Lecture 8 – Unification & Type Inference

Violet Ka I Pun

violet@ifi.uio.no

Substitution and unification

- Subsitiution
  - a number of equations, $\sigma = \{x_1 = t_1, \ldots, x_n = t_n\}$, where the variables on the left are different
  - Applying a substitution σ on a term t, $\sigma(t)$: replace simultaneously all instances of each x_i in t with t_i
  - E.g.: $\sigma = \{x = f(x, y)\}$, $t = g(a(x, y), x)$, $\sigma(t)$
- Unification ('union') of $\{t_1 = t_1', \ldots, t_n = t_n'\}$:
  - a substitution which give syntactic equality, i.e.,
  - a σ such that $\sigma(t_1) = \sigma(t_1'), \ldots, \sigma(t_n) = \sigma(t_n')$.
  - σ is called a unifier
- Using a substitution $\sigma = \{x = s\}$ on a term t, i.e., $\sigma(t)$, written also as $t[x/s]$.

Martelli-Montanari unification algorithm

- Non-deterministic
- The results are equivalent
- Input: $E = \{t_1 = t_1', \ldots, t_n = t_n'\}$
- Output: a unification or NO
- Condition: output should be a most general unifier (mgu) of E (each other unification can be obtained by means of substitution)
- Example: $\{f(X, a) = f(Y, a)\}$ is unified by

  (a) $\{X = a, Y = a\} \quad \rightsquigarrow \quad \{f(a, a) = f(a, a)\}$

  (b) $\{X = b, Y = b\} \quad \rightsquigarrow \quad \{f(b, a) = f(b, a)\}$

  (c) $\{X = Y\} \quad \rightsquigarrow \quad \{f(Y, a) = f(Y, a)\}$

  (d) $\{Y = X\} \quad \rightsquigarrow \quad \{f(X, a) = f(X, a)\}$

! Only (c) and (d) are mgu.

Martelli-Montanari unification algorithm

- Input: $E = \{t_1 = t'_1, \ldots, t_n = t'_n\}$
- Algorithm: transforms E by the following rules:
  - Remove $x = x$ and $c = c$ for variables x and atoms c
  - Turn $f(t_1, \ldots, t_k) = x$ to $x = f(t_1, \ldots, t_k)$ (including f a constant)
  - Replace $f(t_1, \ldots, t_k) = f(s_1, \ldots, s_k)$ with $t_1 = s_1, \ldots, t_k = s_k$
  - If there exists $x = t$ where x does not occur in t ('occurs check'), replace all equations $s = r$ (except $x = t$ itself) with $s[x/t] = r[x/t]$.
- Stop when

NO: if you find a equation where
– occurs check fails: $x = t$ and x occurs in t, or
– $f(\ldots) = g(\ldots)$ and $f =/= g$ (including constants).

YES: and returns the transformed E, if:

1. in each equation, the left hand side is a variable, and
2. each variable which occurs on the left of an equation does not occur anywhere else

Unification: Examples

$$
\begin{array}{lll}
E, t = t & \Rightarrow E & \\
E, f(t_1 ... t_n) = f(s_1 ... s_n) & \Rightarrow E, t_1 = s_1, ..., t_n = s_n & \\
E, f(t_1 ... t_n) = g(s_1 ... s_m) & \Rightarrow NO & f =/= g \vee n \neq m \\
E, f(t_1 ... t_n) = x & \Rightarrow E, x = f(t_1 ... t_n) & \\
E, x = t & \Rightarrow E[x/t], x = t & x \notin Var(t) \\
E, x = t & \Rightarrow NO & x \in Var(t)
\end{array}
$$

1. `p(X,X) = p(Y,Z)` (applying the unifier gives `p(Z,Z) = p(Z,Z)`)
   $\leadsto \{X = Y, X = Z\} \leadsto \{X = Z, Y = Z\}$
2. `p(X,X) = p(Y,f(Y))`
   $\leadsto \{X = Y, X = f(Y)\} \leadsto \{X = Y, Y = f(Y)\}$ (occurs check fails)
3. `mem(H, H:T)) = mem(a,X:[b,c])`

