Notfallblatt

### Komplizierte mgu

A picture containing chart

Description automatically generated

Die funktionstypen sind rechtsassoziativ: a1 = a2 -> a3 -> a4 = a2 -> (a3 -> a4)

1. Man merkt, dass es 2 Regeln für a1 gibt.

Daraus folgt, dass:

1. a2 = a5

2. a6 = a3 -> a4

2. Dann alles öffnen

a2 = a5 = a4 -> (a3 -> a4)  
 a6 = a3 -> a4

a5 = a4 -> (a3 -> a4)

a1 = (a4 -> (a3 -> a4)) -> (a3 -> a4)

3. Unifikator bilden:

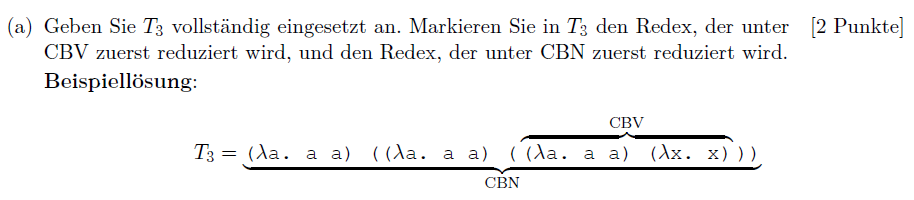
a2 => a4 -> (a3 -> a4)

a6 => a3 -> a4

a5 => a4 -> (a3 -> a4)

a1 => (a4 -> (a3 -> a4)) -> (a3 -> a4)

### CBN vs CBV



CBN: Linkeste, die nicht von Lambda umgeben

CBV: Linkeste, die nicht von Lamnda umgeben UND argument ein Wert ist

# 

## Constraints aus Var, Const, Abs, App ableiten

Text

Description automatically generated

Regel: a4 = a8

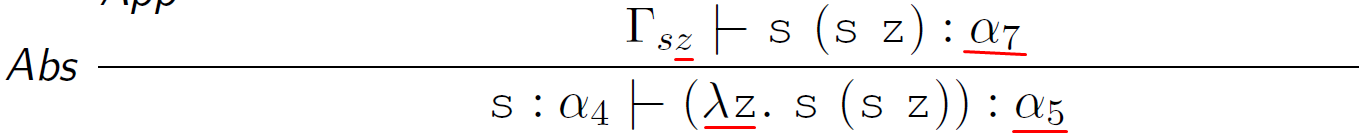
A picture containing text, orange

Description automatically generated

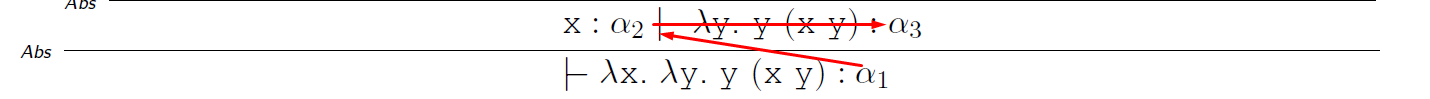
Regel: a8 = bool (genau so mit int)

Text

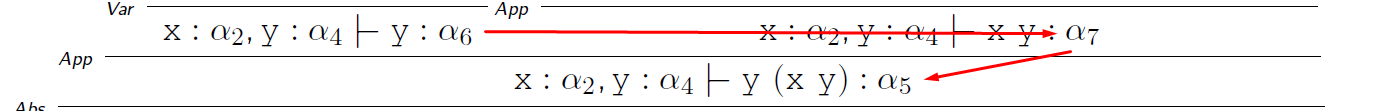
Description automatically generated



Regel: a5 = a6 -> a7 (typ von abgeschnietener lambda -> typ von innerem Ausdruck)



Regel: a1 = a2 -> a3 (Typ von Lambda -> Typ von Inner)



Regel: a6 = a7 --> a5, es kommt weiter keine Regel für a5

### Parser für Gramamtik

Man muss also ganz dumm vorgehen, ohne verschtehen zu versuchen, was die Grammatik macht.

