CSCI-GA.2130-001
Compiler Construction
Lecture 5:
Lexical Analysis II

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
The Magic Behind It All: Finite Automata

• Recognizers: “yes” or “no” about each input string

• Two Flavors:
  – Non-deterministic Finite Automata (NFA)
  – Deterministic Finite Automata (DFA)

• Main parts
  – States
    • Start
    • Accepting or final
  – transitions
Which is Which?
NFA

- Finite set of states \( S \)
- Input alphabet \( \Sigma \)
- Transition function that gives for each state and for each \( \Sigma \cup \{\varepsilon\} \) a set of next states
- A starting state \( S_o \)
- A set of accepting or final state(s)
Another Presentation of NFA: Transition Tables

Transition Table:

<table>
<thead>
<tr>
<th>STATE</th>
<th>$a$</th>
<th>$b$</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>${0,1}$</td>
<td>${0}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>1</td>
<td>$\emptyset$</td>
<td>${2}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>$\emptyset$</td>
<td>${3}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

+ We can easily find the transition
- Lot of space
Acceptance of Input String

Input string $x$ is accepted if and only if:
There is some path in the transition graph from start to one of the accepting states.

Which of the following are accepted: $abb$, $aaa$, $aabb$, $aaabb$, $bbb$?
Example

• For the following NFA indicate all paths labeled $aabb$
DFA

• Special case of NFA
• No moves on $\varepsilon$
• For each state $S$, and input symbol $a$, there is exactly one edge out of $s$ labeled $a$
"Yes" or "No"?
abba
babb
aababb
abbb
# Some Definitions

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$-closure($s$)</td>
<td>Set of NFA states reachable from NFA state $s$ on $\epsilon$-transitions alone.</td>
</tr>
<tr>
<td>$\epsilon$-closure($T$)</td>
<td>Set of NFA states reachable from some NFA state $s$ in set $T$ on $\epsilon$-transitions alone; $= \bigcup_s \in T \epsilon$-closure($s$).</td>
</tr>
<tr>
<td>move($T$, $a$)</td>
<td>Set of NFA states to which there is a transition on input symbol $a$ from some state $s$ in $T$.</td>
</tr>
</tbody>
</table>

![Diagram](image.png)
Simulating NFA

1) $S = \epsilon$-closure($s_0$);
2) $c = \text{nextChar}()$;
3) while ( $c \neq \text{eof}$ ) {
   4) $S = \epsilon$-closure(move($S$, $c$));
   5) $c = \text{nextChar}()$;
4) }
7) if ( $S \cap F \neq \emptyset$ ) return "yes";
8) else return "no";}
Example

Simulate the following NFA on $aabb$

What is the transition table of the above NFA?

```plaintext
1) $S = \epsilon$-closure($s_0$);
2) $c = nextChar();$
3) while ( $c != eof$ ) {
   4) $S = \epsilon$-closure(move($S, c$));
   5) $c = nextChar();$
4) }
6) if ( $S \cap F \neq \emptyset$ ) return "yes";
7) else return "no";
```
NFA -> DFA

• Subset construction: each state of DFA corresponds to a set of NFA states
• For real languages NFA and DFA have approximately the same number of states (although theory has another opinion!)
Subset Constructions

Initially, $\epsilon$-closure($s_0$) is the only state in $D\text{states}$, and it is unmarked;

while (there is an unmarked state $T$ in $D\text{states}$) {
    mark $T$;
    for (each input symbol $a$) {
        $U = \epsilon$-closure(move($T$, $a$));
        if ($U$ is not in $D\text{states}$)
            add $U$ as an unmarked state to $D\text{states}$;
        $D\text{tran}[T,a] = U$;
    }
}

States of the DFA we are constructing
Regular Expression -> NFA
(McNaughton-Yamada-Thompson algorithm)

\[ r = a \]

\[ r = s|t \]

\[ r = st \]

\[ r = s^* \]
Example: \((a \mid b)^* abb\)
Example: \((a|b)^*abb\)
Example: \((a | b)^* abb\)
State Minimization of DFA

- There can be many DFAs that recognize the same language.
- Smaller DFAs are more efficient (storage, speed)
- There is always a unique minimum state DFA
- This minimum-state DFA can be constructed from any DFA that recognizes the language.
How to Do It?

1. Given DFA: start with at least two subgroups: S and S-F
2. Repeat the following algorithm until no more progress can be made

initially, let $\Pi_{\text{new}} = \Pi$;
for ( each group $G$ of $\Pi$ ) {
    partition $G$ into subgroups such that two states $s$ and $t$
    are in the same subgroup if and only if for all
    input symbols $a$, states $s$ and $t$ have transitions on $a$
    to states in the same group of $\Pi$;
    /* at worst, a state will be in a subgroup by itself */
    replace $G$ in $\Pi_{\text{new}}$ by the set of all subgroups formed;
}

Example

\{A,B,C,D\} \{E\}

\{A,B,C\} \{D\} \{E\}

\{A,C\} \{B\} \{D\} \{E\}

<table>
<thead>
<tr>
<th>STATE</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>
Lexical Analyzer Generators

Input buffer

lexeme

lexemeBegin

forward

Automaton simulator

Lex program

Lex compiler

Transition table

Actions
Lexical Analyzer Generators

- Each regular expression $\rightarrow$ NFA
- Combine all NFAs as
- In case of several matches
  - Pick longest
  - Pick earliest in file
Example: aaba
Lex

• Based on DFA not NFA
• Handling lookahead
• For state minimization, initial partition:
  – groups all states that recognizes a particular token
  – places in one group those states that do not indicate any token
So

• We have covered Sections 3.6 -> 3.9
• Skim: 3.7.3, 3.7.5, 3.9.1->3.9.5 and 3.9.8
• Read carefully the rest of: 3.6, 3.7, 3.8, 3.9.6, and 3.9.7