Notfallblatt

Komplizierte mgu

(c) Geben Sie einen allgemeinsten Unifikator für die folgende Constaintmenge in Listenform $[\ldots, \alpha_i \diamond \tau_i, \ldots]$ an.

$$C = \{$$

$$\alpha_5 = \alpha_4 \to \alpha_6,$$

$$\alpha_1 = \alpha_2 \to \alpha_3 \to \alpha_4,$$

$$\alpha_1 = \alpha_5 \to \alpha_6,$$

$$\}$$

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

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

Daraus folgt, dass:

2. Dann alles öffnen

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

$$a6 = a3 -> a4$$

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

3. Unifikator bilden:

CBN vs CBV

(a) Geben Sie T_3 vollständig eingesetzt an. Markieren Sie in T_3 den Redex, der unter CBV zuerst reduziert wird, und den Redex, der unter CBN zuerst reduziert wird. Beispiellösung:

$$T_3 = (\lambda a. a.a) ((\lambda a. a.a) ((\lambda a. a.a) (\lambda x. x)))$$

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

$$Var \ \frac{\Gamma_{sz}(\mathbf{s}) = \underline{\alpha_4}}{\Gamma_{sz} \models \mathbf{s} : \underline{\alpha_8}}$$

Regel: a4 = a8

$$C \frac{\mathsf{true} \in Const}{\Gamma, \mathsf{f} : \alpha_2 \mid \mathsf{-true} : \alpha_8}$$

Regel: a8 = bool (genau so mit int)

$$\Gamma_{sz}=\mathrm{s}:\alpha_4,\mathrm{z}:\alpha_6$$

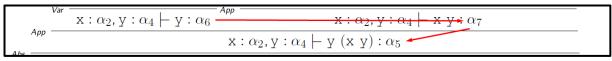
$$\Gamma_{tf}=\mathrm{t}:\alpha_{12},\mathrm{f}:\alpha_{14}$$

$$Abs = \frac{\Gamma_{s\underline{z}} \vdash s (s z) : \underline{\alpha_7}}{s : \alpha_4 \vdash (\underline{\lambda}z. \ s \ (s \ z)) : \underline{\alpha_5}}$$

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

```
Abs \begin{array}{c} \mathbf{x}:\alpha_{2} & \lambda\mathbf{y}.\ \mathbf{y}\ (\mathbf{x}\ \mathbf{y}) + \alpha_{3} \\ & -\lambda\mathbf{x}.\ \lambda\mathbf{y}.\ \mathbf{y}\ (\mathbf{x}\ \mathbf{y}):\alpha_{1} \end{array}
```

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. Terminal symbol 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 Γ für Let?

Gegeben: ein Baum mit LET

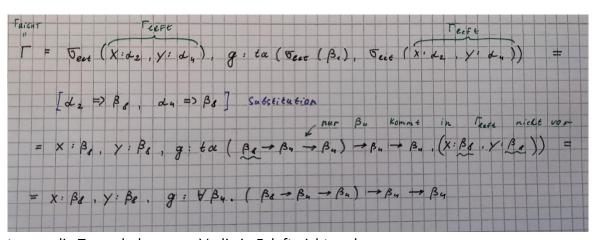
$$\frac{\Gamma \vdash t_1 : \alpha_i}{\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 : \alpha_k}$$

Allgemeine Vorgehensweise:

- Sammle Constraints aus linkem Teilbaum in C_{let}
- 2 Berechne den $mgu \sigma_{let}$ von C_{let}
- **3** Berechne $\Gamma' := \sigma_{let}(\Gamma), \mathbf{x} : ta(\sigma_{let}(\alpha_i), \sigma_{let}(\Gamma))$

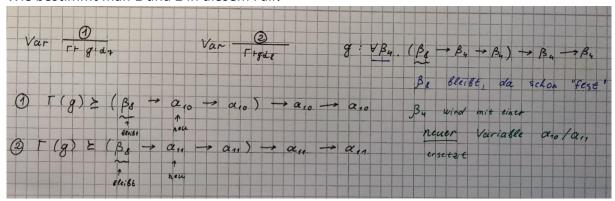
Beispiel:

$$\begin{array}{c} \sigma_{let} = \begin{bmatrix} \\ \beta_1 \diamond (\beta_8 \rightarrow \beta_4 \rightarrow \beta_4) \rightarrow \beta_4 \rightarrow \beta_4, \\ \beta_{10} \diamond \beta_8, \\ \beta_5 \diamond \beta_4, \\ \beta_7 \diamond \beta_4, \\ \beta_9 \diamond \beta_4 \rightarrow \beta_4, \\ \beta_9 \diamond \beta_4 \rightarrow \beta_4, \\ \beta_2 \diamond \beta_8 \rightarrow \beta_4 \rightarrow \beta_4, \\ \beta_2 \diamond \beta_8, \\ \alpha_4 \diamond \beta_8 \\ \end{bmatrix}.$$



ta: nur die Typen bekommen \forall , die in Γ _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?

- Φ Benutze Γ' in rechtem Teilbaum, sammle Constraints in C_{body}
- ⑤ Ergebnisconstraints sind $C'_{let} \cup C_{body} \cup \{\alpha_j = \alpha_k\}$ mit $C'_{let} := \{\alpha_n = \sigma_{let}(\alpha_n) \mid \sigma_{let} \text{ definiert für } \alpha_n\}$

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

let a =
$$\lambda$$
x. λ y. true **in** λ f. f (a true) (a 17)

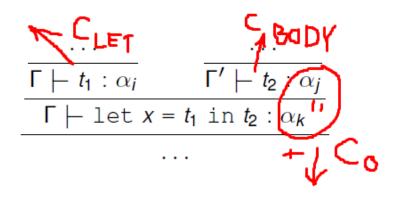
Wie lautet der polymorphe Typ τ_{a}^{poly} von a?

Fast immer mit "scharfem Hinsehen"

$$au^{poly}_{ t a} = orall lpha. orall eta. lpha
ightarrow eta
ightarrow \mathbf{bool}$$

(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

 $F = D \ E = (\lambda x. \ \lambda y. \ y \ (x \ y)) \ (\lambda z. \ z)$ ist nicht typsierbar, da $F \rightarrow y. \ y \ ((\z.z) \ y) \rightarrow y. \ y \ y$

Selbstapplikation ist nie typisierbar