Type Reconstruction

We consider the simply typed lambda calculus with types and terms as follows:

$$T \in Ty = X \mid T \to T$$
$$t \in Ter = x \mid \lambda x{:}T.t \mid t\,t$$

Type Substitutions

A **type substitution** is a function $\sigma \in Ty \to Ty$ with the following properties:

1. $\forall T_1, T_2\colon\ \sigma(T_1 \to T_2) = \sigma T_1 \to \sigma T_1$ (Compatibility)
2. $\{ X \mid \sigma X \neq X \}$ finite (Finiteness)

The symbol σ will always denote a type substitution. We use *TSubst* to denote the set of all type substitutions and define:

$Var\ \sigma \stackrel{\text{def}}{=} \{ X \mid \sigma X \neq X \}$		instantiated variables
$Con\ \sigma \stackrel{\text{def}}{=} \{ (X, \sigma X) \mid \sigma X \neq X \}$		constraint representation
$\sigma \circ \sigma' \stackrel{\text{def}}{=} \lambda T \in Ty.\ \sigma'(\sigma T)$		composition
$Gen\ \sigma \stackrel{\text{def}}{=} \{ \sigma \circ \sigma' \mid \sigma' \in TSubst \}$		generated substitutions
$id \stackrel{\text{def}}{=} \lambda T \in Ty.\ T$		identity substitution

The identity *id* is a type substitution with $Con\ id = \emptyset$ and $Gen\ id = TSubst$. A type substitution σ is **idempotent** if $\sigma = \sigma \circ \sigma$. Note that *id* is idempotent. A type substitution σ is a **renaming** if it is bijective.

Proposition 1 For all $\sigma, \sigma' \in TSubst$:

1. $\sigma \circ \sigma' \in TSubst$
2. $\sigma = \sigma' \iff Con\ \sigma = Con\ \sigma'$
3. $Gen\ \sigma = Gen\ \sigma' \iff \exists$ renaming $\sigma''\colon\ \sigma = \sigma' \circ \sigma''$

Note that $Con\ \sigma$ is a finite representation of σ.

Proposition 2 Let $C = \{X_1 = T_1, \ \ldots\ , \ X_n = T_n\}$ where $X_1, \ldots, X_n$ are pairwise distinct and $X_i \neq T_i$ for all $i \in \{1, \ldots, n\}$. Then there exists exactly one type substitution σ such that $Con\ \sigma = C$.

We say that σ is **more general** than σ' if $Gen\ \sigma' \subseteq Gen\ \sigma$.

Constraints and Unification

A **constraint** is a finite subset of Ty^2. We see a constraint as a finite set of equations between types and take the freedom to write $T = T'$ for a pair (T, T'). The letter C will always denote a constraint. For every type substitution σ we see the set $Con\,\sigma$ as a constraint. This means that every type substitution can be represented as a constraint.

Let C be a constraint. A type substitution σ is called a

- **solution of** C if $\forall (T, T') \in C\colon \sigma T = \sigma T'$
- **principal solution of** C if $Gen\,\sigma$ is the set of all solutions of C.

We say that σ **solves** C if σ is a solution of C. A constraint is called **satisfiable** if it has a solution.

Unification Theorem If a constraint has a solution, then it has a principal solution that is idempotent.

We will develop a **unification algorithm** that computes for a constraint an idempotent and principal solution if there exists one.

Equivalence of constraints is defined as follows:

$$C \approx C' \quad \overset{\text{def}}{\iff} \quad \forall \sigma\colon\ \sigma \text{ solves } C \iff \sigma \text{ solves } C'$$

Proposition 3 (Unification Rules)

1. $C \cup \{T = T\} \approx C$ (Reflexivity)
2. $C \cup \{T_1 \to T_2 = T_1' \to T_2'\} \approx C \cup \{T_1 = T_1',\ T_2 = T_2'\}$ (Decomposition)
3. $C \cup \{X = T\} \approx C[X := T] \cup \{X = T\}$ (Replacement)
4. $C \cup \{T_1 = T_2\} \approx C \cup \{T_2 = T_1\}$ (Symmetry)
5. If C contains an equation $X = T_1 \to T_2$ where X occurs in T_1 or T_2, then C is unsatisfiable.

A constraint C is **solved** if it has the form $\{X_1 = T_1,\ \ldots\ ,\ X_n = T_n\}$ where $X_1, \ldots, X_n$ are pairwise distinct type variables that don't occur in the types $T_1, \ldots, T_n$.

Proposition 4 C solved $\iff$ $\exists \sigma\colon\ C = Con\,\sigma\ \wedge\ \sigma$ idempotent.

Proposition 5 If σ is idempotent, then σ is a principal solution of $Con\,\sigma$.

A constraint is **explicit** if it is solved or has the form $\{X = X \to X\}$.

Proposition 6 An explicit constraint is satisfiable if and only if it is solved.

The unification algorithm simplifies a constraint until an equivalent explicit constraint is obtained. The required simplification steps appear in Proposition 3. Here is an example:

$$\begin{array}{rcll} \{X \to Y = Z \to X \to Z\} & \approx & \{X = Z,\ Y = X \to Z\} & \text{decomposition} \\ & \approx & \{X = Z,\ Y = Z \to Z\} & \text{replacement} \end{array}$$

To obtain a terminating algorithm, we work with two constraints C_1, C_2 satisfying the following conditions:

1. $C_1 \cup C_2$ is equivalent to the initial constraint.
2. C_2 is solved.
3. $\forall (X,T) \in C_2\colon\ X$ doesn't occur in C_1.

