Prolog

Contents

[Prolog Basics 2](#_Toc68014835)

[Terme 2](#_Toc68014836)

[Atome 2](#_Toc68014837)

[Variablen 2](#_Toc68014838)

[Fakten 2](#_Toc68014839)

[Zahlen 2](#_Toc68014840)

[Abfragen 3](#_Toc68014841)

[Konjunktion von Abfragen 3](#_Toc68014842)

[Beispiel 3](#_Toc68014843)

[Regeln 3](#_Toc68014844)

[Syntax 3](#_Toc68014845)

[Beispiel 3](#_Toc68014846)

[Prädikate 3](#_Toc68014847)

[Beispiel 3](#_Toc68014848)

[Listen 4](#_Toc68014849)

[Unendliche Listen 4](#_Toc68014850)

[Arithmetik 4](#_Toc68014851)

[Methoden 5](#_Toc68014852)

[Listenfunktionen 5](#_Toc68014853)

[Member 5](#_Toc68014854)

[Append 5](#_Toc68014855)

[Reverse 5](#_Toc68014856)

[Quicksort 5](#_Toc68014857)

[Listenpermutation 5](#_Toc68014858)

[Prolog Expert 6](#_Toc68014859)

[Cuts 6](#_Toc68014860)

[Generate and Test 6](#_Toc68014861)

[Ausführungsbaum 6](#_Toc68014862)

[Rückwartsausführung 6](#_Toc68014863)

[Aufgaben 6](#_Toc68014864)

[Wolf-Ziege-Kohl 6](#_Toc68014865)

# Prolog Basics

## Terme

### Atome

Atome beginnen mit Kleinbuchstaben:

hans, inge, fritz, fisch

Komplexere Atome: in Hochkommata:

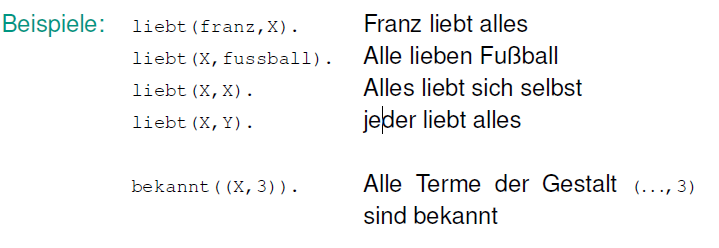
’Hallo Fritz!’

### Variablen

Platzhalter für unbekannte Terme.

Variablen beginnen mit Großbuchstaben oder Unterstrich:

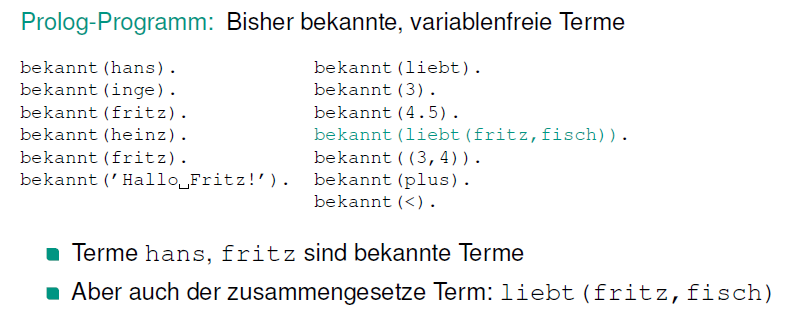
X, Y, \_X, X1, Fisch



### Fakten

Zusammengesetzte Terme:

liebt(fritz,fisch), liebt(fritz,X)



### Zahlen

3, 4.5

## Abfragen

Alle Fakten werden zu Laufzeit in einer Datenbank gehalten. Eine ”Abfrage“ wird durch ein Fragezeichen **?** eingeleitet und mit einem Punkt .

beendet:

**?**liebt(fritz,fisch)**.**

### Konjunktion von Abfragen

Konjunktion von Teilzielen getrennt durch Komma “,”:

?liebt(X,inge), liebt(inge,Y) .

### **Beispiel**

liebt(hans,inge).

liebt(heinz,inge).

liebt(inge,fritz).

liebt(fritz,fisch).

