Assignment 3: Parsing and Single Static Assignment

15-411/611: Course Staff

Due Wednesday, March 1, 2023 (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.

Problem 1: Parsing (30 points)

The famed 15-150 mascot Polly has invaded the compilers course-staff! Of course, he 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]).

$$\begin{array}{lcccl} \gamma_1 & : & E & \rightarrow & \mathbf{x} \\ \gamma_2 & : & E & \rightarrow & \lambda\, \mathbf{x}\, \mathbf{.}\, E \\ \gamma_3 & : & E & \rightarrow & E\, E \\ \gamma_4 & : & E & \rightarrow & \mathbf{(}\, E\, \mathbf{)} \end{array}$$

(a) Polly gets Ethan to review his grammar and 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) James's company Compilers-Я-Us is seeking to acquire Polly's revolutionary language, but the terms of the acquisition require an unambiguous grammar. Help Polly achieve his billion dollar buyout and rewrite the grammar so it is unambiguous[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.

[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. Perhaps there is a way to convert a C0 program to be 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) Convert the abstract assembly program from (a) into SSA with parameterised blocks.

(c) Transform your SSA code into minimal SSA. You may want to use ϕ functions to aid you. It is sufficient to submit the final result of your transformation.

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

(e) Identify the three extended basic blocks in this program and write what they are.