

Lexical Analysis – Part II

EECS 483 – Lecture 3

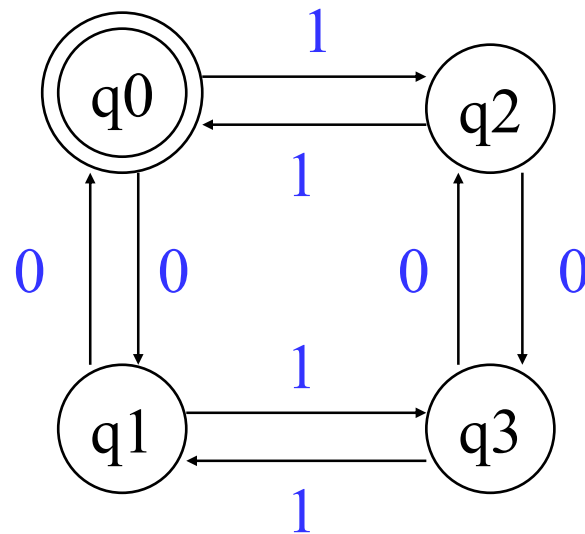
University of Michigan

Wednesday, September 13, 2006

Class Problem From Last Time

Is this a DFA or NFA?

What strings does it recognize?



Lex Notes

- ❖ 64-bit machine compilation of flex file
 - » `gcc -m64 lex.yy.c -lfl`
- ❖ Questions from last time
 - » `[\t]+`, there is a space here
 - So this matches all white space characters except new lines
 - » Flex can detect spaces if you want it to
 - » The period operator, `.`, does match all characters except newline