?liebt(X,inge), liebt(inge,Y) .

-> X = hans, Y = fritz

-> X = heinz, Y = fritz

-> false

## Regeln

### Syntax

term :- termlist .

Wobei :- als ”Wenn“ zu lesen ist und Kommata in termlist als ”Und“, wie in Abfragen.

### Beispiel

Wenn Inge X liebt und wenn X Fisch liebt, dann liebt Hugo X:

liebt(hugo,X) :- liebt(inge,X),liebt(X,fisch).

## Prädikate

Eine Gruppe von Fakten/Regeln mit gleichem Funktor und gleicher

Argumentzahl im Regelkopf heißt ”Prozedur“ oder ”Prädikat“

### Beispiel

grandparent(X,Y) :- parent(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).

parent(X,Y) :- father(X,Y).

mother(inge,emil).

mother(inge,petra).

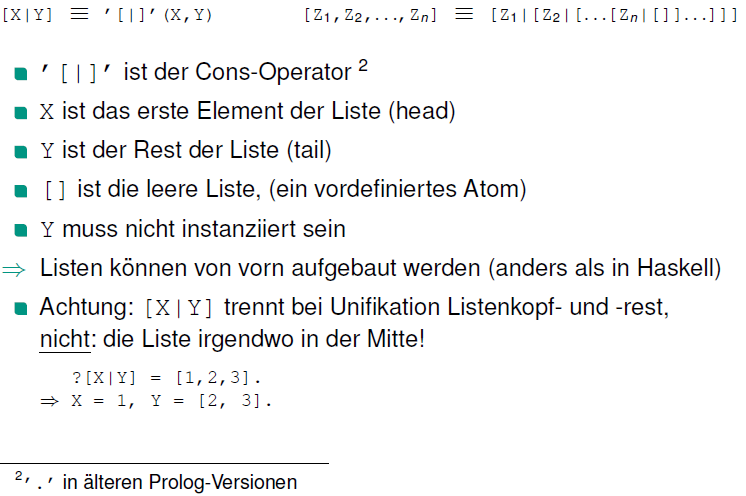
mother(petra,willi).

father(fritz,emil).

father(emil,kunibert).

father, mother, parent, grandparent sind Prädikate

## Listen



### Unendliche Listen

Die rev Funktion lässt sich in beiden Implementierungen rückwärts

ausführen. Die Abfrage

?rev(X,[1|R])

erzeugt alle Listen, deren Umkehrung mit 1 beginnt

X=[1], R=[];

X=[\_137,1], R=[\_137];

X=[\_138,\_137,1], R=[\_137,\_138];

...

Für unbekannte Listenelemente werden nichtinstanziierte (intern

durchnummerierte) Variablen verwendet. Das Ziel der Abfrage ist

unendlich oft re-erfüllbar.

## Arithmetik

# Methoden

## Listenfunktionen

### Member

member(X,[X|R]).

member(X,[Y|R]) :- member(X,R).

X kommt in einer Liste vor, wenn es mit dem ersten Element unifizierbar

ist oder wenn X im Listenrest R vorkommt.

### Append

append([],L,L).

append([X|R],L,[X|T]) :- append(R,L,T).

Die Konkatenation von [] und L ist L. Wenn die Konkatenation von R und

L die Liste T ergibt, dann ergibt die Konkatenation von [X|R] und L die

Liste [X|T].

?append([1,2,3,4],[2,3,4,5],X).

-> (einzige) Ausgabe: X = [1,2,3,4,2,3,4,5]

### Reverse

Naive, aber reicht

rev([],[]).

rev([X|R],Y) :- rev(R,Y1),append(Y1,[X],Y).

Effizienter

rev(X,Y) :- rev1(X,[],Y).

rev1([],Y,Y).

rev1([X|R],A,Y) :- rev1(R,[X|A],Y).

### Quicksort

qsort([],[]).

qsort([X|R],Y) :- split(X,R,R1,R2),

qsort(R1,Y1),

qsort(R2,Y2),

append(Y1,[X|Y2],Y).

split(X,[],[],[]).

