Closure Properties of Regular Languages

Union, Intersection, Difference, Concatenation, Kleene Closure, Reversal, Homomorphism, Inverse Homomorphism
Closure Properties

◆ Recall a closure property is a statement that a certain operation on languages, when applied to languages in a class (e.g., the regular languages), produces a result that is also in that class.

◆ For regular languages, we can use any of its representations to prove a closure property.
Closure Under Union

◆ If L and M are regular languages, so is $L \cup M$.

◆ **Proof**: Let L and M be the languages of regular expressions R and S, respectively.

◆ Then $R + S$ is a regular expression whose language is $L \cup M$. 
Closure Under Concatenation and Kleene Closure

◆ Same idea:
  ▶ RS is a regular expression whose language is LM.
  ▶ R* is a regular expression whose language is L*.
Closure Under Intersection

- If L and M are regular languages, then so is $L \cap M$.

- **Proof:** Let A and B be DFA's whose languages are L and M, respectively.

- Construct C, the product automaton of A and B.

- Make the final states of C be the pairs consisting of final states of both A and B.
Example: Product DFA for Intersection

\[
\begin{array}{c}
\text{A} \\
\rightarrow 0, 1 \\
\text{B} \\
\text{C} \\
\rightarrow 1, 0 \\
\text{D} \\
\end{array}
\]
Closure Under Difference

◆ If L and M are regular languages, then so is \( L - M \) = strings in L but not M.
◆ **Proof**: Let A and B be DFA’s whose languages are L and M, respectively.
◆ Construct C, the product automaton of A and B.
◆ Make the final states of C be the pairs where A-state is final but B-state is not.
Example: Product DFA for Difference

Notice: difference is the empty language
Closure Under Complementation

◆ The \textit{complement} of a language \( L \) (with respect to an alphabet \( \Sigma \) such that \( \Sigma^* \) contains \( L \)) is \( \Sigma^* - L \).

◆ Since \( \Sigma^* \) is surely regular, the complement of a regular language is always regular.
Decision Properties of Regular Languages

General Discussion of “Properties”
The Pumping Lemma
Membership, Emptiness, Etc.
Decision Properties

◆ A *decision property* for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.

◆ Example: Is language $L$ empty?
Example: Use of Closure Property

- We can easily prove $L_1 = \{0^n1^n \mid n \geq 0\}$ is not a regular language.
- $L_2 =$ the set of strings with an equal number of 0’s and 1’s isn’t either, but that fact is trickier to prove.
- Regular languages are closed under $\cap$.
- If $L_2$ were regular, then $L_2 \cap L(0^*1^*) = L_1$ would be, but it isn’t.
The Membership Question

- Our first decision property is the question: “is string w in regular language L?”
- Assume L is represented by a DFA A.
- Simulate the action of A on the sequence of input symbols forming w.
The Emptiness Problem

- Given a regular language, does the language contain any string at all.
- Assume representation is DFA.
- Construct the transition graph.
- Compute the set of states reachable from the start state.
- If any final state is reachable, then yes, else no.
The Infiniteness Problem

◆ Is a given regular language infinite?
◆ Start with a DFA for the language.
◆ **Key idea:** if the DFA has $n$ states, and the language contains any string of length $n$ or more, then the language is infinite.
◆ Otherwise, the language is surely finite.
  ◆ Limited to strings of length $n$ or less.
Proof of Key Idea

If an n-state DFA accepts a string $w$ of length $n$ or more, then there must be a state that appears twice on the path labeled $w$ from the start state to a final state.

Because there are at least $n+1$ states along the path.
Proof – (2)

Then $xy^i z$ is in the language for all $i \geq 0$.

Since $y$ is not $\epsilon$, we see an infinite number of strings in $L$. 
Infiniteness – Continued

◆ We do not yet have an algorithm.
◆ There are an infinite number of strings of length > n, and we can’t test them all.
◆ **Second key idea**: if there is a string of length \( \geq n \) (= number of states) in L, then there is a string of length between n and 2n-1.
Proof of 2\textsuperscript{nd} Key Idea

- Remember:
  - We can choose $y$ to be the first cycle on the path.
  - So $|xy| \leq n$; in particular, $1 \leq |y| \leq n$.
  - Thus, if $w$ is of length $2n$ or more, there is a shorter string in $L$ that is still of length at least $n$.
  - Keep shortening to reach $[n, 2n-1]$.
Completion of Infiniteness Algorithm

◆ Test for membership all strings of length between $n$ and $2n-1$.
  ♦ If any are accepted, then infinite, else finite.

◆ A terrible algorithm.

◆ **Better**: find cycles between the start state and a final state.
Finding Cycles

1. Eliminate states not reachable from the start state.
2. Eliminate states that do not reach a final state.
3. Test if the remaining transition graph has any cycles.
The Pumping Lemma

- We have, almost accidentally, proved a statement that is quite useful for showing certain languages are not regular.
- Called the *pumping lemma for regular languages.*
Statement of the Pumping Lemma

For every regular language \( L \)

There is an integer \( n \), such that

For every string \( w \) in \( L \) of length \( \geq n \)

We can write \( w = xyz \) such that:

1. \( |xy| \leq n \).
2. \( |y| > 0 \).
3. For all \( i \geq 0 \), \( xy^iz \) is in \( L \).
Example: Use of Pumping Lemma

- We have claimed \( \{0^k1^k \mid k \geq 1 \} \) is not a regular language.
- Suppose it were. Then there would be an associated \( n \) for the pumping lemma.
- Let \( w = 0^n1^n \). We can write \( w = xyz \), where \( x \) and \( y \) consist of 0’s, and \( y \neq \varepsilon \).
- But then \( xyyz \) would be in \( L \), and this string has more 0’s than 1’s.
Decision Property: Equivalence

