Compiler Design
Lexical Analysis
Design of a Lexical-Analyzer Generator

conf. dr. ing. Ciprian-Bogdan Chirila

chirila@cs.upt.ro

http://www.cs.upt.ro/~chirila

Outline

- The Structure of the Generated Analyzer
- Pattern Matching Based on NFA's
- DFA's for Lexical Analyzers
- Implementing the Lookahead Operator

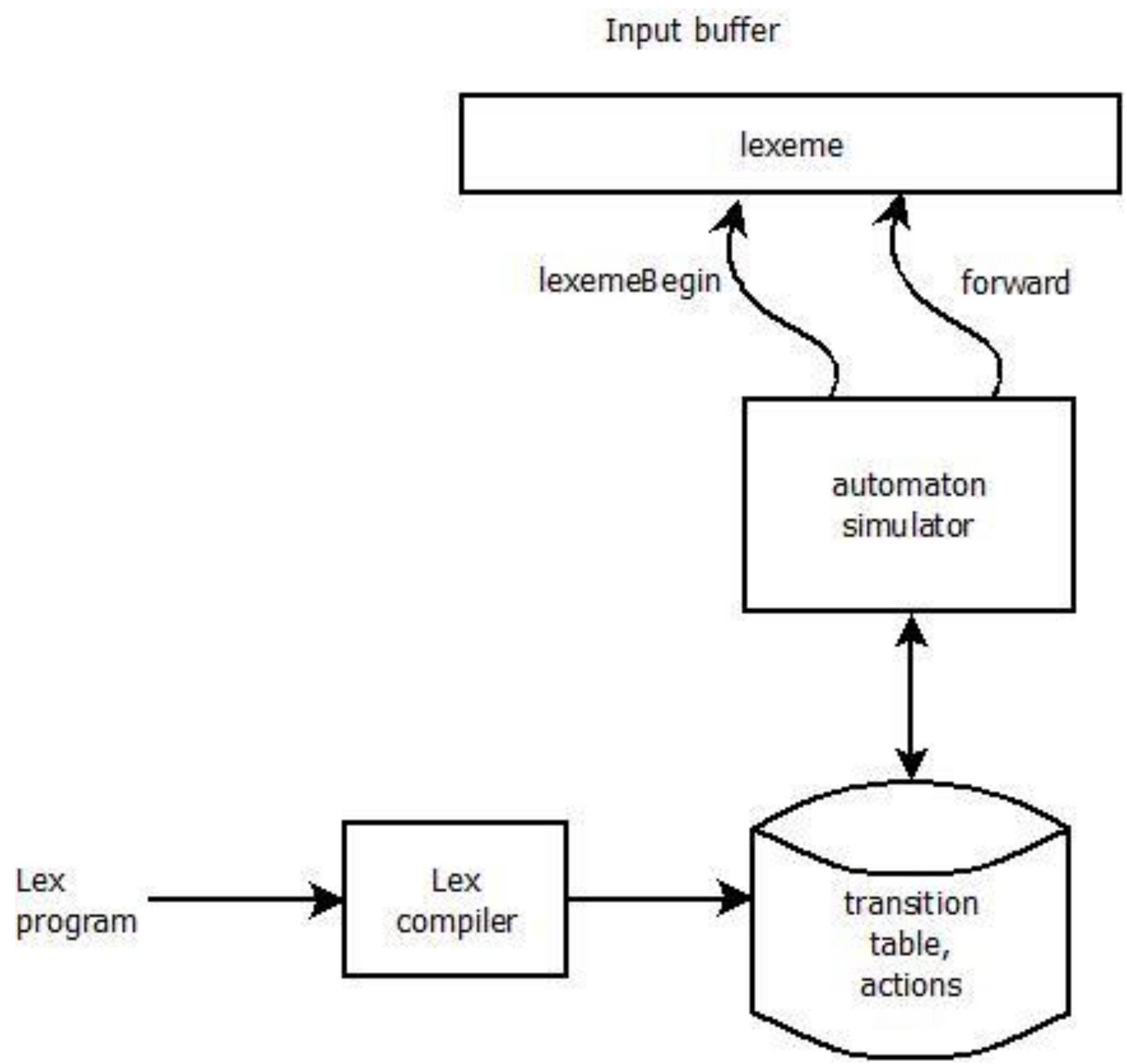
Objectives

- to present the architecture of Lex
- to discuss two approaches
  - NFA based
  - DFA based
    - implementation of Lex

The Structure of the Generated Lexical Analyzer

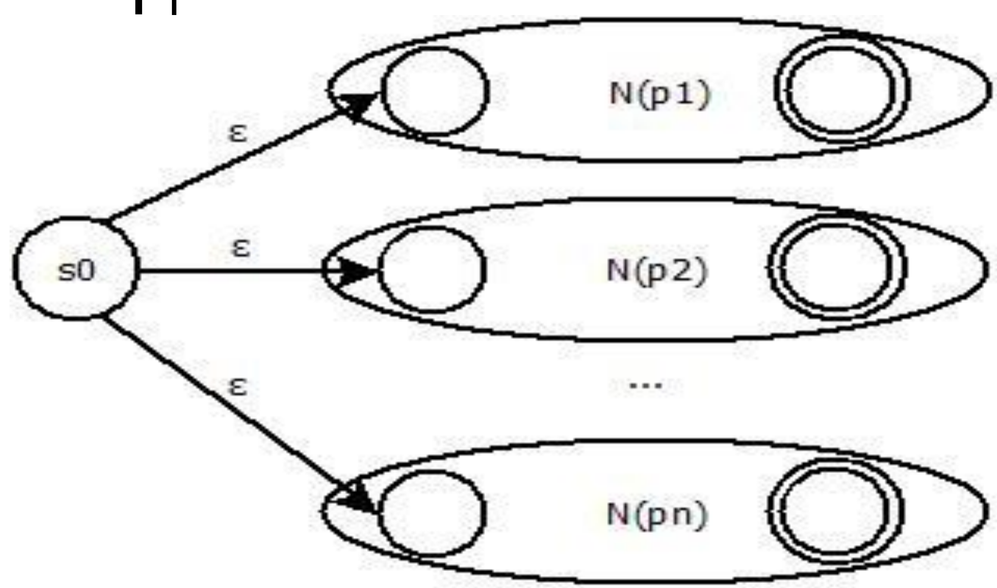
- fixed program that simulates an automaton
  - deterministic
  - nondeterministic
- transition table for the automaton
- functions that are passed directly through Lex to the output (we will see next)
- actions from the input program
  - as fragments of code
  - to be invoked at the appropriate time by the automaton simulator

