Notfallblatt

Komplizierte mgu

(c) Geben Sie einen allgemeinsten Unifikator für die folgende Constaintmenge in Listenform $[\ldots, \alpha_i \Rrightarrow \tau_i, \ldots]$ an. [6 Punkte]

$$
\begin{aligned}
C = \{ & \\
& \alpha_5 = \alpha_4 \to \alpha_6, \\
& \alpha_1 = \alpha_2 \to \alpha_3 \to \alpha_4, \\
& \alpha_1 = \alpha_5 \to \alpha_6, \\
\} &
\end{aligned}
$$

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

(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. [2 Punkte]

Beispiellösung:

$$T_3 = \underbrace{\texttt{(}\lambda\texttt{a. a a) ((}\lambda\texttt{a. a a) (}\overbrace{\texttt{(}\lambda\texttt{a. a a) (}\lambda\texttt{x. x)}}^{\text{CBV}}\texttt{))}}_{\text{CBN}}$$

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

$$\text{Var}\ \frac{\Gamma_{sz}(\mathtt{s}) = \underline{\alpha_4}}{\Gamma_{sz} \vdash \mathtt{s} : \underline{\alpha_8}}$$

Regel: a4 = a8

$$C\ \frac{\mathtt{true} \in Const}{\Gamma, \mathtt{f} : \alpha_2 \vdash \mathtt{true} : \alpha_8}$$

Regel: a8 = bool (genau so mit int)

$$\Gamma_{sz} = \mathtt{s} : \alpha_4, \mathtt{z} : \alpha_6$$
$$\Gamma_{tf} = \mathtt{t} : \alpha_{12}, \mathtt{f} : \alpha_{14}$$

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

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

$$\text{Abs}\ \frac{\mathtt{x} : \alpha_2 \cancel{\vdash \lambda \mathtt{y}.\ \mathtt{y}\ (\mathtt{x}\ \mathtt{y}) :} \alpha_3}{\vdash \lambda \mathtt{x}.\ \lambda \mathtt{y}.\ \mathtt{y}\ (\mathtt{x}\ \mathtt{y}) : \alpha_1}$$

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

$$\text{App}\ \frac{\text{Var}\ \dfrac{}{\mathtt{x} : \alpha_2, \mathtt{y} : \alpha_4 \vdash \mathtt{y} : \alpha_6} \qquad \text{App}\ \dfrac{}{\cancel{\mathtt{x} : \alpha_2, \mathtt{y} : \alpha_4 \vdash \mathtt{x}\ \mathtt{y} :}\ \alpha_7}}{\mathtt{x} : \alpha_2, \mathtt{y} : \alpha_4 \vdash \mathtt{y}\ (\mathtt{x}\ \mathtt{y}) : \alpha_5}$$

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 Γ für Let?

Gegeben: ein Baum mit LET

$$\dfrac{\dfrac{\dots}{\Gamma \vdash t_1 : \alpha_i} \qquad \dfrac{\dots}{\Gamma' \vdash t_2 : \alpha_j}}{\dfrac{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ t_2 : \alpha_k}{\dots}}$$

Allgemeine Vorgehensweise:

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

Beispiel:

$$\textit{Let}\ \dfrac{\dfrac{\vdots}{\mathrm{x}:\alpha_2, \mathrm{y}:\alpha_4 \vdash \lambda \mathrm{f}.\ \lambda \mathrm{z}.\ \mathrm{f\ x\ (f\ y\ z)} : \beta_1} \qquad \dfrac{\dfrac{\textit{Var}\ \dfrac{①}{\Gamma \vdash \mathrm{g} : \alpha_7}}{\vdots} \quad \dots \quad \dfrac{\textit{Var}\ \dfrac{②}{\Gamma \vdash \mathrm{g} : \alpha_8}}{\vdots}}{\Gamma \vdash t : \alpha_6}}{\dfrac{\mathrm{x}:\alpha_2, \mathrm{y}:\alpha_4 \vdash \textbf{let}\ \mathrm{g} = \lambda \mathrm{f}.\ \lambda \mathrm{z}.\ \mathrm{f\ x\ (f\ y\ z)}\ \textbf{in}\ t : \alpha_5}{\vdots}}$$

$$\begin{aligned}
\sigma_{let} = [\ & \beta_1 \Leftrightarrow (\beta_8 \to \beta_4 \to \beta_4) \to \beta_4 \to \beta_4,\\
& \beta_{10} \Leftrightarrow \beta_8,\\
& \beta_5 \Leftrightarrow \beta_4,\\
& \beta_7 \Leftrightarrow \beta_4,\\
& \beta_3 \Leftrightarrow \beta_4 \to \beta_4,\\
& \beta_6 \Leftrightarrow \beta_4 \to \beta_4,\\
& \beta_9 \Leftrightarrow \beta_4 \to \beta_4,\\
& \beta_2 \Leftrightarrow \beta_8 \to \beta_4 \to \beta_4,\\
& \alpha_2 \Leftrightarrow \beta_8,\\
& \alpha_4 \Leftrightarrow \beta_8\\
&].
\end{aligned}$$

$$\underset{\Gamma_{RIGHT}}{\Gamma} = \sigma_{let}(\overbrace{x:\alpha_2, y:\alpha_4}^{\Gamma_{left}}),\ g : ta(\sigma_{let}(\beta_1), \sigma_{let}(\overbrace{x:\alpha_2, y:\alpha_4}^{\Gamma_{left}})) =$$

