CSE 322: Introduction to Formal Models in Computer Science
Assignment #3
April 17, 1994
due: Friday, April 24

1. Exercise 2.2.2.

2. A *DNA sequence* is a string over the alphabet $\Sigma = \{A, C, G, T\}$. A *protein-coding region* is a substring of a DNA sequence that has the "start codon" ATG as a prefix, has one of the three "stop codons" TAA, TAG, or TGA as a suffix, and whose length is a multiple of 3. (Every triple of characters between the start and stop codon encodes one amino acid of the protein. For instance, CTC is the code for the amino acid Leucine.)

   A DNA sequence w has an *open reading frame* if and only if there is some $i \in \{0, 1, 2\}$ such that there is no stop codon that begins at any position in w whose remainder modulo 3 is i. Open reading frames are used by algorithms to locate long protein-coding regions.

   In the following state diagrams, feel free to use the convention of labeling an edge with a subset S of $\Sigma \cup \{e\}$, as in Figure 2-8. This is a shorthand for $|S|$ parallel edges, one for each element of S.

   (a) Give the state diagram for a nondeterministic finite automaton that accepts exactly those DNA sequences that have at least one protein-coding region. For instance, ACTTATG CGG CTG TGG GCA CCG TGATAAC and ATGTTATG CGG CTG TGG GCA TAG should both be accepted, but TTATGCGGCTGTGGGCACCTAG should not be accepted. Try to use as few states as possible. Give a brief explanation of why your automaton works.

   (b) Give the state diagram for a nondeterministic finite automaton that accepts exactly those DNA sequences that have at least one open reading frame. For instance, TTAAG CTA ATG ACA TAT CCGTA should be accepted, but TTAAGCTAATGACATAGCCGTA should not be accepted. Try to use as few states as possible. Give a brief explanation of why your automaton works.

   (If you want a challenge, try to devise from scratch a deterministic finite automaton that accepts either of these languages. There is no need to hand anything in for this challenge, because you could always "cheat" and just use the construction of Theorem 2.2.1 on your nondeterministic finite automaton. If you discover some simpler direct deterministic finite automaton construction for either of these languages, let me know.)

3. Let $M = (K, \Sigma, \Delta, s, F)$ be a nondeterministic finite automaton, let $\sigma \in \Sigma \cup \{e\}$, and let $x, y \in \Sigma^*$. In each of the following proofs, be sure to give a precise justification based

on definitions, theorems, etc. for each step: it is not sufficient to use a justification like "... because y can't affect the computation before it is read".

(a) Prove carefully that $(q, \sigma y) \vdash_M (p, y)$ iff $(q, \sigma) \vdash_M (p, e)$.

(b) Use part 3a to prove carefully that $(q, \sigma xy) \vdash_M (p, xy)$ iff $(q, \sigma x) \vdash_M (p, x)$.

(c) Use part 3b to prove carefully that $(q, xy) \vdash_M^* (p, y)$ iff $(q, x) \vdash_M^* (p, e)$. This proves an important property of finite automaton computations that I have already used a few times in lecture. Hint: Let $x = \sigma x'$, where $\sigma \in \Sigma \cup \{e\}$, and use induction on the length of the computation.

4. Exercise 2.2.9(a). You must use the construction of Theorem 2.2.1. You need only give the state diagram of the deterministic finite automaton, and need only include those states reachable from the start state.

5. Reread Example 2.2.2 in the text.

   (a) Convert the nondeterministic finite automaton shown in Figure 2-8 into an equivalent deterministic finite automaton by the construction of Theorem 2.2.1. You need only give the state diagram of the deterministic finite automaton, and need only include those states reachable from the start state.

   (b) As Example 2.2.2 says, if you were to generalize Figure 2-8 to an alphabet of size n, the nondeterministic finite automaton would have $n + 1$ states. How many states would the equivalent deterministic finite automaton constructed by Theorem 2.2.1 have? How many of those states would be reachable from its start state?