# LiU-FP2016: Lecture 15

## *The Polymorphic Lambda Calculus (System F)*

Henrik Nilsson

University of Nottingham, UK

# This Lecture

- The simply typed lambda calculus.
- Limitations of the simply typed $\lambda$-calculus.
- The polymorphic lambda calculus (System F)
- Examples illustrating the power of system F

# The Simply Typed $\lambda$-Calculus (1)

$$
\begin{array}{lcll}
T & \rightarrow & & \textit{types:} \\
& | & B & \textit{fixed set of base types} \\
& | & T \rightarrow T & \textit{type of functions}
\end{array}
$$

$$
\begin{array}{lcll}
\Gamma & \rightarrow & & \textit{contexts:} \\
& | & \emptyset & \textit{empty context} \\
& | & \Gamma, x : T & \textit{context extension}
\end{array}
$$

Note: Need at least ***one*** base type, or there is no way to construct a type of finite size.

# The Simply Typed $\lambda$-Calculus (2)

| | | | |
|---|---|---|---|
| $t$ | $\rightarrow$ | | *terms:* |
| | $\mid$ | $x$ | *variable* |
| | $\mid$ | $c$ | *constant (optional)* |
| | $\mid$ | $\lambda x : T \,.\, t$ | *abstraction* |
| | $\mid$ | $t\ t$ | *application* |
| $v$ | $\rightarrow$ | | *values:* |
| | $\mid$ | $c$ | *constant (optional)* |
| | $\mid$ | $\lambda x : T \,.\, t$ | *abstraction* |

# The Simply Typed $\lambda$-Calculus (3)

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

$$\frac{c \text{ is a constant of type } T}{\Gamma \vdash c : T} \qquad \text{(T-CONST-c)}$$

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

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

# Example: TWICE (1)

Consider defining a function $\mathrm{twice}$:

$$\mathrm{twice}(f, x) \;=\; f(f(x))$$

# Example: TWICE (1)

Consider defining a function $\mathrm{twice}$:

$$\mathrm{twice}(f, x) \;=\; f(f(x))$$

Easy in the untyped $\lambda$-calculus:

$$\mathtt{TWICE} \;\equiv\; \lambda\mathtt{f}.\lambda\mathtt{x}.\mathtt{f}\,(\mathtt{f}\;\mathtt{x})$$

# Example: TWICE (1)

Consider defining a function $\mathrm{twice}$:

$$\mathrm{twice}(f, x) \quad = \quad f(f(x))$$

Easy in the untyped $\lambda$-calculus:

$$\texttt{TWICE} \quad \equiv \quad \lambda\texttt{f}\,.\,\lambda\texttt{x}\,.\,\texttt{f}\ (\texttt{f}\ \texttt{x})$$

What about the ***simply typed*** $\lambda$-calculus?

$$\texttt{TWICE} \quad \equiv \quad \lambda\texttt{f}:???\,.\,\lambda\texttt{x}:???\,.\,\texttt{f}\ (\texttt{f}\ \texttt{x})$$

# Example: TWICE (1)

Consider defining a function twice:

$$\mathrm{twice}(f, x) \;=\; f(f(x))$$

Easy in the untyped $\lambda$-calculus:

$$\mathtt{TWICE} \;\equiv\; \lambda\mathtt{f}.\lambda\mathtt{x}.\mathtt{f}\;(\mathtt{f}\;\mathtt{x})$$

What about the ***simply typed*** $\lambda$-calculus?

$$\mathtt{TWICE} \;\equiv\; \lambda\mathtt{f}{:}???.\lambda\mathtt{x}{:}???.\mathtt{f}\;(\mathtt{f}\;\mathtt{x})$$

What should the types of the arguments be?

