Type Theory: Impredicative Part (1/5)

An Introduction to System F

Alexandre Miquel
Paris-sud University – Orsay, France
Alexandre.Miquel@lri.fr

Introduction

- **System F:** independently discovered by Girard (1970) and Reynolds (1974)

- Quite different motivations. . .

  Girard: Interpretation of second-order logic

  Reynolds: Functional programming

  . . . connected by the Curry-Howard isomorphism

- Significant influence on the development of Type Theory
  - Interpretation of higher-order logic [Girard, Martin-Löf]
  - Type:Type [Martin-Löf 1971]
  - Martin-Löf Type Theory [1972, 1984, 1990, ..]
  - The Calculus of Constructions [Coquand 1984]

System F: syntax

Types	A, B	$::=$	$\alpha \mid A \to B \mid \forall\alpha\,.\,B$
Terms	t, u	$::=$	$x \mid \underbrace{\lambda x : A\,.\,t \mid tu}_{\text{term abstr./app.}} \mid \underbrace{\Lambda\alpha\,.\,t \mid tA}_{\text{type abstr./app.}}$

$FV(t)$	$\equiv$	set of free (term) variables of the term t
$TV(t)$	$\equiv$	set of free type variables of the term t
$TV(A)$	$\equiv$	set of free type variables of the type A
$t\{x := u\}$	$\equiv$	substitute the term u to the variable x in the term t
$t\{\alpha := A\}$	$\equiv$	substitute the type A to the type variable α in the term t
$B\{\alpha := A\}$	$\equiv$	substitute the type A to the type variable α in the type B

Perform α-conversions to avoid capture of free (term/type) variables!

System F: Typing rules

Contexts $\Gamma, \Delta \quad ::= \quad x_1 : A_1, \ \ldots, \ x_n : A_n$

– Declarations are unordered
– Declaration of type variables is implicit (for each $\alpha \in TV(\Gamma)$)
– Could also declare type variables explicitely: $\alpha : *. \ldots$ (just a matter of taste)

Typing rules are syntax-directed:

$$\frac{}{\Gamma \vdash x : A} \ (x:A)\in\Gamma$$

$$\frac{\Gamma; x : A \vdash t : B}{\Gamma \vdash \lambda x : A \,.\, t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash \Lambda\alpha \,.\, t : \forall\alpha \,.\, B} \ \alpha\notin TV(\Gamma) \qquad \frac{\Gamma \vdash t : \forall\alpha \,.\, B}{\Gamma \vdash tA : B\{\alpha := A\}}$$

Example

The polymorphic identity: id $\equiv \Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,x \quad : \quad \forall\alpha\,.\,\alpha \to \alpha$

$$\dfrac{\dfrac{\dfrac{}{x : \alpha \vdash x : \alpha}}{\vdash \lambda x{:}\alpha\,.\,x : \alpha \to \alpha}}{\vdash \Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,x : \forall\alpha\,.\,\alpha \to \alpha}$$

One has:

id	:	$\forall\alpha\,.\,\alpha \to \alpha$	
id B	:	$B \to B$	for any type B
id $B\,u$	:	B	for any term $u : B$

In particular, when $B \equiv \forall\alpha\,.\,\alpha \to \alpha$ and $u \equiv$ id

id $(\forall\alpha\,.\,\alpha \to \alpha) \quad : \quad (\forall\alpha\,.\,\alpha \to \alpha) \to (\forall\alpha\,.\,\alpha \to \alpha)$

id $(\forall\alpha\,.\,\alpha \to \alpha)$ id $\quad : \quad \forall\alpha\,.\,\alpha \to \alpha$

Properties

- **Substitutivity**
  - *If* $\Gamma, x : A \vdash t : B$ *and* $\Gamma \vdash u : A$, *then* $\Gamma \vdash t\{x := u\} : B$
  - *If* $\Gamma \vdash t : B$, *then* $\Gamma\{\alpha := A\} \vdash t\{\alpha := A\} : B\{\alpha := A\}$
    (for any type variable α *and for any type* A*)*

- **Unicity of type**
  - *If* $\Gamma \vdash t : A$ *and* $\Gamma \vdash t : B$, *then* $A \equiv B$ *(syntactic identity)*

