Prolog

Contents

[Prolog Basics 2](#_Toc68032218)

[Terme 2](#_Toc68032219)

[Atome 2](#_Toc68032220)

[Variablen 2](#_Toc68032221)

[Fakten 3](#_Toc68032222)

[Zahlen 3](#_Toc68032223)

[Abfragen 3](#_Toc68032224)

[Konjunktion von Abfragen 3](#_Toc68032225)

[Beispiel 3](#_Toc68032226)

[Regeln 4](#_Toc68032227)

[Syntax 4](#_Toc68032228)

[Beispiel 4](#_Toc68032229)

[Prädikate 4](#_Toc68032230)

[Beispiel 4](#_Toc68032231)

[Listen 4](#_Toc68032232)

[Unendliche Listen 5](#_Toc68032233)

[Arithmetik und Vergleich 5](#_Toc68032234)

[Unifikation 5](#_Toc68032235)

[Vergleichsoperatoren 5](#_Toc68032236)

[Auswertung mit is 5](#_Toc68032237)

[Methoden 6](#_Toc68032238)

[Listenfunktionen 6](#_Toc68032239)

[Member 6](#_Toc68032240)

[Append 6](#_Toc68032241)

[Reverse 6](#_Toc68032242)

[Quicksort 6](#_Toc68032243)

[Listenpermutation 7](#_Toc68032244)

[even und odd 7](#_Toc68032245)

[Fibonacci 7](#_Toc68032246)

[Prolog Expert 8](#_Toc68032247)

[Generate and Test 8](#_Toc68032248)

[Generate 9](#_Toc68032249)

[Test 9](#_Toc68032250)

[Ausführungsbaum 9](#_Toc68032251)

[Rückwartsausführung 9](#_Toc68032252)

[Der Cut 9](#_Toc68032253)

[Determinismus 9](#_Toc68032254)

[Beispiel 10](#_Toc68032255)

[Blaue, grüne und rote Cuts 10](#_Toc68032256)

[Faustregel 12](#_Toc68032257)

[Exotische Programme 13](#_Toc68032258)

[Differentialrechnung 13](#_Toc68032259)

[Acht Damen Problem 13](#_Toc68032260)

[More Money 14](#_Toc68032261)

[Aufgaben 14](#_Toc68032262)

[Wolf-Ziege-Kohl 14](#_Toc68032263)

# Prolog Basics

## Terme

### Atome

Atome beginnen mit Kleinbuchstaben:

hans, inge, fritz, fisch

Komplexere Atome: in Hochkommata:

’Hallo Fritz!’

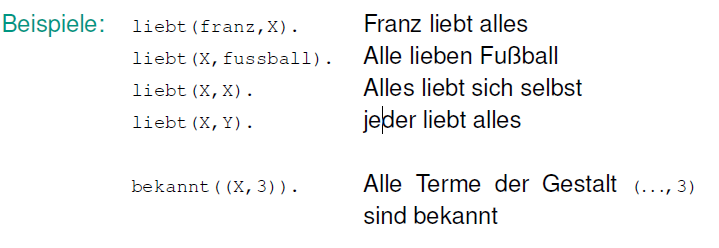
**Zahlen, Variablen und Fakten sind keine Atome! Atome sind: links, fritz, panzer, brot**

### Variablen

Platzhalter für unbekannte Terme.

Variablen beginnen mit Großbuchstaben oder Unterstrich:

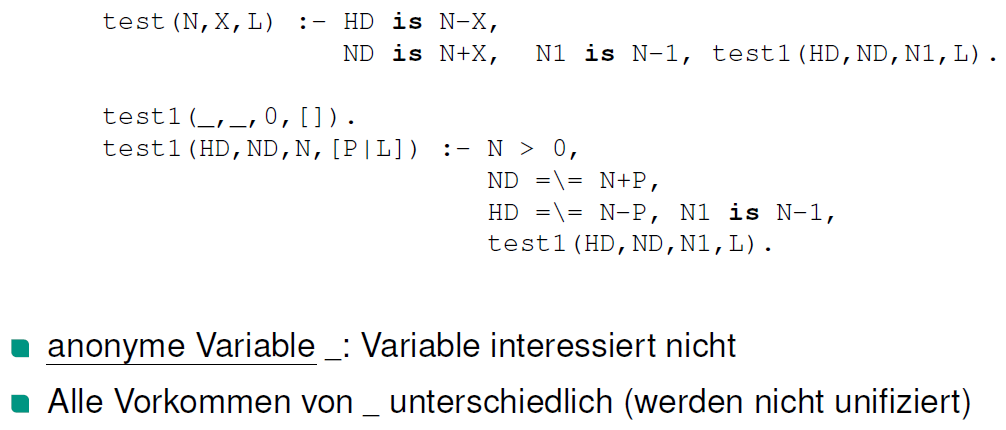
X, Y, \_X, X1, Fisch



### Anonyme Variablen

Anonyme Variablen werden mit \_ ausgedrückt.

