CS375: Logic and Theory of Computing

Fuhua (Frank) Cheng

Department of Computer Science
University of Kentucky
Table of Contents:

- **Week 1: Preliminaries** (set algebra, relations, functions) (read Chapters 1-4)
- **Weeks 3-6: Regular Languages, Finite Automata** (Chapter 11)
- **Weeks 7-9: Context-Free Languages, Pushdown Automata** (Chapters 12)
- **Weeks 10-12: Turing Machines** (Chapter 13)
Table of Contents (conti):

- Weeks 13-14: Propositional Logic (Chapter 6), Predicate Logic (Chapter 7), Computational Logic (Chapter 9), Algebraic Structures (Chapter 10)
6. Regular Languages & Finite Automata

- Constructing Efficient Finite Automata

**Question:** can we get a *most efficient regular expression* for a given NFA?

*as compact as possible*

**YES,** in two steps.

First, transform the given NFA to a DFA

Then, transform the DFA to a *minimum-state DFA*
Transforming an NFA into a DFA

The \( \Lambda \)-closure of a state \( s \), denoted \( \Lambda(s) \), is the set consisting of \( s \) together with all states that can be reached from \( s \) by traversing \( \Lambda \)-edges.

The \( \Lambda \)-closure of a set \( S \) of states, denoted \( \Lambda(S) \), is the union of the \( \Lambda \)-closure of the states in \( S \).

\[
\Lambda(0) = \{0, 1, 2\}
\]

\[
\Lambda(\{0, 1\}) = \Lambda(0) \cup \Lambda(1) = \{0, 1, 2\} \cup \{1, 2\} = \{0, 1, 2\}
\]
6. Regular Languages & Finite Automata

- Constructing Efficient Finite Automata

The same NFA again:

\[ \Lambda(0) = \{0, 1, 2\} \]
\[ \Lambda(1) = \{1, 2\} \]
\[ \Lambda(2) = \{2\} \]
\[ \Lambda(\emptyset) = \emptyset \]

\[ \Lambda(\{1, 2\}) = \{1, 2\} \]
\[ \Lambda(\{0, 1, 2\}) = \{0, 1, 2\} \]
Algorithm: **Transform an NFA into a DFA**

Construct a DFA table $T_D$ from an NFA table $T_N$ as follows:

1. The start state of the DFA is $\Lambda(s)$, where $s$ is the start state of the NFA.

2. If $\{s_1, \ldots, s_n\}$ is a DFA state and $a \in A$, then

   $$T_D(\{s_1, \ldots, s_n\}, a) = \Lambda(T_N(s_1, a) \cup \ldots \cup T_N(s_n, a)).$$

3. A DFA state is final if one of its elements is an NFA final state.
6. Regular Languages & Finite Automata

- Constructing Efficient Finite Automata

**Example.** Given the following NFA.

![NFA Diagram](image)

Construct the DFA transition table $T_D$ and write it in simplified form after renumbering states.

**First:**

- $\Lambda(0) = \{0, 3\}$
- $\Lambda(1) = \{1\}$
- $\Lambda(2) = \{2\}$
- $\Lambda(3) = \{3\}$
6. Regular Languages & Finite Automata

- Constructing Efficient Finite Automata

Then, how to get the States of the DFA?

1. Build the tree on the right
2. Identify all distinct nodes
Example. Given the following NFA.

Construct the DFA transition table $T_D$ and write it in simplified form after renumbering states.

$$\begin{align*}
\Lambda(0) &= \{0,3\} \\
\Lambda(1) &= \{1\} \\
\Lambda(2) &= \{2\} \\
\Lambda(3) &= \{3\}
\end{align*}$$

E.g.,

$$
T_D(\{0,3\}, a) = \Lambda(T_N(0, a) \cup T_N(3, a))
$$

$$
= \Lambda(\emptyset \cup \{3\})
$$

$$
= \Lambda(\{3\})
$$

$$
= \{3\}
$$
Example. Given the following NFA.

