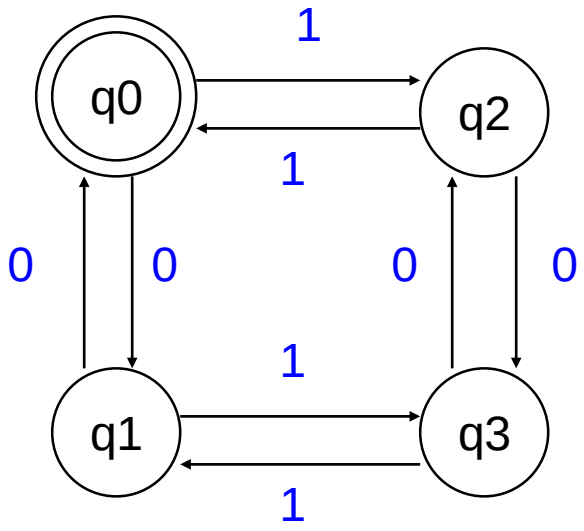
Lexical Analysis – Part II

Yongjun Park
Hanyang University

Class Problem From Last Time

Is this a DFA or NFA?
What strings does it recognize?



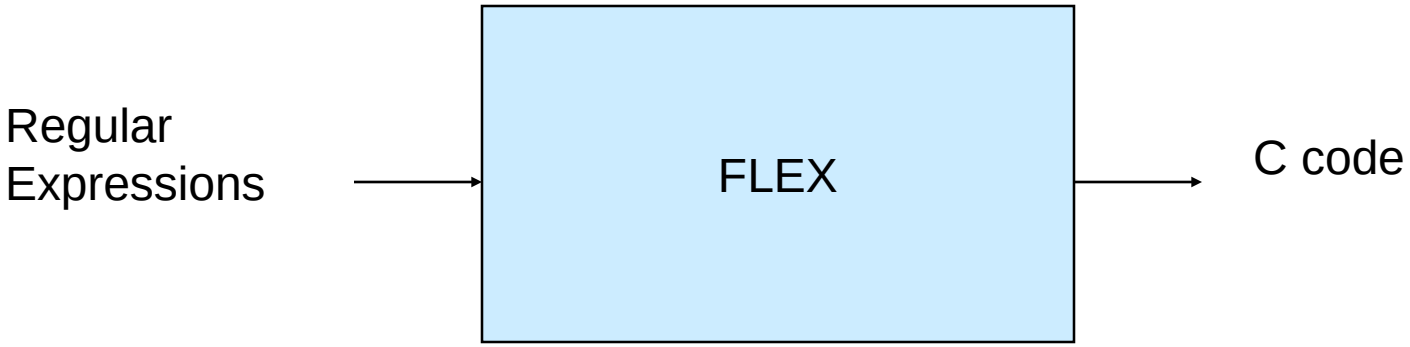
Lex Notes

- **Questions from last time**
  - [\t]+, there is a space here
    - So this matches all white space characters except new lines
  - The period operator, **.** ,does match all characters except newline

Reading (Aho Book)

