CS738: Advanced Compiler Optimizations

Simply Typed Lambda Calculus

Amey Karkare

karkare@cse.iitk.ac.in

http://www.cse.iitk.ac.in/~karkare/cs738
Department of CSE, IIT Kanpur

Reference Book

Types and Programming Languages by Benjamin C. Pierce

Simple Types over Bool

T :=   – Types

Simple Types over Bool

T :=   – Types
        Bool   – Boolean Type

Simple Types over Bool

$$
\begin{array}{lll}
T & := & \text{– Types} \\
& \text{Bool} & \text{– Boolean Type} \\
& T \rightarrow T & \text{– Function Type}
\end{array}
$$

Simple Types over Bool

$$T \quad := \qquad\qquad\qquad \text{– Types}$$

$$\quad\quad \text{Bool} \qquad \text{– Boolean Type}$$

$$\quad\quad T \rightarrow T \qquad \text{– Function Type}$$

type constructor $\rightarrow$ is right-associative, i.e., $T_1 \rightarrow T_2 \rightarrow T_3$
stands for $T_1 \rightarrow (T_2 \rightarrow T_3)$

Examples

For each of the type below, write a function (in your favourite programming language) that has the required type:

► Bool → Bool

Examples

For each of the type below, write a function (in your favourite programming language) that has the required type:

- Bool $\rightarrow$ Bool
- Bool $\rightarrow$ Bool $\rightarrow$ Bool

Examples

For each of the type below, write a function (in your favourite programming language) that has the required type:

- Bool $\rightarrow$ Bool
- Bool $\rightarrow$ Bool $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool) $\rightarrow$ Bool

Examples

For each of the type below, write a function (in your favourite programming language) that has the required type:

- Bool → Bool
- Bool → Bool → Bool
- (Bool → Bool) → Bool
- (Bool → Bool) → Bool → Bool

Examples

For each of the type below, write a function (in your favourite programming language) that has the required type:

- Bool $\rightarrow$ Bool
- Bool $\rightarrow$ Bool $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool) $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool) $\rightarrow$ Bool $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool $\rightarrow$ Bool) $\rightarrow$ Bool

Examples

For each of the type below, write a function (in your favourite programming language) that has the required type:

- Bool $\rightarrow$ Bool
- Bool $\rightarrow$ Bool $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool) $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool) $\rightarrow$ Bool $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool $\rightarrow$ Bool) $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool $\rightarrow$ Bool) $\rightarrow$ Bool

Examples

For each of the type below, write a function (in your favourite programming language) that has the required type:

- Bool $\rightarrow$ Bool
- Bool $\rightarrow$ Bool $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool) $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool) $\rightarrow$ Bool $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool $\rightarrow$ Bool) $\rightarrow$ Bool
- (Bool $\rightarrow$ Bool $\rightarrow$ Bool) $\rightarrow$ Bool
- ((Bool $\rightarrow$ Bool) $\rightarrow$ Bool) $\rightarrow$ Bool

The Abstract Syntax

Simply Typed λ-terms with conditions and Booleans

| t | := | x | – *Variable* |

The Abstract Syntax

Simply Typed λ-terms with conditions and Booleans

| t | := | x | – *Variable* |
| | | $\lambda x : T.\ t$ | – *Abstraction* |

The Abstract Syntax

Simply Typed λ-terms with conditions and Booleans

t	:=	x	– *Variable*
	|	$\lambda x : T.\ t$	– *Abstraction*
	|	t t	– *Application*

The Abstract Syntax

Simply Typed λ-terms with conditions and Booleans

t	:=	x	– *Variable*
		$\lambda x : T.\ t$	– *Abstraction*
		t t	– *Application*
		`true`	– *constant true*

The Abstract Syntax

Simply Typed λ-terms with conditions and Booleans

t	:=	x		– *Variable*
		$\lambda x : T.\, t$		– *Abstraction*
		t t		– *Application*
		`true`		– *constant true*
		`false`		– *constant false*

