
Evolving Automata Using Genetic Programming

by

AMASHINI NAIDOO

Submitted in fulfilment of the academic
requirements for the degree of
Master of Science
in the Faculty of Science and Agriculture,
University of KwaZulu-Natal,
Durban

Supervisor:
Dr. Nelishia Pillay

Abstract

Automata have played a significant role in the field of computer science. The idea of automatically inducing automata has been the object of the computer science community for a number of years. Genetic programming (GP) is an evolutionary algorithm modeled on the Darwinian idea of natural selection and genetic recombination, where individuals are typically represented as tree-structures.

This thesis investigates genetic programming as a method to automatically evolve various classes of automata including finite acceptors, pushdown automata, finite state transducers and Turing machines for benchmark sets of problems. A new approach to evolving automata is introduced whereby each individual in the GP population is represented directly as a graph as opposed to a tree. The methods for evaluation, standard and advanced GP characteristics, and the GP parameters are identified.

Genetic programming proves to be an effective method for inducing automata. The GP systems presented in this thesis successfully induce solutions for finite acceptors; pushdown automata, finite state transducer and Turing machine languages. Furthermore, it is shown that using non-destructive operators and multiple iterations improve the success rate of the GP system.

Preface

The work described in this dissertation was carried out in the School of Computer Science, University of KwaZulu–Natal, Westville, Durban from February 2004 to February 2008 under the supervision of Dr. N. Pillay.

These studies represent original work by the author and have not otherwise been submitted in any form for any degree or diploma to any tertiary institution. Where use has been made of the work of others, it is duly acknowledged in the text.

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Nelishia Pillay, for her guidance and help. She has provided assistance in numerous ways, and I would especially like to thank her for her comments and ideas with regards to this work.

I would like to thank the National Research Foundation, who have financially supported this work. Once again, I would like to thank Dr. Pillay, for her assistance in obtaining this sponsorship.

Finally, I would like to thank my family for supporting me the last four years. A special thank you to my sister, Thrishini Naidoo, who proof read this thesis. Despite having no Computer Science background, she was able to point out several errors and typos. Her help is much appreciated.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 1.1 | Purpose of Study | 1 |
| 1.2 | Objectives | 1 |
| 1.3 | Scope of Study | 2 |
| 1.4 | Contributions to the Study | 2 |
| 1.5 | List of publications | 2 |
| 1.6 | Thesis Layout | 3 |
| 2 | An Overview of Automata | 4 |
| 2.1 | Introduction | 4 |
| 2.2 | Classes of Automata | 4 |
| 2.2.1 | Finite Acceptors | 4 |
| 2.2.1.1 | Deterministic Finite Acceptors | 4 |
| 2.2.1.2 | Nondeterministic Finite Acceptors | 5 |
| 2.2.2 | Finite State Transducers | 6 |
| 2.2.2.1 | Mealy Machine | 7 |
| 2.2.2.2 | Moore Machine | 8 |
| 2.2.3 | Pushdown Automata | 8 |
| 2.2.4 | Turing Machines | 10 |
| 2.3 | The Chomsky Hierarchy (also known as Chomsky–Schützenberger hierarchy) | 12 |
| 2.4 | Applications of Automata | 12 |
| 3 | An Overview of Genetic Programming | 19 |
| 3.1 | Introduction | 19 |
| 3.2 | Brief Overview | 19 |
| 3.3 | Detailed Description | 20 |
| 3.3.1 | Control Models | 20 |
| 3.3.1.1 | Generational Control Model | 20 |
| 3.3.1.2 | Steady State Control Model | 20 |
| 3.3.1.3 | Varying Population Size Control Model | 20 |
| 3.3.2 | Representation of an Individual | 20 |
| 3.3.3 | Initialising the Population | 23 |
| 3.3.4 | Fitness | 25 |
| 3.3.4.1 | Raw Fitness | 25 |
| 3.3.4.2 | Standardized Fitness | 25 |
| 3.3.4.3 | Adjusted Fitness | 26 |
| 3.3.4.4 | Normalized Fitness | 26 |
| 3.3.5 | Selection | 26 |
| 3.3.5.1 | Fitness Proportionate Selection | 26 |
| 3.3.5.2 | Truncation or (μ, λ) Selection | 27 |

| | | |
|----------|---|-----------|
| 3.3.5.3 | Tournament Selection | 27 |
| 3.3.5.4 | Rank Selection | 27 |
| 3.3.6 | Genetic Operators | 27 |
| 3.3.6.1 | Reproduction | 27 |
| 3.3.6.2 | Mutation | 28 |
| 3.3.6.3 | Crossover | 29 |
| 3.3.7 | User Decisions | 29 |
| 4 | Previous Work | 30 |
| 4.1 | Introduction | 30 |
| 4.2 | Finite State Machines | 30 |
| 4.3 | Finite State Transducers | 42 |
| 4.4 | Pushdown Automata | 42 |
| 4.5 | Turing Machines | 49 |
| 5 | Methodology | 56 |
| 5.1 | Introduction | 56 |
| 5.2 | Research Methodology Employed | 56 |
| 5.3 | Language Selection | 57 |
| 5.3.1 | Finite Acceptors | 57 |
| 5.3.2 | Finite State Transducers | 58 |
| 5.3.3 | Pushdown Automata | 59 |
| 5.3.4 | Turing Machines | 60 |
| 5.4 | System Implementation Details | 62 |
| 6 | Genetic Programming Systems for Automata | 63 |
| 6.1 | Introduction | 63 |
| 6.2 | Finite Acceptors | 63 |
| 6.2.1 | Deterministic Finite Acceptors | 63 |
| 6.2.1.1 | Representation of an Individual | 63 |
| 6.2.1.2 | Genetic Programming Operators | 66 |
| 6.2.1.3 | Interpretation | 71 |
| 6.2.1.4 | GP Parameters for DFAs | 72 |
| 6.2.1.5 | Language Set | 73 |
| 6.2.2 | Nondeterministic Finite Acceptors (First Approach) | 75 |
| 6.2.2.1 | Representation of an Individual | 75 |
| 6.2.2.2 | Genetic Programming Operators | 76 |
| 6.2.2.3 | Interpretation | 76 |
| 6.2.2.4 | GP Parameters | 77 |
| 6.2.2.5 | Language set | 77 |
| 6.2.3 | Nondeterministic Finite Acceptors (Second Approach) | 79 |
| 6.2.3.1 | Representation of an Individual | 79 |
| 6.2.3.2 | Genetic Programming Operators | 80 |
| 6.2.3.3 | Interpretation | 80 |
| 6.2.3.4 | GP Parameters | 82 |
| 6.2.3.5 | Language set | 82 |
| 6.3 | Finite State Transducers | 84 |
| 6.3.1 | Representation of An Individual | 84 |
| 6.3.2 | Genetic Programming Operators | 86 |
| 6.3.3 | Interpretation | 89 |
| 6.3.4 | GP Parameters for FSTs | 90 |
| 6.3.5 | Language set | 91 |

| | | |
|----------|---|------------|
| 6.4 | Pushdown Automata | 92 |
| 6.4.1 | Representation of an Individual | 92 |
| 6.4.2 | Genetic Programming Operators | 94 |
| 6.4.3 | Interpretation | 97 |
| 6.4.4 | GP Parameters for PDAs | 98 |
| 6.4.5 | Language set | 99 |
| 6.5 | Turing Machines | 100 |
| 6.5.1 | Turing Machine Acceptors | 100 |
| 6.5.1.1 | Representation of an Individual | 100 |
| 6.5.1.2 | Genetic Programming Operators | 102 |
| 6.5.1.3 | Interpretation | 107 |
| 6.5.1.4 | GP Parameters for TM Acceptors | 108 |
| 6.5.1.5 | Language set | 109 |
| 6.5.2 | Turing Machine Transducers | 110 |
| 6.5.2.1 | Representation of an Individual | 110 |
| 6.5.2.2 | Genetic Programming Operators | 111 |
| 6.5.2.3 | Interpretation | 111 |
| 6.5.2.4 | GP Parameters for TM Transducers | 112 |
| 6.5.2.5 | Language set | 112 |
| 6.6 | Premature Convergence | 112 |
| 7 | Results and Discussion | 114 |
| 7.1 | Finite Acceptors | 114 |
| 7.1.1 | Deterministic Finite Acceptors | 114 |
| 7.1.2 | Modular Approach to Deterministic Finite Acceptors | 130 |
| 7.1.3 | Nondeterministic Finite Acceptors | 134 |
| 7.1.4 | Nondeterministic Finite Acceptors (2) | 144 |
| 7.1.5 | Comparison of Results | 154 |
| 7.2 | Finite State Transducers | 162 |
| 7.3 | Pushdown Automata | 168 |
| 7.4 | Turing Machines | 178 |
| 7.4.1 | Turing Machine Acceptors | 178 |
| 7.4.2 | Turing Machine Transducers | 190 |
| 8 | Conclusion | 200 |
| | Bibliography | 202 |
| A | System Results for Finite Acceptors | 207 |
| A.1 | Deterministic Finite Acceptors | 207 |
| A.2 | Non-deterministic Finite Acceptors (1) | 216 |
| A.3 | Non-deterministic Finite Acceptors (2) | 225 |
| A.4 | Modular Approach for Deterministic Finite Acceptors | 234 |
| A.4.1 | Modular Approach (a) | 234 |
| A.4.2 | Modular Approach (b) | 235 |
| A.4.3 | Modular Approach (c) | 237 |
| B | System Results for Finite State Transducers | 239 |
| C | System Results for Pushdown Automata | 243 |

| | | |
|----------|---|------------|
| D | System Results for Turing Machines | 251 |
| D.1 | Turing Machine Acceptors | 251 |
| D.2 | Turing Machine Transducers | 259 |
| D.3 | Turing Machine Transducers (2) | 263 |
| E | User manual for the GP system software | 267 |
| E.1 | Java version | 267 |
| E.2 | GP systems | 267 |
| E.2.1 | FAs | 268 |
| E.2.1.1 | An example run for FAs | 269 |
| E.2.2 | FSTs | 275 |
| E.2.2.1 | An example run for FSTs | 277 |
| E.2.3 | PDAs | 278 |
| E.2.3.1 | An example run for PDAs | 280 |
| E.2.4 | TMAAs | 281 |
| E.2.4.1 | An example run for TMAAs | 282 |
| E.2.5 | TMTs | 284 |
| E.2.5.1 | An example run for TMTs | 285 |

List of Figures

| | | |
|------|--|----|
| 2.1 | Transition diagram for a DFA | 6 |
| 2.2 | NFA for $L=\{w \mid w \text{ ends in } 10\}$ | 6 |
| 2.3 | Transition Diagram for a Mealy machine | 7 |
| 2.4 | Transition Diagram for a Moore machine | 8 |
| 2.5 | Transition Diagram for a PDA | 9 |
| 2.6 | Turing Machine Tape | 10 |
| 2.7 | Transition Diagram for a Turing Machine | 11 |
| 2.8 | The FA for $P='ABAA'$ | 13 |
| 2.9 | Logic Circuit | 13 |
| 2.10 | FST for the Logic Circuit | 14 |
| 2.11 | Game State Transition | 15 |
| 2.12 | State transition of a rocket projectile from Quake | 16 |
| 2.13 | FSM to control the life-cycle of a MID-let | 17 |
| 2.14 | Virtual FSM Flow Chart | 18 |
| | | |
| 3.1 | Example Representation using a Tree Structure | 21 |
| 3.2 | Example Representation using a Linear Structure | 22 |
| 3.3 | An example PADO Program [3] | 23 |
| 3.4 | Tree initialised using the full method with a maximum depth of 3 | 24 |
| 3.5 | Tree initialised using the grow method with a maximum depth of 3 | 24 |
| 3.6 | Example of Mutation for a Tree Structure | 28 |
| 3.7 | Example of Crossover for a Tree Structure | 29 |
| | | |
| 4.1 | Example Representation of an Individual | 32 |
| 4.2 | Example Translation of a DFA into an S-Expression | 33 |
| 4.3 | Parallel Node | 35 |
| 4.4 | Single state DFA – Parallel Node (Accept State) | 35 |
| 4.5 | Parallel Reject Node | 35 |
| 4.6 | Single state DFA – Parallel-Rej Node (Reject State) | 36 |
| 4.7 | Tree Representation of a DFA for $L=\{\text{even number of } b\text{'s}\}$ | 37 |
| 4.8 | Example growth of a DFA | 38 |
| 4.9 | Example DFA that cannot be evolved using Brave's method | 38 |
| 4.10 | An example of a READ state | 47 |
| 4.11 | An example of a POP State | 47 |
| 4.12 | Tree Representation for $a^n b^n$ | 48 |
| 4.13 | Program Representation for $a^n b^n$ | 48 |
| 4.14 | Transition Diagram for $a^n b^n$ | 49 |
| | | |
| 6.1 | Creation of a DFA | 64 |
| 6.2 | DFA for a^*b^* | 65 |
| 6.3 | Syntax for DFA | 65 |

| | | |
|------|--|-----|
| 6.4 | Resulting Subgraphs | 66 |
| 6.5 | Example of chosen parent individuals and crossover points (step 1 and 2) | 67 |
| 6.6 | Resulting subgraphs of the chosen parent graphs (step 3 and 4) | 68 |
| 6.7 | Resulting children graphs (step 5, 6 and 7) | 69 |
| 6.8 | Chosen parent and mutation point | 70 |
| 6.9 | Mutation - Remaining subgraph | 70 |
| 6.10 | The resulting child graph after mutation | 71 |
| 6.11 | Creation of an NFA1 | 75 |
| 6.12 | NFA1 for a^*b^* | 76 |
| 6.13 | Syntax for NFA1 | 76 |
| 6.14 | Creation of an NFA2 | 79 |
| 6.15 | NFA2 for a^*b^* | 80 |
| 6.16 | Syntax for NFA2 | 80 |
| 6.17 | Creation of an FST | 85 |
| 6.18 | Example Finite State Transducer | 85 |
| 6.19 | Syntax for FST | 85 |
| 6.20 | Example of selected Parents for Crossover | 86 |
| 6.21 | Example of the resulting subgraphs | 87 |
| 6.22 | Example of the offspring created when applying crossover | 88 |
| 6.23 | Example of a selected mutation point in a Parent | 88 |
| 6.24 | Example of the subgraph | 89 |
| 6.25 | Example of the offspring created when applying mutation | 89 |
| 6.26 | Creation of a PDA | 93 |
| 6.27 | PDA for $a^n b^n$ | 93 |
| 6.28 | Syntax for PDA | 94 |
| 6.29 | Chosen parents for Crossover | 94 |
| 6.30 | Resulting Subgraphs | 95 |
| 6.31 | Resulting children after performing Crossover | 96 |
| 6.32 | Example Mutation | 97 |
| 6.33 | Code Example for PDAs | 98 |
| 6.34 | Creation of a Turing Machine | 101 |
| 6.35 | Example Turing Machine Acceptor | 102 |
| 6.36 | Syntax for TM Acceptors | 102 |
| 6.37 | Chosen parents for Crossover | 103 |
| 6.38 | Resulting Subgraphs | 104 |
| 6.39 | Resulting children after performing Crossover | 105 |
| 6.40 | Example Mutation | 106 |
| 6.41 | Example Interpretation (1) | 107 |
| 6.42 | Example Turing Machine Transducer | 110 |
| 6.43 | Example Output Tape for TMT1 | 110 |
| 6.44 | Example Output Tape for TMT2 | 111 |
| 6.45 | Syntax for TM Transducers | 111 |
| 7.1 | Example Solutions for L1 (DFA) | 115 |
| 7.2 | Example Solution for L2 (DFA) | 115 |
| 7.3 | Example Solution for L3 (DFA) | 116 |
| 7.4 | Example Solution for L4 (DFA) | 117 |
| 7.5 | Example Solution for L5 (DFA) | 117 |
| 7.6 | Example Solution for L6 (DFA) | 118 |
| 7.7 | Example Solution for L7 (DFA) | 118 |
| 7.8 | Example Solution for L8 (DFA) | 119 |
| 7.9 | Example Solution for L9 (DFA) | 120 |

| | | |
|------|---|-----|
| 7.10 | Minimized DFA for L9 | 121 |
| 7.11 | Example Solution for L10 (DFA) | 122 |
| 7.12 | Minimized DFA for L10 | 122 |
| 7.13 | Example Solution for L11 (DFA) | 123 |
| 7.14 | Minimized DFA for L11 | 123 |
| 7.15 | Example Solution for L12 (DFA) | 124 |
| 7.16 | Example Solution for L13 (DFA) | 124 |
| 7.17 | Example Solution for L14 (DFA) | 125 |
| 7.18 | Example Solution for L15 (DFA) | 126 |
| 7.19 | Minimized DFA for L15 | 127 |
| 7.20 | Graph for L9 showing the convergence for all seed values | 129 |
| 7.21 | Graph for L15 showing the convergence for all seed values | 129 |
| 7.22 | Example of Modular Components | 131 |
| 7.23 | Example Solutions for L1 (NFA1) | 134 |
| 7.24 | Example Solutions for L2 (NFA1) | 135 |
| 7.25 | Example Solutions for L3 (NFA1) | 136 |
| 7.26 | Example Solutions for L4 (NFA1) | 136 |
| 7.27 | Example Solution for L5 (NFA1) | 137 |
| 7.28 | Example Solution for L6 (NFA1) | 137 |
| 7.29 | Example Solution for L7 (NFA1) | 138 |
| 7.30 | Example Solution for L8 (NFA1) | 138 |
| 7.31 | Example Solution for L9 (NFA1) | 139 |
| 7.32 | Example Solution for L10 (NFA1) | 139 |
| 7.33 | Example Solution for L11 (NFA1) | 140 |
| 7.34 | Example Solution for L12 (NFA1) | 140 |
| 7.35 | Example Solution for L13 (NFA1) | 141 |
| 7.36 | Example Solution for L14 (NFA1) | 141 |
| 7.37 | Example Solution for L15 (NFA1) | 142 |
| 7.38 | Example Solutions for L1 (NFA2) | 144 |
| 7.39 | Example Solution for L2 (NFA2) | 144 |
| 7.40 | Example Solution for L3 (NFA2) | 145 |
| 7.41 | Example Solution for L4 (NFA2) | 145 |
| 7.42 | Example Solution for L5 (NFA2) | 146 |
| 7.43 | Example Solution for L6 (NFA2) | 146 |
| 7.44 | Example Solution for L7 (NFA2) | 147 |
| 7.45 | Example Solution for L8 (NFA2) | 147 |
| 7.46 | Example Solution for L9 (NFA2) | 148 |
| 7.47 | Example Solution for L10 (NFA2) | 148 |
| 7.48 | Example Solution for L11 (NFA2) | 149 |
| 7.49 | Minimized NFA for L11 | 149 |
| 7.50 | Example Solution for L12 (NFA2) | 150 |
| 7.51 | Example Solution for L13 (NFA2) | 150 |
| 7.52 | Example Solution for L14 (NFA2) | 151 |
| 7.53 | Example Solution for L15 (NFA2) | 151 |
| 7.54 | Example Solution for L1 (FST) | 162 |
| 7.55 | Example Solution for L2 (FST) | 163 |
| 7.56 | Example Solution for L3 (FST) | 163 |
| 7.57 | Example Solution for L4 (FST) | 164 |
| 7.58 | Example Solution for L5 (FST) | 164 |
| 7.59 | Example Solution for L6 (FST) | 165 |
| 7.60 | Example Solution for L1 (PDA) | 168 |
| 7.61 | Example Solution for L2 (PDA) | 169 |

| | | |
|------|--|-----|
| 7.62 | Example Solution for L3 (PDA) | 169 |
| 7.63 | Example Solution for L4 (PDA) | 170 |
| 7.64 | Example Solution for L5 (PDA) | 170 |
| 7.65 | Example Solution for L6 (PDA) | 171 |
| 7.66 | Example Solution for L7 (PDA) | 171 |
| 7.67 | Example Solution for L8 (PDA) | 172 |
| 7.68 | Example Solution for L9 (PDA) | 172 |
| 7.69 | Example Solution for L10 (PDA) | 173 |
| 7.70 | Example Solution for L11 (PDA) | 173 |
| 7.71 | Example Solution for L1 (TMA) | 178 |
| 7.72 | Example Solution for L2 (TMA) | 179 |
| 7.73 | Minimized Solution for L2 (TMA) | 180 |
| 7.74 | Example Solution for L3 (TMA) | 180 |
| 7.75 | Example Solution for L4 (TMA) | 181 |
| 7.76 | Example Solution for L5 (TMA) | 182 |
| 7.77 | Example Solution for L6 (TMA) | 182 |
| 7.78 | Example Solution for L7 (TMA) | 183 |
| 7.79 | Example Solution for L8 (TMA) | 184 |
| 7.80 | Example Solution for L9 (TMA) | 185 |
| 7.81 | Minimized Solution for L9 (TMA) | 185 |
| 7.82 | Example Solution for L1 (TMT1) | 190 |
| 7.83 | Example Solution for L1 (TMT2) | 191 |
| 7.84 | Example Solution for L2 (TMT1) | 191 |
| 7.85 | Example Solution for L2 (TMT2) | 192 |
| 7.86 | Example Solution for L3 (TMT1) | 192 |
| 7.87 | Example Solution for L3 (TMT2) | 193 |
| 7.88 | Example Solution for L4 (TMT1) | 193 |
| 7.89 | Example Solution for L5 (TMT1) | 194 |
| 7.90 | Example Solution for L5 (TMT2) | 194 |
| 7.91 | Example Solution Output for L6 (TMT1) | 195 |
| 7.92 | Example Solution for L6 (TMT1) | 195 |
| 7.93 | Example Solution for L6 (TMT2) | 195 |
| | | |
| E.1 | Folder Structure | 267 |
| E.2 | Executable JAR File | 268 |
| E.3 | FAs Application Screenshot | 268 |
| E.4 | Example run for deterministic FAs for L3 | 270 |
| E.5 | Results obtained for L3 using seed 1010 (Deterministic FAs) | 271 |
| E.6 | Example run for deterministic FAs for L15 using the Modular approach | 272 |
| E.7 | Results obtained for L15 using seed 4021 (Modular Approach) | 273 |
| E.8 | Example run for deterministic FAs for L6 (NFA2) | 274 |
| E.9 | Results obtained for L6 using seed 1000 (NFA2) | 275 |
| E.10 | FSTs Application Screenshot | 276 |
| E.11 | Example run for deterministic FSTs for L1 | 277 |
| E.12 | Results obtained for L1 using seed 3908 (FSTs) | 278 |
| E.13 | PDA's Application Screenshot | 279 |
| E.14 | Example run for deterministic PDAs for L1 | 280 |
| E.15 | Results obtained for L1 using seed 5678 (PDAs) | 281 |
| E.16 | TMA's Application Screenshot | 281 |
| E.17 | Example run for deterministic TMAs for L3 | 283 |
| E.18 | Results obtained for L3 using seed 1500 (TMAs) | 283 |
| E.19 | TMT's Application Screenshot | 284 |

| | |
|---|-----|
| E.20 Example run for deterministic TMTs for L3 | 285 |
| E.21 Results obtained for L2 using seed 2000 (TMTs) | 286 |

List of Tables

| | | |
|------|--|-----|
| 2.1 | Transition Table | 5 |
| 2.2 | Chomsky Hierarchy | 12 |
| 2.3 | Transitions for the FST | 15 |
| 4.1 | Genetic Algorithm Parameters for the GIG method | 31 |
| 4.2 | Dupont’s benchmark languages | 31 |
| 4.3 | Genetic Programming Parameters for Dunay’s system | 34 |
| 4.4 | Genetic Programming Parameters for Brave’s system | 34 |
| 4.5 | Grid showing the neighbourhood of ‘e’ | 39 |
| 4.6 | Genetic Algorithm Parameters used by Luke et al. [33] | 40 |
| 4.7 | Genetic Algorithm Parameters for the GARI method | 41 |
| 4.8 | Dunay’s benchmark languages | 43 |
| 4.9 | GA Parameters used by Huijsen | 44 |
| 4.10 | Results of Huijsen’s experiment | 44 |
| 4.11 | Lankhorst’s benchmark languages | 44 |
| 4.12 | Genetic Programming Parameters for Lankhort’s system | 45 |
| 4.13 | Genetic Programming Parameters for Zomorodian’s system | 46 |
| 4.14 | Genetic Algorithm Parameters for Tanomaru’s system | 50 |
| 4.15 | Tanomaru’s benchmark languages | 51 |
| 4.16 | Transition Table used in Experiment 1 | 52 |
| 4.17 | Transition Table used in Experiment 2 | 53 |
| 4.18 | Genetic Programming Parameters for Machado’s system | 55 |
| 5.1 | Language Set for FAs | 58 |
| 5.2 | Language Set for FSTs | 59 |
| 5.3 | Language Set for PDAs | 60 |
| 5.4 | Language Set for TM Acceptors | 61 |
| 5.5 | Language Set for TM Transducers | 62 |
| 6.1 | GP Parameters used for the induction of DFAs | 73 |
| 6.2 | Number of fitness cases used for inducing DFAs | 74 |
| 6.3 | GP Parameters used for the induction of NFAs(First Approach) | 77 |
| 6.4 | Number of fitness cases used for inducing NFAs (First Approach) | 78 |
| 6.5 | Number of fitness cases used for inducing NFAs (Second Approach) | 83 |
| 6.6 | GP Parameters used for the induction of FSTs | 91 |
| 6.7 | Number of fitness cases used for inducing FSTs | 91 |
| 6.8 | GP Parameters used for the induction of PDAs | 98 |
| 6.9 | Number of fitness cases used for inducing PDAs | 99 |
| 6.10 | Interpretation of ‘aabb’ | 108 |
| 6.11 | Interpretation of ‘abab’ | 108 |
| 6.12 | GP Parameters used for the induction of TMAs | 109 |

| | | |
|------|--|-----|
| 6.13 | Number of fitness cases used for inducing TM Acceptors | 109 |
| 6.14 | GP Parameters used for the induction of TMTs | 112 |
| 6.15 | Number of fitness cases used for inducing TM Transducers | 112 |
| 7.1 | Success Rates for DFA Simulations | 127 |
| 7.2 | Multiple iterations required for DFA Simulations | 128 |
| 7.3 | Example Fitness Cases | 130 |
| 7.4 | Success Rates for DFA Simulations – Modular Approach (a) | 132 |
| 7.5 | Success Rates for DFA Simulations – Modular Approach (b) | 132 |
| 7.6 | Success Rates for DFA Simulations – Modular Approach (c) | 132 |
| 7.7 | Success Rates for NFA1 Simulations | 142 |
| 7.8 | Multiple iterations required for NFA1 Simulations | 143 |
| 7.9 | Success Rates for NFA2 Simulations | 152 |
| 7.10 | Multiple iterations required for NFA2 Simulations | 153 |
| 7.11 | Comparison to Human Generated Solutions | 158 |
| 7.12 | Number of redundant nodes | 159 |
| 7.13 | Comparison with previous work | 159 |
| 7.14 | Comparison with Luke et al. (Experiment 1) | 160 |
| 7.15 | Comparison with Dupont’s work | 161 |
| 7.16 | Success Rates for FST Simulations over 10 runs | 165 |
| 7.17 | Comparison of Solutions | 166 |
| 7.18 | Number of redundant nodes | 167 |
| 7.19 | Success Rates for PDA Simulations | 174 |
| 7.20 | Multiple iterations required for PDA Simulations | 174 |
| 7.21 | Comparison with “Human Generated” Solutions (PDAs) | 176 |
| 7.22 | Comparison with Lankhorst’s work | 177 |
| 7.23 | Comparison with Dunay’s work | 177 |
| 7.24 | Success Rates for TMA Simulations | 186 |
| 7.25 | Success Rates for TMA Simulations – Incremental Approach | 186 |
| 7.26 | Multiple iterations required for TMA Simulations | 187 |
| 7.27 | “Human Generated” Solutions (TMAs) | 189 |
| 7.28 | Comparison with Tanomaru’s work | 189 |
| 7.29 | Success Rates for TMT1 Simulations | 196 |
| 7.30 | Multiple iterations required for TMT1 Simulations | 196 |
| 7.31 | Success Rates for TMT2 Simulations | 196 |
| 7.32 | Multiple iterations required for TMT2 Simulations | 197 |
| 7.33 | “Human Generated” Solutions (TMTs) | 198 |
| A.1 | Standard Operator Results for L1 (DFA) | 207 |
| A.2 | Standard Operator Results for L2 (DFA) | 207 |
| A.3 | Non-destructive Results for L2 (DFA) | 208 |
| A.4 | Standard Operator Results for L3 (DFA) | 208 |
| A.5 | Non-destructive Results for L3 (DFA) | 208 |
| A.6 | Standard Operator Results for L4 (DFA) | 209 |
| A.7 | Non-destructive Results for L4 (DFA) | 209 |
| A.8 | Standard Operator Results for L5 (DFA) | 209 |
| A.9 | Non-destructive Results for L5 (DFA) | 210 |
| A.10 | Standard Operator Results for L6 (DFA) | 210 |
| A.11 | Non-destructive Results for L6 (DFA) | 210 |
| A.12 | Standard Operator Results for L7 (DFA) | 211 |
| A.13 | Non-destructive Results for L7 (DFA) | 211 |
| A.14 | Standard Operator Results for L8 (DFA) | 211 |

| | |
|---|-----|
| A.15 Standard Operator Results for L9 (DFA) | 212 |
| A.16 Non-destructive Results for L9 (DFA) | 212 |
| A.17 Standard Operator Results for L10 (DFA) | 212 |
| A.18 Non-destructive Results for L10 (DFA) | 213 |
| A.19 Standard Operator Results for L11 (DFA) | 213 |
| A.20 Non-destructive Results for L11 (DFA) | 213 |
| A.21 Standard Operator Results for L12 (DFA) | 214 |
| A.22 Non-destructive Results for L12 (DFA) | 214 |
| A.23 Standard Operator Results for L13 (DFA) | 214 |
| A.24 Standard Operator Results for L14 (DFA) | 215 |
| A.25 Non-destructive Results for L14 (DFA) | 215 |
| A.26 Standard Operator Results for L15 (DFA) | 215 |
| A.27 Non-destructive Results for L15 (DFA) | 216 |
| A.28 Standard Operator Results for L1 (NFA1) | 216 |
| A.29 Standard Operator Results for L2 (NFA1) | 216 |
| A.30 Standard Operator Results for L3 (NFA1) | 217 |
| A.31 Non-destructive Results for L3 (NFA1) | 217 |
| A.32 Standard Operator Results for L4 (NFA1) | 217 |
| A.33 Non-destructive Results for L4 (NFA1) | 218 |
| A.34 Standard Operator Results for L5 (NFA1) | 218 |
| A.35 Non-destructive Results for L5 (NFA1) | 218 |
| A.36 Standard Operator Results for L6 (NFA1) | 219 |
| A.37 Non-destructive Results for L6 (NFA1) | 219 |
| A.38 Standard Operator Results for L7 (NFA1) | 219 |
| A.39 Non-destructive Results for L7 (NFA1) | 220 |
| A.40 Standard Operator Results for L8 (NFA1) | 220 |
| A.41 Standard Operator Results for L9 (NFA1) | 220 |
| A.42 Non-destructive Results for L9 (NFA1) | 221 |
| A.43 Standard Operator Results for L10 (NFA1) | 221 |
| A.44 Non-destructive Results for L10 (NFA1) | 221 |
| A.45 Standard Operator Results for L11 (NFA1) | 222 |
| A.46 Non-destructive Results for L11 (NFA1) | 222 |
| A.47 Standard Operator Results for L12 (NFA1) | 222 |
| A.48 Non-destructive Results for L12 (NFA1) | 223 |
| A.49 Standard Operator Results for L13 (NFA1) | 223 |
| A.50 Non-destructive Results for L13 (NFA1) | 223 |
| A.51 Standard Operator Results for L14 (NFA1) | 224 |
| A.52 Non-destructive Results for L14 (NFA1) | 224 |
| A.53 Standard Operator Results for L15 (NFA1) | 224 |
| A.54 Non-destructive Results for L15 (NFA1) | 225 |
| A.55 Standard Operator Results for L1 (NFA2) | 225 |
| A.56 Standard Operator Results for L2 (NFA2) | 225 |
| A.57 Standard Operator Results for L3 (NFA2) | 226 |
| A.58 Non-destructive Results for L3 (NFA2) | 226 |
| A.59 Standard Operator Results for L4 (NFA2) | 226 |
| A.60 Non-destructive Results for L4 (NFA2) | 227 |
| A.61 Standard Operator Results for L5 (NFA2) | 227 |
| A.62 Non-destructive Results for L5 (NFA2) | 227 |
| A.63 Standard Operator Results for L6 (NFA2) | 228 |
| A.64 Non-destructive Results for L6 (NFA2) | 228 |
| A.65 Standard Operator Results for L7 (NFA2) | 228 |
| A.66 Non-destructive Results for L7 (NFA2) | 229 |

| | | |
|------|--|-----|
| A.67 | Standard Operator Results for L8 (NFA2) | 229 |
| A.68 | Standard Operator Results for L9 (NFA2) | 229 |
| A.69 | Non-destructive Results for L9 (NFA2) | 230 |
| A.70 | Standard Operator Results for L10 (NFA2) | 230 |
| A.71 | Non-destructive Results for L10 (NFA2) | 230 |
| A.72 | Standard Operator Results for L11 (NFA2) | 231 |
| A.73 | Non-destructive Results for L11 (NFA2) | 231 |
| A.74 | Standard Operator Results for L12 (NFA2) | 231 |
| A.75 | Non-destructive Results for L12 (NFA2) | 232 |
| A.76 | Standard Operator Results for L13 (NFA2) | 232 |
| A.77 | Non-destructive Results for L13 (NFA2) | 232 |
| A.78 | Standard Operator Results for L14 (NFA2) | 233 |
| A.79 | Non-destructive Results for L14 (NFA2) | 233 |
| A.80 | Standard Operator Results for L15 (NFA2) | 233 |
| A.81 | Non-destructive Results for L15 (NFA2) | 234 |
| A.82 | Standard Operator Results for L9 (DFA) - Modular Approach (a) | 234 |
| A.83 | Non-destructive Results for L9 (DFA) - Modular Approach (a) | 234 |
| A.84 | Standard Operator Results for L15 (DFA) - Modular Approach (a) | 235 |
| A.85 | Non-destructive Results for L15 (DFA) - Modular Approach (a) | 235 |
| A.86 | Standard Operator Results for L9 (DFA) - Modular Approach (b) | 235 |
| A.87 | Non-destructive Results for L9 (DFA) - Modular Approach (b) | 236 |
| A.88 | Standard Operator Results for L15 (DFA) - Modular Approach (b) | 236 |
| A.89 | Non-destructive Results for L15 (DFA) - Modular Approach (b) | 236 |
| A.90 | Standard Operator Results for L9 (DFA) - Modular Approach (c) | 237 |
| A.91 | Non-destructive Results for L9 (DFA) - Modular Approach (c) | 237 |
| A.92 | Standard Operator Results for L15 (DFA) - Modular Approach (c) | 237 |
| A.93 | Non-destructive Results for L15 (DFA) - Modular Approach (c) | 238 |
| | | |
| B.1 | Standard Operator Results for L1 (FST) | 239 |
| B.2 | Non-destructive Results for L1 (FST) | 239 |
| B.3 | Standard Operator Results for L2 (FST) | 240 |
| B.4 | Non-destructive Results for L2 (FST) | 240 |
| B.5 | Standard Operator Results for L3 (FST) | 240 |
| B.6 | Standard Operator Results for L4 (FST) | 241 |
| B.7 | Non-destructive Results for L4 (FST) | 241 |
| B.8 | Standard Operator Results for L5 (FST) | 241 |
| B.9 | Standard Operator Results for L6 (FST) | 242 |
| | | |
| C.1 | Standard Operator Results for L1 (PDA) | 243 |
| C.2 | Non-destructive Results for L1 (PDA) | 243 |
| C.3 | Standard Operator Results for L2 (PDA) | 244 |
| C.4 | Non-destructive Results for L2 (PDA) | 244 |
| C.5 | Standard Operator Results for L3 (PDA) | 244 |
| C.6 | Non-destructive Results for L3 (PDA) | 245 |
| C.7 | Standard Operator Results for L4 (PDA) | 245 |
| C.8 | Non-destructive Results for L4 (PDA) | 245 |
| C.9 | Standard Operator Results for L5 (PDA) | 246 |
| C.10 | Non-destructive Results for L5 (PDA) | 246 |
| C.11 | Standard Operator Results for L6 (PDA) | 246 |
| C.12 | Non-destructive Results for L6 (PDA) | 247 |
| C.13 | Standard Operator Results for L7 (PDA) | 247 |
| C.14 | Non-destructive Results for L7 (PDA) | 247 |

| | |
|---|-----|
| C.15 Standard Operator Results for L8 (PDA) | 248 |
| C.16 Non-destructive Results for L8 (PDA) | 248 |
| C.17 Standard Operator Results for L9 (PDA) | 248 |
| C.18 Non-destructive Results for L9 (PDA) | 249 |
| C.19 Standard Operator Results for L10 (PDA) | 249 |
| C.20 Non-destructive Results for L10 (PDA) | 249 |
| C.21 Standard Operator Results for L11 (PDA) | 250 |
| | |
| D.1 Standard Operator Results for L1 (TMA) | 251 |
| D.2 Non-destructive Results for L1 (TMA) | 251 |
| D.3 Standard Operator Results for L1 (TMA) – Incremental | 252 |
| D.4 Non-destructive Results for L1 (TMA) – Incremental | 252 |
| D.5 Standard Operator Results for L2 (TMA) | 252 |
| D.6 Non-destructive Results for L2 (TMA) | 253 |
| D.7 Non-destructive Results for L2 (TMA) – Incremental | 253 |
| D.8 Standard Operator Results for L3 (TMA) | 253 |
| D.9 Non-destructive Results for L3 (TMA) | 254 |
| D.10 Standard Operator Results for L4 (TMA) | 254 |
| D.11 Non-destructive Results for L4 (TMA) | 254 |
| D.12 Standard Operator Results for L4 (TMA) – Incremental | 255 |
| D.13 Non-destructive Results for L4 (TMA) – Incremental | 255 |
| D.14 Standard Operator Results for L5 (TMA) | 255 |
| D.15 Non-destructive Results for L5 (TMA) | 256 |
| D.16 Standard Operator Results for L6 (TMA) | 256 |
| D.17 Non-destructive Results for L6 (TMA) | 256 |
| D.18 Standard Operator Results for L7 (TMA) | 257 |
| D.19 Non-destructive Results for L7 (TMA) | 257 |
| D.20 Standard Operator Results for L8 (TMA) | 257 |
| D.21 Non-destructive Results for L8 (TMA) | 258 |
| D.22 Standard Operator Results for L9 (TMA) | 258 |
| D.23 Non-destructive Results for L9 (TMA) | 258 |
| D.24 Standard Operator Results for L1 (TMT1) | 259 |
| D.25 Non-destructive Results for L1 (TMT1) | 259 |
| D.26 Standard Operator Results for L2 (TMT1) | 259 |
| D.27 Non-destructive Results for L2 (TMT1) | 260 |
| D.28 Standard Operator Results for L3 (TMT1) | 260 |
| D.29 Non-destructive Results for L3 (TMT1) | 260 |
| D.30 Standard Operator Results for L4 (TMT1) | 261 |
| D.31 Non-destructive Results for L4 (TMT1) | 261 |
| D.32 Standard Operator Results for L5 (TMT1) | 261 |
| D.33 Non-destructive Results for L5 (TMT1) | 262 |
| D.34 Standard Operator Results for L6 (TMT1) | 262 |
| D.35 Non-destructive Results for L6 (TMT1) | 262 |
| D.36 Standard Operator Results for L1 (TMT2) | 263 |
| D.37 Non-destructive Results for L1 (TMT2) | 263 |
| D.38 Standard Operator Results for L2 (TMT2) | 263 |
| D.39 Non-destructive Results for L2 (TMT2) | 264 |
| D.40 Standard Operator Results for L3 (TMT2) | 264 |
| D.41 Non-destructive Results for L3 (TMT2) | 264 |
| D.42 Standard Operator Results for L4 (TMT2) | 265 |
| D.43 Non-destructive Results for L4 (TMT2) | 265 |
| D.44 Standard Operator Results for L5 (TMT2) | 265 |

| | |
|--|-----|
| D.45 Non-destructive Results for L5 (TMT2) | 266 |
| D.46 Standard Operator Results for L6 (TMT2) | 266 |
| D.47 Non-destructive Results for L6 (TMT2) | 266 |

List of Algorithms

| | | |
|----|---|----|
| 1 | Genetic Programming Algorithm | 20 |
| 2 | Crossover Algorithm | 66 |
| 3 | Mutation Algorithm | 69 |
| 4 | DFA: isAccepted | 72 |
| 5 | DFA: getFitness | 72 |
| 6 | NFA: isAccepted | 77 |
| 7 | NFA2: isAccepted | 81 |
| 8 | NFA2: getFitness | 82 |
| 9 | FST: getOutputString | 90 |
| 10 | FST: getFitness | 90 |

Chapter 1

Introduction

1.1 Purpose of Study

Automata theory has proved to be an important field in Computer Science, contributing to several areas including compiler construction, natural language processing, networking and game programming. The ability to automatically induce automata has therefore been an object of the Computer Science community for a number of years. There has been considerable work done using evolutionary algorithms as a means to induce automata, however most of the work has concentrated on applying genetic algorithms to the problem domain. This study evaluates genetic programming (GP) as a means of generating the various classes of automata, including finite acceptors, pushdown automata, finite state transducers and Turing machines. Genetic programming is chosen as it is flexible and has solved problems from a wide range of domains. Genetic programming is also a method that is appropriate for difficult search and optimization problems.

1.2 Objectives

The main objectives of this thesis are:

- To test the effectiveness of genetic programming as a method of automatically generating automata.
- To create a system for each class of automaton identifying the representation and the GP parameters.
- To use a direct representation for each automaton.
- To test each system on languages commonly found in literature and to present the results obtained from these tests.

1.3 Scope of Study

This thesis investigates the ability to induce automata using the methods of GP when given a finite set of strings. For automata classed as acceptors, the idea is to generate automata given a set of positive and negative sample sentences, where a positive sentence is defined to be a sentence accepted by the automata and a negative sentence will be rejected by the automata for the specific language. For transducers i.e., automata that generate output, the automata are generated using a finite set of input and output sentences. A GP system is created for each class of automaton. The automata investigated in this thesis includes deterministic finite acceptors, nondeterministic finite acceptors, finite state transducers, nondeterministic pushdown automata, Turing machine acceptors and Turing machine transducers. For each system the representation of the individuals, the methods for evaluation, and the GP parameters are identified. Each GP system created for a specific type of automaton was tested on a set of benchmark languages.

1.4 Contributions to the Study

This thesis makes 2 main contributions:

- It shows that genetic programming provides an effectual method to automatically induce the various types of automata. Furthermore, GP systems for all classes of automata defined in the Chomsky Hierarchy are created with solutions being generated for each type of automaton.
- A new method of evolving automata is introduced where the individuals are represented using the direct representation of an automaton. The seldom used graph structure is utilized where each graph is the actual transition graph of the automaton. The genetic operators using this new graph representation are also defined. Using a different representation has an effect on both the search space and the fitness landscape and thus the overall success of the system.

1.5 List of publications

The following is a list of publications resulting from this study:

- Pillay N., Naidoo A., An Investigation into the Automatic Generation of Solutions to Problems in an Intelligent Tutoring System for Finite Automata, full research paper in the Proceedings of SACLA 2006, Van Belle J. and Brown I. (eds.), pp. 84 - 93, ISBN: 0-620-36150-6 (Award for Best Full Research Paper).
- Naidoo A., Pillay N., Inducing Finite Transducers Using Genetic Programming, in M. Ebner et al. (eds.), EuroGP 2007, Lecture Notes in Computer Science 4445, pp. 371 - 380, Springer-Verlag Berlin Heidelberg, 2007.
- Naidoo A., Pillay N., Evolving Pushdown Automata, in Barnard L. and Botha R. A.(eds), Riding the Wave of Technology, Proceedings of SAICSIT 2007, pp. 83-90, ACM International Conference Proceedings Series, 2007.

- Naidoo A., Pillay N., Evolving Finite Acceptors for Regular Languages. In Neves J., Santos M.F., Machado J.M. (eds.): *New Trends in Artificial Intelligence, APPIA 2007*, pp. 193 - 206, December 2007.
- Naidoo A., Pillay N., Using Genetic Programming for Turing Machine Induction. In O'Neill M. et al. (Eds.): *EuroGP 2008, LNCS 4972*, pp. 350-361, 2008.

1.6 Thesis Layout

Following this introductory chapter, Chapter 2 presents an overview of automata. The definitions of the different classes of automata are explained. For each type of automata, the diagrammatic representation, that being a directed graph, is presented. The Chomsky Hierarchy is explained and a brief overview of some of the areas in Computer Science that makes use of automata are given.

Chapter 3 gives an overview of genetic programming. The algorithm, representation and operators are described. A brief overview is given to introduce genetic programming followed by a detailed description of genetic programming.

Chapter 4 analyses the previous studies done applying evolutionary algorithms to automata induction. The evolutionary parameters are identified as well as the results obtained.

Chapter 5 explains the research methodology used and the experimental setup used to apply genetic programming to induce automata. This chapter also gives the languages that each system was tested on.

Chapter 6 presents the GP systems developed to induce automata. GP systems are presented for finite acceptors, finite state transducers, pushdown automata and Turing machines. This chapter shows the representation of the individuals of the population for each type of automaton. The GP parameters are identified. The GP operators and the interpretation used by each system is explained.

Chapter 7 presents the results of the various systems discussed in Chapter Six. This chapter also discusses the results obtained and does a comparison with results obtained in the previous work done in this area.

Chapter 8 states the conclusion and lists the contributions made.

Chapter 2

An Overview of Automata

2.1 Introduction

An automaton can be described as a mathematical model of a system [21], consisting of a set of states and transitions. This chapter gives an overview of the various classes of automata.

2.2 Classes of Automata

2.2.1 Finite Acceptors

A finite acceptor (FA) is the simplest model of computation which describes a class of languages called ‘regular’. A finite acceptor consists of a finite set of states which are classed to be either accepting (final states) or not accepting. A string of input symbols are taken in by the finite acceptor. If all the input is processed and the current state is an accepting state then the input is accepted; otherwise the input is rejected. The machine is described as defining a language, which would contain all words accepted by the machine. FAs are limited in the languages they can recognize as they do not make use of memory. This results in FAs not having the ability to count. As a result certain simple languages cannot be recognized by an FA e.g., $a^n b^n$ - some number of a’s followed by an equal number of b’s.

2.2.1.1 Deterministic Finite Acceptors

For deterministic finite acceptors (DFA) there is only one transition out of a state for each input symbol of the alphabet (Σ) i.e., $\delta(q,a)$ is unique [9, 21]. This implies that for an input w , the execution is predictable.

Formal Definition [21]

A DFA is defined as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where,

- Q is a set of finite states.
- Σ is the set of input symbols (alphabet).

- δ is the transition function where $\delta : Q \times \Sigma \rightarrow Q$ e.g., $\delta(q,a)=p$ gives the transition from state q on the input a .
- $q_0 \in Q$ is the start state.
- $F \subseteq Q$ is the set of accepting states.

Let M be a DFA such that $M = (Q, \Sigma, \delta, q_0, F)$. String x is said to be accepted by M if $\delta^*(q_0, x) = p$ for some p in F .

2.2.1.2 Nondeterministic Finite Acceptors

Nondeterministic acceptors (NFA) can have multiple (or no) transitions out of a state for each input symbol of the alphabet [52]. Nondeterminism therefore implies that there may not be a unique execution trace (as in DFAs). There are multiple paths of possible executions. For NFAs a string is accepted if there exists at least one execution path that ends in a final state, whereas a string is rejected if all possible execution paths end in a non-final state. NFAs also allow for ϵ -transitions (transitions which do not read a character of the input string).

For every NFA there is an equivalent DFA which accepts exactly the same language L . It is therefore possible to convert any NFA to a DFA. NFAs are necessary as it is easier to construct whereas DFAs can become very complex. DFAs on the other hand are easier to implement and interpret on a computer.

Formal Definition [21]

An NFA is defined as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where,

- Q is a set of finite states.
- Σ is the set of input symbols (alphabet).
- δ is the transition function where $\delta : Q \times (\Sigma \cup \epsilon) \rightarrow \mathcal{P}(Q)$ where $\mathcal{P}(Q)$ is the power set of Q and ϵ is the empty string.
- $q_0 \in Q$ is the start state.
- $F \subseteq Q$ is the set of accepting states.

Representation

A finite acceptor can be represented as a transition diagram or a transition graph. Table 2.1 shows an example transition table. State q_0 is the start state and State q_2 is the accept state.

| | | |
|----------|-------|-------|
| δ | 0 | 1 |
| q_0 | q_0 | q_2 |
| q_1 | q_1 | q_1 |
| q_2 | q_1 | q_1 |

Table 2.1: Transition Table

Figure 2.1 shows a transition diagram for the DFA which accepts the regular language (0^*1) .

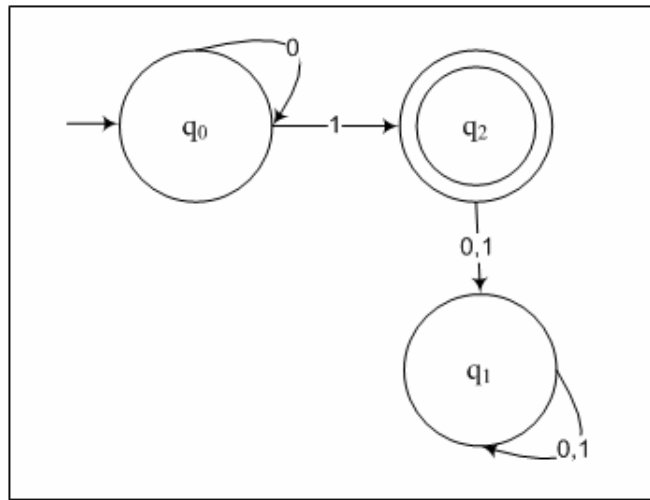


Figure 2.1: Transition diagram for a DFA

A transition diagram for a DFA is a directed graph. The edges represent the transition from one state to another. An accept state is usually represented by a double circle (q_2 in Figure 2.1). The start (initial) state is usually indicated by an arrow (q_0 in Figure 2.1).

Figure 2.2 below shows an example of a transition diagram of an NFA.

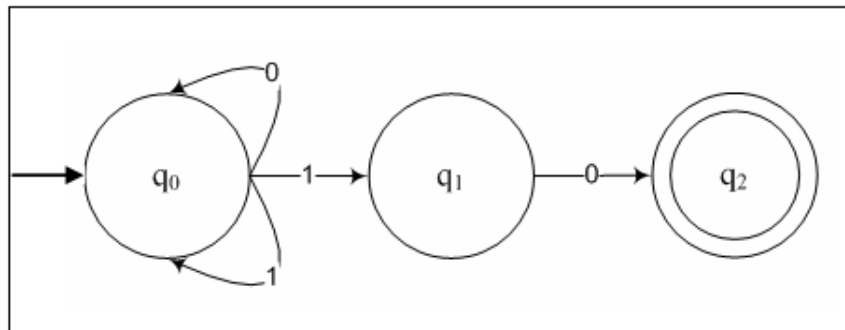


Figure 2.2: NFA for $L=\{w \mid w \text{ ends in } 10\}$

2.2.2 Finite State Transducers

Finite state transducers (FSTs) are finite automata with output. An FST differs from an FA in that it has no final states, the states are not classified as accept or reject states and FSTs cannot be

nondeterministic. There are two types of FSTs: Mealy machines and Moore machines named after G. H. Mealy and Edward F. Moore respectively, who were state machine pioneers.

2.2.2.1 Mealy Machine

A Mealy machine produces an output that depends on the state and the input and therefore the FA has the output written on the edges in a transition diagram. A Mealy machine takes in a string as input and produces a string of equal length on output.

Formal Definition [56]

A Mealy machine is a 6-tuple $(Q, \Sigma, \Gamma, \delta, \omega, q_0)$ where,

- Q is a set of finite states.
- Σ is the set of input symbols (alphabet).
- δ is the transition function where $\delta : Q \times \Sigma \rightarrow Q$.
- $q_0 \in Q$ is the start state.
- Γ is the set of output symbols.
- ω is the output function where $\omega : Q \times \Sigma \rightarrow \Gamma$ e.g., $\omega(q,a)$ gives the output associated with the transition from state q on the input a .

Representation

Figure 2.3 shows an example of a transition diagram for a Mealy machine. The figure depicts the Mealy machine which outputs a '1' for each substring 'bbb'.

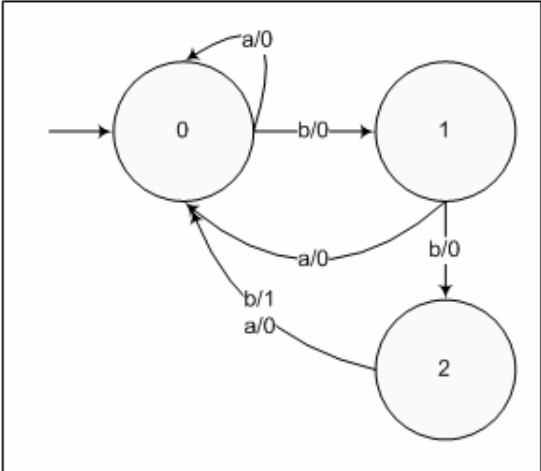


Figure 2.3: Transition Diagram for a Mealy machine

2.2.2.2 Moore Machine

The outputs are determined by the states itself and not the input. The outputs are therefore written in the state itself in a transition diagram. A Moore machine takes in a string of length n as input and produces a string of length $n + 1$ on output.

Formal Definition [57]

The Moore machine is a 6-tuple $(Q, \Sigma, \Gamma, \delta, \omega, q_0)$ where all is as in the Mealy machine except ω which maps $Q \rightarrow \Gamma$.

Representation

Figure 2.4 shows an example of a transition diagram for a Moore machine.

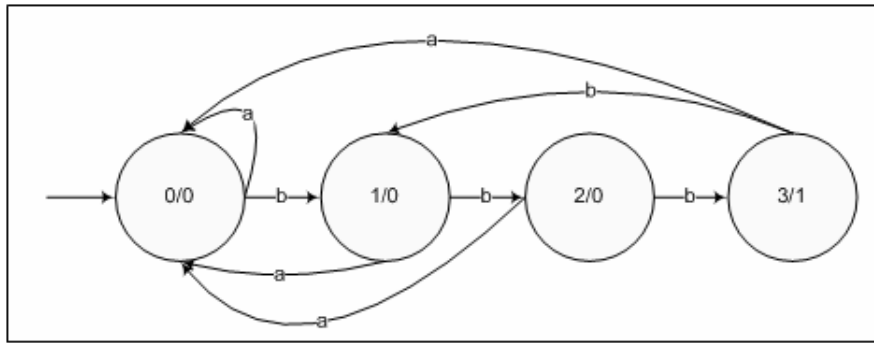


Figure 2.4: Transition Diagram for a Moore machine

Both the Mealy machine and the Moore machine are equivalent and an algorithm exists to convert one to the other.

2.2.3 Pushdown Automata

A pushdown automaton (PDA) describes a class of languages called ‘context-free languages’. The PDA is described as a finite automaton with a stack [21]. The stack is a string of symbols from some alphabet. There are many different interpretations as to when string w is accepted by a PDA. For the purposes of this document we define acceptance of a string if there is some final state where the entire input string has been read and the stack is empty [10].

Formal Definition [10]

A nondeterministic pushdown automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where,

- Q is a set of finite states.
- Σ is the set of input symbols (alphabet).
- Γ is the set of stack symbols.

- δ is the transition function where $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow \mathcal{P}(Q)$ where λ is the empty string.
- $q_0 \in Q$ is the start state.
- F is the set of final (accepting) states.

Representation

As with FAs, a PDA can be represented as a directed graph. For a PDA the edges are labeled with 3 symbols representing the input symbol, the character popped off the stack (λ if no symbol is popped off) and the character pushed onto the stack (λ if no symbol is pushed onto the stack) e.g., $a;A/\lambda$ is the transition when ‘a’ is read, ‘A’ is popped from the stack and nothing is pushed onto the stack. In order for the transition to be executed the symbol popped off the stack must be currently at the top of the stack.

Figure 2.5 shows a transition diagram for the language L where L is a context-free language and consists of all strings where the parentheses are fully balanced ($\Sigma = \{ (,) \}$ and $\Gamma = \{ A \}$).

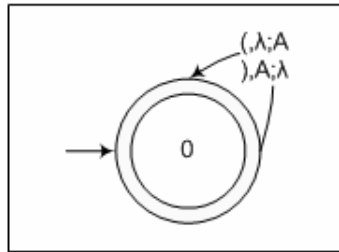


Figure 2.5: Transition Diagram for a PDA

The string ‘ $((()$ ’ is therefore rejected due to the fact that after the entire string is completely processed by the PDA the stack is not empty. The string ‘ $((())$ ’ on the other hand is accepted by the PDA as the stack is empty when the string has been completely read in.

The PDA in Figure 2.5 is considered deterministic as the transitions from state 0 are unique. A PDA is deterministic [10] if:

- For each combination of state, character being read in and stack symbol there is only one combination of destination state and stack action, i.e. $\mathcal{P}(Q)$ has only one element for this combination.
- If there exists a transition $a; \lambda/z_0$ then there exists no other transitions for $a \in \Sigma$ from $q \in Q$.

A PDA is classed as being nondeterministic otherwise.

2.2.4 Turing Machines

A Turing machine was first described by Alan Turing in 1936 before the invention of the computer. A Turing machine makes use of memory called a tape which is divided into cells, each of which contains one symbol. The tape is accessed by a *head* that can read and write symbols to the tape and can move left or right. The cells of the tape are assumed to be filled with the blank symbol 'B' if no symbols have been written to it. The Turing machine tape is infinite to the left and right. Figure 2.6 shows an example of a Turing machine tape.

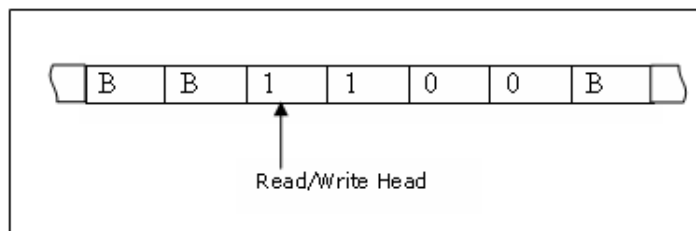


Figure 2.6: Turing Machine Tape

Formal Definition [21]

A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where,

- Q is a set of finite states.
- Σ is the set of input symbols (alphabet).
- Γ is the tape alphabet.
- $B \in \Gamma$ is the blank symbol.
- δ is the transition function where $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ where L and R are left and right respectively which indicates the direction to move the read/write head.
- $q_0 \in Q$ is the start state.
- F is the set of final (accepting) states.

Representation

Turing machines can act as an acceptor or a transducer. As with the previous automata a Turing machine can be represented as a directed graph. The edges are labeled with 3 symbols e.g., $a/X/R$ where 'a' is the symbol read from the tape, 'X' is the symbol to be written to the tape and 'R' is the direction that the head is moved, which in this case is right. Figure 2.7 shows a transition diagram for a Turing machine that accepts the language $a^n b^n$. If the tape is processed and results in a transition to the 'HALT' state the string is accepted by the Turing machine.

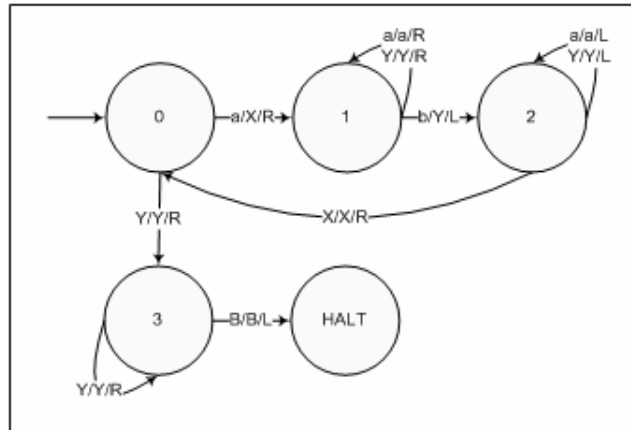


Figure 2.7: Transition Diagram for a Turing Machine

The Turing machine in Figure 2.7 is deterministic. A nondeterministic Turing machine differs from the deterministic Turing machine in that there are many different transitions from a state for a symbol being read from the tape.

Variations of Turing Machines

A Turing machine can have multiple tapes, each of which has its own head. The transition function for an n -tape Turing machine can be defined as

$$\delta : Q \times \Gamma^n \rightarrow Q \times \Gamma^n \times \{L, R\}^n.$$

A Turing machine with multiple tapes reduces the computation time required and in some cases a gain of quadratic speed [12] has been achieved.

A Turing machine transducer with two tapes uses one tape for the input and one for the output [39]. Initially the input tape contains input string M and the output tape is completely blank.

A Turing machine can also have a “stay-option” which adds the option of not moving the tape head at all. In this case, we write an ‘S’ instead of an ‘L’ or ‘R’. This allows the machine to change states without changing the tape head.

Another variation of Turing machines is a multidimensional Turing machine. The Turing machine tape is viewed as having the ability to extend infinitely in more than one dimension [29]. A two-dimensional machine has a transition defined as

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, U, D\}$$

where ‘U’ indicates that the head should move up and ‘D’ indicates that the head should move down.

These variations add nothing to the power of the Turing machine but it does improve the computation time.

2.3 The Chomsky Hierarchy (also known as Chomsky–Schützenberger hierarchy)

There are 4 classes of languages called the “Chomsky Hierarchy” described by Noam Chomsky in 1956 [9, 21, 54]. The hierarchy is summarized in Table 2.2 below.

| Type | Languages | Acceptor |
|------|--|---|
| 0 | Unrestricted grammars (includes all formal grammars) also known as recursively enumerable languages. | Turing machines |
| 1 | Context-sensitive grammars | Turing machines with a bounded tape, called linear bounded automata |
| 2 | Context-free | Nondeterministic pushdown automata |
| 3 | Regular | Finite automata |

Table 2.2: Chomsky Hierarchy

A regular language is also context-free; context-free languages are context-sensitive; context-sensitive languages are recursively enumerable.

2.4 Applications of Automata

Pattern Matching

Pattern matching is the act of identifying the presence of a given pattern where the pattern is rigidly specified. Pattern matching is a very important subject with regards to the wider domain of text processing. Pattern matching is at the core of Web search engines. The famous Knuth–Morris–Pratt (KMP) is a pattern-matching algorithm that efficiently builds these finite automata to search text T for a given search pattern P [2, 4]. Figure 2.8 shows the FA formed when applying the KMP Algorithm to search a text T for the pattern P=‘ABAA’. In Figure 2.8, ‘*’ is used to show an accept state.

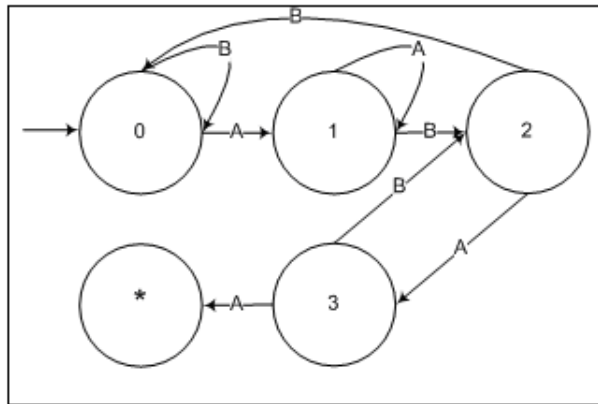


Figure 2.8: The FA for P='ABAA'

Circuit Design

Automata are important in sequential circuit design. State machines are considered to be one of the most frequently used model-building tools. Finite state transducers are the models used to implement digital logic circuits. Figure 2.9 is an example of a logic circuit and Figure 2.10 is the resulting transducer [9].

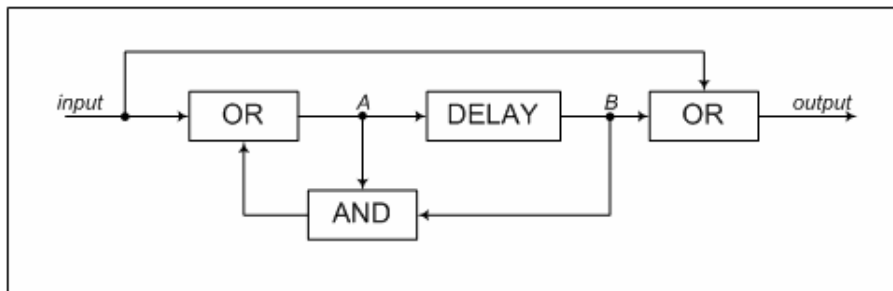


Figure 2.9: Logic Circuit

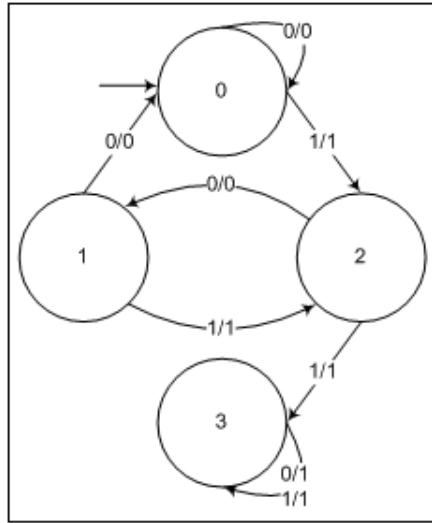


Figure 2.10: FST for the Logic Circuit

The FST in Figure 2.10 is built by firstly identifying the states. Each state indicates if there is current flowing at points A and B in the logic circuit in Figure 2.9. There are 4 possibilities:

State 0: A=0 and B=0

State 1: A=0 and B=1

State 2: A=1 and B=0

State 3: A=1 and B=1

Here '0' indicates that no current is flowing and '1' indicates that current is flowing through the points A and B.

The transitions of the FST are determined by evaluating the operation of the circuit given the various states. The operation of the circuit is as follows:

$$\text{New B} = \text{Old A}$$

$$\text{New A} = (\text{input}) \text{ OR } (\text{Old A AND Old B})$$

$$\text{Output} = (\text{input}) \text{ OR } (\text{Old B})$$

Therefore if we are in State 0, and the input is 0:

$$\text{New B} = \text{Old A} = 0$$

$$\begin{aligned} \text{New A} &= (\text{input}) \text{ OR } (\text{Old A AND Old B}) \\ &= (0) \text{ OR } (0 \text{ AND } 0) = 0 \end{aligned}$$

$$\text{Output} = (\text{input}) \text{ OR } (\text{old B}) = 0 \text{ OR } 0 = 0$$

The transition from State 0 for '0/0' is back to State 0 because New A = 0 and New B = 0.

If we are in State 0, and the input is 1:

$$\text{New B} = \text{Old A} = 0$$

$$\text{New A} = (\text{input}) \text{ OR } (\text{Old A AND Old B})$$

$$= (1) \text{ OR } (0 \text{ AND } 0) = 1$$

$$\text{Output} = (\text{input}) \text{ or } (\text{old B}) = 1 \text{ or } 0 = 1$$

The transition from State 0 for '1/1' is State 2 because New A = 1 and New B = 0.

This exercise is repeated for all 4 states in the FST. Table 2.3 show the results.

| Old State | After input 0 | | After input 1 | |
|-----------|---------------|--------|---------------|--------|
| | New State | Output | New State | Output |
| 0 | 0 | 0 | 2 | 1 |
| 1 | 0 | 0 | 2 | 1 |
| 2 | 1 | 0 | 3 | 1 |
| 3 | 3 | 1 | 3 | 1 |

Table 2.3: Transitions for the FST

Compiler Construction

Various compilers e.g., C/C++ are designed using finite automata. There are various phases involved in compiler construction, two of which can be implemented using automata:

- Lexical Analysis (Scanner): During lexical analysis character sequences are converted into tokens. Comments and whitespace are also removed during this phase. This can be implemented as DFAs seeing that tokens are specified as regular expressions.
- Syntax Analysis: During this phase the output from the lexical analysis phase is input. This input is then analysed to ensure proper syntax. These are expressed as context-free languages and therefore can be implemented as pushdown automata.

Game Programming

In game programming finite automata are used to control functions within a game. A state represents some activity within the game and a transition (event) occurs when something happens in the game world that makes the state change. Figure 2.11 shows a game state transition.

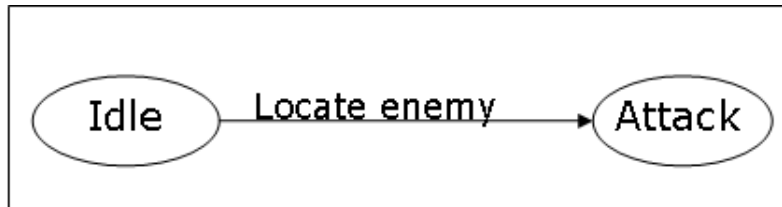


Figure 2.11: Game State Transition

Games like Quake and Doom rely on finite state machines to handle the changing states within the game. Figure 2.12 shows state transitions for a rocket projectile in Quake. Here “Finish Spawning”,

“Collision”, “Exploded”, etc. are all examples of transitions that have to occur before the state of the object within the game is changed [8].

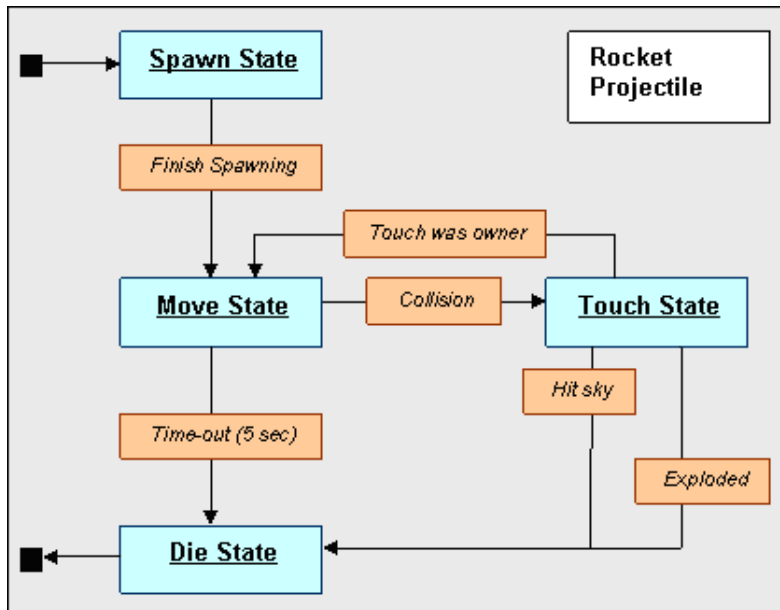


Figure 2.12: State transition of a rocket projectile from Quake

Managing the MID-let life-cycle (Mobile JAVA Applications)

Finite state machines provide a method to manage the life-cycle of a MID-let. MID-lets are Java applications designed to run on wireless devices e.g., mobile phones and PDAs and include java games and screensavers. The basic components of a MID-let are the Java Application Descriptor (JAD) and the Java Archive (JAR) file. The MID-let life-cycle is essential to creating java applications for mobile devices. Using a finite state machine simplifies the handling of the life-cycle. A MID-let has 3 states (Paused, Active and Destroyed) which forms the states of the FSM (Figure 2.13).

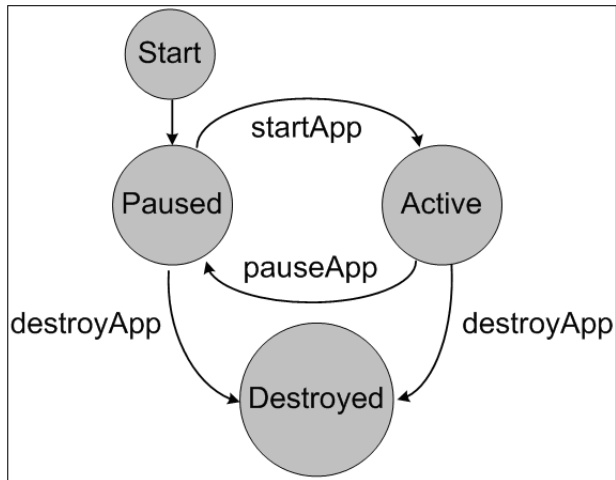


Figure 2.13: FSM to control the life-cycle of a MID-let

Software Design

Finite state machines provide a very powerful method to describe and develop the control logic for applications. “Automata-based programming” [51] uses the characteristics of finite automata during various stages of software development including specification and implementation.

Virtual Finite State Machine (VFMS) is a software design method used to describe the behaviour of a control system. The VFMS method establishes an execution model and is used in complex machine control. Figure 2.14 shows the execution flow chart of a VFMS.

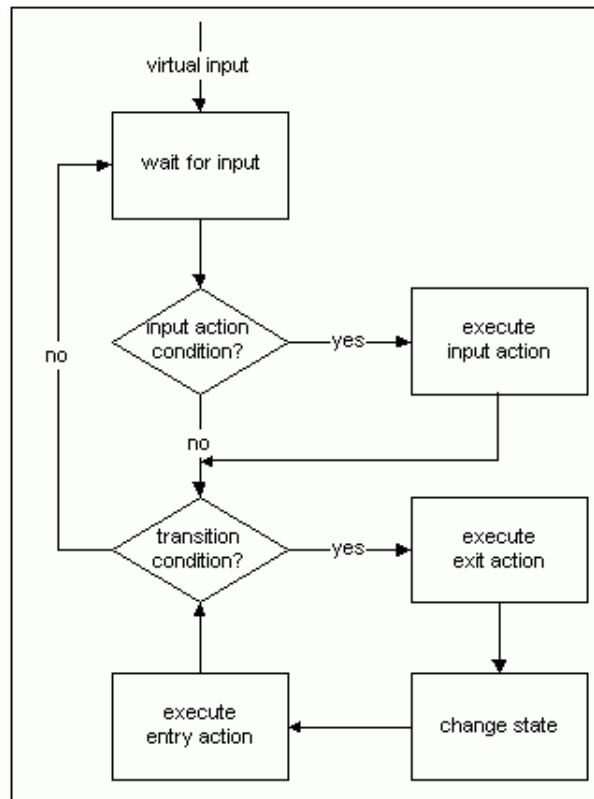


Figure 2.14: Virtual FSM Flow Chart

Biosequence Analysis

Turing machines have been used to recognize a collection of biosequences. Vallejo et al. [49] show how Turing machines can be used in biosequence recognition and analysis. This is discussed further in Chapter 4.

Chapter 3

An Overview of Genetic Programming

3.1 Introduction

Genetic Programming (GP) is an area of computer science that focuses on the automatic learning of computer programs. Automatic programming is an important area in computer science. “Machine Learning” is a broad term used to describe all forms of automatic programming and accordingly GP is a part of this large body of research. Genetic programming has been applied to a number of problem areas including data mining, circuit design and image processing. Genetic programming is an evolutionary algorithm based on Darwin’s theory of natural selection and evolution. This chapter introduces the basic theory of genetic programming.

3.2 Brief Overview

Genetic programming was introduced by Koza [27] and is essentially a genetic algorithm which evolves computer programs. GP maintains a population of individuals where each individual is a potential solution to a particular problem. Each individual of the population is usually represented using a variable-sized structure, usually a parse tree. However, this is not restricted and an individual can be represented using linear or graph-based structures [3]. Traditionally a generational GP algorithm is used, where the initial population is randomly created and this population is then progressively updated through a number of generations until a solution is found or some termination criteria are met. A fitness function is used to evaluate individuals and a fitness measure is assigned to each individual in the population. This fitness measure is then used by the selection method to select the fittest members of the population upon which the genetic operators are applied. The genetic operators are applied to chosen individuals to create offspring for the next generation. There are commonly three genetic operators used, namely, reproduction, mutation and crossover. The overall algorithm is shown below.

Algorithm 1: Genetic Programming Algorithm

Create an initial population.

repeat

Evaluate individuals of the population and assign a fitness to each individual.

Select individuals from the population using the selection method.

Apply genetic operators to the selected individuals.

Insert the result of the genetic operations into the new population.

until the termination criteria are met

3.3 Detailed Description

3.3.1 Control Models

The different methods of conducting a GP run are outlined in this section.

3.3.1.1 Generational Control Model

The generational control model is traditionally used in genetic programming. There is usually a distinct number of generations with a fixed population size. Each new population is then created from the old population.

3.3.1.2 Steady State Control Model

In this method no predetermined number of generations are defined and a single population is maintained where ‘bad performing’ individuals in the population are replaced by newly evolved ones. This method improves performance and is easy to implement [3].

3.3.1.3 Varying Population Size Control Model

This method maintains a single population whose size varies during a run depending on the fitness of the individuals in the population. A poor fitness increases the size of the population.

