Input: Token Spec
- List of regular expressions (RE) in priority order

Output: Lexer
- Reads an input stream and breaks it up into tokens according to REs

Algorithm
- Convert REs into non-deterministic finite automata (NFA)
- Convert NFA to DFA
- Convert DFA into transition table
Lexer Automatic Construction: Example

- RE for tokens:
  \[(a|ab)\]

- NFA:

- DFA:

- Transition Table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Error</td>
</tr>
<tr>
<td>1</td>
<td>Error</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Error</td>
<td>Error</td>
</tr>
</tbody>
</table>
Lexer Automatic Construction: Example

Token Specification

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(ab)</td>
<td>{Action 1}</td>
</tr>
<tr>
<td>(aаб)</td>
<td>{Action 2}</td>
</tr>
<tr>
<td>(a+)</td>
<td>{Action 3}</td>
</tr>
</tbody>
</table>

NFA

\[
\begin{array}{c}
q_0 \quad a \quad \epsilon \\
q_1 \quad a \quad q_2 \\
q_3 \\
q_4 \quad a \quad q_5 \\
q_6 \quad a \quad q_7 \\
q_8 \quad a \\
q_9 \\
\end{array}
\]

DFA

\[
\begin{array}{c}
s_0 \quad a \quad s_1 \\
s_1 \quad a \quad s_3 \\
s_2 \quad a \quad s_3 \\
s_3 \quad a \quad s_4 \\
s_4 \quad a \quad s_4 \\
\end{array}
\]

Example:
Input: \(aab\)

- \(s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_2\)

\(\Sigma = \{a, b\} \)
Theorem
A language $L$ can be described by regular expression if and only if $L$ is the language accepted by a finite automaton.

Algorithms:

- Regular expression $\Rightarrow$ Automaton
  - important for lexer construction
- Automaton $\Rightarrow$ Regular expression
  - interesting method in formal languages theory
• Build the finite automaton inductively, based on the definition of regular expressions
Alternation $R_1 \mid R_2$

Concatenation $R_1 \cdot R_2$

Final States
no final anymore
Alternation $R^*$

Final States
no final anymore
Exercise

Question

- Construct an NFA for the regular expression \((ab)^* \mid b^*\)
Exercise

Question

- Construct an NFA for the regular expression \((ab)^* \mid b^*\)

Answer
DFA Minimization

- Generated DFAs may have a large number of states
- **DFA Minimization**: Converts a DFA to another DFA that:
  - recognizes the same language
  - has a minimum number of states
- Increases time/space efficiency

Both DFAs accept: \(((a \mid b)b \ast a)^*\)
For every regular language \( L \) there exists a unique minimal DFA that recognizes \( L \)
- uniqueness up to renaming of states (isomorphism)

Minimal DFA can be found mechanically
DFA Minimization: Example

- Remove unreachable states: there is no path from initial state to $q_3$

$\Sigma = \{a, b\}$
DFA Minimization: Example

- Remove unreachable states: there is no path from initial state to $q_3$

$\Sigma = \{a, b\}$
DFA Minimization: Example

- Remove unreachable states: there is no path from initial state to $q_3$
- $q_2$, $q_6$ are both accepting sinks with self-loop for any character in $\Sigma$
- Any string reaches $q_2$ or $q_6$ is guaranteed to be accepted later
- $q_2$ and $q_6$ are equivalent states: we can unify them

$\Sigma = \{a, b\}$
DFA Minimization: Example

- Remove unreachable states: there is no path from initial state to $q_3$
- $q_2$, $q_6$ are both accepting sinks with self-loop for any character in $\Sigma$
- Any string reaches $q_2$ or $q_6$ is guaranteed to be accepted later
- $q_2$ and $q_6$ are equivalent states: we can unify them

$\Sigma = \{a, b\}$
DFA Minimization: Example

- Remove unreachable states: there is no path from initial state to $q_3$
- $q_2$, $q_6$ are both accepting sinks with self-loop for any character in $\Sigma$
- Any string reaches $q_2$ or $q_6$ is guaranteed to be accepted later
- $q_2$ and $q_6$ are equivalent states: we can unify them

- If DFA is in $q_1$ or $q_5$:
  - if next character is $a$, it forever accepts in both states
  - if next character is $b$, it forever rejects in both states
- $q_1$ and $q_5$ are equivalent states: we can unify them

\[ \Sigma = \{a, b\} \]
DFA Minimization: Example

- Remove unreachable states: there is no path from initial state to $q_3$
- $q_2$, $q_6$ are both accepting sinks with self-loop for any character in $\Sigma$
- Any string reaches $q_2$ or $q_6$ is guaranteed to be accepted later
- $q_2$ and $q_6$ are equivalent states: we can unify them
- If DFA is in $q_1$ or $q_5$:
  - if next character is $a$, it forever accepts in both states
  - if next character is $b$, it forever rejects in both states
- $q_1$ and $q_5$ are equivalent states: we can unify them

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

Intuition

- Two states are equivalent if all subsequent behavior from those states is the same
- Equivalent states may be unified without affecting DFA’s behavior

Definition

- We say that states $p$ and $q$ are equivalent if for all $w$: $\hat{\delta}(p, w)$ is an accepting state iff $\hat{\delta}(q, w)$ is an accepting state
- $\hat{\delta}$ is the transition function extended for words
DFA Minimization: Procedure

- Write down all pairs of states as a table.
- Every cell in the table denotes if corresponding states are equivalent.
- The table is initially unmarked.
- We mark pair $(p_i, p_j)$ when we discover $p_i$ and $p_j$ are not equivalent.

\[
\begin{array}{ccccccc}
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 \\
\end{array}
\]
1. Start by marking all cells \((q_i, q_j)\) where one of them is final and other is non-final.

2. Look for unmarked pairs \((q_i, q_j)\) such that for some \(c \in \Sigma\), the pair \((\delta(q_i, c), \delta(q_j, c))\) is marked. Then mark \((q_i, q_j)\).

3. Repeat step 2 until no such unmarked pairs remain.
First mark accepting/non-accepting pairs
\((q_1, q_3)\) is unmarked,

\[ q_1 \xrightarrow{b} q_0, \]

\[ q_3 \xrightarrow{b} q_1, \]

and \((q_0, q_1)\) is marked,

so mark \((q_1, q_3)\)
$(q_1, q_3)$ is unmarked, $q_1 \xrightarrow{b} q_0$, $q_3 \xrightarrow{b} q_1$, and $(q_0, q_1)$ is marked, so mark $(q_1, q_3)$
Illustration of minimization algorithm

$(q_2, q_3)$ is unmarked,
$q_2 \xrightarrow{b} q_0$,
$q_3 \xrightarrow{b} q_1$,
and $(q_0, q_1)$ is marked,
so mark $(q_2, q_3)$
(q_2, q_3) is unmarked,
q_2 \xrightarrow{b} q_0,
q_3 \xrightarrow{b} q_1,
and (q_0, q_1) is marked,
so mark (q_2, q_3)
Illustration of minimization algorithm

There is no way to mark the only unmarked pair \((q_1, q_2)\)
Obtain minimized DFA by collapsing \(q_1, q_2\) to a single state
Illustration of minimization algorithm
Convert the following DFA to a DFA with 3 states