

System F

Adrien FRABETTI MATHIEU

Contents

Definition	1
Encodings	1
Empty type	1
Unit type	1
Natural numbers	1
Product type	1
Sum type	1
Existential type	1
Meta-theoretic properties	1
Logical relations	1
Strong normalisation	1
Decidability problems	1
Type checking	1
Type inference	1
Mitigation	1

Definition

Let us define inductively types as follows

$$T \quad := \quad \alpha \mid T \rightarrow T \mid \forall \alpha. T$$

then terms

$$\begin{aligned} t \quad := \quad & \lambda x. t \mid tt \\ & \mid \Lambda \alpha. t \mid tT \\ & \mid x \end{aligned}$$

We define straightforwardly the typing judgement

$$\begin{array}{c} \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \text{Ax} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \rightarrow B} \text{I}_{\rightarrow} \qquad \frac{\Gamma \vdash u : A \rightarrow B \quad \Gamma \vdash v : A}{\Gamma \vdash uv : B} \text{E}_{\rightarrow} \\[10pt] \frac{\Gamma \vdash t : A \quad \alpha \notin \text{fv}(\Gamma)}{\Gamma \vdash \Lambda \alpha. t : \forall \alpha, A} \text{I}_{\forall} \qquad \frac{\Gamma \vdash t : \forall \alpha, A}{\Gamma \vdash tT : A[\alpha \mapsto T]} \text{E}_{\forall} \end{array}$$

Encodings

Impredicative encodings are extremely powerful, as they are enough to define a large class of inductive types with their plain recursor.

The trick is to encode a type as the type of (polymorphic) recursors. Thus, encodings are similar to Church encodings in plain λ -calculus.

It should be noted that, although these encodings are logically enough to be as expressive as inductive types, they are computationally bad encodings, in the sense that they are very wasteful. Natural numbers, for instance, end up being represented in a unary form that additionally has extra overhead than plain unary numbers.

Empty type

The empty type is encoded as

$$\mathbf{0} \quad := \quad \forall \alpha, \alpha$$

Indeed, if I have a term $t : \mathbf{0}$, and any type T , one can derive a witness $tT : T$ of T , which makes $\mathbf{0}$ validate the *ex falso quodlibet* principle.

Unit type

The unit type is encoded as

$$\mathbf{1} \quad := \quad \forall \alpha, \alpha \rightarrow \alpha$$

as well as having a constructor

$$\star \quad := \quad \Lambda \alpha. \lambda x. x : \mathbf{1}$$

We will prove that every closed value $v : \mathbf{1}$ has the canonicity property:

$$v \equiv \star$$

This will stem from a more general adequacy result.

Natural numbers

Natural numbers are encoded as

$$\mathbb{N} := \forall \alpha, (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$$

with

$$\begin{aligned} 0 &:= \Lambda \alpha. \lambda s. \lambda z. z \\ n + 1 &:= \Lambda \alpha. \lambda s. \lambda z. n(\alpha, sz) \end{aligned}$$

Product type

Let U, V be two types. We can define

$$U \times V := \forall \alpha, (U \rightarrow V \rightarrow \alpha) \rightarrow \alpha$$

and, for $u : U$ and $v : V$, we have

$$(u, v) := \Lambda \alpha. \lambda f. f(u, v)$$

Sum type

Let U, V be two types. We can define

$$U + V := \forall \alpha, (U \rightarrow \alpha) \rightarrow (V \rightarrow \alpha) \rightarrow \alpha$$

and

$$\begin{aligned} \iota_1 &:= \lambda u. \Lambda \alpha. \lambda f. \lambda g. fu \\ \iota_2 &:= \lambda v. \Lambda \alpha. \lambda f. \lambda g. gv \end{aligned}$$

Existential type

Let U be a type, we can define

$$\exists \alpha, U := \forall \beta, (\forall \alpha, U \rightarrow \beta) \rightarrow \beta$$

Meta-theoretic properties

Meta-properties of System F are mainly established through two ways.

Firstly, by reduction to λ -calculus: System F introduces type binders and application, but that's an extremely tame calculus because it

- doesn't interfere with itself: reducing a redex $(\Lambda \alpha. t)T$ cannot create an other similar redex
- doesn't interfere with the standard β reduction: reducing a redex as above cannot create a β redex

Hence one can essentially erase all the second order part from a System F term to retrieve important properties of the λ -calculus, such as confluence.

Secondly, by crafting adequate logical relations, as with STLC. Note, however, that historically this second part has been considered a hard problem, because System F's impredicativity breaks the naive way to make logical relations. Indeed, System F's strong normalisation (which follows directly from being able to define logical relations) *need* the meta theory to itself have impredicativity, and it cannot be proved, for instance, in MLTT.

Logical relations

For this section, let $n : \mathbb{N}$ be a natural number, which is a parameter of our logical relation construction. n will define the arity of the logical relation. Common values are $n = 1, 2$.

We will define, by induction on T , an n -ary relation $V\llbracket T \rrbracket_\rho^n$, where $\text{fv}(T) \subseteq \text{dom } \rho$, and ρ is a family of n -ary relations on values of System F, used to interpret free variables of T .

$$\begin{aligned}
V[\![\alpha]\!]_{\rho}^n &:= \rho(\alpha) \\
V[\![A \rightarrow B]\!]_{\rho}^n &:= \{(f_1, \dots, f_n) \in \text{Value}^n \mid \forall (u_1, \dots, u_n) \in V[\![A]\!]_{\rho}^n, (f_1 u_1, \dots, f_n u_n) \in E[\![B]\!]_{\rho}^n\} \\
V[\![\forall \alpha, T]\!]_{\rho}^n &:= \{(u_1, \dots, u_n) \in \text{Value}^n \mid \forall R \subseteq \text{Value}^n, (u_1, \dots, u_n) \in V[\![T]\!]_{\rho[\alpha \mapsto R]}^n\}
\end{aligned}$$

where

$$E[\![T]\!]_{\rho}^n := \left\{ (t_1, \dots, t_n) \in \text{Term}^n \mid \exists (v_1, \dots, v_n) \in V[\![T]\!]_{\rho}^n, \forall i \in \{1, \dots, n\}, t_i \xrightarrow[\beta]^* v_i \right\}$$

Note that, crucially, the interpretation of a universal type requires an impredicative universal quantification in the meta-theory.

Theorem 1 (Fundamental theorem of logical relations)

Let T be a type with free variables in Γ , and $u : \text{Term}$ such that $\vdash u : T$. For any environment ρ , it holds that

$$(u, \dots, u) \in E[\![T]\!]_{\rho}^n$$

Theorem (..) $\Rightarrow$..: Fundamental theorem of logical relations

By induction on T :

- $\vdash u : \alpha$.

Strong normalisation

Decidability problems

System F is a simple extension of STLC, that is expressive enough to be used as a programming language: for instance, the core type system of OCaml is a subset of System F.

Yet, System F is not used for programming languages in practice, because of a fundamental limitation:

Theorem 1

In System F, type checking and type inference are undecidable.

Type checking

The precise statement of the theorem is: for t a term, and T a type, it is undecidable to know whether

$$\vdash t : T$$

Type inference

For t a term, it is undecidable whether there exists a type T such that

$$\vdash t : T$$

Mitigation

Undecidability of System F can be mitigated using an intrinsic presentation of its calculus, rather than an extrinsic one.

However, even intrinsic System F is rarely used in practice, because programming languages that abandon decidable type checking (and thus, in the general case, need user annotations) usually go for even more expressive type theories, for instance with higher kinded types (see System $F\omega$), or even with full type dependency (see the Calculus of Constructions).