Reading

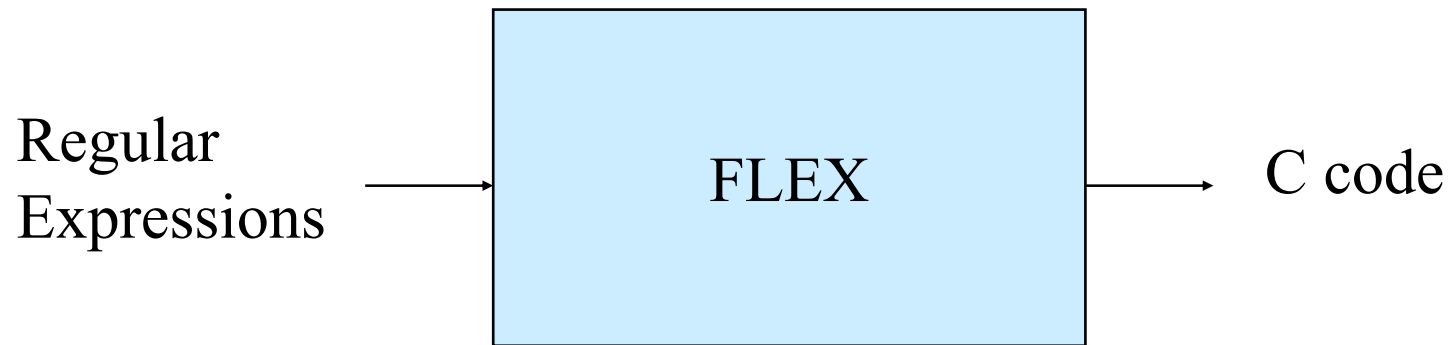
❖ Ch 2

- » Just skim this
- » High-level overview of compiler, which could be useful

❖ Ch 3

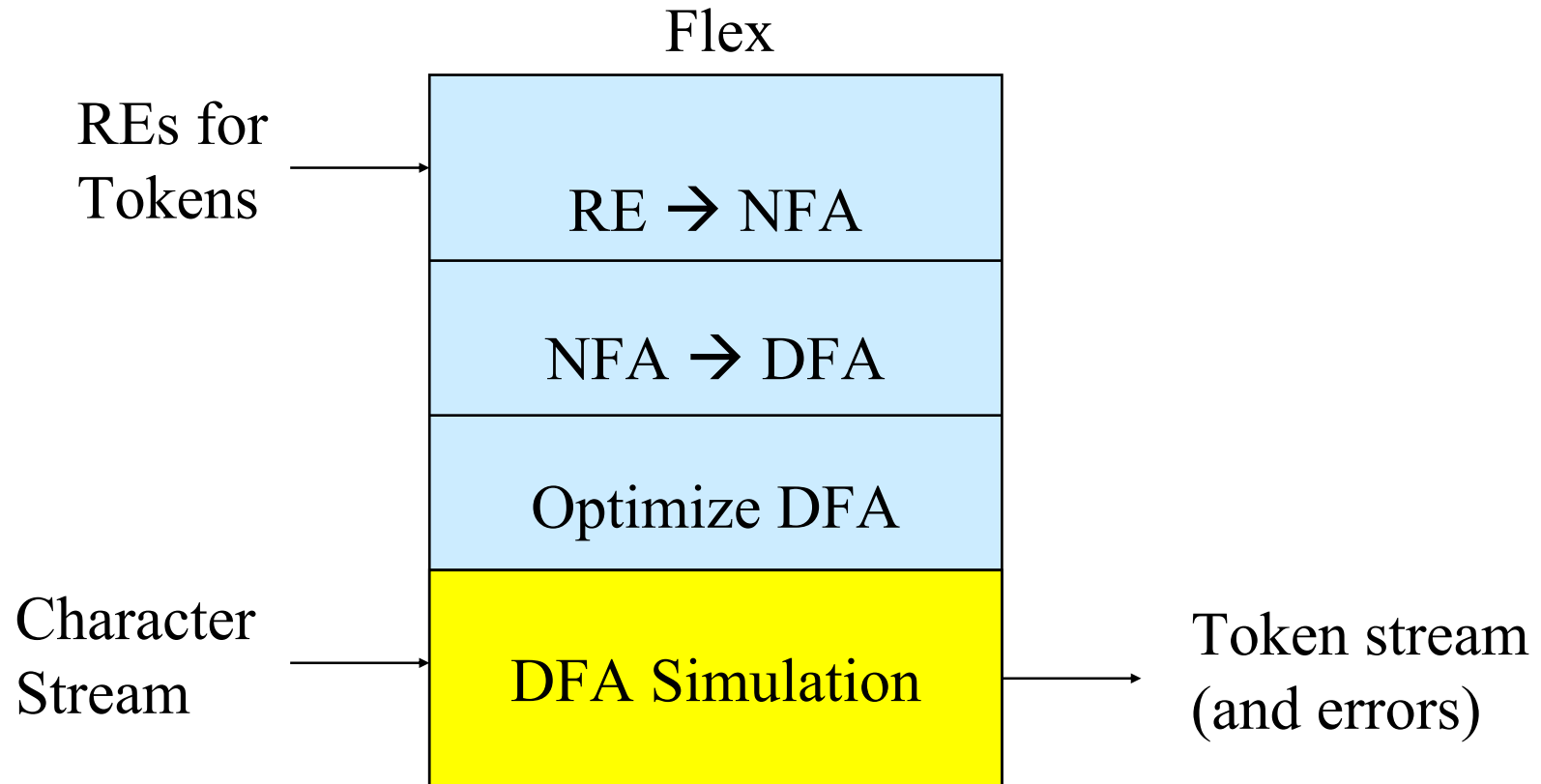
- » Read carefully, more closely follows lecture
- » Go over examples

How Does Lex Work?



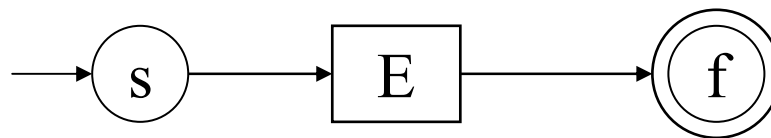
Some kind of DFAs and NFAs
stuff going on inside

How Does Lex Work?



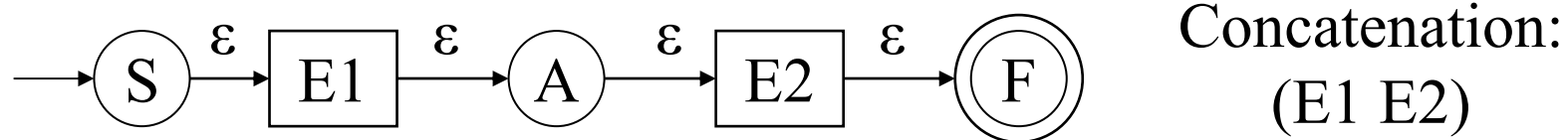
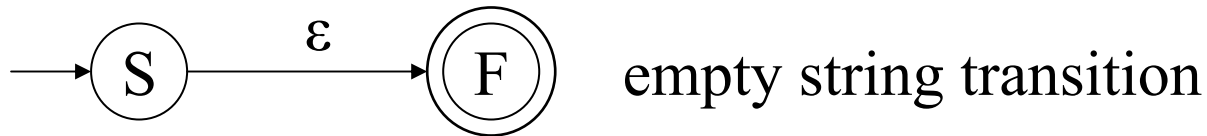
Regular Expression to NFA

