

- 1) My machine is a transforming FSA. Unlike a normal FSA where each node in the FSA diagram can only have one state, each node in the transforming can have multiple (but finite) states. Transitions from a given node can only be taken if the transition is for the node in its current state. Each transition can also change the state of the node it leaves or leave the state as is, including transitions which loop straight back to the same node. A node cannot be changed into or from a final state.

Every other facet of the machine is the same as a FSA. The machine takes an input string and either accepts the string or rejects it. The string is parsed one symbol at a time. The structure of the machine is defined by a series of nodes. These nodes are connected by transitions which define the input symbol necessary to take them. The transitions are also defined by a shape, color, or font (entirely the machine designer's preference as to what is used to differentiate between states of a node). In order to take a given transition from a node, the matching input symbol must be processed and the node must be in the matching state. If, when all of the input symbols are processed, the current node is a final node, the string is accepted. The string is rejected otherwise. Parsing the string begins at a predefined start node.

The tuple is as follows:

Transforming FSA $F = \{Q, S, \Sigma, \delta, q_0, F\}$

where Q is the set of nodes, S is the set of states for each node, Σ is the alphabet of the machine, δ is the set of transition/transformation rules, q_0 is the starting node, and F is the set of final states.

2) This Transforming FSA (TFSA) ranks the same as a standard FSA. Each new state can only replace one node from a normal FSA which performs the same function. It functions just like any other node in that it has a set of incoming and outgoing transitions. The difference is that every node of a normal FSA is active all the time, while only one state of a multi-state node of a TFSA is active at one time. Thus the states of a TFSA node can be easily separate into multiple nodes, creating a normal FSA. Likewise, if two nodes of an FSA match the necessary criterion (discussed in a bit), then they can be combined into one multi-state node. Not all states of an FSA can be combined, but since a TFSA may not actually have multiple states, any FSA is also a TFSA (TFSA nodes can have multiple states, but they don't have to).

The criterion for combining the nodes of an FSA to form a TFSA:

- Final and non-final nodes cannot be combined.
- Nodes can be combined only if the two nodes cannot be accessed through separate paths. That is, if it is possible to access one node without first proceeding through the other or vice versa, then the nodes cannot be combined. This is because each node can only be in one state at a time,

so if the node is in one state while the other state needs to be accessed, it is unavailable. See the examples for, umm, examples.

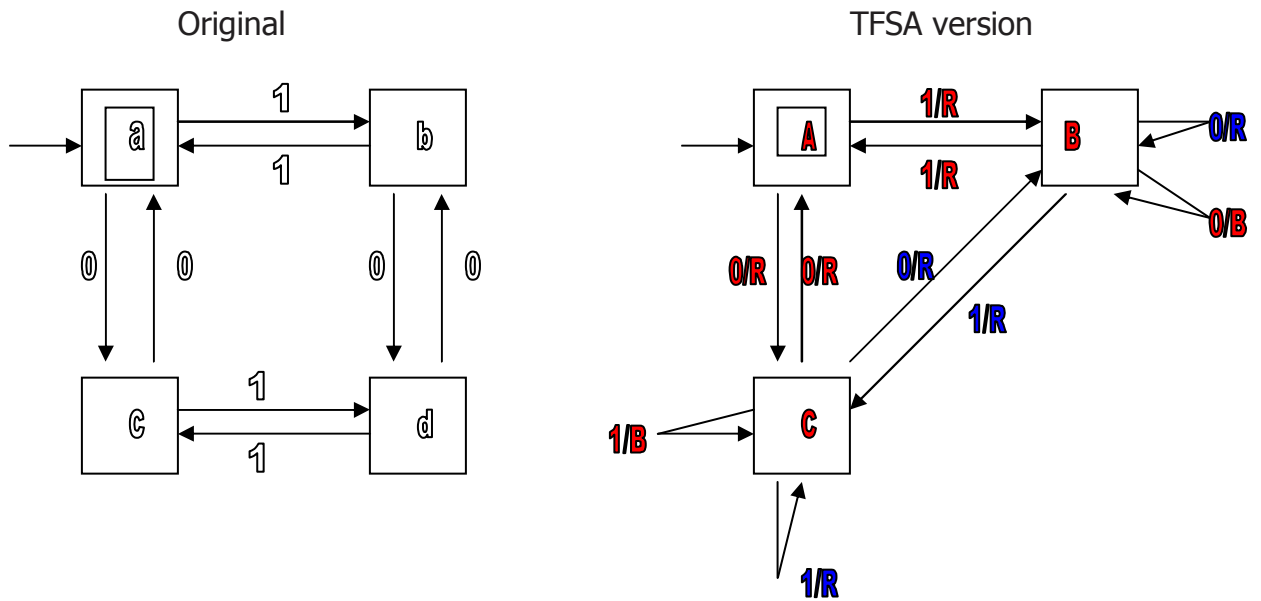
- This isn't a requirement, but it is often easiest to combine multiple nodes into a single multi-state node if the nodes are directly adjacent, especially if they are a one-way chain (they lead directly to one another).
- If you really want to combine two nodes when one of those nodes can be accessed without using the other, then the second node (the when with alternative paths to reach it) must be duplicated as an alternate state for every node which leads into it (see first example).

3. Are these TFSA algorithms? No. While any TFSA can be turned into an algorithm, there are many, many algorithms which cannot be converted into TFSA (for example, $a^i b^i$ where $\Sigma = \{a,b\}$ and $i \in \mathbf{N}$).