Architecture of a Lexical Analyzer Generated by Lex

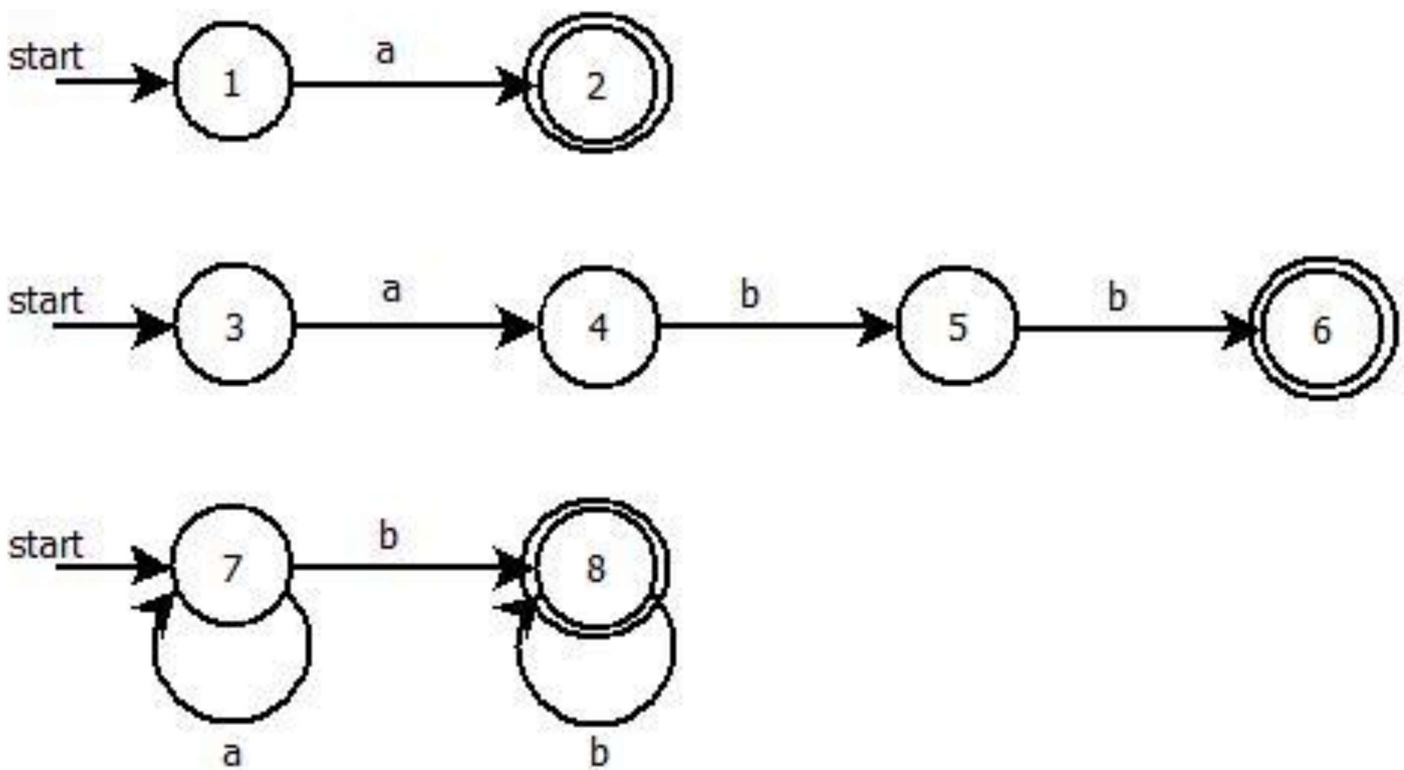


The Generation Process

- each regular expression pattern is transformed into NFA
- all NFAs are combined into one
  - new ε-transitions are added to NFAs N_i for pattern p_i