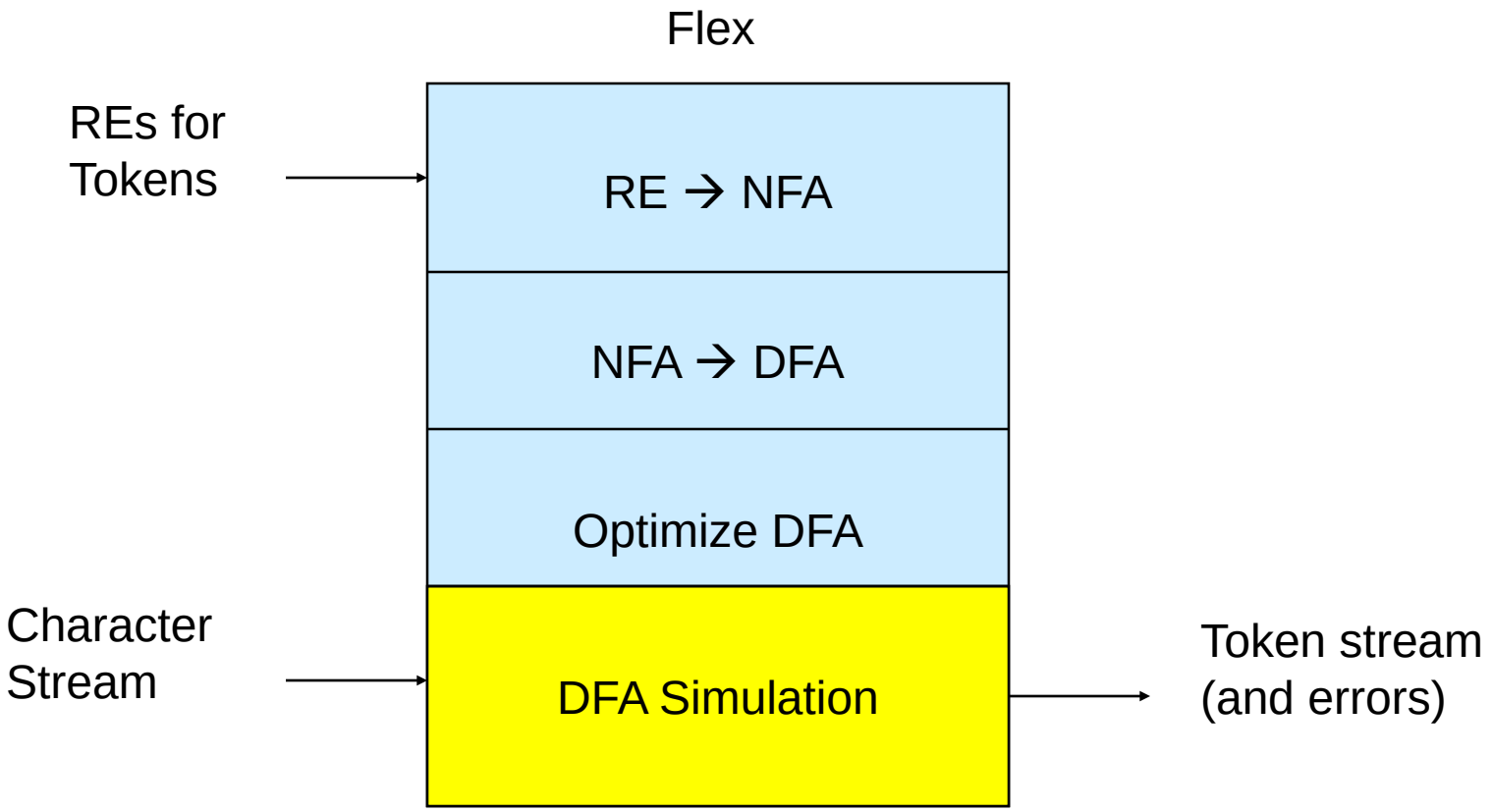
- **Ch 2**
  - Just skim this
  - High-level overview of compiler, which could be useful
- **Ch 3**
  - Read carefully, more closely follows lecture
  - Go over examples

How Does Lex Work?



Some kind of DFAs and NFAs
stuff going on inside

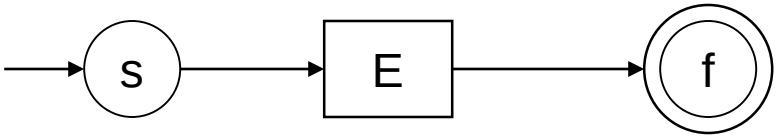
How Does Lex Work?



Regular Expression to NFA

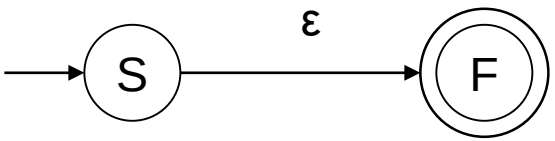
- **Its possible to construct an NFA from a regular expression**

- **Thompson's construction algorithm**
  - Build the NFA inductively
  - Define rules for each base RE
  - Combine for more complex RE's

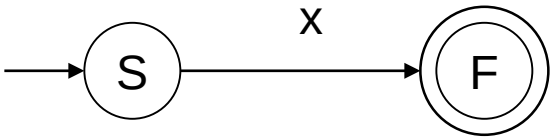


general machine

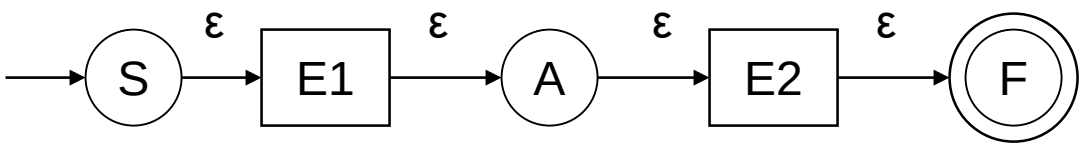
Thompson Construction



empty string transition



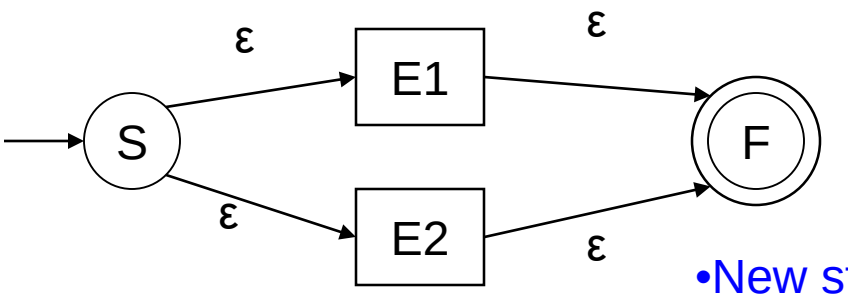
alphabet symbol transition



Concatenation:
(E1 E2)

