Der einfach getypter Lambda-Kalkül

Typprüfung und Typinferenz

Tobias Nipkow und Steffen Smolka

Technische Universität München

Motivation

- Statische Fehlererkennung:
  $(\lambda x.x + 1)$ true

Motivation

- Statische Fehlererkennung:
  $(\lambda x.x + 1)$ true
- Vermeidung von Inkonsistenzen:
  $R := \lambda x.\mathsf{not}(x\ x) \quad \Rightarrow \quad R\ R =_\beta \mathsf{not}(R\ R)$

Motivation

- Statische Fehlererkennung:
  $(\lambda x.x + 1)$ true
- Vermeidung von Inkonsistenzen:
  $R := \lambda x.\text{not}(x\ x) \quad \Rightarrow \quad R\ R =_\beta \text{not}(R\ R)$
- Gibt Garantie für Ergebnis:
  $+ : \text{nat} \to \text{nat} \to \text{nat}$
  $\mathit{prime} : \text{nat} \to \text{bool}$

Formen der Typisierung

- Explizite Typisierung:
  durch Typannotation $\rightarrow$ ***Typprüfung***

Formen der Typisierung

- Explizite Typisierung:
  durch Typannotation $\rightarrow$ ***Typprüfung***

- Implizite Typisierung:
  Typen werden durch Interpreter/Compiler rekonstruiert
  $\rightarrow$ ***Typinferenz***

Terme

t	$::=$	x	Variable
	$\mid$	$\lambda x : T.\ t$	Abstraktion
	$\mid$	$t\ t$	Applikation

Terme

t	$::=$	x	Variable
	$\mid$	$\lambda x : T.\ t$	Abstraktion
	$\mid$	$t\ t$	Applikation
	$\mid$	$\ldots$	Konstante

Typen

$T \quad ::= \quad nat \mid bool \mid \ldots$ Basistypen

Typen

$$
\begin{array}{lcll}
T & ::= & \mathit{nat} \mid \mathit{bool} \mid \ldots & \text{Basistypen} \\
 & \mid & T_1 \to T_2 & \text{Funktionstyp}
\end{array}
$$

Typen

$$
\begin{array}{lcll}
T & ::= & \mathit{nat} \mid \mathit{bool} \mid \ldots & \text{Basistypen} \\
 & \mid & T_1 \to T_2 & \text{Funktionstyp}
\end{array}
$$

Klammerung:

$$T_1 \to T_2 \to T_3 \quad = \quad T_1 \to (T_2 \to T_3)$$

Typen

$$
\begin{array}{lcll}
T & ::= & \mathit{nat} \mid \mathit{bool} \mid \ldots & \text{Basistypen} \\
 & \mid & T_1 \to T_2 & \text{Funktionstyp}
\end{array}
$$

Klammerung:

$$
\begin{array}{rcl}
T_1 \to T_2 \to T_3 & = & T_1 \to (T_2 \to T_3) \\
 & \neq & (T_1 \to T_2) \to T_3
\end{array}
$$

Die Typisierungsrelation

Idee: Termen können Typen zugewiesen werden.

Die Typisierungsrelation

Idee: Termen können Typen zugewiesen werden.

$$t : T$$

Die Typisierungsrelation

Idee: Termen können Typen zugewiesen werden.

$$t : T$$

Beispiele

$\lambda x : nat.\ x :$

Die Typisierungsrelation

Idee: Termen können Typen zugewiesen werden.

$$t : T$$

Beispiele

$\lambda x : nat.\ x :$ $nat \to nat$

Die Typisierungsrelation

Idee: Termen können Typen zugewiesen werden.

$$t : T$$

Beispiele

$\lambda x : nat.\ x :$ $nat \to nat$

$\lambda x : nat \to nat.\ x :$

Die Typisierungsrelation

Idee: Termen können Typen zugewiesen werden.

$$t : T$$

Beispiele

$\lambda x : nat.\ x :$	$nat \to nat$
$\lambda x : nat \to nat.\ x :$	$(nat \to nat) \to (nat \to nat)$