- Given regular languages \( L \) and \( M \), is \( L = M \)?
- Algorithm involves constructing the product DFA from DFA’s for \( L \) and \( M \).
- Let these DFA’s have sets of states \( Q \) and \( R \), respectively.
- Product DFA has set of states \( Q \times R \).
  - I.e., pairs \([q, r]\) with \( q \) in \( Q \), \( r \) in \( R \).
Product DFA – Continued

- Start state = $[q_0, r_0]$ (the start states of the DFA’s for $L$, $M$).
- **Transitions**: $\delta([q,r], a) = [\delta_L(q,a), \delta_M(r,a)]$
  - $\delta_L$, $\delta_M$ are the transition functions for the DFA’s of $L$, $M$.
  - That is, we simulate the two DFA’s in the two state components of the product DFA.
Example: Product DFA
Equivalence Algorithm

- Make the final states of the product DFA be those states \([q, r]\) such that exactly one of \(q\) and \(r\) is a final state of its own DFA.
- Thus, the product accepts \(w\) iff \(w\) is in exactly one of \(L\) and \(M\).
Example: Equivalence
Equivalence Algorithm – (2)

- The product DFA’s language is empty iff \( L = M \).
- But we already have an algorithm to test whether the language of a DFA is empty.
Decision Property: Containment

◆ Given regular languages L and M, is \( L \subseteq M \)?
◆ Algorithm also uses the product automaton.
◆ How do you define the final states \([q, r]\) of the product so its language is empty iff \( L \subseteq M \)?

Answer: q is final; r is not.
The Minimum-State DFA for a Regular Language

- In principle, since we can test for equivalence of DFA’s we can, given a DFA $A$, find the DFA with the fewest states accepting $L(A)$.
- Test all smaller DFA’s for equivalence with $A$.
- But that’s a terrible algorithm.
Efficient State Minimization

◆ Construct a table with all pairs of states.
◆ If you find a string that *distinguishes* two states (takes exactly one to an accepting state), mark that pair.
◆ Algorithm is a recursion on the length of the shortest distinguishing string.
State Minimization – (2)

◆ **Basis:** Mark a pair if exactly one is a final state.

◆ **Induction:** mark \([q, r]\) if there is some input symbol \(a\) such that \([\delta(q,a), \delta(r,a)]\) is marked.

◆ After no more marks are possible, the unmarked pairs are equivalent and can be merged into one state.
Constructing the Minimum-State DFA

◆ Suppose $q_1, \ldots, q_k$ are indistinguishable states.
◆ Replace them by one state $q$.
◆ Then $\delta(q_1, a), \ldots, \delta(q_k, a)$ are all indistinguishable states.
  ◆ Key point: otherwise, we should have marked at least one more pair.
◆ Let $\delta(q, a) = \text{the representative state for that group.}$
**Example: State Minimization**

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\rightarrow)</td>
<td>{1}</td>
<td>{2,4}</td>
</tr>
<tr>
<td></td>
<td>{2,4}</td>
<td>{2,4,6,8}</td>
</tr>
<tr>
<td></td>
<td>{5}</td>
<td>{2,4,6,8}</td>
</tr>
<tr>
<td></td>
<td>{2,4,6,8}</td>
<td>{2,4,6,8}</td>
</tr>
<tr>
<td></td>
<td>{1,3,5,7}</td>
<td>{2,4,6,8}</td>
</tr>
<tr>
<td>*</td>
<td>{1,3,7,9}</td>
<td>{2,4,6,8}</td>
</tr>
<tr>
<td>*</td>
<td>{1,3,5,7,9}</td>
<td>{2,4,6,8}</td>
</tr>
</tbody>
</table>

Here it is with more convenient state names:

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\rightarrow)</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>*</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>*</td>
<td>G</td>
<td>D</td>
</tr>
</tbody>
</table>

Remember this DFA? It was constructed for the chessboard NFA by the subset construction.
Example – Continued

<table>
<thead>
<tr>
<th>r</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>*</td>
<td>F</td>
</tr>
<tr>
<td>*</td>
<td>G</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccccccc}
G & F & E & D & C & B \\
A & xx & & & & \\
B & xx & & & & \\
C & xx & & & & \\
D & xx & & & & \\
E & xx & & & & \\
F & & & & & \\
\end{array}
\]

Start with marks for the pairs with one of the final states F or G.
Example – Continued

Input r gives no help, because the pair [B, D] is not marked.
But input b distinguishes \{A, B, F\} from \{C, D, E, G\}. For example, \[A, C\] gets marked because \[C, F\] is marked.
Example – Continued

<table>
<thead>
<tr>
<th>r</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>*</td>
<td>F</td>
</tr>
<tr>
<td>*</td>
<td>G</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>F</th>
<th>E</th>
<th>D</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>B</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>C</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

[C, D] and [C, E] are marked because of transitions on b to marked pair [F, G].
Example – Continued

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>*</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>*</td>
<td>G</td>
<td>D</td>
</tr>
</tbody>
</table>

[A, B] is marked because of transitions on r to marked pair [B, D].

[D, E] can never be marked, because on both inputs they go to the same state.
Example – Concluded

Replace D and E by H.
Result is the minimum-state DFA.
Eliminating Unreachable States

Unfortunately, combining indistinguishable states could leave us with unreachable states in the “minimum-state” DFA.

Thus, before or after, remove states that are not reachable from the start state.
Clincher

- We have combined states of the given DFA wherever possible.
- Could there be another, completely unrelated DFA with fewer states?
- No. The proof involves minimizing the DFA we derived with the hypothetical better DFA.