The class of languages of finite state automata

Establishing the limits of finite state automata as a model of computation
The plan

We already know (Theorem 11.3) that, given a regular language, there always exists a finite state automaton that recognizes it. Here we intend to answer the question:

Is it possible to build FSAs for recognizing other classes of languages, in addition to regular (for example: context-free)?

– Answering this question yields an understanding of the class of problems admitting an algorithmic solution which can be modeled with an FSA
Let’s introduce the string automata

**Definition 12.1:** A **string automaton** (SA) is a quintuple: 
\[ S = (Q, L, \delta, q_i, F) \]

\[ \delta : Q \times L \rightarrow P(Q) \]

where \( Q \) is a finite set of states, \( q_i \in Q \) is the initial state, \( F \subseteq Q \) the set of accept states, \( \delta \) is a transition relation, called **string transition**; and \( L \) is **a finite language**

– Notice that alphabets are allowed as a special case of finite language. Hence the term: **generalized automaton**
A simple illustration

Let $M = (\{S,A,B,C,D,E\}, \{00,0101,11,00\}, t, S, \{B,E\})$ with $t$ defined by the table:

<table>
<thead>
<tr>
<th>$t$</th>
<th>00</th>
<th>0101</th>
<th>11</th>
<th>000</th>
<th>$\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>${A,D}$</td>
</tr>
<tr>
<td>$A$</td>
<td>${B}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B$</td>
<td></td>
<td></td>
<td>${C}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D$</td>
<td></td>
<td></td>
<td></td>
<td>${E}$</td>
<td></td>
</tr>
<tr>
<td>$E$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>${E}$</td>
</tr>
</tbody>
</table>

After some calculations it can be seen that:

$L(M) = \{000101, 11, 11000, 11000000, \ldots \}$
Are string automata just a fancy but essentially empty generalization?

Not quite!!!!

Think of a network of processors collaborating in the solution of a problem. Each processor is assigned a well-defined sub-problem, which is modeled as an automaton decision problem. Now, assume that you represent your network as a quintuple, where each state of the model represents an automaton (a processor), and that the communications of sub-problem solutions is the transition relation…

What do you get?
Inherited properties: FSA properties that are preserved by SAs

SAs have *many things in common* with FSAs:

(a) There are deterministic and nondeterministic string automata. Their differences are the similar to those that tell DFSA apart from NFSA

(b) String automata compute and accept strings in the same way finite state automata do; and

(c) Two string automata are equivalent if and only if they recognize the same language
Novel properties of string automata

Some SAs properties are not shared with FSAs:

(a) Equivalent SAs may be obtained by applying string operations to the elements in the finite language $L$. This induces some alterations on the transition relation, and eliminates states.

The rest of this lecture is devoted to a particularly useful form of this type of reduction

(b) Also, as pointed out in the definition, unlike finite state automata, a string automaton has a finite set instead of an alphabet in its definition

Think about this!!!
Non-trivial conversion of a DFSA into an equivalent SA

The **trivial conversion is the DFSA itself**. A non-trivial conversion involves **state and transition modifications**. Let assume that

\[ M = (Q, A, \partial, q_0, F) \]

is a formal representation of a deterministic finite state automaton. Non-trivial conversions concentrate in eliminating **“internal states”**. These are states \( q \) satisfying:

\[ (\exists q_1, q_2 \in Q) \land (\exists a, b \in A) \land q \in \partial(q_1, a) \land q_2 \in \partial(q, b) \]

Internal states are eliminated in such the language of the automaton is preserved
Paths through a state

**Definition:** Given an internal state $q$ in a string automaton, we call an open path through state $q$ any sequence $(r, w_1) \rightarrow (q, w_2) \rightarrow s$ satisfying:

$$\exists w_1, w_2 \in L \land q \in \partial(r, w_1) \land s \in \partial(q, w_2) \land r \neq q \land s \neq q$$

