Einführung in PROLOG V

Listen 2

Wir definieren ein Prädikat **append/3**, welches zwei Listen verbindet, indem es sei einfach aneinanderhängt:

```
append( [], L, L).
append( [ K | R ], L2, [ K | L3 ]) :- append( R, L2, L3).
```

Die Query **?- append([a,b,c],[1,2,3,4],X)** ergibt also:
X = [a,b,c,1,2,3,4]

Was geschieht nun im Detail?
Unsere Abbruchbedingung übergibt diesmal nicht die vollständige Lösung an die Variable, sondern initiiert erst die Konstruktion:

```
(1) 1 CALL  append([a, b, c], [1, 2, 3, 4], X)
  (2) 2 CALL  append([b, c], [1, 2, 3, 4], L3)
    (3) 3 CALL  append([c], [1, 2, 3, 4], L3)
      (4) 4 CALL  append([], [1, 2, 3, 4], L3)
      (4) 4 EXIT  append([], [1, 2, 3, 4], [1, 2, 3, 4])
    (3) 3 EXIT  append([c], [1, 2, 3, 4], [c, 1, 2, 3, 4])
  (2) 2 EXIT  append([b, c], [1, 2, 3, 4], [b, c, 1, 2, 3, 4])
(1) 1 EXIT  append([a, b, c], [1, 2, 3, 4], [a, b, c, 1, 2, 3, 4])
```

Das Verwenden von Matching zur Konstruktion einer Struktur ist ein häufiges Vorgehen bei rekursiven Listenoperationen.

Das Prädikat **append/3** ist übrigens bereits in PROLOG eingebaut.

append verwenden

Wir können nun **append** verwenden um Listen aufzuspalten. Zum Beispiel:

```
append(X, Y, [a, b, c, d]).

X=[ ]
Y=[a, b, c, d];

X=[a ]
Y=[ b, c, d];

X=[a, b ]
Y=[ c, d];

X=[a, b, c ]
Y=[ d];

X=[a, b, c, d ]
Y=[ ];
```

Indem PROLOG hier durch matchen versucht, zwei Listen zu bestimmt, die zusammen die zu teilende Liste ergeben, findet PROLOG alle Möglichkeiten die Liste aufzuteilen.
Daher lassen sich mit **append** weitere nützliche Prädikate definieren. Wir können z.B. die Prefixe einer Liste finden lassen:

prefix(P, L) :- append(P, _, L).

prefix(X, [a, b, c]).

X=[ ];
X=[a];
X=[a, b];
X=[a, b, c];

Oder auch die Suffixe:

suffix(S, L) :- append(_, S, L).

Wie wäre die Ausgabe von: **?- suffix(S, [a, b, c]).**

Damit ist es nun einfach ein Prädikat zu schreiben, dass Teillisten einer Liste findet, denn die Teillisten einer Liste L sind die Prefixe von Sufixen von L.

Damit ergibt sich also:

sublist(SubL, L) :- suffix(S, L), prefix(SubL, S).

Listen umdrehen 1

Append ist ein sehr nützliches Prädikat, kann aber auch sehr ineffizient sein. Wenn man betrachtet, wie es arbeitet, fällt auf, dass es nicht zwei Listen in einer simplen Aktion verbindet, sondern sich erst zum ersten Argument hinunterarbeiten muss.

Schauen wir uns diese rekursive Definition an, wie man eine reverse Liste erstellt:

1. Die reverse Liste der leeren Liste ist wieder die leere Liste.
2. Die reverse Liste von [K | R] erhalten wir, wenn wir an die reverse Liste von R hinten K anfügen

Das führt zu folgendem Programm:

```
naiverev([ ], [ ]).
naiverev([K | R], T):-
                naiverev(R RevR),
                append(RevR, [K], T).
```

Dieses Programm funktioniert, aber es ist sehr aufwendig

Listen umdrehen 1

Es ist besser Akkumulatoren zu verwenden, indem wir den Kopf der Liste in die Akkumualtorenliste einfügen. Danach gehen wir durch die Liste, entnehmen jeweils den Kopf und fügen ihn vorne an die Akkumulatorenliste an. Wenn wir die Liste abgearbeitet haben, entspricht die Akkumulatorliste der reversen Liste.

```
accRev([], A, A).
accRev([K | R], A, Rev):-accRev(R, [K|A], Rev).
```

Nun definieren wir noch ein aufrufendes Prädikat:

```
rev(L, Rev):- accRev(L, [], Rev).
```

und vergleichen unsere beiden Prädikate im Tracer.
Ist es notwendig in diesem Fall, den Abbruch vor der rekursiven Definition zu haben?

Übung

Übung 5.1

Wir nennen eine Liste *gedoppelt*, wenn sie aus zwei absoult gleichen Blöcken besteht. [a,b,c,a,b,c] ist gedoppelt (besteht aus [a,b,c] gefolgt von [a,b,c]) ebenso [foo,gubble,foo,gubble]. Allerdings ist [foo,gubble,foo] nicht gedoppelt.
Schreiben Sie ein Prädikat doubled(List) , welches testet, ob eine Liste gedoppelt ist.

Übung 5.2

Ein Palindrom ist ein Wort oder ein Satz, der von hinten, wie von vorne gleich buchstabiert wird. Z.B. `radar`, `Regal-Lager`, `rotator', `eve', und `nurses run' sind alle Palindrome.
Schreiben Sie das Prädikat palindrome(List), welches testet, ob List ein Palindrom ist:
?- palindrome([r,o,t,a,t,o,r]).
sollte als ein yes ergeben.

Übung 5.3

Noch ein logisches Rätsel:
In einer Strasse stehen drei Häuser nebeneinander, die alle unterschidliche Farben haben: Rot, blau und grün.
In den Häusern leben Personen unterschiedlicher Nationalität, die unterschiedliche Haustiere haben.:
Der Engländer lebt im roten Haus
Der Jaguar ist das Haustier der Spanier.
Der Japaner lebt rechts von demjenigen, der eine Schlange hat.
Der Schlangenbesitzer lebt links vom blauen Haus.
Wer besitzt ein Zebra?

Definieren Sie ein Prädikat zebra/1, das Ihnen die Nationalität des Zebrabesitzers offenbart.
Hinweis:
Überlegen Sie sich eine Repräsentation für die Hüser und die Strassen. Kodieren Sie die vier Bedingungen in PROLOG. member und sublist könnten hilfreich sein..