CS 581 Assignment 2

Jessica Fortier

October 18, 2011

1.52 L is a language of finite or infinite index; X is a set of strings.

(a) Proof: If L is recognized by a DFA with k states, L has index at most k.

Consider a DFA $A = (Q, \Sigma, \delta, q_0, F)$, where:

$$\ell(A) = L,$$
$$Q = \{q_1, q_2, ..., q_k\} \rightarrow |Q| = k,$$
$$\text{and strings } s = s_1 s_2 ... s_n | s \in \Sigma^* \wedge n \geq 0.$$

By definition, any string s catalyses a series of n state changes within A according to transition function δ. Let the ending state of this series of transitions be $q_n \in Q$ for a specific s. The deterministic nature of a DFA guarantees that for any string $s \in \Sigma^*$ there is exactly one ending state q_n. Thus consider k sets, one corresponding to each state in Q, which contain the strings over Σ^* such that set Q_n contains all strings ending on state q_n in A, and the union of Q_1 through Q_k is Σ^*. Determinism also guarantees states Q_1 through Q_k to be disjoint, and to cover Σ^* completely.

For any one of these sets, consider the concatenation of a string s in the set and another string z. Because the transitions resulting from the symbols in s end at q_n, the transitions resulting from z would begin at q_n for concatenated string sz, ending at some state $q_m \in Q$. Because the finite automaton is deterministic, the route taken to arrive at q_n is irrelevant to the route taken from q_n in response to the symbols in z, and the ending state q_m for sz is thus identical for all strings in the same set Q_n. Thus, where γ_m is the set of strings s such that sz ends at state q_m, we can say that:

$$\forall x, y \in Q_n \wedge z \in \Sigma^*, \quad \nexists z | xz \in \gamma_m \wedge yz \notin \gamma_m$$

Because q_m is a state in A, which recognizes L, q_m is either an accept state or not; from this binary definition it follows that if q_m is an accept state, then x and $y \in \gamma_m$ must exist in L and if q_m is not an accept state, then x and $y \in \gamma_m \notin L$. Thus the expression above can be rewritten as follows:

$$\forall x, y \in Q_n \wedge z \in \Sigma^*, \quad \nexists z | xz \in L \wedge yz \notin L$$

By this expression we know x and y to be indistinguishable by L for all x and y that share an ending state in A. Thus all strings which share an ending state in A form equivalence classes of L, and there are at most k of these equivalence classes because there are only k states in A. A set that is pairwise distinguishable by L is a set containing at most one element from each of these equivalence classes, because otherwise there would be at least one element indistinguishable by L with another element in the set by the pigeonhole principle and definition of pairwise distinguishability. **Thus a pairwise distinguishable set by L can have only at most k elements, and therefore the index of L must be at most k.**

(b) Constr., Proof: If L has finite index k, L is recognized by a DFA with k states.

Consider a DFA $M = (Q, \Sigma, q_0, \delta, F)$ in the usual manner where $\ell(M) = L$, a language of index k. Take the set $X = \{s_1, ..., s_k\}$ to be pairwise indistinguishable by L, and say that for every string s_i in X, there is a state $q_i \in Q$ which recognizes string s_i and all other strings belonging to the same equivalence class as s_i. Consider that the transitions from one state in M to another adhere to $\delta(q_i, a) = q_j$. This would imply that $s_i a \equiv_L s_j$.

If $s_i a \equiv_L s_j$ holds under the condition that $s_i \not\equiv_L s_j$, then $s_i \not\equiv_L s_i a$, implying that a transition will always lead to one of the k states in M: either to the state from which the transition originated, or to another state corresponding to a string distinguishably by L from the state where the transition originated. In this case, k states are all that is necessary because all transitions are occurring between the existing equivalence classes represented in X.

If this relationship does not hold under these conditions, then there exist no string $s_j \in X$ which is indistinguishable by L from $s_i a$. So, X does not contain an element from each equivalence class of $\Sigma^* / \equiv_L$ (as the equivalence class containing $s_i a$ is not represented), and thus would have to have size $k + 1$ to be the maximum pairwise distinguishable set by L. This is a contradiction to our original assumption.