- ❖ 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



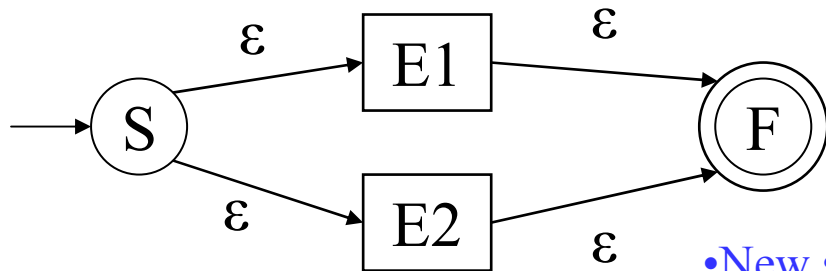
general machine

Thompson Construction



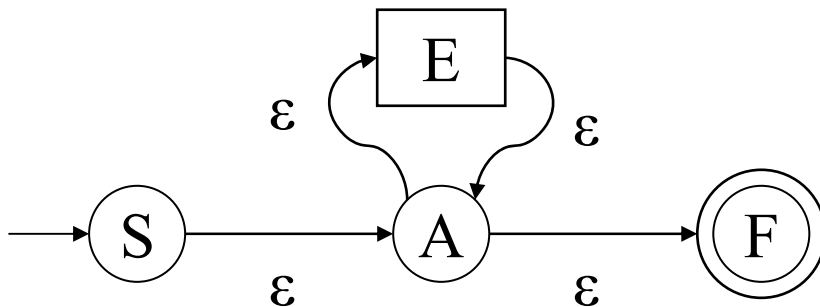
- New start state S ϵ -transition to the start state of E1
- ϵ -transition from final/accepting state of E1 to A, ϵ -transition from A to start state of E2
- ϵ -transitions from the final/accepting state E2 to the new final state F

Thompson Construction - Continued



Alteration: $(E1 \mid E2)$

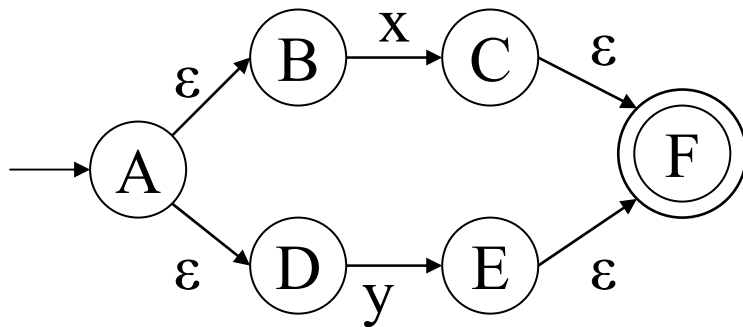
- New start state S ϵ -transitions to the start states of E1 and E2
- ϵ -transitions from the final/accepting states of E1 and E2 to the new final state F



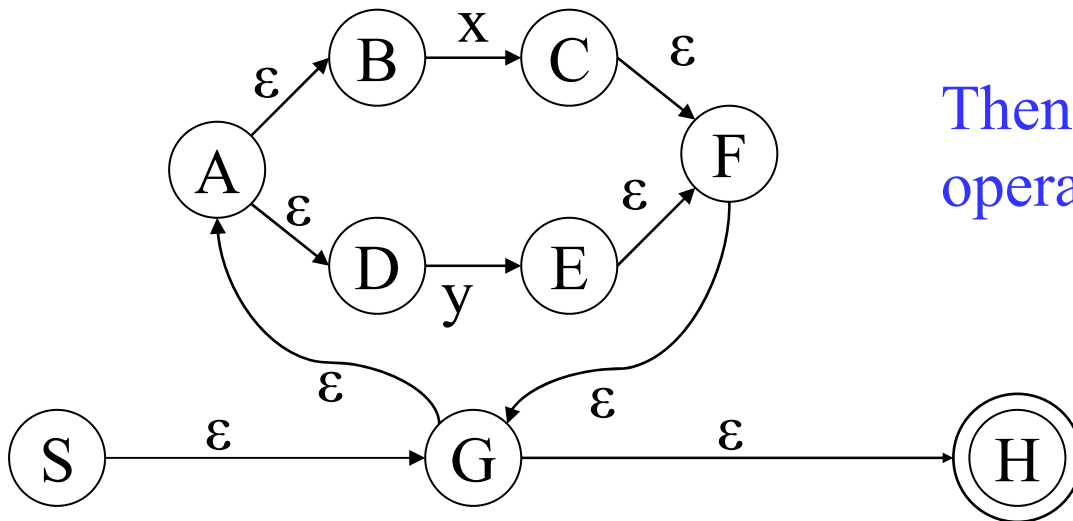
Closure: (E^*)

