Logikprogrammierung

- ▷ Programme und Datenstrukturen
- ▷ PEANO-Arithmetik
- ▷ Arithmetische Ausdrücke
- ▷ Listennotation
- ▷ Listenverarbeitung
- ▷ Beispiele

Programme und Datenstrukturen

Syntax

⟨Datenbasis⟩ ::= ⟨Klausel⟩ ... ⟨Klausel⟩
⟨Klausel⟩ ::= ⟨Kopf⟩ ':-' ⟨Körper⟩ '.' | ⟨Kopf⟩ '.'
⟨Kopf⟩ ::= ⟨Literal⟩
⟨Körper⟩ ::= ⟨Ziel⟩
⟨Ziel⟩ ::= ⟨Ziel⟩ ',' ⟨Ziel⟩ | ⟨Ziel⟩ ';' ⟨Ziel⟩ | ⟨Literal⟩
⟨Literal⟩ ::= ⟨Atom⟩ | ⟨komplexer Term⟩
⟨komplexer Term⟩ ::= ⟨Prädikatsname⟩'('⟨Term⟩',' ... ','⟨Term⟩')'
    | ⟨Operatorausdruck⟩ | ⟨Liste⟩
⟨Term⟩ ::= ⟨Konstante⟩ | ⟨Variable⟩ | ⟨komplexer Term⟩
⟨Konstante⟩ ::= ⟨Atom⟩ | ⟨Zahl⟩
⟨Variable⟩ ::= ⟨benannte Variable⟩ | ⟨unbenannte Variable⟩

Programme und Datenstrukturen

Operatoren: übersichtliche Schreibweise, Nähe zur mathematischen Notation

> Operatoren definieren Termstrukturen:
> `A op B` $\Leftrightarrow$ `op(A,B)`

Operatorausdrücke sind nur eine syntaktische Notationsvariante für eine Termstruktur

Operatortypen:

Präfixoperatoren	`fx`	`lg 10`
Infixoperatoren	`xfy`	`3 + 4`
Postfixoperatoren	`xf`	`4!`

Klauselkopf-Separator ':-'

Konjunktion ','

arithmetische Ordnungsprädikate `< > =< >= =:= =\=`

arithmetische Operatoren `+ - * /`

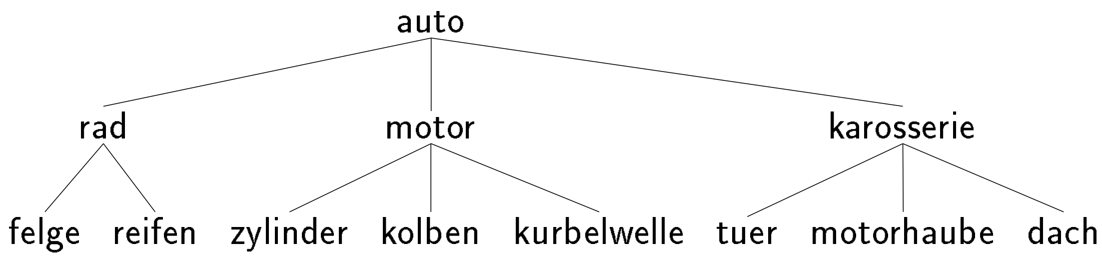
Programme und Datenstrukturen

Terme als komplexe Datenstrukturen:

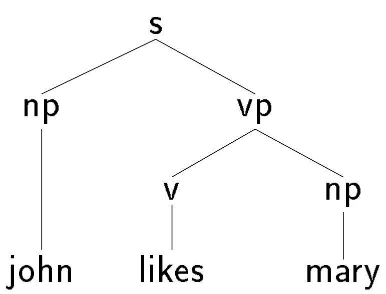
rekursiv eingebettete Terme entsprechen Baumstrukturen

Terme	Verzweigungen des Baumes (Knoten)
Prädikatsnamen	Knotenmarkierungen
äußerster Prädikatsname	Wurzelknoten des Baumes
Argumente	Kanten
Atome als Argument	Blattknoten

```
auto(rad(felge,reifen),
     motor(zylinder,kolben,kurbelwelle),
     karosserie(tuer,motorhaube,dach)).
```




```
s(np(john),
  vp(v(likes),
     np(mary))).
```



Programme und Datenstrukturen

Unifikation (3): Ermitteln einer solchen Variablensubstitution, die die Gleichheit von *zwei Termen* herstellt

- ☐ identischer Prädikatsname
- ☐ gleiche Stelligkeit
- ☐ *rekursive* Unifikation der Argumente

Term / Term	Unifikation	`a(X,c) = a(b,Y)`
Variable / Term	Instanziierung	`X = a(b,c)`
Term / Variable	Instanziierung	`d(e,f) = Y`
Variable / Variable	Sharing	`X = Y`

Beispiele

Term 1	Term 2	Variablensubstitution
`a(X,c)`	`a(b,Y)`	`X=b, Y=c`
`a(X,X)`	`a(d(Y),d(b))`	`X=d(b), Y=b`
`a(X,a(X))`	`a(Y,Y)`	`X=Y=a(X)`

Problemfall: Sharing über mehrere Rekursionsebenen hinweg
Aufbau unendlicher Terme durch Unifikation endlicher Terme

Aufruf von Daten als Programm: `call(Term)`

PEANO-Arithmetik

Axiomatisierung der Arithmetik für natürliche Zahlen

- Anfangselement: 0
- Nachfolgerbeziehung `s(Peano_Term)`

Term	arithmetische Interpretation
0	0
s(0)	1
s(s(0))	2
s(s(s(0)))	3
...	...

