

Incremental construction of DFCA

Andrei Paun^{1,2} Jason R. Smith¹

¹Computer Science Program, Louisiana Tech University, Ruston, LA 71272.

²Institute for Micromanufacturing, 911 Hergot Avenue, Ruston, LA 71272.

Deterministic finite automata (DFA) are widely used in applications where strings quickly tested for acceptance. The most common example would be a spell checker; a word is accepted if it is found in a dictionary and rejected if it is not. Through the use of DFAs, a word can be tested for acceptance without checking the entire dictionary. The only downside is that DFAs require extra memory. Several other directions of research have been conducted recently linking Deterministic Finite Automata with designing the so called “doctor in a cell” by Shapiro et. al.; these results have been reported in 2001 and 2004 in the Nature Journal. Unfortunately, the authors have been limited by the number of states in the automaton simulated, limitation imposed by the current biomolecular technology. We are providing the much needed framework of tools for a new type of deterministic automata that could replace DFA in such experiments, but will be able to perform the same job having less number of states than the DFA. This will provide a clear advantage for such a “doctor in cell”, as the “doctor” based on DFCA will be able to detect more “problems” (and take appropriate actions to fix them) using the same number of states as in the normal DFA.

Recently, the concept of deterministic finite cover automata (DFCA) has been created to alleviate the problem of the memory requirement of the DFAs. In any case where a finite language is being represented, such as the spell checking example, a DFCA can perform the same job as a DFA while requiring less memory. However, the process of building a minimal DFCA (or even a minimal DFA) for a given language can often require large amounts of memory. This is because the most common method is to first build a trie, which is a type of DFA that is often large but easy to build, and then minimize it.

One approach to solving this problem for DFAs is to build them word by word, and minimize them after each word is added. Since this method avoids trie building, the total amount of required memory is greatly reduced. By applying this concept of incremental construction to DFCA, the memory usage can be decreased by an even greater amount.