<u>Def</u> Substitycja s

$s : Var \to Term$

taka że $|\{x \in Var \mid s(x) \neq x\}| < \infty$

piszemy $s = [x_1/t_1, \dots x_n/t_n]$

Substitucja $\sigma : Term \to Term$

$\sigma(a) = a$, a stała

$\sigma(x) = s(x)$, x zmienna

$\sigma(f(t_1, \dots t_n)) = f(\sigma(t_1), \dots \sigma(t_n))$

<u>Def</u> Unifikator

Substitucja σ unifikuje t_1 i t_2 ($t_1 =_\sigma t_2$), jeżeli $\sigma(t_1) \equiv \sigma(t_2)$.

σ jest najbardziej ogólnym unifikatorem dla t_1 i t_2 ($\sigma = mgu(t_1, t_2)$), jeżeli

(i) $t_1 =_\sigma t_2$

(ii) $\forall \sigma' : t_1 =_{\sigma'} t_2 \Rightarrow \exists \tau : \sigma' = \tau \circ \sigma$

<u>Przykład:</u>

(i) $t_1 = p(a, y, z)$, $t_2 = p(x, b, z)$

$\sigma = [x/a, y/b, z/f(a)]$ unifikuje t_1 i t_2

$\sigma' = [x/a, y/b]$ jest $mgu(t_1, t_2)$

$\tau := [z/f(a)] \rightsquigarrow \sigma = \tau \circ \sigma'$

(ii) $f(x, g(a,b)) =_\sigma f(g(y,b), x)$

dla $\sigma = [x / g(a,b), y/a]$

ale nie dla $\sigma' = [x / g(y,b), y/a]$

(iii) $t_1 = (\alpha, Int)$, $t_2 = (\beta, \gamma)$, $t_3 = (\beta, \alpha)$

$mgu(t_1, t_2) = [\beta/\alpha, \gamma/Int]$

$mgu(t_1, t_3) = [\alpha/Int, \beta/Int]$

$mgu(t_2, t_3) = [\gamma/\alpha]$

(iv) $t_1 = (\alpha \to \beta) \to \alpha$, $t_2 = Int \to Char$

nie unifikują się,

ale $mgu(\alpha \to \beta, t_2) = [\alpha/Int, \beta/Char]$

i $mgu(\alpha \to \beta, t_2)(t_1) = (Int \to Char) \to Int$

<u>Algorytm Unifikacji</u>

obliczenie mgu dla t_1 i t_2
na podstawie struktur termów

t_1 i t_2 unifikują się, jeżeli

- mają ten sam operator
- argumenty się unifikują, czyli
  - jeden z nich jest zmienną lub
  - są tą samą stałą lub
  - unifikują się rekurencyjnie

Algorytm Unify

Input: $E = \{ u_1 = v_1, \dots u_n = v_n \}$

Output: $\sigma =$ mgu E, o ile mgu E istnieje

1. $unify(\{\}) = \{\}$
2. $unify(\{ f(t_1, \dots t_n) = g(t'_1, \dots t'_m) \} \cup E) =$

   if $f \neq g$

   then "σ nie istnieje"

   else $unify(\{ t_1 = t'_1, \dots t_n = t'_n \} \cup E)$
3. $unify(E \cup \{ x = t \}) = unify(E \cup \{ t = x \}) =$

   if x wystąpi w t

   then "σ nie istnieje"

   else $unify(E[x/t]) \cup \{[x/t]\}$
4. $unify(E \cup \{ t = t \}) = unify(E)$

Przykład:

$unify(\{ f(x, g(a,b)) = f(g(y,b), x \})$

$= unify(\{ x = g(y,b), g(a,b) = x \})$

$= unify(\{ g(a,b) = g(y,b) \}) \cup [x/g(y,b)]$

$= unify(\{ a = y, b = b \}) \cup [x/g(y,b)]$

$= unify(\{ a = y \}) \cup [x/g(y,b)]$

$= unify(\{\}) \cup [y/a, x/g(y,b]$

$= [y/a, x/g(y,b)]$

czyli $\sigma = [y/a, x/g(a,b)]$

Reguła dla aplikacji

$$\frac{T \vdash f :: \alpha \to \beta \qquad T' \vdash e :: \gamma}{\sigma(T \cup T') \vdash (f\ e) :: \sigma(\beta)} \quad \sigma = mgu(\alpha, \gamma)$$

Przykłady

(i) 1. $\emptyset \vdash (+) :: Int \to (Int \to Int)$ Ax1

2. $\{a :: \alpha\} \vdash a :: \alpha$ Ax2

3. $\{a :: Int\} \vdash (a+) :: Int \to Int$ z 1. i 2.

bo $\sigma = mgu(Int, \alpha) = [\alpha / Int]$

i $\sigma(Int \to Int) = Int \to Int$

4. $\{x :: \beta\} \vdash x :: \beta$ Ax2

5. $\{a :: Int, x :: Int\} \vdash (a+x) :: Int$ z 3. i 4.

6. $\{a :: Int\} \vdash (\backslash x \to a+x) :: Int \to Int$ z 5.