Die Typisierungsrelation

Idee: Termen können Typen zugewiesen werden.

$$t : T$$

Beispiele

$\lambda x : nat.\ x :$	$nat \to nat$
$\lambda x : nat \to nat.\ x :$	$(nat \to nat) \to (nat \to nat)$
$\lambda x : nat.\ \lambda y : nat \to nat.\ y\ x :$	

Die Typisierungsrelation

Idee: Termen können Typen zugewiesen werden.

$$t : T$$

Beispiele

$\lambda x : nat.\ x :$	$nat \to nat$
$\lambda x : nat \to nat.\ x :$	$(nat \to nat) \to (nat \to nat)$
$\lambda x : nat.\ \lambda y : nat \to nat.\ y\ x :$	$nat \to (nat \to nat) \to nat$

Die Typisierungsrelation

Idee: Termen können Typen zugewiesen werden.

$$t : T$$

Beispiele

$\lambda x : nat.\ x :$	$nat \to nat$
$\lambda x : nat \to nat.\ x :$	$(nat \to nat) \to (nat \to nat)$
$\lambda x : nat.\ \lambda y : nat \to nat.\ y\ x :$	$nat \to (nat \to nat) \to nat$

Definition

Ein Term t heißt wohlgetypt,
wenn es einen Typ T gibt, so dass $t : T$.

Die Typisierungsrelation

Idee: Termen können Typen zugewiesen werden.

$$t : T$$

Beispiele

$\lambda x : nat.\ x :$	$nat \to nat$
$\lambda x : nat \to nat.\ x :$	$(nat \to nat) \to (nat \to nat)$
$\lambda x : nat.\ \lambda y : nat \to nat.\ y\ x :$	$nat \to (nat \to nat) \to nat$

Definition

Ein Term t heißt wohlgetypt,
wenn es einen Typ T gibt, so dass $t : T$.

Typchecker

$ty : t \mapsto T$

ty (Versuch)

ty (Versuch)

$ty(t_1\ t_2) =$ Applikation

ty (Versuch)

$ty(t_1\ t_2) =$
  case $ty(t_1)$ of

Applikation

ty (Versuch)

Applikation

$$
\begin{array}{l}
ty(t_1\ t_2) = \\
\quad \text{case } ty(t_1) \text{ of} \\
\quad\quad T \to T' \ \Rightarrow\ \text{if } ty(t_2) = T \\
\quad\quad\quad\quad\quad\quad\ \text{then } T' \text{ else throw } \mathit{Exception}
\end{array}
$$

ty (Versuch)

Applikation

$$
\begin{array}{l}
ty(t_1\ t_2) = \\
\quad \text{case } ty(t_1) \text{ of} \\
\qquad T \to T' \ \Rightarrow\ \text{if } ty(t_2) = T \\
\qquad\qquad\qquad\quad \text{then } T' \text{ else throw } \mathit{Exception} \\
\quad \mid \_ \ \Rightarrow \text{throw } \mathit{Exception}
\end{array}
$$

ty (Versuch)

$ty(t_1\ t_2) =$ Applikation

  case $ty(t_1)$ of

    $T \to T' \Rightarrow$ if $ty(t_2) = T$

      then T' else throw *Exception*

  | _ $\Rightarrow$ throw *Exception*

$ty(\lambda x : T.t) = T \to ty(t)$ Abstraktion

ty (Versuch)

$ty(t_1\ t_2) =$ Applikation

  case $ty(t_1)$ of

    $T \to T' \Rightarrow$ if $ty(t_2) = T$

      then T' else throw *Exception*

  $\mid \_ \Rightarrow$ throw *Exception*

$ty(\lambda x : T.t) = T \to ty(t)$ Abstraktion

$ty(x) = ?$ Variable

ty (Versuch)

$ty(t_1\ t_2) =$ Applikation

case $ty(t_1)$ of

$T \to T' \Rightarrow$ if $ty(t_2) = T$ then T' else throw *Exception*

$\mid \_ \Rightarrow$ throw *Exception*

$ty(\lambda x : T.t) = T \to ty(t)$ Abstraktion

$ty(x) = ?$ Variable

$\Rightarrow$ Die Typen der Variablen müssen verwaltet werden!