3.3.2 Representation of an Individual

According to Banzhaf et al. [3] there are three principle program structures used in GP i.e. tree, linear and graph structures. In most GP systems the tree structure is most commonly used. Trees are constructed by using elements of specific terminal and function sets. Each element of the function set has an arity which is the number of inputs to that function. The terminal set usually comprises the inputs to the functions e.g. constants and functions that have an arity of zero. Function sets are usually application specific. The following are some examples of function sets:

- Boolean Functions e.g. AND, OR, NOT, XOR.
- Arithmetic Functions e.g. +, -, *, ÷.
- Conditional Statements e.g. IF, THEN, ELSE.

Figure 3.1 shows an example representation of an individual using a tree structure. Tree structures are either executed using postfix order or prefix order. The postfix execution order of the tree in Figure 3.1 is: $a\ b\ +\ c\ d\ -\ *$. Prefix execution order executes the tree as: $*\ +\ a\ b\ -\ c\ d$.

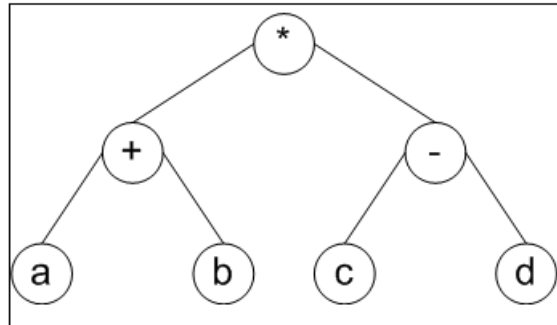


Figure 3.1: Example Representation using a Tree Structure

A linear structure is essentially a chain of instructions. The AIMGP (Automatic Induction of Machine Code with Genetic Programming) system [3] uses linear based structures. The linear program in Figure 3.2 is equivalent to the tree structure used in Figure 3.1. Memory is given to the instructions using a register machine. The instructions will then read values from and write values to the registers e.g. in Figure 3.2, the instruction $a = a + b$ reads a and b from the registers and adds the values together; the result is then placed into register a .

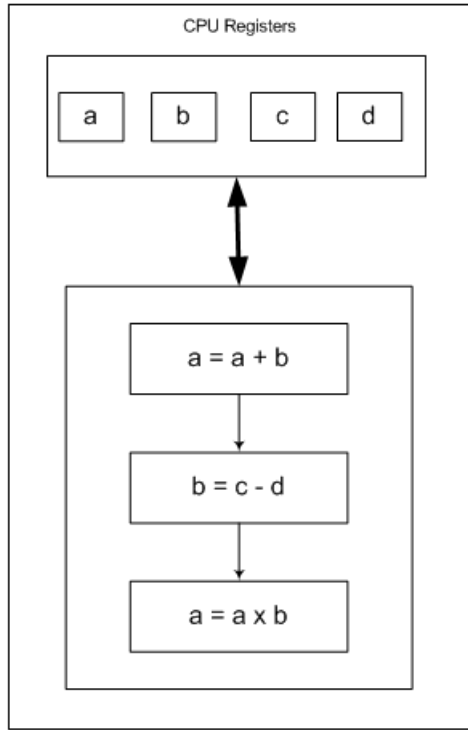


Figure 3.2: Example Representation using a Linear Structure

A graph is a set of vertices connected by lines called edges. The graph structures used in GP are usually directed graphs where an edge indicates the direction from one vertex to another. Graphs can represent complex problems and allows for loops and recursion. Teller et al. [46] used graph-based structures used for their PADO (Parallel Algorithm Discovery and Orchestration) system. The PADO system uses special start and end nodes, makes use of a stack and makes use of indexed memory. The stack is used to transfer data to the nodes. In Figure 3.3 the node labelled '4' pushes 4 onto the stack and the 'Plus' node pops two values from the stack, adds them together and pushes the result onto the stack. The indexed memory can also be used to store data. The 'Read' and 'Write' nodes are used to access the indexed memory. The PADO system decides on an execution path by choosing between the outgoing edges from each node e.g. from the 'Plus' state a choice is made between the node labelled 'A' and the node labelled '6'. A branch-decision function is applied to determine which edge should be chosen based on a number of factors including the stack and memory cells.

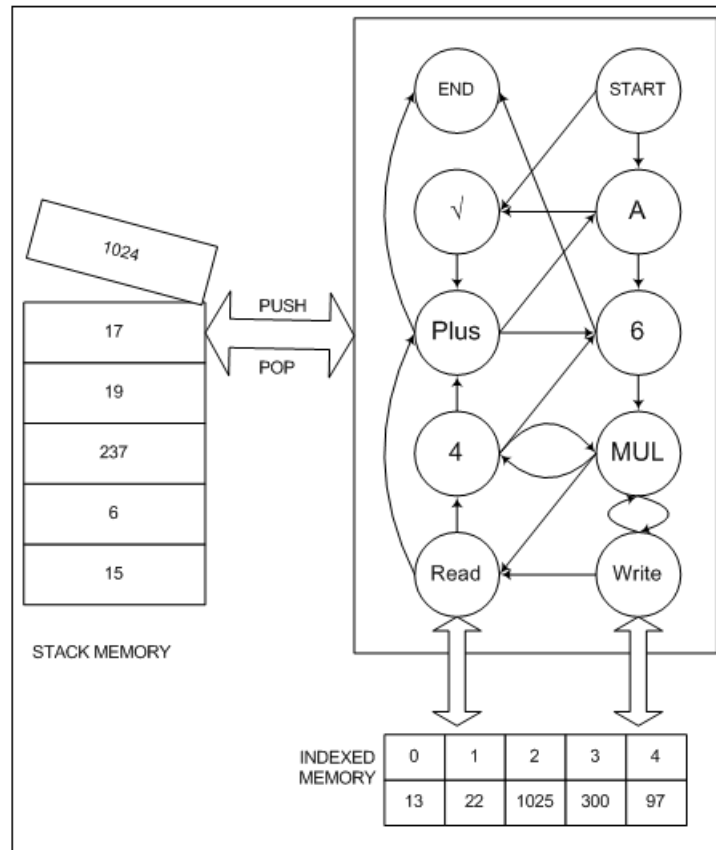


Figure 3.3: An example PADO Program [3]

3.3.3 Initialising the Population

The first step in the GP algorithm involves creating the initial population. The initial population is generated by creating a variety of individuals which are later evolved. The maximum size of the individuals in the initial population must be specified. For tree structures, the maximum size indicates the maximum depth of the tree or the maximum number of nodes in the tree. For the linear structure, the maximum size specifies the maximum number of instructions. For the graph structure, the maximum size indicates the maximum number of nodes in the graph.

The tree structure is created by choosing elements from the function and terminal set. There are three different methods used to create the tree structures i.e. the full method, the grow method and the ramped half-and-half method. Figure 3.4 shows an example of a tree created using the full method with a maximum depth of 3. Nodes are chosen from the function set at random until the maximum depth is reached. Nodes at the maximum depth are chosen from the terminal set. This then results in each branch of the tree reaching the maximum depth specified.

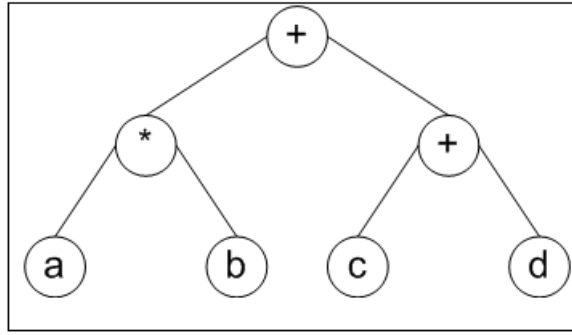


Figure 3.4: Tree initialised using the full method with a maximum depth of 3

Figure 3.5 shows an example of a tree created using the grow method with a maximum depth of 3. Using the grow method, variable sized trees are created. The nodes are chosen from the combination of the function and terminal sets until the maximum depth is reached. At the maximum depth, nodes are only chosen from the terminal set.

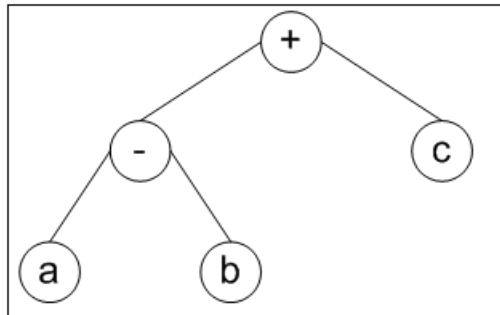


Figure 3.5: Tree initialised using the grow method with a maximum depth of 3

The ramped half-and-half method combines the full and grow method to create an initial population of various shapes and sizes. If the maximum depth is set at 6, an equal number of the population is created having depths 2, 3, 4, 5 and 6. For each depth value half of the trees are created using the grow method and the other half is created using the full method. A depth of 1 is not considered as this produces trivial trees which should not be included in the population. The ramped half-and-half method has an advantage in that the initial population generated has good variety in both size and shape. It is essential to have a good variety of individuals in the initial population as this helps in preventing premature convergence of the GP system.

Linear structures can be initialised using equivalent full and grow methods. CPU registers are used and are the equivalent of the terminals in trees. In the AIMGP system each node in the structure represents a machine code instruction. For each individual, a random length is chosen between two and the maximum length. Machine code instructions are added to the individual until the number of

instructions equals the random length chosen.

Graph structures can also use an equivalent full and grow method. Graphs make use of a maximum number of nodes therefore the full method would create graphs with the number of nodes equal to the maximum number permitted. The grow method on the other hand would create variable sized graphs. Unlike tree structures, a graph node can have more than one outgoing or incoming edges. It is therefore important to set a maximum number of outgoing edges for each node.

3.3.4 Fitness

GP makes use of a fitness function to evaluate each individual of the population. Each individual is then assigned a fitness measure which is used to determine which individuals of the population will be subject to the genetic operators. Fitness is the measure used by GP during evolution to determine how well the individual solves the problem. The fitness is calculated using a training set and is problem specific. This training set is referred to as fitness cases which consist of input and output pairs. Fitness cases are usually a small sample of the entire problem domain. There are usually four measures of fitness i.e. raw fitness, standardized fitness, adjusted fitness and normalized fitness.

3.3.4.1 Raw Fitness

Koza [27] defines raw fitness as “the measurement of fitness that is stated in the natural terminology of the problem itself”. For the artificial ant problem the raw fitness is the amount of food eaten by the artificial ant. A commonly used form of raw fitness is the fitness error. This error is calculated by the following formula [27]:

$$r(i, t) = \sum_{j=1}^N | S(i, j) - C(j) | \quad (3.1)$$

where, $S(i, j)$ is the value returned by the program i for the fitness case j ,
 $C(j)$ is the correct value for the fitness case j , and
 N is the number of fitness cases.

Raw fitness is problem dependent and therefore a better raw fitness may be a smaller value as with the fitness error or a larger value as with the artificial ant problem.

3.3.4.2 Standardized Fitness

Standardized fitness $s(i, t)$ is a fitness function where a value of zero is assigned to the fittest individual. If for a problem the smaller raw fitness value is better, then the standardized fitness equals the raw fitness for that problem i.e. $s(i, t) = r(i, t)$. If for a problem a larger fitness value is better, the standardized fitness is the maximum raw fitness minus the calculated raw fitness for the individual i.e. $s(i, t) = r_{max} - r(i, t)$.

3.3.4.3 Adjusted Fitness

The adjusted fitness is calculated using the standard fitness $s(i, t)$ as follows [27]:

$$a(i, t) = \frac{1}{1 + s(i, t)} \quad (3.2)$$

The adjusted fitness is always within the range zero to one. The fitter individuals of the population will have a higher adjusted fitness. Adjusted fitness is only required if the selection method used is fitness proportionate selection.

3.3.4.4 Normalized Fitness

Normalized fitness is also required if the selection method used is fitness proportionate selection. The normalized fitness is calculated using the adjusted fitness $a(i, t)$ as follows [27]:

$$n(i, t) = \frac{a(i, t)}{\sum_{k=1}^M a(k, t)} \quad (3.3)$$

where, M is the size of the population.

As with adjusted fitness, normalized fitness is between the range zero to one and the fitter individuals have a higher fitness. The sum of the normalized fitness values over the entire population is one.

3.3.5 Selection

There are various selection methods that can be used to decide which individuals to apply the genetic operators to. The most commonly used selection methods are fitness proportionate selection, truncation or (μ, λ) selection, tournament selection and rank selection.

3.3.5.1 Fitness Proportionate Selection

The most popular selection method is the fitness proportionate method. The probability that an individual will be copied into the next generation is calculated by applying the following formula:

$$\frac{f(s_i(t))}{\sum_{j=1}^M f(s_j(t))} \quad (3.4)$$

where, $f(s_i(t))$ is the fitness of individual i .

Usually $f(s_i(t))$ is the normalized fitness of the individual. A pool of individuals is created and parents are randomly chosen from this pool using the normalised fitness.

3.3.5.2 Truncation or (μ, λ) Selection

Each population consists of μ parents which breed λ offspring where $\lambda > \mu$. The best μ individuals are then selected from the λ offspring which are used as parents for the next generation. Truncation selection is usually used for large populations. The value μ is sometimes referred to as the ‘Truncation Threshold’ [43] which indicates the percentage of the population which are chosen as parents. Individuals below this threshold value are discarded and do not create any offspring.

3.3.5.3 Tournament Selection

Tournament selection is performed by randomly selecting a subset of the population. The individuals chosen make up the tournament and the number of individuals chosen is referred to as the tournament size. The individual with the best fitness from this subset of individuals is chosen. The tournament size can be adjusted, thereby adjusting the selection pressure. A smaller tournament size results in a low selection pressure and a larger tournament size leads to a high selection pressure.

3.3.5.4 Rank Selection

Rank selection involves assigning a rank to each individual in the population based on their fitness measure. The selection probability of an individual is then calculated as a function of their rank. There are commonly two types of ranking used i.e. linear and exponential ranking. For linear ranking, the probability of individual i being selected is calculated using the following formula [3, 6]:

$$p_i = \frac{1}{N} \left[p^- + (p^+ - p^-) \frac{i-1}{N-1} \right] \quad (3.5)$$

where, $\frac{p^-}{N}$ is the probability of the worst individual being selected and, $\frac{p^+}{N}$ is the probability of the best individual being selected.

For the population size to stay constant it is required that $p^+ + p^- = 2$.

For exponential ranking the probability is calculated using:

$$p_i = \frac{c-1}{c^{N-1}} c^N - i \quad (3.6)$$

where, c is a selection bias constant and $0 < c < 1$.

3.3.6 Genetic Operators

Genetic operators are used to transform the population of the individuals from one generation to another. There are three principle genetic operators i.e. crossover, mutation and reproduction.

3.3.6.1 Reproduction

The reproduction operator operates on a single individual and produces a single offspring. An individual is selected from the population using one of the selection methods and is copied into the new

population without any alterations.

3.3.6.2 Mutation

The mutation operator operates on a single individual and involves making random changes to the structure of the individual. An individual is randomly selected using one of the selection methods. For tree structures a mutation point is selected randomly and the subtree at the mutation point is removed. A new randomly created subtree is inserted at the mutation point. Figure 3.6 shows an example of mutation using a tree structure.

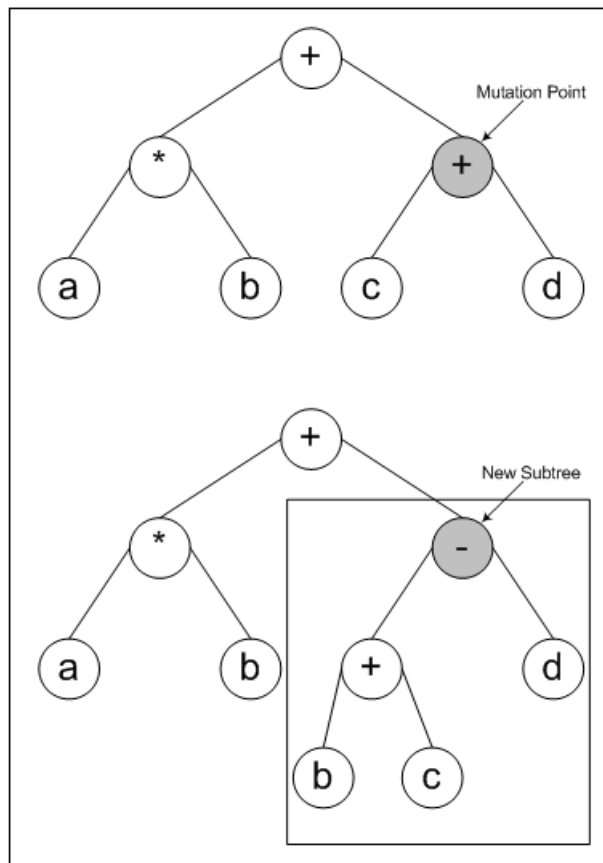


Figure 3.6: Example of Mutation for a Tree Structure

For linear structures mutation involves selecting one instruction from the chosen individual and then making one or more changes to that instruction. These changes can include changing the registers, constants or the operators.

Graph mutation involves randomly selecting a mutation point in the chosen individual. Similar to tree structures where the subtree is removed at the mutation point, the subgraph is removed from the graph structure. A new randomly created subgraph is inserted at the mutation point. This is

discussed further in Chapter 6.

3.3.6.3 Crossover

Crossover involves randomly selecting two parent individuals using one of the selection methods. Crossover involves creating new offspring using parts of the parent individuals. Figure 3.7 shows an example of tree-based crossover. For tree structures, crossover points are chosen in each parent individual. The subtrees rooted at these crossover points are swapped between the two parents.

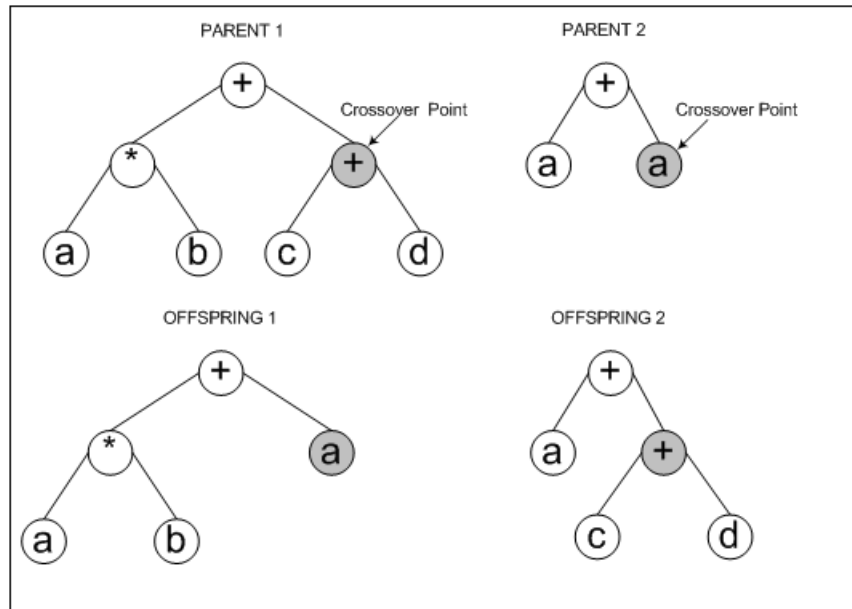


Figure 3.7: Example of Crossover for a Tree Structure

Crossover for linear structures involves swapping linear segments of code between the two parents. For graph structures, instead of swapping subtrees as with the tree structures, crossover involves swapping subgraphs between the two parents. Graph crossover is discussed further in Chapter 6.

3.3.7 User Decisions

The user must define a number of parameters before the GP system begins. Some of the important parameters are listed below:

- Population size.
- Genetic operator application rates.
- Number of generations.
- Maximum individual size.
- Tournament Size.

Chapter 4

Previous Work

4.1 Introduction

This chapter examines the previous research conducted where evolutionary algorithms were applied in an attempt to automatically generate automata. The evolutionary methods applied include genetic programming, genetic algorithms and hill-climbing.

4.2 Finite State Machines

Dupont [15] refers to “Grammatical Inference” as being an instance of “Inductive Learning” (the process of discovering the grammar for sentences defined on a specific alphabet). Grammatical inference has also been referred to as grammar induction, automata induction and automatic language acquisition [18]. Dupont makes use of the evolutionary method of Genetic Algorithms (GA) in an attempt to solve the grammatical inference problem. The method he refers to is that of the GIG method (Grammatical Inference by Genetic Search). The table below shows the GA parameters that Dupont employs.

| | |
|------------------------------------|--|
| Objective | The objective is to evolve a minimal deterministic finite acceptor that accepts a certain language L. |
| Maximum number of Evaluations | 2000 |
| Population Size | 100 |
| Fitness Cases | A sample set of positive and negative sentences. |
| Test Cases | The sample set of sentences used for each language consists of all strings up to a maximal length (l). Dupont uses $l = 9$ for alphabet a, b and $l = 7$ for alphabet a, b, c. |
| Number of Runs | 10 |
| Genetic operator application rates | Crossover=20%, Mutation=1% |

Table 4.1: Genetic Algorithm Parameters for the GIG method

A language set of 15 benchmark languages was used, 7 of which are Tomita’s [7, 15, 14] benchmark languages. The 8 remaining languages are shown in the table below.

| Dupont Language Number | Description | Positive Sample Sentence | Negative Sample Sentence |
|------------------------|--|--------------------------|--------------------------|
| 1 | a^*b | aaaab | aaaba |
| 2 | $(a^* + c^*) b$ | aab, ccccb | acab, cccbc |
| 3 | $(aa)^*(bbb)^*$ | aaaabbb | aaaab |
| 4 | any sentence with an even number of a’s and an odd number of b’s | aababab | abbaaa |
| 5 | $a(aa)^*b$ | aaaaab | aaaab |
| 6 | any sentence over the alphabet a,b with an even number of a’s | aababa | abbbaa |
| 7 | $(aa)^*ba^*$ | aabaa | abba |
| 8 | $bc^*b + ac^*a$ | bccb, accca | bcbccb, abca |

Table 4.2: Dupont’s benchmark languages

Each individual is represented as a partition of a set of states, where each block represents a merging of states. Suppose there are 5 states that represent the states of an automaton. An encoding of (2, 1, 2, 3, 2) of the set of states $\{1, 2, 3, 4, 5\}$ represents the partition $\{\{1, 3, 5\}, \{2\}, \{4\}\}$. Here (2, 1, 2, 3, 2) is renumbered as (1, 2, 1, 3, 1) to have the same left-to-right order as their state of minimal rank. Figure 4.1 below shows how a set of states is encoded. There are 3 blocks: $\{1, 3, 5\}$, $\{2\}$ and $\{4\}$. This signifies that states 1, 3 and 5 are merged to form one state while states 2 and 4 remain

unchanged.

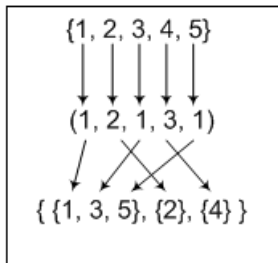


Figure 4.1: Example Representation of an Individual

The genetic operators used are mutation and crossover. Mutation involves the random selection of a state in an individual. This state is moved to a randomly selected block within the individual chosen. Suppose the individual is $\{\{1, \mathbf{3}, 5\}, \{2\}, \{4\}\}$ and state 3 is chosen randomly for mutation. Furthermore, consider that block 2 is selected. The resulting individual will be $\{\{1, 5\}, \{2, \mathbf{3}\}, \{4\}\}$. The number of blocks in a partition can change during mutation whereby the number of blocks can increase or decrease by one.

Crossover involves the random selection of a block in both parent partitions. The union of both chosen blocks is obtained and replaced in the parent partitions accordingly. Suppose the following 2 parents are chosen for crossover, $\{\{1, \mathbf{3}\}, \{2, 4\}, \{5\}\}$ and $\{\{1, \mathbf{4}\}, \{2, 5\}, \{3\}\}$. If block 1 is chosen in both individuals the resulting union of the blocks is $\{1, 3\} \cup \{1, 4\} = \{1, 3, 4\}$. The states that exist in common between the union of the blocks and the two chosen individuals are removed i.e. states 1, 3, and 4 are removed from the 2 parents and are replaced by the block $\{1, 3, 4\}$. The 2 resulting individuals is therefore $\{\{1, \mathbf{3}, 4\}, \{2\}, \{5\}\}$ and $\{\{1, \mathbf{3}, 4\}, \{2, 5\}\}$.

Dupont makes use of a semi-incremental procedure to create the first chromosome of the initial generation. This involves incrementally building up an automaton that accepts all the positive sample sentences. About 50% of the rest of the population of the initial generation are made up of random permutations of the first chromosome and the rest of the population is generated randomly.

Using this method Dupont was successful in generating solutions for 5 of the 15 languages with a mean fitness of 91%.

Dunay et al. [14] uses genetic programming techniques to infer DFAs. The DFA is not directly evolved, but instead the system proposed evolves an S-Expression which represents the DFA. Each S-Expression has an equivalent binary tree which can then be translated into a DFA. Figure 4.2 shows an example translation of a DFA. The root of the tree is the start state and the negative sign indicates an accept state.

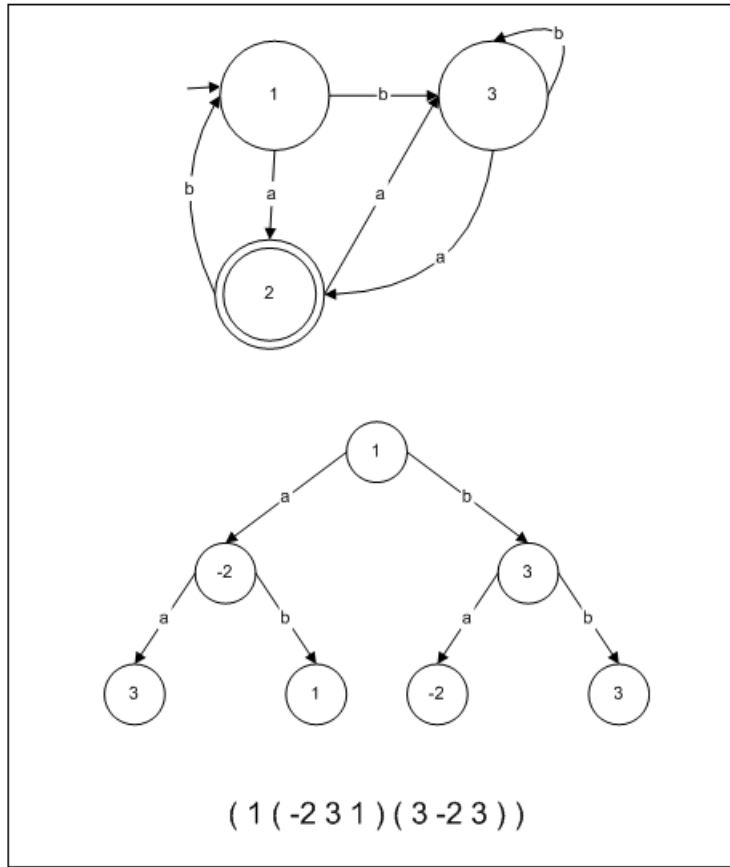


Figure 4.2: Example Translation of a DFA into an S-Expression

Crossover occurs by randomly choosing crossover points in the two parent individuals and swapping the subtrees rooted at that point.

This system was tested on the NB series of regular languages and Tomita's seven benchmark languages. The NB Series contain the language "All strings over {a,b} containing exactly n b's". Table 4.3 gives the GP parameters used to test the system.

| | |
|--|---|
| Objective | The objective is to evolve a program that encodes a deterministic finite acceptor (DFA) that accepts a certain finite language. |
| Number of Generations | 200 |
| Population Size | 1000 |
| Fitness Cases | 120 |
| Number of Runs | 30 |
| Bound on the length of the S-Expressions | 400 symbols including the parentheses |

Table 4.3: Genetic Programming Parameters for Dunay’s system

DFAs were successfully evolved for the first seven languages of the NB series (i.e. for $n = 1$ to $n = 7$) and all seven languages of the Tomita language set.

Brave [7] attempts to evolve deterministic finite acceptors using a method that combines genetic programming and cellular encoding. Brave defines cellular encoding as a genetic programming technique used by Gruau[19] that evolves the architecture, weights, and thresholds of a neural network, where evolution can take place concurrently. This technique is applied to the evolution of deterministic finite acceptors. Brave attempts to solve the benchmarks set by Tomita.

Table 4.4 shows the genetic programming parameters used by Brave in his system.

| | |
|------------------------------------|--|
| Objective | The objective is to evolve a program that encodes a deterministic finite acceptors (DFA) that accepts a certain finite language. |
| Function Set | PARALLEL (ARITY 2), PARALLEL-REJ (ARITY 2), RETRACT (ARITY 1), WRITE-NODE (ARITY 1), TO-NODE (ARITY 1) |
| Terminal Set | DONE |
| Raw Fitness | The number of incorrectly classified sentences. |
| Number of Generations | 100 |
| Population Size | 10000 |
| Genetic Operator Application Rates | Crossover(leaves)=10%, Crossover(nodes)=80%, Mutation=0% |
| Bound on number of Nodes | 300 |

Table 4.4: Genetic Programming Parameters for Brave’s system

The function set used consists of the following:

- **PARALLEL (Arity 2)**: The PARALLEL node creates an accepting state and its 2 children are the pointers to other states. Figure 4.3 below is an example of the PARALLEL node.

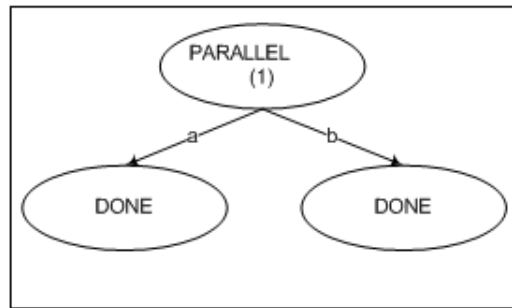


Figure 4.3: Parallel Node

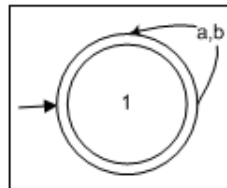


Figure 4.4: Single state DFA – Parallel Node (Accept State)

The PARALLEL node receives a label ‘1’ as this is the first state that is created and therefore indicates a start state. Figure 4.3 shows an example of the PARALLEL node and Figure 4.4 represents the DFA that is created as a result. The state that is created is an accept state.

- **PARALLEL-REJ (Arity 2)**: Similar to the PARALLEL but the state that is created is a reject state. The Figure below is an example of the PARALLEL-REJ node.

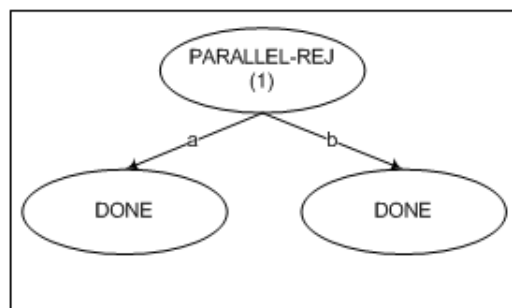


Figure 4.5: Parallel Reject Node

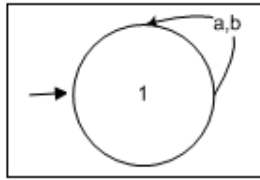


Figure 4.6: Single state DFA – Parallel-Rej Node (Reject State)

Similar to the PARALLEL node the PARALLEL-REJ node is given the label ‘1’ which indicates that this is the first state created and Figure 4.6 is the resulting DFA. A reject state is created.

- **RETRACT (Arity 1)**: Operator that doesn’t create a state but makes the active arc point to the node that it leaves creating a loop.
- **WRITE-NODE (Arity 1)**: Operator that doesn’t create a state but pushes the node at the tail of the active arc onto a stack.
- **TO-NODE (Arity 1)**: Operator that doesn’t create a node but instead makes the active arc point to the node that is on top of the stack. If the stack is empty then the active arc is unchanged.

The terminal set consists of the following:

- **DONE**: This is the only terminal, which simply indicates that the active arc is finished being modified.

Figure 4.7 shows an example representation.

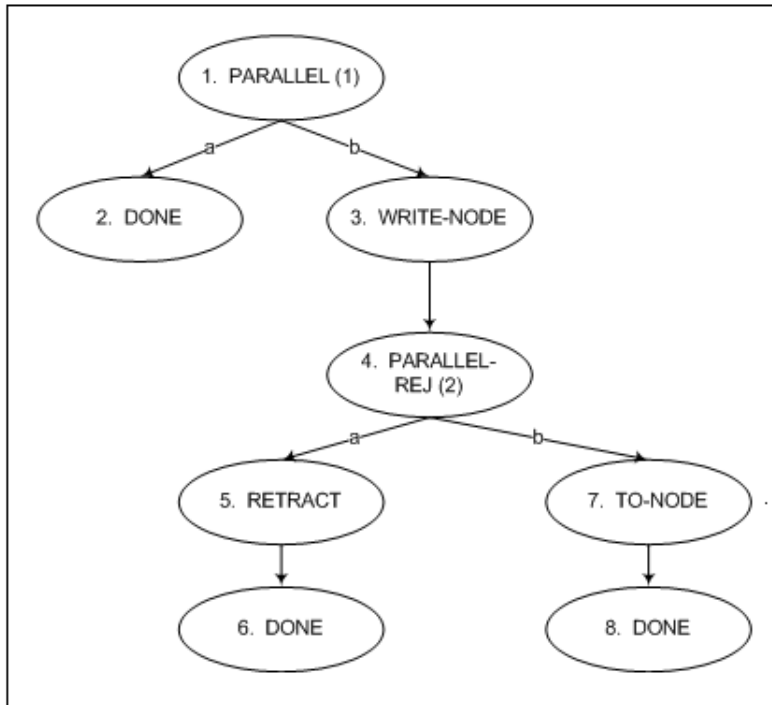


Figure 4.7: Tree Representation of a DFA for $L=\{\text{even number of b's}\}$

The nodes in Figure 4.7 are labelled and indicate the order in which the nodes are evaluated. The program begins with a PARALLEL at node 1. An accept state is created and receives the label '1' as indicated in Figure 4.8(a). Initially 'a' and 'b' point to the node created and the active arc is the output arc 'a'. The left subtree is therefore evaluated. The function node DONE is at node 2 of the tree which indicates no changes to the active arc. Arc 'b' is then activated and the right subtree is evaluated. The WRITE-NODE causes state '1' to be pushed onto the stack and node 4, PARALLEL-REJ creates a reject state which the active arc 'b' points to. The active arc then becomes the output arc 'a' of the second state. Nodes 5 and 6 are then evaluated and the RETRACT node results in the active arc pointing to the state that it left. The output arc 'b' from the second state is then activated and nodes 7 and 8 are evaluated. The TO-NODE results in state '1' being popped off the stack and the active arc is made to point to state '1'. The Figure 4.8 shows the development of the automaton as the tree in Figure 4.7 is evaluated.

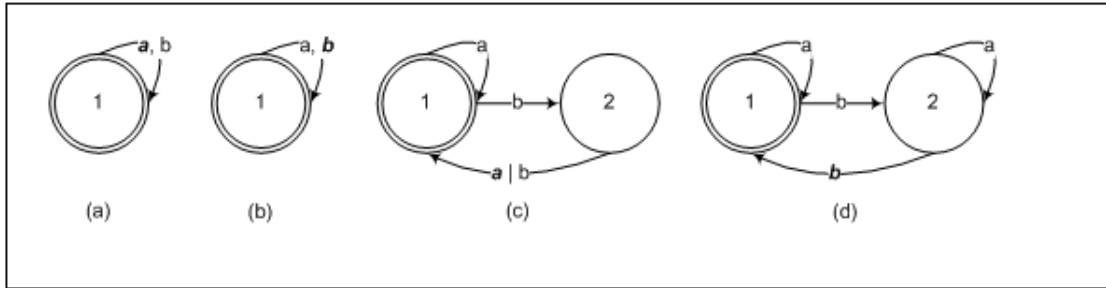


Figure 4.8: Example growth of a DFA

This method was tested on the seven Tomita benchmarks using 500 positive and 500 negative sample sentences. A solution was generated for all but one of the languages in less than 100 generations. The automata produced had a 100% success in classifying the sample strings. The language that didn't generate a solution was Tomita Language 6.

This representation used by Brave does have certain restrictions. Figure 4.9 shows an example DFA that cannot be evolved using this method. Evolving state 4 is impossible and this problem arises as the output arc 'a' is always active before 'b'. Creating state 4 would require state 1 to be on the stack above state 2 and this is not possible using the described representation.

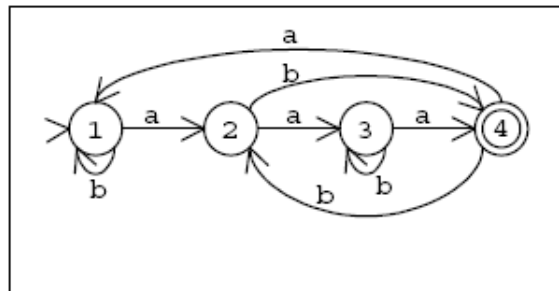


Figure 4.9: Example DFA that cannot be evolved using Brave's method

Belz et al. [5] proposed a genetic algorithm to induce finite state automata with an application to phonotactics. Phonotactics refer to permissible vowel sequences, consonant clusters and syllable structure [58] within a language. Using real linguistic data sets for German syllables and Russian disyllabic words, Belz presents a genetic algorithm to construct finite acceptors from positive data.

The proposed GA places the individuals of the population on a 2D grid of a fixed size and a special GA technique is then applied whereby an individual can only produce offspring with its neighbours.

| | | |
|---|----------|---|
| a | b | c |
| d | e | f |
| g | h | i |

Table 4.5: Grid showing the neighbourhood of ‘e’

Table 4.5 shows an example neighbourhood. Individual ‘e’ can only produce offspring with its neighbours i.e. a, b, c, d, f, g, h or i. Each individual of the 2D grid is considered in turn as a parent P_1 . P_2 is then chosen from P_1 ’s neighbours using rank based selection. Applying genetic operators on P_1 and P_2 , offspring are produced. P_1 is replaced if the fitness of the offspring is greater than or equal to P_1 .

Each individual is represented using a transition matrix with each element of the initial population chosen to be randomly between 2 to 4 states. The transitions between the states are then set randomly. Each transition can either be set to a valid state or to -1, where -1 implies that there is no transition between 2 states.

The fitness of an individual is evaluated using 3 parameters: consistency (C_1), size (C_2) and generalization (C_3). Consistency is determined by the number of strings in the data set that an individual accepts with partial acceptance also being rewarded. Size is determined by the number of states. Generalization is assessed in terms of the size of the language covered by an automaton, where the reward is higher the closer language size is to the specified target size. These 3 parameters are combined by normalizing the criteria to make up one third of the fitness value each. Weights are attached to them to affect the structural characteristics of the automata.

This resulted in successful solutions for both German and Russian languages indicating that GAs can be effectively applied to the automatic discovery of FAs that encode phonological grammars.

Luke et al. [33] propose a method of using gene regulation to induce deterministic finite acceptors. The system uses the traditional genetic algorithm approach. Individuals are represented using a number of genes, where each gene represents a state of a DFA and the gene regulation mechanism determines the transitions. Gene regulation is modelled by defining chemical templates for each gene. The system uses 3 chemical templates i.e. an *expression*, a *reading-0* template and a *reading-1* template where the alphabet for each language is $\{0,1\}$. Each gene has a boolean value indicating if the state is an accept state or a reject state. Table 4.6 outlines the GA parameters used by the system.

| | |
|-----------------------|--|
| Objective | The objective is to evolve a deterministic finite acceptor that accepts a certain language L. |
| Fitness Function | The number of correctly classified sentences. |
| Population Size | 500 |
| Number of generations | 100 |
| Number of runs | 50 runs per language for the first experiment and 20 runs per language for the second experiment |

Table 4.6: Genetic Algorithm Parameters used by Luke et al. [33]

The system was tested using the Tomita language set. In the first experiment the system was tested using a training set of all sentences of size 15 or less. Solutions were successfully induced for all languages except L5 (Any string of even length which, making pairs, has an even number of (01)'s or (10)'s). In the second experiment the system was tested using 100 negative sentences and 100 positive sentences. This resulted in solutions being generated for all seven languages.

Hingston [20] applies a genetic algorithm to infer a regular language from positive (and optionally negative) sample sentences. A prefix tree acceptor (PTA) is initially generated and state merging is applied to generate the finite state automaton. If there are only positive samples in the language set, a problem may arise that the states are merged into one accept state. To prevent this, a greedy search algorithm called "Minimum Message Length (MML)" is applied to the problem. The MML algorithm takes into consideration the following:

- The number of states in the FA.
- The total number of transitions leaving each state in the FA.
- The transition count for each symbol leaving the state.
- For each symbol and state, the next state must be specified.

The method used is named GARI (GA for Regular Inference). The parameters for the system are outlined in Table 4.7

| | |
|------------------------------------|--|
| Objective | The objective is to evolve a minimal deterministic finite acceptor that accepts a certain language L. |
| Maximum number of Evaluations | 2000 |
| Fitness Function | $fitness = exp(-ln(2) \times (\frac{MML}{PTA.MML})^2)$ |
| Fitness cases | Set of positive and negative samples sentences for the first and second experiment. Set of only positive sentences for the third experiment. |
| Test cases | The test cases used was the same as that used by Dupont [15]. |
| Population Size | 200 |
| Maximum number of Evaluations | 5000 |
| Number of generations | 25 for the first experiment and 100 for the second and third experiments. |
| Genetic operator application rates | Crossover=80%, Mutation=20% |

Table 4.7: Genetic Algorithm Parameters for the GARI method

Individuals are represented as sets of pairs of states to be merged. The set of pairs of states decides the partition. The individuals of the initial population are generated by randomly selecting pairs of states from the PTA, which are then merged. The selection method used is the roulette wheel selection (fitness proportionate selection). Elitism is employed whereby the fittest individual of each generation is preserved into the next generation, whereby 5% of the new population is reserved for copies of this individual.

Crossover is the standard one-point crossover and mutation is simply performed by bit-flipping i.e. a mutation point is randomly selected and if the bit is a ‘1’ it is set to ‘0’ and similarly if the bit is ‘0’ it is set to ‘1’.

Hingston tested his system on the seven Tomita languages as well as the additional eight languages proposed by Dupont [15]. In the first experiment the system was tested using 25 generations and this resulted in solutions for 4 of the 15 languages with a mean classification rate for the test set of 89.3%. The second experiment involved testing the system using 100 generations. The results improved slightly with solutions being generated for 5 of the 15 languages and a mean classification rate of 92.5% being achieved. In the final experiment the system was testing using only positive sample sentences. Solutions were generated on 6 of the 15 languages, however the mean classification for the test set of sentences was 84.3%. The system therefore doesn’t generalise well.

Lucas et al. [31] make use of a multi-start random hill-climber to evolve DFAs. The random hill-climber randomly selects an individual to mutate. If the mutated individual has a fitness better or

equal to the original individual it is accepted, otherwise it is rejected and the original individual is retained. If there is no improvement in the fitness for a predefined number of steps the current solution is recorded and the hill-climber is restarted. The system evolves the transition matrix of the DFA. An algorithm referred to as “*Smart Tuning the Output Labels*” was formulated to optimally select the output labels, thereby reducing the size of the search space.

The system was applied to the seven Tomita languages. The fitness function used was the number of correctly classified sentences. Initially the system was tested by fixing the maximum number of states to ten. Next the system was tested by setting the number of states to be exactly that of the minimal DFA for each language. The system was able to find solutions for all seven languages.

4.3 Finite State Transducers

Lucas [30] attempts to evolve finite state transducers (FSTs) using GP and random hill-climbing as the evolutionary algorithm. This is the only known study that investigates the induction of finite transducers. The algorithm is applied to chain-code based recognition of binary images. Chain-codes are used in image processing to classify an object’s boundary and then to analyse the numbers produced by that chain-code. At each step of the evolutionary algorithm the current individual is mutated. The mutated individual replaces the original individual if the fitness is greater than or equal to the original individual. Each individual is represented as a transition table. Mutation can take place in one of three ways namely,

- if the FST is below the maximum size, a randomly created state is connected to the FST,
- an entire row of the transition table is re-randomized¹, and
- a single entry in the table is changed.

The fitness function used takes into consideration three different string distance measures, the strict equality distance, the normalised hamming distance and the normalised edit distance. The system was able to evolve solutions for both the 4-chain and the 8-chain problem.

4.4 Pushdown Automata

Dunay [13] endeavours to induce deterministic pushdown automata (DPDA) using genetic programming. DPDAs were chosen as it is computationally easier to evaluate. The system was tested on the different languages outlined in Table 4.8.

¹The entire row is replaced by a randomly created row.

| Dunay Language Number | Description | Positive Sample Sentence | Negative Sample Sentence |
|-----------------------|-----------------------------|--------------------------|--------------------------|
| 1 | Balanced parentheses | (((())) | ()() |
| 2 | $a^n b^n$ | aabb | aaabbbb |
| 3 | equal number of a's and b's | bbabaa | aaabaa |

Table 4.8: Dunay's benchmark languages

The transition table of the PDA is represented as a binary tree. The representation is similar to that used in Dunay et al. [14] where the S-Expression is evolved which is described in Section 4.2. Each node of the tree is labelled with a 'category' followed by a state number. There are five different categories defined and each category has a unique notation. The notations are as follows:

- '-': This category reads a symbol from the input string and accepts the string if the stack is empty and if it is the last transition.
- '+': This category reads a symbol from the input string and rejects the string if the stack is empty and if it is the last transition.
- 'A': Push an 'A' on the stack.
- 'B': Push a 'B' on the stack.
- 'p': Pop from the stack.

For example 'A0' is a push state numbered '0'. An example S-Expression using this representation is $(-0(A0 +1)(B0 -1))$. Crossover is exactly the same as was done in [14] as is described in Section 4.2. The system was able to induce solutions for all three languages.

Huijsen [22] applies genetic algorithms in an attempt to induce pushdown automata and context-free grammars from a finite number of sample sentences. He was successful in inducing solutions for both deterministic and non-deterministic pushdown automata. Huijsen applies his solution to 3 problems: balanced parentheses, equal number of a's and b's (AB Problem) and ww^R (even length palindromes). The language set L consisted of all sentences from language L with a maximum length of 6. The GA used implements chromosomes as binary integer arrays.

Mutation is performed by simple bit flipping. Crossover is produced by exchanging fragments of 2 parent chromosomes. Huijsen uses two different fitness functions for his experiment: cumulative fitness (the number of correctly classified sentences) and productive fitness (product of percentage of correctly classified positive sentences and percentage of correctly classified negative sentences).

| Method | Population | Chromosome Length | Elite % | Generations | Runs |
|--------|------------|-------------------|---------|-------------|------|
| DPDA | 100 | 200 | 5 | 2000 | 20 |
| NPDA | 100 | 100 | 5 | 2000 | 20 |

Table 4.9: GA Parameters used by Huijsen

Huijsen makes use of a special “*elite mechanism*” whereby a small percentage of the fittest solutions of the old generation is retained in the next generation. Table 4.10 summarizes the results of the experiments performed by Huijsen. This table shows the average generation on which the global optimum was found using cumulative fitness and productive fitness.

| Method | Problem | Average generation to find global optimum | |
|--------|----------------------|---|--------------------|
| | | Cumulative fitness | Productive fitness |
| DPDA | Balanced parentheses | 14.6 | 4.2 |
| | AB Problem | - | 13.7 |
| NPDA | Balanced parentheses | 8.9 | 4.2 |

Table 4.10: Results of Huijsen’s experiment

For the ww^R , the experiment only yielded two optimal NPDAs: after 635 generations of the 3rd run and after 1156 generations of the 7th run.

Marc Lankhorst [28] makes use of a genetic algorithm to infer nondeterministic pushdown automata using a fixed finite set of positive (S_+) and negative (S_-) sample sentences. The reason for inferring nondeterministic pushdown automata is that they accept a full class of context free languages whereas deterministic pushdown automata only accept the deterministic subset.

A language set of ten languages was used. Six of the languages belong to the Tomita benchmark language set. Lankhorst chose to add to the language set a micro natural language (micro-NL) and some common test languages associated with grammatical inference shown in the table below.

| Lankhost Language Number | Description | Positive Sample Sentence | Negative Sample Sentence |
|--------------------------|---|--------------------------|--------------------------|
| 1 | equal number of a’s and b’s | bbabaa | aaabaa |
| 2 | balanced and nested parentheses expressions | ((()((())) | ()(((())()(((|
| 3 | even-length palindromes | abba | bbabbba |

Table 4.11: Lankhorst’s benchmark languages

ϵ -Transitions are not considered and the empty string is omitted from the languages. The parameters used by the GA are listed in Table 4.12.

| | |
|------------------------------------|--|
| Objective | The objective is to evolve a nondeterministic pushdown automata given a language L. |
| Number of Generations | 1000 |
| Population Size | 50 100 for the Micro-NL language and L3 defined in Table 4.11 |
| Genetic Operator Application Rates | Crossover=0.9, Mutation= $1/l$ where l is the length of an individual. |
| Sample set of Sentences | A set of randomly generated positive (S_+) and negative (S_-) sample sentences, with a maximum length of 30. 100 positive sentences and 100 negative sentences was used. |
| Number of runs | 10 |

Table 4.12: Genetic Programming Parameters for Lankhort's system

Each individual is represented by n transitions, where a transition from state p to state q is represented as $\langle q, p, a, Z, Y_1 \dots Y_k \rangle$ where,

- $q, p \in Q$ (the set of states).
- a is the symbol read.
- Z is the top of the stack.
- $Y_1 \dots Y_k$ are the stack symbols.

The length of each individual is therefore $n(k + 4)$. Lankhorst tests his genetic algorithm using both integer and binary encoding for individuals.

The genetic operators used are mutation and crossover. Mutation is random where any integer can be mutated into another integer that is in the gene's range. The crossover used is the standard two-point crossover where two indices are selected in the each individual and everything between the two points are swapped between the parent organisms. The selection method used was stochastic universal sampling using rank based selection. An elitist strategy was employed whereby the fittest individual of the previous generation survived.

The fitness function used by Lankhorst takes into account not only the correctly classified sentences but also the amount of symbols on the stack (the number of symbols remaining on the stack decreases the fitness) and also rewards an automaton for correctly analysing a part of the input.

Lankhorst was able to successfully generate an automaton for all languages in the language set. There were no significant differences in the results with respect to integer or binary encoding of individuals.

Afra Zomorodian [62] describes a method of automatically inducing a Deterministic Pushdown Automata (DPDA) using Genetic Programming. He attempts to find a DPDA given a finite set of positive and negative sample sentences. A sentence will be accepted if it reaches the accept state regardless of whether it reads the entire sentence or not. A sentence will be rejected if it reaches the reject state or reads off the end of the sentence. The approach that is taken in this case is not to generate the solution but rather to generate the ‘constructing program’ for the solution. The examples discussed are the generation of a DPDA for $a^n b^n$ and a DPDA for the problem of balanced parentheses.

| | |
|--|---|
| Objective | The objective is to evolve a program whose output describes a DPDA that accepts a certain finite language. |
| Function Set | READ (ARITY 3), POP (ARITY 3), PUSHA (ARITY 1), PUSHB (ARITY 1), DEC (ARITY 1) |
| Terminal Set | ACCEPT, REJECT, TOLAST |
| Raw Fitness | The number of correctly classified sentences. |
| Number of Generations | 200 |
| Population Size | 3000 |
| Genetic Operator Application Rates | Crossover (leaves) = 10%, Crossover (nodes) = 80%, Mutation = 0% |
| Bound on number of transition states (T) | 75 |
| Standard Fitness | $\frac{1-c}{2}$, where $c = \frac{N_{tp}N_{tn} - N_{fn}N_{fp}}{\sqrt{(N_{tn} + N_{fn})(N_{tn} + N_{fp})(N_{tp} + N_{fn})(N_{tp} + N_{fp})}}$, where N_{tp} = Number of true positives. ² N_{tn} = Number of true negatives. N_{fp} = Number of false positives. N_{fn} = Number of false negatives. |

Table 4.13: Genetic Programming Parameters for Zomorodian’s system

LISP-like macros are used for the function set. The function set consists of the following:

- **READ (Arity 3)**: The READ state creates three pointers to other states. The transition to the next state is determined on the character read off the sample sentence i.e. ‘a’, ‘b’ or ‘ Δ ’ where ‘ Δ ’ represents an empty string.

²Number of true positives is the number correctly classified as accepted. Number of false positives is the number incorrectly classified as accepted. Number of true negatives and number of false negatives are defined accordingly.

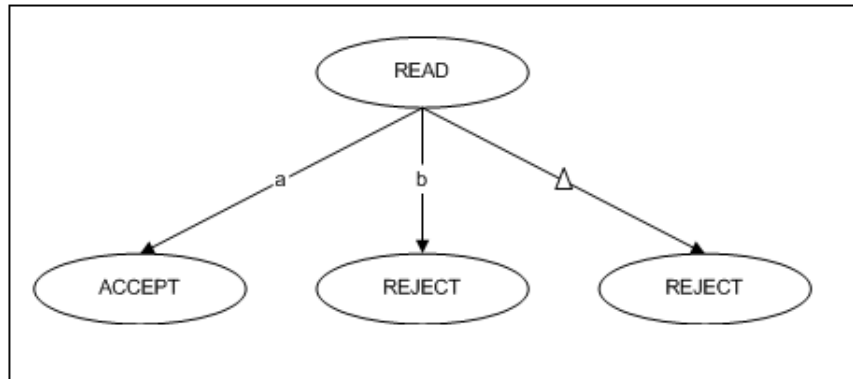


Figure 4.10: An example of a READ state

- **POP (Arity 3)**: Similar to the READ but the character is read and removed from the DPDA stack and not the sample string as in the READ.

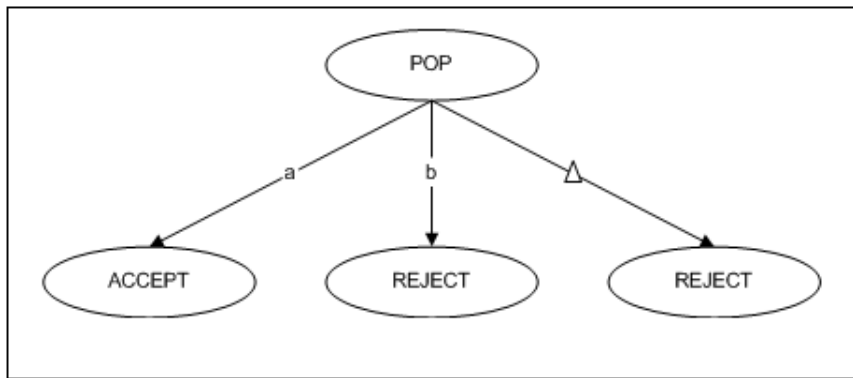


Figure 4.11: An example of a POP State

- **PUSHA, PUSHB (Arity 1)**: Pushes an 'a' or 'b' onto the DPDA stack.
- **DEC (Arity 1)**: Special operator, which doesn't create a state but decrements LASTSTATE where LASTSTATE is a variable that keeps a track of the last state created. The LASTSTATE variable is only updated when a READ or POP state is created. If the current state is the START state then the LASTSTATE variable is not decremented.

The terminal set consists of the following:

- **ACCEPT**: This state is where the language is accepted by the DPDA.
- **REJECT**: This state implies that the language is not accepted by the DPDA.
- **TOLAST**: Returns to the last state specified by the LASTSTATE variable.

Figure 4.12 is an example tree representation for $a^n b^n$ using the method proposed and Figure 4.13 is the corresponding program.

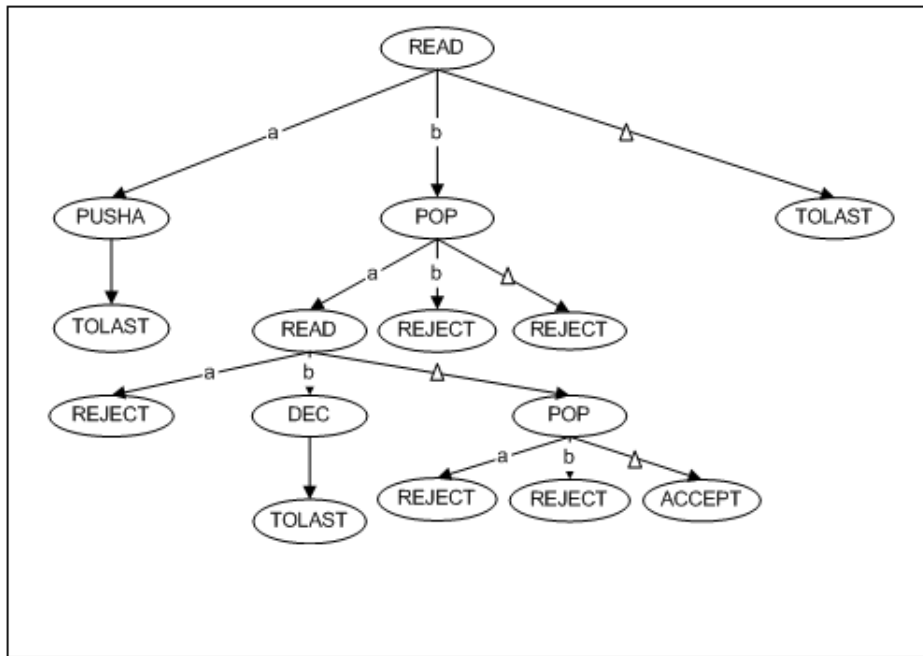


Figure 4.12: Tree Representation for $a^n b^n$

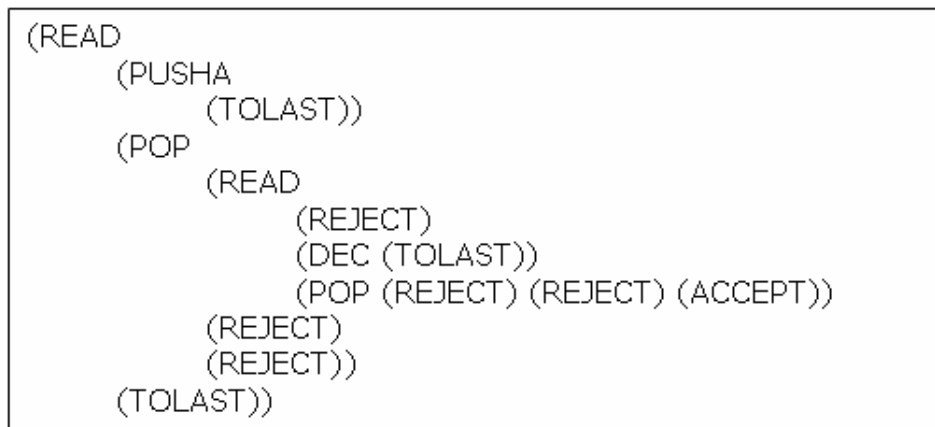


Figure 4.13: Program Representation for $a^n b^n$

The construction of any DPDA begins with a default START state. The DPDA that is constructed from the program is displayed in Figure 4.14.

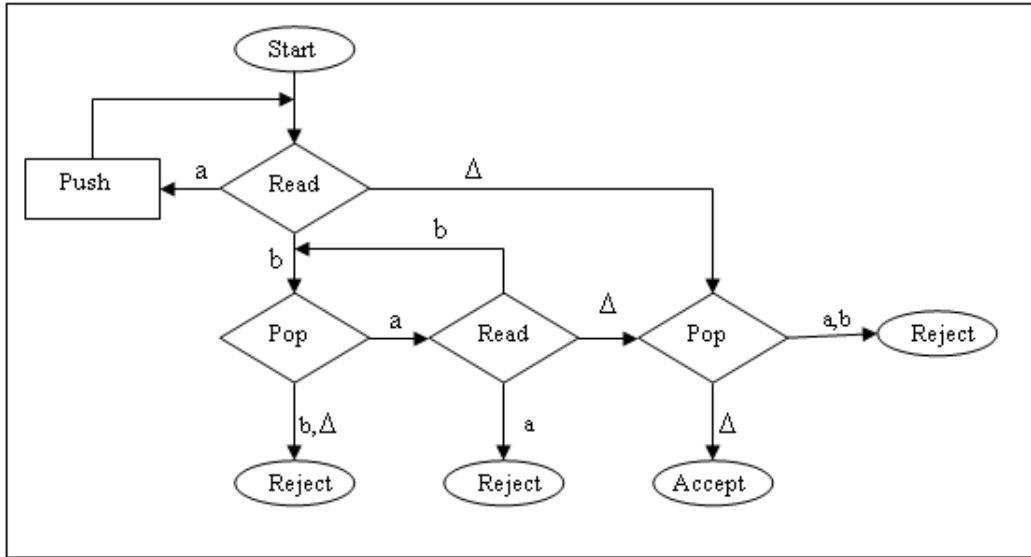


Figure 4.14: Transition Diagram for $a^n b^n$

Using a set of ten positive and nineteen negative sample sentences for the language $a^n b^n$, a solution emerged in generation 11, for one of the runs performed. The best-standardized fitness for an individual in generation 0 was 0.33515 with 16 hits.

For the balanced parentheses problem, the symbol ‘a’ was used for ‘(’ and ‘b’ was used for ‘)’. A set of 11 positive samples and 12 negative samples was used. In one of the runs performed a solution was found at generation 24. The best-standardized fitness of generation 0 was 0.28035 and it had 15 hits.

4.5 Turing Machines

Tanomaru [45] uses a customised genetic algorithm to evolve Turing machines. Each individual is represented as a transition matrix and the initial population is generated randomly with limits on the size of the automaton. During the generation of the t -th generation population $P(t)$ is duplicated and $P'(t)$ is created by applying the genetic operators to the individuals. $P(t + 1)$ is then created by selecting the best automata from $P(t)$ and $P'(t)$.

The genetic operators used are mutation and crossover. Crossover is performed by selecting a crossover point in each individual and the contiguous rows in the transition tables are exchanged. This may result in bad individuals as certain transitions may then refer to non-existent nodes. Mutation is performed by selecting a percentage of the transition table entries and then changing the selected entries randomly. The selection method used was the roulette wheel selection (fitness proportionate selection).

The Turing machines generated have a limit of 30 characters for the tape. Termination of a Tur-

ing machine occurs when the head advances beyond the tape’s limits, when the machine fails to stop within a maximum number of steps, when the machine stops acting or when the machine refers to a non-existing state. A machine stops acting if the machine enters an infinite loop.

The system was tested on two transducers, that is, a two-symbol sorting problem and a unary subtraction problem. Table 4.14 shows the genetic algorithm parameters used by the system.

| | |
|---|---|
| Objective | The objective is to evolve a general Turing machine for a given language L. |
| Number of Generations | 1000 |
| Population Size | 100 |
| Genetic Operator Application Rates | Crossover=90%, Mutation=10% |
| Sample set of Sentences | A set of input and target strings |
| Initial number of states | 10 |
| Maximum number of states | 20 |
| Number of fitness cases | 10 for each language |
| Maximum number of states exchanged during crossover | 3 |
| Number of runs | 10 |

Table 4.14: Genetic Algorithm Parameters for Tanomaru’s system

The two-symbol sorting problem sorts the symbols ‘a’ and ‘b’ in a given string e.g. given the input string ‘bbabbaab’, the target string is ‘aaabbbb’. The system generated Turing machines which not only correctly sorted the ten strings used for learning but also the 20 strings used for testing. The proper unary subtraction problem takes in a string which consists of two adjacent strings of 1s separated by a single 0 character. Each string of 1s represent a natural number n i.e. string ‘1111’ represents the natural number 4. The objective is to generate a Turing machine that gives the subtraction of the two numbers on an input string e.g. given input string ‘1111101’, the target string is ‘1111’. If the first number on the string is less than the second the machine doesn’t output anything i.e., output string is blank. Once again the system was able to successfully induce solutions that correctly classified not only the training set but also the test set of sentences.

Enhancing the genetic algorithm, Tanomaru tested the system on three languages which are shown in Table 4.15.

| Tanomaru Language Number | Description | Positive Sample Sentence | Negative Sample Sentence |
|--------------------------|--------------------------------|--------------------------|--------------------------|
| 1 | awb where $w \in \{a,b\}^*$ | aaaababbb | abbbba |
| 2 | $a^n b^n$ where $n \geq 1$ | aaabbb | aaabbbb |
| 3 | $a^n b^n a^n$ where $n \geq 1$ | aabbaa | aabbbaa |

Table 4.15: Tanomaru’s benchmark languages

The GA is enhanced by applying the “Population Shifting Approach”, defining new mutation operators and applying “1/5 Heuristics”. The population shifting approach collects statistics at each generation. The mutation operators are then applied in such a way as to favour the automata close to the size of the automaton with the best fitness in the previous generation. Crossover was not used in this experiment. Mutation was performed in three different ways:

- Mutation 1: This is same as the first approach i.e. the unenhanced approach with the exception that multiple changes on the same cell are allowed.
- Mutation 2: This mutation allows for the size of the automaton to change by either adding or deleting a state.
- Mutation 3: This mutation removes an individual and generates another with the number of states determined from the performance statistics from the previous generation.

A method called 1/5 Heuristics is used to update the mutation application rates at each generation. This is calculated based on the success rate of each mutation operator. The success rate of the mutation operator is calculated based on the number of individuals produced that are fitter than the parent. If 1/5 of the individuals generated are fitter than the parent, the mutation rate is changed to search more globally. If less than 1/5 of the individuals generated are fitter than the parent, the mutation rate is changed to search more locally.

In this experiment each language was tested using 100 runs and application rates of 70%, 25% and 5% were used for mutation 1, mutation 2 and mutation 3 respectively. These values were adapted at a later stage using the 1/5 heuristics. Enhancing the algorithm in this way led to solutions being generated for all three languages specified in Table 4.15.

Vallejo et al. [49], presents a genetic programming system for biosequence recognition and analysis. Biosequence analysis is used to determine the structure and function of biological molecules. Using a finite set of biosequences an NFA can be constructed to accept the sequences in the set and reject anything else. Vallejo et al. assumes that biosequences are recursively enumerable and therefore there exists a Turing machine that recognizes the collection of biosequences.

Using a collection of African and American HIV sequences, the system is tested using 3 experiments. Experiment 1 attempts to induce Turing machines using a genetic algorithm. The data set used is the

collection of HIV sequences and some counterexamples of sequences. A restricted Turing Machine is used and hence the Turing machine is represented as a 9-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}, t_{size}, s_{max})$ where,

- Q is a finite set of states.
- Σ is the set of input alphabet.
- Γ is the tape alphabet where, $B \in \Gamma$ and $\Sigma \subseteq \Gamma$.
- δ is the transition function where, $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L\}$.
- $q_0 \in Q$ is the start state.
- $q_{accept} \in Q$ is the accept state.
- $q_{reject} \in Q$ is the reject state.
- t_{size} is the tape size.
- s_{max} is the maximum number of computation steps.

During evaluation if the machine tries to move left off the left-hand side of the end of the tape the head stays in the same position for that move despite the transition function indicating that it should move left. Similarly, if the machine tries to move right off the right-hand side of the end of the tape the head stays in the same position despite the transition function indicating that it should move right.

The processing continues until the machine reaches an accept state or a reject state at which point the machine halts or if the machine reaches the maximum computation steps. The maximum computation steps are used to guarantee that the Turing machine doesn't enter into any infinite loops.

Individuals of the population are represented as transition tables.

| δ | a | b |
|----------|---------------|---------------|
| q_1 | (q_1, a, R) | (q_2, b, L) |
| q_2 | (q_3, a, L) | (q_2, b, R) |
| q_3 | (q_2, b, R) | (q_2, a, R) |

Table 4.16: Transition Table used in Experiment 1

The representation of the Turing machine consists of concatenating all the state transitions in the transition table so Table 4.16 yields:

$$(q_1, a, R) (q_2, b, L) (q_3, a, L) (q_2, b, R) (q_2, b, R) (q_2, a, R)$$

The GA system in experiment 1 uses tournament selection to select individuals to apply the genetic operators to. The fitness function used is the number of biosequences accepted. The GA parameters

used included 32 states, 8 symbols, 1024 individuals, 1000 maximum generations, a crossover probability of 0.6, a mutation probability of 0.001 and maximum computations of 64000.

The system proposed in this experiment induced a solution. The Turing machine evolved also accepted several biosequences not included in the training set.

In Experiment 2, a GA to induce a two-way deterministic FA is used. A two-way deterministic FA is a special type of Turing Machine with a read-only tape. The individual is represented as an 8-tuple $(Q, \Sigma, \delta, q_0, q_{accept}, q_{reject}, t_{size}, s_{max})$ where,

- Q is a finite set of states.
- Σ is the set of input alphabet.
- δ is the transition function where, $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L\}$.
- $q_0 \in Q$ is the start state.
- $q_{accept} \in Q$ is the accept state.
- $q_{reject} \in Q$ is the reject state.
- t_{size} is the tape size.
- s_{max} is the maximum number of computation steps.

Once again the individual is represented by concatenating all the state transitions in the transition table.

| δ | a | b |
|----------|------------|------------|
| q_1 | (q_1, R) | (q_2, L) |
| q_2 | (q_3, L) | (q_2, R) |
| q_3 | (q_2, R) | (q_2, R) |

Table 4.17: Transition Table used in Experiment 2

The individual for the transition table in Table 4.17 is therefore:

$$(q_1, R) (q_2, L) (q_3, L) (q_2, R) (q_2, R) (q_2, R)$$

The fitness cases, fitness function and GA parameters are exactly the same as that used in Experiment 1. A two-way deterministic FA was successfully induced which correctly classified all the sequences without modifying the tape.

In Experiment 3, the evolved automaton in Experiment 2 is used to approximate the multiple sequence alignment problem. Sequence alignment is used in bioinformatics to identify the regions of similarity between sequences. The two-way deterministic FA computes each sequence multiple times

by starting at each position in the sequences. This is then repeated for each sequence. If two sequences have a common pattern, the machine will exhibit identical behaviour for both sequences. The machine evolved in Experiment 2 was capable of identifying consensus patterns in biosequences.

Machado et al. [36] studied the feasibility of using Turing machines as an evolutionary model. They test the model on the famous Busy Beaver problem. The Busy Beaver problem was introduced by a Hungarian mathematician, Tibor Radó. He introduced the Busy Beaver problem to find an N -state Turing machine which leaves the largest number of 1's on the tape and then halts [53]. Machado et al. [35, 36] uses graph-based individuals for the representation of the Turing machines.

A Turing Machine is defined by Machado et al. as a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where,

- Q is a set of finite states.
- Σ is the set of input symbols (alphabet).
- Γ is the tape alphabet.
- δ is the transition function.
- $q_0 \in Q$ is the start state.
- F is the set of final (accepting) states.

The model is tested using two different transition functions:

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \quad (4.1)$$

and

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \cup \{L, R\} \quad (4.2)$$

Transition 4.1 is the transition defined in Chapter 2 where the Turing machine writes a new symbol on the tape and moves the head right or left and then enters a new state. Transition 4.2 on the other hand writes a new symbol or moves the tape head and then enters a new state.

Initially the system was tested using Transition 4.1. For the Busy Beaver problem $\Sigma = 1$ and $\Gamma = B, 1$. A maximum number of transitions is defined. If the maximum number of transitions is reached then the machine is considered to be a non-halting machine. The genetic programming parameters used by the system are shown in Table 4.18.

| | |
|-------------------------------------|---|
| Number of Evaluations | 25000000 |
| Population Size | 200 |
| Genetic Operator Application Rates | Crossover = 70%, Mutation = 0.05% |
| Standard Fitness | $f(i) = \begin{cases} \frac{\sigma(i)}{MaxOnes} \times \alpha + \frac{\omega(i)}{MaxT} \times \beta & \text{if } i \text{ halts} \\ \frac{\sigma(i)}{MaxOnes} \times \gamma & \text{if } i \text{ does not halt} \end{cases}$ <p>where $\sigma(i)$ is the number of ones on the tape, $\omega(i)$ is the number of steps performed and α, β and γ are the weightings added to the function.</p> |
| Maximum transitions (<i>MaxT</i>) | 150 |
| Maximum ones (<i>MaxOnes</i>) | 13 |
| Selection Method | Roulette wheel selection |
| Weightings | $\alpha = 1, \beta = 0.3, \gamma = 0.5$ |
| Number of Runs | 30 |

Table 4.18: Genetic Programming Parameters for Machado’s system

The system was successful in finding the 4–state Busy Beaver machine in all runs.

For the 5–state Busy Beaver machine, the system was tested using Transition4.2. The fitness function was altered to be

$$f(i) = h(i) \times [(1 + v(i) \times \alpha) \times \sigma(i) + (1 + v(i)) \times \theta(i) \times \beta + (1 + v(i) \times \gamma) \times \omega(i)] \quad (4.3)$$

where $h(i)$ is 1 if the machine halts before reaching the limit set and 0 otherwise, $v(i)$ is equal to 1 if the machine follows the variant rules defined and 0 otherwise, $\theta(i)$ is the number of transitions used, $\sigma(i)$ is the number of ones on the tape and $\omega(i)$ is the number of steps performed.

MaxT was set to 50000 and *MaxOnes* was set to 500. The weighting values were set to $\alpha = 4, \beta = 2$ and $\gamma = 1$. The system was successful in finding a 5–state Busy Beaver machine with a productivity of 20 i.e., 20 ones were written to the tape before the machine halted.

Chapter 5

Methodology

5.1 Introduction

This chapter examines the methodology used to achieve the objectives outlined in Chapter 1, section 1.2. The languages used to test each system as well as the system implementation details are also given.

5.2 Research Methodology Employed

Research is used to describe the “collection of information about a particular subject, and is usually associated with the output of science and the scientific method” [60]. Johnson [24, 25] identifies four methods of research commonly used in Computer Science namely:

- **Proof by Demonstration** involves building a prototype of the system as an example. The system is iteratively enhanced toward the desired solution.
- **Empiricism** involves generating a hypothesis and identifying the methods required to circumstantiate the hypothesis. The conclusions are then given and the hypothesis is either supported or rejected.
- **Mathematical Proof** uses mathematical models to analyze a system. There are two basic methods of proof used in Computer Science i.e. proof by verification and proof by refutation. Proof by verification uses a sequence of steps to show the truth of a hypothesis. Proof by refutation attempts to refute the correctness of a hypothesis.
- **Hermeneutics** is sometimes described “as the development and study of theories of the interpretation and understanding of texts” [55]. In Computer Science research this method is used to observe the performance of the system developed in the environment it was intended to operate in.

The methodology employed for this study is therefore proof by demonstration. The main aim of this study is to determine if genetic programming can induce an automaton for a given language. A GP

system is built for each type of automaton. Each system is then tested on a set of benchmark problems commonly used in literature. Randomness plays an important role in genetic programming. Due to this randomness a solution may not be found, therefore a number of runs will be performed for each language using different random number generator seeds.

If no solutions are found for at least one seed value for each problem the system is revised by:

- Altering the standard genetic programming parameters including the program representation, application rates of the genetic programming operators, population size and number of generations.
- Applying non-destructive genetic programming operators.
- Applying multiple iterations for seed values.

The revised system is then retested and the results are reported. If no solutions are evolved for a particular language, reasons for this failure are discussed.

5.3 Language Selection

Each GP system is tested on a set of benchmark problems where the fitness cases for each language is a set of positive and negative sentences.

5.3.1 Finite Acceptors

To test the finite acceptors system a set of fifteen benchmark problems were chosen. The fifteen languages chosen consist of the seven benchmark problems defined by Tomita [15],[7] (L1-L7 in Table 5.1) and the eight problems defined by Dupont [15] (L8-L15 in Table 5.1).

