CS_275 Automata and Formal Language Theory

Course Notes
Part II: The Recognition Problem (II)
Chapter II.3.: Finite State Automata (13)

Anton Setzer
(Based on a book draft by J. V. Tucker and K. Stephenson)
Dept. of Computer Science, Swansea University

http://www.cs.swan.ac.uk/~csetzer/lectures/
automataFormalLanguage/current/index.html

March 11, 2015
II.3.1. String Recognition (13.1)

II.3.2. Nondeterministic Finite State Automata (13.2)

II.3.3. Examples of Automata (13.3)

II.3.4. Automata with Empty Move Transitions (13.4)

II.3.5. Deterministic Finite State Automata (13.6)
We want to define a program which recognises strings “start” and “stop”. We start by defining a program which recognises the letter “s”. This can be given by a system given by the following diagram, which will be our first automaton (automata will be introduced soon).

![Automaton Diagram](image-url)
Example Recognising Strings, Step 1

This automaton has the following ingredients.

- States $q_0$, $q_1$.
- State $q_0$ is the starting state, indicated by the arrow into it coming from nowhere.
- State $q_1$, the state indicating that we have recognised letter “s”.
- A transition, which when recognising letter “s”, goes from state $q_0$ to $q_1$. 
Example Recognising Strings, Step 2

In order to recognise the letter “t”, we extend this automaton as follows:

\[ q_0 \xrightarrow{s} q_1 \xrightarrow{t} q_2 \]
Example Recognising Strings, Step 2

- $q_0$ is the start state.
- $q_1$ indicates we have read string “s”.
- $q_2$ indicates we have read string “st”.

Diagram:

- $q_0$ is the start state.
- $q_1$ indicates we have read string “s”.
- $q_2$ indicates we have read string “st”.

$\text{s} \rightarrow q_0 \rightarrow q_1 \rightarrow q_2 \text{ t}$
Example Recognising Strings, Step 3

For the 3rd letter, we have two choices: “a” as part of the word “start”, and “o” as part of the word “stop”.

Diagram:

- $q_0$ to $q_1$ with label $s$
- $q_1$ to $q_2$ with label $t$
- $q_2$ to $q_3$ with label $a$
- $q_2$ to $q_6$ with label $o$
Example Recognising Strings, Step 3

- $q_0$ is the start state.
- $q_1$ indicates we have read string “s”.
- $q_2$ indicates we have read string “st”.
- $q_3$ indicates we have read string “sta”.
- $q_6$ indicates we have read string “sto”.

Diagram:

- $q_0$ is the start state.
- $q_1$ indicates we have read string “s”.
- $q_2$ indicates we have read string “st”.
- $q_3$ indicates we have read string “sta”.
- $q_6$ indicates we have read string “sto”.
We can complete our automaton and obtain the following
Example Recognising Strings, Step 3

This diagram contains a new ingredient: States $q_5$ and $q_7$ are accepting states. If we have processed a word and reached such a state then the word is accepted as a string of the language of the automaton.

(In our example $L = \{\text{start, stop}\}$. )
In order to recognise infinite languages, we need automata with loops. The following automaton recognises $L = \cdots$.

\[ q_0 \quad q_1 \]

Transitions:
- From $q_0$ to $q_0$: $a$
- From $q_0$ to $q_1$: $b$
- From $q_1$ to $q_0$: $b$
- From $q_1$ to $q_1$: $a$

Diagram:

\[ \begin{array}{c}
q_0 \\
\downarrow a \\
q_1 \\
\downarrow b \\
q_0 \\
\downarrow a \\
q_1 \\
\downarrow b \\
\end{array} \]
Automata with Loops

The following automaton recognises $L = \text{???}$

![Diagram of an automaton with states $q_0$ and $q_1$, transitions on $a$ and $b$, and loops on $b$.]
Automata with Loops

The following automaton recognises $L = ???$
The language \{\text{start, stop}\} can as well be recognised by the following \text{nondeterministic automaton}:
The automaton chooses in state $q_0$, non-deterministically, when in state $q_0$ and recognising a letter $s$, whether to go to $q_1$ or $q_6$.

