Typinferenz

Proseminar WS 03/04

Christian Kersten
Betreuer: Andreas Rossberg

Gert Smolka
PS-LAB
Universität des Saarlandes

Typinferenz

Einleitung

Typchecker sind ein wichtiges Mittel, Fehler früh zu erkennen

- **Einfache Typchecker** benötigen dazu sog. **Typannotationen**:

Typinferenz

Einleitung

Typchecker sind ein wichtiges Mittel, Fehler früh zu erkennen

- **Einfache Typchecker** benötigen dazu sog. **Typannotationen**:

  Bsp.: $(\mathsf{fn}\ (x : \mathsf{int}) \Rightarrow fn\ (y : \mathsf{bool}) \Rightarrow \mathsf{if}\ y\ \mathsf{then}\ x\ \mathsf{else}\ 0)\ \ 5\ \ 6$

Typinferenz

Einleitung

Typchecker sind ein wichtiges Mittel, Fehler früh zu erkennen

- **Einfache Typchecker** benötigen dazu sog. **Typannotationen**:

  Bsp.: $(\mathsf{fn}\ (x : \mathsf{int}) \Rightarrow fn\ (y : \underbrace{\mathsf{bool}}) \Rightarrow \mathsf{if}\ y\ \mathsf{then}\ x\ \mathsf{else}\ 0)\ \ 5\ \ \underbrace{6}$

Typinferenz

Einleitung

Typchecker sind ein wichtiges Mittel, Fehler früh zu erkennen

- **Einfache Typchecker** benötigen dazu sog. **Typannotationen**:

  Bsp.: $(\mathsf{fn}\ (x : \mathsf{int}) \Rightarrow fn\ (y : \underbrace{\mathsf{bool}}) \Rightarrow \mathsf{if}\ y\ \mathsf{then}\ x\ \mathsf{else}\ 0)\ \ 5\ \ \underbrace{6}$

- **Vorteil:** leicht zu programmieren und funktioniert

Typinferenz

Einleitung

Typchecker sind ein wichtiges Mittel, Fehler früh zu erkennen

- **Einfache Typchecker** benötigen dazu sog. **Typannotationen**:

  Bsp.: $(\mathsf{fn}\ (x : \mathsf{int}) \Rightarrow fn\ (y : \underbrace{\mathsf{bool}}) \Rightarrow \mathsf{if}\ y\ \mathsf{then}\ x\ \mathsf{else}\ 0)\ \ 5\ \ \underbrace{6}$

- **Vorteil:** leicht zu programmieren und funktioniert
- **Nachteil:** zusätzliche Arbeit für Programmierer

Typinferenz

Einleitung

Typchecker sind ein wichtiges Mittel, Fehler früh zu erkennen

- **Einfache Typchecker** benötigen dazu sog. **Typannotationen**:

  Bsp.: $(\mathsf{fn}\ (x : \mathsf{int}) \Rightarrow fn\ (y : \underbrace{\mathsf{bool}}) \Rightarrow \mathsf{if}\ y\ \mathsf{then}\ x\ \mathsf{else}\ 0)\ \ 5\ \ \underbrace{6}$

- **Vorteil:** leicht zu programmieren und funktioniert
- **Nachteil:** zusätzliche Arbeit für Programmierer
- **Ziel:** Typchecker mit gleicher Funktionalität jedoch ohne Typannotationen

Typinferenz

Einleitung

Typchecker sind ein wichtiges Mittel, Fehler früh zu erkennen

- **Einfache Typchecker** benötigen dazu sog. **Typannotationen**:

  Bsp.: $(\mathsf{fn}\ (x : \mathsf{int}) \Rightarrow fn\ (y : \underbrace{\mathsf{bool}}) \Rightarrow \mathsf{if}\ y\ \mathsf{then}\ x\ \mathsf{else}\ 0)\ \ 5\ \ \underbrace{6}$

- **Vorteil:** leicht zu programmieren und funktioniert
- **Nachteil:** zusätzliche Arbeit für Programmierer
- **Ziel:** Typchecker mit gleicher Funktionalität jedoch ohne Typannotationen

Damit beschäftigt sich **Typinferenz** oder **Typrekonstruktion**

Vorlage: B.C.Pierce *„Types and Programming Languages“* Kapitel 22

Typinferenz

Einleitung

Typchecker sind ein wichtiges Mittel, Fehler früh zu erkennen

- **Einfache Typchecker** benötigen dazu sog. **Typannotationen**:

  Bsp.: $(\mathsf{fn}\ (x : \mathsf{int}) \Rightarrow fn\ (y : \underbrace{\mathsf{bool}}) \Rightarrow \mathsf{if}\ y\ \mathsf{then}\ x\ \mathsf{else}\ 0)\ \ 5\ \ \underbrace{6}$

- **Vorteil:** leicht zu programmieren und funktioniert
- **Nachteil:** zusätzliche Arbeit für Programmierer
- **Ziel:** Typchecker mit gleicher Funktionalität jedoch ohne Typannotationen

Damit beschäftigt sich **Typinferenz** oder **Typrekonstruktion**
Vorlage: B.C.Pierce *„Types and Programming Languages“* Kapitel 22

Bem.: Typinferenz findet man z.B. in ML oder Haskell

Typinferenz

Grundidee:

- **Gegeben sei Term:**

  $\mathsf{t} := \mathsf{fn}\ f \Rightarrow \mathsf{fn}\ x \Rightarrow f(f(x))$

Typinferenz

Grundidee:

- **Gegeben sei Term:**

  $\mathsf{t} := \mathsf{fn}\ f \Rightarrow \mathsf{fn}\ x \Rightarrow f(f(x))$

- **Auffüllen fehlender Annotationen:**

  $\mathsf{t} := \mathsf{fn}\ f : \alpha \Rightarrow \mathsf{fn}\ x : \beta \Rightarrow f(f(x))$

Typinferenz

Grundidee:

- **Gegeben sei Term:**

  $\mathsf{t} := \mathsf{fn}\ f \Rightarrow \mathsf{fn}\ x \Rightarrow f(f(x))$

- **Auffüllen fehlender Annotationen:**

  $\mathsf{t} := \mathsf{fn}\ f : \alpha \Rightarrow \mathsf{fn}\ x : \beta \Rightarrow f(f(x))$

- **Frage:** Kann man eine Substitution σ finden, dass t typisierbar wird?

Typinferenz

Grundidee:

- **Gegeben sei Term:**

  $\mathsf{t} := \mathsf{fn}\ f \Rightarrow \mathsf{fn}\ x \Rightarrow f(f(x))$

- **Auffüllen fehlender Annotationen:**

  $\mathsf{t} := \mathsf{fn}\ f : \alpha \Rightarrow \mathsf{fn}\ x : \beta \Rightarrow f(f(x))$

- **Frage:** Kann man eine Substitution σ finden, dass t typisierbar wird?

- **Lösung:** $\sigma = [\alpha \mapsto \mathsf{Int} \rightarrow \mathsf{Int}, \beta \mapsto \mathsf{Int}]$

Typinferenz

Grundidee:

- **Gegeben sei Term:**

  $\mathsf{t} := \mathsf{fn}\ f \Rightarrow \mathsf{fn}\ x \Rightarrow f(f(x))$

- **Auffüllen fehlender Annotationen:**

  $\mathsf{t} := \mathsf{fn}\ f : \alpha \Rightarrow \mathsf{fn}\ x : \beta \Rightarrow f(f(x))$

- **Frage:** Kann man eine Substitution σ finden, dass t typisierbar wird?

- **Lösung:** $\sigma = [\alpha \mapsto \mathsf{Int} \rightarrow \mathsf{Int}, \beta \mapsto \mathsf{Int}]$

  $\Rightarrow \sigma\mathsf{t} = \mathsf{fn}\ (f : \mathsf{Int} \rightarrow \mathsf{Int})\mathsf{fn}\ (x : \mathsf{Int}) \Rightarrow f(f(x))$

Typinferenz

Grundidee:

- **Gegeben sei Term:**

  $\mathsf{t} := \mathsf{fn}\ f \Rightarrow \mathsf{fn}\ x \Rightarrow f(f(x))$

- **Auffüllen fehlender Annotationen:**

  $\mathsf{t} := \mathsf{fn}\ f : \alpha \Rightarrow \mathsf{fn}\ x : \beta \Rightarrow f(f(x))$

- **Frage:** Kann man eine Substitution σ finden, dass t typisierbar wird?

- **Lösung:** $\sigma = [\alpha \mapsto \mathsf{Int} \rightarrow \mathsf{Int}, \beta \mapsto \mathsf{Int}]$

  $\Rightarrow \sigma\mathsf{t} = \mathsf{fn}\ (f : \mathsf{Int} \rightarrow \mathsf{Int})\mathsf{fn}\ (x : \mathsf{Int}) \Rightarrow f(f(x))$

  $\sigma\mathsf{t} : (\mathsf{Int} \rightarrow \mathsf{Int}) \rightarrow \mathsf{Int} \rightarrow \mathsf{Int}$

Typinferenz

Grundlagen:

Der einfach getypte Lambda-Kalkül

Typinferenz

Grundlagen:

Der einfach getypte Lambda-Kalkül

- **Terme:** $\mathsf{t} ::= x \mid \lambda x.\mathsf{t} \mid \mathsf{t}\ \mathsf{t} \mid \mathsf{succ\ t} \mid \mathsf{pred\ t} \mid \mathsf{iszero\ t} \mid$
  $0 \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if\ t\ then\ t\ else\ t}$

Typinferenz

Grundlagen:

Der einfach getypte Lambda-Kalkül

- **Terme:** $\mathsf{t} ::= x \mid \lambda x.\mathsf{t} \mid \mathsf{t}\ \mathsf{t} \mid \mathsf{succ\ t} \mid \mathsf{pred\ t} \mid \mathsf{iszero\ t} \mid 0 \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if\ t\ then\ t\ else\ t}$
- **Typen:** $\mathsf{T} ::= \mathsf{X} \mid \mathsf{T} \to \mathsf{T} \mid \mathsf{Nat} \mid \mathsf{Bool}$