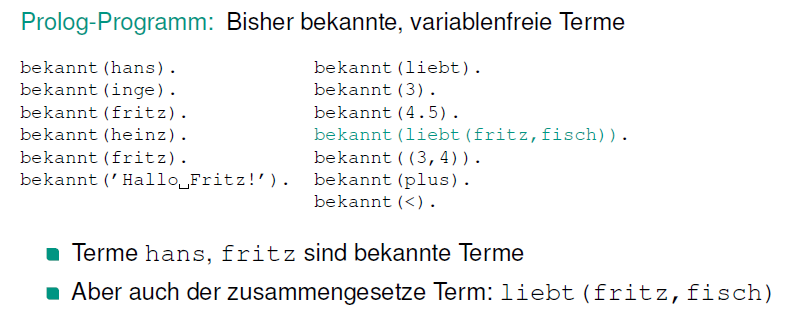
Genau wie bei Haskell: test(\_, X) -> egal was auf der 1.Stelle steht, interessiert nicht, wird nicht verwendet, muss nicht unifiziert werden.



### 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“

testA(X) . - einstelliger Prädikat

testB(X, Z) . - zweistelliger Prädikat

test((A,B,C)) . - Einstelliger Prädikat, Tupel als Argument

### 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).

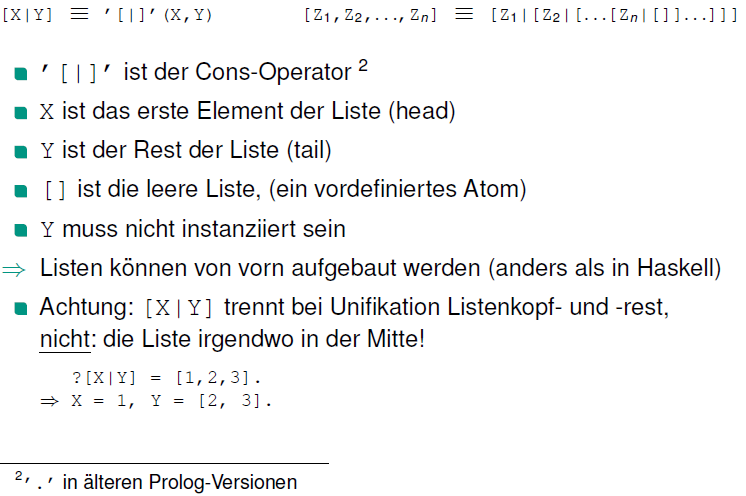
mother(petra,willi).

father(fritz,emil).

father(emil,kunibert).

father, mother, parent, grandparent sind Prädikate

## Listen



Plain: [X, Y, Z, A, B, C]

Head: [X | Rest]

Two heads: [X | **[Y | Rest]**]

### 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 und Vergleich

### Unifikation

Unifikation gilt auch für **uninstanziierte** Variablen

Unifikation: = (wertet die beiden Seiten **NICHT** aus)

?- 3 = 3. -> true

?- 1 + 2 = 3. -> false

Unifiziert nicht: \=

?- 2 \= 3. -> true

### Vergleichsoperatoren

Gilt nur für **vollständig instanziierte** Variablen!

Equal: =:= (wertet die beiden Seiten aus)

?- 1 + 2 =:= 3. -> true

Not equal: =\= (vertet die beiden Seiten aus)

?- 2 + 4 =\= 1. -> true

?- 1 =\= 1 -> false

<, =<, >, >= arithmetischer Vergleich

?- 1+2 <= 1 -> false

### Auswertung mit is

Variablen im **rechten** Term müssen **instanziiert** sein!

(Der linke Term kann uninstanziierte Variablen enthalten)

Auswertung und Unifikation: **is**

?X is 3+3. -> true, X=6

?1 is 3\*3. -> false

?X is 0\*Y -> ERROR: Arguments not sufficiently instantiated

“Zuweisungs”-Teilziel X **is** X + 1 nie erfolgreich!

# 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.

### Member2

member2(X,L) ist erfüllt, wenn L zwei aufeinanderfolgende Vorkommen von X enthält.

member2(X,[X|[X|Rest]]).

member2(X,[Y|Rest]) :- member2(X, Rest).

### 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,[],[],[]).

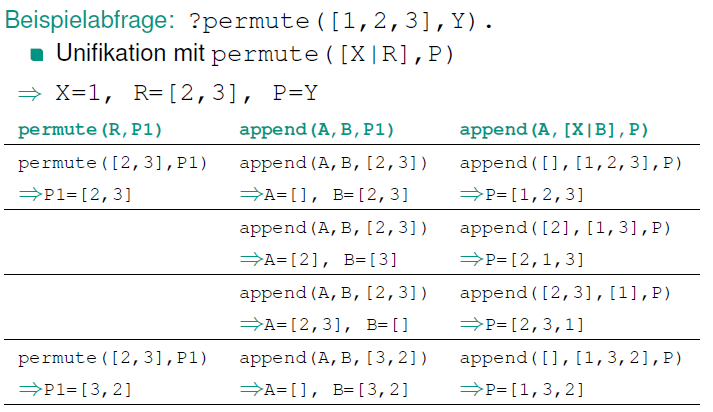
split(X,[H|T],[H|R],Y) :- X>H, split(X,T,R,Y).

