

**TOWARDS CACHE OPTIMIZATION IN FINITE  
AUTOMATA IMPLEMENTATIONS**

By

**ERNEST KETCHA NGASSAM**

**Submitted in partial fulfilment of the requirements for the degree  
Philosophiae Doctor (Computer Science) in the  
Faculty of Engineering, Built Environment and Information Technology  
University of Pretoria, Pretoria**

**September 2006**

**TOWARDS CACHE OPTIMIZATION IN FINITE  
AUTOMATA IMPLEMENTATIONS**

By

**ERNEST KETCHA NGASSAM**

**Supervisor: Bruce W. Watson**

**Co-Supervisor: Derrick G. Kourie**

**Department of Computer Science,  
University of Pretoria**

## ABSTRACT

To the best of our knowledge, the only available implementations of FA-based string recognizers are the so-called conventional table-driven algorithm and, of course, its hardcoded counterpart suggested by Thompson, Penello, and DeRemer in 1967, 1986, and 2004 respectively. However, our early experiments have shown that the performance of both implementations is hampered by the random access nature of the automaton's transition table in the case of table-driven, and also the random access nature of the directly executable instructions that make up each hardcoded state. Moreover, the problem of memory load and instruction load are also performance bottlenecks of these algorithms, since, as the automaton size grows, more space in memory is required to hold data/instructions relevant to the states.

This thesis exploits the notion of cache optimization (that requires good data or instructions organization) in investigating various enhancements of both table-driven and hardcoding.

Functions have been used to formally define the denotational semantics of string recognizers. These functions rely on various so-called *strategy variables* that are integrated into the formal definition of each recognizer. By appropriately selecting these variables, the conventional algorithms may be described, without loss of generality. By specializing these strategy variables, the new and enhanced recognizers can be denotationally described, and resulting algorithms can then be implemented.

We first introduce the so-called *Dynamic State Allocation* (DSA) strategy regarded as a sort of Just-In-time (JIT) implementation of FA-based string recognizers whereby a predefined portion of the memory is reserved for acceptance testing. Then follows the *State pre-Ordering* (SpO) strategy that assumes some prior knowledge on the order in which states would be visited. In this case, acceptance testing takes place once each state have been allocated to its new position in memory. The last strategy referred to as the *Allocated Virtual Caching* (AVC) strategy is based on the premise that a portion of the memory originally occupied by the automaton's states is virtually used as a sort of cache memory in which acceptance testing takes place, enabling therefore, the exploitation of the various performance enhancement notions on which hardware cache memory relies.

It is shown that the algorithms can be classified in a taxonomy tree which is further mapped into a class-diagram that represents the design of a toolkit for FA-based string recognition. Also given in the thesis are empirical results that indicate that the algorithms suggested can, in general, outperform their conventional counterparts when recognizing large and appropriately chosen input strings.

**Keywords:** Finite automata, algorithmic, taxonomy, toolkit, software construction, cache optimization, locality of reference, implementation strategies, performance, string recognizer.

*To Maurice and Madeleine Ngassam;*  
*To Orline Tchamen Ngassam;*  
*To late Pierre Tatchou Tchoukwam;*  
*To late Pauline Tchegnina;*  
*To late Jean Petji;*  
*To my Ancestors.*

## ACKNOWLEDGEMENTS

I would like to thank Derrick G. Kourie and Bruce W. Watson, my supervisors, for their many suggestions and constant support during this research. Their support and guidance from the early years of chaos and confusion until the very end of this research will never be forgotten. They have been impressively valuable in enhancing my research capabilities, my thinking abilities, and of course my commitment to hardwork.

During this whole journey, I had the opportunity to interact with various Universities. Such interactions have indeed been fruitful along the path of achieving this goal. To all colleagues and friends of the **FASTAR** (**F**inite **A**utomata **S**ystems, **T**heoretical and **A**ppplied **R**esearch) Research group of the University of Pretoria (Department of Computer Science) and the Technical University of Eindhoven (Department of Software Construction), to those at the Czech Technical University in Prague (Department of Computer Science), and to those at the University of South Africa (School of Computing, my employer), I would like to take this opportunity to express my gratitude for their constant support. A special thank to Professor Paula Kotzé of the School of Computing at the University of South Africa for her constant support and encouragement throughout my journey of becoming a *researcher*.

I should also mention that my studies were supported by the University of Pretoria through its postgraduate bursary scheme (Postgraduate Student Award) as well as the National Research Foundation (NRF) through the Postgraduate Supervision Bursary Nomination Scheme graciously motivated by my supervisors Professors Derrick G. Kourie and Bruce W. Watson.

Of course, I am grateful to my parents Maurice and Madeleine, for their patience and *love*. Without them this work would never have come into existence (literally).

My thanks also go to my immediate family: my wife Marie Louise Liliane Yemdam-Ketcha, my children Orline Sorel Ketcha, Karl Ryan Ketcha, Ashlyne Shakirah Ketcha, and my niece Carine Ngami Tchouatchoua. Without them, the motivation, the energy and the passion behind this work would have not been materialized.