Thompson Construction - Example

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



First create NFA for $(x | y)$



Then add in the closure operator

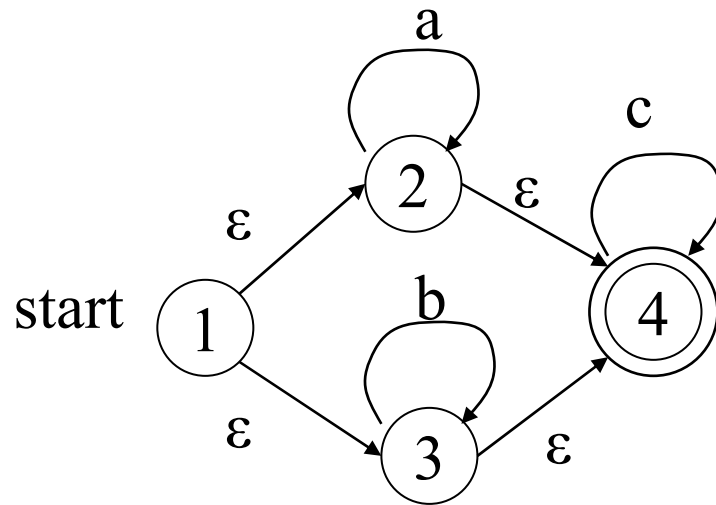
Class Problem

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

NFA to DFA

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

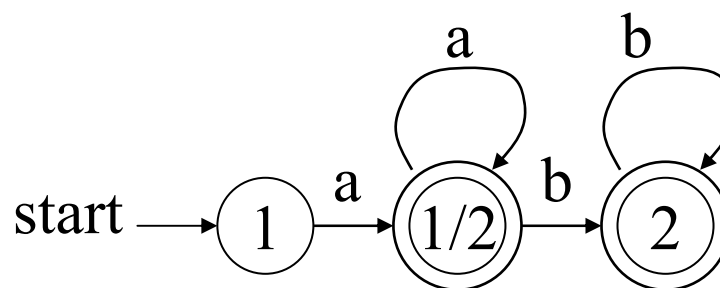
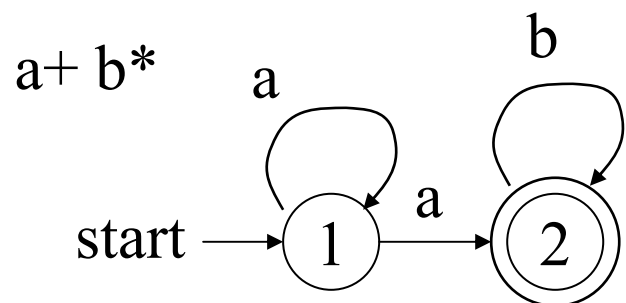
$(a^* | b^*) c^*$



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



$(1,a) \rightarrow 1$ or 2 , create new state $1/2$

$(1/2,a) \rightarrow 1/2$

$(1/2,b) \rightarrow 2$

$(2,a) \rightarrow \text{ERROR}$

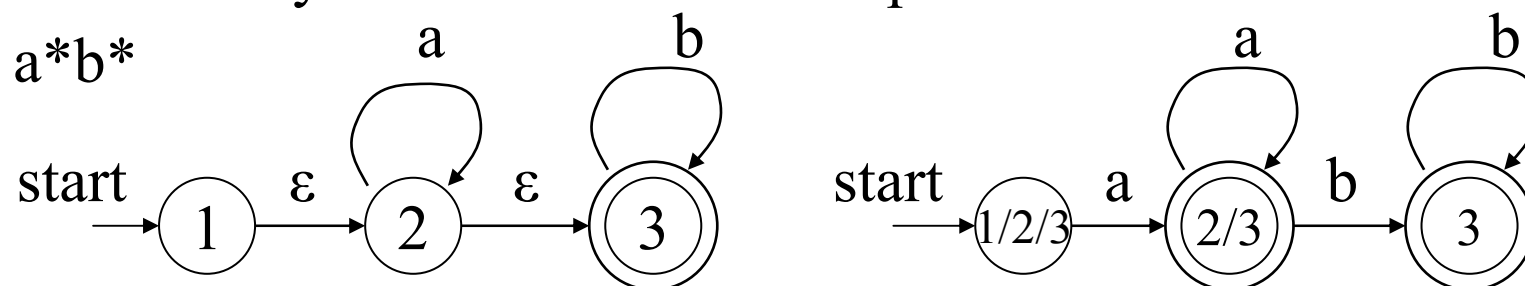
$(2,b) \rightarrow 2$

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

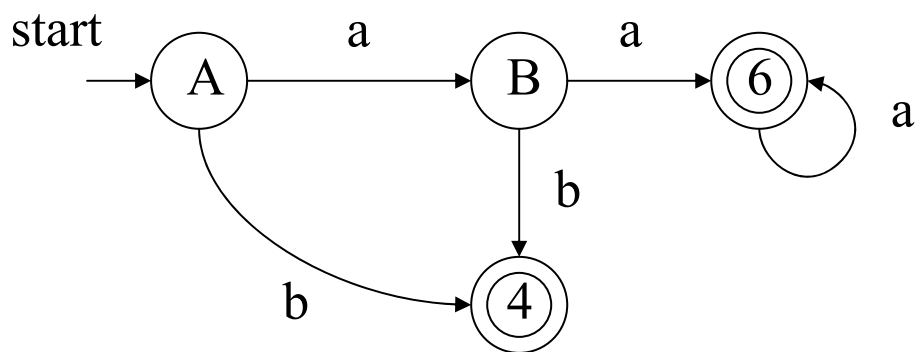
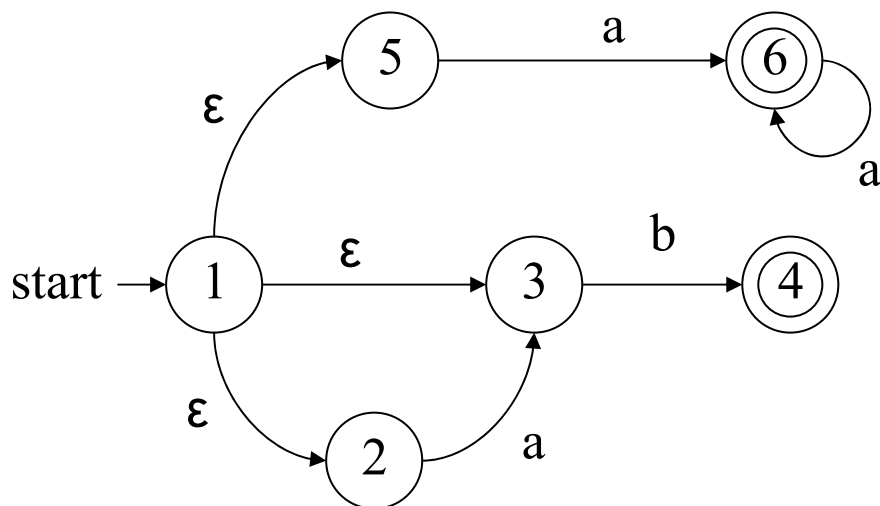
(1/2/3, a) \rightarrow 2/3

