HW 5 **Prof. Caldwell**
Due: 9 September 2010 **COSC 3015**

Problem 0.1. read Chapters 1 and 2 of Bird.

1 Type Inference - A Table Based Algorithm

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

Bird uses letters from the Greek alphabet $\{\alpha, \beta, \gamma, \delta, \cdots\}$ to denote polymorphic type variables. To aid readability we will use lower case Latin letters in a Serif font $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \cdots\}$.

Polymorphic type variables range over types, 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 a can take the value of *any* type. This means, if $f :: \mathsf{a} \to \mathsf{a}$ then any type can be substituted for a and the function f has that type. For example, replacing the polymorphic type variable a by the type String we get $f :: \mathsf{String} \to \mathsf{String}$. Replacing a by the type $\mathsf{a} \to \mathsf{a}$ we get $f :: (\mathsf{a} \to \mathsf{a}) \to (\mathsf{a} \to \mathsf{a})$.

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 $\to$ associates to the right so the type $a \to b \to c$ is parenthesized as $(a \to (b \to c))$.

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

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 ($\to$) can be applied.

> [**Arrow Introduction Rule**] If τ is a type expression and α is a polymorphic type variable ($\alpha \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \cdots\}$) 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 $\tau \to \beta$ 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 $\tau \to \beta$ 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 $(\overset{\alpha}{e_1}\ \overset{\tau}{e_2})$ matches the expression $(\overset{\mathsf{a}}{x}\ \overset{\mathsf{c}}{z})$ by the following mapping:

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

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

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

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

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

x	y	z	expression
a	b	c	$((\overset{\mathsf{a}}{x}\ \overset{\mathsf{c}}{z})\ (\overset{\mathsf{b}}{y}\ \overset{\mathsf{c}}{z}))$
a	$\mathsf{c} \to \mathsf{d}$	c	$((\overset{\mathsf{a}}{x}\ \overset{\mathsf{c}}{z})\ (\overset{\mathsf{c}\to\mathsf{d}}{y}\ \overset{\mathsf{c}}{z}))$

Note that *all* the occurrences of 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 $\tau \to \tau'$ 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 $(\overset{\tau\to\tau'}{e_1}\ \overset{\tau}{e_2})$ by $\overset{\tau'}{(e_1\ e_2)}$.

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

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

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

x	y	z	expression
a	b	c	$((\overset{\mathsf{a}}{x}\ \overset{\mathsf{c}}{z})\ (\overset{\mathsf{b}}{y}\ \overset{\mathsf{c}}{z}))$
a	$\mathsf{c} \to \mathsf{d}$	c	$((\overset{\mathsf{a}}{x}\ \overset{\mathsf{c}}{z})\ (\overset{\mathsf{c}\to\mathsf{d}}{y}\ \overset{\mathsf{c}}{z}))$
a	$\mathsf{c} \to \mathsf{d}$	c	$((\overset{\mathsf{a}}{x}\ \overset{\mathsf{c}}{z})\ \overset{\mathsf{d}}{(y\ z)})$

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

$$\mathsf{a} = (\mathsf{c} \to \mathsf{e})$$

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

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

Now, because $x :: \mathsf{c} \to \mathsf{e}$ and $z :: \mathsf{c}$ 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	$((\overset{\mathsf{a}}{x}\ \overset{\mathsf{c}}{z})\ (\overset{\mathsf{b}}{y}\ \overset{\mathsf{c}}{z}))$
a	$\mathsf{c} \to \mathsf{d}$	c	$((\overset{\mathsf{a}}{x}\ \overset{\mathsf{c}}{z})\ (\overset{\mathsf{c}\to\mathsf{d}}{y}\ \overset{\mathsf{c}}{z}))$
a	$\mathsf{c} \to \mathsf{d}$	c	$((\overset{\mathsf{a}}{x}\ \overset{\mathsf{c}}{z})\ \overset{\mathsf{d}}{(y\ z)})$
$\mathsf{c} \to \mathsf{e}$	$\mathsf{c} \to \mathsf{d}$	c	$((\overset{\mathsf{c}\to\mathsf{e}}{x}\ \overset{\mathsf{c}}{z})\ \overset{\mathsf{d}}{(y\ z)})$
$\mathsf{c} \to \mathsf{e}$	$\mathsf{c} \to \mathsf{d}$	c	$(\overset{\mathsf{e}}{(x\ z)}\overset{\mathsf{d}}{(y\ z)})$

But now, $(x\ z) :: \mathsf{e}$ and $(x\ z)$ is applied to $(y\ z) :: \mathsf{d}$ so $\mathsf{e} = \mathsf{d} \to \mathsf{f}$ 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 $\mathsf{d} \to \mathsf{f}$.

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

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

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

From this table we know the following:

$$
\begin{array}{rcl}
x & :: & \mathsf{c} \to \mathsf{d} \to \mathsf{f} \\
y & :: & \mathsf{c} \to \mathsf{d} \\
z & :: & \mathsf{c} \\
((x\,z)(y\,z)) & :: & \mathsf{f}
\end{array}
$$

We can read off the type of s as

$$s :: (\mathsf{c} \to \mathsf{d} \to \mathsf{f}) \to (\mathsf{c} \to \mathsf{d}) \to \mathsf{c} \to \mathsf{f}$$

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 $\{\mathsf{c} \mapsto \mathsf{a}, \mathsf{d} \mapsto \mathsf{b}, \mathsf{f} \mapsto \mathsf{c}\}$.) This gives the following type.

$$s :: (\mathsf{a} \to \mathsf{b} \to \mathsf{c}) \to (\mathsf{a} \to \mathsf{b}) \to \mathsf{a} \to \mathsf{c}$$

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

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