Finite State Automata and Determinisation

Tim Dawborn

January, 2017
Outline

1. Languages
2. Finite State Automata (FSA)
3. Non-deterministic Finite State Automata (NFA)
4. Regular Expressions
5. Deterministic Finite State Automata (DFA)
6. Determinisation
Alphabets

- An alphabet $\Sigma$ is a set of tokens used by a language
  - $\Sigma = \{0, 1\}$ is the alphabet for binary
  - $\Sigma = \{., 0, 1, \ldots, 9\}$ is the alphabet for a decimal number
  - $\Sigma = \{\text{marco, polo}\}$ is the alphabet for the communications protocol in the game Marco Polo\(^1\)

- What language(s) is this the alphabet for?
  $\Sigma = \{x, 0, 1, \ldots, 9, A, B, \ldots, F\}$

\(^1\) A game involving calling out “Marco!” and “Polo!”, with an optional swimming pool
Strings

- A combination\(^2\) of elements from \(\Sigma\) is called a *string*
  - “0”, “101101”, “111” are all strings of \(\Sigma = \{0, 1\}\)
- The set \(\Sigma^*\) is the infinite set of all combinations of the elements of \(\Sigma\)
  - For the binary alphabet, \(\Sigma = \{0, 1\}\), \(\Sigma^*\) is\(^3\)
    \[\Sigma^* = \{\emptyset, (0), (1), (00), (01), (10), (11), (000), (001), \ldots\}\]

---

\(^2\)Obviously, a contiguous ordered tuple of elements, not a set: “01” is a different string from “10”

\(^3\)Here the empty set will do to represent a null or empty string
Languages

- A language $L$ is a set of items from $\Sigma^*$ which are deemed to be valid ($L \subseteq \Sigma^*$)
  - For hexadecimal, only the strings which begin with “0x” and have at least one more non-“x” token are deemed valid
    - ✓ “0xF9”
    - ✗ “1234”
    - ✓ “0x0”
    - ✗ “0x”
    - ✗ “1xA”
    - ✗ “” (the empty string)
- A language in this context is not a set of rules, syntax, grammar and all: it’s just a set of valid strings
Finite State Automata

- Finite State Automata are representations of a self-contained finite set of states, with rules that govern movement among those states.
- Finite State Automata (FSAs) exist everywhere
- They are used to illustrate
  - the possible states some process can be in
  - how the process moves between these states
Finite State Automata

- E.g. a state machine for PIN code entry on your mobile⁴:

![Diagram of a state machine for PIN code entry]

---

⁴How many people do you know who say “PIN Number”?
Finite State Automata

- FSAs have nodes
Finite State Automata

- **FSAs have nodes**
- **FSAs have edges**
Finite State Automata

- FSAs have nodes
- FSAs have edges
- FSA edges (transitions) have labels
Finite State Automata

- FSAs have nodes
- FSAs have edges
- FSA edges (transitions) have labels
- FSAs have a start node (entry point)
Finite State Automata

- FSAs have nodes
- FSAs have edges
- FSA edges (transitions) have labels
- FSAs have a start node (entry point)
- FSAs have accepting nodes, denoted with the double outline
Finite State Automata

