605.629: Programming Languages

Assignment 5

Sabbir Ahmed

1. [50 pts, ML types]

Determine the ML type for each of the following declarations. Feel free to type the declarations into an ML interpreter to determine the type, but make sure to explain in a couple of sentences why the type is what it is.

Answer

a. `fun a(x,y) = x + y/2.0;`

```
val a = fn : real * real -> real
```

Since `val y/2.0` returns a real, then `x + y/2.0` is also a real.

---

b. `fun b(f) = fn x => f(x)+1;`

```
val b = fn : ('a -> int) -> 'a -> int
```

Since `f` is a function with an unknown type, it is substituted with `'a` and `f(x) + 1` can be inferred as `'a + int` .

---

c. `fun c(w, x, y, z) = if w(x) then x(y) else z;`

`val c = fn : (('a -> 'b) -> bool) * ('a -> 'b) * 'a * 'b -> 'b`
Since `x` is a function with an unknown type, it is substituted with `'a -> 'b`.
Since `w` is a function with an unknown type, it is substituted with `'a -> 'b`. The function is inside a conditional, so it returns a `bool` after taking `x` as a parameter.This can be substituted with `('a -> 'b) -> bool`.
The other parameters are treated as arguments of unknown types and are substituted with `'a` and `'b` respectively, with the entire function return an unknown type.

---

d. `fun addToList(nil, x) = x`
`| addToList(x, h::l) = h::addToList(x,l);`

`val addToList = fn : 'a list * 'b list -> 'b list`
Both of the parameters of the function are lists of unknown types, and are therefore substituted with `'a list` and `'b list`. This function joins the 2 lists and returns it as a list of unknown type `'b list`.

---

e. The `addToList` function above has a bug. Can the type inferred for this function help the programmer notice that the function is implemented incorrectly? How?

Yes, since the intention of the function was to add to a list of unknown type, the function return type should have at least matched with the input parameters. When the return type was inferred to be of a different type, it was an indication that there may be a bug. The online compiler returned a warning for "match nonexhaustive" which suggested that it is possible to break the function with certain matches. This means that calling `addToList([1,2,3], [3])` would result in an error since the first argument expected an empty list. Swapping the arguments to `fun addToList(x, nil) = x | addToList(x, h::l) = h::addToList(x,l);` would fix this issue and the types can be inferred as: `val addToList = fn : 'a list * 'a list -> 'a list`.

[50 pts, parameter passing]

Consider the following program written in Algol-like pseudocode:

```
begin
    integer i;
    procedure foo(integer x, integer y);
        begin
            x := x+1;
            y := x+1;
            x := y;
            i := i+1;
        end
    i := 1;
    foo(i,i);
    print i;
end
```

What would this program print under each of the following parameter passing mechanisms?

Answer

a. Pass-by-value

The program would compute and store the following expressions if the parameters were passed-by-value:

```
i = 1
foo(i, i)
    x = i;      // x = 1
    y = i;      // y = 1
    x = x + 1;  // x = 2
    y = x + 1;  // y = 3
    x = y;      // x = 3
    i = i + 1;  // i = 2
    return i;
```

Therefore, the program would print: 2

b. Pass-by-reference

The program would compute and store the following expressions if the parameters were passed-by-reference:

```
i = 1
foo(&i, &i)
    x = i;      // x = 1
    y = i;      // y = 1
    x = x + 1;  // x = 2, i = 2
    y = x + 1;  // y = 3, i = 3
    x = y;      // x = 3, i = 3
    i = i + 1;  // i = 4
    return i;
```

Therefore, the program would print: 4

---

c. Pass-by-value-result

The program would compute and store the following expressions if the parameters were passed-by-value-result:

```
i = 1
j = i
foo(&j, &j)
    x = j;      // x = 1
    y = j;      // y = 1
    x = x + 1;  // x = 2, j = 2
    y = x + 1;  // y = 3, j = 3
    x = y;      // x = 3, j = 3
    i = i + 1;  // i = 2
    i = j;      // i = 3
    return i;
```

Therefore, the program would print: 3