(ii) 1. $\{f :: \alpha'\} \vdash f :: \alpha'$ Ax2

2. $\emptyset \vdash 3 :: Int$ Ax1

3. $\{f :: Int \to \gamma'\} \vdash (f\ 3) :: \gamma'$ z 1. i 2.

bo $\sigma(\alpha') = \sigma(\alpha \to \beta)$

$\sigma(\gamma) = Int$

czyli $\sigma = mgu(\alpha, \gamma) = [\alpha / Int]$

więc $\sigma(\alpha') = Int \to \beta = Int \to \gamma'$

4. $\emptyset \vdash (\backslash f \to f\ 3) :: (Int \to \gamma') \to \gamma'$ z 3.

(iii) $\emptyset \vdash [] :: [\alpha]$

$\emptyset \vdash \text{'a'} :: Char$

$\emptyset \vdash (:) :: \beta \to [\beta] \to [\beta]$

$\emptyset \vdash (\text{'a'}:) :: [Char] \to [Char]$

$\emptyset \vdash \underbrace{(\text{'a'} : [])}_{=[a]} :: [Char]$

czyli z $\emptyset \vdash length :: [\gamma] \to Int$

dostaniemy $\emptyset \vdash (length\,[\text{'a'}]) :: Int$

ale length jest zdefiniowana rekurencyjnie!

↳ Algorytm Milner'a

- Analyza lewych stron:

$f\ t_1 \ldots t_n = \ldots$

↳ $type(f) = \alpha_1 \to \alpha_2 \to \ldots \to \alpha_n \to \alpha_{n+1}$

t_i variable, czyli $type(t_i) = \beta$ lub

t_i applikacja konstruktora (n.p. $(x:xs)$)

czyli $type(t_i) = [\gamma]$ (jeżeli n.p. $t_i \equiv (x:xs)$)

- Analyza prawych stron:

$\ldots = e_1 e_2$

↳ $type(e_1) = type(e_2) \to type(e_1 e_2)$

Analyza daje zbiór równań dla typów.

Twierdzenie

funkcja f ma typowanie
wtw. zbiór równań ma rozwiązanie
wtw. dla zbioru równań istnieje unifikator.
Najbardziej ogólny unifikator daje najbardziej ogólny typ funkcji f.

Przykład:

length [] = 0
length (x:xs) = 1 + length xs

$length :: \alpha_1 \longrightarrow \alpha_2$

(i) $\alpha_1 = type([]) = [\alpha]$

$\alpha_1 = type(x:xs)$

$type(:) = type(x) \longrightarrow type(x:)$

$type(x:) = type(xs) \longrightarrow type(x:xs)$

↳ $type(:) = type(x) \longrightarrow (type(xs) \longrightarrow type(x:xs))$

$\qquad\qquad \| $

$\qquad \alpha' \longrightarrow [\alpha'] \longrightarrow [\alpha']$

↳ $type(x) = \alpha'$, czyli $x :: \alpha$

$xs :: [\alpha]$, $(x:xs) :: [\alpha]$

(bo $type(x:xs) = type([]) = [\alpha]$, czyli $\alpha = \alpha'$

(ii) $\alpha_2 = type(0) = Int$

$\alpha_2 = type(1 + length\ xs)$

$type(+) = type(1) \longrightarrow type(1+)$

$type(1+) = type(length\ xs) \longrightarrow type(1 + length\ xs)$

$= \alpha_2 \longrightarrow \alpha_2$

$\hookrightarrow type(+) = Int \rightarrow \alpha_2 \longrightarrow \alpha_2$

$= Int \rightarrow Int \rightarrow Int$

wynik: $length :: [\alpha] \rightarrow Int$

uwaga: jest więcej równań, n.p.

$type(length) = type(xs) \rightarrow typ(length\ xs)$

$= [\alpha] \rightarrow \alpha_2$

$= [\alpha] \rightarrow Int$

Przykład:

map f [] = []

map f (x:xs) = (f x) : (map f xs)

$map :: \alpha_1 \longrightarrow \alpha_2 \longrightarrow \alpha_3$

(i) $\alpha_2 = [\alpha]$

oraz $x :: \alpha$, $xs :: [\alpha]$, $(x:xs) :: [\alpha]$

$f :: \gamma\ (= \alpha_1)$

(ii) $\alpha_3 = type([]) = [\beta]$

$\alpha_3 = type((f\,x):(map\ f\ xs))$

$type((f\,x):)$

$= type(map\ f\ xs) \longrightarrow type((f\,x):(map\ f\ xs))$

$= \alpha_3 \longrightarrow \alpha_3$

$= [\beta] \longrightarrow [\beta]$

↳ $type(:) = type(f\,x) \longrightarrow type((f\,x):)$

$= type(f\,x) \longrightarrow ([\beta] \rightarrow [\beta])$

↳ $type(f\,x) = \beta$

$\alpha_1 = type(f)$

$= type(x) \longrightarrow type(f\,x)$

$= \alpha \longrightarrow \beta$

<u>wynik</u>: $map :: (\alpha \rightarrow \beta) \rightarrow [\alpha] \rightarrow [\beta]$