The Curry Simple Types System

May 28, 2019

Simple Types

We start by defining the sets of term and types for this system.

Definition 1 *Given an infinite set of term variables V, the term language is give by the following grammar:*

$$M ::= x \mid MM' \mid \lambda x.M$$

Definition 2 *Let $\mathbb{V}$ be an infinite set of type variables. The* set of simple types, *$\mathbb{T}_C$ is inductively defined from $\mathbb{V}$ in the following way:*

$$\begin{array}{rcl} \alpha \in \mathbb{V} & \Rightarrow & \alpha \in \mathbb{T}_C \\ \tau, \tau' \in \mathbb{T}_C & \Rightarrow & (\tau \to \tau') \in \mathbb{T}_C \end{array}$$

Notation If $\tau_1, \ldots, \tau_n \in \mathbb{T}_C$, then

$$\tau_1 \to \tau_2 \to \cdots \to \tau_n$$

represents

$$(\tau_1 \to (\tau_2 \to \cdots \to (\tau_{n-1} \to \tau_n) \ldots))$$

that is, the type constructor $\to$ is right associative.

Definition 3 *If x is a term variable in $\mathcal{V}$ and τ is a type in $\mathbb{T}_C$ then:*

- *A* statement *is of the form $M : \tau$, where the type τ is called the* predicate, *and the term M is called the* subject *of the statement.*
- *A* declaration *is a statement where the subject is a term variable.*
- *A* basis Γ *is a set of declarations where all the subjects are distinct.*

A basis where the subjects are pairwise distinct, is called *monovalent* (or consistent).

Definition 4 *If $\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$ is a basis, then:*

- *Γ is a partial function, with* domain, *denoted $\mathsf{dom}(\Gamma) = \{x_1, \ldots, x_n\}$, and $\Gamma(x_i) = \tau_i$.*
- *Let $\mathcal{V}_0$ be a set of variables. Then $\Gamma \restriction \mathcal{V}_0 = \{x : \Gamma(x) \mid x \in \mathcal{V}_0\}$.*
- *We define Γ_x as $\Gamma \setminus \{x : \tau\}$.*

Definition 5 *In the Curry type system, we say that M has type τ given the basis Γ, and write*

$$\Gamma \vdash_{\mathcal{C}} M : \tau,$$

if $\Gamma \vdash_{\mathcal{C}} M : \tau$ can be obtained from the following derivation rules:

$$\Gamma \cup \{x : \tau\} \vdash_{\mathcal{C}} x : \tau \qquad (\mathsf{Axiom})$$

$$\frac{\Gamma_x \cup \{x : \tau_1\} \vdash_{\mathcal{C}} M : \tau_2}{\Gamma_x \vdash_{\mathcal{C}} \lambda x.M : \tau_1 \to \tau_2} \qquad (\to \mathsf{Intro})$$

$$\frac{\Gamma \vdash_{\mathcal{C}} M : \tau_1 \to \tau_2 \qquad \Gamma \vdash_{\mathcal{C}} N : \tau_1}{\Gamma \vdash_{\mathcal{C}} MN : \tau_2} \qquad (\to \mathsf{Elim})$$

Example 6 *For the λ-term* $(\lambda xy.x)(\lambda x.x)$ *the following derivation is obtained in the Curry Simple Type System:*

$$\dfrac{\dfrac{\dfrac{\{x:\alpha\to\alpha, y:\beta\}\vdash_{\mathcal{C}} x:\alpha\to\alpha}{\{x:\alpha\to\alpha\}\vdash_{\mathcal{C}} \lambda y.x:\beta\to\alpha\to\alpha}}{\vdash_{\mathcal{C}} \lambda xy.x:(\alpha\to\alpha)\to\beta\to\alpha\to\alpha} \qquad \dfrac{\{x:\alpha\}\vdash_{\mathcal{C}} x:\alpha}{\vdash_{\mathcal{C}} \lambda x.x:\alpha\to\alpha}}{\vdash_{\mathcal{C}} (\lambda xy.x)(\lambda x.x):\beta\to\alpha\to\alpha}$$

Definition 7 (Substitution) *We call* type-substitution *to*

$$\mathbb{S} = [\tau_1/\alpha_1, \ldots, \tau_n/\alpha_n]$$

where $\alpha_1, \ldots, \alpha_n$ *are distinct type variables and* $\tau_1, \ldots, \tau_n$ *are types in* $\mathbb{T}_C$*. If* τ *is a type in* $\mathbb{T}_C$*, then* $\mathbb{S}(\tau)$ *is the type obtained by simultaneously substituting* α_i *by* τ_i*, with* $1 \leq i \leq n$*, in* τ*.*