1. Berechne Indizminge

2. Bestimme mit switch-case mit Indizmenge die Produktionen

**3. eps-Produktion/Ende der Produktion -> einfach return (kein parse von möglichen Follows machen!)**

4. Terminalsymbol T -> expect(T)

5. Nichtterminal -> parseNichtterminal

**A -> B C**

**B -> eps | <A>**

**C -> . | id**

**Indizen: A - egal, B - Follow(B) = [. oder id] und [<], C - [. oder id]**

void parseA() {

parseB();

parseC();

return;

}

void parseB() {

switch(lexer.current)

case DOT:

case IDENT:

return;

case LEFTBR:

expect(LEFTBR);

parseA();

expect(RIGHTBR);

return

default:

error();

}

void parseC() {

switch(lexer.current)

case DOT:

expect(DOT)  
 return;

case IDENT:

expect(IDENT)  
 return;

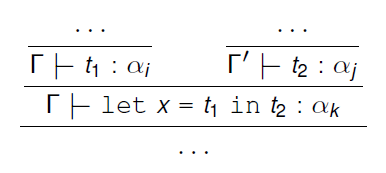
default:

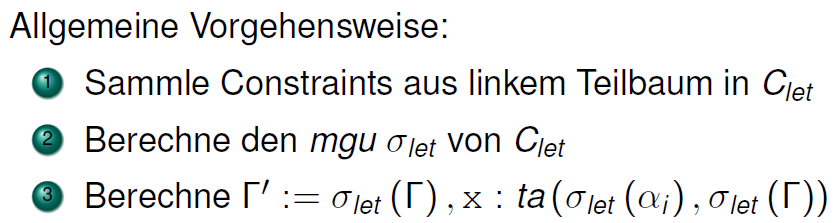
error();

}

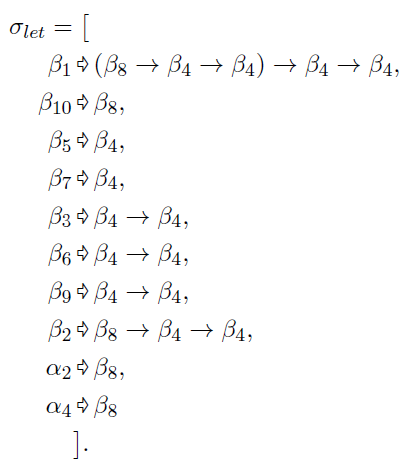
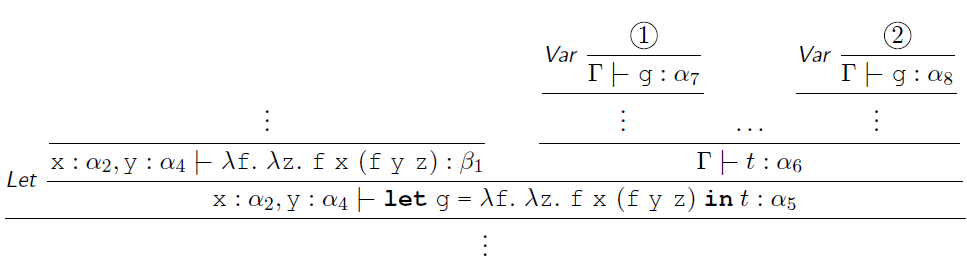
### Wie berechnet man rechte Г für Let?