Addition `add(Summand1,Summand2,Summe)`

Rekursive Definition

```
add(X,0,X).
add(X,s(Y),R):-add(s(X),Y,R).
```

Rekursiver Abstieg über das zweite Argument

PEANO-Arithmetik

Unterspezifizierter Aufruf I

`add(in,in,out)`, z.B. 1+2=Sum

```
?- add(s(0),s(s(0)),Sum).                #1
   add(X1,0,X1).                         fail
   add(X1,s(Y1),R1)                      succ(X1=s(0),
        :-add(s(X1),Y1,R1).              Y1=s(0),R1=Sum)
   ?- add(s(s(0)),s(0),R1).              #2
      add(X2,0,X2).                      fail
      add(X2,s(Y2),R2)                   succ(X2=s(s(0)),
           :-add(s(X2),Y2,R2).           Y2=0,R2=R1=Sum)
      ?- add(s(s(s(0))),0,R2).           #3
         add(X3,0,X3).                   succ(X3=s(s(s(0)))
                                              =R2=R1=Sum)
Sum=s(s(s(0))) ;
         add(X3,s(Y3),R3)                fail
              :-add(s(X3),Y3,R3).
      BT #2
   BT #1
no.
```

PEANO-Arithmetik

Unterspezifizierter Aufruf II

`add(in,out,in),` z.B. `1+Smd=3`

```
?- add(s(0),Smd,s(s(s(0)))).            #1
   add(X1,0,X1).                        fail
   add(X1,s(Y1),R1)                     succ(X1=s(0),
        :-add(s(X1),Y1,R1).             s(Y1)=Smd,R1=s(s(s(0))))
   ?- add(s(s(0)),Y1,s(s(s(0)))).       #2
      add(X2,0,X2).                     fail
      add(X2,s(Y2),R2)                  succ(X2=s(s(0)),
           :-add(s(X2),Y2,R2).          s(Y2)=Y1,
                                        R2=s(s(s(0))))
      ?- add(s(s(s(0))),                #3
               Y2,s(s(s(0)))).
         add(X3,0,X3).                  succ(X3=s(s(s(0))),Y2=0)
Smd=s(s(0)) ;
         add(X3,s(Y3),R3)               succ(X3=s(s(s(0))),
              :-add(s(X3),Y3,R3).       s(Y3)=Y2,R3=s(s(s(0))))
         ?- add(s(s(s(s(0)))),          #4
                  Y3,s(s(s(0)))).
            add(X4,0,X4).               fail
            add(X4,s(Y4),R4)            succ
                 :-add(s(X4),Y4,R4).
            . . .
```

Welche natürliche (!) Zahl muß ich zu 4, 5, ... addieren, damit ich 3 erhalte?!

PEANO-Arithmetik

unterspezifizierter Aufruf III

`add(out,in,in)`, z.B. `Smd+2=3`

```
?- add(Smd,s(s(0)),s(s(s(0)))).        #1
   add(X1,0,X1).                       fail
   add(X1,s(Y1),R1)                    succ(Smd=X1,Y1=s(0),
        :-add(s(X1),Y1,R1).            R=s(s(s(0))))
   ?- add(s(X1),s(0),s(s(s(0)))).      #2
      add(X2,0,X2).                    fail
      add(X2,s(Y2),R2)                 succ(X2=s(X),
           :-add(s(X2),Y2,R2).         Y2=0,R2=s(s(s(0))))
      ?- add(s(s(X2)),0,s(s(s(0)))). #3
         add(X3,0,X3).                 succ(X3=s(s(X2))
                                       =s(s(s(0))))
Smd=s(0) ;
           add(X3,s(Y3),R3)              fail
              :-add(s(X3),Y,R3).
      BT #2
   BT #1
no.
```

> Terminierungsprobleme bei rekursiven Prozeduren treten immer dann auf, wenn das Rekursionsargument nicht instanziiert ist.

PEANO-Arithmetik

Terminierung durch Argumentüberwachung

```
add(X,0,X).
add(X,s(Y),R):-lt(X,R),add(s(X),Y,R).

lt(0,s(_)).
lt(s(X),s(Y)):-lt(X,Y).

?- lt(s(s(0)),s(s(s(0)))).
   lt(0,s(_)).                     fail
   lt(s(X1),s(Y1):-lt(X1,Y1).      succ(X1=s(0),Y1=s(s(0)))
   ?- lt(s(0),s(s(0))).            #1
      lt(0,s(_)).                  fail
      lt(s(X2),s(Y2):-lt(X2,Y2).   succ(X2=0,Y1=s(0))
      ?- lt(0,s(0)).               #2
         lt(0,s(_)).               succ
   yes

?- lt(s(s(0)),s(0)).
   lt(0,s(_)).                     fail
   lt(s(X1),s(Y1):-lt(X1,Y1).      succ(X1=s(0),Y1=0)
   ?- lt(s(0),0).                  #1
      lt(0,s(_)).                  fail
      lt(s(X2),s(Y2):-lt(X2,Y2).   fail
   no
```

Arithmetische Ausdrücke

Infix-Operatoren für Integer-Arithmetik

Addition	+
Substraktion	-
Multiplikation	*
ganzzahlige Division	/
Restklassenbildung	`mod`

Präfix-Operatoren

unäres Plus	+
unäres Minus	-

Operatoren beschreiben nur Termstrukturen

arithmetischer Ausdruck	arithmetische Prozedur
`+(2,3)`	`add(2,3,R).`
`*(4,6)`	`mult(4,6,R).`

