Assignment 2: Lexing and Parsing

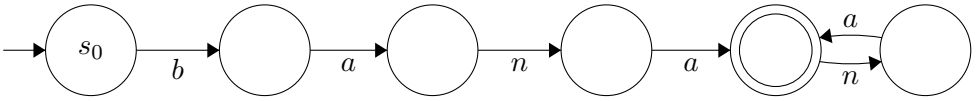
15-411: Compiler Design

Due Thursday, Sept 26, 2019 (11:59PM)

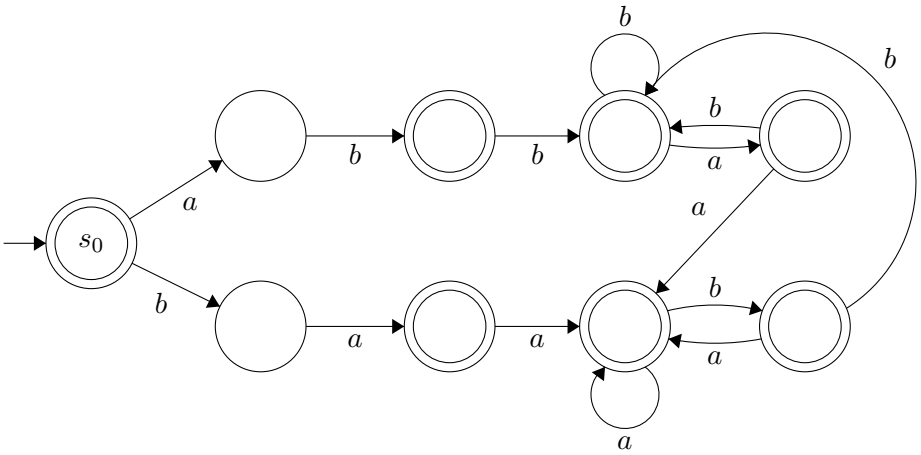
Reminder: Assignments are individual assignments, not done in pairs. The work must be all your own. Hand in your solutions on Gradescope. Please read the late policy for written assignments on the course web page.

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`:



Nick 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. Check out this site for a visualization: http://hackingoff.com/compilers/regular-expression-to-nfa-dfa

Help Nick 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 Nick's DFA.

(b) Jan stumbles across Nick's new language specification and thinks to himself, "Wow, this whole lexing business is far too simple." Reminiscing on his old SIGBOVIK days, Jan 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, Jan quickly finds that his old regular expression–based lexer generator won't properly lex these identifiers. Identify the limitation of Jan's lexer generator and why he cannot decide this language with the regular expression lexer generator.

Problem 2: Grammars (10 points)

In formal language theory, a context-free grammar is said to be in *Chomsky normal form* (first described by Noam Chomsky) if all of its production rules are of the form:

$$A \to BC \textbf{ or } A \to a \textbf{ or } S \to \epsilon$$

where A, B, and C are nonterminal symbols, a is a terminal symbol, S is the start symbol, and ϵ denotes the empty string (if it is in the language).

To convert a grammar to Chomsky normal form, a sequence of simple transformations is applied in a certain order; this is described in most textbooks on automata theory. This conversion is called CNF conversion; the conversion algorithm is often used by algorithms as a preprocessing step (including the CYK parsing algorithm).

But why is this useful? The CYK parsing algorithm, used by some parser generators, runs in $\mathcal{O}(n^3)$ time, where n is the number of tokens in the string. This is one of the best parsing algorithms known in terms of worst-case asymptotic complexity; others exist that are better in average running time.

(a) Which language is generated by the grammar G given by the following rules (where S is the start symbol)?

$$S \to 0S0 \mid 0B00$$

$$B \to 1B \mid 1$$

(b) Convert the grammar G from part (a) into a grammar G' in Chomsky normal form that generates the same language; that is, $L(G) = L(G')$.

(c) Informally argue why $L(G) = L(G')$.

Problem 3: Parsing (30 points)

Sobered by Jan's disastrous foray into lexing, Prachi abandons regular expressions, instead writing a context-free grammar for her new language, C_0^λ, which combines the usability of lambda calculus with the safety of C. She specifies it with the following grammar (noting that **x** is an identifier token and that γ_3 denotes function application[2]).

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

(a) Prachi gets Vijay's administrative assistant to review her 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) Shalom's company Compilers-Я-Us is seeking to acquire Prachi's revolutionary language, but the terms of the acquisition require an unambiguous grammar. Help Prachi achieve her 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.

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

[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.