Therefore a language of index k is recognizable by a DFA with k states.

(c) Proof: L is regular iff its index is the size of the smallest DFA recognizing it.

$p =$ (L is regular).
$q =$ (index of L is the size of the smallest DFA recognizing it);

$\underline{p \to q}$

If L is regular, then L is recognizable by a DFA: M	Definition of regular languages
If M is a DFA, it can be minimized	Definition of a DFA
If M_{min} is the minimization of M, $\ell(M_{min}) = \ell(M)$	Definition of minimization
If M_{min} has k states, L is recognized by a DFA of size k	Definition of regular languages
If L is recognized by a DFA of size k, index of $L \leq k$	Proof (a) above
If X is pairwise distinguishable by L, $\lvert X \rvert \leq \lvert \Sigma^* / \equiv_L \rvert$	Definition of Pairwise Distinguishable
If $\lvert X \rvert \leq \lvert \Sigma^* / \equiv_L \rvert$, $\lvert X \rvert_{max} = \lvert \Sigma^* / \equiv_L \rvert$	Upper bound
If $\lvert X \rvert_{max} = j$, then index of $L = j$	Definition of language index
If $\lvert X \rvert_{max} = j$, then $\lvert M_{min} \rvert \geq j$	Definition of recognition
If M_{min} is minimal, $j = k$	Definition of minimization
If L is regular, then index of $L = j = k = \lvert M_{min} \rvert$	Transitivity

$\underline{q \to p}$

If index of L is the size of the minimal DFA to recognize L, then L is recognized by a DFA
If L is recognized by a DFA, then L is regular

If $p \to q \land q \to p$ then $p \leftrightarrow q$.

2.2 $A = \{a^m b^n c^n | m, n \geq 0\} and B = \{a^n b^n c^m | m, n \geq 0\}$

(a) Illustrate: The class of context-free languages is not closed under intersection.

In language A, the exponents of b and c must be equal, whereas in language B the exponents of a and b must be equal. The intersection, $A \cap B$ is thus the set of strings where both constraints are met. By transitivity, both constraints are fulfilled only when the exponents of all symbols are the same: $a^n b^n c^n$.

Proof by Contradiction: Assume the class of context-free languages to be closed under intersection, and by extension the language $C = \{a^n b^n c^n | n \geq 0\} = A \cap B$ to also be a context-free language.

Contradiction: According to the proof in 2.36, the set of strings represented by language C is shown by the pumping lemma to *not* be context-free. Thus the intersection of two context-free languages is not always context-free, and the class of context-free languages is therefore not closed under intersection.

(b) Illustrate: The class of CFLs isn't closed under complementation.

Proof by Contradiction: Assume the class of context-free languages to be closed under complement such that if language L is context-free, its complement $\overline{L}$ is also context-free.

Let A and B be the context-free languages defined above, and C be the class of context-free languages.

$A,\, B \in C$	Given
$\overline{A}, \overline{B} \in C$	By assumption
$(\overline{A} \cup \overline{B}) \in C$	C is known to be closed under union.
$\overline{(\overline{A} \cup \overline{B})} \in C$	By assumption
$\overline{(\overline{A} \cup \overline{B})} = A \cap B$	DeMorgan's Law
Contradiction: $(A \cap B) \in C$	Transitivity

It was proven in the previous step that the class of context-free languages is not closed under intersection, and that the intersection of these languages A and B in particular is not itself a context-free language. Because of this contradiction, the class of context-free languages is also not closed under complementation.

2.18

(a) Constr., Proof: $C \cap R$ is context-free for context-free C, regular R

A context-free language has a PDA that recognizes it, by definition. Consider the PDA for C: $M_C = (Q_p, \Sigma, \Gamma, \delta_p, q_0, F_p)$. It is essentially a finite automaton that also makes use of a stack to maintain its state, so that the transition function takes parameters q, s, and γ which represent the current state, the input, and the top stack symbol respectively.