$\leadsto \{H = a, H : T = X : [b,c]\}$	$\{H = a, H : T = d : [b,c]\}$
$\leadsto \{H = a, H = X, T = [b,c]\}$	$\{H = a, H = d, T = [b,c]\}$
$\leadsto \{X = a, H = X, T = [b,c]\}$	$\{d = a, H = d, T = [b,c]\}$
$\leadsto \{X = a, H = a, T = [b,c]\}$	

4. `mem(H,H:T) = mem(a,d:[b,c])`

Unification:

$$E, t = t \Rightarrow E$$
$$E, f(t_1 \ldots t_n) = f(s_1 \ldots s_n) \Rightarrow E, t_1 = s_1, \ldots, t_n = s_n$$
$$E, f(t_1 \ldots t_n) = g(s_1 \ldots s_m) \Rightarrow NO \qquad f =/= g \vee n \neq m$$
$$E, f(t_1 \ldots t_n) = x \Rightarrow E, x = f(t_1 \ldots t_n)$$
$$E, x = t \Rightarrow E[x/t], x = t \qquad x \notin Var(t)$$
$$E, x = t \Rightarrow NO \qquad x \in Var(t)$$

$x = c, a = f(x, e), a = f(e, d), b = g(c, d),$ $\quad y = f(a, b)$

$x = c, a = f(c, e), a = f(e, d), b = g(c, d),$ $\quad y = f(a, b)$

$x = c, a = f(c, e), f(c, e) = f(e, d), b = g(c, d),$ $\quad y = f(f(c, e), b)$

$x = c, a = f(c, e), c = e, e = d, b = g(c, d),$ $\quad y = f(f(c, e), b)$

$x = c, a = f(c, d), c = d, e = d, b = g(c, d),$ $\quad y = f(f(c, d), b)$

$x = d, a = f(d, d), c = d, e = d, b = g(d, d),$ $\quad y = f(f(d, d), b)$

$x = d, a = f(d, d), c = d, e = d, b = g(d, d),$

$y = f(f(d, d), g(d, d))$

Type inference (for simple -> types)

We consider only some function expressions and the types of those:

expr	::=	\exprvar -> expr | funapp
funapp	::=	funapp simpexpr | simpexpr | (*) | ...
simpexpr	::=	exprcon | exprvar | (expr)
exprcon	::=	0 | 1 | ... | 3.14 | ... | True | False
exprvar	::=	lower case letter
type	::=	type' -> type | type'
type'	::=	typecon | typevar | (type)
typecon	::=	Int | Bool | ...
typevar	::=	lower case letter

Type inference problem

Given an environment and an expression:
how can we find the type?

environment	::=	{ typeassignment }
typeassignment	::=	exprvar :: type

Initially: $\emptyset \vdash M :: t$,
where $\emptyset$ is the empty environment,
and M is an expression t is a type variable.

What does t equal to?

e.g.,	`\x -> (*) x 2`	::	`Int -> Int`
	`(\x -> (*) x 2) 6`	::	`Int`
	`\x -> \y -> x y`	::	`(a -> b) -> a -> b`

Hindley-Milners algorithm

Reduces

(1) the type inference problem

$$\Gamma \vdash expr :: t \qquad\qquad (\Gamma :: exprvar \to type)$$

to

(2) unification of an equation system

$$E(\Gamma \mid expr :: t) \qquad\qquad \text{derived from (1).}$$

Induction on the form of *expr* (t,a,b are type variables):

$E(\Gamma \mid con :: t) = \{t = \theta(con)\}$

– look up the type of the constant *con*

$E(\Gamma \mid x :: t) = \{t = \Gamma(x)\}$

– the variable x is checked in the constant

$E(\Gamma \mid f\ g :: t) = E(\Gamma \mid g :: a) \cup E(\Gamma \mid f :: a \to t)$