(1/2/3, b) \rightarrow 3

(2/3, a) \rightarrow 2/3

(2/3, b) \rightarrow 3

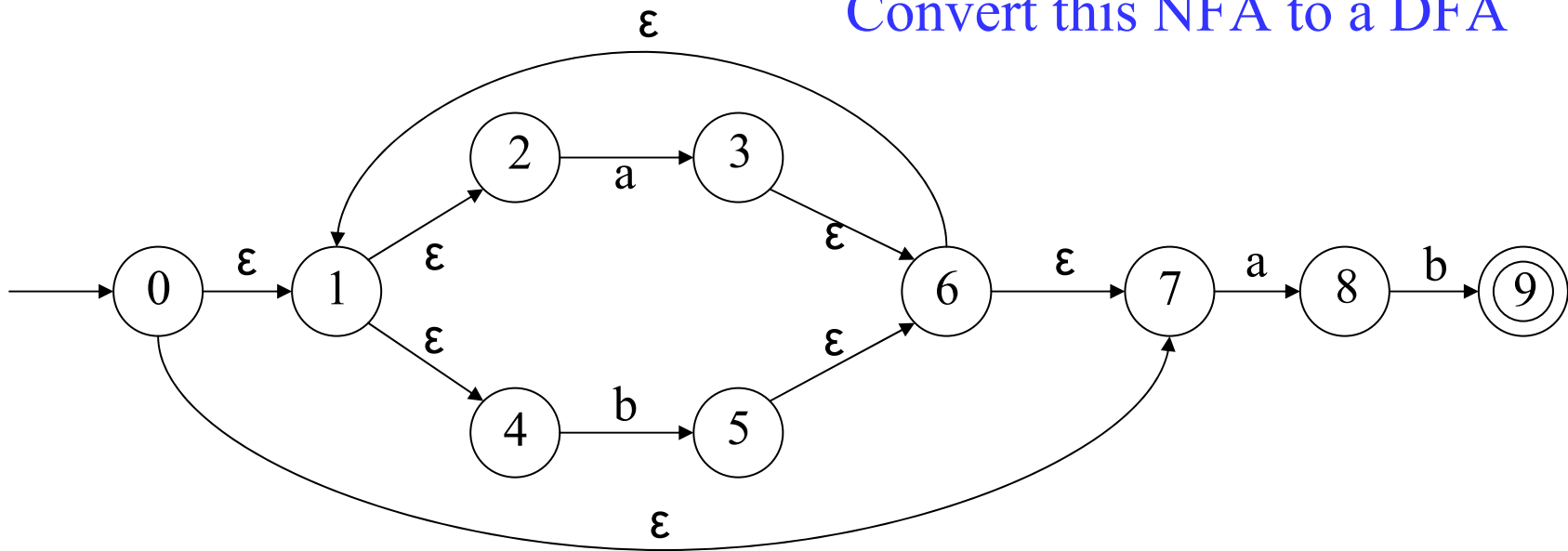
NFA to DFA - Example



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

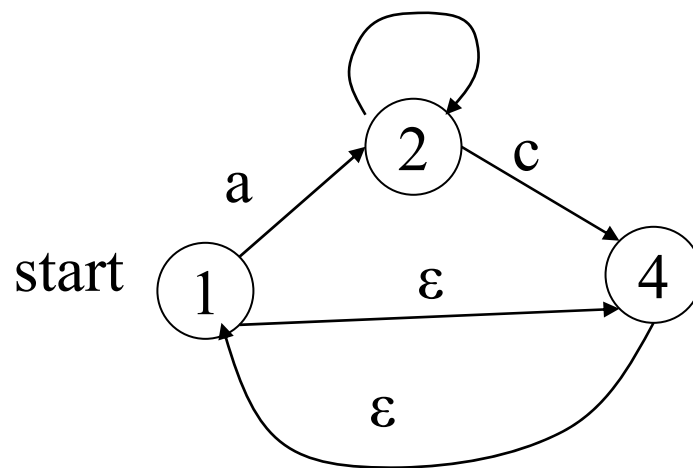
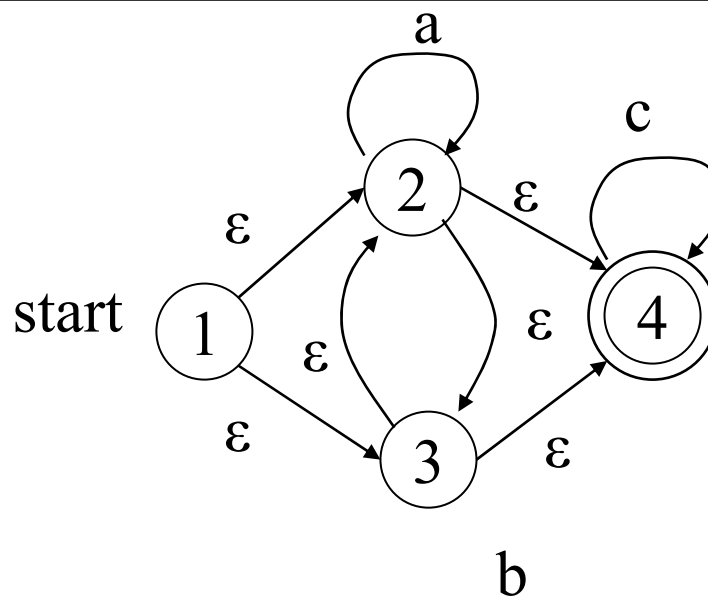
Class Problem