Example



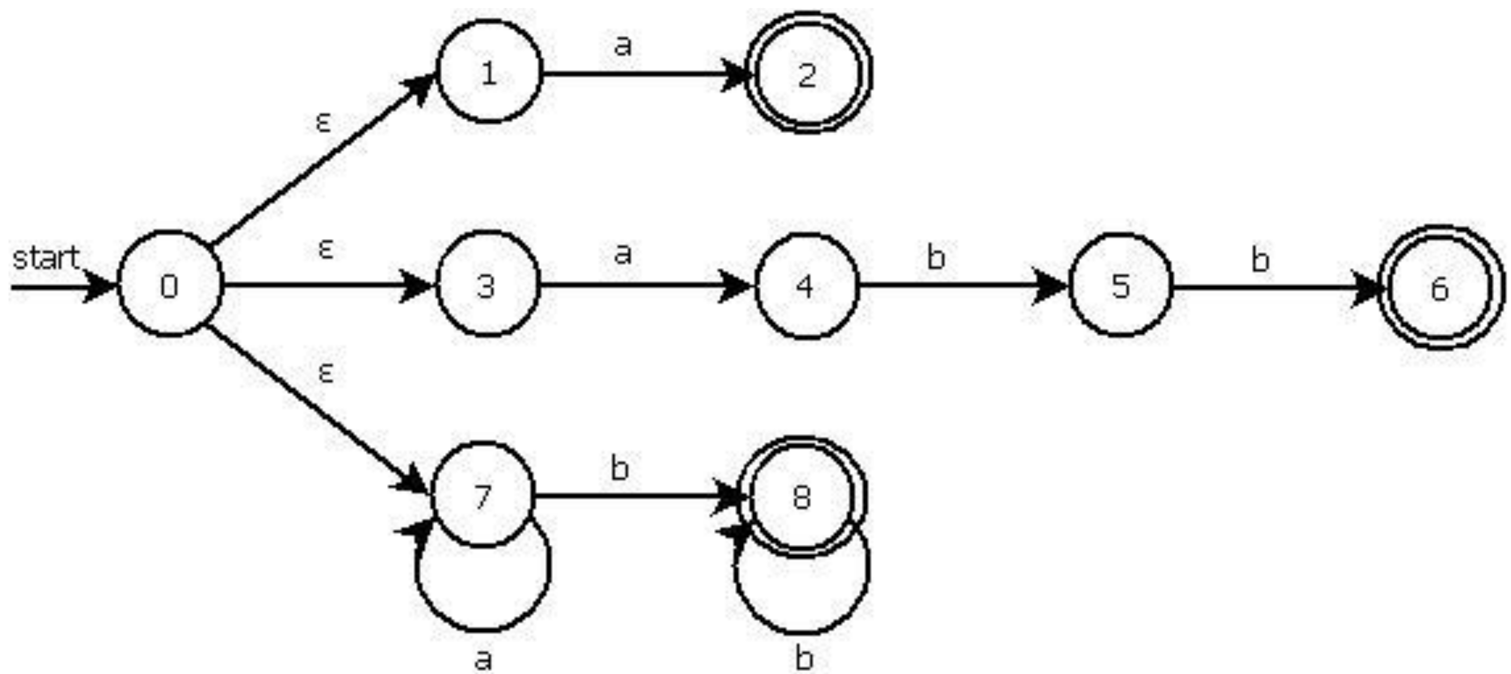
Example

- patterns
  - a {action A_1 for pattern p_1}
  - abb {action A_2 for pattern p_2}
  - a*b+ {action A_3 for pattern p_3}
- when several prefixes on the input matches multiple patterns
  - always prefer a longer prefix to a shorter prefix
  - if the longest possible prefix matches multiple patterns choose the pattern listed first
    - the lexeme "abb" is taken by the second rule

Conflict Resolution

- the three patterns present some conflicts
- abb matches p2 and p3
  - we consider it a lexeme for p2
  - p2 is listed above p3
- aabbbb…
  - we take the longest lexeme until another a is reached
  - we will report the lexeme from the initial a followed by as many b as there are

