Chapter 4b

Lexical analysis

Finite Automata

Finite Automata (FA)

- FA also called Finite State Machine (FSM)
  - Abstract model of a computing entity.
  - Decides whether to accept or reject a string.
  - Every regular expression can be represented as a FA and vice versa.

- Two types of FAs:
  - Non-deterministic (NFA): Has more than one alternative action for the same input symbol.
  - Deterministic (DFA): Has at most one action for a given input symbol.

- Example: how do we write a program to recognize java keyword "int"?

RE and Finite State Automaton (FA)

- Regular expression is a declarative way to describe the tokens
  - It describes what is a token, but not how to recognize the token.
- FA is used to describe how the token is recognized
  - FA is easy to be simulated by computer programs;
- There is a 1-1 correspondence between FA and regular expression
  - Scanner generator (such as lex) bridges the gap between regular expression and FA.

Transition Diagram

- FA can be represented using transition diagram.
- Corresponding to FA definition, a transition diagram has:
  - States represented by circles;
  - An Alphabet (Σ) represented by labels on edges;
  - Transitions represented by labeled directed edges between states. The label is the input symbol;
  - One Start State shown as having an arrow head;
  - One or more Final State(s) represented by double circles.

- Example transition diagram to recognize (a|b)*abb
Simple examples of FA

- \( a \)
- \( a^* \)
- \( a^+ \)
- \( (a|b)^* \)

Procedures of defining a DFA/NFA

- Defining input alphabet and initial state
- Draw the transition diagram
- Check
  - Do all states have out-going arcs labeled with all the input symbols (DFA)
  - Any missing final states?
  - Any duplicate states?
  - Can all strings in the language can be accepted?
  - Are any strings not in the language accepted?
- Naming all the states
- Defining \( (S, \Sigma, \delta, q_0, F) \)

Example of constructing a FA

- Construct a DFA that accepts a language \( L \) over the alphabet \( \{0, 1\} \) such that \( L \) is the set of all strings with \textit{any} number of “0”s followed by \textit{any} number of “1”s.
- Regular expression: \( 0^*1^* \)
- \( \Sigma = \{0, 1\} \)
- Draw initial state of the transition diagram

Example of constructing a FA

- Draft the transition diagram
- Is “111” accepted?
- The leftmost state has missed an arc with input “1”
Example of constructing a FA

• Is “00” accepted?
• The leftmost two states are also final states
  – First state from the left: \( \varepsilon \) is also accepted
  – Second state from the left: strings with “0”s only are also accepted

Start \[ \xrightarrow{0} \] 0 \[ \xrightarrow{1} \] 1

Example of constructing a FA

• The leftmost two states are duplicate
  – their arcs point to the same states with the same symbols

Start \[ \xrightarrow{} \] 0 \[ \xrightarrow{} \] 1

• Check that they are correct
  – All strings in the language can be accepted
    » \( \varepsilon \), the empty string, is accepted
    » strings with “0”s / “1”s only are accepted
  – No strings not in language are accepted

• Naming all the states

Start \[ \xrightarrow{} \] 0 \[ \xrightarrow{} \] 1

How does a FA work

• NFA definition for \((a|b)*abb\)
  – \( S = \{ q_0, q_1, q_2, q_3 \} \)
  – \( \Sigma = \{ a, b \} \)
  – Transitions: \( \text{move}(q_0, a) = \{ q_0, q_1 \}, \text{move}(q_0, b) = \{ q_0 \}, \ldots \)
  – \( s_0 = q_0 \)
  – \( F = \{ q_3 \} \)

• Transition diagram representation
  – Non-determinism:
    » exiting from one state there are multiple edges labeled with same symbol, or
    » There are epsilon edges.
  – How does FA work? Input: ababb

\[
\begin{align*}
\text{move}(0, a) &= 1 \\
\text{move}(1, b) &= 2 \\
\text{move}(2, a) &= 7 \text{ (undefined)} \\
\text{move}(0, a) &= 0 \\
\text{move}(0, b) &= 0 \\
\text{move}(0, a) &= 1 \\
\text{move}(1, b) &= 2 \\
\text{move}(2, b) &= 3 \\
\text{ACCEPT} \\
\text{REJECT}!
\end{align*}
\]

FA for \((a|b)*abb\)

– What does it mean that a string is accepted by a FA?
An FA accepts an input string \( x \) if there is a path from the start state to a final state, such that the edge labels along this path spell out \( x \);
– A path for “aabbb”: \( Q_0 \rightarrow a \rightarrow Q_0 \rightarrow a \rightarrow q_1 \rightarrow b \rightarrow q_2 \rightarrow b \rightarrow q_3 \)
– Is “aab” acceptable?
  \( Q_0 \rightarrow a \rightarrow Q_0 \rightarrow a \rightarrow Q_1 \rightarrow b \rightarrow Q_2 \rightarrow b \rightarrow Q_3 \)

» Final state must be reached;
» In general, there could be several paths.
– Is “aabbb” acceptable?
  \( Q_0 \rightarrow a \rightarrow q_0 \rightarrow a \rightarrow q_1 \rightarrow b \rightarrow q_2 \rightarrow b \rightarrow q_3 \)
» Labels on the path must spell out the entire string.
**Transition table**

- A transition table is a good way to implement a FSA
  - One row for each state, \( S \)
  - One column for each symbol, \( A \)
  - Entry in cell \((S, A)\) gives the state or set of states can be reached from state \( S \) on input \( A \).

- A Nondeterministic Finite Automaton (NFA) has at least one cell with more than one state.

- A Deterministic Finite Automaton (DFA) has a single state in every cell

**DFA (Deterministic Finite Automaton)**

- A special case of NFA where the transition function maps the pair \((\text{state}, \text{symbol})\) to one state.
  - When represented by transition diagram, for each state \( S \) and symbol \( a \), there is at most one edge labeled \( a \) leaving \( S \).
  - When represented transition table, each entry in the table is a single state.
    - There are no \( \varepsilon \)-transition

- Example: DFA for \((a|b)^*abb\)

**DFA to program**

- NFA is more concise, but not as easy to implement;
- In DFA, since transition tables don’t have any alternative options, DFAs are easily simulated via an algorithm.
- Every NFA can be converted to an equivalent DFA
  - What does equivalent mean?
- There are general algorithms that can take a DFA and produce a “minimal DFA.
  - Minimal in what sense?
- There are programs that take a regular expression and produce a program based on a minimal DFA to recognize strings defined by the RE.
- You can find out more in 451 (automata theory) and/or 431 (Compiler design)