CSE331 Automata and Computability

Lecture 3.1: Defining Regular Expressions

Presenter: Azwad Anjum Islam (AAI) *Scribe: Mujtahid Al-Islam Akon (AKO)*

In this lecture, you will learn the formal definition and the properties of regular expression.

We learnt from the previous lecture that-

- Informally, any expression that you can create using OR (|), Concatenation (.) and Closure (*) operators are called regular expression.
- Any string that matches the regular expression must be a member of the associated regular language.
- Every string of the regular language must match with the regular expression.

For example, the expression $(a|b)^*ab$ | ϵ | $baab$ is regular based on the informal definition described above as it contains some symbols (a, b, ϵ) along with only OR, concatenation and closure operators. But this definition is somewhat vague as well as not formal.

Formal Definition of Regular Expression

The definition of Regular Expression is **Inductive**. That means we need to establish a couple of **base statements**, and then define the whole class of regular expression by ***growing*** the definition based on those statements.

1. **Base statements:**
   - The empty set $\boldsymbol{\phi}$ is a regular expression; its language is $\{\}$
   - $\boldsymbol{\epsilon}$ itself is a regular expression; its language is $\{\epsilon\}$
   - Any symbol $\boldsymbol{a}$ from the alphabet $\boldsymbol{\Sigma}$ is a regular expression; its language is $\{a\}$
2. **The Inductive Definition:** If $Rex1$ and $Rex2$ are two regular expressions, then –
   - $\boldsymbol{Rex1|Rex2}$ is a regular expression
   - $\boldsymbol{Rex1 \cdot Rex2}$ is a regular expression

   Again, if $\boldsymbol{Rex}$ is a regular expression, then –
   - $\boldsymbol{Rex^*}$ is a regular expression
   - $(\boldsymbol{Rex})$ is a regular expression (i.e. Putting brackets does not change a regular expression.)

Question 1

$\boldsymbol{(a|b)^*ab}$ | $\boldsymbol{\epsilon}$ | $\boldsymbol{baab}$ - Is this a regular expression?

Answer 1

Observe that it has three parts separated by OR (|). $(a|b)^*ab$; ϵ; and $baab$.

- **Base statements:**
    - $\boldsymbol{\epsilon}$ is a regular expression.
    - $\boldsymbol{a}$ is a regular expression
    - $\boldsymbol{b}$ is a regular expression
- **Part 1 proof**: $(\boldsymbol{a}|\boldsymbol{b})^*\boldsymbol{ab}$

  $\boldsymbol{a}$ is a regular expression. (base case)
  $\boldsymbol{b}$ is a regular expression. (base case)
  ⇒ $\boldsymbol{a}|\boldsymbol{b}$ is a regular expression. (OR)
  ⇒ $(\boldsymbol{a}|\boldsymbol{b})$ is a regular expression. (brackets cannot modify a regular expression)
  ⇒ $(\boldsymbol{a}|\boldsymbol{b})^*$ is a regular expression. ($Kleene\ Closure$)
  ⇒ $(\boldsymbol{a}|\boldsymbol{b})^*\boldsymbol{a}$ is a regular expression. ($Concatenation$)
  ⇒ $(\boldsymbol{a}|\boldsymbol{b})^*\boldsymbol{ab}$ is a regular expression. ($Concatenation$)
- **Part 2 proof:** $\boldsymbol{\epsilon}$
  $\boldsymbol{\epsilon}$ is a regular expression (base case)
- **Part 3 proof:** $\boldsymbol{baab}$
  $\boldsymbol{b}$ is a regular expression. (base case)
  $\boldsymbol{a}$ is a regular expression. (base case)
  ⇒ $\boldsymbol{ba}$ is a regular expression. ($Concatenation$)
  ⇒ $\boldsymbol{baa}$ is a regular expression. ($Concatenation$)
  ⇒ $\boldsymbol{baab}$ is a regular expression. ($Concatenation$)
- **Finally:** $(\boldsymbol{a}|\boldsymbol{b})^*\boldsymbol{ab}$ | $\boldsymbol{\epsilon}$ | $\boldsymbol{baab}$ (Part 1, Part 2 and Part 3 are ORed together)

So, $(a|b)^*ab$ | ϵ | $baab$ is a regular expression.

