CSEP501 : Compiler Construction: Homework 1

Karuna Sagar Krishna

October 14, 2023

Question 1

1a - (a | xy)*

Examples of strings that can be generated:

- ϵ (empty string)
- a
- xy
- axy
- xya
- $aaxyaxya$

Examples of strings that cannot be generated using the same alphabets:

- x
- y
- xay
- ax
- ya
- $axyax$

The regular expression describes the set of all strings consisting of zero of more occurrence of substrings a and xy in any order. Note, this includes empty string.

1b - b(oz)+o

Examples of strings that can be generated:

- *bozo*
- *bozozo*
- *bozozozo*

Examples of strings that cannot be generated using the same alphabets:

- *b*
- *o*
- *bo*
- *bz*
- *boz*
- *bozz*

The regular expression describes the set of all strings that starts with *b*, contains one or more occurrence of substring *oz* and ends with *o*.

1c - ((ϵ | 0)1)*

Examples of strings that can be generated:

- ϵ (empty string)
- 1
- 01
- 011
- 101
- 111010101111

Examples of strings that cannot be generated using the same alphabets:

- 0
- 10
- 001
- 110
- 010

- 0110

The regular expression describes the set of all strings that contains zero or more occurrence of substrings 01 and 1 in any order. Note, this includes empty string. This means that 0 cannot occur consecutively and the string cannot end with 0.

Question 2

2a

```
(a|b)*a(a|b)*a(a|b)*a(a|b)*
```

Since we need atleast 3 a's in the string, not necessarily consecutively, we add these 3 a and then we insert regular expression for any string of a and b before and after these 3 a's.

2b

```
(a|bb)*
```

Essentially, this regular expression describes strings the contains the substring a or bb zero or more times. So this includes the empty string. Further, we can pick and repeat a without any restrictions, but we pick bb substring which causes it to appear as multiples of 2 and we cannot have a in between them.

2c

```
[^aeiou]*a[^aeiou]*e[^aeiou]*i[^aeiou]*o[^aeiou]*u[^aeiou]*
```

As described in the question, first we set the vowels in order. Then we insert other possible sequences of lower case letter. Note, we need to ensure that the vowels appear exactly once and in order, so we need to exclude vowels from other possible sequences i.e. `[^aeiou]*`. We insert this other possible sequence before, after and in between the vowels resulting in the above regular expression.

Question 3

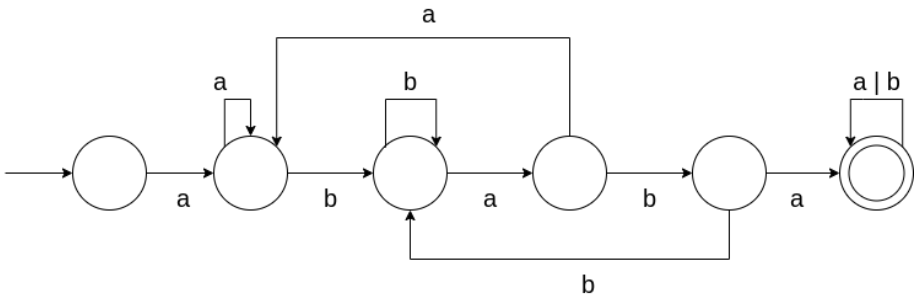
3a

```
a(a|b)*baba(a|b)*
```

The language described in question can be represented by the above regular expression. The regular expression is constructed by fixing the first character to a and a substring $baba$. Then we insert any string of a and b represented

by regular expression `(a|b)*`. Putting together these pieces leads to the above complete regular expression for the language.

The below DFA is a representation of the above regular expression. This DFS was derived using Thompson's construction, Subset construction and Hopcroft's algorithm.



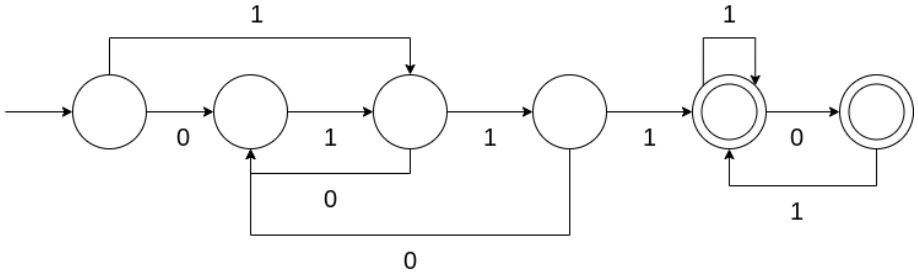
3b

`(01|1)*(0|`ϵ`)111(01|1)*(0|`ϵ`)`

To construct the regular expression, we begin with 111 substring. Then we need to find a regular expression that describes any string of 0 and 1 but does not contain 00 as a substring. If we add this regular expression before and after 111 substring, we should have a complete description of the language.