- □ müssen erst in eine Auswertungsumgebung gestellt werden
- □ Auswertung greift auf die numerischen Operationen der Basismaschine zurück

→ funktionales Berechnungsmodell

Arithmetische Ausdrücke

Auswertungsumgebung: Infixoperator

```
Zahl is Ausdruck
```

Restriktionen

linke Seite: kein komplexer Term (Variable oder numerische Konstante)

rechte Seite: vollständig instanziierter arithmetischer Ausdruck

`3 * 5 - 7`
`X + 2 * Y`, falls `X` mit 4 und `Y` mit 3 instanziiert ist

`is/2` **schlägt fehl** , wenn

- eine der Typrestriktionen für die Argumente verletzt ist
  z.B. linke Seite ist ein komplexer Term
- linke und rechte Seite nicht miteinander unifizieren
  z.B. Ungleichheit der numerischen Werte auf der rechten
  und der linken Seite,
- Auswertung ergibt nichtnumerischen Datentyp

Arithmetische Ausdrücke

Beispiele

`3 is 2 * 5 - 7`	`succ`
`X is 2 * 3 + 1`	`succ(X=7)`
`2 + 1 is 2 * 5 - 7`	`fail`
`X is 5, X is 2 * 6`	`fail`
`X is 1, X is X + 1`	`fail`
`X is 1, X is X * 1`	`succ(X=1)`

hybride Sprache:

- Funktionale Auswertung in einer relationalen Umgebung
- Restriktionen spiegeln die nichtdeklarative Semantik der imperativen Hardwarearchitektur wieder $\rightarrow$ Behandlung der Arithmetik mit außerlogischen Mitteln

Arithmetische Ausdrücke

Beispiel: Fakultät

$$n! = \begin{cases} 1 & \text{für } n = 0 \\ (n-1)! * n & \text{sonst} \end{cases}$$

Induktionsanfang

Induktionsschritt

```
fak(N,Resultat)

fak(0,1).
fak(N,FakN):-N > 0,
             N1 is N - 1,
             fak(N1,FakN1),
             FakN is FakN1 * N.
```

Rekursion über das erste Argument

```
?- fak(2,X).
X=2 ;
no
?-fak(X,2)
instantiation error: N>0
```

> Eine relationale Prozedur, die mit Hilfe einer funktionalen Auswertung definiert wurde, kann ebenfalls nur funktional verwendet werden.

Arithmetische Ausdrücke

Beispiel: Größter gemeinsamer Teiler

$$\mathrm{gcd}(x,y) = \begin{cases} x & \text{wenn } x = y \\ \mathrm{gcd}(y,x) & \text{wenn } x < y \\ \mathrm{gcd}(x-y,y) & \text{sonst} \end{cases}$$

```
ggt(zahl1,zahl2,ggt)

ggt(X,X,X).
ggt(X,Y,Z) :- X<Y, ggt(Y,X,Z).
ggt(X,Y,Z) :- X>Y, X1 is X-Y, ggt(X1,Y,Z).

?- ggt(15,9,X).
   X=3 ;
   no
?- ggt(2,3,X).
   X=1 ;
   no
```

Arithmetische Ausdrücke

Beispiel: Russische Bauernmultiplikation (für natürliche Zahlen größer als 0)

$$\mathrm{rbm}(f_1, f_2) = \begin{cases} f_1 & \text{falls } f_2 = 1 \\ f1 + \mathrm{rbm}(f_1, f_2 - 1) & \text{falls ungeradzahlig}(f_2) \\ \mathrm{rbm}(2f_1, f_2/2) & \text{sonst} \end{cases}$$

```
rbm(Faktor1,Faktor2,Produkt)

rbm(F1,F2,F1) :- F2=:=1.
rbm(F1,F2,R) :- F2>1, odd(F2), F21 is F2-1,
                rbm(F1,F21,R1), R is R1+F1.
rbm(F1,F2,R) :- F2>1, F11 is F1*2, F21 is F2/2,
                rbm(F11,F21,R).

odd(X) :- 1 is X mod 2.
```

Arithmetische Ausdrücke

Beispiel: Russische Bauernmultiplikation (Testläufe)

```
?- rbm(4,7,R).
   R=28 ;
   no
?- rbm(24,157,R).
   R=3768 ;
   no
?- rbm(1,1,R).
   R=1 ;
   no
?- rbm(0,1,R).
   R=0 ;
   no
?- rbm(1,0,R).
   no
?- rbm(-5,4,R).
   R=-20 ;
   no
?- rbm(4,-5,R).
   no
?- rbm(X,5,20).
   Instantiation Error
?- rbm(5,X,20).
   Instantiation Error
```