The language set used to test the system is shown in Table 5.1 below.

| Language Number | Description | Example |
|-----------------|--|---|
| L1 | a^* where $\Sigma=\{a,b\}$ | Accepted: aaaa Rejected: aabaa |
| L2 | $(ba)^*$ where $\Sigma=\{a,b\}$ | Accepted: bababa Rejected: babab |
| L3 | any sentence without an odd number of consecutive a's after an odd number of consecutive b's | Accepted: aaabaabb Rejected: aaabbbba |
| L4 | any sentence over the alphabet a, b without more than two consecutive a's | Accepted: aabaabbbbaa Rejected: babaab |
| L5 | any sentence with an even number of a's and an even number of b's | Accepted: ababbaba Rejected: aababb |
| L6 | any sentence such that the number of a's differs from the number of b's by 0 modulo 3 | Accepted: aaabbbbb Rejected: aaaba |
| L7 | $a^*b^*a^*b^*$ where $\Sigma=\{a,b\}$ | Accepted: aabaaabb Rejected: aabaaba |
| L8 | a^*b where $\Sigma=\{a,b\}$ | Accepted: aaab Rejected: aabaaba |
| L9 | $(a^* + c^*)b$ where $\Sigma=\{a,b,c\}$ | Accepted: aaaab, ccb Rejected: accb |
| L10 | $(aa)^*(bbb)^*$ where $\Sigma=\{a,b\}$ | Accepted: aabbb Rejected: aabaaba |
| L11 | any sentence with an even number of a's and an odd number of b's | Accepted: aabbb Rejected: aabaaba |
| L12 | $a(aa)^*b$ where $\Sigma=\{a,b\}$ | Accepted: aaaaab Rejected: aab |
| L13 | any sentence over the alphabet a, b with an even number of a's | Accepted: aaaabaa Rejected: aab |
| L14 | $(aa)^*ba^*$ | Accepted: aaaabaa Rejected: aab |
| L15 | $bc^*b + ac^*a$ where $\Sigma=\{a,b,c\}$ | Accepted: bccb, accca Rejected: accb |

Table 5.1: Language Set for FAs

5.3.2 Finite State Transducers

The data set consists of languages chosen from [9] and [17].

| Language Number | Description | Example |
|-----------------|--|---------------------------------------|
| L1 | Mealy machine that outputs a '1' for each substring 'aaa'. $\Sigma=\{a,b\}$ | Input: aaaaabaaa Output: 001000001 |
| L2 | Mealy machine that outputs a '1' for each substring 'aab'. $\Sigma=\{a,b\}$ | Input: abaabaab Output: 00001001 |
| L3 | Mealy machine that takes in a binary string in reverse order and outputs the number incremented by 1. | Input: 001 Output: 101 |
| L4 | Mealy machine that outputs a '1' at every position of a double letter. | Input: baaabbabb Output: 001101001 |
| L5 | Mealy machine that takes in a binary string and outputs the ones complement of the string. | Input: 011010 Output: 100101 |
| L6 | Mealy machine that reads in a binary string and outputs an 'E' if the number of 1s read in so far is even and an 'O' if it is odd. | Input: 010001001 Output: EOOOOEEEO |

Table 5.2: Language Set for FSTs

5.3.3 Pushdown Automata

A set of common PDA languages described in literature [9] were chosen to test the system. The language set used to test the system is shown in Table 5.3 below. Some of the languages are common to previous studies [13, 22, 28, 62].

| Language Number | Description | Example |
|-----------------|---|---|
| L1 | $a^n b^n$ where $n \geq 0$ | Accepted: aaabbb Rejected: aabbb |
| L2 | $a^n c b^n$ where $n \geq 0$ | Accepted: aaacbbb Rejected: aaabbb |
| L3 | All palindromes with an odd number of letters. $\Sigma = \{a, b\}$ | Accepted: aababaa Rejected: aabbaa |
| L4 | ss^r where s is in $(a + b)^*$ | Accepted: abba Rejected: baabaab |
| L5 | scs^r where s is in $(a + b)^*$ | Accepted: abcba Rejected: abba |
| L6 | s is in $(a + b)^*$: the number of a's in s is equal to the number of b's in s . | Accepted: bbaaba Rejected: aaababb |
| L7 | $a^n b^{2n}$ where $n \geq 0$ | Accepted: aabbbb Rejected: aabaaba |
| L8 | $\{a^n b^n: n \geq 0\} \cup \{a\}$ | Accepted: a, aabb Rejected: aabbbb |
| L9 | aa^*ba^* | Accepted: aaaabaaa Rejected: aaabaab |
| L10 | $a^n b^{2n}$ where $n \geq 1$ | Accepted: aabbbb Rejected: λ , aabbb |
| L11 | All strings with balanced brackets. $\Sigma = \{(,)\}$ | Accepted: (()(())) Rejected: ()(()) |

Table 5.3: Language Set for PDAs

5.3.4 Turing Machines

For Turing machine acceptors a set of 9 languages is selected from literature [9].

| Language Number | Description | Example |
|-----------------|---|---|
| L1 | $a^n b^n$ where $n \geq 1$ | Accepted: aaabbb Rejected: aabbb |
| L2 | $a^n b^n c^n$ where $n \geq 1$ | Accepted: aaaccbbb Rejected: aaacbbb |
| L3 | aba^*b where $\Sigma = \{a, b\}$ | Accepted: abaab Rejected: baabaab |
| L4 | $a^n b^{2n}$ where $n \geq 1$ | Accepted: aabbbb Rejected: abbab |
| L5 | $w w \in \{a, b\}^*, w$ has even length | Accepted: aabbbb Rejected: aabaaba |
| L6 | $w : w \in \{a, b\}^+$ and the number of a's in w is not equal to the number of b's | Accepted: aaaabaaa Rejected: aabbabab |
| L7 | $a^n b^{n+1}$ where $n \geq 1$ | Accepted: aabbb Rejected: λ , aabbbb |
| L8 | awb where $w \in \{a, b\}^+$ | Accepted: aabbbaba Rejected: λ , aabaa |
| L9 | $a^n b^n a^n$ where $n \geq 1$ | Accepted: aabbaa Rejected: λ , aabba |

Table 5.4: Language Set for TM Acceptors

For Turing machine transducers a set of 6 languages is selected from literature [9, 29].

| Language Number | Description | Example |
|-----------------|--|---------------------------------|
| L1 | Unary addition where the input consists of a string of unary integers separated by a 0 e.g. ‘1101’ represents ‘2 + 1’. | Input: 111011 Output: 111110 |
| L2 | A Turing machine that takes in an integer n in unary and outputs $2n$ ($n \geq 1$) | Input: 111 Output: 111111 |
| L3 | A Turing machine that takes in an integer n in unary and outputs $2n + 1$ ($n \geq 1$) | Input: 11 Output: 11111 |
| L4 | A Turing machine that takes in two integers in unary form separated by 0 as input and outputs the greater of both numbers | Input: 1101 Output: 11 |
| L5 | A Turing machine that performs the following function: $x - y$ if $x > y$ nothing is outputted if $x \leq y$ where the integers x and y are input as unary number separated by a 0 ¹ | Input: 11101 Output: 11 |
| L6 | A Turing machine that performs two-symbol sorting | Input: babba Output: aabbb |

Table 5.5: Language Set for TM Transducers

5.4 System Implementation Details

The GP system was developed using Eclipse 3.1. The system was implemented in Java (JDK version 1.5.0_06). The random number generator used is Java’s standard **Random** class [48] which generates pseudorandom numbers using a 48-bit seed. Seed values are used by the **Random** class such that if two instances of **Random** are created with the same seed, they will generate and return identical sequences of numbers. Simulations were run on an 1.86GHz Intel Pentium Processor with 1GB RAM and a Windows XP operating system.

¹This language is unary subtraction where the input consists of a string of unary integers separated by a 0.

Chapter 6

Genetic Programming Systems for Automata

6.1 Introduction

The chapter that follows describes the genetic programming systems implemented for the various classes of automata. The GP parameters, representation and GP operators are outlined for each system. The GP parameters were chosen by firstly performing a few trial runs and adjusting the parameters as required. The number of fitness cases used for each language are also outlined. The fitness cases for each language were chosen randomly.

6.2 Finite Acceptors

The section that follows describes the genetic programming systems used to induce deterministic finite acceptors (DFA) and nondeterministic finite acceptors (NFA). Two different systems are proposed for NFAs.

6.2.1 Deterministic Finite Acceptors

6.2.1.1 Representation of an Individual

Each individual of the population is represented using a directed graph as opposed to a tree-based structure. Employing a graph structure allows the system to take advantage of the natural form of a finite acceptor's transition diagram. Each vertex of the graph represents a state of a DFA. When a new graph is created, each vertex added to the graph is randomly chosen to be either an accept state or a reject state. The directed edges of the graph represent the transitions of the DFA. All individuals of the population are classed as complete finite acceptors given that for each state there is a transition to another state for each input symbol of the alphabet. The arity of each node is therefore the size of the alphabet. Duplicate individuals are allowed in the initial population. Each state is numbered sequentially where the state numbered zero is the start state. Figure 6.1 shows an example of how

an individual is created. The alphabet of this DFA is $\{a,b\}$. Initially, a start state is added to the graph and is chosen to be either an accept state or a reject state. In Figure 6.1(a), the initial state is numbered '0' and is randomly chosen to be a reject state. Each state added to the graph has an arity of two which is the size of the alphabet. Figure 6.1(a) is incomplete as the transition for 'a' and the transition for 'b' are not connected to an existing state. If the graph is incomplete there are two options for a transition that is not connected to an existing state:

1. Add a new state and connect the transition to the newly created state.
2. Connect the transition to a state that already exists in the graph.

If the number of nodes in the graph reaches the maximum number of nodes allowed only option 2 can be performed. Figure 6.1(b) is an example where option 2 is chosen for the transition for 'a'. As there is only one existing state, 'a' connects to state 0. Figure 6.1(c) is an example where option 1 is chosen for the transition 'b'. Figure 6.1(d) is the resulting DFA.

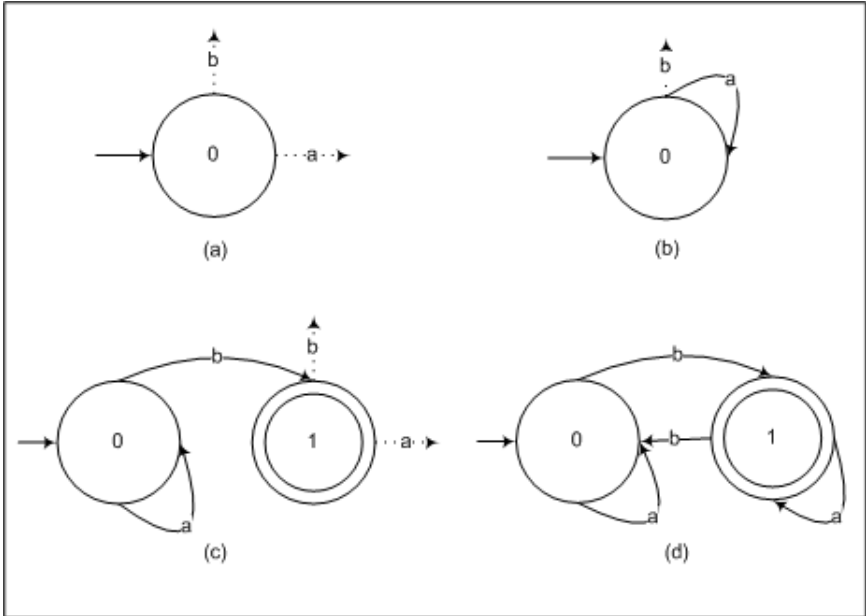


Figure 6.1: Creation of a DFA

Figure 6.2 below shows an example representation of an individual for the language a^*b^* .

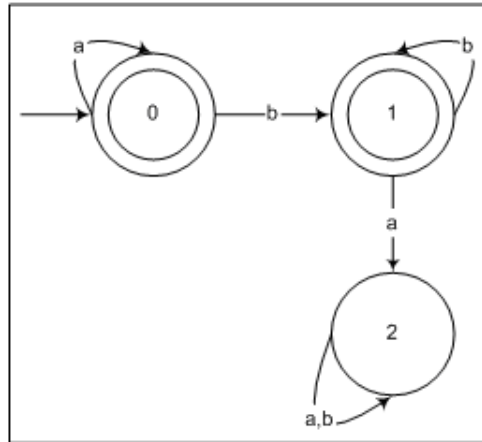


Figure 6.2: DFA for a^*b^*

The following syntax is used by the program for the DFA in Figure 6.2:

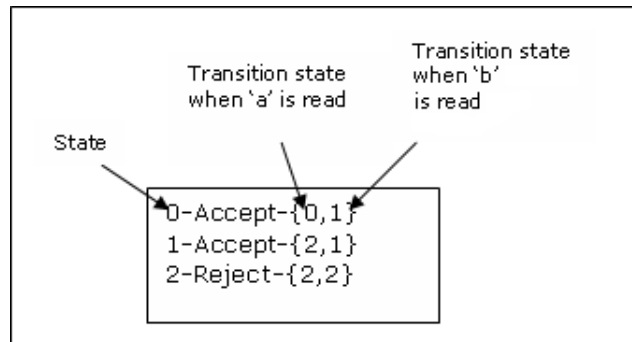


Figure 6.3: Syntax for DFA

The concept of a *subgraph* is brought into play when applying the genetic programming operators, crossover and mutation. Figure 6.4 below shows the resulting subgraphs if the state numbered 1 is chosen as the crossover or mutation point (n) in the parent graph. Subgraph 1 consists of all the nodes numbered less than the chosen node n whereas Subgraph 2 consists of n as well as all the nodes numbered greater than n .

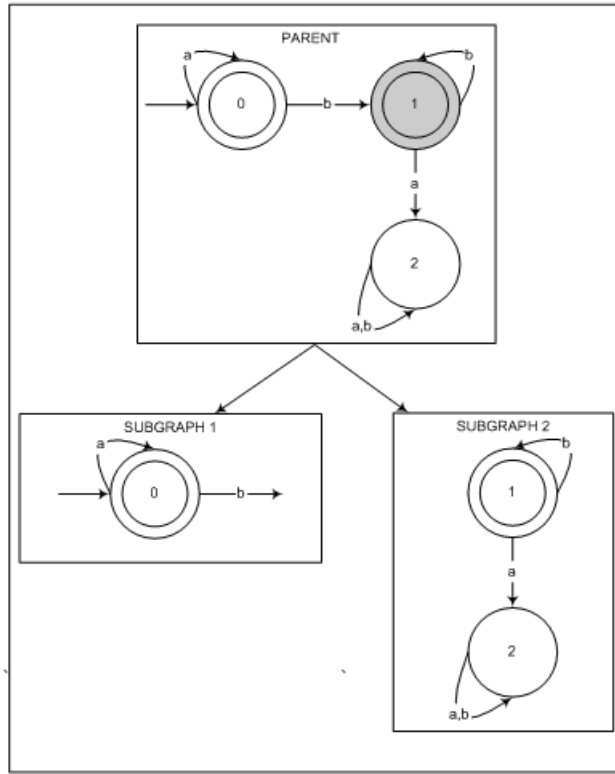


Figure 6.4: Resulting Subgraphs

6.2.1.2 Genetic Programming Operators

The genetic programming operators used are reproduction, crossover and mutation.

Reproduction Reproduction involves making an exact copy of an individual and inserting this copy into the new population.

Crossover Crossover requires an exchange of genetic material of two randomly chosen parent individuals. Crossover in a graph-based system is somewhat more complicated. In this graph-based system crossover involves an exchange of subgraphs. The method used here is similar to the method employed by Teller [26, 46].

The crossover points in each parent individual are chosen randomly. Algorithm 2 shows how crossover is performed.

Algorithm 2: Crossover Algorithm

1. Choose two individuals as parents P_1 and P_2 using a selection method.

2. Select a crossover point CP_1 and CP_2 in each graph P_1 and P_2 respectively. The crossover points are chosen such that the resulting graphs do not exceed the maximum nodes allowed.

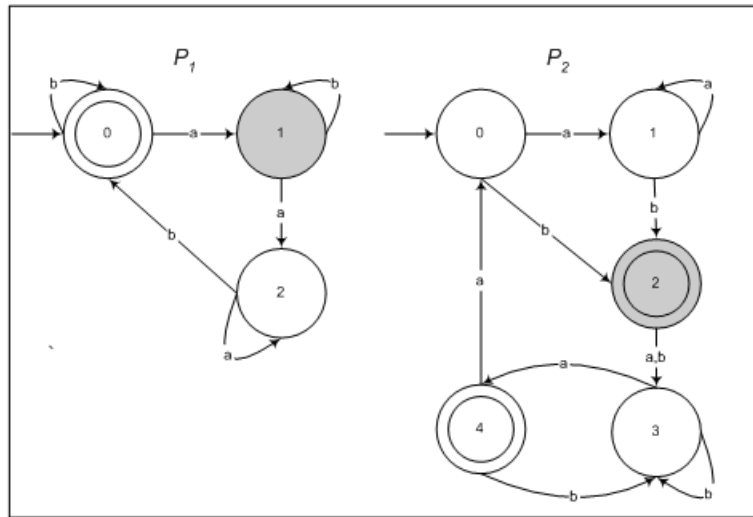


Figure 6.5: Example of chosen parent individuals and crossover points (step 1 and 2)

3. Divide each graph into two node sets (subgraphs) based on the crossover points selected. Call the subgraphs P_1S_1 , P_1S_2 , P_2S_1 , P_2S_2 where P_1S_1 and P_1S_2 are the subgraphs of P_1 and P_2S_1 and P_2S_2 are the subgraphs of P_2 .
4. In each subgraph label all arcs *internal* if they point to nodes within the subgraph, label them as *external* otherwise.

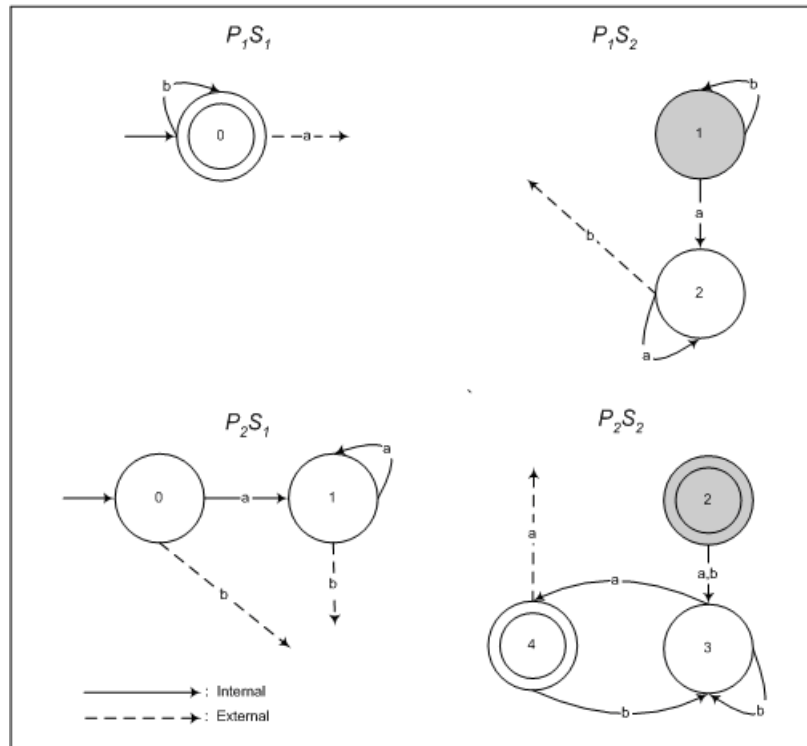


Figure 6.6: Resulting subgraphs of the chosen parent graphs (step 3 and 4)

5. Swap the selected subgraphs of the chosen parent individuals to form the children individuals C_1 and C_2 where C_1 is a combination of subgraphs P_1S_1 and P_2S_2 and C_2 is a combination of subgraphs P_2S_1 and P_1S_2 .
6. All external edges in C_1 whose destination node was CP_1 in P_1 is redirected to point at CP_2 in the newly formed graph C_1 and similarly for all external edges in C_2 whose destination node was CP_2 in P_2 is redirected to point at CP_2 in the newly formed graph C_2 .
7. All remaining external edges of each child graph are redirected to point at a randomly chosen node within the graph and all nodes are renumbered sequentially.

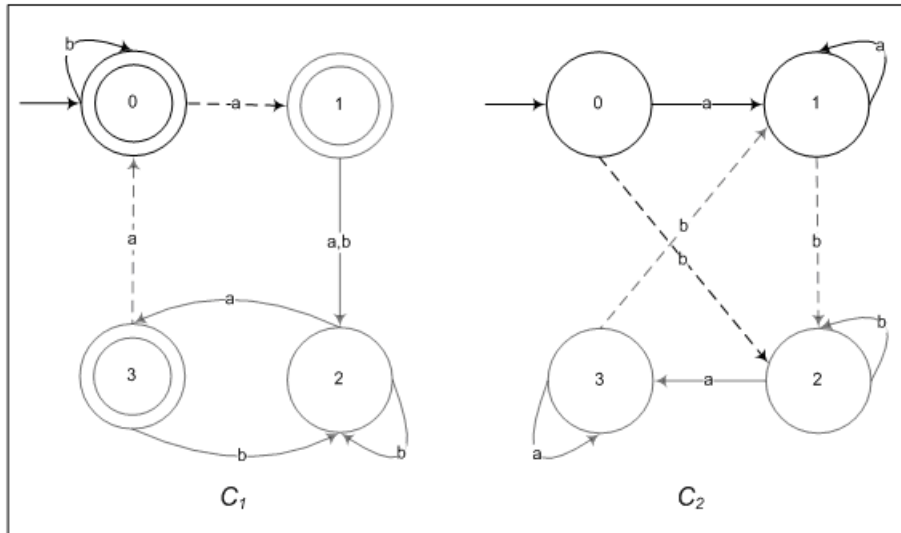


Figure 6.7: Resulting children graphs (step 5, 6 and 7)

Mutation Mutation introduces new genetic material into the population. Unlike crossover, mutation is only performed on one individual.

Algorithm 3: Mutation Algorithm

1. Choose parent P using the chosen selection method.
2. Randomly select mutation point M in the individual P .

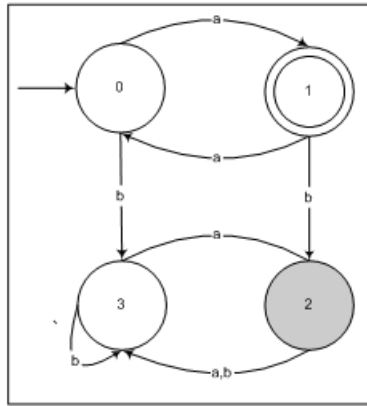


Figure 6.8: Chosen parent and mutation point

3. Remove the subgraph numbered from the mutation point M and label all edges in the remaining subgraph as internal or external as in crossover.

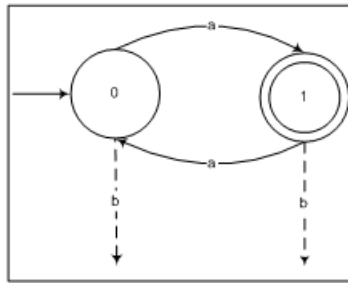


Figure 6.9: Mutation - Remaining subgraph

4. Replace the removed subgraph with a new randomly generated graph.
5. All remaining external edges are redirected to point randomly at an existing node in the newly created child graph.

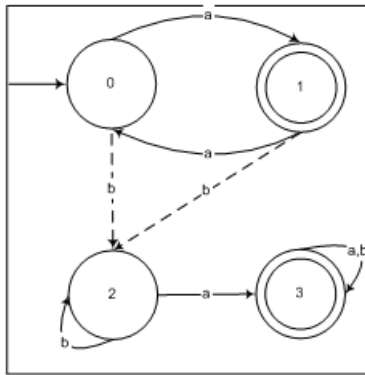


Figure 6.10: The resulting child graph after mutation

6.2.1.3 Interpretation

The fitness value for an individual is the total number of correctly classified sentences. Algorithm 4 below provides details on how a string is classified by the system. If the string has been correctly classified the fitness value of the individual is incremented as is shown by Algorithm 5. If the fitness value is equal to the number of fitness cases then the graph is deemed to be a solution.

Algorithm 4: DFA: isAccepted

Input: String s
Output: Boolean value indicating if the string s is accepted by the Graph G
Node $n \leftarrow$ the initial/start node of Graph g
for $i = 0$ to *length of string s* **do**
 $n \leftarrow n.getTargetNode(s.charAt(i))$ {the target node when the character at position i of the string s is read}
end for
return $n.isAcceptNode()$ {if the node n is an accept or reject node}

Algorithm 5: DFA: getFitness

Input: Graph g
Output: An integer value f representing the fitness of Graph g
 $f \leftarrow 0$
for $j = 0$ to *number of fitness cases* **do**
 Boolean $accepted \leftarrow isAccepted(fitness\ case\ j)$ {See Algorithm 4}
 if fitness case j should be accepted by the FA && $accepted==true$ **then**
 increment f
 else if fitness case j should be rejected by the FA && $accepted==false$ **then**
 increment f
 end if
end for
return f

6.2.1.4 GP Parameters for DFAs

The GP parameters used by the system are outlined in Table 6.1.

| | |
|-------------------------|---|
| Objective | Evolve a finite state machine that accepts a certain language L |
| Population Size | 2000 |
| Selection method | Tournament selection |
| Tournament size | 5 |
| Initial number of Nodes | 5 |
| Max number of Nodes | 10 |
| GP operator rates | Crossover=85%, Reproduction=10%, Mutation=5% |
| Maximum generations | 50 |
| Raw fitness | The number of correctly classified sentences |
| Termination Criteria | A solution has been found or 50 generations are completed. |

Table 6.1: GP Parameters used for the induction of DFAs

6.2.1.5 Language Set

The fitness cases used in attempting to induce a finite state machine are a set of sample sentences. Each set consists of both positive and negative sample sentences where the positive sentences are accepted by the finite state machine and the negative sentences are rejected. The languages used to test the system are shown in Table 5.1 in Chapter 5.

If any brittle solutions were found the number of fitness cases for that language was increased. A solution is termed brittle if, despite correctly classifying all the sentences in the training set, the FA generated does not correctly classify all the sentences in the test set. For each language where $\Sigma = \{a,b\}$, the test set contained all strings up to a maximum length of 15. For each language where $\Sigma = \{a,b,c\}$ on the other hand, the test set contained all strings up to a maximum length of ten.

The number of fitness cases for each language is listed in Table 6.2 below.

| Language | Fitness cases |
|----------|---------------|
| L1 | 100 |
| L2 | 105 |
| L3 | 165 |
| L4 | 110 |
| L5 | 305 |
| L6 | 270 |
| L7 | 105 |
| L8 | 100 |
| L9 | 240 |
| L10 | 220 |
| L11 | 160 |
| L12 | 410 |
| L13 | 100 |
| L14 | 100 |
| L15 | 295 |

Table 6.2: Number of fitness cases used for inducing DFAs

6.2.2 Nondeterministic Finite Acceptors (First Approach)

6.2.2.1 Representation of an Individual

Each individual of the population is represented using a directed graph. The representation of the individual is similar to that used in the system for deterministic finite acceptors. In this first approach to generate NFAs (NFA1), the individuals generated do not have to have a transition for each element of the alphabet [52]. In this system, there is a restriction that an individual must not have multiple transitions for an element of the alphabet. Some literature define this type of acceptors as being a DFA which is not complete [16]. To simplify the interpretation, ϵ -transitions are not allowed. The arity of each node is variable and is randomly chosen to be less than or equal to the size of the alphabet. As with DFAs each state is numbered sequentially where the state numbered zero is the start state for NFAs. Duplicate individuals are allowed in the initial population. Figure 6.11 shows how an individual is created. The creation is similar to that of the DFA graph. The alphabet for this NFA is $\{a,b,c\}$. Initially a start state is added to the graph and is randomly chosen to be either an accept state or a reject state. The arity of the state is also chosen randomly. The maximum arity allowed for each state is three which is the size of the alphabet in this example. In Figure 6.11(a), the start state is chosen to be a reject state and the arity is chosen to be two. The character to be read is chosen randomly for each transition. For the first transition leaving the start state the character is randomly chosen to be 'a'. For the second transition leaving the start state the character is randomly chosen to be 'c'. A character can only be chosen once for transitions leaving a state e.g., for the second transition leaving the start state the character 'a' cannot be chosen as it was used in the first transition. Figure 6.11(d) shows the resulting NFA created.

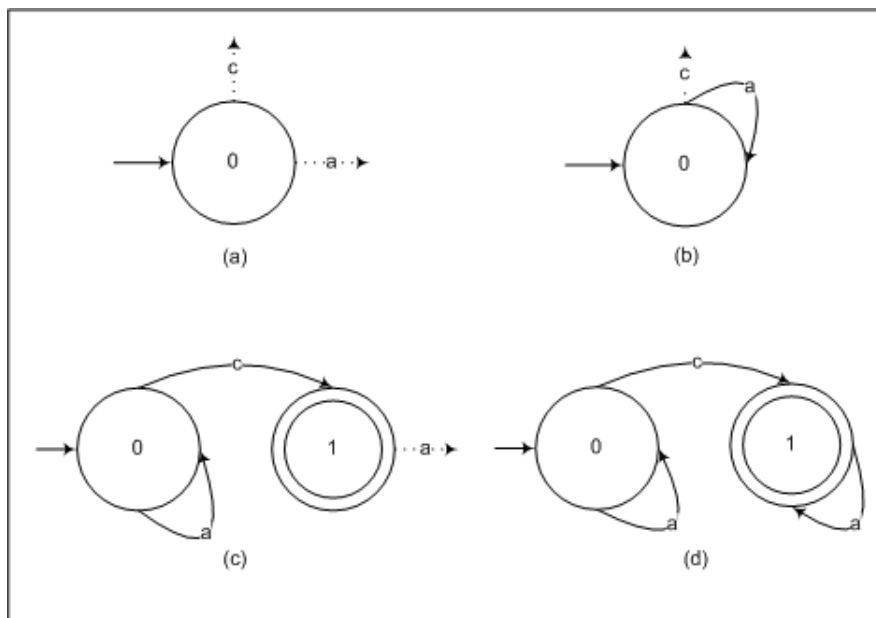


Figure 6.11: Creation of an NFA1

Figure 6.12 below shows an example representation of an individual for the language a^*b^* .

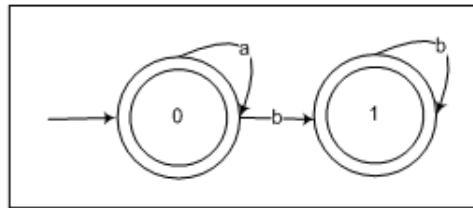


Figure 6.12: NFA1 for a^*b^*

The following syntax is used by the program for the NFA in Figure 6.12:

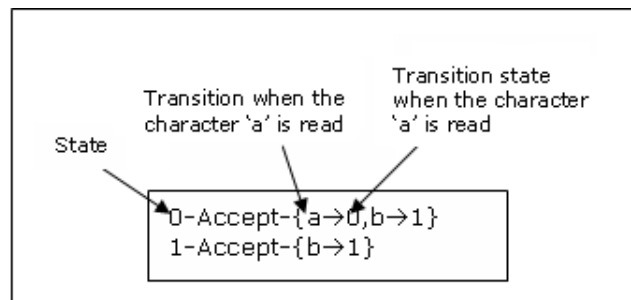


Figure 6.13: Syntax for NFA1

6.2.2.2 Genetic Programming Operators

The genetic programming operators used are reproduction, crossover and mutation. There is no difference in the operation of the GP operators used by the DFA system and this system.

6.2.2.3 Interpretation

The interpretation for this approach is similar to that used by the DFA system and is described in Algorithm 6. For NFA1, a transition may not exist for a character in the alphabet. If there is no transition the *getTargetNode* method returns NULL. An additional check is therefore required to check if the target node is NULL, and therefore false is returned if no transition exists, indicating that the input string is not accepted.

Algorithm 6: NFA: isAccepted

Input: String s
Output: Boolean value indicating if the string s is accepted by the Graph G
Node $n \leftarrow$ the initial/start node of Graph g
for $i = 0$ to *length of string s* **do**
 $n \leftarrow n.getTargetNode(s.charAt(i))$ {the target node when the character at position i of the string s is read}
 if n is NULL **then**
 return false
 end if
end for
return $n.isAcceptNode()$ {if the node n is an accept or reject node}

6.2.2.4 GP Parameters

The GP Parameters used by the system are outlined in Table 6.3.

| | |
|-------------------------|---|
| Objective | Evolve a non-deterministic finite state machine that accepts a certain language L |
| Population Size | 2000 |
| Selection method | Tournament selection |
| Tournament size | 5 |
| Initial number of Nodes | 5 |
| Max number of Nodes | 10 |
| GP operator rates | Crossover=85%, Reproduction=10%, Mutation=5% |
| Maximum generations | 50 |
| Raw fitness | The number of correctly classified sentences |
| Termination Criteria | A solution has been found or 50 generations are completed. |

Table 6.3: GP Parameters used for the induction of NFAs(First Approach)

6.2.2.5 Language set

As with DFAs, the fitness cases used in attempting to induce NFAs is a set of sample sentences. The language set used is the same as that used for DFAs which is shown in Table 5.1 in Chapter 5.

The fitness cases were chosen in a similar method to that used for DFAs. The number of fitness cases used for each language is increased until there are no longer any brittle solutions. The number of fitness cases for each language is listed in Table 6.4 below.

| Language | Fitness cases |
|----------|---------------|
| L1 | 110 |
| L2 | 105 |
| L3 | 130 |
| L4 | 115 |
| L5 | 110 |
| L6 | 270 |
| L7 | 105 |
| L8 | 100 |
| L9 | 200 |
| L10 | 350 |
| L11 | 250 |
| L12 | 100 |
| L13 | 100 |
| L14 | 100 |
| L15 | 295 |

Table 6.4: Number of fitness cases used for inducing NFAs (First Approach)

6.2.3 Nondeterministic Finite Acceptors (Second Approach)

6.2.3.1 Representation of an Individual

Each individual of the population is represented using a directed graph. The representation of the individual is the same as that used in the first approach. In this second approach to generate NFAs (NFA2), the individuals generated do not have to have a transition for each symbol as in the first approach but an individual can also have multiple transitions for a symbol. The system therefore generates completely non-deterministic finite acceptors however, ϵ -transitions are not allowed. This restriction is imposed to reduce the complexity of the interpretation. The arity of each node is variable and is randomly chosen to be less than or equal to a maximum transitions GP parameter. As with DFAs each state is numbered sequentially where the state numbered zero is the start state for NFAs. Duplicate individuals are allowed in the initial population. Figure 6.14 shows how an individual is created. The creation is similar to that of the DFA graph. The alphabet for this NFA is $\{a,b\}$. As with the DFA system and the NFA1 system, a start state is added to the graph and is randomly chosen to be either an accept state or a reject state. The arity of the state is also chosen randomly. In Figure 6.14(a), the start state is chosen to be a reject state and the arity is chosen to be two. The character to be read is chosen randomly for each transition. Both transitions leaving the start state are randomly chosen to be 'a'. Figure 6.14(d) is the resulting NFA.

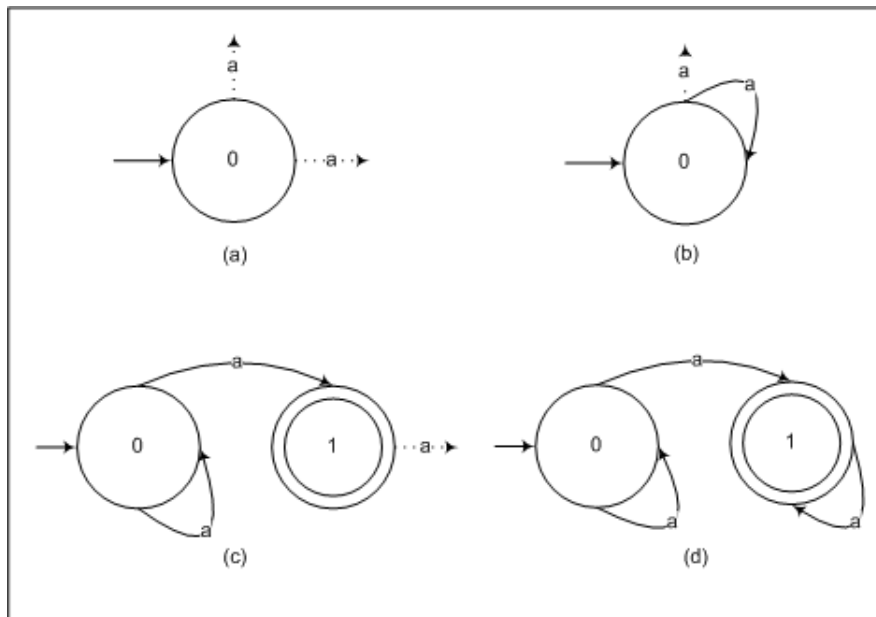


Figure 6.14: Creation of an NFA2

Figure 6.15 below shows an example representation of an individual for the language a^*b^* .

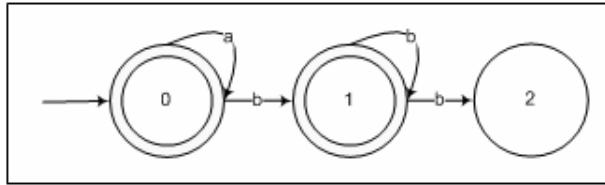


Figure 6.15: NFA2 for a^*b^*

The following syntax is used by the program for the NFA in Figure 6.15:

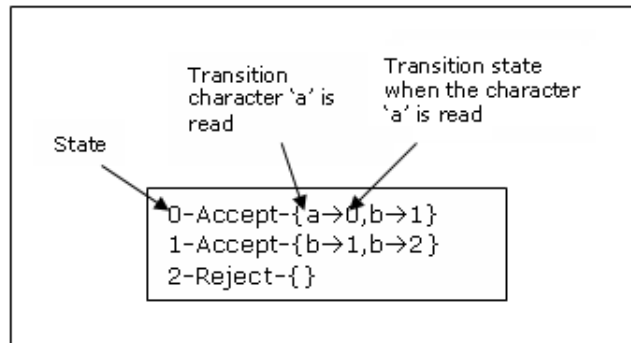


Figure 6.16: Syntax for NFA2

6.2.3.2 Genetic Programming Operators

The genetic programming operators used are reproduction, crossover and mutation. There is no difference in the operation of the GP operators used by the DFA system and this system.

6.2.3.3 Interpretation

As with the DFA system and the NFA1 system, the fitness value for an individual is determined by obtaining the number of correctly classified sentences. Algorithm 7 below provides details on how a string is classified by the system. The algorithm for NFA2 takes into consideration the possibility of having multiple transitions for a character. A string is accepted if at least one of the paths ends in an accept state. A string is rejected if all paths end with a reject state

Algorithm 7: NFA2: isAccepted

Input: String s {fitness case being evaluated}
Input: Node $start$ {node in the graph to start evaluating at}
Input: int $position$ {character position in string s }
Input: int $path$ {used if there is more than one transition for a character}
Output: Boolean value indicating if the string s is accepted by the Graph G
Node $n \leftarrow start.getTargetNode(s.charAt(position), path)$
for $i = position + 1$ to *length of string s* **do**
 int $no_of_paths \leftarrow n.getPaths(s.charAt(i))$ {the number of transitions leaving state n for character at position i in string s }
 if $no_of_paths == 0$ **then**
 return false {no paths so string is rejected}
 else
 for $p = 0$ to no_of_paths **do**
 boolean $acc = isAccepted(s, n, i, p + 1)$
 if $acc == true$ **then**
 return true
 end if
 end for
 return false
 end if
end for
return $n.isAcceptNode()$ {if the node n is an accept or reject node}

If the string has been correctly classified the fitness value of the individual is incremented as is shown by Algorithm 8. If the fitness value is equal to the number of fitness cases then the graph is deemed to be a solution.

Algorithm 8: NFA2: getFitness

Input: Graph g

Output: An integer value f representing the fitness of the graph G

```
int  $f \leftarrow 0$ 
Node  $first \leftarrow$  first node of  $G$ 
for  $j = 0$  to number of fitness cases do
  boolean  $accepted = false$ 
  int  $no\_of\_paths \leftarrow first.getPaths(fitness\_case[j].charAt(0))$ 
  if  $no\_of\_paths == 0$  then
     $accepted = false$ 
  else
    for  $p = 0$  to  $no\_of\_paths$  do
       $accepted = isAccepted(fitness\ case\ j, first, 0, p + 1)$ 
      if  $accepted == true$  then
        break
      end if
    end for
  end if
  if fitness case  $j$  should be accepted by the FA &&  $accepted == true$  then
     $f \leftarrow f + 1$ 
  else if fitness case  $j$  should be rejected by the FA &&  $accepted == false$  then
     $f \leftarrow f + 1$ 
  end if
end for
return  $f$ 
```

6.2.3.4 GP Parameters

The GP parameters used to test the system are exactly the same as that used in the first approach (NFA1). There is one additional GP parameter for this system i.e. maximum transitions, which specifies the maximum number of transitions a node can have. The system was tested with a maximum number of transitions set at 4.

6.2.3.5 Language set

As with DFAs, the fitness cases used in attempting to induce NFAs is a set of sample sentences. The language set used is the same as that used for DFAs which is shown in Table 5.1 in Chapter 5.

The number of fitness cases used for each language is increased until there are no longer any brittle solutions. The number of fitness cases for each language is listed in Table 6.5 below.

| Language | Fitness cases |
|-----------------|----------------------|
| L1 | 115 |
| L2 | 110 |
| L3 | 145 |
| L4 | 150 |
| L5 | 165 |
| L6 | 275 |
| L7 | 115 |
| L8 | 100 |
| L9 | 220 |
| L10 | 240 |
| L11 | 251 |
| L12 | 100 |
| L13 | 170 |
| L14 | 140 |
| L15 | 356 |

Table 6.5: Number of fitness cases used for inducing NFAs (Second Approach)

6.3 Finite State Transducers

The section that follows describes the genetic programming system used to induce Finite State Transducers (FST). The genetic programming system induces Mealy machines. It is unnecessary to induce Moore machines, as an algorithm exists to convert Mealy machines to Moore machines¹.

6.3.1 Representation of An Individual

Each individual of the population is represented using a directed graph. An individual is represented having two characters on the edge e.g. 0/1 where '0' is the input character and '1' is the output character. The arity of each node is the size of the alphabet. All individuals of the population are classed as complete given that for each state there is a transition to another state for each input symbol of the alphabet. Each state is numbered sequentially where the state numbered 0 is the start state. In contrast to finite acceptors, a node is not classified as being an accept or reject state. Figure 6.17 shows an example of how an individual is created. The input and output alphabet of this FST is {0,1}. Initially, a start state is added to the graph as is shown in Figure 6.17(a). Each state added to the graph has an arity of two which is the size of the alphabet. Figure 6.17(a) is incomplete as the transition for '0' and the transition for '1' is not connected to an existing state. As with FAs, if the graph is incomplete there are two options for a transition that is not connected to an existing state:

1. Add a new state and connect the transition to the newly created state.
2. Connect the transition to a state that already exists in the graph.

If the number of nodes in the graph reaches the maximum number of nodes allowed only option 2 can be performed. Figure 6.17(b) is an example where option 2 is chosen for the transition for '0'. As there is only one existing state, '0' connects to state 0. In addition, an output character is randomly chosen for the transition. Figure 6.17(b) shows that '1' has been chosen as the output character. Figure 6.17(c) is an example where option 1 is chosen for the transition '1'. Once again the output character is chosen randomly to be '1'. Figure 6.17(d) is the resulting FST.

¹See Chapter 2, Section 2.2.2 for the formal definitions of a Mealy machine and a Moore machine

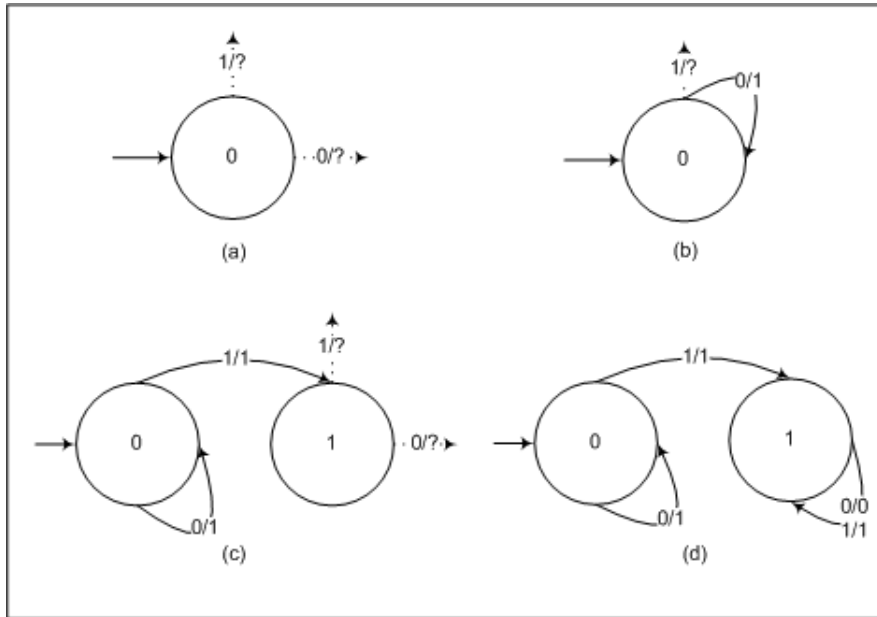


Figure 6.17: Creation of an FST

Figure 6.18 below shows an example representation of an individual.

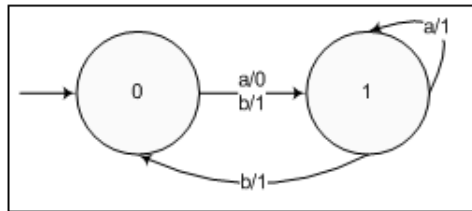


Figure 6.18: Example Finite State Transducer

The following syntax is used by the program for the FST in Figure 6.18

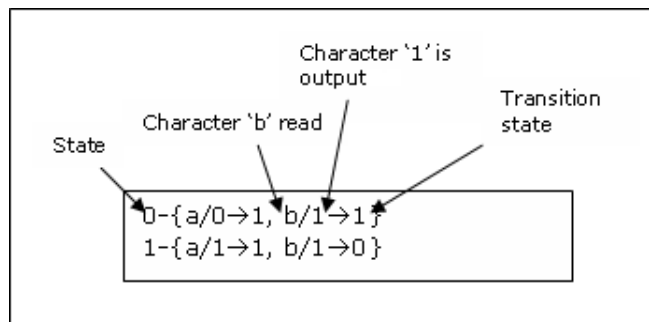


Figure 6.19: Syntax for FST

6.3.2 Genetic Programming Operators

The genetic operators used are reproduction, crossover and mutation. The algorithms for these genetic programming operators are the same as that employed by the GP systems that generate FAs.

To apply crossover, the system randomly selects crossover points in the selected parents as indicated by the shaded nodes in Figure 6.20.

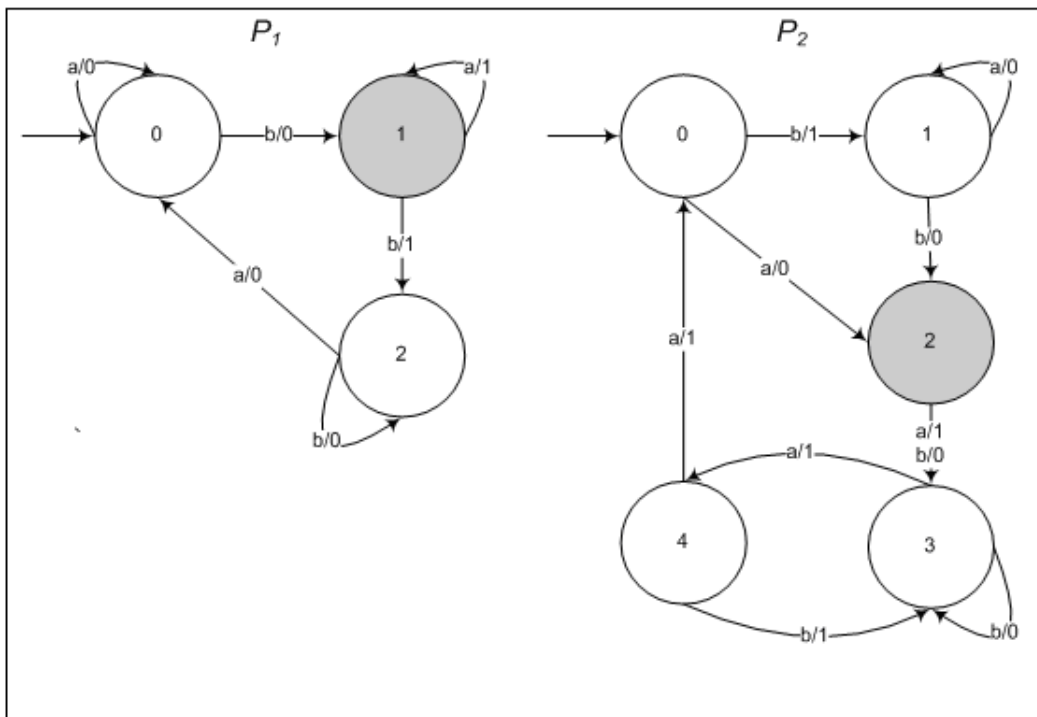


Figure 6.20: Example of selected Parents for Crossover

The subgraphs rooted at these points are swapped. Figure 6.21 shows the resulting subgraphs. As with FAs, in each subgraph the arcs are labelled *internal* if they point to nodes within the subgraph, and are labelled *external* otherwise.

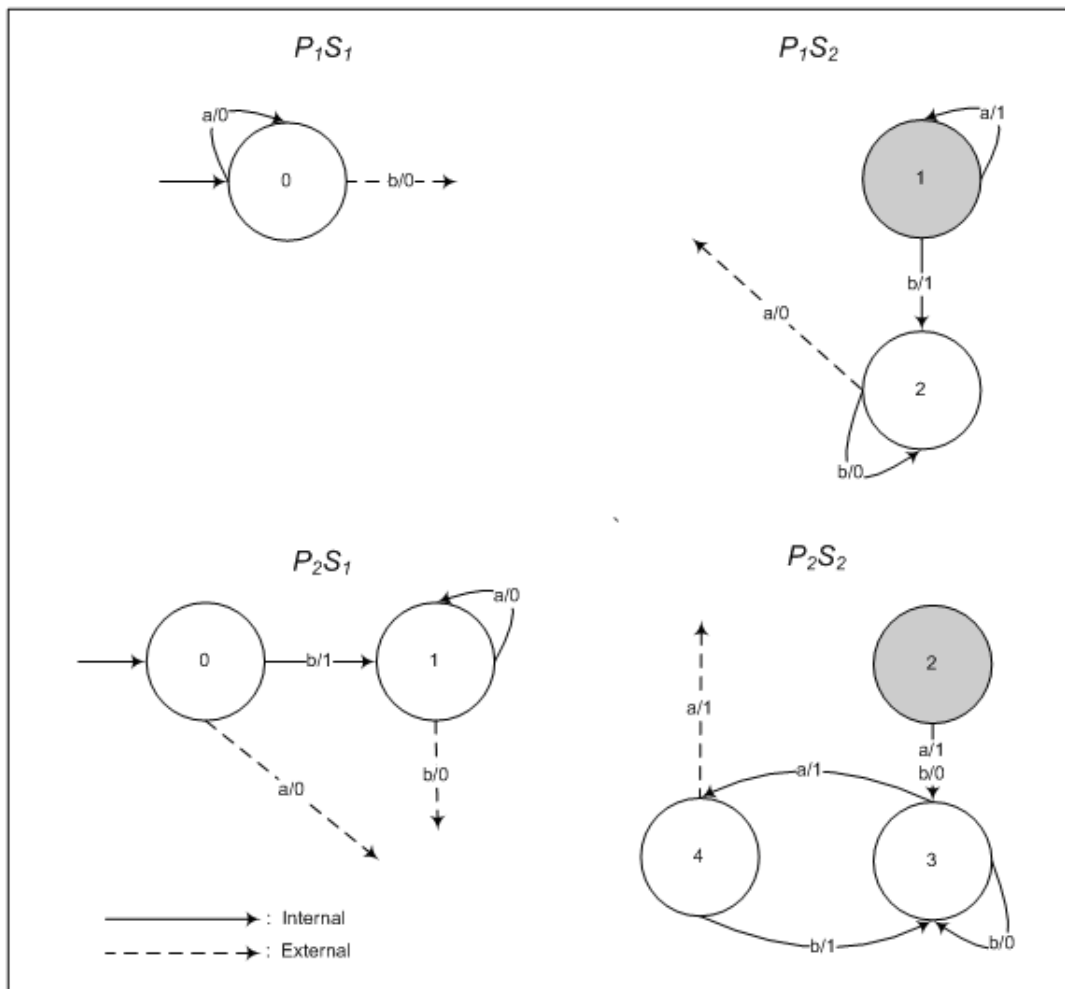


Figure 6.21: Example of the resulting subgraphs

Figure 6.22 illustrates the resulting offspring when swapping the subgraphs.

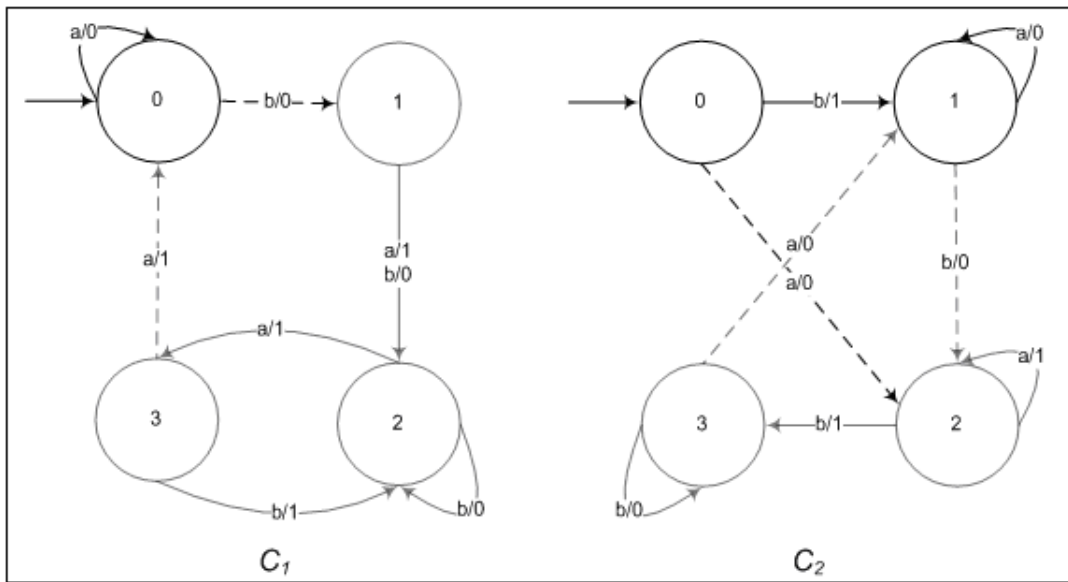


Figure 6.22: Example of the offspring created when applying crossover

For mutation, a mutation point is randomly selected in the parent as shown in Figure 6.23.

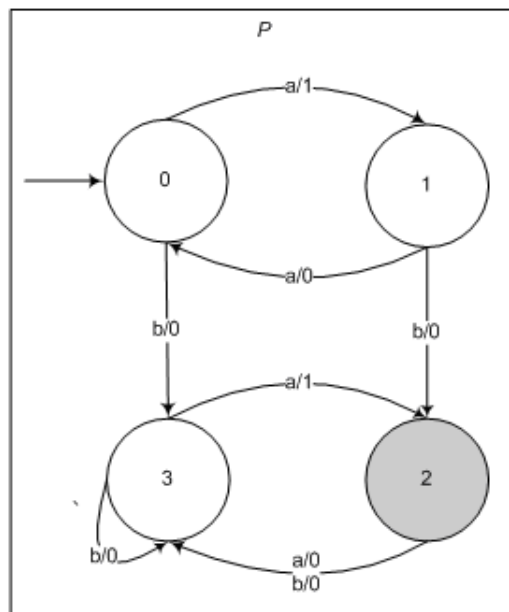


Figure 6.23: Example of a selected mutation point in a Parent

The subgraph rooted at the mutation point is then removed. An example of the remaining subgraph is shown in Figure 6.24.

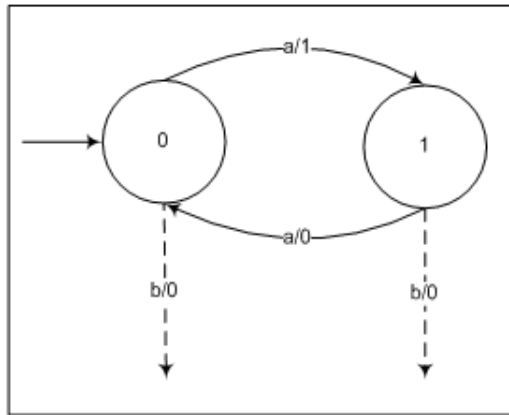


Figure 6.24: Example of the subgraph

A newly created subgraph is then inserted as illustrated in Figure 6.25.

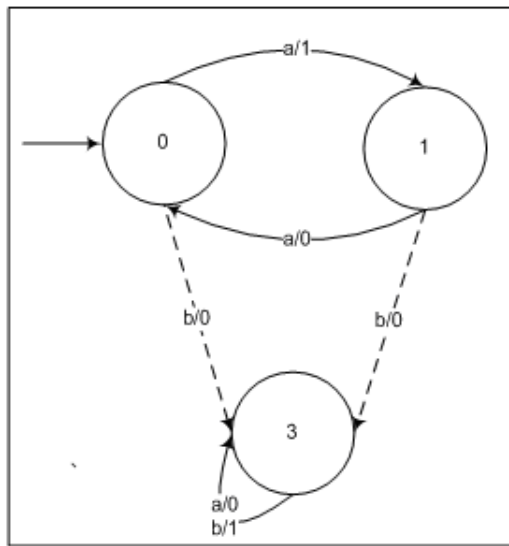


Figure 6.25: Example of the offspring created when applying mutation

6.3.3 Interpretation

The fitness of the graph is dependent on the number of correctly classified sentences. A sentence is correctly classified if the input sentence for the FST produces the expected output sentence.

Algorithm 9: FST: getOutputString

Input: String s
Output: String value is outputted representing the output by the Graph g when s is inputted
Node $n \leftarrow$ the initial/start node of Graph g
String $str \leftarrow$ "" {Empty String}
for $i = 0$ to *length of string s* **do**
 $str \leftarrow str + n.getOutputChar(s.charAt(i))$ {the output character when character at position i of the string s is read}
 $n \leftarrow n.getTargetNode(s.charAt(i))$ {the target node when the character at position i of the string s is read}
end for
return str

Algorithm 10: FST: getFitness

Input: Graph g
Output: An integer value f representing the fitness of Graph g
 $f \leftarrow 0$
for $i = 0$ to *number of fitness cases* **do**
 String $output \leftarrow getOutputString(fitness\ case\ j)$ {See Algorithm 9}
 if $output$ is the same as the expected output for fitness case j **then**
 increment f
 end if
end for
return f

6.3.4 GP Parameters for FSTs

The GP Parameters used by the system are outlined in Table 6.6.

| | |
|-------------------------|---|
| Objective | Evolve a Mealy Machine |
| Population Size | 2000 |
| Selection method | Tournament selection |
| Tournament size | 5 |
| Initial number of Nodes | 3 |
| Max number of Nodes | 6 |
| GP operator rates | Crossover=85%, Reproduction=10%, Mutation=5% |
| Maximum generations | 50 |
| Raw fitness | The number of fitness cases for which the graph provides the target output for the given input. |
| Termination Criteria | A solution has been found or 50 generations are completed. |

Table 6.6: GP Parameters used for the induction of FSTs

6.3.5 Language set

The fitness cases used in attempting to induce a finite state transducer are a set of sample sentences. Each set consists of an input sentence and a corresponding output. The language set used for each language is shown in Table 5.2 in Chapter 5.

As with the FAs, if any brittle solutions were found, the number of fitness cases were increased. For each language the test set contained all input strings up to a maximum length of 15. The number of fitness cases used to test each language is outlined in Table 6.7.

| Language | Fitness cases |
|----------|---------------|
| L1 | 54 |
| L2 | 48 |
| L3 | 20 |
| L4 | 56 |
| L5 | 20 |
| L6 | 20 |

Table 6.7: Number of fitness cases used for inducing FSTs

6.4 Pushdown Automata

The section that follows describes the genetic programming system used to induce nondeterministic pushdown automata (NPDA).

6.4.1 Representation of an Individual

As with finite acceptors, each individual of the population is represented using a directed graph as opposed to a tree-based structure. Each individual is represented having 3 characters on the edge e.g. $a,b;c$ where 'a' is the character that is read, 'b' is the character popped off the stack and 'c' is the character that is pushed onto the stack. The directed edges of the graph represent the transitions of the NPDA. The arity of each node is varied and is randomly chosen to be less than 'maximum transitions', which is a GP parameter. Each state is numbered sequentially where the state numbered 0 is the start state. Figure 6.26 shows an example of how an individual is created. The input alphabet of this PDA is $\{a,b,\lambda\}$ and the stack alphabet of this PDA is $\{A,B,\lambda\}$. Initially, a start state is added to the graph and is chosen to be either an accept state or a reject state. In Figure 6.26(a), the initial state is numbered '0' and is randomly chosen to be a reject state. The arity of the initial state is randomly chosen to be two. Figure 6.26(a) is incomplete as the two transitions are not connected to an existing state. If the graph is incomplete there are two options for a transition that is not connected to an existing state:

1. Add a new state and connect the transition to the newly created state.
2. Connect the transition to a state that already exists in the graph.

If the number of nodes in the graph reaches the maximum number of nodes allowed only option 2 can be performed. Figure 6.26(b) is an example where option 2 is chosen for the first transition. As there is only one existing state, the first transition connects to state 0. The character to be read is randomly chosen to be 'a', the character to be popped off the stack is randomly chosen to be ' λ ' (which implies that nothing will be popped off the stack), and the character to be pushed onto the stack is randomly chosen to be 'A'. Figure 6.26(c) is an example where option 1 is chosen for the second transition. The arity of the newly created state is randomly chosen to be three. Figure 6.26(d) is the resulting PDA.

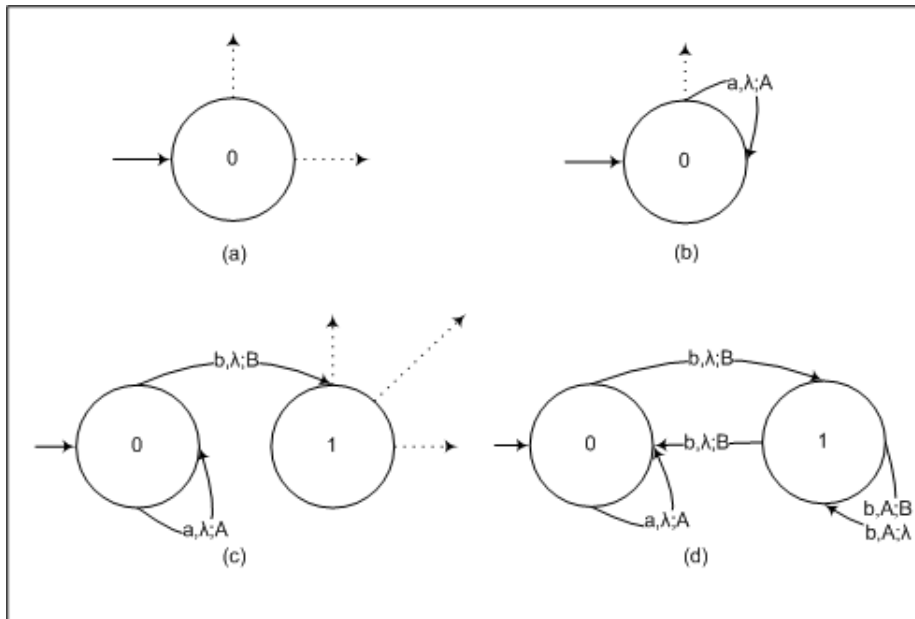


Figure 6.26: Creation of a PDA

Figure 6.27² below shows an example representation of an individual for the language $a^n b^n$.

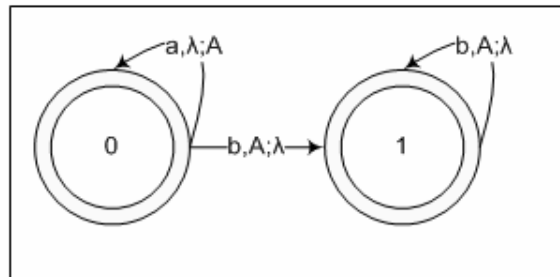


Figure 6.27: PDA for $a^n b^n$

The following syntax is used by the program for the NPDA in Figure 6.27:

²'λ' - In most literature 'λ' represents the empty string. The GP system outputs 'λ' to represent the empty string.

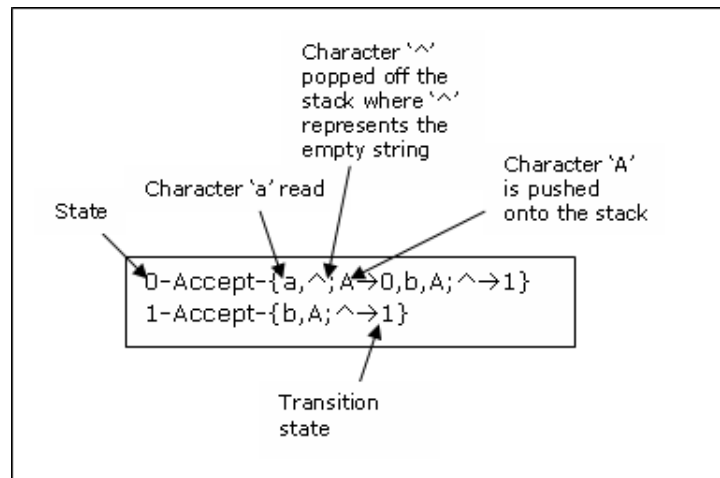


Figure 6.28: Syntax for PDA

6.4.2 Genetic Programming Operators

The genetic programming operators used are reproduction, crossover and mutation.

Crossover The algorithm for crossover is the same as that employed by the GP systems that generate FAs. The parent graphs are chosen using tournament selection. A crossover point is chosen in each graph as is shown in Figure 6.29.

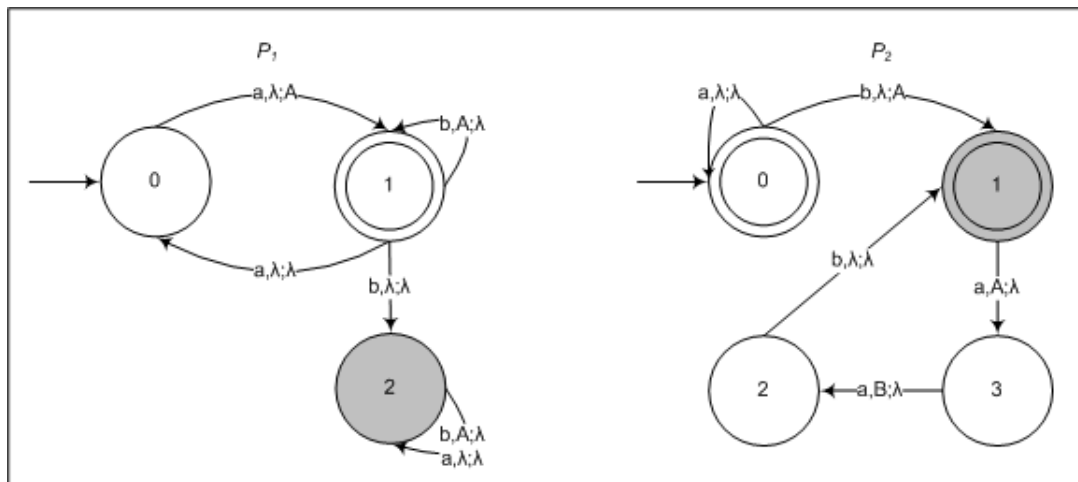


Figure 6.29: Chosen parents for Crossover

Each parent graph is split into two subgraphs indicated by P_1S_1 , P_1S_2 , P_2S_1 and P_2S_2 in Figure 6.30. As with FAs, in each subgraph the arcs are labelled *internal* if they point to nodes within the subgraph, and are labelled *external* otherwise.

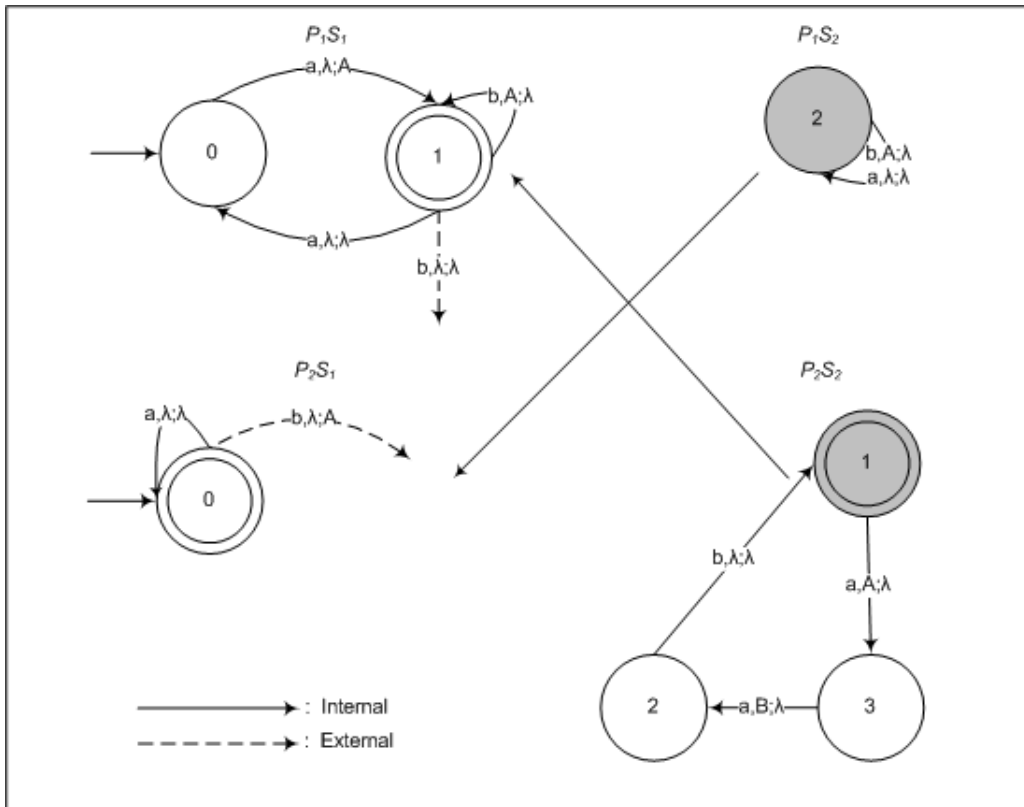


Figure 6.30: Resulting Subgraphs

The children are created by combining the subgraphs P_1S_1 and P_2S_2 to form C_1 and P_2S_1 and P_1S_2 to form C_2 . Figure 6.31 shows the resulting children C_1 and C_2 .

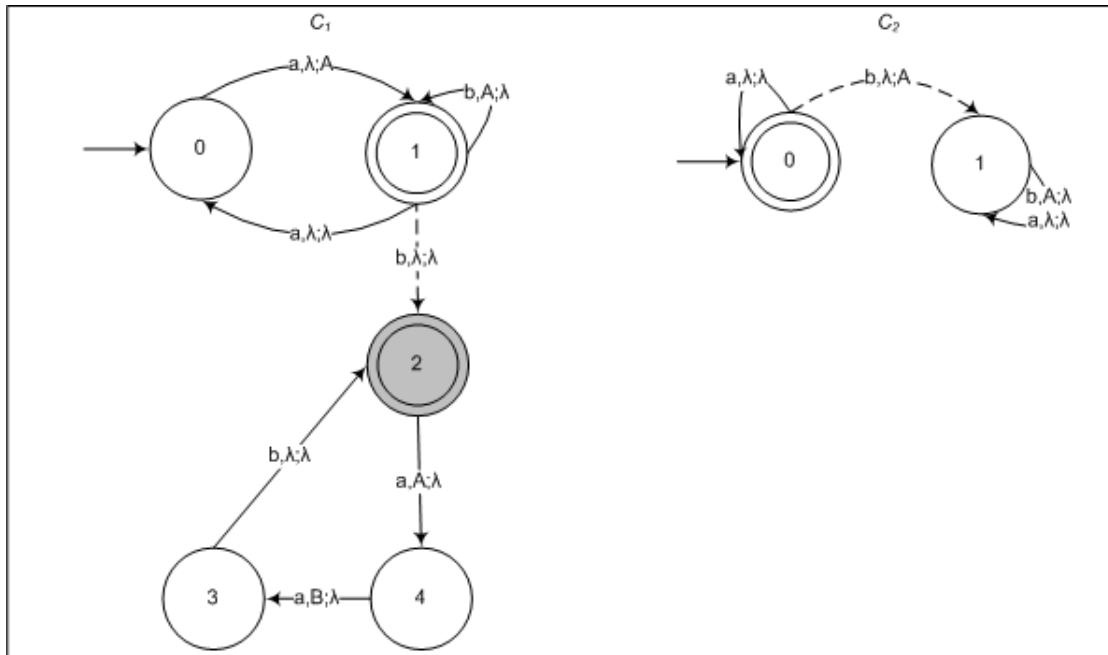


Figure 6.31: Resulting children after performing Crossover

Mutation An individual is selected to undergo mutation using tournament selection. Thereafter a mutation point is randomly chosen in the selected individual as is shown in Figure 6.32a. As with the mutation operator for the FAs, the subgraph rooted at the mutation point is removed and replaced by a new randomly created subgraph as is shown in Figure 6.32b and Figure 6.32c. Mutation can also occur by randomly selecting a transition and altering the transition in one of the following ways:

- Changing the character read.
- Changing the character pushed onto the stack.
- Changing the character popped off the stack.
- Changing the state to which the transition points.

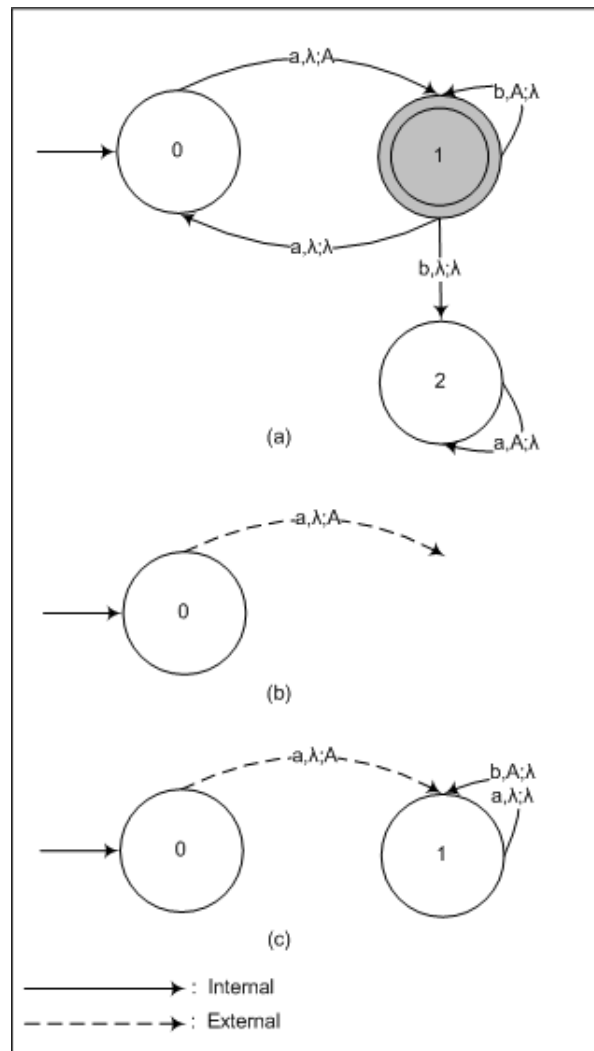


Figure 6.32: Example Mutation

6.4.3 Interpretation

The fitness value for an individual is obtained by calculating the number of correctly classified sentences. During interpretation a *max_evaluations* parameter is used. This ensures that no infinite loop occurs when the empty string is read. The system is tested by setting *max_evaluations* to 50. A 'Path' class is used to record the information needed when multiple transitions are encountered. Figure 6.33 shows the code for the 'Path' class.

```

class Path {
    Node next;
    int posInString;
    int nextPath;
    Stack s;
}

```

Figure 6.33: Code Example for PDAs

A transition is not only dependent on the character being read in but also the character being popped off the stack. If there are multiple transitions, the first transition is evaluated and the remaining transitions are stored in a vector, resulting in a vector of Paths. Each Path is then evaluated until the vector is empty. A string is accepted if one of the 'Paths' end in an accept state and the stack is empty. A string is rejected if all 'Paths' end in a reject state.

6.4.4 GP Parameters for PDAs

The GP Parameters used by the system are outlined in Table 6.8.

| | |
|-------------------------|--|
| Objective | Evolve a pushdown automaton that accepts a certain language L. |
| Population Size | 2000 |
| Selection method | Tournament selection |
| Tournament size | 5 |
| Initial number of Nodes | 2 |
| Max number of Nodes | 5 |
| GP operator rates | Crossover=60%, Reproduction=10%, Mutation=30% |
| Maximum generations | 50 |
| Raw fitness | The number of correctly classified sentences |
| Maximum transitions | 4 for L1, L2, L5, L6, L7, L8, L9, L10, L11, L12 6 for L3 and L4 |
| Termination Criteria | A solution has been found or 50 generations are completed. |
| Alphabet Σ | {a,b} for L1, L3, L4, L6, L7, L8, L9 and L10 {a,b,c} for L2 and L5 {(,)} for L11 |
| Stack Alphabet Γ | {A,B} |

Table 6.8: GP Parameters used for the induction of PDAs

6.4.5 Language set

The fitness cases used in attempting to induce a pushdown automaton are a set of sample sentences. Each set consists of both positive and negative sample sentences where the positive sentences are accepted by the pushdown automaton and the negative sentences are rejected.

