

CS321
Theory of Computation
Exam II, Fall 2008

Name:

1. (a) Consider an NPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$.
- i. [5pt] What is the domain and range of the transition function δ ?

The domain is $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$

The range is $2^{Q \times \Gamma^*}$, i.e. finite subsets of $Q \times \Gamma^*$.

- ii. [5pt] Give a formal definition of $L(M)$ in set notation. You will want to make use of the relation \vdash^* which indicates the reachability of one configuration from another.

$$L(M) = \{w : w \in \Sigma^*, (q_0, w, z) \vdash^* (p, \lambda, u), p \in F, u \in \Gamma^*\}$$

- (b) [5pt] Consider a context-free grammar $G = (V, T, S, P)$. Give a formal definition of $L(G)$ in set notation. You will want to make use of the relation \Rightarrow^* .

$$L(G) = \{w : w \in T^*, S \rightarrow^* w\}$$

- (c) [5pt] A finite-stack NPDA (FS-NPDA) is an NPDA with a finite amount of stack space. Which of the following language classes can an FS-NPDA accept: 1) Regular Languages, 2) Context-Free Languages, 3) Neither. You must explain your answer.

FS-NPDAs are equivalent in power to NFAs, and hence can accept any regular language, but not all context-free languages.

To see this note that given a finite amount of stack space there are only a finite number of possible stacks. Furthermore given any state, any input character, and any one of those finite stacks we can completely describe the set of possible transitions to a next state and next stack. Using this insight we can create an NFA that has a state for every pair (q, v) where q is an FS-NPDA state and v is a possible stack, and has possible transitions between (q, v) and (q', v') only when the corresponding FS-NPDA would allow such a transition. The resulting NFA would simulate the behavior of the FS-NPDA and hence accept the same language.

2. (a) [10pt] Consider the language $L = \{a^m b^{mn} c^n : m \geq 0, n \geq 0\}$. Use the Pumping Lemma to show that L is not regular.

Assume that L is regular for the sake of contradiction and let m be the pumping constant. Consider the string $w = a^m b^m c$, which is in L . Since the first m characters are a's, using our standard argument, the pumping lemma implies that for some $j > 0$, $w_i = a^{m+j(i-1)} b^m c$ is in L for all $i \geq 0$. However for $i = 0$ we get that $w_0 = a^{m-j} b^m c$ which is not in the language, leading to a contradiction. Thus our original assumption was incorrect, showing that L is not regular.

- (b) [10pt] For this problem you may use the fact that **NONE** of the following languages are regular:

$$L_1 = \{a^i b^i : i \geq 0\}, \quad L_2 = \{a^i b^j : i \leq j\}, \quad L_3 = \{a^i b a^i : i \geq 0\}$$

You may also use the fact that regular languages are closed under all of the following operations: *union, intersection, complementation, reversal, homomorphisms, concatenation*. Use closure properties to show that the language

$$L = \{c^i w c^i : i \geq 0, w \in \{a, b\}^*, n_a(w) \neq n_b(w)\}$$

is not regular.

There are several possible solutions to this. Here is one particularly simple one.

Assume that L is regular for the sake of contradiction. Since we know that regular languages are closed under intersection and $L(c^* b c^*)$ is regular then we know that

$$L \cap L(c^* b c^*) = \{c^i b c^i : i \geq 0\}$$

is regular. We now consider the homomorphism h , which maps the character c to a , and keeps other characters unchanged. Since regular languages are closed under homomorphism we get that $h(L \cap L(c^* b c^*)) = \{a^i b a^i : i \geq 0\}$ is regular. However, this language is equal to L_3 above, which we are told is not regular, giving a contradiction. Thus, our original assumption was incorrect, showing that L is not regular.

3. (a) [10pt] Give a context-free grammars for the following two languages

$$L_1 = \{ww^Rc^n : w \in \{a,b\}^*, n \geq 0\}$$

and

$$L_2 = \{a^ib^i : i \geq 0\} \cup \{c^id^j : i \geq 0, j \geq 0\}$$

A grammar for L_1 is as follows:

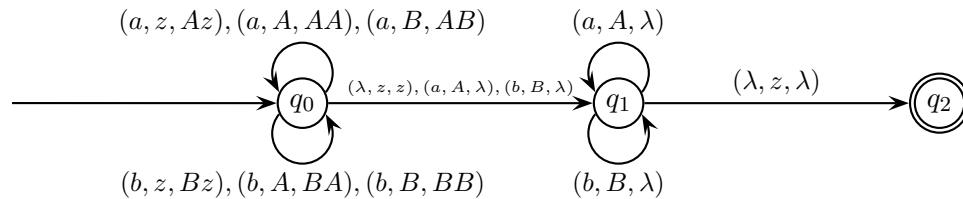
$$\begin{aligned} S &\rightarrow S_1C \\ S_1 &\rightarrow aS_1a \mid bS_1b \mid \lambda \\ C &\rightarrow cC \mid \lambda \end{aligned}$$

A grammar for L_2 is as follows:

$$\begin{aligned} S &\rightarrow S_1 \mid S_2 \\ S_1 &\rightarrow aS_1b \mid \lambda \\ S_2 &\rightarrow cS_2 \mid S_2d \mid \lambda \end{aligned}$$

- (b) [10pt] Give an NPDA for the language $L = \{vv^R : v \in \{a,b\}^*\}$.

