DFAs, REs & Scanning

Finite set of states: $S$
Finite alphabet: $A$
Transition function: $T: S \times A \rightarrow S$
Start State: $s$
Set of final states $f_1..f_n$

$A, a: B$
$B, b: C$
$C, c: D$
$D, a: A$
$D, c: D$  
with
$s = A$  
and
$F = \{A, D\}$

$(ab(cc^*)a)^*$
DFAs, REs & Scanning

\[(ab(cc^*)a)^*\]
DFAs, REs & Scanning

\[(ab(cc^*)a)^*\]
DFAs, REs & Scanning

\[(ab(cc^*)a)^*\]
DFAs, REs & Scanning

\[(ab(cc^*)a)^*\]
DFAs, REs & Scanning

\[(ab(cc^*)a)^*\]
DFAs, REs & Scanning

(ab(cc*)a)*

abcccccabc
DFAs, REs & Scanning

(ab(cc*)a)*

ERROR, No Transition

abcccabcc
A DFA divides the set of all finite strings over its alphabet into two sets: accepted strings vs rejected

To ACCEPT the input string
Consume entire input string - and -
Finish in a FINAL state.

REJECT if either
No transition - or -
Finish in a non-Final state
fun getT oken: token =
    clearbuffer
    return A

fun A: token =
    if endstr {return "A"}
    getchar(x); bufferchar(x)
    if x = a {return B} else error

fun B: token =
    if endstr { error}
    getchar(x); bufferchar(x)
    if x = b {return C} else error

fun C: token
    if endstr { error}
    getchar(x); bufferchar(x)
    if x = c {return D} else error

fun D: token
    if endstr { return "D"}
    getchar(x): bufferchar(x)
    if x = c {return D} else
    if x = a {return A} else error
DFAs, REs & Scanning

DFA examples - Pascal tokens

Letter → Letter or Digit

" : " → " = "
Finite alphabet $A$ plus meta symbols

$\lambda$ the empty string

Finite strings of symbols from $A$ (catenations)

() grouping

| alternatives

* zero or more

+ one or more

“x” literal $x$

$A = \{a,b,c\}$

$$(abc^+a)^*$$

e.g.

“” empty string

abcccccaabcc

abca
DFAs, REs & Scanning

Regular Expression examples

\[ D = 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \]
\[ L = A \mid a \mid B \mid \ldots \mid z \]

ID1 = \( L (L \mid D)^* \)

Letter followed by letters and digits

ID2 = \( L (L \mid D \mid \_\_ (L \mid D))\)^*

Letter followed by letters and digits and underscores but no adjacent underscores or terminated by underscore.

PR = \( D^+ \_ . \ D^+ (\_ \mid \_E \_ (\_ \mid \_+ \mid \_-) \ D^+ ) \)

Pascal real numbers
### DFAs, REs & Scanning

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Regular Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Phi$</td>
<td>${ } ,$</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>${ &quot;&quot; } ,$</td>
</tr>
<tr>
<td>$ab...c$</td>
<td>${ &quot;ab...c&quot; } ,$</td>
</tr>
<tr>
<td>$A \mid B$</td>
<td>$R_A + R_B ,$</td>
</tr>
<tr>
<td>$A \cdot B$</td>
<td>${ab</td>
</tr>
<tr>
<td>$A^*$</td>
<td>${ &quot;&quot; } + {x..z</td>
</tr>
</tbody>
</table>

Where $a,b,...,c$ are in alphabet and $A,B$ are regular expressions

Where $+$ is set union and $R_A$ is the Regular Set for $A$

A set is regular if some regular expression denotes it.
Notes:

Not all sets are regular

E.g.
{ a...ab...b | same number of a’s as b’s }

Modern Computer languages have the property that their set of tokens is a regular set.
A finite automaton can be non-deterministic

Choice for transition or transition on no character read

\[ a | ab | ac \]
DFAs, REs & Scanning  RE to NFA pt 2

A | B

A *

Diagram showing the transition from DFAs, REs & Scanning to RE to NFA pt 2, illustrating the concepts of union and Kleene star operations.
States of the DFA are sets of NFA states

Proc Close = add to a state those states to which you can go on lambda transitions

Proc MakeDeterministic = add states if you can go on lambda or same symbol transitions
Start with initial state
Close it

Now, from state 12 we can reach states 3 or 4 or 5 on an “a” transition so:
Now, from state 345 we can reach 5 on an “a” transition or states 4 or 5 on a “b” transition so:
Now, from state 45 we can reach 5 on an “a” or “b” transition so:
We now have an equivalent DFA
DFAs, REs & Scanning

To automate the construction of a scanner

Define the tokens with a regular expression

Create the equivalent NFA

Construct the equivalent DFA

Program the DFA

Caveat: Then NFA -> DFA construction can require a LARGE number of states

The program LEX does this