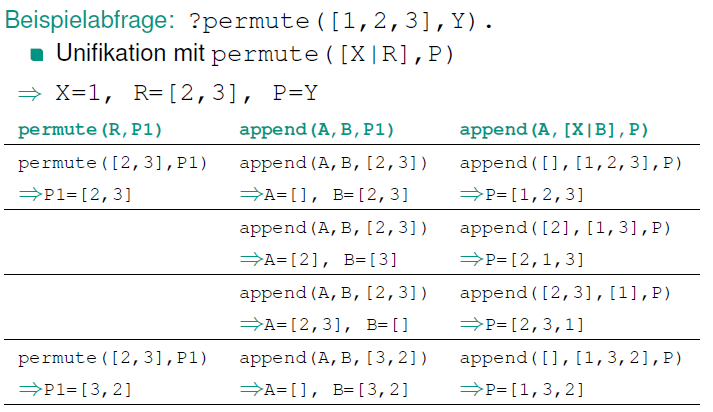
split(X,[H|T],[H|R],Y) :- X>H, split(X,T,R,Y).

split(X,[H|T],R,[H|Y]) :- X=<H, split(X,T,R,Y).

### Listenpermutation

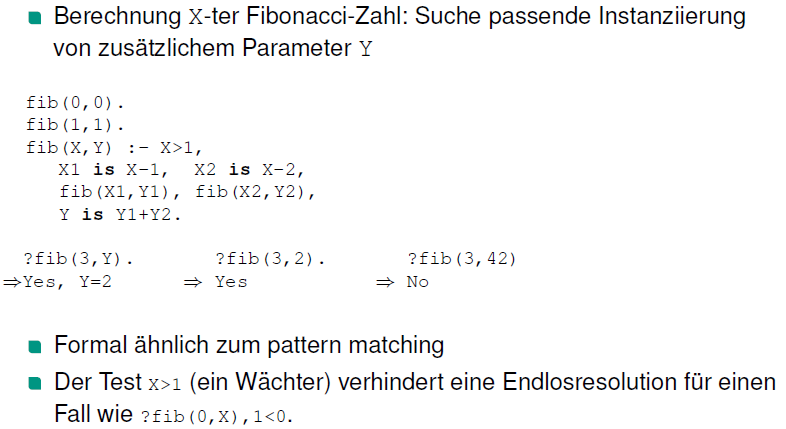
permute([],[]).

permute([X|R],P) :- permute(R,P1),append(A,B,P1),append(A,[X|B],P).



### Fibonacci

Einfach:



Mit Cuts:

fib(0,Y) :- !,Y=0.

fib(1,Y) :- !,Y=1.

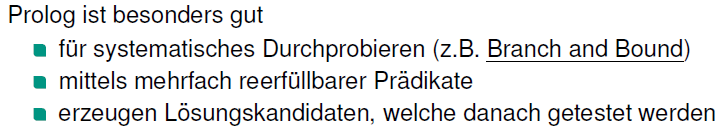
fib(X,Y) :- X1 is X-1, X2 is X-2,

fib(X1,Y1), fib(X2,Y2),

Y is Y1+Y2.

# Prolog Expert

## Generate and Test



Funktion :- Generator-Teil, Tester-Teil

Vermeidung kombinatorischer Explosion: **Generator** möglichst intelligent machen

### Generate

Beispiel für unendlich oft reerfüllbares Prädikat:

nat1(0).

nat2(X) :- nat(Y), X is Y+1.

### Test

Verwendung zum systematischen Durchprobieren natürlicher Zahlen:

sqrt(X,Y) :- nat(Y),

Y2 is Y\*Y, Y3 is (Y+1)\*(Y+1),

Y2 =< X, X < Y3.

?sqrt(27,X). -> liefert X=5

## Ausführungsbaum

## Rückwartsausführung

?append(X,Y,[1,2,3]).

