Typed λ with Booleans

$$
\begin{array}{rcl}
\langle t \rangle & ::= & \langle x \rangle \\
& | & \lambda \langle x \rangle . \langle t \rangle \\
& | & \langle t \rangle \ \langle t \rangle \\
& | & \texttt{true} \\
& | & \texttt{false} \\
& | & \texttt{if} \ \langle t \rangle \ \texttt{then} \ \langle t \rangle \ \texttt{else} \ \langle t \rangle
\end{array}
$$

Where x is a variable in the λ-Calculus sense.

- Exclude numbers to keep things simple for now.

$$
\begin{array}{rcl}
\langle T \rangle & ::= & \langle T \rangle \Rightarrow \langle T \rangle \\
& | & \mathsf{Bool}
\end{array}
$$

Expanding the definition

This grammar allows us to construct some really interesting types!

- $Bool \Rightarrow Bool$
  - A function mapping a Boolean argument to a Boolean result.
- $Bool \Rightarrow Bool \Rightarrow Bool$
  - A function mapping a Boolean argument to a function mapping a Boolean argument to a Boolean result.
  - $\Rightarrow$ is **right associative**, so the above is $Bool \Rightarrow (Bool \Rightarrow Bool)$
- $(Bool \Rightarrow Bool) \Rightarrow (Bool \Rightarrow Bool)$
  - An *operator* on Boolean functions.
- *Plus an infinite number of similar variations!*

How do we type *inputs*? In general there are two approaches:

- **Explicit Typing** *(Used in this course)*.
  - Typing annotations in the syntax functions:

$$\lambda x : T.t$$

- **Implicit Typing** *(Advanced topic in type theory)*.
  - aka via *inference*

Bad Inference Rule

$$\frac{t_2 : T_2}{(\lambda x : T_1 . t_2) : T_1 \Rightarrow T_2}$$

But consider:

$$\lambda x : Bool.\ \texttt{if}\ x\ \texttt{then}\ s_2\ \texttt{else}\ s_3$$

Enter the Context

$$x : Bool \vdash t_2 : T_2$$

- Typing relation becomes a **three-place relation**, i.e.

$$\text{context} \vdash \text{term} : \text{type}$$

In general, need things that look like

$$\{w : T_1, x : T_2, y : T_3\} \vdash z : T_4 \tag{1}$$

where `z` can mention `w`, `x`, `y`.
General form

$$\Gamma \vdash t : T \tag{2}$$

where Γ is a set of variable type relations.

Called either the **typing context** or the **typing environment**.

Well-formed contexts and variables

Formally we have a well-formed context relation:

$$\cdot \quad \texttt{ctx} \tag{C-Empty}$$

$$\frac{\Gamma \quad \texttt{ctx}}{\Gamma, x : T \quad \texttt{ctx}} \tag{C-Extend}$$

Well-formedness is implicitly assumed. Rather than using $\cdot$ for the empty context, we instead leave it blank:

$$\vdash t_1 : T_1$$

This typing rule is now possible:

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \tag{T-Var}$$

Q: what happens if we try to "insert" the same $\texttt{x}$ twice?

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

Let's try (see board):

$$\vdash \lambda x : Bool . \lambda y : Bool . \lambda z : Bool . y$$

$$\frac{\Gamma \vdash t_1 : T_1 \Rightarrow T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1 \; t_2 : T_2} \quad \text{(T-App)}$$

→ *(typed)* *Based on* λ *(5-3)*

Syntax

t	::=		*terms:*
		x	*variable*
		$\lambda x{:}T.t$	*abstraction*
		$t\ t$	*application*
v	::=		*values:*
		$\lambda x{:}T.t$	*abstraction value*
T	::=		*types:*
		$T{\to}T$	*type of functions*
Γ	::=		*contexts:*
		$\varnothing$	*empty context*
		$\Gamma, x{:}T$	*term variable binding*

Evaluation $\boxed{t \longrightarrow t'}$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

Typing $\boxed{\Gamma \vdash t : T}$

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

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

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

Remark: as is, *degenerate*.

Inversion Lemma

LEMMA [Inversion of the Typing Relation]

$$\Gamma \vdash x : R \implies x : R \in \Gamma \qquad \text{(I-Var)}$$

$$\begin{aligned} &\Gamma \vdash (\lambda x : T_1 . t_2) : R \\ &\implies \exists R_2 \mid R = (T_1 \Rightarrow R_2) \wedge \Gamma, x : T_1 \vdash t_2 : R_2 \end{aligned} \qquad \text{(I-Abs)}$$

$$\Gamma \vdash t_1\ t_2 : R \implies \exists T_{11} \mid \Gamma \vdash t_1 : T_{11} \Rightarrow R \wedge \Gamma \vdash t_2 : T_{11} \qquad \text{(I-App)}$$

Uniqueness still holds

THEOREM [Uniqueness of Types] In a given typing context Γ, if all the free variables of a term t are in the domain of Γ, t has at most one type.
Proof Sketch: By induction on term grammar. Crucially relies that each typing rule applies to a single term formation rule.

