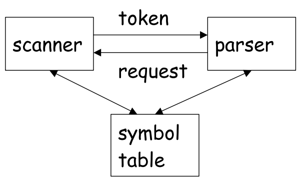
Scanners (Objectives)

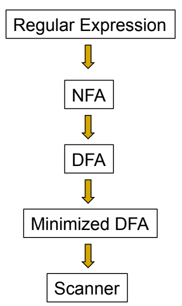
- Review Only
- Given a regular expression, the student will be able to construct its corresponding NFA, DFA and scanner

Role of the Scanner

- token - name representing a class of character strings



Regular Expression to Scanner

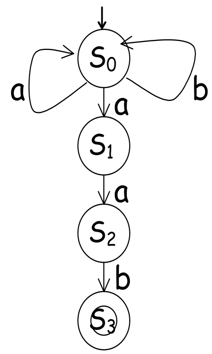


NFAs

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

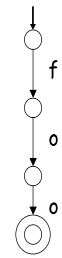
Example NFA

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



5

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;
```

6

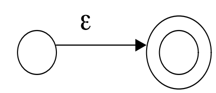
Regular Expressions

- 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 $L(r) \cup L(s)$ (alternation)
    - rs is an RE denoting $L(r) \circ L(s)$ (concatenation)
    - r^* is an RE denoting $L(r)^*$ (Kleen closure)
- RE for example $(a|b)^*aab$

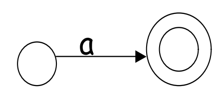
7

Constructing NFAs from REs (Thompson’s construction)

- For RE ε construct



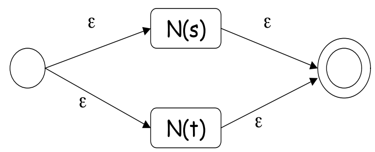
- For RE a construct



8

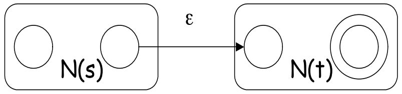
Constructing NFAs from REs (Alternation)

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



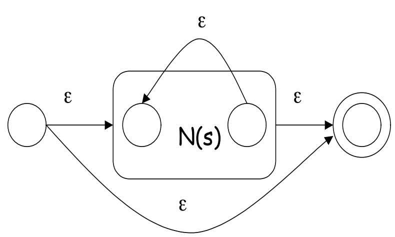
Constructing NFAs from REs (Concatenation)

- For RE st 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 (Kleen Closure)

- For RE s^* construct the NFA

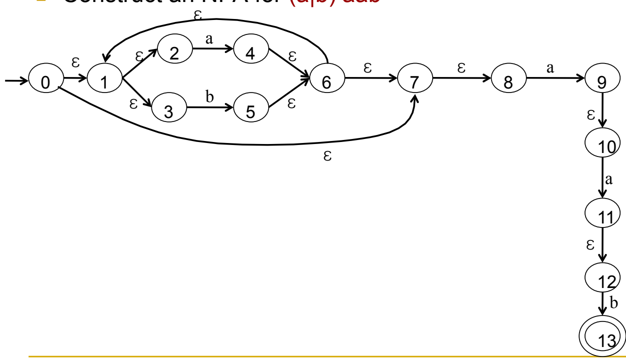


Example

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

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
      }
  }
  return C
}
```

move(q_i,α) = {s_k | □s_j □ q_i, s_j → s_k labeled α}

Example

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

Example

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

$q_0 = \varepsilon - closure(\{s_0\}) = \{s_0, s_1, s_7, s_2, s_3, s_8\} == \{s_0, s_1, s_2, s_3, s_7, s_8\} // start \quad state$

$T[q_0, a] = \varepsilon - closure(\{s_4, s_9\}) = \{s_4, s_9, s_6, s_{10}, s_1, s_7, s_2, s_3, s_8\} == \{s_1, s_2, s_3, s_4, s_6, s_7, s_8, s_9, s_{10}\} = q_1$

$T[q_0, b] = \varepsilon - closure(\{s_5\}) = \{s_5, s_6, s_1, s_7, s_2, s_3, s_8\} == \{s_1, s_2, s_3, s_5, s_6, s_7, s_8\} = q_2$

$T[q_1, a] = \varepsilon - closure(\{s_4, s_9, s_{11}\}) = \{s_4, s_9, s_{11}, s_6, s_{10}, s_{12}, s_1, s_7, s_2, s_3, s_8\} == \{s_1, s_2, s_3, s_4, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}\} = q_3$

$T[q_1, b] = \varepsilon - closure(\{s_5\}) = q_2$

$T[q_2, a] = \varepsilon - closure(\{s_4, s_9\}) = q_1$

$T[q_2, b] = \varepsilon - closure(\{s_5\}) = q_2$

$T[q_3, a] = \varepsilon - closure(\{s_4, s_9, s_{11}\}) = q_3$

$T[q_3, b] = \varepsilon - closure(\{s_5, s_{13}\}) = \{s_5, s_{13}, s_6, s_1, s_7, s_2, s_3, s_8\} == \{s_1, s_2, s_3, s_5, s_6, s_7, s_8, s_{13}\} = q_4 // f\,inal \quad state$