Typinferenz

Typ-Regeln(1)

Ein term $\mathtt{t}$ in einem Kontext Γ ist genau dann typisierbar, wenn es T gibt sodass gilt: $\Gamma \vdash \mathtt{t} : \mathsf{T}$

Typinferenz

Typ-Regeln(1)

Ein term t in einem Kontext Γ ist genau dann typisierbar, wenn es T gibt sodass gilt: $\Gamma \vdash \mathsf{t} : \mathsf{T}$

$$\frac{x{:}\mathsf{T} \in \Gamma}{\Gamma \vdash x{:}\mathsf{T}} \quad (\text{T-VAR})$$

Typinferenz

Typ-Regeln(1)

Ein term t in einem Kontext Γ ist genau dann typisierbar, wenn es T gibt sodass gilt: $\Gamma \vdash \mathsf{t} : \mathsf{T}$

$$\frac{x{:}\mathsf{T} \in \Gamma}{\Gamma \vdash x{:}\mathsf{T}} \quad (\text{T-VAR}) \qquad \frac{\Gamma' \vdash \mathsf{t}_2{:}\mathsf{T}' \quad \Gamma' = \Gamma, x{:}\mathsf{T}}{\Gamma \vdash \lambda\, x{:}\mathsf{T}.\, \mathsf{t}_2{:}\mathsf{T} \to \mathsf{T}'} \quad (\text{T-ABS})$$

Typinferenz

Typ-Regeln(1)

Ein term t in einem Kontext Γ ist genau dann typisierbar, wenn es T gibt sodass gilt: $\Gamma \vdash \mathsf{t} : \mathsf{T}$

$$\frac{x{:}\mathsf{T} \in \Gamma}{\Gamma \vdash x{:}\mathsf{T}} \quad (\text{T-VAR}) \qquad \frac{\Gamma' \vdash \mathsf{t}_2{:}\mathsf{T}' \quad \Gamma' = \Gamma, x{:}\mathsf{T}}{\Gamma \vdash \lambda\, x{:}\mathsf{T}.\, \mathsf{t}_2 : \mathsf{T} \to \mathsf{T}'} \quad (\text{T-ABS})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T} \to \mathsf{T}' \quad \Gamma \vdash \mathsf{t}_2 : \mathsf{T}}{\Gamma \vdash \mathsf{t}_1\, \mathsf{t}_2 : \mathsf{T}'} \quad (\text{T-APP})$$

Typinferenz

Typ-Regeln(1)

Ein term t in einem Kontext Γ ist genau dann typisierbar, wenn es T gibt sodass gilt: $\Gamma \vdash \mathsf{t} : \mathsf{T}$

$$\frac{x{:}\mathsf{T} \in \Gamma}{\Gamma \vdash x{:}\mathsf{T}} \quad (\text{T-VAR}) \qquad \frac{\Gamma' \vdash \mathsf{t}_2{:}\mathsf{T}' \quad \Gamma' = \Gamma, x{:}\mathsf{T}}{\Gamma \vdash \lambda\, x{:}\mathsf{T}.\, \mathsf{t}_2{:}\mathsf{T} \to \mathsf{T}'} \quad (\text{T-ABS})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T} \to \mathsf{T}' \quad \Gamma \vdash \mathsf{t}_2 : \mathsf{T}}{\Gamma \vdash \mathsf{t}_1\, \mathsf{t}_2 : \mathsf{T}'} \quad (\text{T-APP})$$

$$\Gamma \vdash 0 : \mathsf{Nat} \quad (\text{CT-ZERO})$$

Typinferenz

Typ-Regeln(1)

Ein term t in einem Kontext Γ ist genau dann typisierbar, wenn es T gibt sodass gilt: $\Gamma \vdash \mathsf{t} : \mathsf{T}$

$$\frac{x{:}\mathsf{T} \in \Gamma}{\Gamma \vdash x{:}\mathsf{T}} \quad (\text{T-VAR}) \qquad \frac{\Gamma' \vdash \mathsf{t}_2{:}\mathsf{T}' \quad \Gamma' = \Gamma, x{:}\mathsf{T}}{\Gamma \vdash \lambda\, x{:}\mathsf{T}.\, \mathsf{t}_2{:}\mathsf{T} \to \mathsf{T}'} \quad (\text{T-ABS})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T} \to \mathsf{T}' \quad \Gamma \vdash \mathsf{t}_2 : \mathsf{T}}{\Gamma \vdash \mathsf{t}_1\, \mathsf{t}_2 : \mathsf{T}'} \quad (\text{T-APP})$$

$$\Gamma \vdash 0 : \mathsf{Nat} \quad (\text{CT-ZERO}) \qquad \frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\, \mathsf{t}_1 : \mathsf{Nat}} \quad (\text{CT-SUCC})$$

Typinferenz

Typregeln(2)

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{Nat}}{\Gamma \vdash \mathsf{pred}\ \mathsf{t}_1 : \mathsf{Nat}} \quad (\text{CT-PRED})$$

Typinferenz

Typregeln(2)

$$\frac{\Gamma \vdash t_1 : \mathsf{Nat}}{\Gamma \vdash \mathsf{pred}\ t_1 : \mathsf{Nat}} \quad (\text{CT-Pred}) \qquad \frac{\Gamma \vdash t_1 : \mathsf{Nat}}{\Gamma \vdash \mathsf{iszero}\ t_1 : \mathsf{Bool}} \quad (\text{CT-IsZero})$$

Typinferenz

Typregeln(2)

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{Nat}}{\Gamma \vdash \mathsf{pred}\ \mathsf{t}_1 : \mathsf{Nat}} \quad (\text{CT-Pred}) \qquad \frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{Nat}}{\Gamma \vdash \mathsf{iszero}\ \mathsf{t}_1 : \mathsf{Bool}} \quad (\text{CT-IsZero})$$

$$\Gamma \vdash \mathit{true} : \mathsf{Bool} \quad (\text{CT-True})$$

Typinferenz

Typregeln(2)

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{Nat}}{\Gamma \vdash \mathsf{pred}\ \mathsf{t}_1 : \mathsf{Nat}} \quad (\text{CT-Pred}) \qquad \frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{Nat}}{\Gamma \vdash \mathsf{iszero}\ \mathsf{t}_1 : \mathsf{Bool}} \quad (\text{CT-IsZero})$$

$$\Gamma \vdash \mathit{true} : \mathsf{Bool} \quad (\text{CT-True}) \qquad \Gamma \vdash \mathit{false} : \mathsf{Bool} \quad (\text{CT-False})$$

Typinferenz

Typregeln(2)

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{Nat}}{\Gamma \vdash \mathsf{pred}\ \mathsf{t}_1 : \mathsf{Nat}}\ (\text{CT-Pred}) \qquad \frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{Nat}}{\Gamma \vdash \mathsf{iszero}\ \mathsf{t}_1 : \mathsf{Bool}}\ (\text{CT-IsZero})$$

$$\Gamma \vdash \mathit{true} : \mathsf{Bool}\quad (\text{CT-True}) \qquad \Gamma \vdash \mathit{false} : \mathsf{Bool}\quad (\text{CT-False})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{Bool} \qquad \Gamma \vdash \mathsf{t}_2 : \mathsf{T} \quad \Gamma \vdash \mathsf{t}_3 : \mathsf{T}}{\Gamma \vdash \mathsf{if}\ \mathsf{t}_1\ \mathsf{then}\ \mathsf{t}_2\ \mathsf{else}\ \mathsf{t}_3 : \mathsf{T}}\ (\text{CT-If})$$

Typinferenz

Zurück zu Typinferenz

- Gegeben (Γ, t) stellt sich die Frage:

Typinferenz

Zurück zu Typinferenz

- Gegeben (Γ, t) stellt sich die Frage:
  1. Gilt für alle Substitutionen σ:

$$\exists T : \sigma\Gamma \vdash \sigma t : \mathsf{T}$$

Typinferenz

Zurück zu Typinferenz

- Gegeben (Γ, t) stellt sich die Frage:
  1. Gilt für alle Substitutionen σ:

     $$\exists T : \sigma\Gamma \vdash \sigma t : \mathsf{T}$$

  2. Gibt es überhaupt einen Substitution σ:

     $$\exists T : \sigma\Gamma \vdash \sigma t : \mathsf{T}$$

Typinferenz

Zurück zu Typinferenz

- Gegeben (Γ, t) stellt sich die Frage:
  1. Gilt für alle Substitutionen σ:

     $$\exists T : \sigma\Gamma \vdash \sigma t : \mathsf{T}$$

  2. Gibt es überhaupt einen Substitution σ:

     $$\exists T : \sigma\Gamma \vdash \sigma t : \mathsf{T}$$

- **Definition:** Sei Γ ein Kontext und t ein Term, dann nennt man (σ, T) **Lösung**, wenn gilt: $\sigma\Gamma \vdash \sigma t : \mathsf{T}$

Typinferenz

- **Beispiel zu 1:**

Typinferenz

- **Beispiel zu 1:**

$$\lambda f : X \to X.\ \lambda x : X.\ f(f(x))$$

Typinferenz

- **Beispiel zu 1:**

$$\lambda f : X \to X.\ \lambda x : X.\ f(f(x))$$

Dieser Term ist ohne jegliche Einschränkung typisierbar!

Typinferenz

- **Beispiel zu 1:**

  $$\lambda f : X \to X.\ \lambda x : X.\ f(f(x))$$

  Dieser Term ist ohne jegliche Einschränkung typisierbar!

- **Beispiel zu 2:**

  $$\lambda f : Y.\ \lambda x : X.\ f(f(x))$$

Typinferenz

- **Beispiel zu 1:**

  $$\lambda f : X \to X.\ \lambda x : X.\ f(f(x))$$

  Dieser Term ist ohne jegliche Einschränkung typisierbar!

