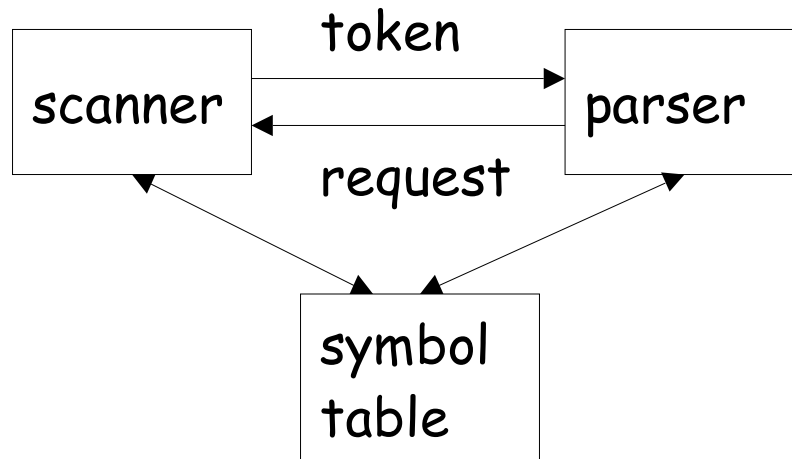


Scanners (Objectives)

- Given a regular expression, the student will be able to construct its corresponding NFA, DFA and scanner
- Given a regular expression, the student will be able to construct its corresponding flex description

Role of the Scanner

- token - name representing a class of character strings



NFAs

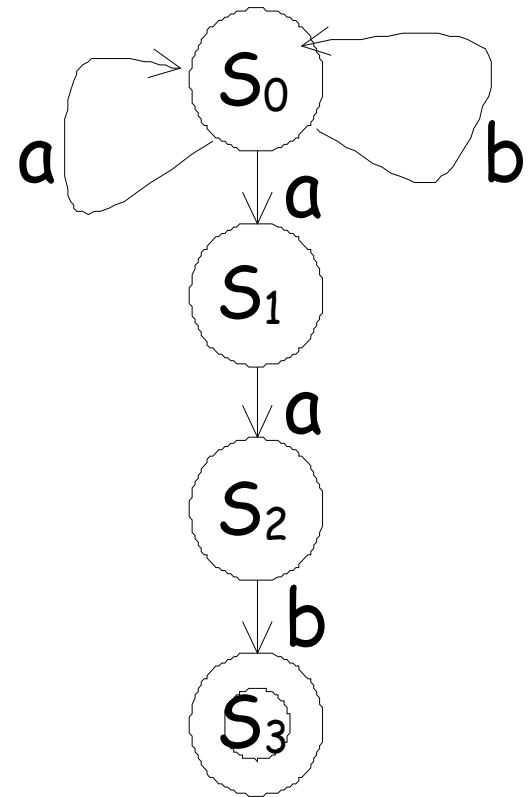
- NFA - nondeterministic finite automaton
 - S - a set of states
 - Σ - an alphabet
 - d - a transition function
 - s_0 - a start state
 - SF - a set of final states $SF \subseteq S$

Example NFA

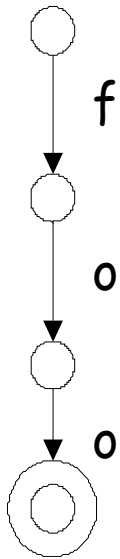
$$S = \{s_0, s_1, s_2, s_3\}$$

$$\Sigma = \{a, b\}$$

$$\delta = \{(s_0, a, s_0), (s_0, a, s_1), \\ (s_0, b, s_0), (s_1, a, s_2), \\ (s_2, b, s_3)\}$$