Question 2

$\boldsymbol{a^2b^4|b^2a^4}$ - Is this a regular expression?

Answer 2

This is basically a short form of $aabbbb$ | $bbaaaa$. As Only $concatenation$ as well as OR are present, from the discussion above, this is a RegEx.

Question 3

$\boldsymbol{a^m b^n c^k}$- Is this a regular expression where $\boldsymbol{m}, \boldsymbol{k} \geq \mathbf{0}, \boldsymbol{n} \geq \mathbf{1}$?

Answer 3

If we check the conditions of $\boldsymbol{m}, \boldsymbol{n}$ & $\boldsymbol{k}$**,** we shall understand that this expression is nothing but this- $\boldsymbol{a^*b} + \boldsymbol{c^*}$. Now, this is obvious that the above one is a RegEx.

Question 4

$\boldsymbol{a^m b^{2m}}$- Is this a regular expression where $\boldsymbol{m} \geq \boldsymbol{0}$?

Answer 4

This one is not regular. Because, there is no way to ensure the relationship between the number of $\boldsymbol{a}$ and $\boldsymbol{b}$**.** This just say that the number of $\boldsymbol{b}$ should be preceded by exactly twice the number of $\boldsymbol{a}$. This kind of relation is not possible in RegEx.

Shorthand Notations

Besides the above three basic operators, there are some shorthand notations widely used in RegEx.

- **? notation:** ? means **zero or one occurrence.** So, $r?$ simply means 0 or 1 occurrence of $\boldsymbol{r}$ where r can be a symbol or a RegEx. In short, $\boldsymbol{r? = r|\epsilon}$**.**
  For example- $ab?c$ is a RegEx which means $a(b|\epsilon)c = \{abc, ac\}$
- **Character Classes:** Sometimes it is convenient to express a lot of related sequential range of characters or symbols using some shorthand notations. The most common such notations even have some common names in practice. Few of them follows-
  - **Digit class:** This is written as $[0-9]$ which is the shorthand for the regular expression- $0|1|2|3|4|5|6|7|8|9$**.** This is often denoted & replaced by the keyword **Digit**. This refers to *any single digit from 0 to 9*.
    Note: Conventionally, a square bracket [ ] means just **any single symbol** picked from what appears inside the bracket.
  - **Alphabet class:** This is written as $[a-zA-Z]$ which is the shorthand for the regular expression- $a|b| \ldots |y|z|A|B| \ldots |Y|Z$. This is often denoted & replaced by the keyword **Alphabet**. This refers to *any single character from either a to z (small letters English) from A to Z (capital letters in English)*.

CSE331	Automata and Computability

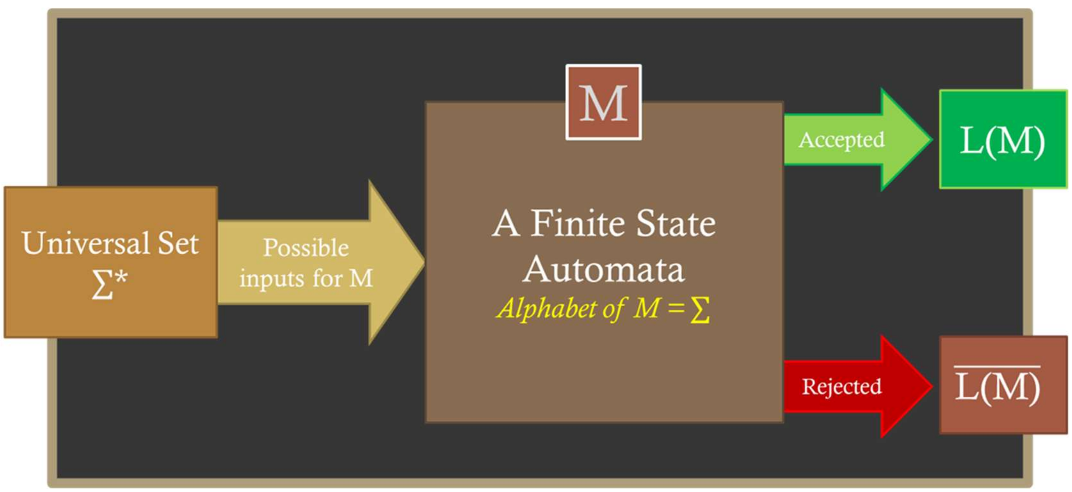
Lecture 3.2: Properties of Regular Expressions

