Design of Digital Circuits and Systems, Quiz 1

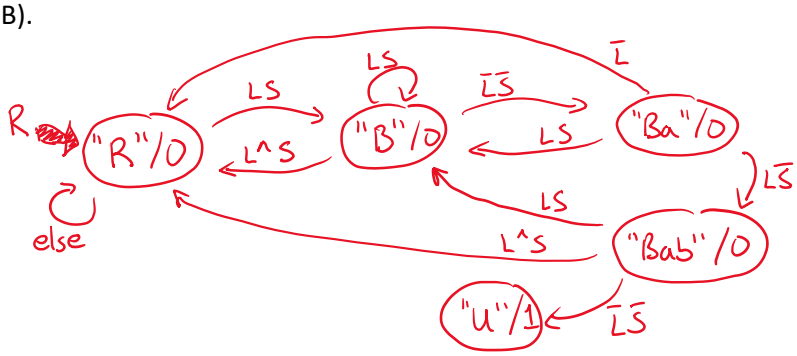
FSMs

Solution Outlines

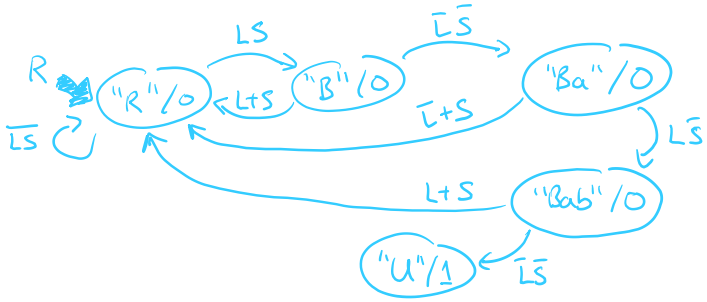
Part A:

This FSM is more naturally expressed as a Moore machine, so the example solutions here will use that, though equivalent Mealy machines were accepted. There were two main design decisions to make: (1) what do when an incorrect combination is entered and (2) what to do after unlocking. The three most common choices for (1) are shown below. Transitions out of the unlocked state are omitted to allow for different options (as described in Part B).

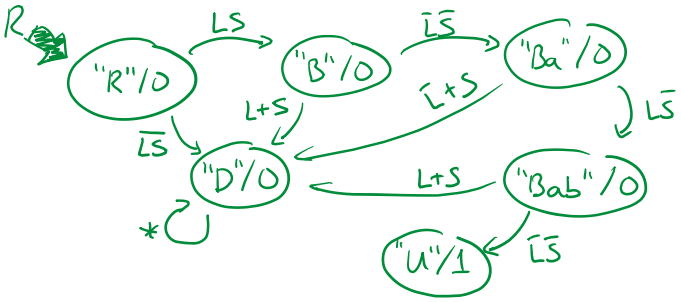
Option 1: Re-enter the sequence (*i.e.*, act like a string recognizer)



Option 2: Go to the reset state



Option 3: Go to a "dead" state



Notes:

- State names: shown above, reset ("R"), unlocked ("U"), dead ("D"). Anything appropriate was accepted (*e.g.*, "Baba" instead of "U", "0" or "Start" instead of "R").
- Output signal: it's Unlock, so should be a 0 when locked and a 1 when unlocked.
- Input values of `0` and `1` could have been given for `L` and `S` – assumed ordering `LS` if not specified.

Part B:

Description needed to match state diagram. The two most common decisions were:

1) All transitions take it to the reset state (or dead state, if applicable).
2) All non-reset transitions (*) remain in the unlocked state.

Explanation of why this choice was made needed to be included. #1 generally for safety to prevent the lock from staying open too long; #2 generally for convenience (better access to contents).

Part C:

Drawback of #1 might be that the lock only stays open for one clock cycle which is a timing issue or annoyance for user.

Drawback of #2 might be that the user forgets to lock it afterward (via the reset signal) or easier to brute force the combination (assuming no dead state) because it will remain unlocked if correct combination is somewhere in string.

Part D:

Problems relating to unlocked behavior (*i.e.*, the unlocked state) were not accepted since the problem asked for "beyond the unlocked behavior," which was covered in parts B and C.

Representative sample of accepted problems (definitely not complete):

- Timing: There is no way to idle – the user *must* set an input combination every clock cycle. This is problematic both when the clock speed is too fast (too difficult to change input combination in time) or too slow (lots of waiting or slow to lock again).
- Security: Easier to brute force combination (if no dead state) because can just keep trying with possibility of eventually finding the correct combination.
- Usability: Assuming there is a dead state, inconvenient to have to reset after every incorrect combination – good for security, annoying for users.
- Usability: Clunky to make user have to press multiple buttons simultaneously.

Part E:

Possible fixes to the listed problems above (again, this is not a comprehensive list of accepted answers):

- Timing: Idling could be introduced by adding an extra input that indicates when the next input selection has been made (e.g., "Go", "Enter", "Select"). Each existing transition on the state diagram would need to have G (asserted) added to it. Every state would idle when $\overline{G}$.
- Security: Add a dead state. All bad inputs would now transition to the dead state and the dead state always transitions to itself except on a reset.
- Usability: Remove dead state (*i.e.*, convert to Option 1 or 2 from Part A). Needed to describe the transition changes.
- Usability: Introduce additional buttons for the user, typically one each for a, b, A, and B. Need to augment every transition to now have 4 inputs AND needed to describe how simultaneous button presses would be handled (*e.g.*, count as bad input, idle).