-> X = [], Y = [1,2,3];

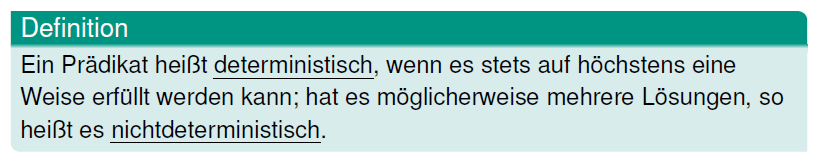
-> X = [1], Y = [2,3];

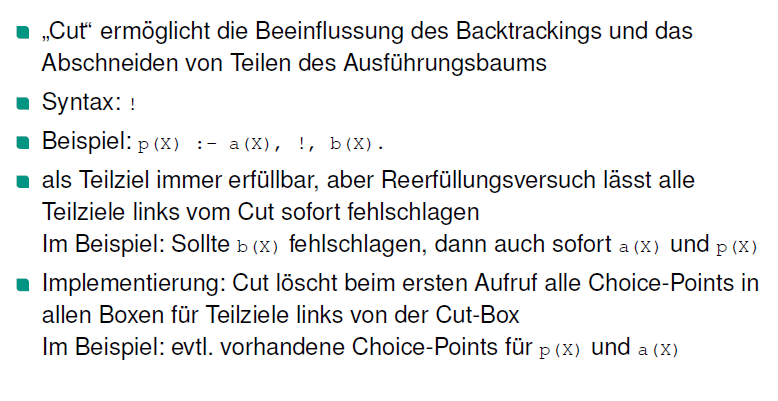
-> X = [1,2], Y = [3];

-> X = [1,2,3], Y = []

## Der Cut

### Determinismus

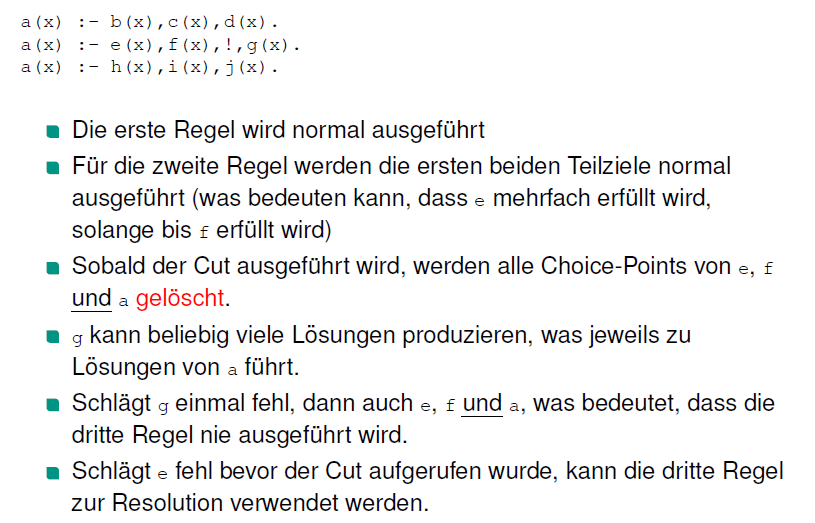




Sehr wichtig: **als Teilziel immer erfüllbar**, aber **Reerfüllungsversuch** lässt alle Teilziele **links vom Cut sofort fehlschlagen**.

Im Beispiel: Sollte b(X) fehlschlagen, dann auch sofort a(X) und p(X)

