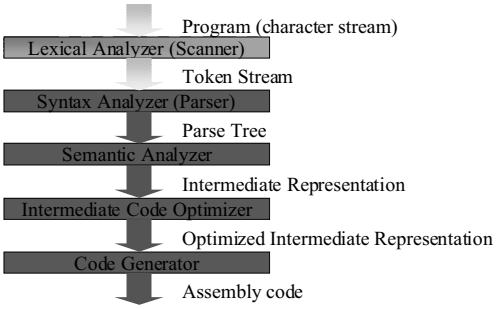
6.035

Fall 2000

Lecture 2: Lexical Analysis

Regular Expressions, NFA's, DFA's and Scanner Generation

Lexical Analysis



What is a Lexical Analyzer?

Source program text ⟹ Tokens

- Example of Tokens
  - Operators = + - > ({ := == <>
  - Keywords if while for int double
  - Numeric literals 43 6.035 -3.6e10 0x13F3A
  - Character literals 'a' '~' '\''
  - String literals "6.891" "Fall 98" "\"\" = empty"
- Example of non-tokens
  - White space space(' ') tab('\t') end-of-line('\n')
  - Comments /*this is not a token*/

Lexical Analyzer needs to….

f	o	r		v	a	r	1		=		1	0			v	a	r	1	<	=	

for ID("var1") eq_op Num(10) ID("var1") leq_op

- Precisely separate the text stream into the correct stream of tokens
  - ID("var1") not ID("var") Num(1)
  - ID("var1") leq_op not ID("var1<=")
- Identify the type of token that matches the input string
  - 10 Num(10)
  - var1 ID("var1")

Lexical Analyzer needs to….

f	o	r		v	a	r	1		=		1	0			v	a	r	1	<	=	

for ID("var1") eq_op Num(10) ID("var1") leq_op

- Describe different types of tokens in different languages
  - FORTRAN DO I=1,10
  - C++ for(int i=1; I<= 10; I++)
  - C-shell foreach i (1 2 3 4 5 6 7 8 9 10)

Lexical Analyzer needs to….

- Use *regular expressions* to precisely describe what strings each type of token can recognize

Examples of Regular Expressions

Regular Expression	Strings matched
a	"a"
a · b	"ab"
a | b	"a" "b"
_	""
a*	"" "a" "aa" "aaa"

Examples of Regular Expressions

Regular Expression	Strings matched
num = 0|1|2|3|4|5|6|7|8|9	"0" "1" "2" "3" …
posint = *num* · *num**	"8" "6035" …
int = (_ | -) · *posint*	"-42" "1024" …
real = *int* · (_ | (. · *posint*))	"-12.56" "12" "1.414"...

14

QUESTION

- Why *posint* = *num* · *num** ?
  Why not *posint* = *num** ?

Few Additional Notations

- "one or more occurrence of" r+ = r · r*
- "zero or one occurrence of" r? = r | _

18

QUESTION

- What regular expression best identifies MIT course numbers?

  num = 0|1|2|3|4|5|6|7|8|9

  1) *class* = *num* · *num**
  2) *class* = *num* · . · *num**
  3) *class* = *num* | . | *num**
  4) *class* = (*num* · . · *num*)*

QUESTION

- What regular expression will match all numbers 0 to 256 and nothing else?

Definition: Formal Languages

- Alphabet _ = finite set of symbols
  - _ = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- String s = finite sequence of symbols from the alphabet
  - s = 6004
- Empty string _ = special string of length zero
- Language L = set of strings over an alphabet
  - L = { 6001, 6002, 6003, 6004, 6035 6891 … }

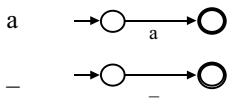
Definition: Regular Expressions

- For a regular expression r, the language L(r) = { all the strings that match r }
  - L((a | _) · b) = {"ab" "b"}
- Suppose r and s are regular expression denoting languages L(r) and L(s)
  - L(r | s) = L(r) U L(s)
  - L(r · s) = { xy | x ∈ L(r) and y ∈ L(s) }
  - L(r*) = { $x_1 x_2 ... x_k$ | x_i ∈ L(r) and k >= 0 }
  - L(_) = {}

More Regular Expressions

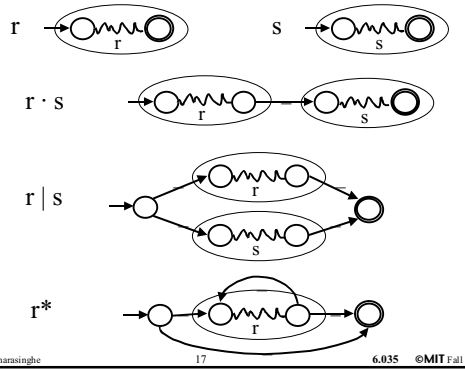
- We know:
  - L(r | s) is the **union** of L(r) and L(s)
  - L(r · s) is the **concatenation** of L(r) and L(s)
  - L(r*) is the **Kleene closure** of L(r)
    - "zero or more occurrence of"

