CS520, Handout 07, 21 September 2006

(modified: 24 September 2006)

Simply Typed Lambda Calculus

Reading: [Pierce, Chapter 9]

Types, Terms, Typing Rules, and Evaluation Rules of $\lambda_{\rightarrow}$ [Pierce, Figure 9-1]

Syntax of Simple Types

T ::=		*types*
	$\bullet$	*base type*
	$T \rightarrow T$	*type of functions*

"$\bullet$" is a type constant – or a metavariable ranging over type constants: $\mathtt{Bool}, \mathtt{Nat}, \ldots$

Syntax of Terms

t ::=		*terms*
	x	*variable*
	$\lambda x : T . t$	*abstraction*
	$t\ t$	*application*
v ::=		*values*
	$\lambda x : T . t$	*abstraction value*

Typing Contexts

Γ ::=		*contexts*
	$\varnothing$	*empty context*
	$\Gamma,\ x : T$	*term-variable binding*

A context Γ is a finite function from term variables to types. $dom(\Gamma)$ is the set of variables to which Γ assigns types.

Typing Rules

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

$$\frac{\Gamma, x : T \vdash t : T'}{\Gamma \vdash (\lambda x : T . t) : T \rightarrow T'} \quad \text{(T-ABS)}$$

$$\frac{\Gamma \vdash t_1 : T \rightarrow T' \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1\ t_2 : T'} \quad \text{(T-APP)}$$

Evaluation Rules

$$\frac{t \Rightarrow t'}{t\ u \Rightarrow t'\ u} \quad \text{(E-APP1)}$$

$$\frac{t \Rightarrow t'}{v\ t \Rightarrow v\ t'} \quad \text{(E-APP2)}$$

$$(\lambda x : T . t)\ v \Rightarrow [x \mapsto v]t \quad \text{(E-AppAbs)}$$

Remark about [Pierce, Section 9.2 and Section 9.3]

The presentation in Sect 9.2 is meant to be for the **pure simply-typed lambda-calculus**, often denoted $\boldsymbol{\lambda}_{\rightarrow}$, but it is not quite. And Fig 9-1 is intended to summarize the whole system $\boldsymbol{\lambda}_{\rightarrow}$, but this summary is not quite complete, as it omits the base type (denoted $\bullet$ on page 2 of these slides) in the syntax of types.

The confusion results from taking $\bullet = \texttt{Bool}$ and giving an example (middle of page 102) and an exercise (Ex. 9.2.2 on page 103) that involve `if-then-else`. With boolean expressions and conditionals added, we do not deal with the pure simply-typed lambda-calculus $\boldsymbol{\lambda}_{\rightarrow}$ as usually understood. Rather, following Pierce's convention, the simply-typed lambda-calculus thus augmented should be called $\boldsymbol{\lambda}\mathbf{B}_{\rightarrow}$.

The results in Sect 9.3 are for the augmented $\boldsymbol{\lambda}\mathbf{B}_{\rightarrow}$ not just for the pure $\boldsymbol{\lambda}_{\rightarrow}$.

For a formal presentation of $\boldsymbol{\lambda}\mathbf{B}_{\rightarrow}$ we need to add to Figure 9-1:

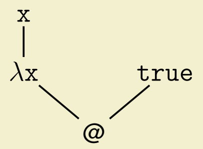
- Figure 3-1 on page 34, for the syntax of boolean terms and their evaluation.
- Figure 8-1 on page 93, for the syntax of types and the typing rules for boolean terms.