The Abstract Syntax

Simply Typed λ-terms with conditions and Booleans

$$
\begin{aligned}
t \quad := \quad & x & \text{– Variable} \\
| \quad & \lambda x : T.\, t & \text{– Abstraction} \\
| \quad & t\, t & \text{– Application} \\
| \quad & \texttt{true} & \text{– constant true} \\
| \quad & \texttt{false} & \text{– constant false} \\
| \quad & \texttt{if}\, t\, \texttt{then}\, t\, \texttt{else}\, t & \text{– conditional}
\end{aligned}
$$

Recap: The Set of Values

$$v ::= \qquad\qquad \text{– } \textit{values}$$
$$\lambda x : T.\, t \quad \text{– } \textit{Abstraction Value}$$

Recap: The Set of Values

$$
\begin{array}{lll}
v & := & \quad \text{– } \textit{values} \\
  & \lambda x : T.\, \mathrm{t} & \quad \text{– } \textit{Abstraction Value} \\
  & |\quad \mathtt{true} & \quad \text{– } \textit{value true}
\end{array}
$$

Recap: The Set of Values

$$
\begin{array}{lll}
v & := & \text{– } \textit{values} \\
& \lambda x : T.\ t & \text{– } \textit{Abstraction Value} \\
| & \texttt{true} & \text{– } \textit{value true} \\
| & \texttt{false} & \text{– } \textit{value false}
\end{array}
$$

Evaluation

$$\frac{t_1 \rightarrow t_1'}{t_1 \; t_2 \rightarrow t_1' \; t_2} \qquad \text{(E-APP1)}$$

Evaluation

$$\frac{t_1 \rightarrow t_1'}{t_1 \; t_2 \rightarrow t_1' \; t_2} \qquad \text{(E-App1)}$$

$$\frac{t_2 \rightarrow t_2'}{v \; t_2 \rightarrow v \; t_2'} \qquad \text{(E-App2)}$$

Evaluation

$$\frac{t_1 \to t_1'}{t_1 \ t_2 \to t_1' \ t_2} \qquad \text{(E-APP1)}$$

$$\frac{t_2 \to t_2'}{v \ t_2 \to v \ t_2'} \qquad \text{(E-APP2)}$$

$$(\lambda x : T_1 . \ t_1) v_2 \to [x \mapsto v_2] t_1 \qquad \text{(E-APPABS)}$$

The Typing Relation

- A *Typing Context* or *Type Environment*, Γ, is a sequence of variables with their types

The Typing Relation

- ▶ A *Typing Context* or *Type Environment*, Γ, is a sequence of variables with their types
- ▶ Γ, $x : T$ denotes extending Γ with a new variable x having type T

The Typing Relation

- A *Typing Context* or *Type Environment*, Γ, is a sequence of variables with their types
- $\Gamma, x : T$ denotes extending Γ with a new variable x having type T
  - The name x is assumed to be distinct from any existing names in Γ

The Typing Relation

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.\, t_2 : T_1 \to T_2} \qquad \text{(T-ABS)}$$

The Typing Relation

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.\, t_2 : T_1 \rightarrow T_2} \qquad \text{(T-ABS)}$$

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

The Typing Relation

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1 . \, t_2 : T_1 \to T_2} \qquad \text{(T-ABS)}$$

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

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1 \, t_2 : T_2} \qquad \text{(T-APP)}$$

Inversion of the Typing Relation

- If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.

Inversion of the Typing Relation

- ▶ If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.
- ▶ If $\Gamma \vdash \lambda x : T_1. \, t_2 : R$, then $R = T_1 \to R_2$ for some R_2 with $\Gamma, x : T_1 \vdash t_2 : R_2$.

Inversion of the Typing Relation

- If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.
- If $\Gamma \vdash \lambda x : T_1. \, t_2 : R$, then $R = T_1 \to R_2$ for some R_2 with $\Gamma, x : T_1 \vdash t_2 : R_2$.
- If $\Gamma \vdash t_1 \, t_2 : R$, then $\exists T_1 \; s.t. \; \Gamma \vdash t_1 : T_1 \to R$ and $\Gamma \vdash t_2 : T_1$.