- **Decidability of typing:** Both problems
  1. *Given* Γ *and* t, *infer a type* A *such that* $\Gamma \vdash t : A$ *is derivable or raise* `Not_typable` *if there is no such type*
  2. *Given* Γ, t *and* A, *check whether the judgment* $\Gamma \vdash t : A$ *is derivable*

are decidable

System F: reduction

Two kind of redexes:

$$\begin{array}{lcll} (\lambda x : A\,.\,t)u & \succ & t\{x := u\} & \text{(first kind)} \\ (\Lambda\alpha\,.\,t)A & \succ & t\{\alpha := A\} & \text{(second kind)} \end{array}$$

Contextual closure:

$$\frac{t \;\succ\; t'}{tu \;\succ\; t'u} \qquad \frac{u \;\succ\; u'}{tu \;\succ\; tu'} \qquad \frac{t \;\succ\; t'}{\lambda x : A\,.\,t \;\succ\; \lambda x : A\,.\,t'}$$

$$\frac{t \;\succ\; t'}{tA \;\succ\; t'A} \qquad \frac{t \;\succ\; t'}{\Lambda\alpha\,.\,t \;\succ\; \Lambda\alpha\,.\,t'}$$

Reflexive & transitive closure:

$$\frac{}{t \;\succ^*\; t} \qquad \frac{t \;\succ^*\; t' \qquad t' \;\succ\; t''}{t \;\succ^*\; t''}$$

Examples

- The polymorphic identity (again):

$$\mathsf{id}\, B\, u \quad \equiv \quad (\Lambda\alpha\,.\,\lambda x : \alpha\,.\,x)\, B\, u \quad \succ \quad (\lambda x : B\,.\,x)\, u \quad \succ \quad u$$

$$\mathsf{id}\,(\forall\alpha\,.\,\alpha{\rightarrow}\alpha)\,\mathsf{id}\,(\forall\alpha\,.\,\alpha{\rightarrow}\alpha)\ \cdots\ \mathsf{id}\,(\forall\alpha\,.\,\alpha{\rightarrow}\alpha)\,\mathsf{id}\,B\,u \quad \succ^* \quad u$$

- A more complex example. . .

$$\begin{array}{l}\big(\Lambda\alpha\,.\,\lambda x : \alpha\,.\,\lambda f : \alpha{\rightarrow}\alpha\,.\,\overbrace{f\,(\cdots\,(f}^{\text{32 times}}\,x)\cdots)\big)\\ \quad(\forall\alpha\,.\,\alpha{\rightarrow}(\alpha{\rightarrow}\alpha){\rightarrow}\alpha)\,(\Lambda\alpha\,.\,\lambda x : \alpha\,.\,\lambda f : \alpha{\rightarrow}\alpha\,.\,f x)\\ \quad(\lambda n : \forall\alpha\,.\,\alpha{\rightarrow}(\alpha{\rightarrow}\alpha){\rightarrow}\alpha\,.\,\Lambda\alpha\,.\,\lambda x : \alpha\,.\,\lambda f : \alpha \rightarrow \alpha\,.\,n\alpha(n\alpha x f)f)\end{array}$$

$$\succ^* \quad \Lambda\alpha\,.\,\lambda x : \alpha\,.\,\lambda f : \alpha{\rightarrow}\alpha\,.\,\underbrace{(f\ \cdots\,(f}_{\text{4 294 967 296 times}}\,x)\cdots)$$

Properties

- **Church-Rosser**
  - *If* $t \succ^* t_1$ *and* $t \succ^* t_2$, *then there is a term* t' *such that* $t_1 \succ^* t'$ *and* $t_2 \succ^* t'$.

  Proof. Roughly the same as for the untyped λ-calculus (adaptation is easy)

- **Subject reduction**
  - *If* $\Gamma \vdash t : A$ *and* $t \succ^* t'$, *then* $\Gamma \vdash t' : A$

  Proof. By induction on the derivation of $\Gamma \vdash t : A$, with $t \succ t'$ (one step reduction)

- **Strong normalization**
  - All the well-typed term of system F are strongly normalizable

  Proof. Girard and Tait's method of reducibility candidates (postponed)

Data structures

- In ML/Haskell approach, the type system constituted by $\longrightarrow$ and $\forall$ (prenex) is not sufficent for programming

  $\Rightarrow$ Must extend the type system with other constructions

  Primitive datatypes: booleans, integers, etc.

  Type constructors: pairs, records, lists, etc.