- You are always “on” one of the nodes, i.e., in one of the states.
- You can travel between the nodes following the directed edges.
- You can only travel along an edge if you see the label that is on the edge.
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“0xF9”</td>
<td>q0 → q1 → x → q2 → q3</td>
<td></td>
</tr>
</tbody>
</table>
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;0xF9&quot;</td>
<td>q₀ → q₁ → q₂ → q₃ → q₃</td>
<td></td>
</tr>
</tbody>
</table>
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“0xF9”</td>
<td>$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_3$</td>
<td>✓</td>
</tr>
</tbody>
</table>
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;0xF9&quot;</td>
<td>q₀ → q₁ → q₂ → q₃ → q₃</td>
<td>✓</td>
</tr>
<tr>
<td>&quot;1234&quot;</td>
<td>q₀ → q₁ → q₂ → q₃ → q₃</td>
<td></td>
</tr>
</tbody>
</table>
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;0xF9&quot;</td>
<td>q₀ → q₁ → q₂ → q₃ → q₃</td>
<td>✓</td>
</tr>
<tr>
<td>&quot;1234&quot;</td>
<td>q₀ → q₀</td>
<td></td>
</tr>
</tbody>
</table>
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“0xF9”</td>
<td>$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_3$</td>
<td>✓</td>
</tr>
<tr>
<td>“1234”</td>
<td>$q_0$</td>
<td>✗</td>
</tr>
</tbody>
</table>
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“0xF9”</td>
<td>q₀ → q₁ → q₂ → q₃ → q₃</td>
<td>✓</td>
</tr>
<tr>
<td>“1234”</td>
<td>q₀</td>
<td>x</td>
</tr>
<tr>
<td>“0x0”</td>
<td>q₀</td>
<td></td>
</tr>
</tbody>
</table>
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“0xF9”</td>
<td>$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_3$</td>
<td>✔</td>
</tr>
<tr>
<td>“1234”</td>
<td>$q_0$</td>
<td>❌</td>
</tr>
<tr>
<td>“0x0”</td>
<td>$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$</td>
<td>✔</td>
</tr>
</tbody>
</table>
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;0xF9&quot;</td>
<td>$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_3$</td>
<td>✔</td>
</tr>
<tr>
<td>&quot;1234&quot;</td>
<td>$q_0$</td>
<td>✗</td>
</tr>
<tr>
<td>&quot;0x0&quot;</td>
<td>$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$</td>
<td>✔</td>
</tr>
</tbody>
</table>
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“0xF9”</td>
<td>$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_3$</td>
<td>✓</td>
</tr>
<tr>
<td>“1234”</td>
<td>$q_0$</td>
<td>x</td>
</tr>
<tr>
<td>“0x0”</td>
<td>$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$</td>
<td>✓</td>
</tr>
<tr>
<td>“”</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“0xF9”</td>
<td>$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_3$</td>
<td>✓</td>
</tr>
<tr>
<td>“1234”</td>
<td>$q_0$</td>
<td>✗</td>
</tr>
<tr>
<td>“0x0”</td>
<td>$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$</td>
<td>✓</td>
</tr>
<tr>
<td>“”</td>
<td>$q_0$</td>
<td>✓</td>
</tr>
</tbody>
</table>
Pattern Recognition using FSAs

Walk the following strings through the FSA. Are they accepted?

<table>
<thead>
<tr>
<th>String</th>
<th>Path</th>
<th>Accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“0xF9”</td>
<td>q₀ → q₁ → q₂ → q₃ → q₃</td>
<td>✓</td>
</tr>
<tr>
<td>“1234”</td>
<td>q₀ → q₁ → q₂ → q₃</td>
<td>✗</td>
</tr>
<tr>
<td>“0x0”</td>
<td>q₀ → q₁ → q₂ → q₃</td>
<td>✓</td>
</tr>
<tr>
<td>“”</td>
<td>q₀</td>
<td>✗</td>
</tr>
</tbody>
</table>
Non-deterministic Finite State Automata

- There are two main types of FSAs
  - Deterministic
  - Non-deterministic
- What we have been looking at so far are examples of non-deterministic finite [state] automata (NFA)
- E.g. NFA for recognising a hexadecimal number:

```
q0 ----> 0 ----> q1
    x         q2
        0-9,A-F  ----> q3
```

Tim Dawborn

Finite State Automata and Determinisation

January, 2017
Properties of NFAs

- Must have a start node
- No restrictions on the edge labels
- Allowed special $\varepsilon$ edge labels (we’ll come back to this)
- Allowed to be “on” multiple states simultaneously
  - non-determinism

Basically an NFA is a FSA that isn’t fully defined: in a DFA (Deterministic Finite (State) Automaton of course), everything has to be determined: if there’s an error condition that has to be explained, there can’t be any ambiguity, and so on.
Non-determinism example

Is the string “ca” accepted by the NFA?
Path:
Accepted:
Non-determinism example

Is the string “ca” accepted by the NFA?
Path: \{q_0\}
Accepted:
Non-determinism example

