Javier Lopez
1/23/23

Quiz 1 study guide

1. Suppose we have a language with only two types of tokens, <ident>, and <int_lit>. Write a regular expression for <ident> where and <ident> is allowed to be any combination of upper and lower case letters and digits that contains at least one letter. For example, 01a, and 1a0 are <ident>s..
   a. Write an RE for <ident> as described above.
   b. Assume that the REs for <int_lit> and <white_space> are the same as in the lecture/project, and <white_space>is handled as usual. Draw the DFA to recognize <ident> and <int_lit> tokens.
2. Some of the following claims about the algebraic properties of regular expressions are correct and some are not. Determine which ones are correct.

rs = sr	Concatenation is commutative	F
r | s = s | r	| is commutative	T
r | (s | t) = (r | s) | t	| is associative	T
r (st) = (rs)t	Concatenation is associative	T
r(s|t) = rs | rt; (s|t)r = sr | tr	Concatenation is distributive	T
Ɛr = rƐ = r	Ɛ is identity for concatenation	T
Ɛ | r = r | Ɛ = r	Ɛ is identity for |	F
r* = (r | Ɛ)*	Ɛ is guaranteed in closure	T
r = r***	* is idempotent	T

3. Consider the lexical specification below:
   <token> ::= <identifier> | <num lit> | <op> | <reserved>
   <num lit> ::= <nonzero digit> <digit>* | **0**
   <digit> ::= **0** | <nonzero digit>
   <nonzero digit> ::= **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**
   <identifier> ::=

```
        <identifier start> <identifier part>*
<identifier start> ::=  A .. Z | a..z | $ | _
<identifier part> ::=
       <identifier start> | <digit>
<op> ::=   + | == | *
<reserved>   ::=  if  | then | else
```

Give the correct tokenization for the following inputs. Assume white space serves as a separator, but is otherwise ignored.

   a. ___00
   b. _ _ _ 0 0
   c. If0 elsea then
   d. 12a
   e. 13then
   f. 13then0
   g. then13a0

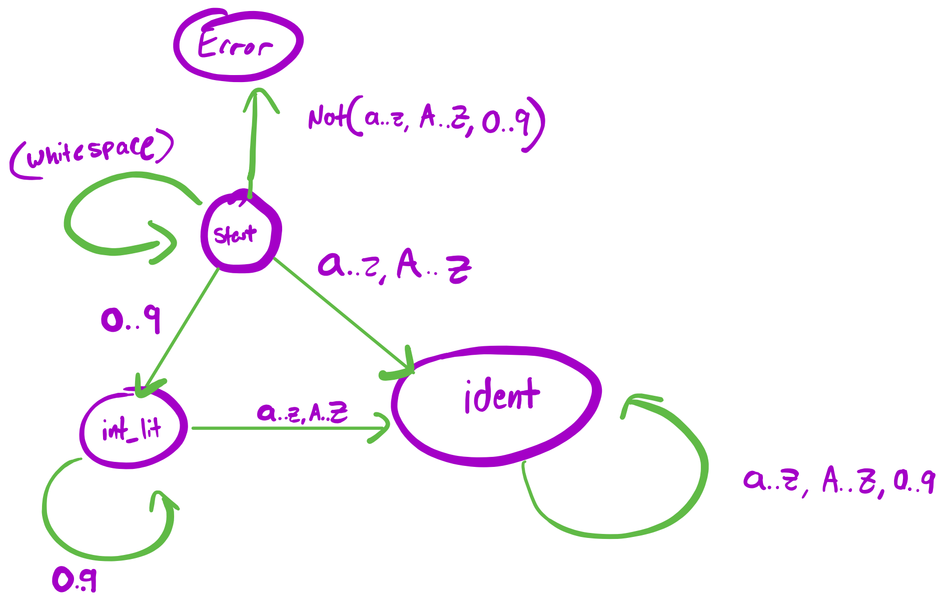
4. What is the benefit of using a Factory class rather than just creating an object directly with new? Explain how to set up a Factory.
5. What does “dynamic” mean in the context of this course
6. What does “static” mean in the context of this course
7. Give at least three differences between old Java switch statements and new switch expressions?
8. We use regular expressions to specify the lexical structure of a programming language, but changed to Context Free Grammars for specifying the phrase structure?
   a. What is the main difference between the two formalisms?
   b. What typical feature of the phrase structure of programming languages can be specified with CFGs but not REs?
9. Suppose that you want to include the following characters (including the quotes) in the input to your scanner, which should be recognized as a string literal.

   “This is a string”

   a. How can you do that in a Junit test using a regular Java String Literal?
   b. How can you do that in a Junit test using a Java text block?
10. What are the decimal ASCII values corresponding to the following character literals in java?
    a. ‘ ‘ (space)
    b. ‘\t’
    c. ‘A’
    d. ‘a’
    e. ‘0’
    f. ‘9’
    g. ‘\n’
    h. ‘\r’
11. What is the difference between the ASCII and unicode character sets?

