Lexical Analysis - 2

- More regular expressions
- Finite Automata
  - NFAs and DFAs
- Scanners
- JLex - a scanner generator

Regular Expressions in JLex

Symbol	Meaning
.	Matches a single character (not newline)
*	Matches 0 or more copies of preceding RE
+	Matches 1 or more copies of preceding RE
?	Matches 0 or 1 occurrence of an RE
"…"	Everything it quotes is matches EXACTLY
^	Matches the beginning of a line
$	Matches the end of a line
[ ]	Character class = matches any character listed; [^] implies a match of any character NOT listed
( )	Groups a series of REs into a new RE

REs in JLex

Symbol - Meaning

{ }	**Control for repeated matching a specific number of times; a{1,3} means match 1,2, or 3 instances of a**
\	**Used to match a metacharacter or control character; \n matches newline; \* is the character ***
|	**Means match either the RE proceeding it OR the RE following it**
RE1/RE2	**Means match RE1 but only when followed by RE2; 0/1 will match the 0 in 01 but not in 02**

Exercise

- Problem: write an RE to match a quoted string such as “Hello”.
  - Need to decide if a quoted string can go across more than 1 line of text
  - Note: JLex REs are line-input-oriented, unlike formal REs
  - Also, JLex REs make the longest matches possible within a string of characters
  - If multiple REs are given to JLex and several match the same longest expression, the first matching RE is used.

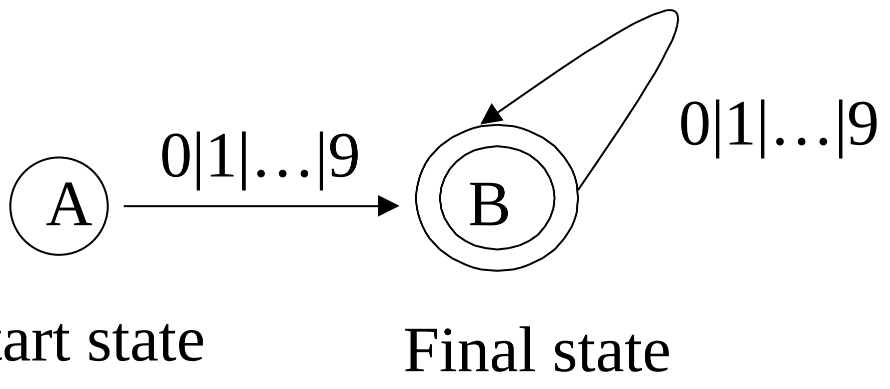
Finite Automata

- Automata that recognize strings defined by a regular expression
- (States, Input symbols, Transitions, Start_state, Set of Final_states)
  - Transitions between states occur on specific input symbols
- Deterministic automata have only 1 transition per state on a specific input and do not allow transition on the empty string

Finite Automata

- **Language *recognized* by automaton is set of strings it *accepts* by starting in the start state, using transitions corresponding to input symbols in the input string, and processing all input and finishing in a final state.**

RE for integers:
$[\ 0\text{-}9\ ]^{+}$



FAs

- **Nondeterministic finite automaton**

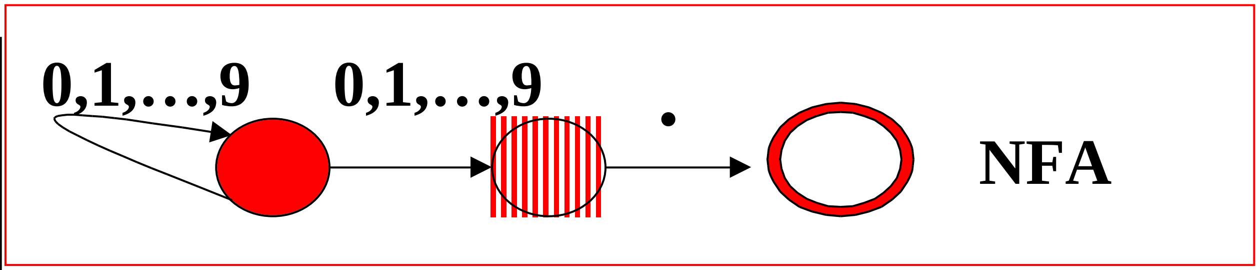
  <{states},
  {input symbols} (terminal symbols of a grammar)
  Transition function ((state,input)--> state),
  Start state
  {Final states}>