The type $\mathbb{S}(\tau)$ *is called an* instance *of the type* τ*. The notion of substitution can be extended to basis in the following way:*

$$\mathbb{S}(\Gamma) \quad = \quad \{x_1 : \mathbb{S}(\tau_1), \ldots, x_n : \mathbb{S}(\tau_n)\} \qquad \text{if } \Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$$

The basis $\mathbb{S}(\Gamma)$ *is called an* instance *of the basis* Γ*.*

Type-checking and Typability

In the Curry Type System, as well in other type systems the following questions arise:

1. Given a term M in Λ, a type τ and a basis Γ, do we have $\Gamma \vdash_{\mathcal{C}} M : \tau$?
2. Given a term M in Λ, is there a type τ and a basis Γ, such that $\Gamma \vdash_{\mathcal{C}} M : \tau$?

The first question concerns *type checking* and the second *typability*, and will be discuss in this section.

Type checking and typability in the Curry Type System are both decidable problems. Moreover, for the typability problem there exists a function that, for any typable term M, returns the most general type for M in this system. Such type is called the *principal type* of the term.

We first introduce the notions of principal pair and principal type.

Definition 8 (Principal pair) *Let* M *be a term in* Λ*. Then* (Γ, τ) *is called a* principal pair *for* M *if:*

1. $\Gamma \vdash_{\mathcal{C}} M : \tau$*;*
2. *If* $\Gamma' \vdash_{\mathcal{C}} M : \tau'$*, then* $\exists\ \mathbb{S}.\ (\mathbb{S}(\Gamma) \subseteq \Gamma'$ *and* $\mathbb{S}(\tau) \equiv \tau')$*.*

Note that, if (Γ, τ) is a principal pair for a term M, then $\mathsf{fv}(M) = \mathsf{dom}(\Gamma)$, where $\mathsf{fv}(M)$ denotes the set of free variables of M.

Definition 9 (Principal type) *Let* M *be a closed term in* Λ*. Then* τ *is called a* principal type *for* M *if:*

1. $\vdash_{\mathcal{C}} M : \tau$*;*
2. *If* $\vdash_{\mathcal{C}} M : \tau'$*, then* $\exists\ \mathbb{S}.\ (\mathbb{S}(\tau) \equiv \tau')$*.*

The principal type of a term M is a characterisation of the set of types that can be assigned to the term M. Note that every type that can be assigned to M, can be obtained from the principal type of M by applying a type-substitution.

The type-substitution $\mathbb{S}$ is found using Robinson's of first-order unification, which is decidable.

Type-inference

We now present a principal type algorithm for the Curry Type System, based on Robinson's unification algorithm that given a term M, return its principal typing.

To type-check a given type for a given term, one can use the type inference algorithms to calculate the principal type of the term and then verify if the given type is an instance of the principal type.

Unification

Robinson's unification algorithm plays a key role in type inference. We will present a definition of the unification algorithm for the particular case where the terms we want to unify are simple types.

Definition 10 (Unifier) *A unifier between two types τ_1 and τ_2, is a type-substitution $\mathbb{S}$ such that $\mathbb{S}\,\tau_1 = \mathbb{S}\,\tau_2$. If two types have a unifier then we say that they are unifiable. We call $\mathbb{S}\,\tau_1$ (or $\mathbb{S}\,\tau_2$), a common instance.*

Example 11 *The types $(\alpha \to \beta \to \alpha)$ and $((\gamma \to \gamma) \to \delta)$ are unifiable. For the substitution $\mathbb{S} = [(\gamma \to \gamma)/\alpha, \beta \to (\gamma \to \gamma)/\delta]$, the common instance is $((\gamma \to \gamma) \to \beta \to (\gamma \to \gamma))$.*

Definition 12 (Most general unifier) *$\mathbb{S}$ is a most general unifier (mgu) of τ_1 and τ_2 if, for any other unifier $\mathbb{S}_1$, of τ_1 and τ_2, there is a substitution $\mathbb{S}_2$ such that $\mathbb{S}_1 = \mathbb{S}_2 \circ \mathbb{S}$.*

Example 13 *Consider the types $\tau_1 = (\alpha \to \alpha)$ e $\tau_2 = (\beta \to \gamma)$. The substitution $\mathbb{S}' = [(\alpha_1 \to \alpha_2)/\alpha, (\alpha_1 \to \alpha_2)/\beta, (\alpha_1 \to \alpha_2)/\gamma]$ is a unifier of τ_1 and τ_2, but it is not the mgu.*

The mgu of τ_1 and τ_2 is $\mathbb{S} = [\alpha/\beta, \alpha/\gamma]$. The common instance of τ_1 and τ_2 by $\mathbb{S}'$, $(\alpha_1 \to \alpha_2) \to (\alpha_1 \to \alpha_2)$ is an instance of $(\alpha \to \alpha)$.