Convert this NFA to a DFA



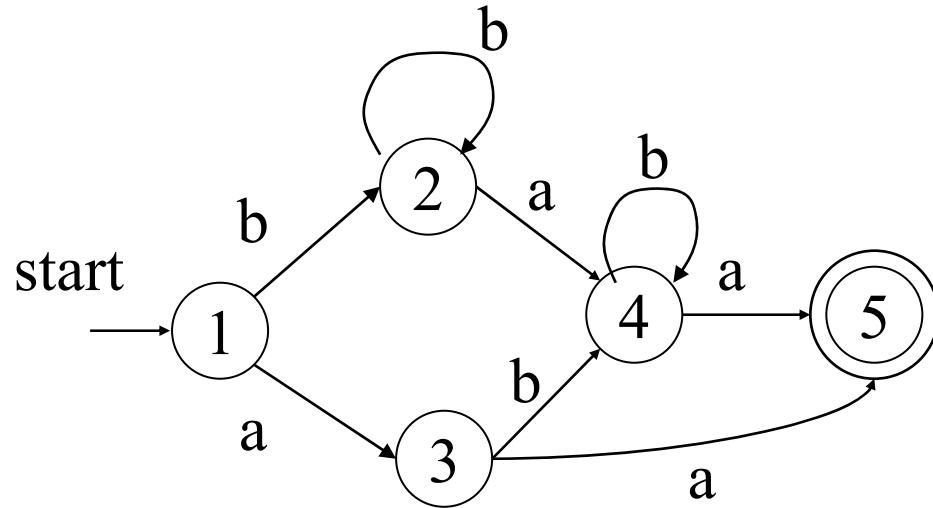
NFA to DFA Optimizations

- ❖ 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

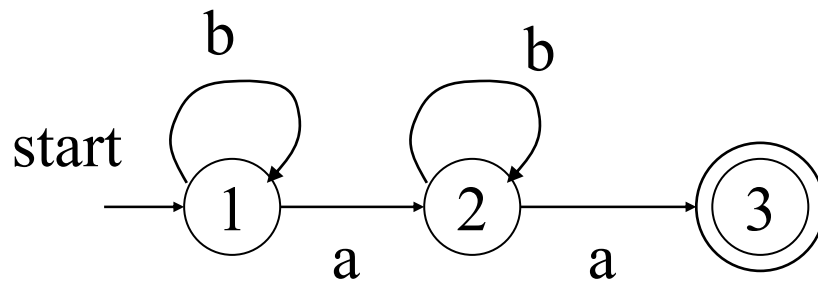


State Minimization

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

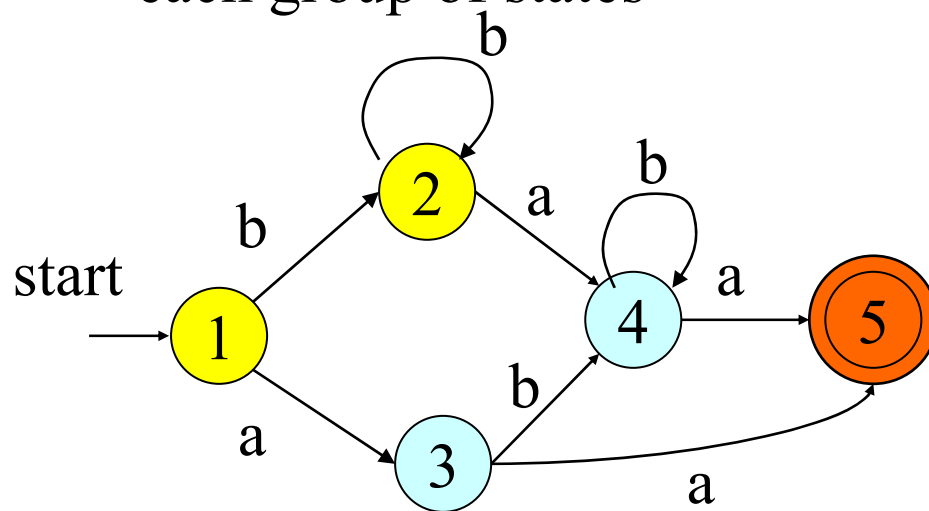


