NFA and regex

- the Boolean algebra of languages
- regular expressions
KISS – DFA

Deterministic Finite Automaton

Exactly one start state, and from each state, \( q \), for each token, \( t \), there is exactly one transition from \( s \) with label \( t \).
Two examples

Input sequence is accepted if it ends with a zero.

Input sequence is accepted if it ends with a one.
The complement of a regular language is regular

L₀: even numbers
   = 0 mod 2

L₁: odd numbers
   = 1 mod 2
Three examples

Which binary numbers are accepted?

<table>
<thead>
<tr>
<th>mod 3</th>
<th>x2</th>
<th>x2 + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
The complement of a regular language is regular

If \( A \subseteq \Sigma^* \) is recognised by \( M \) then \( \overline{A} = \Sigma^* \setminus A \) is recognised by \( \overline{M} \)

where \( \overline{M} \) and \( M \) are identical except that the accepting states of \( \overline{M} \) are the non-accepting states of \( M \) and vice-versa.
By three or not by three?

divisible by three

not divisible by three
The intersection of two regular languages is regular

\[ L_0 = 0 \mod 3 \]
\[ L_1 = 1 \mod 3 \]
\[ L_2 = 2 \mod 3 \]
The intersection of two regular languages is regular

\[ \equiv \]

\begin{align*}
\text{divisible by 6} \\
\text{divisible by 2} \\
\text{and} \\
\text{divisible by 3}
\end{align*}
The intersection of two regular languages is regular

Run both machines in parallel?

Build one machine that simulates two machines running in parallel!

Keep track of the state of each machine.
The intersection of two regular languages is regular
intersection of languages

run the two machines in parallel
when a string is in both languages,
both are in an accepting state
intersection of languages

run the two machines in parallel
when a string is in both languages,
both are in an accepting state
intersection of two regular languages is regular

run two machines in synchrony
The regular languages $A \subseteq \Sigma^*$ form a Boolean Algebra

- Since they are closed under intersection and complement.
Are all languages $A \subseteq \Sigma^*$ regular?

• A finite machine can only do so much or rather, so little.

Consider the language with two symbols $\Sigma = \{0, 1\}$, consisting of the all strings that contain equal numbers of zeros and ones.

How can we show that this language is not regular?
Are all languages \( A \subseteq \Sigma^* \) regular?

Consider the language with two symbols \( \Sigma = \{0,1\} \), consisting of the all strings that contain equal numbers of zeros and ones.

Suppose we have a DFA that recognises this language.

Let \( s_n \) be the state the machine reaches after an input of \( n \) zeros.

If the machine is in state \( s_n \) and we give it an input of \( n \) ones it will be in an accepting state.
Are all languages $A \subseteq \Sigma^*$ regular?

$A =$ strings that contain equal numbers of zeros and ones.

We have a DFA that recognises this language.

$s_n$ is the state the machine after an input of $n$ zeros.

If the machine is in state $s_n$ and we give it an input of $n$ ones it will be in an accepting state.

If $n \neq m$ what can we say about $s_n$ and $s_m$?
Are all languages $A \subseteq \Sigma^*$ regular?

$A =$ strings that contain equal numbers of zeros and ones.
We have a DFA that recognises this language.

$s_n$ is the state the machine after an input of $n$ zeros.

If the machine is in state $s_n$ (after an input of $n$ zeros),
and we give it an input of $m$ ones
it will be in an accepting state iff $m = n$.

Therefore, if $n \neq m$ then $s_n \neq s_m$ (why?)

So our machine must have infinitely many states!!

An FSM with $n$ states cannot count beyond $n-1$. 
Which languages $A \subseteq \Sigma^*$ are regular?

What kind of answer can we give to a question like this?

If we have a machine
then the language it recognises is regular.

For some languages, $L$, e.g. our $#0s = #1s$ example,
we can argue that no FSM can recognise $L$.
But that is not a general argument.

Instead of finding a property,
to characterise the regular languages,
we will find a set of rules that generate
the set of regular languages.
Which languages $A \subseteq \Sigma^*$ are regular?

We give a set of rules, and show that they generate all regular languages.

First we work with general NFA to show that the rules are **sound** – any language generated by the rules is regular.

Then we show that for any NFA, $M$, there is a DFA, $\mathcal{P}(M)$, that recognises the same language. So, any language generated by the rules is recognised by some DFA.

Then we show that the rules are **complete** – any regular language is generated by the rules.
A natural language is a set of finite sequences of words.

A formal language is a set of finite sequences of symbols.

A formal language is regular iff it is the language recognised by some Finite State Machine.
KISS – start simple

NFA any number of start states and accepting states

Any NFA with no accepting states, e.g. the NFA with no states, recognises the empty language $\emptyset = \{}$.

The NFA with one state - starting and accepting, and no transitions $\xrightarrow{}$ recognises the empty string $\{\varepsilon\}$.

The NFA with two states and one transition, start to stop $\xrightarrow{a}$ recognises $\{"a"\}$.
The following languages are regular

• The empty language $\emptyset = \{\}$

• The language that includes only the empty string $\{\varepsilon\}$.

• The language that includes only the one-letter string $\{"a"\}$
NFA any number of start states and accepting states
sequence
RS
alternation $R|S$
iteration \( R^* \)
rules for regular languages

The following languages are regular

• The empty language $\emptyset = \{\}$
• The language that includes only the empty string $\{\varepsilon\}$.
• The language that includes only the one-letter string $\{"a"\}$

If $R$ and $S$ are regular so are

• $R \mid S = R \cup S$
• $RS = \{rs \mid r \in R \text{ and } s \in S\}$
• $R^*$ generated by rules
  • $\varepsilon \in R^*$
  • if $x \in R^*$ and $r \in R$ then $xr \in R^*$
patterns for regular languages

regular expressions for sets of strings

• "" empty language $\emptyset = \{\}$
• $\varepsilon$ empty string $\{\varepsilon\}$
• $a$ one-letter string $\{"a"\}$
• $R \mid S$ union $R \cup S$
• $RS$ concatenation $\{ rs \mid r \in R \text{ and } s \in S \}$
• $R^*$ iteration

precedence: $R \mid ST^* = R \mid (S(T^*))$
A Decimal Number

\[ (+\mid-)(\d|\d)\d^*(\varepsilon|\d^*\d) \]

where \( \d \) is \( 0|1|2|3|4|5|6|7|8|9 \)
\[(a|b)^*a\]

\[a(a|b)^*\]

\[(a|b)^*a(bc)^*\]

\[(a|b)^*ab(cb)^*\]

\[a(da|bc)^*\]
The following equations hold for any sets of strings $R, S, T$

- $\emptyset | S = \emptyset$
- $\emptyset S = \emptyset$
- $\varepsilon S = \emptyset$
- $\varepsilon^* = \emptyset$
- $\{\}^* = \emptyset$
- $R (S | T) = R S | R T$
- $S R | T R = S R | S T$
- $S^* S | \varepsilon = S^* S$
rules for regular languages

The following equations hold for any sets of strings $R, S, T$

- $\emptyset | S = \emptyset | S = S$
- $\emptyset S = S \emptyset = \emptyset$
- $\epsilon S = S \epsilon = S$
- $\epsilon^* = \epsilon$
- $\emptyset^* = \emptyset$
- $R (S | T) = R S | R T$
- $(S | T) R = S R | T R$
- $S^* = S^* S | \epsilon = S S^* | \epsilon$