Finally I wish to thank the following for their *unconditional* support and love: Lysette Tchatat Ngassam, Guy Tchoukwam Ngassam, Laurent Wandja Ngassam, Mirabelle Soumbé Ngassam, Orline Tchamen Ngassam (*my angel*), and Aimé Floriant Noungo Ngassam. To them I wish to say, *the Ngassam's dream has finally come true, and the real battle has just begun*.

Pretoria, South Africa  
Semptember 7, 2006

Ernest Ketcha Ngassam

## TABLE OF CONTENTS

	Page
<b>ABSTRACT</b> .....	<b>iii</b>
<b>ACKNOWLEDGEMENTS</b> .....	<b>v</b>
<b>LIST OF TABLES</b> .....	<b>x</b>
<b>LIST OF FIGURES</b> .....	<b>xi</b>
<b>I Prologue</b>	<b>1</b>
<b>CHAPTER</b>	
<b>1. INTRODUCTION</b> .....	<b>2</b>
1.1 The Problem .....	3
1.2 The Kind of Automata .....	4
1.3 Objective of the thesis .....	4
1.4 Methodology .....	5
1.5 Thesis Outline .....	6
<b>2. PRELIMINARIES</b> .....	<b>8</b>
2.1 Introduction .....	8
2.2 Formal elements of string processing .....	8
2.3 Operation of superscalar processors and performance metrics .....	12
2.3.1 Basic principles of computer architecture .....	13
2.3.2 Operation of a superscalar processor .....	15
2.3.3 Operational diagram of a superscalar processor .....	16
2.3.4 Performance metrics identification .....	18
2.4 Summary of the chapter .....	20
<b>II FA-based String Processing Algorithms</b>	<b>21</b>
<b>3. CORE ALGORITHMS</b> .....	<b>22</b>

3.1	The table-driven algorithm . . . . .	22
3.2	The hardcoded algorithm . . . . .	23
3.2.0.1	An Example . . . . .	26
3.3	The mixed-mode algorithm . . . . .	29
3.4	Functional description of core recognizers . . . . .	31
3.5	Summary of the Chapter . . . . .	32
<b>4.</b>	<b>DYNAMIC STATE ALLOCATION . . . . .</b>	<b>34</b>
4.1	DSA Characterization . . . . .	34
4.1.1	Properties of the DSA strategy . . . . .	36
4.2	The TD-DSA algorithm . . . . .	38
4.3	The HC-DSA Algorithm . . . . .	41
4.4	The MM-DSA Algorithm . . . . .	44
4.5	Illustrative Example . . . . .	46
4.6	A theoretical assessment . . . . .	48
4.7	Summary of the Chapter . . . . .	49
<b>5.</b>	<b>STATES PRE-ORDERING . . . . .</b>	<b>50</b>
5.1	The SpO-based Characterization . . . . .	50
5.2	Properties of the SpO strategies . . . . .	51
5.3	The TD-SpO algorithm . . . . .	53
5.4	The HC-SpO algorithm . . . . .	56
5.5	The MM-SpO algorithm . . . . .	59
5.6	Theoretical Assessment . . . . .	61
5.7	Summary of the Chapter . . . . .	62
<b>6.</b>	<b>ALLOCATED VIRTUAL CACHING . . . . .</b>	<b>64</b>
6.1	The AVC-based Characterization . . . . .	64
6.2	Properties of the AVC strategies . . . . .	66
6.3	The Table-driven-AVC algorithm . . . . .	68
6.4	The hardcoded-AVC algorithm . . . . .	71
6.5	The mixed-mode-AVC algorithm . . . . .	74
6.6	Illustrative example . . . . .	76
6.7	Theoretical assessment . . . . .	80
6.8	Summary of the Chapter . . . . .	81
<b>7.</b>	<b>TAXONOMY OF FA-BASED STRING PROCESSORS . . . . .</b>	<b>82</b>
7.1	Related Work . . . . .	82
7.2	New Characterization of FA-based String Processors . . . . .	85
7.2.1	Derivation of the TD algorithms . . . . .	87
7.2.1.1	The TD-SpO-AVC algorithm . . . . .	89
7.2.1.2	The bounded TD-DSA-SpO algorithm . . . . .	89
7.2.1.3	The bounded TD-DSA-SpO-AVC algorithm . . . . .	90
7.2.1.4	The bounded TD-DSA-AVC algorithm . . . . .	91
7.2.1.5	The unbounded TD-DSA-SpO algorithm . . . . .	92

