Topics: Finite Automata: Deterministic and Non
Approach: Work Exercises, Talk about Ideas
Main Ideas: DFA/NFA, multiple Representations

1. Admin
   No Recitation this week
   * quiz today, * outdoor lab Thursday

2. An opening question:
   a. Are there any English words that contain all five vowels in alphabetical order, with no repetitions?
   b. Can you design a function/machine to decide the set of all such strings?

3. The course so far:
   a. What is a Turing Machine?
   b. A Language is a set of strings
   c. Problems can be expressed as "is this string in this language?"
   d. Two big questions:
      - Can a Turing Machine answer a specific problem?
      - If it can, how long will it take?
      If we limit time, can we still solve it?

4. And now for something completely different
   a. What if we limit the machine instead of the time?
   b. Take away two things:
      - No more writing to the tape
      - No more backing up (i.e read a symbol once)
   c. Two big questions:
      - What languages can a finite automaton decide?
      - If it can, how long will it take?

5. Let’s get to work:
   a. We do 6 worksheets with discussion

Can decide some languages, can recognize many languages, can co-recognize many languages. Some problems take exponential time, some take polynomial time.

What languages can a FA decide? Are there languages it can recognize but not decide?
How much time does a FA take to accept/reject a string? How much space does it take?
Version 0: The English/Set Theory Description

\[ ST = \left\{ w \in \Sigma^* \mid w \text{ starts with ‘st’} \right\} \quad \text{where} \quad \Sigma = \{a, b, c, \ldots, z\} \]

Version 1: The State Diagram

Version 2: A Regular Expression

\[ \text{st} (a \cup b \cup \ldots \cup z)^* \]

or

\[ \text{st} \Sigma^* \quad \text{where} \quad \Sigma = \{a, b, c, \ldots, z\} \]

and some people write

\[ \text{st} (a|b|\ldots|z)^* \]

Version 3: The Formal Definition

\[ \Sigma \quad \text{(alphabet):} \quad a-z \]

States: \(q_0, q_1, q_2, q_3\)

Start State: \(q_0\)

Accept States: \(\{q_2\}\)

\[ \delta \quad \text{(Transitions)} \]

<table>
<thead>
<tr>
<th>In</th>
<th>Sees</th>
<th>GoesTo</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>s</td>
<td>(q_1)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>t</td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>[a-z]</td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_0)</td>
<td>[^s]</td>
<td>(q_3)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>[^t]</td>
<td>(q_3)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>[a-z]</td>
<td>(q_3)</td>
</tr>
</tbody>
</table>

Version 4: The C++ Code

```cpp
#include <iostream>
using namespace std;

/* SwST find all strings that start with "st" */

bool starts_with_st( string ); // decl

int main()
{
    string s;
    while( cin >> s ){  
        s=s+'' ;
        if ( starts_with_st( s ) )
            cout << s << endl;
    }
    return 0;
}

// starts_with_st --
// returns true if arg starts with "st"
enum states { START, SAW_S, ACC, REJ };

bool starts_with_st( string s )
{
    enum states state = START;
    int i = 0;
    while( s[i] != ' ' ){
        if ( state == START )
            if ( s[i] == 's' )
                state = SAW_S;
            else
                state = REJ;
        else if ( state == SAW_S )
            if ( s[i] == 't' )
                state = ACC;
            else
                state = REJ;
        else if ( state == ACC || state == REJ )
            state = ACC;
            else
                state = REJ;
        else if ( state == ACC || state == REJ )
            i++;  // next char does not matter
    }
    return ( state == ACC );
}
```

Definitions and Skills for Today

**DFA**
A Deterministic Finite Automaton is a 5-tuple of \((\Sigma, Q, \delta, q_0, A)\)

**Regular Language**
A language that is recognized by a DFA

**NFA**
A Nondeterministic Finite Automaton is a DFA with addition of 0 or more \(\varepsilon\) transitions and 0 or more non-deterministic transitions

**Translating**
You should be able to translate from any one of these five boxes to all of the other four.
Example 2: Translating a State Diagram into Other Notations

Version 0: The English/Set Theory Description (Call this regular language $L_2$)

Version 1: The State Diagram

Version 2: A Regular Expression

(\text{note: } [\text{\`set}] \text{ means complement of set})

Version 3: The Formal Definition

\begin{align*}
\Sigma \text{ (alphabet):} & \quad a-z \\
\text{States:} & \\
\text{Start State:} & \\
\text{Accept States:} & \\
\begin{array}{|c|c|c|}
\hline
\text{In} & \text{Sees} & \text{GoesTo} \\
\hline
\hline
\hline
\hline
\hline
\end{array}
\end{align*}

Version 4: Code (optional for today)

Use this space for notes and/or scratch work

Definitions and Skills for Today

\begin{itemize}
  \item DFA  
  A Deterministic Finite Automaton is a 5-tuple of $(\Sigma, Q, \delta, q_0, A)$
  \item Regular Language  
  A language that is recognized by a DFA
  \item NFA  
  A Nondeterministic Finite Automaton is a DFA with addition of 0 or more $\varepsilon$ transitions and 0 or more non-deterministic transitions
  \item Translating  
  You should be able to translate from any one of these five boxes to all of the other four.
\end{itemize}
Example 3: Understanding an RE, Translating to Other Forms

Version 0: The English/Set Theory Description

Version 1: The State Diagram

Version 2: A Regular Expression

\[ st(a \cup b \cup \ldots \cup z)^*er \]

or

\[ st\Sigma^*er \quad \text{where } \Sigma = \{a, b, c, \ldots, z\} \]

and some people write

\[ st(a|b|\ldots|z)^*er \]

Version 3: The Formal Definition

\[ \Sigma (\text{alphabet}): \ a-z \]

States:
Start State:
Accept States:

<table>
<thead>
<tr>
<th>δ (Transitions)</th>
<th>In</th>
<th>Sees</th>
<th>GoesTo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Version 4: Code (optional for today)

Use this space for notes and/or scratch work

Question: Can you use your diagrams/RE/Table from Examples 1 and 2 to answer this problem?

