Compilers: Week 2

Ian Quah (itq)

June 15, 2017

Topic 1

Lexical Analysis - Introduction

1. String → Lexical analyzer → (⟨class, string⟩**a.k.a** Token) → parser
2. **E.g**: foo = 42 → <ID, "FOO" > < OP, "="> < Int,"42">
3. (a) **Operators**
   (b) Whitespace
   (c) Keywords
   (d) Identifiers
   (e) Numbers
   (f) (- open parens
   (g)) - closed parens
   (h) ; - semi-colon

   The last 3 are often token classes of their own

Topic 2

Lexical Analysis - Examples

1. **Lookahead**

   Necessary to disambiguate certain cases: might be the case that only much later in the program two cases are different.

   Necessary to decide where one token ends and next begins.

   Always necessary but we want to bound it as it increases complexity like crazy
2. **Goal**

   Partition input string into lexemes

   identify the token of each lexeme
3. Left-to-right scan makes lookahead necessary

Topic 3

Lexical Analysis - Regular Languages

1. Regular Languages:
   (a) Generally use Regexp to define them
   (b) Specifies what set of languages are in a token class
   (c) Single char:
   'c' = { "c"}: a language containing just the single character "c"
   (d)
   (e) ϵ
   ϵ = {""}: the language containing the empty string. Is **NOT** null set
   (f) union
   A + B = {a | a ∈ A} ∪ {b | b ∈ B}
   (g) Concat
   AB = {ab | a ∈ A ∧ b ∈ B }
   (h) Iteration a.k.a Kleene closure
   $A^* = \cup_{i \geq 0} A^i$ A with itself i times.
   $A^0 == \epsilon$
2. Regular expressions over Σ are the smallest set of expressions including:
   R = ϵ
   | 'c' s.t c ∈ Σ
   | R + R
   | RR
   | R*
   All this is known as a grammar
3. Consider a language with the alphabet 0, 1 s.t: Σ = {0, 1}
   (a) 1^* = "", 1, 11, 111,....
   (b) (1 + 0) 1 = {ab — a ∈ ∧ b ∈ B} = {11, 10}
   (c) $(1 + 0)^*$ 1 = "", 0 + 1, (1 + 0)(1 + 0), ... Σ^*
   Σ^* is the set of all strings you can form out of the alphabet

Topic 4

Lexical Analysis - Formal Languages

1. **Definition: Formal Language**

   Let Σ be a set of characters(an alphabet)

   A language over Σ is a set of strings drawn from that alphabet

   difference between formal, regular, language, expression

2. **Definition: Meaning Function, L** Using the regexp notation from earlier:

   (a) L: expr $\rightarrow$ Sets of Strings
   (b) L('c') = { "c"}
   (c) L(ϵ) = {""}
   (d) union $L(A + B) = \{a \mid a \in L(A)\} \cup \{b \mid b \in L(B)\}$
   (e) Concat $L(AB) = \{ab \mid a \in L(A) \wedge b \in L(B)\}$
   (f) Iteration a.k.a Kleene closure $L(A^*) = \cup_{i \geq 0} L(A^i)$

   Why is L important?

   (a) More E than M: many ways to express same meaning
   (b) Makes clear what is syntax and what is semantics : important (consider how Roman numerals and current number system have same meaning, but different notation)

   This allows us to separate the two and consider different ways of representation, allowing us to explore different notations

Topic 5

Lexical Analysis - Lexical Specifications

1. Strings
   (a) if... then... else...
   (b) **Two equivalent forms** (concat then union)
   'i' 'f' + 'e' 'l' 's' 'e'
   'if' + 'else'
2. Digits
   (a) digit = '0' + '1' + ... + '9' Let digits refer to regexp containing all above
   - Refers to single digit
   (b) digit*
   - denotes almost all digits, except the first letter could be "",
   - to fix: digit digit* or digit+ (based on standard regexp)
3. Identifiers
   (a) Strings of letters or digits starting with an letter
   letter = 'a' + 'b' + ... + 'z' + 'A' + ... + 'Z' == [a-zA-Z]
   (b) Identifier: letter(letter + digit)*
4. Whitespace
   (a) non-empty sequence of blanks, newlines, and tabs
   (b) blank : ' '
   (c) newlines : '\n'
   (d) tabs : '\t'

Constructing our lexer: Lexical Specification

1. Write a rexp for the lexemes of each token class

Token Class	Lexeme
Number	digit$^+$
keyword	"if" + "else"
identifier	letter(letter + digit)*
OpenPar	"("

2. Construct R, matching all lexems for all tokens

$$R = Keyword + identifier + Number + ... \quad (1)$$
$$= R_1 + R_2 + ... \quad (2)$$

Which is the union of all the lexems from Step 1

Topic 5 continued on next page. . .