Example of a Typing Derivation

Using the rules of $\boldsymbol{\lambda}\mathbf{B}_{\rightarrow}$

$$\dfrac{\dfrac{\dfrac{(\mathtt{x}:\mathtt{Bool}) \in (\mathtt{x}:\mathtt{Bool})}{\mathtt{x}:\mathtt{Bool} \vdash \mathtt{x}:\mathtt{Bool}}\ \textsc{T-Var}}{\vdash \lambda\mathtt{x}:\mathtt{Bool}.\,\mathtt{x}:\mathtt{Bool}\rightarrow\mathtt{Bool}}\ \textsc{T-Abs} \qquad \dfrac{}{\vdash \mathtt{true}:\mathtt{Bool}}\ \textsc{T-True}}{\vdash (\lambda\mathtt{x}:\mathtt{Bool}.\,\mathtt{x})\ \mathtt{true}:\mathtt{Bool}}\ \textsc{T-App}$$

The **typing derivation** has the same shape as the **abstract syntax tree** (upside down):



i.e., the typing derivation is **syntax-directed**.

Safety = Progress + Preservation [Pierce, Section 9.3]

Progress [Pierce, Theorem 9.3.5]
Let t be a closed term. If t is typable, i.e., $\vdash t : T$ is derivable for some type T, then either t is a value or else $t \Rightarrow t'$ for some term t'.

The progress theorem fails if t is not closed. For example, the term $(\mathtt{x}\ \mathtt{true})$ is a normal form but not a value.

Preservation [Pierce, Theorem 9.3.9]
If $\Gamma \vdash t : T$ is derivable and $t \Rightarrow t'$, then $\Gamma \vdash t' : T$ is derivable.

This is "preservation" in the strong sense that the **typing**, not just **typability**, is preserved. This form of stronger preservation is sometimes called **subject-reduction**.

Safety = Progress + Preservation (continued)

Two fundamental technical lemmas are used in the proof of the preservation theorem.

Weakening Lemma [Pierce, Lemma 9.3.7]
If $\Gamma \vdash t : T$ is derivable and $x \notin dom(\Gamma)$, then $\Gamma, x : U \vdash t : T$ is derivable.

Substitution Lemma [Pierce, Lemma 9.3.8]
If $\Gamma, x : U \vdash t : T$ and $\Gamma \vdash u : U$ are derivable, then $\Gamma \vdash [x \mapsto u]t : T$ is derivable.

Curry-Howard Correspondence

[Pierce, Section 9.4]

Also called **formulas-as-types** correspondence.

INTUITIONISTIC LOGIC	TYPED LAMBDA CALCULUS
formulas	types
formula $T \supset U$	type $T \to U$
formula $T \wedge U$	type $T \times U$
proof of formula T	term t of type T
formula T is provable	type T is inhabited by some term
$\vdots$	$\vdots$

Intuitionistic propositional logic, sometimes called **implicational logic**.

Syntax of Propositional Formulas

(compare with the **syntax of types**)

T ::=		*propositional formulas*
	a	*propositional variable*
	b	*propositional variable*
	$\ldots$	*propositional variable*
	$T \to T$	*implication*

Inference Rules

(compare with the **typing rules**)

$$\frac{T \in \Gamma}{\Gamma \vdash T} \quad (\text{Ax})$$

$$\frac{\Gamma,\ T \vdash T'}{\Gamma \vdash T \to T'} \quad (\to\text{INTRO})$$

$$\frac{\Gamma \vdash T \to T' \qquad \Gamma \vdash T}{\Gamma \vdash T'} \quad (\to\text{ELIM})$$