Inversion of the Typing Relation

- If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.
- If $\Gamma \vdash \lambda x : T_1. t_2 : R$, then $R = T_1 \to R_2$ for some R_2 with $\Gamma, x : T_1 \vdash t_2 : R_2$.
- If $\Gamma \vdash t_1\ t_2 : R$, then $\exists T_1\ s.t.\ \Gamma \vdash t_1 : T_1 \to R$ and $\Gamma \vdash t_2 : T_1$.
- If $\Gamma \vdash \texttt{true} : R$, then $R = \text{Bool}$.

Inversion of the Typing Relation

- ▶ If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.
- ▶ If $\Gamma \vdash \lambda x : T_1. \ t_2 : R$, then $R = T_1 \rightarrow R_2$ for some R_2 with $\Gamma, x : T_1 \vdash t_2 : R_2$.
- ▶ If $\Gamma \vdash t_1 \ t_2 : R$, then $\exists T_1 \ s.t. \ \Gamma \vdash t_1 : T_1 \rightarrow R$ and $\Gamma \vdash t_2 : T_1$.
- ▶ If $\Gamma \vdash \texttt{true} : R$, then $R = \text{Bool}$.
- ▶ If $\Gamma \vdash \texttt{false} : R$, then $R = \text{Bool}$.

Inversion of the Typing Relation

- If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.
- If $\Gamma \vdash \lambda x : T_1. t_2 : R$, then $R = T_1 \to R_2$ for some R_2 with $\Gamma, x : T_1 \vdash t_2 : R_2$.
- If $\Gamma \vdash t_1\ t_2 : R$, then $\exists T_1\ s.t.\ \Gamma \vdash t_1 : T_1 \to R$ and $\Gamma \vdash t_2 : T_1$.
- If $\Gamma \vdash \texttt{true} : R$, then $R = \text{Bool}$.
- If $\Gamma \vdash \texttt{false} : R$, then $R = \text{Bool}$.
- If $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : R$, then

Inversion of the Typing Relation

- If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.
- If $\Gamma \vdash \lambda x : T_1.\ t_2 : R$, then $R = T_1 \to R_2$ for some R_2 with $\Gamma, x : T_1 \vdash t_2 : R_2$.
- If $\Gamma \vdash t_1\ t_2 : R$, then $\exists T_1\ s.t.\ \Gamma \vdash t_1 : T_1 \to R$ and $\Gamma \vdash t_2 : T_1$.
- If $\Gamma \vdash \texttt{true} : R$, then $R = \text{Bool}$.
- If $\Gamma \vdash \texttt{false} : R$, then $R = \text{Bool}$.
- If $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : R$, then
    - $\Gamma \vdash t_1 : \text{Bool}$

Inversion of the Typing Relation

- If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.
- If $\Gamma \vdash \lambda x : T_1.\ t_2 : R$, then $R = T_1 \rightarrow R_2$ for some R_2 with $\Gamma, x : T_1 \vdash t_2 : R_2$.
- If $\Gamma \vdash t_1\ t_2 : R$, then $\exists T_1\ s.t.\ \Gamma \vdash t_1 : T_1 \rightarrow R$ and $\Gamma \vdash t_2 : T_1$.
- If $\Gamma \vdash \texttt{true} : R$, then $R = \text{Bool}$.
- If $\Gamma \vdash \texttt{false} : R$, then $R = \text{Bool}$.
- If $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : R$, then
    - $\Gamma \vdash t_1 : \text{Bool}$
    - $\Gamma \vdash t_2 : R$

