Polymorphism

Last revised: 2026-07-01

The motivation for (parametric) **polymorphism** is to *reuse* a single term for different types. For instance, in STLC we have distinct identity functions $\mathrm{id}_A \coloneqq \lambda x{:}A.\ x$ and $\mathrm{id}_B \coloneqq \lambda x{:}B.\ x$ for distinct types A and B. With polymorphism, we can construct a single term $\mathrm{id} \coloneqq \Lambda\alpha.\ \lambda x{:}\alpha.\ x : \forall\alpha.\alpha \to \alpha$ that works on all types.

Unfortunately, typing checking in full polymorphism is undecidable. Nonetheless, we can impose restrictions on the shape of polymorphic types and introduce *let expressions* to recover decidability. Under such systems, we can type terms such as $\textbf{let}\ f = \lambda x.\ x\ \textbf{in}\ (f\ 0, f\ \mathrm{true}) : \mathrm{Nat} \times \mathrm{Bool}$.

1 System F

In full polymorphism (a.k.a $\lambda 2$ endowed with *System F*), types are defined to be either a *type variable*, an *arrow type*, or *universally quantified type*:

$$A, B ::= \alpha \mid A \to B \mid \forall\alpha.A$$

The arrow type has the following introduction and elimination rules (*à la Church*)[4]:

$$\frac{\Gamma, \alpha : \star \vdash M : A}{\Gamma \vdash \lambda\alpha.\ M : \forall\alpha.A}\ \forall\mathrm{I}$$

$$\frac{\Gamma \vdash M : \forall\alpha.A \quad \Gamma \vdash B\ \mathrm{ty}}{\Gamma \vdash M\ B : A[B/\alpha]}\ \forall\mathrm{E}$$

Example 1A. In such polymorphism, we can build types that are semantically equilavent to the **bottom type** and the **top type**.

$$\bot \cong \forall\alpha.\alpha \quad \top \cong \forall\alpha.\alpha \to \alpha$$

To see why $\forall\alpha.\alpha$ correspond to the bottom type (i.e. empty), suppose $M : \forall\alpha.\alpha$. Then $M\ \bot$ has the bottom type, contracting that the bottom type has no terms. For $\top$, note that, by parametricity, $\Lambda\alpha.\ \lambda x.\ x$ is the only term of type $\forall\alpha.\alpha \to \alpha$.

[4] i.e. with explicit type annotations

1.1 Curry Style Implicit Arguments

In constrast to the Church style terms with explicit type annotations, System F *à la Curry* consists of polymorphic terms with *implicit type arguments*.

$$\frac{\Gamma, \alpha : \star \vdash M : A}{\Gamma \vdash M : \forall \alpha.A}\ \forall \mathrm{I}$$

$$\frac{\Gamma \vdash M : \forall \alpha.A \quad \Gamma \vdash B\ \mathrm{ty}}{\Gamma \vdash M : A[B/\alpha]}\ \forall \mathrm{E}$$

Definition 11A. The conversion from Church style to Curry style is called **type erasure**, defined as follows:

$$\begin{aligned} |x| &:= x & |\lambda x{:}A.\, M| &:= \lambda x.\, |M| \\ |\Lambda \alpha.\, M| &:= M & |M\ N| &:= |M|\ |N| \\ |M\ A| &:= M \end{aligned}$$

We have that type erasure perserves type checking.

Lemma 11A. If $\Gamma \vdash M : A$ à la Church, then $\Gamma \vdash |M| : A$ à la Curry. Conversely, if $\Gamma \vdash N : A$ à la Curry, then there exists a term M à la Church such that $\Gamma \vdash M : A$ and $|M| \equiv N$.

While Curry style polymorphism is convenient, its type checking problem is undecidable [Wel99]. To recover decidablility, we need a weaker form of polymorphism, found in the *Hindley–Milner type system*.

2 Hindley–Milner Type System

In the *Hindley–Milner (HM) type system*, the universal quantifiers only appear at outermost level. Thus types are defined as follows:

$$\begin{aligned} \tau, \sigma &::= \alpha \mid \tau \to \tau \\ A, B &::= \tau \mid \forall \alpha.A \end{aligned}$$

with the following rules:

$$\frac{\Gamma, \alpha : \star \vdash M : \tau}{\Gamma \vdash \lambda\alpha.M : \forall\alpha.\tau}$$

$$\frac{\Gamma \vdash M : \forall\alpha.\tau}{\Gamma \vdash M\,\sigma : \tau[\sigma/\alpha]}$$

Since the universal quantifier only appears outside, we usually omit them. As such, we can assume that *all type variables are implicitly quantified.*

Theorem 2A. Type checking in HM is decidable, and is given by Algorithm W.

Nonetheless, we can obtain additional flexibility in the HM type system with a syntactical technique called *let polymorphism.*

2.1 Let Polymorphism

Intuitively, *let polymorphism* allows a function to have a polymorphic type only if the function is immediately applied. We extend the calculus with a *let expression*, $\textbf{let}\ x = M\ \textbf{in}\ N$, that is semantically equivalent to the redex $(\lambda x.\ N)\ M$. Its typing rules is as follows:

$$\frac{\Gamma \vdash M : A \quad \Gamma, x : A \vdash N : \tau}{\Gamma \vdash \textbf{let}\ x = M\ \textbf{in}\ N : \tau}$$

This gives us access to the term $\lambda x{:}A.\ N$ without forming it and type it as $A \to \tau$. The type checking problem remains decidable.

Bibliography

[Wel99] J. Wells, "Typability and type checking in System F are equivalent and undecidable," *Annals of Pure and Applied Logic*, vol. 98, no. 1, pp. 111–156, 1999, doi: https://doi.org/10.1016/S0168-0072(98)00047-5.