As with FAs, if any brittle solutions were found the number of fitness cases for that language was increased. For each language where $\Sigma = \{a,b\}$, the test set contained all strings up to a maximum length of 15. For each language where $\Sigma = \{a,b,c\}$ on the other hand, the test set contained all strings up to a maximum length of ten. The number of fitness cases used for each language is indicated in Table 6.9.

| Language | Fitness cases |
|----------|---------------|
| L1 | 56 |
| L2 | 157 |
| L3 | 85 |
| L4 | 385 |
| L5 | 163 |
| L6 | 262 |
| L7 | 155 |
| L8 | 69 |
| L9 | 70 |
| L10 | 201 |
| L11 | 51 |

Table 6.9: Number of fitness cases used for inducing PDAs

6.5 Turing Machines

6.5.1 Turing Machine Acceptors

The section that follows describes the genetic programming system used to induce deterministic Turing machine acceptors.

6.5.1.1 Representation of an Individual

Each individual of the population is represented using a directed graph as opposed to a tree-based structure. The system implements Turing Machines with two tapes³, where the first tape is the read-only input tape and the second is a read-write tape. Turing machines with two tapes reduces the computation time required. Each individual is therefore represented having two transitions on each edge. The first transition, which is used for the first tape (Tape A) has 2 characters on the edge e.g. a/b and the second tape (Tape B) can have either 2 or 3 characters e.g. a/b or a/c/b where ‘a’ is the character that is read from the tape, ‘b’ is the symbol that specifies the direction to move the head and ‘c’ is the symbol written to the tape. The symbol ‘b’ can be either ‘R’, ‘L’ or ‘S’ where ‘R’ indicates that the head moves right, ‘L’ indicates that the head moves left and ‘S’ indicates that the head is stationary and doesn’t move. Each state is numbered sequentially where the state numbered 1 is the start state and state 0 is the ‘HALT’ state. In the system all graphs are created having one ‘HALT’ state. The ‘HALT’ state has no outgoing transitions. The arity of a state is the number of outgoing transitions. The number of transitions leaving a state is chosen randomly to be less than or equal to the number of possible combinations for the input characters on Tape A and Tape B e.g., if $\Sigma = \{a, b, B\}$ there are nine possible combinations i.e. $\{a, b\}, \{a, a\}, \{a, B\}, \{b, a\}, \{b, b\}, \{b, B\}, \{B, B\}, \{B, a\}, \{B, b\}$. A chosen combination can only be used once for each transition leaving a particular state thereby ensuring that the Turing machine created is deterministic. A change from one state to another only occurs if the input characters of the transitions match the input characters on Tape A and Tape B.

Figure 6.34 shows an example of how an individual is created. The input and output alphabet of this TM is $\{a, b, B\}$. Initially, a ‘HALT’ state and a start state is added to the graph. In Figure 6.34(a), the start state is numbered ‘1’ and the ‘HALT’ state is numbered ‘0’. The ‘HALT’ state is given an arity of zero and the arity of the start state is chosen randomly to be two. The maximum transitions permitted in this example is nine as there are nine possible combinations. Figure 6.34(a) is incomplete as the transitions are not connected to an existing state. If the graph is incomplete there are two options for a transition that is not connected to an existing state:

1. Add a new state and connect the transition to the newly created state.
2. Connect the transition to a state that already exists in the graph.

If the number of nodes in the graph reaches the maximum number of nodes allowed only option 2 can be performed. Figure 6.34(b) is an example where option 2 is chosen for the first transition. There are two existing states, and the first transition is randomly chosen to connect to state 0. The input

³See Chapter 2, Section 2.2.4 for the definition of a multi-tape Turing machine.

characters are randomly chosen from the combinations which in this case is chosen to be $\{b,B\}$ where 'b' is the character input from Tape A and 'B' is the character input from Tape B. The movement of the head for each tape is chosen randomly to be 'S' and 'L' for Tape A and Tape B respectively. This transition will only be followed from state 1 if the input characters from Tape A and Tape B are 'b' and 'B' respectively. The combination $\{b,B\}$ cannot be used again for any of the transitions leaving state 1 as this would make the Turing machine nondeterministic. For this transition, no characters are written to Tape B. Figure 6.34(c) is an example where option 1 is chosen for the second transition. The input characters are once again chosen randomly from the remaining possible combinations to be $\{a,B\}$. The character to be written to Tape B is randomly chosen to be 'a'. Figure 6.34(d) is the resulting TM.

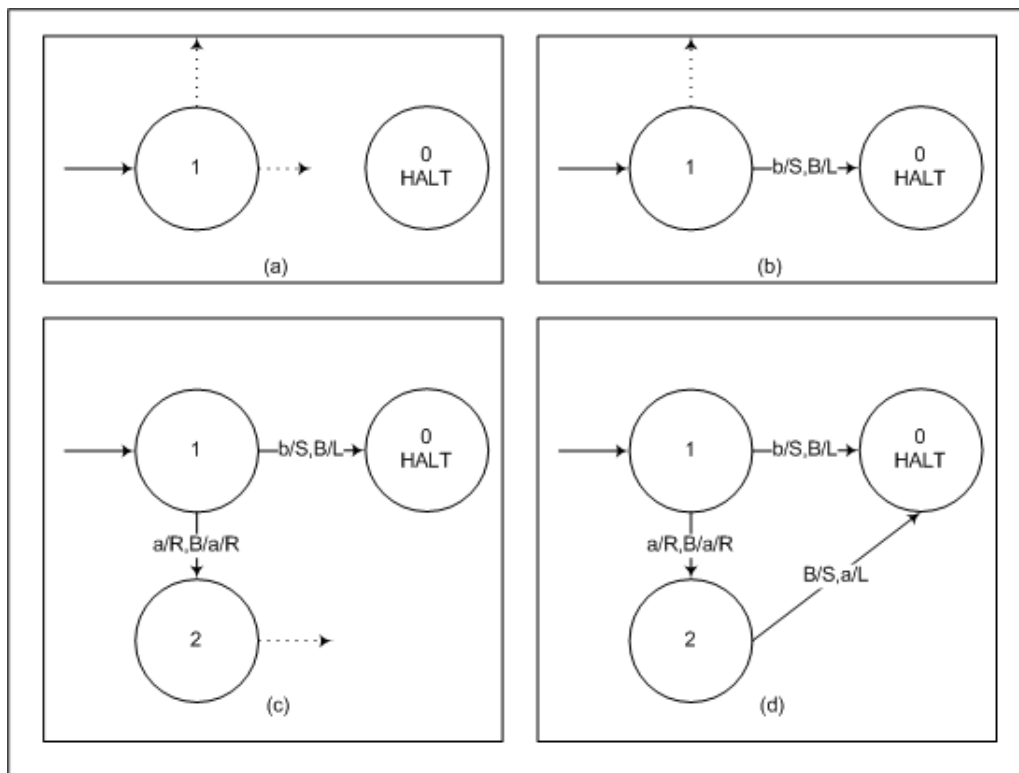


Figure 6.34: Creation of a Turing Machine

Figure 6.35 below shows an example representation of an individual.

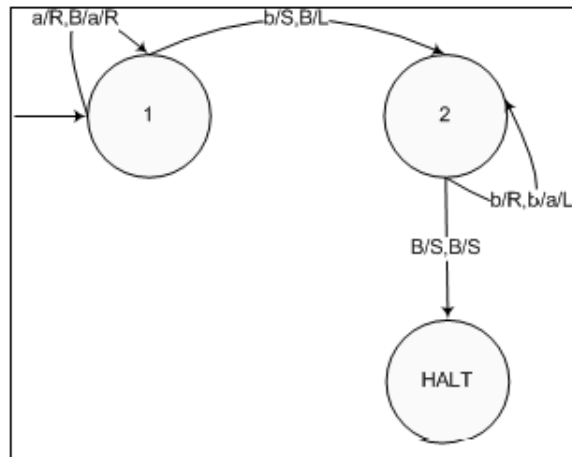


Figure 6.35: Example Turing Machine Acceptor

The following syntax is used by the program for the TM Acceptor in Figure 6.35:

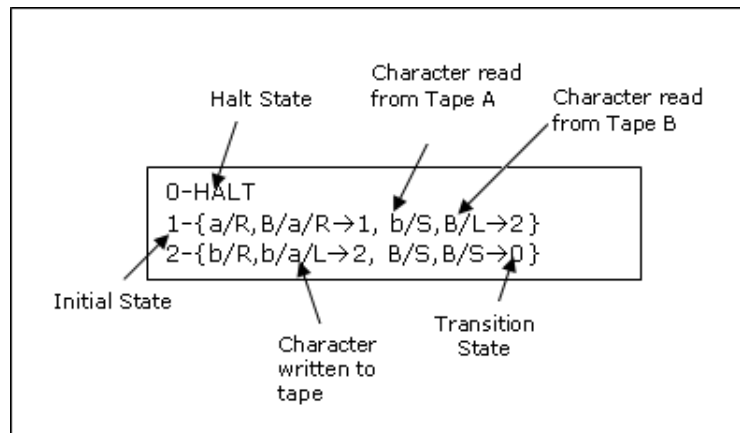


Figure 6.36: Syntax for TM Acceptors

6.5.1.2 Genetic Programming Operators

The genetic programming operators used are reproduction, crossover and mutation.

Crossover A crossover point is chosen in each graph as is shown in Figure 6.37. In each graph the 'HALT' state (state 0) is excluded from being selected as a crossover point.

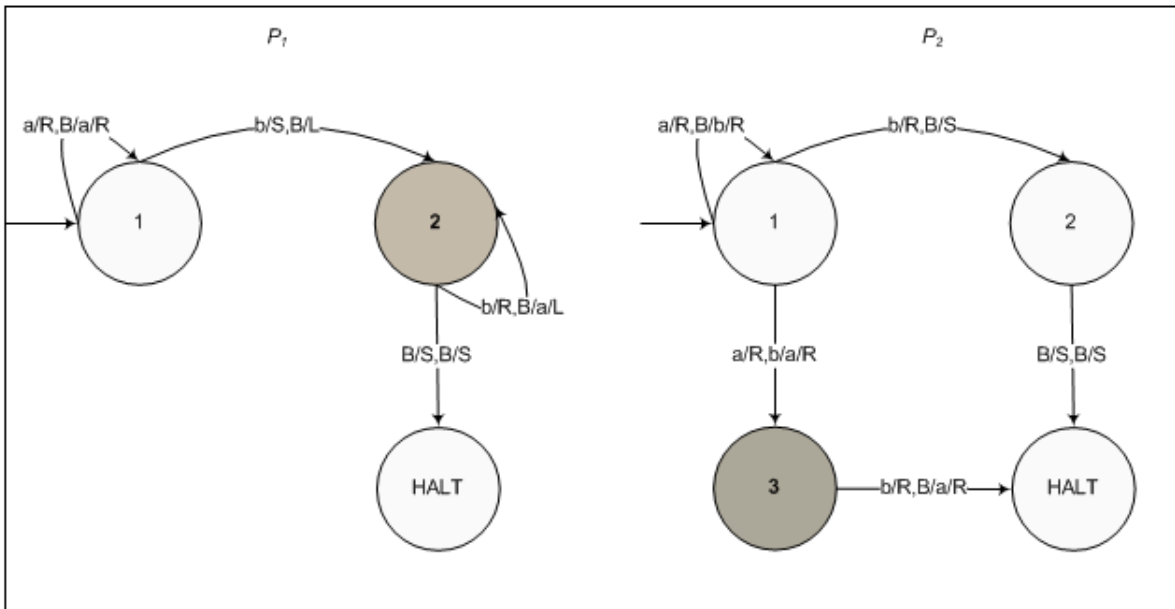


Figure 6.37: Chosen parents for Crossover

Each parent graph is split into two subgraphs indicated by P_1S_1 , P_1S_2 , P_2S_1 and P_2S_2 in Figure 6.38. As with FAs, in each subgraph the arcs are labelled *internal* if they point to nodes within the subgraph, and are labelled *external* otherwise.

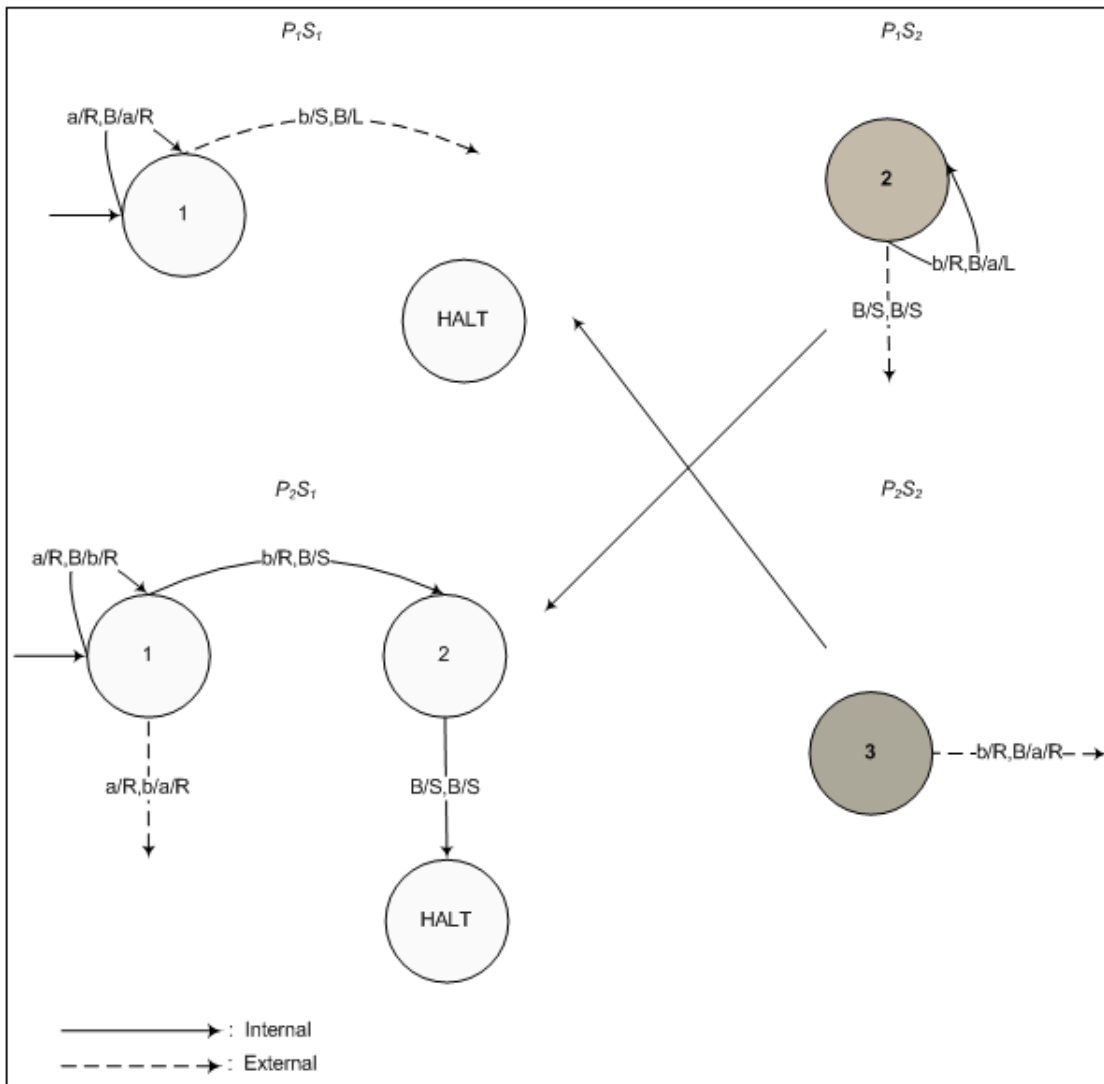


Figure 6.38: Resulting Subgraphs

The children are created by combining the subgraphs P_1S_1 and P_2S_2 to form C_1 and P_2S_1 and P_1S_2 to form C_2 . Figure 6.39 shows the resulting children C_1 and C_2 .

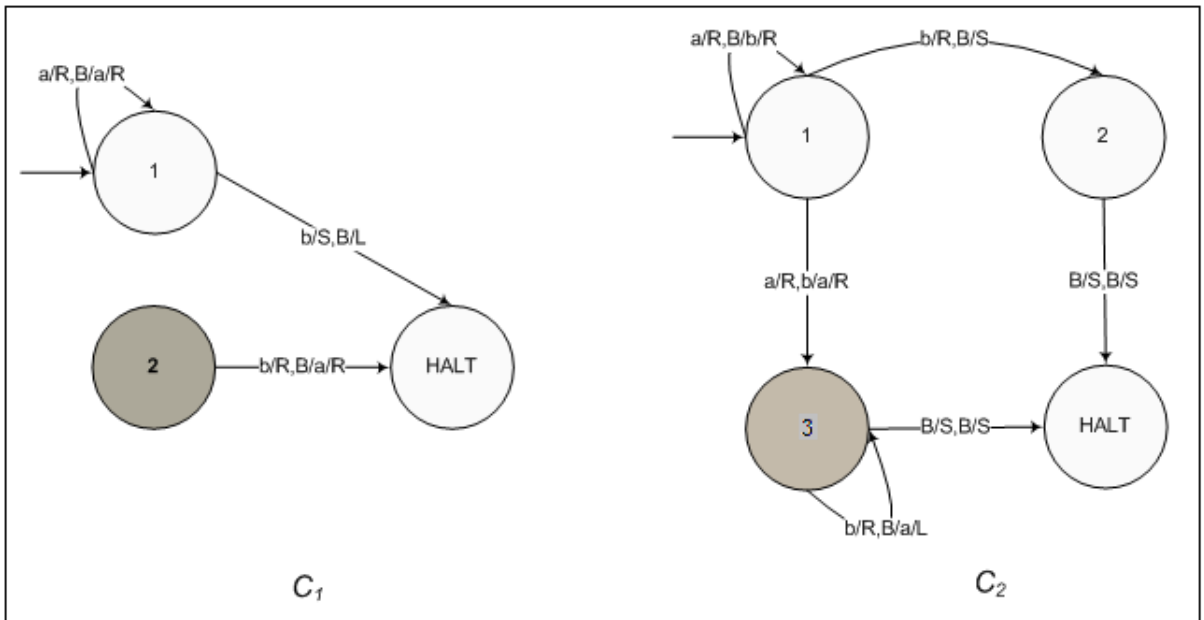


Figure 6.39: Resulting children after performing Crossover

Mutation An individual is selected to undergo mutation using tournament selection. Thereafter a mutation point is randomly chosen in the selected individual. As with the mutation operator for the FAs, the subgraph rooted at the mutation point is removed and replaced by a new randomly created subgraph as is shown in Figure 6.40. Figure 6.40(c) shows the resulting child. The resulting child has no incoming transitions for the HALT state and thus the HALT state is a redundant state as it can never be reached. This also implies that all sentences will be rejected by this Turing machine.

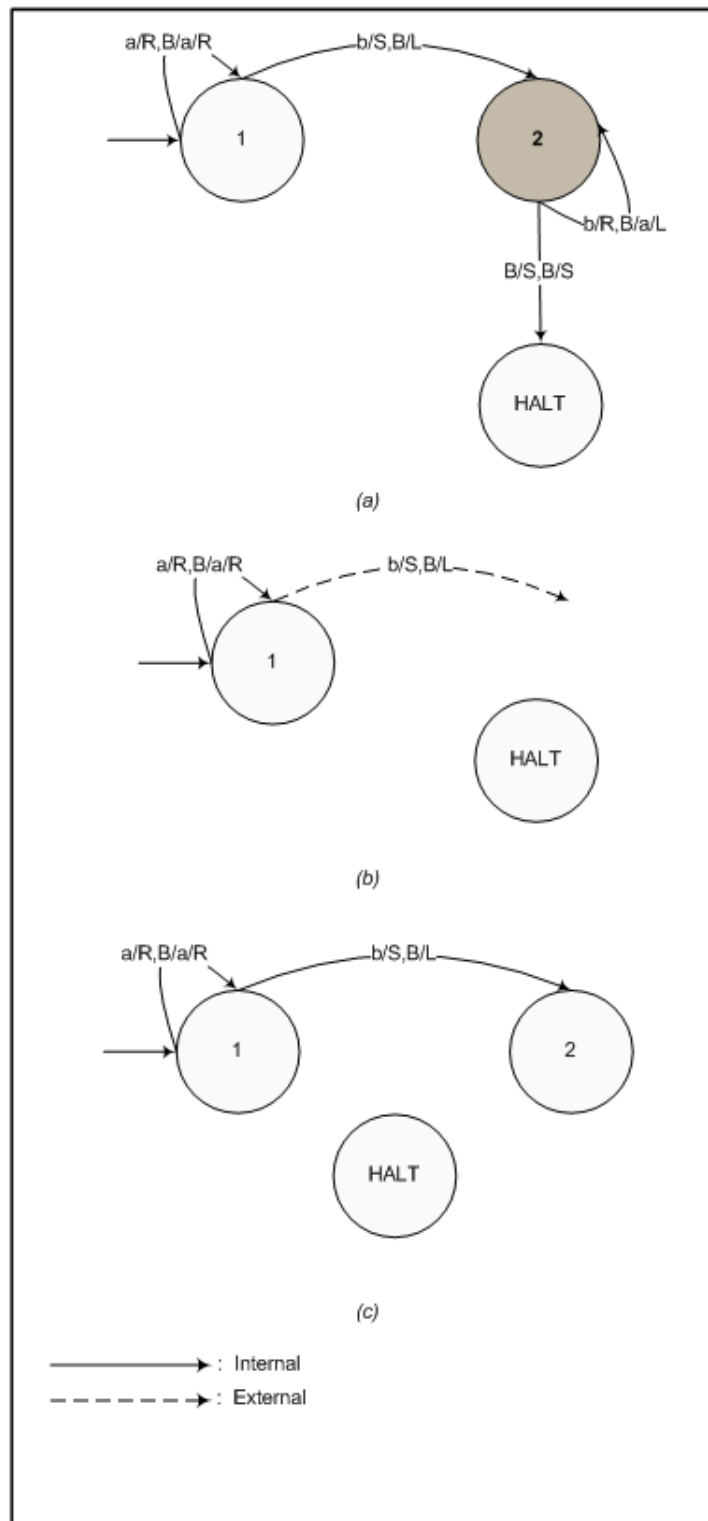


Figure 6.40: Example Mutation

6.5.1.3 Interpretation

The fitness of the graph is dependent on the number of correctly classified sentences. During interpretation the number of movements on the tape is restricted to prevent any non-halting Turing machines. The restriction on the number of movements is calculated to be *length of the input string* \times 100. If the movement goes off the left or right of either tape, a blank 'B' symbol is added to the tape thereby extending the tape as needed. During interpretation a Turing Machine halts if there is no outgoing arc from the current state for the character read from the tape A and the character read from tape B or if a 'HALT' state is reached. A sentence is accepted if a machine ends in the 'HALT' state. If there is no outgoing arc from the current state of the machine, the machine rejects the sentence.

Table 6.10 shows the interpretation of a sentence ('aabb') for the Turing machine acceptor presented in Figure 6.41. Initially the sentence 'aabb' is placed on Tape A with the head pointing to the initial 'a' in the sentence. Blank symbols are placed to the right and left of the sentence. Tape B, on the other hand, is blank. Since the input characters are 'a' and 'B' from Tape A and Tape B respectively, the Turing machine follows the transition with the corresponding input characters. The transition followed is shown in Figure 6.41. The head moves right on Tape A. The character 'a' is written to Tape B and the head moves right. Table 6.10 shows each transition made, and the contents of the tape after each transition. The arrows indicate the characters read from the tapes. The machine ends in the 'HALT' state which implies that the sentence is accepted.

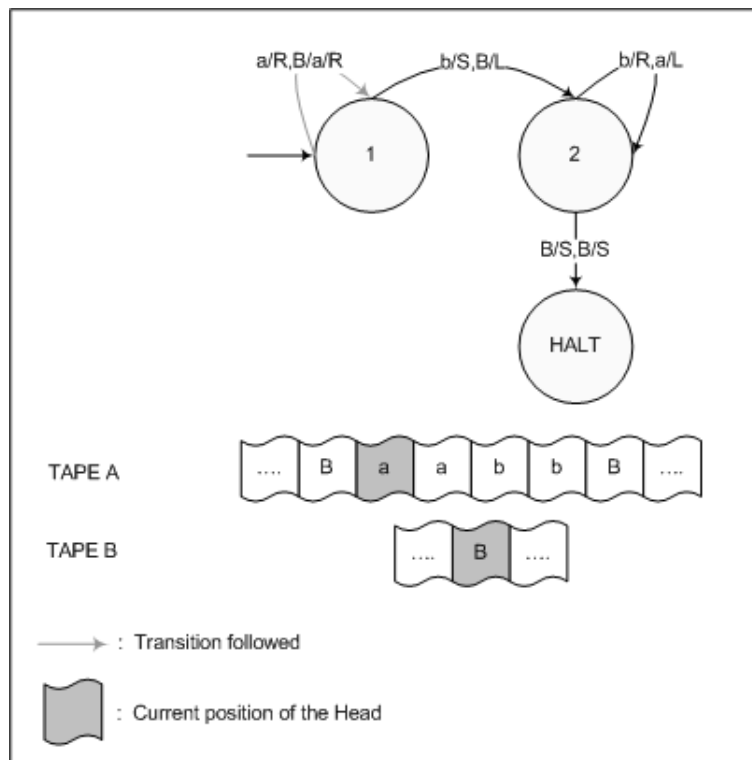


Figure 6.41: Example Interpretation (1)

| State | Tape A | Tape B | Transition State |
|-------|----------------------------|----------------------|------------------|
| 1 | ...Ba a bbB... ↑ | ... B ... | 1 |
| 1 | ...Ba a bbB... ↑ | ...Ba B ... | 1 |
| 1 | ...Ba a bbB... ↑ | ...Baa B ... | 2 |
| 2 | ...Ba a bbB... ↑ | ...Baa a B... | 2 |
| 2 | ...Ba a bbB... ↑ | ...Baa B ... | 2 |
| 2 | ...Baabb B ... ↑ | ... B aaB... | 0 |

Table 6.10: Interpretation of ‘aabb’

Table 6.11 shows the interpretation of the sentence ‘abab’ on the Turing machine in Figure 6.41. The sentence is rejected by the Turing machine as there is no transition from state 2 for the input characters ‘a’ from Tape A and ‘B’ from Tape B.

| State | Tape A | Tape B | Transition State |
|-------|-----------------------------|--------------------|------------------|
| 1 | ...Ba a babB... ↑ | ... B ... | 1 |
| 1 | ...Ba a babB... ↑ | ...Ba B ... | 2 |
| 2 | ...Ba a babB... ↑ | ...Ba B ... | 2 |
| 2 | ...Bab a bB... ↑ | ... B aB... | - |

Table 6.11: Interpretation of ‘abab’

6.5.1.4 GP Parameters for TM Acceptors

The GP Parameters used by the system are outlined in Table 6.12.

| | |
|-------------------------|---|
| Objective | Evolve a Turing Machine Acceptor |
| Population Size | 2000 |
| Selection method | Tournament selection |
| Tournament size | 5 |
| Initial number of Nodes | 8 |
| Max number of Nodes | 15 |
| GP operator rates | Crossover=50%, Reproduction=10%, Mutation=40% |
| Maximum generations | 50 for L1 and L3–L9, 100 for L2 |
| Tape Alphabet | {a,b,B} for L1 and L3–L9, {a,b,c,B} for L2 |
| Raw fitness | The number of correctly classified sentences |
| Termination Criteria | A solution has been found or the maximum number of generations are completed. |

Table 6.12: GP Parameters used for the induction of TMAs

6.5.1.5 Language set

The fitness cases used in attempting to induce a Turing machine acceptor are a set of sample sentences. Each set consists of both positive and negative sample sentences where the positive sentences are accepted by the Turing machine acceptor and the negative sentences are rejected. The language set used to test the system is shown in Table 5.4 in Chapter 5.

As with FAs and PDAs, if any brittle solutions were found the number of fitness cases for that language was increased. For each language where $\Sigma = \{a,b\}$, the test set contained all strings up to a maximum length of 15. For each language where $\Sigma = \{a,b,c\}$ on the other hand, the test set contained all strings up to a maximum length of 10. The number of fitness cases used to test each language is outlined in Table 6.13.

| Language | Fitness cases |
|----------|---------------|
| L1 | 405 |
| L2 | 186 |
| L3 | 190 |
| L4 | 265 |
| L5 | 85 |
| L6 | 40 |
| L7 | 276 |
| L8 | 92 |
| L9 | 106 |

Table 6.13: Number of fitness cases used for inducing TM Acceptors

6.5.2 Turing Machine Transducers

The section that follows describes the genetic programming system used to induce deterministic Turing machine (TM) transducers.

6.5.2.1 Representation of an Individual

Each individual of the population is represented using a directed graph as opposed to a tree-based structure. The system implements Turing Machines with two tapes, where the first tape is the read-only input tape and the second is the output tape of the transducer. Each individual is therefore represented having two transitions on each edge. The first transition, which is used for the first tape (Tape A) has 2 characters on the edge e.g. a/b and the second tape (Tape B) can have either 2 or 3 characters e.g. a/b or a/c/b where 'a' is the character that is read from the Turing machine tape, 'b' is the symbol that specifies the direction to move the head and 'c' is the symbol written to the tape. The symbol 'b' can be either 'R', 'L' or 'S' where 'R' indicates that the head moves right, 'L' indicates that the head moves left and 'S' indicates that the head is stationary and doesn't move. Each state is numbered sequentially where the state numbered 1 is the start state and state 0 is the 'HALT' state. Figure 6.42 below shows an example representation of an individual.

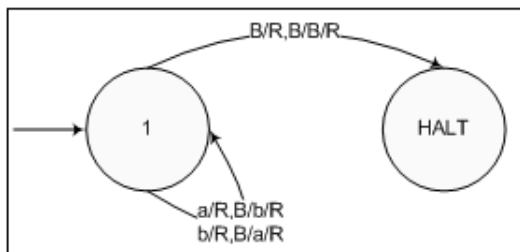


Figure 6.42: Example Turing Machine Transducer

In this system, two methods of interpreting how to read the output is used. The first method (TMT1) reads the output as being the non-blank section of the second tape [41], irrespective of the position of the head. Figure 6.43 shows an example of the interpretation of the output for TMT1.

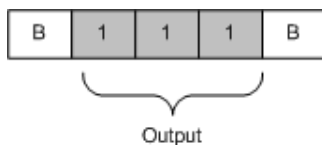


Figure 6.43: Example Output Tape for TMT1

The second method (TMT2) uses the head of the second tape to indicate the start of the output. Figure 6.44 shows an example of the interpretation of the output for TMT2.

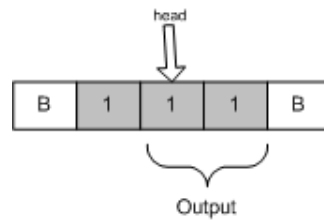


Figure 6.44: Example Output Tape for TMT2

The following syntax is used by the program for the TM Transducer in Figure 6.42:

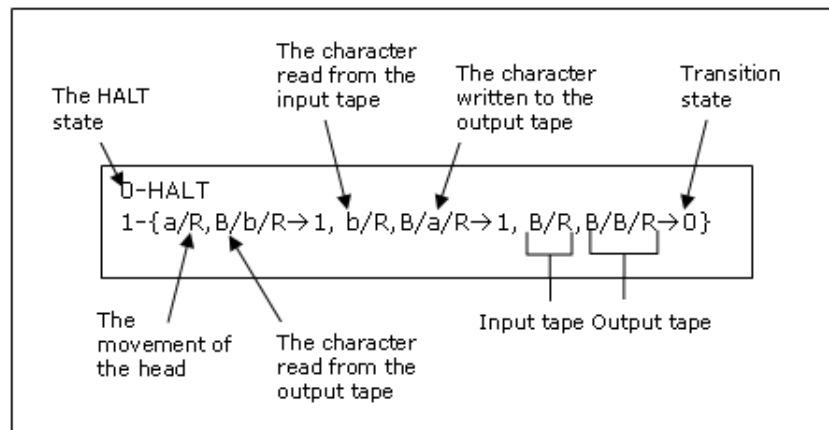


Figure 6.45: Syntax for TM Transducers

6.5.2.2 Genetic Programming Operators

The genetic programming operators used are reproduction, crossover and mutation. The operators are performed in the same way as that of the Turing Machine Acceptors.

6.5.2.3 Interpretation

The fitness of the graph is dependent on the number of correctly classified sentences. A sentence is correctly classified if the input sentence of the Turing machine transducer produces the expected output sentence on the output tape. During interpretation the number of movements on the tape is restricted to prevent any non-halting Turing machines. If the movement goes off the left or right of either tape a blank 'B' symbol is added to the tape thereby extending the tape as needed. During interpretation a Turing Machine halts if there is no outgoing arc from the current state for the character read from the tape A and the character read from tape B or if a 'HALT' state is reached.

The interpretation works similar to the Turing machine acceptor except that the output on Tape B is used to determine if a sentence is correctly classified. Therefore using the same Turing machine in Figure 6.41, if the input sentence is 'aabb', the output string is 'aa' for both TMT1 and TMT2.

6.5.2.4 GP Parameters for TM Transducers

The GP Parameters used by the system are outlined in Table 6.14.

| | |
|-------------------------|--|
| Objective | Evolve a Turing Machine Transducer |
| Population Size | 2000 |
| Selection method | Tournament selection |
| Tournament size | 5 |
| Initial number of Nodes | 8 |
| Max number of Nodes | 15 |
| GP operator rates | Crossover=50%, Reproduction=10%, Mutation=40% |
| Maximum generations | 50 |
| Tape Alphabet | {0,1,B} for L1, L4 and L5, {1,B} for L2 and L3, and {a,b,B} for L6 |
| Raw fitness | The number of correctly classified sentences |
| Termination Criteria | A solution has been found or 50 generations are completed. |

Table 6.14: GP Parameters used for the induction of TMTs

6.5.2.5 Language set

The fitness cases used in attempting to induce a Turing machine transducer are a set of sample sentences. Each set consists of an input sentence and a corresponding output. The language set used to test the system is shown in Table 5.5 in Chapter 5 and the number of fitness cases used to test each language is outlined in Table 6.15.

| Language | Fitness cases – TMT1 | Fitness cases – TMT2 |
|----------|----------------------|----------------------|
| L1 | 10 | 11 |
| L2 | 3 | 5 |
| L3 | 4 | 3 |
| L4 | 22 | 30 |
| L5 | 18 | 11 |
| L6 | 50 | 54 |

Table 6.15: Number of fitness cases used for inducing TM Transducers

6.6 Premature Convergence

Premature convergence is a common problem experienced with evolutionary algorithms. The population converges to a suboptimal solution. To prevent premature convergence, the following mechanisms have been built into the system:

- Non-destructive operators – Premature convergence may occur due to the destructive effects of crossover or mutation [37], i.e. these genetic operators may produce offspring that have a

worse fitness than their parents. Therefore, non-destructive versions of crossover and mutation have been implemented by ensuring that the offspring produced are at least as fit as the parent. This is achieved by iteratively calling up the mutation or crossover operator until the operator produces offspring that has a fitness that is more than or equal to their parents. A limit is set on the number of iterations. If the limit is reached and the fitness of the child has not improved the parent is just copied to the new population.

- Multiple runs – Mahfound [37] identifies the loss of genetic variation in the population as a being cause of premature convergence. This selection variance or selection noise is a result of the random nature of evolutionary algorithms whereby the population of some runs contain the components necessary for the induction of a solution algorithm while others may not. The system therefore performs a number of iterations per seed to deal with selection variance.

Chapter 7

Results and Discussion

7.1 Finite Acceptors

The system was tested using ten random seed values for each language in the language set. The ten random seed values used are 1000, 1010, 2000, 2500, 3000, 3007, 3500, 4000, 4021 and 5529. The seed values were chosen randomly. The program was tested using both the standard genetic operators and non-destructive operators. The non-destructive operators are implemented by ensuring that the fitness of the child is better than or equal to the fitness of the parent when applying crossover and mutation. Non-destructive operators are applied to avoid the destructive effect of crossover and mutation. If no solutions were found for a particular seed value multiple iterations (the system was tested using ten iterations of each seed value for the two character languages and twenty iterations for the three character languages) for this seed were performed to deal with selection variance. All the tables referred to in this section can be found in Appendix A.

7.1.1 Deterministic Finite Acceptors

The results for each language are outlined below:

L1 (a*): Solutions were evolved for all of the random seed values. All solutions were found in the initial population; therefore no genetic operators were used for this language. In effect, solutions for L1 are found using a ‘random search’ of the search space of the problem.

Figure 7.1 presents some example solutions for L1.

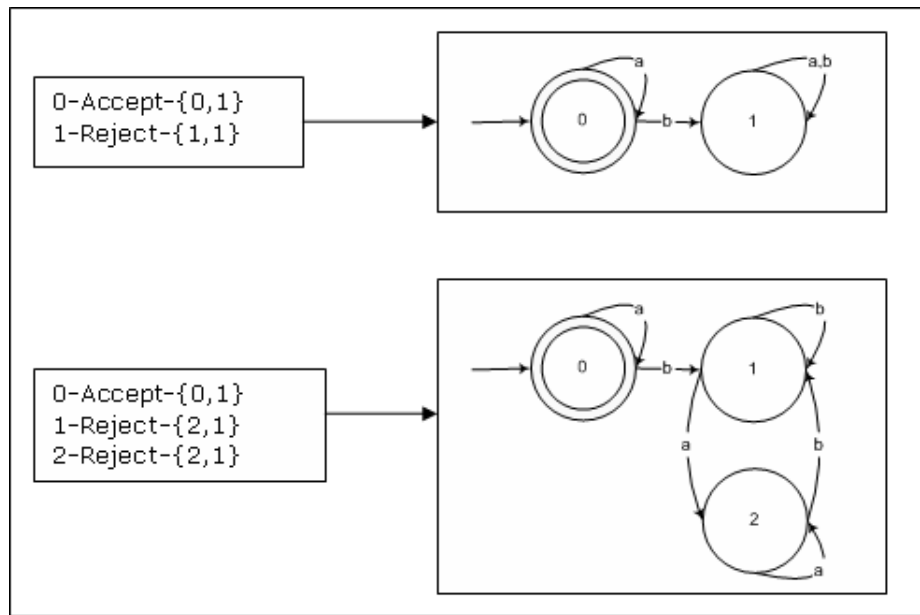


Figure 7.1: Example Solutions for L1 (DFA)

L2 (ba)*: Solutions were generated for all seed values using the standard genetic operators as well as non-destructive operators. Table A.2 and Table A.3 outlines the seed values as well as the generation on which a solution was found. Multiple iterations were not required for L2 as all seed values induced solutions on the initial iteration.

Figure 7.2 presents an example solution for L2.

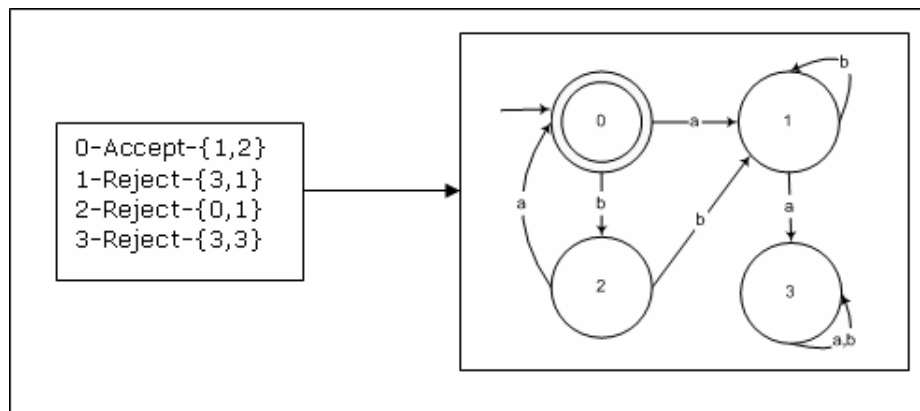


Figure 7.2: Example Solution for L2 (DFA)

L3 (any sentence without an odd number of consecutive a's after an odd number of consecutive b's): Table A.4 outlines the results obtained for L3 using the standard genetic operators

and Table A.5 outlines the results when applying non-destructive operators. Solutions were induced for all seed values when using multiple iterations.

Figure 7.3 presents an example solution for L3.

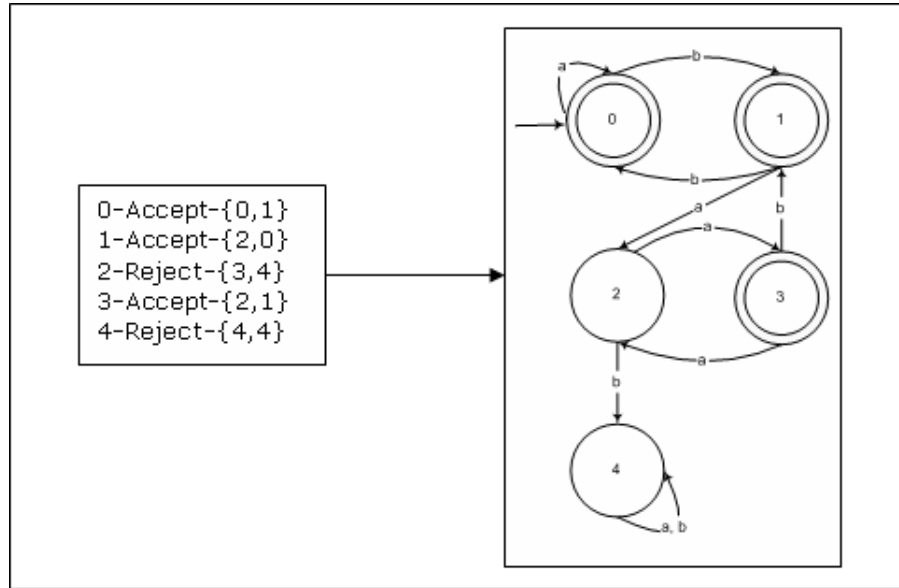


Figure 7.3: Example Solution for L3 (DFA)

L4 (any sentence over the alphabet a, b without more than two consecutive a's): Solutions were obtained for all seed values using non-destructive operators and the standard genetic operators. Table A.6 shows the results obtained for L4 when applying the standard genetic operators and Table A.7 presents the results obtained when using non-destructive operators.

Figure 7.4 presents an example solution for L4.

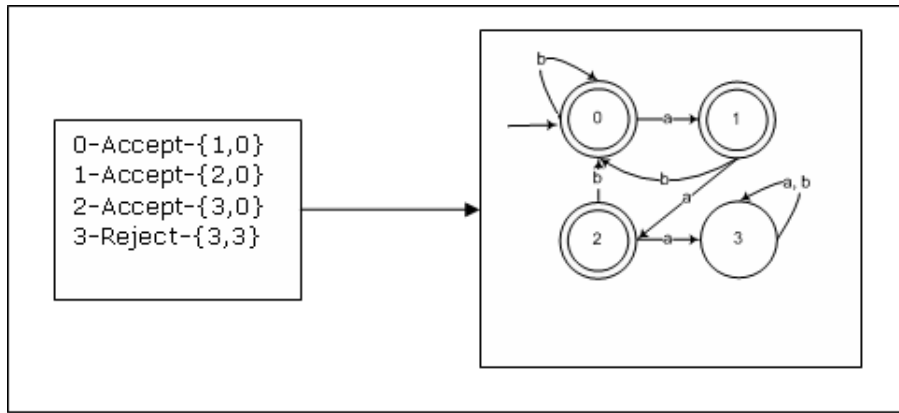


Figure 7.4: Example Solution for L4 (DFA)

L5 (any sentence with an even number of a's and an even number of b's): Solutions were generated for all seed values using either the standard genetic operators or non-destructive operators for each seed value. Table A.8 presents the results obtained for L5 applying the standard genetic operators and Table A.9 presents the results when applying non-destructive operators.

Figure 7.5 presents an example solution for L5.

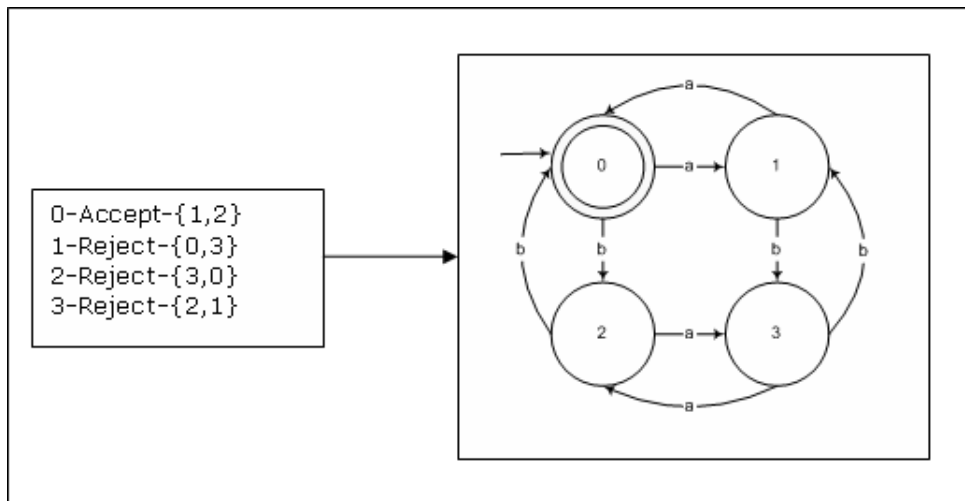


Figure 7.5: Example Solution for L5 (DFA)

L6 (any sentence such that the number of a's differs from the number of b's by 0 modulo 3): Solutions were obtained for all seed values using either the standard genetic operators or non-destructive operators. Table A.10 and Table A.11 shows the results obtained for L6.

Figure 7.6 presents an example solution for L6.

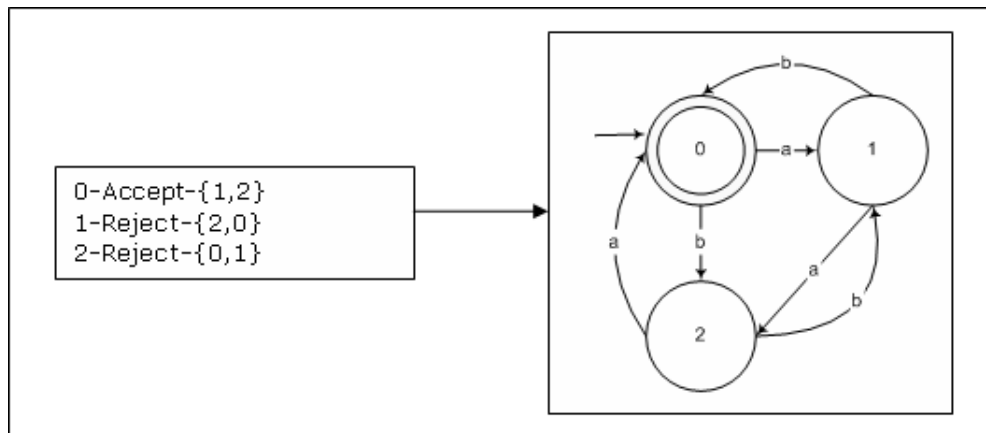


Figure 7.6: Example Solution for L6 (DFA)

L7 ($a^*b^*a^*b^*$): Solutions were found for all seeds using either the standard genetic operators or non-destructive operators. Table A.12 shows the results obtained for L7 when applying the standard genetic operators and Table A.13 shows the results using non-destructive operators.

Figure 7.7 presents an example solution for L7.

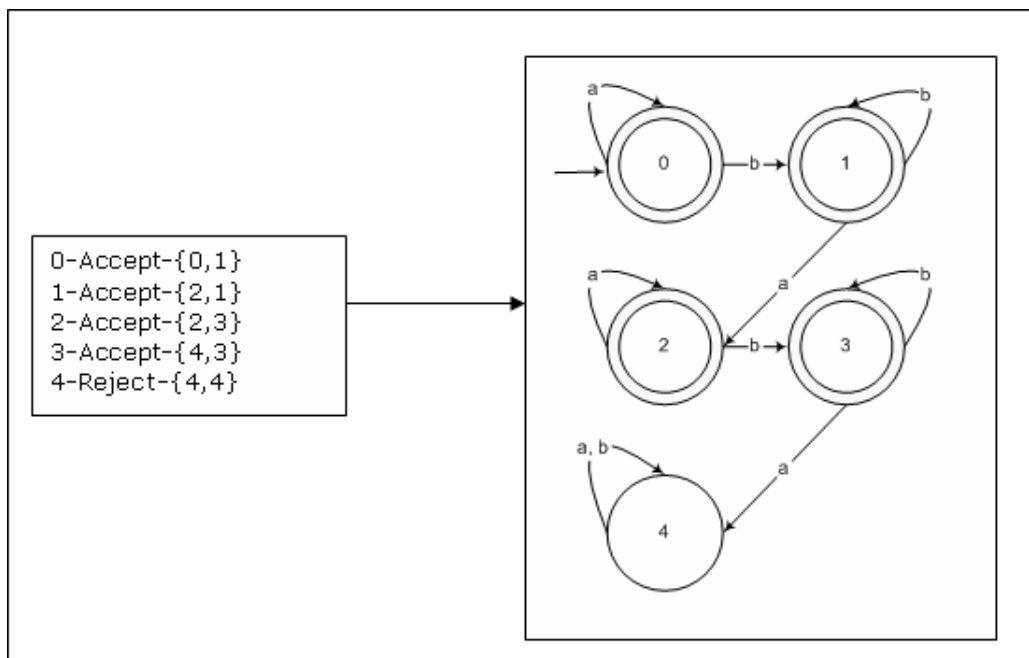


Figure 7.7: Example Solution for L7 (DFA)

L8 (a^*b): Solutions were evolved for all of the random seed values during either initial generation of the population or the second generation of the population. Using non-destructive operators for

this language is not necessary as the results yielded are exactly the same as when using the standard genetic operators.

Figure 7.8 presents an example solution for L8.

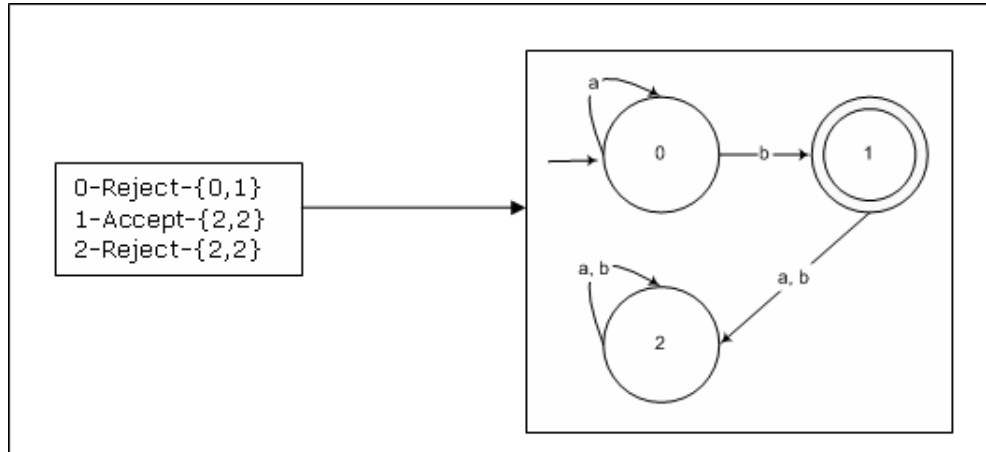


Figure 7.8: Example Solution for L8 (DFA)

L9 $((a^* + c^*)b$): No solutions were evolved for the seed values using the standard genetic operators. Non-destructive operators produced two solutions when using multiple iterations.

Figure 7.9 presents an example solution.

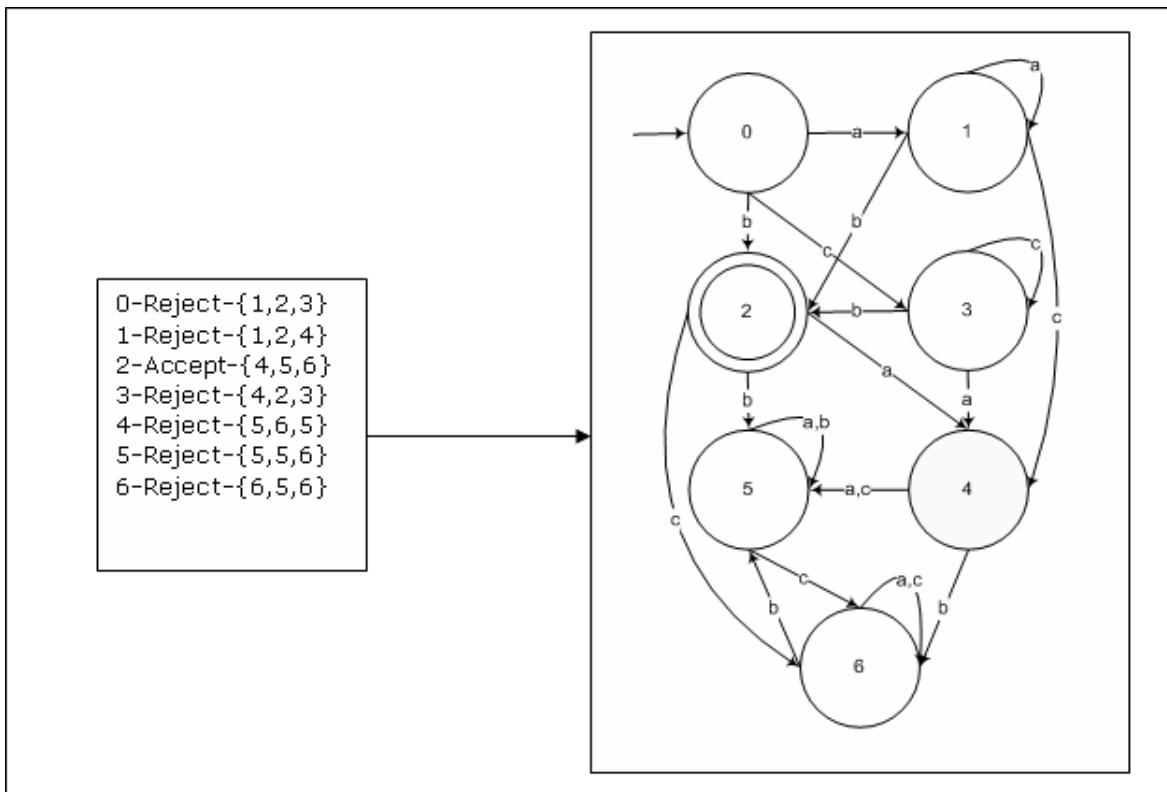


Figure 7.9: Example Solution for L9 (DFA)

The solution generated by the system has redundant states. This can be eliminated by applying the minimization algorithm. Figure 7.10 shows the minimized DFA for L9 of the DFA presented in Figure 7.9.

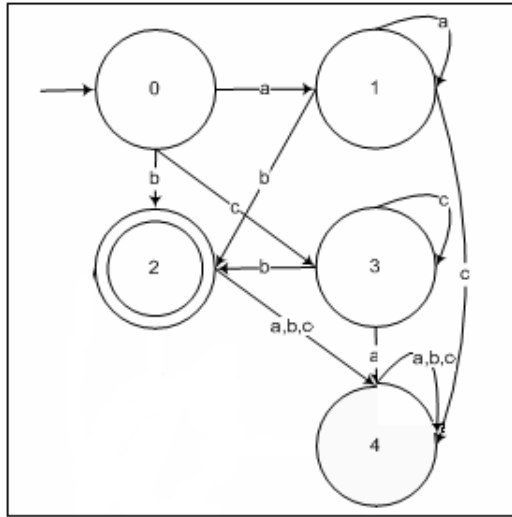


Figure 7.10: Minimized DFA for L9

L10 ((aa)*(bbb)*): Solutions were evolved for five seed values using the standard genetic operators. Solutions were generated for all seed values using non-destructive crossover as well as non-destructive crossover and mutation. Table A.17 shows the results obtained for L10 when applying the standard genetic operators and Table A.18 shows the results using non-destructive operators.

Figure 7.11 presents an example solution for L10 and Figure 7.12 shows the minimized DFA for L10.

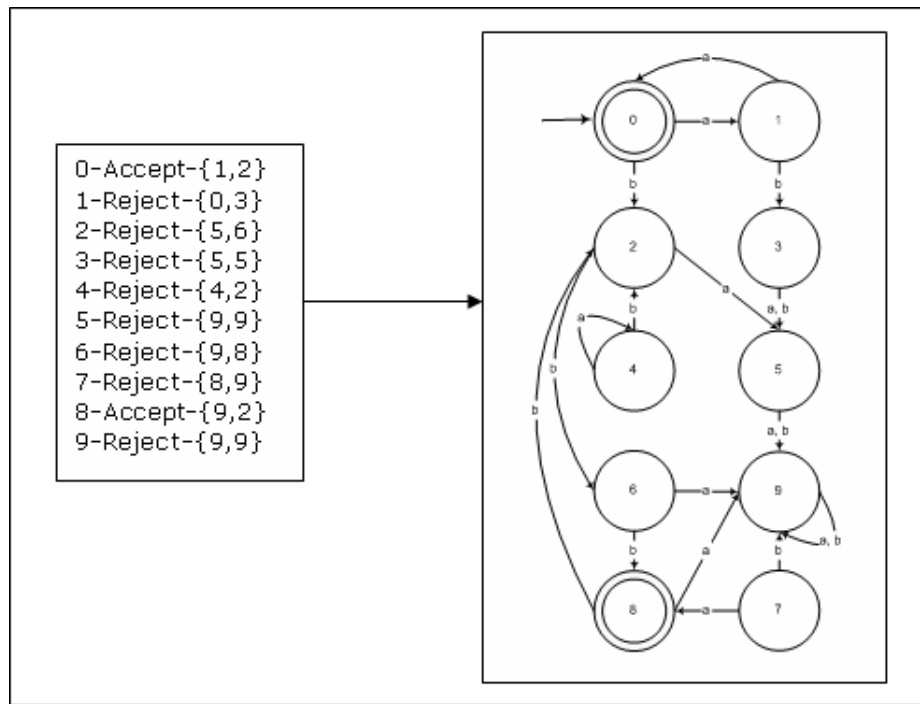


Figure 7.11: Example Solution for L10 (DFA)

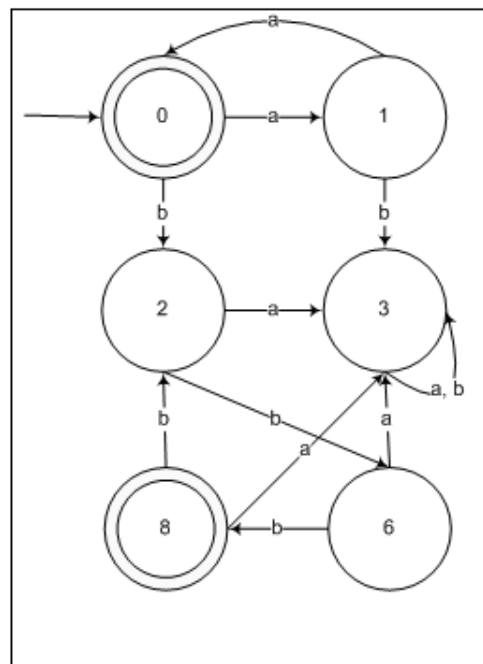


Figure 7.12: Minimized DFA for L10

L11 (any sentence with an even number of a's and an odd number of b's): Solutions were evolved for all seed values using the standard genetic operators and non-destructive operators, the results of which are outlined in Table A.19 and Table A.20.

Figure 7.13 presents an example solution for L11 and Figure 7.14 shows the minimized DFA for L11.

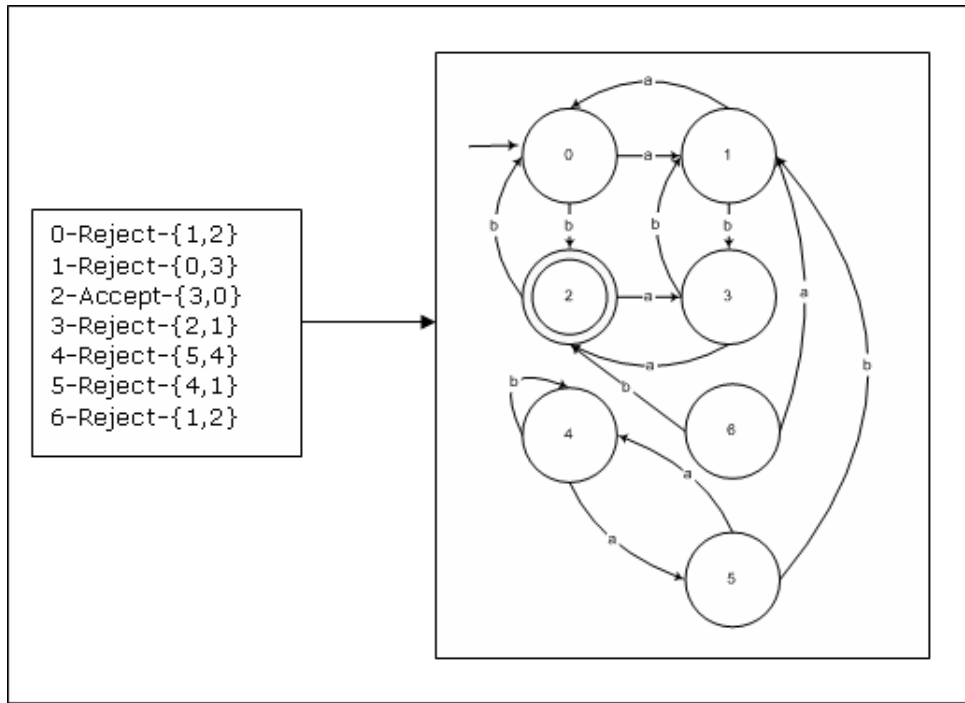


Figure 7.13: Example Solution for L11 (DFA)

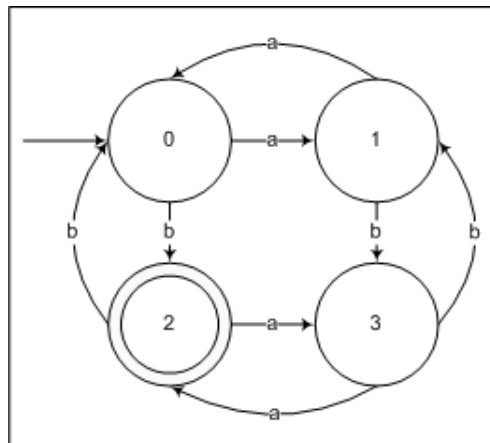


Figure 7.14: Minimized DFA for L11

L12 ($a(aa)^*b$): Solutions were found for all seed values. Table A.21 shows the results for L12 when applying the standard genetic operators and Table A.22 shows the results when applying non-destructive operators.

Figure 7.15 presents an example solution for L12.

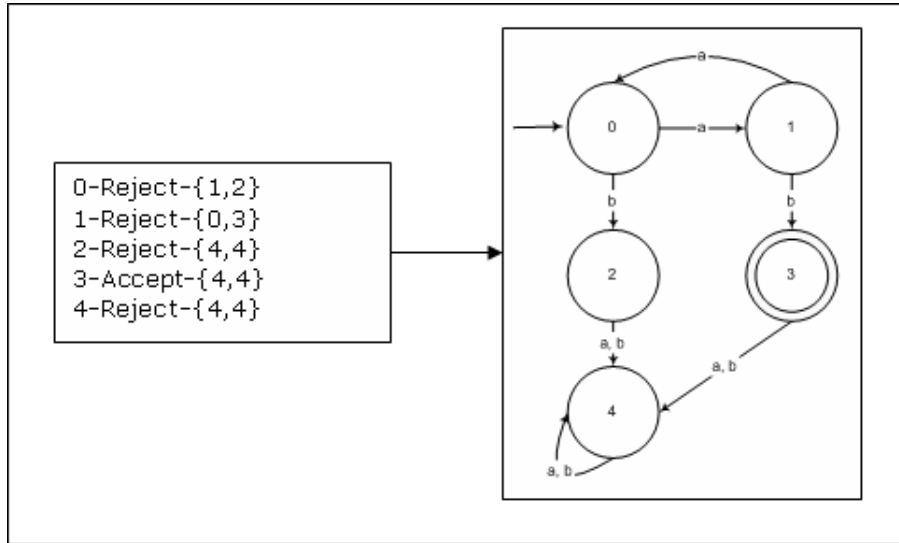


Figure 7.15: Example Solution for L12 (DFA)

L13 (any sentence over the alphabet a, b with an even number of a 's): Solutions were evolved for all of the random seed values. As in L1, all solutions were found in the initial population; therefore no genetic operators were used for this language.

Figure 7.16 presents an example solution for L13.

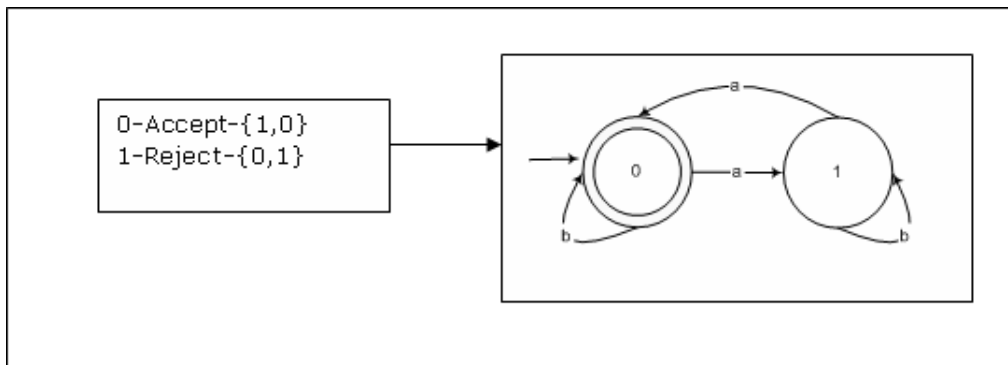


Figure 7.16: Example Solution for L13 (DFA)

L14 ($(aa)^*ba^*$): Solutions were evolved for all seed values using both the standard genetic operators

and non-destructive operators. Table A.24 presents the results obtained for L14 using the standard genetic operators and Table A.25 presents the results using non-destructive operators.

Figure 7.17 presents an example solution for L14.

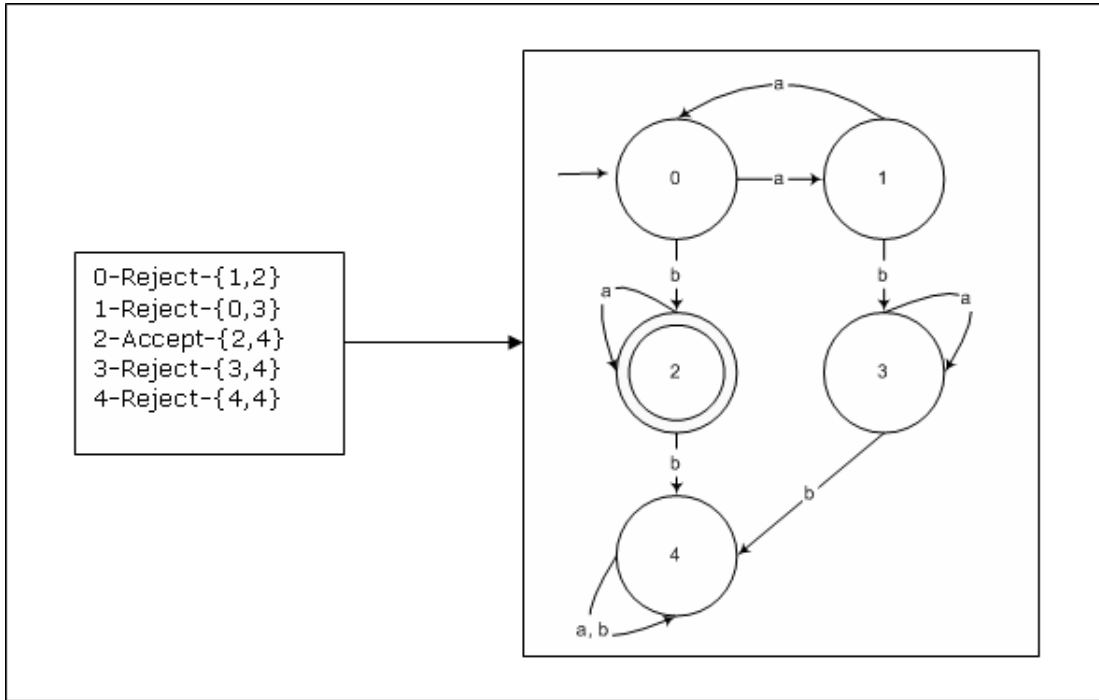


Figure 7.17: Example Solution for L14 (DFA)

L15 ($bc^*b + ac^*a$): Solutions were evolved for three of the ten seeds using the standard genetic operators. Solutions were found for three seed values using non-destructive mutation. Table A.26 and Table A.27 shows the results for L15. Figure 7.18 presents an example solution for L15 and Figure 7.19.

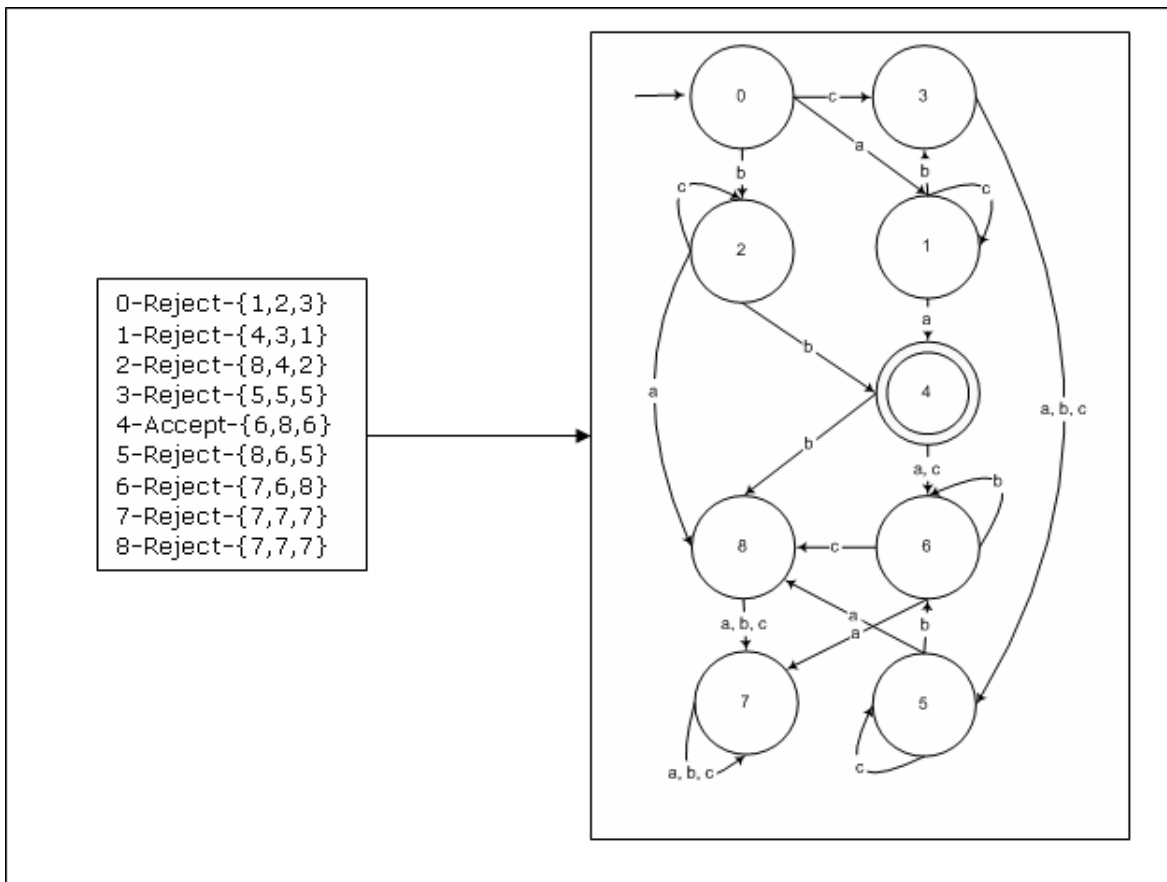


Figure 7.18: Example Solution for L15 (DFA)

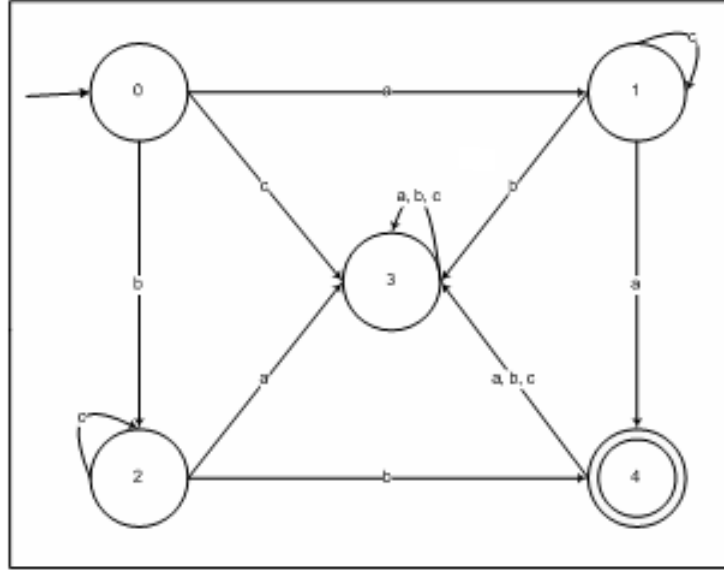


Figure 7.19: Minimized DFA for L15

Table 7.1 lists the success rates for each language. The success rate is calculated as a percentage of the seed values that produced solutions over ten iterations (except for L9 and L15 which show the success rate over twenty iterations).

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 100% | 100% | 100% | 100% |
| L2 | 100% | 100% | 100% | 100% |
| L3 | 100% | 100% | 100% | 100% |
| L4 | 100% | 100% | 100% | 100% |
| L5 | 100% | 100% | 100% | 100% |
| L6 | 100% | 100% | 100% | 100% |
| L7 | 100% | 100% | 100% | 100% |
| L8 | 100% | 100% | 100% | 100% |
| L9 | 0% | 0% | 10% | 10% |
| L10 | 50% | 80% | 100% | 100% |
| L11 | 100% | 100% | 100% | 100% |
| L12 | 100% | 100% | 100% | 100% |
| L13 | 100% | 100% | 100% | 100% |
| L14 | 100% | 100% | 100% | 100% |
| L15 | 30% | 30% | 10% | 10% |

Table 7.1: Success Rates for DFA Simulations

Table 7.2 shows the percentage of seed values that required multiple iterations. Multiple iterations were required for a majority of the languages to overcome premature convergence caused by selection

variance.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 0% | 0% | 0% | 0% |
| L2 | 0% | 0% | 0% | 0% |
| L3 | 80% | 30% | 30% | 30% |
| L4 | 20% | 0% | 0% | 0% |
| L5 | 60% | 40% | 10% | 20% |
| L6 | 0% | 0% | 0% | 0% |
| L7 | 10% | 0% | 10% | 10% |
| L8 | 0% | 0% | 0% | 0% |
| L9 | 100% | 100% | 100% | 100% |
| L10 | 80% | 60% | 60% | 60% |
| L11 | 30% | 40% | 20% | 10% |
| L12 | 80% | 80% | 30% | 30% |
| L13 | 0% | 0% | 0% | 0% |
| L14 | 50% | 30% | 20% | 20% |
| L15 | 80% | 90% | 100% | 100% |

Table 7.2: Multiple iterations required for DFA Simulations

The two character languages generate solutions on either the first iteration or when applying multiple iterations for a random seed value. There is an improvement in the results obtained for the languages when non-destructive operators are used for the two character languages.

Altering the GP parameters (i.e. increasing the population size, increasing the number of generations, etc.) does not improve the results obtained for both the two character and three character languages. Increasing the number of iterations for each seed value to twenty for L9 and L15 slightly improves the results.

The results obtained for L9 and L15 are poor. No solutions were induced for L9 using the standard genetic operators. L15, on the other hand, performed slightly better with a 30% success rate. Figure 7.20 and Figure 7.21 shows the progression of fitness values of the best individual for L9 and L15 respectively when applying non-destructive operators. The graphs show the change in fitness value over the 50 generations for all seed values on the initial iteration. The graphs clearly show the premature convergence of the system as there is no change in the fitness value for the best individual after the 30th generation for L9 and after the 10th generation for L15.

