LEX8: Lexical Analyzer

Lexical Analysis

CMPT 379: Compilers

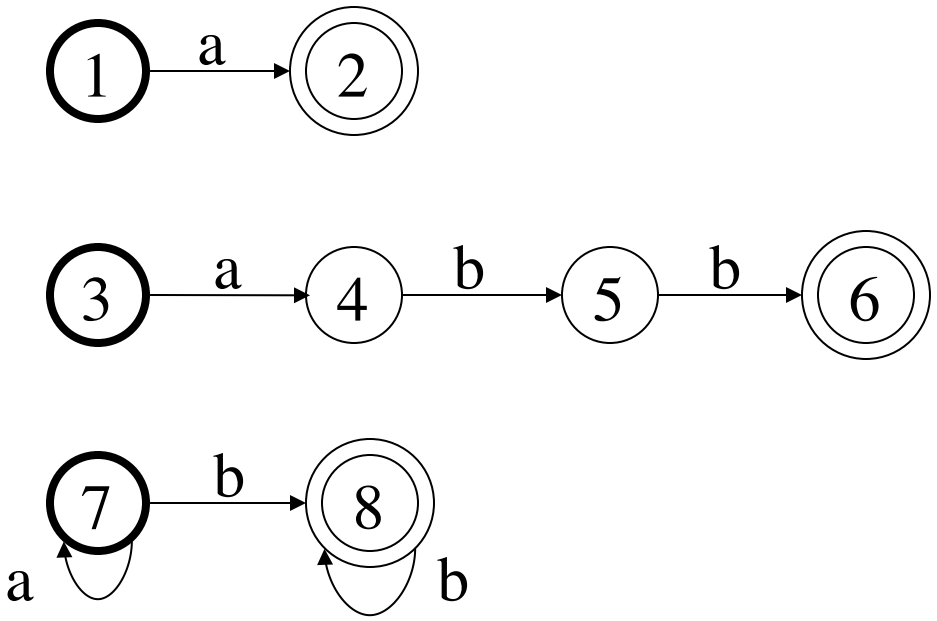
Instructor: Anoop Sarkar

anoopsarkar.github.io/compilers-class

Lexical Analyzer using NFAs

- For each token convert its regexp into a DFA or NFA
- Create a new start state and create a transition on ε to the start state of the automaton for each token
- For input $i_1, i_2, \ldots, i_n$ run NFA simulation which returns some final states (each final state indicates a token)
- If no final state is reached then raise an error
- Pick the final state (token) that has the longest match in the input,
  - e.g. prefer DFA #8 over all others because it read the input until i_{30} and none of the other DFAs reached i_{30}
  - If two DFAs reach the same input character then pick the one that is listed first in the ordered list