3. Let input be x_1, ... x_n

$$\text{For} i \leq i \leq n : \tag{3}$$
$$if(x_1, ..., x_i \in L(R)) : \text{remove} x_1, ...x_i \tag{4}$$
$$\tag{5}$$

   (a) Maximal munch:
      i. for some i $\neq$ j, and valid (x_1,...x_i) and valid (x_1, ..., x_j) which do we take?
      ii. We take max(i, j)
   (b) Which token?
      i. Recall R = R_1 + ... + R_n
      ii. given some valid string (x_1, ..., x_n) s.t $\in$ L(R_i) and $\in$ L(R_j)
      iii. typically just order s.t when pattern matching we choose the one listed first. **Priority ordering**
      iv. E.g "if" is both in the language of keywords and identifiers, but we order it s.t keywords is first
   (c) What if no rule matches?
      i. x_1, ..., x_i $\notin$ L(R) ??
      ii. Define an Error string, s.t [all strings that are not matched]
      iii. Has lowest priority in **priority ordering**

Topic 6

Finite Automata

1. Can be seen as analagous to langages, where depending on the string, we can say if it's in the language or not
2. note: final state has two circles. Does not have self loop because if we get a string that repeats last character, it might not be in the language (which self loop would imply)
3. **Termination**
   (a) String is finished, but not in accepting state
   (b) Is in a "dead end"

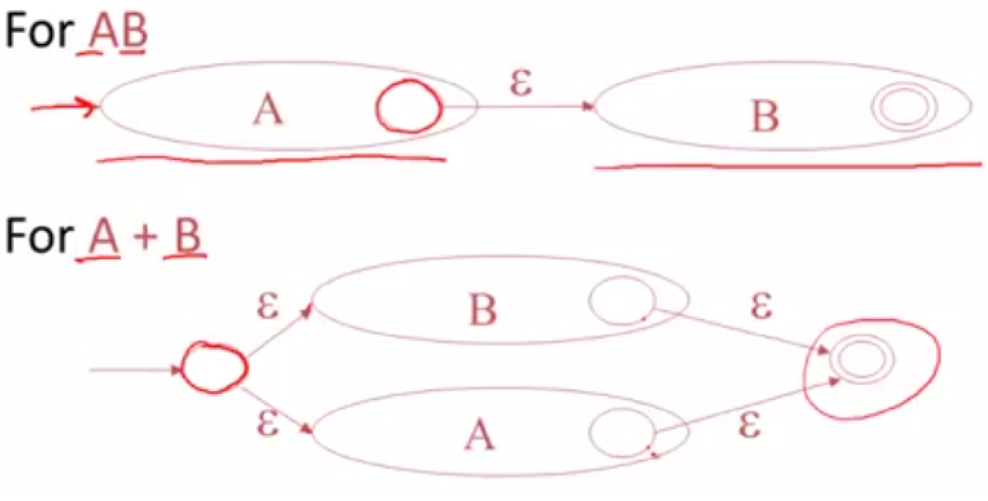
Topic 7

Regexps into NFAs

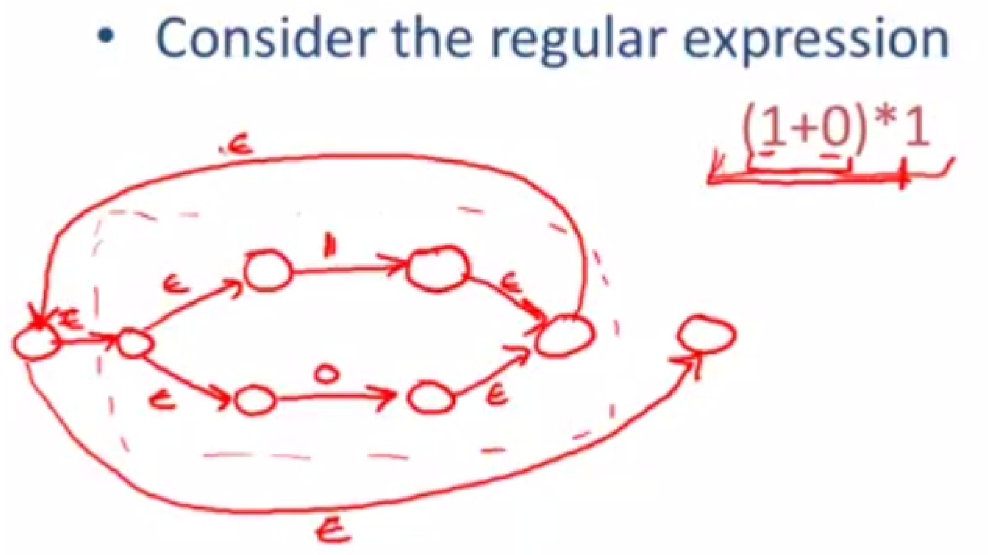
1. NFA = non-deterministic finite automaton
2. For each kind of rexp, build a NFA
   (a) ϵ: input $\rightarrow^{\epsilon}$ (fin)
   (b) A: input $\rightarrow^{A}$ A(fin)
   (c) AB: input $\rightarrow^{A}$ A(not fin) $\rightarrow^{\epsilon}$ B(fin)