Is the string “ca” accepted by the NFA?
Path: \( \{ q_0 \} \to \{ q_1 \} \)
Accepted:
Non-determinism example

Is the string “ca” accepted by the NFA?
Path: \( \{q_0\} \rightarrow \{q_1\} \rightarrow \{q_3\} \)
Accepted:
Non-determinism example

Is the string “ca” accepted by the NFA?
Path: \(q_0 \rightarrow q_1 \rightarrow q_3\)
Accepted: Yes
Non-determinism example

Is the string “ac” accepted by the NFA?
Path:
Accepted:

```
      q2
     /  \
   a,b   \\
  /      \
q0 ---- a ---- q1
     |   |   |
     b   c   b
     |   |   |
     q3 ---- b,c ---- q3
```

Accepted: Yes
Non-determinism example

Is the string “ac” accepted by the NFA?
Path: \{q_0\}
Accepted:
Non-determinism example

Is the string “ac” accepted by the NFA?
Path: \( \{ q_0 \} \rightarrow \{ q_0, q_2 \} \)
Accepted:
Non-determinism example

Is the string “ac” accepted by the NFA?
Path: \( \{ q_0 \} \rightarrow \{ q_0, q_2 \} \rightarrow \{ q_1, q_3 \} \)
Accepted:
Non-determinism example

Is the string “ac” accepted by the NFA?
Path: \[\{q_0\} \rightarrow \{q_0, q_2\} \rightarrow \{q_1, q_3\}\]
Accepted: Yes
Non-determinism example

Is the string “aabc” accepted by the NFA?

Path:

Accepted:
Non-determinism example

Is the string “aabc” accepted by the NFA?
Path: \{q_0\}
Accepted:
Non-determinism example

Is the string “aabc” accepted by the NFA?
Path: \(\{q_0\} \rightarrow \{q_0, q_2\}\)
Accepted:
Non-determinism example

Is the string “aabc” accepted by the NFA?

Path: \( \{ q_0 \} \rightarrow \{ q_0, q_2 \} \rightarrow \{ q_0, q_2 \} \)

Accepted:
Non-determinism example

Is the string “aabc” accepted by the NFA?
Path: \( \{q_0\} \rightarrow \{q_0, q_2\} \rightarrow \{q_0, q_2\} \rightarrow \{q_0, q_1, q_2\} \)
Accepted:
Non-determinism example

Is the string “aabc” accepted by the NFA?
Path: \( \{q_0\} \to \{q_0, q_2\} \to \{q_0, q_2\} \to \{q_0, q_1, q_2\} \to \{q_1, q_3\} \)
Accepted:

```
q0 ——— a,b ——— q2
  |     \       |
  |        \     |
  |         \    |
  b ——— c ———|
  \         |
  \       |
  \     |
  \     |
  c ——— a ——— q1
  |     \       |
  \     \       |
  \     \       |
  \     \       |
  \     \       |
  \     \       |
  \     \       |
  \     \       |
  b,c ——— q3
```

Accepted: Yes
Non-determinism example

Is the string “aabc” accepted by the NFA?
Path: \[\{q_0\} \rightarrow \{q_0, q_2\} \rightarrow \{q_0, q_2\} \rightarrow \{q_0, q_1, q_2\} \rightarrow \{q_1, q_3\}\]
Accepted: Yes
\( \varepsilon \) transitions

- NFAs can have a special transition label \( \varepsilon \)
- \( \varepsilon \) allows a transition *without* consuming any input tokens

In English, what pattern does this NFA therefore allow us to match?
ε transitions

- NFAs can have a special transition label ε
- ε allows a transition *without* consuming any input tokens

In English, what pattern does this NFA therefore allow us to match?
- Any string starting with a “c”, followed by one or more instances of “ab”, followed by a “c”
Your turn – Woo!

Construct a NFA which accepts only binary numbers with an even number of zero’s.
Your turn – Woo!

Construct a NFA which accepts only binary numbers with an even number of zero’s.
Your turn – Woo!