$[\alpha_2 \Rightarrow \beta_8,\ \alpha_4 \Rightarrow \beta_8]$ Substitution

nur β_4 kommt in Γ_{left} nicht vor

$$= x:\beta_8,\ y:\beta_8,\ g : ta((\beta_8 \to \beta_4 \to \beta_4) \to \beta_4 \to \beta_4, (x:\beta_8, y:\beta_8)) =$$

$$= x:\beta_8,\ y:\beta_8,\ g : \forall \beta_4.\ (\beta_8 \to \beta_4 \to \beta_4) \to \beta_4 \to \beta_4$$

ta: nur die Typen bekommen ∀, die in Γ_left nicht vorkommen.
In Beispiel also nur B4, da B8 kommt vor.

Wie bestimmt man 1 und 2 in diesem Fall?

$$\text{Var}\ \frac{①}{\Gamma \vdash g : \alpha_7} \qquad\qquad \text{Var}\ \frac{②}{\Gamma \vdash g : \alpha_8}$$

$g : \underline{\forall \beta_4}. (\beta_8 \to \beta_4 \to \beta_4) \to \beta_4 \to \beta_4$

β_8 bleibt, da schon "fest"

β_4 wird mit einer **neuer** Variable $\alpha_{10} / \alpha_{11}$ ersetzt

① $\Gamma(g) \succeq (\underbrace{\beta_8}_{\text{bleibt}} \to \underbrace{\alpha_{10}}_{\text{neu}} \to \alpha_{10}) \to \alpha_{10} \to \alpha_{10}$

② $\Gamma(g) \succeq (\underbrace{\beta_8}_{\text{bleibt}} \to \underbrace{\alpha_{11}}_{\text{neu}} \to \alpha_{11}) \to \alpha_{11} \to \alpha_{11}$

Wie bestimmt man die ganze Constraintsmenge?

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

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

```
let a = λx. λy. true in λf. f (a true) (a 17)
```

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

Fast immer mit „scharfem Hinsehen“

hier wäre es: $\tau_{\mathrm{a}}^{poly} = \forall\alpha.\forall\beta.\alpha \to \beta \to \mathbf{bool}$

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

C_let, C_body und C_0

$$\dfrac{\dfrac{\overset{C_{LET}}{\ldots}}{\Gamma \vdash t_1 : \alpha_i} \qquad \dfrac{\overset{C_{BODY}}{\ldots}}{\Gamma' \vdash t_2 : \alpha_j}}{\dfrac{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ t_2 : \alpha_k}{\ldots}}$$

$+\ C_0$

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

```
public static int execute(ValueCombinator combinator, Integer[] array) {
   return Arrays.asList(array)
      .stream().reduce((a, b) -> combinator.combine(a, b)).get();
}
```

Linksfaktorisierung

$$\begin{array}{rcll} \textit{Expression} & \to & \ldots \mid \ldots \mid \ldots & \\ \textit{Statement} & \to & \ldots \mid \ldots \mid \ldots & \text{(Keine dieser Alternativen beginnt mit } \texttt{do}\text{)} \\ & \mid & \texttt{do}\ \textit{Statement}\ \texttt{while (}\textit{Expression}\texttt{) ;} & \\ & \mid & \texttt{do}\ \textit{Statement}\ \texttt{until (}\textit{Expression}\texttt{) ;} & \end{array}$$

Linksfaktorisiert:

$$\begin{array}{rcll} \textit{Expression} & \to & \ldots \mid \ldots \mid \ldots & \\ \textit{Statement} & \to & \ldots \mid \ldots \mid \ldots & \text{(Keine dieser Alternativen beginnt mit } \texttt{do}\text{)} \\ & \mid & \texttt{do}\ \textit{Statement}\ \textit{WhileOrUntil} & \\ \textit{WhileOrUntil} & \to & \texttt{while (}\textit{Expression}\texttt{) ;} & \\ & \mid & \texttt{until (}\textit{Expression}\texttt{) ;} & \end{array}$$

Lambda <--> Typ

i. $(\alpha \to \gamma \to \delta) \to (\alpha \to \beta \to \gamma) \to \alpha \to \beta \to \delta$

ii. $(\gamma \to \gamma \to \gamma) \to (\alpha \to \gamma) \to (\beta \to \gamma) \to \alpha \to \beta \to \gamma$

iii. $((\alpha \to \beta) \to \gamma) \to \alpha \to \beta \to \gamma$

iv. $(\alpha \to \beta) \to (\alpha \to \gamma) \to (\beta \to \gamma \to \delta) \to \alpha \to \delta$

i. $\lambda f.\ \lambda g.\ \lambda a.\ \lambda b.\ f\ a\ (g\ a\ b)$

ii. $\lambda p.\ \lambda q.\ \lambda r.\ \lambda x.\ \lambda y.\ p\ (q\ x)\ (r\ y)$

iii. $\lambda f.\ \lambda x.\ \lambda y.\ f\ (\lambda z.\ y)$

iv. $\lambda c.\ \lambda d.\ \lambda f.\ \lambda b.\ f\ (c\ b)\ (d\ b)$

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

Selbstapplikation ist nie typisierbar