15-453
FORMAL LANGUAGES, AUTOMATA AND COMPUTABILITY
NON-DETERMINISM and REGULAR OPERATIONS

THURSDAY JAN 16
UNION THEOREM
The union of two regular languages is also a regular language

“Regular Languages Are Closed Under Union”

INTERSECTION THEOREM
The intersection of two regular languages is also a regular language
Complement **THEOREM**

The complement of a regular language is also a regular language

In other words,

if L is regular than so is \( \neg L \),

where \( \neg L = \{ w \in \Sigma^* | w \notin L \} \)

Proof ?
L(M) = \{ w \mid w \text{ begins with } 1 \}
Suppose our machine reads strings from right to left…

What language would be recognized then?
Suppose our machine reads strings from right to left…

What language would be recognized then?

L(M) = \{ w | w begins with 1\}

L(M) = \{ w | w ends with 1\}  Is L(M) regular?
L(M) = \{ w \mid w \text{ ends with 1} \}

Is L(M) regular?
THE REVERSE OF A LANGUAGE

Reverse: \( L^R = \{ w_1 \ldots w_k | w_k \ldots w_1 \in L, w_i \in \Sigma \} \)

If \( L \) is recognized by a normal DFA, then \( L^R \) is recognized by a DFA reading from right to left!

Can every “Right-to-Left DFA” be replaced by a normal DFA??
REVERSE THEOREM

The reverse of a regular language is also a regular language

``Regular Languages Are Closed Under Reverse``

If a language can be recognized by a DFA that reads strings from right to left, then there is an “normal” DFA that accepts the same language
REVERSING DFAs

Assume $L$ is a regular language. Let $M$ be a DFA that recognizes $L$.

**Task:** Build a DFA $M^R$ that accepts $L^R$.

If $M$ accepts $w$, then $w$ describes a directed path in $M$ from *start* to an *accept* state.
Assume $L$ is a regular language. Let $M$ be a DFA that recognizes $L$.

**Task:** Build a DFA $M^R$ that accepts $L^R$.

If $M$ accepts $w$, then $w$ describes a directed path in $M$ from start to an accept state.

**First Attempt:**
Try to define $M^R$ as $M$ with the arrows reversed. 
Turn start state into a final state.
Turn final states into start states.
$M^R$ IS NOT ALWAYS A DFA!

It could have many start states

Some states may have too many outgoing edges,

or none at all!
What happens with 100?
What happens with 100?

We will say that this machine accepts a string if there is some path that reaches an accept state from a start state.
NONDETERMINISM is BORN!

What happens with 100?

We will say that this machine **accepts** a string if there is some path that reaches an accept state from a start state.
Finite Automata and Their Decision Problems

Abstract: Finite automata are considered in this paper as instruments for classifying finite tapes. Each one-tape automaton defines a set of tapes, a two-tape automaton defines a set of pairs of tapes, et cetera. The structure of the defined sets is studied. Various generalizations of the notion of an automaton are introduced and their relation to the classical automata is determined. Some decision problems concerning automata are shown to be solvable by effective algorithms; others turn out to be unsolvable by algorithms.

Introduction

Turing machines are widely considered to be the abstract prototype of digital computers; workers in the field, however, have felt more and more that the notion of a Turing machine is too general to serve as an accurate model of actual computers. It is well known that even for simple calculations it is impossible to give an a priori upper bound on the amount of tape a Turing machine will need for any given computation. It is precisely this feature that renders Turing's concept unrealistic.

In the last few years the idea of a finite automaton has appeared in the literature. These are machines having a method of viewing automata but have retained throughout a machine-like formalism that permits direct comparison with Turing machines. A neat form of the definition of automata has been used by Burks and Wang and by E. F. Moore, and our point of view is closer to theirs than it is to the formalism of nerve-nets. However, we have adopted an even simpler form of the definition by doing away with a complicated output function and having our machines simply give “yes” or “no” answers. This was also used by Myhill, but our generalizations to the “nondeterministic,” “two-way,” and “many-tape”
Theorem 19. There is no effective method of deciding whether the set of tapes definable by a two-tape, two-way automaton is empty or not.