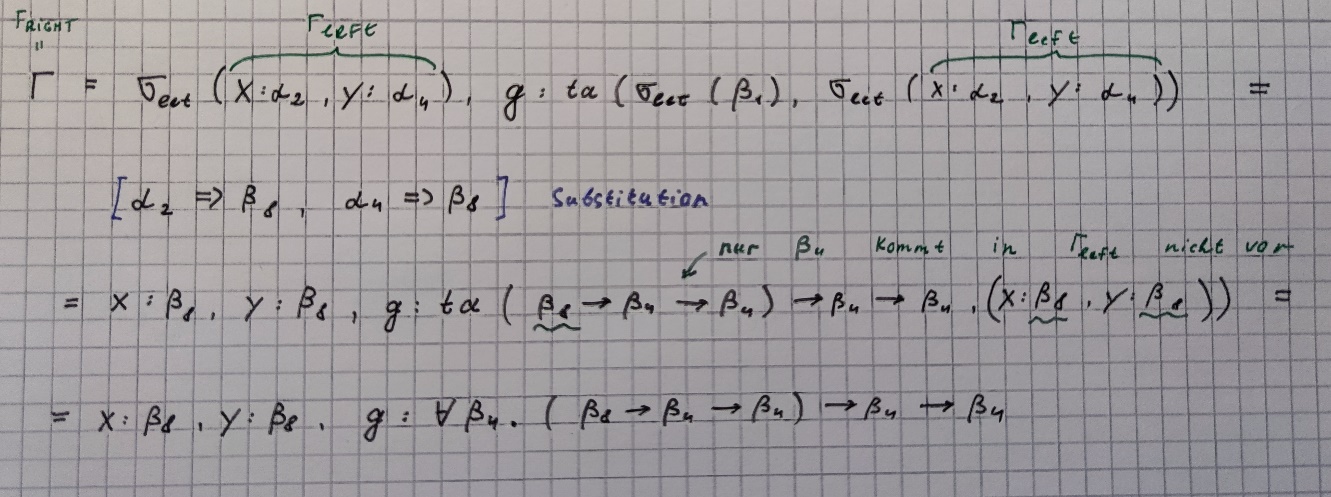
Gegeben: ein Baum mit LET





Beispiel:

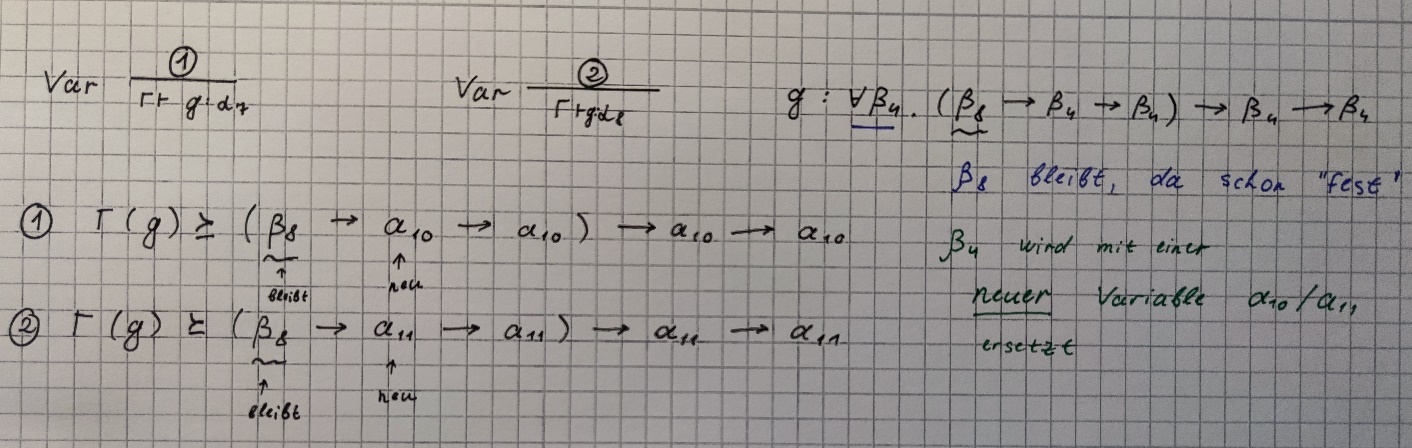




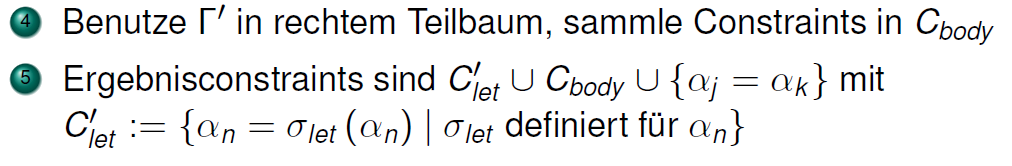
ta: nur die Typen bekommen ∀, die in Г\_left nicht vorkommen.