(d) A + B: ϵ either to A or B then ϵ to fin

(e) AB and A+B as an NFA



(f) **Example:** (1 + 0)*1



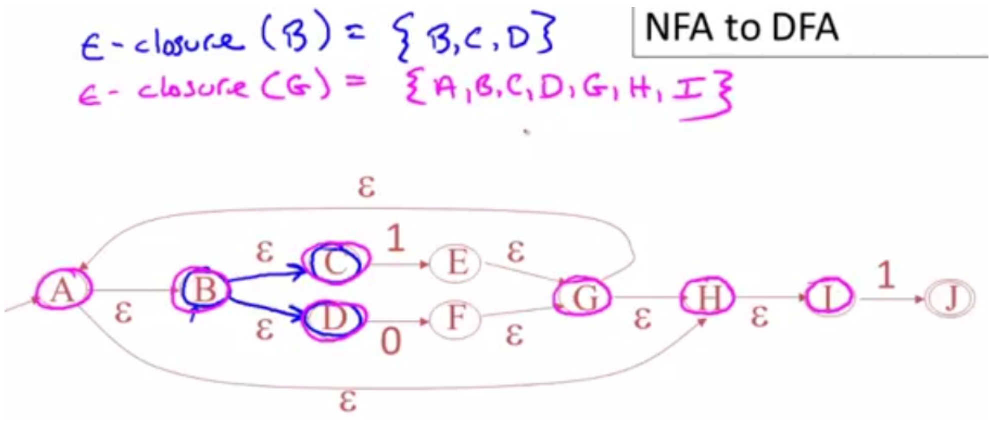
Topic 8

NFA to DFA

ϵ-closure

1. Defn: all states reachable by consuming an ϵ from current state.

   Example


ϵ-closure(B) = {B, C, D}
ϵ-closure(G) = {A, B, C, D, G, H, I}
NFA to DFA
A
B
C
D
E
F
G
H
I
J
ε
1
0

NFA Details

1. NFA may be in many states at once because non-deterministic
2. How many states?

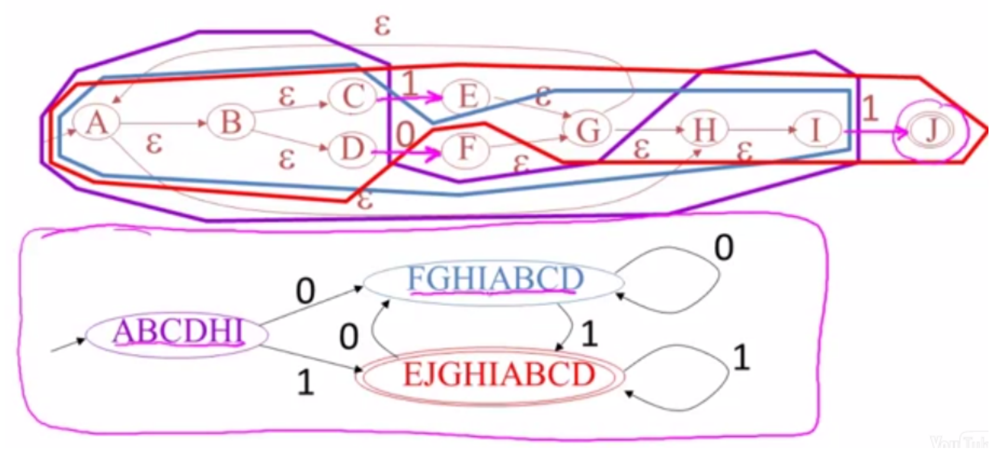
   N states

   $|S| \leq$ N
3. How many subsets?

   2^N - 1 subsets May be many, but is *finite*
4. Comparison between NFA and DFA

Automata	NFA	DFA
States	S	subsets of S (except empty set)
Start	s $\in$ S	ϵ-closure(s)
Final	F $\subset$ S	$\{X \mid X \cap F \neq \emptyset\}$
Transition function	$a(X) = \{y \mid x \in X \wedge X \to^a Y\}$	$X \to^a Y$ if $Y = \epsilon$-closure(a(X))

5. **NFA to DFA**
   (a) Begin with the start state, then determine the set of states in the ϵ closure
   (b) Consider all possible transitions from the "blob", and generate new blobs
   (c) Enumerate all the blobs
   (d) Finish



The reason we defined the final state that way is that we don't care about any specific final state, we just want to get to a final state.

Topic 9

Quiz Hints

1. First ignore the ϵ then find the number of possible strings. Then, consider with 1, 2, 3 and 4 epsilons.
2. Consider each "level" to be another rule where it goes from level to level. Note, it does not loop on itself so if it's not fully eaten up by the end it's not accepted - this is in line with what happens if you try to construct a DFA
3. Remember priority. Also, when they say tokenize here, they mean tokenize the entire thing (so the output of tokenization is the same as the input)
4. Just trace it
5. I considered the start state to be 1 state as well
6. Remember + means 1 or more and * means 0 or more
7. -
8. -
9. -
10. I honestly don't remember how I did this without just working it out and spending a lot of time on this, sorry
11. Read the definition
12. Read the definition (the name itself is helpful here: non-deterministic)