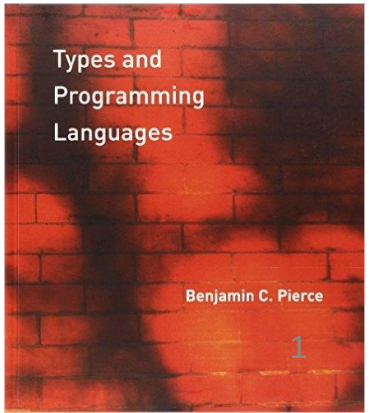
Concepts in Programming Languages – Recitation 7:

Typed Lambda Calculus

Oded Padon & Mooly Sagiv

(original slides by Kathleen Fisher, John Mitchell, Shachar Itzhaky, S. Tanimoto)

Reference:
Types and Programming Languages
by Benjamin C. Pierce, Chapter 9



Untyped Lambda Calculus - Syntax

t ::=		terms
	x	variable
	λ x. t	abstraction
	t t	application

- Terms can be represented as abstract syntax trees
- Syntactic Conventions:
  - Applications associates to left :
    $e_1\ e_2\ e_3 \equiv (e_1\ e_2)\ e_3$
  - The body of abstraction extends as far as possible:
    $\lambda x.\ \lambda y.\ x\ y\ x \equiv \lambda x.\ (\lambda y.\ ((x\ y)\ x))$

Simple Types

T ::=		types
	Bool	type of Booleans
	$T \rightarrow T$	type of functions

$T_1 \rightarrow T_2 \rightarrow T_3 = T_1 \rightarrow (T_2 \rightarrow T_3)$

Simple Typed Lambda Calculus

t ::=	terms	T::=	types
x	variable	Bool	atomic type for Booleans
λ x: T. t	abstraction	T $\rightarrow$ T	types of functions
t t	application		

Tying Relation $\vdash$

- Type fact: t:T means term t has type T (e.g. 1+3:Int, f:Int$\rightarrow$Int)
- Typing relation $\vdash$: relates sets of type facts and type facts
- $\Gamma \vdash$ t:T means under Γ (context, environment), term t has type T
- $\vdash$ t:T means term t has type T under the empty environment (no assumptions)
  - Can t have free variables?

Typing Relation Examples

- Type fact: t:T means term t has type T (e.g. 1+3:Int, f:Int→Int)
- $\Gamma \vdash$ t:T means under Γ (context, environment), term t has type T
- $\vdash$ t:T means closed term t has type T under the empty environment

Examples

- x:Int, y:Int $\vdash$ x+y : ?
- x:Int, y:Bool $\vdash$ x+y : ?
- x:Int $\vdash$ x+y : ?
- x:Int, y:Int, b:Bool $\vdash$ if b then x else y : ?
- x:Int, y:Bool, b:Bool $\vdash$ if b then x else y : ?
- x:Int, b:Bool $\vdash$ if b then x else y : ?
- x:Int, f:Int →Bool $\vdash$ f x : ?
- x:Int, f: Int → Int → Bool $\vdash$ f x x : ?
- x:Int, f: Int → Int → Bool $\vdash$ f x: ?
- f: Int → Int → Bool $\vdash$ f x x : ?
- $\vdash$ 1+2 : ?
- $\vdash$ true : ?
- x:Int, y:Int, f: Int → Int → Bool $\vdash$ 1+2 : ?

Formally Defining $\vdash$

t ::=	terms
x	variable
λ x: T. t	abstraction
t t	application
T::=	types
$T \rightarrow T$	types of functions
Γ::=	context
$\varnothing$	empty context
Γ, x : T	term variable binding

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

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

$$\frac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-APP)}$$

Adding Booleans

t ::=	terms
x	variable
λ x: T. t	abstraction
t t	application
true	constant true
false	constant false
if t then t else t	conditional
T::=	types
Bool	type of Booleans
T $\rightarrow$ T	types of functions
Γ::=	context
$\varnothing$	empty context
Γ, x : T	term variable binding