We now present the Unification algorithm we will use to define type inference. The function UNIFY, given two types τ_1 and τ_2, returns the mgu of τ_1 and τ_2 if it exists, and fails otherwise.

Definition 14 (Unification Algorithm) *Let τ_1 and τ_2 be two types. The unification function $\mathsf{UNIFY}(\tau_1, \tau_2)$ is inductively defined as:*

$$
\begin{array}{lcl}
\mathsf{UNIFY}(\alpha, \tau) & = & [\tau/\alpha] \quad \textit{if } \alpha \notin \mathsf{fv}(\tau) \\
 & = & Id \quad \textit{(identity function) if } \tau = \alpha \\
 & = & fail \quad \textit{otherwise} \\
 & & \\
\mathsf{UNIFY}(\tau_1 \to \tau_2, \alpha) & = & \mathsf{UNIFY}(\alpha, \tau_1 \to \tau_2) \\
 & & \\
\mathsf{UNIFY}(\sigma_1 \to \sigma_2, \tau_1 \to \tau_2) & = & \textit{let} \\
 & & \quad \mathbb{S} = \mathsf{UNIFY}(\sigma_2, \tau_2) \\
 & & \textit{in} \;\; \mathsf{UNIFY}(\mathbb{S}\,\sigma_1, \mathbb{S}\,\tau_1) \circ \mathbb{S}
\end{array}
$$

Note that let $\mathbb{S} = \mathsf{UNIFY}(\sigma_2, \tau_2)$ in $\mathsf{UNIFY}(\mathbb{S}\,\sigma_1, \mathbb{S}\,\tau_1) \circ \mathbb{S}$, fails if one of the calls of UNIFY fails.

Milner's Type-Inference Algorithm

The following algorithm, defines a function that, given a term M returns a basis Γ and a type τ such that $\Gamma \vdash M : \tau$, is the principal typing of M.

Definition 15 *Let Γ be a basis, M a term and τ a type. Let UNIFY be the unification function defined above. The function $T(M) = (\Gamma, \tau)$ defines a type inference algorithm for the simply typed λ-calculus, in the following way:*

1. *If M is a variable x, then $\Gamma = \{x : \alpha\}$ and $\tau = \alpha$, where α is a new variable;*
2. *If $M \equiv M_1 M_2$, $T(M_1) = (\Gamma_1, \tau_1)$ e $T(M_2) = (\Gamma_2, \tau_2)$, then let $\{v_1, \ldots, v_n\} = \mathsf{dom}(M_1) \cap \mathsf{dom}(M_2)$, such that $v_1 : \delta_1, \ldots, v_n : \delta_n \in \Gamma_1$ and $v_1 : \delta'_1, \ldots, v_n : \delta'_n \in \Gamma_2$. Then*

   $T(M) = (\mathbb{S} \circ \mathbb{S}_n \circ \cdots \circ \mathbb{S}_1(\Gamma_1 \cup \Gamma_2), \mathbb{S}\alpha)$, where

   - $\mathbb{S}_1 = \mathsf{UNIFY}(\delta_1, \delta'_1)$;
   - $\mathbb{S}_2 = \mathsf{UNIFY}(\mathbb{S}_1(\delta_1), \mathbb{S}_1(\delta'_1))$;
   - ...
   - $\mathbb{S}_n = \mathsf{UNIFY}(\mathbb{S}_{n-1} \circ \cdots \circ \mathbb{S}_1(\delta_n), \mathbb{S}_{n-1} \circ \cdots \circ \mathbb{S}_1(\delta'_n))$;
   - $\mathbb{S} = \mathsf{UNIFY}(\mathbb{S}_n \circ \cdots \circ \mathbb{S}_1(\tau_1), \mathbb{S}_n \circ \cdots \circ \mathbb{S}_1(\tau_2) \to \alpha)$ *and α is a new variable;*
3. *If $M \equiv \lambda x.N$ and $T(N) = (\Gamma_N, \sigma)$ then:*

   a) *If $x \notin \mathsf{dom}(\Gamma_N)$, then $T(M) = (\Gamma_N, \alpha \to \sigma)$, where α is a new variable;*
   b) *If $\{x : \tau\} \in \Gamma_N$, $T(M) = (\Gamma_N \setminus \{x : \tau\}, \tau \to \sigma)$.*