Lösung

Kontext Γ,

Lösung

Kontext Γ, bildet Variablen auf Typen ab.

Lösung

Kontext Γ, bildet Variablen auf Typen ab.

Beispiele

- $\Gamma := [x : \mathit{nat}, y : \mathit{bool}]$
  $\Gamma(x) = \mathit{nat}$
  $\Gamma(y) = \mathit{bool}$

Lösung

Kontext Γ, bildet Variablen auf Typen ab.

Beispiele

- $\Gamma := [x : \mathit{nat}, y : \mathit{bool}]$
  $\Gamma(x) = \mathit{nat}$
  $\Gamma(y) = \mathit{bool}$

- $\Gamma' := \Gamma[y : \mathit{nat}, z : \mathit{bool}]$

Lösung

Kontext Γ, bildet Variablen auf Typen ab.

Beispiele

- $\Gamma := [x : \mathit{nat}, y : \mathit{bool}]$
  $\Gamma(x) = \mathit{nat}$
  $\Gamma(y) = \mathit{bool}$

- $\Gamma' := \Gamma[y : \mathit{nat}, z : \mathit{bool}]$
  $\Gamma'(y) = \mathit{nat}$
  $\Gamma'(z) = \mathit{bool}$

$ty(t_1\ t_2) =$ Applikation

case $ty(t_1)$ of

$T \to T' \Rightarrow$ if $ty(t_2) = T$

then T' else throw *Exception*

$\mid \_ \Rightarrow$ throw *Exception*

$ty(\lambda x : T.t) = T \to ty(t)$ Abstraktion

$ty(x) =\ ?$ Variable

$ty(\Gamma, t_1\ t_2) =$ Applikation

  case $ty(\Gamma, t_1)$ of

    $T \to T' \Rightarrow$ if $ty(\Gamma, t_2) = T$

      then T' else throw *Exception*

    $\mid \_ \Rightarrow$ throw *Exception*

$ty(\Gamma, \lambda x : T.t) = T \to ty(\Gamma, t)$ Abstraktion

$ty(\Gamma, x) =\ ?$ Variable

$$ty(\Gamma, t_1\ t_2) = \qquad \text{Applikation}$$
$$\quad \text{case } ty(\Gamma, t_1) \text{ of}$$
$$\quad\ T \to T' \ \Rightarrow\ \text{if } ty(\Gamma, t_2) = T$$
$$\qquad\qquad\qquad \text{then } T' \text{ else throw } \mathit{Exception}$$
$$\quad\ \mid \_ \ \Rightarrow\ \text{throw } \mathit{Exception}$$

$$ty(\Gamma, \lambda x : T.t) = T \to ty(\Gamma, t) \qquad \text{Abstraktion}$$

$$ty(\Gamma, x) = \Gamma(x) \qquad \text{Variable}$$

ty

$$ty(\Gamma, t_1\ t_2) = \qquad \text{Applikation}$$

$$\begin{array}{l} \quad \text{case } ty(\Gamma, t_1) \text{ of} \\ \quad\ T \to T' \ \Rightarrow\ \text{if } ty(\Gamma, t_2) = T \\ \qquad\qquad\qquad\ \text{then } T' \text{ else throw } \mathit{Exception} \\ \quad\ \mid \_ \ \Rightarrow\ \text{throw } \mathit{Exception} \end{array}$$

$$ty(\Gamma, \lambda x : T.t) = T \to ty(\Gamma[x : T], t) \qquad \text{Abstraktion}$$