– a is a fresh type variable, to avoid name conflicts

$E(\Gamma \mid \backslash x \to ex :: t) = \{t = a \to b\} \cup E(\Gamma, x :: a \mid ex :: b)$

– a, b are fresh type variables

Example: $\backslash\mathtt{x} \to \mathtt{x}$

$$
\begin{array}{lll}
(t1) & E(\Gamma \mid con :: t) & = \{t = \theta(con)\} \\
(t2) & E(\Gamma \mid x :: t) & = \{t = \Gamma(x)\} \\
(t3) & E(\Gamma \mid f\ g :: t) & = E(\Gamma \mid g : c) \cup E(\Gamma \mid f :: c \to t) \\
(t4) & E(\Gamma \mid \backslash x \to ex :: t) & = \{t = a \to b\} \cup E(\Gamma, x :: a \mid ex :: b)
\end{array}
$$

To find the type t for $\backslash x \to x$, ask GHCi $> : t \ \ \backslash x \to x$ ...

$$
\begin{array}{rl}
 & E(\emptyset \mid \backslash x \to x :: t) \ = \\
(t4) & \{t = a \to b\} \cup E(x :: a \mid x :: b) \ = \\
(t2) & \{t = a \to b\} \cup \{b = a\} \\
= & \{t = a \to b, b = a\} \ = \\
(MM) & \{t = a \to a, b = a\}
\end{array}
$$

i.e. the answer is $\backslash x \to x :: a \to a$

Example: $\backslash\ \mathtt{x} \to \backslash\ \mathtt{y} \to \mathtt{x}\ (\mathtt{x}\ \mathtt{y})$

$$
\begin{array}{lll}
\text{(t1)} & E(\Gamma \mid con :: t) & = \{t = \theta(con)\} \\
\text{(t2)} & E(\Gamma \mid x :: t) & = \{t = \Gamma(x)\} \\
\text{(t3)} & E(\Gamma \mid f\ g :: t) & = E(\Gamma \mid g :: c) \cup E(\Gamma \mid f :: c \to t) \\
\text{(t4)} & E(\Gamma \mid \backslash x \to ex :: t) & = \{t = a \to b\} \cup E(\Gamma, x :: a \mid ex :: b)
\end{array}
$$

E.g., $E(\emptyset \mid \backslash\mathtt{x}\ \mathtt{->}\ \backslash\mathtt{y}\ \mathtt{->}\ \mathtt{x}\ (\mathtt{x}\ \mathtt{y}) :: \tau\ ) =$

(t4) $\{\tau = a \to b\} \cup E(x :: a \mid \backslash\mathtt{y}\ \mathtt{->}\ \mathtt{x}\ (\mathtt{x}\ \mathtt{y}) :: b)$

(t4) $\{b = c \to d, \tau = a \to b\} \cup E(x :: a, y :: c \mid \mathtt{x}\ (\mathtt{x}\ \mathtt{y})\ \mathtt{::}\ d)$

(t3) $\{b = c \to d, \tau = a \to b\}$ . . . . [i]

$\cup\ E(x :: a, y :: c \mid \mathtt{x}, e \to d)$ . . . . [ii]

$\cup\ E(x :: a, y :: c \mid \mathtt{x}\ \mathtt{y} :: e)$ . . . . [iii]

(t2) for [ii]: $\{a = e \to d\}$

(t3) for [iii]: $E(x :: a, y :: c \mid \mathtt{x} :: f \to e) \cup E(x :: a, y :: c \mid \mathtt{y} :: f)$

(t2) gives $\{a = f \to e, c = f\}$

$[i] \cup [ii] \cup [iii] = \{f = c, a = f \to e, a = e \to d, b = c \to d, \tau = a \to b\}$