- **Beispiel zu 2:**

  $$\lambda f : Y.\ \lambda x : X.\ f(f(x))$$

  Dieser Term ist nur typisierbar, wenn σ sogenannte **Constraints** erfüllt .
  Mögliche Löungen wären: $\sigma_1 = [Y \mapsto X \to X]$
  $\sigma_2 = [Y \mapsto \mathsf{Nat} \to \mathsf{Nat}, X \mapsto \mathsf{Nat}]$

Typinferenz

Constraint-Based Typing

Bedeutung für die Lösung eines Typisierungsproblems?

Typinferenz

Constraint-Based Typing

Bedeutung für die Lösung eines Typisierungsproblems?

- Lösung ein zwei Schritten:

Typinferenz

Constraint-Based Typing

Bedeutung für die Lösung eines Typisierungsproblems?

- Lösung ein zwei Schritten:

  1. **Bestimmung der Constraints** bzw. Constraintmenge, die eine mögliche Lösung erfüllen muss

Typinferenz

Constraint-Based Typing

Bedeutung für die Lösung eines Typisierungsproblems?

- Lösung ein zwei Schritten:
  1. **Bestimmung der Constraints** bzw. Constraintmenge, die eine mögliche Lösung erfüllen muss
  2. **Herleitung einer Lösung** aus der gerade bestimmten Constraintmenge

Typinferenz

Constraint-Based Typing

- **Definition:**
  Eine **Constraintmenge** C ist eine Menge von Gleichungen.
  σ erfüllt Constraint S $=$ T genau dann, wenn gilt: $\sigma S = \sigma$T.
  σ erfüllt C genau dann, wenn sie jede Gleichung in C erfüllt.

Typinferenz

Constraint-Based Typing

- **Definition:**
  Eine **Constraintmenge** C ist eine Menge von Gleichungen.
  σ erfüllt Constraint $\mathsf{S} = \mathsf{T}$ genau dann, wenn gilt: $\sigma\mathsf{S} = \sigma\mathsf{T}$.
  σ erfüllt C genau dann, wenn sie jede Gleichung in C erfüllt.

- **Ziel:** Algorithmus, der (Γ, t) nimmt und (C, S) berechnet, sodass gilt:

$$\Gamma \vdash t : \mathsf{S}|_X C$$

  d.h. sei (σ, T) Lösung, dann muss σ Constraintmenge C erfüllen.

Typinferenz

Constraint-Based Typing

- **Definition:**
  Eine **Constraintmenge** C ist eine Menge von Gleichungen.
  σ erfüllt Constraint $\mathsf{S} = \mathsf{T}$ genau dann, wenn gilt: $\sigma\mathsf{S} = \sigma\mathsf{T}$.
  σ erfüllt C genau dann, wenn sie jede Gleichung in C erfüllt.

- **Ziel:** Algorithmus, der (Γ, t) nimmt und (C, S) berechnet, sodass gilt:

$$\Gamma \vdash t : \mathsf{S}|_X C$$

  d.h. sei (σ, T) Lösung, dann muss σ Constraintmenge C erfüllen.

- **Bem:** Wir nennen $(\Gamma, t, \mathsf{S}, C)$ dann auch **Constraintproblem**

Typinferenz

Constraint Basierte Typisierung(1)

$$\frac{x : \mathsf{T} \in \Gamma}{\Gamma \vdash x : \mathsf{T}|_{\emptyset}\{\}} \quad (\text{CT-VAR})$$

Typinferenz

Constraint Basierte Typisierung(1)

$$\frac{x : \mathsf{T} \in \Gamma}{\Gamma \vdash x : \mathsf{T} | _{\emptyset} \{\}} \quad (\text{CT-VAR}) \qquad \frac{\Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}} C}{\Gamma \vdash \lambda\, x : \mathsf{T}_1 \,.\, \mathsf{t}_2 : \mathsf{T}_1 \to \mathsf{T}_2 | _{\mathcal{X}} C} \quad (\text{CT-ABS})$$

Typinferenz

Constraint Basierte Typisierung(1)

$$\frac{x : \mathsf{T} \in \Gamma}{\Gamma \vdash x : \mathsf{T} | _{\emptyset} \{\}} \quad (\text{CT-Var}) \qquad \frac{\Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}} C}{\Gamma \vdash \lambda\, x : \mathsf{T}_1 \,.\, \mathsf{t}_2 : \mathsf{T}_1 \to \mathsf{T}_2 | _{\mathcal{X}} C} \quad (\text{CT-Abs})$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 | _{\mathcal{X}_1} C_1 \quad \Gamma \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_2} C_2 \\ C' = C_1 \cup C_2 \cup \{\mathsf{T}_1 = \mathsf{T}_2 \to X\} \qquad X \text{ neu}\end{array}}{\Gamma \vdash \mathsf{t}_1\, \mathsf{t}_2 : X | _{\mathcal{X}_1 \cup \mathcal{X}_2 \cup \{X\}} C'} \quad (\text{CT-App})$$

Typinferenz

Constraint Basierte Typisierung(1)

$$\frac{x : \mathsf{T} \in \Gamma}{\Gamma \vdash x : \mathsf{T} | _{\emptyset} \{\}} \quad (\text{CT-VAR}) \qquad \frac{\Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}} C}{\Gamma \vdash \lambda\, x : \mathsf{T}_1 \,.\, \mathsf{t}_2 : \mathsf{T}_1 \to \mathsf{T}_2 | _{\mathcal{X}} C} \quad (\text{CT-ABS})$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 | _{\mathcal{X}_1} C_1 \quad \Gamma \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_2} C_2 \\ C' = C_1 \cup C_2 \cup \{\mathsf{T}_1 = \mathsf{T}_2 \to X\} \qquad X \text{ neu}\end{array}}{\Gamma \vdash \mathsf{t}_1\, \mathsf{t}_2 : X | _{\mathcal{X}_1 \cup \mathcal{X}_2 \cup \{X\}} C'} \quad (\text{CT-APP})$$

$$\Gamma \vdash 0 : \mathsf{Nat} | _{\emptyset} \{\} \quad (\text{CT-ZERO})$$

Typinferenz

Constraint Basierte Typisierung(1)

$$\frac{x : \mathsf{T} \in \Gamma}{\Gamma \vdash x : \mathsf{T}|_{\emptyset}\{\}} \quad (\text{CT-VAR}) \qquad \frac{\Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2|_{\mathcal{X}} C}{\Gamma \vdash \lambda\, x : \mathsf{T}_1 \,.\, \mathsf{t}_2 : \mathsf{T}_1 \to \mathsf{T}_2|_{\mathcal{X}} C} \quad (\text{CT-ABS})$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1|_{\mathcal{X}_1} C_1 \quad \Gamma \vdash \mathsf{t}_2 : \mathsf{T}_2|_{\mathcal{X}_2} C_2 \\ C' = C_1 \cup C_2 \cup \{\mathsf{T}_1 = \mathsf{T}_2 \to X\} \qquad X \text{ neu}\end{array}}{\Gamma \vdash \mathsf{t}_1\, \mathsf{t}_2 : X|_{\mathcal{X}_1 \cup \mathcal{X}_2 \cup \{X\}} C'} \quad (\text{CT-APP})$$

$$\Gamma \vdash 0 : \mathsf{Nat}|_{\emptyset}\{\} \quad (\text{CT-ZERO}) \qquad \frac{\begin{array}{c}\Gamma \vdash \mathsf{t}_1 : \mathsf{T}|_{\mathcal{X}} C \\ C' = C \cup \{\mathsf{T} = \mathsf{Nat}\}\end{array}}{\Gamma \vdash \mathsf{succ}\, \mathsf{t}_1 : \mathsf{Nat}|_{\mathcal{X}} C'} \quad (\text{CT-SUCC})$$

Typinferenz

Constraint-Based Typing Inferenzregeln (2)

$$\frac{\begin{array}{c}\Gamma \vdash \mathsf{t}_1 : \mathsf{T} | _{\mathcal{X}} C \\ C' = C \cup \{\mathsf{T} = \mathsf{Nat}\}\end{array}}{\Gamma \vdash \mathsf{pred}\ \mathsf{t}_1 : \mathsf{Nat} | _{\mathcal{X}} C'} \quad (\text{CT-PRED})$$

Typinferenz

Constraint-Based Typing Inferenzregeln (2)

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T} | _{\mathcal{X}} C \qquad C' = C \cup \{\mathsf{T} = \mathsf{Nat}\}}{\Gamma \vdash \mathsf{pred}\ \mathsf{t}_1 : \mathsf{Nat} | _{\mathcal{X}} C'} \quad (\text{CT-PRED}) \qquad \frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T} | _{\mathcal{X}} C \qquad C' = C \cup \{\mathsf{T} = \mathsf{Nat}\}}{\Gamma \vdash \mathsf{iszero}\ \mathsf{t}_1 : \mathsf{Bool} | _{\mathcal{X}} C'} \quad (\text{CT-ISZER}$$

Typinferenz

Constraint-Based Typing Inferenzregeln (2)

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T} | _{\mathcal{X}} C \qquad C' = C \cup \{\mathsf{T} = \mathsf{Nat}\}}{\Gamma \vdash \mathsf{pred}\ \mathsf{t}_1 : \mathsf{Nat} | _{\mathcal{X}} C'} \quad (\text{CT-PRED})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T} | _{\mathcal{X}} C \qquad C' = C \cup \{\mathsf{T} = \mathsf{Nat}\}}{\Gamma \vdash \mathsf{iszero}\ \mathsf{t}_1 : \mathsf{Bool} | _{\mathcal{X}} C'} \quad (\text{CT-ISZER}$$

$$\Gamma \vdash \mathit{true} : \mathsf{Bool} | _{\emptyset} \{\} \quad (\text{CT-TRUE})$$