Construct a \textit{NFA} which accepts only binary numbers with an even number of zero’s.

\begin{center}
\begin{tikzpicture}
\node[state, initial] (q0) at (0,0) {$q_0$};
\node[state, right of=q0] (q1) {$q_1$};
\path[->]
(q0) edge [loop above] node {1} ()
(q0) edge [below] node {0} (q1)
(q1) edge [loop above] node {1} ()
(q1) edge [below] node {0} (q0);
\end{tikzpicture}
\end{center}

As above, except for one’s instead of zero’s.
Your turn – Woo!

Construct a NFA which accepts only binary numbers with an even number of zero’s.

As above, except for one’s instead of zero’s.
Your turn

Construct a NFA which accepts only binary numbers with an even number of zero’s OR an even number of one’s.
Your turn

Construct a NFA which accepts only binary numbers with an even number of zero’s OR an even number of one’s.
You remember the power of regular expressions? These things can save the world! regex’s rule.
Regular Expressions as NFAs

- Regular expressions can easily be represented using NFAs
- We can group regular expressions into 4 different components
  - **Character** a single character: /a/
  - **Concatenation** two adjacent expressions: /ab/
  - **Union** two OR’d expressions: /a|b/
  - **Kleene star** zero or more repetitions: /a*/

/a(bc|d*)\*e/ can be viewed as
And it really does matter

Knowing the right way to do regular expressions, using good computer science, means that you don’t have to make major errors leading to really really bad performance, in a hardly-used bit of software, like say, Perl.

This is the time taken to match the sequence $a^n a^n$ against $a^n$ where the superscript represents string repeats, so $a^2 a^2 = a?a?aa$

Source: Russ Cox / rsc@swtch.com

For a great discussion of this, see http://swtch.com/~rsc/regexp/regexp1.html

---

Tim Dawborn

Finite State Automata and Determinisation

January, 2017
RE to NFA: Character

\[
/a/
\]

- Graph with two states, q0 and q1, connected by an arrow labeled 'a'.
**RE to NFA: Concatenation**

The regular expression `/ab/` becomes:

```
\[ \begin{array}{c}
\text{q0} \xrightarrow{a} \text{q1} \\
\text{q2} \xrightarrow{b} \text{q3}
\end{array} \]
```

becomes:

```
\[ \begin{array}{c}
\text{q0} \xrightarrow{a} \text{q1} \xrightarrow{\epsilon} \text{q2} \xrightarrow{b} \text{q3}
\end{array} \]
```
RE to NFA: Union

/a|b/

q0 \rightarrow a \rightarrow q1 \quad + \quad q2 \rightarrow b \rightarrow q3

becomes

q4 \rightarrow e \rightarrow q0 \rightarrow a \rightarrow q1 \quad e \quad q2 \rightarrow e \rightarrow b \rightarrow q3 \quad e \rightarrow q5
RE to NFA: Kleene Star

/a*/

\[ q_0 \xrightarrow{a} q_1 \]

becomes

\[ q_2 \xrightarrow{\epsilon} q_0 \xrightarrow{a} q_1 \xrightarrow{\epsilon} q_3 \]
An example conversion

\[/a(bc|d^*)^*e/\]

becomes

![Diagram of a finite state automaton](image-url)
DFAs vs NFAs

- How do DFAs differ from NFAs?
  - Deterministic (not non-deterministic)
  - No ε edges
  - Every edge from a node must have a unique label (can’t be in multiple states)
  - Every node must have an outward edge for each token of the alphabet (it is completely described)
DFAs vs. NFAs

- E.g. DFA for recognising a hexadecimal number:
Why do we need DFAs? What’s wrong with NFAs?

- The non-deterministic aspect of NFAs makes them bad for modern-day computation
- Computers cannot efficiently perform the “be in multiple states at once”
- In a DFA, you can only be in one state at a time
DFAs can be harder to manually construct than NFAs, due to their restrictions.

Construct a DFA over $\Sigma = \{a, b\}$ accepting strings ending in “bb”.
DFAs can be harder to manually construct than NFAs, due to their restrictions.

Construct a DFA over $\Sigma = \{a, b\}$ accepting strings ending in “bb”.

