

# Prolog

## 4. Kapitel: Listen

Dozentin: Wiebke Petersen

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

### Listen in Prolog

- Listen sind sehr mächtige Datenstrukturen in Prolog.
- Listen sind endliche Sequenzen von Elementen:

```
% Liste mit atomaren Elementen:
[mia, vincent, jules, mia]
% Liste mit verschiedenen Termen als Elemente:
[mia, 2, mother(jules), X, 1.7]
% leere Liste:
[]
% Listenelemente koennen Listen sein:
[mia, [[3,4,paul], mother(jules)], X]
```

- Listen können **verschachtelt** sein (Listen können Listen als Elemente haben)
- Die Reihenfolge der Elemente ist wichtig  $[a,b,c] \neq [b,a,c]$  (Unterschied zu Mengen).
- Dasselbe Element kann mehrfach in einer Liste vorkommen (Unterschied zu Mengen).

### Zusammenfassung Kapitel 3

- Wir haben gelernt, dass die Rekursion eine essentielle Programmieretechnik 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

### Unifikation / Matching von Listen

Zwei Listen sind unifizierbar,

- wenn sie dieselbe Länge haben und
- wenn die korrespondierenden Listenelemente unifizierbar sind.

```
?- [a,b,X]=[Y,b,3].
X = 3,
Y = a
?- [[a,b,c],b,3]=[Y,b,3].
Y = [a, b, c]
?- [a,b,c] = X. % Variablen koennen mit Listen unifiziert werden.
X=[a,b,c]
?- [a,b,X,c]=[Y,b,3].
false.
?- [a,c,3]=[Y,b,3].
false.
```

- Die **Länge** einer Lister ist die Zahl ihrer Elemente.

## Listenerlegung in Prolog

- Prolog hat einen eingebauten Operator '!' (**Listenkonstruktor**) mit dem man eine Liste in **Kopf** (*head*) und **Rest** (*tail*) zerlegen kann.
- Der **Kopf** ist das erste Element der Liste.
- Der **Rest** ist die Restliste.

```
?- [Head|Tail] = [mia, vincent, jules, mia].
Head = mia,
Tail = [vincent, jules, mia].
```

- Eine leere Liste hat keinen Head und lässt sich somit nicht zerlegen:

```
?- [Head|Tail] = [].
false.
```

- Man kann mit '!' auch mehr als ein Element am Anfang abtrennen:

```
?- [First,Second|Tail] = [mia, vincent, jules, mia].
First = mia,
Second = vincent,
Tail = [jules, mia].
```

► Übung: Syntax von Listen

► Übung: Matching von Listen

## Anonyme Variable

- Die Variable '\_' ist die anonyme Variable in Prolog.
- Sie kommt immer dann zum Einsatz, wenn ein Wert zwar variabel sein soll, später aber nicht mehr benötigt wird.
- Die anonyme Variabel erhöht die Lesbarkeit der Programme.
- Anders als bei anderen Variablen ist jedes Vorkommen der anonymen Variabel unabhängig von den anderen. Sie kann also immer wieder anders initialisiert werden:

```
isst_gerne(X,X) = isst_gerne(a,b).
false.

isst_gerne(_,_) = isst_gerne(a,b).
true.
```

Hinweis: Variablen wie `_X`, die mit einem Unterstrich beginnen sind nicht anonym, sie führen aber im Gegensatz zu anderen Variablen beim Konsultieren einer Wissensbasis nicht zu der Warnung: „`singleton variables`:“.