Presenter: Azwad Anjum Islam (AAI) *Scribe: Mujtahid Al-Islam Akon (AKO)*

In this lecture, you will learn some properties of regular language as well as regular expression.

Properties of regular language

1. The **union** of any two regular languages is also a regular language. i.e. if L_1 and L_2 are two regular languages, this property follows that $L_1 \cup L_2$ must be a regular language too.
   Proof: Obvious by definition. (recall the previous lecture)
2. The **complement** of a regular language is also a regular language. i.e. if L is a regular language, then this property dictates that the complement of that language, $\overline{L}$ must be a regular language too.
   Proof: We can use the concept of an abstract finite state machine to prove it.



Recall the above this diagram of a finite automaton that we have seen earlier. Let, this automaton recognizes the regular language $\boldsymbol{L}$. In the above diagram-

- M = the name of the machine
- $\Sigma =$ the alphabet set for language L
- $L(M) =$ the set of strings **accepted** by the machine M
- $\overline{L(M)} =$ the set of strings that the machine M **rejects**.

Now, consider another finite state machine built on top of the above machine. Let's call this new machine **Arsenal**!

- we have built **Arsenal** following that logic that whenever M accepts something, **Arsenal** is going to reject that and vice versa.

- **Arsenal** is obviously a finite state machine because we have just added one more step after the finite state machine M i.e. whatever the original machine M produces as output, **Arsenal** will **reverse** it**.**
- In short, Arsenal will accept only those strings rejected by the machine M. So, the language of this **Arsenal** is nothing but $\overline{L(M)}$.

$(\boldsymbol{Proved})$

Prove/disprove it by yourself: Is the **intersection** of two regular languages regular?

Properties of Regular expressions

Let, R, M, N are some regular expressions. Now,

- OR operation is commutative but $Concatenation$ is not.
  So, $R + M = M + R$
  However, $R \cdot M \neq M \cdot R$.
- Both OR and $Concatenation$ obeys the associative law.
  So, $(R + M) + N = R + (M + N)$
  Also, $(RM)N = R(MN)$
- The identity of $\boldsymbol{OR}$ is empty language $\emptyset$. So, $\emptyset + R = R + \emptyset = R$
- The identity of $\boldsymbol{Concatenation}$ is empty string ϵ. So, $\epsilon R = R\epsilon = R$
- The annihilator of $\boldsymbol{Concatenation}$ is empty language ϕ. So, $\phi R = R\phi = \phi$
- OR follows the idempotent law. So, $R + R = R$
- Regular expressions obey the distributive law of $Concatenation$. On the contrary, it is not true for OR.
  So, $(M + N)R = MR + NR$
  and, $R(M + N) = RM + RN$
  However, $MN + R \neq (M + R)(N + R)$
  and, $R + MN \neq (R + M)(R + N)$
- **Some identities:**
  - $R^* = (R^*)^*$
  - $\emptyset^* = \epsilon$
  - $\epsilon^* = \epsilon$
  - $R^*R^* = R^*$
  - $R^*R = R^+$
  - $(M + N)^* = (M^*N^*)^* = (M^* + N^*)^*$

Is the intersection of two regular languages regular?

Proof using De Morgan's law:

Let, L & M are two regular language.

From the properties of regular language, we get,

$\Rightarrow \bar{L}$ & $\bar{M}$ are regular.

$\Rightarrow \bar{L} \cup \bar{M}$ is regular

$\Rightarrow \overline{\bar{L} \cup \bar{M}}$ is regular.