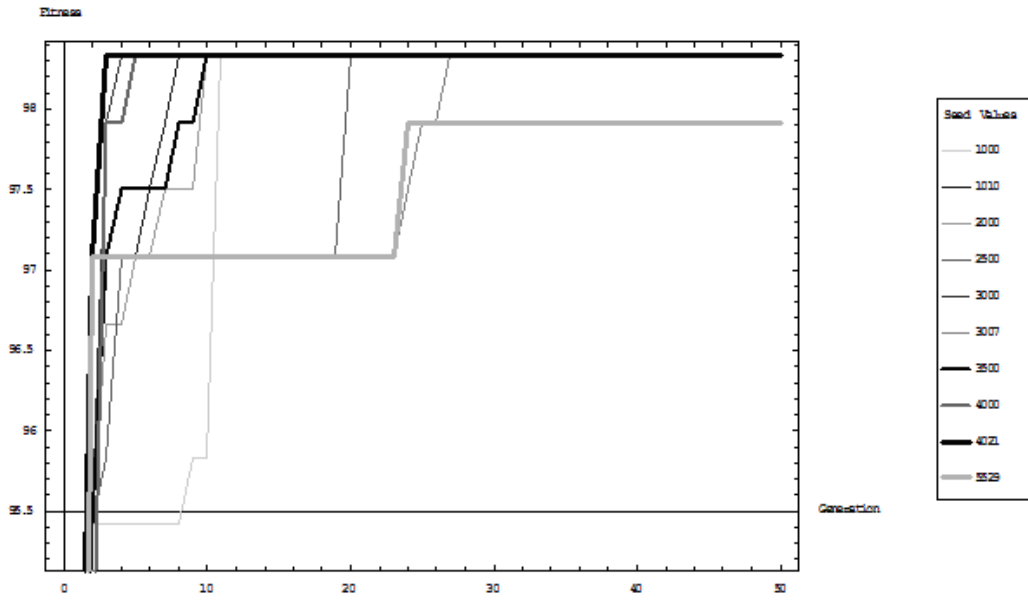


Figure 7.20: Graph for L9 showing the convergence for all seed values

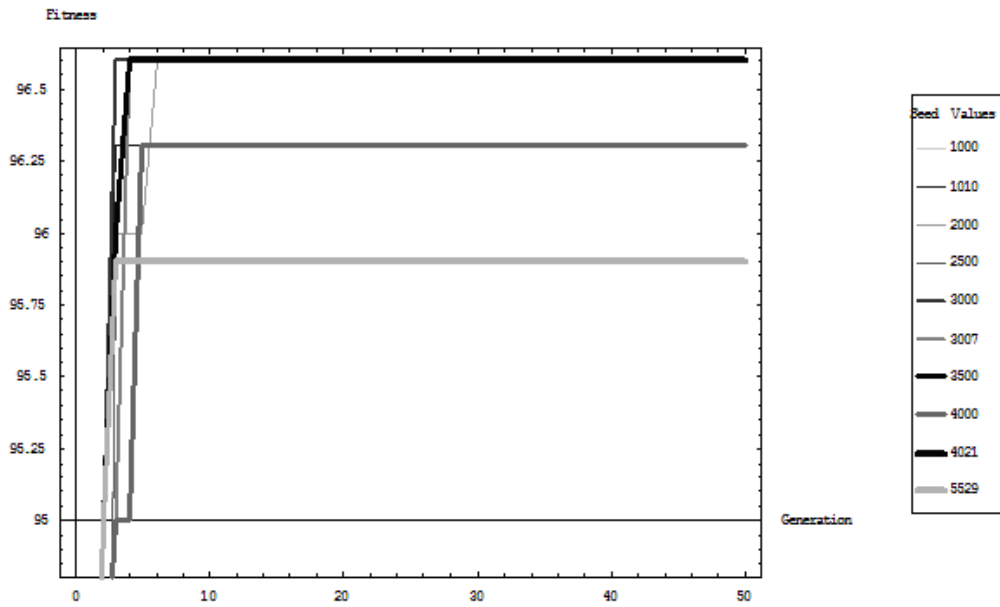


Figure 7.21: Graph for L15 showing the convergence for all seed values

L9 and L15 are both three character languages and therefore has a larger search space. This is probably the reason for the low success rates.

7.1.2 Modular Approach to Deterministic Finite Acceptors

Angeline et al. [1], proposes using a modular approach for problems that have a large search space. Applying a modular approach to the three character languages i.e. L9 $((a^* + c^*)b)$ and L15 $(bc^*b + ac^*a)$ provided more positive results for these two languages. The alphabet for each language was broken up into 2 character components i.e., the language 'abc' was broken up into 'ab', 'ac' and 'bc'. Separate runs were performed for each component and then for the overall language. When attempting to find a solution for each component only the fitness cases that contain the relevant alphabet are used. Table 7.3 shows some example fitness cases for L9. When the run for 'ab' is performed only the fitness cases numbered 1, 4 and 8 are used. A solution is found for the component 'ab' if these three fitness cases are correctly classified. When the run for 'ac' is performed only the fitness case numbered 5 is used and when the run for 'bc' is performed only the fitness cases numbered 2, 6 and 7 are used. The fitness case numbered 3 is not used for any of the runs performed for the components, however this fitness case is used when performing the run for the overall language. All the fitness cases listed below are used when performing the run for the overall language.

| Number | Input | Output |
|--------|--------|--------|
| 1 | aaab | 1 |
| 2 | ccccb | 1 |
| 3 | aacb | 0 |
| 4 | aaaba | 0 |
| 5 | ccaa | 0 |
| 6 | bbcbcb | 0 |
| 7 | cbc | 0 |
| 8 | babb | 0 |

Table 7.3: Example Fitness Cases

Each component is equivalent to a specific language e.g., for L9 the system will attempt to find solutions for a^*b , c^*b , and $(a^* + c^*)$ which are the component languages of $(a^* + c^*)b$.

To use these components the following methods were employed:

- a. Any solutions generated for these components were placed in the initial population for the overall language.
- b. Encapsulate the components i.e. represent the component solutions as a single node with the possibility of these nodes being added during creation of the initial population or during mutation.
- c. Combining both (a) and (b).

Figure 7.22 shows an example of solutions found for the modular components of L9. For method (a), these three solutions will be placed in the initial population when performing the run for the overall language, L9. For method (b), these three solutions are used as a single node which can be added during creation and mutation. During creation and mutation, the system randomly chooses between adding a new node or pointing a transition to an existing node (see Section 6.2.1.1 as shown in Chapter 6). If a new node is chosen to be added, the system has the option of adding an accept, reject or one of the encapsulated nodes. The encapsulated node is only added if the number of nodes in the individual doesn't exceed the maximum nodes genetic programming parameter.

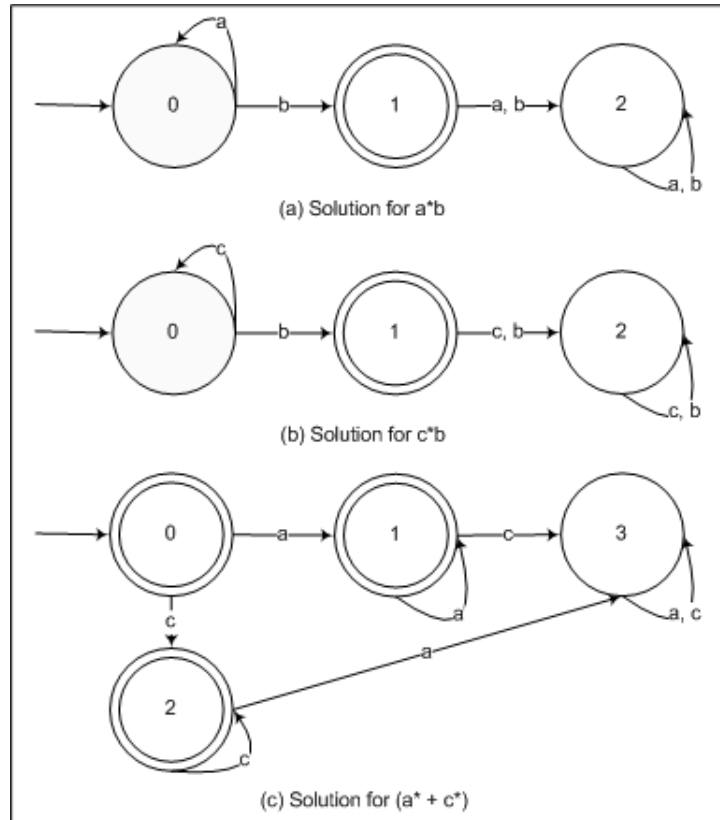


Figure 7.22: Example of Modular Components

The GP parameters are the same as that used by the systems previously (see Table 6.1). If no solutions are found for all of the components, the run for the overall language is not performed. There is no reason to continue the run as this would lead to the same results as the original GP system.

The GP parameters are the same as that used by the systems previously (see Table 6.1). If no solutions are found for all of the components, the run for the overall language is not performed. There is no reason to continue the run as this would lead to the same results as the original GP system.

The results for (a) are outlined in Table A.82 and Table A.83 for L9 and Table A.84 and Table A.85

for L15. For L9 $((a^* + c^*)b)$, solutions were found using standard operators for one of the ten seed values. There were also solutions found for seven of the ten seed values when applying non-destructive crossover. Using non-destructive crossover and mutation led to solutions being generated for four seed values. This undoubtedly produces more encouraging results when compared to the system without modularization where only one solution was found. L15 $(bc^*b + ac^*a)$ also shows an improvement in the results with one solution being evolved when applying the standard genetic operators. When applying non-destructive crossover and mutation, the system yields solutions for nine seed values. Table 7.4 shows the success rates for L9 and L15 when using method (a).

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L9 | 10% | 0% | 70% | 40% |
| L15 | 10% | 80% | 60% | 90% |

Table 7.4: Success Rates for DFA Simulations – Modular Approach (a)

The results for (b) are outlined in Table A.86 and Table A.87 for L9 and Table A.88 and Table A.89 for L15. No solutions were found using standard operators for L9. However, solutions were generated for all seed values when applying non-destructive mutation and crossover. For L15, solutions were evolved for 8 of the 10 seed values when applying the standard genetic operators. Applying non-destructive crossover and non-destructive crossover and mutation led to solutions being generated for all seed values. Table 7.5 shows the success rates for L9 and L15 when using method (b).

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L9 | 0% | 70% | 90% | 100% |
| L15 | 80% | 90% | 100% | 100% |

Table 7.5: Success Rates for DFA Simulations – Modular Approach (b)

The results for (c) are outlined in Table A.90 and Table A.91 for L9 and Table A.92 and Table A.93 for L15. Solutions were found using standard operators for L9 for 4 seed values. Solutions were generated for all seed values when applying non-destructive operators for L9. For L15, solutions were evolved for 6 of the 10 seed values when applying the standard genetic operators. Applying non-destructive mutation and non-destructive crossover led to solutions being generated for all seed values. Table 7.6 shows the success rates for L9 and L15 when using method (c).

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L9 | 40% | 90% | 100% | 100% |
| L15 | 60% | 100% | 100% | 90% |

Table 7.6: Success Rates for DFA Simulations – Modular Approach (c)

The results illustrate that the proposed system is capable of successfully evolving deterministic finite

acceptors when applying a modular approach with the use of non-destructive operators to the three character languages.

7.1.3 Nondeterministic Finite Acceptors¹ (1)

The results for each language are outlined below.

L1 (a*): Solutions were evolved for all of the random seed values. All solutions were found in the initial population; therefore no genetic operators were used for this language.

Figure 7.23 presents some example solutions for L1.

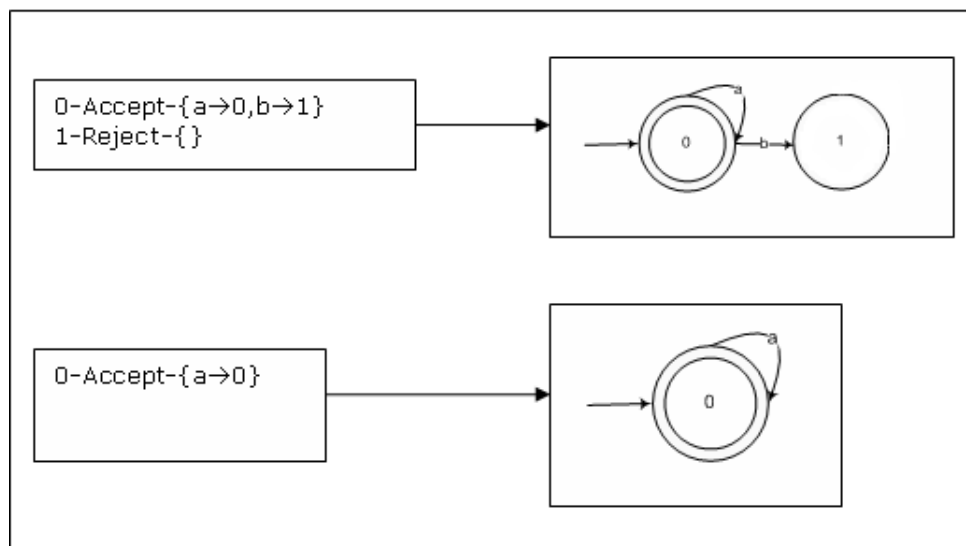


Figure 7.23: Example Solutions for L1 (NFA1)

L2 (ba)*: As with L1, solutions were generated for all seed values on the initial generation of the population for L2.

Figure 7.24 presents some example solutions for L2.

¹See Section 6.2.2 and Section 6.2.3 in Chapter 6 for an explanation of the difference between NFA1 and NFA2.

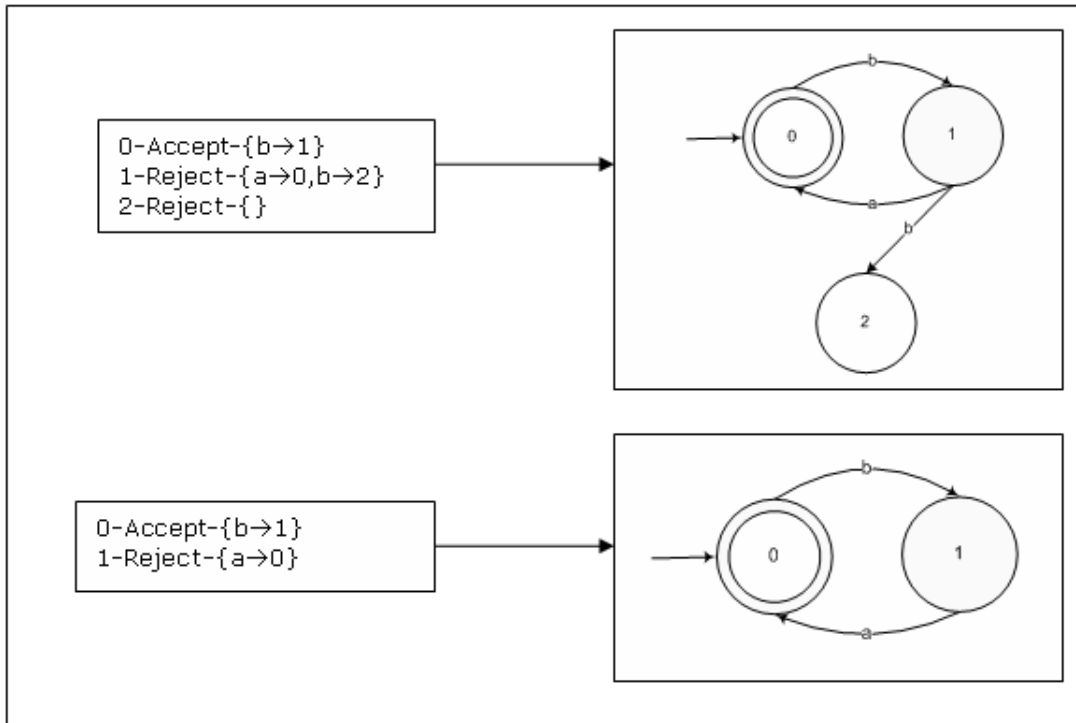


Figure 7.24: Example Solutions for L2 (NFA1)

L3 (any sentence without an odd number of consecutive a's after an odd number of consecutive b's): Solutions were also induced for all seed values using the standard genetic operators and non-destructive operators. Table A.30 and Table A.31 outlines the results for L3.

Figure 7.25 presents an example solution for L3.

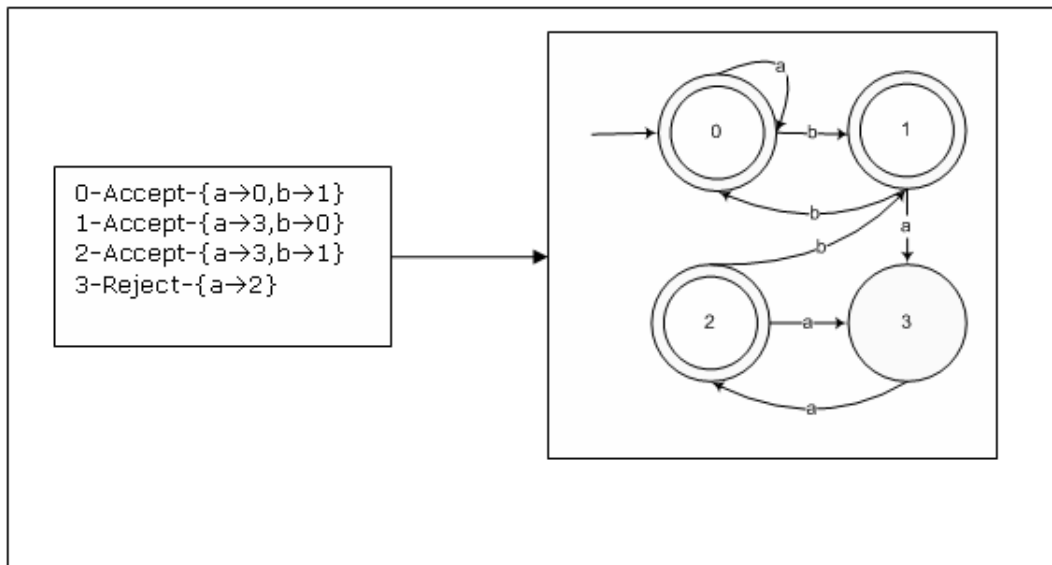


Figure 7.25: Example Solutions for L3 (NFA1)

L4 (any sentence over the alphabet a, b without more than two consecutive a's): Solutions were obtained for all seed values using either the standard genetic operators or non-destructive operators. Table A.32 and Table A.33 shows the results obtained for L4.

Figure 7.26 presents an example solution for L4.

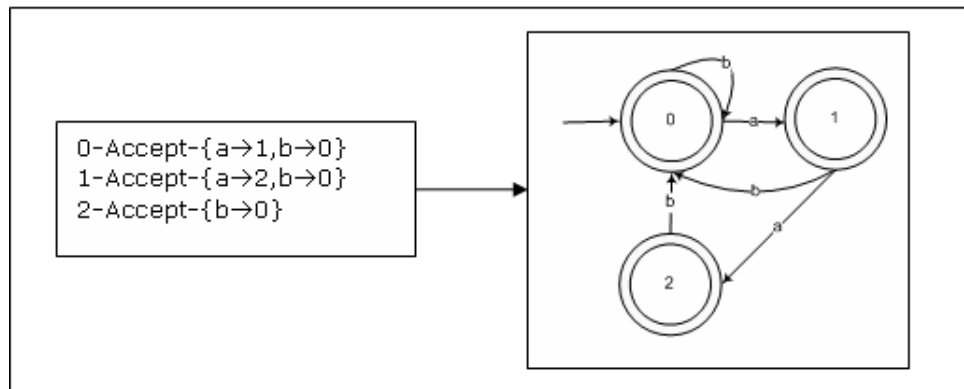


Figure 7.26: Example Solutions for L4 (NFA1)

L5 (any sentence with an even number of a's and an even number of b's): Solutions were obtained for three seed values using the standard genetic operators and for all seed values using non-destructive crossover and non-destructive crossover and mutation. Table A.34 presents the results for L5 when applying the standard genetic operators and Table A.35 presents the results when applying

non-destructive operators. Figure 7.27 presents an example solution for L5.

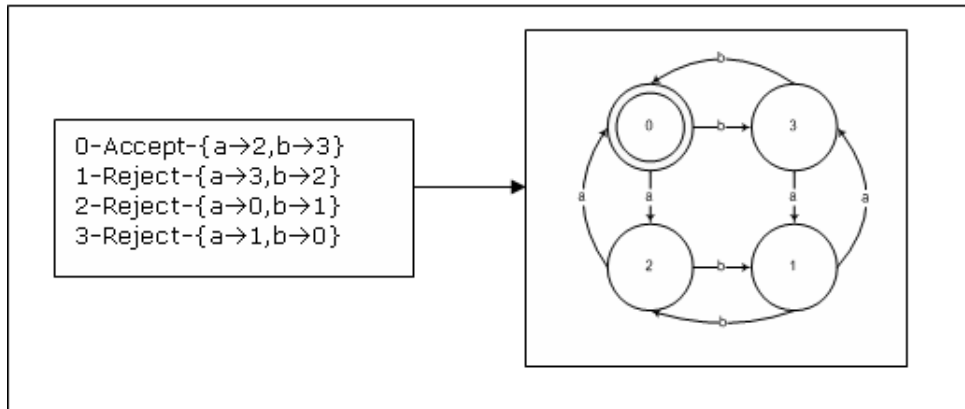


Figure 7.27: Example Solution for L5 (NFA1)

L6 (any sentence such that the number of a's differs from the number of b's by 0 modulo 3): Nine seed values produced solutions for the standard genetic operators and all seed values produced solutions for non-destructive operators. The results are outlined in Table A.36 and Table A.37. Figure 7.28 presents an example solution for L6.

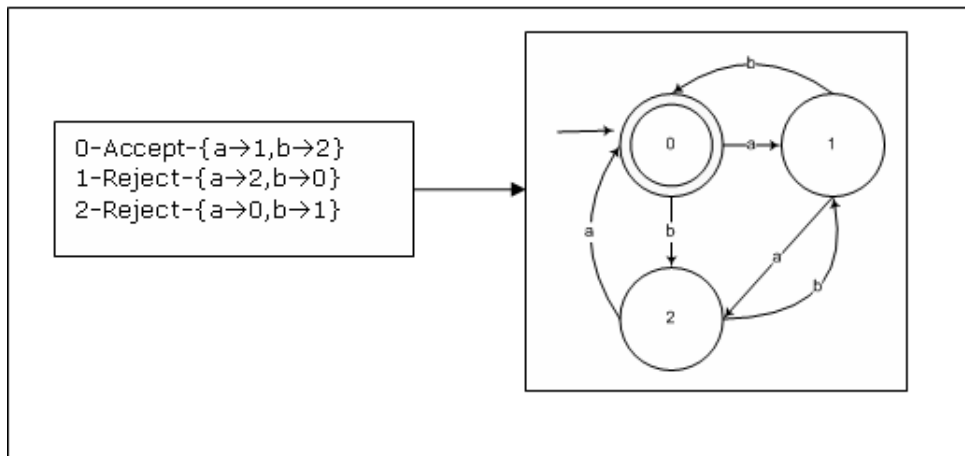


Figure 7.28: Example Solution for L6 (NFA1)

L7 ($a^*b^*a^*b^*$): Solutions were generated for all seed values using either the standard genetic operators or non-destructive operators. The results for L7 are presented in Table A.38 and Table A.39. Figure 7.29 presents an example solution for L7.

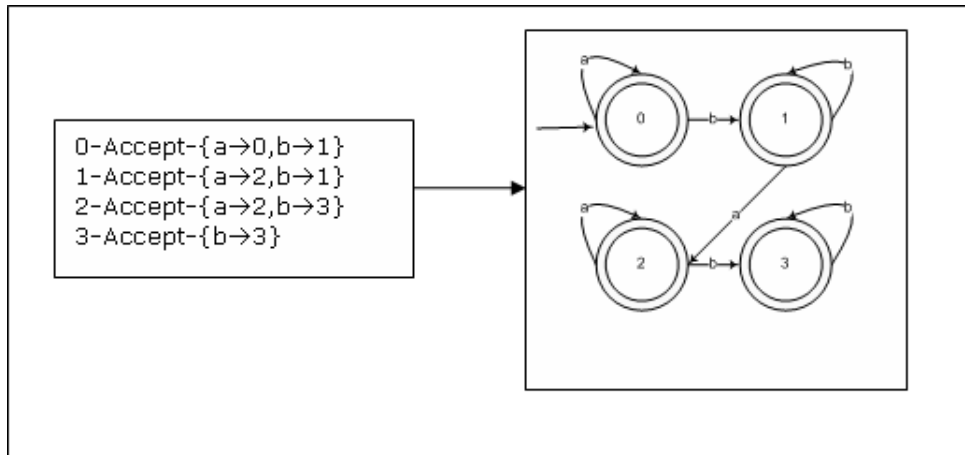


Figure 7.29: Example Solution for L7 (NFA1)

L8 (a^*b): Solutions were generated for all seed values during initial generation of the population therefore L8 did not require any genetic programming operators. Figure 7.30 presents an example solution for L8.

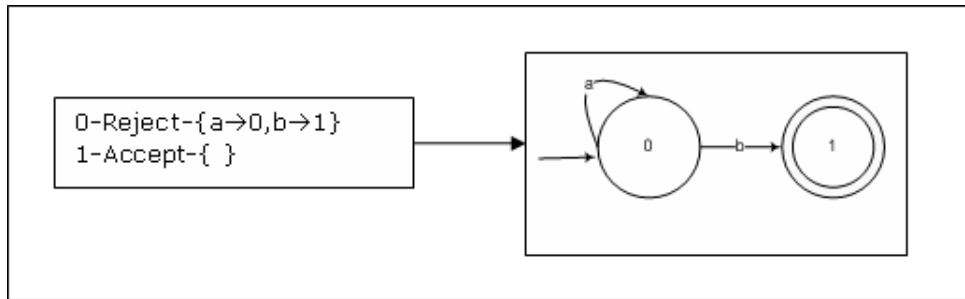


Figure 7.30: Example Solution for L8 (NFA1)

L9 ($(a^* + c^*)b$): Three solutions were generated for L9 using the standard genetic operators. Solutions were generated for all of the ten seed values using non-destructive crossover and mutation. Table A.41 and Table A.42 present the results for L9. Figure 7.31 presents an example solution for L9.

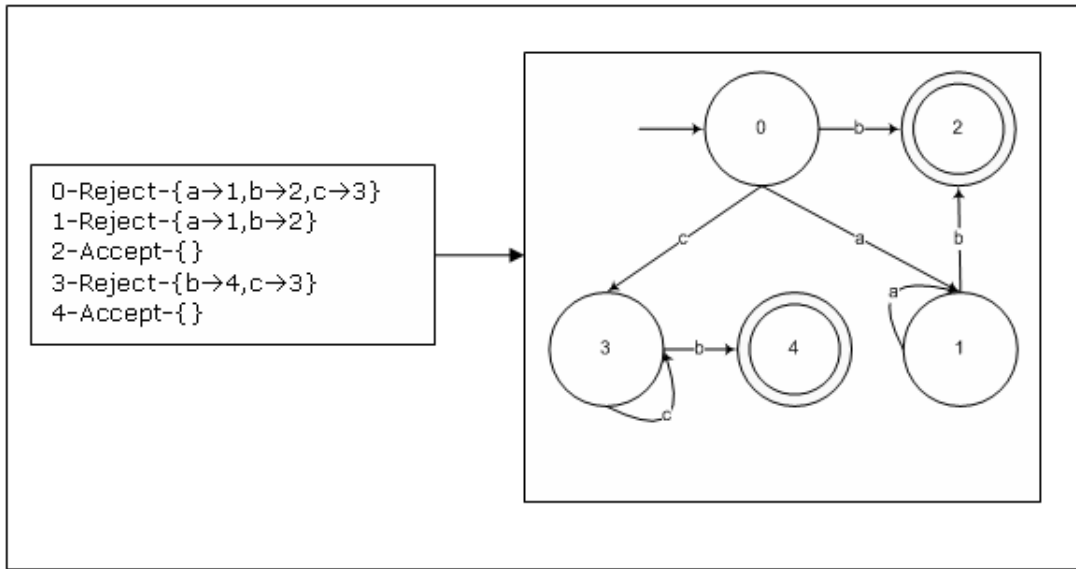


Figure 7.31: Example Solution for L9 (NFA1)

L10 ((aa)*(bbb)*): Solutions were induced for nine of the ten seed values using the standard genetic operators and for all seed values using non-destructive operators. The results for L10 are outlined in Table A.43 and Table A.44. Figure 7.32 presents an example solution for L10.

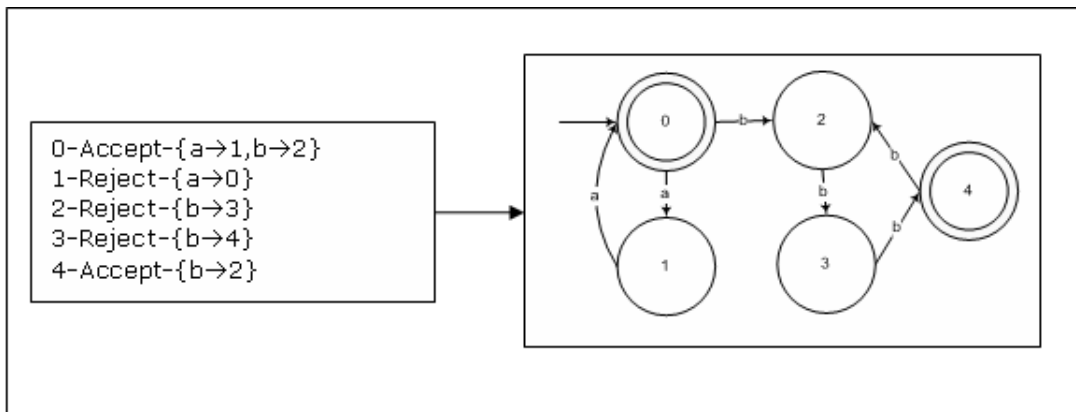


Figure 7.32: Example Solution for L10 (NFA1)

L11 (any sentence with an even number of a's and an odd number of b's): Solutions were found for two seed values for L11 using the standard genetic operators. Solutions were generated for all seed values using non-destructive mutation and crossover. The results are outlined in Table A.45 and Table A.46. Figure 7.33 presents an example solution for L11.

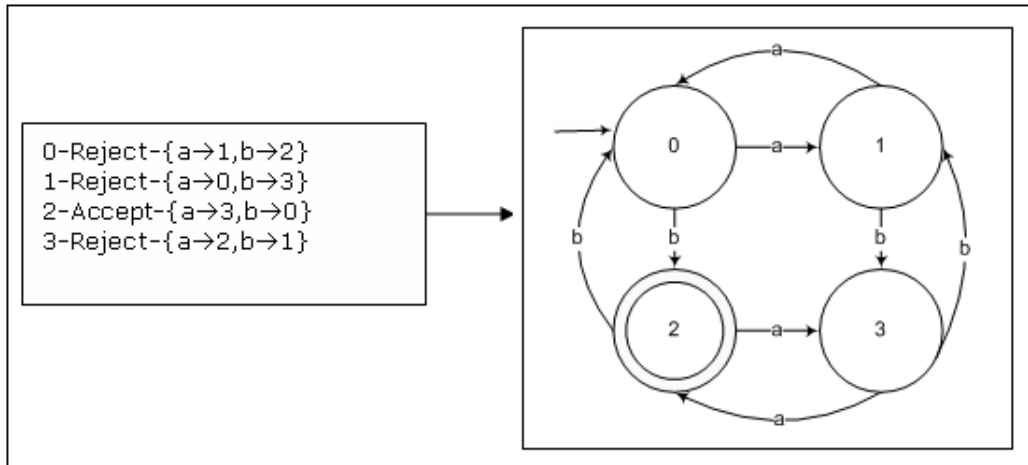


Figure 7.33: Example Solution for L11 (NFA1)

L12 ($a(aa)^*b$): Solutions were generated for all seed values using both the standard genetic operators and non-destructive operators. The results are outlined in Table A.47 and Table A.48. Figure 7.34 presents an example solution for L12.

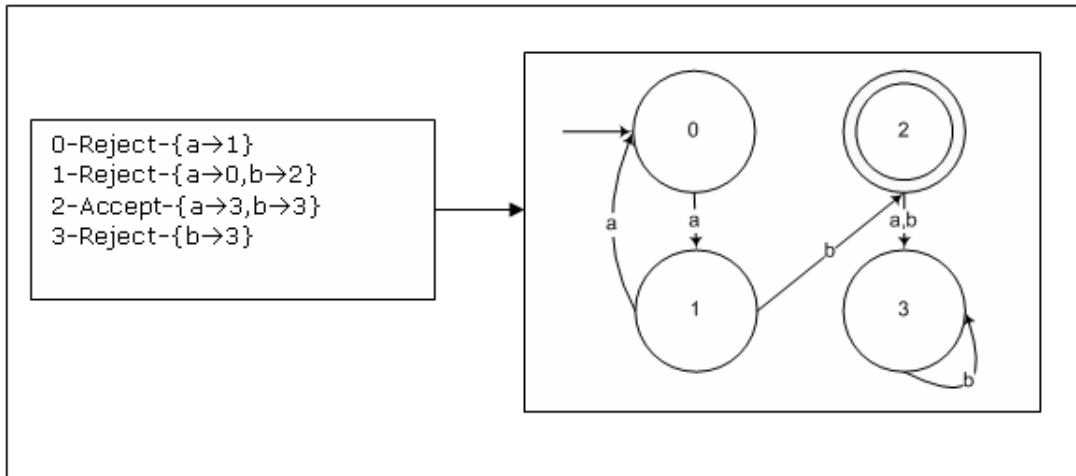


Figure 7.34: Example Solution for L12 (NFA1)

L13 (any sentence over the alphabet a, b with an even number of a 's): Solutions were generated for all seed values using both the standard genetic operators and non-destructive operators. The results for L13 are presented in Table A.49 and Table A.50. Figure 7.35 presents an example solution for L13.

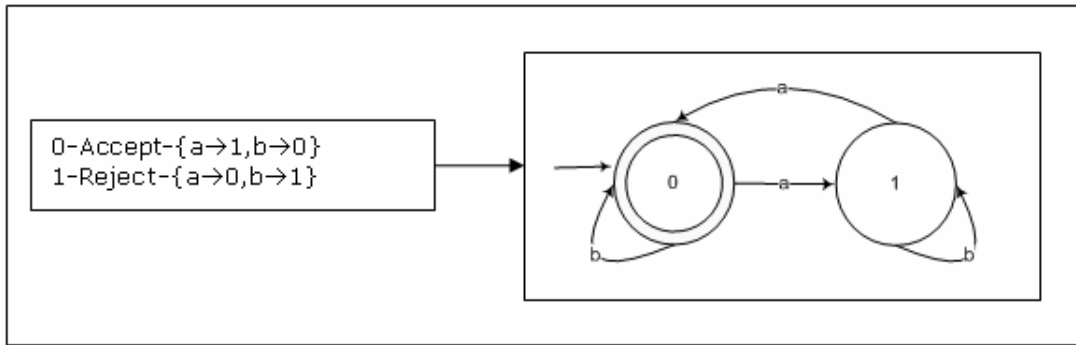


Figure 7.35: Example Solution for L13 (NFA1)

L14 ((aa)*ba*): Solutions were generated for all seed values using the standard genetic operators and non-destructive operators. Table A.51 and Table A.52 show the results for L14. Figure 7.36 presents an example solution for L14.

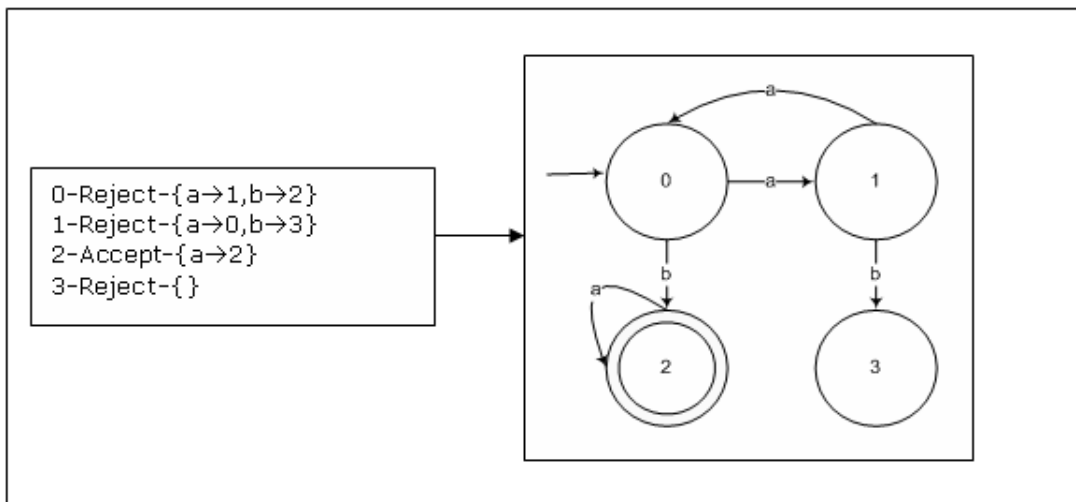


Figure 7.36: Example Solution for L14 (NFA1)

L15 (bc*b + ac*a): Two solutions were found using the standard genetic operators. Solutions were found for all seed values using non-destructive crossover and mutation. Table A.53 shows the results for the seed values using the standard genetic operators and Table A.54 shows the results using non-destructive operators. Figure 7.37 presents an example solution for L15.

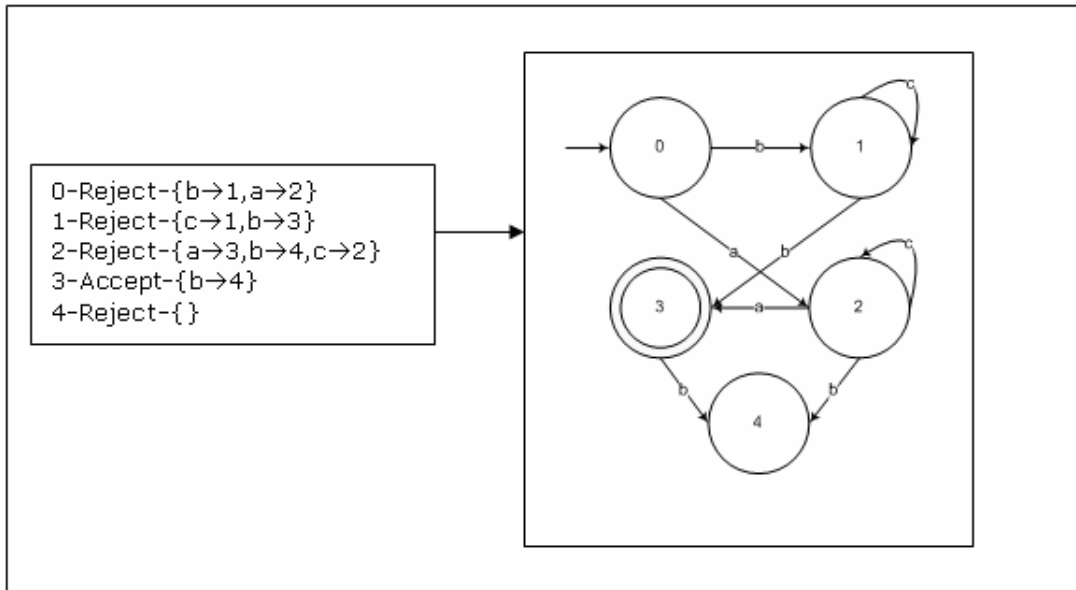


Figure 7.37: Example Solution for L15 (NFA1)

Solutions were found for all languages in the language set. Changing the genetic programming parameters does not improve the results for the system. Table 7.7 lists the success rates for each language. The success rate is calculated as a percentage of the seed values that produced solutions (using ten iterations for each seed value for all languages except for L9 and L15 which used twenty iterations). Using non-destructive crossover and mutation resulted in a 100% success rate for all languages.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 100% | 100% | 100% | 100% |
| L2 | 100% | 100% | 100% | 100% |
| L3 | 100% | 100% | 100% | 100% |
| L4 | 100% | 100% | 100% | 100% |
| L5 | 30% | 90% | 100% | 100% |
| L6 | 90% | 100% | 100% | 100% |
| L7 | 100% | 100% | 100% | 100% |
| L8 | 100% | 100% | 100% | 100% |
| L9 | 20% | 50% | 90% | 100% |
| L10 | 90% | 100% | 100% | 100% |
| L11 | 20% | 50% | 100% | 100% |
| L12 | 100% | 100% | 100% | 100% |
| L13 | 100% | 100% | 100% | 100% |
| L14 | 100% | 100% | 100% | 100% |
| L15 | 10% | 40% | 80% | 100% |

Table 7.7: Success Rates for NFA1 Simulations

Table 7.8 shows the percentage of seed values that required multiple iterations. Multiple iterations

were required for seven of the fifteen languages to overcome premature convergence caused by selection variance.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 0% | 0% | 0% | 0% |
| L2 | 0% | 0% | 0% | 0% |
| L3 | 40% | 30% | 0% | 0% |
| L4 | 0% | 0% | 0% | 0% |
| L5 | 100% | 90% | 60% | 60% |
| L6 | 80% | 60% | 10% | 0% |
| L7 | 0% | 0% | 0% | 0% |
| L8 | 0% | 0% | 0% | 0% |
| L9 | 100% | 100% | 90% | 90% |
| L10 | 80% | 10% | 10% | 10% |
| L11 | 100% | 90% | 70% | 50% |
| L12 | 0% | 0% | 0% | 0% |
| L13 | 0% | 0% | 0% | 0% |
| L14 | 20% | 0% | 0% | 0% |
| L15 | 100% | 100% | 90% | 80% |

Table 7.8: Multiple iterations required for NFA1 Simulations

7.1.4 Nondeterministic Finite Acceptors (2)

The results for each language in the language set are outlined below:

L1 (a*): Solutions were evolved for all of the random seed values. All solutions were found in the initial population; therefore no genetic operators were used for this language.

Figure 7.38 presents an example solution for L1.

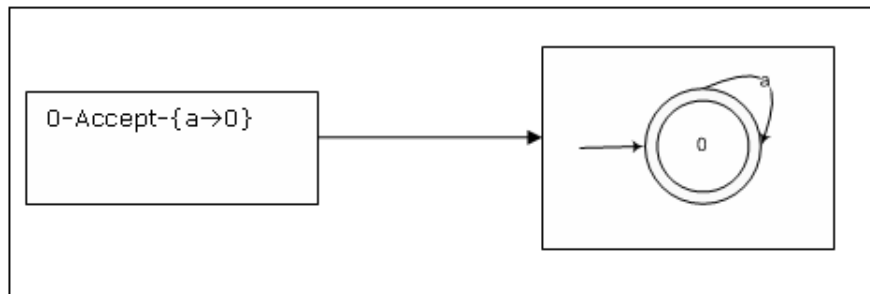


Figure 7.38: Example Solutions for L1 (NFA2)

L2 (ba)*: As with L1, solutions were generated for all seed values on the initial generation of the population for L2.

Figure 7.39 presents an example solution for L2.

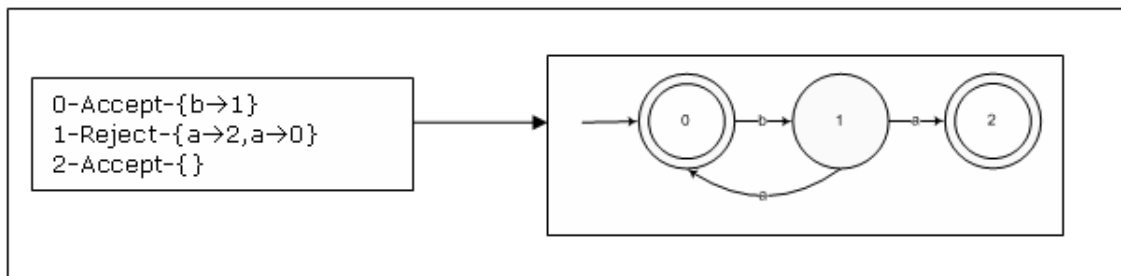


Figure 7.39: Example Solution for L2 (NFA2)

L3 (any sentence without an odd number of consecutive a's after an odd number of consecutive b's): Solutions were generated for all seeds using the standard genetic operators and non-destructive operators. Table A.57 and Table A.58 outline the results obtained for L3. Figure 7.40 presents an example solution for L3.

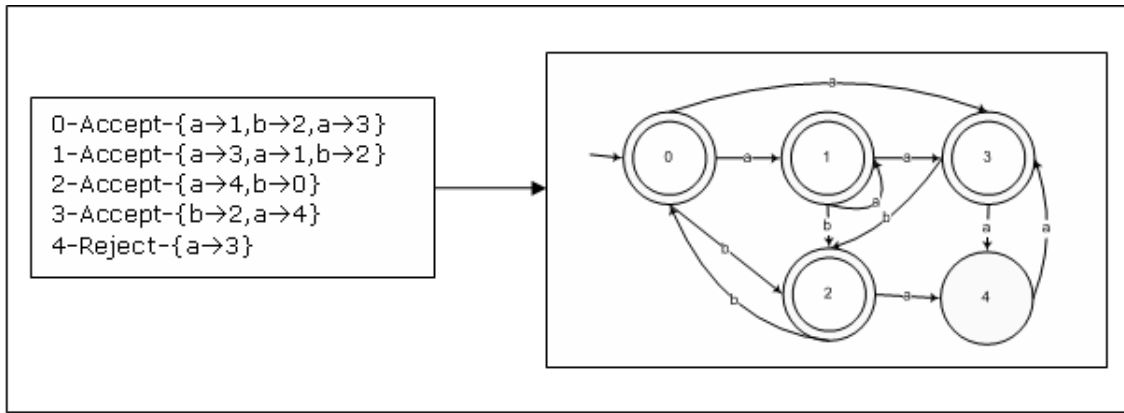


Figure 7.40: Example Solution for L3 (NFA2)

L4 (any sentence over the alphabet a, b without more than two consecutive a's): Solutions were obtained for all seed values using either the standard genetic operators or non-destructive operators. Table A.59 and Table A.60 shows the results obtained for L4. Figure 7.41 presents an example solution for L4.

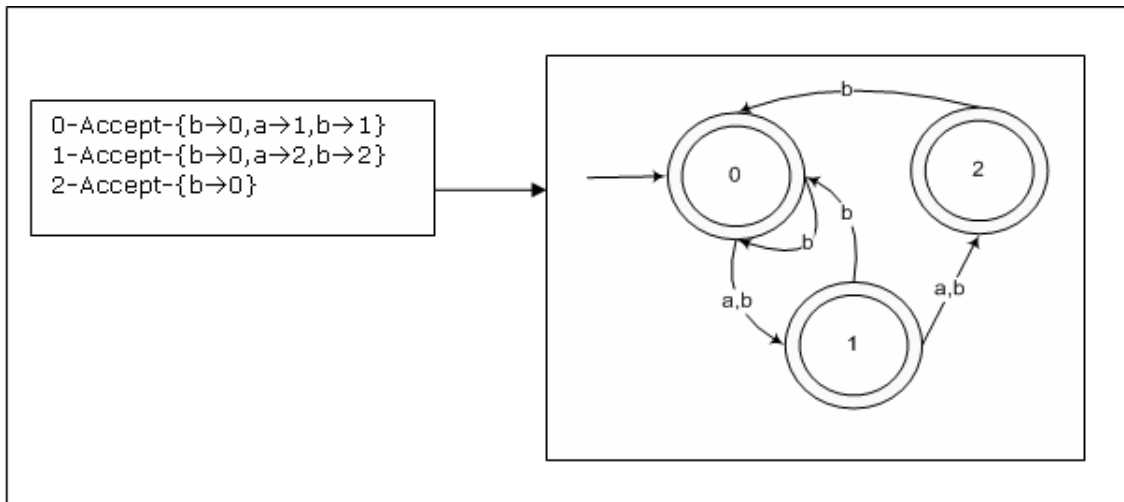


Figure 7.41: Example Solution for L4 (NFA2)

L5 (any sentence with an even number of a's and an even number of b's): No solutions were evolved using the standard genetic operators whereas solutions were evolved for all seed values using non-destructive crossover and mutation. Table A.61 and Table A.62 presents the results for L5. Figure 7.42 presents an example solution for L5.

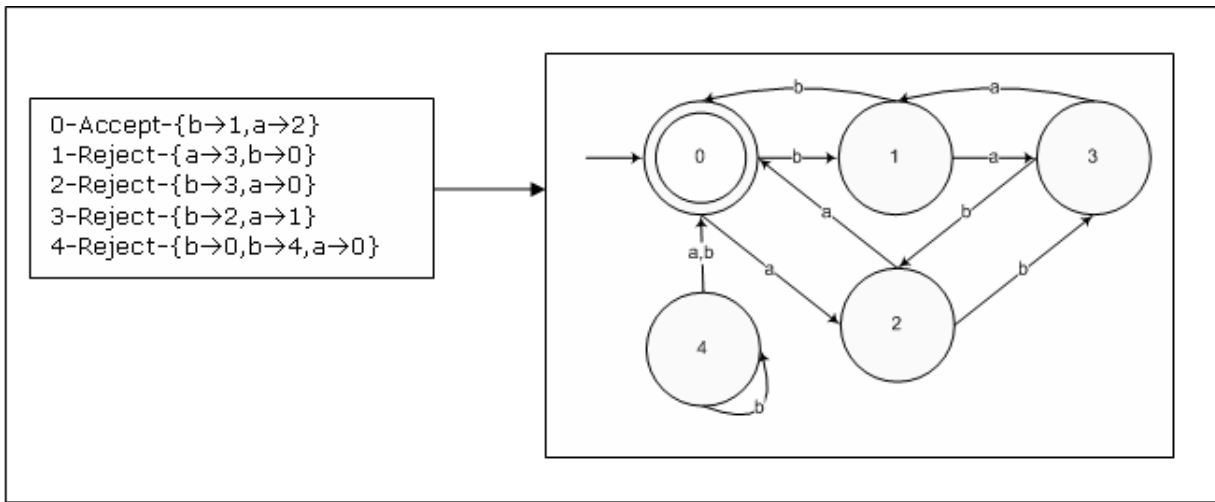


Figure 7.42: Example Solution for L5 (NFA2)

L6 (any sentence such that the number of a's differs from the number of b's by 0 modulo 3): Solutions were found for seven of the ten seed values using the standard genetic operators. Solutions were found for all seed values using non-destructive operators. The results are outlined in Table A.63 and Table A.64. Figure 7.43 presents an example solution for L6.

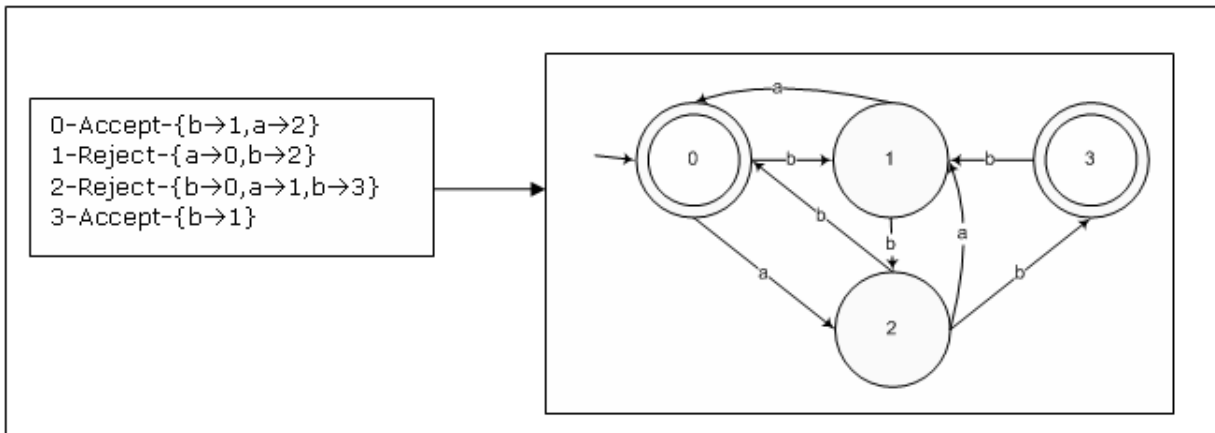


Figure 7.43: Example Solution for L6 (NFA2)

L7 ($a^*b^*a^*b^*$): Solutions were generated for all seed values using either the standard genetic operators or non-destructive operators. The results for L7 are presented in Table A.65 and Table A.66. Figure 7.44 presents an example solution for L7.

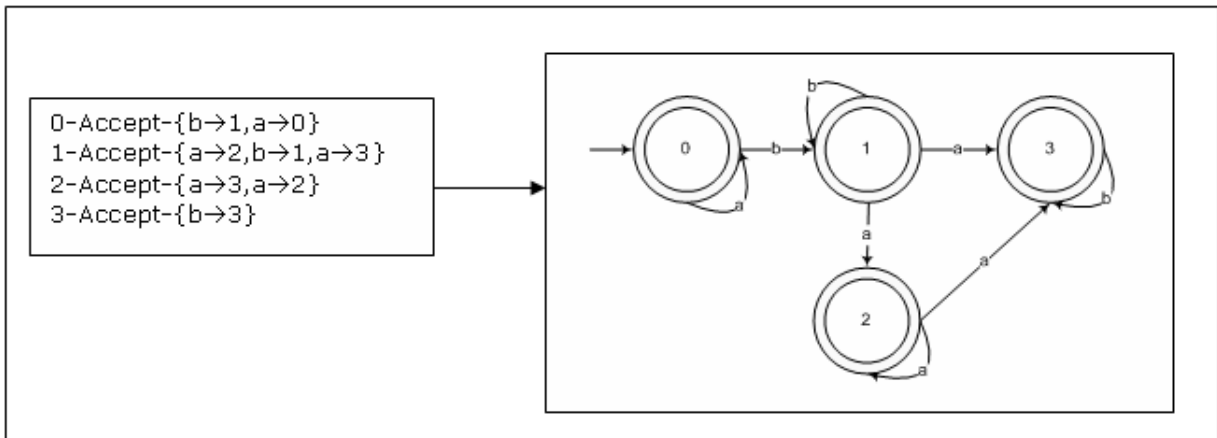


Figure 7.44: Example Solution for L7 (NFA2)

L8 (a^*b): Solutions were generated for all seed values on the initial generation of the population. Figure 7.45 presents an example solution for L8.

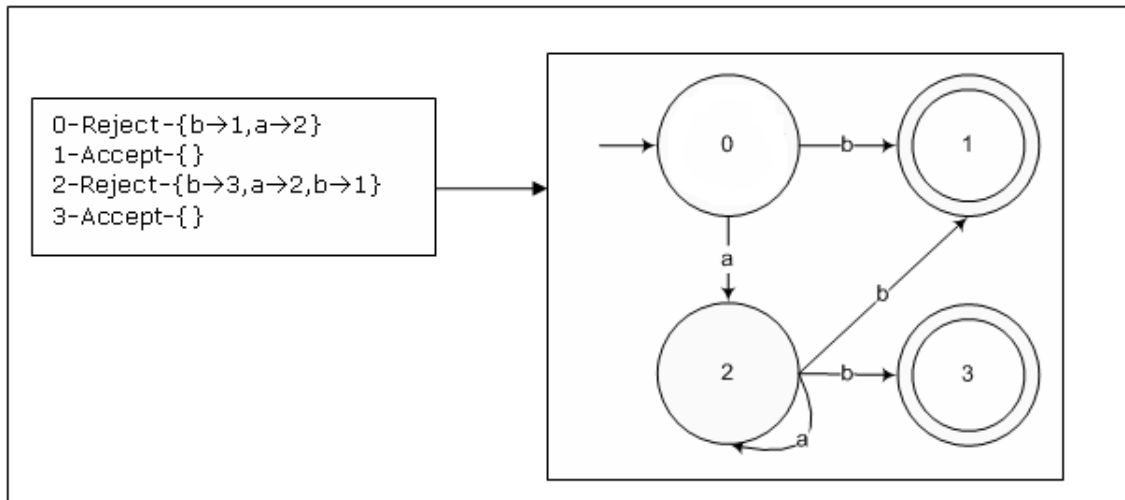


Figure 7.45: Example Solution for L8 (NFA2)

L9 ($(a^* + c^*)b$): Two seed values led to solutions being generated for L9 for the standard genetic operators. Solutions were generated for eight of the ten seed values using non-destructive crossover and mutation. Table A.68 and Table A.69 present the results for L9. Figure 7.46 presents an example solution for L9.

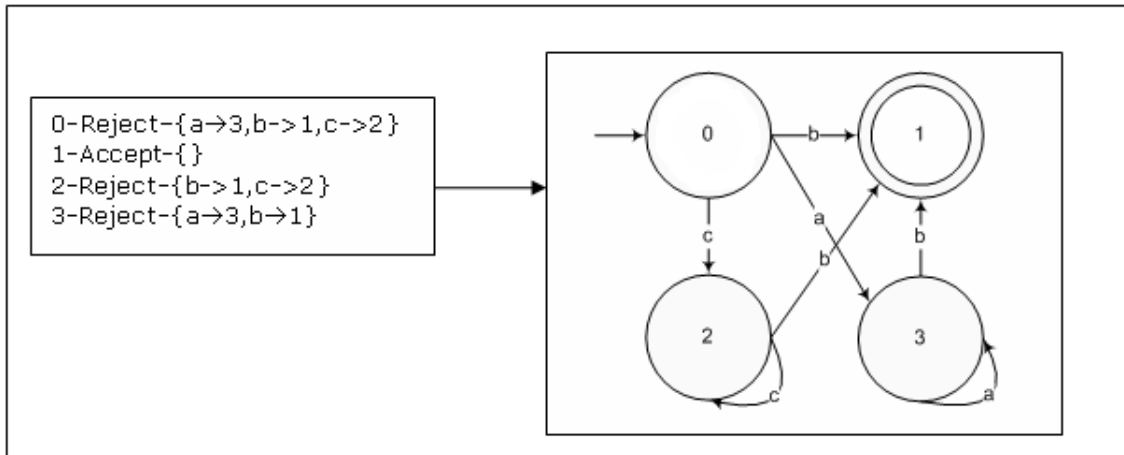


Figure 7.46: Example Solution for L9 (NFA2)

L10 $((aa)^*(bbb)^*$): No solutions were evolved when using the standard genetic operators. Solutions were evolved for nine of the ten seed values using non-destructive crossover and mutation. The results for L10 are outlined in Table A.70 and Table A.71. Figure 7.47 presents an example solution for L10.

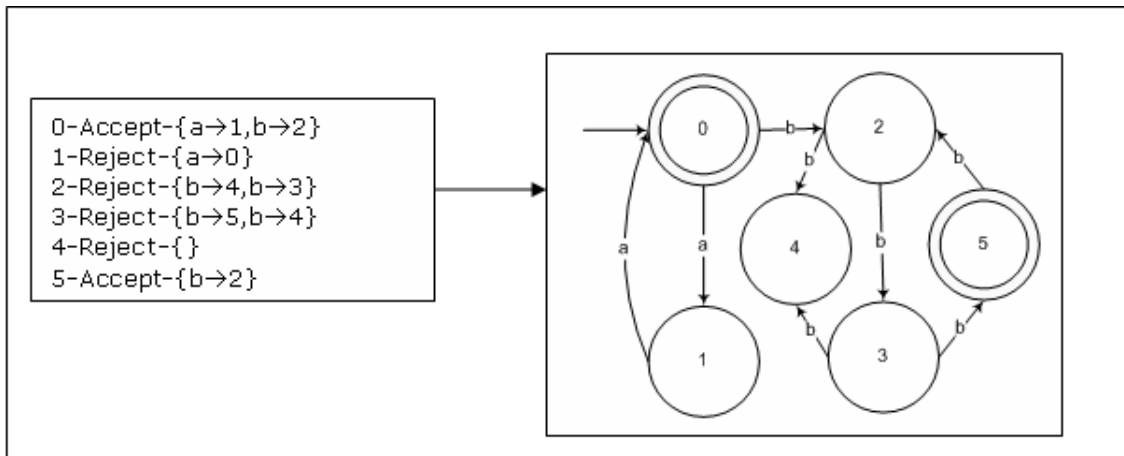


Figure 7.47: Example Solution for L10 (NFA2)

L11 (any sentence with an even number of a's and an odd number of b's): Only one seed value produced solutions when applying the standard genetic operators. Solutions were found for all seed values using non-destructive operators. Table A.72 and Table A.73 outline the results for L11. Figure 7.48 presents an example solution for L11. Figure 7.49 presents the minimized NFA for L11.

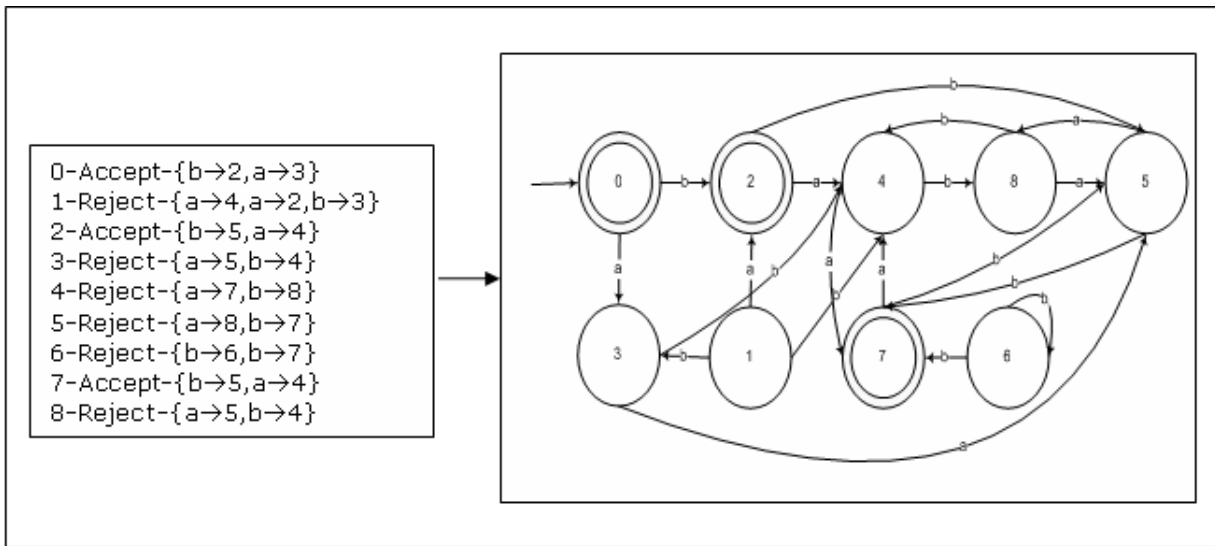


Figure 7.48: Example Solution for L11 (NFA2)

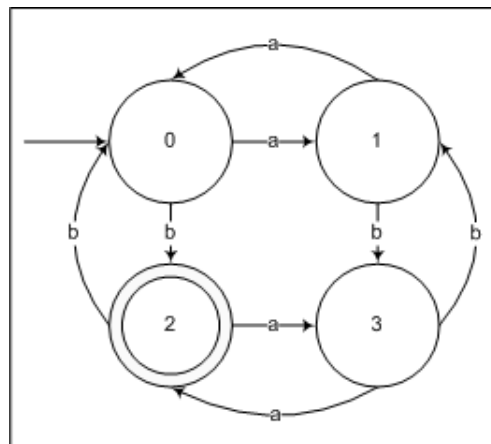


Figure 7.49: Minimized NFA for L11

L12 (a(aa)*b): Solutions were generated for all seed values using the standard genetic operators and non-destructive operators. The results are outlined in Table A.74 and Table A.75. Figure 7.50 presents an example solution for L12.

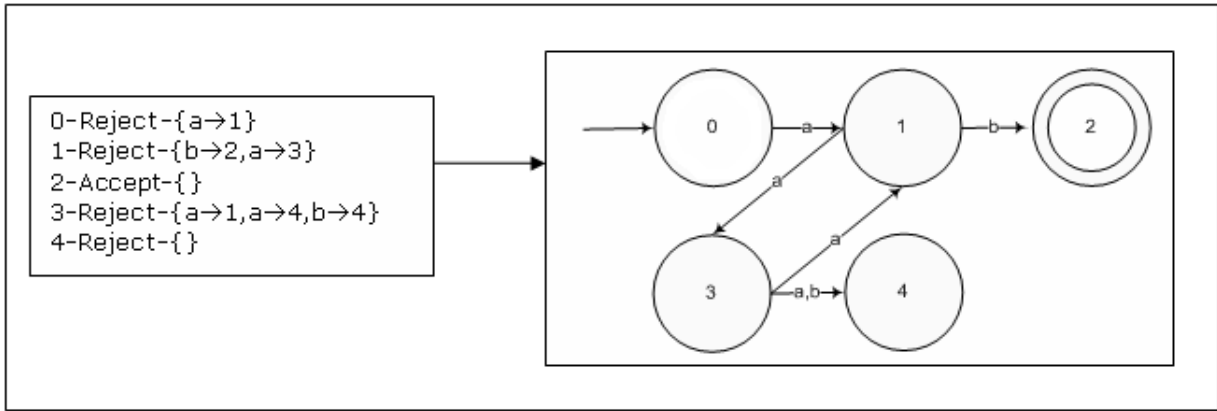


Figure 7.50: Example Solution for L12 (NFA2)

L13 (any sentence over the alphabet a, b with an even number of a's): Solutions were generated for all seed values using both the standard genetic operators and non-destructive operators. The results for L13 are presented in Table A.76 and Table A.77. Figure 7.51 presents an example solution for L13.

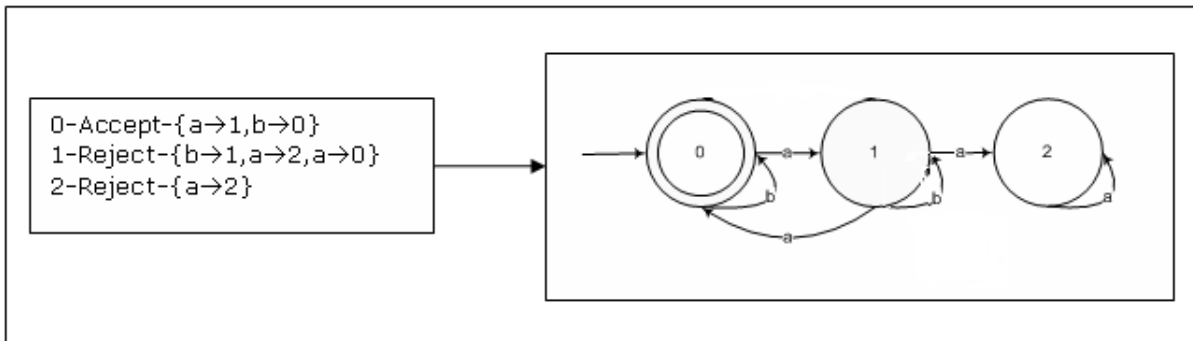


Figure 7.51: Example Solution for L13 (NFA2)

L14 ((aa)*ba*): Solutions were generated for all seed values using the standard genetic operators and non-destructive operators. Table A.78 and Table A.79 show the results for L14. Figure 7.52 presents an example solution for L14.

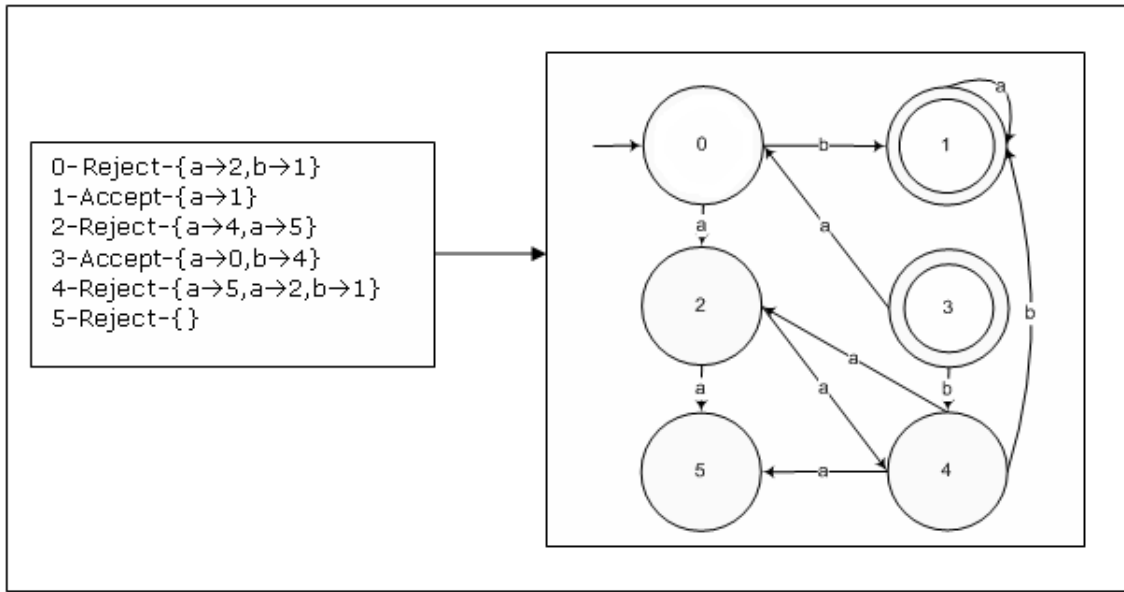


Figure 7.52: Example Solution for L14 (NFA2)

L15 ($bc^*b + ac^*a$): Solutions were generated for two seed values when applying the standard genetic operators and for all seed values when applying non-destructive crossover and non-destructive crossover and mutation. Table A.80 presents the results for the standard genetic operators and Table A.81 presents the results when using non-destructive operators. Figure 7.53 presents an example solution for L15.

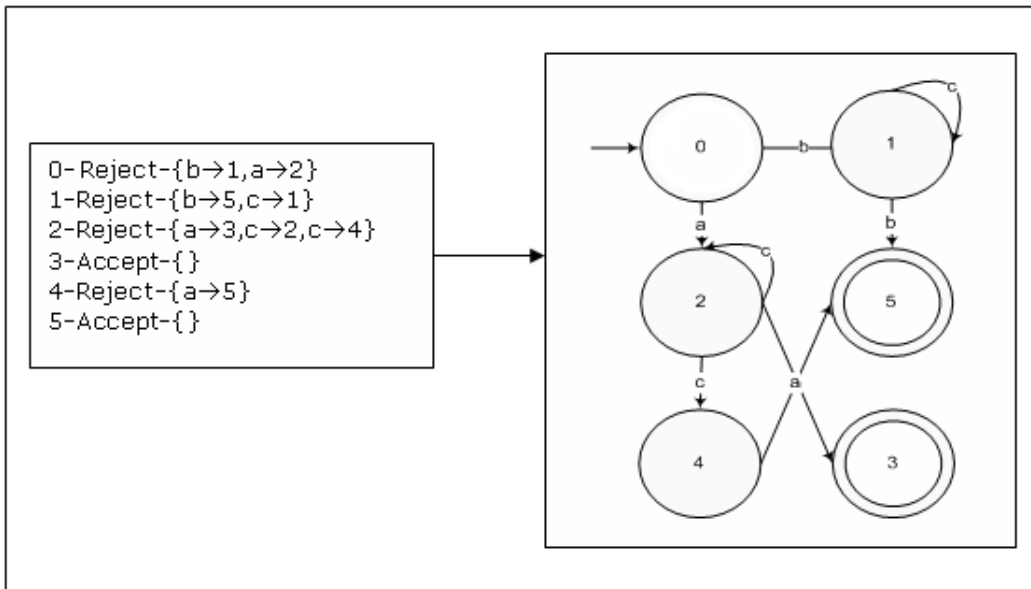


Figure 7.53: Example Solution for L15 (NFA2)

Solutions were induced for all languages in the language set. It is noticeable that applying the non-destructive operators produces improved results when compared to the results produced using the standard genetic operators. Altering the GP parameters for the languages do not improve the results obtained.

Table 7.9 lists the success rates for each language. The success rate is calculated as a percentage of the seed values that produced solutions (using ten iterations for each seed value for all languages except for L9 and L15 which used twenty iterations for each seed value).

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 100% | 100% | 100% | 100% |
| L2 | 100% | 100% | 100% | 100% |
| L3 | 100% | 100% | 100% | 100% |
| L4 | 100% | 100% | 100% | 100% |
| L5 | 0% | 30% | 80% | 100% |
| L6 | 70% | 100% | 100% | 100% |
| L7 | 100% | 100% | 100% | 100% |
| L8 | 100% | 100% | 100% | 100% |
| L9 | 20% | 10% | 70% | 80% |
| L10 | 0% | 20% | 20% | 90% |
| L11 | 10% | 20% | 100% | 100% |
| L12 | 100% | 100% | 100% | 100% |
| L13 | 100% | 100% | 100% | 100% |
| L14 | 100% | 100% | 100% | 100% |
| L15 | 20% | 90% | 100% | 100% |

Table 7.9: Success Rates for NFA2 Simulations

Table 7.10 shows the percentage of seed values that required multiple iterations. Multiple iterations were required for nine of the fifteen languages to overcome premature convergence caused by selection variance.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 0% | 0% | 0% | 0% |
| L2 | 0% | 0% | 0% | 0% |
| L3 | 60% | 40% | 0% | 10% |
| L4 | 0% | 0% | 0% | 0% |
| L5 | 100% | 100% | 80% | 80% |
| L6 | 90% | 70% | 40% | 30% |
| L7 | 10% | 0% | 0% | 0% |
| L8 | 0% | 0% | 0% | 0% |
| L9 | 100% | 100% | 80% | 100% |
| L10 | 100% | 100% | 100% | 100% |
| L11 | 100% | 100% | 70% | 80% |
| L12 | 40% | 0% | 0% | 0% |
| L13 | 0% | 0% | 0% | 0% |
| L14 | 40% | 20% | 0% | 0% |
| L15 | 100% | 100% | 100% | 80% |

Table 7.10: Multiple iterations required for NFA2 Simulations

7.1.5 Comparison of Results

All systems were tested using the same language set and the same GP parameters. For each of the GP systems, the same methods i.e. destructive, non-destructive crossover, non-destructive mutation, and non-destructive crossover and mutation were used. Each solution was evolved in under a minute for DFAs and NFA1 and within five minutes for NFA2 due to the search space being larger for NFA2.

Below is a comparison between the three systems for each language in the language set.

L1 (a^*): In all three approaches solutions for L1 were generated during initial generation of the population for all seed values used to test the system. All systems were able to evolve the minimal FA for L1.

L2($(ba)^*$): All systems used produced a 100% success rate for L2. Solutions were generated for L2 during initial generation of the population for nondeterministic acceptors (both approaches). When using the DFA system for L2, solutions were found for all seed values. Unlike the nondeterministic systems these results were obtained after the evolution of a few generations. The nondeterministic systems are therefore the fastest² method to generate solutions for L2.

L3 (any sentence without an odd number of consecutive a's after an odd number of consecutive b's): All systems produced a 100% success rate for L3. However, the first approach to generating nondeterministic finite acceptors was the fastest method for generating solutions for L3 with solutions being generated for all seed values on the first iteration when using non-destructive crossover and non-destructive crossover and mutation.

L4 (any sentence over the alphabet a, b without more than two consecutive a's): All systems produced a 100% success rate for L4. Multiple iterations for each seed value were only required for destructive operators in the DFA system.

L5 (any sentence with an even number of a's and an even number of b's): It is apparent from the results generated for L5 that the DFA system was most successful in generating solutions for this language. The NFA1 system did produce better results with an increase in the population size; however the DFA system still remained the better approach for this language with a 100% success rate.

L6 (any sentence such that the number of a's differs from the number of b's by 0 modulo 3): As with L5, the DFA system produces the best results for L6 with solutions being generated for all seed values using destructive operators and non-destructive operators. The NFA systems produce a 100% success rate when using non-destructive operators whereas destructive operators led to a 90% and 70% success rate for NFA1 and NFA2 respectively.

L7 ($a^*b^*a^*b^*$): All systems produced a 100% success rate for L7. The DFA system required multiple iterations for 10% of the seed values for destructive operators, non-destructive crossover and non-destructive crossover and mutation. The NFA1 system on the other hand did not require any

²The fastest method implies that this method took the least number of generations to evolve the solution.

multiple iterations as all solutions were found on the initial generation and the NFA2 system only required multiple iterations for 10% of the seed values for destructive operators.

L8 (a^*b): The NFA systems induce solutions to L8 for all seed values during initial generation of the population. The DFA system generates all solutions either during initial generation of the population or on the 2nd generation of the population.

L9 ($(a^* + c^*)b$): Only two DFA solutions were found for L9 using non-destructive operators. However, applying a modular approach for DFAs improves the results for L9 with a 100% success rate using non-destructive operators. The NFA1 system produced a 100% success rate for L9 using non-destructive crossover and mutation. The NFA2 system on the other hand produced an 80% success rate using non-destructive crossover and mutation.

L10 ($(aa)^*(bbb)^*$): It is noticeable that the NFA1 system produces the best results for L10. The system was able to induce solutions using destructive operators with a success rate of 90%. Non-destructive operators led to a success rate of 100%. The DFA system was able to produce a 100% success rate for non-destructive crossover and non-destructive crossover and mutation. Destructive operators on the other hand had a 50% success rate. The NFA2 system produced a success rate of 90% when applying non-destructive crossover and mutation. However, no solutions were generated when applying destructive operators.

L11 (any sentence with an even number of a's and an odd number of b's): For this language the DFA system produced the best results with all methods generating solutions for all seed values. The NFA systems produced comparable results using non-destructive operators, but only had a 20% success rate for NFA1 and a 10% success rate for NFA2 using destructive operators.