Constructing a NFA for a regular expression

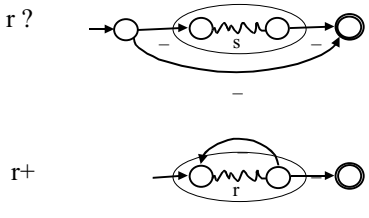


If r and s are regular expressions with the NFA's

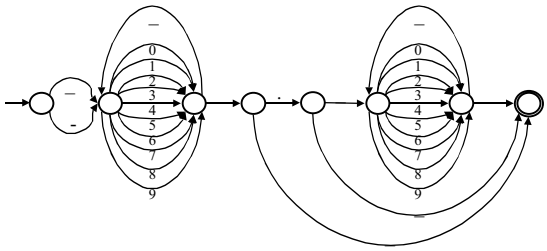
Constructing a NFA for a regular expression



Constructing a NFA for a regular expression

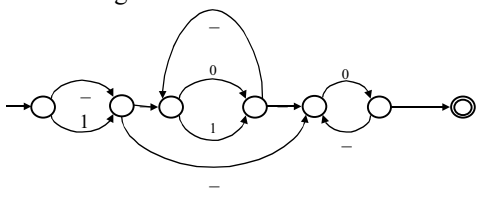


(-| _) ·(0|1|2|3|4|5|6|7|8|9)+ · (. ·(0|1|2|3|4|5|6|7|8|9)*)?

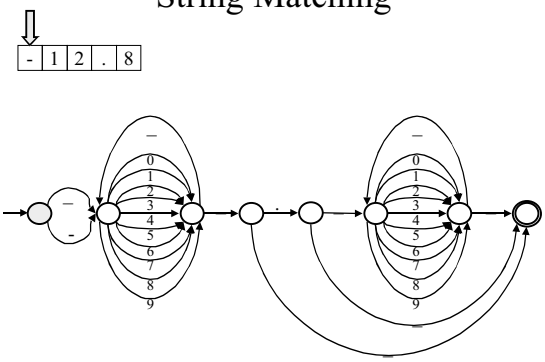


QUESTION

- What is the regular expression matched by the following NFA?



String Matching

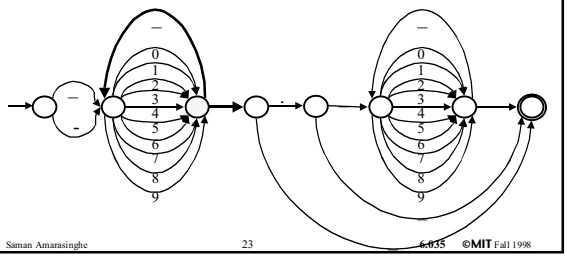


Implementing a Lexical Analyzer

- Need to find which strings match the regular expressions
- Create a NFA for to match the regular expression
- Unfortunately, NFA does not have a simple implementation

Non-deterministic

- Two possible choices
- Need to know about the future
  - No simple algorithm can pick the right path



Implementing a Lexical Analyzer

- Need to find which strings match the regular expressions
- Create a NFA for to match the regular expression
- Unfortunately, NFA does not have a simple implementation
- Need to create a Deterministic Finite Automaton (DFA) From a NFA

Constructing a DFA from a NFA

- Why do we need a DFA?
  - Easy to implement
  - Current state + input symbol uniquely identifies the next state
- How do you construct a DFA from a NFA?

1. Closure

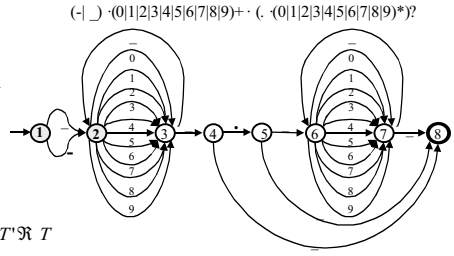
- The closure of a state is the set of states that can be reached from that state without consuming any of the input
  - Closure(S) is the smallest set T such that

$$T = S \cup \bigcup_{s \in T} edge(s, \varepsilon)$$

- Algorithm

$T \leftarrow S$
repeat
  $T' \leftarrow T$
  $T \leftarrow T' \cup \bigcup_{s \in T'} edge(s, \varepsilon)$
until $T = T'$

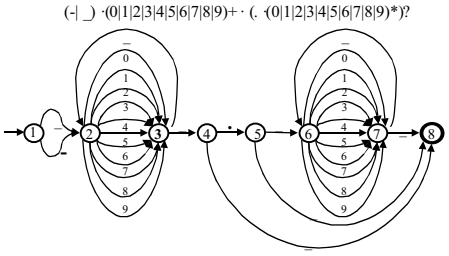
S = {1}
T = {1, 2}
T' = {1, 2}



$T \leftarrow S$
repeat
  $T' \leftarrow T$
  $T \leftarrow T' \cup \bigcup_{s \in T'} edge(s, \varepsilon)$
until $T = T'$

Question: What is closure(3)?