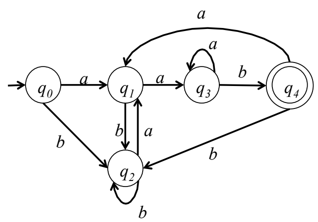
$T[q_4, a] = \varepsilon - closure(\{s_4, s_9\}) = q_1$

$T[q_4, b] = \varepsilon - closure(\{s_5\}) = q_2$

Example

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

	a	b
q_0	q_1	q_2
q_1	q_3	q_2
q_2	q_1	q_2
q_3	q_3	q_4
q_4	q_1	q_2



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 $S = \{s_1, s_2, \ldots, s_n\}$ into partitions $P = \{p_1, p_2, \ldots, p_m\}$ such that all states $s_i \in p_i$ are equivalent.

DFA Minimization

```
T ← {S_F, S – S_F}
repeat
  P ← T
  T ← ∅
  for each set p ∈ P {
    for each α ∈ Σ
        partition p by α into p_1, ..., p_k
    T ← T ∪ p_1 ∪ ... ∪ p_k
  }
until T == P
```


Example

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

Example

- Minimize the DFA for (a|b)*aab.
  - Initially, $p_0=\{q_4\}$ // final states
    $p_1=\{q_0, q_1, q_2, q_3\}$

p_1	a
q_0	p_1
q_1	p_1
q_2	p_1
q_3	p_1

No change

p_1	b
q_0	p_1
q_1	p_1
q_2	p_1
q_3	p_0

partition p_1
into $p_1=\{q_3\}$ and $p_2=\{q_0, q_1, q_2\}$

Example

- Minimize the DFA for (a|b)*aab.
  - $p_0=\{q_4\}$
    $p_1=\{q_3\}$
    $p_2=\{q_0, q_1, q_2\}$

p_2	a
q_0	p_2
q_1	p_1
q_2	p_2

partition p_2
into $p_2=\{q_1\}$ and $p_3=\{q_0, q_2\}$

Example

- Minimize the DFA for (a|b)*aab.
  - $p_0=\{q_4\}$
    $p_1=\{q_3\}$
    $p_2=\{q_1\}$
    $p_3=\{q_0, q_2\}$

p_3	a
q_0	p_2
q_2	p_2

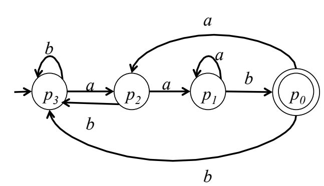
No change

p_3	b
q_0	p_3
q_2	p_3

No change

Example

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



DFA to Scanner

- Emit table driven or direct code

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

state	p_0	p_1	p_2	p_3
class-a				
class-b				
other				

Code for Scanner

```
char = NextChar(); state = start state; done = false; lexeme = “”;
while not(done) do
    class = ClassTable[char];
    state = TransTable[class,state];
    switch (state) {
      case p1,p2,p3: lexeme += char;
                     char = NextChar(); break;
      case p0: char = NextChar();
               if (char == EOF) then
                 token_type = VALID;
                 done = true;
               endif
      case se: token_type = ERROR; done = true;
    }
    return token_type, lexeme;
end
```


Real Issues

- Note that there is a rule to scan the longest-possible tokens, meaning you return only when the next character can't be used to continue the current token
  - the next character will generally need to be saved for the next token
- In some cases, you may need to peek at more than one character of look-ahead in order to know whether to proceed
  - In Pascal, for example, when you have a 3 and you a see a dot
    - do you proceed (in hopes of getting 3.14)?
      or
    - do you stop (in fear of getting 3..5)?

What else is hard?

- PL/I has no reserved words

  if then then then = else; else else = then;

- Fortran ignores blanks

  d   o10 i=1   ,2   5

  do 10 i = 1 .   25

Is Fortran really that hard to scan?

```
       INTEGERFUNCTIONA
       PARAMETER(A=6,B=2)
       IMPLICIT CHARACTER*(A-B)(C-D)
       INTEGER FORMAT(10), IF(10), DO9E1
       FORMAT(4H)=(3)
       FORMAT(4   )=(3)
       DO9E1=1
       DO9E1=1,2
       IF(X)=1
       IF(X)H=1
       IF(X)300,200
300    END
C      THIS IS A “COMMENT CARD”
       $FILE(1)
200    END
```

Example due to Dr. F.K. Zadeck, from Eng.
A Compiler, Cooper and Torczon

Practice Problem

- For the regular expression $(aa)^* \mid (ab)^*$, construct an NFA using the method discussed in class, convert that NFA and then minimize the resulting DFA.