Listennotation

extensionale Elementaufzählung

```
[a]
[a,b]
[a,b,c]
```

Unifikation zweier Listen durch paarweise Unifikation ihrer Elemente

erfolgreich: wenn alle Elementpaare unifizieren

Fehlschlag: wenn wenigstens ein Elementpaar nicht unifiziert

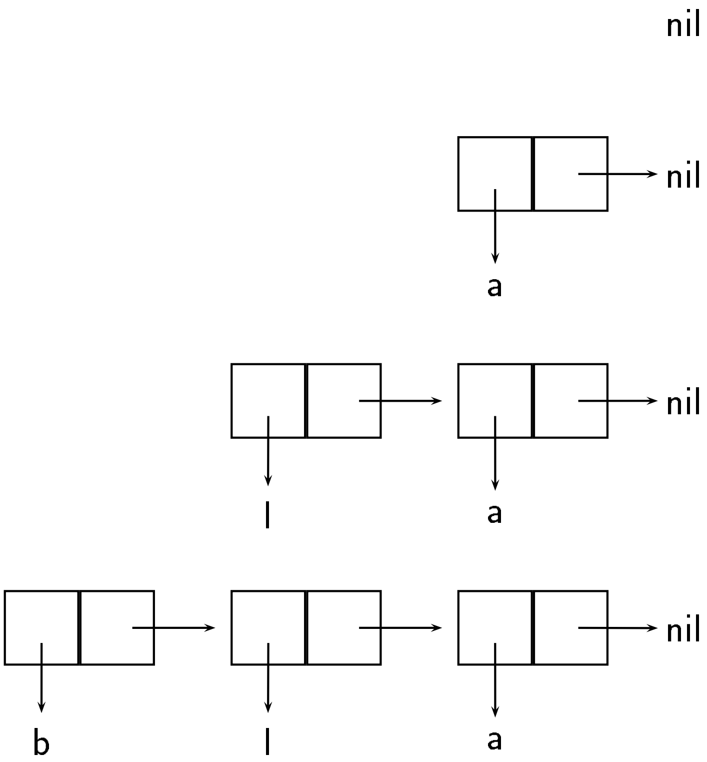
```
?- [a,b,c]=[a,b,c].
yes
?- [a,X,c]=[Y,b,c].
X=b, Y=a ;
no
?- [a,X,c]=[X,b,c].
no
?- [a,b,c]=[a,b].
no
```

leere Liste: `[ ]`

Listenverarbeitung

Listen als wichtigste rekursive Datenstruktur

- die leere Liste ist eine Liste
- ein *geordnetes Paar* aus einem Element und einer Liste ist eine Liste



`[ ]`

`[a]`

`[l,a]`

`[b,l,a]`

Restlistenseparator `[Listenkopf|Restliste]`

alternative (aber umständliche) Notation: `[b|[l|a|[ ]]]`

Listenverarbeitung

Unifikation zweier Listen durch paarweise Unifikation von Kopf und Restliste

erfolgreich: wenn Listenkopf und Restliste unifizieren

Fehlschlag:

- wenn der Unifikationspartner keine Liste ist
- wenn der Unifikationspartner eine Liste ist, aber der Listenkopf oder die Restliste nicht unifizieren

```
?- [a,b,c]=[X|Y].
   X=a,Y=[b,c] ;
   no
?- [a,b,c]=[X,Y|_].
   X=a, Y=b ;
    no
```

Elementselektion durch Unifikation (Dekomposition einer Liste)

```
db([sue,bob,tom]).
?- db([X|_]).
   X=sue ;
   no
?- db([_,_,X|_]).
   X=tom ;
   no
```

Listenverarbeitung

Konstruktion von Listen durch Unifikation (Komposition)

```
?- L=[sue,bob,tom],L1=[jim|L].
   L=[sue,bob,tom],L1=[jim,sue,bob,tom] ;
   no
?- [X,Y|R]=[sue,bob,tom],L1=[X,Y,jim|R].
   X=sue,Y=bob,R=[tom],L1=[sue,bob,jim,tom] ;
   no
```

Unifikation rekursiv eingebetteter Listen

```
?- [[a,b]|X]=[X,a,b]
   X=[a,b] ;
   no
?- [O,[P,Q],[a,O]]=[[b,Q],[R,[c,P]],[a,[b,[R,b]]]]
   O=[b,[c,b]],P=b,Q=[c,b],R=c ;
no
```

Listenverarbeitung

Beispiel 1: erstes Element einer Liste: `first(Element,Liste)`

Prozedurdefinition:

```
first(E,L):-
        L=[Kopf|Rest],
        Kopf=E.
```

Vereinfacht zu:

```
first(E,[E|_]).
```

Vollständig spezifizierter Aufruf:

```
?- first(a,[a,b,c]).
   yes
?- first(b,[a,b,c]).
   no
```

Listenverarbeitung

Beispiel 1: erstes Element einer Liste

Partiell unterspezifizierter Aufruf:

```
?- first(X,[a,b,c]).
   X=a ;
   no
?- first(a,X).
   X=[a|X] ;
   no
```

Vollständig unterspezifizierter Aufruf:

```
?- first(X,Y).
   X=X1, Y=[X1|Y1] ;
   no
?- first(X,X).
   . . .
```

Listenverarbeitung

Beispiel 2: Element einer Liste: `in_list(Element,Liste)`

Prozedurdefinition

```
in_list(E,L):-
          L=[Kopf|Rest],
          Kopf=E.

in_list(E,L):-
          L=[Kopf|Rest],
          in_list(E,Rest).
```

Vereinfacht zu:

```
in_list(E,[E,_]).

in_list(E,[Kopf|Rest]):-
          in_list(E,Rest).
```

Vollständig spezifizierter Aufruf:

```
?- in_list(b,[a,b,c]).
   yes
?- in_list(d,[a,b,c]).
   no
```

Listenverarbeitung

Beispiel 2: Element einer Liste

Partiell unterspezifizierter Aufruf:

```
?- in_list(X,[a,b,c]).
   X=a ;
   X=b ;
   X=c ;
   no
?- in_list(a,X).
   X=[a|X1] ;
   X=[X2,a|X3] ;
   X=[X2,X4,a|X5] ;
   X=[X2,X4,X6,a|X7]
   yes
```

