Prolog

3. Kapitel: Rekursion

Dozentin: Wiebke Petersen

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

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

Einführung

- Ein wichtiges Konzept für das Lösen von Aufgaben bzw. für die Definition mächtiger Prädikate ist die **Rekursion**.
- Ein Prädikat ist **rekursiv definiert**, wenn in einer der definierenden Regeln das Prädikat im Regelkörper aufgerufen wird.
- Rekursion ist eine **Problemlösungsstrategie**. Die Grundidee ist das Zurückführen einer allgemeinen Aufgabe auf eine einfachere Aufgabe derselben Klasse (Schleifen).
- Rekursion ermöglicht es kompakte Prädikatsdefinitionen zu schreiben und Redundanz zu vermeiden.

rekursive Definitionen

Beispiel: natürliche Zahlen

- 0 ist eine natürliche Zahl. **(Ankerregel)**
- Wenn n eine natürliche Zahl ist, dann ist auch der Nachfolger von n (also $n+1$) eine natürliche Zahl. **(rekursive Regel)**
- Nichts sonst ist eine natürliche Zahl. **(Ausschlussregel)**

Beispiel: transitive Relation "Vorfahr"

A ist ein Vorfahr von B, wenn

- A ein Elternteil von B ist. **(Ankerregel)**
- wenn A ein Vorfahr von C und C ein Elternteil von B ist. **(rekursive Regel)**
- Sonst ist A kein Vorfahr von B. **(Ausschlussregel)**

rekursive Prädikate in Prolog

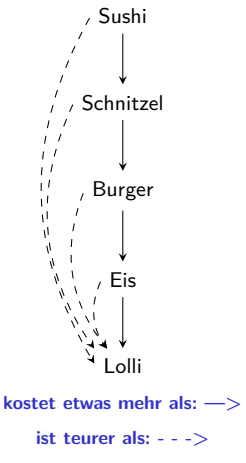
- Ein Prädikat ist **rekursiv definiert**, wenn in einer der definierenden Regeln das Prädikat im Regelkörper aufgerufen wird.
- Das Prädikat `teurer/2` ist rekursiv definiert.

```
kostet_etwas_mehr(eis,lolli).
kostet_etwas_mehr(burger,eis).
kostet_etwas_mehr(schnitzel,burger).
kostet_etwas_mehr(sushi,schnitzel).

teurer(X,Y):-
    kostet_etwas_mehr(X,Y).

teurer(X,Y):-
    kostet_etwas_mehr(X,Z),
    teurer(Z,Y).
```


Vorteile rekursiver Prädikate



Vorteile rekursiver Prädikate

```
kostet_etwas_mehr(eis,lolli).
kostet_etwas_mehr(burger,eis).
kostet_etwas_mehr(schnitzel,burger).
kostet_etwas_mehr(sushi,schnitzel).

% nichtrekursive Definition von teurer/2
teurer(X,Y):-
    kostet_etwas_mehr(X,Y).

teurer(X,Y):-
    kostet_etwas_mehr(X,A),
    kostet_etwas_mehr(A,Y).

teurer(X,Y):-
    kostet_etwas_mehr(X,A),
    kostet_etwas_mehr(A,B),
    kostet_etwas_mehr(B,Y).

teurer(X,Y):-
    kostet_etwas_mehr(X,A),
    kostet_etwas_mehr(A,B),
    kostet_etwas_mehr(B,C),
    kostet_etwas_mehr(C,Y).
```

```
kostet_etwas_mehr(eis,lolli).
kostet_etwas_mehr(burger,eis).
kostet_etwas_mehr(schnitzel,burger).
kostet_etwas_mehr(sushi,schnitzel).

% rekursive Definition von teurer/2
teurer(X,Y):-
    kostet_etwas_mehr(X,Y).

teurer(X,Y):-
    kostet_etwas_mehr(X,Z),
    teurer(Z,Y).
```


deklarative und prozedurale Bedeutung einer Wissensbasis

deklarative Bedeutung

- Unter der deklarativen Bedeutung versteht man die Bedeutung, die 'gemeint' oder die 'ausgedrückt' ist, wenn man die Wissensbasis als Menge logischer Aussagen liest.
- Die deklarative Bedeutung eines Prologprogramms kann extensional als die Menge aller Aussagen definiert werden, die sich **logisch** aus der Theorie der Wissensbasis (sprich Sammlung von logischen Aussagen) ableiten lassen.

