Finite-State Automata

CS236 - Discrete Structures
Instructor: Brett Decker
Spring 2021

Finite-State Automata

Recall that we explored language recognition with finite-state machines. Finite-state automata are a special kind of machine used in language recognition.

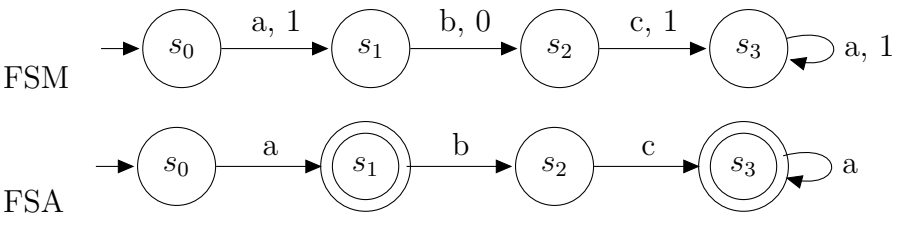
Finite-State Automaton: Definition 3, Section 13.3.3*

A *finite state automaton* $M = (S, I, f, s_0, F)$ consists of the following:

a finite set S of *states*
a finite *input alphabet* I
a *transition function* f that assigns to each state and input pair a new state
an *initial state* s_0
a set of *final states* F that is a subset of S

Finite-State Automaton

The statement that F is a subset of S means that F is comprised of some number of states that are in S (it could be none to all). In our state diagrams, final states will have a double circle for visual recognition. Note that there is no output for finite-state automata. We use finite-state automata for language recognition because they are a simplification of the FSM recognizers. Let's convert the following FSM to a FSA:



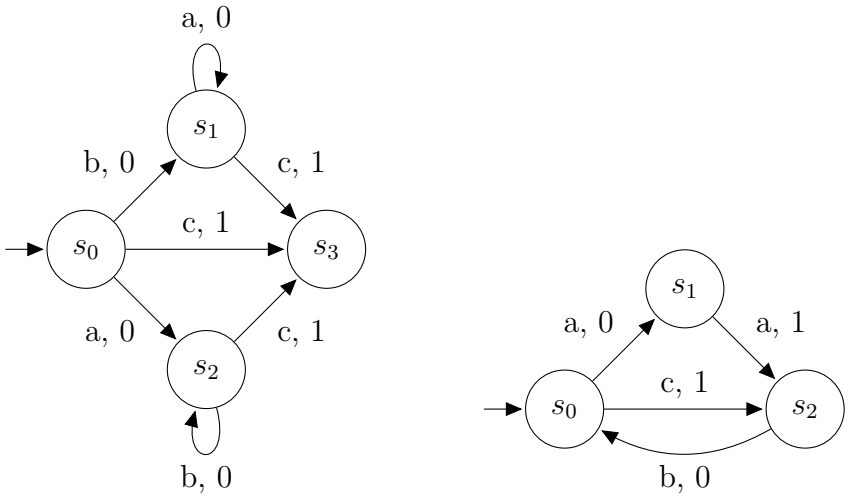
The two machines above recognize the exact same language. Thus we can use finite-state automata with regular expressions as well. To be clear, in the FSA above $S = \{s_0, s_1, s_2, s_3\}$ (same as the FSM) and $F = \{s_1, s_3\}$. I, f, and s_0 are also unchanged.

Language Recognition: Definition 4, Section 13.3.4*

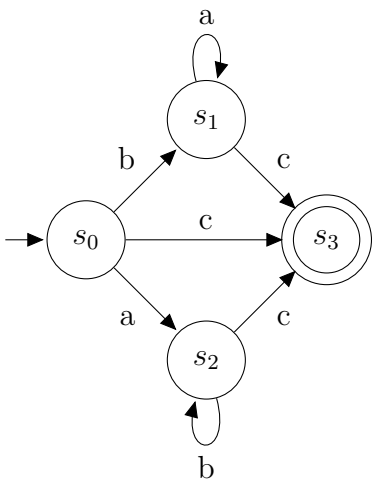
A string x is said to be *recognized* or *accepted* by the machine $M = (S, If, s_0, F)$ if it takes the initial state s_0 to a final state, that is, $f(s_0, x)$ is a state in F. The *language recognized* or *accepted* by the machine M, denoted by $L(M)$, is the set of all strings taht are recognized by M.

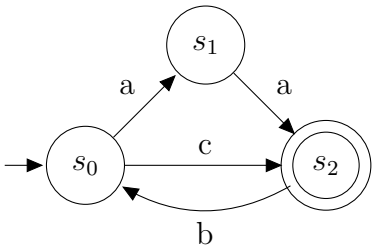
Finite-State Automaton Example:

Let's convert the FSMs from the Language Recognition Example (in the FSM lecture notes) to their corresponding FSAs. Here are the FSMs:



Recall that an FSA does not produce any output. It either accepts or rejects the input string. To convert from the FSMs into FSA we simply need to identify which transitions in the FSMs output a 1, which indicates acceptance of the input string. If all transitions into an individual state output a one, then this entire state is an acceptance state. Else, we will have to split the state into *two* states – one accepting state, one non-accepting state – in the FSA. In both examples given, all transitions output a one so we have the simple case. Here are the matching FSA:





Note that the above FSA are slightly simpler and more concise than their corresponding FSMs.

Finite-State Automata Equivalence: Definition 4, Section 13.3.4*

Two finite-state automata are called *equivalent* if they recognize the same language.

Finite-State Automata Equivalence

Note that the definition of equivalence for two finite-state automata is regardless of the number of states and transitions.

Conclusion

Finite-state automata theory is useful for a surprising number of recognition problems. For example, FSA theory can be used with formal verification for model checking. CS252 discusses more automata theory for those interested. See the book* for further examples and details.

*All definitions are from *Discrete Mathematics and Its Applications*, by Kenneth H. Rosen.