Typinferenz

Constraint-Based Typing Inferenzregeln (2)

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T} | _{\mathcal{X}} C \qquad C' = C \cup \{\mathsf{T} = \mathsf{Nat}\}}{\Gamma \vdash \mathsf{pred}\ \mathsf{t}_1 : \mathsf{Nat} | _{\mathcal{X}} C'} \quad (\text{CT-PRED}) \qquad \frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T} | _{\mathcal{X}} C \qquad C' = C \cup \{\mathsf{T} = \mathsf{Nat}\}}{\Gamma \vdash \mathsf{iszero}\ \mathsf{t}_1 : \mathsf{Bool} | _{\mathcal{X}} C'} \quad (\text{CT-ISZER}$$

$$\Gamma \vdash \mathit{true} : \mathsf{Bool} | _{\emptyset} \{\} \quad (\text{CT-TRUE}) \qquad \Gamma \vdash \mathit{false} : \mathsf{Bool} | _{\emptyset} \{\} \quad (\text{CT-FALSE})$$

Typinferenz

Constraint-Based Typing Inferenzregeln (2)

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T} | _{\mathcal{X}} C \qquad C' = C \cup \{\mathsf{T} = \mathsf{Nat}\}}{\Gamma \vdash \mathsf{pred}\ \mathsf{t}_1 : \mathsf{Nat} | _{\mathcal{X}} C'} \quad (\text{CT-PRED})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T} | _{\mathcal{X}} C \qquad C' = C \cup \{\mathsf{T} = \mathsf{Nat}\}}{\Gamma \vdash \mathsf{iszero}\ \mathsf{t}_1 : \mathsf{Bool} | _{\mathcal{X}} C'} \quad (\text{CT-ISZER}$$

$$\Gamma \vdash \mathit{true} : \mathsf{Bool} | _{\emptyset} \{\} \quad (\text{CT-TRUE}) \qquad \Gamma \vdash \mathit{false} : \mathsf{Bool} | _{\emptyset} \{\} \quad (\text{CT-FALSE})$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 | _{\mathcal{X}_1} C_1 \\ \Gamma \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_2} C_2 \quad \Gamma \vdash \mathsf{t}_3 : \mathsf{T}_3 | _{\mathcal{X}_3} C_3 \\ C' = C_1 \cup C_2 \cup C_3 \cup \{\mathsf{T}_1 = \mathsf{Bool}, \mathsf{T}_2 = \mathsf{T}_3\}\end{array}}{\Gamma \vdash \mathsf{if}\ \mathsf{t}_1\ \mathsf{then}\ \mathsf{t}_2\ \mathsf{else}\ \mathsf{t}_3 : \mathsf{T}_2 | _{\mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3} C'} \quad (\text{CT-IF})$$

Typinferenz

Constraint-Based Typing

Beispiel: sei $\Gamma = \emptyset$ und $\mathsf{t} := \lambda\,\mathsf{x} : \mathsf{X}.\ \lambda\,\mathsf{y} : \mathsf{Y}.\ \mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0$

Typinferenz

Constraint-Based Typing

Beispiel: sei $\Gamma = \emptyset$ und $\mathsf{t} := \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0$

$$\frac{}{\emptyset \vdash \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0 : \quad \mid} \quad (\text{CT-Abs})$$

Typinferenz

Constraint-Based Typing

Beispiel: sei $\Gamma = \emptyset$ und $\mathsf{t} := \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0$

$$\frac{(\mathsf{x}:\mathsf{X}) \vdash \lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0 : \quad |}{\emptyset \vdash \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0 : \quad |} \quad (\text{CT-Abs})$$

Typinferenz

Constraint-Based Typing

Beispiel: sei $\Gamma = \emptyset$ und $\mathsf{t} := \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0$

$$\dfrac{\dfrac{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0 : \quad |}{(\mathsf{x}:\mathsf{X}) \vdash \lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0 : \quad |}(\text{CT-Abs})}{\emptyset \vdash \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0 : \quad |}(\text{CT-Abs})$$

Typinferenz

Constraint-Based Typing

Beispiel: sei $\Gamma = \emptyset$ und $\mathsf{t} := \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0$

$$\dfrac{\dfrac{\dfrac{}{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{x} : \mathsf{X}|_{\emptyset}\emptyset} \quad \dfrac{}{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{y} : \mathsf{Y}|_{\emptyset}\emptyset} \quad \dfrac{}{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash 0 : \mathsf{Nat}|_{\emptyset}\emptyset} \quad C = \{\mathsf{X} = \mathsf{Bool}, \mathsf{Y} = \mathsf{Nat}\}}{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0 : \quad |}\,(\text{CT-If})}{(\mathsf{x}:\mathsf{X}) \vdash \lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0 : \quad |}\,(\text{CT-Abs})}{\emptyset \vdash \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0 : \quad |}\,(\text{CT-Abs})$$

Typinferenz

Constraint-Based Typing

Beispiel: sei $\Gamma = \emptyset$ und $\mathsf{t} := \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{x} : \mathsf{X}|_{\emptyset}\emptyset} \quad \dfrac{}{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{y} : \mathsf{Y}|_{\emptyset}\emptyset} \quad \dfrac{}{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash 0 : \mathsf{Nat}|_{\emptyset}\emptyset} \quad C = \{\mathsf{X} = \mathsf{Bool}, \mathsf{Y} = \mathsf{Nat}\}}{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0 : \mathsf{Nat} \mid_{\emptyset} \{\mathsf{X} = \mathsf{Bool}, \mathsf{Y} = \mathsf{Nat}\}}(\text{CT-If})}{(\mathsf{x}:\mathsf{X}) \vdash \lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0 : \quad \mid}(\text{CT-Abs})}{\emptyset \vdash \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0 : \quad \mid}(\text{CT-A}$$

Typinferenz

Constraint-Based Typing

Beispiel: sei $\Gamma = \emptyset$ und $\mathsf{t} := \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0$

$$\dfrac{\dfrac{\dfrac{\dfrac{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{x}:\mathsf{X}|_{\emptyset}\emptyset \quad (\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{y}:\mathsf{Y}|_{\emptyset}\emptyset \quad (\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash 0:\mathsf{Nat}\,|_{\emptyset}\,\emptyset \quad C = \{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\}}{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0:\mathsf{Nat}\,|_{\emptyset}\,\{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\}}(\text{CT-If})}{(\mathsf{x}:\mathsf{X}) \vdash \lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0:\mathsf{Y}\to\mathsf{Nat}\,|_{\emptyset}\,\{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\}}(\text{CT-Abs})}{\emptyset \vdash \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0:\quad|}(\text{CT-A}$$

Typinferenz

Constraint-Based Typing

Beispiel: sei $\Gamma = \emptyset$ und $\mathsf{t} := \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0$

$$\dfrac{\dfrac{\dfrac{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{x}:\mathsf{X}|_{\emptyset}\emptyset \quad (\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{y}:\mathsf{Y}|_{\emptyset}\emptyset \quad (\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash 0:\mathsf{Nat}|_{\emptyset}\emptyset \quad C = \{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\}}{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0:\mathsf{Nat}\,|_{\emptyset}\,\{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\}}(\text{CT-If})}{(\mathsf{x}:\mathsf{X}) \vdash \lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0:\mathsf{Y}\to\mathsf{Nat}\,|_{\emptyset}\,\{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\}}(\text{CT-Abs})}{\emptyset \vdash \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0:\mathsf{X}\to\mathsf{Y}\to\mathsf{Nat}\,|_{\emptyset}\,\{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\}}(\text{CT-A}$$

Typinferenz

Constraint-Based Typing

Beispiel: sei $\Gamma = \emptyset$ und $\mathsf{t} := \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0$

$$\dfrac{\dfrac{\dfrac{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{x}:\mathsf{X}|_{\emptyset}\emptyset \quad (\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{y}:\mathsf{Y}|_{\emptyset}\emptyset \quad (\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash 0:\mathsf{Nat}|_{\emptyset}\emptyset \quad C = \{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\}}{(\mathsf{x}:\mathsf{X},\mathsf{y}:\mathsf{Y}) \vdash \mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0:\mathsf{Nat}\,|_{\emptyset}\,\{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\}}(\text{CT-If})}{(\mathsf{x}:\mathsf{X}) \vdash \lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0:\mathsf{Y}\to\mathsf{Nat}\,|_{\emptyset}\,\{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\}}(\text{CT-Abs})}{\emptyset \vdash \lambda\,\mathsf{x}:\mathsf{X}.\,\lambda\,\mathsf{y}:\mathsf{Y}.\,\mathsf{if}\,\mathsf{x}\,\mathsf{then}\,\mathsf{y}\,\mathsf{else}\,0:\mathsf{X}\to\mathsf{Y}\to\mathsf{Nat}\,|_{\emptyset}\,\{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\}}(\text{CT-A}$$

d.h. wir haben nun folgendes Constraintproblem (Γ, t, S, C):

$$(\emptyset, \mathsf{t}, \mathsf{X}\to\mathsf{Y}\to\mathsf{Nat}, \{\mathsf{X}=\mathsf{Bool},\mathsf{Y}=\mathsf{Nat}\})$$

Typinferenz

Constraint-Based Typing

- **Definition:** Gegeben sei $\Gamma \vdash t : S \mid C$. Eine **Lösung** für (Γ, t, S, C) ist nun ein Paar (σ, T), sodass gilt: σ erfüllt C und $\sigma S = T$

Typinferenz

Constraint-Based Typing

- **Definition:** Gegeben sei $\Gamma \vdash t : S \mid C$. Eine **Lösung** für (Γ, t, S, C) ist nun ein Paar (σ, T), sodass gilt: σ erfüllt C und $\sigma S = T$

- **Theorem:** [Soundness of Constraint Typing]
  Gegeben sei $\Gamma \vdash t : S \mid C$. Wenn (σ, T) eine Lösung für (Γ, t, S, C) ist, dann ist sie auch Lösung für (Γ, t)