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).



### even und odd

RICHTIG:

even(0).

even(X) :- X>0, X1 is X-1, odd(X1).

odd(1).

odd(X) :- X>1, X1 is X-1, even(X1).

?even(2) -> Yes

Aber Achtung: Arithmetische Ausdrücke **unifizieren nicht** mit Konstanten!

FALSCH:

~~even(0)~~.

~~even(X)~~ :- X>0, odd(X-1).

~~odd(1)~~.

~~odd(X)~~ :- X>1, even(X-1).

?even(2) -> No

### atom

atom(Term) -> True if *Term* is bound to an atom.

?- atom(pizza). -> true.

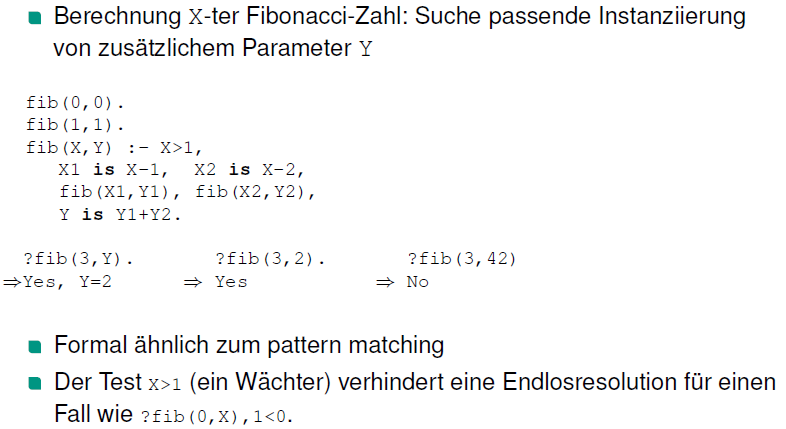
?- atom(likes(mary, pizza)). -> false.

?- atom(235). -> false.

**Zahlen, Variablen und Fakten sind keine Atome! Atome sind: links, fritz, panzer, brot**

### Fibonacci

Einfach:



Mit Cuts:

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

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

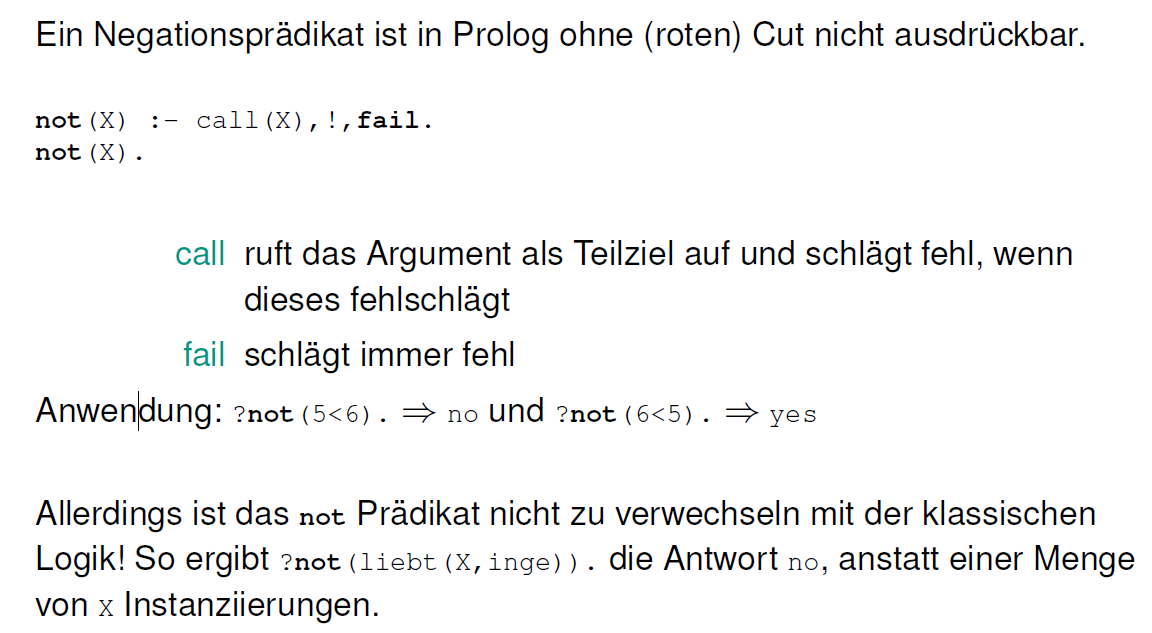
fib(X,Y) :- X1 is X-1, X2 is X-2,

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

Y is Y1+Y2.

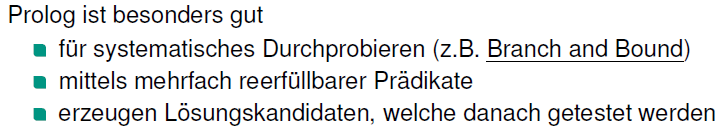
not(X) :- call(X),!,fail.

not(X).

