I think you should be more explicit here in step two.
Enter 15 cents in dimes or nickels
Press S or B for a candy bar
Basic transitions on N (nickel), D (dime), B (butterfinger), S (snickers)
Adding output to states: N – Nickel, S – Snickers, B – Butterfinger
Adding additional “unexpected” transitions to cover all symbols for each state
State Minimization

- Many different FSMs (DFAs) for the same problem
- Take a given FSM and try to reduce its state set by combining states
  - Algorithm will always produce the unique minimal equivalent machine (up to renaming of states) but we won’t prove this
State Minimization Algorithm

1. Put states into groups based on their outputs (or whether they are final states or not)
2. Repeat the following until no change happens
   a. If there is a symbol \( s \) so that not all states in a group \( G \) agree on which group \( s \) leads to, split \( G \) into smaller groups based on which group the states go to on \( s \)
3. Finally, convert groups to states
Put states into groups based on their outputs (or whether they are final states or not)
Put states into groups based on their outputs (or whether they are final states or not)
State Minimization Example

<table>
<thead>
<tr>
<th>present state</th>
<th>next state 0</th>
<th>next state 1</th>
<th>next state 2</th>
<th>next state 3</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>S0</td>
<td>S3</td>
<td>S1</td>
<td>S5</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S1</td>
<td>S3</td>
<td>S2</td>
<td>S4</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>S1</td>
<td>S0</td>
<td>S4</td>
<td>S5</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S5</td>
<td>1</td>
</tr>
<tr>
<td>S5</td>
<td>S1</td>
<td>S4</td>
<td>S0</td>
<td>S5</td>
<td>0</td>
</tr>
</tbody>
</table>

Put states into groups based on their outputs (or whether they are final states or not)

If there is a symbol s so that not all states in a group G agree on which group s leads to, split G based on which group the states go to on s
State Minimization Example

Put states into groups based on their outputs (or whether they are final states or not)

If there is a symbol $s$ so that not all states in a group $G$ agree on which group $s$ leads to, split $G$ based on which group the states go to on $s$
State Minimization Example

Put states into groups based on their outputs (or whether they are final states or not).

If there is a symbol \( s \) so that not all states in a group \( G \) agree on which group \( s \) leads to, split \( G \) based on which group the states go to on \( s \).
State Minimization Example

Put states into groups based on their outputs (or whether they are final states or not).

If there is a symbol $s$ so that not all states in a group $G$ agree on which group $s$ leads to, split $G$ based on which group the states go to on $s$.
Put states into groups based on their outputs (or whether they are final states or not)

If there is a symbol \( s \) so that not all states in a group \( G \) agree on which group \( s \) leads to, split \( G \) based on which group the states go to on \( s \)
### State Minimization Example

#### State Transition Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>S0</td>
<td>S3</td>
<td>S1</td>
<td>S5</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S1</td>
<td>S3</td>
<td>S2</td>
<td>S4</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>S1</td>
<td>S0</td>
<td>S4</td>
<td>S5</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S5</td>
<td>1</td>
</tr>
<tr>
<td>S5</td>
<td>S1</td>
<td>S4</td>
<td>S0</td>
<td>S5</td>
<td>0</td>
</tr>
</tbody>
</table>

Finally convert groups to states:

Can combine states S0-S4 and S3-S5.

In table replace all S4 with S0 and all S5 with S3
Minimized Machine

<table>
<thead>
<tr>
<th>State</th>
<th>Next State 0</th>
<th>Next State 1</th>
<th>Next State 2</th>
<th>Next State 3</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>S0</td>
<td>S3</td>
<td>S1</td>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S1</td>
<td>S3</td>
<td>S2</td>
<td>S0</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>S1</td>
<td>S0</td>
<td>S0</td>
<td>S3</td>
<td>0</td>
</tr>
</tbody>
</table>

State transition table

state
A Simpler Minimization Example
A Simpler Minimization Example

Split states into final/non-final groups

Every symbol causes the DFA to go from one group to the other so neither group needs to be split
Minimized DFA

\[ s_0 \rightarrow s_1 \]

\[ s_3 \rightarrow s_2 \]

\[ 0,1 \]
Another way to look at DFAs

Definition: The label of a path in a DFA is the concatenation of all the labels on its edges in order.

Lemma: $x$ is in the language recognized by a DFA iff $x$ labels a path from the start state to some final state.
Nondeterministic Finite Automata (NFA)

- Graph with start state, final states, edges labeled by symbols (like DFA) but
  - Not required to have exactly 1 edge out of each state labeled by each symbol—can have 0 or >1
  - Also can have edges labeled by empty string $\varepsilon$
- Definition: $x$ is in the language recognized by an NFA if and only if $x$ labels a path from the start state to some final state

![NFA Diagram](image-url)
Consider This NFA

What language does this NFA accept?
Consider This NFA

What language does this NFA accept?

$10(10)^* \cup 111 (0 \cup 1)^*$
NFA ε-moves

- States: s₀, s₁, t₀, t₁, t₂, q
- Transitions:
  - s₀: 0,1, ε → s₁, 2
  - s₁: 0,1, ε → t₁, 2
  - t₀: 0, ε → t₀, 2
  - t₁: 0, 1 → t₂, 1
  - t₂: 2, 1 → t₂, 2
NFA $\epsilon$-moves

Strings over \{0,1,2\} w/ even # of 2's OR sum to 0 mod 3
Three ways of thinking about NFAs

• Outside observer: Is there a path labeled by $x$ from the start state to some final state?

• Perfect guesser: The NFA has input $x$ and whenever there is a choice of what to do it magically guesses a good one (if one exists)

• Parallel exploration: The NFA computation runs all possible computations on $x$ step-by-step at the same time in parallel
NFA for set of binary strings with a 1 in the 3rd position from the end
NFA for set of binary strings with a 1 in the 3rd position from the end
Compare with the smallest DFA

\[ \begin{align*}
  &s_3 \xrightarrow{1} s_2 \xrightarrow{0,1} s_1 \xrightarrow{0,1} s_0 \\
\end{align*} \]
Parallel Exploration view of an NFA

Input string  0101100

0 1 0 1 1 0 0