University of Virginia - cs3102: Theory of Computation	Spring 2010
Regular Languages Notes	**4 February**

Upcoming Schedule:

- **Tuesday, 16 February:** Problem Set 2 is due at the beginning of class.
- Reading for next week: Sections 2.1 and 2.2.

Proofs

A *proof* is a convincing argument. An argument is a sequence of clearly stated claims. For an argument to be convincing, the first claim must follow from the given assumptions, each subsequent claim must follow from the previous claims, and the final claim must be equivalent to the statement you are proving. Please read through the comments on PS1 carefully to understand what makes a convincing proof.

Most of the theorems we prove in cs3102 have short, convincing proofs. If you feel like you need a long and convoluted proof you are probably either attempting to prove something that is not true, missing an easier way to prove what you want, or not finding the clearest way to present your idea. If you still cannot find a shorter proof, when you are writing a long (more than 3 lines) proof, you should include a sentence or two at the beginning explaining your overall proof idea, and enough prose in your proof to make it clear to a reader how the steps in your argument are connected, and how they show the statement you are proving is true.

The proofs we do in cs3102 will involve only a few main types of argument:

- **Proof by Construction** — If you can express the statement you are proving in the form, *An X exists*, then you can prove the statement by showing how to make an *X*. Many of the constructions in this class are of the form, *for any Y, there is an X* (e.g., for any *NFA*, there is *a DFA that recognizes the same language*). For a construction proof to be convincing, it needs to argue that the result of the construction is indeed an X (often, the way it is constructed is enough to be convincing for this already), and the construction method has to be general enough to be clear that it works for all possible Y. (Proof by reduction, which we will use frequently in the later parts of this course, is a special kind of construction proof.)

- **Proof by Contradiction** — To prove X, start by assuming X is not true, and show that it leads to a conclusion that is obviously untrue. In the PS1 comments, we used proof by contradiction for Problem 4b. In this class, we will often combine the proof by construction and proof by contradiction strategies by showing that if we had an X, we could use it to build a Y, but we know Y cannot exist. This proves that X cannot exist.

- **Proof by Induction** — We use proof by induction when we need to prove something is true for all elements of some infinite set that can be generated inductively. To

prove something is true for all elements in the infinite set we need to (1) *basis step* prove it is true for some finite number of elements of the set (usually just one); and (2) *induction step* prove that if it is true for some element of the set, it is also true for all elements of the set that can be generated from that element.

NFAs (from Class 4)

Definition: A *nondeterministic finite automaton* (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0, and F are defined as they are for a DFA, and δ is defined as:

> δ**:** $Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$ —- a function from a state and alphabet symbol to **a set of states** that is a member of $\mathcal{P}(Q)$, the power set of Q.

Computation Model for an NFA. The NFA $A = (Q, \Sigma, \delta, q_0, F)$ accepts the language $\mathcal{L}(A) = \{w | \delta^*(q_0, w) \cap F \neq \emptyset\}$ where δ^*: $Q \times \Sigma^* \longrightarrow \mathcal{P}(Q)$ is the *extended transition function* defined by:

> $\delta^*(q, \epsilon) = \{q\}$
> For $w = ax$ where $a \in \Sigma$ and $x \in \Sigma^*$, $\delta^*(q, ax) = \bigcup_{q_i \in \delta(q,a)} \delta^*(q_i, x)$

Equivalence of NFAs and DFAs. Show that the set of languages that can be recognized by some NFA is equal to the set of languages that can be recognized by some DFA.