L12 ($a(aa)^*b$): All systems produced a 100% success rate for L12. The NFA1 system clearly produces the fastest results seeing that solutions are generated for all seed values on the first run using any of the methods (destructive or non-destructive operators).

L13 (any sentence over the alphabet a, b with an even number of a's): All systems produced a 100% success rate for L13. All DFA solutions were generated during initial population generation. For the NFA systems, on the other hand, results were obtained after the evolution of a few generations.

L14 ($(aa)^*ba^*$): All systems produced a 100% success rate for L14 using both the standard genetic operators and non-destructive operators.

L15 ($bc^*b + ac^*a$): The DFA system produced a 30% success rate when applying destructive operators and non-destructive mutation. Using a modular approach for DFAs improved the success rate to 100%. The NFA1 system produced a 100% success rate when applying non-destructive crossover and non-destructive crossover and mutation and an 80% success rate for destructive operators. The

NFA2 system produced the best results with a 100% success rate for all methods.

Table 7.11 shows the comparison between human generated solutions [11], and the DFAs and NFAs generated by the GP system. The table above shows the minimized DFAs and NFAs generated by the GP systems. The human generated solutions are nondeterministic and it is clear that the minimized NFA solutions generated by the system are the same. The DFAs on the other hand are similar in structure containing at most one extra reject state.

| | Human generated solutions | GP Generated DFA | GP Generated NFA |
|----|---------------------------|------------------|------------------|
| L1 | | | |
| L2 | | | |
| L3 | | | |
| L4 | | | |
| L5 | | | |
| L6 | | | |
| L7 | | | |
| L8 | | | |

| | Human generated solutions | GP Generated DFA | GP Generated NFA |
|-----|---------------------------|------------------|------------------|
| L9 | | | |
| L10 | | | |
| L11 | | | |
| L12 | | | |
| L13 | | | |
| L14 | | | |
| L15 | | | |

Table 7.11: Comparison to Human Generated Solutions

Table 7.12 shows the average number of redundant nodes for each language when comparing the solutions generated by the GP systems to the minimized FAs. The minimized NFAs are compared to the NFAs generated by the NFA1 system. The average number of redundant nodes are calculated by getting the average number of nodes for all solutions generated by the GP systems and subtracting the number of nodes in the minimized solution. The systems evolve solutions with redundant nodes on average for fourteen of the languages for DFAs and for ten languages for NFAs.

| Language | Number of redundant nodes | |
|----------|---------------------------|------|
| | DFAs | NFAs |
| L1 | 1 | - |
| L2 | 1 | - |
| L3 | 2 | 2 |
| L4 | 1 | - |
| L5 | 1 | 3 |
| L6 | 1 | 2 |
| L7 | 2 | 1 |
| L8 | 1 | - |
| L9 | 2 | 2 |
| L10 | 2 | 3 |
| L11 | 1 | - |
| L12 | 3 | 1 |
| L13 | - | 1 |
| L14 | 3 | 1 |
| L15 | 4 | 1 |

Table 7.12: Number of redundant nodes

Table 7.13 shows a comparison of success rates for the Tomita languages between the GP system presented in this thesis and the previous work done.

| Language | Brave | Luke et al. (2) | Dunay et al. | Lucas et al. | | GP System (DFA) |
|----------|-------|-----------------|--------------|--------------|--------|-----------------|
| | | | | Smart | nSmart | |
| L1 | 100% | 100% | 100% | 81.8% | 100% | 100% |
| L2 | 100% | 100% | 100% | 88.8% | 95.5% | 100% |
| L3 | 100% | 75% | 100% | 71.8% | 90.8% | 100% |
| L4 | 100% | 100% | 96.7% | 61.1% | 100% | 100% |
| L5 | 100% | 5% | 63.3% | 65.9% | 100% | 100% |
| L6 | 0% | 65% | 93.3% | 61.9% | 100% | 100% |
| L7 | 100% | 70% | 96.7% | 62.9% | 82.9% | 100% |

Table 7.13: Comparison with previous work

Comparing the DFA results to the previous work done by Brave [7], the GP systems presented here uses less fitness cases and a smaller population. Brave achieved a success rate of 100% for all of the Tomita languages except L6 (any sentence such that the number of a's differs from the number of b's by 0 modulo 3). No solutions were evolved for L6 and this failure was attributed to the representation

used in the Brave system. The GP system presented in this thesis for DFAs on the other hand, was able to achieve a 100% success rate for all seven Tomita languages.

Luke et al. [33], as indicated in Chapter 4, perform two experiments³ using the seven Tomita languages. In the first experiment results were evolved for all languages except L5 (any sentence with an even number of a's and an even number of b's). Table 7.14 shows a comparison between the DFA GP system and the system proposed by Luke et al. The table shows the average and best classification rates for each system. The classification rate is defined as being the number of sentences an individual can correctly classify. The best classification rate is therefore the number of sentences the best individual in the population can correctly classify.

| Language | Luke et al. | | GP System (DFA) | |
|----------|-------------|--------|-----------------|------|
| | Average | Best | Average | Best |
| L1 | 88.39% | 100% | 100% | 100% |
| L2 | 84.00% | 100% | 100% | 100% |
| L3 | 66.28% | 100% | 100% | 100% |
| L4 | 65.25% | 100% | 100% | 100% |
| L5 | 68.65% | 82.94% | 100% | 100% |
| L6 | 95.94% | 100% | 100% | 100% |
| L7 | 67.69% | 100% | 100% | 100% |

Table 7.14: Comparison with Luke et al. (Experiment 1)

Luke et al. do not specify the success rates for experiment 1, so it is difficult to do a comparison. The classification rate of the DFA system however, clearly performs better than that of Luke et al. The better performance could be attributed to the larger population size used by the DFA system. Luke et al. use a population size four times smaller. The number of generations and the number of runs used to test the system proposed by Luke et al., however, are larger than that used by the DFA system.

For experiment 2, Luke et al. were able to generate solutions for all seven languages. The success rates are compared with the DFA GP system in Table 7.13.

Dunay et al. [14] were able to generate solutions for all seven Tomita languages. The success rates are shown in Table 7.13. Dunay et al. represented each individual as a tree and hence found difficulty in representing back pointers. They attribute the poor success rate of L5 to this difficulty.

Lucas et al. [31] were able to successfully generate solutions for all seven Tomita languages. Table 7.13 shows the success rate of the proposed systems. Lucas et al. tested the system using two approaches i.e., Smart and nSmart. The Smart approach set the maximum number of nodes to ten and the nSmart approach sets the number of states to be exactly that of the number of states in the minimal DFA. The nSmart approach did outperform the Smart approach but required some knowledge of the DFA before implementation.

³For more details on the difference between the two experiments see Section 4.2 in Chapter 4.

The system proposed by Dupont [15] presents results as the average classification rate over ten runs as well as the best classification over the ten runs. Table 7.15 shows a comparison between Dupont and the GP systems showing the best classification rates of each system. This indicates that the Dupont system was able to achieve solutions for only five of the fifteen languages. This implies that the solutions generated by the Dupont system do not generalize well.

| Language | Dupont | GP Systems |
|----------|--------|------------|
| L1 | 100% | 100% |
| L2 | 100% | 100% |
| L3 | 99.4% | 100% |
| L4 | 90.6% | 100% |
| L5 | 83.5% | 100% |
| L6 | 95% | 100% |
| L7 | 100% | 100% |
| L8 | 100% | 100% |
| L9 | 99.6% | 100% |
| L10 | 99.9% | 100% |
| L11 | 72.2% | 100% |
| L12 | 99.8% | 100% |
| L13 | 100% | 100% |
| L14 | 99.9% | 100% |
| L15 | 99.5% | 100% |

Table 7.15: Comparison with Dupont’s work

Hingston [20] tests his system on all fifteen benchmark problems with a only four of the fifteen languages producing solutions. Both Dupont and Hingston give the classification rates so a comparison of the success rates is not possible.

For all languages the GP systems presented in this thesis for FAs performed comparably, and in some cases outperformed the other methods implemented for this purpose.

7.2 Finite State Transducers

The system was applied using ten seed values for each language in the language set. The ten random seed values used are 1580, 2978, 3478, 3908, 4600, 5210, 6541, 6908, 7500 and 8080. The program was tested using both the standard genetic operators and non-destructive operators. The non-destructive operators are implemented by ensuring that the fitness of the child is better than or equal to the fitness of the parent when applying crossover and mutation. Non-destructive operators are applied to avoid the destructive effects of crossover and mutation. If no solutions were found for a particular seed value multiple iterations for this seed were performed to deal with selection variance.

The results for each language in the language set are outlined below and the tables mentioned in this section can be found in Appendix B:

L1 (Mealy machine that outputs a '1' for each substring 'aaa'. $\Sigma=\{a,b\}$): Solutions were evolved for all of the random seed values. Table B.1 presents the results obtained for L1 using the standard genetic operators. Multiple iterations were required for the standard genetic operators. Table B.2 show the results for L1 with non-destructive operators. Solutions for all seed values were evolved on the initial iteration therefore multiple iterations were not required.

Figure 7.54 shows an example solution.

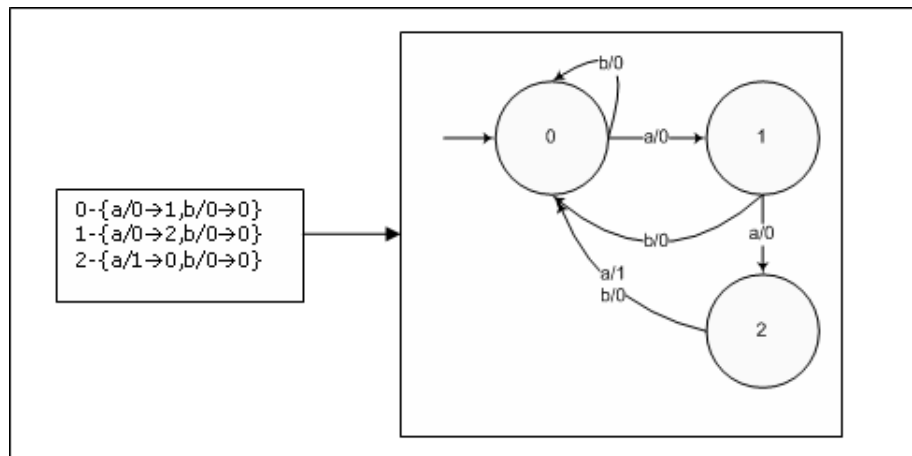


Figure 7.54: Example Solution for L1 (FST)

L2 (Mealy machine that outputs a '1' for each substring 'aab'. $\Sigma=\{a,b\}$): Solutions were generated for all seed values using the standard genetic operators and non-destructive operators on the first run. Table B.3 and Table B.4 presents the results obtained for L2. Multiple iterations are not required for L2 as all solutions were generated on the initial iteration for each seed value.

Figure 7.55 shows an example solution.

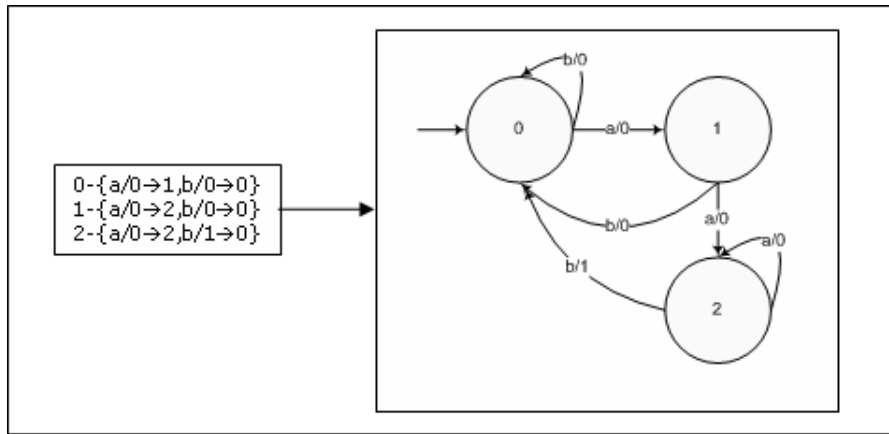


Figure 7.55: Example Solution for L2 (FST)

L3 (Mealy machine that takes in a binary string in reverse order and outputs the number incremented by 1): Solutions were generated for all seed values either during initial generation of the population or on the second generation of the population. The results for non-destructive operators are the same as when applying the standard genetic operators. Table B.5 shows the results for L3.

Figure 7.56 shows an example solution.

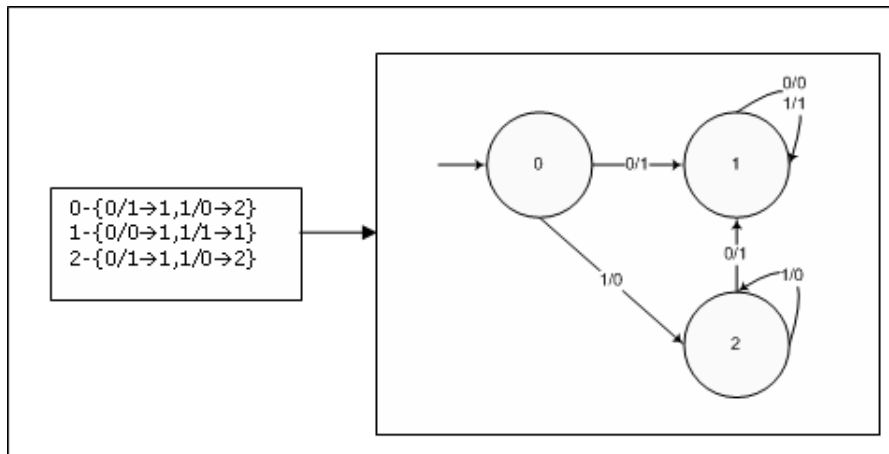


Figure 7.56: Example Solution for L3 (FST)

L4 (Mealy machine that outputs a '1' at every position of a double letter.): Solutions were generated for all seed values using the standard genetic operators or non-destructive operators. Table B.6 and Table B.7 shows the results obtained for L4.

Figure 7.57 shows an example solution.

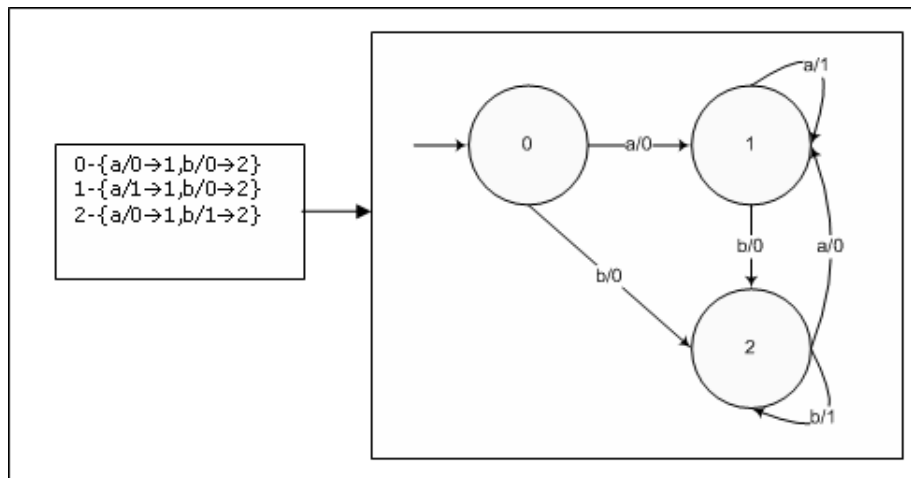


Figure 7.57: Example Solution for L4 (FST)

L5 (Mealy machine that takes in a binary string and outputs the one's complement of the string): Solutions were induced for all seed values during the initial generation of the population. Therefore the genetic operators are not used and hence solutions for L5 are generated using random search.

Figure 7.58 shows an example solution.

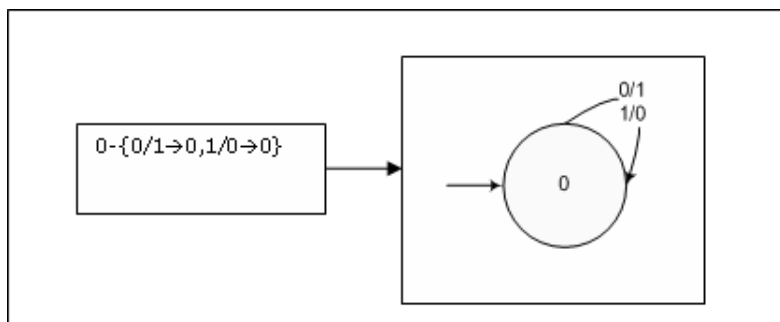


Figure 7.58: Example Solution for L5 (FST)

L6 (Mealy machine that reads in a binary string and outputs an 'E' if the number of 1s read in so far is even and an 'O' if it is odd): Solutions were induced for all seed values during the initial generation of the population. As with L5, solutions for L6 are generated using random search.

Figure 7.59 shows an example solution.

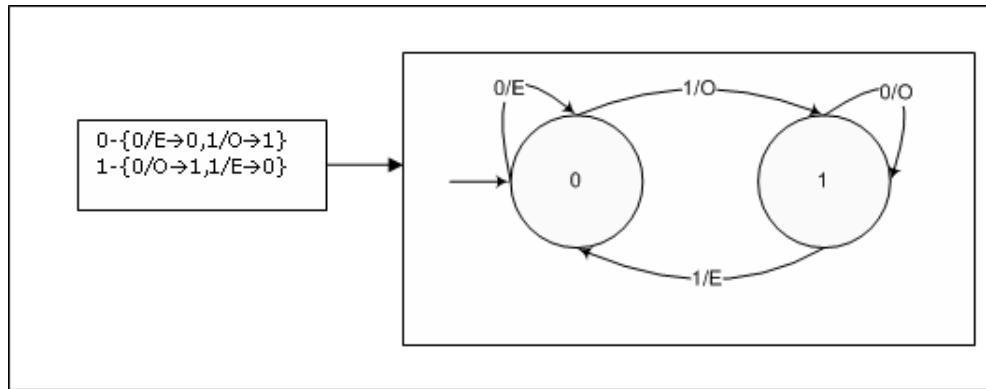


Figure 7.59: Example Solution for L6 (FST)

Solutions were successfully generated for all languages in the language set. Furthermore the system was able to generate the minimal FST for each language. A single run for each language took less than a minute to complete. It is also noticeable that multiple iterations were only required for L1, indicating that the problem of selection variance did not adversely affect this system. Table 7.16 shows the success rates for both standard and non-destructive operators when using only one iteration for each seed value. Non-destructive operators are not necessary for L5 and L6. All solutions for L5 and L6 were found on the initial generation of the population and therefore no genetic operators were applied.

| Language | Standard Operators | Non-destructive Operators |
|----------|--------------------|---------------------------|
| L1 | 100% | 100% |
| L2 | 100% | 100% |
| L3 | 100% | 100% |
| L4 | 100% | 100% |
| L5 | 100% | ⁴ - |
| L6 | 100% | - |

Table 7.16: Success Rates for FST Simulations over 10 runs

Table 7.17 compares the generated solutions to “Human generated” solutions as shown in [9] and [17]. The GP system was not only capable of generating the minimal transducer but was also able to evolve solutions equivalent to the “Human generated” solutions. None of the FST examples presented in this section have redundant nodes. This is not always true for all solutions however. Some solutions generated by the system did contain a few redundant nodes.

⁴Non-destructive operators were not necessary for this language.

| | "Human Generated" Solution | GP Generated Solution |
|----|----------------------------|-----------------------|
| L1 | | |
| L2 | | |
| L3 | | |
| L4 | | |
| L5 | | |
| L6 | | |

Table 7.17: Comparison of Solutions

Table 7.18 shows the average number of redundant nodes for each language when comparing the

solutions generated by the GP systems to the human generated solutions. The average number of redundant nodes are calculated by getting the average number of nodes for all solutions generated by the GP systems and subtracting the number of nodes in the human generated solution. The systems evolve solutions with redundant nodes on average for two languages.

| Language | Number of redundant nodes |
|-----------------|----------------------------------|
| L1 | 1 |
| L2 | 1 |
| L3 | - |
| L4 | - |
| L5 | - |
| L6 | - |

Table 7.18: Number of redundant nodes

7.3 Pushdown Automata

The system was applied using ten seed values for each language in the language set. The ten random seed values used are 1097, 1111, 2121, 2134, 3003, 3012, 5678, 7777, 8530 and 9090. The seed values were chosen randomly. The program was tested using both the standard genetic operators and non-destructive operators. The non-destructive operators are implemented by ensuring that the fitness of the child is better than or equal to the fitness of the parent when applying crossover and mutation. Non-destructive operators are applied to avoid the destructive effects of crossover and mutation. If no solutions were found for a particular seed value multiple iterations (the system was tested using ten iterations for each seed value except L3 where twenty iterations were applied) for this seed were performed to deal with selection variance. A string is accepted by the PDA if at least one path ends in an accept state. A string is rejected by the PDA if all paths end in a reject state. All the tables referred to in this section can be found in Appendix C.

L1 ($a^n b^n$ where $n \geq 0$): The results for L1 are outlined in Table C.1 and Table C.2. Solutions were induced for all seed values using both the standard genetic operators and non-destructive operators. Figure 7.60 presents an example solution for L1.

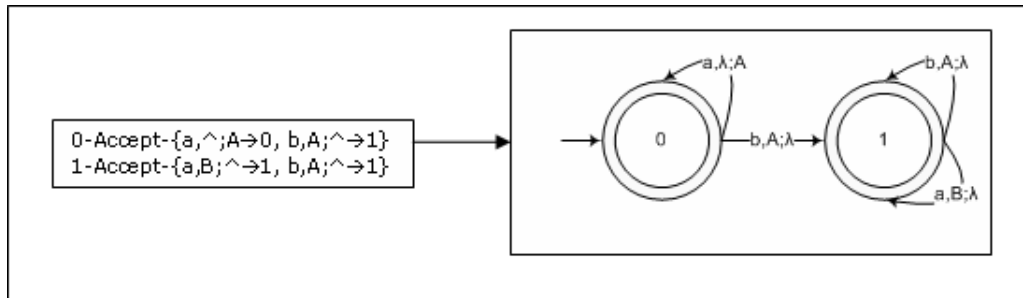


Figure 7.60: Example Solution for L1 (PDA)

L2 ($a^n c b^n$ where $n \geq 0$): Table C.3 and Table C.4 outline the results for L2. Solutions were induced for all seed values when applying both the standard genetic operators and non-destructive operators. Figure 7.61 presents an example solution for L2.

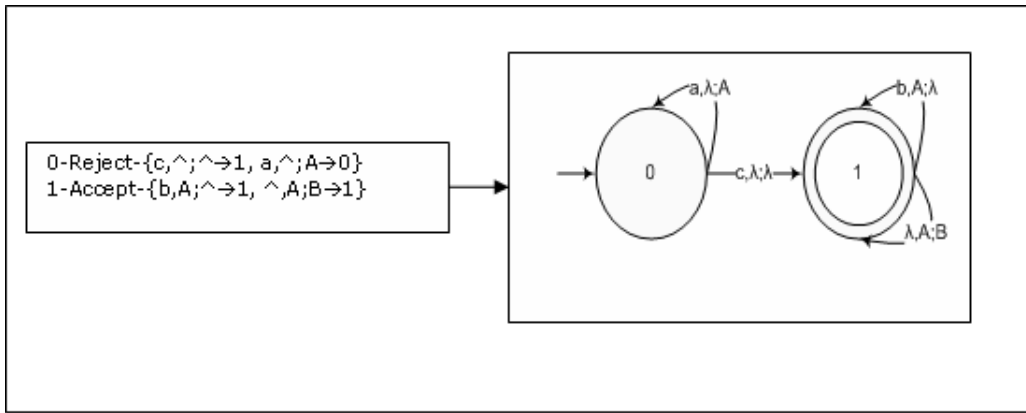


Figure 7.61: Example Solution for L2 (PDA)

L3 (All palindromes with an odd number of letters. $\Sigma=\{a,b\}$): Table C.5 and Table C.6 outline the results obtained for L3. Applying the standard genetic operators yielded solutions for four of the ten seed values whereas applying non-destructive crossover and non-destructive mutation resulted in solutions for all seed values. Figure 7.62 presents an example solution for L3.

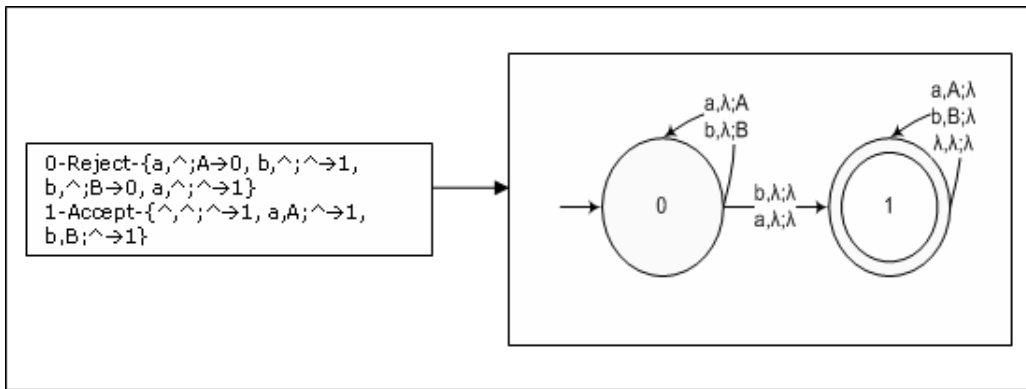


Figure 7.62: Example Solution for L3 (PDA)

L4 (ss^r where s is in $(a + b)^*$): Solutions were induced for six of the ten seed values using the standard genetic operators and for nine of the ten seed values using non-destructive crossover and mutation. Table C.7 and Table C.8 outline the results obtained for L4. Figure 7.63 presents an example solution for L4.

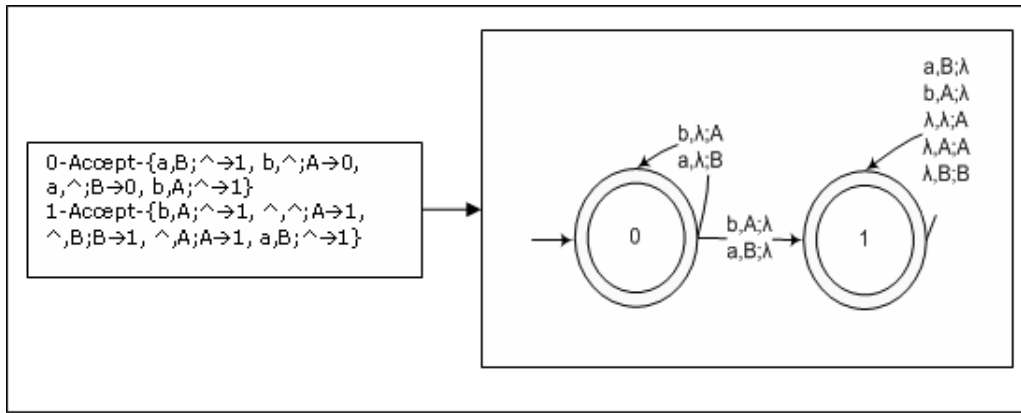


Figure 7.63: Example Solution for L4 (PDA)

L5(scs^r where s is in $(a + b)^*$): The results for L5 are outlined in Table C.9 and Table C.10. Solutions were induced for five of the ten seed values for the standard genetic operators and all seed values using non-destructive crossover and non-destructive mutation. Figure 7.64 presents an example solution for L5.

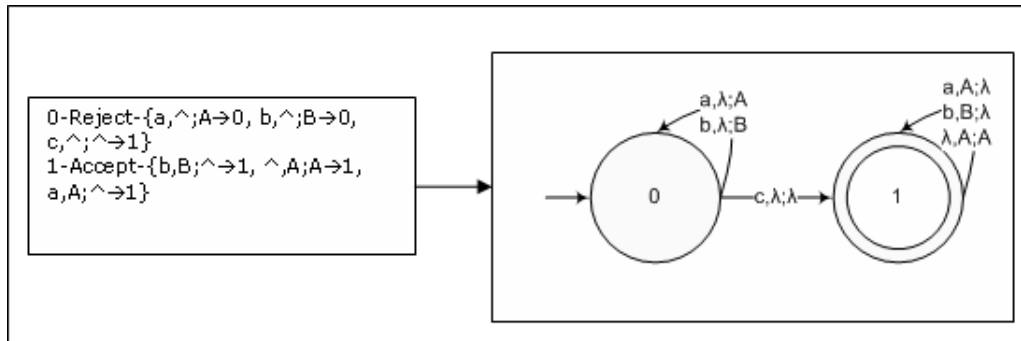


Figure 7.64: Example Solution for L5 (PDA)

L6(s is in $(a + b)^*$: the number of a's in s is equal to the number of b's in s): Solutions were found for all seed values using the standard genetic and non-destructive operators. Table C.11 and Table C.12 show the results obtained for L6. Figure 7.65 presents an example solution for L6.

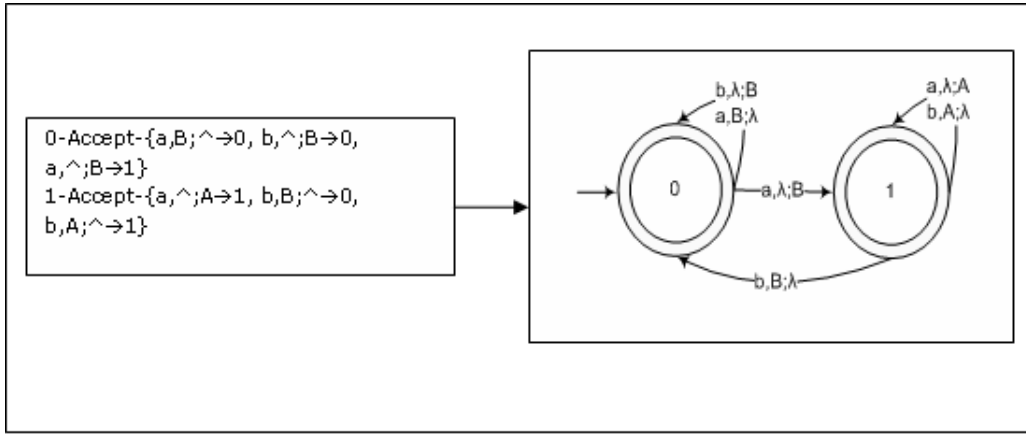


Figure 7.65: Example Solution for L6 (PDA)

L7($a^n b^{2n}$ where $n \geq 0$): Solutions were found for all seed values using the standard genetic operators and non-destructive operators. Table C.13 and Table C.14 show the results obtained for L7. Figure 7.66 presents an example solution for L7.

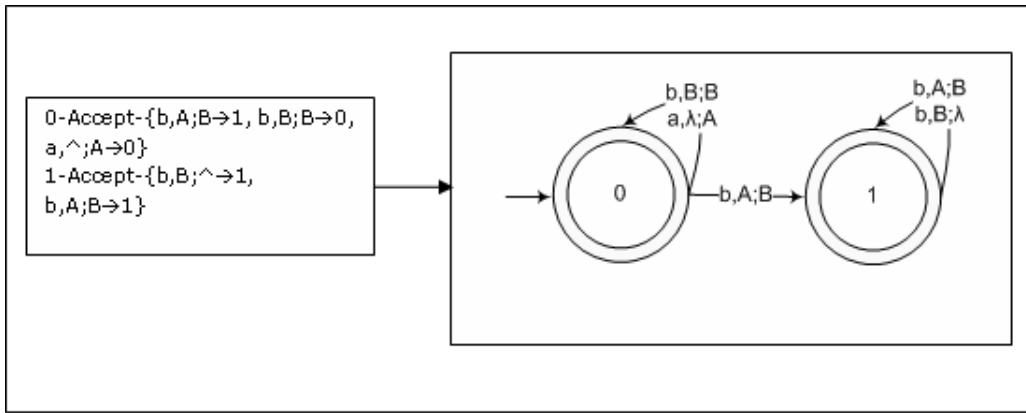


Figure 7.66: Example Solution for L7 (PDA)

L8($a^n b^n$: $n \geq 0 \cup \{a\}$): Table C.15 and Table C.16 show the results obtained for L8. Solutions were obtained for all seed values using the standard genetic operators and non-destructive operators. Figure 7.67 presents an example solution for L8.

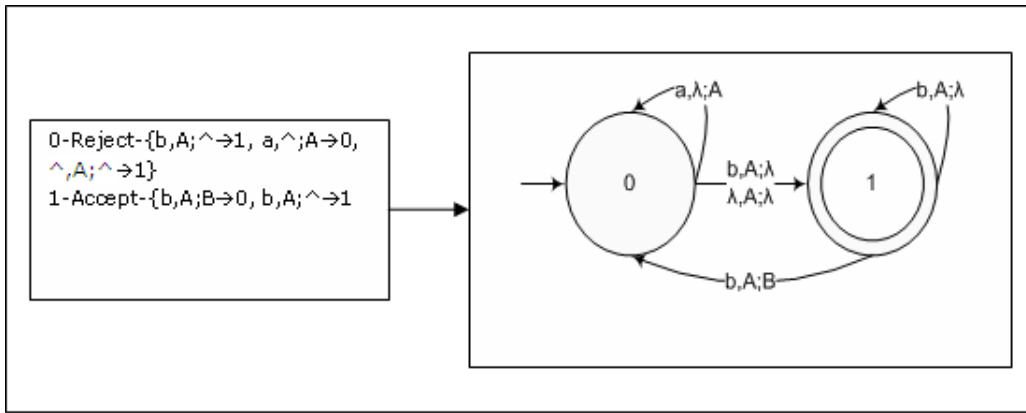


Figure 7.67: Example Solution for L8 (PDA)

L9(aa^*ba^*): Solutions were generated for all seed values using both the standard genetic operators and non-destructive operators. Table C.17 and Table C.18 outline the results obtained for L9. Figure 7.68 presents an example solution for L9.

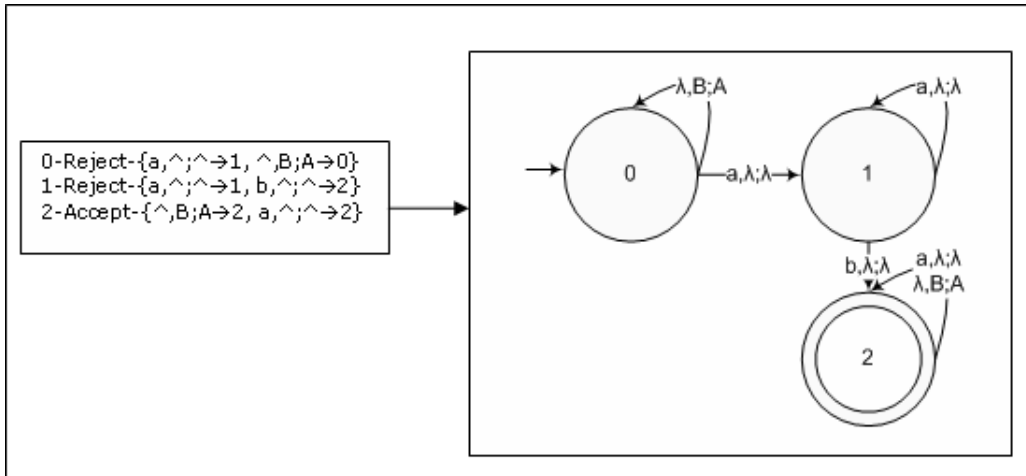


Figure 7.68: Example Solution for L9 (PDA)

L10($a^n b^{2n}$ where $n \geq 1$): Table C.19 and Table C.20 present the results obtained for L10. Solutions were induced for all seed values using the standard genetic operators and non-destructive operators. Figure 7.69 presents an example solution for L10.

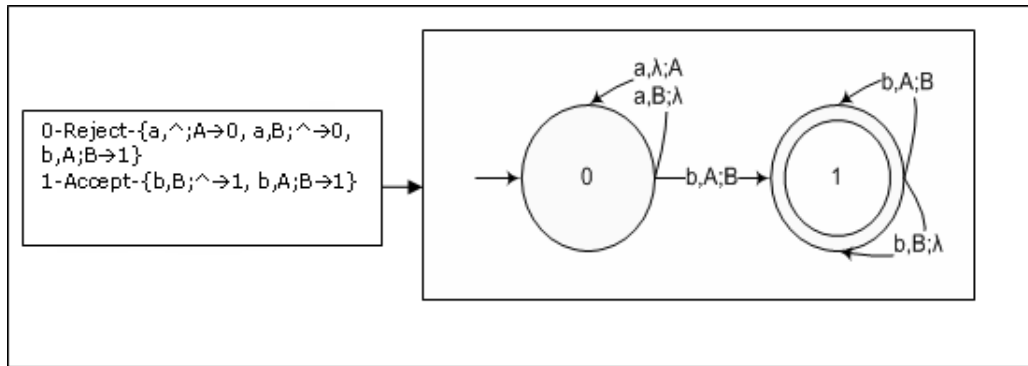


Figure 7.69: Example Solution for L10 (PDA)

L11(All strings with balanced brackets. $\Sigma = \{ (,) \}$): Solutions were generated for all seed values during initial generation of the population. Non-destructive operators are therefore not used for this language. In effect, solutions for L11 are found using a ‘random search’ of the search space of the problem. Figure 7.70 presents an example solution for L11.

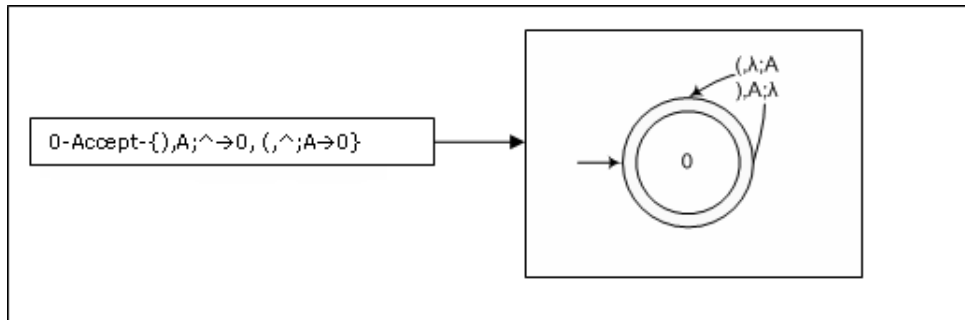


Figure 7.70: Example Solution for L11 (PDA)

The system successfully evolved PDAs for all languages. The system was able to generate both NPDAs and DPDAs for all languages except L3, L4 and L6. L3, L4 and L6 can only be recognised by NPDAs. A single run for a language took about 3–5 minutes to complete for all languages except L3, L4 and L6, which ranged from 15–20 minutes. It is noticeable that the languages that require NPDAs need a longer runtime indicating that the non-determinism leads to an increase in the search space, therefore making the search more difficult.

Table 7.19 lists the success rates for each language. The success rate is calculated as a percentage of the seed values that produced solutions over 20 iterations for L3 and ten iterations for the other ten languages.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 100% | 100% | 100% | 100% |
| L2 | 100% | 100% | 100% | 100% |
| L3 | 40% | 100% | 100% | 80% |
| L4 | 60% | 80% | 70% | 90% |
| L5 | 50% | 100% | 100% | 90% |
| L6 | 100% | 100% | 100% | 100% |
| L7 | 100% | 100% | 100% | 100% |
| L8 | 100% | 100% | 100% | 100% |
| L9 | 100% | 100% | 100% | 100% |
| L10 | 100% | 100% | 100% | 100% |
| L11 | 100% | 100% | 100% | 100% |

Table 7.19: Success Rates for PDA Simulations

Note that the only languages that did not achieve a 100% success rate were L3, L4 and L5. These three languages are the palindrome languages i.e. odd palindrome, ss^r and scs^r . Table 7.20 gives the percentage of runs requiring more than one iteration. It is clear that selection variance resulted in premature convergence for L3, L4, L5 and L6 and the use of multiple iterations per run was needed to escape local optima for this domain.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 10% | 0% | 0% | 0% |
| L2 | 10% | 0% | 0% | 0% |
| L3 | 100% | 80% | 80% | 100% |
| L4 | 90% | 70% | 80% | 90% |
| L5 | 100% | 70% | 60% | 70% |
| L6 | 70% | 30% | 60% | 60% |
| L7 | 30% | 0% | 10% | 10% |
| L8 | 40% | 0% | 0% | 20% |
| L9 | 0% | 0% | 0% | 0% |
| L10 | 80% | 0% | 20% | 10% |
| L11 | 0% | 0% | 0% | 0% |

Table 7.20: Multiple iterations required for PDA Simulations

Table 7.21 shows the GP generated solution compared to a ‘‘Human Generated’’ solution for each language. The solutions generated compare well to human generated solutions however the system generated solutions usually contain redundant transitions. This is consistent with studies performed by Huijsen [22] and Zomorodian [62]. For example, the solution for L4 presented in Figure 7.63, has the transition $\{\lambda, \lambda, A\}$ which is unnecessary and is never traversed. The transition $\{\lambda, A, A\}$ is also unnecessary as this transition just pops and pushes an ‘A’ onto the stack thereby increasing the runtime of the system.

| | "Human Generated" | GP Generated |
|----|-------------------|--------------|
| L1 | | |
| L2 | | |
| L3 | | |
| L4 | | |
| L5 | | |
| L6 | | |

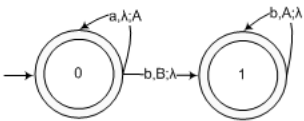
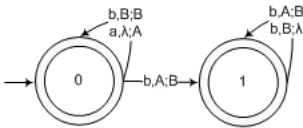
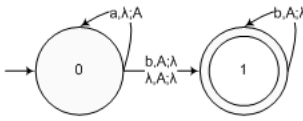
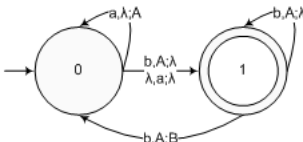
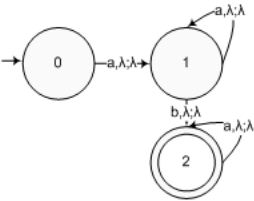
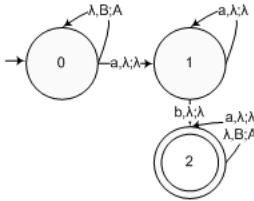
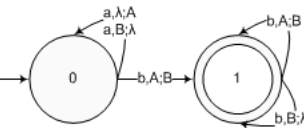
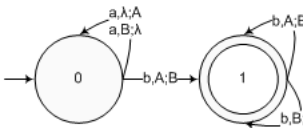
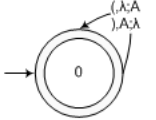
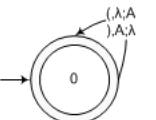
| | "Human Generated" | GP Generated |
|-----|---|--|
| L7 |  |  |
| L8 |  |  |
| L9 |  |  |
| L10 |  |  |
| L11 |  |  |

Table 7.21: Comparison with "Human Generated" Solutions (PDAs)

Comparing these results obtained to the previous work done by Lankhorst [28] Table 7.22 shows the average classification⁵ for the three languages that are common to both systems. Lankhorst uses 100 positive sentences and 100 negative sentences to train each language, whereas the GP system required 51 sentences (including the empty string) for the 'Balanced Brackets' language and more than 200 sentences for the other two languages. This indicates that a larger number of fitness cases is needed to produce solutions that generalise well.

⁵The average classification for the GP system is calculated by taking the average of the best fitness over the 10

| Language | Lankhorst | | GP System | |
|-----------------------------|-----------|---------|--------------------|-----------------|
| | Binary | Integer | Standard Operators | Non-destructive |
| Equal number of a's and b's | 97% | 95.5% | 100% | 100% |
| Balanced Brackets | 100% | 97.5% | 100% | 100% |
| Even-length Palindrome | 92% | 88% | 98.6% | 99% |

Table 7.22: Comparison with Lankhorst's work

Table 7.23 shows a comparison between the work done by Dunay [13] and the GP system presented in this section. Dunay presents his results as a percentage of the 'induced machines that correctly classifies the target language'.

| Language | Dunay | GP System | |
|-----------------------------|-------|--------------------|-----------------|
| | | Standard Operators | Non-destructive |
| Equal number of a's and b's | 81% | 100% | 100% |
| Balanced Brackets | 100% | 100% | 100% |
| $a^n b^n$ | 60% | 100% | 100% |

Table 7.23: Comparison with Dunay's work

The GP system performs just as well for the balanced brackets language and outperforms the other methods for the other languages. Huijsen [22] tested his system on three languages i.e. balanced brackets, equal number of a's and b's and ww^r . The results are presented as the average generation on which a solution was obtained. For the ww^r , two optimal NPDAs were found. For the system presented in this thesis, solutions were found for nine of the ten runs performed when applying non-destructive operators for ww^r (L4). Unfortunately, the success rates for L6 (Equal number of a's and b's) and L10 (Balanced brackets) are not specified in Huijsen[22] and hence a comparison is not possible.

For all languages the GP system presented in this thesis for PDAs performed comparably, and in some cases outperformed the other methods implemented.

iterations for each seed value.

7.4 Turing Machines

7.4.1 Turing Machine Acceptors

The system was applied using ten seed values for each language in the language set. The ten random seed values used are 1500, 2000, 3500, 3900, 4020, 4850, 5249, 6300, 8400 and 9077. The program was tested using both the standard genetic operators and non-destructive operators. The non-destructive operators are implemented by ensuring that the fitness of the child is better than or equal to the fitness of the parent when applying crossover and mutation. Non-destructive operators are applied to avoid the destructive effects of crossover and mutation. If no solutions were found for a particular seed value multiple iterations were performed for this seed to deal with selection variance. The system was also tested using an incremental fitness method where the fitness cases are ordered by length of the input string. When evaluating an individual the system starts off with the shortest fitness cases first, and only once an individual satisfies this will it be tested on fitness cases of larger length. A string is accepted by the Turing machine if a path leads to the HALT state and is rejected otherwise. All the tables referred to in this section can be found in Appendix D.

L1 ($a^n b^n$): No solutions were found when employing the standard genetic operators. Solutions were found for nine seed values using non-destructive crossover and mutation. Using incremental fitness the results were slightly better for the standard operators. The results using the standard fitness are presented in Table D.1 and Table D.2 and the results using incremental fitness are presented in Table D.3 and Table D.4. Figure 7.71 presents an example solution for L1.

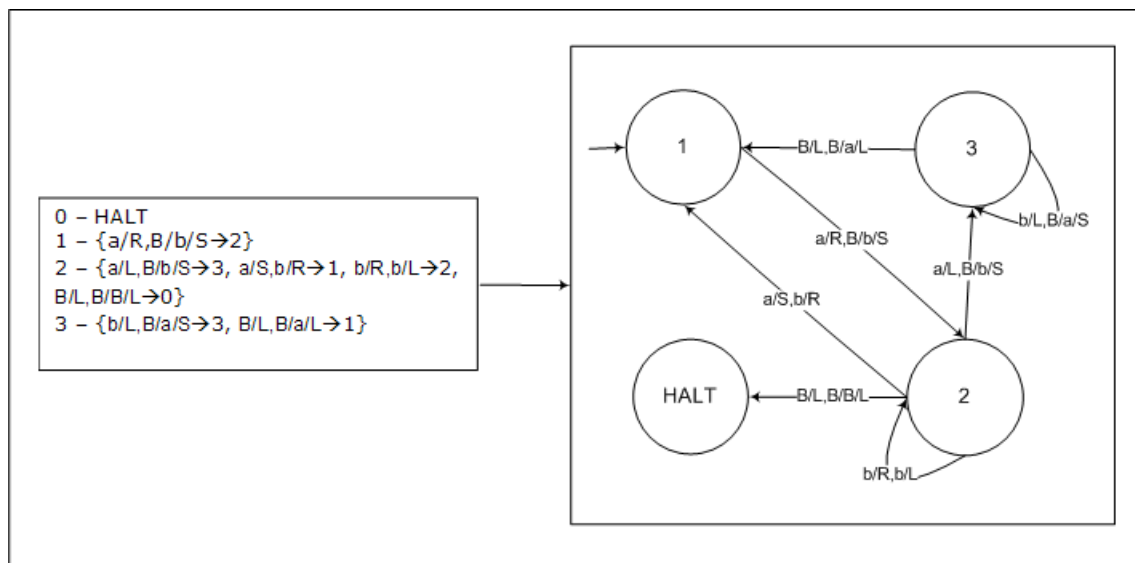


Figure 7.71: Example Solution for L1 (TMA)

L2 ($a^n b^n c^n$): No solutions were found for L2. The results for L2 are shown in Table D.5 and Table D.6. Only one solution is induced using the incremental fitness method for non-destructive crossover and

mutation one solution is induced using non-destructive crossover. The results are shown in Table D.6 Figure 7.72 presents an example solution for L2 obtained using the incremental method.

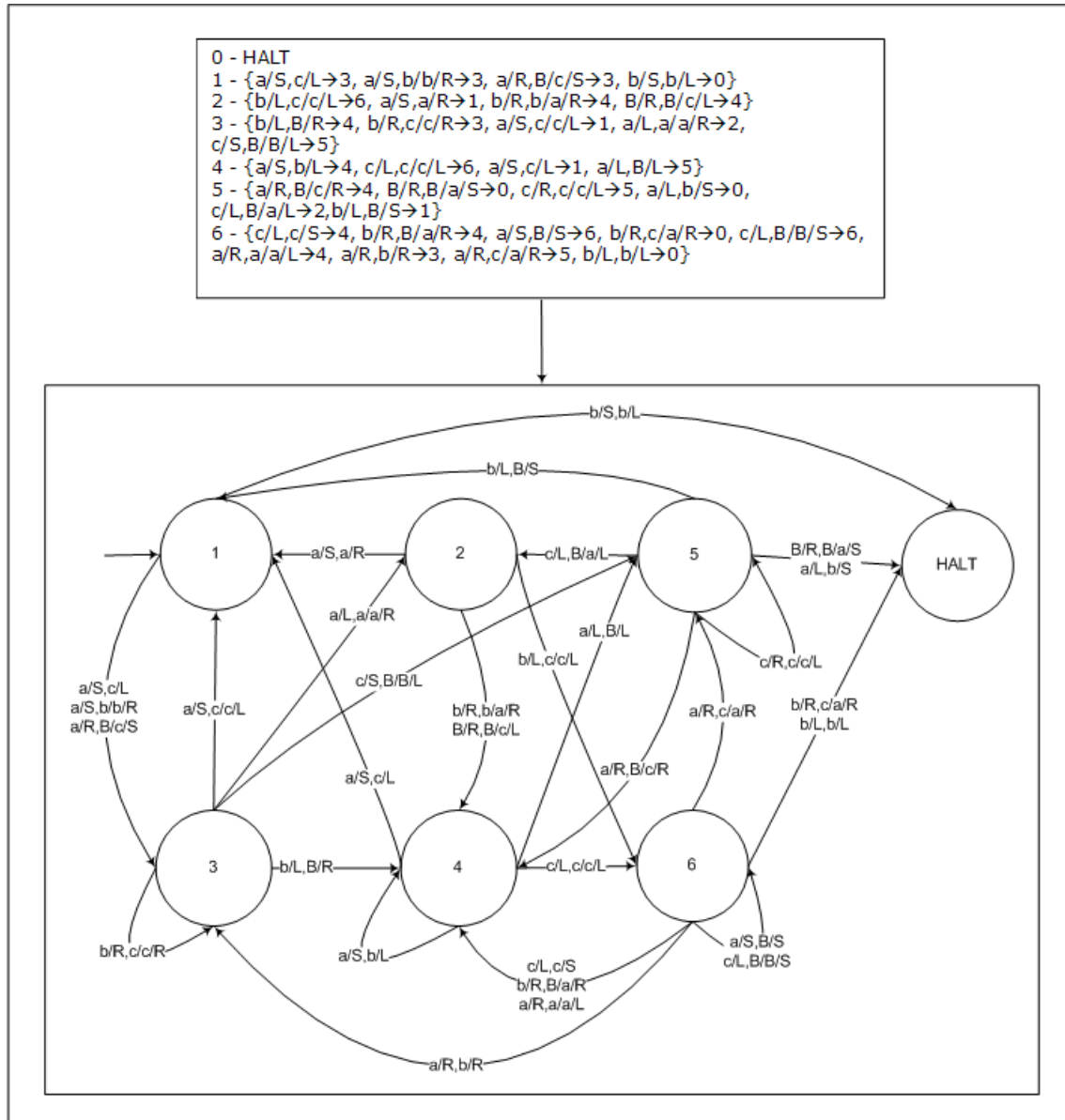


Figure 7.72: Example Solution for L2 (TMA)

The solution generated for L2 has a few redundant nodes and unnecessary transitions. State 2 and state 6 can never be reached. Some of the transitions e.g., the transition a/S,b/b/R leaving state 1 will never be traversed as 'b' is never written to the output tape. Figure 7.73 shows a minimized solution for L2.

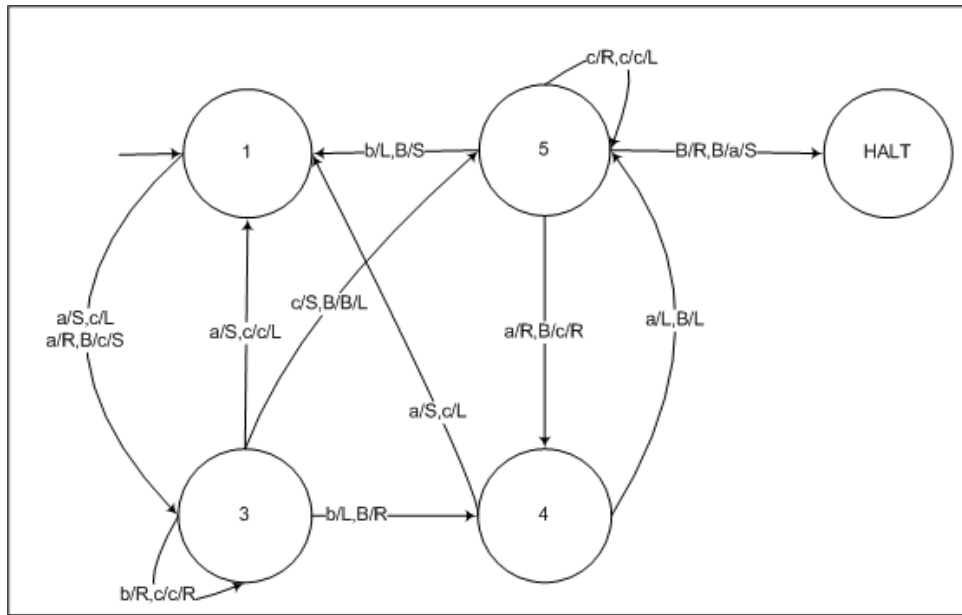


Figure 7.73: Minimized Solution for L2 (TMA)

L3 (aba*b): Solutions were induced for all seed values using either the standard genetic programming operators or non-destructive operators. Table D.8 and Table D.9 show the results obtained for L3. The incremental method didn't improve the results for L3. Figure 7.74 presents an example solution for L3.

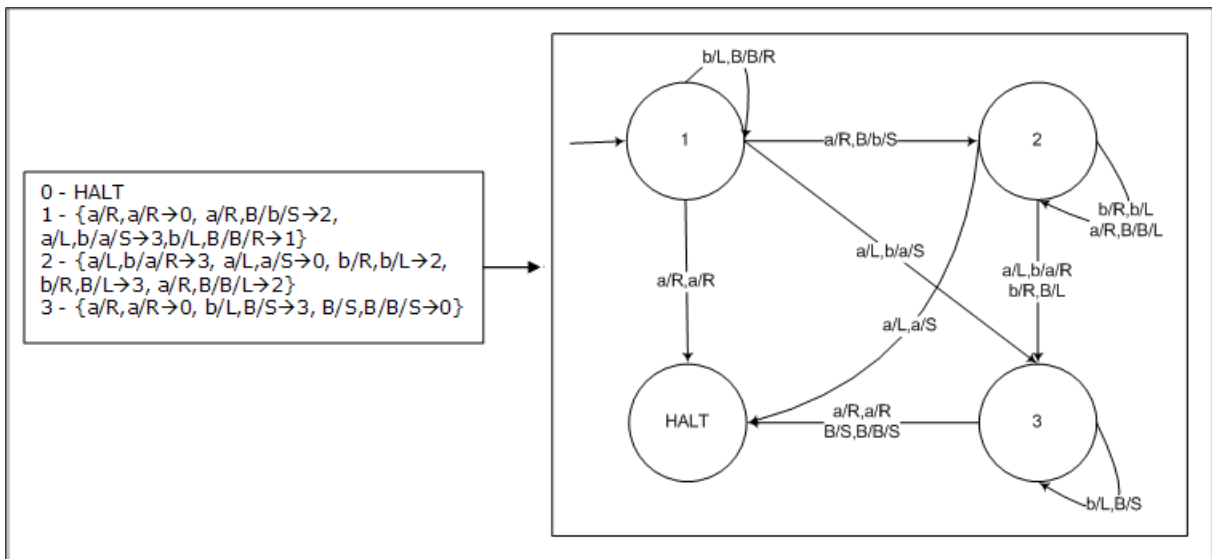


Figure 7.74: Example Solution for L3 (TMA)

L4 ($a^n b^{2^n}$): No solutions were found using the standard genetic operators and four seed values produced solutions using non-destructive operators. The results for L4 are presented in Table D.10 and Table D.11. Using the incremental method L4 performs much better with seven of the seed values producing solutions when using non-destructive crossover and mutation. The results for the incremental approach are shown in Table D.12 and Table D.13. Figure 7.75 shows an example solution.

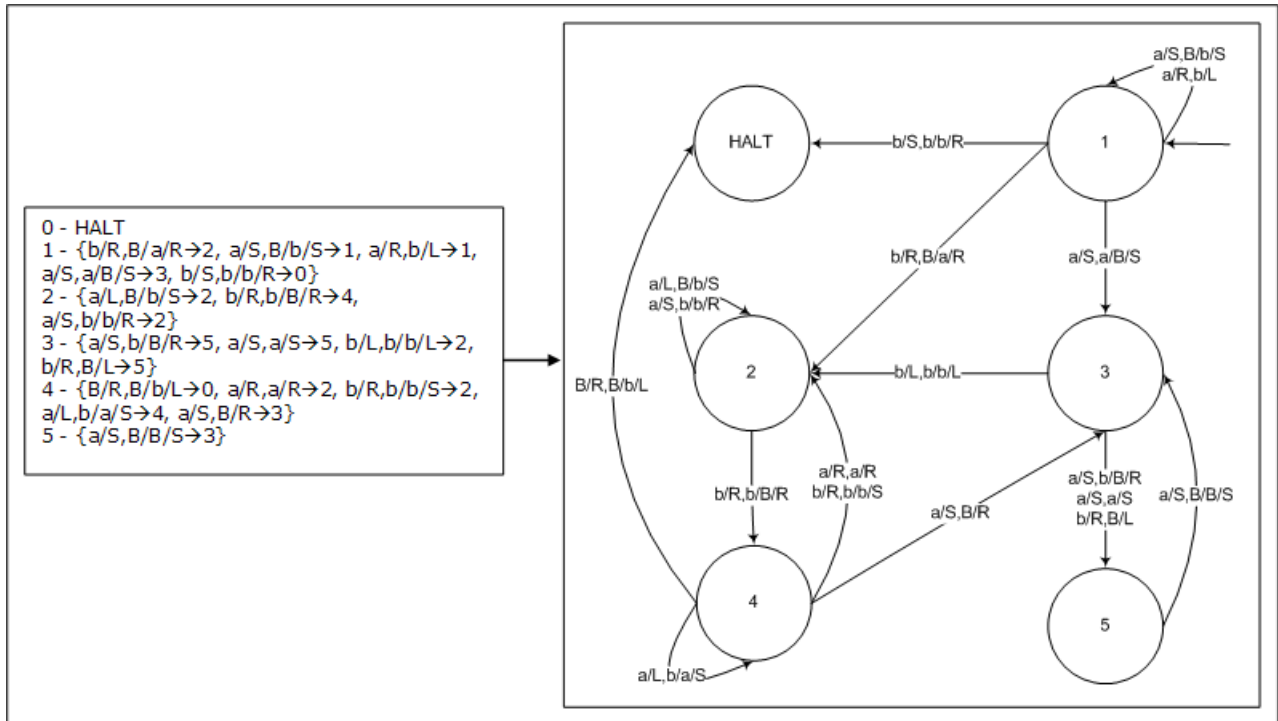


Figure 7.75: Example Solution for L4 (TMA)

L5 ($w|w \in \{a, b\}^*$ where w has even length): Solutions were induced for all seed values using either the standard genetic operators or non-destructive operators. Table D.14 and Table D.15 present the results for L5. The incremental method doesn't improve on the results obtained for L5. Figure 7.76 presents an example solution for L5.

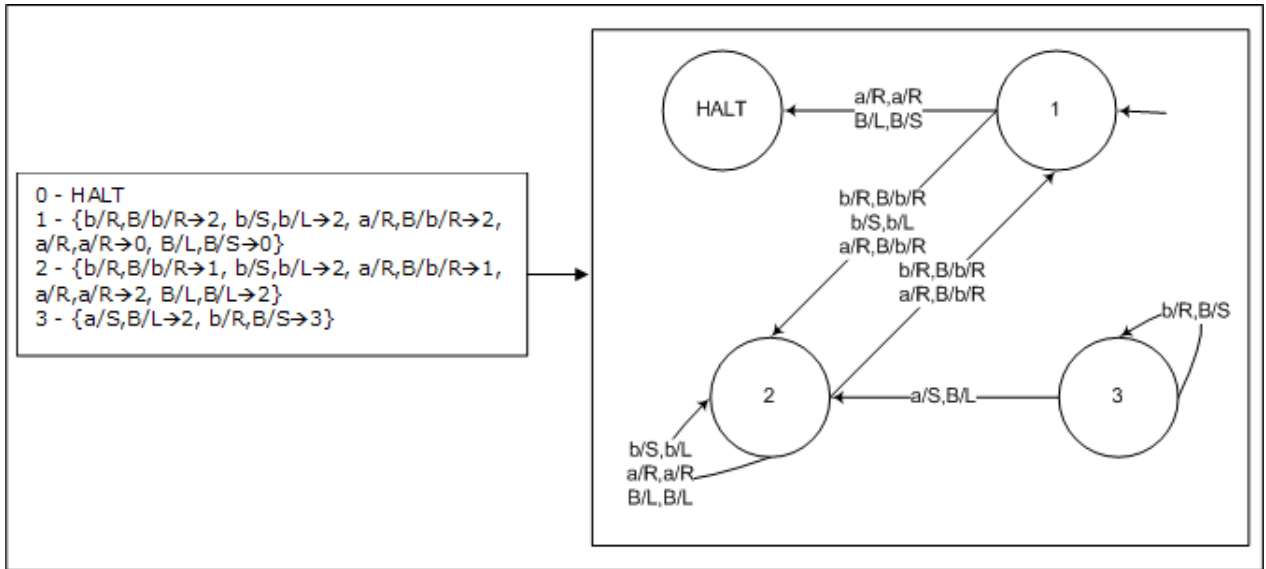


Figure 7.76: Example Solution for L5 (TMA)

L6 ($w : w \in \{a, b\}^+$ and the number of a's in w is not equal to the number of b's): Only one solution for L6 was found which is shown in Figure 7.77. This solution was obtained using non-destructive crossover and mutation on the initial iteration. The incremental method didn't improve the results obtained for L6. Table D.16 and Table D.17 show the results for L6.

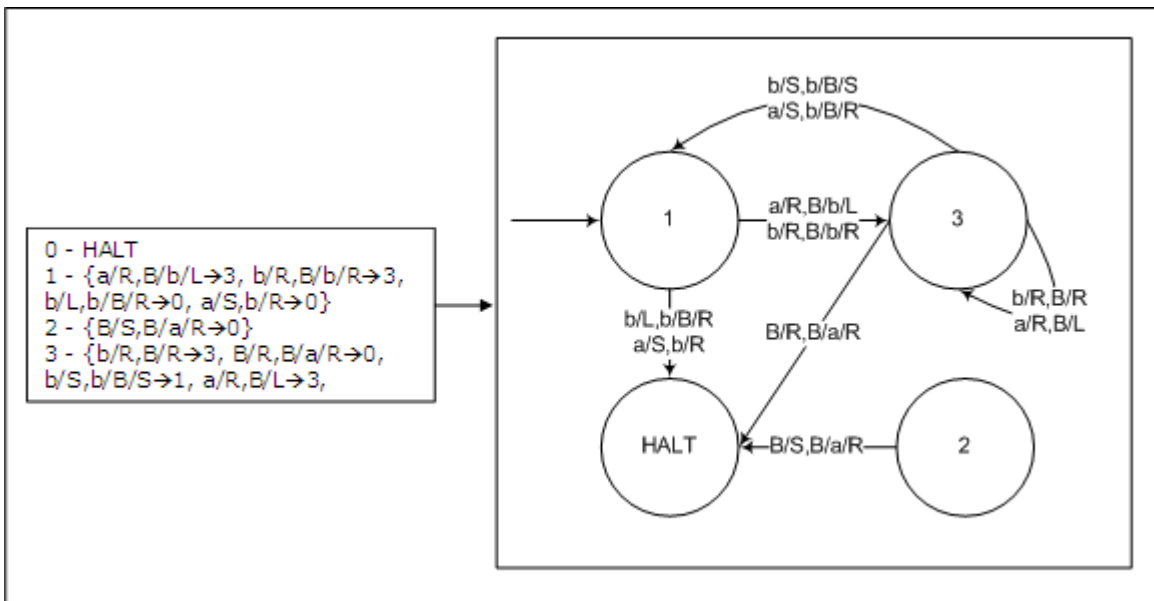


Figure 7.77: Example Solution for L6 (TMA)

L7 ($a^n b^{n+1}$): No solutions were found for L7 using the standard genetic operators. Solutions were

found for all seed values using non-destructive crossover and mutation. The results are shown in Table D.18 and Table D.19. The incremental method doesn't improve the results of L7. Figure 7.78 shows an example solution. Node 5 has been removed from the transition diagram as it can never be reached.

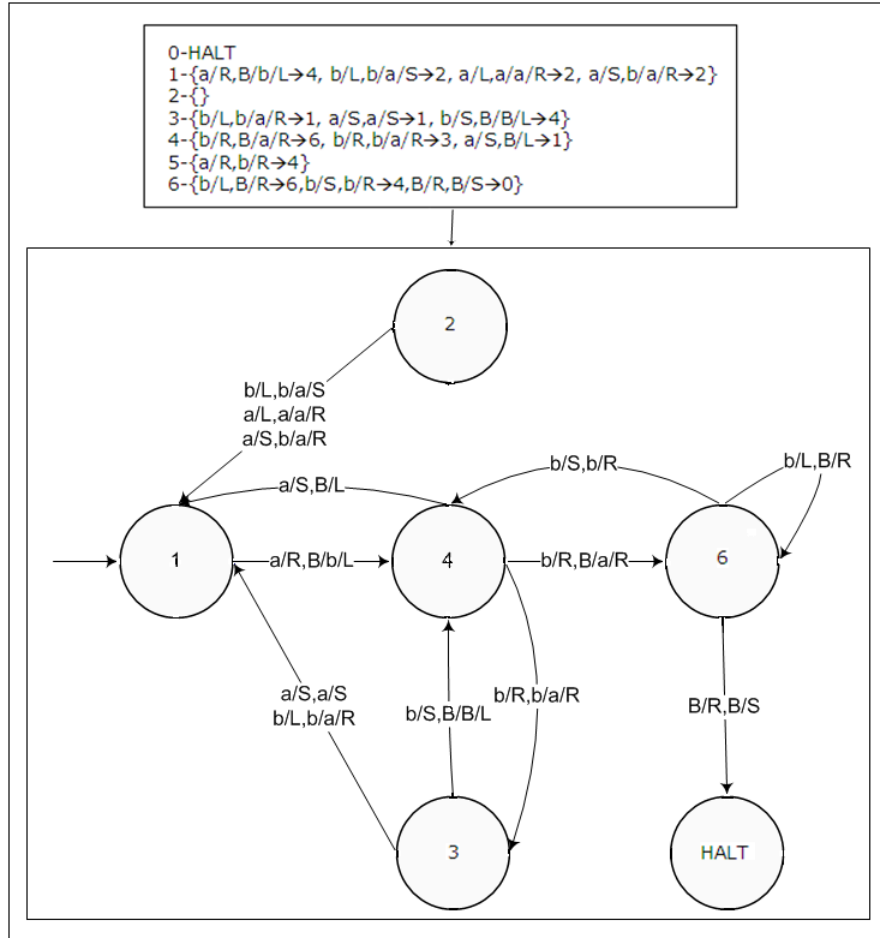


Figure 7.78: Example Solution for L7 (TMA)

L8 (awb where $w \in \{a, b\}^*$): Seven of the ten seed values produced solutions for L8 using the standard genetic operators. All seed values produced solutions using non-destructive operators. Table D.20 and Table D.21 shows the results obtained for L8. The incremental method doesn't improve the results for L8. Figure 7.79 shows an example solution.

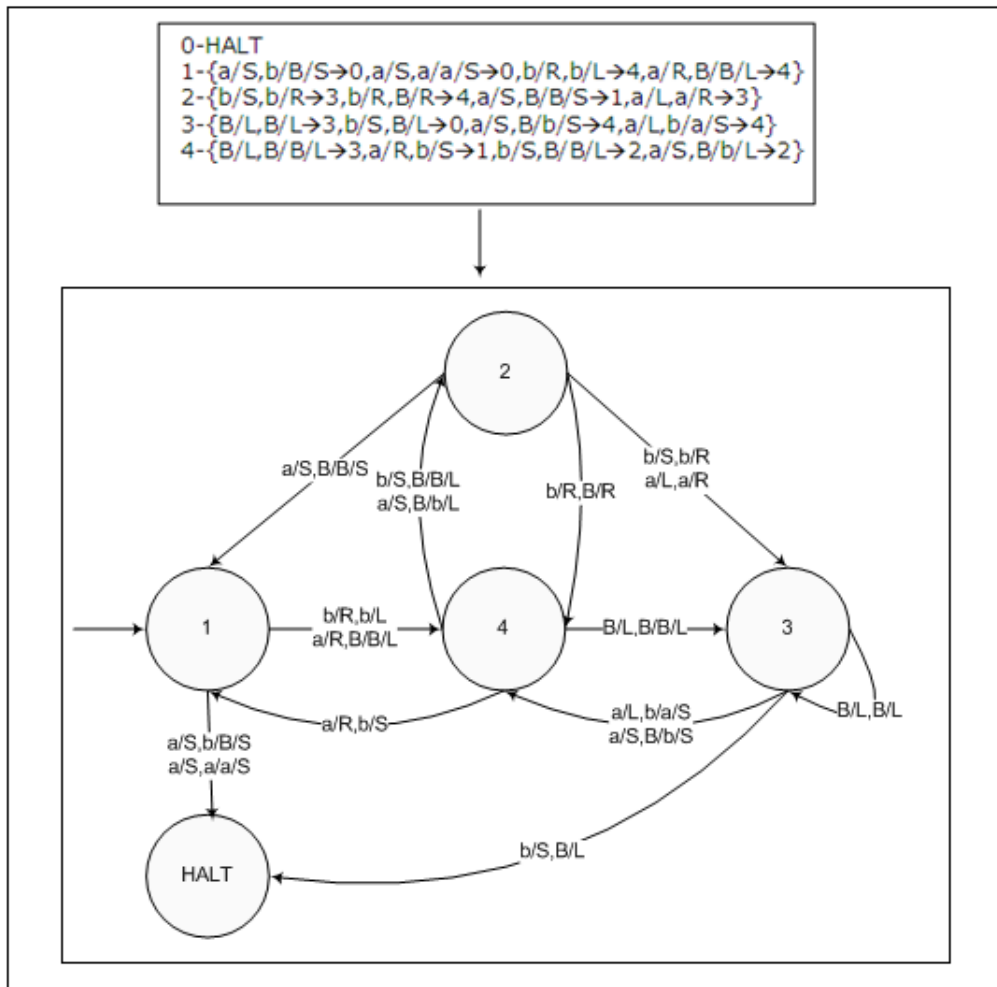


Figure 7.79: Example Solution for L8 (TMA)

L9 ($a^n b^n a^n$): Only one solution was found for L9 using non-destructive crossover and mutation. The results are shown in Table D.22 and Table D.23. The incremental method doesn't improve the results for L9. The output from the system is shown in Figure 7.80.

```

0-HALT
1-{a/R,B/b/L→3,b/R,b/L→4,a/R,a/b/L→0,a/S,b/b/S→7}
2-{a/R,a/B/R→5,B/S,B/S→6,a/S,B/R→2,a/R,b/S→3}
3-{a/L,B/b/R→1,b/R,b/b/R→8,b/R,B/a/L→5}
4-{b/L,B/B/L→7,a/R,b/L→4,a/R,a/b/S→2,a/R,B/R→7,B/S,B/a/R→3}
5-{a/S,a/a/R→1,b/L,b/a/S→3,B/S,B/R→8,a/R,B/B/S→6,a/L,b/S→4}
6-{b/L,B/b/L→10,B/L,B/a/R→0,b/S,b/b/L→11}
7-{a/R,a/B/R→7,B/S,B/R→1,a/S,B/R→13,a/R,b/S→10}
8-{a/L,B/a/R→8,b/R,b/b/R→7,b/R,B/a/L→8}
9-{a/S,b/B/S→8,B/L,B/S→9,a/S,a/B/R→9,a/S,B/a/S→8}
10-{a/R,b/R→11,b/L,b/b/R→9,a/R,B/S→7,a/R,a/b/R→12}
11-{a/S,b/R→7,a/R,B/B/R→11,a/S,a/B/L→8,b/R,B/B/L→8,b/S,b/L→13}
12-{a/S,b/S→11,B/L,B/B/S→11,a/S,a/a/R→11,a/S,B/a/S→12}
13-{a/R,b/R→5,b/S,b/B/L→8,a/R,B/B/S→5,a/L,a/b/R→7}

```

Figure 7.80: Example Solution for L9 (TMA)

This produces a complicated transition diagram that is not very intuitive. Removing the unnecessary nodes and transitions for L9 produces the diagram shown in Figure 7.81.

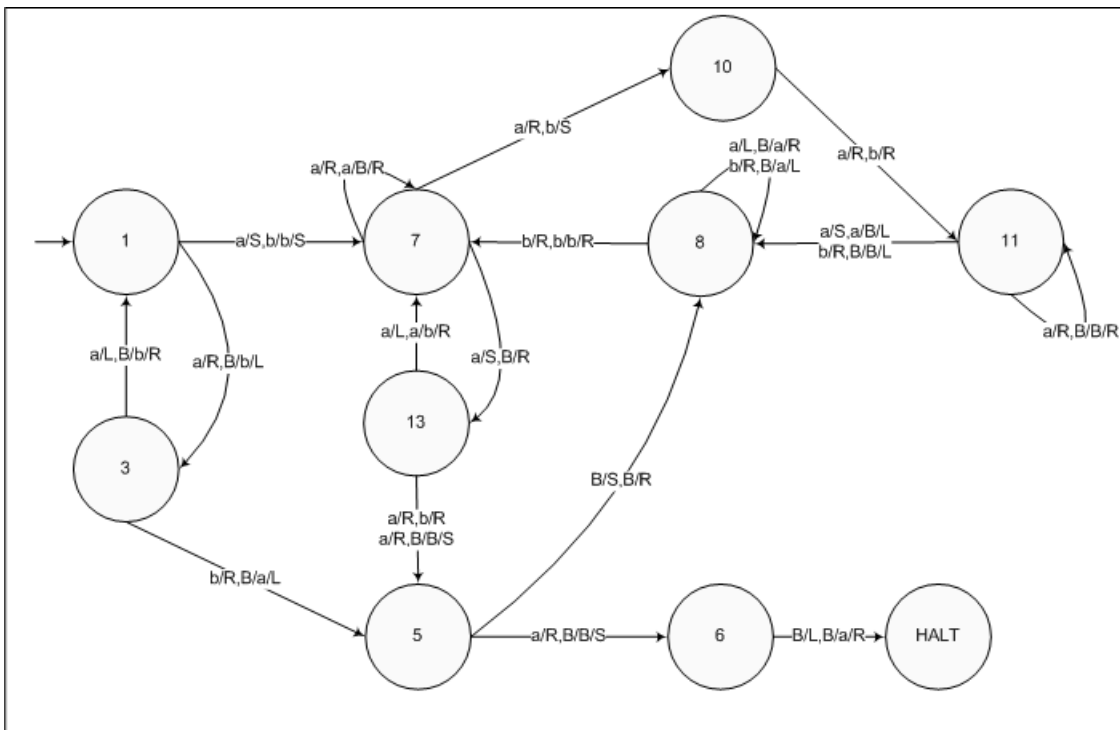


Figure 7.81: Minimized Solution for L9 (TMA)

The system successfully evolved TMAs for all languages. A single run, with one iteration, took between 30 minutes to an hour. This is due to the complexity involved in interpreting each Turing machine

individual. Table 7.24 lists the success rates for each language. The success rate is calculated as a percentage of the seed values that produced solutions over ten iterations for the languages. The system was able to produce a good success rate for L1, L3, L4 (when using the incremental approach), L5, L7 and L8. L2, L6 and L9 have much lower success rates in comparison to the other languages. These languages have a large search space which could be the reason for the poor results. L2 is a three character language which is also a contributing factor to the large search space. For L2 and L9, insufficient positive⁶examples in the fitness cases could also be a contributing factor to the poor results. When analysing L2 and L9, it is noticeable that the best individuals for all runs fail to correctly classify some of the positive fitness cases whereas all the negative fitness cases are correctly classified.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 0% | 70% | 80% | 90% |
| L2 | 0% | 0% | 0% | 0% |
| L3 | 100% | 100% | 100% | 100% |
| L4 | 0% | 10% | 20% | 20% |
| L5 | 100% | 100% | 100% | 100% |
| L6 | 0% | 0% | 0% | 10% |
| L7 | 0% | 40% | 80% | 90% |
| L8 | 70% | 100% | 100% | 100% |
| L9 | 0% | 0% | 0% | 10% |

Table 7.24: Success Rates for TMA Simulations

Table 7.25 shows the success rates when applying an incremental approach for L1, L2 and L4. All languages show an improvement in the success rates.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 20% | 70% | 60% | 100% |
| L2 | 0% | 0% | 10% | 10% |
| L4 | 10% | 60% | 30% | 70% |

Table 7.25: Success Rates for TMA Simulations – Incremental Approach

Table 7.26 gives the percentage of runs requiring more than one iteration. All languages required multiple iterations.