An open path may have an internal loop at $q$. This is the case of $(\exists w_1, w_2, w_3) q \in \partial(r, w_1) \land q \in \partial(q, w_3) \land s \in \partial(q, w_2)$

We call closed or loop path around $q$, a path of the form of $(q, w_1) \rightarrow (s, w_2) \rightarrow q$. This kind of path occurs if $(\exists w_1, w_2) s \in \partial(q, w_1) \land q \in \partial(s, w_2)$

A loop path may involve a much larger number of states.
Method 12.1
State elimination method

The following conversion rules operate on strings that represent regular expressions

1. For each path through a fixed state \( q \):
   - If the path comes from
     \[
     (\exists w_1, w_2) q \in \partial(r, w_1) \land s \in \partial(q, w_2) \quad \text{(no loop in } q)\]
   - replace these transitions with
     \[
     s \in \partial(r, w_1 \circ w_2)\]
   - If the path comes from
     \[
     (\exists w_1, w_2, w_3) q \in \partial(r, w_1) \land q \in \partial(q, w_3) \land s \in \partial(q, w_2) \quad \text{(loop : } w_3)\]
   - replace these transitions with
     \[
     s \in \partial(r, w_1 \circ w^*_3 \circ w_2)\]
   - If the path contains a loop path around \( q \), this is:
     \[
     (\exists w_1, w_2, w_3) q \in \partial(r, w_1) \land s \in \partial(q, w_2) \land q \in \partial(s, w_3)\]
   - Replace with
     \[
     s \in \partial(r, w_1 \circ (w_2 \circ w_3)^*)\]
Method 12.1 (cont.)

2. Replace each pair of string transitions obtained in Step 1 that are of the form:
   \[ s \in \partial(r, \sigma_1) \land s \in \partial(r, \sigma_2) \]
   with a the string transition
   \[ s \in \partial(r, \sigma_1 + \sigma_2) \]
Consider the following string transitions:

\[ T(r, 00) = \{q\}; \quad T(r, 01) = \{s\}; \]
\[ T(q, 10) = \{s\}; \quad T(q, 11) = \{s\} \]

There is two state paths: \((r, 00) \rightarrow (q, 10) \rightarrow s\) and \((r, 00) \rightarrow (q, 11) \rightarrow s\)

By applying Step 1 to each path, we get

\[ T(r, 00 \circ 10) = \{s\}; \quad T(r, 00 \circ 11) = \{s\} \]

These two new transitions together with the old one \((T(r, 01) = \{s\})\) are now modified by Step 2, to get the string transition:

\[ T(r, 00 \circ 10 + 00 \circ 11 + 01) = \{s\} \]
Illustration 2

Consider the following transitions:

\[ T(r, 00) = \{q\}; \quad T(t, 01) = \{q\}; \]
\[ T(q, 10) = \{s\}; \quad T(q, 11) = \{p\} \]

There are four paths: 
- \( (r, 00) \to (q, 10) \to s \),
- \( (r, 00) \to (q, 11) \to p \),
- \( (t, 01) \to (q, 10) \to s \), and 
- \( (t, 01) \to (q, 11) \to p \)

By applying Step 1 to each path we get:

\[ T(r, 00 \circ 10) = \{s\}; \quad T(r, 00 \circ 11) = \{p\} \]
\[ T(t, 01 \circ 10) = \{s\}; \quad T(t, 01 \circ 11) = \{p\} \]

Step 2 does not apply, since no pair of transitions share the start and end states.
Illustration 3

Consider the following string transitions:

\[ T(r, 00) = \{ q \}; \quad T(t, 01) = \{ q \}; \]
\[ T(q, 10) = \{ q \}; \quad T(q, 11) = \{ s \} \]

There are two state paths:

- \((r, 00) \rightarrow (q, 11) \rightarrow s \) (loop: 10) and
- \((t, 01) \rightarrow (q, 11) \rightarrow s \) (loop: 10)

By applying Step 1 in Method 12.1 we get:

\[ T(r, 00 \circ 10^* \circ 11) = \{ s \} \quad \text{and} \quad T(t, 01 \circ 10^* \circ 11) = \{ s \} \]