Lexical Analysis using NFAs

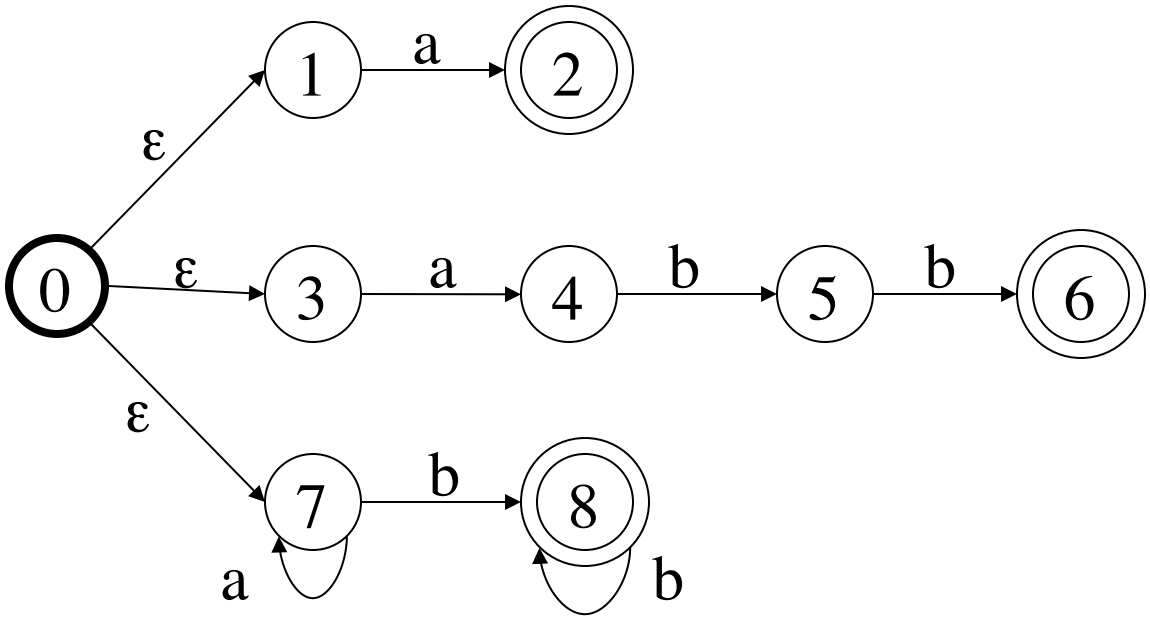


TOKEN_A = a

TOKEN_B = abb

TOKEN_C = a*b+

Lexical Analysis using NFAs



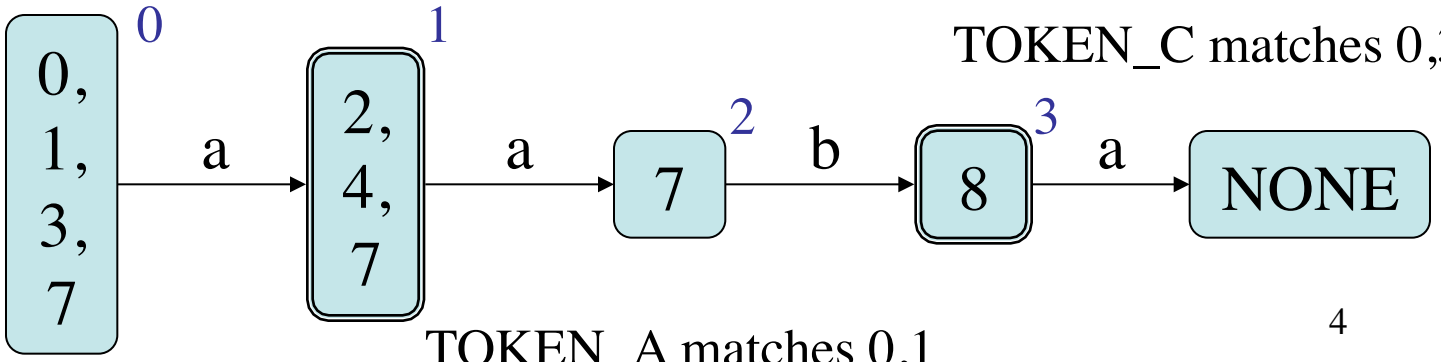
TOKEN_A = a

TOKEN_B = abb

TOKEN_C = a*b+

Input: aaba

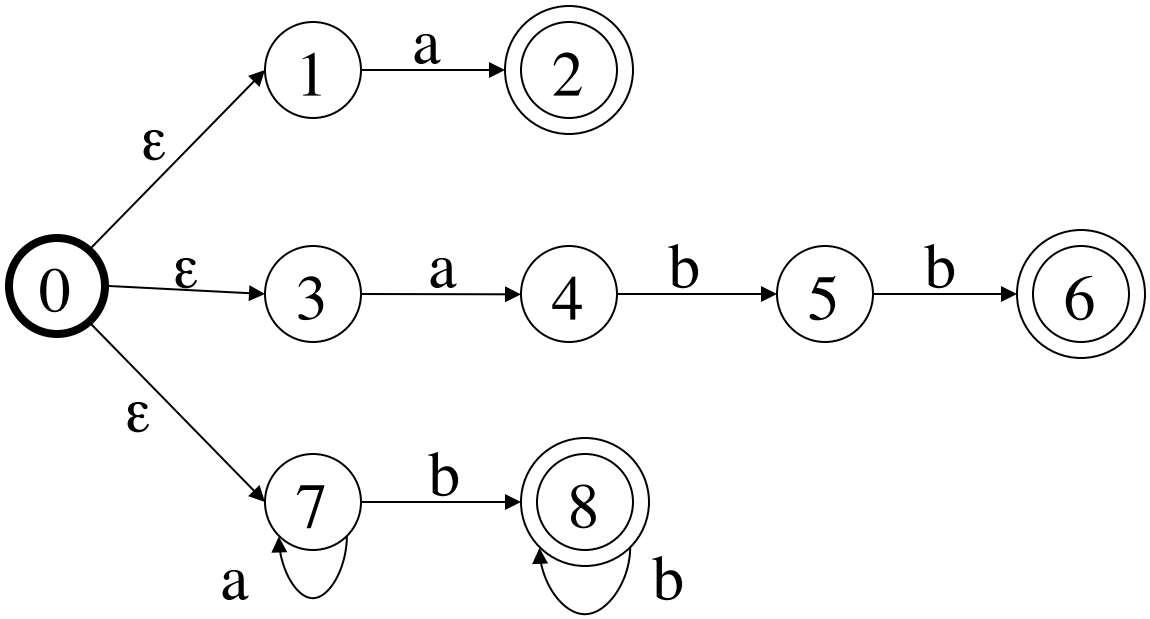
$_0a_1a_2b_3a_4$



TOKEN_C matches 0,3

TOKEN_A matches 0,1

Lexical Analysis using NFAs



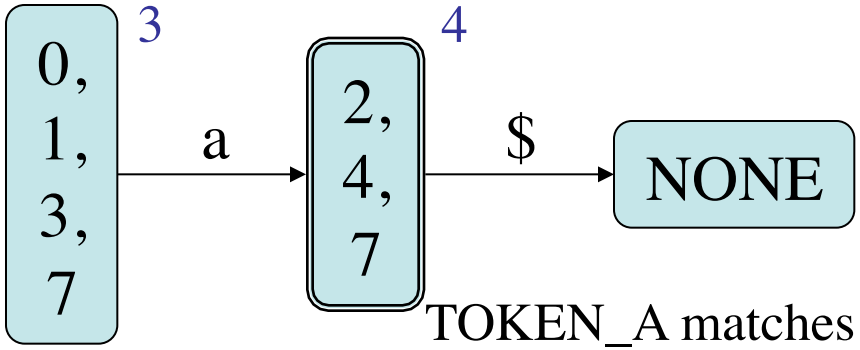
TOKEN_A = a

TOKEN_B = abb

TOKEN_C = a*b+

Input: aaba

$_0a_1a_2b_3a_4$



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

TOKEN_A matches 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 orig NFA final states with each DFA state

Lexical Analyzer using DFAs

- The DFA recognizer has to 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 r_2
- e.g. *a*b+/a*c* accepts a string *bac* 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 where r_1 matched but not r_2
- Reset to start state and start from position *p*

Summary

- Token ⇒ Pattern
- Pattern ⇒ Regular Expression
- Regular Expression ⇒ NFA
  - Thompson's Rules
- NFA ⇒ DFA
  - Subset construction
- DFA ⇒ minimal DFA
  - Minimization

⇒ Lexical Analyzer (multiple patterns)