



UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

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 Tchetgnia, & 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 uninterminated 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 <i>FAs</i> 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 <i>DFA</i> String Recognition	10
2.2.3 Non-Deterministic Finite Automata (NFA)	11
2.2.4 Equivalence <i>DFA</i> and <i>NFA</i>	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



2.4.2	String Keyword Pattern Matching and Finite Automata . . .	16
2.4.3	Summary of the section	17
2.5	Lexical Analysis	17
2.5.1	Summary of the section	18
2.6	Context Free Grammars	18
2.6.1	Definition	19
2.6.2	Context Free Grammars and Regular expressions	20
2.6.3	Push Down Automata	20
2.6.4	Parsing and Code Generation	23
2.7	Related Work	29
2.7.1	Pennello	29
2.7.2	Horspool and Whitney	33
2.7.3	Bhamidiapaty and Proebsting	35
2.8	Summary of the chapter	36
3	Problem Domain Restriction	38
3.1	Introduction	38
3.2	The Table-Driven Algorithm	39
3.3	The Hardcoded Algorithm	40
3.4	Comparison of Hardcoding and Table-Driven Algorithms	42
3.5	Problem Restriction	44
3.6	Single Symbol Recognition	46
3.7	Hardcoding Single Symbol Recognition	48
3.8	Summary	49
4	Tools and Methodology	50
4.1	Introduction	50
4.2	Hardware Considerations	50
4.3	Software Considerations	51
4.4	The Intel Pentium Read Time Stamp Counter Instruction	52
4.5	Random Number Generation	52
4.6	Methodology	54
4.7	Chapter Summary	55
5	Implementation	57
5.1	Introduction	57
5.2	The Random Transition Array	57
5.3	Table-Driven Implementation	59
5.4	Hardcoded Implementations	61
5.4.1	Use of the Nested Conditional Statements	66
5.4.2	Use of the Switch Statements	67

5.4.3	Use of a Jump Table	69
5.4.4	Use of a Linear Search	72
5.4.5	Use of a Direct Jump	74
5.5	Data Collection	76
5.6	summary of the chapter	78
6	Experimental Results	79
6.1	Introduction:	79
6.2	Table-Driven Experimental Results	79
6.3	Hardcoding Experimental Results	80
6.3.1	High-Level Language Hardcoding	81
6.3.2	Low-Level Language Hardcoding	83
6.3.3	Overall Results of Hardcoding	87
6.4	Final Results	88
6.5	Summary of the chapter	89
7	String Recognition Experiments	94
7.1	Introduction:	94
7.2	Exercising Memory on Intel Pentium Architecture	95
7.2.1	A Simple Experiment and Results	97
7.3	The String Recognition Experiment	104
7.3.1	Experimental Results	107
7.4	Summary of the Chapter	111
8	Summary and Future Work	113
8.1	Summary and Conclusion	113
8.2	Future Work	118
A	Random Number Generator	120
B	Data Collected	126
	Bibliography	137

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 <i>NCSs</i>	82
6.3	Rejecting symbol performance for <i>NCSs</i>	83
6.4	Performance based on <i>ASs</i> for <i>SSs</i>	84
6.5	Performance based on <i>RSs</i> for <i>SSs</i>	85
6.6	Performance based on hardcoding implementation in high-level language	86
6.7	Performance based on <i>ASs</i> for <i>JT</i>	87
6.8	Performance based on <i>RSs</i> for <i>JT</i>	88
6.9	Performance based on <i>ASs</i> for <i>LS</i>	89
6.10	Performance based on <i>RSs</i> for <i>LS</i>	90
6.11	Performance based on <i>ASs</i> for <i>DJ</i>	91
6.12	Performance based on <i>RSs</i> for <i>DJ</i>	91



6.13	Performance of low-level hardcoded implementations	92
6.14	Performance based on hardcoding implementation	92
6.15	Average processing speed per implementation technique	93
7.1	Hardcoded time against automaton size for two symbols alphabet . .	101
7.2	Table-driven time against automaton size for two symbols alphabet .	104
7.3	table-driven and hardcoded multiple states for two symbols alphabet	105
7.4	Table-driven and hardcoded performance using linear search	108
7.5	Table-driven and hardcoded performance using direct index	110
7.6	Table-driven and hardcoded performance using direct index	110
7.7	Performance based on searching and direct indexing	111