Example

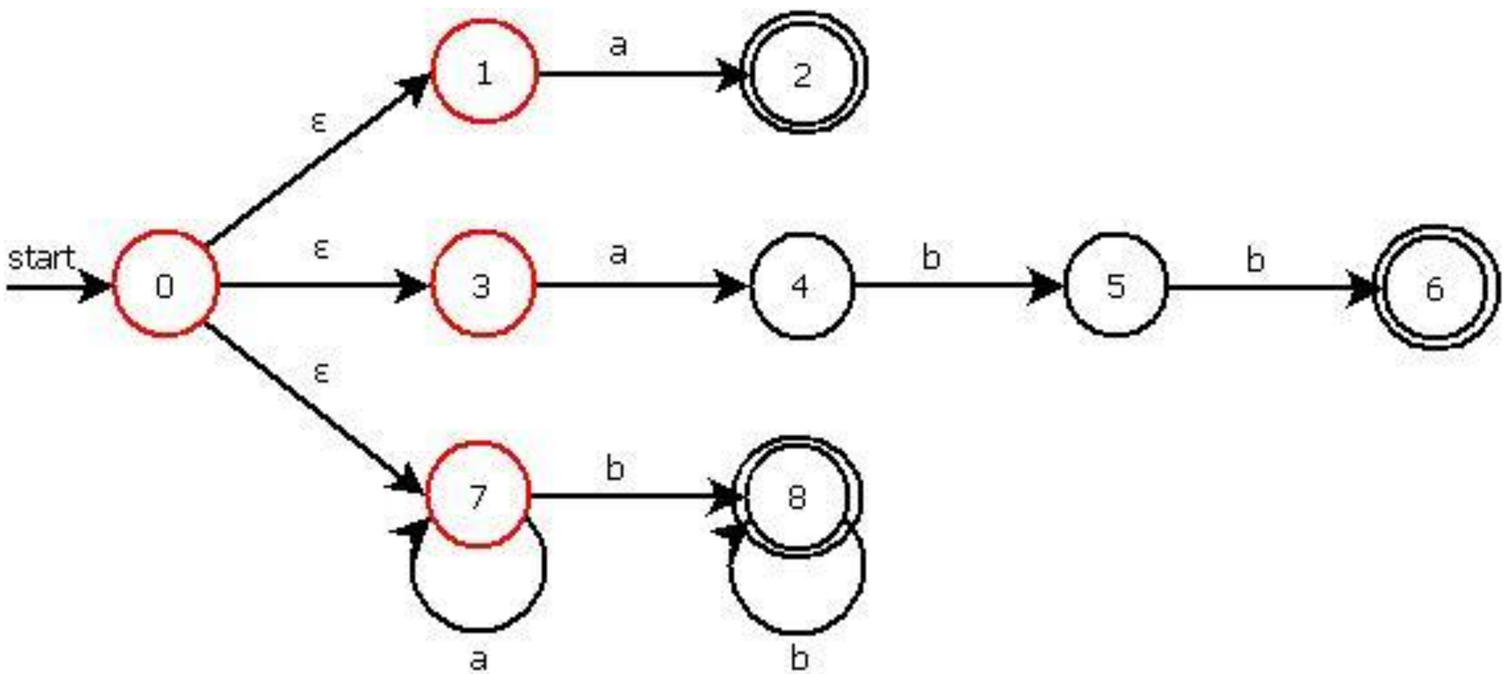


Pattern Matching Based on NFA's

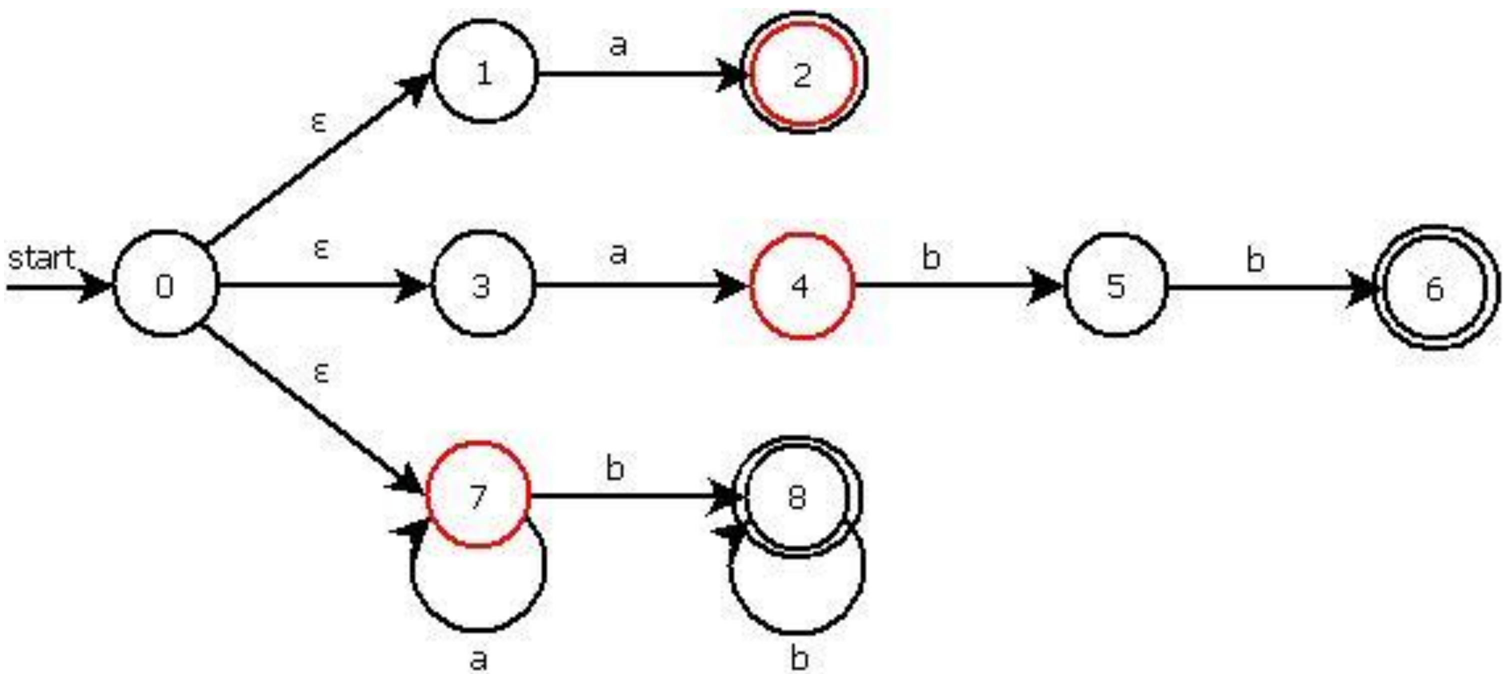
- NFA simulation algorithm

```
S=ε-closure(s0);
c=nextChar();
while(c!=eof)
{
  S=ε-enclosure(move(S,c));
  c=nextChar();
}
if(S∩F!=ø) return "yes";
else return "no";
```

