# Assignment 2
# SSA, Lexing and Parsing

15-411: Compiler Design
Frank Pfenning
Flávio Cruz, Maxime Serrano, Rokhini Prabhu, Tae Gyun Kim

Due Thursday, September 25, 2014 (11:59pm)

**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 (30 points)

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

$$\begin{array}{lcl} a_0 & = & n \\ a_{i+1} & = & \left\{ \begin{array}{ll} a_i/2 & \text{if } a_i \text{ is even} \\ 3a_i + 1 & \text{if } a_i \text{ is odd} \end{array} \right. \end{array}$$

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 eventually 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 abstract assembly code as used in Lecture 6. You do not need to follow a specific instruction selection algorithm, but the correspondence of the source to abstract assembly code should be direct and easy to discern.

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

(c) Convert the program into SSA, using the form with parameterized labels. Feel free to either use the control flow graph or the abstract assembly form to show your result. You do not need to show intermediate steps, only your final form, but it is easier to assign partial credit if you show your work. For full credit, your result should have the minimal number of parameters for each label.

(d) Apply the de-SSA transformation to obtain a program where labels are no longer parameterized. Again, you may show your result in the form of a control flow graph or in textual form.

(e) 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;
```

If we apply *neededness analysis* from Lecture 5 which code will be flagged as *not needed*?

(f) [Extra credit] The original code for `collatz` has a bug. Describe and fix this bug in the source.

## Problem 2 (30 points)

The following table defines the tokens for a simple language with $\Sigma = \{[a{-}z], [0{-}9], ., \oplus, (, )\}$.

| token | regular expression |
|---|---|
| $\langle num \rangle$ | ≡ 0 \| [1-9][0-9]* |
| $\langle lam \rangle$ | ≡ lam |
| $\langle ident \rangle$ | ≡ [a-z][a-z0-9]* |
| $\langle dot \rangle$ | ≡ . |
| $\langle lp \rangle$ | ≡ ( |
| $\langle rp \rangle$ | ≡ ) |
| $\langle binop \rangle$ | ≡ $\oplus$ |

(a) Construct a DFA that accepts tokens from this language using Brzozowski derivatives. Show the calculation of the derivatives, combining the treatment of character ranges when all characters in the range have the same derivatives.

You may show the resulting DFA either graphically, or by explicitly writing down the states and transitions. You may omit transitions to the (non-accepting) state with regular expression 0, where all further derivatives remain 0.

In either notation, the connection to the Brzozowski derivates should be clear. Also, note for each final state which token was recognized. See Lecture 7 for an example notation.

Here is the grammar for the language whose tokens were defined in part a, using x for identifiers $\langle ident \rangle$ and n for numbers $\langle num \rangle$.

$$\begin{array}{llll}
\gamma_1 & : & \langle \mathrm{E} \rangle & \rightarrow & \mathrm{n} \\
\gamma_2 & : & \langle \mathrm{E} \rangle & \rightarrow & \mathrm{x} \\
\gamma_3 & : & \langle \mathrm{E} \rangle & \rightarrow & \mathrm{lam}\ \mathrm{x}\ .\ \langle \mathrm{E} \rangle \\
\gamma_4 & : & \langle \mathrm{E} \rangle & \rightarrow & \langle \mathrm{E} \rangle\ \langle \mathrm{E} \rangle \\
\gamma_5 & : & \langle \mathrm{E} \rangle & \rightarrow & (\ \langle \mathrm{E} \rangle\ ) \\
\gamma_6 & : & \langle \mathrm{E} \rangle & \rightarrow & \langle \mathrm{E} \rangle \oplus \langle \mathrm{E} \rangle
\end{array}$$

(b) This grammar is ambiguous. Give two examples of ambiguous expressions along with two ways in which each can be parsed. The two examples should illustrate different sources of ambiguity in the grammar.

(c) Some additional information is needed to make the grammar unambiguous. Make some design decisions that resolve the ambiguity and state your decisions clearly, in words. Use examples to illustrate your choices.

(d) Rewrite the grammar so that it accepts the same language of strings but is unambiguous. The new grammar should adhere to your decisions in part (c). You do not need to be concerned about whether one can parse left-to-right, just whether the grammar ambiguous.