Vollständig unterspezifizierter Aufruf:

```
?- in_list(X,Y).
   X=X1, Y=[X1|Y1] ;
   X=X1, Y=[X2,X1|Y2] ;
   X=X1, Y=[X2,X3,X1|Y3]
   yes
```

(oftmals) eingebautes Prädikat `member(Element,Liste)`

Listenverarbeitung

Beispiel 3: letztes Element einer Liste: `last(Element,Liste)`

Prozedurdefinition

```
last(E,L):-
        L=[Kopf|Rest],
        E=Kopf,
        Rest=[ ].

last(E,L):-
        L=[_|Rest],
        last(E,Rest).
```

Vereinfacht zu:

```
last(E,[E]).

last(E,[_|Rest]):-
        last(E,Rest).
```

Vollständig spezifizierter Aufruf:

```
?- last(c,[a,b,c]).
   yes
?- last(d,[a,b,c]).
   no
```

Listenverarbeitung

Beispiel 3: letztes Element einer Liste

Partiell unterspezifizierter Aufruf:

```
?- last(X,[a,b,c]).
   X=c ;
   no
?- last(a,X).
   X=[a] ;
   X=[X1,a] ;
   X=[X1,X2,a] ;
   X=[X1,X2,X3,a]
   yes
```

Vollständig unterspezifizierter Aufruf:

```
?- last(X,Y).
   X=X1, Y=[X1] ;
   X=X1, Y=[X2,X1] ;
   X=X1, Y=[X2,X3,X1] ;
   X=X1, Y=[X2,X3,X4,X1]
   yes
```

Listenverarbeitung

Beispiel 4: Länge einer Liste: `length(Zahl,Liste)`

Prozedurdefinition

```
length(N,L):-
        L=[ ],
        N is 0.

length(N,L):-
        L=[Kopf|Rest],
        length(N1,Rest),
        N is N1+1.
```

Vereinfacht zu:

```
length(0,[ ]).

length(N,[Kopf|Rest]):-
        length(N1,Rest),
        N is N1+1.
```

Vollständig spezifizierter Aufruf:

```
?- length(3,[a,b,c]).
   yes
?- length(4,[a,b,c]).
   no
```

Listenverarbeitung

Beispiel 4: Länge einer Liste

Partiell unterspezifizierter Aufruf:

```
?- length(X,[a,b,c]).
   X=3 ;
   no
?- length(3,X).
   X=[X1,X2,X3] ;
   . . .
```

Vollständig unterspezifizierter Aufruf:

```
?- length(X,Y).
   X=0, Y=[ ] ;
   X=1, Y=[Y1] ;
   X=2, Y=[Y1,Y2] ;
   X=3, Y=[Y1,Y2,Y3]
   yes
```

Listenverarbeitung

Beispiel 5: Verketten zweier Listen: `app(Liste1, Liste2, Resultat)`

Prozedurdefinition

```
app(L1,L2,R):-
        L1=[ ],
        R=L2.

app(L1,L2,R]):-
        L1=[Kopf|Rest1],
        app(Rest1,L2,ZR),
        R = [Kopf|ZR].
```

Vereinfacht zu:

```
app([ ],L,L).

app([Kopf|Rest1],L2,[Kopf|ZR]):-
        app(Rest1,L2,ZR).
```

Aufruf mit einem unterspezifizierten Argument:

```
?- app([a,b],[c,d,e],X).
   X=[a,b,c,d,e] ;
   no
?- app(X,[c,d,e],[a,b,c,d,e]).
   X=[a,b] ;
   no
?- app([a,b],X,[a,b,c,d,e]).
   X=[c,d,e] ;
   no
```

Listenverarbeitung

Beispiel 5: Verketten zweier Listen

Aufruf mit zwei unterspezifizierten Argumenten:

```
?- app(X,Y,[a,b,c,d,e]).
   X=[ ],Y=[a,b,c,d,e] ;
   X=[a],Y=[b,c,d,e] ;
   X=[a,b],Y=[c,d,e] ;
   X=[a,b,c],Y=[d,e] ;
   X=[a,b,c,d],Y=[e] ;
   X=[a,b,c,d,e],Y=[ ] ;
   no
```

Rekursion über dem ersten *und* dem dritten Argument

```
?- app(X,[c,d,e],Y).
   X=[ ], Y=[d,e] ;
   X=[X1], Y=[X1,d,e] ;
   X=[X1,X2], Y=[X1,X2,d,e] ;
   X=[X1,X2,X3], Y=[X1,X2,X3,d,e]
yes
?- app(X,Y,Z).
   X=[ ], Y=Y1, Z=Y1 ;
   X=[X1], Y=Y1, Z=[X1|Y1] ;
   X=[X1,X2], Y=Y1, Z=[X1,X2|Y1] ;
   X=[X1,X2,X3], Y=Y1, Z=[X1,X2,X3|Y1]
yes
```

(oftmals) eingebautes Prädikat `append(Liste1,Liste2,Resultat)`

Beispiel: Endliche Automaten

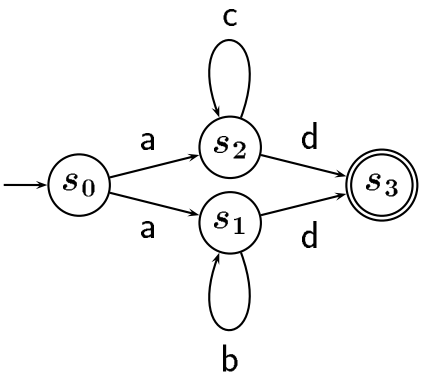
Repräsentation des Automaten:

Startzustand: `a(zustand)`

Zielzustand: `e(zustand)`

Zustandsübergang: `t(zustand-alt, symbol, zustand-neu)`

Beispiel: `a(b*|c*)d`




```
a(s0).

e(s3).

t(s0,a,s1).
t(s0,a,s2).
t(s1,d,s3).
t(s2,d,s3).
t(s1,b,s1).
t(s2,c,s2).
```

Beispiel: Endliche Automaten

Repräsentation der Zeichenfolge als Liste: `[a, b, b, b, d]`

Prozedur: `generate(Liste)`
Überprüft ob `Liste` durch den gegebenen Automaten erzeugt wird.

```
generate(Wort):-
        a(Start),
        gen(Wort,Start,Ziel),
        e(Ziel).

gen(Wort,Start,Ziel):-
        Wort=[ ],
        Start=Ziel.

gen(Wort,Start,Ziel):-
        Wort=[Kopf|Rest],
        t(Start,Kopf,Zwischenzustand),
        gen(Rest,Zwischenzustand,Ziel).
```

Vereinfachung zu:

```
gen([ ],Start,Start).

gen([Kopf|Rest],Start,Ziel):-
        t(Start,Kopf,Zwischenzustand),
        gen(Rest,Zwischenzustand,Ziel).
```

Beispiel: Endliche Automaten

Vollständig spezifizierter Aufruf: Test, ob gegebene Zeichenkette durch den Automaten erzeugt wird.

```
?- generate([a, b, a]).
   yes
?- generate([a,b]).
   no
?- generate([b,a]).
   no
```

Unterspezifizierter Aufruf: Generierung der durch den Automaten erzeugbaren Zeichenketten

```
?- generate(X).
   X=[a,d] ;
   X=[a,b,d] ;
   X=[a,b,b,d] ;
   X=[a,b,b,b,d]
   yes
```

Beispiel: Fahrplanauskunft

Repräsentation der Zugverbindungen

```
train(Zug-Nr, Abf-ort, Abf-zeit, Ank-ort, Ank-zeit)
```

Operator zur Zeitdarstellung: `10:56, 9:04`

```
:-op(200,xfx,':').
```

Fahrplan-Datenbank

```
train(ic803,   hamburg,      9:03,  wittenberge, 10:21).
train(ice585,  hamburg,      9:08,  hannover,    10:25).
train(ice575,  hamburg,      9:36,  hannover,    10:48).
train(ic803,   wittenberge, 10:23,  berlin,      11:32).
train(ic704,   wittenberge, 11:31,  hamburg,     12:47).
train(ir2633,  wittenberge, 10:33,  magdeburg,   11:53).
train(rb3815,  wittenberge, 11:02,  magdeburg,   12:38).
train(ic803,   berlin,      11:40,  leipzig,     13:34).
train(re4407,  magdeburg,   11:04,  leipzig,     12:43).
train(ir2633,  magdeburg,   12:03,  leipzig,     13:19).
train(ic645,   magdeburg,   12:43,  leipzig,     13:59).
train(ir2584,  magdeburg,   12:01,  hannover,    13:35).
train(ice940,  magdeburg,   12:26,  hannover,    13:54).
train(ir2447,  hannover,    10:22,  magdeburg,   11:59).
train(ice884,  hannover,    10:33,  hamburg,     11:52).
train(ice672,  hannover,    11:10,  hamburg,     12:22).
train(ice680,  hannover,    13:33,  hamburg,     14:52).
train(ice76,   hannover,    14:10,  hamburg,     15:22).
train(ic645,   hannover,    11:07,  magdeburg,   12:35).
train(ir2341,  hannover,    11:22,  magdeburg,   12:56).
```

Beispiel: Fahrplanauskunft

Prozedur 1: Über welche Zwischenstationen kommt man von A nach B?

```
connection(Startort, Zielort,Wegstrecke)

connection(A,B,W):-
        train(_,A,_,B,_).
        W=[A,B].

connection(A,B,W):-
        train(_,A,_,C,_),
        connection(C,B,W1)
        W=[A|W1].
```

Vereinfacht zu:

```
connection(A,B,[A,B]):-
        train(_,A,_,B,_).

connection(A,B,[A|W]):-
        train(_,A,_,C,_),
        connection(C,B,W).
```

→ transitive Hülle über der Erreichbarkeitsrelation

Beispiel: Fahrplanauskunft

Prozedur 1: Erreichbarkeit

Aufruf

```
?- connection(hamburg,leipzig,W).
   W = [hamburg, wittenberge, berlin, leipzig] ;
   W = [hamburg, wittenberge, hamburg, wittenberge,
        berlin, leipzig] ;
   W = [hamburg, wittenberge, hamburg, wittenberge,
        hamburg, wittenberge, berlin, leipzig] ;
   W = [hamburg, wittenberge, hamburg, wittenberge,
        hamburg, wittenberge, hamburg, wittenberge,
        berlin, leipzig]
   yes
```

Beispiel: Fahrplanauskunft

Prozedur 2: Zyklenüberwachung: `conn_z(Start, Ziel, Wegstr)`
Hilfsprozedur: `conn1(Start, Ziel, Bish_Wegstr, Ges_Wegstr)`