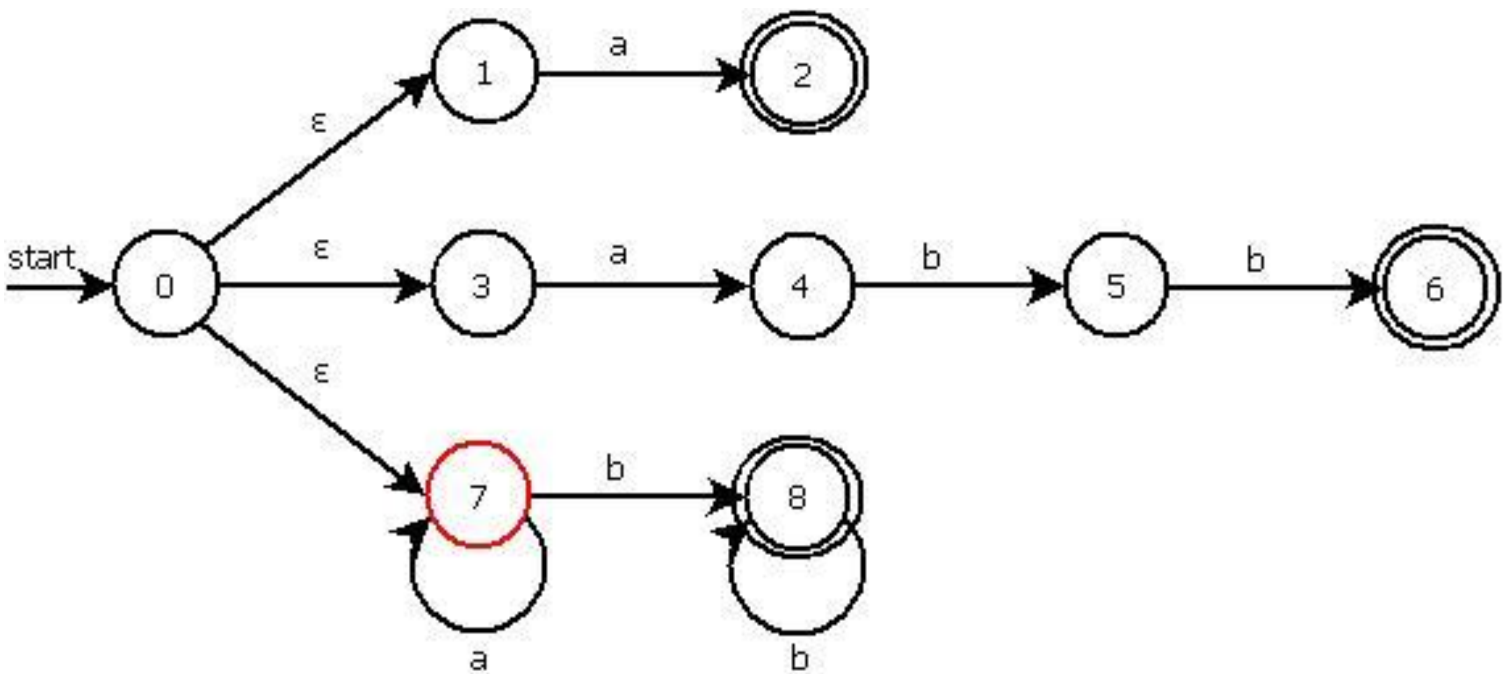
Example input a a b a



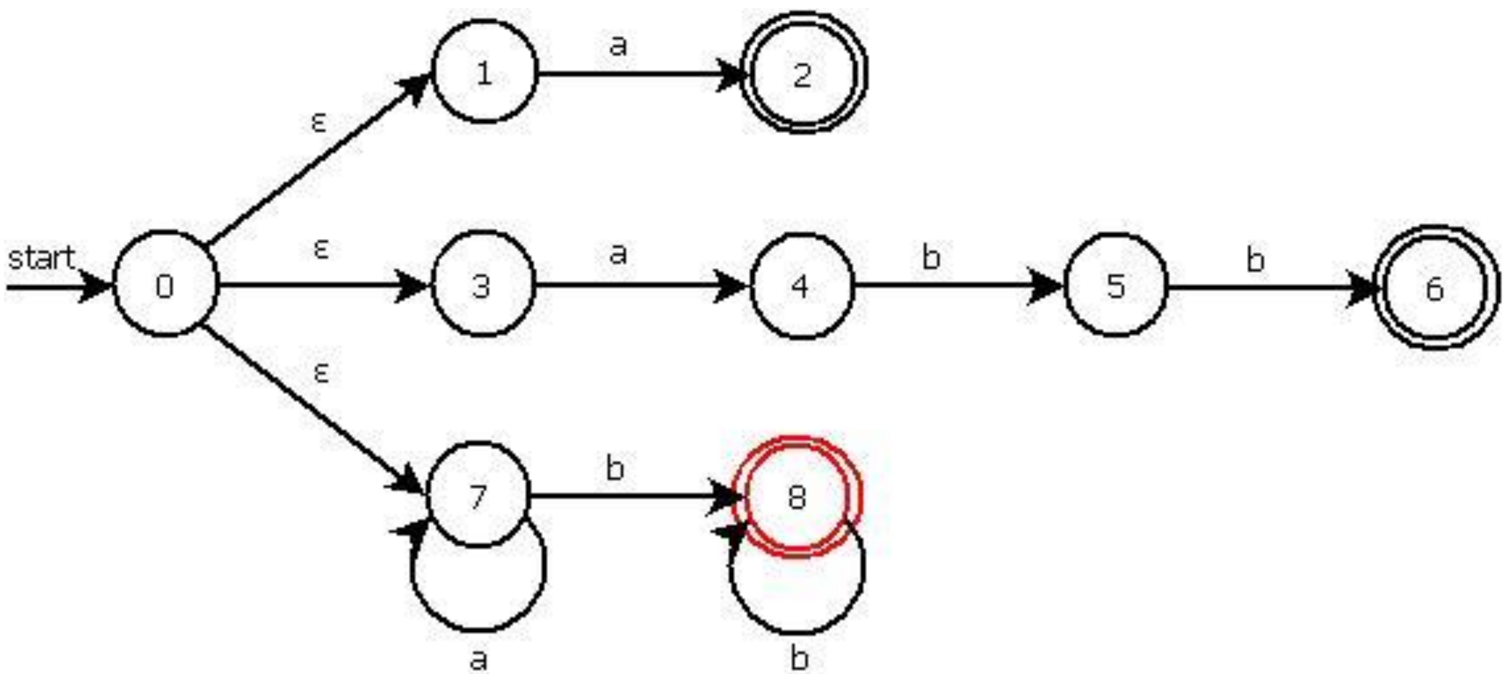
Example input a a b a



Example input a a b a



Example input a a b a



- pattern a*b+ was found !!!