An argument similar to the above one will show that the class of sets of pairs of tapes definable by two-way, two-tape automata is closed under Boolean operations. In view of Theorem 17, this implies that there are sets definable by two-way automata which are not definable by any one-way automaton; thus no analogue to Theorem 15 holds.

References


Revised manuscript received August 8, 1958
At each state, we can have *any* number of out arrows for each letter $\sigma \in \Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.
Possibly many start states

\[ L(M) = \{0^i1^j \mid i \in \{0,1\}, j \geq 0\} \]
NFA EXAMPLES

L(M)={1,00}
A non-deterministic finite automaton (NFA) is a 5-tuple \( N = (Q, \Sigma, \delta, Q_0, F) \)

- \( Q \) is the set of states
- \( \Sigma \) is the alphabet
- \( \delta : Q \times \Sigma \epsilon \rightarrow 2^Q \) is the transition function
- \( Q_0 \subseteq Q \) is the set of start states
- \( F \subseteq Q \) is the set of accept states

\[ 2^Q \text{ is the set of all possible subsets of } Q \]

\[ \Sigma \epsilon = \Sigma \cup \{ \epsilon \} \]
Let \( w \in \Sigma^* \) and suppose \( w \) can be written as \( w_1 \ldots w_n \) where \( w_i \in \Sigma_\varepsilon \) (\( \varepsilon \) = empty string).

Then \( N \) accepts \( w \) if there are \( r_0, r_1, \ldots, r_n \in Q \) such that:

1. \( r_0 \in Q_0 \)
2. \( r_{i+1} \in \delta(r_i, w_{i+1}) \) for \( i = 0, \ldots, n-1 \), and
3. \( r_n \in F \)

\( L(N) \) = the language recognized by \( N \)

= set of all strings machine \( N \) accepts

A language \( L \) is recognized by an NFA \( N \) if \( L = L(N) \).
Deterministic Computation

accept or reject

Non-Deterministic Computation

accept

reject
Deterministic Computation

accept or reject

Non-Deterministic Computation

accept
\[ N = (Q, \Sigma, \delta, Q_0, F) \]

\[ Q = \{q_1, q_2, q_3, q_4\} \]

\[ \Sigma = \{0, 1\} \]

\[ Q_0 = \{q_1, q_2\} \]

\[ F = \{q_4\} \subseteq Q \]

\[ \delta(q_2, 1) = \{q_4\} \]

\[ \delta(q_3, 1) = \emptyset \quad \delta(q_3, \varepsilon) = \{q_2\} \]

\[ \delta(q_1, 0) = \{q_3\} \]

\[ 00 \in L(N) \]

\[ 01 \in L(N) \]
$N = (Q, \Sigma, \delta, Q_0, F)$

$Q = \{q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$

$Q_0 = \{q_1, q_2\}$

$F = \{q_4\} \subseteq Q$

\[
\begin{array}{|c|c|c|c|}
\hline
\delta & 0 & 1 & \varepsilon \\
\hline
q_1 & \{q_3\} & \emptyset & \emptyset \\
q_2 & \emptyset & \{q_4\} & \emptyset \\
q_3 & \{q_4\} & \emptyset & \{q_2\} \\
q_4 & \emptyset & \emptyset & \emptyset \\
\hline
\end{array}
\]
\( N = (Q, \Sigma, \delta, Q_0, F) \)

- \( Q = \{q_1, q_2, q_3, q_4\} \)
- \( \Sigma = \{0, 1\} \)
- \( Q_0 = \{q_1, q_2\} \)
- \( F = \{q_4\} \subseteq Q \)

\[
\begin{array}{|c|c|c|c|}
\hline
\delta & 0 & 1 & \varepsilon \\
\hline
q_1 & \{q_2, q_3\} & \emptyset & \emptyset \\
q_2 & \emptyset & \{q_4\} & \emptyset \\
q_3 & \{q_4\} & \emptyset & \emptyset \\
q_4 & \emptyset & \emptyset & \emptyset \\
\hline
\end{array}
\]
$$N = (Q, \Sigma, \delta, Q_0, F)$$

$$Q = \{q_1, q_2, q_3, q_4\}$$

$$\Sigma = \{0, 1\}$$

$$Q_0 = \{q_1, q_2\}$$

$$F = \{q_4\} \subseteq Q$$

<table>
<thead>
<tr>
<th>(\delta)</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_1)</td>
<td>{q_2, q_3}</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(\emptyset)</td>
<td>{q_4}</td>
</tr>
<tr>
<td>(q_3)</td>
<td>{q_4}</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(q_4)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>
We allow *multiple* start states for NFAs, and Sipser allows only one.