Step 2 does not apply since \( r \) and \( t \) are different states.
Method 12.2
Converting a DFSA into a two-state DSA

Given a deterministic finite state automaton:
1. Add a new start state with a null transition to the original start state, and then, transform the automaton into an automaton with a single, new accept state.
2. While there is an internal state:
   1. Select an internal state
   2. Eliminate it by applying Method 12.1
   3. Update the transitions and language of the string automata
3. Return the resulting two-state string automaton
Let’s consider the deterministic finite state automaton
\[ M = (\{A, B, C, D\}, \{0, 1\}, T, A, \{D\}) \]
where \( T \) is defined in the table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{B}</td>
<td>{C}</td>
</tr>
<tr>
<td>B</td>
<td>{D}</td>
<td>{B}</td>
</tr>
<tr>
<td>C</td>
<td>{A}</td>
<td>{D}</td>
</tr>
<tr>
<td>D</td>
<td>{C}</td>
<td>{C}</td>
</tr>
</tbody>
</table>

We will apply Method 12.2 to transform it into a two-state string automaton, over strings that are regular expressions
Example (cont)

First we add a new start state $S$, and a new single accept state $H$. We express this in the transition table:

<table>
<thead>
<tr>
<th>T</th>
<th>0</th>
<th>1</th>
<th>$\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td></td>
<td></td>
<td>${A}$</td>
</tr>
<tr>
<td>$A$</td>
<td>${B}$</td>
<td>${C}$</td>
<td></td>
</tr>
<tr>
<td>$B$</td>
<td>${D}$</td>
<td>${B}$</td>
<td></td>
</tr>
<tr>
<td>$C$</td>
<td>${A}$</td>
<td>${D}$</td>
<td></td>
</tr>
<tr>
<td>$D$</td>
<td>${C}$</td>
<td>${B}$</td>
<td>${H}$</td>
</tr>
<tr>
<td>$H$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As a result, all the original states are now internal states.
Example (cont.)

We select first state C:

There are three paths through C:
(a) (A, 1) → (C, 0) → A (loop path)
(b) (A, 1) → (C, 1) → D (open path, no loop)
(c) (D, 0) → (C, 1) → D (loop path)

Transformations:
Step 1:  
(a) T(A, 1 □ 0) = \{A\}
(b) T(A, 1 □ 1) = \{D\}
(c) T(D, 0 □ 1) = \{D\}

Step 2:  Does not apply
Example (cont.)

Updated automaton:

$$M=\langle \{S, A, B, D, H\}, \{0, 1, 1 \circ 0, 1 \circ 1 + 0 \circ 1\}, T, S, \{H\}\rangle$$

where $T$ is defined by the table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1o0</th>
<th>1o1</th>
<th>0o1</th>
<th>λ</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>{A}</td>
</tr>
<tr>
<td>A</td>
<td>{B}</td>
<td></td>
<td>{A}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>{D}</td>
<td>{B}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>{D}</td>
<td></td>
<td>{D}</td>
<td></td>
<td>{H}</td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example (cont.)

Next, we select state B
There is one path through state B:

\[(A,0) \rightarrow (B,1) \rightarrow (B,0) \rightarrow D\) (internal loop in B)

The elimination of B yields the string transition:

Step 1:

\[T(A, 0 \circ 1^* \circ 0) = \{D\}\]

Step 2: This transformation together with \(T(A, 1 \circ 1) = \{D\}\), which was obtained after eliminating C; yield, in turn:

\[T(A, 0 \circ 1^* \circ 0 + 1 \circ 1) = \{D\}\]
Example (cont.)