- Die (virtuell) zurückgelegte Wegstrecke wird in einer Liste `Bisherige_Wegstrecke` (3. Argument) gesammelt.
- Initialzustand ist der Ausgangsort.
- Das Aufsammeln muß bereits beim rekursiven Abstieg erfolgen
- Für jeden potentiellen (Zwischen-)zielort erfolgt Test auf Nichtenthaltensein in der bisher zurückgelegten Wegstrecke: `not(member(Ort,Wegstrecke)`.
- Das Resultat wird auf der vierten Argumentsposition zurückgereicht.

```
conn_z(A,B,W):-                          % neue Prozedur
        conn1(A,B,[A],GW),               % neues Argument
        reverse(GW,W).

conn1(A,B,BW,[B|BW]):-                   % neues Argument
        train(_,A,_,B,_),
        not(member(B,BW)).               % neues Ziel

conn1(A,B,BW,GW):-                       % neues Argument
        train(_,A,_,C,_),
        not(member(C,BW)),               % neues Ziel
        conn1(C,B,[C|BW],GW).            % neues Argument
```

Beispiel: Fahrplanauskunft

Prozedur 2: Zyklenüberwachung

- Nach der erfolgreichen Suche muß die Liste umgedreht werden:
  `reverse(Liste_alt,Liste_neu).`

```
reverse(L,R):-
        rev(L,[ ],R).

rev([ ],L,L).
rev([X|L],A,R):-
        rev(L,[X|A],R).
```

Aufruf:

```
?- conn_z(hamburg,leipzig,W).
   W = [hamburg, wittenberge, berlin, leipzig] ;
   W = [hamburg, wittenberge, magdeburg, leipzig] ;
   W = [hamburg, wittenberge, magdeburg, leipzig] ;
   W = [hamburg, wittenberge, magdeburg, leipzig] ;
   W = [hamburg, wittenberge, magdeburg, leipzig] ;
   W = [hamburg, wittenberge, magdeburg, leipzig] ;
   W = [hamburg, wittenberge, magdeburg, leipzig] ;
   W = [hamburg, hannover, magdeburg, leipzig] ;
   W = [hamburg, hannover, magdeburg, leipzig] ;
   ...
   W = [hamburg, hannover, magdeburg, leipzig] ;
   W = [hamburg, hannover, magdeburg, leipzig] ;
   W = [hamburg, hannover, magdeburg, leipzig] ;
   No
```

Beispiel: Fahrplanauskunft

Prozedur 3: Überwachung der Zeitrestriktionen beim Umsteigen:

`conn_zt(Start, Ziel, Frueheste-Abfahrt,Wegstr)`

Hilfsprozedur: `conn2(Start, Ziel, Bish_Wegstr, Ges_Wegstr)`

- ☐ Ankunftszeit
- ☐ Vergleich für strukturierte Zeitangaben `before(Zeitang,Zeitang)`

Prozedurdefinition:

```
before(Xh:Xm,Yh:Ym):-
        Xh<Yh.

before(H:Xm,H:Ym):-
        Xm<Ym.
```

Aufruf:

```
?- before(10:20,10:22).
   yes
?- before(10:20,10:18).
   no
```

Beispiel: Fahrplanauskunft

Prozedur 3: Zeitrestriktionen

- Zeitvergleich erfolgt zwischen zwei unterschiedlichen Zugverbindungen; Ankunftszeit muß zwischengespeichert werden
- Erweiterte Struktur der Wegstreckenliste
  - initial: `[Abfahrtsort,Früheste_Abfahrt]`
  - für jede Etappe wird folgende Information zur Wegstreckenliste hinzugefügt: `[Zielort, Ankunftszeit, Abfahrtszeit]`
- Vergleichszeit wird aus der letzten Eintragung der Wegstreckenliste entnommen: `BW=[_,Tf,|_]`

```
conn_zt(A,B,Tf,W):-                          % neues Argument
        conn2(A,B,[A,Tf]GW),                 % neue Struktur
        reverse(GW,[_|W]).                   % neue Struktur

conn2(A,B,BW,[B,Tan,Tab|BW]):-               % neue Struktur
        train(_,A,Tab,B,Tan),
        BW=[_,Tf|_],                         % neues Ziel
        before(Tf,Tab),                      % neues Ziel
        not(member(B,BW)).

conn2(A,B,BW,GW):-                           % neue Struktur
        train(_,A,Tab,C,Tan),
        not(member(C,BW)),
        BW=[_,Tf|_],                         % neues Ziel
        before(Tf,Tab),                      % neues Ziel
        conn2(C,B,[C,Tan,Tab|BW],GW).        % neue Struktur
```

Beispiel: Fahrplanauskunft

Prozedur 3: Zeitrestriktionen

Aufruf:

```
?- conn_zt(hamburg,leipzig,8:00,W).
   W = [hamburg, 9:3, 10:21, wittenberge, 10:23, 11:32,
            berlin, 11:40, 13:34, leipzig] ;
   W = [hamburg, 9:3, 10:21, wittenberge, 10:33, 11:53,
            magdeburg, 12:3, 13:19, leipzig] ;
   W = [hamburg, 9:3, 10:21, wittenberge, 10:33, 11:53,
            magdeburg, 12:43, 13:59, leipzig] ;
   W = [hamburg, 9:3, 10:21, wittenberge, 11:2, 12:38,
            magdeburg, 12:43, 13:59, leipzig] ;
   W = [hamburg, 9:8, 10:25, hannover, 11:7, 12:35,
            magdeburg, 12:43, 13:59, leipzig] ;
   W = [hamburg, 9:36, 10:48, hannover, 11:7, 12:35,
            magdeburg, 12:43, 13:59, leipzig] ;
   No
```