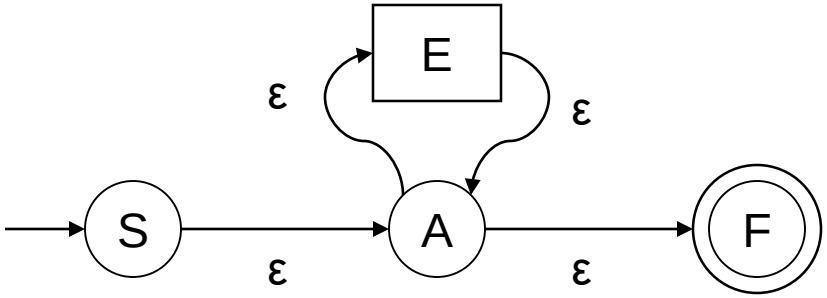
- New start state S ε-transition to the start state of E1
- ε-transition from final/accepting state of E1 to A, ε-transition from A to start state of E2
- ε-transitions from the final/accepting state E2 to the new final state F

Thompson Construction - Continued



Alteration: (E1 | E2)

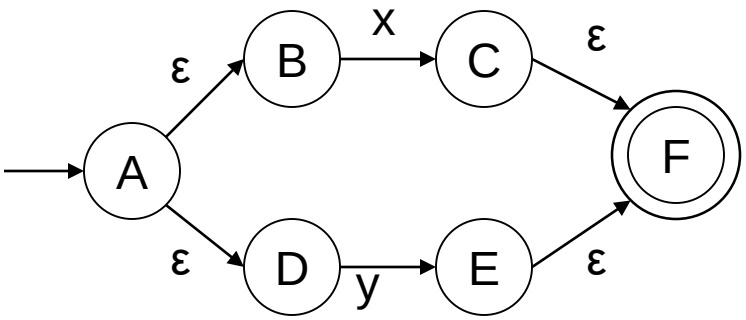
- New start state S ε-transitions to the start states of E1 and E2
- ε-transitions from the final/accepting states of E1 and E2 to the new final state F



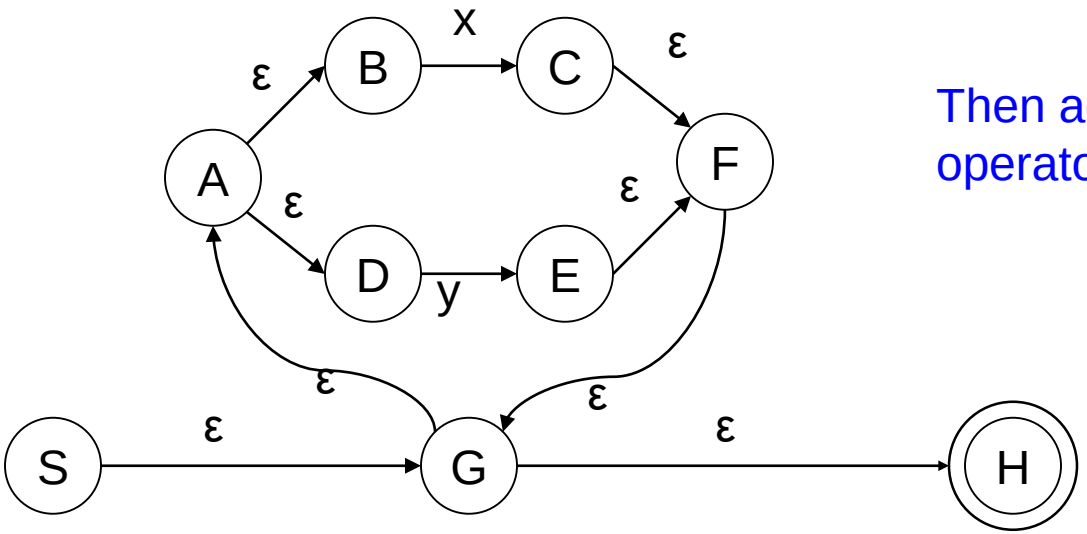
Closure: (E*)

Thompson Construction - Example

Develop an NFA for the RE: (x | y)*



First create NFA for (x | y)



Then add in the closure operator

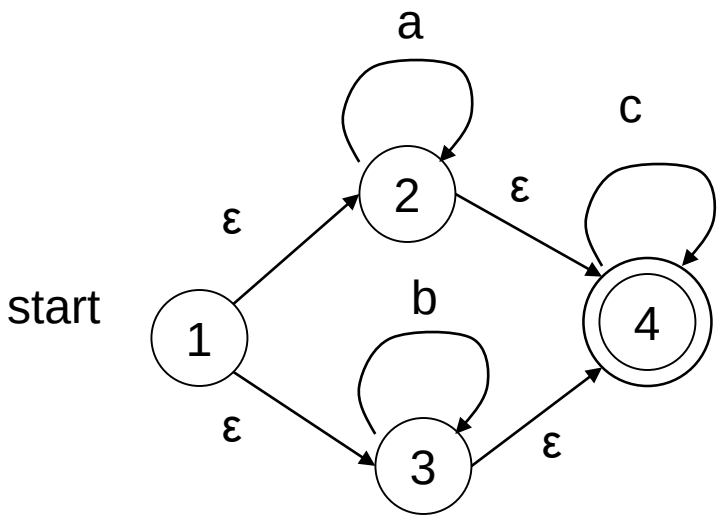
Class Problem

Develop an NFA for the RE: (\+? | -?) d+

NFA to DFA

- **Remove the non-determinism**
- **2 problems**
  - States with multiple outgoing edges due to same input
  - ε transitions