Construct the DFA transition table $T_D$ and write it in simplified form after renumbering states.

$$
\begin{align*}
\Lambda(0) &= \{0,3\} \\
\Lambda(1) &= \{1\} \\
\Lambda(2) &= \{2\} \\
\Lambda(3) &= \{3\}
\end{align*}
$$

$$
T_D(\{0,3\}, b) = \Lambda(T_N(0, b) \cup T_N(3, b))
$$

\begin{align*}
&= \Lambda(\{0,1\} \cup \emptyset) \\
&= \Lambda(\{0,1\}) \\
&= \Lambda(0) \cup \Lambda(1) \\
&= \{0,3\} \cup \{1\} = \{0,1,3\}
\end{align*}
6. Regular Languages & Finite Automata

- Constructing Efficient Finite Automata

**Example.** Given the following NFA.

Construct the DFA transition table $T_D$ and write it in simplified form after renumbering states.

$$\Lambda(0) = \{0, 3\}$$
$$\Lambda(1) = \{1\}$$
$$\Lambda(2) = \{2\}$$
$$\Lambda(3) = \{3\}$$

**e.g.,**

$$T_D(\{3\}, a) = \Lambda(T_N(3, a))$$
$$= \Lambda(\{3\})$$
$$= \Lambda(3)$$
$$= \{3\}$$
Example. Given the following NFA.

Construct the DFA transition table $T_D$ and write it in simplified form after renumbering states.

$\Lambda(0) = \{0,3\}$
$\Lambda(1) = \{1\}$
$\Lambda(2) = \{2\}$
$\Lambda(3) = \{3\}$

e.g., $T_D(\{3\}, b) = \Lambda(T_N(3, b)) = \Lambda(\emptyset) = \emptyset$
6. Regular Languages & Finite Automata

- Constructing Efficient Finite Automata

**Example.** Given the following NFA.

Construct the DFA transition table $T_D$ and write it in simplified form after renumbering states.

\[
\begin{align*}
\Lambda(0) &= \{0,3\} \\
\Lambda(1) &= \{1\} \\
\Lambda(2) &= \{2\} \\
\Lambda(3) &= \{3\}
\end{align*}
\]

\[
T_D(\{0,1,3\}, a) = \Lambda(T_N(0, a) \cup T_N(1, a) \cup T_N(3, a))
\]

\[
= \Lambda(\emptyset \cup \{2\} \cup \{3\})
\]

\[
= \Lambda(\{2,3\})
\]

\[
= \Lambda(2) \cup \Lambda(3) = \{2\} \cup \{3\} = \{2,3\}
\]
Example. Given the following NFA.

Construct the DFA transition table $T_D$ and write it in simplified form after renumbering states.

Hence
Example. Transform the following NFA into a DFA.
Example. Transform the following NFA into a DFA.

\[ \begin{align*}
\Lambda(0) &= \{0, 3\} \\
\Lambda(1) &= \{1\} \\
\Lambda(2) &= \{2\} \\
\Lambda(3) &= \{3\}
\end{align*} \]
6. Regular Languages & Finite Automata

- Constructing Efficient Finite Automata

**Quiz.** Transform the following NFA into a DFA.

Then, build the tree

Distinct nodes are: \{0, 3\}, \{1,2,3\}, \{2,3\}, \{2\}, \emptyset
Example. Transform the following NFA into a DFA.

Hence, solution:

\[
\begin{array}{c|ccc}
T_D & a & b \\
\hline
S & \{0, 3\} & \{1, 2, 3\} & \emptyset \\
F & \{1, 2, 3\} & \{2, 3\} & \{2\} \\
F & \{2, 3\} & \{2, 3\} & \emptyset \\
F & \{2\} & \emptyset & \emptyset \\
\end{array}
\]
Algorithm: Transform a DFA to a minimum-state DFA

1. Construct the following sequence of sets of possible equivalent pairs of distinct states:

\[ E_0 \supset E_1 \supset \cdots \supset E_k = E_{k+1} \]

where

\[ E_0 = \left\{ \{s, t\} \mid \text{s and t are either both final or both non-final} \right\} \]

and

\[ E_{i+1} = \left\{ \{s, t\} \in E_i \mid \{T(s, a), T(t, a)\} \in E_i \text{ or } T(s, a) = T(t, a) \right\} \]

for every \( a \in A \}

\( E_k \) represents the distinct pairs of equivalent states from which an equivalence relation \( \sim \) can be generated.
6. Regular Languages & Finite Automata

- Constructing Efficient Finite Automata

\[ E_0 = \{ \{s, t\} / s & t \text{ are either both final or both non-final}\} \]

\[ E_0 = \{ \{0,4\}, \{1,2\}, \{1,3\}, \{2,3\} \} \]

\[ E_0 = \{ \{0,0\}, \{4,4\}, \{0,4\}, \{4,0\}, \{1,1\}, \{2,2\}, \{3,3\}, \{1,2\}, \{2,1\}, \{1,3\}, \{3,1\}, \{2,3\}, \{3,2\} \} \]
To be a pair in $E_{i+1}$, $s$ and $t$ must be mapped to the same state or states in the same group in $E_i$ by every $a \in A$. 