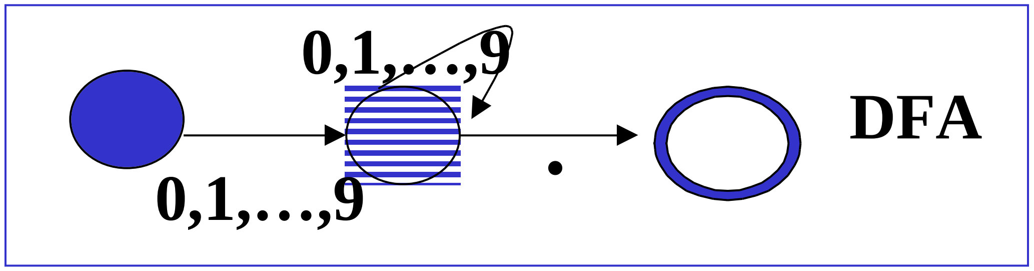
- **NFA allows more than 1 transition on the same input symbol and/or transitions on ε**
- **Deterministic FA allows only 1 transition per input symbol and no ε transitions**

FAs

- Theoretical results:
  - Set of languages recognizable by NFAs is same as those recognizable by DFAs.
  - There is an algorithm to check for equivalence of two languages recognized by 2 different FAs.

RE for reals:
$([0-9]^{+} \backslash. [0-9]^{*}) \mid ([0-9]^{*} \backslash. [0-9]^{+})$
Shown: $[0-9]^{+}$.





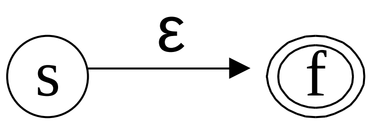
Practical FAs

- **Encode transitions as a table**
  - **Each column is an input symbol**
  - **Each row is a state**
  - **Entry at (*s1*,*i1*) is state to transition to when in state *s1* and see input *i1***
- **Scanner has to try to find longest match in input to a possible token**
  - **May have to look beyond end of token to do this!**

