Assignment 3: Parsing and Single Static Assignment

15-411/611: Course Staff

Due Tuesday, February 25th, 2025 (11:59PM)

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

As a reminder, **only submissions typeset in LaTeXwill be accepted**.

Problem 1: Parsing (30 points)

The famed 15-150 mascot Polly has invaded the compilers course-staff and is in a glorious language battle with Opal the Otter!

Of course, Polly is not a fan of C0 instead preferring SML. But rather than changing the entire class to compile to SML, he decides to instead write a new language. Using context-free grammars Polly comes up with the language, C_0^{λ}, which combines the usability of lambda calculus with the safety of C. He specifies it with the following grammar (noting that **x** is an identifier token and that γ_3 denotes function application[1]).

$$\begin{array}{lcll} \gamma_1 & : & \langle \text{E} \rangle & \rightarrow \quad \textbf{x} \\ \gamma_2 & : & \langle \text{E} \rangle & \rightarrow \quad \lambda\ \textbf{x}\ \textbf{.}\ \langle \text{E} \rangle \\ \gamma_3 & : & \langle \text{E} \rangle & \rightarrow \quad \langle \text{E} \rangle\ \langle \text{E} \rangle \\ \gamma_4 & : & \langle \text{E} \rangle & \rightarrow \quad \textbf{(}\ \langle \text{E} \rangle\ \textbf{)} \end{array}$$

(a) Polly gets Opal to review his grammar, and Opal uncovers a number of problems. Show two ambiguities in the above grammar by providing for each ambiguity two possible parse trees for the same string.

(b) Clarabelle the 15-210 mascot is seeking to use Polly's revolutionary language, having found much success with SML. However, Clarabelle requires an unambiguous grammar. As compiler writers, help Polly fix his language and rewrite the grammar so it is unambiguous. **You must accomplish this by modifying currently existing rules, and/or adding up to one more rule for a total of 5 rules.**[2].

For each ambiguity you found in (a), identify which of the two parse trees will be accepted by your new grammar. The grammar should describe the same set of strings as the original grammar.

(c) The 15-312 mascots, Progress and Preservation, also want to use Polly's language (being fans of lambda calculus). However, Progress and Preservation attempt to use a left-to-right recursive descent parser and discover that it loops infinitely.

Identify one reason why the top-down parser might loop infinitely on the grammar, and explain how to fix it.

[1] For example, **f x** is the function **f** applied to the argument **x**.

[2] Hint: your new grammar may or may not have the precedence you'd expect from lambda calculus.

Problem 2: Stuck in the Middle (20 points)

After Polly's successful release of C_0^{λ} she wonders if she dismissed C0 a little to quickly. Opal informed her of a way to convert a C0 program to be more faster and more functional...

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

$$c_0 = n$$

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

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

The following is a valid three-address abstract assembly translation:

```
collatz(n):
    r <- n
    goto .loop
.loop:
    if (n > 1) then .body else .done
.body:
    if (n > r) then .l1 else .l2
.l1:
    r <- n
    goto .l2
.l2:
    m <- n % 2
    if (m == 0) then .l3 else .l4
.l3:
    n <- n / 2
    goto .loop
.l4:
    n <- n * 3
    n <- n + 1
    goto .loop
.done:
    ret r
```

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

(b) Show the dominator tree of the program pictorially. Then, for each basic block, write its corresponding dominance frontier.

(c) Convert the abstract assembly program from (a) into SSA, inserting Φ-functions and renaming variables as necessary. It is sufficient to submit the final result of your transformation.

(d) Apply the de-SSA transformation by replacing the Φ-functions back into sequential moves.