G54FOP: Lecture 14 *The Polymorphic Lambda Calculus (System F)*

Henrik Nilsson

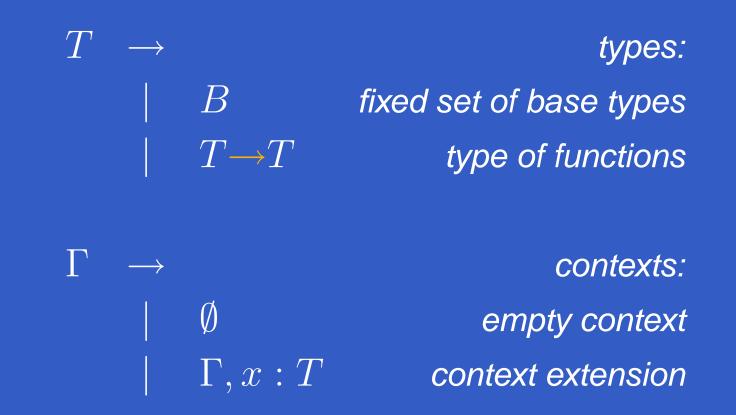
University of Nottingham, UK

G54FOP: Lecture 14 - p.1/17

This Lecture

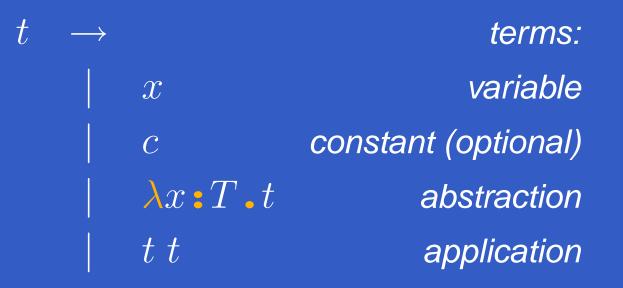
- Limitations of the simply typed λ -calculus.
- The polymorphic lambda calculus (System F)
- Examples illustrating the power of system F

Rcp: The Simply Typed λ -Calculus (1)



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

Rcp: The Simply Typed λ -Calculus (2)



 $v \rightarrow values:$ $\mid c \qquad constant (optional)$ $\mid \lambda x \cdot T \cdot t \qquad abstraction$

Rcp: The Simply Typed λ -Calculus (3)

$$\begin{array}{ll} \frac{x:T\in\Gamma}{\Gamma\vdash x:T} & (\text{T-VAR})\\ \frac{c \text{ is a constant of type }T}{\Gamma\vdash c:T} & (\text{T-CONST-c})\\ \frac{\Gamma,x:T_1\vdash t_2:T_2}{\Gamma\vdash\lambda x \bullet T_1 \bullet t_2:T_1 \to T_2} & (\text{T-ABS})\\ \frac{\Gamma\vdash t_1:T_{11}\to T_{12}}{\Gamma\vdash t_1:t_2:T_{12}} & \Gamma\vdash t_2:T_{11} & (\text{T-APP}) \end{array}$$

G54FOP: Lecture 14 – p.5/17

Consider defining a function twice:

twice(f, x) = f(f(x))

Consider defining a function twice:

twice(f, x) = f(f(x))

Easy in the untyped λ -calculus:

TWICE $\equiv \lambda \mathbf{f} \cdot \lambda \mathbf{x} \cdot \mathbf{f} (\mathbf{f} \cdot \mathbf{x})$

Consider defining a function twice:

twice(f, x) = f(f(x))

Easy in the untyped λ -calculus:

TWICE $\equiv \lambda f. \lambda x. f(fx)$

What about the simply typed λ -calculus?

TWICE $\equiv \lambda f:??.\lambda x:??.f(fx)$

Consider defining a function twice:

twice(f, x) = f(f(x))

Easy in the untyped λ -calculus:

TWICE $\equiv \lambda f.\lambda x.f(fx)$

What about the simply typed λ -calculus?

TWICE $\equiv \lambda f:??.\lambda x:??.f(fx)$

What should the types of the arguments be?

Consider defining a function twice:

twice(f, x) = f(f(x))

Easy in the untyped λ -calculus:

TWICE $\equiv \lambda f. \lambda x. f(fx)$

What about the simply typed λ -calculus?

TWICE $\equiv \lambda f:??.\lambda x:??.f(fx)$

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

G54FOP: Lecture 14 - p.6/17

Suppose Bool, Nat $\in B$.

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.

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**:

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**:

TWICEBOOL $\equiv \lambda f:Bool \rightarrow Bool.\lambda x:Bool.f(fx)$

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**:

TWICEBOOL $\equiv \lambda f:Bool \rightarrow Bool.\lambda x:Bool.f(f x)$

TWICENAT $\equiv \lambda f: Nat \rightarrow Nat. \lambda x: Nat. f(fx)$

G54FOP: Lecture 14 - p.7/17

Suppose **Bool**, $Mat \in B$.