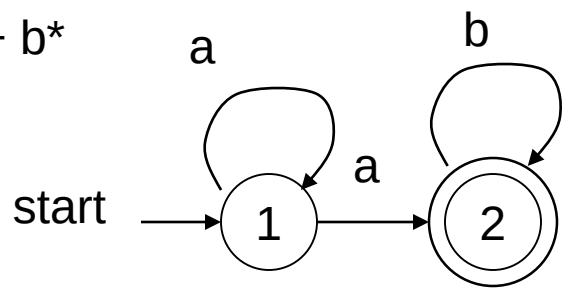
(a*| b*) c*



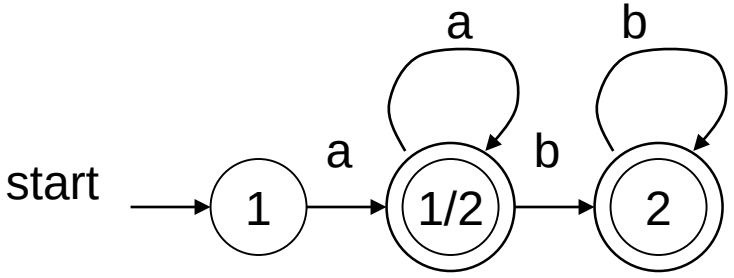
NFA to DFA (2)

- **Problem 1: Multiple transitions**
  - Solve by subset construction
  - Build new DFA based upon the power set of states on the NFA
  - Move (S,a) is relabeled to target a new state whenever single input goes to multiple states

a+ b*



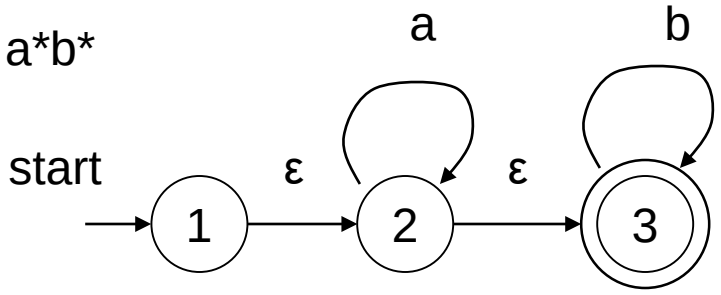
(1,a) → 1 or 2, create new state 1/2
(1/2,a) →1/2
(1/2,b) → 2



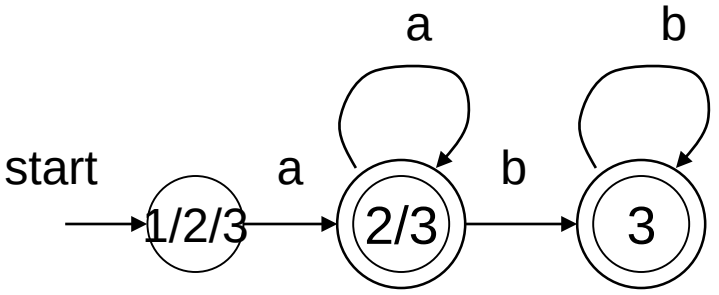
(2,a) → ERROR
(2,b) → 2
Any state with “2” in name is a final state

NFA to DFA (3)

- **Problem 2: ε transitions**
  - Any state reachable by an ε transition is "part of the state"
  - ε-closure - Any state reachable from S by ε transitions is in the ε-closure; treat ε-closure as 1 big state, always include ε-closure as part of the state