## Anonyme Variable

Beispielproblem: Wir wollen das 2. und 4. Element einer Liste herausgreifen:

```
% ohne anonyme Variabel erhaelt man Werte fuer alle
% Variablen der Anfrage.
?- [X1,X2,X3,X4|T] = [mia, vincent, jules, mia, otto, lena].
X1 = mia,
X2 = vincent,
X3 = jules,
X4 = mia,
T = [otto,lena].

% mit anonymer Variable nur die gesuchten.
?- [_,X2,_,X4|_] = [mia, vincent, jules, mia, otto, lena].
X2 = vincent,
X4 = mia.
```

► Übung

## Prädikat: member/2

- `member/2` ist ein rekursiv definiertes Prädikat, das überprüft, ob ein Element (ein Term) in einer Liste vorkommt:

```
% member/2, member(Term,List)
member(X, [X|_]).
member(X, [_|T]) :-
    member(X,T).
```

- Wie ist das Programm zu verstehen?
- Der Fakt `member(X, [X|_])` besagt, dass etwas ein Element einer Liste ist, wenn es das erste Element (der Head) der Liste ist.
- Die Regel `member(X, [_|T]) :- member(X,T)` besagt, dass etwas ein Element einer Liste ist, wenn es ein Element der Restliste (des Tails) ist.
- Jedes Element einer Liste ist entweder erstes Element oder ein Element im Tail.
- Vorsicht: `member/2` ist ein in der Library `lists` vordefiniertes Prädikat, das von manchen Prologimplementierungen automatisch geladen wird. Verwenden sie daher besser einen anderen Namen, z.B. `my_member/2`.





- Wir haben Listen als mächtige Datenstrukturen in Prolog kennengelernt und mithilfe des Listenkonstruktors `|` dekonstruiert.
- Wir haben gelernt, Prädikate zu definieren, die Listen rekursiv verarbeiten und das wichtige Prädikat `member/2` kennengelernt.
- Wir haben die anonyme Variable `_` kennengelernt.
- **Keywords:** Listenkonstruktor, Kopf (Head), Restliste (Tail), rekursive Listenverarbeitung, `member/2`, anonyme Variable.
- **Wichtig:** Die rekursive Verarbeitung von Listen ist eine zentrale Programmier-technik in Prolog.
- **Ausblick Kapitel 5:** Arithmetik

- Welche der folgenden Ausdrücke sind syntaktisch wohlgeformte Listen in Prolog?
- Wie lang sind die Listen?

```

1 [1|[2,3,4]]
2 [1,2,3|[]]
3 [1|2,3,4]
4 [1|[2|[3|[4]]]]
5 [1,2,3,4|[]]
6 [[]|[]]
7 [[1,2]|4]
8 [[1,2],[3,4]|[5,6,7]]

```

▸ zurück

Was antwortet Prolog auf die folgenden Anfragen?

```

1 ?- [a,b,c,d] = [a,[b,c,d]].
2 ?- [a,b,c,d] = [a|[b,c,d]].
3 ?- [a,b,c,d] = [a,b,[c,d]].
4 ?- [a,b,c,d] = [a,b|[c,d]].
5 ?- [a,b,c,d] = [a,b,c,[d]].
6 ?- [a,b,c,d] = [a,b,c|[d]].
7 ?- [a,b,c,d] = [a,b,c,d,[]].
8 ?- [a,b,c,d] = [a,b,c,d|[]].
9 ?- [a,b,c,d] = [a,b,X].
10 ?- [a,b,c,d] = [a,b|X].
11 ?- [a,b,c,d] = [a,b,[c,d]].
12 ?- [a,b,c,d] = [a|[b|[c,d]]].
13 ?- [[die,Y]|Z]=[X,katze],[ist,weg]].
14 ?- [a|B]=[A|b]. % Vorsicht: ?- is_list([a|b]). liefert 'false'.
15 ?- [anna,X]=[Y|[maria]].

```

▸ zurück

Was antwortet Prolog auf die folgenden Anfragen?

```

1 ?- [] = __.
2 ?- [] = [ _ ].
3 ?- [] = [ _ | [ ] ].
4 ?- [ _ ] = [ _ | [ ] ].
5 ?- [ _ , X , _ , Y | _ ] = [dead(zed), [2, [b, chopper]], [], []].
6 ?- [ _ , X , _ , Y | _ ] = [dead(zed), [2, [b, chopper]], [], []].
7 ?- [ _ , _ , [ _ | X ] | _ ] = [ [], dead(zed), [2, [b, chopper]], [], Z, [2, [b, chopper]] ].

