

Please read these notes before class on September 12. The actual homework is due Tuesday September 17th. There will be more problems assigned Friday.

1 Type Inference - A Table Based Method

In class I informally talked about a method to determine the polymorphic type of a Haskell expression (program). We make the method more precise here by presenting two rules and a table based method for deriving types.

We will use lower case Latin letters in a Serif font $\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \dots\}$ to denote polymorphic type variables.

Polymorphic type variables range over types, *i.e.* they stand for any type. In the same way that an variable declared to be of type `int` in a C++ program can take on the value of any `int`, the polymorphic type variable \mathbf{a} can take the value of *any* type. This means, if $f :: \mathbf{a} \rightarrow \mathbf{a}$ then any type can be substituted for \mathbf{a} and the function f has that type. For example, replacing the polymorphic type variable \mathbf{a} by the type `String` we get $f :: \text{String} \rightarrow \text{String}$. Replacing \mathbf{a} by the type $\mathbf{a} \rightarrow \mathbf{a}$ we get $f :: (\mathbf{a} \rightarrow \mathbf{a}) \rightarrow (\mathbf{a} \rightarrow \mathbf{a})$.

Recall, function application associates to the left and so the term $x\ z(y\ z)$ is parenthesized as $((x\ z)(y\ z))$. Also, recall that the function type constructor \rightarrow associates to the right so the type $\mathbf{a} \rightarrow \mathbf{b} \rightarrow \mathbf{c}$ is parenthesized as $(\mathbf{a} \rightarrow (\mathbf{b} \rightarrow \mathbf{c}))$.

1.1 The Method

We start by constructing an initial table that has a columns on the left labeled at the top by the names of the formal parameters and whose first row entries are labeled by different polymorphic type variables. There is also a column on the right where the entry in each row is a labeled Haskell expression. This expression is the body of the function definition with sub-expressions tagged with types (known so far). It may be more readable to fully parenthesize the expression.

For example, consider the function s defined as follows.

$$s\ x\ y\ z = x\ z\ (y\ z)$$

The initial table for this function appears as follows:

x	y	z	expression
\mathbf{a}	\mathbf{b}	\mathbf{c}	$((\overset{\mathbf{a}}{x}\ \overset{\mathbf{c}}{z})\ (\overset{\mathbf{b}}{y}\ \overset{\mathbf{c}}{z}))$

There are two rules for constructing the next row of the table. In the table above, the rule that refines a type to an arrow type (\rightarrow) can be applied.

[Arrow Introduction Rule] If τ is a type expression and α is a polymorphic type variable ($\alpha \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \dots\}$) and there is an application of labeled expressions e_1 and e_2 in the right column having the form $(\overset{\alpha}{e_1}\ \overset{\tau}{e_2})$, then make a new row by copying the last row and replacing all occurrences of α by the type $\tau \rightarrow \beta$ where β is a new variable name not appearing anywhere in the row being copied.

The justification for the arrow introduction rule goes like this: If $e_2 :: \tau$ then the application $(e_1\ e_2)$ is well-typed if and only if e_1 is a function whose domain is τ . We do not know the range (yet) so we just choose a fresh polymorphic variable name and wait to figure it out later. So, we

create a new row from the one above by copying it and changing all occurrences of the type variable α to the type $\tau \rightarrow \beta$ where β is a completely new variable.

There are two places this rule can be applied in the last row of the example. The pattern of the rule $(e_1^\alpha e_2^\tau)$ matches the expression $(\overset{\mathbf{a}}{x} \overset{\mathbf{c}}{z})$ by the following mapping:

$$\{e_1 \mapsto x, \alpha \mapsto \mathbf{a}, e_2 \mapsto z, \tau \mapsto \mathbf{c}\}$$

Also, the pattern $(e_1^\alpha e_2^\tau)$ matches the expression $(\overset{\mathbf{b}}{y} \overset{\mathbf{c}}{z})$ by the mapping:

$$\{e_1 \mapsto y, \alpha \mapsto \mathbf{b}, e_2 \mapsto z, \tau \mapsto \mathbf{c}\}$$

Either application of the rule may be chosen; for no particular reason, we choose the second.