1. Every DFA has an equivalent NFA. (Proof by construction - trivial.)
2. Every NFA has an equivalent DFA. (Proof by construction below, and in book, Theorem 1.39)

Given $N = (Q, \Sigma, \delta, q_0, F)$, and NFA recognizing some language A. We prove that every NFA has an equivalent DFA by showing how to construct a DFA N' from N that recognizes the same language A. $N' = (Q', \Sigma', \delta', q_0', F')$ defined as:

1. $Q' = \mathcal{P}(Q)$ — we have a state in Q' to represent each possible subset of states in Q. The label for each state in Q' is a set.
2. $\Sigma' = \Sigma$ — the alphabet is the same
3. $\delta' : Q' \times \Sigma' \rightarrow Q'$ is defined to capture all possible states resulting from δ transitioning from the input state: $\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$
4. $q_0' = E(q_0)$
5. $F' = \{$states in Q' that correspond to subsets of states that include a state in $F\}$

where $E : Q' \rightarrow Q'$: is the epsilon-transition function defined by:

$$E(q) = q \cup \bigcup_{r \in \delta(q,\epsilon)} E(r)$$

Example. Prove that the regular languages are closed under reversal. That is, if L is a regular language, then $L^R = \{w | w^R \in L\}$ is a regular language. (Hint: think about what you need to do to construct an NFA that recognizes the reverse of the language recognized by some DFA A, and use the equivalence of NFAs and DFAs.)

Non-regular Languages (Class 5)

Pumping Lemma

> If A is a regular language, then there is a number p (the pumping length) where for any string $s \in A$ and $|s| \geq p$, s may be divided into three pieces, $s = xyz$, such that $|y| > 0$, $|xy| \leq p$, and for any $i \geq 0$, $xy^iz \in A$.

Informal argument: if $s \in A$, some part of s that appears within the first p symbols must correspond to a loop in a DFA that recognizes A. We call y the string along that loop. Since it is a loop in the DFA, we can go around the loop any number of times without changing the acceptance result for A.

Game view: Suppose A is some language. Player one picks $p \in N$, the maximum number of states, and attempts to design a DFA M that recognizes A using p or fewer states. Player two picks a string s. Player two wins if she can find a string s that is not processed correctly by M (that is, either $s \in A$ and M rejects, or $s \notin A$ and M accepts). If there is a strategy player two can use to always win no matter what value player one picks for n, then A is not a regular language. Otherwise, A is regular.

We use the pumping lemma to prove a language is non-regular using proof by contradiction. The proof steps will always be similar to:

1. Assume A is regular.
2. Then, the pumping lemma is true for A: there is some number p, such that or any string $s \in A$ and $|s| \geq p$, s may be divided into three pieces, $s = xyz$, such that $|y| > 0$, $|xy| \leq p$, and for any $i \geq 0$, $xy^iz \in A$.
3. **The creative part:** Identify a string s that leads to a contradiction of the pumping lemms. The string s can be built using p. The proof must show that there is *no possible way* to divide $s = xyz$ such that $|y| > 0$ and $xy^iz \in A$ for any $i \geq 0$.

We can condense the first 2 steps into the statement, "Assume A is regular and p is the pumping length for A."

Example. Prove the language $\{a^n b^n | n \geq 0\}$ is not regular.

Assume $\{a^n b^n | n \geq 0\}$ is regular and p is the pumping length for A.

Choose $s =$ [ ]. $s \in A$ since [ ].

Since the pumping lemma requires that the repeating string is found in the first p symbols, $|xy| \leq p$. However we choose x and y, y can only contain as. Since y only contains as, $xy^i z$ always contains p 1s, but the number of 0s increases with i. Thus, we have a contradiction: the pumping lemma says that for any $s \in A$ we should be able to find $s = xyz$ such that $xy^i z \in A$ and $|y| > 1$ and $|xy| \leq p$, but we have shown that no matter how we pick x and y (subject to the constraints), we can produce strings that are not in A.

Example. Prove the language $\{w | w$ has more as than bs$\}$ is not regular.

Regular Languages Summary

- A *language* is a set of strings. A machine *recognizes* a language if for all possible strings, it correct decides if the string is in the language.
- A language is *regular* if and only if some DFA recognizes that language.
- DFAs, NFAs, and Regular Expressions are equally powerful: they can recognize exactly the same set of languages (the regular languages). We can prove this by showing how to construct a DFA that recognizes the same language as any NFA, etc. in both directions for each pair of machines.
- To prove a language is regular: construct a DFA, NFA or RE that recognizes it.
- To prove a language is not regular: show that recognizing it requires keeping track of infinite state (hard to be completely convincing in most cases) or use the pumping lemma to get a contradiction.