4. I'll be using colors to define the different state types. For example:

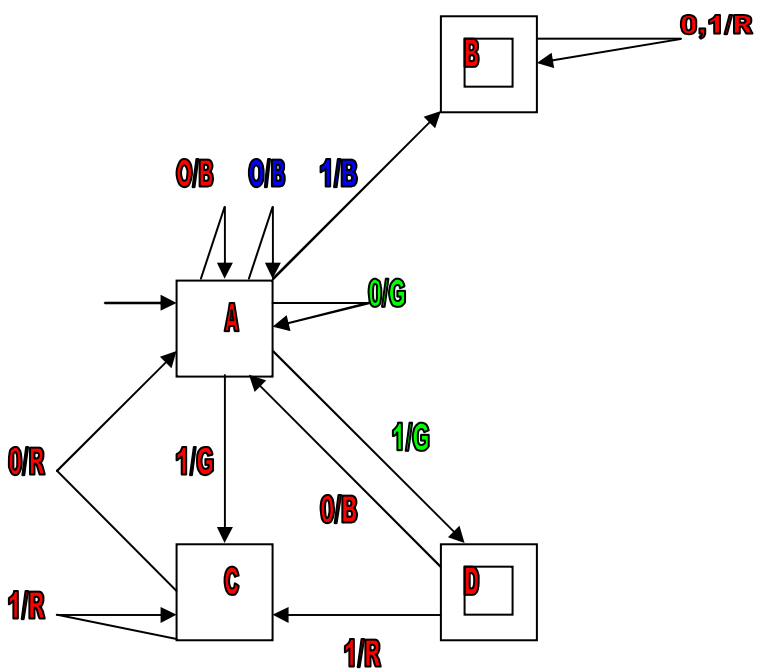
1/R means that the node must currently be blue to take that path and that the node will change to red after the path is taken. For reference, B is blue, R is red, G is green, and Y is yellow. All nodes begin in the RED state.

This machine accepts strings of 0 and 1 which contain an even number of 0's and 1's



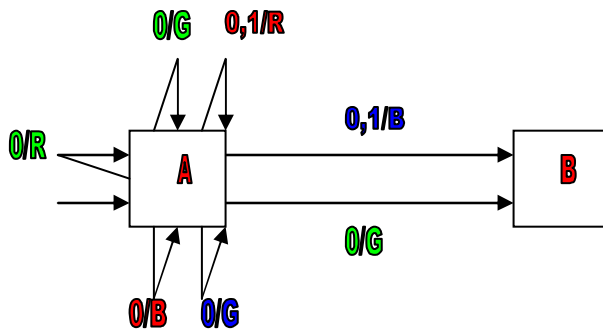
Notice that node **d**, because it can be reached by either **b** or **c**, must be duplicated as a second state on both **b** and **c** in the TFSA.

This machine accepts strings of 0 and 1 which begin or end with 01.

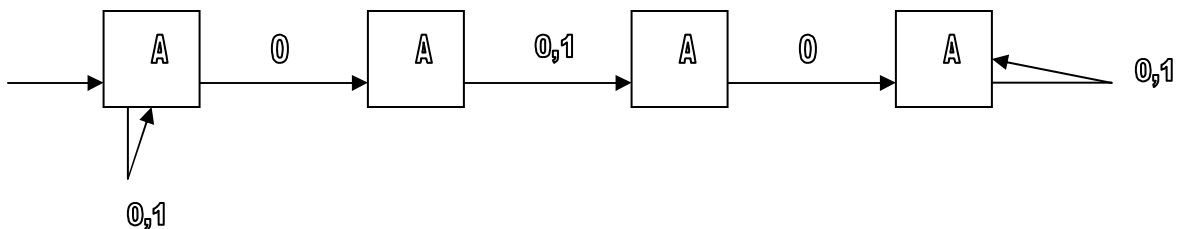


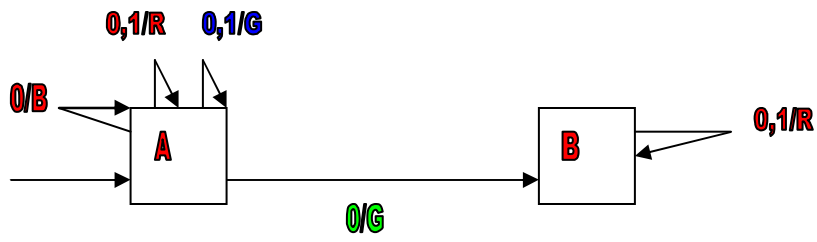
This machine is a TFSA of the following NFA (note that I did not minimize the machine, I simply did a straight conversion)

	>p	q	r	s*	t*
0	{p,q}	{r,s}	{p,r}	\emptyset	\emptyset
1	{p}	{t}	{t}	\emptyset	\emptyset



- Well, as I've shown, TFSA's are equivalent to FSA's. In the same way, NFA's are equivalent to TNFA's. The same rules apply to combining nodes to create the TNFA or dividing them to create the NFA (of course, those epsilon transitions and extra normal transitions can make reducing the nodes into multistate nodes challenging).
- Convert the following NFA into a TFSA. The TFSA should have no more than 2 nodes (you can leave it nondeterministic).





Convert the following TFSA into a DFA.

