Assignment 2: Lexing and Parsing

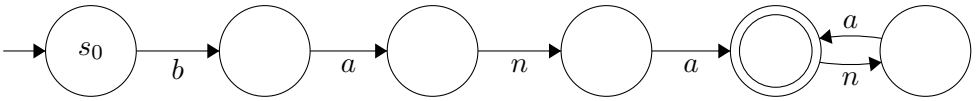
15-411: Compiler Design
Rob Simmons, Will Crichton, Grant Della Silva, Matt Bryant, Anshu Bansal

Due Thursday, October 1, 2015 (11:00pm)

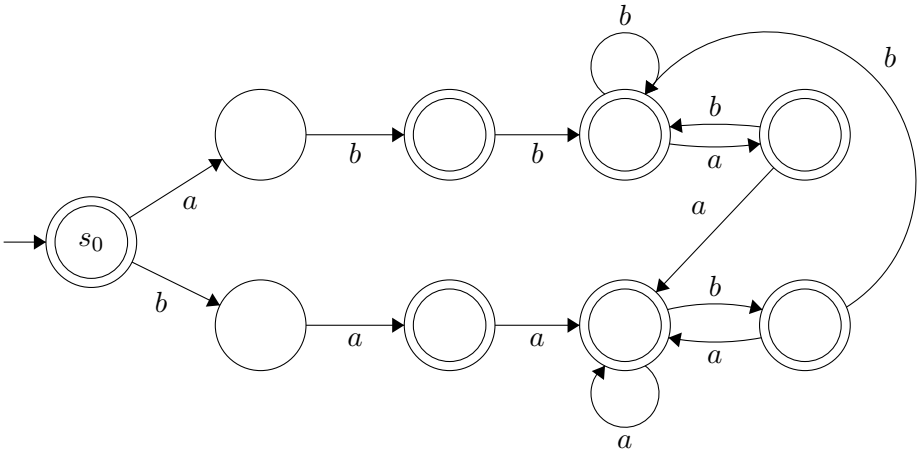
Reminder: Assignments are individual assignments, not done in pairs. The work must be all your own. Hand in your solutions as a PDF file on Autolab. Please read the late policy for written assignments on the course web page.

Problem 1: 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 or FLAC[1]. For example, the DFA for the regex `ban(an)*a` would be:



Anshu has decided to create a new language C_{naught} which is startlingly similar to C_0 but utilizes new and exciting tokens. To lex his identifiers, he hand-crafts the following DFA which accepts a 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. Also check out this neat site: http://hackingoff.com/compilers/regular-expression-to-nfa-dfa

Help Anshu simplify his language specification by finding a simple regular expression that accepts L. Note that although a DFA is required to a transition for every character in Σ defined for every state, we omit certain transitions for brevity—these enter a permanent failure state, i.e. a string that enters (like "aa" in the above DFA) it is guaranteed to never be accepted.

(b) Rob stumbles across Anshu's new language specification and thinks to himself, "Wow, this whole lexing business is far too simple." Reminiscing on his old SIGBOVIK days, Rob makes a new language $C_{\aleph_0}$ which requires that all identifiers must be of the form $a^n b^n c^n$ for $n > 0$. However, Rob quickly finds that his old regular expression-based lexer generator won't properly lex these identifiers. Identify the limitation of Rob's lexer generator and briefly describe the kind of tool he needs to solve this problem.

Problem 2: Parsing (40 points)

After Rob's disastrous foray into lexing, Matt figures he's safe by just writing a context free grammar for his new language, C_0^λ which combines all the usability of lambda calculus with all the safety of C. He specifies it with the following productions (note that x is an identifier token and n is a number token):

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

(a) Matt gets Grant's pet bison 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) Will's company Compilers-R-Us is seeking to acquire Matt's revolutionary new language, but the terms of the acquisition require that the grammar is unambiguous. Help Matt 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.

(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 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.

(d) Prove that the reduce/reduce conflict in (c) exists using the example from the notes.

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