Can easily convert NFA with many start states into one with a single start state:
UNION THEOREM FOR NFAs?
UNION THEOREM FOR NFAs?
NFAs ARE SIMPLER THAN DFAs

A DFA that recognizes the language \{1\}:
NFAs ARE SIMPLER THAN DFAs

An NFA that recognizes the language \{1\}:

A DFA that recognizes the language \{1\}:
Theorem: Every NFA has an equivalent* DFA

Corollary: A language is regular iff it is recognized by an NFA

Corollary: L is regular iff $L^R$ is regular

* N is equivalent to M if $L(N) = L(M)$
FROM NFA TO DFA

Input: NFA $N = (Q, \Sigma, \delta, Q_0, F)$
Output: DFA $M = (Q', \Sigma, \delta', q_0', F')$

To learn if NFA accepts, we could do the computation in parallel, maintaining the set of all possible states that can be reached.

Idea:

$Q' = 2^Q$
FROM NFA TO DFA

Input: NFA $N = (Q, \Sigma, \delta, Q_0, F)$

Output: DFA $M = (Q', \Sigma, \delta', q_0', F')$

$Q' = 2^Q$

$\delta' : Q' \times \Sigma \rightarrow Q'$

$\delta'(R, \sigma) = \bigcup_{r \in R} \varepsilon(\delta(r, \sigma))$

$q_0' = \varepsilon(Q_0)$

$F' = \{ R \in Q' \mid f \in R \text{ for some } f \in F \}$

For $R \subseteq Q$, the $\varepsilon$-closure of $R$, $\varepsilon(R) = \{ q \text{ that can be reached from some } r \in R \text{ by traveling along zero or more } \varepsilon \text{ arrows} \}$
EXAMPLE OF $\varepsilon$-CLOSURE

$\varepsilon(\{q_0\}) = \varepsilon(\{q_1\}) = \varepsilon(\{q_2\}) =$
EXAMPLE OF $\varepsilon$-CLOSURE

$\varepsilon(\{q_0\}) = \{q_0, q_1, q_2\}$

$\varepsilon(\{q_1\}) = \{q_1, q_2\}$

$\varepsilon(\{q_2\}) = \{q_2\}$
Given: NFA $N = ( \{1,2,3\}, \{a,b\}, \delta, \{1\}, \{1\} )$

Construct: Equivalent DFA $M$

$$M = (2^{\{1,2,3\}}, \{a,b\}, \delta', \{1,3\}, \ldots)$$

$\mathcal{E}(\{1\}) = \{1,3\}$
Given: NFA $N = (\{1,2,3\}, \{a,b\}, \delta, \{1\}, \{1\})$

Construct: Equivalent DFA $M$

$M = (2^{\{1,2,3\}}, \{a,b\}, \delta', \{1,3\}, ...)$

$\varepsilon(\{1\}) = \{1,3\}$
Given: NFA $N = (\{1,2,3\}, \{a,b\}, \delta, \{1\}, \{1\})$

Construct: Equivalent DFA $M$

$M = (2^{\{1,2,3\}}, \{a,b\}, \delta', \{1,3\}, \ldots)$

$\epsilon(\{1\}) = \{1,3\}$

$\epsilon(\{1\}) = \{1,3\}$
Given: NFA $N = (\{1,2,3\}, \{a, b\}, \delta, \{1\}, \{1\})$

Construct: equivalent DFA $M = (Q', \Sigma, \delta', q_0', F')$

$$q_0' = \epsilon(\{1\}) = \{1,3\}$$

$\delta'$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${1}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${2}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${3}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${1,2}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${1,3}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${2,3}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${1,2,3}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Given: NFA  \( N = ( \{1,2,3\}, \{a,b\}, \delta, \{1\}, \{1\} ) \)

Construct: equivalent DFA  \( M = (Q', \Sigma, \delta', q_0', F') \)

\[ \delta' \]

