Lecture 25: Non-regularity and limits of FSMs

- **Highlights**
  - NFAs from Regular Expressions
    - \((01 \cup 1)^*0\)
  - "Subset construction": NFA to DFA

- **Diagrams**
  - NFA and DFA examples for subsets and regular expressions.
DFAs ≡ regular expressions

We have shown how to build an optimal DFA for every regular expression

- Build NFA
- Convert NFA to DFA using subset construction
- Minimize resulting DFA

Theorem: A language is recognized by a DFA if and only if it has a regular expression

Generalized NFAs

- Like NFAs but allow
  - Parallel edges
  - Regular Expressions as edge labels
    - NFAs already have edges labeled λ or a
- An edge labeled by A can be followed by reading a string of input chars that is in the language represented by A
- A string x is accepted iff there is a path from start to final state labeled by a regular expression whose language contains x

Starting from an NFA

Add new start state and final state

Then eliminate original states one by one, keeping the same language, until it looks like:

Final regular expression will be A
only two simplification rules

- **Rule 1**: For any two states \(q_1\) and \(q_2\) with parallel edges (possibly \(q_1 = q_2\)), replace

\[
\begin{array}{c}
q_1 \quad A \quad q_2
\end{array}
\]

by

\[
\begin{array}{c}
q_1 \quad \text{AUB} \quad q_2
\end{array}
\]

- **Rule 2**: Eliminate non-start/final state \(q_3\) by replacing all

\[
\begin{array}{c}
q_1 \quad A \quad B \quad C \quad q_2
\end{array}
\]

by

\[
\begin{array}{c}
q_1 \quad \text{AB}^*\text{C} \quad q_2
\end{array}
\]

for every pair of states \(q_1, q_2\) (even if \(q_1 = q_2\))

---

**converting an NFA to a regular expression**

Consider the DFA for the mod 3 sum

- Accept strings from \(\{0,1,2\}^*\) where the digits mod 3 sum of the digits is 0

\[
\begin{array}{c}
t_0 \quad \lambda \quad t_1 \quad \lambda \quad t_2
\end{array}
\]

---

**splicing out a node**

Label edges with regular expressions

\[
\begin{align*}
t_0 \rightarrow t_1 \rightarrow t_0 & : 10^*2 \\
t_0 \rightarrow t_1 \rightarrow t_2 & : 10^*1 \\
t_2 \rightarrow t_1 \rightarrow t_0 & : 20^*2 \\
t_2 \rightarrow t_1 \rightarrow t_2 & : 20^*1
\end{align*}
\]

---

**finite automaton without \(t_1\)**

Final regular expression:

\[ (0 \cup 10^*2 \cup (2 \cup 10^*1)(0 \cup 20^*1)^*(1 \cup 20^*2))^* \]
what can finite state machines do?

- We’ve seen how we can get DFAs to recognize all regular languages

- What about some other languages we can generate with CFGs?
  - \{0^n 1^n : n \geq 0 \}?
  - binary palindromes?
  - strings of balanced parentheses?

A=\{0^n 1^n : n \geq 0 \} cannot be recognized by any DFA

Consider the infinite set of strings
\[ S=\{\lambda, 0, 00, 000, 0000, \ldots \} \]

Claim: No two strings in \( S \) can end at the same state of any DFA for \( A \)

Proof:
Suppose \( n \neq m \) and \( 0^n \) and \( 0^m \) end at the same state \( p \) of some DFA for \( A \).

Since \( 0^n 1^n \) is in \( A \), following \( 1^n \) after state \( p \) must lead to a final state.

But then the DFA would also accept \( 0^m 1^n \)
which is a contradiction to the DFA recognizing \( A \).

Given claim, the # of states of any DFA for \( A \) must be \( \geq |S| \)
which is not finite, which is impossible for a DFA.

B = \{binary palindromes\} can’t be recognized by any DFA

Consider the infinite set of strings
\[ S=\{\lambda, 0, 00, 000, 0000, \ldots \}=\{0^n : n \geq 0 \} \]

Claim: No two strings in \( S \) can end at the same state of any DFA for \( B \)

Proof:
Suppose \( n \neq m \) and \( 0^n \) and \( 0^m \) end at the same state \( p \) of some DFA for \( B \).