The accepted language is the set of strings such that for each of them we obtain an accepting state for at least one non-deterministic choice.
If we try to accept the word “stop” by moving \( q_0 \xrightarrow{s} q_1 \xrightarrow{t} q_2 \) we get stuck at \( q_2 \).

That we fail to accept a word for one specific non-deterministic choice doesn’t imply that this word is not in the language.

A word is not accepted only if for all non-deterministic choices the corresponding run of the automaton doesn’t accept the string.
Why Nondeterministic Automata?

- When translating regular grammars into automata, we will obtain non-deterministic automata.
- We will show later that from a non-deterministic automaton we can obtain an equivalent deterministic automaton.
- Non-deterministic machine models play an important role in the theory of algorithms and complexity.
  - In some cases (as for automata), deterministic and non-deterministic are equivalent.
  - Sometimes they are not.
  - In other cases it is an open problem whether they are equivalent.
  - One example is the famous open “P = NP?” problem. Whether the problems recognised by deterministic (P) and non-deterministic (NP) Turing machines in polynomial time coincide or not.
In many exams I have seen students writing transitions, which are labelled by strings rather than single terminals:

Transitions of an automaton are labelled only by single elements of the alphabet.

Solution is to introduce intermediate states:
II.3.1. String Recognition (13.1)

II.3.2. Nondeterministic Finite State Automata (13.2)

II.3.3. Examples of Automata (13.3)

II.3.4. Automata with Empty Move Transitions (13.4)

II.3.5. Deterministic Finite State Automata (13.6)
Definition NFA

A non-deterministic finite state automaton, in short NFA \((Q, q_0, F, T, \rightarrow)\) is given by

- A finite set \(Q\) of states.
- A single initial state \(q_0 \in Q\).
- A set \(F \subseteq Q\) of accepting states.
- A finite set of terminal symbols \(T\).

\(\rightarrow\) is a relation between states \(q \in Q\), \(a \in T\) and states \(q' \in Q\). We write \(q \xrightarrow{a} q'\) if \(q, a, q'\) are in this relation.

More formally \(\rightarrow\) is a subset of \(Q \times T \times Q\) and \(q \xrightarrow{a} q'\) is an abbreviation for \((q, a, q') \in \rightarrow\).
Presentation of NFA by Picture

An NFA can be presented by a picture, like the following diagram:

\[ \text{Alphabet} = \{a, \ldots, z\} \]

- The arrow from nowhere into state \( q_0 \) denotes that \( q_0 \) is the initial state.
- Circles denote states.
- Arrows from a state \( q \) to \( q' \) labelled by \( a \) mean that \( q \xrightarrow{a} q' \).
- Double circle like for \( q_5 \) and \( q_9 \) denote the accepting states.
- The alphabet needs to be stated, unless all elements of the alphabet occur as labels of arrows, or it is clear from the context.
## Presentation of NFA by Table

<table>
<thead>
<tr>
<th>automaton</th>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>$q_0, \ldots, q_n$</td>
</tr>
<tr>
<td>terminals</td>
<td>$a_0, \ldots, a_m$</td>
</tr>
<tr>
<td>start</td>
<td>$q_i$</td>
</tr>
<tr>
<td>final</td>
<td>$q_{i_0}, q_{i_1}, \ldots, q_{i_m}$</td>
</tr>
<tr>
<td>transitions</td>
<td>$q_{k_1} \xrightarrow{a_{l_1}} q_{m_1}$</td>
</tr>
<tr>
<td></td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>
Example

