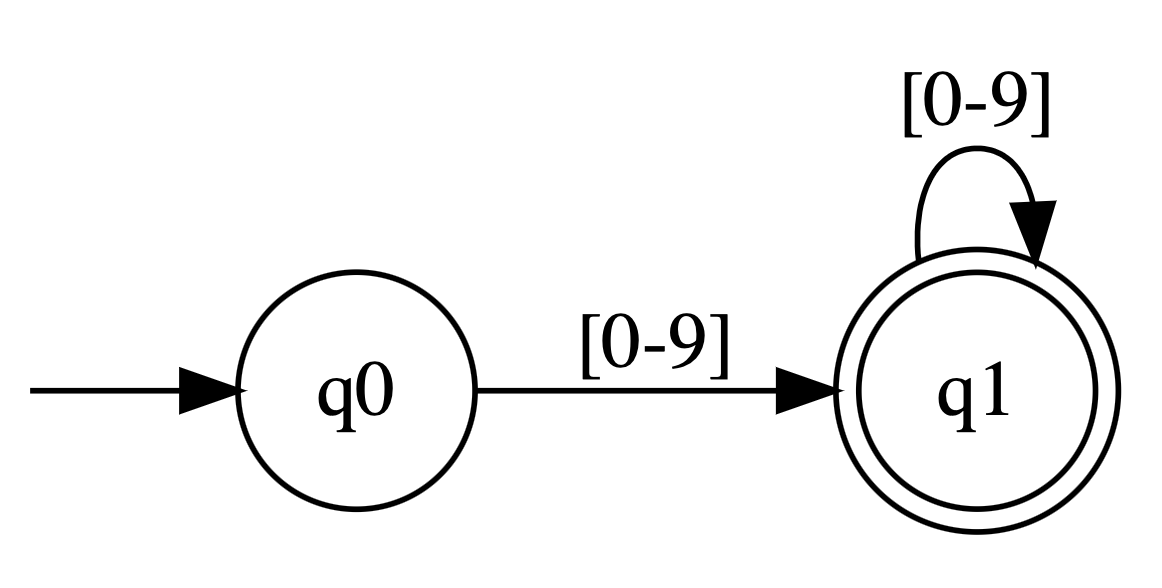
**My FSA Design Documents - Angel Villa**

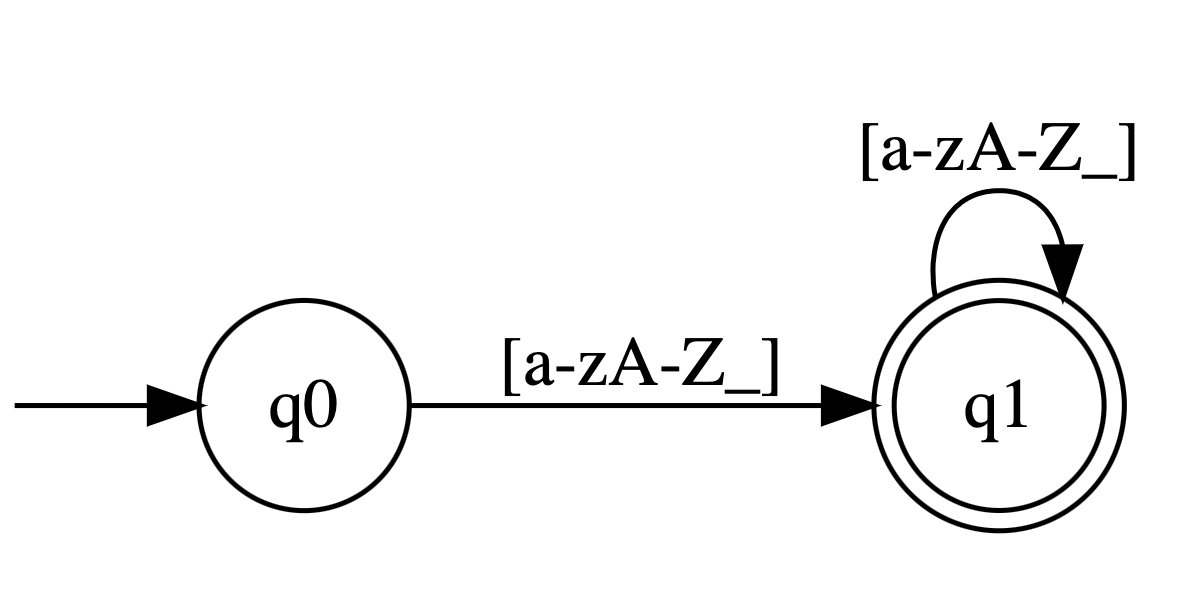
*Two FSA graphs, each one showing how a program would navigate when detecting specific values, letters, or symbols.*