prozedurale Bedeutung

- Unter der prozeduralen Bedeutung versteht man die Bedeutung, die sich daraus ergibt, was Prolog mit einer Wissensbasis 'tut'.
- Die Prozedurale Bedeutung eines Prologprogramms kann extensional als die Menge aller Anfragen (Aussagen) definiert werden, für die der Prolog-Interpreter eine Variablenbelegung findet, die zu der Ausgabe `true.` führt.

deklarative und prozedurale Bedeutung

```
es_regnet :- es_regnet.
es_regnet.
```

deklarative Bedeutung: Die Wissensbasis besteht aus zwei Aussagen: 'Wenn es regnet, dann regnet es.' und 'es regnet'. Aus diesen Aussagen lässt sich ableiten, dass es regnet.

prozedurale Bedeutung: Auf welche Aussagen wird Prolog mit '`true`.' antworten? Was passiert bei der Anfrage '`?- es_regnet.`'?

deklarative und prozedurale Bedeutung

- Das Ziel bei der Entwicklung von Prolog war eine deklarative Programmiersprache.
- Aber, die deklarative und die prozedurale Bedeutung eines Prologprogramms stimmen nicht immer überein.
- Trotzdem spricht man bei Prolog von einer deklarativen oder logischen Programmiersprache, da sie diesem Ziel nah gekommen ist. (Wer mehr zu dem Thema wissen will, warum es keine bessere Lösung gibt, sollte sich mit dem Problem der Unentscheidbarkeit der Prädikatenlogik befassen.)

Rekursive Prädikate prozedural

```
teurer(X,Y):-
    kostet_etwas_mehr(X,Y).

teurer(X,Y):-
    kostet_etwas_mehr(X,Z),
    teurer(Z,Y).
```

- Erste Regel: Wenn X ist teurer als Y bewiesen werden soll, reicht es X kostet etwas mehr als Y zu beweisen.
- Zweite Regel: Wenn X ist teurer als Y bewiesen werden soll, kann dieses Problem in zwei Teilprobleme zerlegt werden. Gesucht ist ein Z, so dass X etwas mehr kostet als Z (Teilproblem 1) und dass Z teurer ist als Y (Teilproblem 2).

▸ Übung

prozedural $\neq$ deklarativ

Konsequenz für die Prologprogrammierung:

- zunächst sollte man immer das Problem beschreiben (deklarativ),
- anschließend muss man sich Gedanken über die Arbeitsweise (prozedural) des Prolog-Interpreters machen und das Programm gegebenenfalls anpassen.

Definieren harmloser rekursiver Prädikate:

- Rekursive Prädikate benötigen immer mindestens zwei Klausel: rekursive Klausel plus Anker- oder Ausstiegsklausel.
- Die Ankerklausel sollte immer vor der rekursiven Klausel stehen (sonst droht eine Endlosschleife).
- Im Regelkörper der rekursiven Klausel ist es oft sinnvoll, den rekursive Aufruf ans Ende zu setzen.

Beispiel: Definition natürlicher Zahlen

Natürliche Zahlen

- 0 ist eine natürliche Zahl. **(Ankerregel)**
- Wenn n eine natürliche Zahl ist, dann ist auch der Nachfolger von n eine natürliche Zahl. **(rekursive Regel)**
- Nichts sonst ist eine natürliche Zahl. **(Ausschlussregel)**

Wir verwenden `succ/1` zur Kodierung natürlicher Zahlen:

```
0 => 0
1 => succ(0)
2 => succ(succ(0))
3 => succ(succ(succ(0)))
...
```

Ziel: Ein Prädikat `numeral/1`, welches überprüft ob das Argument eine natürliche Zahl in der `succ`-Darstellung ist.