The automaton for alphabet \( \{a, \ldots, z\} \) given by the diagram

is represented by the table on the next slide.
Example

automaton $A_{start,stop}$

states $q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7$

terminals $a, \ldots, z$

start $q_0$

final $q_5, q_9$

transitions

$q_0 \xrightarrow{s} q_1$
$q_0 \xrightarrow{s} q_6$
$q_1 \xrightarrow{t} q_2$
$q_2 \xrightarrow{a} q_3,$
$q_2 \xrightarrow{r} q_3,$
$q_3 \xrightarrow{t} q_4,$
$q_4 \xrightarrow{t} q_5,$
$q_6 \xrightarrow{t} q_7,$
$q_7 \xrightarrow{o} q_8,$
$q_8 \xrightarrow{p} q_9.$
The Extended Transition Relation

We define an extended transition relation which determines, whether the NFA from a state $q$ when reading word $w \in T^*$ can reach state $q'$:

**Definition**

Let $A = (Q, q_0, F, T, \rightarrow)$ be an NFA. Then we define $\rightarrow^*$ between states $q \in Q$, words $w \in T^*$ and $q' \in Q$.

We write $q \xrightarrow{w}\rightarrow^* q'$, if $q, w, q'$ are in this relation.

- If $w = a_0 \cdots a_n$, then

  $$q \xrightarrow{w}\rightarrow^* q' \text{ iff } q \xrightarrow{a_0} q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n = q'$$

  for some $q_i \in Q$

Especially $q \xrightarrow{\varepsilon} q'$ iff $q' = q$. 

The Extended Transition Relation

Again, more formally \( \rightarrow^* \) is a subset of \( Q \times T^* \times Q \) and \( q \overset{w}{\rightarrow}^* q' \) means that \((q, w, q') \in \rightarrow^*\)
For the automaton

we give some definitions of $\rightarrow^*$:
Example

For each of the words given in the following all possible transitions from $q_0$ will be given:

$\bullet \quad q_0 \xrightarrow{\epsilon}^* q_0$,  
$\bullet \quad q_0 \xrightarrow{s}^* q_1$, $q_0 \xrightarrow{s}^* q_6$,  
$\bullet \quad q_0 \xrightarrow{st}^* q_2$, $q_0 \xrightarrow{st}^* q_7$,  
$\bullet \quad q_0 \xrightarrow{sta}^* q_3$,  
$\bullet \quad q_0 \xrightarrow{star}^* q_4$,  
$\bullet \quad q_0 \xrightarrow{start}^* q_5$,  
$\bullet \quad q_0 \xrightarrow{sto}^* q_8$,  
$\bullet \quad q_0 \xrightarrow{stop}^* q_9$. 
The Language accepted by an NFA

Definition

Let \( A = (Q, q_0, F, T, \rightarrow) \) be an NFA. The **language accepted by** \( A \) is defined as

\[
L(A) := \{ w \in T^* \mid q_0 \xrightarrow{w}^* q \text{ for some } q \in F \}
\]
Operational Understanding of NFAs

To an NFA $A = (Q, q_0, F, T, \rightarrow)$ corresponds a non-deterministic program for checking whether an input string is in $L(A)$. The program will have

- a variable $q \in Q$,
- and a pointer to a position in the input string,
- a Boolean variable \textit{stopped}, which will be true if the automaton has stopped without having reached the end of the input string.

A run of the NFA on an input string $s$ is as given by the pseudo-code on the next slide:
Operational Understanding of NFAs

\{
\text{- Initialisation -} \}
\begin{align*}
q &:= q_0; \\
p &:= \text{beginning of word } s; \\
\text{stopped} &:= \text{false};
\end{align*}

\begin{algorithm}
\begin{algorithmic}
\While{\((p \neq \text{end of word } s) \land \neg \text{stopped}\)}
\State \(a := \text{next symbol in } s \text{ at position } p;\)
\If{\(\neg \exists q'. (q \xrightarrow{a} q')\)}
\State \(\text{stopped} := \text{true};\)
\Else
\State \(\text{choose } q' \text{ s.t. } q \xrightarrow{a} q';\)
\State \(q := q';\)
\State \(p := \text{position of next symbol in } s \text{ from position } p;\)
\EndIf
\EndWhile
\end{algorithmic}
\end{algorithm}
Operational Understanding of NFAs

- If at the end of the execution $stopped = \text{true}$ (therefore we have not reached the end of the string), the string is not accepted in this run.
- Otherwise the string is accepted if $q \in F$.