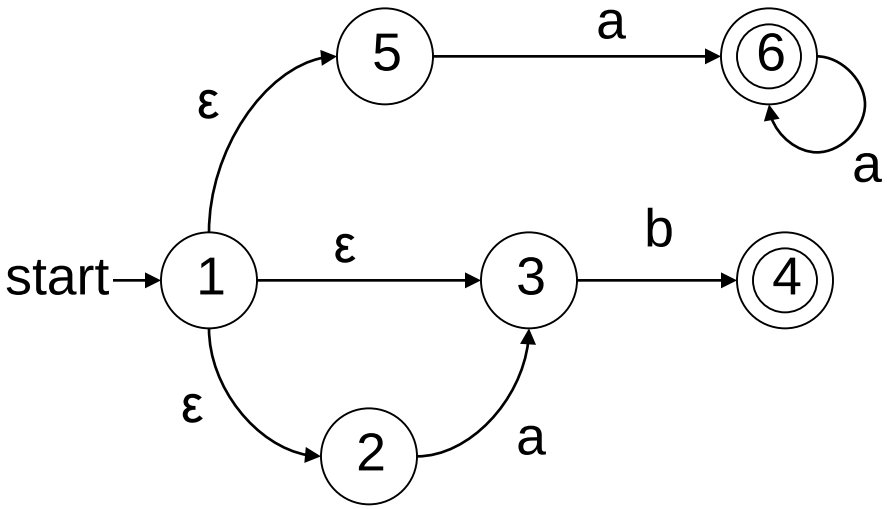


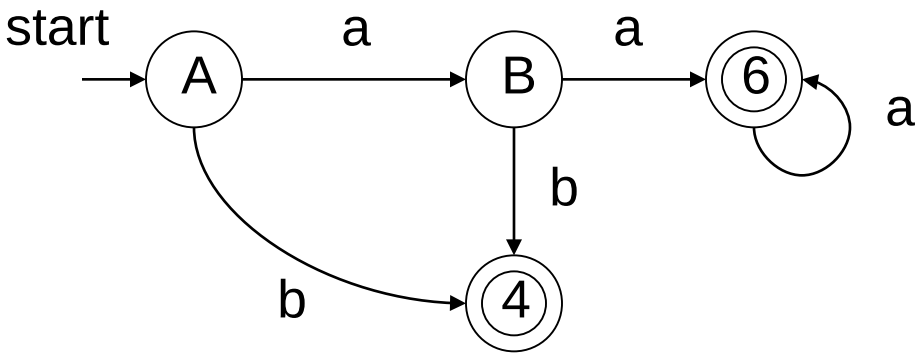
ε-closure(1) = {1,2,3}
ε-closure(2) = {2,3}
create new state 1/2/3
create new state 2/3



(1/2/3, a) → 2/3
(1/2/3, b) → 3
(2/3, a) → 2/3
(2/3, b) → 3

NFA to DFA - Example

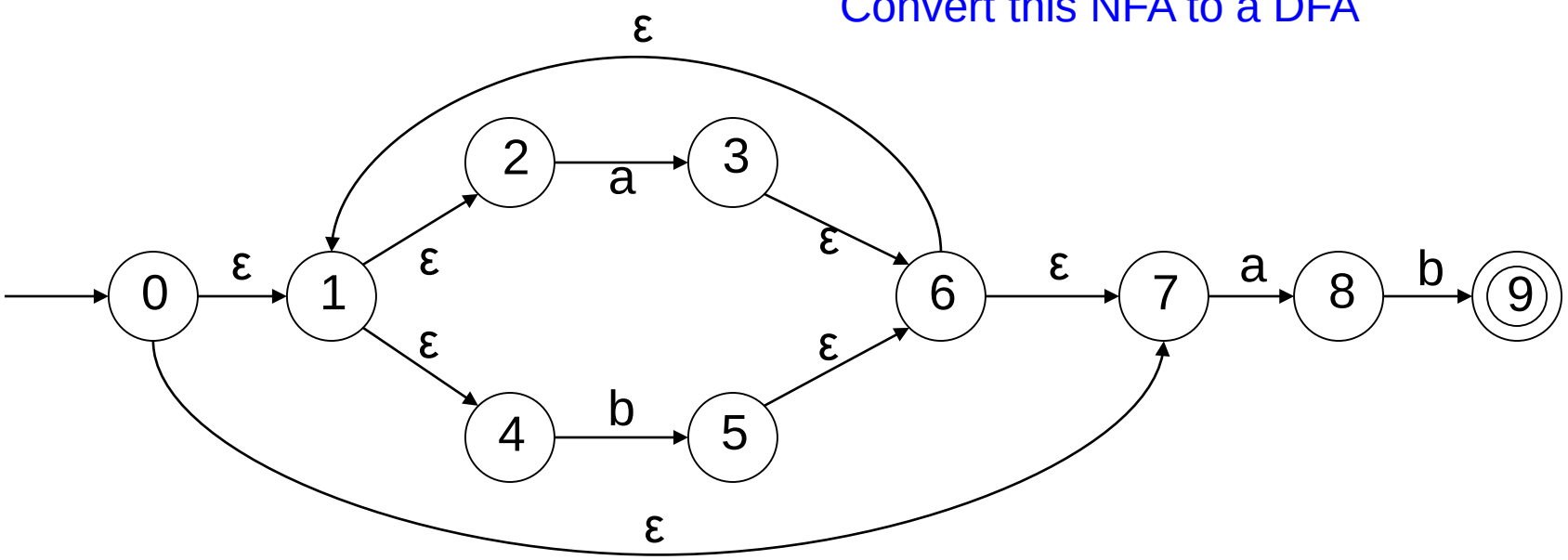




- ε-closure(1) = {1, 2, 3, 5}
- Create a new state A = {1, 2, 3, 5} and examine transitions out of it
- move(A, a) = {3, 6}
- Call this a new subset state = B = {3, 6}
- move(A, b) = {4}
- move(B, a) = {6}
- move(B, b) = {4}
- Complete by checking move(4, a); move(4, b); move(6, a); move(6, b)