We apply the arrow introduction rule setting $\mathbf{b} = \mathbf{c} \rightarrow \mathbf{d}$. The polymorphic type variable \mathbf{d} is new. We create a new row in the table by copying the last row and replacing all occurrences of \mathbf{b} by the type $(\mathbf{c} \rightarrow \mathbf{d})$. This yields the following table.

x	y	z	expression
\mathbf{a}	\mathbf{b}	\mathbf{c}	$((\overset{\mathbf{a}}{x} \overset{\mathbf{c}}{z}) (\overset{\mathbf{b}}{y} \overset{\mathbf{c}}{z}))$
\mathbf{a}	$\mathbf{c} \rightarrow \mathbf{d}$	\mathbf{c}	$((\overset{\mathbf{a}}{x} \overset{\mathbf{c}}{z}) (\overset{\mathbf{c} \rightarrow \mathbf{d}}{y} \overset{\mathbf{c}}{z}))$

Note that *all* the occurrences of \mathbf{b} have been changed.

Now the arrow introduction rule could be applied again to the application $(x z)$. Instead, we introduce the second rule which is a simplification rule that eliminates arrow types from the right side.

[Arrow Elimination Rule] If τ and τ' are type expressions and there is an labeled application of expression e_1 of type $\tau \rightarrow \tau'$ to expression e_2 of type τ , then create a new row in the table by copying the last row and replacing the labeled application $(\overset{\tau \rightarrow \tau'}{e_1} \overset{\tau}{e_2})$ by $(e_1^{\tau'} e_2)$.

The justification for the rule is simply that $(e_1 e_2)$ must have type τ' if $e_1 :: \tau \rightarrow \tau'$ and $e_2 :: \tau$.

In the running example, the arrow elimination rule has one match in the last row of the example table. Here is the matching:

$$\{e_1 \mapsto y, e_2 \mapsto z, \tau \mapsto \mathbf{c}, \tau' \mapsto \mathbf{d}\}$$

Applying the arrow elimination rule to the last row in the table above yields the following table.

x	y	z	expression
\mathbf{a}	\mathbf{b}	\mathbf{c}	$((\overset{\mathbf{a}}{x} \overset{\mathbf{c}}{z}) (\overset{\mathbf{b}}{y} \overset{\mathbf{c}}{z}))$
\mathbf{a}	$\mathbf{c} \rightarrow \mathbf{d}$	\mathbf{c}	$((\overset{\mathbf{a}}{x} \overset{\mathbf{c}}{z}) (\overset{\mathbf{c} \rightarrow \mathbf{d}}{y} \overset{\mathbf{c}}{z}))$
\mathbf{a}	$\mathbf{c} \rightarrow \mathbf{d}$	\mathbf{c}	$((\overset{\mathbf{a}}{x} \overset{\mathbf{c}}{z}) (\overset{\mathbf{d}}{y} z))$

As the next step, we apply the arrow introduction rule. Since x is applied to an argument of type \mathbf{c} it must be a function of type $\mathbf{c} \rightarrow \mathbf{e}$ where \mathbf{e} is a fresh type variable. Thus we use the arrow introduction rule setting

$$\mathbf{a} = (\mathbf{c} \rightarrow \mathbf{e})$$

To create the next row of the table, copy the last row and replace all occurrences of \mathbf{a} by the type $(\mathbf{c} \rightarrow \mathbf{e})$.

x	y	z	expression
a	b	c	$((\overset{a}{x} \overset{c}{z}) (\overset{b}{y} \overset{c}{z}))$
a	$c \rightarrow d$	c	$((\overset{a}{x} \overset{c}{z}) (\overset{c \rightarrow d}{y} \overset{c}{z}))$
a	$c \rightarrow d$	c	$((\overset{a}{x} \overset{c}{z}) (\overset{d}{y} z))$
$c \rightarrow e$	$c \rightarrow d$	c	$((\overset{c \rightarrow e}{x} \overset{c}{z}) (\overset{d}{y} z))$