Unification: $\backslash\ \mathtt{x} \to \backslash\ \mathtt{y} \to \mathtt{x}\ (\mathtt{x}\ \mathtt{y})$

$$
\begin{array}{lll}
E, t = t & \Rightarrow E & \\
E, f(t_1 \dots t_n) = f(s_1 \dots s_n) & \Rightarrow E, t_1 = s_1, \dots, t_n = s_n & \\
E, f(t_1 \dots t_n) = g(s_1 \dots s_m) & \Rightarrow \bot & f \not\equiv g \lor n \neq m \\
E, f(t_1 \dots t_n) = x & \Rightarrow E, x = f(t_1 \dots t_n) & \\
E, x = t & \Rightarrow E[x/t], x = t & x \notin Var(t) \\
E, x = t & \Rightarrow \bot & x \in Var(t)
\end{array}
$$

$$
\begin{array}{ll}
f = c, a = f \to e, a = e \to d, b = c \to d, & \tau = a \to b \\
f = c, a = c \to e, a = e \to d, b = c \to d, & \tau = a \to b \\
f = c, a = c \to e, c \to e = e \to d, b = c \to d, & \tau = (c \to e) \to b \\
f = c, a = c \to e, c = e, e = d, b = c \to d, & \tau = (c \to e) \to b \\
f = c, a = c \to d, c = d, e = d, b = c \to d, & \tau = (c \to d) \to b \\
f = d, a = d \to d, c = d, e = d, b = d \to d, & \tau = (d \to d) \to b \\
f = d, a = d \to d, c = d, e = d, b = d \to d, & \\
 & \tau = (d \to d) \to d \to d
\end{array}
$$

$\backslash\ \mathtt{x} \to \backslash\ \mathtt{y} \to \mathtt{x}\ (\mathtt{x}\ \mathtt{y}) :: (d \to d) \to d \to d$

Example: $\backslash\mathtt{x} \rightarrow \mathtt{x}\ \mathtt{x}$

$$
\begin{array}{lll}
(\text{t1}) & E(\Gamma \mid con :: t) & = \{t = \theta(con)\} \\
(\text{t2}) & E(\Gamma \mid x :: t) & = \{t = \Gamma(x)\} \\
(\text{t3}) & E(\Gamma \mid f\ g :: t) & = E(\Gamma \mid g :: c) \cup E(\Gamma \mid f :: c \rightarrow t) \\
(\text{t4}) & E(\Gamma \mid \backslash x \rightarrow ex :: t) & = \{t = a \rightarrow b\} \cup E(\Gamma, x :: a \mid ex :: b)
\end{array}
$$

To find the type t for $\backslash x \rightarrow x\ x$, ask GHCi $>: t\ \ \backslash x \rightarrow x\ x$ ...

$$
\begin{array}{ll}
 & E(\emptyset \mid \backslash x \rightarrow x\ x :: t)\ = \\
(\text{t4}) & \{t = a \rightarrow b\} \cup E(x :: a \mid x\ x :: b)\ = \\
(\text{t3}) & \{t = a \rightarrow b\} \cup E(x :: a \mid x :: c) \cup E(x :: a \mid x :: c \rightarrow b)\ = \\
(\text{t2}) & \{t = a \rightarrow b\} \cup \qquad \{c = a\} \cup \qquad \{c \rightarrow b = a\} \\
(\text{MM}) & \{t = c \rightarrow b, a = c, c = c \rightarrow b\}
\end{array}
$$

and the answer is

Occurs check: cannot construct the infinite type $c \sim c \rightarrow b$

- Self-application is not allowed Haskell

Unification: $\backslash x \to x\ x$

$$
\begin{array}{lll}
E, t = t & \Rightarrow E & \\
E, f(t_1 ... t_n) = f(s_1 ... s_n) & \Rightarrow E, t_1 = s_1, ..., t_n = s_n & \\
E, f(t_1 ... t_n) = g(s_1 ... s_m) & \Rightarrow \perp & f \not\equiv g \vee n \neq m \\
E, f(t_1 ... t_n) = x & \Rightarrow E, x = f(t_1 ... t_n) & \\
E, x = t & \Rightarrow E[x/t], x = t & x \notin Var(t) \\
E, x = t & \Rightarrow \perp & x \in Var(t)
\end{array}
$$

$\{c = a, a = c \to b, \tau = a \to b\}$

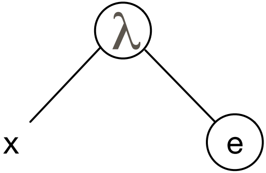
$\{c = a, a = a \to b, \tau = a \to b\}$

Occurs check: cannot construct the infinite type $c \sim c \to b$

Self-application is impossible in Haskell – it does not have any type.