****

# 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];

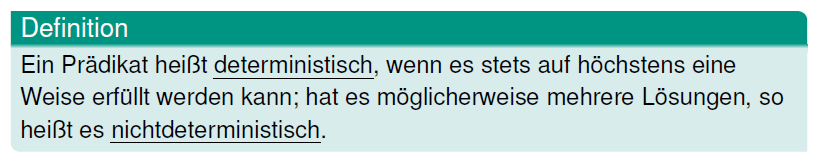
-> X = [1], Y = [2,3];

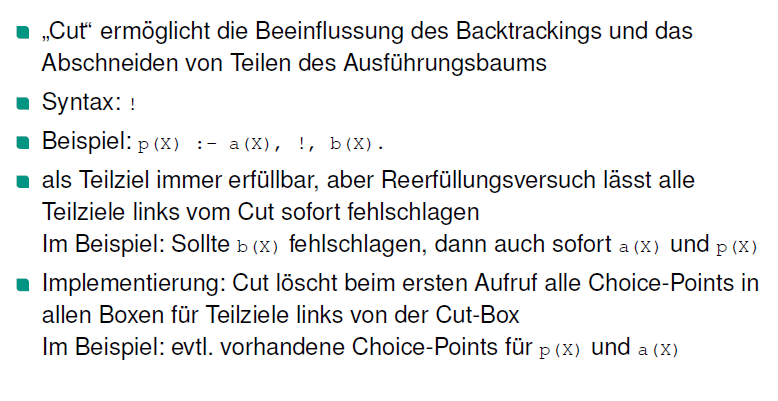
-> X = [1,2], Y = [3];

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

## Der Cut

### Determinismus

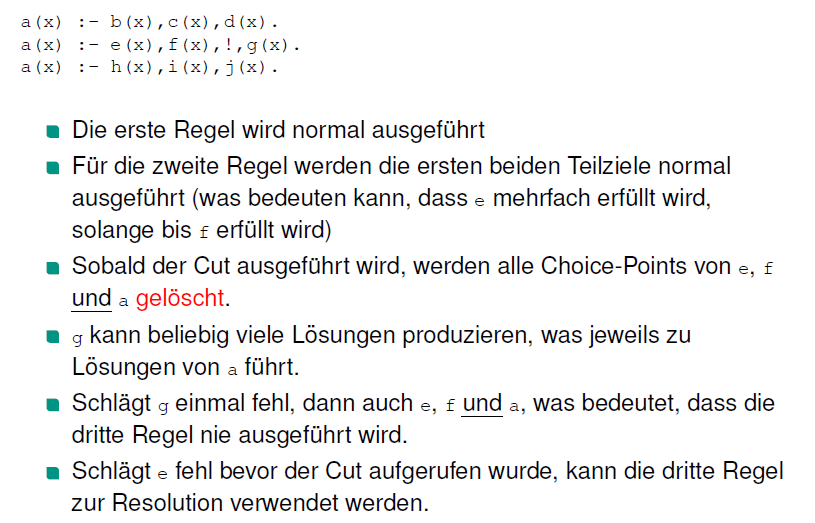




Sehr wichtig: **als Teilziel immer erfüllbar**, aber **Reerfüllungsversuch** lässt alle Teilziele **links vom Cut sofort fehlschlagen**.