# Example: TWICE (1)

Consider defining a function twice:

$$\mathrm{twice}(f, x) \;=\; f(f(x))$$

Easy in the untyped $\lambda$-calculus:

$$\mathtt{TWICE} \;\equiv\; \lambda\mathtt{f}.\lambda\mathtt{x}.\mathtt{f}\;(\mathtt{f}\;\mathtt{x})$$

What about the ***simply typed*** $\lambda$-calculus?

$$\mathtt{TWICE} \;\equiv\; \lambda\mathtt{f}:???.\lambda\mathtt{x}:???.\mathtt{f}\;(\mathtt{f}\;\mathtt{x})$$

What should the types of the arguments be?
Can **`TWICE`** be used for, say, both **`Bool`** and **`Nat`**?

# Example: TWICE (2)

Suppose `Bool`, `Nat` $\in B$.

# Example: TWICE (2)

Suppose **`Bool`**, **`Nat`** $\in B$.

What matters is that the types would be different even if we were to encode them in the base calculus.

# Example: TWICE (2)

Suppose `Bool`, `Nat` $\in B$.

What matters is that the types would be different even if we were to encode them in the base calculus.

Thus we need a ***separate*** definition for ***each*** type at which we want to use `TWICE`:

# Example: TWICE (2)

Suppose $\texttt{Bool}, \texttt{Nat} \in B$.

What matters is that the types would be different even if we were to encode them in the base calculus.

Thus we need a ***separate*** definition for ***each*** type at which we want to use **`TWICE`**:

$$\texttt{TWICEBOOL} \equiv \lambda\texttt{f:Bool}\to\texttt{Bool.}\lambda\texttt{x:Bool.f}\,(\texttt{f x})$$

# Example: TWICE (2)

Suppose $\texttt{Bool}, \texttt{Nat} \in B$.

What matters is that the types would be different even if we were to encode them in the base calculus.

Thus we need a ***separate*** definition for ***each*** type at which we want to use **`TWICE`**:

$$\begin{aligned}
\texttt{TWICEBOOL} &\equiv \lambda\texttt{f}:\texttt{Bool}\rightarrow\texttt{Bool}.\lambda\texttt{x}:\texttt{Bool}.\texttt{f}\,(\texttt{f}\;\texttt{x}) \\
\texttt{TWICENAT} &\equiv \lambda\texttt{f}:\texttt{Nat}\rightarrow\texttt{Nat}.\lambda\texttt{x}:\texttt{Nat}.\texttt{f}\,(\texttt{f}\;\texttt{x})
\end{aligned}$$

# Example: TWICE (2)

Suppose $\texttt{Bool}, \texttt{Nat} \in B$.

What matters is that the types would be different even if we were to encode them in the base calculus.

Thus we need a ***separate*** definition for ***each*** type at which we want to use **TWICE**:

$$
\begin{aligned}
\texttt{TWICEBOOL} &\equiv \lambda\texttt{f}:\texttt{Bool}\to\texttt{Bool}.\lambda\texttt{x}:\texttt{Bool}.\texttt{f}\ (\texttt{f}\ \texttt{x}) \\
\texttt{TWICENAT} &\equiv \lambda\texttt{f}:\texttt{Nat}\to\texttt{Nat}.\lambda\texttt{x}:\texttt{Nat}.\texttt{f}\ (\texttt{f}\ \texttt{x}) \\
\texttt{TWICENATFUN} &\equiv \lambda\texttt{f}:(\texttt{Nat}\to\texttt{Nat})\to(\texttt{Nat}\to\texttt{Nat}). \\
&\qquad \lambda\texttt{x}:\texttt{Nat}\to\texttt{Nat}.\texttt{f}\ (\texttt{f}\ \texttt{x})
\end{aligned}
$$

# Example: TWICE (3)

We have been forced to define ***essentially the same*** function over and over.

# Example: TWICE (3)

We have been forced to define ***essentially the same*** function over and over.

Common CS sensibility suggests ***abstraction*** over the ***varying*** part; i.e., here ***the type***!

# Example: TWICE (3)

We have been forced to define ***essentially the same*** function over and over.