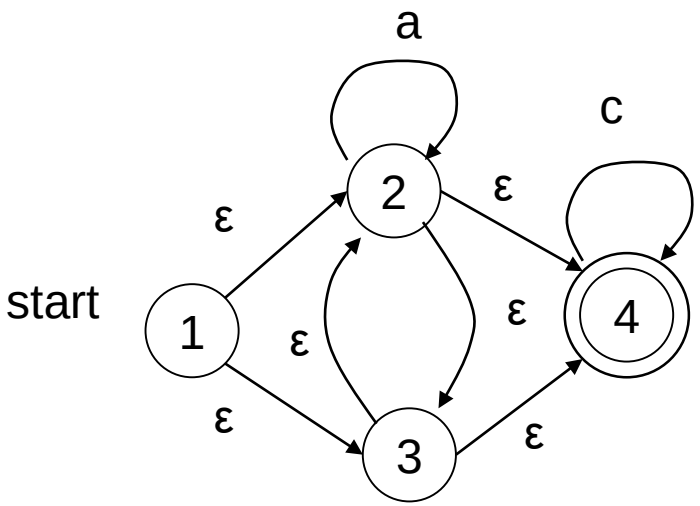
Class Problem

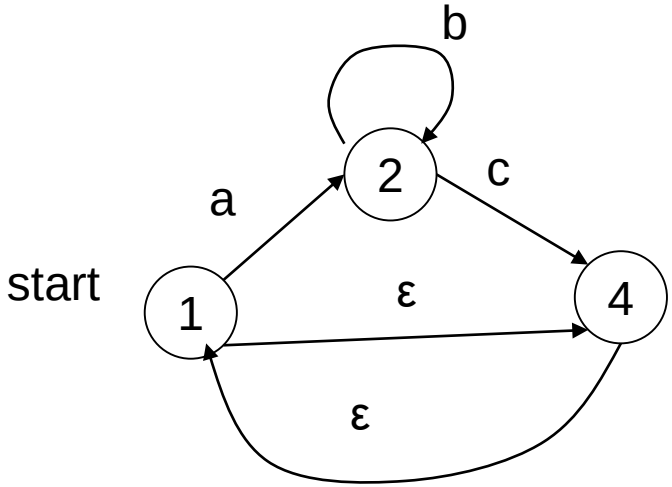
Convert this NFA to a DFA



NFA to DFA Optimizations

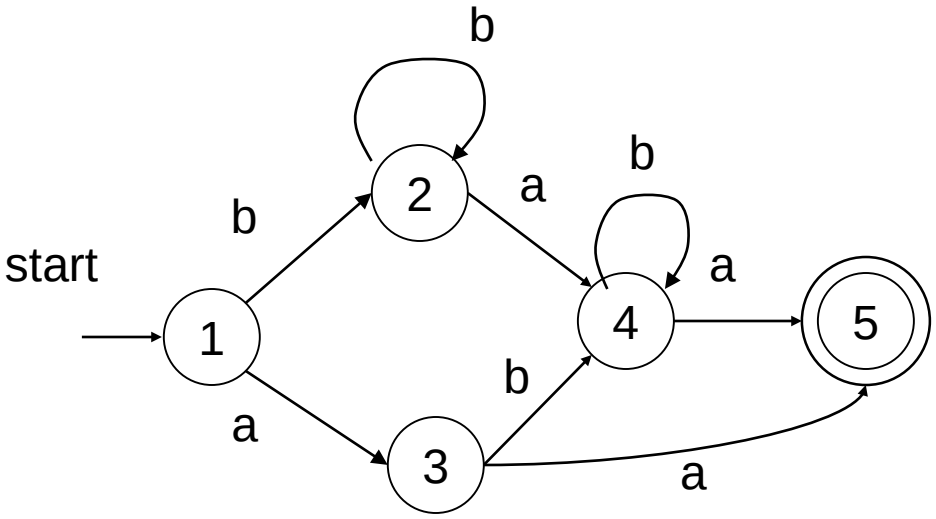
- **Prior to NFA to DFA conversion:**
- **Empty cycle removal**
  - Combine nodes that comprise cycle
  - Combine 2 and 3
- **Empty transition removal**
  - Remove state 4, change transition 2-4 to 2-1

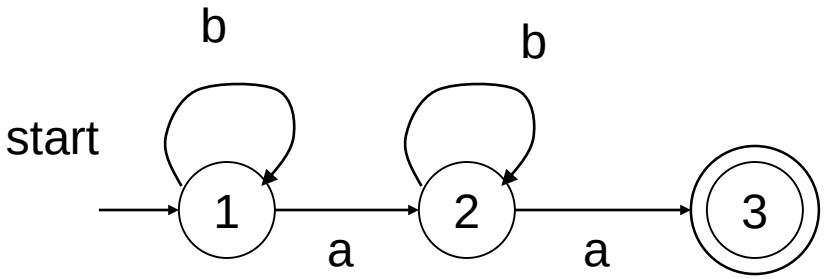




State Minimization

- **Resulting DFA can be quite large**
  - Contains redundant or equivalent states

