Programming Languages Hw5

Arya Banaeizadeh 9431029

John Mitchell, Concepts in Programming Languages

6.1 ML Types

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

(int * int) -> int

This function takes to arguments [(‘a * ‘a)] and then returns a term which has + and * operations involving x and y and an integer number (concluding + and * are of type int -> (int -> int) and thus ‘a cannot be anything but int.

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

(real * real) -> real

This function takes to arguments [(‘a * ‘a)] and then returns a term which has + and / operations involving x and y and a real number (concluding + and * are of type real -> (real -> real) and thus ‘a cannot be anything but int.

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

('a -> 'b) -> 'a -> 'b

This is a function which takes an argument and then takes another argument and applies the former to the latter thus f is a function (‘a -> ‘b) and y is applicable to f so y is of type ‘a and f(y) is of type ‘b.

(d) fun d(f,x) = f(f(x)));

(‘a -> ‘a) * ‘a -> ‘a

The function takes two argument and f is a function judging by the return statement. Thus (‘a -> ‘b) and because we have f(x) we’ll deduce that x type is ‘a if we apply the f(x) we’ll get ‘b as a result and ‘b is again applied to f so ‘b should be equal to ‘a and thus the return statement is of type ‘a .

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

6.2 Polymorphic Sorting

Knowing that the code will run correctly and it’s a sort function with less and list argument it’s evident that the type is (bool * a’ list) -> a’ list.

But if we want to be more accurate we’ll examine the code

1) sort(less, nil) = nil type is (bool * a’ list) -> a’ list
2) 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))
```

inset returns a list of some type (say (‘b list)) we’ll just have to prove that the ‘b list and ‘a list are of one type. Insert type is (‘a * (a’ list)) -> (‘a list) because if it takes a and l as arguments they are both present in the return type. Sort(less,a :: l) = insert(a, sort(less, l)) thus insert type is equal to sort type.

6.4 Polymorphic Fixed Point

a)

(int -> int) -> int -> int

b)

(('a -> 'b) -> 'a -> 'b) -> 'a -> 'b

6.6 Parse Graph

(@s are indexed from the top)

f(g) : int

@1 : int

@2 : int -> int (1)

+ : int -> int -> int (2)

(1),(2) : g(g) @3 : int

g would be ‘b -> int but this is impossible since g is the argument of itself that takes itself as an argument and return an int because ‘b being equal to ‘b -> int is logically impossible therefore it the program will get stuck.

6.7 Type Inference and Bugs

the type would be: (‘a list * ‘a) -> ‘b list

the abstraction of the function does not raise an error but since in the second pattern x is not used in the return term the type will become different. This’ll come to programmer’s aid when the programmer realizes that append return type shouldn’t be different than the original list type.

6.8 Type Inference and Debugging

The 1st pattern is inaccurate, the type would be (‘a -> ‘a) * ‘b -> ‘b and ‘b can be anything (even a list) on the other hand reduce(f, (x :: y)) is defined as f(x, reduce(f, y)) which has the type: ((‘a*’a list) -> ‘a list)*…). so there is a match redundancy in this function.

6.9 Polymorphism in C

(‘a list * int * (‘a * ‘a) -> int) -> int

int should be changed to bool if the function is implemented in ml. Also int n wouldn’t be necessary in ml. Resulting the type : (‘a list * (‘a * ‘a) -> bool) -> int

If C was polymorphic we would assume that void*[] is an abstract list type and thus the type in ml notation would be a'list. n type is int. and less type is of ('a * 'a) -> int. min return type is int and therefore we have ('a list * int * ('a * 'a) -> int) -> int

6.11 Dynamic Typing in ML

a)

```
fun atom (Nil) = Symbol("T")

| atom(Number(x)) = Symbol("T")

| atom(Symbol(s)) = Symbol("T")

| atom(Function(f)) = Symbol("T")

| atom(Cons(d)) = Nil;
```

b)

```
fun islist (Nil) = Symbol("T")

| islist(Number(x)) = Nil

| islist(Symbol(s)) = Nil

| islist(Function(f)) = Nil

| islist(Cons(a,b)) = islist(b);
```

c)

```
car(cons(a,b)) = a
```

cons type is LISP * LISP -> LISP. Cons(a,b) type is LISP and a is of type LISP this car is of type LISP -> LISP

d)

```
fun lambda(x) = Cons(x,Symbol("A"))
```

'A in lisp refers to quote A which is the same thing as the Symbol("A").