Common CS sensibility suggests ***abstraction*** over the ***varying*** part; i.e., here ***the type***!

Thus, we would like to do something like:

$$\mathbf{TWICEPOLY} \equiv \Lambda \mathbf{T}.\lambda \mathbf{f}:\mathbf{T}\to\mathbf{T}.\lambda \mathbf{x}:\mathbf{T}.\mathbf{f}\,(\mathbf{f}\;\mathbf{x})$$

# Example: TWICE (3)

We have been forced to define ***essentially the same*** function over and over.

Common CS sensibility suggests ***abstraction*** over the ***varying*** part; i.e., here ***the type***!

Thus, we would like to do something like:

$$\texttt{TWICEPOLY} \;\equiv\; \Lambda\texttt{T}.\,\lambda\texttt{f:T}\!\rightarrow\!\texttt{T}.\,\lambda\texttt{x:T}.\,\texttt{f}\;(\texttt{f}\;\texttt{x})$$

Now:

$$\begin{aligned}
\texttt{TWICEBOOL} &\equiv \texttt{TWICEPOLY}\,[\texttt{Bool}] \\
\texttt{TWICENAT} &\equiv \texttt{TWICEPOLY}\,[\texttt{Nat}] \\
\texttt{TWICENATFUN} &\equiv \texttt{TWICEPOLY}\,[\texttt{Nat}\!\rightarrow\!\texttt{Nat}]
\end{aligned}$$

# System F: Abstract Syntax (1)

| | | |
|---|---|---|
| $T \;\rightarrow$ | | *types:* |
| $\mid$ | $B \mid T{\rightarrow}T$ | *[as for simply typed]* |
| $\mid$ | $X$ | *type variable* |
| $\mid$ | $\forall X \,.\, T$ | *universally quantified type* |
| $\Gamma \;\rightarrow$ | | *contexts:* |
| $\mid$ | $\emptyset \mid \Gamma, x : T$ | *[as for simply typed]* |
| $\mid$ | $\Gamma, X$ | *extension with type variable* |

# System F: Abstract Syntax (2)

$$
\begin{array}{llll}
t & \rightarrow & & \textit{terms:} \\
 & | & x \mid c \mid \lambda x\!:\!T\,.\,t \mid t\ t & \textit{[as for simply typed]} \\
 & | & \Lambda X\,.\,t & \textit{type abstraction} \\
 & | & t\ [T] & \textit{type application}
\end{array}
$$

$$
\begin{array}{llll}
v & \rightarrow & & \textit{values:} \\
 & | & c \mid \lambda x\!:\!T\,.\,t & \textit{[as for simply typed]} \\
 & | & \Lambda X\,.\,t & \textit{type abstraction value}
\end{array}
$$

# System F: Typing Rules

T-VAR, (T-CONST-c), T-ABS, T-APP are as before (omitted):

# System F: Typing Rules

T-VAR, (T-CONST-c), T-ABS, T-APP are as before (omitted):

Additional typing rules:

# System F: Typing Rules

T-VAR, (T-CONST-c), T-ABS, T-APP are as before (omitted):

Additional typing rules:

$$\frac{\Gamma, X \vdash t : T}{\Gamma \vdash \Lambda X \,.\, t : \forall X \,.\, T} \qquad \text{(T-TABS)}$$

# System F: Typing Rules

T-VAR, (T-CONST-c), T-ABS, T-APP are as before (omitted):

Additional typing rules:

$$\frac{\Gamma, X \vdash t : T}{\Gamma \vdash \Lambda X\,.\,t : \forall X\,.\,T} \quad \text{(T-TABS)}$$

$$\frac{\Gamma \vdash t_1 : \forall X\,.\,T_{12}}{\Gamma \vdash t_1\ [T_2] : [X \mapsto T_2]\ T_{12}} \quad \text{(T-TAPP)}$$