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

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

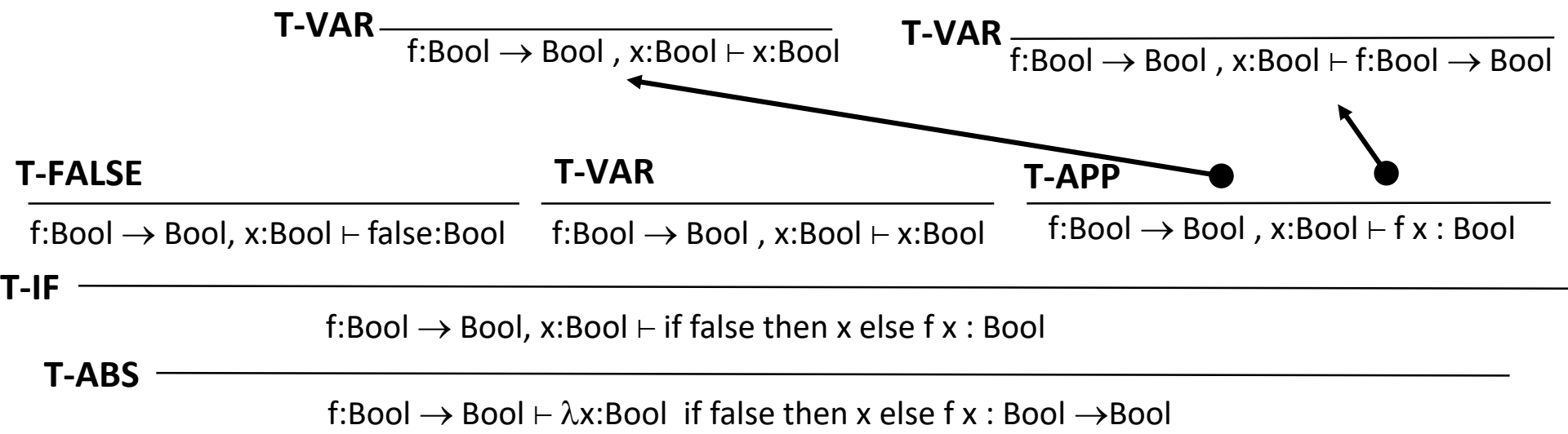
$$\frac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-APP)}$$

$$\Gamma \vdash \text{true} : \text{Bool} \quad \text{(T-TRUE)}$$

$$\Gamma \vdash \text{false} : \text{Bool} \quad \text{(T-FALSE)}$$

$$\frac{\Gamma \vdash t_1 : \text{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \quad \text{(T-IF)}$$

Example Typing Tree



$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}\ \textbf{(T-VAR)} \qquad \frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}}\ \textbf{(T-APP)} \qquad \frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.\ t_2 : T_1 \to T_2}\ \textbf{(T-ABS)}$$

$$\frac{\Gamma \vdash t_1 : \text{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T}\ \textbf{(T-IF)} \qquad \Gamma \vdash \text{false} : \text{Bool}\ \textbf{(T-FALSE)}$$

Typing is Static

- $\vdash$ if true then 1 else false : Int ?
- $\vdash$ if false then 1 else false : Bool ?

- "Compile time" vs. "Run time"

- What about operational semantics?
  - Ignore types, evaluate by beta reduction

Curry-Howard Isomorphism

Beautiful connection between type systems (PL) and proof systems (logic)

Propositional Logic	*Simply-Typed Lambda Calculus*
$\dfrac{\begin{array}{c}\overline{A}^{\,i}\\ \vdots\\ B\end{array}}{A \to B}\,(\to I),i$	$\dfrac{\begin{array}{c}\overline{x : A}^{\,i}\\ \vdots\\ B\end{array}}{\lambda x.M : A \to B}\,(\to I),i$
$\dfrac{A \quad A \to B}{B}\,(\to E)$	$\dfrac{N : A \quad M : A \to B}{MN : B}\,(\to E)$

conjunction	product type
$A \wedge B$	$A \times B$
disjunction	sum type
$A \vee B$	$A + B$
implication	function type
$A \supset B$	$A \to B$