$E_{i+1} = \{ \{s, t\} \in E_i \mid \{T(s, a), T(t, a)\} \in E_i \text{ or } T(s, a) = T(t, a) \text{ for every } a \in A \}$
6. Regular Languages & Finite Automata

- Constructing Efficient Finite Automata

E₁ = ?

E₁ =
Theoretically,
$E_1 = \{ \{0,0\}, \{4,4\}, \{1,1\}, \{2,2\}, \{3,3\}, \{1,2\}, \{2,1\}, \{1,3\}, \{3,1\}, \{2,3\}, \{3,2\} \}$

or simply, $E_1 = \{ \{1,2\}, \{1,3\}, \{2,3\} \}$
6. Regular Languages & Finite Automata

- Constructing Efficient Finite Automata

One-element groups cannot be further reduced and the three-element group will remain the same.

Hence, \( E_2 = E_1 \)

\( S \) is partitioned by \( \{0\}, \{1, 2, 3\}, \{4\} \).
Algorithm: Transform a DFA to a minimum-state DFA

2. The equivalence classes form the states of the minimum state DFA with transition table $T_{\text{min}}$ defined by

$$T_{\text{min}}([s], a) = [T(s, a)].$$

3. The start state is the class containing the start state of the given DFA.

4. A final state is any class containing a final state of the given DFA.
Example. Transform the given DFA into a minimum-state DFA.

\[ E_2 = E_1 \]

So \( S \) is partitioned by \( \{0\}, \{1, 2, 3\}, \{4\} \).
The minimum-state DFA has three states: [0], [1], [4].
Example. Transform the given DFA into a minimum-state DFA.

\[
\begin{align*}
T_{\text{min}}([s], a) &= [T(s, a)] \\
T_{\text{min}}([0], a) &= [T(0, a)] = [1] \\
T_{\text{min}}([0], b) &= [T(0, b)] = [4] \\
T_{\text{min}}([1], a) &= [T(1, a)] = [2] = [1]
\end{align*}
\]
Example. Transform the given DFA into a minimum-state DFA.

### Min-state Table

<table>
<thead>
<tr>
<th></th>
<th>$T_{Min}$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>$F$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

### Renamed Table

<table>
<thead>
<tr>
<th></th>
<th>$T_{Min}$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$F$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

### Min-state DFA graph

- $S$ → 0 (on $a$, $b$)
- 0 → 1 (on $a$, $b$)
- 1 → 2 (on $b$)
- 2 → 0 (on $a$, $b$)
Example. Transform the given DFA into a minimum-state DFA.

Question: What regular expression equality arises from the two DFAs?

Answer: \( a + aa + (aaa + aab + ab)(a + b)^* = a(a + b)^*. \)
Prove: \[ a + aa + (aaa + aab + ab)(a + b)^* = a(a + b)^* \]

LHS = \[ a + aa + aa(a+b)(a+b)^* + ab(a+b)^* \]

= \[ a + aa + aa(a + b)^+ + ab(a+b)^* \]

= \[ a + aa(\Lambda + (a + b)^+ ) + ab(a+b)^* \]

= \[ a + aa(a+b)^* + ab(a+b)^* \]

= \[ a + a(a + b)^+ \]

= \[ a(\Lambda + (a + b)^+) \]

= \[ a (a + b)^* \]

= RHS
Question: Is the following DFA a minimum-state DFA?

Answer. No.
Use the minimum-state algorithm.

\[ E_0 = \{ \{0, 1\} \}, \quad E_1 = \{ \{0, 1\} \} = E_0. \]

The partition is the whole set of states \( \{0, 1\} = [0] \).
Therefore, we have
Question: Is the following DFA a minimum-state DFA?
Question: Is the following DFA a minimum-state DFA?

E₀ = ?

E₁ = ?
Question: Is the following DFA a minimum-state DFA?

\[ E_0 = \]

\[ E_1 = ? \]
Question: Is the following DFA a minimum-state DFA?
Question: Is the following DFA a minimum-state DFA?

Answer: NO
End of Regular Language and Finite Automata III