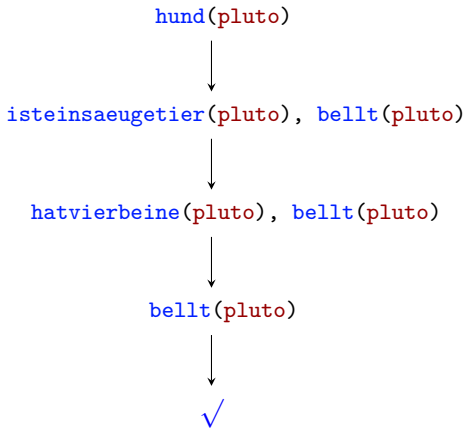

Suchbaum einer Beweissuche

```
istehrlich(pluto).
hatvierbeine(pluto).
bellt(pluto).

hund(X) :-
    isteinsaeugetier(X),
    bellt(X).

isteinsaeugetier(X) :-
    hatvierbeine(X).

?- hund(pluto).
```



Warum sagt man das Prolog die Strategie Tiefensuche anwendet?

Trace einer Beweissuche

```
istehrlich(pluto).
hatvierbeine(pluto).
bellt(pluto).

hund(X) :-
    isteinsaeugetier(X),
    bellt(X).

isteinsaeugetier(X) :-
    hatvierbeine(X).

?- hund(pluto).
```

```
[trace] 8 ?- hund(pluto).
   Call: (7) hund(pluto) ?
   Call: (8) isteinsaeugetier(pluto) ?
   Call: (9) hatvierbeine(pluto) ?
   Exit: (9) hatvierbeine(pluto) ?
   Exit: (8) isteinsaeugetier(pluto) ?
   Call: (8) bellt(pluto) ?
   Exit: (8) bellt(pluto) ?
   Exit: (7) hund(pluto) ?
true.
```


Beispiel einer Beweissuche (Entscheidungspunkt)

- Beweissuche für die Anfrage: `term(X).`

```
term1(a).
term1(b).

term2(a).
term2(b).

term3(b).

term(X) :- term1(X), term2(X), term3(X).

?- term(X).
```


Suchbaum einer Beweissuche (Entscheidungspunkt)

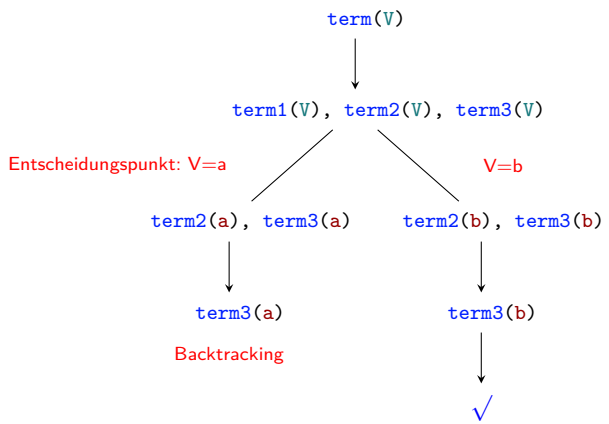
```
term1(a).
term1(b).

term2(a).
term2(b).

term3(b).

term(X) :-
    term1(X),
    term2(X),
    term3(X).

?- term(X).
```



▸ Übung 1 ▸ Übung 2

Beweissuche: Entscheidungspunkt

```
term1(a).
term1(b).

term2(a).
term2(b).

term3(b).

term(X) :-
    term1(X),
    term2(X),
    term3(X).

?- term(X).
```

```
[trace] 19 ?- term(X).
   Call: (7) term(_G11414) ?
   Call: (8) term1(_G11414) ?
   Exit: (8) term1(a) ?
   Call: (8) term2(a) ?
   Exit: (8) term2(a) ?
   Call: (8) term3(a) ?
   Fail: (8) term3(a) ?
   Redo: (8) term1(_G11414) ?
   Exit: (8) term1(b) ?
   Call: (8) term2(b) ?
   Exit: (8) term2(b) ?
   Call: (8) term3(b) ?
   Exit: (8) term3(b) ?
   Exit: (7) term(b) ?
X = b.
```

Zusammenfassung: Kapitel 2

Wir haben gelernt wie komplexe Strukturen durch Matching in Prolog aufgebaut werden können und wie die Beweisführung in Prolog funktioniert.

- **Keywords**: Beweisführung, Beweisstrategie (top-down, left-to-right, depth-first), Matching, Unifikation, Backtracking.
- **Wichtig**: Der Ablauf des Matchings und der Beweisführung (inkl. Backtracking) sind essentiell für die Programmierung in Prolog.
- **Ausblick Kapitel 3**: Rekursion

Übung: Matching

Welche der folgenden Paare von Termen matchen?

```
1  bread = bread
2  'Bread' = bread
3  'bread' = bread
4  Bread = bread
5  bread = sausage
6  food(bread) = bread
7  food(bread) = X
8  food(X) = food(bread)
9  food(bread,X) = food(Y,sausage)
10 food(bread,X,beer) = food(Y,sausage,X)
11 food(bread,X,beer) = food(Y,kahuna_burger)
12 food(X) = X
13 meal(food(bread),drink(beer)) = meal(X,Y)
14 meal(food(bread),X) = meal(X,drink(beer))
```

zurück

Übung: Die Negation des Unifikationsoperators

Welche der folgenden Anfragen führen zu `true`.?

```
?- a \= a.
?- 'a' \= a.
?- A \= a.
?- f(a) \= a.
?- f(a) \= A.
?- f(A) \= f(a).
?- g(a,B,c) \= g(A,b,C).
?- g(a,b,c) \= g(A,C).
?- f(X) \= X.
```

zurück

Übung: Beweis und Suchbaum

Gegeben ist folgende Wissensbasis:

```
house_elf(dobby).
witch(hermione).
witch(mcGonagall).
witch(rita_skeeter).
wizard(goofy).
magic(X):-house_elf(X).
magic(X):-wizard(X).
magic(X):-witch(X).
```

Welche der folgenden Anfragen lassen sich beweisen, wie werden eventuelle Variablen belegt?

```
1 ?- magic(house_elf).
2 ?- wizard(harry).
3 ?- magic(wizard).
4 ?- magic(mcGonagall).
5 ?- magic(Hermione).
```

Zeichne den Suchbaum für die 4. Anfrage. Gib alle Lösungen für die 5. Anfrage in der Reihenfolge an, in der sie Prolog ausgeben würde.

Bearbeiten sie auch die Aufgaben 2.3 und 2.4 aus “Learn Prolog Now!” (Übungssitzung).