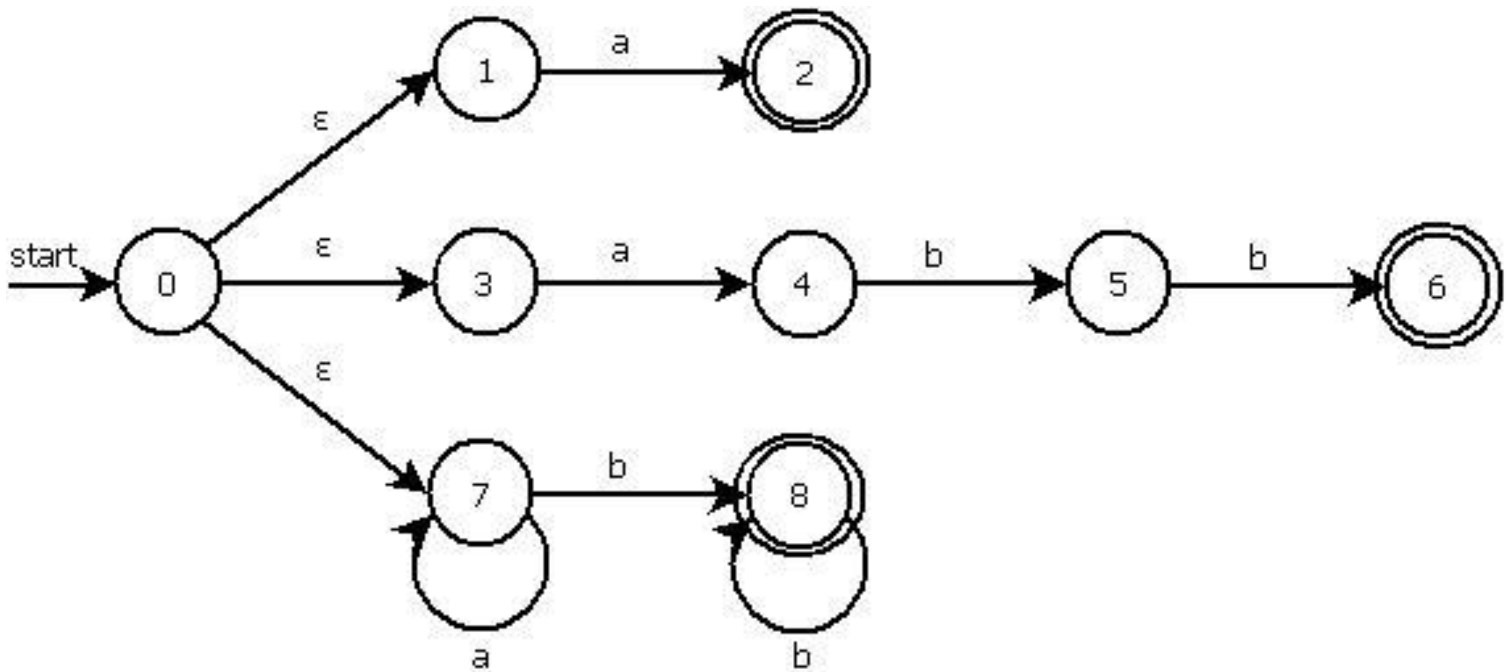
DFAs Architecture for Lexical Analyzers

- to convert NFA for all patterns into DFA
  - by using the subset construction algorithm
- within each DFA state having one or more NFA accepting states
  - to determine the first pattern whose accepting state is represented
  - to make that pattern the output of the DFA state

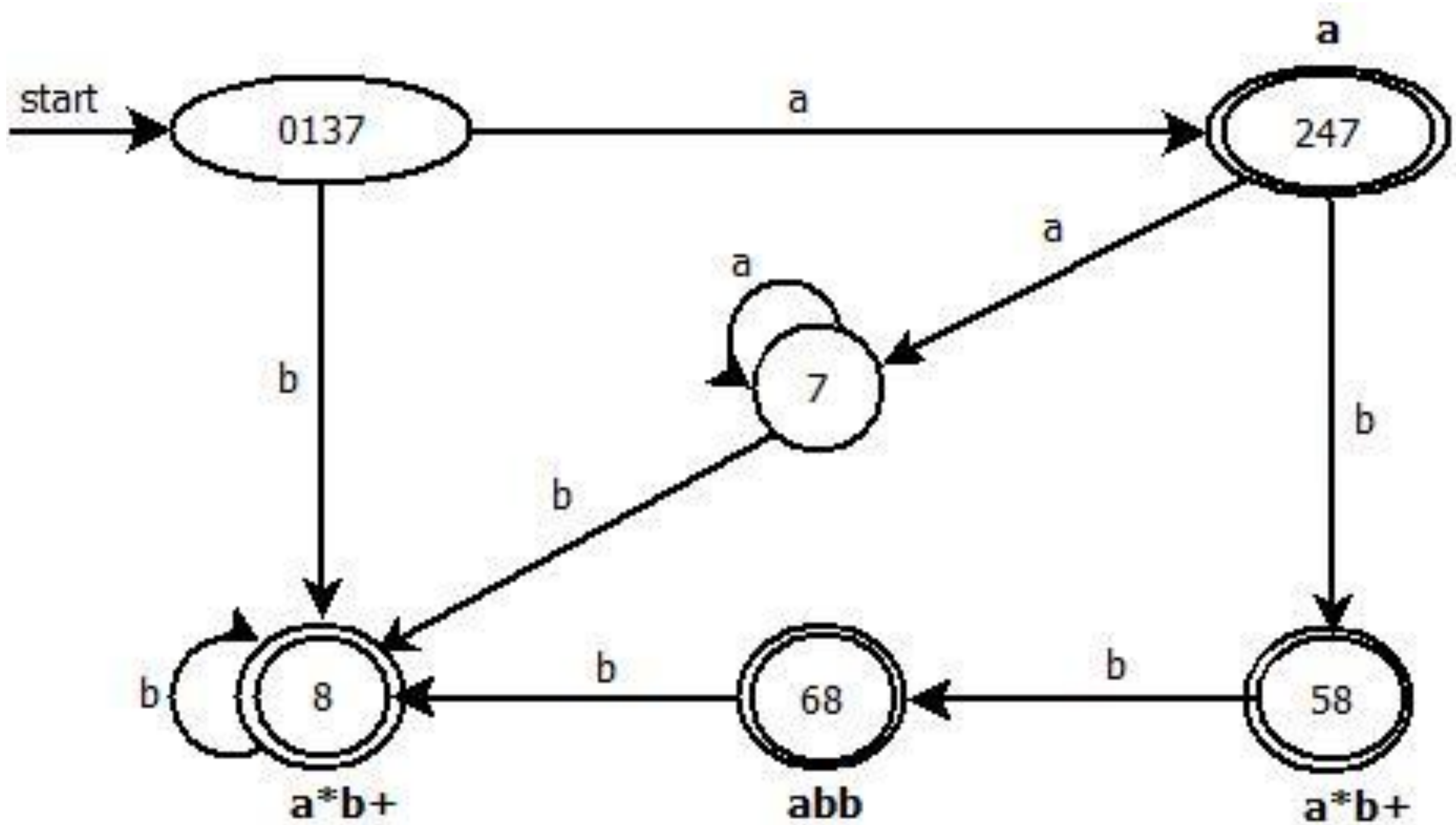
The Subset Construction Algorithm

```
while(there is an unmarked state T in Dstates)
{
  mark T;
  for(each input symbol a)
  {
      U=ε-closure(move(T,a));
      if (U is not in Dstates)
            add U as unmarked state to Dstates;
      Dtran[T,a]=U;
  }
}
```

