Closure properties of regular languages

Proposition:
Let $\Sigma$ be some finite alphabet.
If $L \subseteq \Sigma^*$ is regular, then so is $\overline{L} = \Sigma^* \setminus L$.

Proof:
**Theorem:**
Let $\Sigma$ be some finite alphabet. If $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$ are regular, then so is $L_1 \cup L_2$.

**Proof:**

---

**The mindset**

---

**Step 1:** Imagining ourselves as a DFA
**Example**

$L_1 =$ strings with even number of 1’s

$L_2 =$ strings with length divisible by 3.

Closed under union

**Input:** 101001

Accept
Main idea:
Construct a DFA that keeps track of both at once.

Step 2: Formally defining the DFA
Closed under union

**Proof:** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA deciding $L_1$ and $M' = (Q', \Sigma, \delta', q'_0, F')$ be a DFA deciding $L_2$. We construct a DFA $M'' = (Q'', \Sigma, \delta'', q''_0, F'')$ that decides $L_1 \cup L_2$, as follows:

More closure properties

**Closed under union:**

**Closed under concatenation:**

**Closed under star:**

super awesome vs regular

What is the relationship between super awesome and regular?
**Theorem:**
Can define regular languages recursively as follows:

**Closed under concatenation**

**Theorem:**
Let $\Sigma$ be some finite alphabet. If $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$ are regular, then so is $L_1 L_2$.

**The mindset**

*Imagine yourself as a DFA.*

**Rules:**
1) Can only scan the input once, from left to right.
2) Can only remember “constant” amount of information.
   
   should not change based on input length
**Step 1:** Imagining ourselves as a DFA

Given \( w \in \Sigma^* \), we need to decide if
\[
\text{for } u \in L_1, v \in L_2.
\]

**Problem:** Don’t know where \( u \) ends, \( v \) begins.
When do you stop simulating \( M_1 \) and start simulating \( M_2 \) ?

Suppose God tells you \( u \) ends at \( w_3 \).

<table>
<thead>
<tr>
<th>( w )</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( w_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( w_3 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( w_4 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( w_5 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( w_6 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( w_7 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( w_8 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( w_9 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( w_{10} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**thread:**
**Step 2**: Formally defining the DFA

\[
M_1 = (Q, \Sigma, \delta, q_0, F) \quad M_2 = (Q', \Sigma, \delta', q'_0, F')
\]

\[
\begin{align*}
Q'' &= \\
\delta'' : \\
q''_0 &= \\
F'' &=
\end{align*}
\]