Combining Regular Languages

Union: \( \cup \)
The Union of two Regular Languages is a Regular Language. Proof by definition and construction.

Concatenation: \( \circ \)
\( A \circ B \) is defined to be the set of all strings \( ab \) where \( a \in A \) and \( b \in B \). If \( A \) and \( B \) are regular languages, then \( A \circ B \) is a regular language. Proof by definition and construction.

Star: \( * \)
\( A^* \) is defined to be the set of all strings \( a_1a_2\ldots a_n \) where \( a_i \in A \). If \( A \) is a regular language, the \( A^* \) is a regular language. Proof by construction.
Example 4: Reading a Transition Table, Translate to Other Forms; Meet an NFA

Version 0: The English/Set Theory Description

Version 1: The State Diagram

Version 2: A Regular Expression

or

and some people write

Version 3: The Formal Definition

<table>
<thead>
<tr>
<th>( \Sigma ) (alphabet):</th>
<th>a-z</th>
</tr>
</thead>
<tbody>
<tr>
<td>States:</td>
<td>( q_0, q_1, q_2, q_3 )</td>
</tr>
<tr>
<td>Start State:</td>
<td>( q_0 )</td>
</tr>
<tr>
<td>Accept States:</td>
<td>{ ( q_3 ) }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \delta ) (Transitions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>( q_0 )</td>
</tr>
<tr>
<td>( q_0 )</td>
</tr>
<tr>
<td>( q_1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
</tr>
<tr>
<td>( q_3 )</td>
</tr>
</tbody>
</table>

Non Determinism

Multiple Transitions
A NFA (non-deterministic finite automaton) is a finite automaton that contains multiple transitions that go from the same state on the same input.

Epsilon Transitions: \( \varepsilon \)
An epsilon transition (\( \varepsilon \)) is a transition that does not consume any input. The machine can move from one state to another ‘for free’.

NFA/DFA Equivalence
Any NFA can be converted into a DFA if you create a bunch more states and more transitions. Therefore, any language that can be decided by a NFA can be decided by a DFA.

Version 4: Code (optional for today)
Use this space for notes and/or scratch work
Example 5: NFA with Epsilon Transition

Version 0: The English/Set Theory Description

Version 1: The State Diagram

Version 2: A Regular Expression

```
\text{or}
\text{and some people write}
```

Version 3: The Formal Definition

<table>
<thead>
<tr>
<th>( \Sigma ) (alphabet):</th>
<th>a-z</th>
</tr>
</thead>
<tbody>
<tr>
<td>States:</td>
<td></td>
</tr>
<tr>
<td>Start State:</td>
<td></td>
</tr>
<tr>
<td>Accept States:</td>
<td></td>
</tr>
</tbody>
</table>

\[
\delta \text{ (Transitions)}
\]

<table>
<thead>
<tr>
<th>In</th>
<th>Sees</th>
<th>GoesTo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Explorations/Experiments

☞: Draw a tree showing branching to indicate multiple paths. Trace paths for some strings in the language, and some not in the language.

☞: Redraw the diagram as a DFA by removing the \( \epsilon \) transitions.

Multiple Paths in an NFA

- **Multiple Paths**: Consider an input string of “a” for this machine. What path or paths does the machine follow? Do any accept? Do any reject?

- **One Accept is Enough**: An NFA accepts a string if any path accepts.

- **Epsilon Transitions**: An epsilon transition uses no input symbols. The machine reads no input; it just jumps to the next state. If there are multiple epsilon transitions from one state, the machine splits into multiple machines and runs the paths in parallel. See the Calvin and Hobbes duplicator handout.
Example 6: A More Complicated Example with Many Solutions

Version 0: The English/Set Theory Description

\[ ST = \left\{ w \in \Sigma^n \mid \text{w starts with 'st' or 'er', contains exactly one 'c', ends with 'er'} \right\} \]

where \( \Sigma = \{a, b, c, \ldots, z\} \)

Version 1: The State Diagram

Version 2: A Regular Expression

\[ \text{or} \]

\[ \text{and some people write} \]

Version 3: The Formal Definition

\[ \Sigma \text{ (alphabet): a-z} \]

States:
Start State:
Accept States:

\[ \delta \text{ (Transitions)} \]

<table>
<thead>
<tr>
<th>In</th>
<th>Sees</th>
<th>GoesTo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Possible/Feasible

**Every Regular Language is Decidable**
How can you show the machine will never loop?

**Running Time is Always Polynomial**
How long does a DFA take to accept or reject a string?

**Running Space is Always Polynomial**
How much tape storage does a DFA require to accept or reject a string?

Version 4: Code (optional for today)
Use this space for notes and/or scratch work
Or outline proofs of the claims in the box at left.