Beispiel: Definition natürlicher Zahlen

Natürliche Zahlen

- 0 ist eine natürliche Zahl. **(Ankerregel)**
- Wenn n eine natürliche Zahl ist, dann ist auch der Nachfolger von n eine natürliche Zahl. **(rekursive Regel)**
- Nichts sonst ist eine natürliche Zahl. **(Ausschlussregel)**

Ziel: Ein Prädikat `numeral/1`, welches überprüft ob das Argument eine Zahl in der `succ`-Darstellung ist.

```
% Ankerklausel: 0 ist eine Zahl
numeral(0).

% rekursive Klausel: Der Nachfolger einer Zahl ist eine Zahl
numeral(succ(X)) :- numeral(X).
```


Beispiel: Definition natürlicher Zahlen

Das Programm

```
numeral(0).
numeral(succ(X)) :- numeral(X).
```

kann zur Generierung von Zahlen genutzt werden:

```
?- numeral(X).
X = 0 ;
X = succ(0) ;
X = succ(succ(0)) ;
X = succ(succ(succ(0))) ;
X = succ(succ(succ(succ(0)))) ;
X = succ(succ(succ(succ(succ(0))))) ;
X = succ(succ(succ(succ(succ(succ(0)))))) ;
X = succ(succ(succ(succ(succ(succ(succ(0))))))) ;
X = succ(succ(succ(succ(succ(succ(succ(succ(0)))))))) ;
X = succ(succ(succ(succ(succ(succ(succ(succ(succ(0))))))))) ;
...
```


Beispiel: Addition natürlicher Zahlen

Ziel: Ein Prädikat `add/3`, welches drei Zahlen in der `succ/0`-Darstellung als Argument nimmt.

Das dritte Argument soll die Summe der beiden ersten sein.

```
?- add(succ(0), succ(succ(0)), X).
X = succ(succ(succ(0))).


?- add(succ(succ(0)), succ(0), X).
X = succ(succ(succ(0))).

?- add(0, succ(succ(0)), X).
X = succ(succ(0)).
```

Beispiel: Addition natürlicher Zahlen

Ziel: Ein Prädikat `add/3`, welches drei Zahlen in der `succ`-Darstellung als Argument nimmt.

Das dritte Argument soll die Summe der beiden ersten sein.

- **Ankerklausel:** Wenn das erste Argument `0` ist, dann ist das zweite Argument gleich dem dritten Argument.
- **Rekursive Klausel:** Wenn die Summe von `X` und `Y` gleich `Z` ist, dann ist die Summe von `succ(X)` und `Y` gleich `succ(Z)`.

```
% Ankerklausel
add(0,Y,Y).

% rekursive Klausel
add(succ(X),Y,succ(Z)):-
    add(X,Y,Z).
```


Beispiel: Addition natürlicher Zahlen

```
% Ankerklausel
add(0,Y,Y).

% rekursive Klausel
add(succ(X),Y,succ(Z)):-
    add(X,Y,Z).
```

Was geschieht bei folgenden Anfragen?

```
?- add(succ(succ(0)) , succ(succ(succ(0))) , Z).
?- add(X,succ(succ(0)) , succ(succ(succ(0)))).
?- add(succ(succ(0)) , Y , succ(succ(succ(0)))).
?- add(X , Y , succ(succ(succ(0)))).
?- add(X , succ(succ(succ(0))) , Z).
?- add(succ(succ(succ(0))) , Y , Z).
?- add(X , Y , Z).
```

▸ Übung: add ▸ Übung: greater_than

Zurück zur prozeduralen und deklarativen Bedeutung

Zur Erinnerung: Bei der Beweisführung arbeitet sich Prolog

- durch die Wissensbasis von oben nach unten,
- innerhalb der einzelnen Klauseln von links nach rechts durch die Teilziele.

Die Reihenfolge der Klauseln und der Teilziele innerhalb der Klauseln beeinflusst ihre prozedurale Verarbeitung!

```
et(albert,kevin).
et(lena,albert).
et(marie,lena).

vorfahr1(X,Y):- et(X,Y).
vorfahr1(X,Z):-
    et(X,Y),
    vorfahr1(Y,Z).

vorfahr2(X,Z):-
    et(X,Y),
    vorfahr2(Y,Z).
vorfahr2(X,Y):- et(X,Y).
```

```
vorfahr3(X,Y):- et(X,Y).
vorfahr3(X,Z):-
    vorfahr3(Y,Z),
    et(X,Y).

vorfahr4(X,Z):-
    vorfahr4(Y,Z),
    et(X,Y).
vorfahr4(X,Y):- et(X,Y).
```

Wie beeinflusst die Reihenfolge das prozedurale Verhalten der Prädikatsdefinitionen? ▸ Übung

Wiederholung: rekursive Prädikate

Ziele: Rekursive Prädikate,

- die nicht zu Endlosschleifen führen,
- die möglichst früh terminieren,
- die mit offenen Variablen aufgerufen werden können.