Scanners and NFAs



➤ Scanner Code

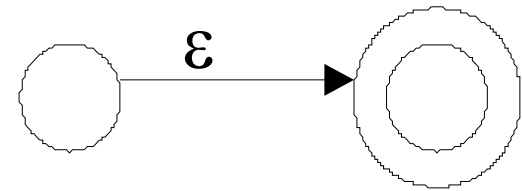
```
c = NextChar();  
if (c != 'f') then stop;  
else  
  c = NextChar();  
  if (c != 'o') then stop;  
  else  
    c = NextChar();  
    if (c != 'o') then stop;  
    else  
      report success;
```

Regular Expressions

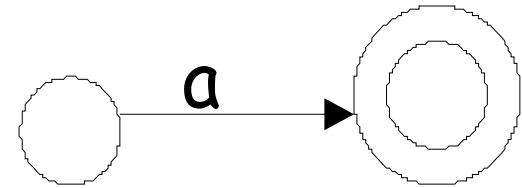
- Regular Expression (RE) Definitions
 - ϵ - the null symbol
 - Σ - the alphabet over which the RE is defined
 - if $a \in \Sigma$ then a is the RE denoting $\{a\}$
 - $L(r)$ is the language denoted by RE r
 - if r and s are REs then
 - $r|s$ is an RE denoting $L(r) \cup L(s)$
 - rs is an RE denoting $L(r) \circ L(s)$
 - r^* is an RE denoting $L(r)^*$
- RE for example $(a|b)^*aab$

Constructing NFAs from REs (Base cases)

- For RE ϵ construct

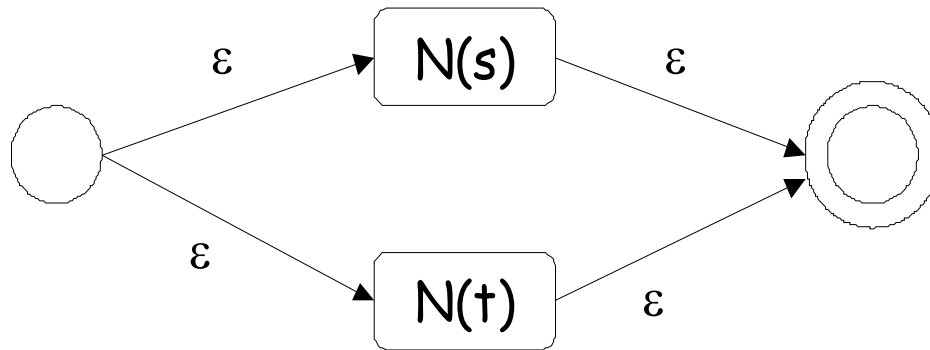


- For RE a construct



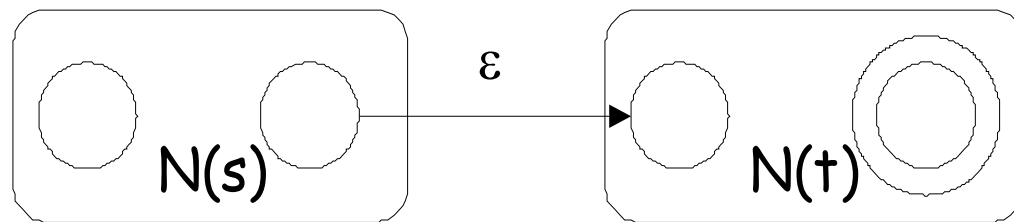
Constructing NFAs from REs (Or)