**FOR NUMERICAL VALUES:**

****

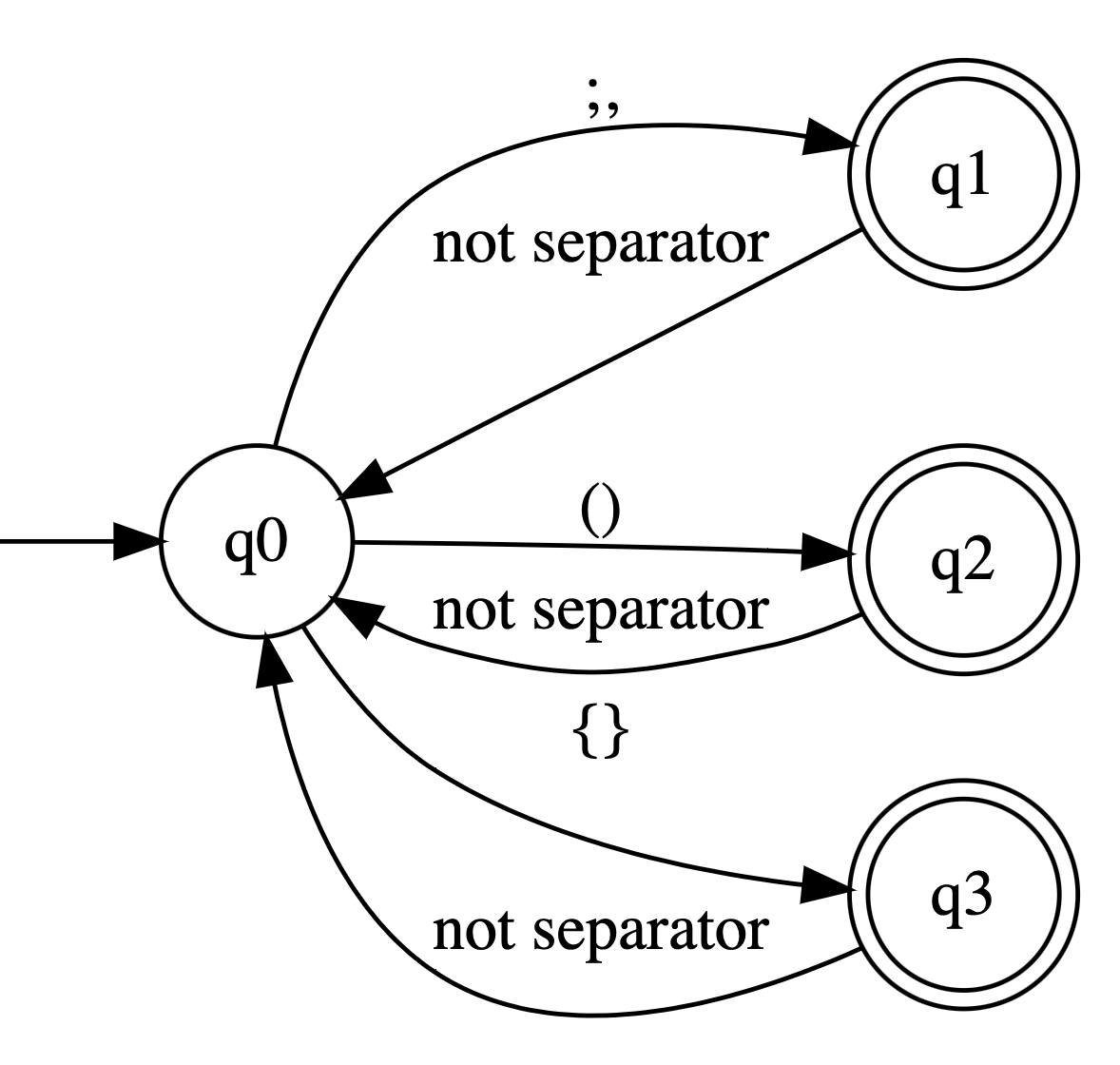
* The way this FSA works is that it starts off in state 0, then whenever a numerical input 0-9 moves to state q1, and continues looping if a continuation of numerical values are inputted. Then once numerical values are done, it just hits q1 as its final state.

**FOR ALPHABETICAL CHARACTERS:**



* The way this FSA works is that it starts off in state 0, then whenever a alphabetical input from A-Z whether lowercase or uppercase, moves to state q1, and continues looping if a continuation of alphabeticals are inputted. Then once alphabetical values are done, it just hits q1 as its final state.

**FOR SEPARATOR CHARACTERS:**

****

* Here’s another example of a slightly more “complicated” FSA that works for separators. In this FSA, if it is the separators “;” or “,” then the FSA will go to state 1, and will only go back to q0 if it’s not a separator which means q1 would be the final state.
* It will do the same for the separators “()” and “{}” and only move to states q2 or q3 if it’s the correct separator.
* This FSA can get a lot more complicated if we include EVERY single number, alphabet, or symbol in the FSA diagram. Then it would have many more states, as well as a lot of arrows returning to different stages at every step.
* Technically the FSA would expand exponentially, as every new state in the FSA would also include all the other symbols that could be next, making the FSA huge.
* This FSA could also be condensed to detect ALL the separator characters in one go, and instead just have q0 -> q1 if we grouped it all together. I just separated it into this FSA to show how they can get much bigger.