Design and Implementation of Programming Languages

Prof. Mehran S. Fallah

Assignment #5

Types in Programming / Type Safety / Type Checking / Type Inference / Polymorphism / Overloading

Done with **love** by

Ali Gholami

with Stu. #

9531504

Department of Computer Engineering and Information Technology

Amirkabir University of Technology

January 2018

The following are problems from the book, **Concepts in Programming Languages, John Mitchell**.

6.1 **ML Types**

Explain the ML type for each of the following declarations:

(a) $fun\ a(x, y) = x + 2 * y;$

(b) $fun\ b(x, y) = x + y/2.0;$

(c) $fun\ c(f) = fn\ y => f(y);$

(d) $fun\ d(f, x) = f\big(f(x)\big);$

(e) $fun\ e(x, y, b) = if\ b(y)\ then\ x\ else\ y;$

Write a short explanation to show that you understand why the function has the type you give.

Answer. (a)

According to the function declaration, the arguments passed to this function are x and y which are considered as $integers$ according to the type inference system of the ML. A point can be the number 2 which is appeared on the left of the multiplication operator. This simply means that this is the integer multiplication and the result will be an $integer$. So the whole function will be resulting into the type of $integer$.

$$val\ a\ =\ fn: int * int \rightarrow int$$

Answer. (b)

From the given floating point number as the second operand of the operator division operator, the result should be a floating point number of course. The addition is also called upon two float numbers also. The float numbers are represented as $real$ in ML.

$$val\ b\ =\ fn: real * real \rightarrow real$$

Answer. (c)

The function declared with the name c, will accept an argument with some type, in which ML will understand that some type should be a closure(function type) since it is being applied on something else. The result will be a function which is ready to grab a new argument and apply the $some\ type$ function on it.

$$val\ c\ =\ fn: (\ 'a \rightarrow\ 'b) \rightarrow\ 'a \rightarrow 'b$$

Answer. (d)

The resulting type and the type being called by the function $f,$ should be the same.

$$val\ d\ =\ fn: (\ 'a \rightarrow\ 'a) *\ 'a \rightarrow 'a$$

Answer. (e)

The function grabs three argument. The first one is any type, the second one is also any other type but the third one will be applied on that any other type and the resulting value is a Boolean for sure. The whole result should be in the same type, so that any type with the second argument will be in the same type. Note that the function return types can be polymorphic, but the return branches should have the same type.

$$val\ e\ =\ fn:\ 'a * 'a * ('a \rightarrow bool) \rightarrow 'a$$

6.2 **Polymorphic Sorting**

This function performing insertion sort on a list takes as arguments a comparison function less and a list l of elements to be sorted. The code compiles and runs correctly:

```
fun sort(less, nil) = nil | sort(less, a :: l) =
  let
    fun insert(a, nil) = a :: nil | insert(a, b :: l) =
        if less(a, b) then a :: (b :: l) else b :: insert(a, l)
  in
    insert(a, sort(less, l))
  end;
```

What is the type of this sort function? Explain briefly, including the type of the subsidiary function insert. You do not have to run the ML algorithm on this code; just explain why an ordinary ML programmer would expect the code to have this type.

Answer

The above function sort accepts an argument of type function which is a Boolean one, since it is being evaluated inside a $if - then - else$ command. The second argument is a list for sure but the resulting final type of the function sort, will be the same as the type of the argument being passed to it, a list of the same type. That is because it is being called by a part of that list, inside the insert function and the returning results of the insert function are the list elements concatenated together in an incremental order.

$$val\ sort\ =\ fn:\ ('a\ *\ 'a \rightarrow bool) *\ 'alist \rightarrow 'a\ list$$

6.4 **Polymorphic Fixed Point**

A fixed point of a function f is some value x such that $x = f(x)$. There is a connection between recursion and fixed points that is illustrated by this ML definition of the factorial function $factorial: int \rightarrow int$.

```
fun Y f x = f (Y f) x;
fun F f x = if x = 0 then 1 else x * f(x - 1);
val factorial = Y F;
```

The first function, Y, is a fixed-point operator. The second function, F, is a function on functions whose fixed point is factorial. Both of these are curried functions; using the ML syntax we could also write the function F as

$$fun\ F(f) = \ fn\ x \rightarrow \ if\ x = 0\ then\ 1\ else\ x * f(x - 1)$$

This F is a function that, when applied to argument f, returns a function that, when applied to argument x, has the value given by the expression $if\ x = 0\ then\ 1\ else\ x * f(x - 1)$.