$\therefore$ $\boldsymbol{L \cap M}$ **is regular**. [$\because$ From Morgan's law, $\overline{x \cup y} = \bar{x} \cap \bar{y}$]

$(\boldsymbol{Proved})$

CSE331	Automata and Computability

Lecture 4.1: Constructing Regular Expressions - Part 1

Presenter: Azwad Anjum Islam (AAI) — *Scribe: Mujtahid Al-Islam Akon (AKO)*

In this lecture, you will learn to construct regular expressions through some examples.

$\boxed{\boldsymbol{Problem\ 1}}$

Let, $\boldsymbol{L}$ be a language, where L is the set of **all binary strings**. Is L a regular language? If yes, what is its regular expression?

$\boxed{Answer}$

A valid binary string means **one or more occurrences** of 0 or 1. Recall that one or more occurrences means positive closure. So, the result should be $(0|1)^+$.

As we have found a regular expression, the language must be regular.

$\boxed{\boldsymbol{Problem\ 2}}$

Let, $L = \{a, aa, aaa, aaaa, aaaaa, \ldots, ab, abab, ababab, abababab, abababab, \ldots\}$

Is L a regular language? If yes, what is its regular expression?

$\boxed{Answer}$

$L = \{a, aa, aaa, aaaa, aaaaa, \ldots, ab, abab, ababab, abababab, abababab, \ldots\}$

Notice that the strings of this language consists of either one or more occurrences of $\boldsymbol{a}$ **or** one or more occurrences of ab.

one or more occurrences of $a = a^+$

one or more occurrences of $ab = (ab)^+$

$\boldsymbol{OR}$ing them, we get the result, $\boldsymbol{a}^+|(\boldsymbol{ab})^+$

As we have found a regular expression, the language must be regular.

For the following examples, assume the alphabet set, $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$.

$\boxed{\boldsymbol{Problem\ 3}}$

What if L was the language of all binary strings that **starts with $\mathbf{0}$**?

$\boxed{Answer}$

To start a binary number with 0, we need a zero first. This zero should be followed by any binary string of length ranging from zero to anything. Binary string with zero or any length means $(0|1)$*. After prepending a 0, we get the result, $\mathbf{0}(0|1)$*

$\boxed{\boldsymbol{Problem\ 4}}$

What about the language of all binary strings that ends with two consecutive $\mathbf{1}$s?

$\boxed{Answer}$

A RegEx for any binary string that can be generated including empty string (why?) is $(0|1)$*. Appending two 1s we get the desired result, $(0|1)$*$\mathbf{11}$.

To be continued …

CSE331	Automata and Computability

Lecture 4.2: Constructing Regular Expressions - Part 2

Presenter: Azwad Anjum Islam (AAI) *Scribe: Mujtahid Al-Islam Akon (AKO)*

In this lecture, we shall continue learning to construct regular expressions through some more examples.

For the following examples, assume the alphabet set, $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$

$\boxed{\boldsymbol{Problem\ 5}}$

Construct a regular expression for all binary strings that **contain the substring $\mathbf{1011}$**.

$\boxed{Answer}$

If we add any binary string before **or** after $\mathbf{1011}$, it will give the desired binary string recognized by the language. We already know that <u>*any binary string*</u> of length 0 or more is $(0|1)$*.

So, the accepting pattern should be, <u>*any binary string*</u> 1011 <u>*any binary string*</u>. Replacing <u>*any binary string*</u> part with $(0|1)$*, we get the result, $(0|1)$*$\mathbf{1011}(0|1)$*.

$\boxed{\boldsymbol{Problem\ 6}}$

Construct a regular expression for all binary strings that **contain the subsequence $\mathbf{1011}$**.

$\boxed{Answer}$

From the concept of subsequence, we understand that, <u>***any binary string***</u> part (mentioned in the previous example) can even appear in each side of each symbol of the string, $\mathbf{1011}$.

