CSci 423 Homework 4

Due: 1:00 pm, Wednesday, 10/10
Eric Shih

1. (8 points) Prove that the language A of strings of 0s and 1s whose length is a perfect square is not regular

   We will use proof by contradiction. Assume that A is a regular language, therefore the pumping lemma applies.

   Let p be the constant in the pumping lemma. Select $s = (0 \cup 1)^{p^2}$. $|s| \leq p, s = xyz, |xyz| = p^2, |y| > 0$, and $|xy| < p$.

   According to the pumping lemma, $\forall i \leq 0$, $xy^iz \in A$. Therefore when y is pumped, s should still be a perfect square, thus $|y|$ to any perfect square should result in the next perfect square.

   $|s| = p^2$, so the next perfect square will be $(p+1)^2$; letting $i = 2$: $p^2 + 1 \leq |xy^2z| \leq p^2 + p$.

   $|xy^2z|$ can't be the correct perfect square p^2 nor can it be the next perfect square $(p+1)^2$ (p^2+2p+1) since $|xy^2z| \leq p^2 + p \leq p^2 + 2p + 1$. This is a contradiction and the pumping lemma does not apply. Thus we have shown that A is not regular.

2. (8 points) Problem 1.46 (c) on page 90.

   Consider $\overline{L}$ where all strings are palindromes. By the closure properties, if $\overline{L}$ is not regular, L is not regular.

   Assume $\overline{L}$ is regular, so the pumping lemma applies. Let p be the constant in the pumping lemma and $s = xyz$ where $\forall i \leq 0$, $xy^iz \in \overline{L}$, $|y| > 0$, and $|xy| < p$.

   Let $s = 0^p10^p$, y will be a substring in the first half of s containing at least one 0 and up to p number of 0s. z will be 10^p. When $i = 0$, there will be at least one 0 removed from the first part of x, leaving xz resulting in anywhere from $0^{p-1}10^p$ to 10^p, none of which are palindromes.

   Thus the pumping lemma does not hold, proving that $\overline{L}$ is not regular, and by closure L.

3. (16 points) Problem 1.49 (a) and (b) on page 90.
   a) Strings accepted in B are:

k	1^ky
1	$1\ldots1\ldots$
2	$11\ldots1\ldots1\ldots$
3	$111\ldots1\ldots1\ldots1\ldots$
4	$1111\ldots1\ldots1\ldots1\ldots1\ldots$

   $\forall w \in B$, $w = 1^ky = 11^{k-1}y$. From this w can give the regular expression $1(0\cup1)^*1(0\cup1)^*$ proving B to be regular.

4. (8 points) Problem 1.54 (a) on page 91.

   F can be defined by 3 subsets:

- $i = 0$, $F_0 = b^*c^*$, this subset is regular.
- $i = 1$, $F_1 = ab^nc^n$, found to be not regular in a previous proof.
- $i \geq 2$, $F_2 = aa^+b^*c^*$, this subset is regular.

Assuming F is regular, $F = F_0 \cup F_1 \cup F_2$ and each subset is mutually disjoint. This leads to the conclusion that $F_1 = F - F_0 - F_2$. By the closure property, if F, F_0, and F_2 are regular then F_1 should be as well. But this is not the case; we know that F_1 is not regular from a previous proof, thus providing a contradiction, which in turn proves that F is not regular.

Reading Summary 2

Regular expressions are used for applications that search for patterns in text. Compiled into deterministic or non-deterministic automata, they are used to produce a program that recognizes patterns in text. Two of the most important classes of regular expression-based applications are lexical analyzers and text searches. UNIX notation is commonly used to make the applications for regular expressions easier. The UNIX extensions of character classes allow large sets of characters to be represented very succinctly.

The lexical analyzer scans the source program and recognizes all tokens, such as keywords and identifiers. A lexical-analyzer generator takes a high-level description of a lexical analyzer and produces a functioning lexical analyzer. lex and flex are examples of these sort of operations. They are very useful because they are able to generate an efficient function that breaks source programs into tokens. This shortens the amount of time that is spent generating a proper lexical analyzer. The conversion of the expressions to an automaton begins by building an automaton for all expressions. This only tells us that a token has been recognized, but the use of the NFA epsilon state, exactly which specific token can be recognized. One simple problem to this strategy is that more than one token can be recognized, but this can be solved by giving priority to the first expression listed.

Searching for patterns in text is another popular use for regular-expression notation. Despite not being as developed as lexical analyzers, using regular expressions for text searches will offer efficient automaton-based implementations. This technology is normally used to describe classes of patterns at a high level, and to allow for easy modifications. In the case of a Web application, usually a large amount of text needs to be scanned in order to figure out what a query is asking. In the example of finding a street address, a regular expression can be used to find what a street is. A set of regular expressions can be used to define what a street is, but can also easily be modified to add terms to the recognizer to improve its effectiveness.

Regular expressions are a powerful tool in producing automated programs. Two very commonly used applications are lexical analyzers and text searches. Regular expressions are used in lexical analyzers to simplify the generation of tokens. Text searches use regular expressions to search for wanted patterns in large amounts of text. Both applications use regular expressions to simplify the creation of the individual program, but also allows for easy modifications or changes.