- In system F, these primitive datatypes are actually definable

  $\oplus$ No extension of the type system is needed

  $\oplus$ Keeps the simplicity of the system

  $\ominus$ Much less flexible than ML/Haskell approach

  $\Rightarrow$ Illustrates the strength of the system

Booleans (1/3)

$$
\begin{array}{lcll}
\mathsf{Bool} & \equiv & \forall\gamma\,.\,\gamma \to \gamma \to \gamma & \\
\mathsf{true} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,x & : \quad \mathsf{Bool} \\
\mathsf{false} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,y & : \quad \mathsf{Bool}
\end{array}
$$

$$
\mathtt{if}_A\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \quad \equiv \quad u A t_1 t_2
$$

From these definitions, we easily derive the typing rule

$$
\frac{\Gamma \vdash u : \mathsf{Bool} \qquad \Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash \mathtt{if}_A\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 : A}
$$

as well as the reduction rules

$$
\frac{u \ \succ^* \ \mathsf{true}}{\mathtt{if}_A\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \ \succ^* \ t_1} \qquad \frac{u \ \succ^* \ \mathsf{false}}{\mathtt{if}_A\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \ \succ^* \ t_2}
$$

Booleans (2/3)

Objection: Could do the same in untyped λ-calculus!

$$\left.\begin{array}{l} \text{true} \equiv \lambda x, y \,.\, x \\ \text{false} \equiv \lambda x, y \,.\, y \\[1em] \texttt{if } t \texttt{ then } u_1 \texttt{ else } u_2 \equiv t u_1 u_2 \end{array}\right\} \text{Enjoys the same reduction rules}$$

So, why making life complicated with all these types ?

Remark: In untyped λ-calculus, nothing prevents the following computation:

$$\texttt{if } \underbrace{\lambda x \,.\, x}_{\text{bad bool}} \texttt{ then } t_1 \texttt{ else } t_2 \equiv (\lambda x \,.\, x) t_1 t_2 \succ \underbrace{t_1 t_2}_{\text{meaningless result}}$$

Question: Is the type system of system F sufficent to avoid this problem ?

Booleans (3/3)

Principle (that should be satisfied by any good functional programming language)

> *When a program P of type A evaluates to a value v, then v has one of the canonical forms expected by the type A.*

In ML/Haskell for instance, a value produced by a program of type `bool` will always be `true` or `false` (i.e. the canonical forms of type `bool`).

In system F: The subject-reduction property ensures that the normal form of a term of type Bool is a term of type Bool. To conclude, it suffices to check that:

> **Lemma.** – *The terms* $\Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,x$ (true) *and* $\Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,y$ (false) *are the only closed normal terms of type* $\forall\gamma\,.\,\gamma \to \gamma \to \gamma$ (Bool)

Proof. Case analysis on the derivation.

Cartesian product

Given two types A and B, we set:

$$\begin{array}{lcl} A \times B & \equiv & \forall\gamma\,.\,(A \to B \to \gamma) \to \gamma \\ \langle t_1, t_2 \rangle & \equiv & \Lambda\gamma\,.\,\lambda f : A \to B \to \gamma\,.\,f t_1 t_2 \\ \mathsf{fst} & \equiv & \lambda p : A \times B\,.\,pA(\lambda x : A\,.\,\lambda y : B\,.\,x) \quad : \quad A \times B \to A \\ \mathsf{snd} & \equiv & \lambda p : A \times B\,.\,pB(\lambda x : A\,.\,\lambda y : B\,.\,y) \quad : \quad A \times B \to B \end{array}$$

From these definitions, we easily derive the typing rule and the reduction rules:

$$\frac{\Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : B}{\Gamma \vdash \langle t_1, t_2 \rangle : A \times B} \qquad \mathsf{fst}\langle t_1, t_2 \rangle \ \succ^* \ t_1 \qquad \mathsf{snd}\langle t_1, t_2 \rangle \ \succ^* \ t_2$$

Again, we easily check that:

Lemma. – *The closed normal terms of type $A \times B$ are of the form $\langle t_1, t_2 \rangle$, where t_1 and t_2 are closed normal terms of type A and B, respectively.*

