Tutorium 10: Übungsaufgaben

Paul Brinkmeier

8. Januar 2020

Tutorium Programmierparadigmen am KIT

Heutiges Programm

Programm

- Wiederholung Typinferenz
- Typinferenz mit Let
- Übungsaufgaben

Typinferenz

Wiederholung: Typinferenz

Vorgehensweise zur Typinferenz:

- Stelle Typherleitungsbaum auf
  - In jedem Schritt werden neue Typvariablen α_i angelegt
  - Statt die Typen direkt im Baum einzutragen, werden Gleichungen in einem Constraint-System eingetragen
- Unifiziere Constraint-System zu einem Unifikator
  - Robinson-Algorithmus, im Grunde wie bei Prolog

Robinson-Algorithmus

```
unify [] = []
unify [lhs = rhs | rest] =
  if lhs == rhs then unify rest
  if (lhs == Var a) and a not in fv(rhs):
    unify (apply [a => rhs] rest) ++ [a => rhs]
  if (rhs == Var a) and a not in fv(lhs):
    unify (apply [a => lhs] rest) ++ [a => lhs]
  if (lhs == a -> b) and (rhs == c -> d):
    unify (rest ++ [a = c, b = d])
  otherwise:
    fail
```

Erzeugt Unifikator zu einem Constraint-System.

Typinferenz: Übungsaufgabe

$$\frac{\dots}{\texttt{f} : \mathrm{int} \to \beta \vdash \lambda \texttt{x}.\texttt{f x} : \alpha_1}\textsc{Abs}$$

- „Finde den allgemeinsten Typen α_1 von $\lambda\texttt{x}.\texttt{f x}$"

Erinnerung:

- Baum mit durchnummerierten α_i aufstellen
- Constraints sammeln:

$$\frac{\Gamma(x) = \sigma \quad \sigma \succeq \iota}{\Gamma \vdash x : \tau}\textsc{Var} \qquad \frac{\Gamma \vdash f : \xi \quad \Gamma \vdash x : \phi}{\Gamma \vdash f\ x : \alpha}\textsc{App} \qquad \frac{\Gamma, p : \pi \vdash b : \beta}{\Gamma \vdash \lambda p.b : \alpha}\textsc{Abs}$$

Constraint: $\{\iota = \tau\}$ Constraint: $\{\xi = \phi \to \alpha\}$ Constraint: $\{\alpha = \pi \to \beta\}$

- Constraint-System auflösen (Robinson-Algorithmus)

Wozu brauchen wir Let?

- Findet den allgemeinsten Typen von α_1 von $(\lambda \texttt{x}.\texttt{x}\ \texttt{x})\ \lambda \texttt{y}.\texttt{y}$

Wozu brauchen wir Let?

- Findet den allgemeinsten Typen von α_1 von $(\lambda\mathtt{x}.\mathtt{x}\ \mathtt{x})\ \lambda\mathtt{y}.\mathtt{y}$
- $\rightsquigarrow$ Geht nicht, $\lambda\mathtt{y}.\mathtt{y}$ hat eine feste Struktur, kann nicht auf sich selbst angewendet werden
- Wir müssten $\mathtt{x}$ einen Typen geben, den man mehrmals verwenden kann: $\mathtt{x} : \forall\alpha.\alpha \to \alpha$

Wozu brauchen wir Let?

- Findet den allgemeinsten Typen von α_1 von $(\lambda \mathtt{x}.\mathtt{x}\ \mathtt{x})\ \lambda \mathtt{y}.\mathtt{y}$
- $\rightsquigarrow$ Geht nicht, $\lambda \mathtt{y}.\mathtt{y}$ hat eine feste Struktur, kann nicht auf sich selbst angewendet werden
- Wir müssten $\mathtt{x}$ einen Typen geben, den man mehrmals verwenden kann: $\mathtt{x} : \forall \alpha . \alpha \to \alpha$
- Dafür gibt es $\textsc{Let}$:

$$\frac{\Gamma \vdash y : \pi \quad \Gamma' \vdash b : \beta}{\Gamma \vdash \mathrm{let}\ x = y\ \mathrm{in}\ b : \tau}\textsc{Let}$$

- Mit den Constraints ist das hier etwas komplizierter:
  - Finde Unifikator $\sigma_{\textsc{Let}}$ und allg. Typ π für y
  - $\Gamma' = \sigma_{\textsc{Let}}(\Gamma)\ ,\ x : ta(\sigma_{\textsc{Let}}(\pi), \sigma_{\textsc{Let}}(\Gamma))$
  - $ta(\tau, \Gamma)$ bindet alle in Γ freien Typvariablen mit einem $\forall$ in τ
  - Bspw. $ta(\alpha \to \beta, x : \beta, y : \delta) = \forall \alpha . \alpha \to \mathrm{int}$