

Preliminary Experiments in Hardcoding Finite Automata

E. Ketcha Ngassam, Bruce. W. Watson, and Derrick. G. Kourie

Department of Computer Science, University of Pretoria, Pretoria 0002, South Africa
E-mail: {eketcha, bwatson, dkourie}@cs.up.ac.za

Abstract. Various experiments in hardcoding a single row of a transition table of a finite state machine directly into symbol-recognizing code are presented. Measurements are provided to show the time efficiency gains by various hardcoded versions over the traditional table-driven algorithm.

1 The hardcoding Experiment

The most commonly used algorithm to determine whether an input string is in the language of a finite automaton (FA) assumes that the transition function of the FA is represented as a table in memory. However, if the same table is to be repeatedly used to recognize strings, the question arises: “Would it not be more efficient to use a variation of the algorithm that encapsulates the transition table information into the code itself –i.e. to rely on an algorithm that hardcodes the FA?”

We have started to explore this question by comparing the timing behavior of five hardcoded algorithms against the table-driven algorithm on a Linux platform. The table-driven algorithm and two hardcoded algorithms were written in C++ and compiled using the gnu C++ optimizing compiler. The remaining three hardcoded algorithms were written in NASM assembler. Given the current input symbol, the first C++ hardcoded algorithm relies on the switch statement to determine the next state. The second C++ hardcoded algorithm relies on a sequence of nested if-then-else statements to determine the next state from the input symbol. The first and second assembler algorithms mimic these high-level versions in assembler. The switch statement is mimicked at the assembler level by making use of a jump table. The nested if-then-else statements are mimicked by assembler code that essentially executes a linear search through the list of accepting symbols in the current state. Finally, the third assembler algorithm relies on a direct jump statement to execute a block of code whose location is computed from the current input symbol.

All of these algorithms have been carefully designed to reflect the work to be done by some FA in accepting or rejecting a *single* character of some input string when the FA is in some arbitrary state. To provide code that would test whether a complete string is a member of the FAs language, one could simply extend any one of the algorithms in a relatively straightforward linear fashion.

An experiment was carried out in which the following steps were repeated several hundred times.

- Generate a row of a transition table such that the size and content of the set of accepting symbols are randomly determined; and each transition leads to some randomly determined next state. Regard the set of accepting symbols as the problem size.
- Generate the code that is associated with this transition row for each of the five hardcoded algorithms.
- Run this code as well as the relevant code for the table-driven algorithm, for each of the possible input symbols allowed by the alphabet under test. Each run is repeated twenty times. In each case the time taken to execute the code was recorded.

The resulting data revealed that the performance of each of the algorithms was substantially unrelated to the problem size. As a result, it seemed reasonable to base further comparisons of the six different coding possibilities on average values taken over all problem sizes. In each case, the *jump table* version is more than twice as fast as its nearest rival (the *linear search* version) and more than forty times faster than any of the high-level implementation versions, whether table-driven or hardcoded. Globally speaking, hardcoding implementations in assembler are faster than other methods. Moreover, hardcoding to a high-level language does not appear to be worthwhile, since the standard table-driven implementation seems to be slightly faster.

2 Conclusion

This study provides *prima facie* evidence that hardcoding in assembler could lead to significant performance improvements in relation to the table-driven algorithm. However, it would be naïve to assume that the recognition of an entire string merely involves a linear scale-up of the times obtained in these experiments. What needs to be explored is the paging, data caching and instruction caching impact on the table-driven approach (which uses a relatively small program but relies on a very large transition table as its data) and on the various hardcoded approaches (where the transition table is embedded into larger programs).