So, the required format should be,
***"<u>any binary string</u>* 1 <u>*any binary string*</u> 0 <u>*any binary string*</u> 1 <u>*any binary string*</u> 1 <u>*any binary string*</u>"**.
So, the result is $(0|1)$* $\mathbf{1}$ $(0|1)$* $\mathbf{0}$ $(0|1)$* $\mathbf{1}$ $(0|1)$* $\mathbf{1}$ $(0|1)$*

N.B. The following solution is also correct. Can you find out why?

$(0^*)\mathbf{1}(1^*)\mathbf{0}(0^*)1(0^*)\mathbf{1}(0|1)^*$

$\boxed{\textbf{\textit{Problem 7}}}$

Construct a regular expression for all binary strings **that do NOT contain consecutive $\mathbf{1}$s**.

$\boxed{\textit{Answer}}$

You have already known that $(0|1)^*$ generates all possible strings with $\mathbf{0}$ & $\mathbf{1}$

i.e. $\{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\}$

1. Notice that the above RegEx would work well for this problem except the strings marked red are problematic as they contain more than **one** consecutive 1's. To solve this issue, what you can do is replacing the **1** with **10**.
   After applying this idea, we get,
   $(0|\mathbf{10})^* = \{\epsilon, 0, 10, 00, 010, 100, \mathbf{1010}, 000, 0010, 0100, \mathbf{01010}, \ldots\}$
   Notice that, **no** consecutive 1s is there anymore.
2. Now, notice again that $(0|10)^*$ could not cover some strings those should have been covered to complete the answer. It could not cover the strings where **1 is present at the end.** For example, $\{1, 01, 001, 101, \ldots\}$, all of them are valid strings and should be accepted but are missed by $(0|10)^*$. So, we need to add a new term cover them and we can do so by appending a $\mathbf{1}$ with $(0|10)^*$ i.e. $(\mathbf{0}|\mathbf{10})^*\mathbf{1}$.

Combing the two parts, we get,

$(0|10)^*$ **|** $(0|10)^*1$

$\Rightarrow (\mathbf{0}|\mathbf{10})^*(\boldsymbol{\epsilon}|\mathbf{1})$

Another correct solution is $(\boldsymbol{\epsilon}|\mathbf{1})(\mathbf{0}|\mathbf{01})^*$. Find out why by yourself.

Note: Always verify your regular expressions with some strings from the language. Don't miss the corner cases.

$\boxed{\textbf{\textit{Problem 8}}}$

Construct a regular expression for all binary strings **that do NOT contain consecutive $\mathbf{0}$s**.

$\boxed{\textit{Answer}}$

$(\mathbf{1}|\mathbf{01})^*(\boldsymbol{\epsilon}|\mathbf{0})$ **(**Similar to **Problem 7**).

$\boxed{\boldsymbol{Problem\ 9}}$

Construct a regular expression for all binary strings in which **the number of 0's is divisible by 3**.

$\boxed{Answer}$

As long as you keep getting 1s, everything is fine. But whenever a **0** appears, we need to wait for 2 more 0s to accept the string. So far, we get $(1|\mathbf{000})^*$. But these 0s do not need to come consecutively. So, we put any number of 1s around each of them.

Applying all these, finally we get $(\mathbf{1}|\mathbf{0}\ 1^*\ \mathbf{0}\ 1^*\ \mathbf{0})^*$.

$\boxed{\boldsymbol{Problem\ 10}}$

Construct a regular expression for **all mobile phone numbers of Bangladesh**.

For example: $+8801515654784$, 01911720127

$\boxed{Answer}$

Consider the following cases-

- The strings can contain either +88 or nothing. So, we get $(+88|\epsilon)$ or in short (+88)?.
- Then the string must be followed by 01**.**
- Then we need to add the operator codes. For example
  - GP: 3,7
  - Robi: 6, 8
  - Banglalink: 9
  - Teletalk: 5

  So, we need to append **(3|5|6|7|8|9)**
- At last, we need to add last 8 digits. They can be any decimal digits from 0 to 9. So, we can again use a shorthand notation $[0-9]^8$.

Now concatenating all of them we get our desired **RE,** $(+88)?\,01(3|5|6|7|8|9|)[0|9]^8$

To test by yourself how regular expressions work in real life as well as to play with it, you can explore https://regexr.com.

____