Typinferenz

Constraint-Based Typing

- **Definition:** Gegeben sei $\Gamma \vdash t : S \mid C$. Eine **Lösung** für (Γ, t, S, C) ist nun ein Paar (σ, T), sodass gilt: σ erfüllt C und $\sigma S = T$

- **Theorem:** [Soundness of Constraint Typing]
  Gegeben sei $\Gamma \vdash t : S \mid C$. Wenn (σ, T) eine Lösung für (Γ, t, S, C) ist, dann ist sie auch Lösung für (Γ, t)

- **Theorem:** [Completeness of Constraint Typing]
  Gegeben sei $\Gamma \vdash t : S \mid_X C$. Wenn (σ, T) eine Lösung für (Γ, t) und $dom(\sigma) \cap X = \emptyset$, dann gibt es auch eine Lösung (σ', T) für (Γ, t, S, C), sodass $\sigma' \setminus X = \sigma$

Typinferenz

Constraint-Based Typing

- **Definition:** Gegeben sei $\Gamma \vdash t : S \mid C$. Eine **Lösung** für (Γ, t, S, C) ist nun ein Paar (σ, T), sodass gilt: σ erfüllt C und $\sigma S = T$

- **Theorem:** [Soundness of Constraint Typing]
  Gegeben sei $\Gamma \vdash t : S \mid C$. Wenn (σ, T) eine Lösung für (Γ, t, S, C) ist, dann ist sie auch Lösung für (Γ, t)

- **Theorem:** [Completeness of Constraint Typing]
  Gegeben sei $\Gamma \vdash t : S \mid_X C$. Wenn (σ, T) eine Lösung für (Γ, t) und $dom(\sigma) \cap X = \emptyset$, dann gibt es auch eine Lösung (σ', T) für (Γ, t, S, C), sodass $\sigma' \setminus X = \sigma$

Bem: Man übersetzt ein Typproblem in ein Constraintproblem, das genau dann lösbar ist, wenn das Typproblem lösbar ist

Typinferenz

Unifikation

- Constraintmenge C bestimmt, stellt sich die Frage:

Typinferenz

Unifikation

- Constraintmenge C bestimmt, stellt sich die Frage:
- Ist C überhaupt lösbar, wenn ja wie bekommt man eine Lösung?

Typinferenz

Unifikation

- Constraintmenge C bestimmt, stellt sich die Frage:
- Ist C überhaupt lösbar, wenn ja wie bekommt man eine Lösung?
- Diese Überlegung führt zu einem Algorithmus namens ***Unification***

Typinferenz

Unifikation

- Constraintmenge C bestimmt, stellt sich die Frage:
- Ist C überhaupt lösbar, wenn ja wie bekommt man eine Lösung?
- Diese Überlegung führt zu einem Algorithmus namens ***Unification***
- *Unification* liefert zu einer Constraintmenge C, falls lösbar, eine ***allgemeinste Lösung*** (σ, T)

Typinferenz

Unifikation

- Constraintmenge C bestimmt, stellt sich die Frage:
- Ist C überhaupt lösbar, wenn ja wie bekommt man eine Lösung?
- Diese Überlegung führt zu einem Algorithmus namens ***Unification***
- *Unification* liefert zu einer Constraintmenge C, falls lösbar, eine ***allgemeinste Lösung*** (σ, T)
- **Definition:** Eine Lösung (σ, T) heisst **allgemeinste Lösung**, wenn für alle weiteren Lösungen (σ', T') gilt:
  $\exists \sigma'' : \sigma' = \sigma'' \circ \sigma$.
  T heisst dann **allgemeinster Typ**.

Typinferenz

Unifikations-Algorithmus

$$
\begin{array}{lcl}
\text{unify}(C) & = & \text{if } C = \emptyset \text{ then } [\,] \\
& & \text{else let } \{S = T\} \cup C' = C \text{ in} \\
& & \quad \text{if } S = T \\
& & \quad\quad \text{then unify}(C') \\
& & \quad \text{else if } S = X \text{ and } X \notin FV(T) \\
& & \quad\quad \text{then unify}([X \mapsto T]C') \circ [X \mapsto T] \\
& & \quad \text{else if } T = X \text{ and } X \notin FV(S) \\
& & \quad\quad \text{then unify}([X \mapsto S]C') \circ [X \mapsto S] \\
& & \quad \text{else if } S = S_1 \to S_2 \text{ and } T = T_1 \to T_2 \\
& & \quad\quad \text{then unify}(C' \cup \{S_1 = T_1, S_2 = T_2\}) \\
& & \quad \text{else} \\
& & \quad\quad \text{fail}
\end{array}
$$

Typinferenz

Unifikation

zurück zu unserm Beispiel:

$$\emptyset \vdash \lambda\, x : X.\ \lambda\, y : Y.\ \text{if } x \text{ then } y \text{ else } 0 : X \to Y \to \text{Nat} \mid_{\emptyset} \{X = \text{Bool}, Y = \text{Nat}\}$$

Typinferenz

Unifikation

zurück zu unserm Beispiel:

$$\emptyset \vdash \lambda\, x : X.\ \lambda\, y : Y.\ \text{if } x \text{ then } y \text{ else } 0 : X \to Y \to \text{Nat} \mid_{\emptyset} \{X = \text{Bool}, Y = \text{Nat}\}$$

$$\text{unify}(\{X = \text{Bool}, Y = \text{Nat}\}) = [X \mapsto \text{Bool}, Y \mapsto \text{Nat}]$$

Typinferenz

Unifikation

zurück zu unserm Beispiel:

$$\emptyset \vdash \lambda\,\mathsf{x} : \mathsf{X}.\ \lambda\,\mathsf{y} : \mathsf{Y}.\ \mathsf{if}\ \mathsf{x}\ \mathsf{then}\ \mathsf{y}\ \mathsf{else}\ 0 : \mathsf{X} \to \mathsf{Y} \to \mathsf{Nat}\ |_{\emptyset}\ \{\mathsf{X} = \mathsf{Bool}, \mathsf{Y} = \mathsf{Nat}\}$$

$$\mathsf{unify}(\{\mathsf{X} = \mathsf{Bool}, \mathsf{Y} = \mathsf{Nat}\}) = [\mathsf{X} \mapsto \mathsf{Bool}, \mathsf{Y} \mapsto \mathsf{Nat}]$$

damit erhalten wir als allgemeinste Lösung für $(\emptyset, t)$:

$$([\mathsf{X} \mapsto \mathsf{Bool}, \mathsf{Y} \mapsto \mathsf{Nat}], \mathsf{Bool} \to \mathsf{Nat} \to \mathsf{Nat})$$

Let Polymorphismus

Typinferenz

Let Polymorphismus

- Wir erweitern unsere Regeln um:

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2} \quad (\text{T-Let})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 | X_1 C_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2 | \mathcal{X}_2 C_2 \qquad C = C_1 \cup C_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2 | \mathcal{X}_1 \cup \mathcal{X}_2 C} \quad (\text{CT-Let}$$

Typinferenz

Let Polymorphismus

- Wir erweitern unsere Regeln um:

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2} \quad (\text{T-Let})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 | _{X_1} C_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_2} C_2 \qquad C = C_1 \cup C_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_1 \cup \mathcal{X}_2} C} \quad (\text{CT-Let}$$

- somit sind folgende Terme typisierbar:

Typinferenz

Let Polymorphismus

- Wir erweitern unsere Regeln um:

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2} \quad (\text{T-Let})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 | _{X_1} C_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_2} C_2 \qquad C = C_1 \cup C_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_1 \cup \mathcal{X}_2} C} \quad (\text{CT-Let}$$

- somit sind folgende Terme typisierbar:

  - $\mathsf{t}_1 := \mathsf{let}\ f = \lambda\, x : X\, .\, x\ \mathsf{in}\ f\ 5$

Typinferenz

Let Polymorphismus

- Wir erweitern unsere Regeln um:

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2} \quad (\text{T-Let})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 | _{X_1} C_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_2} C_2 \qquad C = C_1 \cup C_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_1 \cup \mathcal{X}_2} C} \quad (\text{CT-Let}$$

- somit sind folgende Terme typisierbar:

$$\text{- } \mathsf{t}_1 := \mathsf{let}\ f = \lambda\, x : X\, .\, x\ \mathsf{in}\ f\ 5 \qquad\qquad C_1 = \{X \to X = \mathsf{Nat} \to Y\}$$

Typinferenz

Let Polymorphismus

- Wir erweitern unsere Regeln um:

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2} \quad (\text{T-Let})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 | _{X_1} C_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_2} C_2 \qquad C = C_1 \cup C_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_1 \cup \mathcal{X}_2} C} \quad (\text{CT-Let}$$

- somit sind folgende Terme typisierbar:

  - $\mathsf{t}_1 := \mathsf{let}\ f = \lambda\, x : X\, .\, x\ \mathsf{in}\ f\ 5 \qquad C_1 = \{X \to X = \mathsf{Nat} \to Y\}$
  - $\mathsf{t}_2 := \mathsf{let}\ f = \lambda\, x : X\, .\, x\ \mathsf{in}\ f\ \mathsf{true}$

Typinferenz

Let Polymorphismus

- Wir erweitern unsere Regeln um:

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2} \quad (\text{T-Let})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 | _{X_1} C_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_2} C_2 \qquad C = C_1 \cup C_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_1 \cup \mathcal{X}_2} C} \quad (\text{CT-Let})$$

- somit sind folgende Terme typisierbar:

  - $\mathsf{t}_1 := \mathsf{let}\ f = \lambda\, x : X\, .\, x\ \mathsf{in}\ f\ 5 \qquad C_1 = \{X \to X = \mathsf{Nat} \to Y\}$
  - $\mathsf{t}_2 := \mathsf{let}\ f = \lambda\, x : X\, .\, x\ \mathsf{in}\ f\ \mathsf{true} \qquad C_2 = \{X \to X = \mathsf{Bool} \to Y\}$

Typinferenz

Let Polymorphismus

- Wir erweitern unsere Regeln um:

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2} \quad (\text{T-Let})$$

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1 | _{X_1} C_1 \qquad \Gamma, x : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_2} C_2 \qquad C = C_1 \cup C_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}_1 \cup \mathcal{X}_2} C} \quad (\text{CT-Let})$$

- somit sind folgende Terme typisierbar:

  - $\mathsf{t}_1 := \mathsf{let}\ f = \lambda x : X\ .\ x\ \mathsf{in}\ f\ 5 \qquad C_1 = \{X \to X = \mathsf{Nat} \to Y\}$
  - $\mathsf{t}_2 := \mathsf{let}\ f = \lambda x : X\ .\ x\ \mathsf{in}\ f\ \mathsf{true} \qquad C_2 = \{X \to X = \mathsf{Bool} \to Y\}$

  d.h. wir können f in **verschiedenen** Programmen mit unterschiedlichen Typen verwenden

Typinferenz

Let Polymorphismus

- Unser Ziel ist es jedoch f im selben Programm mit unterschiedlichen Typen zu verwenden !