Example of a Proof in Implicational Logic

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{b}\to\mathsf{c} \in \mathsf{a}\to\mathsf{b},\mathsf{b}\to\mathsf{c},\mathsf{a}}{\mathsf{a}\to\mathsf{b},\mathsf{b}\to\mathsf{c},\mathsf{a}\vdash\mathsf{b}\to\mathsf{c}}(\text{Ax}) \quad \dfrac{\dfrac{\mathsf{a}\to\mathsf{b} \in \mathsf{a}\to\mathsf{b},\mathsf{b}\to\mathsf{c},\mathsf{a}}{\mathsf{a}\to\mathsf{b},\mathsf{b}\to\mathsf{c},\mathsf{a}\vdash\mathsf{a}\to\mathsf{b}}(\text{Ax}) \quad \dfrac{\mathsf{a} \in \mathsf{a}\to\mathsf{b},\mathsf{b}\to\mathsf{c},\mathsf{a}}{\mathsf{a}\to\mathsf{b},\mathsf{b}\to\mathsf{c},\mathsf{a}\vdash\mathsf{a}}(\text{Ax})}{\mathsf{a}\to\mathsf{b},\mathsf{b}\to\mathsf{c},\mathsf{a}\vdash\mathsf{b}}(\to\text{ELIM})}{\mathsf{a}\to\mathsf{b},\mathsf{b}\to\mathsf{c},\mathsf{a}\vdash\mathsf{c}}(\to\text{ELIM})}{\mathsf{a}\to\mathsf{b},\mathsf{b}\to\mathsf{c}\vdash\mathsf{a}\to\mathsf{c}}(\to\text{INTRO})}{\mathsf{a}\to\mathsf{b}\vdash(\mathsf{b}\to\mathsf{c})\to(\mathsf{a}\to\mathsf{c})}(\to\text{INTRO})}{\vdash(\mathsf{a}\to\mathsf{b})\to(\mathsf{b}\to\mathsf{c})\to\mathsf{a}\to\mathsf{c}}(\to\text{INTRO})$$

Conclusion: $(\mathsf{a}\to\mathsf{b})\to(\mathsf{b}\to\mathsf{c})\to\mathsf{a}\to\mathsf{c}$ is a **tautology** of implicational logic

i.e., it is provable (from the empty sequence of assumptions).

Formulas-as-Types Correspondence (not in [Pierce, Section 9.4])

If T is a simple type (now built up from the base types $\mathsf{a}, \mathsf{b}, \ldots$)
then the following statements are equivalent:

- $\vdash t : T$ is a derivable for some term t (in the simply-typed lambda calculus),
- T is a tautology of implicational logic (with "$\to$" read as implication).

The preceding is a theorem that requires a proof (omitted here).

Erasure and Typability [Pierce, Section 9.5]

Erasure of Types [Pierce, Definition 9.5.1]

$$\begin{aligned}\mathtt{erase}(x) &= x\\ \mathtt{erase}(\lambda x : T.\, t) &= \lambda x.\, \mathtt{erase}(t)\\ \mathtt{erase}(t\ u) &= \mathtt{erase}(t)\ \mathtt{erase}(u)\end{aligned}$$

Evaluation Commutes With Erasure [Pierce, Theorem 9.5.2]

1. If $t \Rightarrow u$ then $\mathtt{erase}(t) \Rightarrow \mathtt{erase}(u)$.
2. If $\mathtt{erase}(t) \Rightarrow u'$ then there is a simply-typed term u such that $t \Rightarrow u$ and $\mathtt{erase}(u) = u'$.

Typability [Pierce, Definition 9.5.3]
Let t be a term of the untyped lambda calculus. We say t is typable in $\boldsymbol{\lambda}_{\rightarrow}$ if there is a simply-typed term t', a type T and a context Γ, such that $\mathtt{erase}(t') = t$ and $\Gamma \vdash t' : T$ is derivable.

Comments on Type Inhabitation (not in [Pierce])

A type T is inhabited if there is a closed well-typed term t such that $\vdash t : T$.

Remark: The term t must be closed, otherwise the type T is always inhabited. (Why?)

Equivalent Questions: Let T be an arbitrary simple type.

- Is T inhabited?
- Is there a closed well-typed term which can be assigned type T?
- Is T a tautology of implicational logic?

Fact 1: A base type is not inhabited.

Fact 2: A type where every base type occurs at most once is not inhabited.

Fact 3: The type $((\mathrm{a} \rightarrow \mathrm{b}) \rightarrow \mathrm{a}) \rightarrow \mathrm{a}$ is not inhabited.