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**:

TWICEBOOL $\equiv \lambda f:Bool \rightarrow Bool.\lambda x:Bool.f(fx)$

TWICENAT $\equiv \lambda f: Nat \rightarrow Nat. \lambda x: Nat. f(fx)$

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

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*!

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:

TWICEPOLY = $\Lambda \mathbf{T} \cdot \lambda \mathbf{f} : \mathbf{T} \rightarrow \mathbf{T} \cdot \lambda \mathbf{x} : \mathbf{T} \cdot \mathbf{f} (\mathbf{f} \mathbf{x})$

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:

TWICEPOLY $\equiv \Lambda \mathbf{T} \cdot \lambda \mathbf{f} : \mathbf{T} \rightarrow \mathbf{T} \cdot \lambda \mathbf{x} : \mathbf{T} \cdot \mathbf{f} (\mathbf{f} \mathbf{x})$ Now:

TWICEBOOL \equiv TWICEPOLY [Bool]TWICENAT \equiv TWICEPOLY [Nat]TWICENATFUN \equiv TWICEPOLY [Nat \rightarrow Nat]

System F: Abstract Syntax (1)

$$\begin{array}{cccc} T & \to & \\ & | & B & | & T \to T \\ & | & \forall X \bullet T \end{array}$$

types: [as for simply typed] universally quantified type

 $\begin{array}{cccc} \Gamma & \to & & \text{contexts:} \\ & \mid & \emptyset \mid \ \Gamma, x:T & & \text{[as for simply typed]} \\ & \mid & \Gamma, X & & \text{extension with type variable} \end{array}$

System F: Abstract Syntax (2)

values: [as for simply typed] type abstraction value

G54FOP: Lecture 14 - p.10/17

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

G54FOP: Lecture 14 – p.11/17

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

Additional typing rules:

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

Additional typing rules:

$$\frac{\Gamma, X \vdash t : T}{\Gamma \vdash \Lambda X \cdot t : \forall X \cdot T}$$
(T-TABS

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

Additional typing rules:

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

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

G54FOP: Lecture 14 – p.11/17

System F: Evaluation Rules

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

- $\frac{t_1 \longrightarrow t'_1}{t_1 t_2 \longrightarrow t'_1 t_2} \quad (E-APP1)$ $\frac{t_2 \longrightarrow t'_2}{v_1 t_2 \longrightarrow v_1 t'_2} \quad (E-APP2)$
- $\begin{array}{c} (\lambda x : T_{11} \cdot t_{12}) \ v_2 \longrightarrow [x \mapsto v_2] t_{12} & (\text{E-APPABS}) \\ \\ \frac{t_1 \longrightarrow t'_1}{t_1 \ [T_2] \longrightarrow t'_1 \ [T_2]} & (\text{E-TAPP}) \end{array} \end{array}$

 $(\Lambda X \bullet t_{12}) [T_2] \longrightarrow [X \mapsto T_2] t_{12}$ (E-TAPPABS)

Exercise

Given

type check ID [Nat] 5 in context Γ_1 . (On whiteboard)

System F: Church Booleans (1)

Recall untyped encoding:

TRUE	\equiv	$\lambda t. \lambda f.t$
FALSE	\equiv	$\lambda t.\lambda f.f$

We need to:

assign a common type to these two terms;

need to work for *arbitrary* argument types.
 Any ideas?

CBOOL \equiv ???

System F: Church Booleans (1)

Recall untyped encoding:

TRUE $\equiv \lambda t.\lambda f.t$ **FALSE** $\equiv \lambda t.\lambda f.f$

We need to:

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

Parametrise on the type:

CBOOL $\equiv \forall x.x \rightarrow x \rightarrow x$

System F: Church Booleans (2)

- **CBOOL** $\equiv \forall x.x \rightarrow x \rightarrow x$
 - TRUE : CBOOL
 - **TRUE** $\equiv \Lambda \mathbf{x} \cdot \lambda \mathbf{t} \cdot \mathbf{x} \cdot \lambda \mathbf{f} \cdot \mathbf{x} \cdot \mathbf{t}$
- FALSE : CBOOL
- **FALSE** $\equiv \Lambda \mathbf{x} \cdot \lambda \mathbf{t} \cdot \mathbf{x} \cdot \lambda \mathbf{f} \cdot \mathbf{x} \cdot \mathbf{f}$
 - NOT : CBOOL \rightarrow CBOOL NOT $\equiv \lambda b:$ CBOOL $\Lambda x \cdot \lambda t: x \cdot \lambda f: x \cdot b [x] f t$



System F is strongly normalizing, like the simply typed λ -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** \rightarrow **CBOOL**
- Provide a suitable definition of logical conjunction, AND.