Typinferenz

Let Polymorphismus

- Unser Ziel ist es jedoch f im selben Programm mit unterschiedlichen Typen zu verwenden !

  - let $f = \lambda x : X \,.\, x$ in
    let $a =\ f$ 5 in
      let $b =\ f$ true in ...

Typinferenz

Let Polymorphismus

- Unser Ziel ist es jedoch f im selben Programm mit unterschiedlichen Typen zu verwenden !

  - let $f = \lambda x : X \,.\, x$ in
    let $a =\ f$ 5 in
      let $b =\ f$ true in ...

  $C_1 = \{X \to X = \mathsf{Nat} \to Y\}$

Typinferenz

Let Polymorphismus

- Unser Ziel ist es jedoch f im selben Programm mit unterschiedlichen Typen zu verwenden !

- let $f = \lambda x : X \,.\, x$ in
  let $a =\ f$ 5 in $\qquad C_1 = \{X \to X = \mathsf{Nat} \to Y\}$
  let $b =\ f$ true in ... $\qquad C_2 = \{X \to X = \mathsf{Bool} \to Y\}$

Typinferenz

Let Polymorphismus

- Unser Ziel ist es jedoch f im selben Programm mit unterschiedlichen Typen zu verwenden !

  - let $f = \lambda x : X \,.\, x$ in
    let $a =\ f$ 5 in $\qquad C_1 = \{X \to X = \mathsf{Nat} \to Y\}$
    let $b =\ f$ true in ... $\qquad C_2 = \{X \to X = \mathsf{Bool} \to Y\}$

  führt zu folgendem Constraint

  $$C = \{X \to X = \mathsf{Nat} \to Y \ \wedge X \to X = \mathsf{Bool} \to Z\}$$

Typinferenz

Let Polymorphismus

- Unser Ziel ist es jedoch f im selben Programm mit unterschiedlichen Typen zu verwenden !

  - let $f = \lambda x : X \,.\, x$ in
    let $a =\ f$ 5 in
      let $b =\ f$ true in ...

  $C_1 = \{X \to X = \mathsf{Nat} \to Y\}$

  $C_2 = \{X \to X = \mathsf{Bool} \to Y\}$

  führt zu folgendem Constraint

  $C = \{X \to X = \mathsf{Nat} \to Y \ \wedge X \to X = \mathsf{Bool} \to Z\}$ unerfüllbar !!

Typinferenz

Let Polymorphismus

- Wir können unser Ziel durch folgende Regeln erreichen:

Typinferenz

Let Polymorphismus

- Wir können unser Ziel durch folgende Regeln erreichen:

$$\frac{X \,\mathsf{neu} \qquad \Gamma, x : X \vdash \mathsf{t}_1 : \mathsf{T}|_{\mathcal{X}} C}{\Gamma \vdash \lambda\, x.\, \mathsf{t}_1 \;\; : X \to \mathsf{T}|_{\mathcal{X} \cup \{X\}} C} \quad (\text{CT-AbsInf})$$

Typinferenz

Let Polymorphismus

- Wir können unser Ziel durch folgende Regeln erreichen:

$$\frac{X \,\mathsf{neu} \qquad \Gamma, x : X \vdash \mathsf{t}_1 : \mathsf{T}|_{\mathcal{X}} C}{\Gamma \vdash \lambda\, x.\, \mathsf{t}_1 \;\; : X \to \mathsf{T}|_{\mathcal{X} \cup \{X\}} C} \quad (\text{CT-AbsInf})$$

$$\frac{\Gamma \vdash [x \mapsto \mathsf{t}_1]\mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\, x = \mathsf{t}_1 \,\mathsf{in}\, \mathsf{t}_2 : \mathsf{T}_2} \quad (\text{T-LetPoly})$$

Typinferenz

Let Polymorphismus

- Wir können unser Ziel durch folgende Regeln erreichen:

$$\frac{X \,\mathsf{neu} \qquad \Gamma, x : X \vdash \mathsf{t}_1 : \mathsf{T}|_{\mathcal{X}} C}{\Gamma \vdash \lambda\, x.\, \mathsf{t}_1 \;: X \to \mathsf{T}|_{\mathcal{X} \cup \{X\}} C} \quad (\text{CT-AbsInf})$$

$$\frac{\Gamma \vdash [x \mapsto \mathsf{t}_1]\mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\, x = \mathsf{t}_1 \,\mathsf{in}\, \mathsf{t}_2 : \mathsf{T}_2} \quad (\text{T-LetPoly})$$

$$\frac{\Gamma \vdash [x \mapsto \mathsf{t}_1]\mathsf{t}_2 : \mathsf{T}_2|_{\mathcal{X}} C}{\Gamma \vdash \mathsf{let}\, x = \mathsf{t}_1 \,\mathsf{in}\, \mathsf{t}_2 : \mathsf{T}_2|_{\mathcal{X}} C} \quad (\text{CT-LetPoly})$$

Typinferenz

Let Polymorphismus

- Wir können unser Ziel durch folgende Regeln erreichen:

$$\frac{X \text{ neu} \qquad \Gamma, x : X \vdash \mathsf{t}_1 : \mathsf{T} | _{\mathcal{X}} C}{\Gamma \vdash \lambda\, x.\, \mathsf{t}_1 \ : X \to \mathsf{T} | _{\mathcal{X} \cup \{X\}} C} \quad (\text{CT-AbsInf})$$

$$\frac{\Gamma \vdash [x \mapsto \mathsf{t}_1]\mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2} \quad (\text{T-LetPoly})$$

$$\frac{\Gamma \vdash [x \mapsto \mathsf{t}_1]\mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}} C}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2 | _{\mathcal{X}} C} \quad (\text{CT-LetPoly})$$

- $\mathsf{let}\ f = \lambda\, x\, .\, x\ \mathsf{in}$
  $\mathsf{let}\ a = \ f\ 5\ \mathsf{in}$ $\qquad C_1 = \{X_1 \to X_1 = \mathsf{Nat} \to Y\}$
  $\mathsf{let}\ b = \ f\ \mathsf{true}\ \mathsf{in} \ldots$ $\qquad C_2 = \{X_2 \to X_2 = \mathsf{Bool} \to Z\}$

Typinferenz

Let Polymorphismus

- leider gibt es noch ein kleines Problem:

Typinferenz

Let Polymorphismus

- leider gibt es noch ein kleines Problem:

  - folgender Term wäre typisierbar

    let $x =$ <utter garbage> in 5

Typinferenz

Let Polymorphismus

- leider gibt es noch ein kleines Problem:

  - folgender Term wäre typisierbar

    let $x =$ <utter garbage> in 5

  - deshalb noch eine kleine Änderung

$$\frac{\Gamma \vdash [x \mapsto \mathsf{t}_1]\mathsf{t}_2 : \mathsf{T}_2 \quad \Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1}{\Gamma \vdash \mathsf{let}\ x = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathsf{T}_2} \quad (\text{T-LetPoly})$$

HM(X)

General Framework HM(X)

Einleitung

- Viele Typsysteme erweitern das Hindley Milner System mit Constraints

General Framework HM(X)

Einleitung

- Viele Typsysteme erweitern das Hindley Milner System mit Constraints
- Beispiele: *record systems*, *overloading* und *subtyping*

General Framework HM(X)

Einleitung

- Viele Typsysteme erweitern das Hindley Milner System mit Constraints
- Beispiele: *record systems*, *overloading* und *subtyping*
- Für all diese Typsysteme muss man

General Framework HM(X)

Einleitung

- Viele Typsysteme erweitern das Hindley Milner System mit Constraints
- Beispiele: *record systems*, *overloading* und *subtyping*
- Für all diese Typsysteme muss man
  1. einen Inferenzalgorithmus herleiten

General Framework HM(X)

Einleitung

- Viele Typsysteme erweitern das Hindley Milner System mit Constraints
- Beispiele: *record systems*, *overloading* und *subtyping*
- Für all diese Typsysteme muss man
  1. einen Inferenzalgorithmus herleiten
  2. Eigenschaften beweisen wie z.B. Soundness oder Completeness

General Framework HM(X)

Einleitung

- Viele Typsysteme erweitern das Hindley Milner System mit Constraints
- Beispiele: *record systems*, *overloading* und *subtyping*
- Für all diese Typsysteme muss man
  1. einen Inferenzalgorithmus herleiten
  2. Eigenschaften beweisen wie z.B. Soundness oder Completeness
- Aus diesem Grund entwickelten *M. Odersky*, *M. Sulzmann* und *M. Wehr* den **General Framework HM(X)**.

General Framework HM(X)

Was ist HM(X)?

- **generische Erweiterung** des Hindley-Milner-Typsystems mit Constraints

General Framework HM(X)

Was ist HM(X)?