⁶Positive fitness cases refer to the strings accepted by the Turing machine e.g. for L2 ‘abc’, ‘aabbcc’, ‘aaaaabbbbcccc’ are positive fitness cases. Negative fitness cases are the strings rejected by the Turing machine.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 100% | 60% | 80% | 70% |
| L2 | 100% | 100% | 100% | 100% |
| L3 | 60% | 10% | 40% | 30% |
| L4 | 100% | 100% | 90% | 90% |
| L5 | 60% | 40% | 40% | 20% |
| L6 | 100% | 100% | 100% | 90% |
| L7 | 100% | 80% | 80% | 60% |
| L8 | 80% | 30% | 10% | 20% |
| L9 | 100% | 100% | 100% | 90% |

Table 7.26: Multiple iterations required for TMA Simulations

Table 7.27 shows the “Human generated” solutions for TMAs. One of the difficulties with the solutions generated by the GP system is the unnecessary nodes and transitions called ‘structural redundancies’. All of the solutions generated by the GP system have these redundancies. These redundancies increase the complexity of the solutions generated. For example, the example presented for L5 in Figure 7.76, performs some redundant steps, in rejecting ‘bab’. Some of the solutions generated by the GP system have structures that are quite different to the human generated solutions. The solution generated by the GP system for L3 (aba*b) has four nodes, as it makes use of the second tape whereas the “Human generated” solution has five nodes as using the second tape isn’t necessary for this language. The solution generated by the system for L3, on the other hand, makes use of the second tape by initially writing a ‘b’ to the tape when the first ‘a’ is read. This then allows for a single state to be used for the ‘ba*’ portion of language. The transition ‘b/R,b/L’ is followed as a result when the ‘b’ is read, and thereafter the ‘a/R,B/B/L’ transition is followed for each ‘a’ character read.

| Language | Human Generated Solutions |
|----------|---------------------------|
| L1 | |
| L2 | |
| L3 | |
| L4 | |
| L5 | |
| L6 | |

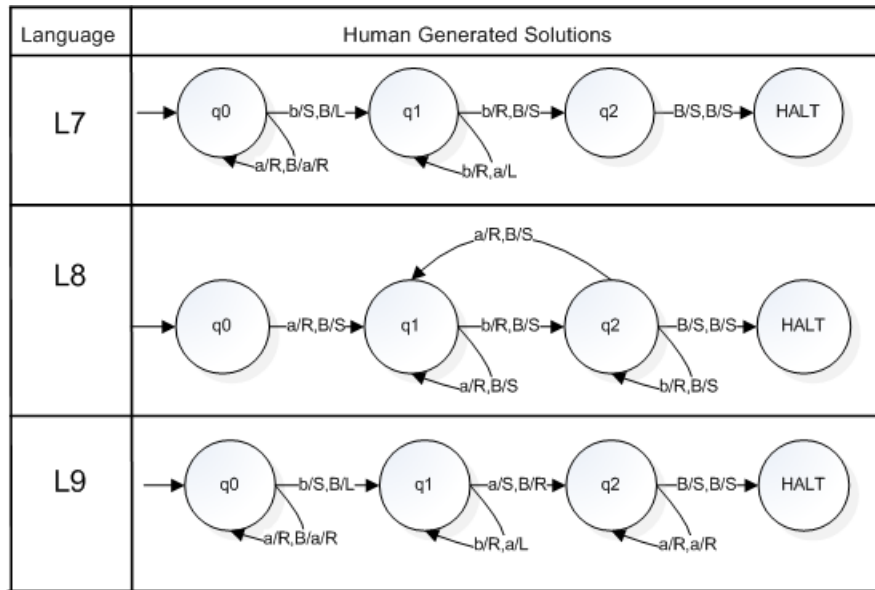


Table 7.27: “Human Generated” Solutions (TMAs)

Comparing these results to the work done by Tanomaru [45], Table 7.28 shows a comparison of the success rates of the three languages common to both systems. The GP system outperformed Tanomaru’s system for two of the languages. Tanomaru’s system with population shifting produced better results for the language $a^n b^n a^n$. Note that the number of runs performed per problem in the Tanomaru study is a hundred while ten runs were performed in this study.

| Language | Tanomaru | | GP System |
|----------------------------|---------------------------|--------------|-----------|
| | Experiment 1 ⁷ | Experiment 2 | |
| $a^n b^n$ | 9% | 82% | 90% |
| awb where $w \in a, b^*$ | 41% | 62% | 100% |
| $a^n b^n a^n$ | 1% | 38% | 10% |

Table 7.28: Comparison with Tanomaru’s work

⁷Experiment 1 is performed without population shifting and Experiment 2 is performed with population shifting

7.4.2 Turing Machine Transducers

The system was applied using ten seed values for each language in the language set. The ten random seed values used are 1500, 2000, 3500, 3900, 4020, 4850, 5249, 6300, 8400 and 9077. The seed values were chosen randomly. The program was tested using both the standard genetic operators and non-destructive operators. The non-destructive operators are implemented by ensuring that the fitness of the child is better than or equal to the fitness of the parent when applying crossover and mutation. Non-destructive operators are applied to avoid the destructive effects of crossover and mutation. If no solutions were found for a particular seed value multiple iterations for this seed were performed to deal with selection variance. The TM does not need to end in a HALT state to generate a solution. The TM may halt as a result of not being able to find a transition with the required combination of tape characters.

L1 (Unary Addition where the input consists of a string of unary integers separated by a 0): The results for L1 produced by TMT1⁸ are outlined in Table D.24 and Table D.25. The results produced by TMT2 are outlined in Table D.36 and Table D.37. Solutions were induced for all seed values using non-destructive operators for TMT1. Solutions were induced for all seed values using non-destructive crossover and mutation. Figure 7.82 and Figure 7.83 show example solutions for L1 for TMT1 and TMT2 respectively.

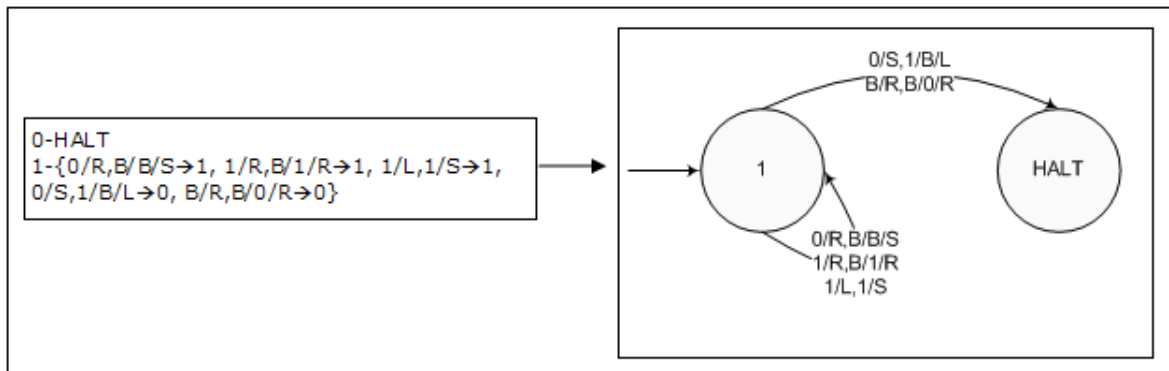


Figure 7.82: Example Solution for L1 (TMT1)

⁸See Chapter 6, Section 6.5.2 for the explanation on the difference between the TMT1 and TMT2 system.

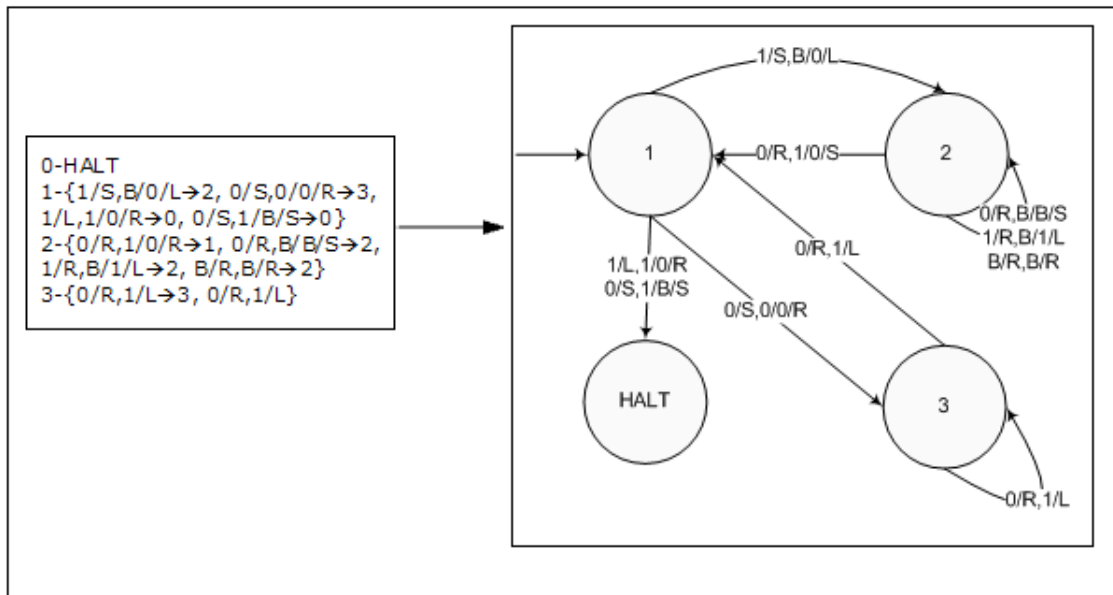


Figure 7.83: Example Solution for L1 (TMT2)

L2 (A Turing machine that takes in an integer n in unary and outputs $2n$ ($n \geq 1$)): The results for L2 produced by TMT1 are outlined in Table D.26 and Table D.27 and the results produced by TMT2 are outlined in Table D.38 and Table D.39. Solutions were induced for all seed values using standard operators and non-destructive operators. Figure 7.84 shows an example solution for TMT1 and Figure 7.85 shows an example solution for TMT2. The HALT state is not used for TMT1 and TMT2 and is therefore not drawn in the transition diagrams.

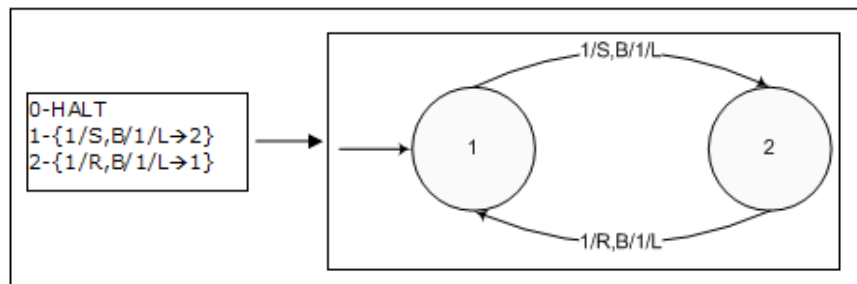


Figure 7.84: Example Solution for L2 (TMT1)

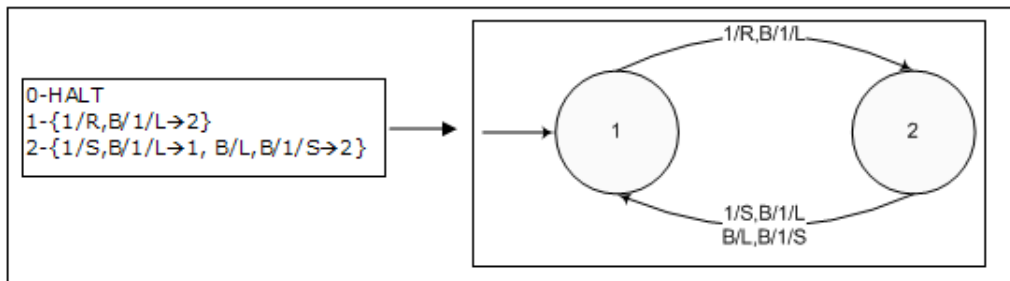


Figure 7.85: Example Solution for L2 (TMT2)

L3 (A Turing machine that takes in an integer n in unary and outputs $2n + 1$ ($n \geq 1$)): The results for L3 are outlined in Table D.28 and Table D.29 for TMT1. The results produced by TMT2 are outlined in Table D.40 and Table D.41. Solutions were induced for all seed values using standard operators and non-destructive operators. Figure 7.86 and Figure 7.87 shows example solutions for TMT1 and TMT2 respectively.

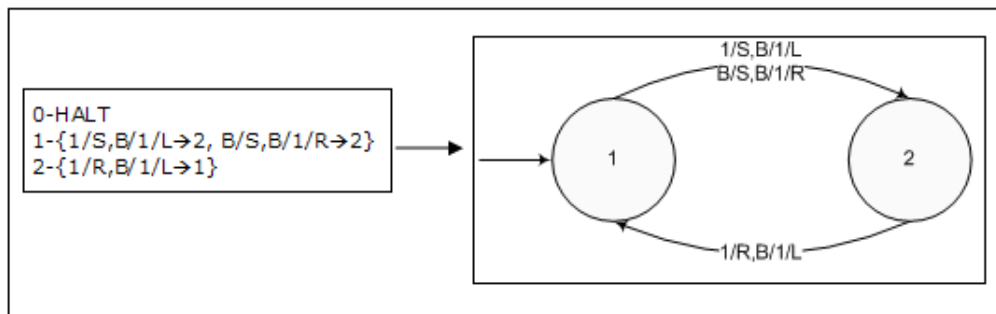


Figure 7.86: Example Solution for L3 (TMT1)

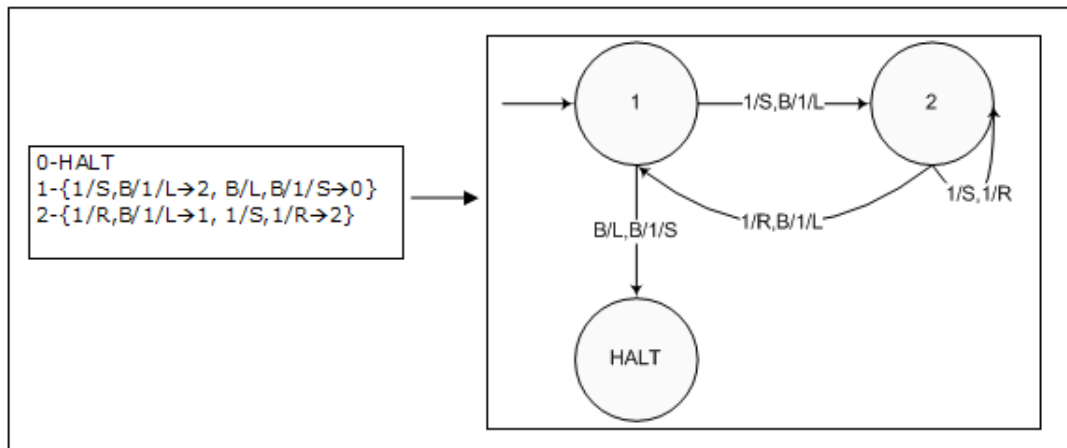


Figure 7.87: Example Solution for L3 (TMT2)

L4 (A Turing machine that takes in two integers in unary form separated by 0 as input and outputs the greater of both numbers): The results for TMT1 are outlined in Table D.30 and Table D.31. Solutions were induced for 9 of the 10 seed values using standard operators and for all seed values when employing non-destructive operators. Figure 7.88 shows an example solution for TMT1. The results for TMT2 are outlined in Table D.42 and Table D.43. No solutions were produced using this method.

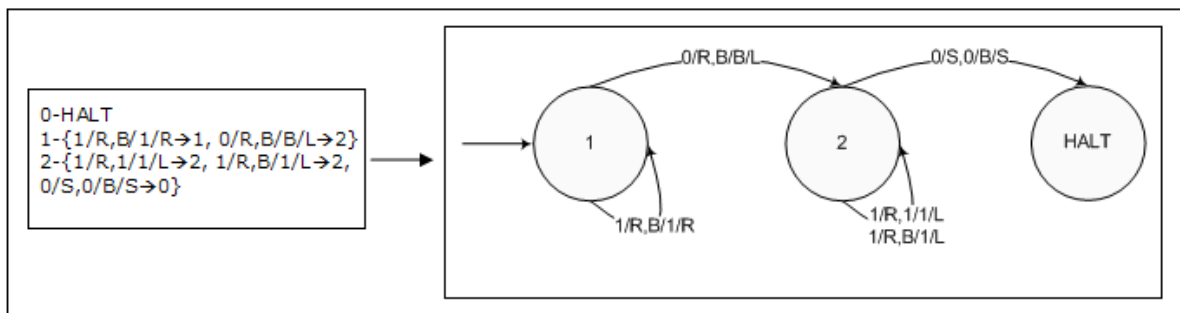


Figure 7.88: Example Solution for L4 (TMT1)

L5 (Unary subtraction where the input consists of a string of unary integers separated by a 0): The results using TMT1 for L5 are outlined in Table D.32 and Table D.33. Solutions were induced for one seed value using standard operators and for 4 of the 10 seed values when employing non-destructive crossover and for non-destructive crossover and mutation. The results for TMT2 are shown in Table D.44 and Table D.45. The results for TMT2 are much better with solutions being generated for four seed values using standard operators and for all seed values using non-destructive crossover and mutation. Figure 7.89 and Figure 7.90 shows example solutions for L5 for TMT1 and TMT2 respectively.

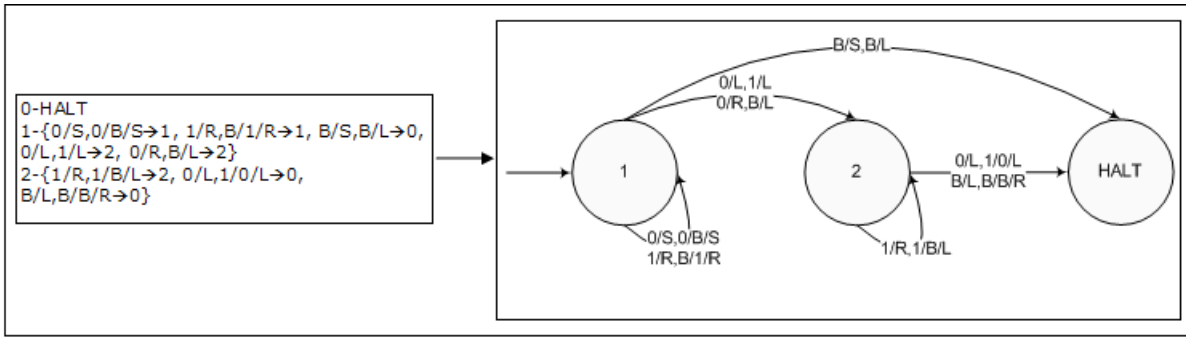


Figure 7.89: Example Solution for L5 (TMT1)

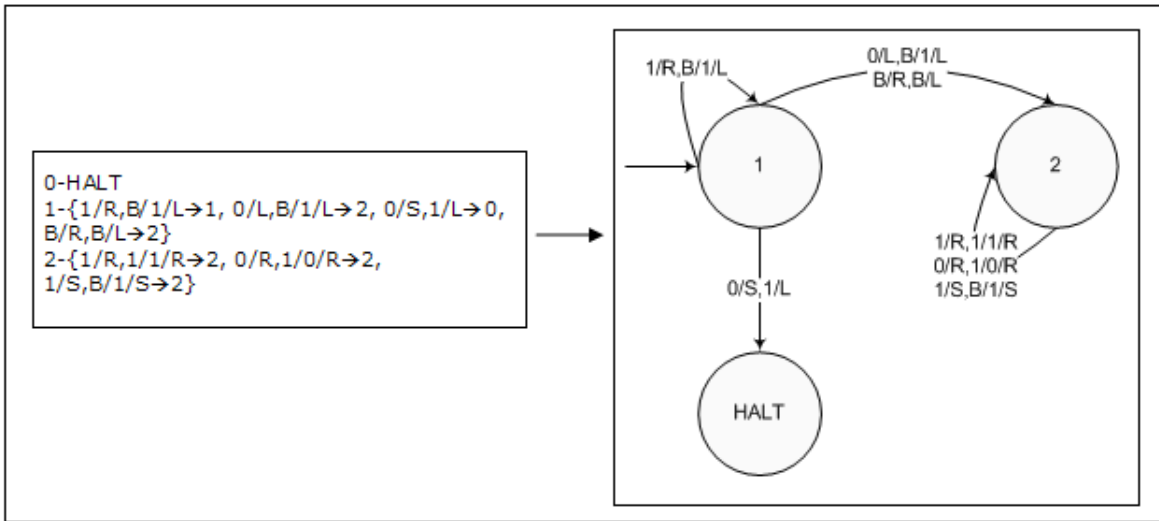


Figure 7.90: Example Solution for L5 (TMT2)

L6 (A Turing machine that performs two-symbol sorting): The results using TMT1 for L6 are outlined in Table D.34 and Table D.35. Only one solution was induced for L6 using non-destructive crossover. The results for TMT2 are shown in Table D.46 and Table D.47. Two solutions were induced for L6 using this method with non-destructive crossover and mutation. Figure 7.91 shows the output for the solution generated. A majority of the nodes and transitions are redundant and will never be reached or traversed. Figure 7.92 shows the transition diagram with all the unnecessary nodes removed. Figure 7.93 shows example solutions for L6 for TMT2.

```

0-HALT
1-{a/R,b/a/L→2,a/R,B/b/L→1,b/S,b/b/R→11,b/R,B/b/L→1,B/L,B/R→8}
2-{b/L,b/S→8,b/R,B/B/L→3,a/R,B/B/R→4,a/R,a/R→3,a/S,b/a/R→4}
3-{a/R,b/a/L→5,a/R,B/a/L→2,b/S,b/b/S→6,b/R,B/b/L→7,B/L,B/R→3}
4-{a/L,b/R→7,b/L,B/L→11,a/R,B/b/R→6,a/R,a/L→4}
5-{b/S,B/S→9,B/R,B/B/R→11,a/R,b/L→11,a/L,B/B/L→11}
6-{b/L,B/a/R→12,b/R,b/L→10,a/L,b/B/R→9}
7-{a/R,b/a/L→12,a/R,B/a/L→12,b/S,b/b/S→11,b/R,B/b/L→11,
B/L,B/R→10}
8-{b/L,b/S→8,a/L,b/a/R→8}
9-{B/L,B/a/L→12,b/L,B/L→8,a/S,a/S→12,a/R,B/L→9}
10-{a/R,b/b/S→8,b/S,B/B/R→8,a/S,a/a/R→8,a/L,B/a/R→9}
11-{b/L,b/b/L→8}
12-{a/L,a/a/R→9,a/L,B/B/L→11,a/S,b/a/R→9}

```

Figure 7.91: Example Solution Output for L6 (TMT1)

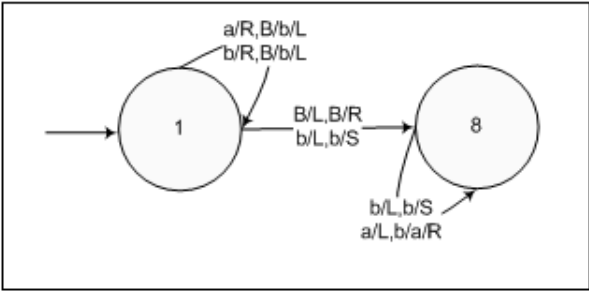


Figure 7.92: Example Solution for L6 (TMT1)

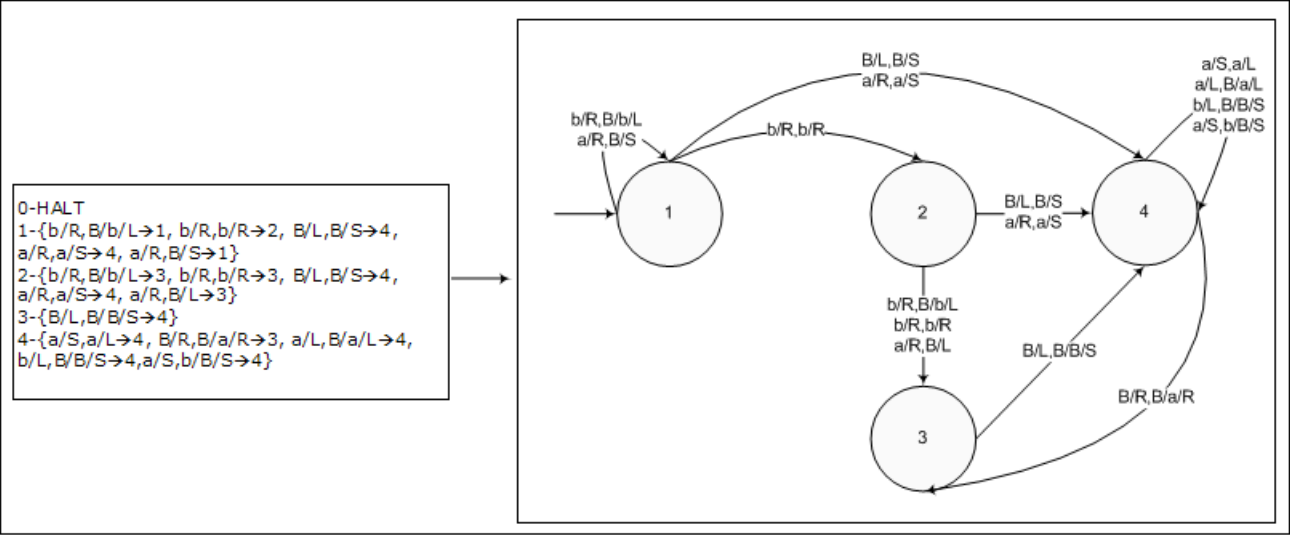


Figure 7.93: Example Solution for L6 (TMT2)

The GP system was able to produce solutions for all languages in the language set. Unlike the TMAs, a single run, with one iteration, took between 5 minutes to 10 minutes for L1–L5. L6, however, took between 30 minutes to an hour to complete a single run. This can be attributed to the fact that L6 has many more fitness cases. Table 7.29 lists the success rates for each language for TMT1. The success rate is calculated as a percentage of the seed values that produced solutions over 10 iterations for the languages.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 70% | 100% | 100% | 100% |
| L2 | 100% | 100% | 100% | 100% |
| L3 | 100% | 100% | 100% | 100% |
| L4 | 90% | 100% | 100% | 100% |
| L5 | 10% | 30% | 40% | 30% |
| L6 | 0% | 0% | 10% | 0% |

Table 7.29: Success Rates for TMT1 Simulations

Table 7.30 gives the percentage of runs requiring more than one iteration.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 80% | 40% | 50% | 30% |
| L2 | 0% | 0% | 0% | 0% |
| L3 | 70% | 80% | 90% | 0% |
| L4 | 80% | 60% | 30% | 50% |
| L5 | 100% | 90% | 90% | 90% |
| L6 | 100% | 100% | 100% | 100% |

Table 7.30: Multiple iterations required for TMT1 Simulations

Table 7.31 lists the success rates for each language for TMT2.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 40% | 80% | 90% | 100% |
| L2 | 100% | 100% | 100% | 100% |
| L3 | 100% | 100% | 100% | 100% |
| L4 | 0% | 0% | 0% | 0% |
| L5 | 40% | 80% | 90% | 100% |
| L6 | 0% | 0% | 0% | 20% |

Table 7.31: Success Rates for TMT2 Simulations

Table 7.32 gives the percentage of runs requiring more than one iteration.

| Language | Standard Operators | Non-Destructive | | |
|----------|--------------------|-----------------|----------------|------------------------|
| | | Mutation only | Crossover only | Crossover and Mutation |
| L1 | 100% | 80% | 70% | 60% |
| L2 | 30% | 10% | 0% | 0% |
| L3 | 70% | 20% | 50% | 30% |
| L4 | 100% | 100% | 100% | 100% |
| L5 | 100% | 90% | 90% | 80% |
| L6 | 100% | 100% | 100% | 100% |

Table 7.32: Multiple iterations required for TMT2 Simulations

TMT1 performs slightly better for L1. For L2 and L3 both methods produces a 100% success rate, however more iterations were required for TMT2. No solutions were found for L4 using TMT2, however for TMT1, L4 has a 100% success rate when using non-destructive operators and a 90% success rate using standard operators. For L5, on the other hand, TMT2 produces better results. Even though a low number of fitness cases were used for L2 and L3 the success rate is still high. The success rate for L6 (the two-symbol sorting problem) is poor. Tanomaru [45] suggests that the large search space has an effect on the ability of the system to generate TMs for the two-symbol sorting problem. A solution for L6 involves going through the input tape twice to sort the symbols in the correct order. This could possibly add to the difficulty in finding a solution.

Table 7.33 shows the “Human generated” solutions for the Turing machine transducers for TMT1. Once again, as with the Turing machine acceptors, the GP system produces structural redundancies. L6 is an example of the large number of redundant nodes and transitions produced by the GP system for TMT1. There are eleven redundant nodes for the solution generated for L6. The redundant nodes and transitions add to the complexity of the solutions. For some languages these unnecessary transitions are traversed by the system which add to the runtime of the system.

| Language | Human Generated Solutions |
|----------|--|
| L1 | <pre> graph LR start(()) --> q0((q0)) q0 -- "1/R, B/1/R 0/R, B/S" --> q0 q0 -- "B/S, B/0/R" --> HALT((HALT)) </pre> |
| L2 | <pre> graph LR start(()) --> q0((q0)) q0 -- "1/S, B/1/R" --> q1((q1)) q1 -- "1/R, B/1/R" --> q0 q1 -- "B/S, B/S" --> HALT((HALT)) </pre> |
| L3 | <pre> graph LR start(()) --> q0((q0)) q0 -- "1/S, B/1/R" --> q1((q1)) q1 -- "1/R, B/1/R" --> q0 q1 -- "B/S, B/1/R" --> HALT((HALT)) </pre> |
| L4 | <pre> graph LR start(()) --> q0((q0)) q0 -- "0/R, B/L" --> q1((q1)) q0 -- "1/R, B/1/R" --> q0 q1 -- "1/R, 1/L 1/R, B/1/L" --> q1 q1 -- "B/S, B/S" --> HALT((HALT)) </pre> |
| L5 | <pre> graph LR start(()) --> q0((q0)) q0 -- "0/R, B/L" --> q1((q1)) q0 -- "1/R, B/1/R" --> q0 q1 -- "1/R, 1/B/L" --> q1 q1 -- "B/S, B/S" --> HALT((HALT)) </pre> |
| L6 | <pre> graph LR start(()) --> q0((q0)) q0 -- "B/L, B/S" --> q1((q1)) q0 -- "a/R, B/a/R b/R, B/S" --> q0 q1 -- "b/L, B/b/R a/L, B/S" --> q1 q1 -- "B/S, B/S" --> HALT((HALT)) </pre> |

Table 7.33: “Human Generated” Solutions (TMTs)

Tanomaru's [45] system was tested on two languages i.e. the two-symbol sorting problem and the unary subtraction problem. The system was successful in evolving solutions for both problems. He gives the results for the ten best runs, which all generated solutions. The GP system was able to produce similar results for the unary subtraction problem, however the two-symbol sorting problem did perform significantly worse. The GP system was only able to achieve a 20% success rate for this language. One of the differences between the GP system and the system presented by Tanomaru is the fact that the GP system uses two tapes. By using one tape the Tanomaru system effectively performs a bubble sort on the tape. When using two tapes however, this is not possible, unless the system generates a solution that effectively copies the input string to the output tape and then performs the bubble sort on the output tape.

Chapter 8

Conclusion

The main aim of this thesis was to evaluate genetic programming as a means of evolving various classes of automata. The various GP systems were able to evolve deterministic finite acceptors, non-deterministic finite acceptors, finite state transducers, pushdown automata, Turing machine acceptors and Turing machine transducers. All of the GP systems successfully made use of directed graphs to represent the individuals, which is the main difference between this study and the previous work done.

All GP systems presented in this thesis was able to successfully induce solutions for all the languages in the language sets defined. The use of non-destructive and multiple iterations has been successfully employed to overcome premature convergence. A modular approach was also attempted for the 3-character FA languages with good success. A success rate of 80% or better was achieved for the FAs, FSTs and PDAs. The Turing machines on the other hand had a poor success rate for certain languages. This can probably be attributed to the large search space for Turing machines.

Most of the solutions generated by the GP systems contain structural redundancies, which not only adds to the complexity of the solutions, but also increases the processing time of the systems. These redundancies in the solutions generated by the FA systems can easily be removed by applying the minimizing algorithm. The FST system, on the other hand, was able to generate human competitive solutions and the minimal transducer was found for most of the languages in the language set. The PDA and Turing machine systems, produced solutions with redundant nodes and transitions which are unnecessarily traversed. Some of the solutions produced by the GP system differed from the typical human solutions.

Methods of reducing the search space for Turing machines needs to be investigated further. Removing the structural redundancies could possibly reduce the search space and improve the results obtained by the Turing machine GP systems. Implementing syntax shortcuts as described in the JFLAP tutorial [23] could also reduce the search space. Future work for improving the success rates for TMs could also include employing a modular approach similar to that applied to FAs for L9 and L15 as it worked well for both these languages. Future work could also include applying this approach to specific applications domains e.g., natural language processing or compiler construction. Using the FA system as

part of an intelligent tutoring system, has been examined in [42], and can also be investigated further.

Bibliography

- [1] P. J. Angeline and J. B. Pollack. Evolutionary module acquisition. In D. Fogel and W. Atmar, editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 154–163, La Jolla, CA, USA, 25-26 1993.
- [2] S. Baase and A. van Gelder. *Computer Algorithms (Introduction to design and analysis)*, 3rd Edition. Addison-Wesley, Reading Massachusetts, 2000. ISBN 0-201-61244-5.
- [3] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming An Introduction: On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, California, 1998.
- [4] S. Bates. Knuth–Morris–Pratt algorithm. Available: <http://www.nist.gov/dads/HTML/knuthMorrisPratt.html>, Retrieved: 2005.
- [5] A. Belz and B. Eskikaya. A Genetic Algorithm for Finite State Automata Induction with an Application to Phonotactics. In B. Keller, editor, *Proceedings of the ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing*, pages 9–17, Saarbruecken, Germany, 1998.
- [6] T. Blickle and L. Thiele. A Comparison of Selection Schemes Used in Genetic Algorithms. Available: <http://citeseer.ist.psu.edu/article/blickle95comparison.html>, Retrieved: 2006, Last Updated: 1995.
- [7] S. Brave. Evolving Deterministic Finite Automata Using Cellular Encoding. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 39–44, Stanford University, CA, USA, 1996. MIT Press.
- [8] J. Brownlee. Finite State Machines *fsm*. Available: <http://ai-depot.com/FiniteStateMachines/>, Retrieved: 2004, Last Updated: 2003.
- [9] D. I. A. Cohen. *Introduction to Computer Theory, 2nd edition*. John Wiley and Sons, Inc., Hunter College, City University of New York, 1991. ISBN 0-471-13772-3. 634 pp.
- [10] PlanetMath contributors. Non-deterministic pushdown automaton. Available: <http://planetmath.org/encyclopedia/PushdownAutomaton.html>, Retrieved: 2004.

- [11] F. Coste. Grammatical Inference Benchmarks Repository and Links. Available: http://www.irisa.fr/symbiose/people/coste/gi_benchs.html, Retrieved: 2006.
- [12] H. G. Dales and G. Oliveri, editors. *Truth in Mathematics*. Oxford University Press, USA, 1998. ISBN 1-85233-074-0. 916 pp.
- [13] B. D. Dunay. Context Free Language Induction with Genetic Programming. In *Proceedings of the International Conference on Tools with Artificial Intelligence*, pages 828–831, New Orleans, LA, 1994. IEEE Computer Society Press.
- [14] B.D. Dunay, F.E. Petry, and B.P. Buckles. Regular Language Induction with Genetic Programming. In *Proceedings of the first IEEE Conference on Evolutionary Computation*, pages 396–400, Orlando, Florida, USA, 1994. IEEE Press.
- [15] P. Dupont. Regular Grammatical Inference from Positive and Negative Samples by Genetic Search: the GIG method. In *ICGI '94: Proceedings of the Second International Colloquium on Grammatical Inference and Applications*, pages 236–245, London, UK, 1994. Springer-Verlag. ISBN 3-540-58473-0.
- [16] L. Fegaras. Design and Construction of Compilers – lecture notes. Available: <http://lambda.uta.edu/cse5317/notes/node8.html>, Retrieved: 2005.
- [17] M. L. Forcada. Neural Networks: Automata and Formal Methods of Computation. Available: <http://www.dlsi.ua.es/~Emlf/nafmc/pbook.pdf>, Retrieved: 2005, Last Updated: 2002.
- [18] ICGI Steering Group. Grammatical induction community. Available: <http://labh-curien.univ-st-etienne.fr/informatique/gi/>, Retrieved: 2007.
- [19] F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis, France, 1994.
- [20] P. Hingston. A Genetic Algorithm for Regular Inference. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 1299–1306, San Francisco, California, USA, 2001. M. Kaufmann. ISBN 1-55860-774-9.
- [21] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Jones and Bartlett Publishers, 1979. ISBN 0-201-02988-X.
- [22] W. Huijsen. Genetic grammatical inference. In G. Bouma and G. van Noord, editors, *CLIN IV. Papers from the Fourth CLIN Meeting, Vakgroep Alfa-informatica*, pages 59–72, University of Groningen, 1994.
- [23] JFLAP. Building A Turing Machine. Available: <http://www.jflap.org/tutorial/turing/one/index.html>, Retrieved: 2007.
- [24] C. Johnson. Basic Research Skills in Computing Science. Available: http://www.dcs.gla.ac.uk/~johnson/teaching/research_skills/basics.html, Retrieved: 2006.

- [25] C. Johnson. What is Research in Computing Science? Available: http://www.dcs.gla.ac.uk/~johnson/teaching/research_skills/research.html, Retrieved: 2006.
- [26] W. Kantschik and W. Banzhaf. Linear-Graph GP – A New GP Structure. In *EuroGP '02: Proceedings of the 5th European Conference on Genetic Programming*, pages 83–92, London, UK, 2002. Springer-Verlag. ISBN 3-540-43378-3.
- [27] J. R. Koza. *Genetic Programming: On the programming of Computers by Means of Natural Selection*. MIT Press, London, England, 1992.
- [28] M. Lankhorst. A Genetic Algorithm for Induction of Nondeterministic Pushdown Automata. In *University of Groningen, Computer Science Report CS-R 9502, The Netherlands*, 1995.
- [29] P. Linz. *An Introduction to Formal Languages and Automata, Third Edition*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 2001. 418 pp.
- [30] S. M. Lucas. Evolving Finite State Transducers: Some Initial Explorations. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, and E. Costa R. Poli, editors, *Proceedings of the Sixth European Conference on Genetic Programming (EuroGP-2003)*, volume 2610 of *LNCS*, pages 130–141, Essex, UK, 2003. Springer Verlag. ISBN 3-540-00971-X.
- [31] S. M. Lucas and T. Reynolds. Learning DFA: Evolution versus Evidence Driven State Merging. In *Proceedings of the 2003 Congress on Evolutionary Computing (CEC 2003)*, pages 351–358. IEEE Press, 2003. ISBN 1-55860-774-9.
- [32] S. M. Lucas and T. Reynolds. Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 27, pages 1063–1074, Washington, DC, USA, 2005. IEEE Computer Society.
- [33] S. Luke, S. Hamahashi, and H. Kitano. “Genetic” Programming. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1098–1105, Orlando, Florida, USA, 1999. M. Kaufmann. ISBN 1-55860-611-4.
- [34] P. Machado, F. B. Pereira, A. Cardoso, and E. Costa. Busy Beaver – The Influence of Representation. In *European Workshop on Genetic Programming, EuroGP'99*, volume 1598, pages 29–38, Goteborg, Sweden, 1999. Springer-Verlag. ISBN 3-540-65899-8.
- [35] P. Machado, F. B. Pereira, E. Costa, and A. Cardoso. Graph-Based Crossover: A Case Study with the Busy Beaver Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 1149–1155, Orlando, USA, 1999.
- [36] P. Machado, F. B. Pereira, J. Tavares, and A. Cardoso. Evolutionary Turing Machines: The quest for busy beavers. In *Recent Developments in Biologically Inspired Computing*, 2004.
- [37] S. W. Mahfoud. Niching Methods for Genetic Algorithms. In *IlliGal Report No. 95001*, Urbana, IL, 1995. University of Illinois at Urbana-Champaign.

- [38] J. Martin. *Introduction to Languages and the Theory of Computation, 3rd Edition*. McGraw-Hill, Avenue of the Americas, New York, 2003. ISBN 0-07-119854-7. 543 pp.
- [39] A. Meduna. *Automata and Languages*. Springer-Verlag, London, 2000. ISBN 1-85233-074-0. 916 pp.
- [40] M. S. Olivier. *Information Technology Research: A Practical Guide*. Johannesburg, 1999. 176 pp.
- [41] L. Ong. Computational Complexity: Lecture notes. Available: <http://users.comlab.ox.ac.uk/luke.ong/teaching/complexity>, Retrieved: 2007, Last Updated: 1999.
- [42] N. Pillay and A. Naidoo. An Investigation into the Automatic Generation of Solutions to Problems in an Intelligent Tutoring System for Finite Automata. In J. Van Belle and I. Brown, editors, *Proceedings of SACLA 2006*, pages 84–93, 2006.
- [43] H. Pohlheim. Evolutionary algorithms. In *GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB Documentation*. GEATbx.
- [44] Conor Ryan and Michael O’Neill. Grammatical evolution: A steady state approach. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Stanford University Bookstore.
- [45] J. Tanomaru. Evolving Turing Machines from Examples. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution*, volume 1363 of *LNCS*, Nimes, France, 1993. Springer-Verlag.
- [46] A. Teller and M. Veloso. PADO: A New Learning Architecture for Object Recognition. In Katsushi Ikeuchi and Manuela Veloso, editors, *Symbolic Visual Learning*, pages 81–116. Oxford University Press, 1996.
- [47] S. Toida. CS390 Web Course Study Materials: Introduction to Theoretical Computer Science/Theory of Computation. Available: http://www.cs.odu.edu/~toida/nerzic/390teched/web_course.html, Retrieved: 2006.
- [48] Unknown. Class Random: Java 2 Platform Standard Ed. 5.0. Available: <http://java.sun.com/j2se/1.5.0/docs/api/java/util/Random.html>, Retrieved: 2006.
- [49] E. E. Vallejo and F. Ramos. Evolving Turing machines for Biosequences Recognition and Analysis. In J. F. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. G. B. Tettamanzi, and W. B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP’2001*, volume 2038 of *LNCS*, pages 192–203, Lake Como, Italy, 2001. Springer-Verlag. ISBN 3-540-41899-7.
- [50] M. Walker. Introduction to Genetic Programming. Available: http://www.massey.ac.nz/~mgwalker/gp/intro/introduction_to_gp.pdf, Retrieved: 2004, Last Updated: 2001.
- [51] Wikipedia. Automata-Based Programming. Available: http://en.wikipedia.org/w/index.php?title=Automata-Based_Programming&oldid=162432941, Retrieved: 2006.

- [52] Wikipedia. Automata Theory. Available: http://en.wikipedia.org/w/index.php?title=Automata_theory&oldid=175917463, Retrieved: 2006.
- [53] Wikipedia. Busy Beaver. Available: http://en.wikipedia.org/w/index.php?title=Busy_beaver&oldid=175048268, Retrieved: 2007.
- [54] Wikipedia. Chomsky hierarchy. Available: http://en.wikipedia.org/w/index.php?title=Chomsky_hierarchy&oldid=84419317, Retrieved: 2006.
- [55] Wikipedia. Hermeneutics. Available: <http://en.wikipedia.org/w/index.php?title=Hermeneutics&oldid=175814090>, Retrieved: 2006.
- [56] Wikipedia. Mealy Machine. Available: http://en.wikipedia.org/w/index.php?title=Mealy_machine&oldid=175239728, Retrieved: 2006.
- [57] Wikipedia. Moore Machine. Available: http://en.wikipedia.org/w/index.php?title=Moore_machine&oldid=175242259, Retrieved: 2006.
- [58] Wikipedia. Phonotactics. Available: <http://en.wikipedia.org/w/index.php?title=Phonotactics&oldid=87472504>, Retrieved: 2006.
- [59] Wikipedia. Premature convergence. Available: http://en.wikipedia.org/w/index.php?title=Premature_convergence&oldid=177328985, Retrieved: 2007.
- [60] Wikipedia. Research. Available: <http://en.wikipedia.org/w/index.php?title=Research&oldid=176123253>, Retrieved: 2006.
- [61] Wikipedia. Virtual finite state machine. Available: http://en.wikipedia.org/w/index.php?title=Virtual_finite_state_machine&oldid=88856938, Retrieved: 2006.
- [62] A. Zomorodian. Context-Free Language Induction by Evolution of Deterministic Push-Down Automata Using Genetic Programming. In E. V. Siegel and J. R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 127–133, MIT, Cambridge, MA, USA, 1995. AAAI.

Appendix A

System Results for Finite Acceptors

A.1 Deterministic Finite Acceptors

| Seed | Run | Generation | Best Fitness |
|------|-----|----------------|--------------|
| 1000 | 1 | 1 ¹ | 100% |
| 1010 | 1 | 1 | 100% |
| 2000 | 1 | 1 | 100% |
| 2500 | 1 | 1 | 100% |
| 3000 | 1 | 1 | 100% |
| 3007 | 1 | 1 | 100% |
| 3500 | 1 | 1 | 100% |
| 4000 | 1 | 1 | 100% |
| 4021 | 1 | 1 | 100% |
| 5529 | 1 | 1 | 100% |

Table A.1: Standard Operator Results for L1 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 1 | 100% |
| 1010 | 1 | 1 | 100% |
| 2000 | 1 | 2 | 100% |
| 2500 | 1 | 2 | 100% |
| 3000 | 1 | 2 | 100% |
| 3007 | 1 | 2 | 100% |
| 3500 | 1 | 2 | 100% |
| 4000 | 1 | 2 | 100% |
| 4021 | 1 | 3 | 100% |
| 5529 | 1 | 2 | 100% |

Table A.2: Standard Operator Results for L2 (DFA)

¹1 is the initial generation of the population

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 1010 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 2000 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 2500 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 3000 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 3007 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 3500 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 4000 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 4021 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 5529 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |

Table A.3: Non-destructive Results for L2 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 5 | 15 | 100% |
| 1010 | 2 | 12 | 100% |
| 2000 | 2 | 7 | 100% |
| 2500 | 1 | 9 | 100% |
| 3000 | 4 | 13 | 100% |
| 3007 | 2 | 23 | 100% |
| 3500 | 2 | 19 | 100% |
| 4000 | 2 | 13 | 100% |
| 4021 | 1 | 10 | 100% |
| 5529 | 3 | 20 | 100% |

Table A.4: Standard Operator Results for L3 (DFA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 11 | 2 | 4 | 3 | 6 | 100% |
| 1010 | 1 | 8 | 1 | 4 | 1 | 8 | 100% |
| 2000 | 1 | 31 | 1 | 6 | 1 | 7 | 100% |
| 2500 | 2 | 41 | 2 | 15 | 3 | 22 | 100% |
| 3000 | 3 | 8 | 1 | 4 | 1 | 29 | 100% |
| 3007 | 1 | 8 | 1 | 4 | 1 | 4 | 100% |
| 3500 | 2 | 2 | 2 | 5 | 2 | 8 | 100% |
| 4000 | 1 | 33 | 1 | 6 | 1 | 5 | 100% |
| 4021 | 1 | 11 | 1 | 11 | 1 | 51 | 100% |
| 5529 | 1 | 6 | 1 | 3 | 1 | 5 | 100% |

Table A.5: Non-destructive Results for L3 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 2 | 6 | 100% |
| 1010 | 1 | 7 | 100% |
| 2000 | 1 | 3 | 100% |
| 2500 | 1 | 5 | 100% |
| 3000 | 1 | 2 | 100% |
| 3007 | 1 | 5 | 100% |
| 3500 | 1 | 4 | 100% |
| 4000 | 2 | 5 | 100% |
| 4021 | 1 | 6 | 100% |
| 5529 | 1 | 5 | 100% |

Table A.6: Standard Operator Results for L4 (DFA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 5 | 1 | 4 | 1 | 4 | 100% |
| 1010 | 1 | 5 | 1 | 4 | 1 | 3 | 100% |
| 2000 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 2500 | 1 | 6 | 1 | 4 | 1 | 3 | 100% |
| 3000 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 3007 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 3500 | 1 | 4 | 1 | 4 | 1 | 3 | 100% |
| 4000 | 1 | 6 | 1 | 3 | 1 | 3 | 100% |
| 4021 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 5529 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |

Table A.7: Non-destructive Results for L4 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 9 | 100% |
| 1010 | 3 | 8 | 100% |
| 2000 | 1 | 15 | 100% |
| 2500 | 3 | 15 | 100% |
| 3000 | 5 | 7 | 100% |
| 3007 | 3 | 9 | 100% |
| 3500 | 1 | 5 | 100% |
| 4000 | 4 | 12 | 100% |
| 4021 | 2 | 7 | 100% |
| 5529 | 1 | 10 | 100% |

Table A.8: Standard Operator Results for L5 (DFA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 1010 | 1 | 12 | 1 | 4 | 2 | 4 | 100% |
| 2000 | 2 | 6 | 1 | 3 | 1 | 3 | 100% |
| 2500 | 4 | 5 | 1 | 3 | 1 | 5 | 100% |
| 3000 | 1 | 7 | 1 | 4 | 1 | 4 | 100% |
| 3007 | 2 | 7 | 1 | 4 | 2 | 4 | 100% |
| 3500 | 2 | 7 | 4 | 4 | 3 | 3 | 100% |
| 4000 | 1 | 9 | 1 | 4 | 1 | 8 | 100% |
| 4021 | 1 | 6 | 1 | 6 | 1 | 4 | 100% |
| 5529 | 1 | 9 | 1 | 5 | 1 | 3 | 100% |

Table A.9: Non-destructive Results for L5 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 8 | 100% |
| 1010 | 1 | 12 | 100% |
| 2000 | 1 | 5 | 100% |
| 2500 | 1 | 5 | 100% |
| 3000 | 1 | 5 | 100% |
| 3007 | 1 | 6 | 100% |
| 3500 | 1 | 5 | 100% |
| 4000 | 1 | 5 | 100% |
| 4021 | 1 | 1 | 100% |
| 5529 | 1 | 4 | 100% |

Table A.10: Standard Operator Results for L6 (DFA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 5 | 1 | 3 | 1 | 3 | 100% |
| 1010 | 1 | 3 | 2 | 3 | 1 | 3 | 100% |
| 2000 | 1 | 5 | 1 | 3 | 1 | 2 | 100% |
| 2500 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 3000 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 3007 | 1 | 27 | 1 | 3 | 1 | 3 | 100% |
| 3500 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 4000 | 1 | 3 | 1 | 5 | 1 | 3 | 100% |
| 4021 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 5529 | 1 | 3 | 1 | 3 | 1 | 2 | 100% |

Table A.11: Non-destructive Results for L6 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 5 | 100% |
| 1010 | 1 | 5 | 100% |
| 2000 | 1 | 13 | 100% |
| 2500 | 1 | 6 | 100% |
| 3000 | 1 | 7 | 100% |
| 3007 | 1 | 4 | 100% |
| 3500 | 1 | 6 | 100% |
| 4000 | 1 | 6 | 100% |
| 4021 | 2 | 3 | 100% |
| 5529 | 1 | 7 | 100% |

Table A.12: Standard Operator Results for L7 (DFA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 3 | 2 | 4 | 1 | 3 | 100% |
| 1010 | 1 | 3 | 1 | 4 | 1 | 3 | 100% |
| 2000 | 1 | 8 | 1 | 3 | 1 | 4 | 100% |
| 2500 | 1 | 3 | 1 | 3 | 1 | 4 | 100% |
| 3000 | 1 | 14 | 1 | 3 | 1 | 4 | 100% |
| 3007 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 3500 | 1 | 5 | 1 | 4 | 1 | 4 | 100% |
| 4000 | 1 | 6 | 1 | 3 | 1 | 4 | 100% |
| 4021 | 1 | 4 | 1 | 3 | 2 | 4 | 100% |
| 5529 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |

Table A.13: Non-destructive Results for L7 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 2 | 100% |
| 1010 | 1 | 1 | 100% |
| 2000 | 1 | 2 | 100% |
| 2500 | 1 | 1 | 100% |
| 3000 | 1 | 2 | 100% |
| 3007 | 1 | 2 | 100% |
| 3500 | 1 | 2 | 100% |
| 4000 | 1 | 2 | 100% |
| 4021 | 1 | 1 | 100% |
| 5529 | 1 | 1 | 100% |

Table A.14: Standard Operator Results for L8 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|----------------|------------|--------------|
| 1000 | - ² | - | 98.3% |
| 1010 | - | - | 98.3% |
| 2000 | - | - | 98.3% |
| 2500 | - | - | 98.3% |
| 3000 | - | - | 98.3% |
| 3007 | - | - | 98.3% |
| 3500 | - | - | 98.3% |
| 4000 | - | - | 98.3% |
| 4021 | - | - | 98.3% |
| 5529 | - | - | 98.3% |

Table A.15: Standard Operator Results for L9 (DFA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | - | - | - | - | - | - | 98.3% |
| 1010 | - | - | - | - | - | - | 98.3% |
| 2000 | - | - | - | - | - | - | 99.6% |
| 2500 | - | - | - | - | 18 | 3 | 100% |
| 3000 | - | - | - | - | - | - | 98.3% |
| 3007 | - | - | - | - | - | - | 98.3% |
| 3500 | - | - | - | - | - | - | 98.3% |
| 4000 | - | - | - | - | - | - | 98.3% |
| 4021 | - | - | 5 | 3 | - | - | 100% |
| 5529 | - | - | - | - | - | - | 99.6% |

Table A.16: Non-destructive Results for L9 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 97.8% |
| 1010 | 4 | 32 | 100% |
| 2000 | 1 | 12 | 100% |
| 2500 | 1 | 36 | 100% |
| 3000 | 10 | 42 | 100% |
| 3007 | - | - | 97.8% |
| 3500 | - | - | 99.6% |
| 4000 | - | - | 97.8% |
| 4021 | - | - | 97.8% |
| 5529 | 8 | 25 | 100% |

Table A.17: Standard Operator Results for L10 (DFA)

²- indicates that there are no solutions for the seed value over all iterations

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 19 | 1 | 6 | 3 | 7 | 100% |
| 1010 | 7 | 16 | 3 | 14 | 2 | 14 | 100% |
| 2000 | - | - | 1 | 5 | 1 | 5 | 100% |
| 2500 | 6 | 34 | 8 | 6 | 3 | 16 | 100% |
| 3000 | - | - | 3 | 7 | 7 | 9 | 100% |
| 3007 | 1 | 30 | 2 | 11 | 3 | 9 | 100% |
| 3500 | 7 | 31 | 1 | 6 | 1 | 6 | 100% |
| 4000 | 3 | 45 | 3 | 5 | 1 | 10 | 100% |
| 4021 | 1 | 17 | 3 | 11 | 3 | 40 | 100% |
| 5529 | 1 | 16 | 1 | 5 | 1 | 10 | 100% |

Table A.18: Non-destructive Results for L10 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 7 | 100% |
| 1010 | 1 | 9 | 100% |
| 2000 | 9 | 6 | 100% |
| 2500 | 1 | 12 | 100% |
| 3000 | 9 | 7 | 100% |
| 3007 | 1 | 28 | 100% |
| 3500 | 1 | 15 | 100% |
| 4000 | 1 | 4 | 100% |
| 4021 | 5 | 6 | 100% |
| 5529 | 1 | 15 | 100% |

Table A.19: Standard Operator Results for L11 (DFA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 5 | 1 | 3 | 1 | 3 | 100% |
| 1010 | 3 | 18 | 2 | 4 | 1 | 27 | 100% |
| 2000 | 5 | 13 | 1 | 4 | 1 | 45 | 100% |
| 2500 | 2 | 8 | 2 | 4 | 1 | 4 | 100% |
| 3000 | 3 | 9 | 1 | 12 | 2 | 4 | 100% |
| 3007 | 1 | 5 | 1 | 3 | 1 | 3 | 100% |
| 3500 | 1 | 8 | 1 | 9 | 1 | 5 | 100% |
| 4000 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 4021 | 1 | 34 | 1 | 4 | 1 | 3 | 100% |
| 5529 | 1 | 6 | 1 | 3 | 1 | 3 | 100% |

Table A.20: Non-destructive Results for L11 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 5 | 5 | 100% |
| 1010 | 9 | 5 | 100% |
| 2000 | 8 | 5 | 100% |
| 2500 | 1 | 45 | 100% |
| 3000 | 9 | 3 | 100% |
| 3007 | 4 | 8 | 100% |
| 3500 | 4 | 22 | 100% |
| 4000 | 8 | 9 | 100% |
| 4021 | 1 | 2 | 100% |
| 5529 | 10 | 9 | 100% |

Table A.21: Standard Operator Results for L12 (DFA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 6 | 38 | 1 | 21 | 1 | 9 | 100% |
| 1010 | 3 | 22 | 1 | 5 | 2 | 35 | 100% |
| 2000 | 4 | 46 | 1 | 11 | 1 | 6 | 100% |
| 2500 | 3 | 36 | 1 | 7 | 1 | 18 | 100% |
| 3000 | 1 | 3 | 1 | 9 | 2 | 13 | 100% |
| 3007 | 3 | 7 | 2 | 17 | 1 | 12 | 100% |
| 3500 | 6 | 5 | 2 | 26 | 2 | 3 | 100% |
| 4000 | 6 | 17 | 2 | 3 | 1 | 13 | 100% |
| 4021 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 5529 | 5 | 6 | 1 | 21 | 1 | 6 | 100% |

Table A.22: Non-destructive Results for L12 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 1 | 100% |
| 1010 | 1 | 1 | 100% |
| 2000 | 1 | 1 | 100% |
| 2500 | 1 | 1 | 100% |
| 3000 | 1 | 1 | 100% |
| 3007 | 1 | 1 | 100% |
| 3500 | 1 | 1 | 100% |
| 4000 | 1 | 1 | 100% |
| 4021 | 1 | 1 | 100% |
| 5529 | 1 | 1 | 100% |

Table A.23: Standard Operator Results for L13 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 3 | 100% |
| 1010 | 4 | 5 | 100% |
| 2000 | 1 | 3 | 100% |
| 2500 | 5 | 4 | 100% |
| 3000 | 3 | 3 | 100% |
| 3007 | 3 | 7 | 100% |
| 3500 | 4 | 5 | 100% |
| 4000 | 1 | 3 | 100% |
| 4021 | 1 | 5 | 100% |
| 5529 | 1 | 4 | 100% |

Table A.24: Standard Operator Results for L14 (DFA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 4 | 1 | 4 | 1 | 4 | 100% |
| 1010 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 2000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 2500 | 1 | 44 | 1 | 3 | 2 | 4 | 100% |
| 3000 | 3 | 5 | 2 | 3 | 2 | 3 | 100% |
| 3007 | 1 | 21 | 1 | 4 | 1 | 21 | 100% |
| 3500 | 2 | 4 | 2 | 4 | 1 | 4 | 100% |
| 4000 | 1 | 7 | 1 | 2 | 1 | 2 | 100% |
| 4021 | 2 | 40 | 1 | 48 | 1 | 3 | 100% |
| 5529 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |

Table A.25: Non-destructive Results for L14 (DFA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 98.3% |
| 1010 | 1 | 33 | 100% |
| 2000 | - | - | 97.6% |
| 2500 | - | - | 96.6% |
| 3000 | 1 | 15 | 100% |
| 3007 | - | - | 97.6% |
| 3500 | - | - | 97.2% |
| 4000 | 13 | 13 | 100% |
| 4021 | - | - | 97.2% |
| 5529 | - | - | 96.6% |

Table A.26: Standard Operator Results for L15 (DFA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | - | - | - | - | - | - | 98.6% |
| 1010 | - | - | - | - | - | - | 98% |
| 2000 | - | - | 8 | 6 | - | - | 100% |
| 2500 | 13 | 25 | - | - | - | - | 100% |
| 3000 | 1 | 10 | - | - | - | - | 100% |
| 3007 | - | - | - | - | 3 | 40 | 100% |
| 3500 | - | - | - | - | - | - | 99% |
| 4000 | 2 | 35 | - | - | - | - | 100% |
| 4021 | - | - | - | - | - | - | 99% |
| 5529 | - | - | - | - | - | - | 99.7% |

Table A.27: Non-destructive Results for L15 (DFA)

A.2 Non-deterministic Finite Acceptors (1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 1 | 100% |
| 1010 | 1 | 1 | 100% |
| 2000 | 1 | 1 | 100% |
| 2500 | 1 | 1 | 100% |
| 3000 | 1 | 1 | 100% |
| 3007 | 1 | 1 | 100% |
| 3500 | 1 | 1 | 100% |
| 4000 | 1 | 1 | 100% |
| 4021 | 1 | 1 | 100% |
| 5529 | 1 | 1 | 100% |

Table A.28: Standard Operator Results for L1 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 1 | 100% |
| 1010 | 1 | 1 | 100% |
| 2000 | 1 | 1 | 100% |
| 2500 | 1 | 1 | 100% |
| 3000 | 1 | 1 | 100% |
| 3007 | 1 | 1 | 100% |
| 3500 | 1 | 1 | 100% |
| 4000 | 1 | 1 | 100% |
| 4021 | 1 | 1 | 100% |
| 5529 | 1 | 1 | 100% |

Table A.29: Standard Operator Results for L2 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 20 | 100% |
| 1010 | 1 | 6 | 100% |
| 2000 | 1 | 7 | 100% |
| 2500 | 3 | 42 | 100% |
| 3000 | 2 | 37 | 100% |
| 3007 | 3 | 7 | 100% |
| 3500 | 2 | 8 | 100% |
| 4000 | 1 | 32 | 100% |
| 4021 | 1 | 8 | 100% |
| 5529 | 1 | 7 | 100% |

Table A.30: Standard Operator Results for L3 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 10 | 1 | 3 | 1 | 4 | 100% |
| 1010 | 1 | 6 | 1 | 4 | 1 | 4 | 100% |
| 2000 | 1 | 7 | 1 | 10 | 1 | 5 | 100% |
| 2500 | 1 | 4 | 1 | 6 | 1 | 4 | 100% |
| 3000 | 1 | 7 | 1 | 4 | 1 | 4 | 100% |
| 3007 | 2 | 23 | 1 | 3 | 1 | 3 | 100% |
| 3500 | 2 | 6 | 1 | 5 | 1 | 3 | 100% |
| 4000 | 3 | 8 | 1 | 5 | 1 | 7 | 100% |
| 4021 | 1 | 12 | 1 | 4 | 1 | 4 | 100% |
| 5529 | 1 | 8 | 1 | 4 | 1 | 4 | 100% |

Table A.31: Non-destructive Results for L3 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 4 | 100% |
| 1010 | 1 | 3 | 100% |
| 2000 | 1 | 5 | 100% |
| 2500 | 1 | 3 | 100% |
| 3000 | 1 | 5 | 100% |
| 3007 | 1 | 4 | 100% |
| 3500 | 1 | 3 | 100% |
| 4000 | 1 | 2 | 100% |
| 4021 | 1 | 4 | 100% |
| 5529 | 1 | 3 | 100% |

Table A.32: Standard Operator Results for L4 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 1010 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 2000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 2500 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 3000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 3007 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 3500 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 4000 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 4021 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 5529 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |

Table A.33: Non-destructive Results for L4 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 90% |
| 1010 | - | - | 90% |
| 2000 | 8 | 21 | 100% |
| 2500 | - | - | 84.5% |
| 3000 | - | - | 86.4% |
| 3007 | - | - | 90% |
| 3500 | 8 | 20 | 100% |
| 4000 | - | - | 91.8% |
| 4021 | - | - | 90% |
| 5529 | 5 | 30 | 100% |

Table A.34: Standard Operator Results for L5 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 8 | 32 | 8 | 11 | 5 | 3 | 100% |
| 1010 | - | - | 2 | 9 | 6 | 4 | 100% |
| 2000 | 3 | 24 | 1 | 4 | 1 | 3 | 100% |
| 2500 | 10 | 47 | 1 | 32 | 5 | 33 | 100% |
| 3000 | 9 | 50 | 2 | 7 | 4 | 43 | 100% |
| 3007 | 3 | 7 | 1 | 11 | 1 | 25 | 100% |
| 3500 | 10 | 32 | 2 | 23 | 6 | 6 | 100% |
| 4000 | 7 | 6 | 4 | 27 | 1 | 34 | 100% |
| 4021 | 9 | 5 | 1 | 8 | 1 | 10 | 100% |
| 5529 | 1 | 49 | 2 | 41 | 5 | 24 | 100% |

Table A.35: Non-destructive Results for L5 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 2 | 4 | 100% |
| 1010 | 6 | 23 | 100% |
| 2000 | 2 | 13 | 100% |
| 2500 | 1 | 12 | 100% |
| 3000 | 3 | 22 | 100% |
| 3007 | 6 | 10 | 100% |
| 3500 | 1 | 6 | 100% |
| 4000 | 4 | 40 | 100% |
| 4021 | 4 | 8 | 100% |
| 5529 | - | - | 84.8% |

Table A.36: Standard Operator Results for L6 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 2 | 6 | 1 | 10 | 1 | 46 | 100% |
| 1010 | 3 | 3 | 2 | 4 | 1 | 5 | 100% |
| 2000 | 3 | 4 | 1 | 3 | 1 | 4 | 100% |
| 2500 | 1 | 47 | 1 | 4 | 1 | 5 | 100% |
| 3000 | 3 | 51 | 1 | 5 | 1 | 4 | 100% |
| 3007 | 2 | 5 | 1 | 5 | 1 | 4 | 100% |
| 3500 | 1 | 8 | 1 | 3 | 1 | 3 | 100% |
| 4000 | 1 | 13 | 1 | 8 | 1 | 3 | 100% |
| 4021 | 4 | 22 | 1 | 4 | 1 | 4 | 100% |
| 5529 | 1 | 7 | 1 | 6 | 1 | 4 | 100% |

Table A.37: Non-destructive Results for L6 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 3 | 100% |
| 1010 | 1 | 5 | 100% |
| 2000 | 1 | 4 | 100% |
| 2500 | 1 | 5 | 100% |
| 3000 | 1 | 6 | 100% |
| 3007 | 1 | 3 | 100% |
| 3500 | 1 | 4 | 100% |
| 4000 | 1 | 3 | 100% |
| 4021 | 1 | 4 | 100% |
| 5529 | 1 | 5 | 100% |

Table A.38: Standard Operator Results for L7 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 1010 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 2000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 2500 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 3000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 3007 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 3500 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 4000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 4021 | 1 | 5 | 1 | 3 | 1 | 3 | 100% |
| 5529 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |

Table A.39: Non-destructive Results for L7 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 1 | 100% |
| 1010 | 1 | 1 | 100% |
| 2000 | 1 | 1 | 100% |
| 2500 | 1 | 1 | 100% |
| 3000 | 1 | 1 | 100% |
| 3007 | 1 | 1 | 100% |
| 3500 | 1 | 1 | 100% |
| 4000 | 1 | 1 | 100% |
| 4021 | 1 | 1 | 100% |
| 5529 | 1 | 1 | 100% |

Table A.40: Standard Operator Results for L8 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 6 | 7 | 100% |
| 1010 | - | - | 98.1% |
| 2000 | - | - | 99.5% |
| 2500 | - | - | 97.7% |
| 3000 | 16 | 4 | 100% |
| 3007 | 2 | 6 | 100% |
| 3500 | - | - | 98.1% |
| 4000 | - | - | 98.1% |
| 4021 | - | - | 98.1% |
| 5529 | - | - | 98.1% |

Table A.41: Standard Operator Results for L9 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | - | - | 6 | 4 | 9 | 3 | 100% |
| 1010 | 5 | 3 | 2 | 3 | 2 | 3 | 100% |
| 2000 | 16 | 13 | 2 | 4 | 2 | 5 | 100% |
| 2500 | 3 | 6 | 9 | 4 | 9 | 3 | 100% |
| 3000 | 3 | 4 | 5 | 3 | 4 | 5 | 100% |
| 3007 | 5 | 5 | 6 | 3 | 9 | 4 | 100% |
| 3500 | 6 | 18 | - | - | 1 | 5 | 100% |
| 4000 | - | - | 1 | 4 | 4 | 19 | 100% |
| 4021 | - | - | 3 | 3 | 5 | 7 | 100% |
| 5529 | 16 | 5 | 3 | 4 | 4 | 4 | 100% |

Table A.42: Non-destructive Results for L9 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 8 | 21 | 100% |
| 1010 | 3 | 17 | 100% |
| 2000 | - | - | 98.3% |
| 2500 | 5 | 18 | 100% |
| 3000 | 1 | 20 | 100% |
| 3007 | 7 | 10 | 100% |
| 3500 | 3 | 13 | 100% |
| 4000 | 9 | 17 | 100% |
| 4021 | 1 | 14 | 100% |
| 5529 | 2 | 15 | 100% |

Table A.43: Standard Operator Results for L10 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 10 | 1 | 4 | 1 | 5 | 100% |
| 1010 | 1 | 10 | 1 | 4 | 1 | 4 | 100% |
| 2000 | 1 | 27 | 1 | 6 | 1 | 3 | 100% |
| 2500 | 1 | 14 | 1 | 3 | 1 | 3 | 100% |
| 3000 | 1 | 10 | 1 | 4 | 1 | 4 | 100% |
| 3007 | 1 | 8 | 3 | 4 | 3 | 4 | 100% |
| 3500 | 1 | 16 | 1 | 4 | 1 | 4 | 100% |
| 4000 | 2 | 12 | 1 | 10 | 1 | 5 | 100% |
| 4021 | 1 | 30 | 1 | 14 | 1 | 7 | 100% |
| 5529 | 1 | 9 | 1 | 3 | 1 | 3 | 100% |

Table A.44: Non-destructive Results for L10 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 83.3% |
| 1010 | 7 | 36 | 100% |
| 2000 | - | - | 83.3% |
| 2500 | - | - | 83.3% |
| 3000 | - | - | 85.7% |
| 3007 | - | - | 85.7% |
| 3500 | - | - | 83.3% |
| 4000 | - | - | 83.3% |
| 4021 | - | - | 83.3% |
| 5529 | 8 | 51 | 100% |

Table A.45: Standard Operator Results for L11 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | - | - | 4 | 4 | 1 | 6 | 100% |
| 1010 | 1 | 11 | 5 | 4 | 5 | 4 | 100% |
| 2000 | - | - | 1 | 6 | 1 | 5 | 100% |
| 2500 | - | - | 10 | 5 | 1 | 12 | 100% |
| 3000 | - | - | 3 | 9 | 4 | 19 | 100% |
| 3007 | 9 | 34 | 2 | 5 | 2 | 4 | 100% |
| 3500 | - | - | 1 | 36 | 1 | 4 | 100% |
| 4000 | 6 | 11 | 2 | 5 | 2 | 21 | 100% |
| 4021 | - | - | 2 | 4 | 2 | 5 | 100% |
| 5529 | - | - | 1 | 6 | 1 | 6 | 100% |

Table A.46: Non-destructive Results for L11 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 1 | 100% |
| 1010 | 1 | 8 | 100% |
| 2000 | 1 | 1 | 100% |
| 2500 | 1 | 2 | 100% |
| 3000 | 1 | 3 | 100% |
| 3007 | 1 | 5 | 100% |
| 3500 | 1 | 7 | 100% |
| 4000 | 1 | 2 | 100% |
| 4021 | 1 | 2 | 100% |
| 5529 | 1 | 2 | 100% |

Table A.47: Standard Operator Results for L12 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 1010 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 2000 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 2500 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 3000 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 3007 | 1 | 10 | 1 | 3 | 1 | 3 | 100% |
| 3500 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 4000 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 4021 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 5529 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |

Table A.48: Non-destructive Results for L12 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 3 | 100% |
| 1010 | 1 | 2 | 100% |
| 2000 | 1 | 1 | 100% |
| 2500 | 1 | 3 | 100% |
| 3000 | 1 | 1 | 100% |
| 3007 | 1 | 2 | 100% |
| 3500 | 1 | 3 | 100% |
| 4000 | 1 | 1 | 100% |
| 4021 | 1 | 1 | 100% |
| 5529 | 1 | 2 | 100% |

Table A.49: Standard Operator Results for L13 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 1010 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 2000 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 2500 | 1 | 3 | 1 | 3 | 1 | 2 | 100% |
| 3000 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 3007 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 3500 | 1 | 3 | 1 | 3 | 1 | 2 | 100% |
| 4000 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 4021 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 5529 | 1 | 2 | 1 | 3 | 1 | 2 | 100% |

Table A.50: Non-destructive Results for L13 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 3 | 100% |
| 1010 | 1 | 7 | 100% |
| 2000 | 1 | 9 | 100% |
| 2500 | 1 | 4 | 100% |
| 3000 | 1 | 14 | 100% |
| 3007 | 1 | 4 | 100% |
| 3500 | 1 | 32 | 100% |
| 4000 | 1 | 31 | 100% |
| 4021 | 4 | 4 | 100% |
| 5529 | 8 | 10 | 100% |

Table A.51: Standard Operator Results for L14 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 5 | 1 | 2 | 1 | 2 | 100% |
| 1010 | 1 | 2 | 1 | 3 | 1 | 3 | 100% |
| 2000 | 1 | 13 | 1 | 3 | 1 | 3 | 100% |
| 2500 | 1 | 10 | 1 | 2 | 1 | 2 | 100% |
| 3000 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 3007 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 3500 | 1 | 35 | 1 | 3 | 1 | 3 | 100% |
| 4000 | 1 | 18 | 1 | 3 | 1 | 3 | 100% |
| 4021 | 1 | 6 | 1 | 4 | 1 | 4 | 100% |
| 5529 | 1 | 4 | 1 | 2 | 1 | 2 | 100% |

Table A.52: Non-destructive Results for L14 (NFA1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 5 | 46 | 100% |
| 1010 | - | - | 97.7% |
| 2000 | - | - | 98.9% |
| 2500 | - | - | 99.8% |
| 3000 | 12 | 5 | 100% |
| 3007 | - | - | 99.3% |
| 3500 | - | - | 97.7% |
| 4000 | - | - | 99.1% |
| 4021 | 12 | 30 | 100% |
| 5529 | - | - | 99.1% |

Table A.53: Standard Operator Results for L15 (NFA1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | - | - | 10 | 8 | 2 | 6 | 100% |
| 1010 | 14 | 6 | 1 | 7 | 1 | 4 | 100% |
| 2000 | - | - | 2 | 6 | 4 | 5 | 100% |
| 2500 | - | - | 13 | 3 | 2 | 3 | 100% |
| 3000 | - | - | 8 | 17 | 2 | 7 | 100% |
| 3007 | - | - | 9 | 6 | 10 | 3 | 100% |
| 3500 | 9 | 21 | 8 | 5 | 1 | 6 | 100% |
| 4000 | 2 | 11 | 5 | 3 | 6 | 4 | 100% |
| 4021 | 8 | 8 | 8 | 4 | 9 | 8 | 100% |
| 5529 | 4 | 8 | - | - | 2 | 5 | 100% |

Table A.54: Non-destructive Results for L15 (NFA1)

