CS310 : Automata Theory 2019

Lecture 6: Regular expressions

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2019-01-16

Regular expressions

We have been using automatons to define languages.

We talk about languages without automatons, *e. g.*, programming languages.

Now let us talk about languages using some other means.

We define regular expressions to declare languages.

Topic 6.1

Regular expressions

Regular expressions : give a word

We can declare singleton languages by simply writing the member word.

Example 6.1

We write 01 *instead of writing* $\{01\}$.

More examples:

- 1
- 011101
- *all*
- ϵ

Regular expressions : empty language

We have already seen $\emptyset$ denotes the language without any element.

Regular expressions : union of languages

If we have two languages, we can describe the union of the languages using the operator '+'.

Example 6.2

For example, Consider language $\{01, 10, 0, \epsilon\}$.

$$L(\underbrace{01 + 10 + 0 + \epsilon}_{\textit{regular expression}}) = \{01, 10, 0, \epsilon\}.$$

More examples:

- $L(1 + 0) = \{1, 0\}$
- $L(011 + 101) = \{011, 101\}$
- $L(all + none) = \{all, none\}$
- $L(\epsilon + 1 + 1) = \{\epsilon, 1\}$

Regular expressions : concatenation

Definition 6.1

For regular expressions R_1 and R_2, let regular expression R_1R_2 define the language consist of words obtained by concatenating a word from L_1 with a word from L_2, e. g.,

$$L(R_1R_2) \triangleq \{ww' | w \in L(R_1) \wedge w' \in L(R_2)\}.$$

Example 6.3

For example, $L((0+1)1) = \{01, 11\}$.

Exercise 6.1

List the words in the following languages

- $L(\ (1+0)(10+11)\ ) =$
- $L(\ (1+0)\emptyset\ ) =$
- $L(\ \epsilon + (1+01)01\ ) =$

Regular expressing : arbitary concatenations

We can define concatenations of unbounded number of languages.

Definition 6.2

Let $R_1, R_2, \ldots, R_n$ be regular expressions.

$$L(R_1 R_2 \ldots R_n) \triangleq L(R_1(R_2 ... R_n))$$

Base case for $n = 2$ is already defined in the previous slide.

Exercise 6.2

- $L(\ (1+0)1(0+1)\ ) =$
- $L(\ (1+0)\epsilon(0+1)\ ) =$

Exercise 6.3

What is the value of $L(R_1 R_2 \ldots R_n)$ if $n = 1$? or $n = 0$?

Regular expressions : Kleene star

Definition 6.3
For regular expression R, let R^* define the language consist of words obtained by concatenating words from $L(R)$ some number of times, e. g.,

$$L(R^*) \triangleq \bigcup_{n \geq 0} L(\underbrace{R \dots R}_{n}).$$

Example 6.4
For example, $(01)^*$ *defines* $\{\epsilon, 01, 0101, 010101, ...\}$.

Exercise 6.4

- $01^* =$
- $(0+1)^* =$
- $(0+1)^*0 =$
- $\emptyset^* =$
- $\epsilon^* =$

Regular expressions as grammar

1. Alphabet
2. Empty language $\emptyset$
3. Union $L_1 + L_2$
4. Concatenation $L_1 L_2$
5. Kleene star L^*

We can write down the above means of constructing regular expressions as a grammar

$$L ::= \Sigma \mid \emptyset \mid L + L \mid LL \mid L^*$$

- L is being recursively defined
- $|$ separates different constructors
- Ls occuring in right hand sides are subexpressions
- Different occurrences of L are not required to be the same expressions

Example: regular expressions

Example 6.5

Let $\Sigma = \{a, b, c\}$. Regular expressions for

- *single letter words*
$$a + b + c \quad \text{or} \quad \Sigma$$
- *words that contain at least one a.*
$$\Sigma^* a \Sigma^*$$
- *words that contain at least one a or one b.*
$$\Sigma^* (a + b) \Sigma^*$$
- *words that contain at least one a and one b.*
$$\Sigma^* a \Sigma^* b \Sigma^* + \Sigma^* b \Sigma^* a \Sigma^*$$

Exercise: regular expressions

Exercise 6.5

Let $\Sigma = \{0, 1\}$. Give regular expressions for

- *words such that no prefix have the difference between the number of* 0*s and* 1*s more than 1.*
- *words such that no prefix have the difference between the number of* 0*s and* 1*s more than 2*

Precedence order

We saw the use of parentheses to make regular expressions unambiguous.

We can reduce the burden of writing parentheses by defining precedence order. The following is decreasing order of precedence.

- Kleene star
- Concatenation
- Union

Example 6.6

- $(01) + (0)^* = 01 + 0^*$
- $(10)^* \neq 10^*$
- $(01) + (10) = 01 + 10$
- $1(01) = 101$

Exercise 6.6

Drop as many parentheses as possible.

- $(01)^* =$
- $(10) + (1 + 01) =$
- $((10)1)^*(0 + 1) =$
- $((10)^*)^* =$

Topic 6.2

Regular expressions in programming languages

Practical regular expressions!

Before understanding

the importance of regular expression

in the theory of computation, we should see

the practical importance of regular expressions.

Regular expressions for pattern matching

Regular expressions are widely used to find patterns in texts.

All languages provide libraries to handle regular expressions.

We will present regular expressions in Python syntax (similar to PHP).

Matching problem

Let L be a given language and a word w.

Find all[†] longest subwords w' of w such that $w' \in L$.

[†]In case of overlapping subwords that are in L, one may be expected to report only the earlier one.

Example 6.7

Consider language $L = 01(01)^*$ *and word*

011110011110001001010101100101100100111000000000

The blue highlighted words are the longest subwords that are in L.

Commentary: Longest is an important term. Often subwords of the matched word.

Regular expressions in PHP/Python

- alphanumeric letters match with themselves
- macros for special characters, *e. g.*, `\n` for newline
- `|` denotes union
- `( )` denotes groupings
- `*` denotes zero or more times repetition
- `+` denotes one or more times repetition, *i.e.*, `L+ = LL*`
- `?` denotes zero or one time, *i.e.*, `L? = (|L)`

Example 6.8

The following are the regular expressions in PHP/Python

`0|1`	either 0 or 1
`(0|1)*`	repeated 0 or 1
`0(0|1)+1`	repeated 0 or 1 at least once
`0|1|2|3|4|5|6|7|8|9`	any numeric digit

Notational faluda : $|$ and $+$ in Python

We have already used $|$ and $+$ in our earlier notation.

However in python,

- $|$ is used to denote union instead of $+$
- $+$ is used to denote one or more repetition of words.

A lot of computer science is about keeping up with the notation.

Regular matching in action

Please open the following URL in your browser

`http://regex101.com/`

Exercise 6.7

Find the following string in the class pizza post for this lecture and paste in the second box of the webpage.

01111001111000100101010110010110010011100000000

Write regular expressions to find

- *substrings that are has only 0's and longer than 2*
- *substrings that do not contain 11.*

Regular expressions macros

Writing regular expressions can be cumbersome

the programming languages provide macros

`\d`	matches with any digit
`\w`	matches with any alphabet
`[a-s]`	matches with range of characters
`[^a-s]`	matches with characters not in the range
`.`	matches with any character except newline
`a{3,6}`	repeat a from 3 to 6 times
`^`	indicates start of string or after newline
`$`	indicates end of string or before newline

- …. dozens more (look into the right drop down and click on i)

Back to theory

Now we will get back to the study of languages.

How regular expressions relate to NFAs?

To be continued...

End of Lecture 6