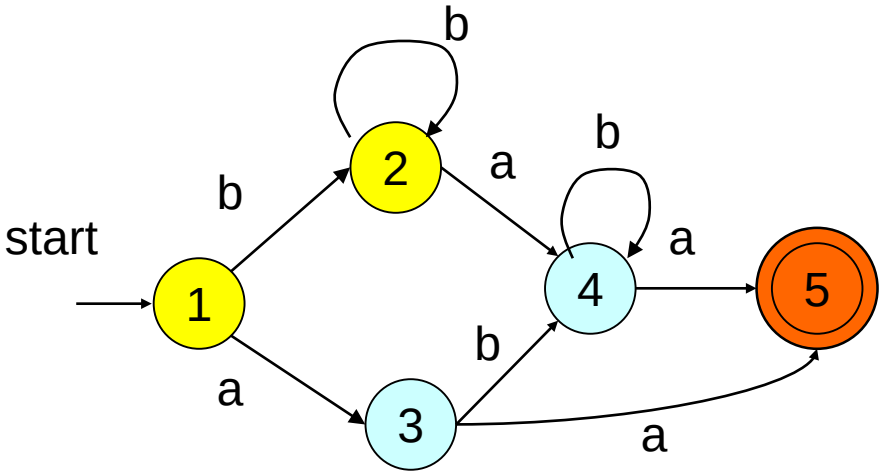




Both DFAs accept
b*ab*a

State Minimization (2)

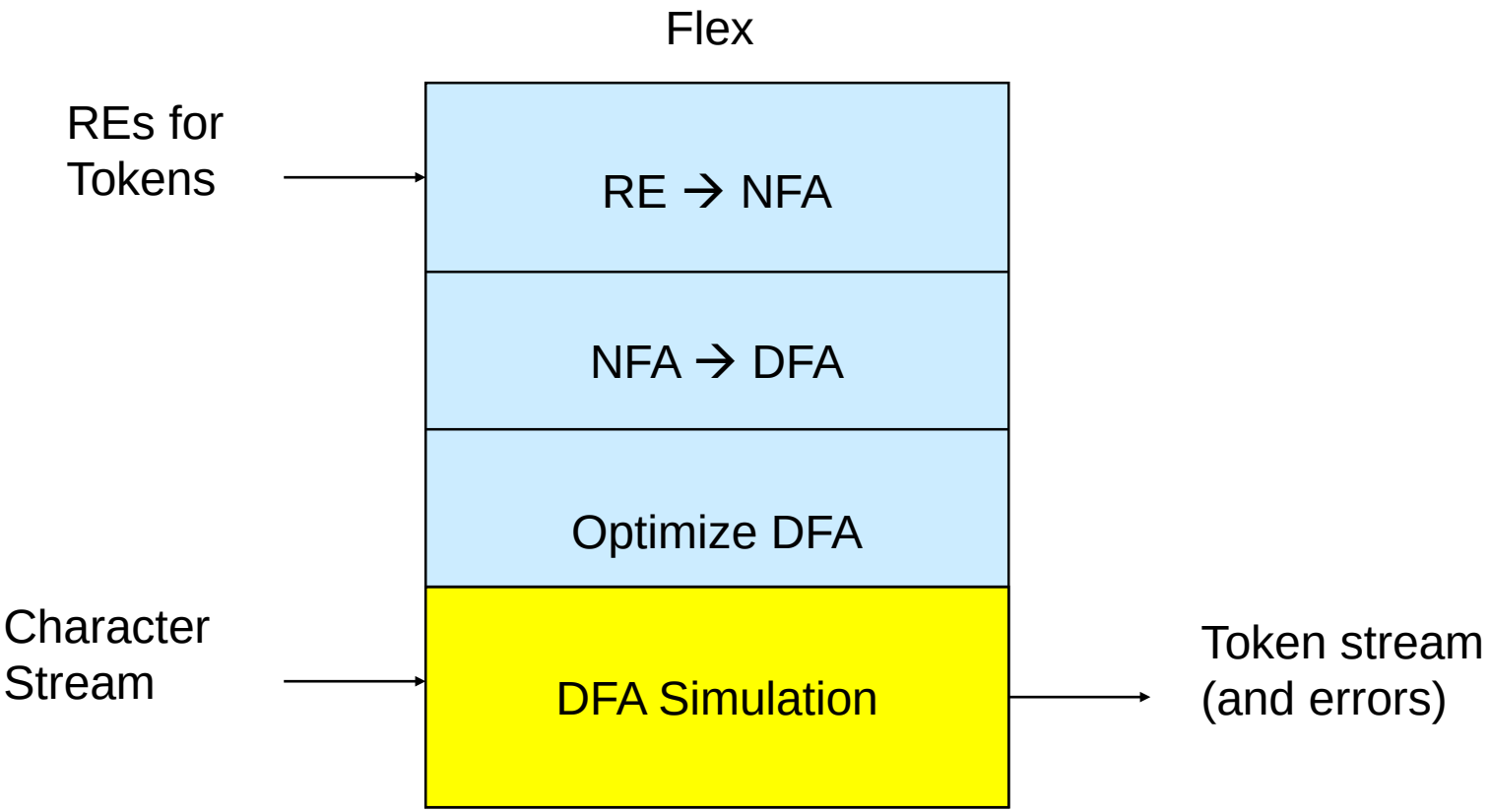
- **Idea – find groups of equivalent states and merge them**
  - All transitions from states in group G1 go to states in another group G2
  - Construct minimized DFA such that there is 1 state for each group of states



