Notes on Sipser: §4.1, “Decidable Languages”
CSC 341, “Automata, Formal Languages, and Computational Complexity”
Department of Computer Science - Grinnell College
August 31, 2020

In the proof of Theorem 4.1, the Turing machine \( M \) is a general simulator for deterministic finite automata. It may seem counterintuitive that a single Turing machine can simulate any deterministic finite automaton, even one with a larger alphabet and more states than the Turing machine itself has.

The larger alphabet is not a problem, since the Turing machine can use many symbols from its own alphabet to represent a single symbol in the deterministic finite automaton’s alphabet. Since the deterministic finite automaton’s input is also encoded, there is no need to assume that a symbol of the deterministic finite automaton’s alphabet fits into a single tape square on the Turing machine that simulates it.

Similarly, the states of the deterministic finite automaton do not have to be stored inside the states of the Turing machine, since they can (and indeed must) be recorded instead somewhere on the Turing machine’s tape. The Turing machine may make many transitions in the process of simulating a single transition of the deterministic finite automaton, as it looks up the current state of the DFA and the symbol that is currently under the DFA’s head in the encoded transition function, recovers the new state of the DFA, copies it to the part of the tape where that information is stored, and advances the simulated head to the next tape square. At no point is it necessary to assume that the states of the Turing machine represent or reflect the individual states of the deterministic finite automaton.

• The proof of Theorem 4.2 reveals a previously unsuspected advantage of proofs by construction, where the construction method is itself expressed as an algorithm. Since the construction used in Theorem 1.39 is an algorithm, and since (according to the Church-Turing Thesis) every algorithm can be implemented as a Turing machine, Sipser feels entitled to incorporate that construction into the design of the Turing machine \( N \) as a kind of preprocessor, converting the input \( \langle B, w \rangle \) to \( N \) into an input \( \langle C, w \rangle \) suitable for Turing machine \( M \) from the proof of Theorem 4.1. Preprocessing inputs like this will play an increasingly important role in many of our proofs.

• The proof of Theorem 4.9 illustrates a slightly different kind of preprocessing. Each Turing machine \( M_G \) has an encoding of the grammar \( G \) stored in its finite-state control and begins just by prepending it to the encoding for \( w \) to produce the encoding for \( \langle G, w \rangle \) that the embedded Turing machine \( S \) needs. The encoding for any one context-free grammar \( G \) has a fixed, finite size, \( M_G \) can be provided with states that simply write \( \langle G \rangle \) onto the tape once \( w \) has been pushed far enough to the right to leave room for it.

Of course, the machine \( M_G \) is not general; it decides only the one context-free grammar \( G \). We need a different Turing machine for every choice of \( G \). But we already have the general Turing machine that can test whether \( G \) derives \( w \) for arbitrary \( G \) and \( w \). Sipser constructed it in the proof of Theorem 4.7.