In this case, we say that the typing relation is *syntax directed*.

LEMMA [Canonical Forms]

1. If v is a value of type $Bool$, then v is either `true` or `false`.
2. If v is a value of type $T_1 \Rightarrow T_2$, then v has shape $\lambda x : T_1.t_2$.

Note that type $T_1 \Rightarrow T_2$ may have infinitely many values as inhabitants.

Progress

THEOREM [Progress for the Simply Typed λ-Calculus]
Suppose $\cdot \vdash t : T$. Either t is a value, or else there is some t' such that $t \to t'$.

A later theorem will let us generalize from the empty context. Terms typeable in the empty context are called **closed**.

Proof by Induction on Typing Derivations. Each evaluation rule is examined in term. Details use use inversion and, for the one tricky case of `T-AppAbs`, canonical forms are needed.

More about contexts

LEMMA [Permutation invariance]
If $\Gamma \vdash t : T$ and Δ is a permutation of Γ then $\Delta \vdash t : T$. Moreover, the latter derivation has the same depth as the former.

Proof Sketch: induction on typing derivations.

We add extra "facts" without changing conclusions: **LEMMA [Weakening]**
If $\Gamma \vdash t : T$ and $x \notin dom(\Gamma)$, then $\Gamma, x : S \vdash t : T$. Moreover, the latter derivation has the same depth as the former.

Proof Sketch: Induction on typing derivations.

Points of variations show up here, i.e. linear types, union types, dependent types, etc.

LEMMA [Preservation of Types Under Substitution]

$$\Gamma, x : S \vdash t : T \land \Gamma \vdash s : S \implies \Gamma \vdash [x \mapsto s]t : T \tag{3}$$

- Proof will proceed by induction over typing derivations, and using a case analysis over typing rules.

As a reminder:

$$
\begin{array}{lcll}
[x \mapsto s]x & = & s & \\
[x \mapsto s]y & = & y & \text{if } y \neq x \\
[x \mapsto s](\lambda y.t_1) & = & \lambda y.\ [x \mapsto s]t_1 & \text{if } y \neq x \text{ and } y \notin FV(s) \\
[x \mapsto s](t_1\ t_2) & = & [x \mapsto s]t_1\ [x \mapsto s]t_2 &
\end{array}
$$

Substitution Lemma II

T-True, T-False, T-If, T-App straightforward.

T-Var: $t = z \wedge z : T \in (\Gamma, x : S)$

- Case $x = z$
  - $[x \mapsto s]z$ would then evaluate to s.
  - $x = z \wedge z = t \implies x = t$
  - Via the uniqueness of types, $x : S \wedge t : T \implies S = T$
  - Substituting into lemma statement:

$$\Gamma, x : S \vdash x : S \wedge \Gamma \vdash s : S \implies \Gamma \vdash s : S$$

- Now consider $x \neq z$
  - $[x \mapsto s]z$ would then evaluate to z (and from there to t).

$$\Gamma, x : S \vdash t : T \wedge \Gamma \vdash s : S \implies \Gamma \vdash t : T$$

- We can now conclude by weakening.

Substitution Lemma III

T-Abs: $t = \lambda y : T_3.t_1 \;\wedge\; T = T_3 \Rightarrow T_4 \;\wedge\; \Gamma, x : S, y : T_3 \vdash t_1 : T_4$
By our meta-rule of substitutions in λ expressions, we derive:

$$x \neq y \qquad\qquad y \notin FV(s)$$

Using the the permutation lemma on the rightmost equation:

$$\Gamma, y : T_3, x : S \vdash t_1 : T_4$$

Using the weakening lemma on $\Gamma \vdash s : S$:

$$\Gamma, y : T_3 \vdash s : S$$

By the induction hypothesis:

$$\Gamma, y : T_3 \vdash [x \mapsto s]t_1 : T_4.$$

Recall T-Abs:

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1 . t_2 : T_1 \Rightarrow T_2}$$

Applying this to last equation $\Gamma, y : T_3 \vdash [x \mapsto s]t_1 : T_4$, get

$$\Gamma \vdash \lambda y : T_3 . [x \mapsto s]t_1 : T_3 \Rightarrow T_4$$

The definition of substitution is:

$$[x \mapsto s](\lambda y : T_3 . t_1) = \lambda y : T_3 . [x \mapsto s]t_1$$

- The LHS has type $T_3 \Rightarrow T_4$ from our original case analysis.
- The RHS has the same type from above.