HW Solution

CS/ECE 374: Algorithms & Models of Computation, Spring 2019 Version: **1.0**

Submitted by:

- ≪**Alan Lee**≫: ≪**alanlee2**≫
- ≪**Joshua Burke**≫: ≪**joshuab3**≫
- ≪**Gerald Kozel**≫: ≪**gjkozel2**≫

(100 PTS.) Fun with parity.

Given $L \subseteq \{0,1\}^*$, define $even_0(L)$ to be the set of all strings in $\{0,1\}^*$ that can be obtained by taking a string in L and inserting an even number of 0's (anywhere in the string). Similarly, define $odd_0(L)$ to be the set of all strings x in $\{0,1\}^*$ that can be obtained by taking a string in L and inserting an odd number of 0's.

(Example: if $01101 \in L$, then $01010000100 \in even_0(L)$.)

(Another example: if L is 1^*, then $even_0(L)$ can be described by the regular expression $(1^*01^*0)^*1^*$.)

The purpose of this question is to show that if $L \subseteq \{0,1\}^*$ is regular, then $even_0(L)$ and $odd_0(L)$ are regular.

1 (30 PTS.) For each of the base cases of regular expressions $\emptyset$, ε, 0, and 1, give regular expressions for $even_0(L(r))$ and $odd_0(L(r))$.

2 (60 PTS.) Given regular expressions for $e_j = even_0(L(r_j))$ and $o_j = odd_0(L(r_j))$, for $j \in \{1,2\}$, give regular expressions for

(i) $even_0(L(r_1 + r_2))$
(ii) $odd_0(L(r_1 + r_2))$
(iii) $even_0(L(r_1 r_2))$
(iv) $odd_0(L(r_1 r_2))$
(v) $even_0(L(r_1^*))$
(vi) $odd_0(L(r_1^*))$

Give brief justification of correctness for each of the above.

3 (10 PTS.) Using the above, describe (shortly) a recursive algorithm that given a regular expression r, outputs a regular expression for $even_0(L(r))$ (similarly describe the algorithm for computing $odd_0(L(r))$).

8 Solution:

A $even_0(\emptyset) = \emptyset$

$odd_0(\emptyset) = \emptyset$

$even_0(\varepsilon) = (00)^*$

$odd_0(\varepsilon) = 0(00)^*$

$even_0(0) = 0(00)^*$

$odd_0(0) = 00(00)^*$

$even_0(1) = (00)^*1(00)^*$

$odd_0(1) = (0(00)^*)^*1(0(00)^*)^*$

B $even_0(L(r_1 + r_2)) = even_0(L(r_1) \cup L(r_2)) = even_0(L(r_1)) \cup even_0(L(r_2)) = e_1 \cup e_2$. Because of the closure under union property, if L and M are regular languages, so is L ∪ M. this gives us the expression above.

$odd_0(L(r_1 + r_2)) = odd_0(L(r_1) \cup L(r_2)) = odd_0(L(r_1)) \cup odd_0(L(r_2)) = o_1 \cup o_2$. Because of the closure under union property, if L and M are regular languages, so is L ∪ M. this gives us the expression above.

$even_0(L(r_1r_2)) = even_0(L(r_1)L(r_2)) = even_0(L(r_1))even_0(L(r_2)) = e_1e_2$. Because of the closure under concatenation, if R and S are regular expressions using languages L and M, then RS is a regular expression whose language is LM

$odd_0(L(r_1r_2)) = odd_0(L(r_1)L(r_2)) = odd_0(L(r_1))odd_0(L(r_2)) = o_1o_2$. Because of the closure under concatenation, if R and S are regular expressions using languages L and M, then RS is a regular expression whose language is LM

$even_0(L(r_1^*)) = even_0(L(r_1)^*) = even_0(L(r_1))* = e_1^*$. Because of the closure under concatenation and Kleene Closure, if R is a regular expressions using language L, then R^* is a regular expression whose language is L^*

$odd_0(L(r_1^*)) = odd_0(L(r_1)^*)) = odd_0(L(r_1))^* = o_1^*$. Because of the closure under concatenation and Kleene Closure, if R is a regular expressions using language L, then R^* is a regular expression whose language is L^*

C For every component of the regular expression input, we will iterate over the entire regex. we start with the four base regular expression cases stated in part A and if one of them is encountered as the current iteration, then we add the corresponding value we found in part A to a dynamic storage element. If the regular expression fits into one of the classifications that are described in part B, then we go into the respective result found in part C in that case. We will further add these elements to our dynamic storage element. After the regular expression has been fully iterated over, we will return our dynamic storage element as the result. This works for both odd and even cases.