LEX8: Lexical Analyzer

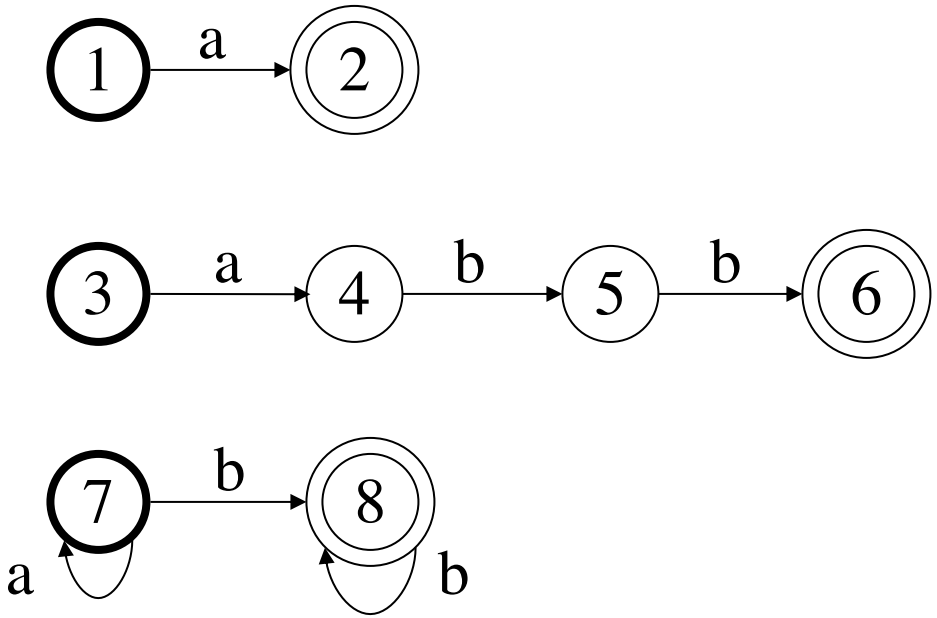
Lexical Analysis

CMPT 379: Compilers

Instructor: Anoop Sarkar

anoopsarkar.github.io/compilers-class

Lexical Analysis using NFAs

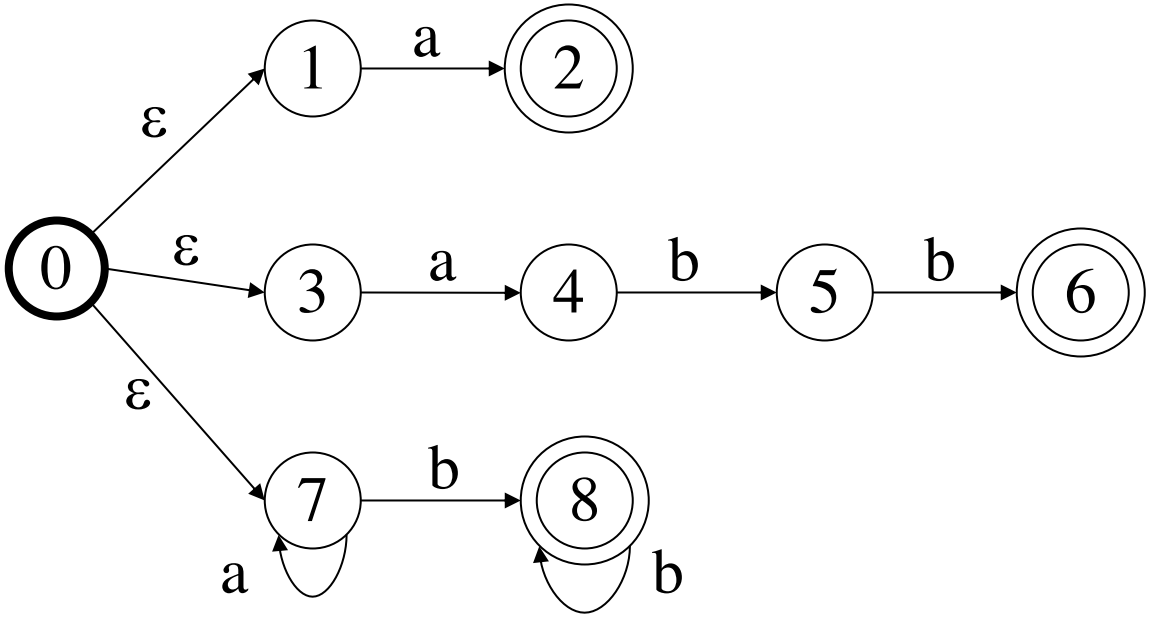


TOKEN_A = a

TOKEN_B = abb

TOKEN_C = a*b+

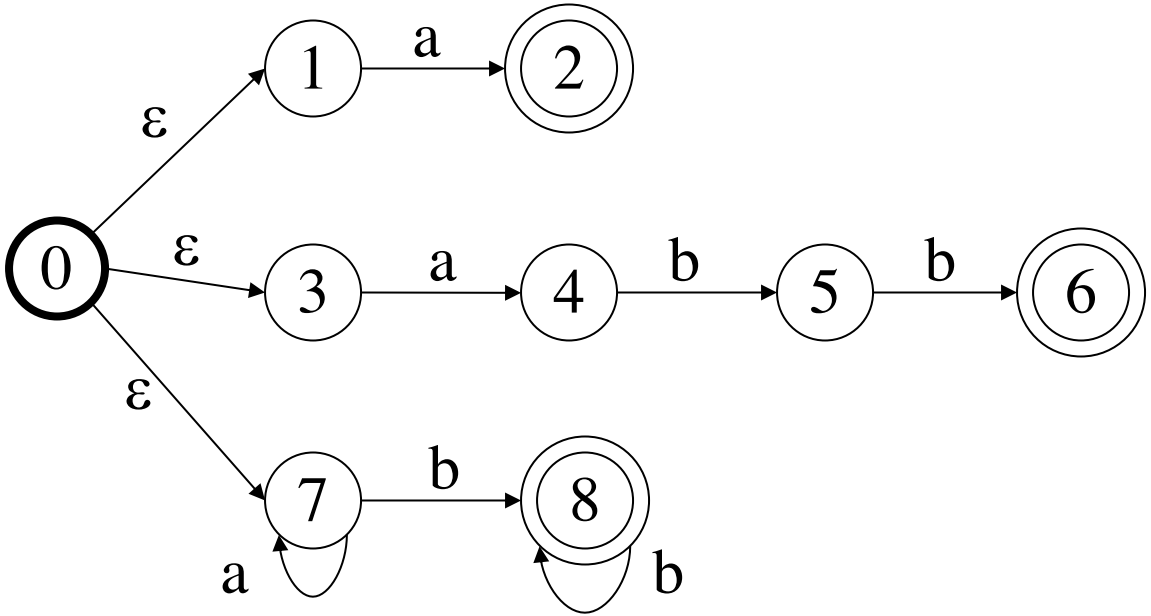
Lexical Analysis using NFAs



TOKEN_A = a

TOKEN_B = abb

TOKEN_C = a*b+



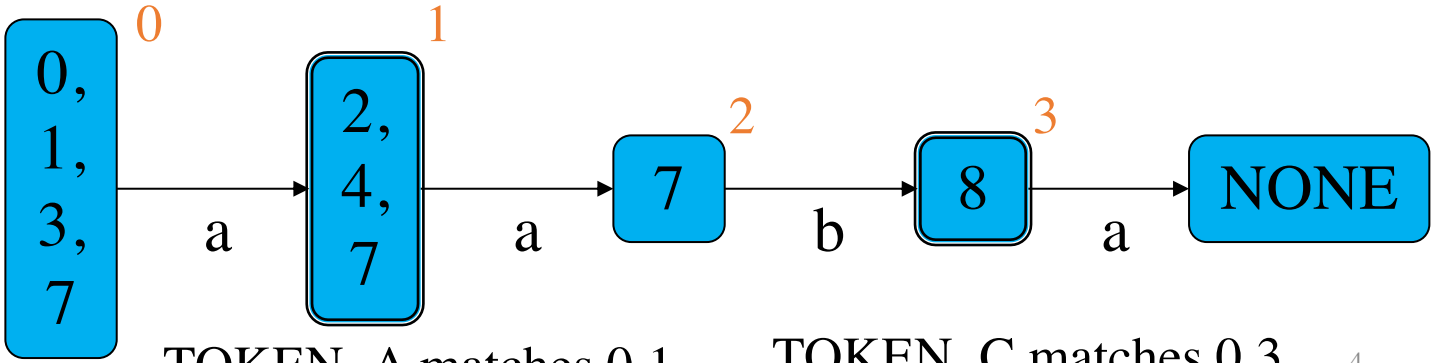
TOKEN_A = a

TOKEN_B = abb

TOKEN_C = a*b+

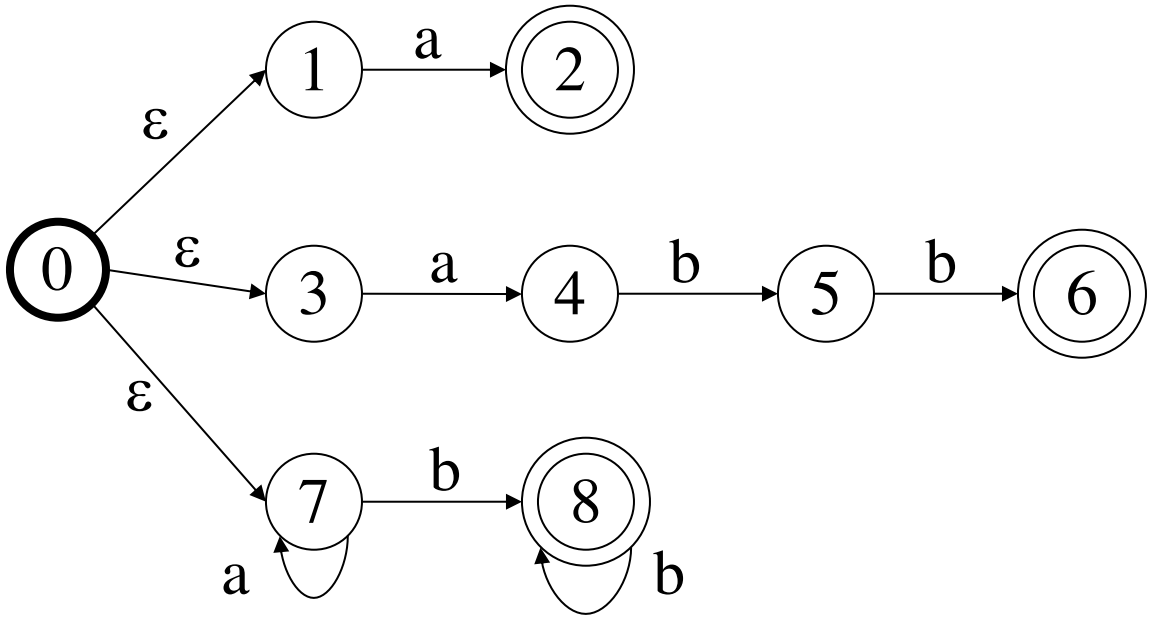
Input: aaba

$_0a_1a_2b_3a_4$



TOKEN_A matches 0,1

TOKEN_C matches 0,3



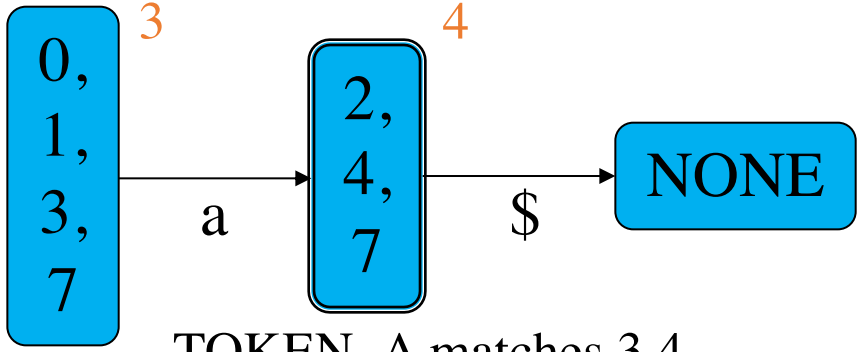
TOKEN_A = a

TOKEN_B = abb

TOKEN_C = a*b+

Input: aaba

$_0a_1a_2b_3a_4$



TOKEN_A matches 3,4

Output:
TOKEN_C aab [0,3]
TOKEN_A a [3,4]

Lexical Analyzer using DFAs

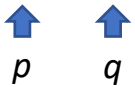
- Each token is defined using a regexp r_i
- Merge all regexps into one big regexp
