DCC888 – Constraint Based Analysis

1. Consider the program below, which has been implemented in Python. Finish the implementation of the function `create_binding()`, in such a way that the call `obj.get()` at line 19 invoke each of the functions defined at lines 4, 10 and 15 of the program.

```
1  class A:
2    def __init__(self):
3      self.x = 0
4    def get(self):
5      return self.x
6
7  class B:
8    def __init__(self, start):
9      self.x = start
10   def get(self):
11     return self.x
12   def inc(self):
13     self.x += 1
14
15 def foo(obj):
16   return obj.x
17
18 def bar(obj):
19   return obj.get()
20
21 create_binding()
```

Line 4:
def create_binding():

Line 15:
def create_binding():

Line 10:
def create_binding():

2. Consider the two programs below, which have been written in SML/NJ, a functional programming language:

   - ```
     let f = fn x => fn y => x y
       in let g = fn x => x + 1
         in f g 1
     ```
   - ```
     let f = fn x => x 1
     in let g = fn y => y + 2
        in let h = fn z => z + 3
            in (f g) + (f h)
     ```

   (a) Add labels to these programs, in such a way that each label describes a syntactic block that can be bound to a function.

   (b) Find a solution to the control flow analysis problem for these programs. Your solution must associate labels or variable names to sets of functions that can be bound to them.

3. A solution to the control flow analysis satisfies a function application such as $(t_1^{l_1} t_2^{l_2})^l$ if the following rules hold true:

$$
\begin{aligned}
&(C, R) \models t_1^{l_1} \ \wedge \ (C, R) \models t_2^{l_2} \ \wedge \\
&\qquad\qquad (\forall (\mathit{fn}\ x \Rightarrow t_0^{l_0}) \in C(l_1) : (C, R) \models t_0^{l_0} \ \wedge \ C(l_2) \subseteq R(x) \ \wedge \ C(l_0) \subseteq C(l))
\end{aligned}
$$

However, these rules do not take into consideration the ordering in which the expressions are evaluated. For instance, in a language that admits parameter passing by value, an expression such as $E_1 E_2$ requires first the evaluation of E_1, before we move on to the evaluation of E_2. Yet, if E_1 does not produce any closure, then we do not need to evaluate E_2. Is it possible to modify the rule above, to evaluate the operand l_2 only if the operator l_1 produces a closure? How could this be done?

4. Prove that the algorithm to solve constraints based on the constraint graph terminates. Use an argument based on *saturation*. The algorithm uses a few data-structures to record information. This structures keep track of a finite amount of information during the execution of the algorithm. Once the information is inserted in the data-structure, this information will never be removed from it.

5. In an object oriented programming language, it is desirable to know what are the implementations of a method `m`, once we find a call such as `o.m(a, b, c)`. A way to solve this problem relies on a control flow analysis. However, there are other ways – much faster, yet less precise – to find the targets of method calls. In this question, you must discuss different techniques to find out the different implementations of `m`, without resorting to a full-fledged control flow analysis.

6. This question refers to the program below, which was written in Python.

```
1  class A:
2    def __init__(self):
3      self.x = 0
4
5    def get(self, term):
6      return self.x + term
7
8
9  class B:
10   def __init__(self, start):
11     self.x = start
12   def get(self, factor):
13
14     return self.x * factor
15
16 class C:
17   def __init__(self):
18
19     self.x = -1
20   def get(self):
21     return self.x
```

```
22 def foo1(obj):
23   return obj.x
24
25
26 def foo2(obj, factor):
27   return obj.x * factor
28
29
30 def bar(obj1, obj2):
31   return obj1.get() + obj2.get(2.0)
32
33 a = A()
34
35 if int(raw_input("Enter: ")) > 1:
36   a = B(1)
37   B.get = foo1
38 c = C()
39
40 C.get = foo2
41 bar(a, c)
```

(a) What are the possible bindings for `obj1`, the first argument of `bar`, at line 41?

(b) What are the possible bindings for `obj2`, the second argument of `bar`, at line 41?

(c) An exception can happen in this program, due to an incompatible method invocation. How can this event take place? You may consider to run the program, to observe the actual exception happening.

(d) Can the constraint based analysis be adapted to flag the possibility of such exception happening? In this case, the constraint based analysis is a bit similar to the type system of a statically typed language. Can you explain this similarity?