What problem are we solving?

Translate a sequence of characters

if m = 0 then 1 else if m = 1 then 1 else fib (m - 1) + fib (m - 2)

into a sequence of tokens

IF, IDENT “m”, EQUAL, INT 0, THEN, INT 1, ELSE, IF, IDENT “m”, EQUAL, INT 1, THEN, INT 1, ELSE, IDENT “fib”, LPAREN, IDENT “m”, SUB, INT 1, RPAREN, ADD, IDENT “fib”, LPAREN, IDENT “m”, SUB, INT 2, RPAREN

implemented with some data type

type token =
  | INT of int| IDENT of string | LPAREN | RPAREN
  | ADD | SUB | EQUAL | IF | THEN | ELSE
  | ...
Regular expression $e$ over alphabet $\Sigma$

\[ e \rightarrow \phi \mid \varepsilon \mid a \mid e + e \mid ee \mid e^* \quad (a \in \Sigma) \]

\[ M(e) \subseteq \Sigma^* \]

\[ M(\phi) = \{ \} \]

\[ M(\varepsilon) = \{ \varepsilon \} \]

\[ M(a) = \{ a \} \]

\[ M(e_1 + e_2) = M(e_1) \cup M(e_2) \]

\[ M(e_1e_2) = \{ w_1w_2 \mid w_1 \in M(e_1), w_2 \in M(e_2) \} \]

\[ M(e^0) = \{ \varepsilon \} \]

\[ M(e^{n+1}) = M(ee^n) \]

\[ M(e^*) = \bigcup_{n \geq 0} M(e^n) \]
Regular Expression (RE) Examples

\[ M ((a + b)^* abb) = \]
\[ \{abb, aabb, baabb, aaabb, ababb, baabb, bbabb, aaaaabb, \ldots \} \]

\[ M ((\Xi + \otimes)^* \Xi \otimes \otimes) = \]
\[ \{\Xi \otimes \otimes, \Xi \Xi \otimes \otimes, \otimes \Xi \Xi \Xi \otimes \otimes, \]
\[ \Xi \Xi \Xi \Xi \otimes \otimes, \Xi \otimes \Xi \otimes \otimes, \otimes \Xi \Xi \Xi \otimes \otimes, \]
\[ \otimes \otimes \Xi \otimes \otimes, \Xi \Xi \Xi \Xi \Xi \otimes \otimes, \ldots \} \]
Review of Finite Automata (FA)

\[ M = (Q, \Sigma, \delta, q_0, F) \]

- \( Q \): states
- \( \Sigma \): alphabet
- \( q_0 \in Q \): start state
- \( F \subseteq Q \): final states

\[ \forall q \in Q, a \in \Sigma, \delta(q, a) \in Q \]

for deterministic FA (DFA)

\[ \forall q \in Q, a \in (\Sigma \cup \{\varepsilon\}), \delta(q, a) \subseteq Q \]

for nondeterministic FA (NFA)
An NFA accepting

\[ M(a^* + b^* + caa^* + cbb^*) \]
A bit of notation

For deterministic FA.

\[ \varepsilon \]
\[ q \rightarrow q \]

\[ aw \]
\[ q_1 \rightarrow q_3 \quad \text{if} \quad \delta(q_1, a) = q_2 \quad \text{and} \quad q_2 \rightarrow q_3 \]

\[ L(M) = \{ w \mid \exists q \in F, q_0 \rightarrow w \} \]

For nondeterministic FA.

\[ \varepsilon \]
\[ q \rightarrow q \]

\[ w \]
\[ q_1 \rightarrow q_3 \quad \text{if} \quad q_2 \in \delta(q_1, \varepsilon) \quad \text{and} \quad q_2 \rightarrow q_3 \]

\[ aw \]
\[ q_1 \rightarrow q_3 \quad \text{if} \quad q_2 \in \delta(q_1, a) \quad \text{and} \quad q_2 \rightarrow q_3 \]

\[ L(M) = \{ w \mid \exists q \in F, q_0 \rightarrow w \} \]
A regular expression. A nondeterministic FA accepting $M(e)$ with a single final state.

The construction is done by induction on the structure of $e$. 
$N(\emptyset) =$ \hspace{1cm} \begin{array}{c}
\text{\circle{\hspace{1cm} q_0 \hspace{1cm} q_1}}
\end{array}

$N(\varepsilon) =$ \hspace{1cm} \begin{array}{c}
\text{\circle{\hspace{1cm} q_0 \hspace{1cm} q_1}}
\end{array}

$N(\alpha) =$ \hspace{1cm} \begin{array}{c}
\text{\circle{\hspace{1cm} q_0 \hspace{1cm} q_1}}
\end{array}$
\[ N(e_1 + e_2) = \]
\[ N(e_1 e_2) = \]

\[ N(e_1) \quad N(e_2) \]
\[ N(e^*) = \]

The diagram shows a non-deterministic finite automaton (NFA) with transitions labeled by \(\varepsilon\) (epsilon) arrows.
$N(((a + b)^* \, abb))$
Review of NFA $\rightarrow$ DFA

\[ M = (Q, \Sigma, \delta, q_0, F) \]

\[ M' = (Q', \Sigma, \delta', q'_0, F') \]

\[ Q' = \{S \mid S \subseteq Q\} \]

\[ \varepsilon - \text{closure}(S) = \{q' \in Q \mid \exists q \in S, q \xrightarrow{\varepsilon} q'\} \]

\[ \delta'(S, a) = \varepsilon - \text{closure}(\{q' \in \delta(q, a) \mid q \in S\}) \]

\[ q'_0 = \varepsilon - \text{closure}\{q_0\} \]

\[ F' = \{S \subseteq Q \mid S \cap F \neq \emptyset\} \]
How do we compute $\varepsilon$−\textit{closure}(S)?