The machine operates by pushing some number of characters onto the stack so that they are essentially in reverse order. The machine then non-deterministically chooses a moment to begin popping off those characters and comparing them to the remaining sequence of input characters. If finally the bottom of the stack is reached and the input is completely read in then the string must be of the form vv^R and should be accepted. For any string of this form there is always a way for the machine to guess a point to stop pushing and begin popping from the stack and reach the accept condition. For strings not of that form, there will never be a way to make a guess that reaches an accept condition. This example is also in your book so you can look there for another explanation.



4. Consider the following context-free grammar G for arithmetic expressions with terminal set $T = \{a, b, +, *, (,)\}$, variable set $V = \{S, E\}$, and start symbol S .

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow E + E \mid E * E \\ E &\rightarrow (E) \\ E &\rightarrow a \mid b \end{aligned}$$

- (a) [10pt] Show that G is ambiguous.

Below we show that there are two distinct derivation trees for the string $w = a + a + a$, which shows that this grammar is ambiguous.

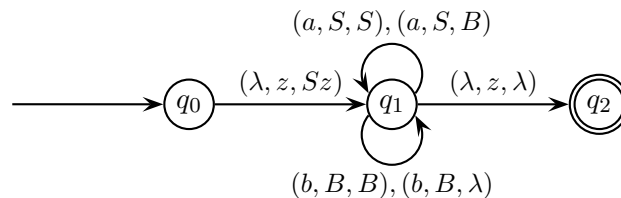
$$\begin{array}{c} S \\ | \\ E \\ / \quad | \quad \backslash \\ E \quad + \quad E \\ / \quad | \quad \backslash \quad | \\ E \quad + \quad E \quad a \\ | \quad \quad | \\ a \quad \quad a \end{array} \qquad \begin{array}{c} S \\ | \\ E \\ / \quad | \quad \backslash \\ E \quad + \quad E \\ | \quad / \quad | \quad \backslash \\ a \quad E \quad + \quad E \\ | \quad \quad | \\ a \quad \quad a \end{array}$$

- (b) [5pt] Explain why it is undesirable for the above grammar to be ambiguous.

Often the derivation tree of a string is treated as the meaning of the string and is used to guide computation of later processes. When there is not a distinct derivation tree for a string, this means that there can be multiple meanings for a string, for which the later process might produce different answers. Often this is undesirable. For example, if the later process is an evaluator of mathematical expressions, we would like to guarantee that no matter how a numerical expression string is parsed that we get the same answer. However, that might not be the case for an ambiguous grammar. As described in class, for example, $1 + 2 * 3$ can give different results depending on whether one first does the multiplication or the addition, which would typically be determined by the derivation tree of the string.

- (c) [10pt] Using the procedure described in class and the book, construct a non-deterministic pushdown automaton (NPDA) M that is equivalent to the following grammar.

$$\begin{aligned} S &\rightarrow aS \mid aB \\ B &\rightarrow bB \mid b \end{aligned}$$



5. [15pt] Let REMOVE-A be an operation on languages such that REMOVE-A(L) is a new language where the strings of L have been modified by deleting all of the a's. For example, if $L = \{a, ab, aabb, bbb\}$ then REMOVE-A(L) = $\{\lambda, b, bb, bbb\}$. Show that the set of regular languages is closed under the REMOVE-A operation. (If you don't understand the operation from this description, please come ask me.)

For any regular language L we know that it has an NFA N . Consider creating a new NFA N' that is an exact copy of N , except that any transition labeled by an 'a' is replaced by a λ transition.

Now consider any string $w \in L$. We know that there is a path in N for w from the initial state to a final state that reads in all of w . Now consider that same path in N' . This path leads from the start state to a final state and reads in all characters in w with the a's removed, as desired. This shows that REMOVE-A(L) $\subseteq L(N')$.

Now consider any string $w \notin L$. We know that there is no path in N that reads in all of w and ends in a final state. From this it is easy to see that in N' there will be not path to the final state for the string corresponding to w with the a's removed. This shows that $L(N') \subseteq \text{REMOVE-A}(L)$.

Together these show that $L(N') = \text{REMOVE-A}(L)$.