Retrospect

Natural Operational Semantics $\rightarrow$

- $\rightarrow$ is defined **inductively** using **inference rules**, with both **syntactic** conditions on S and **semantic** conditions on s

$$[\text{ass}_{\text{ns}}] \quad \langle x := a,\ s\rangle \rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$[\text{skip}_{\text{ns}}] \quad \langle \texttt{skip},\ s\rangle \rightarrow s$$

$$[\text{comp}_{\text{ns}}] \quad \frac{\langle S_1,\ s\rangle \rightarrow s',\ \langle S_2,\ s'\rangle \rightarrow s''}{\langle S_1;S_2,\ s\rangle \rightarrow s''}$$

$$[\text{if}^{\text{tt}}_{\text{ns}}] \quad \frac{\langle S_1,\ s\rangle \rightarrow s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \rightarrow s'} \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$[\text{if}^{\text{ff}}_{\text{ns}}] \quad \frac{\langle S_2,\ s\rangle \rightarrow s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \rightarrow s'} \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

$$[\text{while}^{\text{tt}}_{\text{ns}}] \quad \frac{\langle S,\ s\rangle \rightarrow s',\ \langle \texttt{while}\ b\ \texttt{do}\ S,\ s'\rangle \rightarrow s''}{\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \rightarrow s''} \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$[\text{while}^{\text{ff}}_{\text{ns}}] \quad \langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \rightarrow s \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

Structural Operational Semantics ⇒

- ⇒ is defined **inductively** using **inference rules**, with both **syntactic** conditions on S and **semantic** conditions on s

$$[\text{ass}_{\text{sos}}] \qquad \langle x := a,\ s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$[\text{skip}_{\text{sos}}] \qquad \langle \texttt{skip},\ s\rangle \Rightarrow s$$

$$[\text{comp}^1_{\text{sos}}] \qquad \frac{\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_1';S_2,\ s'\rangle}$$

$$[\text{comp}^2_{\text{sos}}] \qquad \frac{\langle S_1,\ s\rangle \Rightarrow s'}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_2,\ s'\rangle}$$

$$[\text{if}^{\text{tt}}_{\text{sos}}] \qquad \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_1,\ s\rangle \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$[\text{if}^{\text{ff}}_{\text{sos}}] \qquad \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_2,\ s\rangle \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

$$[\text{while}_{\text{sos}}] \qquad \langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \Rightarrow \langle \texttt{if}\ b\ \texttt{then}\ (S;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else}\ \texttt{skip},\ s\rangle$$

λ-Calculus: Non-Deterministic Operational Semantics

(E-AppAbs) $(\lambda\, x.\, t_1)\; t_2 \Rightarrow [x \mapsto t_2]\; t_1$

$$\frac{t \Rightarrow t'}{\lambda\, x.\, t \Rightarrow \lambda\, x.\, t'}\ \text{(E-Abs)}$$

$$\text{(E-App}_1\text{)}\quad \frac{t_1 \Rightarrow t'_1}{t_1\ t_2 \Rightarrow t'_1\ t_2} \qquad \frac{t_2 \Rightarrow t'_2}{t_1\ t_2 \Rightarrow t_1\ t'_2}\ \text{(E-App}_2\text{)}$$

λ-Calculus: Lazy Evaluation Operational Semantics

(E-AppAbs) $(\lambda\, x.\, t_1)\; t_2 \Rightarrow [x \mapsto t_2]\; t_1$

(E-App$_1$)
$$\frac{t_1 \Rightarrow t'_1}{t_1\; t_2 \Rightarrow t'_1\; t_2}$$

λ-Calculus: Call-by-Value Operational Semantics

(E-AppAbs) $(\lambda\, x.\, t_1)\ v_2 \Rightarrow [x \mapsto v_2]\ t_1$

(E-App$_1$) $$\frac{t_1 \Rightarrow t'_1}{t_1\ t_2 \Rightarrow t'_1\ t_2}$$

$$\frac{t_2 \Rightarrow t'_2}{v_1\ t_2 \Rightarrow v_1\ t'_2}$$ (E-App$_2$)

Simply Typed λ-Calculus

t ::=	terms
x	variable
λ x: T. t	abstraction
t t	application
true	constant true
false	constant false
if t then t else t	conditional

T::=	types
Bool	type of Booleans
$T \to T$	types of functions

Γ::=	context
$\varnothing$	empty context
Γ, x : T	term variable binding

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

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

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-APP)}$$

$$\Gamma \vdash \text{true} : \text{Bool} \quad \text{(T-TRUE)}$$

$$\Gamma \vdash \text{false} : \text{Bool} \quad \text{(T-FALSE)}$$

$$\frac{\Gamma \vdash t_1 : \text{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \quad \text{(T-IF)}$$

Retrospect Conclusion:

For the past 8 weeks we have been formalizing programming languages Syntax, Semantics, and Type Rules using ***Inductive Definitions***

Q: What's the difference between a mathematician and a computer scientist?

A: The mathematician sometimes proves theorems without induction ☺