(a) What type will the ML compiler deduce for F?
(b) What type will the ML compiler deduce for Y?

Answer. (a)

Before getting into the $ML\ REPL$ output, we can understand from the definition of F, that the argument x is an integer for sure. On the other hand, the result of the function is an integer in one branch, so it has to be the same on the $other$ branch as well. Considering this, we can understand that $x * f(x - 1)$ is an integer. Since x is an $integer$, the multiplication will be called upon two $integer$ arguments. Finally, the return value of f has to be an integer as well to match the result. So, this is wise that we get the following output from the $ML\ REPL$.

$$val\ F\ = \ fn : (int \rightarrow \ int) \rightarrow \ int \rightarrow \ int$$

Answer. (b)

Since things are getting a little bit complicated, we'll find the type for the below definition first.

$$fun\ Y\ f(x)\ = \ f(Y\ f);$$

It's obvious that the function Y, accepts a function with an argument of type $'a$ and the return type of $'b$. So, the input for the function Y is $('a \rightarrow 'b)$. The return value for the function Y, is the application of the function f with type $('a \rightarrow 'b)$ on the function application of Y on f again, which gives us the final below results.

$$(('a \rightarrow 'b)\ \rightarrow 'b) \rightarrow 'a \rightarrow 'b$$

Inferring the same strategy for the function given in the problem, we can understand that the type deduction result will be as following.

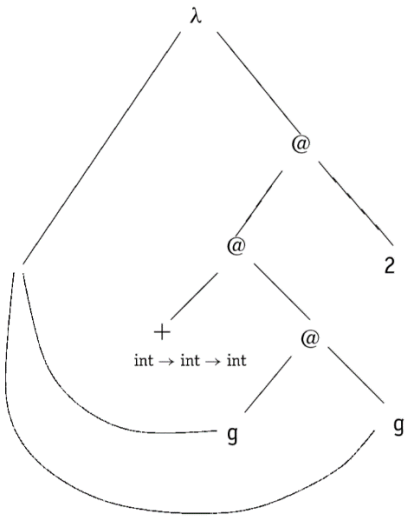
$$val\ Y\ = \ fn : \Big(('a \rightarrow \ 'b) \rightarrow \ 'a \rightarrow \ 'b\Big) \rightarrow \ 'a \rightarrow 'b$$

6.6 **Parse Graph**

Use the following parse graph to follow the steps of the ML type-inference algorithm on the function declaration

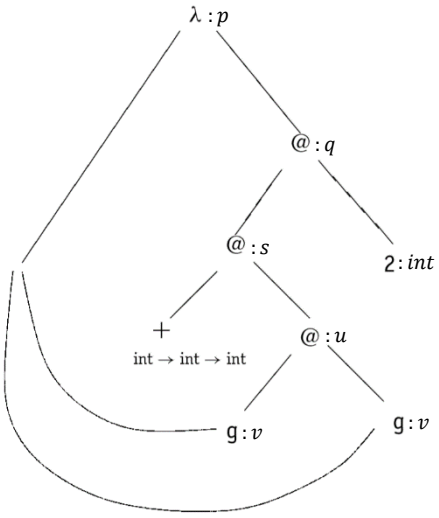
$$fun\ f(g) = g(g) + 2;$$

What is the output of the type checker?



Answer

Applying the type inference algorithm on the above parse tree, we can derive the following constraint and decoration on the parse tree.