Since \( 0^n 1^n \) is in \( B \), following \( 1^n \) after state \( p \) must lead to a final state.

But then the DFA would also accept \( 0^m 1^n \)
which is a contradiction since the DFA recognizes \( B \).

Given claim, the # of states of any DFA for \( B \) must be \( \geq |S| \)
which is not finite, which is impossible for a DFA.

general: how to show language \( L \) has no DFA

- Find a “hard” infinite set \( S=\{s_0, s_1, \ldots, s_n, \ldots\} \) of strings that might be prefixes of strings in \( L \)

- Show that \( S \) is hard by showing that no two strings \( s_n \neq s_m \) in \( S \) can end at the same state of any DFA recognizing \( L \)
  - For each pair \( s_n \neq s_m \) find an extender string \( t \)
    depending on \( n, m \) so that exactly one of \( s_n t \) and \( s_m t \) is in \( L \)

- Conclude that any DFA for \( L \) would need \( \geq |S| \) states which is not finite, and so impossible
P = \{strings of balanced parentheses\}

pattern matching

\begin{itemize}
  \item Given
    \begin{itemize}
      \item a string, \textbf{s}, of \textit{n} characters
      \item a pattern, \textbf{p}, of \textit{m} characters
      \item usually \textit{m} \textless \textless \textit{n}
    \end{itemize}
  \item Find
    \begin{itemize}
      \item all occurrences of the pattern \textbf{p} in the string \textbf{s}
    \end{itemize}
  \item Obvious algorithm:
    \begin{itemize}
      \item try to see if \textbf{p} matches at each of the positions in \textbf{s}
      \item stop at a failed match and try the next position
    \end{itemize}
\end{itemize}

\begin{align*}
  \text{string } \textbf{s} &= x \ y \ x \ x \ y \ y \ x \ y \ x \ x \ y \ x \ y \ x \ y \ x \ x \\
  \text{pattern } \textbf{p} &= x \ y \ x \ y \ x \ y \ x \ y \ x \ x
\end{align*}

\begin{align*}
  \text{string } \textbf{s} &= x \ y \ x \ x \ y \ x \ x \ y \ y \ x \ x \ y \ x \ y \ x \ y \ x \ x \\
  \text{string } \textbf{s} &= x \ y \ x \ x \ y \ x \ x \ y \ y \ x \ x \ y \ x \ y \ x \ y \ x \ x \\
  \text{string } \textbf{s} &= x \ y \ x \ x \ y \ x \ x \ y \ y \ x \ x \ y \ x \ y \ x \ y \ x \ x
\end{align*}
string \( s = x \ y \ x \ x \ y \ x \ y \ x \ y \ x \ y \ x \ y \ x \ x \ x \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)

string \( s = x \ y \ x \ x \ y \ x \ y \ x \ y \ x \ y \ x \ y \ x \ y \ x \ x \ x \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)

string \( s = x \ y \ x \ x \ y \ x \ y \ x \ y \ x \ y \ x \ y \ x \ x \ x \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)

string \( s = x \ y \ x \ x \ y \ x \ y \ x \ y \ x \ y \ x \ y \ x \ x \ x \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)
   \( x \ y \ x \ y \)
string $s = \text{x y x x y x y y x y y x y y x y x y x x}$

String $s = \text{x y x x y x y y x x y x y y x x}$

string $s = \text{x y x x y x y x y x x y x y y x y y y}$

string $s = \text{x y y x y x x y y}$
Worst-case time $O(mn)$
better pattern matching via finite automata

• Build a DFA for the pattern (preprocessing) of size $O(m)$
  – Keep track of the ‘longest match currently active’
  – The DFA will have only $m+1$ states

• Run the DFA on the string $n$ steps

• Obvious construction method for DFA will be $O(m^2)$
  but can be done in $O(m)$ time.

• Total $O(m+n)$ time
preprocessing the pattern

pattern \( p = x \ y \ x \ y \ y \ x \ y \ x \ y \ x \ x \)

preprocessing the pattern

pattern \( p = x \ y \ x \ y \ y \ x \ y \ x \ y \ x \ x \)

generalizing

- Can search for arbitrary combinations of patterns
  - Not just a single pattern
  - Build NFA for pattern then convert to DFA 'on the fly'.

  Compare DFA constructed above with subset construction for the obvious NFA.