# System F: Evaluation Rules

E-APP1, E-APP2, E-APPABS are as before:

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

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \qquad \text{(E-APP2)}$$

$$(\lambda x\!:\!T_{11}\,.\,t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \qquad \text{(E-APPABS)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ [T_2] \longrightarrow t_1'\ [T_2]} \qquad \text{(E-TAPP)}$$

$$(\Lambda X\,.\,t_{12})\ [T_2] \longrightarrow [X \mapsto T_2]\ t_{12} \qquad \text{(E-TAPPABS)}$$

# Exercise

Given

$$
\begin{aligned}
\mathtt{ID} &\equiv \Lambda \mathtt{T}.\lambda \mathtt{x}:\mathtt{T}.\mathtt{x} \\
\Gamma_1 &= \emptyset, \mathtt{Nat}, \mathtt{5} : \mathtt{Nat}
\end{aligned}
$$

type check $\mathtt{ID}\ [\mathtt{Nat}]\ \mathtt{5}$ in context $\Gamma_1$.

(On whiteboard)

# System F: Church Booleans (1)

Recall untyped encoding:

$$\begin{array}{rcl} \texttt{TRUE} & \equiv & \lambda\texttt{t}.\lambda\texttt{f}.\texttt{t} \\ \texttt{FALSE} & \equiv & \lambda\texttt{t}.\lambda\texttt{f}.\texttt{f} \end{array}$$

We need to:

- assign a ***common*** type to these two terms;
- need to work for ***arbitrary*** argument types.

Any ideas?

$$\texttt{CBOOL} \equiv ???$$

# System F: Church Booleans (1)

Recall untyped encoding:

$$\begin{aligned}\texttt{TRUE} &\equiv \lambda\texttt{t}.\lambda\texttt{f}.\texttt{t}\\ \texttt{FALSE} &\equiv \lambda\texttt{t}.\lambda\texttt{f}.\texttt{f}\end{aligned}$$

We need to:

- assign a ***common*** type to these two terms;
- need to work for ***arbitrary*** argument types.

Parametrise on the type:

$$\texttt{CBOOL} \equiv \forall\texttt{X}.\texttt{X}\to\texttt{X}\to\texttt{X}$$

# System F: Church Booleans (2)

$$\mathtt{CBOOL} \equiv \forall \mathtt{X}.\mathtt{X}\to\mathtt{X}\to\mathtt{X}$$

$$\mathtt{TRUE} : \mathtt{CBOOL}$$
$$\mathtt{TRUE} \equiv \Lambda\mathtt{X}.\lambda\mathtt{t}:\mathtt{X}.\lambda\mathtt{f}:\mathtt{X}.\mathtt{t}$$

$$\mathtt{FALSE} : \mathtt{CBOOL}$$
$$\mathtt{FALSE} \equiv \Lambda\mathtt{X}.\lambda\mathtt{t}:\mathtt{X}.\lambda\mathtt{f}:\mathtt{X}.\mathtt{f}$$

$$\mathtt{NOT} : \mathtt{CBOOL}\to\mathtt{CBOOL}$$
$$\mathtt{NOT} \equiv \lambda\mathtt{b}:\mathtt{CBOOL}.\Lambda\mathtt{X}.\lambda\mathtt{t}:\mathtt{X}.\lambda\mathtt{f}:\mathtt{X}.\mathtt{b}\,[\mathtt{X}]\;\mathtt{f}\;\mathtt{t}$$

# Normalization

System F is strongly normalizing, like the simply typed $\lambda$-calculus.

# Homework

- Given `1` : `Nat` and `2` : `Nat`, write down a type-correct application of `TRUE` to `1` and `2` such that the result is `1`.
- Evaluate the above term using the evaluation rules.
- Prove `TRUE` : `CBOOL`.
- Prove `NOT` : `CBOOL`→`CBOOL`
- Provide a suitable definition of logical conjunction, `AND`.