Both DFAs accept b^*ab^*a



State Minimization (2)

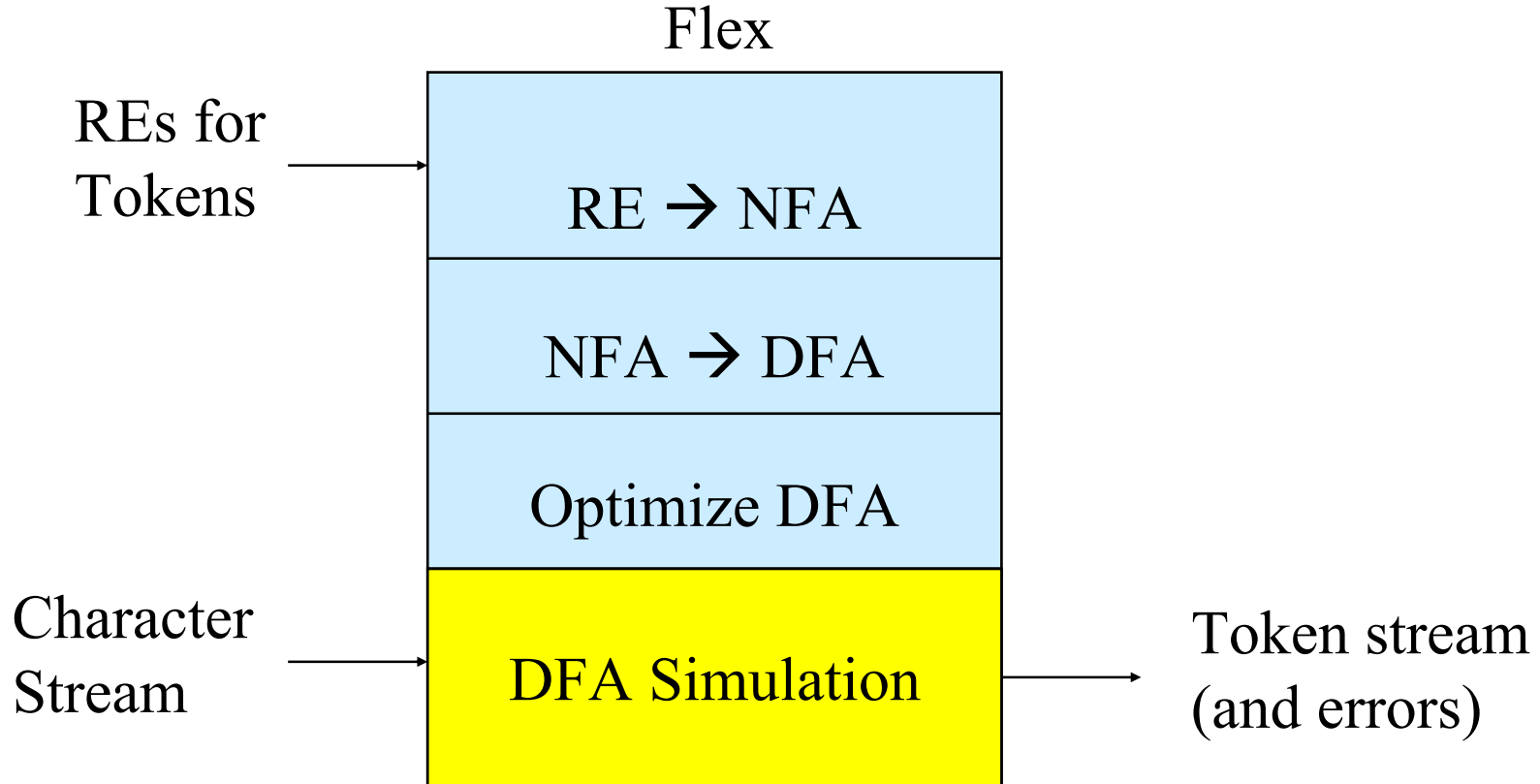
- ❖ Idea – find groups of equivalent states and merge them
 - » All transitions from states in group G1 go to states in another group G2
 - » Construct minimized DFA such that there is 1 state for each group of states