In Beispiel also nur B4, da B8 kommt vor.

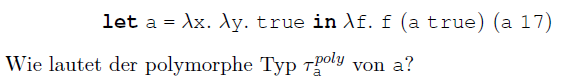
Wie bestimmt man 1 und 2 in diesem Fall?



Wie bestimmt man die ganze Constraintsmenge?



Wie bestimmt man Polymorphe typ von a, wenn es keine mgu gibt?

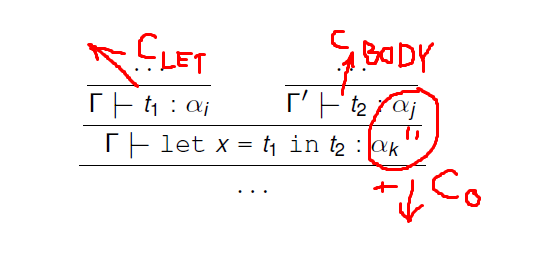


Fast immer mit „scharfem Hinsehen“

hier wäre es: 

(es werden zwei beliebige Variablen genommen und dann immer true zurückgegeben)

### C\_let, C\_body und C\_0



### Race Conditions

Accessing a **resource or shared memory** in parallel by different threads kann solche Situationen verursachen.

Critical sections protect – Shared memory, File access, Hardware access

**synchonized** Block: nur 1 thread kann in einem synchonized Block gleichzeitig befinden

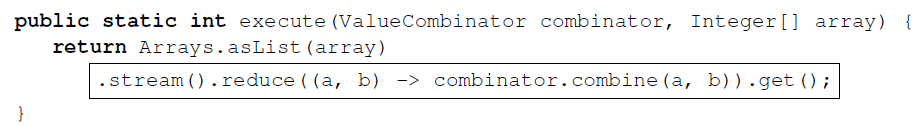
A simple way to avoid concurrency problems is to only share **immutable data**, which cannot be changed.

Defensive Copies (That copy will throw an exception if a modification is tried to be performed):  
public List<String> getSomeList() {

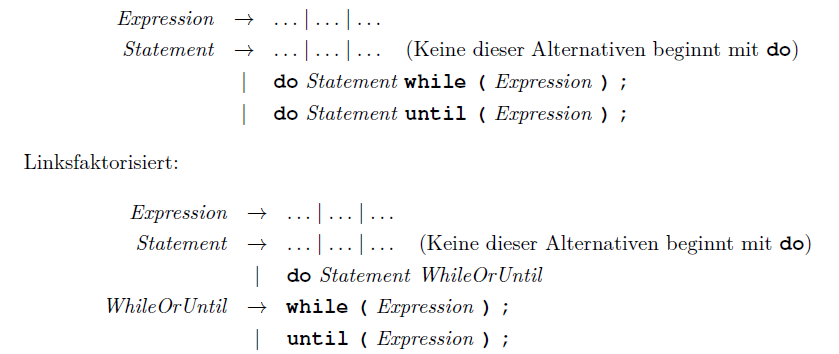
return java.util.Collections.unmodifiableList(someList);

}

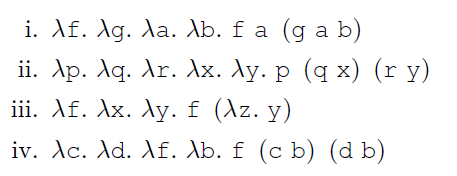
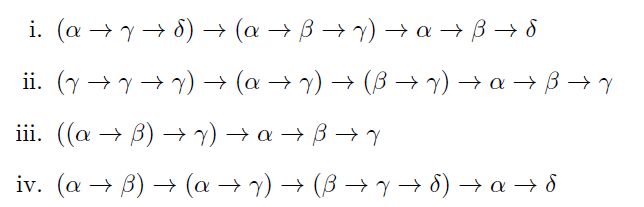
### Reduce mit Kombinator



### Linksfaktorisierung



### Lambda <--> Typ



 ist nicht typsierbar, da F -> \y. y ((\z.z) y) -> \y. y y

**Selbstapplikation ist nie typisierbar**