Definieren harmloser rekursiver Prädikate:

- Rekursive Prädikate benötigen immer mindestens zwei Klausel: rekursive Klausel plus Anker- oder Ausstiegslkausel.
- Die Ankerklausel sollte immer vor der rekursiven Klausel stehen (sonst droht Endlosschleife).
- Im Regelkörper der rekursiven Klausel ist es oft sinnvoll, den rekursive Aufruf ans Ende zu setzen.

Zusammenfassung Kapitel 3

- Wir haben gelernt, dass die Rekursion eine essentielle Programmiertechnik in Prolog ist.
- Wir wissen, dass die Rekursion uns das Schreiben von kompakten und präzisen Programmen ermöglicht.
- Wichtig ist die deklarative sowie prozedurale Bedeutung einer Wissensbasis zu verstehen.

- **Keywords**: Rekursion, Problemlösungsstrategie, deklarative / prozedurale Bedeutung.
- **Wichtig**: Die Rekursion ist ein äußerst wichtiges Grundkonzept in Prolog.
- **Ausblick Kapitel 4**: Listen

Übung: rekursive Prädikate

- Wie lauten die Antworten auf die Anfragen? In welcher Reihenfolge werden die Ergebnisse für die letzte Anfrage ausgegeben?

```
verdaut(X,Y) :- hatgegessen(X,Y).
verdaut(X,Y) :- hatgegessen(X,Z),
                verdaut(Z,Y).

hatgegessen(moskito,blut(john)).
hatgegessen(frosch,moskito).
hatgegessen(storch,frosch).
```

```
1 ?- verdaut(storch,frosch).
2 ?- verdaut(storch,moskito).
3 ?- verdaut(frosch,X).
4 ?- verdaut(X,Y).
```

zurück

Übung: greater_than

Definieren Sie ein Prädikat `greater_than/2`, das zwei natürliche Zahlen in der `succ/1`-Notation nimmt und überprüft, ob die erste Zahl größer ist als die zweite:

```
?- greater_than(succ(succ(succ(0))),succ(0)).
true.
?- greater_than(succ(succ(0)),succ(succ(succ(0)))).
false.
```

zurück

Übung: Vorfahr

Betrachten Sie die folgende Definitionsvariante für das Prädikat `vorfahr/2`. Welche Probleme ergeben sich für diese Variante?

```
vorfahr5(X,Y):-
    et(X,Y).

vorfahr5(X,Y):-
    vorfahr5(X,Z),
    vorfahr5(Z,Y).
```

zurück

Bearbeiten sie die Aufgabe 3.3 auf der “Learn Prolog Now!” Seite (Übungssitzung).

Übung: Addition

Bei der derzeitigen Definition des Prädikats add/3 erhalten Sie auf manche Anfragen mit mehr als einer Variablen konkrete Zahlen als Antworten, für andere erhalten Sie lediglich eine Angabe über die Beziehungen, die zwischen den Variablenbelegungen herrschen müssen:

```
% keine konkrete Zahl als Antwort
?- add(succ(0),Y,Z).
Z = succ(Y).

% konkrete Zahlen als Antwort
?- add(X,succ(0),Z).
X = 0,
Z = succ(0) ;
X = succ(0),
Z = succ(succ(0)) ;
X = succ(succ(0)),
Z = succ(succ(succ(0))) ;
...
```

- Können Sie die Definition von add/3 so anpassen, dass Sie immer konkrete Zahlen als Antwort erhalten?

zurück

Bearbeiten sie auch die Aufgabe 3.5 sowie die Aufgaben der ‘Practical Session’ zu Kapitel 3 aus “Learn Prolog Now!” (Übungssitzung).