EECS 483 — Review Set 4

March 16, 2018

1 Runtime Organization

1. In class we discussed partitioning memory into *code*, *static data*, *stack* and *heap* areas. In at most three paragraphs, support or refute the claim that any program that could be compiled for such a four-way partition could also be compiled for a model in which memory is divided up into *code*, *static data* and *heap* areas only. Your argument should consider the compilation of global variables, local variables, function calls, and dynamically-allocated objects.

2. Consider the addition of Java- or C++-style *exception handling* to Cool. For each of the following sorts of variables, separately indicate the effect of raising an exception on its *lifetime* and also its *scope*.

   (a) local variables

   (b) function arguments

   (c) dynamically-allocated objects

2 Stack Discipline

1. Consider the following *incorrect* code generation rule for a function definition:

```
cgen(def f(x1, ..., xn) = e) =
        f_entry:
                push ra
                mov fp <- sp
                cgen(e)
                ra <- top
                add sp <- sp z
                return
```

   Give a simple program that will behave incorrectly when compiled using the above rule.

2. Consider the following two alternate definitions for the NumTemps function, NT:

   (a) $NT(e_1 + e_2) = \max(NT(e_1), 2 + NT(e_2))$

   (b) $NT(e_1 + e_2) = \max(1 + NT(e_1), NT(e_2))$

   For each alternate definition, indicate in at most one paragraph what would happen if that definition were used for code generation.

3. In at most one paragraph, explain how code generation would change if function parameters were pushed on the stack in the *opposite* order of what we described in class.

4. Consider the following function prologue:

```
Main.test??:  mov fp <- sp
              pop r0
              li r2 <- 6
              sub sp <- sp r2
              push ra
              push fp
              push r0
```

   Ignoring optimizations, to which of the following function definition(s) could it correspond?

   (a)
```
test34(x : Int, y : Int, z : Int) : Int {
  let a : Int, b : Int, c : Int, d : Int in
    x + y + z + a + b + c + d
} ;
```

   (b)
```
test24(x : Int, y : Int) : Int {
  let a : Int, b : Int, c : Int, d : Int in
    x + y + a + b + c + d
} ;
```

   (c)
```
test22(x : Int, y : Int) : Int {
  let a : Int, b : Int in
    x + y + a + b
} ;
```

   (d)
```
test31(x : Int, y : Int, z : Int) : Int {
  let a : Int in
    x + y + z + a
} ;
```

3 Typechecking Rules

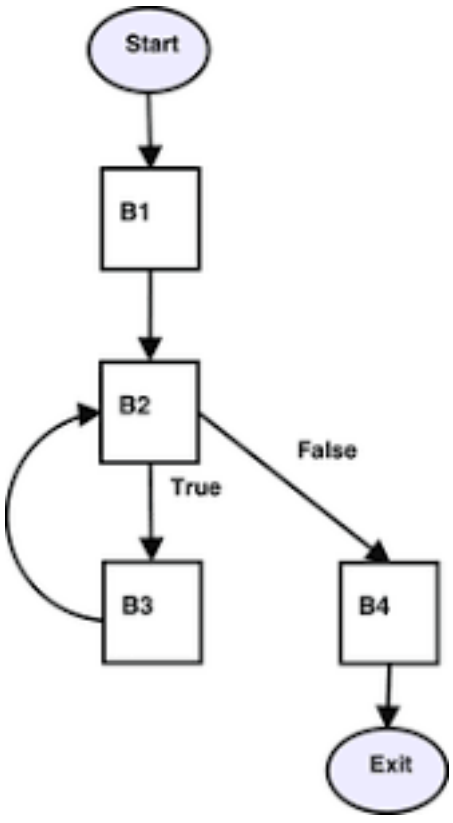
1. Suppose we change the behavior of *while* so that it returns the number of times the loop body is executed. Write the typing rule for this alternate *while* expression.

2. Consider the following alternate typing rule for *if*. Indicate whether it is unsound (i.e., allows unsafe programs to proceed to code generation) or incomplete (i.e., mistakenly prevents safe programs from being processed), both, or neither. If it is incomplete and/or unsound, give a program that demonstrates as much.

$$\frac{\begin{array}{c} O \vdash e_0 : Bool \\ O \vdash e_1 : T \\ O \vdash e_2 : T \end{array}}{O \vdash \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \text{lub}(Bool, T)}$$

4 Dataflow Analysis

1. Consider the following control-flow graph:



Consider carrying out constant propagation on the graph with respect to a variable x. You must map each of the following statements to exactly one of the basic blocks ($B1 \ldots B4$):

(a) $x \leftarrow 1$

(b) $x \leftarrow x + 1$

(c) $x \leftarrow x - 2$

(d) $x \leftarrow 3$

... such that the resulting dataflow analysis will only have $x = \top$ (i.e., "unknown") on one edge (the edge leaving the *Start* node). (Do not worry about the *if* conditional.)

2. Consider *live variable analysis* and *constant propagation*. Construct the smallest control-flow graph you can (in terms of number of nodes) such that:

   - Each node contains only a single assignment of the form $x \leftarrow 1$ (or 2, or 3, etc.)
   - The start node contains $x \leftarrow 0$
   - There is an edge in the control flow graph such that liveness concludes x is dead on that edge but constant propagation concludes $x = \top$ on that edge.
   - The control flow graph is *acyclic*.