Inversion of the Typing Relation

- If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.
- If $\Gamma \vdash \lambda x : T_1. \, t_2 : R$, then $R = T_1 \rightarrow R_2$ for some R_2 with $\Gamma, x : T_1 \vdash t_2 : R_2$.
- If $\Gamma \vdash t_1 \, t_2 : R$, then $\exists T_1 \; s.t. \; \Gamma \vdash t_1 : T_1 \rightarrow R$ and $\Gamma \vdash t_2 : T_1$.
- If $\Gamma \vdash \texttt{true} : R$, then $R = \text{Bool}$.
- If $\Gamma \vdash \texttt{false} : R$, then $R = \text{Bool}$.
- If $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : R$, then
    - $\Gamma \vdash t_1 : \text{Bool}$
    - $\Gamma \vdash t_2 : R$
    - $\Gamma \vdash t_3 : R$

- For each of the term t below, find context Γ and type T such that
$$\Gamma \vdash t : T$$

Exercises

- For each of the term t below, find context Γ and type T such that

$$\Gamma \vdash t : T$$

  - t is $\lambda x.\ x$

Exercises

- For each of the term t below, find context Γ and type T such that

$$\Gamma \vdash t : T$$

  - t is $\lambda x.\, x$
  - t is $(\,(x\,z)\,(y\,z)\,)$

Exercises

- For each of the term t below, find context Γ and type T such that

$$\Gamma \vdash t : T$$

  - t is $\lambda x.\ x$
  - t is $(\ (x\ z)\ (y\ z)\ )$
  - t is $\lambda y.\ x$

Exercises

▶ For each of the term t below, find context Γ and type T such that

$$\Gamma \vdash t : T$$

  ▶ t is $\lambda x.\ x$
  ▶ t is $(\ (x\ z)\ (y\ z)\ )$
  ▶ t is $\lambda y.\ x$
  ▶ t is $x\ x$

Uniqueness of Types

- In a given type context Γ, A term t, such that the free variables of t are in Γ, has at most one type.

Uniqueness of Types

- In a given type context Γ, A term t, such that the free variables of t are in Γ, has at most one type.
- If t is typeable, then its type is unique.

Uniqueness of Types

- In a given type context Γ, A term t, such that the free variables of t are in Γ, has at most one type.
- If t is typeable, then its type is unique.
- Moreover, there is just one derivation of this typing built from the inference rules.

Some Properties

- **Permutation:** If $\Gamma \vdash t : T$ and Δ is a permutation of Γ, then $\Delta \vdash t : T$.

- **Permutation:** If $\Gamma \vdash t : T$ and Δ is a permutation of Γ, then $\Delta \vdash t : T$.
    - The derivation with Δ has the same depth as the derivation with Γ.

Some Properties

- ▶ **Permutation:** If $\Gamma \vdash t : T$ and Δ is a permutation of Γ, then $\Delta \vdash t : T$.
    - ▶ The derivation with Δ has the same depth as the derivation with Γ.
- ▶ **Weakening:** If $\Gamma \vdash t : T$ and $x \notin \text{domain}(\Gamma)$, then $\Gamma, x : S \vdash t : T$.

Some Properties

- **Permutation:** If $\Gamma \vdash t : T$ and Δ is a permutation of Γ, then $\Delta \vdash t : T$.
    - The derivation with Δ has the same depth as the derivation with Γ.
- **Weakening:** If $\Gamma \vdash t : T$ and $x \notin \text{domain}(\Gamma)$, then $\Gamma, x : S \vdash t : T$.
    - The derivation with $\Gamma, x : S$ has the same depth as the derivation with Γ.

Progress

- **Progress:** A well-typed term is not stuck.

Progress

- **Progress:** A well-typed term is not stuck.
    - If $\vdash t : T$, then t is either a value or there exists some t′ such that $t \to t'$.

Preservation

- **Preservation of Types under Substitution:** If
  $\Gamma, x : S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.

Preservation

- **Preservation of Types under Substitution:** If
  $\Gamma, x : S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.
- **Preservation:** If a well-typed term takes a step of
  evaluation, then the resulting term is also well-typed.

Preservation

- **Preservation of Types under Substitution:** If
  $\Gamma, x : S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.
- **Preservation:** If a well-typed term takes a step of
  evaluation, then the resulting term is also well-typed.
  - If $\Gamma \vdash t : T$ and $t \rightarrow t'$, then $\Gamma \vdash t' : T$.