1) a) $[(A...Z) | (a...z) | (0...9)]^* \ [(A...Z) | (a...z)] \ [(A...Z) | (a...z) | (0...9)]^*$

b)



2)
- $rs = sr$; Concatenation is commutative $\Rightarrow$ FALSE
- $r|s = s|r$; $|$ is commutative $\Rightarrow$ TRUE
- $r|(s|t) = (r|s)|t$; $|$ is associative $\Rightarrow$ TRUE
  - $\hookrightarrow$ $r|s \,|\, r|t = r|t \,|\, s|t$
  - $(r,s,t) = (r,t,s)$
- $r(st) = (rs)t$; Concatenation is associative $\Rightarrow$ TRUE
- $r(s|t) = rs|rt$; $(s|t)r = sr|tr$; Concat. is distributive $\Rightarrow$ TRUE
- $\varepsilon r = r\varepsilon = r$; ε is identity for concatenation $\Rightarrow$ TRUE
- $\varepsilon|r = r|\varepsilon = r$; ε is identity for $|$ $\Rightarrow$ FALSE
- $r^* = (r|\varepsilon)^*$; ε is guaranteed in closure $\Rightarrow$ TRUE
- $r^{**} = r^*$; * is idempotent $\Rightarrow$ TRUE

3) a) $\underbrace{\_\_\_\_00}_{\text{ident}}$ ⇒ 1 token : <identifier>

b) $\underbrace{\_}_{\text{iden}}\ \underbrace{\_}_{\text{iden}}\ \underbrace{\_}_{\text{iden}}\ \underbrace{0}_{\text{num lit}}\ \underbrace{0}_{\text{num lit}}$ ⇒ 5 token : <identifier><identifier><identifier> (num lit> (num lit>

c) $\underbrace{\text{if0}}_{\text{ident}}\ \underbrace{\text{elsea}}_{\text{ident}}\ \underbrace{\text{then}}_{\text{reserved}}$ ⇒ 3 token : <identifier><identifier><reserved>

d) $\underbrace{12}_{\text{num lit}}\underbrace{a}_{\text{identifier}}$ ⇒ 2 tokens : <num lit> <idenifier>

e) $\underbrace{13}_{\text{num lit}}\underbrace{\text{then}}_{\text{reserved}}$ ⇒ 2 tokens : (num lit> <reserved>

f) $\underbrace{13}_{\text{num lit}}\underbrace{\text{then0}}_{\text{ident}}$ ⇒ 2 tokens: (num lit)<identifier>

g) $\underbrace{\text{then13a0}}_{\text{ident}}$ ⇒ 1 token: <identifier>

4) The benefit of the factory class as rather than making a new class is that a factory is an an interface or abstract class with whose intent is on creating objects that let subclasses decided what classes are inherited. Instead of creating a "new" instantiation of a class, factory abstracts the this in its methods so you can define an object in terms of the factory and determine which class is instantiated inside the factory itself. This reduces the need to change several words when chasing classes by only requiring a change in the factory itself

You set up a factory as you would a normal class and create a method for creating a "new Instantiation

Ex:
```
class XFactory {
    static IX makeX() {
        return  new X1();
    }
}
```

5) Dynamic means during runtime

6) Static means before runtime

7)

- Switch statements don't return anything, expression return values
- Statements require "break" to differentiate cases, expressions don't
- In switch statements, cases could only have one possible value at a time to determine behavior, expressions can have multiple possible values per each new case

8) a. RE’s cannot incorporate recursion/nested statements while CFG’s can

b. Recursive bahavior and balancing parentheses

9)

a) Java String literal:

```
@Test void
singleCharTokensWithWhiteSpace() throws LexicalException{
IScanner scanner = CompilerComponentFactory.makeScanner("\”This is a string\””);

{Arbitrary code here}
 }
```

b) Java Text Block:

```
@Test void
singleCharTokensWithWhiteSpace() throws LexicalException{
IScanner scanner = CompilerComponentFactory.makeScanner(“””
”This is a string” “””);

{Arbitrary code here}
 }
```

10) a) ‘ ’ space ⇒ 32

b) ‘\t’ ⇒ 9

c) ‘A’ ⇒ 65

d) ‘a’ ⇒ 97

e) ‘0’ ⇒ 48

F) ‘9’ ⇒ 57

g) ‘\n’ ⇒ 10

h) ‘\r’ ⇒ 13

11)

ASCII represents letters (a-z), uppercase letters (A-Z), digits (0-9), and symbols (punctuation) while Unicode has the same thing whilst also representing letters of English, Arabic, Greek etc