Now, because $x :: c \rightarrow e$ and $z :: c$ we know that the application $(x z)$ has type e . We simplify the table by applying the arrow elimination rule as follows:

x	y	z	expression
a	b	c	$((\overset{a}{x} \overset{c}{z}) (\overset{b}{y} \overset{c}{z}))$
a	$c \rightarrow d$	c	$((\overset{a}{x} \overset{c}{z}) (\overset{c \rightarrow d}{y} \overset{c}{z}))$
a	$c \rightarrow d$	c	$((\overset{a}{x} \overset{c}{z}) (\overset{d}{y} z))$
$c \rightarrow e$	$c \rightarrow d$	c	$((\overset{c \rightarrow e}{x} \overset{c}{z}) (\overset{d}{y} z))$
$c \rightarrow e$	$c \rightarrow d$	c	$((\overset{e}{x} \overset{d}{y} z))$

But now, $(x z) :: e$ and $(x z)$ is applied to $(y z) :: d$ so $e = d \rightarrow f$ where f is a new type variable. To create the next line of the table we apply the arrow introduction rule by copying the last line of the table and replacing all occurrences of e by $d \rightarrow f$.

x	y	z	expression
a	b	c	$((\overset{a}{x} \overset{c}{z}) (\overset{b}{y} \overset{c}{z}))$
a	$c \rightarrow d$	c	$((\overset{a}{x} \overset{c}{z}) (\overset{c \rightarrow d}{y} \overset{c}{z}))$
a	$c \rightarrow d$	c	$((\overset{a}{x} \overset{c}{z}) (\overset{d}{y} z))$
$c \rightarrow e$	$c \rightarrow d$	c	$((\overset{c \rightarrow e}{x} \overset{c}{z}) (\overset{d}{y} z))$
$c \rightarrow e$	$c \rightarrow d$	c	$((\overset{e}{x} \overset{d}{y} z))$
$c \rightarrow (d \rightarrow f)$	$c \rightarrow d$	c	$((\overset{d \rightarrow f}{x} \overset{d}{y} z))$

But now we see that $(x z) :: d \rightarrow f$ and it is applied to $(y z) :: d$ so the term $((x z)(y z)) :: f$. We use the arrow elimination rule to create a new row in the table as follows:

x	y	z	expression
a	b	c	$((\overset{a}{x} \overset{c}{z}) (\overset{b}{y} \overset{c}{z}))$
a	$c \rightarrow d$	c	$((\overset{a}{x} \overset{c}{z}) (\overset{c \rightarrow d}{y} \overset{c}{z}))$
a	$c \rightarrow d$	c	$((\overset{a}{x} \overset{c}{z}) (\overset{d}{y} z))$
$c \rightarrow e$	$c \rightarrow d$	c	$((\overset{c \rightarrow e}{x} \overset{c}{z}) (\overset{d}{y} z))$
$c \rightarrow e$	$c \rightarrow d$	c	$((\overset{e}{x} \overset{d}{y} z))$
$c \rightarrow (d \rightarrow f)$	$c \rightarrow d$	c	$((\overset{f}{x} \overset{d}{y} z))$

From this table we know the following:

$$\begin{array}{ll} x & :: \text{c} \rightarrow \text{d} \rightarrow \text{f} \\ y & :: \text{c} \rightarrow \text{d} \\ z & :: \text{c} \\ ((x\ z)(y\ z)) & :: \text{f} \end{array}$$

We can read off the type of s as

$$s :: (\text{c} \rightarrow \text{d} \rightarrow \text{f}) \rightarrow (\text{c} \rightarrow \text{d}) \rightarrow \text{c} \rightarrow \text{f}$$

Since c , d and f are type polymorphic type variables, we can uniformly rename them to make the type more readable (we use the mapping $\{\text{c} \mapsto \text{a}, \text{d} \mapsto \text{b}, \text{f} \mapsto \text{c}\}$.) This gives the following type.

$$s :: (\text{a} \rightarrow \text{b} \rightarrow \text{c}) \rightarrow (\text{a} \rightarrow \text{b}) \rightarrow \text{a} \rightarrow \text{c}$$

Problem 1.1. Use this method to compute the types for the following Haskell functions.

1. $k\ x\ y = x$
2. $compose\ f\ g\ x = f\ (g\ x)$
3. $flip\ f\ x\ y = f\ y\ x$