To find the regular expression that does not produce 00 substring, we can say that if we start with 0 then we need to follow it with a 1, but a 1 can stand on its own. So this leads to `(01|1)*`. However, the drawback with this is that a single 0 cannot stand on its own. So we add `0|`ϵ as suffix resulting in `(01|1)*(0|`ϵ`)`. Putting together all these pieces leads to the above complete regular expression for the language.

The below DFA is a representation of the above regular expression. This DFS was derived using Thompson's construction, Subset construction and Hopcroft's algorithm.



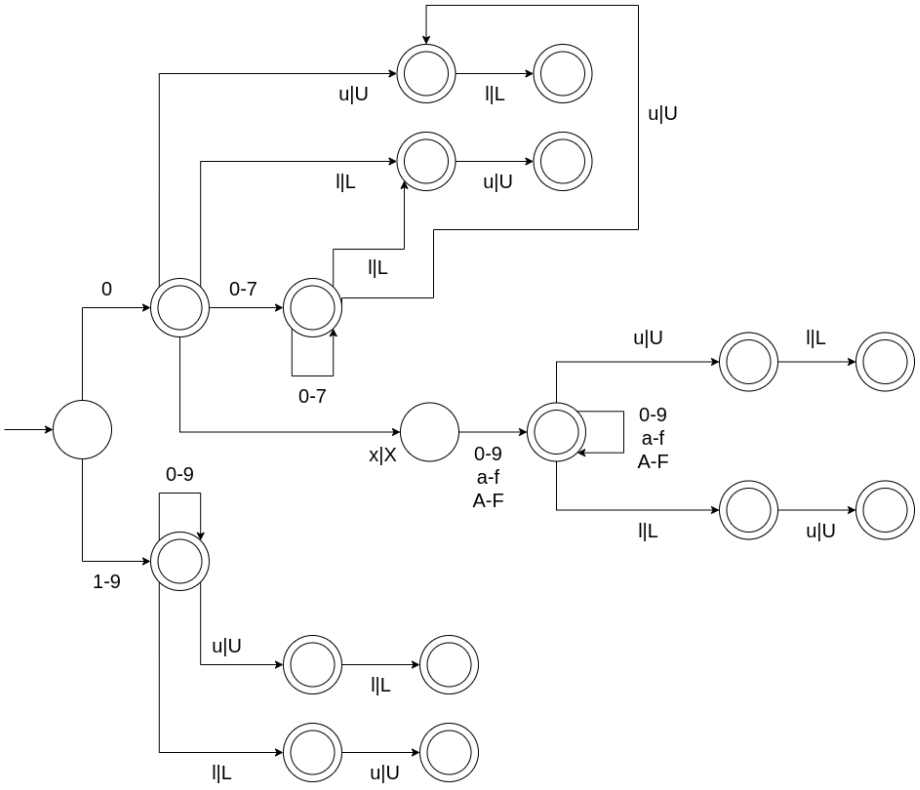
Question 4

4a

```
Octal ::= 0[0-7]*
Decimal ::= [1-9][0-9]*
Hexadecimal ::= 0[xX][0-9a-fA-F]+
Unsigned ::= u|U
Long ::= l|L
Suffix ::= (Unsigned? Long?) | (Long? Unsigned?)
IntegerConstant ::+ (Octal | Decimal | Hexadecimal) Suffix?
```

We start with the definition of octal, decimal and hexadecimal as specified in the question. The suffix (if any) for unsigned and long can occur once and in any order, so we need to consider both possible orders. Putting this together leads to the above regular expression for integer constants.

4b



Question 5

5a

```
CommentMiddle ::= ([^_])|(-[^_])|(---*[^>_])
HtmlComment ::= <!--CommentMiddle*--(-*)>
```

There is a well defined starting and ending character sequence for HTML comments. So focusing on the middle section of the comment, we form a regular expression that either allows all characters except − or allows − but not followed by − or allows −− but not followed by >. But this does not cover a single − which is a valid comment content. This is true for a multiple repetitions of − as well. To cover this, we interpret the ending character sequence to allow two or more − followed by >.

5b