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

### Beispiel

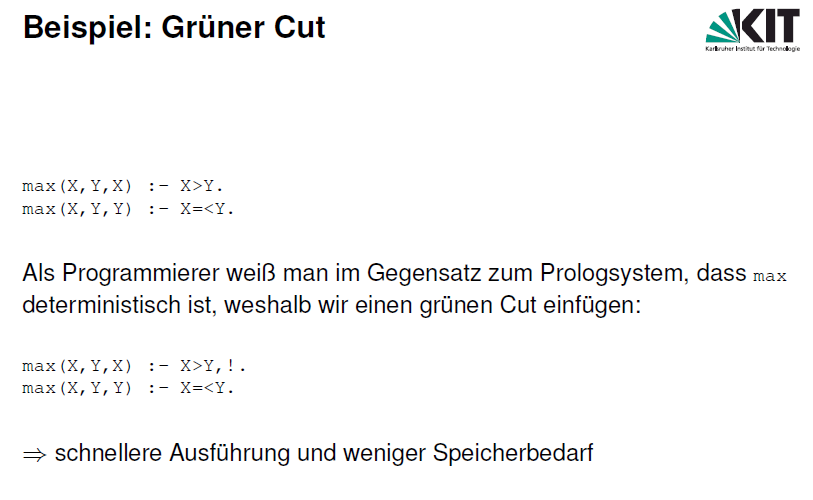


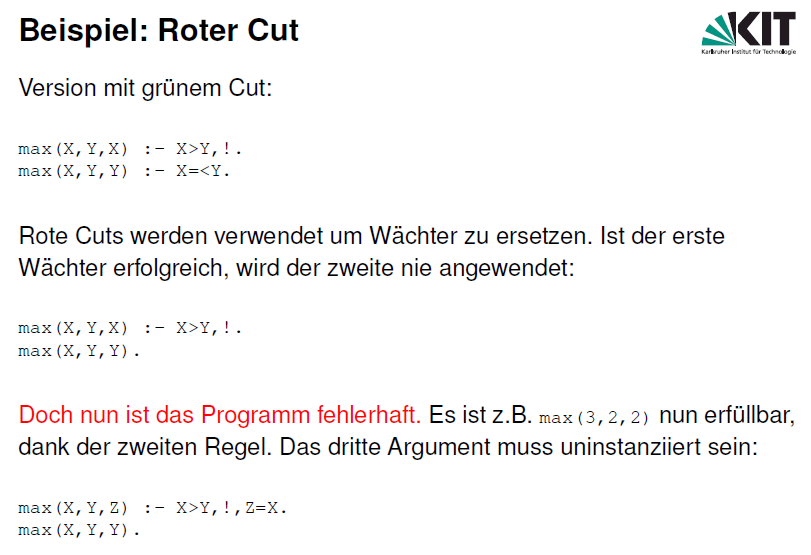
### Blaue, grüne und rote Cuts

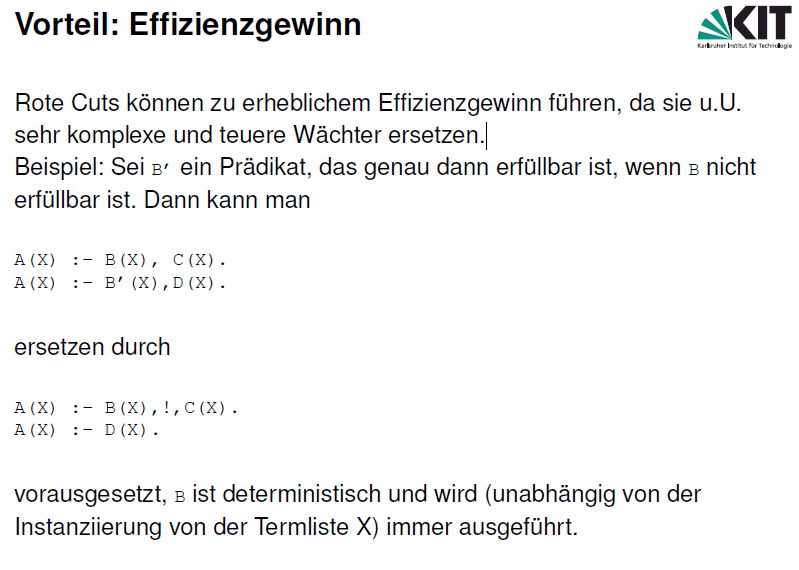
Blauer Cut: beeinflusst weder Programmlaufzeit, noch -verhalten