A regular language has a DFA that recognizes it, by definition. Consider the DFA for R: $M_R = (Q_d, \Sigma, \delta_d, q_0, F_d)$. The transition function here is only a function of current state q, and input s.

Consider that the two machines are merged into a PDA so that they share a combined state space $Q = Q_p \cup Q_d$. Because the transition function of the R's DFA is independent of the stack, it is possible to transition through the the state space Q for recognition of R regardless of the contents of the stack. Because R's DFA does not use the stack, it is possible to transition through the state space Q as before to recognize the language C. Thus it is possible that both machines may coexist and end on states they did before. Now consider the set of states $F = F_p \cap F_d \subseteq Q$ for the new machine. Because we seek to model the language $C \cap R$, we only accept on states that are acceptable to both machines. This is a valid PDA, therefore the language $C \cap R$ is context-free.

(b) Illustrate: $A = \{w | w \in \{a, b, c\}^* \wedge\}$ is not context-free for $\#(a) = \#(b) = \#(c)$

With a PDA it is possible to track equal numbers of symbols by pushing a symbol onto the stack any time a specific symbol is received as input to the PDA. Consider the language $B = \{w | w \in \{a, b\}^*$ and contains equal numbers of a's and b's. Let's initially push a marker \$ onto the stack. Then, for every a received as input push a # onto the stack, and for every b received as input pop the stack. Of course it's possible that at some point in a valid string, there may have been more b's consumed than a's, so we watch for if the stack marker is popped and handle appropriately. Regardless, any state in which the stack is empty with the exception of the stack marker is an accept state, because all stack symbols generated have been consumed, implying that the number of a's and b's is the same.

This works by virtue of the stack having two functions: push, and pop. Thus we assign push to one input symbol, and pop to the other, which allows us to use the size of the stack to tally the net amount of each. Because language A would require three values to be tracked, and those values mathematically could not cancel one another out, there is no way to utilize the stack for memory on all three symbols.

2.35 Proof: $|\ell(G)| = \infty$ if G generates a string in $\geq 2^b$ steps

G is a context-free grammar with b variables in Chomsky normal form: every rule of G must take the form either $A \rightarrow BC$ or $A \rightarrow a$ for variables A, B, C and terminal symbol a including start variable S where only $S \rightarrow \varepsilon$ is permitted, and $B, C \neq S$.

Because of this form, each rule followed can result in at most two more steps, such as in the case $A \rightarrow BC$. Consider a parse tree illustrating these rules in G, beginning with S (ex: $S \rightarrow AB$ would give the tree with S at the root, and A and B as the children of S). In this construction, every node contains a variable, and variables which yield terminating symbols are said to have no children. In this way, a node signifies a substitution being performed and substitutions involving terminal symbols are not double counted (ex: the node for some variable $A | A \rightarrow a$ would be counted to signify it initiated a substitution, but if a child node containing a were added then that terminal substitution would be counted again, even though no substitutions were performed subsequent to the exchange of A for a). It is known that the maximum number of nodes in a binary tree of height h is the complete binary tree with $2^{h+1} - 1 = n$ nodes, so it follows that this also describes the minimum height h of a binary tree with n nodes.

Assume a finite language $\ell(G)$ contains a string whose generation has $n \geq 2^b$ steps. The number of steps is equivalent to the number of variable substitutions made, and thus can be represented by the total number of nodes present in the tree for this string generation. We know that $2^b - 1$ nodes is the maximum number of nodes possible in a tree of height $(b-1)$: $\quad 2^{(b-1)+1} - 1$. Thus it follows that a tree of $n \geq 2^b$ nodes has a height of at least b.

A height of at least b really means that there are at least $b + 1$ tiers in the tree, and thus the path from the start variable S to the final variable which is substituted for a terminal symbol is actually $\geq (b + 1)$ variables deep. By the pigeonhole principle, this necessitates that at least one of G's b variables be repeated, indicating recursion and further indicating that language $\ell(G)$ is infinite.