Hardcoding Finite Automata

By

ERNEST KETCHA NGASSAM

Supervisors:

Bruce W. Watson and Derrick G. Kourie

Submitted in partial fulfilment of the requirements for the degree of
MAGISTER SCIENTIA (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria
NOVEMBER 2003
To Maurice & Madeleine Ngassam, late Pierre Tatchou Tchoukwam, late Pauline Tchetnia, & late Jean Petji.
Abstract

The so-called conventional approach to implement finite automata is by mean of a matrix to represent the transition function. Of course, if the transition table is very sparse, linked lists might be used as an alternative. Such approaches therefore depend on the computer’s main memory capabilities to optimally hold the table for better processing.

For various computational problems using finite automata as a basic solution-model, the processing may be an important factor to be considered. This work aims to investigate a relatively new implementation approach that relies on hardcoding. A hardcoded algorithm uses simple instructions to represent the transition table. The algorithm is written in such a way that the transition matrix is part of its instructions as opposed to the traditional table-driven approach in which the table is external data that is to be accessed by the algorithm. This work includes a general performance analysis of both approaches through an empirical study. We firstly investigate the processing speed required to accept or reject a symbol by some randomly generated single states of some automata. Then, a more advanced experiment is performed based on the previous, for the test of acceptance of randomly generated strings by randomly generated finite automata.

The main result of this work is that the hardcoded implementations of finite automata outperform the table-driven implementation up to some threshold. This therefore emphasizes that many applications using finite automata as basic model may be optimized by replacing the table-driven implementation with a hardcoded implementation, resulting to better performances.

Keywords: Hardcoding, Automata, Pattern matching, Lexical analyzer, Algorithms, Experimentation, Performance, Grammars, Language, Parsing
Acknowledgements

I would like to thank Derrick G. Kourie and Bruce W. Watson, my supervisors, for their constant supports and unterminated suggestions during this research.

All my gratitude to Professor Bruce Watson for providing me the idea leading to the achievement of this work. Many thanks to Professor Derrick Kourie for its constant proof reading and critics throughout this research.

Of course, I am grateful to my parents, Maurice and Madeleine Ngassam, for their patience and love. Without them this work would never have come into existence. My thanks go also to my daughter Orline Ketcha, my son Ryan Ketcha, and my wife Liliane Ketcha who provided me with all the moral support needed to achieve such a work.

Finally, I wish to thank the following: Lisette Ngassam, Guy Ngassam, Laurent Ngassam, Mirabelle Ngassam, Orline Ngassam and Floriant Ngassam for their constant support and love through the path of achieving this goal.

Pretoria
October 31, 2003

Ernest Ketcha Ngassam
# Table of Contents

Abstract iii

Acknowledgements iv

Table of Contents v

List of Tables viii

List of Figures ix

1 Introduction 1

1.1 The Problem ............................................. 1
1.2 FAs in Context ............................................ 3
1.3 Objective of the dissertation 4
1.4 Methodology .............................................. 4
1.5 Dissertation Outline ...................................... 5

2 Background and Related Work 7

2.1 Introduction .............................................. 7
2.2 Finite Automata ........................................... 8

2.2.1 Deterministic Finite Automaton (DFA) ................. 9
2.2.2 Complexity of DFA String Recognition ............... 10
2.2.3 Non-Deterministic Finite Automata (NFA) ......... 11
2.2.4 Equivalence DFA and NFA ............................ 12

2.3 Finite Automata and Regular Expressions ............... 13

2.3.1 Operands and Operators of a Regular Expression 13
2.3.2 Equivalence of Finite Automata and Regular Expressions 14
2.3.3 Summary of the Section ............................... 15

2.4 Pattern Matching ......................................... 15

2.4.1 General Pattern Matching Algorithm ................ 15
# List of Tables

3.1 Evaluation of algorithm 3 and 4 ........................................ 43
B.1 The Table-driven Experiment Data ........................................ 128
B.2 The Switch Statements Data ........................................... 129
B.3 The Nested Conditional Statements Data ................................. 130
B.4 The Jump Table Data .................................................. 131
B.5 The Linear Search Data ............................................... 132
B.6 The Direct Jump Data .................................................. 133
B.7 Averaged Data collected independently to the problem size .......... 134
B.8 Sample Data for the two-alphabet symbols Experiments ............... 135
B.9 Sample Data for the String Recognition Experiment with Searching and Direct Indexing ........................................... 136
List of Figures

2.1 A finite automaton ........................................... 9
2.2 A state transition diagram .................................. 9
2.3 A Push Down Automaton .................................... 21

3.1 A state in the transition diagram of some finite automaton ...... 47
3.2 A transition array for a state of some automaton ............... 47

4.1 Process diagram indicating how the hardcoded implementation were compared to the table-driven implementation .................. 56

6.1 Average processing speed for Table-driven implementation (accepting and rejecting symbols) ............................ 80
6.2 Accepting symbol performance for NCSs ........................ 82
6.3 Rejecting symbol performance for NCSs ......................... 83
6.4 Performance based on ASs for SSs ............................. 84
6.5 Performance based on RSs for SSs ............................. 85
6.6 Performance based on hardcoding implementation in high-level language 86
6.7 Performance based on ASs for JT .............................. 87
6.8 Performance based on RSs for JT .............................. 88
6.9 Performance based on ASs for LS .............................. 89
6.10 Performance based on RSs for LS ............................ 90
6.11 Performance based on ASs for DJ ............................. 91
6.12 Performance based on RSs for DJ ............................. 91