CSC236 Winter 2020 Assignment #3

Isaac Ng
ngisaac1

April 2, 2020

1. grep and many other software implementations of regular expressions include the question mark , '?', as a special symbol which marks the preceding expression as optional. For example, the regular expression dog(gy)? matches the strings 'dog' and 'doggy'.

   Let $\mathcal{REQ}$ be an extension of our familiar language of regular expressions with the question mark operator added. We will formally define the set $\mathcal{REQ}$ by extending the definition of $\mathcal{RE}$ (definition 7.6 in the Vassos course notes) to add the following induction step: If $R \in \mathcal{REQ}$, then $(R)? \in \mathcal{REQ}$.

   (a) Definition 7.7 in the Vassos course notes is a recursive definition of the language denoted by a regular expression $R \in \mathcal{RE}$. Give an extended version of this definition for $\mathcal{REQ}$.

   Solution:
   Language $\mathcal{L}(R)$ is defined by structural induction on R:
   Basis: If R is of $\mathcal{REQ}$, then either $R = \emptyset, R = \epsilon, R = a$ for some $a \in \Sigma$. For each of these cases, we define $\mathcal{L}(R)$:

   $\mathcal{L}(\emptyset) = \emptyset$ (the empty language, consisting of no strings)
   $\mathcal{L}(\epsilon) = \{\epsilon\}$ (the language consisting of just the empty string)
   $\forall a \in \Sigma, \mathcal{L}(a) = \{a\}$ (the language consisting of just the one-symbol string a)

   Induction Step: If R is of $\mathcal{REQ}$, then either $R = (S + T), R = (ST), R = S^*, R = (S)?$ for some regular expressions ($\mathcal{REQ}$) S and T, where we can assume that $\mathcal{L}(S), \mathcal{L}(T)$ have been defined, inductively. For each of the four cases, we define $\mathcal{L}(R)$:

   $\mathcal{L}((S + T)) = \mathcal{L}(S) \cup \mathcal{L}(T)$
   $\mathcal{L}((ST) = \mathcal{L}(S) \circ \mathcal{L}(T)$

$\mathcal{L}(S^*) = (\mathcal{L}(S))^*$
$\mathcal{L}((S)?) = \mathcal{L}(\epsilon) \cup \mathcal{L}(S)$

(b) Show that $\mathcal{REQ}$ has no more expressive power than $\mathcal{RE}$, by proving the following statement: $\forall R_1 \in \mathcal{REQ}, \exists R_2 \in \mathcal{RE}, \mathcal{L}(R_2) = \mathcal{L}(R_1)$. Your proof should use structural induction.

Solution:
Let $R_1 \in \mathcal{REQ}$. I want to show that there is some $R_2 \in \mathcal{RE}$ that is equivalent to R_1
I want to prove this with structural induction.

Basis: $R_1 = \emptyset, R_1 = \epsilon, R_1 = a$ for some $a \in \Sigma$
Let $R_2 = R_1$ for each case.
From the definition of $\mathcal{RE}$ as defined in Vassos course notes Definition 7.7, it is clear that $R_2 \in \mathcal{RE}$.
Since $R_2 = R_1$, it follows that $\mathcal{L}(R_2) = \mathcal{L}(R_1)$.
I have proven my basis.

Inductive Step: $R_1 = (S + T), R_1 = (ST), R_1 = S^*, R_1 = (S)?$ for some regular expressions ($\mathcal{REQ}$) S and T, where we can assume that $\mathcal{L}(S), \mathcal{L}(T)$ have been proven to be in both $\mathcal{REQ}$ and $\mathcal{RE}$, inductively. For each of the four cases, we define R_2:

Case 1: $R_1 = (S + T), R_1 = (ST), R_1 = S^*$
I define $R_2 = R_1$.
Since S and T are both in $\mathcal{RE}$ by my assumption, it follows that all three cases have $R_2 \in \mathcal{RE}$ by definition of regular expression as defined structurally in Vassos Course Notes Definition 7.7.
Since $R_2 = R_1$, it follows that $\mathcal{L}(R_2) = \mathcal{L}(R_1)$.
I have proven what I want for this case.