Basic strategy: identify distinguishing transitions

Putting It All Together

Remaining issues: how to Simulate, multiple REs, producing a token stream, longest match, rule priority



Simulating the DFA

* Straight-forward translation of DFA to C program
* Transitions from each state/input can be represented as table
  - Table lookup tells where to go based on current state/input

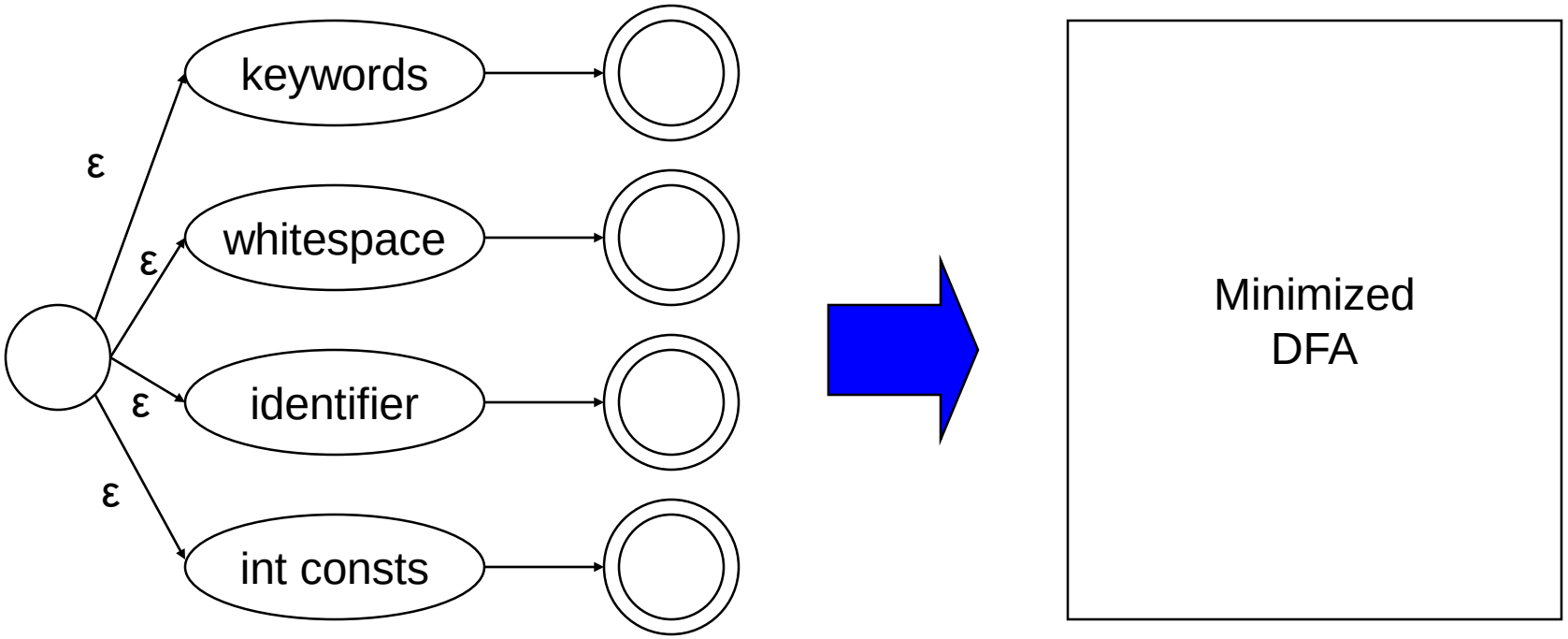
```
trans_table[NSTATES][NINPUTS];
accept_states[NSTATES];
state = INITIAL;

while (state != ERROR) {
        c = input.read();
        if (c == EOF) break;
        state = trans_table[state][c];
}
return accept_states[state];
```

Not quite this simple but close!

Handling Multiple REs

Combine the NFAs of all the regular expressions into a single NFA



Remaining Issues

- **Token stream at output**
  - Associate tokens with final states
  - Output corresponding token when reach final state
- **Longest match**
  - When in a final state, look if there is a further transition. If no, return the token for the current final state
- **Rule priority**
  - Same longest matching token when there is a final state corresponding to multiple tokens
  - Associate that final state to the token with highest priority