```

▸ zurück

## Übung: einfache Listenprädikate

- Schreiben sie ein Prädikat `third/2`, das gelingt, wenn das zweite Argument eine Liste ist und das erste Argument das dritte Element dieser Liste ist.

```
?- third(a, [b,c,a,d,e]).
true.
```

- Schreiben sie ein Prädikat `tausch12/2`, das zwei Listen als Argumente nimmt und gelingt, wenn sich die beiden Listen nur in der Reihenfolge der ersten beiden Elemente unterscheiden.

```
?- tausch([a,b,c,d], [b,a,c,d]).
true.
```

## Übung: member

Zeichnen sie die Suchbäume zu den folgenden Anfragen:

```
?- member(b, [c,b,a,y]).
?- member(x, [a,b,c]).
?- member(X, [a,b,c]).
```

▶ zurück

## Übung: auf `member/2` beruhende Prädikate

Für die Definition der folgenden Prädikate kann `member/2` verwendet werden.

- Schreibe ein Prädikat `all_members/2`, das zwei Listen L1 und L2 nimmt und gelingt, wenn alle Elemente von L1 auch Element von L2 sind.

```
?- all_members([a,c], [a,b,c,d]).
true.
?- all_members([a,e], [a,b,c,d]).
false.
?- all_members(a, [a,b,c,d]).
false.
```

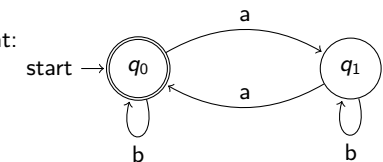
- Schreibe ein Prädikat `set_equal/2`, das zwei Listen L1 und L2 nimmt und gelingt, wenn die beiden Listen als Mengen betrachtet gleich sind (also die gleichen Elemente haben).

```
?- set_equal([a,b,a], [b,b,b,a]).
true.
?- set_equal([a,b,c], [b,b,b,a]).
false.
?- set_equal([a,b], [c,a,b]).
false.
```

▶ zurück

## Übung: endliche Automaten

Gegeben sei der folgende endliche Automat:



Dieser Automat akzeptiert die Sprache aller Sequenzen über dem Alphabet  $\{a, b\}$ , die eine gerade Anzahl von `a`'s beinhalten.

Repräsentieren Sie diesen Automaten in Prolog und schreiben Sie ein Prädikat `fsa_accept/1`, das als Argument eine Liste nimmt und gelingt, wenn die Liste eine von dem Automaten akzeptierte Sequenz ist.

```
?- fsa_accept([a,b,a,a,b,b,a]).
true.
?- fsa_accept([a,a,c]).
false.
?- fsa_accept([a,b]).
false.
```



## Einige Lösungen von Foliensatz 2

### Grammatikaufgabe:

```
% Erarbeitet mit Cora Scholl
% Regelsystem:
s(s(NP,VP)) :- np(NP,NUM), vp(VP,NUM).
np(np(A,N),NUM) :- artikel(A,NUM), nomen(N,NUM).
vp(vp(V,NP),NUM) :- verb(V,NUM), np(NP,_NUM2).
vp(vp(V),NUM) :- verb(V,NUM).
```

```
% Lexikon:
nomen(katze,sg).
nomen(katzen,pl).
nomen(maus,sg).
nomen(maeuse,pl).
artikel(die,sg).
artikel(die,pl).
artikel(eine,sg).
artikel(viele,pl).
verb(klaut,sg).
verb(klauen,pl).
verb(jagt,sg).
verb(jagen,pl).
```

### Ecken des Rechtecks:

```
% Ecke oben links:
?- horizontal(line(point(6,3),OL)),
   vertical(line(point(1,1),OL)).
% Ecke unten rechts:
?- horizontal(line(point(1,1),UR)),
   vertical(line(point(6,3),UR)).
% beide Ecken in einer Anfrage:
?- horizontal(line(point(6,3),OL)),
   vertical(line(point(1,1),OL)),
   horizontal(line(point(1,1),UR)),
   vertical(line(point(6,3),UR)).
```