<table>
<thead>
<tr>
<th>( \delta' )</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {1} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {2} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {3} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {1,2} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {1,3} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {2,3} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {1,2,3} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( q_0' = \varepsilon(\{1\}) = \{1,3\} \)
Given: NFA  \( N = ( \{1,2,3\}, \{a,b\}, \delta, \{1\}, \{1\} ) \)

Construct: equivalent DFA  \( M = (Q', \Sigma, \delta', q_0', F') \)

\[ \begin{array}{c|cc}
\delta' & a & b \\
\hline
\emptyset & \emptyset & \emptyset \\
\{1\} & \emptyset & \{2\} \\
\{2\} & \{2,3\} & \{3\} \\
\{3\} & \{1,3\} & \emptyset \\
\{1,2\} & \{2,3\} & \{2,3\} \\
\{1,3\} & \{1,3\} & \{2\} \\
\{2,3\} & \{1,2,3\} & \{3\} \\
\{1,2,3\} & \{1,2,3\} & \{2,3\} \\
\end{array} \]

\( q_0' = \varepsilon(\{1\}) = \{1,3\} \)
Given: NFA \( N = ( \{1,2,3\}, \{a,b\}, \delta, \{1\}, \{1\} ) \)

Construct: equivalent DFA \( M = (Q', \Sigma, \delta', q_0', F') \)
NFAs CAN MAKE PROOFS MUCH EASIER!

Remember this on your Homework!
REGULAR LANGUAGES CLOSED UNDER CONCATENATION

Concatenation: $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$

Given DFAs $M_1$ and $M_2$, connect accept states in $M_1$ to start states in $M_2$

$L(N) = L(M_1) \cdot L(M_2)$
REGULAR LANGUAGES CLOSED UNDER CONCATENATION

**Concatenation:** $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$

Given DFAs $M_1$ and $M_2$, connect accept states in $M_1$ to start states in $M_2$

$L(N) = L(M_1) \cdot L(M_2)$
REGULAR LANGUAGES CLOSED UNDER CONCATENATION

**Concatenation:** $A \cdot B = \{ vw | v \in A \text{ and } w \in B \}$

Given DFAs $M_1$ and $M_2$, connect accept states in $M_1$ to start states in $M_2$

$L(N) = L(M_1) \cdot L(M_2)$
RLs ARE CLOSED UNDER STAR

Star: \( A^* = \{ s_1 \ldots s_k \mid k \geq 0 \text{ and each } s_i \in A \} \)

Let \( M \) be a DFA, and let \( L = L(M) \)

Can construct an NFA \( N \) that recognizes \( L^* \)
RLs ARE CLOSED UNDER STAR

Star: \( A^* = \{ s_1 \ldots s_k \mid k \geq 0 \text{ and each } s_i \in A \} \)

Let \( M \) be a DFA, and let \( L = L(M) \)

Can construct an NFA \( N \) that recognizes \( L^* \)
**RLs ARE CLOSED UNDER STAR**

**Star:** \( A^* = \{ s_1 \ldots s_k \mid k \geq 0 \text{ and each } s_i \in A \} \)

Let \( M \) be a DFA, and let \( L = L(M) \)

Can construct an NFA \( N \) that recognizes \( L^* \)
Formally:

Input: \( M = (Q, \Sigma, \delta, q_1, F) \)

Output: \( N = (Q', \Sigma, \delta', \{q_0\}, F') \)

\[
Q' = Q \cup \{q_0\}
\]

\[
F' = F \cup \{q_0\}
\]

\[
\delta'(q, a) = \begin{cases} 
\{\delta(q, a)\} & \text{if } q \in Q \text{ and } a \neq \varepsilon \\
\{q_1\} & \text{if } q \in F \text{ and } a = \varepsilon \\
\{q_1\} & \text{if } q = q_0 \text{ and } a = \varepsilon \\
\emptyset & \text{if } q = q_0 \text{ and } a \neq \varepsilon \\
\emptyset & \text{else}
\end{cases}
\]
Show: \( L(N) = L^* \) where \( L = L(M) \)

1. \( L(N) \supseteq L^* \)

2. \( L(N) \subseteq L^* \)
1. \( L(N) \supseteq L^* \) (where \( L = L(M) \))

Assume \( w = w_1 \ldots w_k \) is in \( L^* \), where \( w_1, \ldots, w_k \in L \)