Now the resulting constraints are

$p = (v * v) \rightarrow q$ $\quad$ $int \rightarrow int \rightarrow int = u \rightarrow s$

$s = int \rightarrow q$ $\quad$ $v = v \rightarrow u$

Using $unification$, we'll get the following result.

$$u = int, q = int$$

$$s = int \rightarrow int$$

Note that the inferred constraint $v = v \rightarrow u$ won't let the type inference mechanism to complete and we'll get the following error.

$stdIn: 1.13 - 26.4\ Error:\ operator\ is\ not\ a\ function\ [circularity]$

$operator: 'Z$

$in\ expression:$

$g\ g$

6.7 Type Inference and Bugs

What is the type of the following ML function?

$$fun\ append(nil, l)\ =\ l$$

$$|\ append(x :: l, m)\ =\ append(l, m);$$

Write one or two sentences to explain succinctly and informally why append has the type you give. This function is intended to append one list onto another. However, it has a bug. How might knowing the type of this function help the programmer to find the bug?

Answer

While appending a list to another, the resulting thing has to be a list also. Using the above append declaration, appending a null list to something else, will result in that thing only. Let the first element of the list to be appended is x and the rest of the list is called l. While appending that list to a list called m, we have to use the prepend operator :: to concatenate the first element to the rest of the appending things. So the true append function will be as the following.

$$fun\ append(nil, l)\ =\ l$$

$$|\ append(x :: l, m)\ =\ x :: append(l, m);$$

In which results in the following type.

$$val\ append\ =\ fn : 'a\ list\ *\ 'a\ list\ ->\ 'a\ list$$

6.8 **Type Inference and Debugging**

The reduce function takes a binary operation, in the form of a function f, and a list, and produces the result of combining all elements in the list by using the binary operation. For example:

$$reduce\ plus[1,2,3] = 1 + 2 + 3 = 6$$

if plus is defined by

$$fun\ plus\ (x, y: int) = x + y$$

A friend of yours is trying to learn ML and tries to write a reduce function. Here is his incorrect definition:

```
fun reduce (f, x) = x
| reduce (f, (x :: y)) = f (x, reduce (f, y));
```

He tells you that he does not know what to return for an empty list, but this should work for a nonempty list: If the list has one element, then the first clause returns it. If the list has more than one element, then the second clause of the definition uses the function f. This sounds like a reasonable explanation, but the type checker gives you the following output:

$$val\ reduce = fn: \left(\left(('a *' a\ list) \rightarrow' a\ list\right) *' a\ list\right) \rightarrow 'a\ list$$

How can you explain your friend that his code is wrong?

Answer

I'll try to explain him using the function declaration itself. According to the function declaration, the second argument of the function in the second pattern of the function reduce, has to be a list to be separated using the operator **::** and while reaching the final element, this second argument will be null. So the base case for the above recursive function, has to be the null case. On the other hand, using the type inference results, we can understand that the returning value of the reduce function does not have to be the same $'a\ list$ like the list which is passed to it! Because a function is being applied on every single element of the list and the resulting value be different for sure. So, the application of f on the x, has to be changed to the application of f on the $null$ in the base case. Using this approach, the $ML\ REPL$ will give use the following results.

$$val\ reduce\ =\ fn\ :\ ('a\ *\ 'b\ list\ ->\ 'b\ list)\ *\ 'a\ list\ ->\ 'b\ list$$

Note that the resulting list has to be in a different type since function are meant to act like that.

6.9 **Polymorphism in C**

In the following C min function, the type void is used in the types of two arguments. However, the function makes sense and can be applied to a list of arguments in which

void has been replaced with another type. In other words, although the C type of this function is not polymorphic, the function could be given a polymorphic type if C had a polymorphic type system. Using ML notation for types, write a type for this min function that captures the way that min could be meaningfully applied to arguments of various types. Explain why you believe the function has the type you have written.

```
int min(void *a[], /* a is an array of pointers to data of unknown type */
        int n, /* n is the length of the array */
        int (*less)(void *, void *) /* parameter less is a pointer to
function */)
/* that is used to compare array elements */
{
    int i;
    int m;
    m = 0;
    for (i = 1; i < n; i++)
        if (less(a[i], a[m]))
            m = i;
    return (m);
}
```

