# Assignment 3: Middle End

15-411: Compiler Design
Rob Simmons, Will Crichton, Grant Della Silva, Matt Bryant, Anshu Bansal

Due Thursday, October 15, 2015 (11:00pm)

**Reminder:** Assignments are individual assignments, not done in pairs. The work must be all your own. Hand in your solutions as a PDF file on Autolab. Please read the late policy for written assignments on the course web page.

## Problem 1: Stuck in the Middle (60 points)

We generate a *Collatz sequence* $c_i$, starting from some positive integer $n$, with the following mathematical definition:

$$a_0 = n$$

$$a_{i+1} = \begin{cases} a_i/2 & \text{if } a_i \text{ is even} \\ 3a_i + 1 & \text{otherwise} \end{cases}$$

The *stopping time* of a Collatz sequence is the smallest index $i$ such that $a_i = 1$. It is currently not known if every Collatz sequence reaches 1 (and thereby stops). The following C0 function is intended to compute the *maximum number* in the Collatz sequence for $n$ before it stops.

```
int collatz(int n)
//@requires n >= 1;
{
  int r = 0;
  while (n > 1) {
    if (n > r) r = n;
    if (n % 2 == 0)
      n = n / 2;
    else
      n = 3*n + 1;
  }
  return r;
}
```

(a) Compile this program to an intermediate representation with labels, infinite temps, and arbitrary nested expressions, but only gotos (`goto` $Lab$) and conditionals (`if` $e1$ `?` $e2$ `then` $Lab1$ `else` $Lab2$). You do not need to follow a specific code generation algorithm, but the correspondence of the source to the intermediate representation should be direct and easy to discern. The code should be arranged in basic blocks that begin with a label and end with a goto, conditional, or return.

(b) Compile your intermediate representation to abstract three-address assembly code with infinite temps but without nested expressions. Your code should still be arranged into basic blocks.

(c) Show the control flow graph of the program in (b) pictorially, carefully encapsulating each basic block. Label each basic block with the label from the abstract assembly code.

(d) Convert the abstract assembly program into SSA with parameterized labels.

(e) Rearrange the SSA program to use Φ-functions.

(f) Transform your SSA code into minimal SSA, removing unnecessary Φ-functions.

(g) Transform your code out of SSA with Φ-functions and back into SSA with parameterized labels.

(h) Apply the de-SSA transformation to obtain a program where labels are no longer parameterized.

(i) Modify this code to use extended basic blocks, where every label is the target of more than one jump. (Aside: you could have transformed your code into extended basic blocks after step (a) or after step (b). How would this have influenced later stages?)

(j) If we are not interested in the maximum value in the sequence, but just a confirmation that it stopped, we can transform the program by replacing the last line with `return 0`. Apply *neededness analysis* from lecture and show which code will be flagged as *not needed*. Clearly indicate where temps are defined and used, and which temps are needed on which lines. You do not need to show intermediate steps, only your final answer, but it is easier to assign partial credit if you show your work.

(k) Even if we assume that integers are infinite-precision, the original code for `collatz` has a bug. Describe and fix this bug in the source.