A string is accepted if there exist a sequence of non-deterministic choices, such that the string is accepted in the corresponding run.
Operational Understanding of NFAs

One can now see that a string $s$ is accepted by some run of the program for automaton $A$ iff $s \in L(A)$. 
II.3.1. String Recognition (13.1)

II.3.2. Nondeterministic Finite State Automata (13.2)

II.3.3. Examples of Automata (13.3)

II.3.4. Automata with Empty Move Transitions (13.4)

II.3.5. Deterministic Finite State Automata (13.6)
Example 1: An Automaton accepting \{1, 2, 3\}
Example 1: An Automaton accepting \{1, 2, 3\}

Display style:

<table>
<thead>
<tr>
<th>automaton</th>
<th>( A^{1,2,3} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>( q_0, q_1, q_2, q_3 )</td>
</tr>
<tr>
<td>terminals</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>start</td>
<td>( q_0 )</td>
</tr>
<tr>
<td>final</td>
<td>( q_1, q_2, q_3 )</td>
</tr>
</tbody>
</table>
| transitions| \( q_0 \xrightarrow{1} q_1 \)  \\
|            | \( q_0 \xrightarrow{2} q_2 \)  \\
|            | \( q_0 \xrightarrow{3} q_3 \)  |
Simplified Version

Graph:
- Two states: $q_0$ and $q_1$
- Arrows:
  - From $q_0$ to $q_1$: label 1
  - From $q_1$ to $q_0$: label 2
  - From $q_0$ to $q_0$: label 3
Simplified Version

Shorter pictorial presentation of the same automaton:
## Simplified Version

### Display style:

<table>
<thead>
<tr>
<th>automaton</th>
<th>$A^{1,2,3'}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>$q_0, q_1$</td>
</tr>
<tr>
<td>terminals</td>
<td>$1, 2, 3$</td>
</tr>
<tr>
<td>start</td>
<td>$q_0$</td>
</tr>
<tr>
<td>final</td>
<td>$q_1$</td>
</tr>
</tbody>
</table>
| transitions       | $q_0 \xrightarrow{0} q_1$
|                   | $q_0 \xrightarrow{1} q_1$
|                   | $q_0 \xrightarrow{2} q_1$ |
Example 2: Automaton accepting ???

$q_0$

0, 1, ..., 9
Simplified Version

Display style:

<table>
<thead>
<tr>
<th>term</th>
<th>notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>automaton</td>
<td>$A$</td>
</tr>
<tr>
<td>states</td>
<td>$q_0$</td>
</tr>
<tr>
<td>terminals</td>
<td>$0, \ldots, 9$</td>
</tr>
<tr>
<td>start</td>
<td>$q_0$</td>
</tr>
<tr>
<td>final</td>
<td>$q_0$</td>
</tr>
<tr>
<td>transitions</td>
<td>$q_0 \xrightarrow{a} q_0$ (for $a \in {0, 1, \ldots, 9}$)</td>
</tr>
</tbody>
</table>
Example 3: Automaton accepting $$0, 1, \ldots, 9$$
Display Style

Display style:

<table>
<thead>
<tr>
<th>Automaton</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>$q_0, q_1, q_2$</td>
</tr>
<tr>
<td>Terminals</td>
<td>$0, \ldots, 9$</td>
</tr>
<tr>
<td>Start</td>
<td>$q_0$</td>
</tr>
<tr>
<td>Final</td>
<td>$q_1, q_2$</td>
</tr>
<tr>
<td>Transitions</td>
<td></td>
</tr>
<tr>
<td>$q_0 \xrightarrow{0} q_2$</td>
<td></td>
</tr>
<tr>
<td>$q_0 \xrightarrow{a} q_1, \ (a \in {1, \ldots, 9})$</td>
<td></td>
</tr>
<tr>
<td>$q_1 \xrightarrow{a} q_1, \ (a \in {0, \ldots, 9})$</td>
<td></td>
</tr>
</tbody>
</table>
II.3.1. String Recognition (13.1)