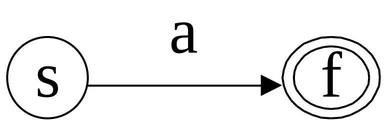
RE to NFA Conversion

- **Straightforward translation using composition operators of REs**

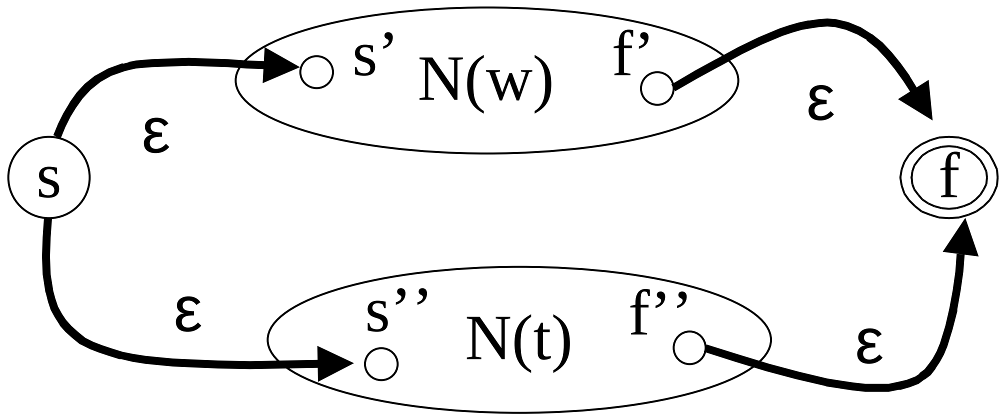
For RE ε,



For RE a,
terminal symbol,

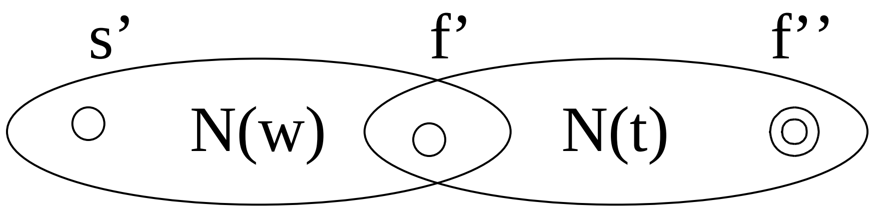


For w,t REs with corresponding NFAs N(w), N(t), w|t yields,

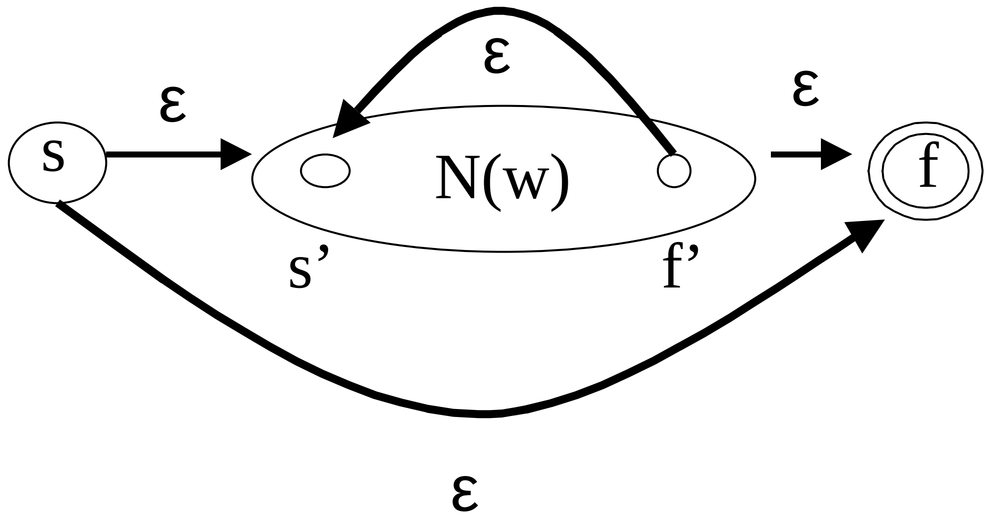


RE to NFA Conversion

For w,t REs with corresponding NFAs N(w), N(t), w t yields,



For w RE, w* yields,



For (w) RE, use N(w).

RE to NFA Conversion

- **These drawings follow the Aho, Sethi, Ullman Compiler text and are equivalent to those in Appel**
- **For w^+ use fact that $w^+ = w\ w^*$**
- **For w? use fact that w? = $w \mid \varepsilon$**
- **[abc] = $a \mid b \mid c$**
- **For “abc” use fact that “abc” = a b c**

How does an NFA compute?

- **Start off in the start state**
- **Compute set S of all states reachable on ε transitions.**
- **Given next input symbol is *a*, calculate set of states T, reachable as transition(s,*a*) where s   S**
- **Repeat steps 2,3 until input is exhausted. If final set of states contains a final state, then string has been recognized.**

NFA to DFA Conversion

- Deterministic computation is desirable if we want a write a scanner as a program
  - Need to convert NFA to equivalent DFA
  - Then can simulate DFA recognition process using tables in program to describe transitions
  - If process ends up in a final state, a token has been recognized

NFA to DFA Conversion

- *Intuition:* whenever there is an ε transition out of a state s, the NFA may go to any of the states reachable in this manner without consuming any input symbols. Call these states the *ε-closure* of state s.
  - By looking at ε-closures, we form sets of related states in the NFA; these become states in the corresponding DFA
  - Edges in the DFA correspond to sets of edges in the NFA (connecting different ε-closure sets of states)

NFA to DFA Conversion

- **DFA derived is *not* the most efficient (smallest possible) , but is usually of practical size**
- **There are ways of obtaining an optimal DFA by minimizing the numbers of states**

Conversion Algorithm

- **Need two primitive functions**
  - **ε−*closure*(T), for T a set of states in the NFA**
    - **Returns a set of NFA states reachable from state s   T by ε-transitions**
  - ***move*(T, *a*), for T a set of states in the NFA**
    - **returns a set of NFA states to which there is a transition on *a* from some NFA state s   T**
- **Build set of states (D) and transitions (Dtrans) for the DFA**

Conversion Algorithm

```
Assume all states in NFA are unmarked initially.
Let S = ε-closure(start state of NFA).
Let D = {S}.
while    an unmarked state T      D do
    Mark T;
        input symbols a do
    { U = ε-closure (move (T, a));
      if  U      D then {add unmarked U to D};
      Dtrans(T,a) = U;
    }
endwhile
```

Possible Problems

- Theoretically, for NFA having n states can get DFA with 2^n states, but this doesn't happen in practice.
- Token is recognized if the ending state of the DFA contains an original final state of the NFA.
  - In case of choice, use final state which represents the earliest rule in the list of productions for tokens

Optimal DFA

- There are algorithms for constructing the minimal (smallest) DFA
  - Idea:
    - Assume every state can transition on every input (can create an error state to do this).
    - Try to prove that computation starting at states T and S differs on at least 1 input. If cannot find such an input can merge S and T. Resulting state has the union of their transitions. (ASU, pp141ff)