$$ty(\Gamma, x) = \Gamma(x) \qquad \text{Variable}$$

Formalisierung von *ty*

Gesucht: Mathematische Definition der Typisierungsrelation

Formalisierung von *ty*

Gesucht: Mathematische Definition der Typisierungsrelation

Notation

$$ty(\Gamma, t) = T \quad \rightsquigarrow \quad \Gamma \vdash t : T$$

Formalisierung von *ty*

Gesucht: Mathematische Definition der Typisierungsrelation

Notation

$$ty(\Gamma, t) = T \quad \rightsquigarrow \quad \Gamma \vdash t : T$$

Definition der Relation durch Typisierungsregeln

Formalisierung von *ty*

Gesucht: Mathematische Definition der Typisierungsrelation

Notation

$$ty(\Gamma, t) = T \quad \rightsquigarrow \quad \Gamma \vdash t : T$$

Definition der Relation durch Typisierungsregeln
= Schlussregeln für $\Gamma \vdash t : T$

Typisierungsregeln Var und Abs

$$ty(\Gamma, x) = \Gamma(x) \quad \rightsquigarrow \quad \frac{\Gamma(x) \text{ ist definiert}}{\Gamma \vdash x : \Gamma(x)} \qquad \text{Var}$$

Typisierungsregeln Var und Abs

$$ty(\Gamma, x) = \Gamma(x) \quad \rightsquigarrow \quad \frac{\Gamma(x) \text{ ist definiert}}{\Gamma \vdash x : \Gamma(x)} \qquad \text{Var}$$

$$ty(\Gamma, \lambda x : T.t) = T \to ty(\Gamma[x : T], t) \quad \rightsquigarrow \quad \frac{\Gamma[x : T] \vdash t : T'}{\Gamma \vdash \lambda x : T.t \ : \ T \to T'} \qquad \text{Abs}$$

Typisierungsregel App

$$ty(\Gamma, t_1\ t_2) = \text{case } ty(\Gamma, t_1) \text{ of}$$
$$\quad T \to T' \Rightarrow \text{ if } ty(\Gamma, t_2) = T \text{ then } T' \text{ else throw } \mathit{Exception}$$
$$\quad \mid \_ \Rightarrow \text{throw } \mathit{Exception}$$

$$\rightsquigarrow$$

$$\frac{\Gamma \vdash t_1 : T \to T' \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1\ t_2 : T'}\ \text{App}$$

Alle Typisierungsregeln

$$\frac{\Gamma(x) \text{ ist definiert}}{\Gamma \vdash x : \Gamma(x)} \quad \text{Var}$$

$$\frac{\Gamma[x : T] \vdash t : T'}{\Gamma \vdash \lambda x : T.t \ : \ T \to T'} \quad \text{Abs}$$

$$\frac{\Gamma \vdash t_1 : T \to T' \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1 \ t_2 \ : \ T'} \quad \text{App}$$

Eigenschaften der Typisierungsregeln

- Von unten nach oben angewandt entsprechen die Regeln genau dem Algorithmus *ty*

Eigenschaften der Typisierungsregeln

- Von unten nach oben angewandt entsprechen die Regeln genau dem Algorithmus *ty*
- Für jede Art von Term gibt es genau eine Regel

Eigenschaften der Typisierungsregeln

- Von unten nach oben angewandt entsprechen die Regeln genau dem Algorithmus *ty*
- Für jede Art von Term gibt es genau eine Regel
  $\Rightarrow$ Typprüfung ist deterministisch

Eigenschaften der Typisierungsregeln

- Von unten nach oben angewandt entsprechen die Regeln genau dem Algorithmus *ty*
- Für jede Art von Term gibt es genau eine Regel
  $\Rightarrow$ Typprüfung ist deterministisch
- Bei Anwendung einer Regel auf einen Term t werden ausschließlich Teilterme von t betrachtet

Eigenschaften der Typisierungsregeln

- Von unten nach oben angewandt entsprechen die Regeln genau dem Algorithmus *ty*
- Für jede Art von Term gibt es genau eine Regel
  $\Rightarrow$ Typprüfung ist deterministisch
- Bei Anwendung einer Regel auf einen Term t werden ausschließlich Teilterme von t betrachtet
  $\Rightarrow$ Typprüfung terminiert

Zentrale Sätze

- Der Typ eines wohlgetypten Termes ist eindeutig bestimmt

Zentrale Sätze

- Der Typ eines wohlgetypten Termes ist eindeutig bestimmt
- Type Preservation:
  Wenn $\Gamma \vdash t : T$ und $t \to_\beta t'$, dann $\Gamma \vdash t' : T$.