- Given NFAs $N(s)$ and $N(t)$ for REs s and t
The NFA for $s|t$ is



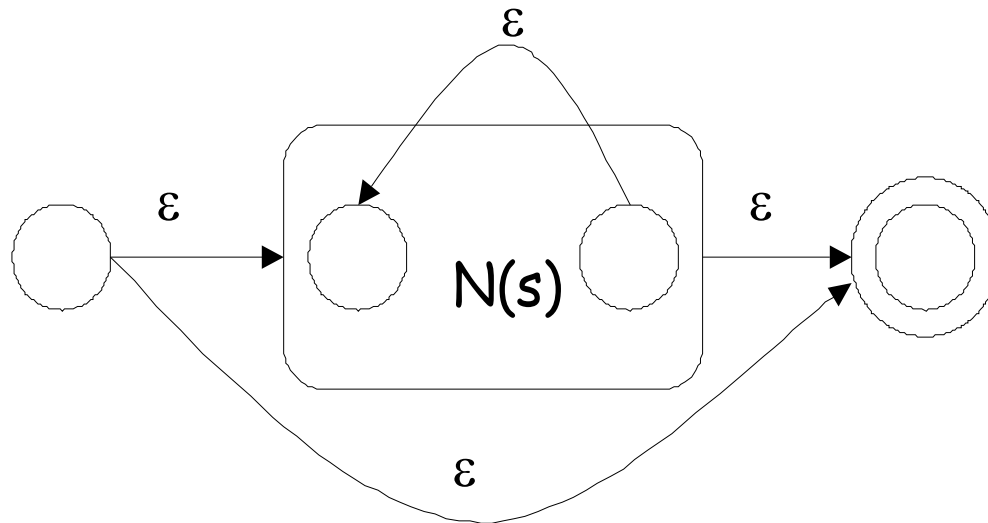
Constructing NFAs from REs (Concatenation)

- For RE $s|t$ construct the following NFA where s_0 of $N(s)$ is the start state and S_F of $N(t)$ are the end states and there is an ε -transition from S_F of $N(s)$ to s_0 of $N(t)$



Constructing NFAs from REs (Kleene Closure)

- For RE s^* construct the NFA



Example

- Construct an NFA for $(a|b)^*aab$

DFAs from NFAs

```
q0 ← ε-closure({s0})
Q ← {q0}
while ∃ unmarked qi ∈ Q {
  mark qi
  for each α ∈ Σ
    t ← ε-closure(move(qi, α))
    if t ∉ Q
      Q ∪ = {t}
      T[qi, α] ← t
}
```

```
ε-closure(T) {
  push states in T on stack
  C = T
  while stack not empty do
  {
    t = pop T
    for each t → u labeled ε
      if u ∉ C then {
        C ∪ = {u}
        push u onto stack
      }
  }
}
```

Example

- Convert NFA for $(a|b)^*aab$ to a DFA

DFA Minimization

- After conversion from an NFA to a DFA, there may be unnecessary states.
- We can minimize the number of states to reduce the space requirement of the DFA.
- Two states are said to be equivalent if they produce the same behavior on any input string.
- The algorithm will partition the set of states $S = \{s_1, s_2, \dots, s_n\}$ into partitions $P = \{p_1, p_2, \dots, p_m\}$ such that all states $s_i \in p_i$ are equivalent.

DFA Minimization

$T \leftarrow \{S_F, S - S_F\}$

repeat

$P \leftarrow T$

$T \leftarrow \emptyset$

for each set $p \in P$ {

for each $\alpha \in \Sigma$

partition p by α into p_1, \dots, p_k

$T \leftarrow T \cup p_1 \cup \dots \cup p_k$

}

until $T == P$

Example

- Minimize the DFA for $(a|b)^*aab$.

DFA to Scanner

- Perform minimization on the DFA after conversion from NFA (not done here)
- Emit **table driven** or direct code

char	a	b	other
class	class-a	class-b	other

state	q_0	q_1	q_2	q_3
class-a				
class-b				
other				

Code for Scanner

```
char = NextChar(); state = q0; done = false; str = "";
while not(done) do
  class = ClassTable[char];
  state = TransTable[class,state];
  switch (state) {
    case q0,q1,q2: lexeme += char;
                    char = NextChar(); break;
    case q4: char = NextChar;
              if (char == EOF) then
                token_type = VALID;
                done = true;
              endif
    case se: token_type = ERROR; done = true;
  }
  return token_type, lexeme;
end
```