We show \( N \) accepts \( w \) by induction on \( k \)

Base Cases:

- \( k = 0 \) (\( w = \varepsilon \))
- \( k = 1 \) (\( w \in L \))
Assume $w = w_1\ldots w_k$ is in $L^*$, where $w_1,\ldots, w_k \in L$

We show $N$ accepts $w$ by induction on $k$

**Base Cases:**
- $k = 0$ \hspace{1cm} ($w = \epsilon$)
- $k = 1$ \hspace{1cm} ($w \in L$)

**Inductive Step:**
Assume $N$ accepts all strings $v = v_1\ldots v_k \in L^*$, $v_i \in L$
and let $v = v_1\ldots v_k v_{k+1} \in L^*$, $u_j \in L$

Since $N$ accepts $v_1\ldots v_k$ (by induction) and $M$ accepts $v_{k+1}$, $N$ must accept $v$
2. \( L(N) \subseteq L^* \) (where \( L = L(M) \))

Assume \( w \) is accepted by \( N \), we show \( w \in L^* \)

If \( w = \varepsilon \) or \( w \in L \), then \( w \in L^* \)
2. $L(N) \subseteq L^*$ (where $L = L(M)$)

Assume $w$ is accepted by $N$, we show $w \in L^*$

If $w = \epsilon$ or $w \in L$, then $w \in L^*$

If $w \neq \epsilon$ or $w \notin L$

write $w$ as $w = uv$,

where $v$ is the substring read after the last $\epsilon$-transition
2. $L(N) \subseteq L^*$ (where $L = L(M)$)

Assume $w$ is accepted by $N$, we show $w \in L^*$

If $w = \varepsilon$ or $w \in L$, then $w \in L^*$

If $w \neq \varepsilon$ or $w \not\in L$
write $w$ as $w=uv$, where $v$ is the substring read
after the last $\varepsilon$-transition
Assume $w$ is accepted by $N$, we show $w \in L^*$

If $w = \varepsilon$ or $w \in L$, then $w \in L^*$

If $w \neq \varepsilon$ or $w \notin L$
write $w$ as $w=uv$,
where $v$ is the substring read
after the last $\varepsilon$-transition

$$\varepsilon \quad \varepsilon \quad \varepsilon$$

accept
Assume \( w \) is accepted by \( N \), we show \( w \in L^* \)

If \( w = \varepsilon \) or \( w \in L \), then \( w \in L^* \)

If \( w \neq \varepsilon \) or \( w \notin L \)

write \( w \) as \( w = uv \),

where \( v \) is the substring read after the last \( \varepsilon \)-transition

\[ 2. L(N) \subseteq L^* \quad \text{(where} \ L = L(M) \text{)} \]
2. $L(N) \subseteq L^*$ (where $L = L(M)$)

Assume $w$ is accepted by $N$, we show $w \in L^*$

If $w = \varepsilon$ or $w \in L$, then $w \in L^*$

If $w \neq \varepsilon$ or $w \notin L$
write $w$ as $w = uv$, where $v$ is the substring read after the last $\varepsilon$-transition

By induction

$u \in L^*$

$v \in L$
2. \( L(N) \subseteq L^* \)  (where \( L = L(M) \))

Assume \( w \) is accepted by \( N \), we show \( w \in L^* \).

If \( w = \varepsilon \) or \( w \in L \), then \( w \in L^* \).

If \( w \neq \varepsilon \) or \( w \notin L \), write \( w \) as \( w = uv \), where \( v \) is the substring read after the last \( \varepsilon \)-transition.

By induction, \( u \in L^* \) and \( v \in L \).

So, \( w = uv \in L^* \).
REGULAR LANGUAGES ARE CLOSED UNDER THE REGULAR OPERATIONS

**Union:** \( A \cup B = \{ w \mid w \in A \text{ or } w \in B \} \)

**Intersection:** \( A \cap B = \{ w \mid w \in A \text{ and } w \in B \} \)

**Negation:** \( \neg A = \{ w \in \Sigma^* \mid w \notin A \} \)

**Reverse:** \( A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A \} \)

**Concatenation:** \( A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \} \)

**Star:** \( A^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in A \} \)
Read Chapters 1.3 and 1.4 of the book for next time