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

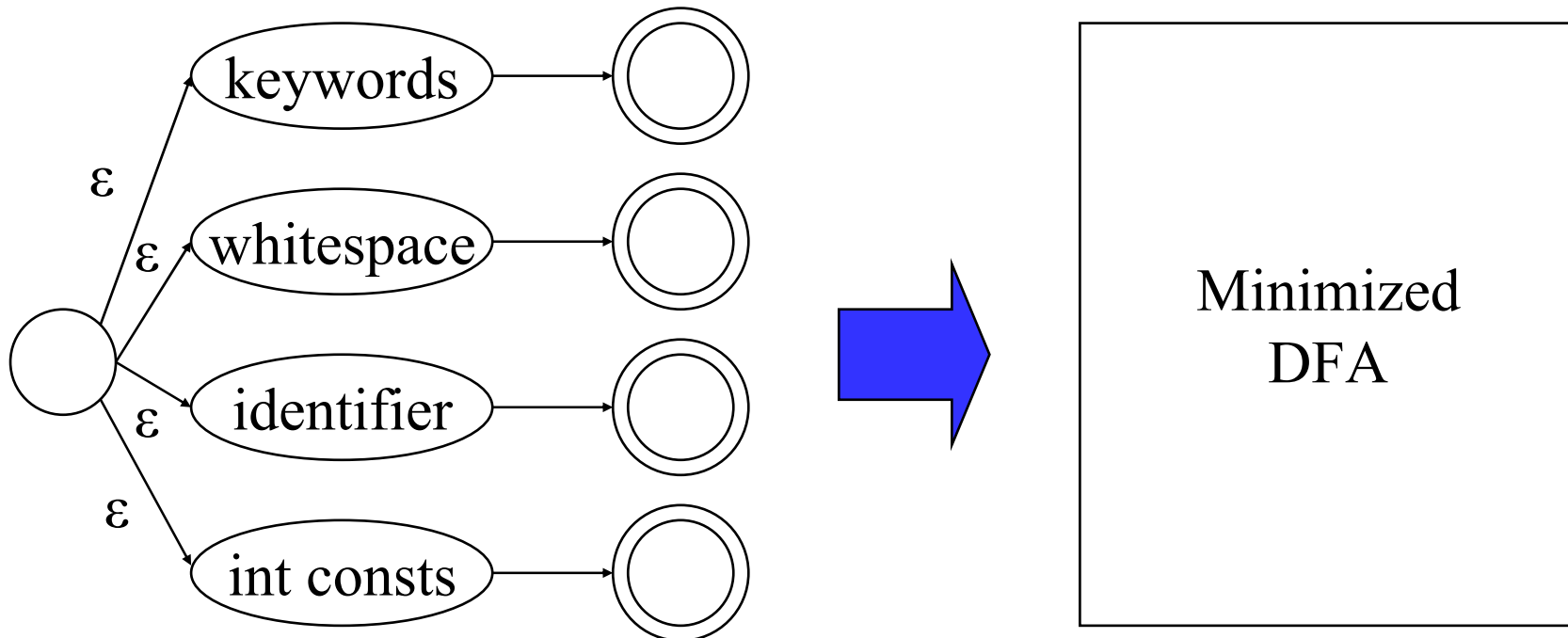
```
trans_table[NSTATES][NINPUTS];
accept_states[NSTATES];
state = INITIAL;

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

Project 1

- ❖ P1 handout available under projects link on course webpage
 - » Base file has a bunch of links, so make sure you get everything
- ❖ Your job is to write a lexical analyzer and parser for a language called C - -
 - » Flex and bison will be used to construct these
 - We'll talk about bison next week
 - Can start on the flex part immediately
 - » You will produce a stylized output explained in the Spec
 - » Detect various simple errors
 - » Due Wednes, 9/28

C - - (A few of the Highlights)

❖ Subset of C

- » Allowed keywords: char, else, extern, for, while, if, int, return, void
- » So, no floating point, structs, unions, switch, continue, break, and a bunch of other stuff
- » All C punctuation/operators are supported (including ++) with the exception of ‘?:’ operators
- » No include files – manually declare libc/libm functions with externs
- » Only 1 level of pointers, ie `int *x`, not `int **x`

Project Grading

- ❖ You'll turn in 2 files
 - » `uniquename.l`, `uniquename.y`
- ❖ Details of grading still to be worked out
 - » But, as a rough estimate
 - $\text{Grade} = \text{explanation} * (\text{features} + \text{correctness})$
 - Correctness: do you pass the testcases, we provide some to you, but not all
 - Features: how much of the spec did you implement
 - Explanation: Interview, do you understand the concepts, can you explain the source code

Doing Your Own Work

- ❖ Each person should work independently on Project 1
 - » You are encouraged to help each other with flex/bison syntax, operation, etc.
 - » You can discuss the project, corner cases, etc.
 - » But the code should be yours
- ❖ We will not police this
 - » But, it will be obvious in the interview who did not write the code or did not understand what they wrote