Assignment 2: Lexing and Parsing

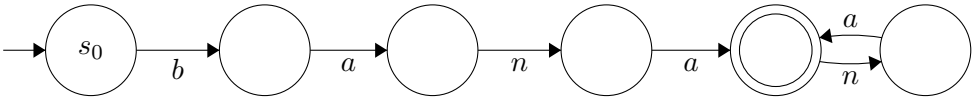
15-411/611: Course Staff

Due Tuesday, October 6, 2020 (11:59PM)

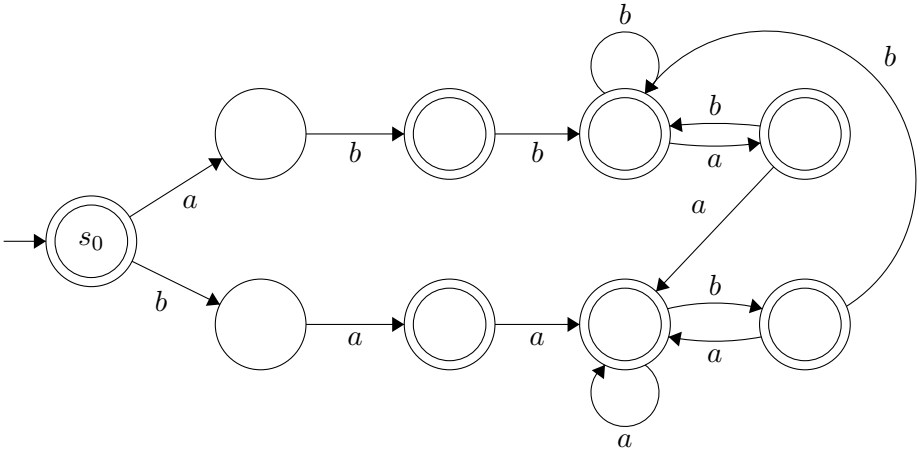
Reminder: Assignments are individual assignments, not done in pairs. The work must be all your own. Hand in your solutions on Gradescope. Please read the late policy for written assignments on the course web page.

Problem 1: Lexing (20 points)

(a) In class, we discussed how lexers can use regular expressions to parse an input corpus into tokens. A common way of implementing a regular expression parser is with deterministic finite automata, or DFAs, which you might have seen in 251[1]. For example, here is a DFA that accepts the language described by the regex `ban(an)*a`:



Pranav has decided to create a new language C_{naught}, which is similar to C_0 but employs new and exciting tokens. To lex his identifiers, he handcrafts the following DFA that accepts some language L over the alphabet $\Sigma = \{a, b\}$.



[1]To learn more about DFAs, you can read about it in the textbook or ask the TAs. Seriously, our office hours aren't very busy (and Komal, a former 251 TA, would love to talk to you about DFAs). Check out this site for a visualization: http://hackingoff.com/compilers/regular-expression-to-nfa-dfa

Help Pranav simplify his language specification by finding a simple regular expression that denotes L. Although usual definitions of DFAs require that a transition be given for every character in Σ at every state, we omit certain transitions for brevity. When these transitions are taken, the DFA enters a permanent failure state. For example, a string that begins with the prefix "aa" will never be accepted by Pranav's DFA.

(b) Seth stumbles across Pranav's new language specification and thinks to himself, "Wow, this whole lexing business is far too simple." Reminiscing on his old SIGBOVIK days, Seth makes a new language $C_{\aleph_0}$ which requires that all identifiers be of the form $a^m b^n c^{m+n}$ for $m, n > 0$. However, Seth quickly finds that his old regular expression–based lexer generator won't properly lex these identifiers. Identify the limitation of Seth's lexer generator and why he cannot decide this language with the regular expression lexer generator.

Problem 2: Parsing (30 points)

Sobered by Seth's disastrous foray into lexing, Henry abandons regular expressions, instead writing a context-free grammar for his new 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[2]).

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

(a) Henry gets Albert's administrative assistant 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) Stephen's company Compilers-Я-Us is seeking to acquire Henry's revolutionary language, but the terms of the acquisition require an unambiguous grammar. Help Henry achieve his billion dollar buyout and rewrite the grammar so it is unambiguous[3]. 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) Not content to let the TAs have all the fun, you set out to write your own grammar, but run into some familiar pitfalls. Describe an unambiguous grammar G with fewer than 6 productions that contains a reduce/reduce conflict in a shift-reduce parser with lookahead 1. The language $L(G)$ must be context free, but not regular. You may use your parser generator of choice to help.

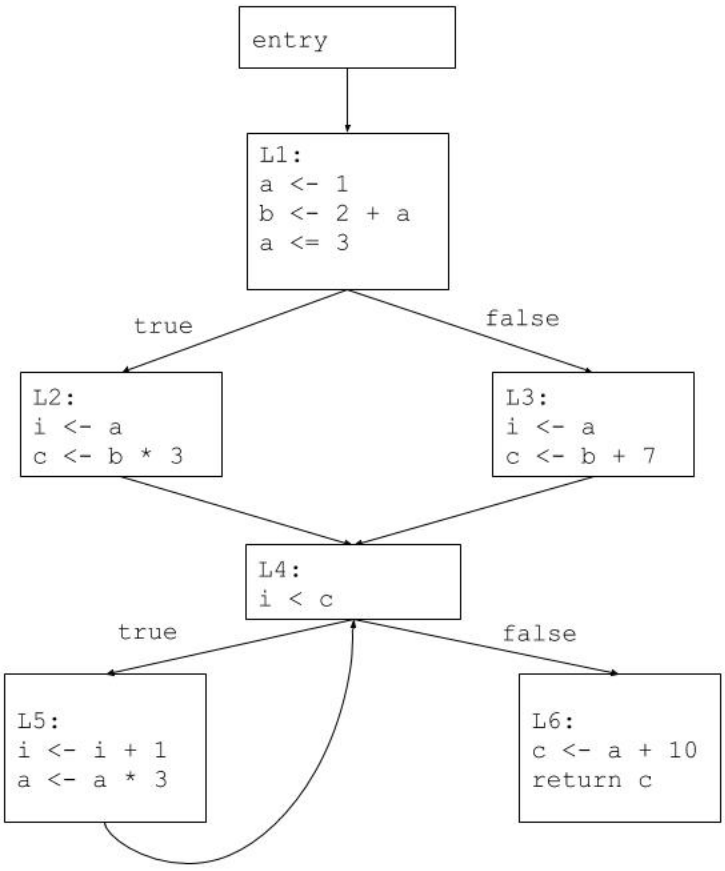
[2]For example, **f x** is the function **f** applied to the argument **x**.

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

(d) Prove that the reduce/reduce conflict in (c) exists by giving two conflicting derivations. (Compare the example given in the lecture notes.)

Problem 3: Dataflow Analysis (20 points)

The following is a CFG for a sample program, and we want to use dataflow analysis to find the available expressions at each basic block. Remember that available expressions is a **forward must** analysis, and the "facts" are expressions of the form "x op y"



(a) Provide the Gen and Kill sets for each basic block.

(b) Provide the In and Out sets for each basic block.

(c) List the best order to visit the nodes when solving this problem and why.