Beispiel: Fahrplanauskunft

Prozedur 4: Entfernen der Zwischenhalts von der Wegstreckenliste:

- Erneut erweiterte Struktur der Wegstreckenliste:
  `[... , Ort, Ank-zeit, Zugnummer, Abf-zeit, Ort, ...]`
- Folgen zwei Etappen mit gleicher Zugnummer aufeinander, so wird der Zwischenstopp von der Liste entfernt: `clear(Alte_Liste, Neue_Liste)`

```
clear([O2,Tan2,Zug,Tab1,O1,Tan1,Zug|L],R):-
        clear([O2,Tan2,Zug|L],C).
                % die ersten beiden Etappen benutzen den
                % gleichen Zug
                % streiche die Zwischenetappe

clear([O2,Tan2,Zug2,Tab1,O1,Tan1,Zug1|L],
                [O2,Tan2,Zug2,Tab1|R]):-
        Zug1\=Zug2,
        clear([O1,Tan1,Zug1|L],R).
                % die ersten beiden Etappen benutzen
                % unterschiedliche Z"uge
                % Rekursionsschritt: s"aubere die um eine
                % Etappe verk"urzte Liste

clear([O2,Tan2,Zug2,Tab1,O1,Tf],[O2,Tan2,Zug2,Tab1,O1,Tf]).
                % Rekursionsabschlu"s: die Liste enth"alt
                % nur noch eine Etappe
```

Beispiel: Fahrplanauskunft

Prozedur 4: ohne Zwischenhalts

Prozedurdefinition:

```
conn_ztc(A,B,Tf,W):-
        conn3(A,B,[A,Tf],GW),
        clear(GW,GWc),                         % neues Ziel
        reverse(GWc,[_|W]).

conn3(A,B,BW,[B,Tan,Zug,Tab|BW]):-         % neue Struktur
        train(Zug,A,Tab,B,Tan),
        BW=[_,Tf|_],
        before(Tf,Tab),
        not(member(B,BW)).

conn3(A,B,BW,GW):-
        train(Zug,A,Tab,C,Tan),
        not(member(C,BW)),
        BW=[_,Tf|_],
        before(Tf,Tab),
        conn3(C,B,[C,Tan,Zug,Tab|BW],GW).   % neue Struktur
```

Beispiel: Fahrplanauskunft

Prozedur 4: ohne Zwischenhalts

Aufruf:

```
?- conn_ztc(hamburg,leipzig,8:00,W).
   W = [hamburg, 9:3, ic803, 13:34, leipzig] ;
   W = [hamburg, 9:3, ic803, 10:21, wittenberge,
         10:33, ir2633, 13:19, leipzig] ;
   W = [hamburg, 9:3, ic803, 10:21, wittenberge,
         10:33, ir2633, 11:53, magdeburg,
         12:43, ic645, 13:59, leipzig] ;
   W = [hamburg, 9:3, ic803, 10:21, wittenberge,
         11:2, rb3815, 12:38, magdeburg,
         12:43, ic645, 13:59, leipzig] ;
   W = [hamburg, 9:8, ice585, 10:25, hannover,
         11:7, ic645, 13:59, leipzig] ;
   W = [hamburg, 9:36, ice575, 10:48, hannover,
         11:7, ic645, 13:59, leipzig] ;
   No
```

Beispiel: Fahrplanauskunft

Prozedur 5: Entfernen der Zwischenhalts bereits beim Listenaufbau

- Hilfsprozedur zum Listenaufbau
  `buildw(Ort, Ank_Zeit, Zug, Abf_Zeit, Bish_Weg,Gesamt_Weg)`

```
buildw(O2,Tan2,Zug,Tab1,[O1,Tan1,Zug|L],[O2,Tan2,Zug|L]).

buildw(O2,Tan2,Zug2,Tab1,[O1,Tan1,Zug1|L],
                [O2,Tan2,Zug2,Tab1,O1,Tan1,Zug1|L]):-
        Zug1\=Zug2.

buildw(O2,Tan2,Zug2,Tab1,W,R):-
        W=[O1,Tf],
        R=[O2,Tan2,Zug2,Tab1|W].
```

Beispiel: Fahrplanauskunft

Prozedur 5: ohne Zwischenhalts (Variante 2)

Prozedurdefinition:

```
conn_ztb(A,B,Tf,W):-
        conn4(A,B,[A,Tf],GW),              % Ziel entfernt
        reverse(GW,[_|W]).

conn4(A,B,BW,GW):-
        train(Zug,A,Tab,B,Tan),
        W=[_,Tf|_],
        before(Tf,Tab),
        not(member(B,BW)),
        buildw(B,Tan,Zug,Tab,BW,GW).       % neues Ziel

conn4(A,B,BW,GW):-
        train(Zug,A,Tab,C,Tan),
        not(member(C,BW)),
        BW=[_,Tf|_],
        before(Tf,Tab),
        buildw(C,Tan,Zug,Tab,BW,BWneu),    % neues Ziel
        conn4(C,B,BWneu,GW).
```

Weiterentwicklungen:

- ☐ Vergleich verschiedener Wegstrecken, Auswahl, Optimierung
- ☐ Kriterien: Umsteigen, Abfahrtzeit, Ankunftszeit, Gesamtzeit, Wartezeit, ...
- ☐ Vergleich partieller Wegstrecken