### Beispiel

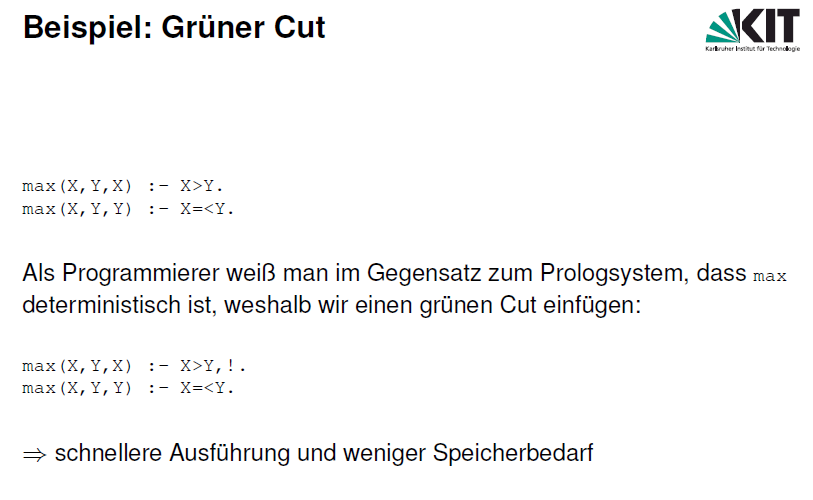


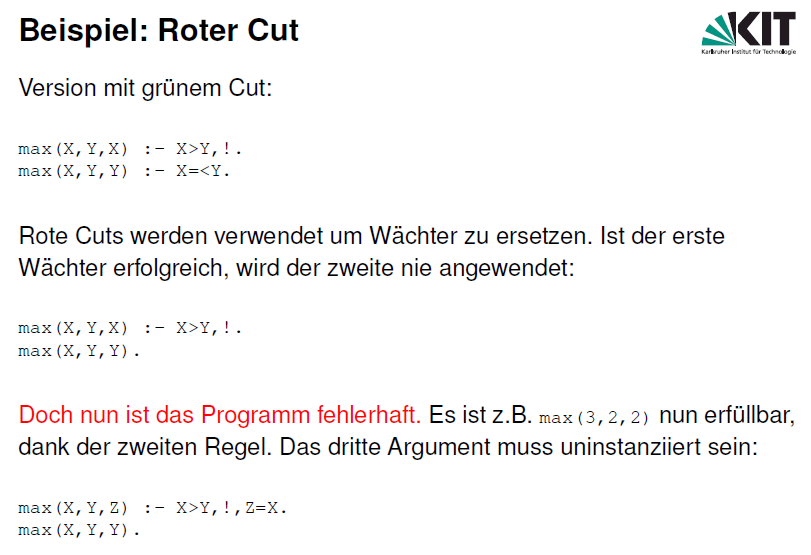
### Blaue, grüne und rote Cuts

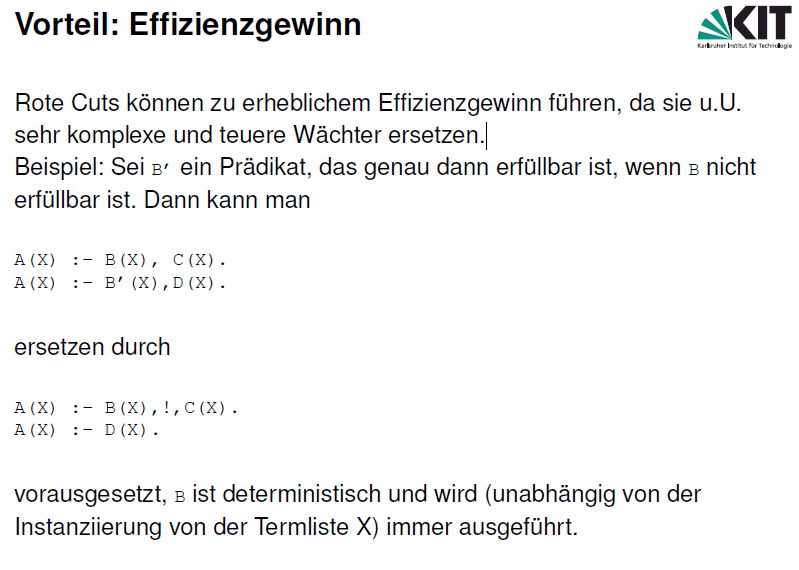
Blauer Cut: beeinflusst weder Programmlaufzeit, noch -verhalten

Grüner Cut: beeinflusst Programmlaufzeit, aber nicht -verhalten

Roter Cut: beeinflusst das Programmverhalten







### Faustregel

Faustregel: Der Cut darf erst kommen, wenn man weiß, dass **man in der richtigen Regel ist**, aber muss **vor der Instanziierung der Ausgabevariablen** stehen.

# Aufgaben

## Wolf-Ziege-Kohl