- **generische Erweiterung** des Hindley-Milner-Typsystems mit Constraints
- spezielle Instanzen erhält man über den Parameter X

General Framework HM(X)

Was ist HM(X)?

- **generische Erweiterung** des Hindley-Milner-Typsystems mit Constraints
- spezielle Instanzen erhält man über den Parameter X
- HM(X) steht ein generischer Inferenzalgorithmus zur Verfügung. Er liefert unter best. Bedingungen für X jeweils den allgemeinsten Typ eines Terms

General Framework HM(X)

HM(X)

- **Syntax des HM(X) Frameworks:**

General Framework HM(X)

HM(X)

- **Syntax des HM(X) Frameworks:**

Values	$v ::= x \mid \lambda x.e$
Expressions	$e ::= v \mid e\, e \mid \mathsf{let}\ x = e\ \mathsf{in}\ \mathsf{e}$
Types	$\tau ::= \alpha \mid \tau \rightarrow \tau \mid \mathsf{T}\bar{\tau}$
Type schemes	$\sigma ::= \tau \mid \forall \alpha.C \Rightarrow \sigma$

General Framework HM(X)

HM(X)

- **Syntax des HM(X) Frameworks:**

Values	$v ::= x \mid \lambda x.e$
Expressions	$e ::= v \mid e\,e \mid \mathsf{let}\ x = e\ \mathsf{in}\ \mathsf{e}$
Types	$\tau ::= \alpha \mid \tau \rightarrow \tau \mid \mathsf{T}\bar{\tau}$
Type schemes	$\sigma ::= \tau \mid \forall\alpha.C \Rightarrow \sigma$

  - C soll dabei aus der Menge S der *solved forms* stammen, die je nach Constraintsystem variiert.
    Jedes $C \in S$ muss dabei mindestens erfüllbar sein.

General Framework HM(X)

Typregeln für HM(X)

- **Typregeln(1):**

General Framework HM(X)

Typregeln für HM(X)

- **Typregeln(1):**

(Var) $$C, \Gamma \vdash x : \sigma \;\; (x : \sigma \in \Gamma)$$

(Sub) $$\frac{C, \Gamma \vdash e : \tau \quad C \vdash^e (\tau \preceq \tau')}{C, \Gamma \vdash e : \tau'}$$

(Abs) $$\frac{C, \Gamma_x . x : \tau \vdash e : \tau'}{C, \Gamma_x \vdash \lambda x.e : \tau \to \tau'}$$

(App) $$\frac{C, \Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad C, \Gamma \vdash e_2 : \tau_1}{C, \Gamma \vdash e_1\, e_2 \;\; : \tau_2}$$

General Framework HM(X)

Typregeln für HM(X)

- **Typregeln(2):**

General Framework HM(X)

Typregeln für HM(X)

- **Typregeln(2):**

(Let) $$\frac{C, \Gamma_x \vdash e : \sigma \quad C, \Gamma_x.x : \sigma \vdash e' : \tau'}{C, \Gamma_x \vdash let\ x = e\ in\ e' : \tau'}$$

(∀ Intro) $$\frac{C \wedge D, \Gamma \vdash e : \tau \quad \bar{\alpha} \notin (C) \cup fv(\Gamma)}{C \wedge \exists \bar{\alpha}.D,\ \Gamma \vdash e : \forall \bar{\alpha}.D \Rightarrow \tau}$$

(∀ Elim) $$\frac{C, \Gamma \vdash e : \forall \bar{\alpha}.D \Rightarrow \tau' \quad C \vdash^e [\bar{\tau}/\bar{\alpha}]D}{C, \Gamma \vdash e : [\bar{\tau}/\bar{\alpha}]\tau'}$$

General Framework HM(X)

Konkrete Typsysteme

- wie bekommt man konkrete Typsysteme ?

General Framework HM(X)

Konkrete Typsysteme

- wie bekommt man konkrete Typsysteme ?

  1. für X ein Constraintsystem wählen

General Framework HM(X)

Konkrete Typsysteme

- wie bekommt man konkrete Typsysteme ?

  1. für X ein Constraintsystem wählen
  2. Menge S der *solved forms* bestimmen

General Framework HM(X)

Konkrete Typsysteme

- wie bekommt man konkrete Typsysteme ?

  1. für X ein Constraintsystem wählen
  2. Menge S der *solved forms* bestimmen
  3. Bedeutung für $\preceq$ wählen

General Framework HM(X)

Konkrete Typsysteme

- wie bekommt man konkrete Typsysteme ?

  1. für X ein Constraintsystem wählen
  2. Menge S der *solved forms* bestimmen
  3. Bedeutung für $\preceq$ wählen
  4. Syntax anpassen

General Framework HM(X)

Konkrete Typsysteme

- wie bekommt man konkrete Typsysteme ?

  1. für X ein Constraintsystem wählen
  2. Menge S der *solved forms* bestimmen
  3. Bedeutung für $\preceq$ wählen
  4. Syntax anpassen

- als Beispiele:

General Framework HM(X)

Konkrete Typsysteme

- wie bekommt man konkrete Typsysteme ?
  1. für X ein Constraintsystem wählen
  2. Menge S der *solved forms* bestimmen
  3. Bedeutung für $\preceq$ wählen
  4. Syntax anpassen
- als Beispiele:
  - Hindley/Milner

General Framework HM(X)

Konkrete Typsysteme

- wie bekommt man konkrete Typsysteme ?
  1. für X ein Constraintsystem wählen
  2. Menge S der *solved forms* bestimmen
  3. Bedeutung für $\preceq$ wählen
  4. Syntax anpassen
- als Beispiele:
  - Hindley/Milner
  - Subtyping

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 1 [Hindley/Milner]:**

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 1 [Hindley/Milner]:**
  - für X wählt man HERBRAND

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 1 [Hindley/Milner]:**
  - für X wählt man HERBRAND
  - S ist die Menge, die nur true enthält also $\emptyset$

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 1 [Hindley/Milner]:**
  - für X wählt man HERBRAND
  - S ist die Menge, die nur true enthält also $\emptyset$
  - $\preceq$ entspricht syntaktischer Gleichheit $=$

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 1 [Hindley/Milner]:**
  - für X wählt man HERBRAND
  - S ist die Menge, die nur true enthält also $\emptyset$
  - $\preceq$ entspricht syntaktischer Gleichheit $=$
  - keine zusätzliche Terme oder Typen

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 1 [Hindley/Milner]:**
  - für X wählt man HERBRAND
  - S ist die Menge, die nur true enthält also $\emptyset$
  - $\preceq$ entspricht syntaktischer Gleichheit $=$
  - keine zusätzliche Terme oder Typen

(Var) $$C, \Gamma \vdash x : \sigma \ \ (x : \sigma \in \Gamma)$$

(Sub) $$\frac{C, \Gamma \vdash e : \tau \quad C \vdash^e (\tau \preceq \tau')}{C, \Gamma \vdash e : \tau'}$$

(Abs) $$\frac{C, \Gamma_x . x : \tau \vdash e : \tau'}{C, \Gamma_x \vdash \lambda x.e : \tau \rightarrow \tau'}$$

$$\vdots$$

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 1 [Hindley/Milner]:**
  - für X wählt man HERBRAND
  - S ist die Menge, die nur true enthält also $\emptyset$
  - $\preceq$ entspricht syntaktischer Gleichheit $=$
  - keine zusätzliche Terme oder Typen

(Var) $\Gamma \vdash x : \sigma \ \ (x : \sigma \in \Gamma)$

(Sub) $$\frac{\Gamma \vdash e : \tau \quad \vdash^e (\tau \preceq \tau')}{\Gamma \vdash e : \tau'}$$

(Abs) $$\frac{\Gamma_x . x : \tau \vdash e : \tau'}{\Gamma_x \vdash \lambda x.e : \tau \to \tau'}$$

$$\vdots$$

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 1 [Hindley/Milner]:**
  - für X wählt man HERBRAND
  - S ist die Menge, die nur true enthält also $\emptyset$
  - $\preceq$ entspricht syntaktischer Gleichheit $=$
  - keine zusätzliche Terme oder Typen

(Var) $\Gamma \vdash x : \sigma \ \ (x : \sigma \in \Gamma)$

(Sub) $$\frac{\Gamma \vdash e : \tau \quad \vdash^e (\tau = \tau')}{\Gamma \vdash e : \tau'}$$

(Abs) $$\frac{\Gamma_x . x : \tau \vdash e : \tau'}{\Gamma_x \vdash \lambda x.e : \tau \to \tau'}$$

$$\vdots$$

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 2 [Subtyping]:**

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 2 [Subtyping]:**
  - für X wählt man SC

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 2 [Subtyping]:**
  - für X wählt man SC
  - S ist die Menge aller erfüllbarer Constraints in SC

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 2 [Subtyping]:**
  - für X wählt man SC
  - S ist die Menge aller erfüllbarer Constraints in SC
  - $\preceq$ entspricht Gleichheit $<$:

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 2 [Subtyping]:**
  - für X wählt man SC
  - S ist die Menge aller erfüllbarer Constraints in SC
  - $\preceq$ entspricht Gleichheit $<$:
  - für alle *records* $\{l_1 : \tau_1, \ldots, l_n : \tau_n\}$

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 2 [Subtyping]:**
  - für X wählt man SC
  - S ist die Menge aller erfüllbarer Constraints in SC
  - $\preceq$ entspricht Gleichheit $<$:
  - für alle *records* $\{l_1 : \tau_1, \ldots, l_n : \tau_n\}$
    - $\{l_1 : \tau_1, \ldots, l_n : \tau_n\} \quad \in \tau$

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 2 [Subtyping]:**
  - für X wählt man SC
  - S ist die Menge aller erfüllbarer Constraints in SC
  - $\preceq$ entspricht Gleichheit $<$:
  - für alle *records* $\{l_1 : \tau_1, \ldots, l_n : \tau_n\}$
    - $\{l_1 : \tau_1, \ldots, l_n : \tau_n\} \quad \in \tau$
    - $f_{l_1 \ldots l_n} : \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \{l_1 : \tau_1, \ldots, l_n : \tau_n\} \quad \in \Gamma_0$