Grüner Cut: beeinflusst Programmlaufzeit, aber nicht -verhalten

Roter Cut: beeinflusst das Programmverhalten

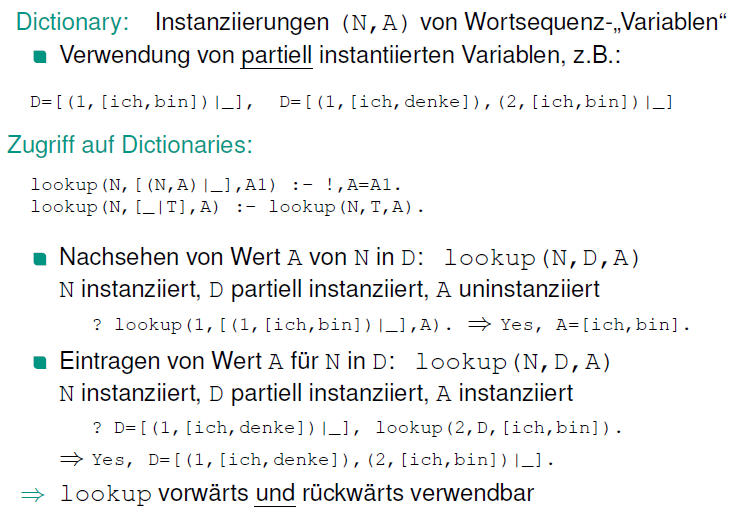






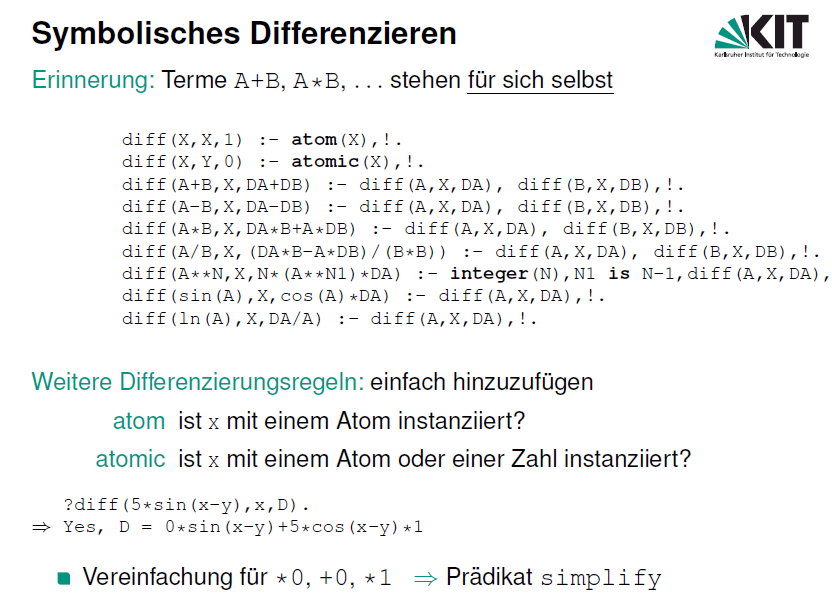
### Faustregel

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

Dictionaries

# Exotische Programme

### Differentialrechnung

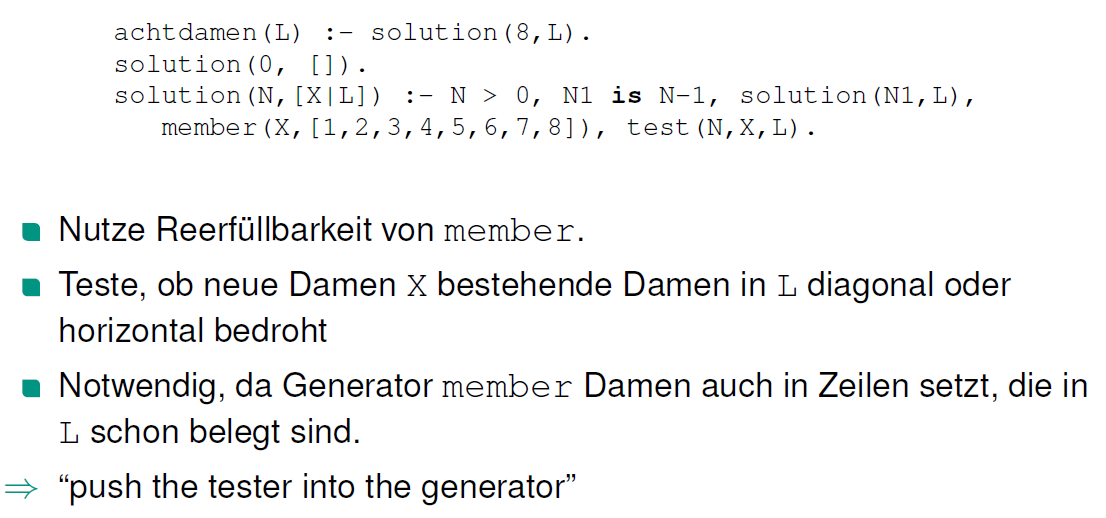