  - $R = (r_1 \mid r_2 \mid \ldots \mid r_n)$
- Convert *R* to an NFA, then DFA, then minimize
  - remember original NFA final states with each DFA state

Lexical Analyzer using DFAs

- The DFA recognizer must find the *longest leftmost match* for a token
  - continue matching and report the last final state reached once DFA simulation cannot continue
  - e.g. longest match: <*print*> and not <*pr*>, <*int*>
  - e.g. leftmost match: for input string *aabaaaaab* the regexp a^+b will match *aab* and not *aaaaab*
- If two patterns match the same token, pick the one that was listed earlier in R
  - e.g. prefer final state (in the original NFA) of r_2 over r_3

Lookahead operator

- Implementing r_1/r_2: match r_1 when followed by string in r_2
- e.g. `a*b+/a*c` accepts a string `bac` (token=b) but not `abd`
- The lexical analyzer matches $r_1 \varepsilon r_2$ up to position *q* in the input



- But remembers the position *p* in the input for the r_1 match
- Reset and start from position *p* for next token

TOKEN_A = (ab)*a

TOKEN_B = (ab)*a(ca)*

TOKEN_C = bab(bab)*

TOKEN_D = a*ba(ba)*

Q: Use the ordered token definitions shown here and provide the tokenized output for the input string *abacababababa* using the greedy longest match lexical analysis method.

Summary

- Token $\Rightarrow$ Pattern
- Pattern $\Rightarrow$ Regular Expression
- Regular Expression $\Rightarrow$ NFA
  - Thompson's Rules
- NFA $\Rightarrow$ DFA
  - Subset construction
- DFA $\Rightarrow$ minimal DFA
  - Minimization

$\Rightarrow$ **Lexical Analyzer (multiple patterns)**

Extra Slides

TOKEN_A = (ab)*a

TOKEN_B = (ab)*a(ca)*

TOKEN_C = bab(bab)*

TOKEN_D = a*ba(ba)*

Q: Use the ordered token definitions shown here and provide the tokenized output for the input string *abacababababa* using the greedy longest match lexical analysis method.

A:
TOKEN_B (abaca)
TOKEN_D (bababa)