General Framework HM(X)

Konkrete Typsysteme

- **Beispiel 2 [Subtyping]:**
  - für X wählt man SC
  - S ist die Menge aller erfüllbarer Constraints in SC
  - $\preceq$ entspricht Gleichheit $<$:
  - für alle *records* $\{l_1 : \tau_1, \ldots, l_n : \tau_n\}$
    - $\{l_1 : \tau_1, \ldots, l_n : \tau_n\} \quad \in \tau$
    - $f_{l_1 \ldots l_n} : \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \{l_1 : \tau_1, \ldots, l_n : \tau_n\} \quad \in \Gamma_0$
    - $f_l : \{l : \tau\} \rightarrow \tau \quad \in \Gamma_0$

General Framework HM(X)

Typinferenz HM(X)

$$(\textsc{Var}) \qquad \frac{\begin{array}{c} x : (\forall \bar{\alpha}.D \Rightarrow \tau) \in \Gamma \quad \bar{\beta}\, new \\ (C, \psi) = normalize(D, [\bar{\beta}/\bar{\alpha}]) \end{array}}{\psi_{|fv(\Gamma)}, C, \Gamma \vdash^{W} x : \psi\tau}$$

(Abs)

(App)

General Framework HM(X)

Typinferenz HM(X)

$$(\textsc{Var}) \qquad \frac{x : (\forall \bar{\alpha}.D \Rightarrow \tau) \in \Gamma \quad \bar{\beta}\, new \quad (C, \psi) = normalize(D, [\bar{\beta}/\bar{\alpha}])}{\psi_{|fv(\Gamma)}, C, \Gamma \vdash^W x : \psi\tau}$$

$$(\textsc{Abs}) \qquad \frac{\psi, C, \Gamma_x.x : \alpha \vdash^W e : \tau \quad \alpha\, new}{\psi_{\setminus\{\alpha\}}, C, \Gamma_x \vdash^W \lambda x.e : \psi(\alpha) \to \tau}$$

$$(\textsc{App})$$

General Framework HM(X)

Typinferenz HM(X)

$$(\textsc{Var}) \qquad \frac{x : (\forall \bar{\alpha}.D \Rightarrow \tau) \in \Gamma \quad \bar{\beta}\, new \quad (C, \psi) = normalize(D, [\bar{\beta}/\bar{\alpha}])}{\psi_{|fv(\Gamma)}, C, \Gamma \vdash^W x : \psi\tau}$$

$$(\textsc{Abs}) \qquad \frac{\psi, C, \Gamma_x.x : \alpha \vdash^W e : \tau \quad \alpha\, new}{\psi_{\setminus\{\alpha\}}, C, \Gamma_x \vdash^W \lambda x.e : \psi(\alpha) \to \tau}$$

$$(\textsc{App}) \qquad \frac{\begin{array}{c}\psi_1, C_1, \Gamma \vdash^W e_1 : \tau_1 \quad \psi_2, C_2, \Gamma \vdash^W e_2 : \tau_2 \\ \psi' = \psi_1 \sqcup \psi_2 \\ D = C_1 \wedge C_2 \wedge (\tau_1 \preceq \tau_2 \to \alpha) \quad \alpha\, new \\ (C, \psi) = normalize(D, \psi')\end{array}}{\psi_{|fv(\Gamma)}, C, \Gamma \vdash^W e_1\, e_2 : \psi(\alpha)}$$

General Framework HM(X)

Typinferenz HM(X)

$$\text{(Let)} \quad \frac{\begin{array}{c}\psi_1, C_1, \Gamma_x \vdash^W e : \tau \quad (C_2, \sigma) = gen(C_1, \psi_1\Gamma, \tau) \\ \psi_2, C_3, \Gamma_x.x : \sigma \vdash^W e' : \tau' \\ \psi' = \psi_1 \sqcup \psi_2 \quad D = C_2 \wedge C_3 \\ (C, \psi) = normalize(D, \psi')\end{array}}{\psi_{|fv(\Gamma)}, C, \Gamma \vdash^W let\, x = e\, in\, e' : \psi(\tau')}$$

General Framework HM(X)

Typinferenz HM(X)

$$\frac{\begin{array}{c}\psi_1, C_1, \Gamma_x \vdash^W e : \tau \quad (C_2, \sigma) = gen(C_1, \psi_1\Gamma, \tau) \\ \psi_2, C_3, \Gamma_x.x : \sigma \vdash^W e' : \tau' \\ \psi' = \psi_1 \sqcup \psi_2 \quad D = C_2 \wedge C_3 \\ (C, \psi) = normalize(D, \psi')\end{array}}{\psi_{|fv(\Gamma)}, C, \Gamma \vdash^W let\, x = e\, in\, e' : \psi(\tau')} \text{(Let)}$$

- gen muss nur folgendes erfüllen:

$$gen(C, \Gamma, \sigma) = (D \wedge \exists\bar{\alpha}.C', \forall\bar{\alpha}.C' \Rightarrow \sigma)$$

General Framework HM(X)

Typinferenz HM(X)

- **Typinferenz unter HM(X):**

General Framework HM(X)

Typinferenz HM(X)

- **Typinferenz unter HM(X):**
  - Typproblem in Constraintproblem umwandeln

General Framework HM(X)

Typinferenz HM(X)

- **Typinferenz unter HM(X):**
  - Typproblem in Constraintproblem umwandeln
  - Constraintproblem lösen

General Framework HM(X)

Typinferenz HM(X)

- **Typinferenz unter HM(X):**
  - Typproblem in Constraintproblem umwandeln
  - Constraintproblem lösen
  - **Schwierigkeit:**

General Framework HM(X)

Typinferenz HM(X)

- **Typinferenz unter HM(X):**
  - Typproblem in Constraintproblem umwandeln
  - Constraintproblem lösen
  - **Schwierigkeit:**
    - Constraintgeneration für alle Typsysteme gleich

General Framework HM(X)

Typinferenz HM(X)

- **Typinferenz unter HM(X):**
  - Typproblem in Constraintproblem umwandeln
  - Constraintproblem lösen
  - **Schwierigkeit:**
    - Constraintgeneration für alle Typsysteme gleich
    - Constraintproblem lösen für jedes System spezifisch !!

General Framework HM(X)

Typinferenz HM(X)

- **Typinferenz unter HM(X):**
  - Typproblem in Constraintproblem umwandeln
  - Constraintproblem lösen
  - **Schwierigkeit:**
    - Constraintgeneration für alle Typsysteme gleich
    - Constraintproblem lösen für jedes System spezifisch !!

    Beispiele: - Bei Hindley/Milner ist $S = \emptyset$,
    deshalb Lösungen von der Form $(\psi, \emptyset, \tau)$
    d.h. alle Gleichungen müssen gelöst werden !!
    - Bei *subtyping* besteht S aus allen unter
    SC erfüllbaren Constraintmengen,
    d.h. nur Erfüllbarkeitsprüfung erforderlich !!

General Framework HM(X)

Typinferenz HM(X)

- **Normalisierung:**

General Framework HM(X)

Typinferenz HM(X)

- **Normalisierung:**
  - Hindley/Milner
    - Unification

General Framework HM(X)

Typinferenz HM(X)

- **Normalisierung:**
  - Hindley/Milner
    - Unification
  - Subtyping:

General Framework HM(X)

Typinferenz HM(X)

- **Normalisierung:**
  - Hindley/Milner
    - Unification
  - Subtyping:
    - $normalize(C, \phi)$
      $= (\phi C, id) \quad if\ \phi C$ erfüllbar
      $= fail$ sonst

General Framework HM(X)

Zusammenfassung

HM(X) hat somit folgende Vorteile:

- Viele Typsysteme können als Instanz von HM(X) ausgedrückt werden

General Framework HM(X)

Zusammenfassung

HM(X) hat somit folgende Vorteile:

- Viele Typsysteme können als Instanz von HM(X) ausgedrückt werden
- Für alle diese Instanzen steht ein Inferenzalg. zur Verfügung

General Framework HM(X)

Zusammenfassung

HM(X) hat somit folgende Vorteile:

- Viele Typsysteme können als Instanz von HM(X) ausgedrückt werden
- Für alle diese Instanzen steht ein Inferenzalg. zur Verfügung
- Eigenschaften wie Soundness, Completeness oder Principal Typ sind gegeben, wenn X best. Bedingungen erfüllt

General Framework HM(X)

Zusammenfassung

HM(X) hat somit folgende Vorteile:

- Viele Typsysteme können als Instanz von HM(X) ausgedrückt werden
- Für alle diese Instanzen steht ein Inferenzalg. zur Verfügung
- Eigenschaften wie Soundness, Completeness oder Principal Typ sind gegeben, wenn X best. Bedingungen erfüllt
- Experimentieren mit neuen Typsystemen bei wesentlich geringerem Aufwand

General Framework HM(X)

Zusammenfassung

HM(X) hat somit folgende Vorteile:

- Viele Typsysteme können als Instanz von HM(X) ausgedrückt werden
- Für alle diese Instanzen steht ein Inferenzalg. zur Verfügung
- Eigenschaften wie Soundness, Completeness oder Principal Typ sind gegeben, wenn X best. Bedingungen erfüllt
- Experimentieren mit neuen Typsystemen bei wesentlich geringerem Aufwand
- HM(X) kann erweitert werden, sodass man auch z.B. mit recursiven Typsystemen arbeiten kann

Referenzen

- Benjamin Pierce
  Types and Programming Languages, Chapter 22.
  MIT Press 2002

- M. Odersky, M. Sulzmann and M. Wehr
  Type inference with constrained types
  Theory and Practice of Object Systems, 5(1), 1999