II.3.2. Nondeterministic Finite State Automata (13.2)

II.3.3. Examples of Automata (13.3)

II.3.4. Automata with Empty Move Transitions (13.4)

II.3.5. Deterministic Finite State Automata (13.6)
Definition DFA

Let $A = (Q, q_0, F, T, \rightarrow)$ be an NFA. $A$ is a **deterministic finite state automaton**, in short **DFA**, if for all $q \in Q$ and $a \in T$ there exist at most one $q'$ s.t.

$$q \xrightarrow{a} q'$$

So deterministic finite state automata are those automata corresponding to real programs: we have never to make a choice.
Example (DFA)

The following NFA is a DFA:

Note that there is no transition from $q_0$ labelled by letter $\neq s$. For a DFA from each state and element of the alphabet there need to be at most one transition – it is possible to have no transition.
Example (not a DFA)

The following NFA is not a DFA:
Theorem II.3.5.1.

Let $A$ be an NFA.
Then there exists a DFA $A'$ s.t. $L(A) = L(A')$.
$A'$ can be computed from $A$. 
Example

Consider the NFA above which was not an DFA:
Definition of the DFA

- We define a DFA as follows:
- The states $Q'$ of the DFA are all possible sets of states $Q$ of the NFA.
  - Note that since the $Q$ is finite, $Q'$ is finite.
- The initial state is the set of initial states, here $\{q_0\}$. 
Definition of the DFA

- From a set of states $B$ we can make an $a$-transition to the set of states $q$, which can be reached from any state $q \in B$ by an $a$-transition.

- For instance
  - $\{q_0\} \xrightarrow{s} \{q_1, q_6\}$ since the set of states we can reach by an $s$-transition from $q_0$ are states $q_1, q_6$.
  - $\{q_1, q_6\} \xrightarrow{t} \{q_2, q_7\}$ since the set of states we can reach by a $t$-transition from any of the states $q_1$ or $q_6$ are the states $q_2, q_7$.

- A set of states is an accepting state, if at least one of the states in it is in $F$.
  - For instance $\{q_9\}$ or $\{q_5\}$ or $\{q_4, q_9\}$ are accepting.
Optimisation of the DFA

- It is not necessary to add all subsets of $Q$ to $Q'$:
  - We can omit $\emptyset$:
    - $\emptyset$ is not an accepting state, and $\emptyset \xrightarrow{a} \emptyset$.
    - So $\emptyset$ is a *sink* of the DFA.
      It’s non-accepting and all transitions from it go back to it (why?). So we can’t escape from it (why?).
    - We can omit it and all transitions into it.
  - We can as well omit all states which are unreachable from the initial state.
- So we get a (usually much smaller) automaton, by starting from the initial state $\{q_0\}$, and successively determine all transitions of the DFA which don’t end up in $\emptyset$.
- We give two examples and then introduce an algorithm for transforming an NFA into a DFA.
Example above Repeated
Using the technique above we obtain the following automaton:

Note that this was the original DFA above (up to renaming of states).
Example 2

Consider the following NFA accepting $L = \ldots$:

\[
\begin{array}{c}
  q_0 \\
  \downarrow \\
  a \\
\end{array}
\quad \xrightarrow{a,b} \quad
\begin{array}{c}
  q_1 \\
  \uparrow \\
  b \\
\end{array}
\]
Corresponding DFA

\[
\begin{align*}
\{q_0\} &\xrightarrow{a} \{q_0, q_1\} \\
\{q_0, q_1\} &\xrightarrow{a} \{q_0, q_1\} \\
\{q_0\} &\xrightarrow{b} \{q_1\} \\
\{q_1\} &\xrightarrow{b} \{q_1\}
\end{align*}
\]
Algorithm for Obtaining from an NFA an Equivalent DFA

