Outline

- Recap
  - The lexical-analyzer generator Lex
- Implementing lexical-analyzer generators
- Summary and homework
Implementing Lexical-Analyzer Generators

- Regular expressions $\rightarrow$ Nondeterministic finite automata
- Nondeterministic finite automata $\rightarrow$ Deterministic finite automata
- Deterministic finite automata $\rightarrow$ A lexer

- Regular expressions $\rightarrow$ Deterministic finite automata
- Deterministic finite automata $\rightarrow$ A lexer
MYT Algorithm

- Constructing an NFA from a regular expression $r$ by McNaughton-Yamada-Thompson algorithm
  - Organizing $r$ into its constituent sub-expressions (parse tree)
    - Sub-expressions with no operators
    - Operators
  - Using basic rules to construct NFA for sub-expressions with no operators
  - Using inductive rules to construct larger NFA based on the constructed NFA for operations of sub-expressions
An Example: $(a|b)^*abb$
Another Example

- Form the NFA for the regular expression

  \textit{letter}(\textit{letter}|\textit{digit})^{*}
Implementing Lexical-Analyzer Generators

- Regular expressions $\rightarrow$ Nondeterministic finite automata
- Nondeterministic finite automata $\rightarrow$ Deterministic finite automata
- Deterministic finite automata $\rightarrow$ A lexer
Conversion of NFA to DFA

- Subset construction algorithm
  - Input: An NFA \( \mathcal{N} \)
  - Output: A DFA \( \mathcal{D} \) accepting the same language as \( \mathcal{N} \)
  - Algorithm: construct a transition table \( \mathcal{D}_{\text{tran}} \) corresponding to \( \mathcal{D} \)

Initially, \( \varepsilon\)-closure(\( s_0 \)) is the only state in \( \mathcal{D}_{\text{states}} \), and it is unmarked; while ( there is an unmarked state \( T \) in \( \mathcal{D}_{\text{states}} \) ) {
  mark \( T \);
  for ( each input symbol \( a \) ) {
    \( U = \varepsilon\text{-closure}(\text{move}(T, a)) \);
    if ( \( U \) is not in \( \mathcal{D}_{\text{states}} \) ) add \( U \) as an unmarked state to \( \mathcal{D}_{\text{states}} \);
    \( \mathcal{D}_{\text{tran}}[T, a] = U \);
  }
}
\( \varepsilon \)-closure\((s) \) and \( \varepsilon \)-closure\((T) \)

- \( \varepsilon \)-closure\((s) \): a set of NFA states reachable from NFA state \( s \) on \( \varepsilon \)-transitions alone
- \( \varepsilon \)-closure\((T) \): a set of NFA states reachable from some NFA state \( s \) in the set \( T \) on \( \varepsilon \)-transitions alone
  - \( \bigcup_{s \in T} \varepsilon \)-closure\((s) \)

push all states of \( T \) onto stack;
initialize \( \varepsilon \)-closure\((T) \) to \( T \);
while ( stack is not empty ) {
  pop \( t \), the top element, off stack;
  for ( each state \( u \) with an edge from \( t \) to \( u \) labeled \( \varepsilon \) )
    if ( \( u \) is not in \( \varepsilon \)-closure\((T) \) ) {
      add \( u \) to \( \varepsilon \)-closure\((T) \);  push \( u \) onto stack;
    }
}
move(T, a)

- A set of NFA states to which there is a transition on input symbol a from some state s in T
Conversion of An NFA Accepting \((a|b)^*abb\) to A DFA

- Draw the state transition diagram
Another Example

- Convert the NFA for the regular expression \textit{letter(letter|digit)}^* to a DFA
Simulation of An NFA

An input string $x$ terminated by $\texttt{eof}$. An NFA $N$ with a start state $s_0$, accepting states $F$, and $\varepsilon$-closure() and move() functions.

\[
S = \varepsilon\text{-closure}(s_0);
\]
\[
c = \text{nextChar}();
\]
\[
\textbf{while} \ ( \ c \neq \texttt{eof} ) \ \{ \quad S = \varepsilon\text{-closure}(\text{move}(S, c)); \ c = \text{nextChar}();
\}
\]
\[
\textbf{if} \ ( \ S \cap F \neq \emptyset ) \ \textbf{return} \ \text{“yes”};
\]
\[
\textbf{else} \ \text{return} \ \text{“no”};
\]
Outline

- Recap
- Implementing lexical-analyzer generators
- Summary and homework
Flex

- Fast lexical analyzer generator
Conversion of NFA to DFA

- Subset construction algorithm
  - Input: An NFA $N$
  - Output: A DFA $D$ accepting the same language as $N$
  - Algorithm: construct a transition table $D_{tran}$ corresponding to $D$

Initially, $\epsilon$-closure($s_0$) is the only state in $D_{states}$, and it is unmarked;
while ( there is an unmarked state $T$ in $D_{states}$ ) {
    mark $T$;
    for ( each input symbol $a$ ) {
        $U = \epsilon$-closure(move($T, a$));
        if ( $U$ is not in $D_{states}$ ) add $U$ as an unmarked state to $D_{states}$;
        $D_{tran}[T, a] = U$;
    }
}
Reading Assignment

- For today’s class
  - Sections 3.7 and 3.8
- For next Tuesday’s class
  - Chapter 4
Homework (Due on 02/19 at 11:55 PM)

5.1. (10 points). Using flex and based on the Example 3.8 (pages 128-129 in the textbook), generate a lexer that scans the following input stream and outputs the following output stream.

- Input stream: if i>0 then i=1 else i=0

Please provide a readme file explaining how you generate and test your lexer.

5.2. (10 points) Convert the NFA for the regular expression \texttt{letter}\texttt{(letter|digit)}* to a DFA.