$\varepsilon$−\textit{closure}(S):

push all elements of $S$ onto a stack

result := $S$

while stack not empty

pop $q$ off the stack

for each $u \in \delta(q, \varepsilon)$

if $u \notin \text{result}$

then result := $\{u\} \cup \text{result}$

push $u$ on stack

return result

Look familiar?
It’s just a version of transitive closure!
$DFA(N((a + b)^* a bb))$
Given $e$ and $w$, is $w \in L(e)$?

Solution: construct NFA from $e$, then DFA, then run the DFA on $w$.

But is this a solution to the “lexing problem? No!
Something closer to the “lexing problem”

Given $e_1, e_2 \cdots, e_k$ and $W$

find $(i_1, w_1), (i_2, w_2), \ldots, (i_n, w_n)$ so that

$w = w_1w_2\ldots w_n$ and $\forall i \exists j \ w_i \in L(e_{i,j})$

and what else?

The expressions are ordered by priority. Why? Is “if” a variable or a keyword? Need priority to resolve ambiguity (so “if” matched keyword RE before identifier RE.

We need to do a longest match. Why? Is “ifif” a variable or two “if” keywords?
Given \( e_1, e_2 \cdots, e_k \) and \( W \)

find \( (i_1, w_1), (i_2, w_2), \ldots, (i_n, w_n) \) so that

1) \( w = w_1 w_2 \cdots w_n \)

2) \( \forall j, w_j \in L(e_{i_j}) \)

3) \( w_j \in L(e_s) \rightarrow i_j \leq s \) (priority rule)

4) \( \forall j : \forall u \in \text{prefix} \ (w_{j+1} w_{j+2} \cdots w_n), u \neq \varepsilon \)

\[ \rightarrow \forall s, w_j, u \notin L(e_s) \] (longest match)
**Keyword: if**

This FA is really shorthand for:

```
1 \(\xrightarrow{i} 2\) \(\xrightarrow{f} 3\)
```

```
1 \(\xrightarrow{i} 2\) \(\xrightarrow{f} 3\)
```

```
\(\Sigma-\{i\}\) \(\Sigma-\{f\}\) \(\Sigma\)
```

```
\(\text{"dead state"} 0\)
```

\(\Sigma\)
## Define Tokens with Regular Expressions (Finite Automata)

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Finite Automata</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keyword:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>if</td>
<td>![Finite Automata Diagram for &quot;if&quot;]</td>
<td>KEY(IF)</td>
</tr>
<tr>
<td>then</td>
<td>![Finite Automata Diagram for &quot;then&quot;]</td>
<td>KEY(then)</td>
</tr>
<tr>
<td><strong>Identifier:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[a-zA-Z][a-zA-Z0-9]*</td>
<td>![Finite Automata Diagram for &quot;Identifier&quot;]</td>
<td>ID(s)</td>
</tr>
<tr>
<td>Regular Expression</td>
<td>Finite Automata</td>
<td>Token</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
<tr>
<td>number: [0-9][0-9]*</td>
<td><img src="image" alt="Diagram" /></td>
<td>NUM(n)</td>
</tr>
<tr>
<td>real: (([0-9]+ \cdot \ [0-9]*)</td>
<td><img src="image" alt="Diagram" /></td>
<td>NUM(n)</td>
</tr>
<tr>
<td>| (([0-9]* \cdot \ [0-9]+)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
No Tokens for “White-Space”

White-space with one line comments starting with %

![Diagram](image)
Constructing a Lexer

**INPUT:** \( e_1, e_2, \ldots, e_k \)

an **ordered** list of regular expressions

Highest priority first, lowest last

\[ NFA_1, NFA_2, \ldots, NFA_k \]

use priority \( NFA \)

\( DFA \)

minimize \( DFA \) (we have not discussed minimization)
(1) **Keyword**: then

(2) **Ident**: [a-z][a-z]*

(2) **White-space**: ' ' 
What about longest match?

Start in initial state, Repeat:
1) read input until dead state is reached. Emit token associated with last accepting state.
2) reset state to start state.

| = current position, $ = EOF

Input

|then thenx$ 1 0
|then thenx$ 2 2
|then thenx$ 3 3
|then thenx$ 4 4
|then thenx$ 5 5
|then thenx$ 0 5 EMIT KEY(THEN)
|then thenx$ 1 0 RESET
|then thenx$ 7 7
|then thenx$ 0 7 EMIT WHITE(' ')  
|then thenx$ 1 0 RESET
|then thenx$ 2 2
|then thenx$ 3 3
|then thenx$ 4 4
|then thenx$ 5 5
|then thenx$ 6 6
|then thenx$ 0 6 EMIT ID(thenx)