Disjoint union

Given two types A and B, we set:

$$A + B \quad \equiv \quad \forall\gamma\,.\,(A \to \gamma) \to (B \to \gamma) \to \gamma$$

$$\begin{array}{lcllll}
\mathsf{inl}(v) & \equiv & \Lambda\gamma\,.\,\lambda f : A \to \gamma\,.\,\lambda g : B \to \gamma\,.\,f v & : & A + B & (\text{with } v : A) \\
\mathsf{inr}(v) & \equiv & \Lambda\gamma\,.\,\lambda f : A \to \gamma\,.\,\lambda g : B \to \gamma\,.\,g v & : & A + B & (\text{with } v : B)
\end{array}$$

$$\mathtt{case}_C\ u\ \mathtt{of}\ \ \mathsf{inl}(x) \mapsto t_1 \ \mid\ \mathsf{inr}(y) \mapsto t_2 \quad \equiv \quad uC(\lambda x : A\,.\,t_1)(\lambda y : B\,.\,t_2)$$

From these definitions, we derive the expected typing rule and reduction rules

$$\frac{\Gamma \vdash u : A + B \qquad \Gamma,\ x : A \vdash t_1 : C \qquad \Gamma,\ y : B \vdash t_2 : C}{\Gamma \ \vdash\ \mathtt{case}_C\ u\ \mathtt{of}\ \ \mathsf{inl}(x) \mapsto t_1 \ \mid\ \mathsf{inr}(y) \mapsto t_2 \ :\ C}$$

$$\begin{array}{lcl}
\mathtt{case}_C\ \mathsf{inl}(v)\ \mathtt{of}\ \ \mathsf{inl}(x) \mapsto t_1 \ \mid\ \mathsf{inr}(y) \mapsto t_2 & \succ^* & t_1\{x := v\} \\
\mathtt{case}_C\ \mathsf{inr}(v)\ \mathtt{of}\ \ \mathsf{inl}(x) \mapsto t_1 \ \mid\ \mathsf{inr}(y) \mapsto t_2 & \succ^* & t_2\{y := v\}
\end{array}$$

[Same remark as before for the canonical forms of type $A + B$]

Finite types

For any integer $n \geq 0$ we set

$$\begin{array}{lcll} \mathsf{Fin}_n & \equiv & \forall\gamma\,.\,\underbrace{\gamma \to \cdots \to \gamma}_{n\ \mathsf{times}} \to \gamma & \\ \mathbf{e}_1 & \equiv & \Lambda\gamma\,.\,\lambda x_1 : \gamma \ldots \lambda x_n : \gamma\,.\,x_1 & : \ \mathsf{Fin}_n \\ & \vdots & & \\ \mathbf{e}_n & \equiv & \Lambda\gamma\,.\,\lambda x_1 : \gamma \ldots \lambda x_n : \gamma\,.\,x_n & : \ \mathsf{Fin}_n \end{array}$$

Again, $\mathbf{e}_1, \ldots, \mathbf{e}_n$ are the only closed normal terms of type Fin_n.

In particular:

$$\begin{array}{lclcll} \mathsf{Fin}_2 & \equiv & \forall\gamma\,.\,\gamma \to \gamma \to \gamma & \equiv & \mathsf{Bool} & \text{(type of booleans)} \\ \mathsf{Fin}_1 & \equiv & \forall\gamma\,.\,\gamma \to \gamma & \equiv & \mathsf{Unit} & \text{(unit data-type)} \\ \mathsf{Fin}_0 & \equiv & \forall\gamma\,.\,\gamma & \equiv & \bot & \text{(empty data-type)} \end{array}$$

(Notice that there is no closed normal term of type $\bot$.)

Natural numbers

In system F the type of Church numerals is defined by

$$\begin{array}{lcll}
\mathsf{Nat} & \equiv & \forall\gamma\,.\,\gamma \to (\gamma \to \gamma) \to \gamma & \\
\overline{0} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,x & \\
\overline{1} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,f x & \\
\overline{2} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,f(f x) & \\
 & \vdots & & \\
\overline{n} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,\underbrace{f(\cdots(f}_{n\ \text{times}}\,x)\cdots) & : \quad \mathsf{Nat} \\
 & \vdots & &
\end{array}$$

$\Rightarrow$ The terms $\overline{0}$, $\overline{1}$, $\overline{2}$, . . . are the only closed normal terms of type Nat.

The corresponding iterator (or recursor) is given by

$$\begin{array}{lcl}
\mathsf{iter} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,\lambda n:\mathsf{Nat}\,.\,n\gamma x f \\
 & & : \quad \forall\gamma\,.\,\gamma \to (\gamma{\to}\gamma) \to \mathsf{Nat} \to \gamma
\end{array}$$