### Acht Damen Problem

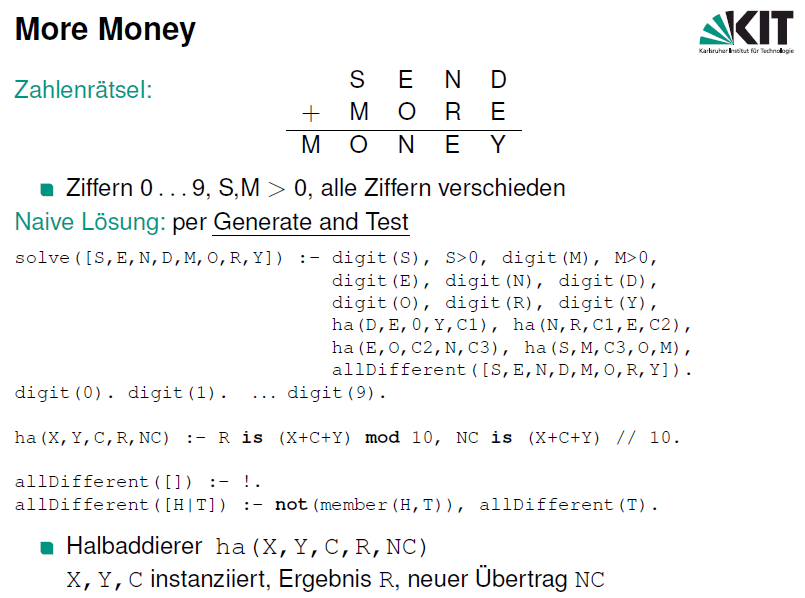
**Einfach, naiv:**



**Effizienter:**



## More Money

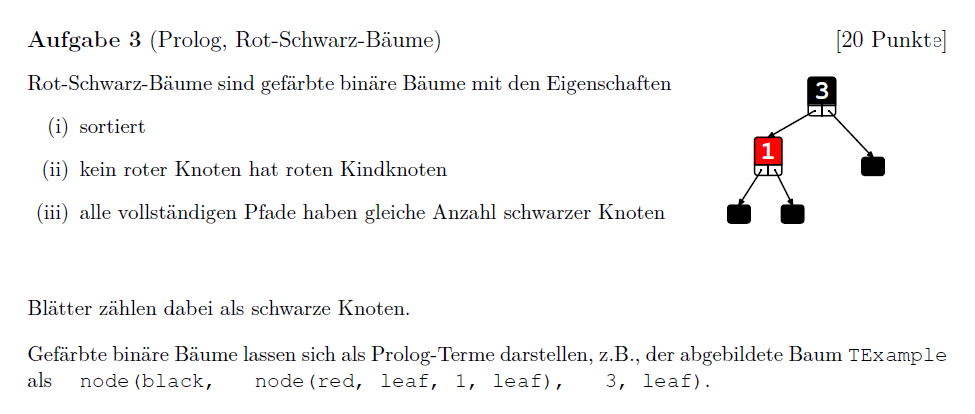


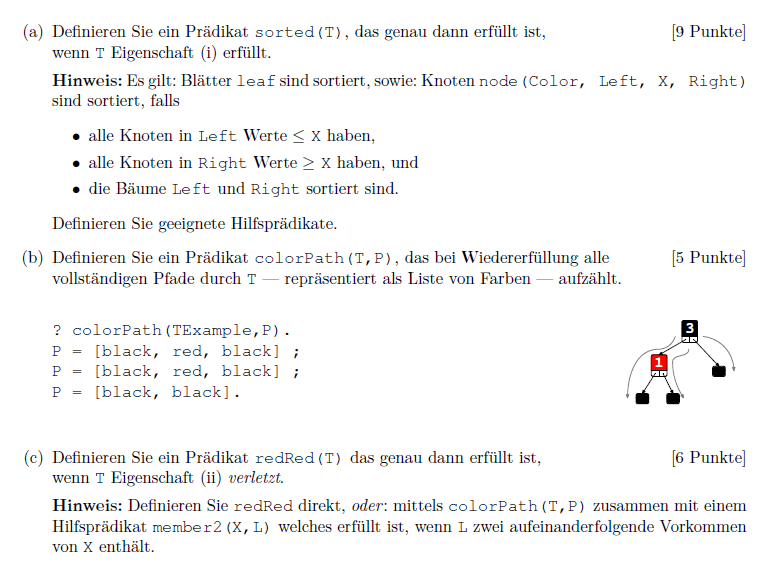
# Aufgaben

Siege Übungsblatt 6

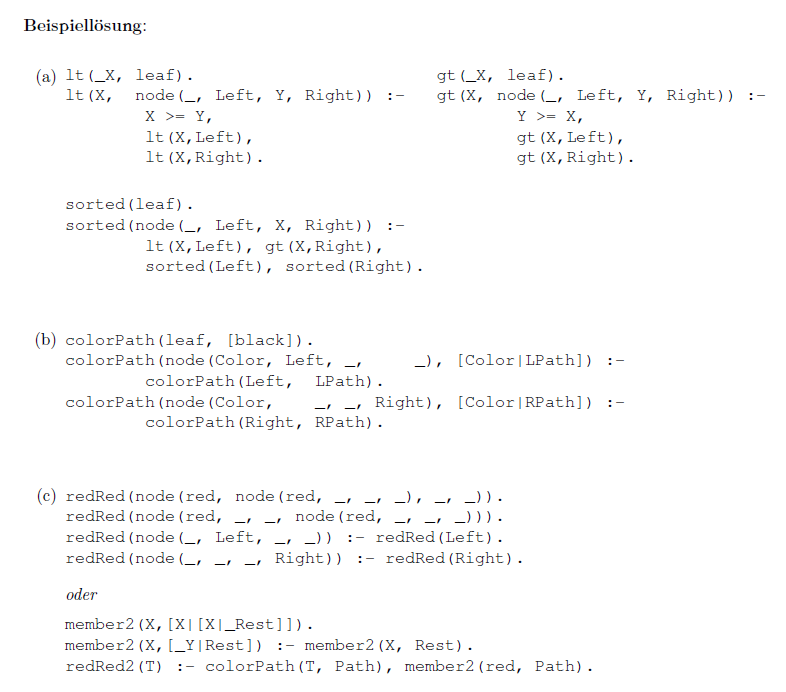
## Rot-Schwarz-Bäume

### Aufgabe



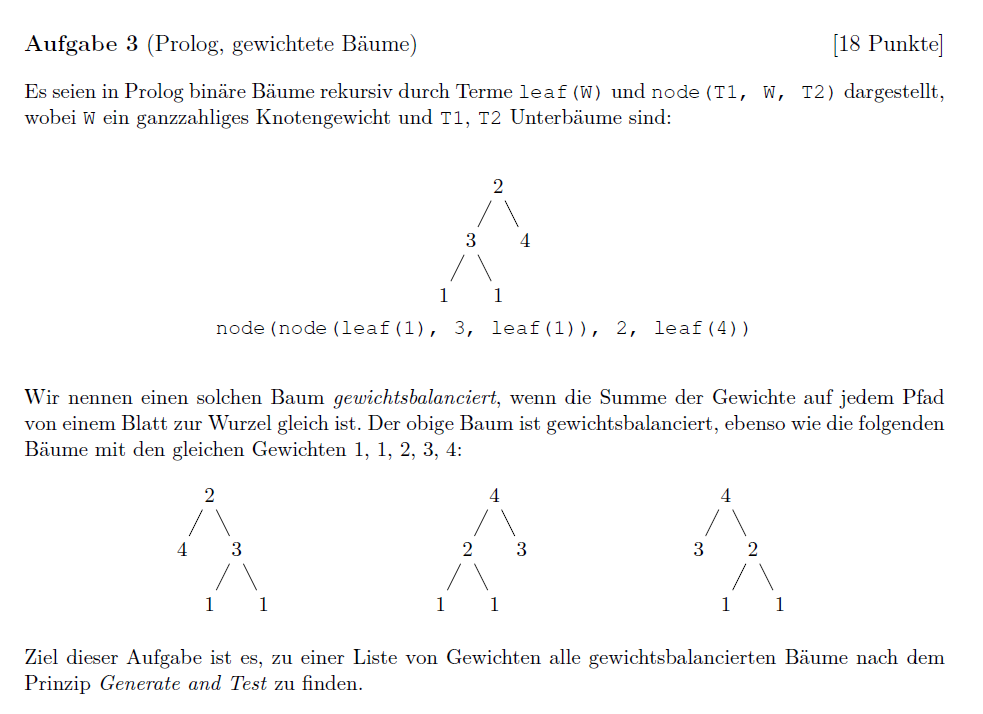


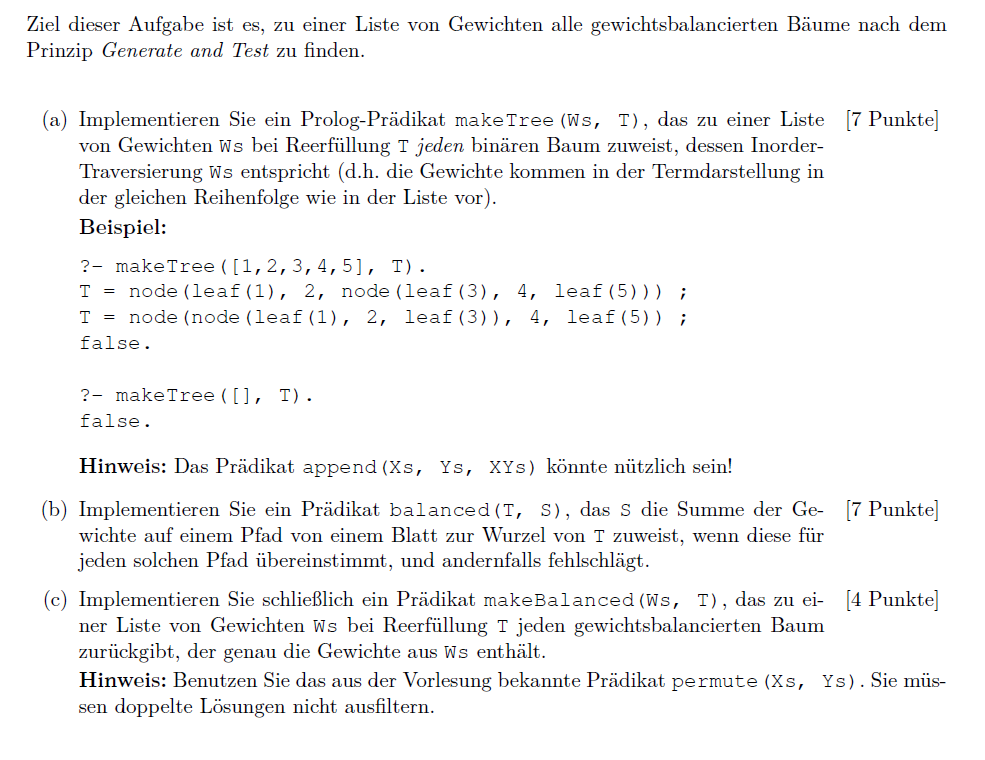