Zentrale Sätze

- Der Typ eines wohlgetypten Termes ist eindeutig bestimmt
- Type Preservation:
  Wenn $\Gamma \vdash t : T$ und $t \to_\beta t'$, dann $\Gamma \vdash t' : T$.
  $\Rightarrow$ keine Typfehler zur Laufzeit

Zentrale Sätze

- Der Typ eines wohlgetypten Termes ist eindeutig bestimmt
- Type Preservation:
  Wenn $\Gamma \vdash t : T$ und $t \to_\beta t'$, dann $\Gamma \vdash t' : T$.
  $\Rightarrow$ keine Typfehler zur Laufzeit
- Strong Normalization:
  Beta-Reduktion terminiert auf wohlgetypten Termen

Zentrale Sätze

- Der Typ eines wohlgetypten Termes ist eindeutig bestimmt
- Type Preservation:
  Wenn $\Gamma \vdash t : T$ und $t \rightarrow_\beta t'$, dann $\Gamma \vdash t' : T$.
  $\Rightarrow$ keine Typfehler zur Laufzeit
- Strong Normalization:
  Beta-Reduktion terminiert auf wohlgetypten Termen
  (Keine Selbstapplikation, daher keine Rekursion möglich)

Zentrale Sätze

- Der Typ eines wohlgetypten Termes ist eindeutig bestimmt
- Type Preservation:
  Wenn $\Gamma \vdash t : T$ und $t \rightarrow_\beta t'$, dann $\Gamma \vdash t' : T$.
  $\Rightarrow$ keine Typfehler zur Laufzeit
- Strong Normalization:
  Beta-Reduktion terminiert auf wohlgetypten Termen
  (Keine Selbstapplikation, daher keine Rekursion möglich)
- Nicht alle berechenbare Funktionen sind als wohlgetypte λ-Terme darstellbar.

Zentrale Sätze

- Der Typ eines wohlgetypten Termes ist eindeutig bestimmt
- Type Preservation:
  Wenn $\Gamma \vdash t : T$ und $t \rightarrow_\beta t'$, dann $\Gamma \vdash t' : T$.
  $\Rightarrow$ keine Typfehler zur Laufzeit
- Strong Normalization:
  Beta-Reduktion terminiert auf wohlgetypten Termen
  (Keine Selbstapplikation, daher keine Rekursion möglich)
- Nicht alle berechenbare Funktionen sind als wohlgetypte λ-Terme darstellbar.
- Hinzufügen eines primitiven Fixpunktoperators macht den einfach getypten λ-Kalkül Turing-vollständig

Implizite Typisierung

$$\lambda x : T.\ t \quad \rightsquigarrow \quad \lambda x.\ t$$

Problem

$\lambda x.\ x : \ ?$

Problem

$\lambda x.\ x : \ \mathit{bool} \to \mathit{bool}$

Problem

$\lambda x.\ x : \ \mathit{nat} \to \mathit{nat}$

Problem

$\lambda x.\ x : \ (\mathit{bool} \to \mathit{nat}) \to (\mathit{bool} \to \mathit{nat})$

Lösung: Typvariablen

Typen

$$
\begin{array}{lcll}
T & ::= & \alpha \mid \beta \mid \ldots & \text{Typvariablen} \\
 & \mid & \ldots & \text{wie vorher}
\end{array}
$$

Typinferenz

= Berechnung der fehlenden Typen

Typinferenz

= Berechnung der fehlenden Typen

Methode:

- Beginne mit $t : \alpha$ und

Typinferenz

= Berechnung der fehlenden Typen

Methode:

- Beginne mit $t : \alpha$ und
- berechne („inferiere") α inkrementell

Typinferenz

= Berechnung der fehlenden Typen

Methode:

- Beginne mit $t : \alpha$ und
- berechne („inferiere") α inkrementell
  bei der Rückwärts-Anwendung der Typisierungsregeln

Typinferenz

= Berechnung der fehlenden Typen

Methode:

- Beginne mit $t : \alpha$ und
- berechne („inferiere") α inkrementell
  bei der Rückwärts-Anwendung der Typisierungsregeln
  durch Unifikation der Typen.