Answer

There are three major arguments being passed to the function min. The first argument is a list of some type which we call it an $'a\ list$. The second argument has the type of the integer which will be int in the ML language. The third argument is function which is getting two parameters of unknown types which we'll call both of them as $'a$ since they are the elements of that array we just give it a name. The returning value for the function $less$ is an integer. The returning value for the whole min function is an integer also. So, I'll expect the following thing to be the type of the above min function.

$$val\ min\ =\ fn : 'a\ list\ *\ int\ *\ ('a\ list\ *\ 'a\ list \rightarrow\ int) \rightarrow\ int$$

6.11 **Dynamic Typing in ML**

Many programmers believe that a run-time typed programming language like Lisp or Scheme is more expressive than a compile-time typed language like ML, as there is no type system to $get\ in\ your\ way$. Although there are some situations in which the flexibility if Lisp or Scheme is a tremendous advantage, we can also make the opposite argument. Specifically, ML is more expressive than Lisp or Scheme because we can define an ML data type for Lisp or Scheme expressions.

Here is a type declaration for pure historical Lisp

$$
\begin{aligned}
&datatype\ LISP\ =\ Nil\\
&\quad |\ Symbol\ of\ string\\
&\quad |\ Number\ of\ int\\
&\quad |\ Cons\ of\ LISP\ *\ LISP\\
&\quad |\ Function\ of\ (LISP \rightarrow\ LISP)
\end{aligned}
$$

(a) Write an ML declaration for the Lisp function $atom$ that tests whether its argument is an atom. (Everything except a cons cell is an atom – The word atom comes from the Greek word $atomos$, meaning indivisible. In Lisp, symbols, numbers, nil, and functions cannot be divided into smaller pieces, so they are considered to be atoms.) Your function should have type $LISP \rightarrow LISP$, returning atoms Symbol($"T"$) or nil.

Answer. (a)

Using pattern matching style, we'll get:

$$
\begin{aligned}
&fun\ isAtom(Symbol(x))\ =\ Symbol("T")|\\
&isAtom(Nil)\ =\ Symbol("T")\ |\\
&\ isAtom(Function(x))\ =\ Symbol("T")\ |\\
&isAtom(Number(x))\ =\ Symbol("T")\ |\\
&\ isAtom\big(Cons(x,y)\big)\ =\ Nil;
\end{aligned}
$$

(b) Write an ML declaration for the Lisp function $islist$ that tests whether its argument is a proper list. A proper list is either nil or a cons cell whose cdr is a proper list. Note that not all list like structures built from cons cells are proper lists. For instance, $(cons\ \big(symbol(\text{A}),\ Symbol(\text{B})\big))$ is not a proper list and its $islist$ should evaluate to Nil. On the other hand, $(cons\ \Big(Symbol(\text{A}), \big(cons\ (Symbol(\text{B}), Nil)\big)\Big))$ is a proper list, and so your function should evaluate to Symbol($"T"$). Your function should have type $LISP \rightarrow LISP$ as before.

Answer. (b)

$$fun\ islist(Function(x)) = Nil \mid$$

$$islist(Number(x)) = Nil \mid$$

$$islist(Symbol(x)) = Nil \mid$$

$$islist(Nil) = Symbol("T") \mid$$

$$islist(Cons(x, y)) = islist(y);$$

(c) Write an ML declaration for Lisp car function and explain briefly. The function should have type $LISP \rightarrow LISP$.

Answer. (c)

$$fun\ car(Cons(x, y)) = x$$

(d) Write Lisp expression $(lambda\ (x)\ (cons\ x\ 'A))$ as an ML expression of type $LISP \rightarrow LISP$. Note that $'A$ means something completely different in Lisp and ML. The $'A$ here is part of a Lisp expression, not an ML expression. Explain briefly.

Answer. (d)

The following ML function is used with a variable named x, which creates a Cons in the Lisp with that x variable and the string symbol 'A.

$$fn\ x => Cons(x, Symbol("'A"));$$