- **Input:** An NFA \((Q, q_0, F, T, \rightarrow)\).
- **Output:** A DFA \((Q', q'_0, F', T', \rightarrow')\).
- \(T' := T, q'_0 := \{q_0\}\).
- We first determine iteratively

  \[
  Q' \quad \text{(which consists of subsets of } Q) \\
  \rightarrow'
  \]

  Simultaneously we define iteratively the set of states

  \[
  \text{Treated}(Q') \subseteq Q'
  \]

  These are the set of states for which we have defined \(\rightarrow'\) originating from it completely.
- Initially let

  \[
  Q' := \{q'_0\} \\
  \rightarrow' := \emptyset \\
  \text{Treated}(Q') := \emptyset
  \]
Algorithm for Obtaining an Equivalent DFA from an NFA

- As long as there is a state
  \[ \{q_1, \ldots, q_n\} \in Q' \setminus \text{Treated}(Q') \]
  select one and do the following:
  - If for some \( a \in T \) there is at least one \( i \) and at least one transition \( q_i \xrightarrow{a} q' \) for some \( a \in T \) do the following:
    - For each such \( a \) let
      \[ \{q_1', \ldots, q_m'\} := \{q' \in Q \mid \exists i \in \{1, \ldots, n\}. q_i \xrightarrow{a} q'\} \]
      So \( \{q_1', \ldots, q_m'\} \) are all states which are obtained by an \( a \)-transition from at least one state in \( \{q_1, \ldots, q_n\} \).
    - If
      \[ \{q_1', \ldots, q_m'\} \not\in Q' , \]
      then add \( \{q_1', \ldots, q_m'\} \) to \( Q' \).
    - Then add to \( \rightarrow' \)
      \[ \{q_1, \ldots, q_n\} \xrightarrow{a} \{q_1', \ldots, q_m'\} \]
  - Once this has been done for all \( a \), then add \( \{q_1, \ldots, q_n\} \) to \( \text{Treated}(Q') \).
Algorithm for Obtaining an Equivalent DFA from an NFA

- If there was no such $a$ then add $\{q_1, \ldots, q_n\}'$ to $\text{Treated}(Q')$ without adding any element to $\rightarrow'$. 

- Once $Q' = \text{Treated}(Q')$, so the loop above has been completed, we complete the definition of the DFA by defining

$$F' := \{\{q_1, \ldots, q_n\} \in Q' \mid \{q_1, \ldots, q_n\} \cap F \neq \emptyset\}$$

So $F'$ is the set of states in $Q'$, which contain at least one state in $F$. 
II.3.5. Deterministic Finite State Automata (13.6)

Proof Idea of Equivalence between NFA and generated DFA

- One can easily see that in the DFA we have that for all words $w \in T^*$

  \[ \{q_0\} \xrightarrow{w}^* \{q_1, \ldots, q_n\} =: B \]

  iff

  \[ B = \{ q \mid q_0 \xrightarrow{w}^* q \} \]

  so $B$ is the set of states which we can reach from $q_0$ by a $w$-transition.

- Therefore in the DFA we reach an accepting state (one containing an element of $F$) by a $w$-transition,

  iff the NFA can reach an accepting state by a $w$-transition.

- Therefore the language accepted by this DFA is the same as the language accepted by the original NFA.

- A formal proof can be found in the Additional Material.
DFA are special cases of NFA

- Note that every DFA is an NFA.
- If you are
  - asked to define an NFA and derive from it a DFA,
  - and your first idea is already deterministic, i.e. a DFA,
- then your solution can be used as an answer for both the NFA and the DFA question.
- So you don’t need to make your initial solution non-deterministic.