![DFA diagram]
NFA ≠ DFA

- At first glance, it would appear that NFAs are more powerful than DFAs. Why?
At first glance, it would appear that NFAs are more powerful than DFAs. Why?

- multiple edges with the same label coming off a node
- \(\varepsilon\) edges
- ability to be in multiple states at once
At first glance, it would appear that NFAs are more powerful than DFAs. Why?
- multiple edges with the same label coming off a node
- $\epsilon$ edges
- ability to be in multiple states at once

In 1959, Rabin and Scott proved that NFAs and DFAs have the same expressive power.

The proof is surprisingly simple: we need to show that for each NFA there is a DFA, and vice-versa, that accept precisely the same languages.
Determinisation Algorithm

- This algorithm converts any NFA into an equivalent DFA

\[
\begin{array}{c}
q_1 \\
\downarrow \\
q_2 \\
\downarrow \\
q_3 \\
\uparrow \\
q_4
\end{array}
\]

\[
\begin{array}{ccc}
e & 0 & e \\
0 & 0 & 1 \\
1 & e & 0
\end{array}
\]

\[
\begin{array}{c}
\{q_1, q_2, q_3\} \\
\{q_2, q_4\} \\
\{q_2, q_3\} \\
\{q_4\} \\
\{\} \\
\end{array}
\]

\[
\begin{array}{ccc}
0,1 & 0,1 & 0,1 \\
0 & 0 & 0 \\
1 & 1 & 1
\end{array}
\]

Tim Dawborn
Finite State Automata and Determinisation
January, 2017
Move $M(\text{states, token})$

Given a set of states and an input token, what set of states do you end up at

Epsilon-Closure $EC(\text{states})$

Given a set of states, what set of states do you get by expanding all $\varepsilon$ transitions

$$M(\{q_0, q_1\}, "a") = \{q_2, q_3\} \quad \text{EC}(\{q_2, q_3\}) = \{q_0, q_2, q_3\}$$
Algorithm Idea

- We need to remove the concept of non-determinism in the DFA
- Achieved by creating a state in the DFA for every possible set of states in the NFA
  - Q: If the NFA has \( n \) states, what’s the maximum number of states its corresponding DFA could have?
Algorithm Idea

- We need to remove the concept of non-determinism in the DFA
- Achieved by creating a state in the DFA for every possible set of states in the NFA
  - Q: If the NFA has $n$ states, what’s the maximum number of states its corresponding DFA could have?
  - A: $|\mathcal{P}(n)| = 2^n$
Pseudocode

Require: nfa

dfa ← a new DFA object
dfa.start ← EC(nfa.start)
todo ← [dfa.start]
while todo ≠ ∅ do
    states = pop the next item off todo
    for all σ ∈ Σ_{NFA} \ {ε} do
        s = EC(M(states, σ))
        if s ≠ ∅ then
            Add the edge states $\xrightarrow{\sigma}$ s to dfa
            Add s to todo if it’s new
        end if
    end for
end while
return dfa
Example

dodo

\{0\}
Example

done

\{0\}
Example

todo

\{0\}
\{0, 2\}
\{1\}
Example

todo
\{0\}
\{0, 2\}
\{1\}
\{0, 1, 2\}
\{1, 3\}
Example

todo

\{0\}
\{0, 2\}
\{1\}
\{0, 1, 2\}
\{1, 3\}
\{3\}
Example

todo

\{0\}
\{0, 2\}
\{1\}
\{0, 1, 2\}
\{1, 3\}
\{3\}
\{0, 2, 3\}
Example

todo

\{0\}
\{0, 2\}
\{1\}
\{0, 1, 2\}
\{1, 3\}
\{3\}
\{0, 2, 3\}
Example

**todo**

\{0\}
\{0, 2\}
\{1\}
\{0, 1, 2\}
\{1, 3\}
\{3\}
\{0, 2, 3\}
Example

\[\text{todo}\]
\[
\{0\},
\{0, 2\},
\{1\},
\{0, 1, 2\},
\{1, 3\},
\{3\},
\{0, 2, 3\}\]
End of presentation.