Case 2: $R_1 = (S)?$
This means that R_1 is either empty, or equal to S: $\mathcal{L}(R_1) = \mathcal{L}(\epsilon) \cup \mathcal{L}(S)$
I define $R_2 = (S + T)$ where T is the language with empty strings: $\mathcal{L}(T) = \{\epsilon\} = \mathcal{L}(\epsilon)$.
By definition of Vassos course notes Definition 7.7, $R_2 \in \mathcal{RE}$
This means that R_2 is either empty or equal to S: $\mathcal{L}(R_2) = \mathcal{L}(T) \cup \mathcal{L}(S) = \mathcal{L}(\epsilon) \cup \mathcal{L}(S)$
As we can see, R_1 and R_2 define the same Regular Expression and

share the same language.
Therefore, I have shown what I want for this case.

Through structural induction, I have proven that all regular expressions in $\mathcal{REQ}$ can be written as a regular expression in $\mathcal{RE}$.

2. Given a DFSA $M = (Q, \Sigma, \delta, s, F)$, we will say that M is **frumious** if the following is true:

   $\forall a \in \Sigma, \exists q_1 \in Q, \forall q_2 \in Q, \delta(q_2, a) = q_1$

   (a) Give a short English description of what it means for a DFSA to be frumious.

   Solution:
   For each letter, a, in the given alphabet, there is some state, q, in which all other states go to q when a is input.

   (b) If M is frumious, what can we say about the language accepted by M, $\mathcal{L}(M)$?

   Solution:
   Informally, this tells us that the state reached by the machine for any string x is determined entirely by the last input/alphabet of string x.
   Thus, we can say that $\forall a \in \Sigma, (\Sigma^* a) \subset \mathcal{L}(M)$ or $(\Sigma^* a) \cap \mathcal{L}(M) = \emptyset$.
   In other words, either all strings ending with a given letter are in the language or none of them are.

   (c) How many distinct languages over the alphabet $\{0, 1\}$ can be recognized by frumious DFSAs? Briefly explain your answer.

   Solution:
   All languages but the empty language and the empty set over the alphabet $\{0, 1\}$ can be recognized by frumious DFSAs. If I choose M to be frumious with the set of accepting states F being both q_0 and q_1, that is strings ending in 0 and strings ending in 1, then all languages (with the exception of the empty language and the empty set) would be recognized by frumious language M. The empty language and the empty set are not well defined, since they do not necessarily end in 0 or 1, which means that they would not be recognized. If the machine starts at an empty state q_s corresponding to s and this

is included to F, then these two languages can be recognized by M, meaning all languages could be recognized.

3. Suppose L is an infinite regular language. Does it follow that there exists a finite language S such that $L = SS^*$? If yes, prove it. If no, find a counterexample language L and prove that it cannot be formed this way.

Solution:
No, I want to prove this statement by contradiction.

Let L be an infinite regular language.
I choose L to be the language on alphabet $\{a, b\}$ where the last alphabet is b and the rest is a.
Some examples of this language would be $b, ab, aab, aaab$.
I assume there exists some finite language S such that $L = SS^*$
Since S^* Kleene star concatenates zero or more strings from S, I can write $S^* = S^*S$.
Therefore, can write $L = SS^*S = LS$

Case 1: S can be the empty string (S contains an or operation involving the empty string)
This allows for $L = LS = L$ because S can be empty.
However, this means that $L \subset SS^*$ but not $L = SS^*$.
Notice that $L = SS^*$ by our assumption. This means that S must also have the possibility to not be empty. (Because L is not the empty string)
So, this means that some strings in $SS*$ will be equivalent but not all strings.
If we have S to be non-empty, then $L \neq LS = SS^*S = SS^*$.
This is because the language L contains only one 'b' at the end while LS will have a 'b' in the middle of the string.

Case 2: S cannot be the empty string
This means that LS will contain the letter 'b' within the string rather than ONLY at the end; L ends with b and S is non-empty.
Therefore, $L \neq LS = SS^*S = SS^*$.

I have shown by contradiction, that for an infinite regular language L, there is not always a finite language S such that $L \neq SS^*$.