7.2.1.6	The unbounded TD-DSA-SpO-AVC algorithm . . . . .	93
7.2.1.7	The unbounded TD-DSA-AVC algorithm . . . . .	93
7.2.2	Derivation of the hardcoded algorithms . . . . .	94
7.2.2.1	The HC-SpO-AVC algorithm . . . . .	95
7.2.2.2	The bounded HC-DSA-SpO algorithm . . . . .	96
7.2.2.3	The bounded HC-DSA-SpO-AVC algorithm . . . . .	97
7.2.2.4	The bounded HC-DSA-AVC algorithm . . . . .	98
7.2.2.5	The unbounded HC-DSA-SpO algorithm . . . . .	98
7.2.2.6	The unbounded HC-DSA-AVC algorithm . . . . .	99
7.2.2.7	The unbounded HC-DSA-SpO-AVC algorithm . . . . .	99
7.2.3	Derivation of the mixed-mode algorithms . . . . .	100
7.3	The taxonomy . . . . .	102
7.4	Summary of the Chapter . . . . .	105
<b>8.</b>	<b>TOOLKIT DESIGN FOR FA-BASED STRING RECOGNIZERS . . . . .</b>	<b>107</b>
8.1	Motivation and Related Work . . . . .	107
8.2	The Architectural Design . . . . .	109
8.2.1	The Package PkgRecognizer . . . . .	110
8.2.2	The Package PkgTableDriver . . . . .	113
8.2.2.1	The first level of the TD class-diagram . . . . .	114
8.2.2.2	The second level of the TD class-diagram . . . . .	115
8.2.2.3	The last level of the TD class-diagram . . . . .	116
8.2.3	The Package PkgHardCoder . . . . .	117
8.2.4	The Package PkgMixedModer . . . . .	118
8.2.5	A Detailed Toolkit's Architecture . . . . .	121
8.3	Summary of the Chapter . . . . .	123
<b>III</b>	<b>Performance of DFA-based String Processors . . . . .</b>	<b>124</b>
<b>9.</b>	<b>INTRODUCTION AND MOTIVATIONS . . . . .</b>	<b>125</b>
9.1	The Software and Hardware Context . . . . .	125
9.2	Performance Measurement . . . . .	128
9.3	Summary of the Chapter . . . . .	128
<b>10.</b>	<b>EXPERIMENTAL RESULTS . . . . .</b>	<b>130</b>
10.1	Introduction . . . . .	130
10.2	The Table-driven Experiments . . . . .	130
10.3	The Hardcoded Experiments . . . . .	136
10.4	The Mixed-mode Experiments . . . . .	140
10.5	Summary of the Chapter . . . . .	142



<b>IV Epilogue: Conclusion and Future Work</b>	<b>145</b>
<b>11.SUMMARY AND CONCLUSION</b>	<b>146</b>
11.1 Summary	146
11.2 Conclusion	147
<b>12.FUTURE WORK</b>	<b>149</b>
12.1 Projects of limited scale	149
12.2 Medium-scale projects	149
12.3 Advanced research projects	150
12.4 End note	150
<b>BIBLIOGRAPHY</b>	<b>151</b>

## LIST OF TABLES

Table	Page
3.1 The transition table $\Delta$ of the TD recognizer of example 3.2.0.1 .....	27
4.1 The arrays $\delta$ , $m$ , and $d$ for the DSA Example.....	47
6.1 The arrays $\delta$ , $m$ , $i$ , and $c$ initially for the AVC example .....	77
6.2 The arrays $\delta$ , $m$ , $i$ , and $c$ after the second iteration for the AVC example .....	78
6.3 The arrays $\delta$ , $m$ , $i$ , and $c$ after the third iteration for the AVC example .....	79
6.4 The arrays $\delta$ , $m$ , $i$ , and $c$ after the fifth iteration for the AVC example .....	79
6.5 The arrays $\delta$ , $m$ , $i$ , and $c$ after the sixth iteration for the AVC .....	80
7.1 The derived TD-based algorithms .....	88
7.2 The range of HC-based algorithms .....	95
7.3 A range of MM algorithms .....	101
10.1 Rate at which states are visited for the first time .....	131

## LIST OF FIGURES

Figure	Page
2.1 Operation diagram of superscalar processors based on information gathered from [PH05, HP03] .....	17
3.1 A State diagram that accepts the string <i>void</i> .....	27
4.1 A State diagram for testing the string <i>abcbaabcbaabcba</i> .....	46
6.1 A State diagram for testing the string <i>abcabcaabccaabcc</i> .....	76
7.1 A taxonomy of FA-based String Processing Algorithms.....	103
8.1 A high-level toolkit’s view based on interacting packages .....	110
8.2 The class diagrams of <code>PkgRecognizer</code> .....	111
8.3 The <code>TableDriver</code> class diagram .....	113
8.4 The <code>HardCoder</code> class diagram .....	118
8.5 An extract <code>MixedModer</code> class diagram .....	119
8.6 An extract FA-based String Recognizers class-diagram .....	122
9.1 The structure of a NASM code generator .....	128
10.1 Total number of states vs: number of newly visited states in segment 1, 2 , 3 and 4 (I, II, III & IV) .....	132
10.2 Performances of TD vs: DSA (I & II), SPO (III), and AVC (IV) .....	135
10.3 Performances of TD vs: DSA-SpO (I & II), DSA-AVC (III & IV), DSA-SpO-AVC (V & VI), and SpO-AVC (VII) .....	137
10.4 Performances of HC vs: HC-DSA (I & II), SpO (III), and AVC (IV) .....	139

10.5 Performances of HC, TD and MM (where 25% of the states are  
HC) .....141

10.6 Performances of TD and MM (where 25% of the states are HC, and  
the first 75% of states are frequently visited) .....143