# Type Inference: Algorithm W and Algorithm U

by *Yinyanghu* on **March 13, 2014**

Tagged as: Algorithm, Lambda Calculus, Type Inference, Logic, Polymorphism, Type System, Hindley-Milner, Programming Language.

## Type Substitution

A type substitution $\sigma$ maps type variables to types or type variables.

- $\sigma(X) = T$, if $(X \to T) \in \sigma$
- otherwise, $\sigma(X) = X$

The composition of two substitutions is

$\sigma \circ \gamma = \{X \to \sigma(T) \mid (X \to T) \in \gamma\} \cup \{X \to T \mid (X \to T) \in \sigma, X \notin domain(\gamma)\}$

When apply the substitution $\sigma$ to term $T$, $\sigma T$ is $[X_i \to \sigma(X_i)]T$, where $X_i$ denotes the variables in $T$.

Similarly, when apply the substitution $\sigma$ to context $\Gamma$, $\sigma\Gamma$ is $\{\sigma T_i\}$, where $T_i \in \Gamma$.

## Algorithm W

This is the original type inference algorithm of Damas and Miller.

- The input to the algorithm is an expression and a typing environment (context) which is a set of assumptions, i.e. bindings of type variables to type expressions.
- The output of the algorithm is a type for the given expression, and a substitution of type expressions for types which results in the overall type.

$W(\Gamma, expr) = (\sigma, T)$, where

- $\Gamma$ is the context
- $expr$ is the expression to be typed
- $\sigma$ is the substitution of type expressions for type variables which gives the value of $T$
- $T$ is the type of $expr$

Several cases:

- if $expr$ is a **variable**, say $x$, then

  $W(\Gamma, expr) = (\emptyset, instantiate(x))$, where

  - $instantiate(x) = [a_1 \to b_1][a_2 \to b_2] \cdots [a_n \to b_n]T_x$,
  - $x : (\forall a_1, a_2, \cdots, a_n)T_x \in \Gamma$,
  - and $b_1, b_2, \cdots, b_n$ are fresh variables
- if $expr$ is an **abstraction** ($\lambda$ expression), say $\lambda x.f$, then

  $W(\Gamma, expr) = (\sigma_1, \sigma_1 X \to T_f)$, where

  - $W(\Gamma \cup \{x : X\}, f) = (\sigma_1, T_f)$,
  - and $X$ is a fresh variable
- if $expr$ is an **application**, say $expr = f g$, then

  $W(\Gamma, expr) = (\sigma_3 \circ \sigma_2 \circ \sigma_1, \sigma_3 X)$, where

  - $W(\Gamma, f) = (\sigma_1, T_f)$,
  - $W(\sigma_1 \Gamma, g) = (\sigma_2, T_g)$,
  - $U(\sigma_2 T_f, T_g \to X) = \sigma_3$,
  - and $X$ is a fresh variable
- if $expr$ is a **conditional**, say if $cond$ then $f$ else $g$, then

  $W(\Gamma, expr) = (\sigma_5 \circ \sigma_4 \circ \sigma_3 \circ \sigma_2 \circ \sigma_1, \sigma_5 T_g)$, where

  - $W(\Gamma, cond) = (\sigma_1, T_{cond})$,
  - $U(T_{cond}, Bool) = \sigma_2$,
  - $W(\sigma_2\sigma_1\Gamma, f) = (\sigma_3, T_f)$,
  - $W(\sigma_3\sigma_2\sigma_1\Gamma, g) = (\sigma_4, T_g)$,
  - and $U(\sigma_4 T_f, T_g) = \sigma_5$
- if $expr$ is a **fix-point expression**, say fix $x.f$, then

  $W(\Gamma, expr) = (\sigma_2 \circ \sigma_1, \sigma_2 \circ \sigma_1 \circ X)$, where

  - $W(\Gamma \cup \{x : X\}, f) = (\sigma_1, T_f)$,
  - $U(\sigma_1 X, T_f) = \sigma_2$,
  - and $X$ is a fresh variable
- if $expr$ is a **let expression**, say let $x = f$ in $g$, then

  $W(\Gamma, expr) = (\sigma_2 \circ \sigma_1, T_g)$, where

  - $W(\Gamma, f) = (\sigma_1, T_f)$,
  - $W(\sigma_1\Gamma \cup \{x : poly(T_f)\}, g) = (\sigma_2, T_g)$, where $poly(T_f) = (\forall x_1, x_2, \cdots, x_n)T_f$, and $x_1, x_2, \cdots, x_n$ are the free variables in $T_f$ which do not appear in $\sigma_1\Gamma$.

## Algorithm U

- Algorithm U solves unification which is what we need to complete our description of Algorithm W.
- The input to the algorithm is two type expressions.
- The output of the algorithm is a substitution or an error if we cannot find an unification.

$U(T_1, T_2) = \sigma$, where

- $T_1, T_2$ are the type expressions to be unified
- $\sigma$ is the substitution if we find an unification of $T_1$ and $T_2$

Also several cases:

- if both $T_1$ and $T_2$ are base type, then
  - $U(T_1, T_2) = \emptyset$, if $T_1 = T_2$
  - otherwise, we find an error
- if both $T_1$ and $T_2$ are type variables, then
  - $U(T_1, T_2) = \emptyset$, if $T_1 = T_2$
  - if $T_1$ **occurs** in $T_2$, or $T_2$ **occurs** in $T_1$, then we find an error: **circularity** (e.g. $\lambda x.\, xx$)
  - otherwise, $U(T_1, T_2) = \{T_1 \to T_2\}$
- if $T_1$ and $T_2$ have the same type constructor $C$, i.e. $T_1 = C(A_1, A_2, \cdots, A_k)$ and $T_2 = C(B_1, B_2, \cdots, B_k)$, then

  $U(T_1, T_2) = \sigma_k \circ \sigma_{k-1} \circ \cdots \sigma_1$, where $\sigma_i = U(A_i, B_i)$
- otherwise, we find an error

---

- *Damas* and *Milner* proved that Algorithm W computes the **principal type** scheme for a given expression and context.
- According to *Cardelli(1985)*, the order of type inference does NOT affect the final result and it solves the system of type constraints.
- This version taken from *Field* and *Harrison*, also treats expressions involving the fix-point operator **fix**.

## Reference


- Algorithm W
- Wikipedia: Hindley-Milner type system
- University of Waterloo, CS442 Lecture Note
- Lecture 22: Type Inference and Unification
- Lecture 26: Type Inference and Unification


Yinyanghu, 2014