S = {3}
T = ??



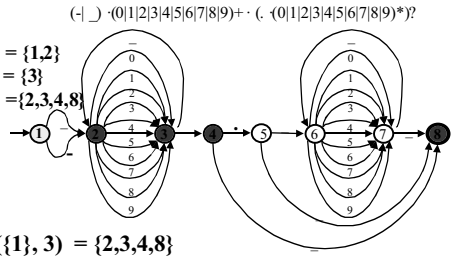
2. DFAedge

- Given a symbol and a state, what states can you reach?

$$DFAedge(d, c) = closure\left(\bigcup_{s \in d} edge(s, c)\right)$$

What is DFAedge({1}, 3)?

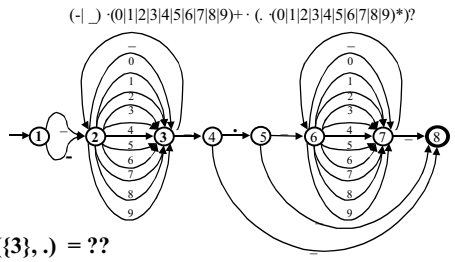
d = {1, 2}
closure({1}) = {1,2}
edge({2}, 3) = {3}
closure({3}) ={2,3,4,8}



DFAedge({1}, 3) = {2,3,4,8}

Question: What is DFAedge({3}, .)?

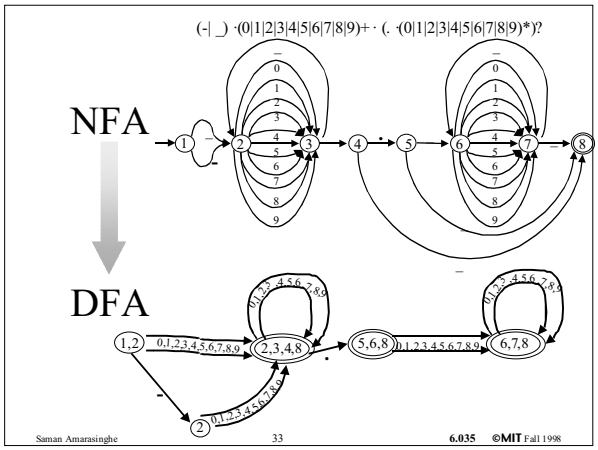
d = {3}



DFAedge({3}, .) = ??

3. DFA

```
states[0] ← I q
states[1] ← closure(bg)
p ← 1
j ← 0
while j ≤ p
  foreach c ∈ Σ
    e ← DFAedge(states[j], ch)
    if e = states[i] for some i ≤ p
    then
      trans[j,c] ← i
    else
      p ← p+1
      states[p] ← e
      trans[j,c] ← p
    j ← j+1
```

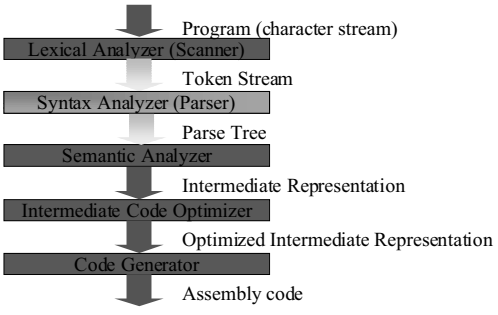
In practice

- Uses automated tools to construct a lexical analyzer
  - Given a set of tokens defined using regular expressions, the tools will generate a character stream tokenizer by constructing a DFA
- Common scanner generator tools
  - lex in C
  - JLex in java
- Tonight, will talk about how to use JLex

Summary

- Lexical analyzer create tokens out of a text stream
- Tokens are defined using regular expressions
- Regular expressions can be mapped to Nondeterministic Finite Automatons (NFA)
  - by simple construction
- NFA is transformed to a DFA
  - Transformation algorithm
  - Executing a DFA is straightforward

Next 3 Lectures



Recitation on Programming and Collaboration Tools

- **TONIGHT!**
- From 7:30PM to 9:30PM
- Room: HERE!
- Will Discuss
  - CVS
  - Make
  - Athena Java tools
  - 6.035 locker
  - JLEX for the 1st assignment

Roadmap

Monday 9/4	Tuesday 9/5	Wednesday 9/6 **Room 3-370** L1: Intro to compilers, course admin. info., lexical analysis	Thursday 9/7 **Room 3-270** **L2: TODAY** **7:30 – 9:30**
Monday 9/11	Tuesday 9/12 **Room 3-270** **L3: Syntax analysis, bottom-up parsing** **Next Lecture**	Wednesday 9/13 **Room 3-370** L4: LR(0) Paring Algorithms and Parsing Tables	Thursday 9/14 **Room 3-270** L5: LR(1) & LALR(1) Paring Algorithms

Scanner Segment assigned:

Project due on 9/11