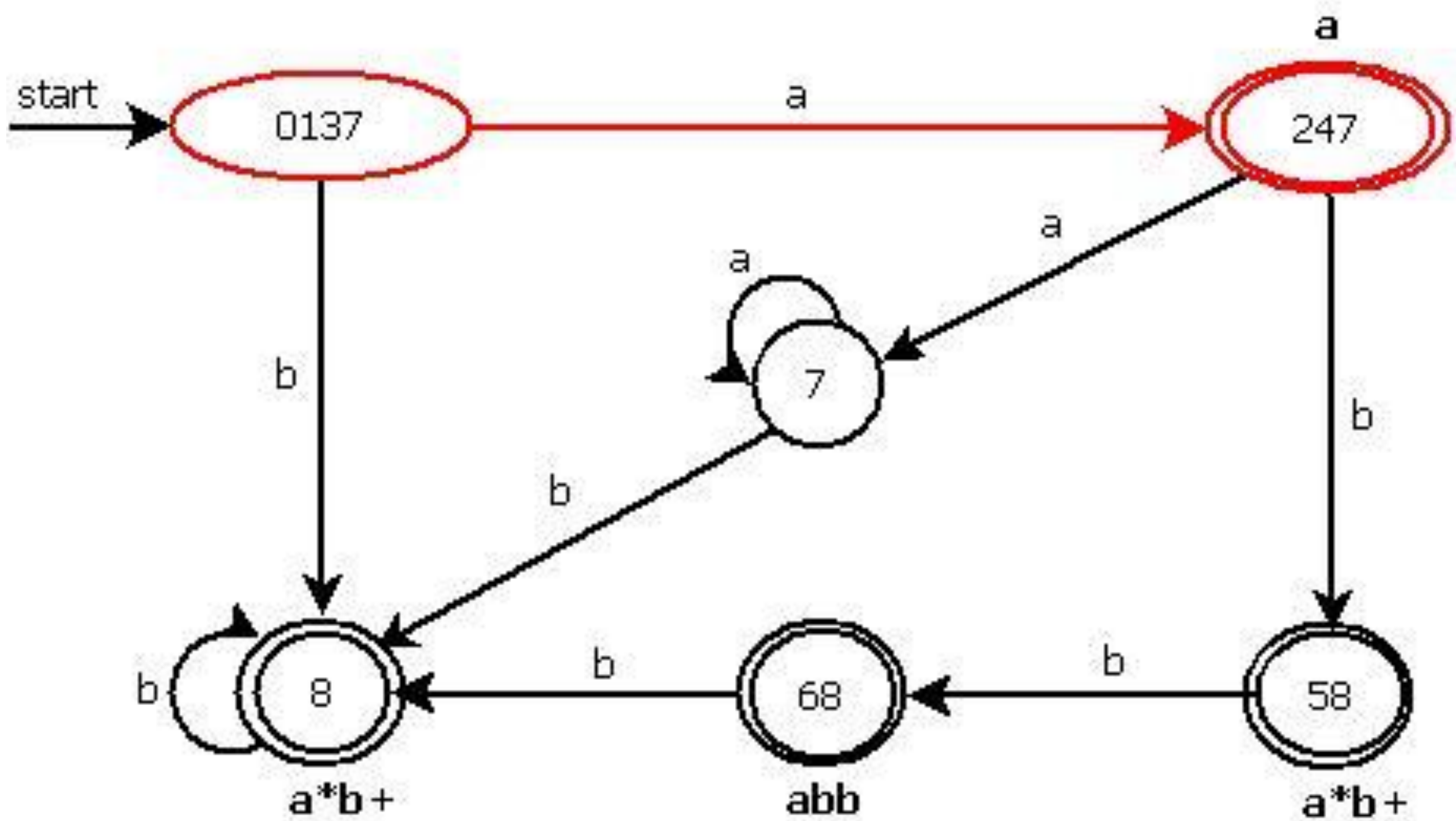
NFA Example



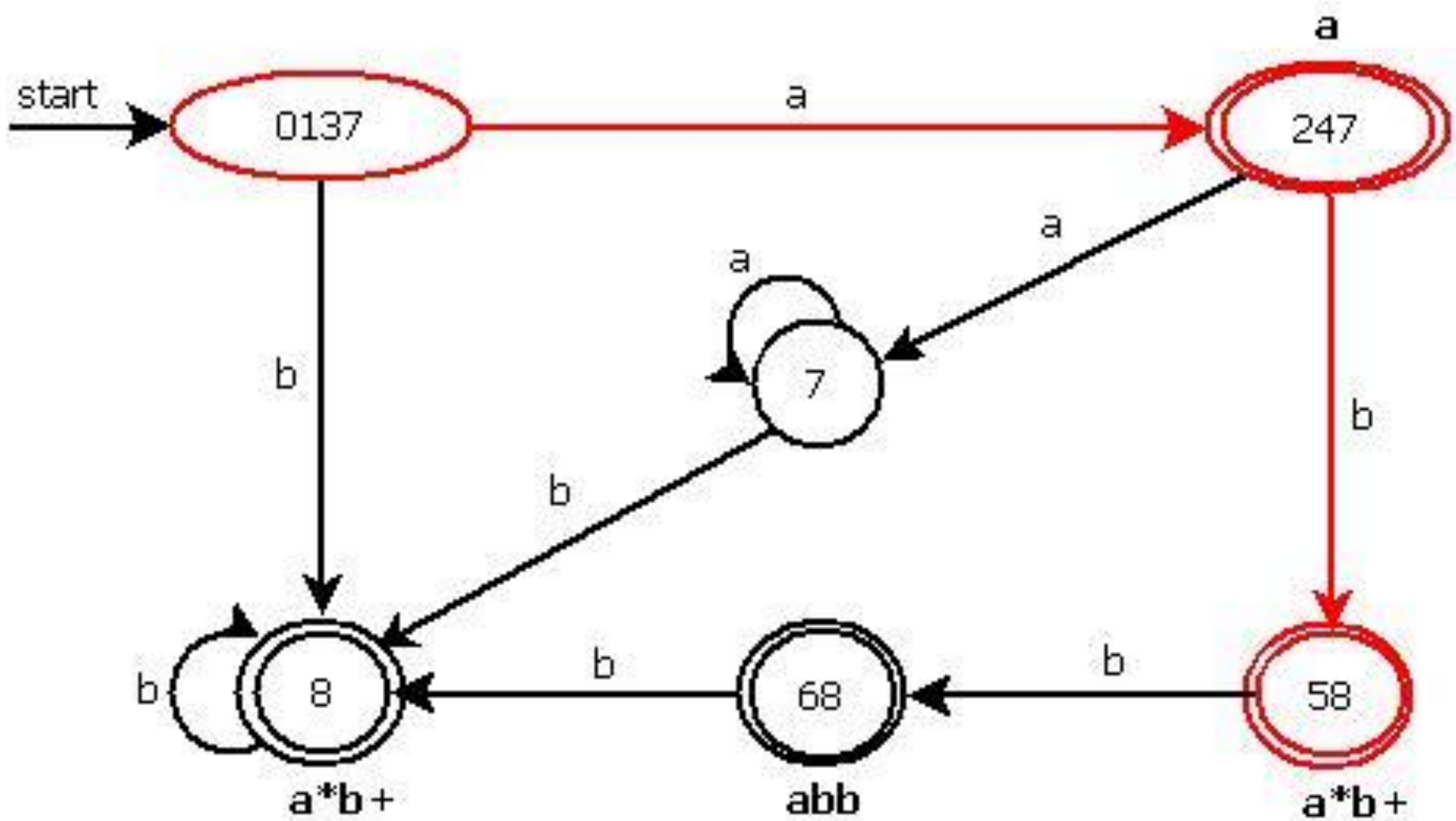
NFA to DFA Example



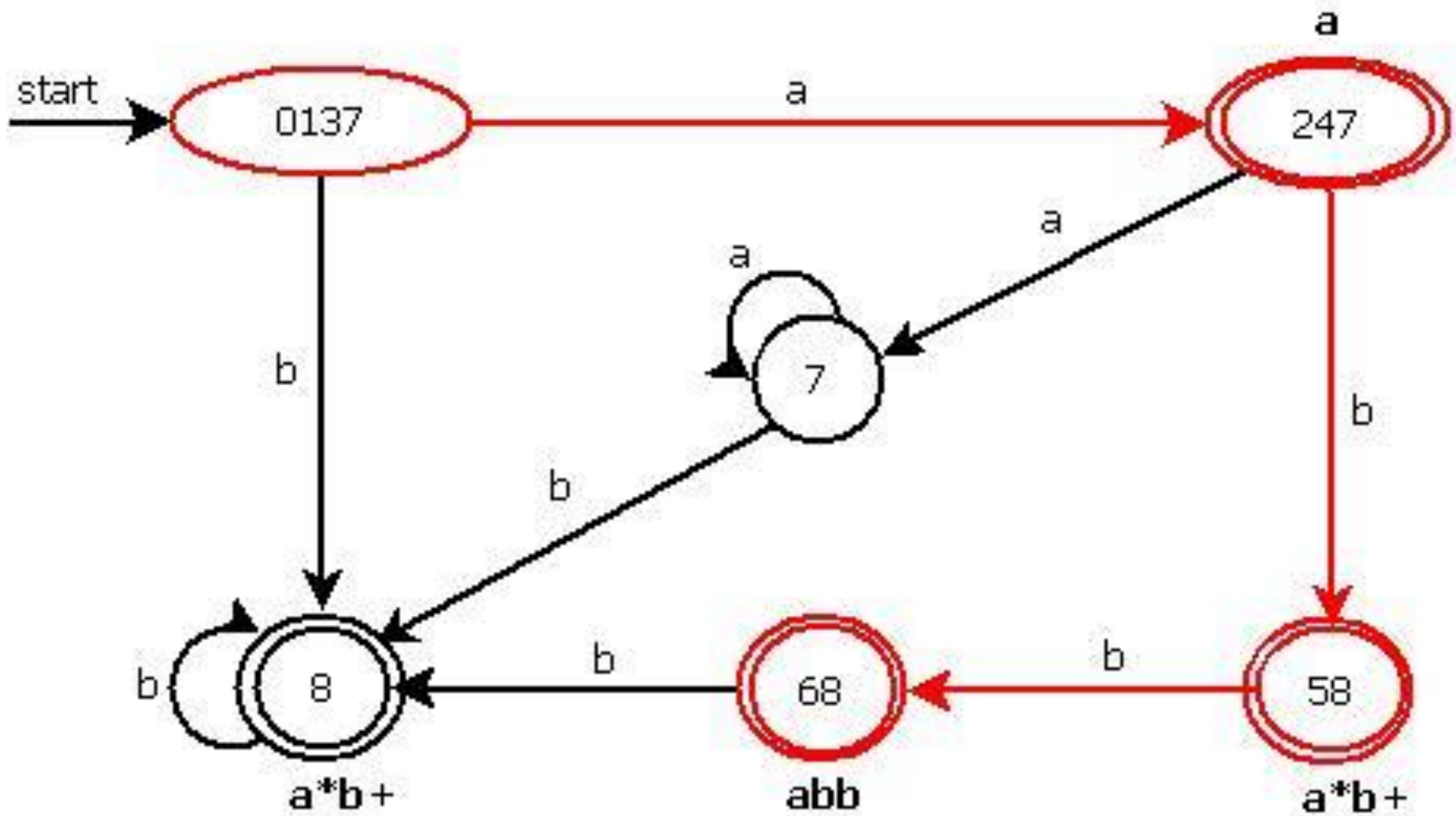
DFA Simulation Example a b b a



DFA Simulation Example a b b a



DFA Simulation Example a b b a



Dead States in DFA's

- the automaton not quite a DFA
  - no transitions on every state x every input
- we have omitted
  - transitions to the dead state ∅
  - from the dead state ∅ to itself

Bibliography

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman – Compilers, Principles, Techniques and Tools, Second Edition, 2007