Assignment 3

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at here. I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located here.

Exercise 1

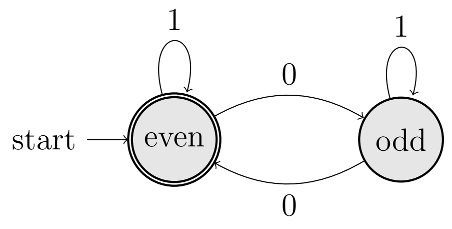
(40 points) Let L be the language over the alphabet $A = \{0, 1\}$ consisting of all words containing an even number of zeroes.

Part (a)

Draw a finite state acceptor that accepts the language L. Carefully label all the states including the starting state and the finishing states as well as all the transitions. Make sure you justify it accepts all strings in the language L and no others.

Solution

The finite state acceptor should be left in an accepting state only if it has received an even number of zeros. To model this, we need only two states:



Initially, the finite state acceptor is in the *even* state, as there have been zero 0 characters received. It stays in the *even* state until a 0 character is received. After the first 0 character, the internal state transitions to *odd*. Again, it stays in this state until a subsequent 0 character is passed to it, which returns the internal state to *even* again. The string is only accepted if the internal state finishes on the *even* state.

In this way, only strings with an even number of 0 characters leave the internal state as *even*, thus only these strings are accepted for the language L.

Part (b)

Devise a regular grammar in normal form that generates the language L. Be sure to specify the start symbol, the non-terminals and all the production rules.

Solution

The initial state of the finite state acceptor is the $\langle E\rangle$ state. This is also the only accepting state. It has three production rules for transitioning:

(1) $\langle E\rangle \to 1\langle E\rangle$

(2) $\langle E\rangle \to 0\langle O\rangle$

(3) $\langle E\rangle \to \epsilon$

On receiving a 1 character, the acceptor remains in the $\langle E\rangle$ state: The number of 0 characters received has not changed. On receiving a 0 character, the acceptor transitions to the $\langle O\rangle$ state: The number of 0 characters received has increased by one, from an even number to an odd number, taking the acceptor out of an accepting state. Finally, as this is an accepting state, it also has a production rule of type (iii).

The only other state is when the acceptor has so far received an odd number of 0 characters, labeled the $\langle O\rangle$ state. It has two production rules for transitioning:

(4) $\langle O\rangle \to 1\langle O\rangle$

(5) $\langle O\rangle \to 0\langle E\rangle$

On receiving a 1 character, the acceptor remains in the $\langle O\rangle$ state: The number of 0 characters received has not changed. On receiving a 0 character, the acceptor transitions back to the $\langle E\rangle$ state: The number of 0 characters received has again increased by one, now from an odd number to an even number, returning the acceptor to an accepting state.

Part (c)

Write down a regular expression that gives the language L and justify.

Solution

A regular expression that matches only language L would be $(1 \cup 01^*0)^*$. To understand this expression, we will break it down into its components:

The inner expression $1 \cup 01^*0$ accepts either a single 1 character or two 0 characters separated by an arbitrary number of 1 characters. Recalling the finite state acceptor diagram, this expression insures that the internal state of the acceptor either remains in the *even* state or transitions to the *odd* state and then returns to the *even* state again.

Wrapping this expression in a Kleene star allows this pattern to repeat. This covers all words in L.

Part (d)

Prove that language L is regular from the definition.

Solution

To prove L is a regular language, we must show that a finite sequence $L_1, L_2, \ldots, L_m$ exists where $L = L_m$ and $\forall i,\ 1 \leq i \leq m,\ L_i$ satisfies one:

1. L_i is a finite set.
2. $L_i = L_j^*$ where $1 \leq j \leq i$.
3. $L_i = L_j \circ L_k$ where $1 \leq j, k \leq i$.
4. $L_i = L_j \cup L_k$ where $1 \leq j, k \leq i$.

As shown before, L can be given by $(1 \cup 01^*0)^*$. L is regular if $(1 \cup 01^*0)^*$ can be shown to give a regular language using the rules above.

01^*0 gives a regular language, as 1^* follows the second property and $0 \circ 1^* \circ 0$ simply concatenates finite sets which combines the first and third properties. Then $1 \cup 01^*0$ must give a regular language, as 1 is a finite set and $1 \cup 01^*0$ follows from the fourth property. With this, $(1 \cup 01^*0)^*$ must give a regular language through applying the second property to $1 \cup 01^*0$.

Exercise 2

(20 points) Let M be the language over the alphabet $\{ a, r, c \}$ given by $M = \{ a^i r^j c^k \mid i, j, k \geq 0 \wedge i = 2j - k \}$.

Part (a)

Use the Pumping Lemma to show this language is not regular.

Solution

Assuming language M is regular, then it must have a pumping length p. Therefore all strings in M longer than p must satisfy the pumping conditions.

Consider a word $w = a^p r^j c^k$. Clearly, $|w| \geq p$. For w to also be a word in M, then $p = 2j - k$. As w is a string in M longer than p, it must satisfy the pumping conditions.

Suppose w can be split up so that $w = xuy$. According to the pumping conditions, $|xu| \leq p$ and $|u| > 0$. Therefore, $xu = a^+$ as the a substring of w has a length of p. Clearly, $u = a^+$ as well. Then y must equal the remaining $a^* r^j c^k$.

But if that is the case, the pumping condition $\forall i \; xu^i y \in M$ could not hold. Changing the value of i would result in a different length of a symbols in the string. This does not follow the requirements of language M that $i = 2j - k$ where $a^i r^j c^k \in M$. Therefore, language M does not have the pumping property and, as such, is not a regular language.

Part (b)

Write down the production rules of a context-free grammar that generates exactly M and justify your answer.

Solution

To design production rules to generate exactly M, we must first consider the requirements of the language. Words in M follow the rule $w = a^i r^j c^k$ where $i = 2j - k$. If we rearrange this to $2j = i + k$, you can notice that for every

r character in a word, there must be two of either of the other characters in the alphabet.

It is helpful to now consider how patterns may form between words in M from this. To follow the above rule, a word must keep this equation 'balanced'. If another r character is added, a combination of two a and c characters must be added to compensate. To capture this behaviour in context-free grammar, we will split the additional r characters added between prepending or appending to the previously added r characters. This allows us to compensate for the r character with two equal characters.

To do this, the starting symbol $\langle S\rangle$ is split into two recursive symbols, $\langle A\rangle$ and $\langle C\rangle$. The $\langle A\rangle$ symbol continuously prepends two a characters to a prepended r. The $\langle C\rangle$ symbol continuously appends two c characters to an appended r. As we compensate each r character with two alternative characters, we must add an additional production rule to cover when there are an odd number of a and c characters.

(1) $\langle S\rangle \to \langle A\rangle\langle C\rangle$

(2) $\langle S\rangle \to a\langle A\rangle r\langle C\rangle c$

These recursive symbols work by either continuing the chain or inserting an empty string.

(3) $\langle A\rangle \to aa\langle A\rangle r$

(4) $\langle A\rangle \to \epsilon$

(5) $\langle C\rangle \to r\langle C\rangle cc$

(6) $\langle C\rangle \to \epsilon$

These six production rules cover the language M. Any number of $aa\langle A\rangle r$ or $r\langle C\rangle cc$ iterations does not unbalance $i = 2j - k$. Additionally, the alternative starting symbol production rule offsets these iterations by the three characters arc, which gives the initially missed words that contain an odd number of a and odd number of c characters.