Updated automaton:
\[ M=\left(\{S, A, D, H\}, \{1, 1\cdot0, 0\cdot1, 1\cdot1+0\cdot1^*\cdot0\}, T, S, \{H\}\right) \]
where \( T \) is defined by the table:

<table>
<thead>
<tr>
<th>( T )</th>
<th>1</th>
<th>1\cdot0</th>
<th>0\cdot1</th>
<th>1\cdot1+0\cdot1^*\cdot0</th>
<th>( \lambda )</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>{A}</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>{A}</td>
<td></td>
<td>{D}</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>{D}</td>
<td>{D}</td>
<td></td>
<td>{H}</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example (cont.)

Next, we select state $D$
There are two paths through state $D$:

$(A, 1 \circ 1 + 0 \circ 1^* \circ 0) \rightarrow (D, 1) \rightarrow (D, \lambda) \rightarrow H$ (internal loop in $D$)

$(A, 1 \circ 1 + 0 \circ 1^* \circ 0) \rightarrow (D, 0 \circ 1) \rightarrow (D, \lambda) \rightarrow H$ (internal loop in $D$)

Thus, the elimination of $D$ yields the string transition:

Step 1:

$T(A, (1 \circ 1 + 0 \circ 1^* \circ 0) \circ 1) = \{H\}$

$T(A, (1 \circ 1 + 0 \circ 1^* \circ 0) \circ 0 \circ 1) = \{H\}$

Step 2: After factorizations:

$T(A, (1 \circ 1 + 0 \circ 1^* \circ 0) \circ (1 + 0 \circ 1)) = \{D\}$
Example (cont.)

Updated automaton:
$M = (\{S, A, H\}, \{1 \cdot 0, (1 \cdot 1 + 0 \cdot 1^* \cdot 0) \cdot (1 + 0 \cdot 1)\}, T, S, \{H\})$

where $T$ is defined by the table:

<table>
<thead>
<tr>
<th>T</th>
<th>$1 \cdot 0$</th>
<th>$(1 \cdot 1 + 0 \cdot 1^* \cdot 0) \cdot (1 + 0 \cdot 1)$</th>
<th>$\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td>$\lambda$</td>
<td>${A}$</td>
</tr>
<tr>
<td>A</td>
<td>${A}$</td>
<td>$\lambda$</td>
<td>${H}$</td>
</tr>
<tr>
<td>H</td>
<td></td>
<td>$\lambda$</td>
<td></td>
</tr>
</tbody>
</table>
Example (cont.)

Finally, we are left with state A

There is only one path through state A:
\[(S, \lambda) \rightarrow (A, (1\cdot0)) \rightarrow (A, 1\cdot1 + 0\cdot1*0) \cdot (1 + 0\cdot1)) \rightarrow H \] (internal loop in A)

The elimination of A yields the string transition:
Step 1:
\[T(S, (1\cdot0)* \cdot (1\cdot1 + 0\cdot1*0) \cdot (1 + 0\cdot1)) = \{H\}\]

Step 2: Does not apply
Example (conclusion)

The **two-state string automaton** that is **equivalent** to the original **finite state automaton** is

\[ M = (\{S, H\}, \{(1 \cdot 0)^* \cdot (1 \cdot 1 + 0 \cdot 1^* \cdot 0) \cdot (1 + 0 \cdot 1)\}, T, S, \{H\}) \]

Where \( T(S, (1 \cdot 0)^* \cdot (1 \cdot 1 + 0 \cdot 1^* \cdot 0) \cdot (1 + 0 \cdot 1)) = \{H\} \)

**Thus,**

\[ L(M) = L((1 \cdot 0)^* \cdot (1 \cdot 1 + 0 \cdot 1^* \cdot 0) \cdot (1 + 0 \cdot 1)) \]
The limits of the finite state automata model

**Theorem 12.1**: Each finite state automaton is equivalent to a two-state string automaton over a regular expression.

**Proof**: If the automaton is nondeterministic, apply Theorem 11.1 and convert it to a deterministic automaton. Then, apply Method 12.2. If it is deterministic, just apply Method 12.2.
Conclusion

By combining Corollary 11.1 and Theorem 12.1 we conclude that:

**Corollary 12.1:** A language is regular if and only if there exists a finite state automaton that recognizes it