We always start with $C, \emptyset$. If the initial constraint is satisfiable, we end up with $\emptyset, C'$ where C' is solved and equivalent to the initial constraint. Otherwise, we end up with $\{X = X \to X\}, \emptyset$. The solved part of the constraint grows which each replacement step eliminating a variable from the unsolved part.

Proposition 7 If $C = \{X_1 = T_1,\ \ldots\ ,\ X_n = T_n\}$ is solved and X and $X_1, \ldots, X_n$ don't occur in T, then $C[X := T] \cup \{X = T\}$ is solved.

Typings and Constraint Extraction

Proposition 8 $\Gamma \vdash t : T \implies \sigma\Gamma \vdash \sigma t : \sigma T$

Given Γ and t, we call a type substitution σ a

- **typing of** Γ, t if $\exists T\colon \sigma\Gamma \vdash \sigma t : T$.
- **principal typing of** Γ, t if $Gen\ \sigma$ is the set of all typings of Γ, t.

Note that a principal typing of Γ, t is a typing of Γ, t that represents all other typings of Γ, t. Example: id is a principal typing of $\emptyset, \lambda x\!:\!X.x$.

Typing Theorem If Γ and t have a typing, then they have a principal typing that is idempotent.

A **type reconstruction algorithm** that computes for Γ, t an idempotent and principal typing if there exists one can be obtained by combining a unification algorithm with an **extraction algorithm** that computes for Γ and t a constraint C such that the solutions of C are the typings of Γ, t up to a minor modification.

We will describe the extraction algorithm through inference rules defining an **extraction relation** "$\Gamma, t \rightsquigarrow T, C$" satisfying the following properties:

- $FV\ t \subseteq Dom\,\Gamma \implies \exists T, C\colon\ \Gamma, t \rightsquigarrow T, C$ (Totality)
- $\Gamma, t \rightsquigarrow T, C\ \wedge\ \sigma$ solves $C \implies \sigma\Gamma \vdash \sigma t : \sigma T$ (Soundness)

· $\Gamma, t \rightsquigarrow T, C \ \wedge \ \sigma\Gamma \vdash \sigma t : T' \implies$ (Completeness)
$\exists \sigma' \colon \ \sigma'$ solves $C \ \wedge \ \forall X \in TV(\Gamma, t) \colon \ \sigma X = \sigma' X$

where $TV(\Gamma, t)$ denotes the set of all type variables that occur in Γ, t.

If σ is a type substitution and V is a set of type variables, the type substitution $\sigma|V$ called the **restriction of σ to V** is defined as follows:

$$Con(\sigma|V) = \{ (X, T) \in Con\, \sigma \mid X \in V \}$$

Extraction Theorem Let $\Gamma, t \rightsquigarrow T, C$. Then:

1. Γ, t has a typing $\iff$ C has a solution.
2. σ principal solution of $C \implies \sigma|TV(\Gamma, t)$ principal typing of Γ, t.

The Extraction Theorem is a consequence of the soundness and completeness of the extraction relation. The Typing Theorem follows with the totality of the extraction relation from the Extraction Theorem and the Unification Theorem. The theorems tell us that a type reconstruction algorithm can be obtained from a type extraction algorithm and a unification algorithm.

The extraction relation is defined by the following inference rules:

$$\frac{\Gamma x = T}{\Gamma, x \ \rightsquigarrow \ T, \emptyset} \qquad \frac{\Gamma[x := T], t \ \rightsquigarrow \ T', C}{\Gamma, \lambda x{:}T.t \ \rightsquigarrow \ T \to T', C}$$

$$\frac{\Gamma, t_1 \rightsquigarrow T_1, C_1 \quad \Gamma, t_2 \rightsquigarrow T_2, C_2 \quad X \text{ fresh}}{\Gamma, \ t_1 t_2 \ \rightsquigarrow \ X, \ C_1 \cup C_2 \cup \{T_1 = T_2 \to X\}}$$

where "X fresh" is an abbreviation for the lengthy condition

$$V_1 \cap W = \emptyset \ \wedge \ V_2 \cap W = \emptyset \ \wedge \ V_1 \cap V_2 = \emptyset \ \wedge \ X \notin V_1 \cup V_2 \cup W$$
$$\text{where} \ \ V_1 = TV(T_1, C_1) - TV(\Gamma, t_1)$$
$$V_2 = TV(T_2, C_2) - TV(\Gamma, t_2)$$
$$W = TV(\Gamma, t_1 t_2)$$

The inference rules defining the extraction relation also define an algorithm that for Γ and t computes T and C such that $\Gamma, t \rightsquigarrow T, C$.

The soundness and completeness properties of the extraction relation can be shown by induction on $|t|$. The totality property requires a proof of the stronger claim

$$FV\ t \subseteq Dom\, \Gamma \ \wedge \ V \text{ finite} \implies \exists T, C \colon \ \Gamma, t \rightsquigarrow T, C \ \wedge \ TV(T, C) \cap V \subseteq TV(\Gamma, t)$$

which once again can be carried out by induction on $|t|$.

Principal Types

A type T is a **principal type** of a closed term t if

$$\forall T' \colon \ (\exists \sigma \colon \ \emptyset \vdash \sigma t : T') \iff (\exists \sigma \colon \ T' = \sigma T)$$

Proposition 8 Let σ be a principal typing of $\emptyset$, t and $\emptyset \vdash \sigma t : T$. Then T is a principal type of t.