Lambda expression:

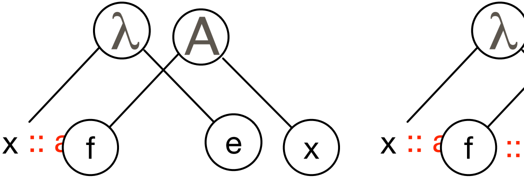
`\ x -> e :: a -> b`



- A function: domain -> range
- Must be of function type
- Type of domain
  = type of variable `x`
- Type of range
  = type of function body e

Function application:

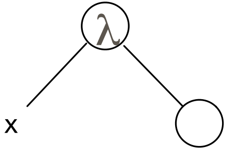
`f x :: b`



- `f` must of function type
- a -> b
- a is the type of the domain of `f`, i.e., the arguement `x`
- b is the type of the range of `f`, i.e., the result of the function application

How does it work?

```
f x = 2 + x
f   = \ x -> 2 + x
f   = \ x -> (+) 2 x
f   :: Int -> Int
```



1. Assign type variables to each node
2. Generate equations (using the two rules from the previous slide)

   ```
   f = Int
   e = f -> d
       Int -> Int -> Int
   d = b -> c
       Int -> Int
   c = Int
   b = Int
   a = b -> c
       Int -> Int
   ```

3. Solve the equations by unification

Try `f g x = (g (g x))`

Extension to other types

$$
\begin{array}{llcl}
\text{(t1)} & E(\Gamma \mid con :: t) & = & \{t = \theta(con)\} \\
\text{(t2)} & E(\Gamma \mid x :: t) & = & \{t = \Gamma(x)\} \\
\text{(t3)} & E(\Gamma \mid f\ g :: t) & = & E(\Gamma \mid g : a) \cup E(\Gamma \mid f :: a \rightarrow t) \\
\text{(t4)} & E(\Gamma \mid \backslash x \rightarrow ex :: t) & = & \{t = a \rightarrow b\} \cup E(\Gamma, x :: a \mid ex :: b) \\
\text{(t5)} & E(\Gamma \mid (ex1, ex2) :: t) & = & \{t = (a, b)\} \\
 & & & \quad \cup\ E(\Gamma \mid ex1 :: a) \cup E(\Gamma \mid ex2 :: b) \\
\text{(t6)} & E(\Gamma \mid x : xs :: t) & = & \{t = [a]\} \cup E(\Gamma \mid x :: a) \cup E(\Gamma \mid xs :: [a]) \\
\text{(t7)} & E(\Gamma \mid [] :: t) & = & \{t = [a]\}
\end{array}
$$

a, b are fresh everywhere

$$
\begin{array}{l}
E(\emptyset \mid [1,2] :: t) = E(\emptyset \mid 1 : 2 : [] :: t) = \\
\quad \{t = [a]\} \cup E(1 :: a) \quad \cup \quad E(2 : [] :: [a]) \ = \\
\quad \{t = [a]\} \cup \{a = \theta(1)\} \cup \{[a] = [b]\} \cup \{2 :: b\} \quad \cup \quad E([] : [b]) = \\
\quad \{t = [a]\} \cup \{a = \theta(1)\} \cup \{[a] = [b]\} \cup \{b = \theta(2)\} \cup \{[b] = [c]\} = \\
\quad \{\ t = [Int], \quad a = Int, \qquad [Int] = [Int], \qquad b = Int, \qquad [Int] = [Int]\ \}
\end{array}
$$