### Lösung



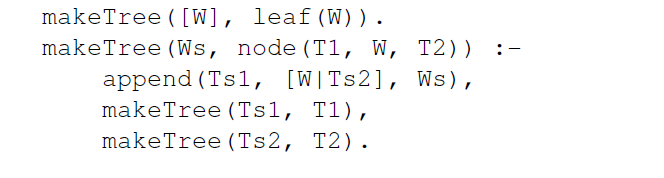
## Gewichtete Bäume

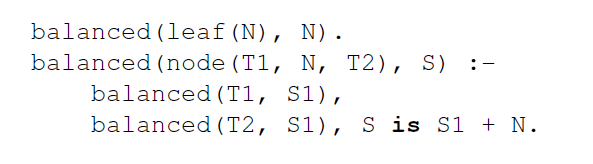
### Aufgabe

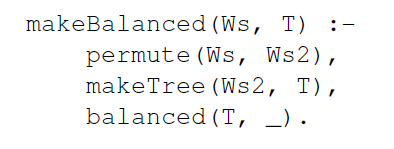




### Lösung







### Erklärung

makeTree(List, T) -> List = (Value), nur 1 Element -> T = leaf(Value)

makeTree(List, T) -> List = (LeftPart, Value, RightPart), LeftPart and RightPart not empty,

T = node(makeTree(LeftPart), Value, makeTree(RightPart))