Beispiel

$\lambda x.\ \lambda y.\ y\ x : ?$

Unifikation

Typinferenz erzeugt Menge von Gleichungen zwischen Typen.

Unifikation

Typinferenz erzeugt Menge von Gleichungen zwischen Typen.
Lösungsmethode Unifikation:

Unifikation

Typinferenz erzeugt Menge von Gleichungen zwischen Typen.
Lösungsmethode Unifikation:

$\alpha = T$: Ersetze α überall durch T.

Unifikation

Typinferenz erzeugt Menge von Gleichungen zwischen Typen.
Lösungsmethode Unifikation:

$\alpha = T$: Ersetze α überall durch T.
Aber nur, falls α nicht in T vorkommt!

Unifikation

Typinferenz erzeugt Menge von Gleichungen zwischen Typen.
Lösungsmethode Unifikation:

$\alpha = T$: Ersetze α überall durch T.
Aber nur, falls α nicht in T vorkommt!

$T = \alpha$: wie $\alpha = T$

Unifikation

Typinferenz erzeugt Menge von Gleichungen zwischen Typen.
Lösungsmethode Unifikation:

$\alpha = T$:	Ersetze α überall durch T. Aber nur, falls α nicht in T vorkommt!
$T = \alpha$:	wie $\alpha = T$
$T_1 \to T_2 = T_3 \to T_4$:	unifiziere $T_1 = T_3$ und $T_2 = T_4$

Selbstanwendung?

Frage

Warum ist keine Selbstandwendung (und daher keine Rekursion) in wohlgetypten Termen möglich?

Selbstanwendung?

Frage

Warum ist keine Selbstandwendung (und daher keine Rekursion) in wohlgetypten Termen möglich?

Erklärung

$$\lambda \quad x \quad . \quad x \quad x$$

Selbstanwendung?

Frage

Warum ist keine Selbstandwendung (und daher keine Rekursion) in wohlgetypten Termen möglich?

Erklärung

$$\begin{array}{cccccc} \lambda & x & . & x & & x \\ & \alpha & & & & \end{array}$$

Selbstanwendung?

Frage

Warum ist keine Selbstandwendung (und daher keine Rekursion) in wohlgetypten Termen möglich?

Erklärung

$$\begin{array}{ccccc} \lambda & x & . & x & x \\ & \alpha & & \alpha & \alpha \end{array}$$

Selbstanwendung?

Frage

Warum ist keine Selbstandwendung (und daher keine Rekursion) in wohlgetypten Termen möglich?

Erklärung

$$\begin{array}{cccccc} \lambda & x & . & x & & x \\ & \alpha & & \alpha & & \alpha \\ & & & \underbrace{\qquad\qquad} & & \\ & & & \alpha = \alpha \to \beta & & \end{array}$$

Haupteigenschaft der Typinferenz

Satz Jeder wohlgetypte Term hat einen allgemeinsten Typ.

Haupteigenschaft der Typinferenz

Satz Jeder wohlgetypte Term hat einen allgemeinsten Typ.

Beweisidee Weil Unifikation eine allgemeinste Lösung eines Gleichungsystems liefert.

Fazit

- Implizite Typisierung bietet die Vorteile von Typen ohne zusätzlichen Notationsaufwand.

Fazit

- Implizite Typisierung bietet die Vorteile von Typen ohne zusätzlichen Notationsaufwand.

- Typinferenz Standard in funktionalen Programmiersprachen.

Fazit

- Implizite Typisierung bietet die Vorteile von Typen ohne zusätzlichen Notationsaufwand.
- Typinferenz Standard in funktionalen Programmiersprachen.
- Warnung:
  Was das Schreiben erleichtert, kann das Lesen erschweren!