A.3 Non-deterministic Finite Acceptors (2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 1 | 100% |
| 1010 | 1 | 1 | 100% |
| 2000 | 1 | 1 | 100% |
| 2500 | 1 | 1 | 100% |
| 3000 | 1 | 1 | 100% |
| 3007 | 1 | 1 | 100% |
| 3500 | 1 | 1 | 100% |
| 4000 | 1 | 1 | 100% |
| 4021 | 1 | 1 | 100% |
| 5529 | 1 | 1 | 100% |

Table A.55: Standard Operator Results for L1 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 1 | 100% |
| 1010 | 1 | 1 | 100% |
| 2000 | 1 | 1 | 100% |
| 2500 | 1 | 1 | 100% |
| 3000 | 1 | 1 | 100% |
| 3007 | 1 | 1 | 100% |
| 3500 | 1 | 1 | 100% |
| 4000 | 1 | 1 | 100% |
| 4021 | 1 | 1 | 100% |
| 5529 | 1 | 1 | 100% |

Table A.56: Standard Operator Results for L2 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 4 | 9 | 100% |
| 1010 | 3 | 16 | 100% |
| 2000 | 3 | 9 | 100% |
| 2500 | 1 | 30 | 100% |
| 3000 | 6 | 10 | 100% |
| 3007 | 1 | 8 | 100% |
| 3500 | 1 | 13 | 100% |
| 4000 | 1 | 16 | 100% |
| 4021 | 3 | 9 | 100% |
| 5529 | 3 | 15 | 100% |

Table A.57: Standard Operator Results for L3 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 9 | 1 | 4 | 1 | 6 | 100% |
| 1010 | 1 | 9 | 1 | 4 | 1 | 4 | 100% |
| 2000 | 1 | 12 | 1 | 4 | 1 | 4 | 100% |
| 2500 | 3 | 8 | 1 | 5 | 1 | 5 | 100% |
| 3000 | 2 | 12 | 1 | 5 | 1 | 5 | 100% |
| 3007 | 2 | 7 | 1 | 5 | 1 | 5 | 100% |
| 3500 | 3 | 10 | 1 | 4 | 2 | 4 | 100% |
| 4000 | 1 | 7 | 1 | 4 | 1 | 4 | 100% |
| 4021 | 1 | 11 | 1 | 5 | 1 | 5 | 100% |
| 5529 | 1 | 19 | 1 | 4 | 1 | 4 | 100% |

Table A.58: Non-destructive Results for L3 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 3 | 100% |
| 1010 | 1 | 4 | 100% |
| 2000 | 1 | 2 | 100% |
| 2500 | 1 | 3 | 100% |
| 3000 | 1 | 1 | 100% |
| 3007 | 1 | 2 | 100% |
| 3500 | 1 | 3 | 100% |
| 4000 | 1 | 2 | 100% |
| 4021 | 1 | 3 | 100% |
| 5529 | 1 | 3 | 100% |

Table A.59: Standard Operator Results for L4 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 1010 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 2000 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 2500 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 3000 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 3007 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 3500 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 4000 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 4021 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 5529 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |

Table A.60: Non-destructive Results for L4 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 81.2% |
| 1010 | - | - | 85.5% |
| 2000 | - | - | 81.2% |
| 2500 | - | - | 88.5% |
| 3000 | - | - | 81.2% |
| 3007 | - | - | 95.2% |
| 3500 | - | - | 80% |
| 4000 | - | - | 93.3% |
| 4021 | - | - | 86.1% |
| 5529 | - | - | 86.1% |

Table A.61: Standard Operator Results for L5 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | - | - | - | - | 10 | 30 | 100% |
| 1010 | 3 | 8 | 2 | 29 | 3 | 6 | 100% |
| 2000 | 7 | 7 | 2 | 5 | 3 | 8 | 100% |
| 2500 | - | - | 1 | 6 | 2 | 39 | 100% |
| 3000 | - | - | 6 | 5 | 1 | 29 | 100% |
| 3007 | - | - | 1 | 7 | 1 | 16 | 100% |
| 3500 | - | - | 7 | 14 | 3 | 5 | 100% |
| 4000 | - | - | - | - | 5 | 5 | 100% |
| 4021 | 7 | 11 | 2 | 7 | 2 | 7 | 100% |
| 5529 | - | - | 2 | 15 | 3 | 33 | 100% |

Table A.62: Non-destructive Results for L5 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 2 | 18 | 100% |
| 1010 | 3 | 39 | 100% |
| 2000 | 3 | 10 | 100% |
| 2500 | - | - | 85.5% |
| 3000 | - | - | 81.1% |
| 3007 | 7 | 6 | 100% |
| 3500 | 7 | 12 | 100% |
| 4000 | - | - | 88.7% |
| 4021 | 1 | 43 | 100% |
| 5529 | 7 | 8 | 100% |

Table A.63: Standard Operator Results for L6 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 5 | 44 | 1 | 7 | 1 | 3 | 100% |
| 1010 | 2 | 15 | 2 | 4 | 2 | 5 | 100% |
| 2000 | 9 | 5 | 1 | 6 | 1 | 6 | 100% |
| 2500 | 4 | 9 | 1 | 4 | 1 | 4 | 100% |
| 3000 | 1 | 10 | 1 | 4 | 2 | 4 | 100% |
| 3007 | 2 | 6 | 2 | 4 | 1 | 18 | 100% |
| 3500 | 3 | 49 | 2 | 4 | 2 | 6 | 100% |
| 4000 | 2 | 8 | 3 | 4 | 1 | 5 | 100% |
| 4021 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 5529 | 1 | 6 | 1 | 7 | 1 | 5 | 100% |

Table A.64: Non-destructive Results for L6 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 3 | 100% |
| 1010 | 1 | 5 | 100% |
| 2000 | 1 | 4 | 100% |
| 2500 | 1 | 4 | 100% |
| 3000 | 1 | 4 | 100% |
| 3007 | 2 | 4 | 100% |
| 3500 | 1 | 4 | 100% |
| 4000 | 1 | 4 | 100% |
| 4021 | 1 | 3 | 100% |
| 5529 | 1 | 3 | 100% |

Table A.65: Standard Operator Results for L7 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 1010 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 2000 | 1 | 5 | 1 | 3 | 1 | 3 | 100% |
| 2500 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 3000 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 3007 | 1 | 5 | 1 | 3 | 1 | 3 | 100% |
| 3500 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 4000 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 4021 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 5529 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |

Table A.66: Non-destructive Results for L7 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 1 | 100% |
| 1010 | 1 | 1 | 100% |
| 2000 | 1 | 1 | 100% |
| 2500 | 1 | 1 | 100% |
| 3000 | 1 | 1 | 100% |
| 3007 | 1 | 1 | 100% |
| 3500 | 1 | 1 | 100% |
| 4000 | 1 | 1 | 100% |
| 4021 | 1 | 1 | 100% |
| 5529 | 1 | 1 | 100% |

Table A.67: Standard Operator Results for L8 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 9 | 5 | 100% |
| 1010 | - | - | 99.5% |
| 2000 | - | - | 98.2% |
| 2500 | - | - | 98.2% |
| 3000 | - | - | 98.6% |
| 3007 | - | - | 98.2% |
| 3500 | - | - | 98.2% |
| 4000 | - | - | 98.2% |
| 4021 | - | - | 97.7% |
| 5529 | 6 | 6 | 100% |

Table A.68: Standard Operator Results for L9 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | - | - | - | - | - | - | 99.5% |
| 1010 | - | - | 20 | 18 | 13 | 4 | 100% |
| 2000 | - | - | 13 | 3 | - | - | 100% |
| 2500 | - | - | 2 | 3 | 4 | 20 | 100% |
| 3000 | - | - | - | - | 2 | 21 | 100% |
| 3007 | - | - | - | - | 13 | 3 | 100% |
| 3500 | 5 | 5 | 1 | 4 | 2 | 6 | 100% |
| 4000 | - | - | 14 | 6 | 15 | 6 | 100% |
| 4021 | - | - | 1 | 18 | 6 | 6 | 100% |
| 5529 | - | - | 3 | 8 | 4 | 3 | 100% |

Table A.69: Non-destructive Results for L9 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 97.1% |
| 1010 | - | - | 94.6% |
| 2000 | - | - | 97.1% |
| 2500 | - | - | 95.8% |
| 3000 | - | - | 98.3% |
| 3007 | - | - | 98.3% |
| 3500 | - | - | 97.9% |
| 4000 | - | - | 98.3% |
| 4021 | - | - | 98.3% |
| 5529 | - | - | 98.3% |

Table A.70: Standard Operator Results for L10 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | - | - | - | - | 8 | 4 | 100% |
| 1010 | - | - | - | - | 6 | 6 | 100% |
| 2000 | - | - | 4 | 14 | 9 | 5 | 100% |
| 2500 | 2 | 34 | 4 | 5 | 3 | 5 | 100% |
| 3000 | - | - | - | - | 7 | 5 | 100% |
| 3007 | - | - | - | - | 2 | 5 | 100% |
| 3500 | 8 | 11 | - | - | 3 | 35 | 100% |
| 4000 | - | - | - | - | - | - | 99.6% |
| 4021 | - | - | - | - | 3 | 10 | 100% |
| 5529 | - | - | - | - | 4 | 5 | 100% |

Table A.71: Non-destructive Results for L10 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 83.3% |
| 1010 | - | - | 83.3% |
| 2000 | - | - | 83.3% |
| 2500 | - | - | 83.3% |
| 3000 | - | - | 83.3% |
| 3007 | - | - | 83.3% |
| 3500 | - | - | 83.3% |
| 4000 | - | - | 83.3% |
| 4021 | - | - | 83.3% |
| 5529 | 7 | 50 | 100% |

Table A.72: Standard Operator Results for L11 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | - | - | 1 | 7 | 2 | 4 | 100% |
| 1010 | - | - | 3 | 5 | 6 | 4 | 100% |
| 2000 | - | - | 4 | 8 | 4 | 6 | 100% |
| 2500 | - | - | 1 | 45 | 10 | 13 | 100% |
| 3000 | 6 | 7 | 6 | 4 | 2 | 47 | 100% |
| 3007 | - | - | 1 | 7 | 1 | 32 | 100% |
| 3500 | 6 | 16 | 7 | 8 | 4 | 20 | 100% |
| 4000 | - | - | 4 | 5 | 2 | 18 | 100% |
| 4021 | - | - | 3 | 5 | 2 | 5 | 100% |
| 5529 | - | - | 4 | 34 | 1 | 9 | 100% |

Table A.73: Non-destructive Results for L11 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 2 | 3 | 100% |
| 1010 | 1 | 6 | 100% |
| 2000 | 1 | 2 | 100% |
| 2500 | 1 | 5 | 100% |
| 3000 | 2 | 7 | 100% |
| 3007 | 1 | 7 | 100% |
| 3500 | 2 | 6 | 100% |
| 4000 | 1 | 2 | 100% |
| 4021 | 1 | 2 | 100% |
| 5529 | 3 | 24 | 100% |

Table A.74: Standard Operator Results for L12 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 2 | 1 | 3 | 1 | 2 | 100% |
| 1010 | 1 | 4 | 1 | 3 | 1 | 2 | 100% |
| 2000 | 1 | 2 | 1 | 2 | 1 | 1 | 100% |
| 2500 | 1 | 4 | 1 | 3 | 1 | 2 | 100% |
| 3000 | 1 | 3 | 1 | 3 | 1 | 1 | 100% |
| 3007 | 1 | 7 | 1 | 3 | 1 | 2 | 100% |
| 3500 | 1 | 3 | 1 | 2 | 1 | 1 | 100% |
| 4000 | 1 | 2 | 1 | 2 | 1 | 1 | 100% |
| 4021 | 1 | 2 | 1 | 2 | 1 | 1 | 100% |
| 5529 | 1 | 4 | 1 | 4 | 1 | 2 | 100% |

Table A.75: Non-destructive Results for L12 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 4 | 100% |
| 1010 | 1 | 3 | 100% |
| 2000 | 1 | 3 | 100% |
| 2500 | 1 | 4 | 100% |
| 3000 | 1 | 4 | 100% |
| 3007 | 1 | 3 | 100% |
| 3500 | 1 | 4 | 100% |
| 4000 | 1 | 4 | 100% |
| 4021 | 1 | 3 | 100% |
| 5529 | 1 | 4 | 100% |

Table A.76: Standard Operator Results for L13 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 1010 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 2000 | 1 | 2 | 1 | 3 | 1 | 3 | 100% |
| 2500 | 1 | 3 | 1 | 3 | 1 | 2 | 100% |
| 3000 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 3007 | 1 | 3 | 1 | 3 | 1 | 2 | 100% |
| 3500 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 4000 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 4021 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 5529 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |

Table A.77: Non-destructive Results for L13 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 1 | 18 | 100% |
| 1010 | 1 | 31 | 100% |
| 2000 | 3 | 2 | 100% |
| 2500 | 4 | 6 | 100% |
| 3000 | 2 | 3 | 100% |
| 3007 | 1 | 11 | 100% |
| 3500 | 2 | 12 | 100% |
| 4000 | 1 | 1 | 100% |
| 4021 | 1 | 7 | 100% |
| 5529 | 1 | 4 | 100% |

Table A.78: Standard Operator Results for L14 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 1 | 4 | 1 | 3 | 1 | 4 | 100% |
| 1010 | 1 | 37 | 1 | 4 | 1 | 4 | 100% |
| 2000 | 2 | 16 | 1 | 4 | 1 | 2 | 100% |
| 2500 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 3000 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 3007 | 1 | 10 | 1 | 4 | 1 | 4 | 100% |
| 3500 | 1 | 4 | 1 | 5 | 1 | 2 | 100% |
| 4000 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 4021 | 2 | 4 | 1 | 3 | 1 | 15 | 100% |
| 5529 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |

Table A.79: Non-destructive Results for L14 (NFA2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 98.9% |
| 1010 | 10 | 5 | 100% |
| 2000 | - | - | 99.7% |
| 2500 | - | - | 98.9% |
| 3000 | - | - | 97.2% |
| 3007 | - | - | 99.7% |
| 3500 | - | - | 98.6% |
| 4000 | - | - | 98.9% |
| 4021 | 15 | 10 | 100% |
| 5529 | - | - | 99.2% |

Table A.80: Standard Operator Results for L15 (NFA2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 2 | 10 | 4 | 15 | 2 | 4 | 100% |
| 1010 | - | - | 6 | 21 | 7 | 14 | 100% |
| 2000 | 14 | 16 | 3 | 3 | 5 | 10 | 100% |
| 2500 | 3 | 13 | 4 | 13 | 2 | 10 | 100% |
| 3000 | 19 | 7 | 6 | 3 | 2 | 4 | 100% |
| 3007 | 10 | 7 | 5 | 4 | 4 | 4 | 100% |
| 3500 | 8 | 7 | 5 | 4 | 1 | 4 | 100% |
| 4000 | 4 | 5 | 2 | 3 | 3 | 9 | 100% |
| 4021 | 6 | 12 | 10 | 8 | 1 | 37 | 100% |
| 5529 | 12 | 50 | 8 | 3 | 7 | 3 | 100% |

Table A.81: Non-destructive Results for L15 (NFA2)

A.4 Modular Approach for Deterministic Finite Acceptors

A.4.1 Modular Approach (a)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 97.1% |
| 1010 | - | - | 97.5% |
| 2000 | - | - | 98.3% |
| 2500 | - | - | 97.5% |
| 3000 | 2 | 4 | 100% |
| 3007 | - | - | 98.3% |
| 3500 | - | - | 98.3% |
| 4000 | - | - | 97.1% |
| 4021 | - | - | 98.3% |
| 5529 | - | - | 95.4% |

Table A.82: Standard Operator Results for L9 (DFA) - Modular Approach (a)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | - | - | 8 | 3 | - | - | 100% |
| 1010 | - | - | - | - | - | - | 97.9% |
| 2000 | - | - | 2 | 3 | 5 | 3 | 100% |
| 2500 | - | - | 7 | 3 | - | - | 100% |
| 3000 | - | - | - | - | 8 | 15 | 100% |
| 3007 | - | - | 2 | 3 | - | - | 100% |
| 3500 | - | - | 10 | 4 | 8 | 3 | 100% |
| 4000 | - | - | 1 | 19 | - | - | 100% |
| 4021 | - | - | - | - | 2 | 5 | 100% |
| 5529 | - | - | 1 | 3 | - | - | 100% |

Table A.83: Non-destructive Results for L9 (DFA) - Modular Approach (a)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 97.6% |
| 1010 | - | - | 99.7% |
| 2000 | - | - | 98% |
| 2500 | - | - | 99% |
| 3000 | - | - | 99% |
| 3007 | - | - | 96.6% |
| 3500 | - | - | 96.9% |
| 4000 | - | - | 97.6% |
| 4021 | - | - | 99.7% |
| 5529 | 4 | 14 | 100% |

Table A.84: Standard Operator Results for L15 (DFA) - Modular Approach (a)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 5 | 9 | 9 | 3 | 2 | 5 | 100% |
| 1010 | 1 | 45 | 7 | 11 | 5 | 5 | 100% |
| 2000 | 5 | 10 | 4 | 4 | 5 | 4 | 100% |
| 2500 | 10 | 9 | 8 | 6 | - | - | 100% |
| 3000 | 5 | 8 | 2 | 4 | 10 | 5 | 100% |
| 3007 | 1 | 12 | - | - | 3 | 27 | 100% |
| 3500 | - | - | - | - | 8 | 3 | 100% |
| 4000 | - | - | - | - | 1 | 3 | 100% |
| 4021 | 4 | 14 | - | - | 2 | 5 | 100% |
| 5529 | 6 | 48 | 5 | 38 | 2 | 5 | 100% |

Table A.85: Non-destructive Results for L15 (DFA) - Modular Approach (a)

A.4.2 Modular Approach (b)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 97.1% |
| 1010 | - | - | 97.9% |
| 2000 | - | - | 98.3% |
| 2500 | - | - | 98.3% |
| 3000 | - | - | 97.1% |
| 3007 | - | - | 97.1% |
| 3500 | - | - | 97.9% |
| 4000 | - | - | 97.9% |
| 4021 | - | - | 98.3% |
| 5529 | - | - | 97.9% |

Table A.86: Standard Operator Results for L9 (DFA) - Modular Approach (b)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 6 | 7 | 5 | 4 | 4 | 3 | 100% |
| 1010 | 6 | 24 | 6 | 6 | 1 | 5 | 100% |
| 2000 | 9 | 11 | 1 | 4 | 3 | 3 | 100% |
| 2500 | - | - | - | - | 5 | 4 | 100% |
| 3000 | 4 | 4 | 5 | 3 | 6 | 4 | 100% |
| 3007 | 5 | 7 | 8 | 4 | 6 | 4 | 100% |
| 3500 | 7 | 28 | 1 | 4 | 1 | 25 | 100% |
| 4000 | - | - | 1 | 11 | 6 | 5 | 100% |
| 4021 | 2 | 23 | 4 | 3 | 4 | 4 | 100% |
| 5529 | - | - | 8 | 4 | 2 | 4 | 100% |

Table A.87: Non-destructive Results for L9 (DFA) - Modular Approach (b)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 99.7% |
| 1010 | 8 | 51 | 100% |
| 2000 | 4 | 33 | 100% |
| 2500 | 2 | 17 | 100% |
| 3000 | 1 | 19 | 100% |
| 3007 | - | - | 99% |
| 3500 | 3 | 23 | 100% |
| 4000 | 2 | 6 | 100% |
| 4021 | 4 | 11 | 100% |
| 5529 | 10 | 34 | 100% |

Table A.88: Standard Operator Results for L15 (DFA) - Modular Approach (b)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 2 | 7 | 3 | 4 | 7 | 6 | 100% |
| 1010 | 7 | 45 | 7 | 4 | 1 | 13 | 100% |
| 2000 | 3 | 18 | 1 | 4 | 3 | 7 | 100% |
| 2500 | 1 | 6 | 4 | 4 | 2 | 4 | 100% |
| 3000 | - | - | 3 | 4 | 3 | 5 | 100% |
| 3007 | 8 | 8 | 7 | 3 | 7 | 6 | 100% |
| 3500 | 3 | 19 | 5 | 41 | 2 | 38 | 100% |
| 4000 | 7 | 11 | 2 | 15 | 1 | 10 | 100% |
| 4021 | 3 | 12 | 1 | 4 | 4 | 5 | 100% |
| 5529 | 2 | 10 | 5 | 5 | 2 | 5 | 100% |

Table A.89: Non-destructive Results for L15 (DFA) - Modular Approach (b)

A.4.3 Modular Approach (c)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | - | - | 98.3% |
| 1010 | 3 | 8 | 100% |
| 2000 | - | - | 97.9% |
| 2500 | - | - | 98.3% |
| 3000 | 6 | 5 | 100% |
| 3007 | 5 | 16 | 100% |
| 3500 | 6 | 42 | 100% |
| 4000 | - | - | 98.3% |
| 4021 | - | - | 97.9% |
| 5529 | - | - | 97.9% |

Table A.90: Standard Operator Results for L9 (DFA) - Modular Approach (c)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 10 | 6 | 2 | 4 | 1 | 6 | 100% |
| 1010 | 2 | 20 | 8 | 7 | 2 | 11 | 100% |
| 2000 | 5 | 14 | 5 | 26 | 4 | 4 | 100% |
| 2500 | 4 | 25 | 4 | 3 | 1 | 39 | 100% |
| 3000 | 2 | 5 | 8 | 4 | 7 | 3 | 100% |
| 3007 | 5 | 38 | 2 | 4 | 2 | 5 | 100% |
| 3500 | 8 | 4 | 3 | 3 | 2 | 4 | 100% |
| 4000 | - | - | 1 | 5 | 1 | 4 | 100% |
| 4021 | 2 | 14 | 4 | 4 | 2 | 4 | 100% |
| 5529 | 7 | 7 | 1 | 4 | 2 | 9 | 100% |

Table A.91: Non-destructive Results for L9 (DFA) - Modular Approach (c)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1000 | 4 | 5 | 100% |
| 1010 | 1 | 6 | 100% |
| 2000 | 3 | 10 | 100% |
| 2500 | 2 | 38 | 100% |
| 3000 | - | - | 98.6% |
| 3007 | - | - | 99.7% |
| 3500 | 10 | 6 | 100% |
| 4000 | - | - | 99.7% |
| 4021 | 1 | 23 | 100% |
| 5529 | - | - | 99.7% |

Table A.92: Standard Operator Results for L15 (DFA) - Modular Approach (c)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1000 | 8 | 11 | 1 | 5 | 4 | 3 | 100% |
| 1010 | 6 | 27 | 2 | 5 | 2 | 6 | 100% |
| 2000 | 4 | 8 | 4 | 5 | 2 | 3 | 100% |
| 2500 | 1 | 5 | 8 | 5 | 1 | 4 | 100% |
| 3000 | 3 | 7 | 3 | 4 | 3 | 4 | 100% |
| 3007 | 1 | 6 | 1 | 4 | 3 | 4 | 100% |
| 3500 | 7 | 7 | 6 | 4 | - | - | 100% |
| 4000 | 4 | 8 | 3 | 4 | 3 | 6 | 100% |
| 4021 | 5 | 11 | 1 | 4 | 4 | 4 | 100% |
| 5529 | 1 | 12 | 6 | 4 | 1 | 3 | 100% |

Table A.93: Non-destructive Results for L15 (DFA) - Modular Approach (c)

Appendix B

System Results for Finite State Transducers

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1580 | 1 | 5 | 100% |
| 2978 | 1 | 5 | 100% |
| 3478 | 3 | 1 | 100% |
| 3908 | 1 | 7 | 100% |
| 4600 | 1 | 8 | 100% |
| 5210 | 3 | 9 | 100% |
| 6541 | 1 | 7 | 100% |
| 6908 | 1 | 23 | 100% |
| 7500 | 1 | 7 | 100% |
| 8080 | 3 | 5 | 100% |

Table B.1: Standard Operator Results for L1 (FST)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1580 | 1 | 5 | 1 | 3 | 1 | 3 | 100% |
| 2978 | 1 | 6 | 1 | 3 | 1 | 3 | 100% |
| 3478 | 1 | 11 | 1 | 3 | 1 | 3 | 100% |
| 3908 | 1 | 7 | 1 | 3 | 1 | 3 | 100% |
| 4600 | 1 | 6 | 1 | 3 | 1 | 3 | 100% |
| 5210 | 1 | 6 | 1 | 3 | 1 | 3 | 100% |
| 6541 | 1 | 7 | 1 | 3 | 1 | 3 | 100% |
| 6908 | 1 | 7 | 1 | 3 | 1 | 3 | 100% |
| 7500 | 1 | 9 | 1 | 3 | 1 | 3 | 100% |
| 8080 | 1 | 6 | 1 | 3 | 1 | 3 | 100% |

Table B.2: Non-destructive Results for L1 (FST)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1580 | 1 | 5 | 100% |
| 2978 | 1 | 6 | 100% |
| 3478 | 1 | 6 | 100% |
| 3908 | 1 | 4 | 100% |
| 4600 | 1 | 5 | 100% |
| 5210 | 1 | 5 | 100% |
| 6541 | 1 | 5 | 100% |
| 6908 | 1 | 1 | 100% |
| 7500 | 1 | 4 | 100% |
| 8080 | 1 | 5 | 100% |

Table B.3: Standard Operator Results for L2 (FST)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1580 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 2978 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 3478 | 1 | 5 | 1 | 3 | 1 | 3 | 100% |
| 3908 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 4600 | 1 | 5 | 1 | 3 | 1 | 3 | 100% |
| 5210 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 6541 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 6908 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 7500 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 8080 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |

Table B.4: Non-destructive Results for L2 (FST)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1580 | 1 | 2 | 100% |
| 2978 | 1 | 1 | 100% |
| 3478 | 1 | 2 | 100% |
| 3908 | 1 | 2 | 100% |
| 4600 | 1 | 2 | 100% |
| 5210 | 1 | 1 | 100% |
| 6541 | 1 | 1 | 100% |
| 6908 | 1 | 1 | 100% |
| 7500 | 1 | 1 | 100% |
| 8080 | 1 | 1 | 100% |

Table B.5: Standard Operator Results for L3 (FST)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1580 | 1 | 2 | 100% |
| 2978 | 1 | 6 | 100% |
| 3478 | 1 | 4 | 100% |
| 3908 | 1 | 2 | 100% |
| 4600 | 1 | 41 | 100% |
| 5210 | 1 | 2 | 100% |
| 6541 | 1 | 2 | 100% |
| 6908 | 1 | 5 | 100% |
| 7500 | 1 | 6 | 100% |
| 8080 | 1 | 3 | 100% |

Table B.6: Standard Operator Results for L4 (FST)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1580 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 2978 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 3478 | 1 | 3 | 1 | 3 | 1 | 2 | 100% |
| 3908 | 1 | 2 | 1 | 3 | 1 | 3 | 100% |
| 4600 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 5210 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 6541 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 6908 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 7500 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 8080 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |

Table B.7: Non-destructive Results for L4 (FST)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1580 | 1 | 1 | 100% |
| 2978 | 1 | 1 | 100% |
| 3478 | 1 | 1 | 100% |
| 3908 | 1 | 1 | 100% |
| 4600 | 1 | 1 | 100% |
| 5210 | 1 | 1 | 100% |
| 6541 | 1 | 1 | 100% |
| 6908 | 1 | 1 | 100% |
| 7500 | 1 | 1 | 100% |
| 8080 | 1 | 1 | 100% |

Table B.8: Standard Operator Results for L5 (FST)

| Seed | Run | Generation | Best Fitness |
|-------------|------------|-------------------|---------------------|
| 1580 | 1 | 1 | 100% |
| 2978 | 1 | 1 | 100% |
| 3478 | 1 | 1 | 100% |
| 3908 | 1 | 1 | 100% |
| 4600 | 1 | 1 | 100% |
| 5210 | 1 | 1 | 100% |
| 6541 | 1 | 1 | 100% |
| 6908 | 1 | 1 | 100% |
| 7500 | 1 | 1 | 100% |
| 8080 | 1 | 1 | 100% |

Table B.9: Standard Operator Results for L6 (FST)

Appendix C

System Results for Pushdown Automata

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1097 | 1 | 5 | 100% |
| 1111 | 2 | 6 | 100% |
| 2121 | 1 | 4 | 100% |
| 2134 | 1 | 8 | 100% |
| 3003 | 1 | 4 | 100% |
| 3012 | 1 | 3 | 100% |
| 5678 | 1 | 8 | 100% |
| 7777 | 1 | 4 | 100% |
| 8530 | 1 | 1 | 100% |
| 9090 | 1 | 2 | 100% |

Table C.1: Standard Operator Results for L1 (PDA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1097 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 1111 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 2121 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 2134 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 3003 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 3012 | 1 | 2 | 1 | 3 | 1 | 3 | 100% |
| 5678 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 7777 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |
| 8530 | 1 | 1 | 1 | 1 | 1 | 1 | 100% |
| 9090 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |

Table C.2: Non-destructive Results for L1 (PDA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1097 | 1 | 4 | 100% |
| 1111 | 1 | 17 | 100% |
| 2121 | 1 | 47 | 100% |
| 2134 | 2 | 7 | 100% |
| 3003 | 1 | 7 | 100% |
| 3012 | 1 | 7 | 100% |
| 5678 | 1 | 50 | 100% |
| 7777 | 1 | 7 | 100% |
| 8530 | 1 | 18 | 100% |
| 9090 | 1 | 23 | 100% |

Table C.3: Standard Operator Results for L2 (PDA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1097 | 1 | 5 | 1 | 2 | 1 | 2 | 100% |
| 1111 | 1 | 9 | 1 | 3 | 1 | 3 | 100% |
| 2121 | 1 | 4 | 1 | 6 | 1 | 3 | 100% |
| 2134 | 1 | 6 | 1 | 6 | 1 | 3 | 100% |
| 3003 | 1 | 3 | 1 | 4 | 1 | 4 | 100% |
| 3012 | 1 | 4 | 1 | 4 | 1 | 4 | 100% |
| 5678 | 1 | 5 | 1 | 2 | 1 | 2 | 100% |
| 7777 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 8530 | 1 | 4 | 1 | 3 | 1 | 3 | 100% |
| 9090 | 1 | 4 | 1 | 4 | 1 | 5 | 100% |

Table C.4: Non-destructive Results for L2 (PDA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1097 | 16 | 14 | 100% |
| 1111 | - | - | 92.9% |
| 2121 | 5 | 19 | 100% |
| 2134 | - | - | 94.1% |
| 3003 | 8 | 34 | 100% |
| 3012 | 7 | 14 | 100% |
| 5678 | - | - | 94.1% |
| 7777 | - | - | 91.8% |
| 8530 | - | - | 95.3% |
| 9090 | - | - | 91.8% |

Table C.5: Standard Operator Results for L3 (PDA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1097 | 1 | 19 | 8 | 35 | 14 | 34 | 100% |
| 1111 | 2 | 17 | 4 | 10 | 14 | 23 | 100% |
| 2121 | 3 | 6 | 5 | 15 | 2 | 11 | 100% |
| 2134 | 11 | 8 | 2 | 49 | - | - | 100% |
| 3003 | 3 | 16 | 5 | 28 | - | - | 100% |
| 3012 | 4 | 6 | 17 | 21 | 9 | 3 | 100% |
| 5678 | 11 | 6 | 1 | 35 | 2 | 49 | 100% |
| 7777 | 15 | 11 | 4 | 27 | 3 | 27 | 100% |
| 8530 | 4 | 27 | 1 | 21 | 5 | 11 | 100% |
| 9090 | 1 | 12 | 3 | 30 | 4 | 34 | 100% |

Table C.6: Non-destructive Results for L3 (PDA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1097 | 3 | 43 | 100% |
| 1111 | 4 | 27 | 100% |
| 2121 | 6 | 49 | 100% |
| 2134 | 1 | 21 | 100% |
| 3003 | - | - | 96.9% |
| 3012 | 5 | 38 | 100% |
| 5678 | - | - | 94.3% |
| 7777 | - | - | 99% |
| 8530 | - | - | 95.1% |
| 9090 | 10 | 28 | 100% |

Table C.7: Standard Operator Results for L4 (PDA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1097 | 3 | 13 | - | - | 4 | 35 | 100% |
| 1111 | 2 | 40 | - | - | 5 | 48 | 100% |
| 2121 | 1 | 21 | 6 | 45 | 1 | 10 | 100% |
| 2134 | 2 | 12 | 6 | 35 | 2 | 28 | 100% |
| 3003 | 1 | 11 | 1 | 18 | 10 | 21 | 100% |
| 3012 | - | - | 3 | 17 | 7 | 8 | 100% |
| 5678 | 2 | 11 | 1 | 50 | 3 | 23 | 100% |
| 7777 | 5 | 17 | 7 | 5 | 5 | 5 | 100% |
| 8530 | - | - | - | - | - | - | 95% |
| 9090 | 1 | 6 | 4 | 21 | 8 | 9 | 100% |

Table C.8: Non-destructive Results for L4 (PDA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1097 | - | - | 93.9% |
| 1111 | - | - | 93.9% |
| 2121 | 2 | 18 | 100% |
| 2134 | 3 | 35 | 100% |
| 3003 | 10 | 19 | 100% |
| 3012 | - | - | 93.9% |
| 5678 | 6 | 35 | 100% |
| 7777 | - | - | 93.9% |
| 8530 | - | - | 93.3% |
| 9090 | 2 | 31 | 100% |

Table C.9: Standard Operator Results for L5 (PDA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1097 | 1 | 4 | 4 | 31 | 1 | 23 | 100% |
| 1111 | 3 | 11 | 1 | 20 | - | - | 100% |
| 2121 | 2 | 20 | 1 | 29 | 5 | 4 | 100% |
| 2134 | 2 | 4 | 7 | 37 | 8 | 11 | 100% |
| 3003 | 3 | 8 | 1 | 13 | 8 | 13 | 100% |
| 3012 | 1 | 11 | 3 | 12 | 3 | 12 | 100% |
| 5678 | 2 | 6 | 5 | 13 | 1 | 4 | 100% |
| 7777 | 7 | 24 | 2 | 24 | 3 | 3 | 100% |
| 8530 | 4 | 24 | 3 | 44 | 7 | 4 | 100% |
| 9090 | 1 | 22 | 1 | 4 | 1 | 3 | 100% |

Table C.10: Non-destructive Results for L5 (PDA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1097 | 4 | 48 | 100% |
| 1111 | 1 | 26 | 100% |
| 2121 | 4 | 49 | 100% |
| 2134 | 2 | 24 | 100% |
| 3003 | 5 | 9 | 100% |
| 3012 | 3 | 50 | 100% |
| 5678 | 2 | 20 | 100% |
| 7777 | 2 | 38 | 100% |
| 8530 | 1 | 8 | 100% |
| 9090 | 1 | 8 | 100% |

Table C.11: Standard Operator Results for L6 (PDA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1097 | 2 | 5 | 1 | 43 | 4 | 4 | 100% |
| 1111 | 1 | 9 | 5 | 10 | 6 | 17 | 100% |
| 2121 | 1 | 16 | 3 | 15 | 2 | 45 | 100% |
| 2134 | 1 | 5 | 1 | 4 | 1 | 3 | 100% |
| 3003 | 1 | 10 | 4 | 25 | 1 | 3 | 100% |
| 3012 | 1 | 6 | 10 | 39 | 3 | 6 | 100% |
| 5678 | 1 | 6 | 1 | 24 | 3 | 20 | 100% |
| 7777 | 3 | 8 | 3 | 5 | 1 | 4 | 100% |
| 8530 | 2 | 5 | 4 | 35 | 3 | 10 | 100% |
| 9090 | 1 | 4 | 3 | 16 | 1 | 5 | 100% |

Table C.12: Non-destructive Results for L6 (PDA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1097 | 1 | 25 | 100% |
| 1111 | 1 | 12 | 100% |
| 2121 | 1 | 19 | 100% |
| 2134 | 3 | 11 | 100% |
| 3003 | 1 | 27 | 100% |
| 3012 | 3 | 25 | 100% |
| 5678 | 1 | 22 | 100% |
| 7777 | 1 | 4 | 100% |
| 8530 | 2 | 9 | 100% |
| 9090 | 1 | 20 | 100% |

Table C.13: Standard Operator Results for L7 (PDA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1097 | 1 | 6 | 1 | 9 | 1 | 4 | 100% |
| 1111 | 1 | 7 | 1 | 3 | 1 | 3 | 100% |
| 2121 | 1 | 8 | 1 | 3 | 1 | 4 | 100% |
| 2134 | 1 | 6 | 1 | 3 | 1 | 3 | 100% |
| 3003 | 1 | 8 | 1 | 3 | 1 | 3 | 100% |
| 3012 | 1 | 3 | 1 | 7 | 1 | 3 | 100% |
| 5678 | 1 | 7 | 2 | 5 | 1 | 4 | 100% |
| 7777 | 1 | 5 | 1 | 14 | 2 | 4 | 100% |
| 8530 | 1 | 5 | 1 | 8 | 1 | 4 | 100% |
| 9090 | 1 | 7 | 1 | 3 | 1 | 3 | 100% |

Table C.14: Non-destructive Results for L7 (PDA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1097 | 1 | 27 | 100% |
| 1111 | 1 | 42 | 100% |
| 2121 | 1 | 19 | 100% |
| 2134 | 6 | 27 | 100% |
| 3003 | 1 | 13 | 100% |
| 3012 | 1 | 18 | 100% |
| 5678 | 1 | 9 | 100% |
| 7777 | 2 | 12 | 100% |
| 8530 | 2 | 19 | 100% |
| 9090 | 2 | 10 | 100% |

Table C.15: Standard Operator Results for L8 (PDA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1097 | 1 | 3 | 1 | 8 | 1 | 3 | 100% |
| 1111 | 1 | 5 | 1 | 20 | 1 | 4 | 100% |
| 2121 | 1 | 3 | 1 | 23 | 1 | 3 | 100% |
| 2134 | 1 | 4 | 1 | 6 | 1 | 4 | 100% |
| 3003 | 1 | 4 | 1 | 7 | 2 | 7 | 100% |
| 3012 | 1 | 4 | 1 | 5 | 1 | 3 | 100% |
| 5678 | 1 | 5 | 1 | 31 | 2 | 3 | 100% |
| 7777 | 1 | 11 | 1 | 13 | 1 | 9 | 100% |
| 8530 | 1 | 5 | 1 | 24 | 1 | 26 | 100% |
| 9090 | 1 | 3 | 1 | 17 | 1 | 14 | 100% |

Table C.16: Non-destructive Results for L8 (PDA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1097 | 1 | 15 | 100% |
| 1111 | 1 | 8 | 100% |
| 2121 | 1 | 13 | 100% |
| 2134 | 1 | 5 | 100% |
| 3003 | 1 | 9 | 100% |
| 3012 | 1 | 9 | 100% |
| 5678 | 1 | 5 | 100% |
| 7777 | 1 | 5 | 100% |
| 8530 | 1 | 10 | 100% |
| 9090 | 1 | 4 | 100% |

Table C.17: Standard Operator Results for L9 (PDA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1097 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 1111 | 1 | 2 | 1 | 3 | 1 | 2 | 100% |
| 2121 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 2134 | 1 | 2 | 1 | 3 | 1 | 3 | 100% |
| 3003 | 1 | 2 | 1 | 3 | 2 | 3 | 100% |
| 3012 | 1 | 3 | 1 | 5 | 1 | 3 | 100% |
| 5678 | 1 | 3 | 1 | 3 | 2 | 3 | 100% |
| 7777 | 1 | 2 | 1 | 3 | 1 | 2 | 100% |
| 8530 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 9090 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |

Table C.18: Non-destructive Results for L9 (PDA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1097 | 4 | 12 | 100% |
| 1111 | 1 | 37 | 100% |
| 2121 | 2 | 14 | 100% |
| 2134 | 2 | 50 | 100% |
| 3003 | 4 | 32 | 100% |
| 3012 | 1 | 21 | 100% |
| 5678 | 4 | 16 | 100% |
| 7777 | 2 | 24 | 100% |
| 8530 | 3 | 27 | 100% |
| 9090 | 5 | 20 | 100% |

Table C.19: Standard Operator Results for L10 (PDA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1097 | 1 | 8 | 1 | 16 | 1 | 5 | 100% |
| 1111 | 1 | 8 | 2 | 4 | 1 | 6 | 100% |
| 2121 | 1 | 10 | 1 | 28 | 1 | 3 | 100% |
| 2134 | 1 | 8 | 1 | 12 | 1 | 6 | 100% |
| 3003 | 1 | 7 | 1 | 9 | 1 | 10 | 100% |
| 3012 | 1 | 7 | 1 | 5 | 1 | 3 | 100% |
| 5678 | 1 | 7 | 1 | 4 | 1 | 11 | 100% |
| 7777 | 1 | 9 | 2 | 43 | 2 | 3 | 100% |
| 8530 | 1 | 8 | 1 | 4 | 1 | 4 | 100% |
| 9090 | 1 | 9 | 1 | 22 | 1 | 5 | 100% |

Table C.20: Non-destructive Results for L10 (PDA)

| Seed | Run | Generation | Best Fitness |
|-------------|------------|-------------------|---------------------|
| 1097 | 1 | 1 | 100% |
| 1111 | 1 | 1 | 100% |
| 2121 | 1 | 1 | 100% |
| 2134 | 1 | 1 | 100% |
| 3003 | 1 | 1 | 100% |
| 3012 | 1 | 1 | 100% |
| 5678 | 1 | 1 | 100% |
| 7777 | 1 | 1 | 100% |
| 8530 | 1 | 1 | 100% |
| 9090 | 1 | 1 | 100% |

Table C.21: Standard Operator Results for L11 (PDA)

Appendix D

System Results for Turing Machines

D.1 Turing Machine Acceptors

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 97.3% |
| 2000 | - | - | 97.5% |
| 3500 | - | - | 97.5% |
| 3900 | - | - | 97.8% |
| 4020 | - | - | 97.5% |
| 4850 | - | - | 97.8% |
| 5249 | - | - | 97.5% |
| 6300 | - | - | 97.5% |
| 8400 | - | - | 97.5% |
| 9077 | - | - | 97.5% |

Table D.1: Standard Operator Results for L1 (TMA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | - | - | 1 | 29 | 3 | 43 | 100% |
| 2000 | - | - | - | - | 10 | 45 | 100% |
| 3500 | 3 | 10 | 2 | 43 | 5 | 7 | 100% |
| 3900 | 1 | 45 | 8 | 50 | 1 | 6 | 100% |
| 4020 | 1 | 10 | 5 | 42 | - | - | 100% |
| 4850 | 1 | 25 | 3 | 49 | 1 | 22 | 100% |
| 5249 | - | - | - | - | 4 | 31 | 100% |
| 6300 | 6 | 5 | 1 | 19 | 3 | 13 | 100% |
| 8400 | 7 | 50 | 7 | 49 | 5 | 15 | 100% |
| 9077 | 1 | 4 | 7 | 5 | 1 | 3 | 100% |

Table D.2: Non-destructive Results for L1 (TMA)

| Seed | Run | Generation |
|-------------------|-----|------------|
| 3500 ¹ | 10 | 35 |
| 3900 | 10 | 16 |

Table D.3: Standard Operator Results for L1 (TMA) – Incremental

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | |
|------|---------------|------------|----------------|------------|----------------------|------------|
| | Run | Generation | Run | Generation | Run | Generation |
| 1500 | - | - | 10 | 40 | 7 | 8 |
| 2000 | 1 | 5 | - | - | 1 | 35 |
| 3500 | 1 | 8 | 2 | 5 | 5 | 9 |
| 3900 | 1 | 32 | 10 | 11 | 1 | 13 |
| 4020 | 6 | 3 | - | - | 3 | 38 |
| 4850 | 5 | 17 | - | - | 1 | 23 |
| 5249 | - | - | 5 | 50 | 3 | 7 |
| 6300 | - | - | 5 | 8 | 5 | 4 |
| 8400 | 2 | 5 | - | - | 1 | 7 |
| 9077 | 7 | 8 | 5 | 9 | 1 | 3 |

Table D.4: Non-destructive Results for L1 (TMA) – Incremental

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 98.4% |
| 2000 | - | - | 98.9% |
| 3500 | - | - | 98.4% |
| 3900 | - | - | 98.4% |
| 4020 | - | - | 98.4% |
| 4850 | - | - | 98.4% |
| 5249 | - | - | 98.4% |
| 6300 | - | - | 98.4% |
| 8400 | - | - | 98.4% |
| 9077 | - | - | 98.9% |

Table D.5: Standard Operator Results for L2 (TMA)

¹Only the seed values where solutions were found are shown in the tables with Incremental solutions.

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | - | - | - | - | - | - | 98.9% |
| 2000 | - | - | - | - | - | - | 98.9% |
| 3500 | - | - | - | - | - | - | 98.9% |
| 3900 | - | - | - | - | - | - | 98.9% |
| 4020 | - | - | - | - | - | - | 98.9% |
| 4850 | - | - | - | - | - | - | 98.9% |
| 5249 | - | - | - | - | - | - | 98.9% |
| 6300 | - | - | - | - | - | - | 98.9% |
| 8400 | - | - | - | - | - | - | 98.9% |
| 9077 | - | - | - | - | - | - | 98.9% |

Table D.6: Non-destructive Results for L2 (TMA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | |
|------|---------------|------------|----------------|------------|----------------------|------------|
| | Run | Generation | Run | Generation | Run | Generation |
| 1500 | - | - | - | - | 2 | 15 |
| 4850 | - | - | 3 | 42 | - | - |

Table D.7: Non-destructive Results for L2 (TMA) – Incremental

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | 1 | 12 | 100% |
| 2000 | 1 | 42 | 100% |
| 3500 | 6 | 42 | 100% |
| 3900 | 1 | 17 | 100% |
| 4020 | 6 | 29 | 100% |
| 4850 | 7 | 33 | 100% |
| 5249 | 1 | 28 | 100% |
| 6300 | 2 | 24 | 100% |
| 8400 | 2 | 7 | 100% |
| 9077 | 2 | 9 | 100% |

Table D.8: Standard Operator Results for L3 (TMA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | 1 | 6 | 1 | 4 | 1 | 4 | 100% |
| 2000 | 1 | 5 | 1 | 7 | 2 | 6 | 100% |
| 3500 | 1 | 10 | 1 | 6 | 1 | 5 | 100% |
| 3900 | 1 | 6 | 1 | 5 | 1 | 6 | 100% |
| 4020 | 2 | 5 | 2 | 44 | 1 | 4 | 100% |
| 4850 | 1 | 12 | 1 | 39 | 2 | 4 | 100% |
| 5249 | 1 | 25 | 1 | 43 | 1 | 5 | 100% |
| 6300 | 1 | 32 | 2 | 7 | 1 | 9 | 100% |
| 8400 | 1 | 8 | 2 | 4 | 1 | 5 | 100% |
| 9077 | 1 | 5 | 2 | 15 | 2 | 7 | 100% |

Table D.9: Non-destructive Results for L3 (TMA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 98.5% |
| 2000 | - | - | 98.1% |
| 3500 | - | - | 98.1% |
| 3900 | - | - | 98.1% |
| 4020 | - | - | 99.6% |
| 4850 | - | - | 98.1% |
| 5249 | - | - | 98.1% |
| 6300 | - | - | 98.1% |
| 8400 | - | - | 98.1% |
| 9077 | - | - | 98.5% |

Table D.10: Standard Operator Results for L4 (TMA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | - | - | - | - | - | - | 98.5% |
| 2000 | - | - | - | - | - | - | 98.5% |
| 3500 | - | - | - | - | - | - | 98.9% |
| 3900 | 3 | 32 | - | - | 9 | 10 | 100% |
| 4020 | - | - | - | - | - | - | 99.6% |
| 4850 | - | - | - | - | 1 | 16 | 100% |
| 5249 | - | - | - | - | - | - | 98.5% |
| 6300 | - | - | 6 | 34 | - | - | 100% |
| 8400 | - | - | - | - | - | - | 99.2% |
| 9077 | - | - | 1 | 37 | - | - | 100% |

Table D.11: Non-destructive Results for L4 (TMA)

| Seed | Run | Generation |
|------|-----|------------|
| 5249 | 2 | 27 |

Table D.12: Standard Operator Results for L4 (TMA) – Incremental

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | |
|------|---------------|------------|----------------|------------|----------------------|------------|
| | Run | Generation | Run | Generation | Run | Generation |
| 1500 | 5 | 7 | - | - | 2 | 18 |
| 2000 | - | - | 2 | 50 | 5 | 33 |
| 3500 | 9 | 25 | - | - | - | - |
| 3900 | - | - | - | - | 4 | 10 |
| 4020 | 10 | 11 | - | - | 7 | 18 |
| 4850 | - | - | - | - | - | - |
| 5249 | 4 | 12 | 4 | 29 | 2 | 11 |
| 6300 | - | - | - | - | 2 | 24 |
| 8400 | 1 | 13 | - | - | 1 | 26 |
| 9077 | 1 | 37 | 3 | 12 | - | - |

Table D.13: Non-destructive Results for L4 (TMA) – Incremental

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | 1 | 14 | 100% |
| 2000 | 7 | 10 | 100% |
| 3500 | 3 | 18 | 100% |
| 3900 | 10 | 47 | 100% |
| 4020 | 3 | 49 | 100% |
| 4850 | 1 | 30 | 100% |
| 5249 | 1 | 47 | 100% |
| 6300 | 1 | 37 | 100% |
| 8400 | 3 | 43 | 100% |
| 9077 | 5 | 39 | 100% |

Table D.14: Standard Operator Results for L5 (TMA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | 1 | 9 | 1 | 5 | 1 | 4 | 100% |
| 2000 | 2 | 4 | 1 | 9 | 1 | 8 | 100% |
| 3500 | 2 | 22 | 2 | 7 | 1 | 30 | 100% |
| 3900 | 2 | 6 | 1 | 22 | 1 | 5 | 100% |
| 4020 | 1 | 5 | 3 | 45 | 1 | 7 | 100% |
| 4850 | 1 | 19 | 2 | 10 | 1 | 51 | 100% |
| 5249 | 1 | 5 | 2 | 35 | 1 | 4 | 100% |
| 6300 | 1 | 7 | 1 | 5 | 2 | 6 | 100% |
| 8400 | 2 | 15 | 1 | 4 | 3 | 4 | 100% |
| 9077 | 1 | 6 | 1 | 26 | 1 | 4 | 100% |

Table D.15: Non-destructive Results for L5 (TMA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 80% |
| 2000 | - | - | 80% |
| 3500 | - | - | 80% |
| 3900 | - | - | 85% |
| 4020 | - | - | 80% |
| 4850 | - | - | 80% |
| 5249 | - | - | 80% |
| 6300 | - | - | 85% |
| 8400 | - | - | 85% |
| 9077 | - | - | 87.5% |

Table D.16: Standard Operator Results for L6 (TMA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | - | - | - | - | - | - | 95% |
| 2000 | - | - | - | - | - | - | 95% |
| 3500 | - | - | - | - | 1 | 10 | 100% |
| 3900 | - | - | - | - | - | - | 97.5% |
| 4020 | - | - | - | - | - | - | 95% |
| 4850 | - | - | - | - | - | - | 95% |
| 5249 | - | - | - | - | - | - | 97.5% |
| 6300 | - | - | - | - | - | - | 95% |
| 8400 | - | - | - | - | - | - | 97.5% |
| 9077 | - | - | - | - | - | - | 97.5% |

Table D.17: Non-destructive Results for L6 (TMA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 97.8% |
| 2000 | - | - | 97.8% |
| 3500 | - | - | 99.3% |
| 3900 | - | - | 97.4% |
| 4020 | - | - | 99.6% |
| 4850 | - | - | 97.8% |
| 5249 | - | - | 97.8% |
| 6300 | - | - | 97.8% |
| 8400 | - | - | 97.4% |
| 9077 | - | - | 97.8% |

Table D.18: Standard Operator Results for L7 (TMA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | - | - | 10 | 7 | 2 | 45 | 100% |
| 2000 | - | - | - | - | 2 | 46 | 100% |
| 3500 | - | - | 10 | 14 | 7 | 34 | 100% |
| 3900 | - | - | 6 | 19 | 1 | 26 | 100% |
| 4020 | 6 | 7 | 1 | 7 | 4 | 5 | 100% |
| 4850 | - | - | 7 | 37 | 1 | 39 | 100% |
| 5249 | - | - | - | - | 1 | 41 | 100% |
| 6300 | 8 | 19 | 4 | 28 | - | - | 100% |
| 8400 | 1 | 9 | 6 | 3 | 2 | 39 | 100% |
| 9077 | 1 | 7 | 1 | 9 | 1 | 4 | 100% |

Table D.19: Non-destructive Results for L7 (TMA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | 4 | 43 | 100% |
| 2000 | - | - | 88% |
| 3500 | - | - | 91.3% |
| 3900 | 3 | 20 | 100% |
| 4020 | - | - | 91.3% |
| 4850 | 3 | 39 | 100% |
| 5249 | 1 | 34 | 100% |
| 6300 | 4 | 10 | 100% |
| 8400 | 5 | 10 | 100% |
| 9077 | 1 | 23 | 100% |

Table D.20: Standard Operator Results for L8 (TMA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | 3 | 36 | 1 | 20 | 1 | 6 | 100% |
| 2000 | 1 | 35 | 2 | 3 | 1 | 11 | 100% |
| 3500 | 1 | 15 | 1 | 28 | 1 | 40 | 100% |
| 3900 | 2 | 22 | 1 | 16 | 2 | 4 | 100% |
| 4020 | 1 | 6 | 1 | 31 | 1 | 45 | 100% |
| 4850 | 1 | 20 | 1 | 6 | 3 | 8 | 100% |
| 5249 | 1 | 7 | 1 | 7 | 1 | 9 | 100% |
| 6300 | 3 | 10 | 1 | 11 | 1 | 48 | 100% |
| 8400 | 1 | 24 | 1 | 9 | 1 | 6 | 100% |
| 9077 | 1 | 7 | 1 | 23 | 1 | 28 | 100% |

Table D.21: Non-destructive Results for L8 (TMA)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 96.2% |
| 2000 | - | - | 96.2% |
| 3500 | - | - | 96.2% |
| 3900 | - | - | 96.2% |
| 4020 | - | - | 96.2% |
| 4850 | - | - | 96.2% |
| 5249 | - | - | 96.2% |
| 6300 | - | - | 96.2% |
| 8400 | - | - | 97.2% |
| 9077 | - | - | 96.2% |

Table D.22: Standard Operator Results for L9 (TMA)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | - | - | - | - | - | - | 97.2% |
| 2000 | - | - | - | - | - | - | 98.1% |
| 3500 | - | - | - | - | - | - | 97.2% |
| 3900 | - | - | - | - | 1 | 43 | 100% |
| 4020 | - | - | - | - | - | - | 98.1% |
| 4850 | - | - | - | - | - | - | 97.2% |
| 5249 | - | - | - | - | - | - | 98.1% |
| 6300 | - | - | - | - | - | - | 98.1% |
| 8400 | - | - | - | - | - | - | 97.2% |
| 9077 | - | - | - | - | - | - | 98.1% |

Table D.23: Non-destructive Results for L9 (TMA)

D.2 Turing Machine Transducers

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | 4 | 10 | 100% |
| 2000 | 9 | 13 | 100% |
| 3500 | - | - | 90% |
| 3900 | - | - | 60% |
| 4020 | - | - | 50% |
| 4850 | 1 | 42 | 100% |
| 5249 | 1 | 26 | 100% |
| 6300 | 3 | 18 | 100% |
| 8400 | 7 | 47 | 100% |
| 9077 | 3 | 18 | 100% |

Table D.24: Standard Operator Results for L1 (TMT1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | 2 | 21 | 2 | 12 | 1 | 30 | 100% |
| 2000 | 3 | 12 | 1 | 47 | 3 | 31 | 100% |
| 3500 | 2 | 4 | 1 | 47 | 2 | 3 | 100% |
| 3900 | 1 | 45 | 3 | 4 | 1 | 21 | 100% |
| 4020 | 2 | 6 | 2 | 8 | 1 | 37 | 100% |
| 4850 | 1 | 37 | 1 | 4 | 2 | 7 | 100% |
| 5249 | 1 | 9 | 3 | 3 | 1 | 25 | 100% |
| 6300 | 1 | 39 | 1 | 7 | 1 | 33 | 100% |
| 8400 | 1 | 6 | 1 | 47 | 1 | 5 | 100% |
| 9077 | 1 | 46 | 2 | 22 | 1 | 9 | 100% |

Table D.25: Non-destructive Results for L1 (TMT1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | 1 | 4 | 100% |
| 2000 | 1 | 2 | 100% |
| 3500 | 1 | 2 | 100% |
| 3900 | 1 | 6 | 100% |
| 4020 | 1 | 6 | 100% |
| 4850 | 1 | 9 | 100% |
| 5249 | 1 | 44 | 100% |
| 6300 | 1 | 7 | 100% |
| 8400 | 1 | 9 | 100% |
| 9077 | 1 | 7 | 100% |

Table D.26: Standard Operator Results for L2 (TMT1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | 1 | 2 | 1 | 3 | 1 | 2 | 100% |
| 2000 | 1 | 2 | 1 | 2 | 1 | 2 | 100% |
| 3500 | 1 | 2 | 1 | 3 | 1 | 3 | 100% |
| 3900 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 4020 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 4850 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 5249 | 1 | 3 | 1 | 3 | 1 | 3 | 100% |
| 6300 | 1 | 3 | 1 | 4 | 1 | 3 | 100% |
| 8400 | 1 | 3 | 1 | 3 | 1 | 2 | 100% |
| 9077 | 1 | 3 | 1 | 2 | 1 | 2 | 100% |

Table D.27: Non-destructive Results for L2 (TMT1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | 1 | 8 | 100% |
| 2000 | 3 | 48 | 100% |
| 3500 | 2 | 20 | 100% |
| 3900 | 2 | 17 | 100% |
| 4020 | 2 | 26 | 100% |
| 4850 | 2 | 24 | 100% |
| 5249 | 1 | 49 | 100% |
| 6300 | 1 | 8 | 100% |
| 8400 | 2 | 37 | 100% |
| 9077 | 3 | 7 | 100% |

Table D.28: Standard Operator Results for L3 (TMT1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | 1 | 7 | 1 | 4 | 1 | 8 | 100% |
| 2000 | 1 | 31 | 1 | 6 | 1 | 4 | 100% |
| 3500 | 2 | 3 | 2 | 9 | 1 | 9 | 100% |
| 3900 | 1 | 5 | 1 | 49 | 1 | 7 | 100% |
| 4020 | 1 | 16 | 1 | 4 | 1 | 4 | 100% |
| 4850 | 1 | 7 | 1 | 12 | 1 | 5 | 100% |
| 5249 | 1 | 8 | 1 | 4 | 1 | 4 | 100% |
| 6300 | 1 | 5 | 1 | 4 | 1 | 3 | 100% |
| 8400 | 2 | 7 | 1 | 14 | 1 | 44 | 100% |
| 9077 | 1 | 41 | 1 | 12 | 1 | 45 | 100% |

Table D.29: Non-destructive Results for L3 (TMT1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | 2 | 50 | 100% |
| 2000 | 1 | 9 | 100% |
| 3500 | 5 | 11 | 100% |
| 3900 | 7 | 15 | 100% |
| 4020 | 4 | 44 | 100% |
| 4850 | 1 | 47 | 100% |
| 5249 | 2 | 19 | 100% |
| 6300 | - | - | 86.4% |
| 8400 | 5 | 37 | 100% |
| 9077 | 10 | 15 | 100% |

Table D.30: Standard Operator Results for L4 (TMT1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | 2 | 12 | 1 | 7 | 2 | 4 | 100% |
| 2000 | 1 | 39 | 2 | 15 | 1 | 3 | 100% |
| 3500 | 3 | 6 | 1 | 10 | 1 | 43 | 100% |
| 3900 | 3 | 8 | 1 | 7 | 2 | 4 | 100% |
| 4020 | 1 | 7 | 1 | 8 | 2 | 12 | 100% |
| 4850 | 1 | 44 | 5 | 5 | 3 | 4 | 100% |
| 5249 | 5 | 11 | 1 | 7 | 1 | 7 | 100% |
| 6300 | 2 | 11 | 1 | 8 | 1 | 3 | 100% |
| 8400 | 2 | 23 | 1 | 6 | 1 | 5 | 100% |
| 9077 | 1 | 7 | 2 | 4 | 2 | 8 | 100% |

Table D.31: Non-destructive Results for L4 (TMT1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 55.6% |
| 2000 | - | - | 72.2% |
| 3500 | - | - | 55.6% |
| 3900 | - | - | 61.1% |
| 4020 | - | - | 61.1% |
| 4850 | - | - | 61.1% |
| 5249 | 8 | 49 | 100% |
| 6300 | - | - | 86.4% |
| 8400 | - | - | 66.7% |
| 9077 | - | - | 61.1% |

Table D.32: Standard Operator Results for L5 (TMT1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | - | - | - | - | 1 | 9 | 100% |
| 2000 | - | - | 1 | 19 | - | - | 100% |
| 3500 | 9 | 48 | 3 | 40 | 8 | 12 | 100% |
| 3900 | - | - | - | - | - | - | 77.8% |
| 4020 | - | - | - | - | 2 | 7 | 100% |
| 4850 | 7 | 39 | 6 | 42 | - | - | 100% |
| 5249 | - | - | - | - | - | - | 88.9% |
| 6300 | - | - | - | - | 4 | 12 | 100% |
| 8400 | - | - | - | - | - | - | 88.9% |
| 9077 | 1 | 49 | 5 | 13 | - | - | 100% |

Table D.33: Non-destructive Results for L5 (TMT1)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 24% |
| 2000 | - | - | 76% |
| 3500 | - | - | 40% |
| 3900 | - | - | 50% |
| 4020 | - | - | 24% |
| 4850 | - | - | 44% |
| 5249 | - | - | 42% |
| 6300 | - | - | 42% |
| 8400 | - | - | 42% |
| 9077 | - | - | 34% |

Table D.34: Standard Operator Results for L6 (TMT1)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | - | - | - | - | - | - | 86% |
| 2000 | - | - | - | - | - | - | 90% |
| 3500 | - | - | - | - | - | - | 92% |
| 3900 | - | - | - | - | - | - | 98% |
| 4020 | - | - | - | - | - | - | 96% |
| 4850 | - | - | - | - | - | - | 80% |
| 5249 | - | - | - | - | - | - | 76% |
| 6300 | - | - | 8 | 19 | - | - | 100% |
| 8400 | - | - | - | - | - | - | 84% |
| 9077 | - | - | - | - | - | - | 98% |

Table D.35: Non-destructive Results for L6 (TMT1)

D.3 Turing Machine Transducers (2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 54.5% |
| 2000 | 8 | 25 | 100% |
| 3500 | - | - | 54.5% |
| 3900 | - | - | 54.5% |
| 4020 | 5 | 40 | 100% |
| 4850 | - | - | 54.5% |
| 5249 | 5 | 46 | 100% |
| 6300 | - | - | 18.8% |
| 8400 | 8 | 49 | 100% |
| 9077 | - | - | 81.8% |

Table D.36: Standard Operator Results for L1 (TMT2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | 2 | 47 | 10 | 13 | 1 | 51 | 100% |
| 2000 | 3 | 22 | 1 | 9 | 1 | 47 | 100% |
| 3500 | - | - | - | - | 5 | 6 | 100% |
| 3900 | 5 | 22 | 5 | 9 | 2 | 5 | 100% |
| 4020 | 6 | 28 | 3 | 41 | 5 | 5 | 100% |
| 4850 | 2 | 23 | 3 | 7 | 2 | 4 | 100% |
| 5249 | - | - | 1 | 28 | 2 | 4 | 100% |
| 6300 | 8 | 45 | 1 | 47 | 1 | 19 | 100% |
| 8400 | 1 | 20 | 3 | 24 | 2 | 31 | 100% |
| 9077 | 1 | 21 | 2 | 12 | 1 | 39 | 100% |

Table D.37: Non-destructive Results for L1 (TMT2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | 2 | 26 | 100% |
| 2000 | 1 | 30 | 100% |
| 3500 | 1 | 5 | 100% |
| 3900 | 2 | 38 | 100% |
| 4020 | 2 | 29 | 100% |
| 4850 | 1 | 6 | 100% |
| 5249 | 1 | 8 | 100% |
| 6300 | 1 | 48 | 100% |
| 8400 | 1 | 4 | 100% |
| 9077 | 1 | 4 | 100% |

Table D.38: Standard Operator Results for L2 (TMT2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | 1 | 7 | 1 | 3 | 1 | 3 | 100% |
| 2000 | 2 | 4 | 1 | 4 | 1 | 4 | 100% |
| 3500 | 1 | 5 | 1 | 3 | 1 | 11 | 100% |
| 3900 | 1 | 3 | 1 | 5 | 1 | 9 | 100% |
| 4020 | 1 | 5 | 1 | 8 | 1 | 6 | 100% |
| 4850 | 1 | 10 | 1 | 10 | 1 | 4 | 100% |
| 5249 | 1 | 7 | 1 | 12 | 1 | 39 | 100% |
| 6300 | 1 | 6 | 1 | 5 | 1 | 4 | 100% |
| 8400 | 1 | 3 | 1 | 4 | 1 | 9 | 100% |
| 9077 | 1 | 4 | 1 | 7 | 1 | 3 | 100% |

Table D.39: Non-destructive Results for L2 (TMT2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | 4 | 31 | 100% |
| 2000 | 4 | 48 | 100% |
| 3500 | 3 | 32 | 100% |
| 3900 | 1 | 22 | 100% |
| 4020 | 1 | 27 | 100% |
| 4850 | 1 | 34 | 100% |
| 5249 | 4 | 39 | 100% |
| 6300 | 2 | 35 | 100% |
| 8400 | 10 | 21 | 100% |
| 9077 | 6 | 40 | 100% |

Table D.40: Standard Operator Results for L3 (TMT2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | 1 | 7 | 2 | 33 | 1 | 19 | 100% |
| 2000 | 2 | 22 | 1 | 7 | 1 | 3 | 100% |
| 3500 | 3 | 11 | 1 | 16 | 2 | 7 | 100% |
| 3900 | 1 | 27 | 3 | 23 | 1 | 11 | 100% |
| 4020 | 1 | 10 | 1 | 10 | 1 | 9 | 100% |
| 4850 | 1 | 19 | 1 | 3 | 1 | 33 | 100% |
| 5249 | 1 | 11 | 1 | 47 | 1 | 9 | 100% |
| 6300 | 1 | 19 | 2 | 33 | 1 | 45 | 100% |
| 8400 | 1 | 25 | 5 | 11 | 2 | 17 | 100% |
| 9077 | 1 | 11 | 2 | 11 | 3 | 23 | 100% |

Table D.41: Non-destructive Results for L3 (TMT2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 63.3% |
| 2000 | - | - | 80% |
| 3500 | - | - | 80% |
| 3900 | - | - | 63.3% |
| 4020 | - | - | 63.3% |
| 4850 | - | - | 53.3% |
| 5249 | - | - | 70% |
| 6300 | - | - | 70% |
| 8400 | - | - | 70% |
| 9077 | - | - | 53.3% |

Table D.42: Standard Operator Results for L4 (TMT2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | - | - | - | - | - | - | 86.7% |
| 2000 | - | - | - | - | - | - | 83.3% |
| 3500 | - | - | - | - | - | - | 96.7% |
| 3900 | - | - | - | - | - | - | 90% |
| 4020 | - | - | - | - | - | - | 93.3% |
| 4850 | - | - | - | - | - | - | 83.3% |
| 5249 | - | - | - | - | - | - | 83.3% |
| 6300 | - | - | - | - | - | - | 83.3% |
| 8400 | - | - | - | - | - | - | 90% |
| 9077 | - | - | - | - | - | - | 90% |

Table D.43: Non-destructive Results for L4 (TMT2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 69.2% |
| 2000 | - | - | 73.1% |
| 3500 | 2 | 27 | 100% |
| 3900 | 9 | 51 | 100% |
| 4020 | - | - | 69.2% |
| 4850 | 5 | 27 | 100% |
| 5249 | - | - | 61.5% |
| 6300 | 6 | 4 | 100% |
| 8400 | - | - | 61.5% |
| 9077 | - | - | 61.5% |

Table D.44: Standard Operator Results for L5 (TMT2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | - | - | 2 | 8 | 8 | 19 | 100% |
| 2000 | 2 | 5 | 2 | 9 | 7 | 22 | 100% |
| 3500 | 2 | 36 | 8 | 6 | 4 | 36 | 100% |
| 3900 | 4 | 49 | 6 | 24 | 5 | 4 | 100% |
| 4020 | - | - | 8 | 33 | 1 | 10 | 100% |
| 4850 | 2 | 27 | 2 | 17 | 3 | 34 | 100% |
| 5249 | 1 | 13 | 5 | 39 | 6 | 20 | 100% |
| 6300 | 2 | 15 | 6 | 32 | 1 | 27 | 100% |
| 8400 | 4 | 18 | 1 | 11 | 6 | 50 | 100% |
| 9077 | 3 | 15 | - | - | 4 | 21 | 100% |

Table D.45: Non-destructive Results for L5 (TMT2)

| Seed | Run | Generation | Best Fitness |
|------|-----|------------|--------------|
| 1500 | - | - | 22.2% |
| 2000 | - | - | 22.2% |
| 3500 | - | - | 22.2% |
| 3900 | - | - | 22.2% |
| 4020 | - | - | 22.2% |
| 4850 | - | - | 22.2% |
| 5249 | - | - | 22.2% |
| 6300 | - | - | 22.2% |
| 8400 | - | - | 22.2% |
| 9077 | - | - | 22.2% |

Table D.46: Standard Operator Results for L6 (TMT2)

| Seed | Mutation Only | | Crossover Only | | Crossover & Mutation | | Best Fitness |
|------|---------------|------------|----------------|------------|----------------------|------------|--------------|
| | Run | Generation | Run | Generation | Run | Generation | |
| 1500 | - | - | - | - | - | - | 53.7% |
| 2000 | - | - | - | - | - | - | 50% |
| 3500 | - | - | - | - | - | - | 83.3% |
| 3900 | - | - | - | - | - | - | 53.7% |
| 4020 | - | - | - | - | 2 | 10 | 100% |
| 4850 | - | - | - | - | - | - | 94.4% |
| 5249 | - | - | - | - | - | - | 55.6% |
| 6300 | - | - | - | - | 8 | 7 | 100% |
| 8400 | - | - | - | - | - | - | 64.8% |
| 9077 | - | - | - | - | - | - | 46.3% |

Table D.47: Non-destructive Results for L6 (TMT2)

Appendix E

User manual for the GP system software

E.1 Java version

The system was implemented in Java (JDK version 1.5.0_06). To run the software the JDK needs to be installed.

E.2 GP systems

Each GP system has a folder structure as shown in Figure E.1.

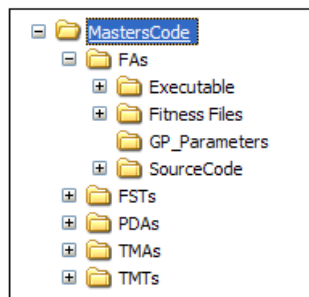


Figure E.1: Folder Structure

Each GP system has an *Executable* folder. Figure E.2 shows an example of the contents of the FAs' GP system.



Figure E.2: Executable JAR File

E.2.1 FAs

Double-click on the *FiniteAcceptors.jar* file found in the *FAs\Executable* folder. Alternatively, from the command prompt, *cd* to the *FAs\Executable* and run the command *java -jar FiniteAcceptors.jar*. A screen as shown in Figure E.3 will be displayed.

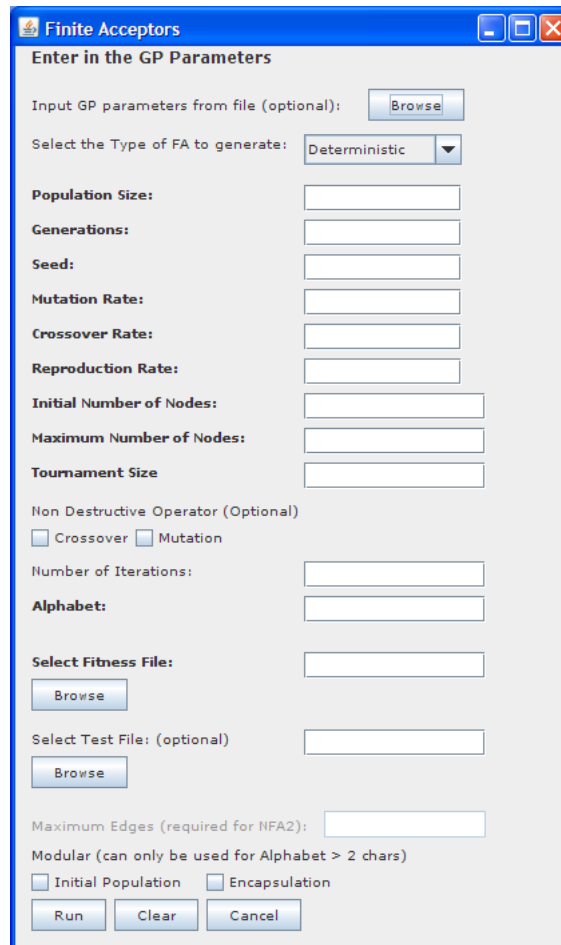


Figure E.3: FAs Application Screenshot

To execute a run perform the following steps:

- Choose the type of FA to generate from the drop down list. There are 3 options i.e., *Deterministic*,

Nondeterministic(1) and Nondeterministic(2).

- Enter in the parameter values for *Population size*, *Generations*, *Mutation Rate*, *Crossover Rate*, *Reproduction Rate*, *Initial number of Nodes*, *Maximum number of Nodes* and *Tournament size*. These values can be entered in manually or can be entered in from a file. To enter these values from a file click on the *Browse* for the field *Input GP Parameters from file* and select the appropriate file. A file with the GP Parameters used in this thesis can be found in the *FAs\GP Parameters* folder.
- Enter in the random *Seed* value for the run.
- If the run being performed uses the non-destructive methods, click on the checkbox labelled *Crossover* or/and the checkbox labelled *Mutation*.
- Enter in the alphabet of the language being tested e.g. 'ab'.
- Enter in the number of iterations used for each seed value.
- Choose the fitness case file by clicking on the *Browse* button below the label *Choose Fitness File*.
- If you wish to test any solutions generated provide a test file by clicking on the *Browse* button under the label *Choose Test File*.
- If the type of FA chosen is the nondeterministic(2) the option of *Maximum Edges* becomes available. Enter in the number of edges/transitions allowed to leave a state.
- If the type of FA chosen is deterministic and is a 3 character language the modular approach can be used. If you wish to use the modular approach select the type of modular approach by clicking on the checkbox *Initial Population* or/and *Encapsulation*.
- Click on the *Run* button to begin the run.

E.2.1.1 An example run for FAs

Figure E.4 shows an example with all the necessary fields filled. These parameters are for the L3 and the run is executed using the fitness case file used for this thesis.

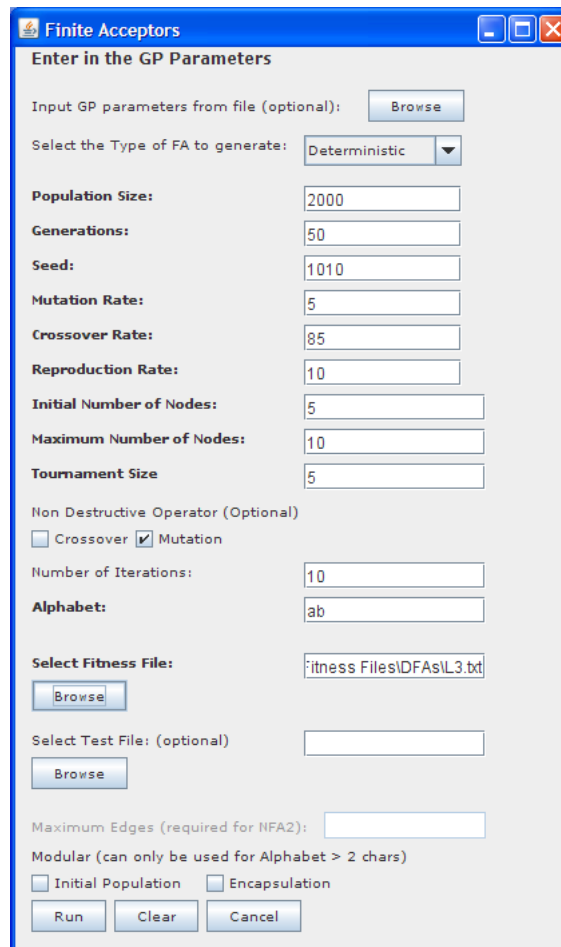


Figure E.4: Example run for deterministic FAs for L3

The results for this run is shown in Figure E.5.

```
Results
seed: 1010 iteration: 1 generation: 8

Solution 1:
0-Accept-{0,1}
1-Accept-{2,0}
2-Reject-{5,4}
3-Accept-{6,5}
4-Reject-{4,4}
5-Accept-{2,1}
6-Accept-{7,0}
7-Accept-{8,6}
8-Accept-{9,6}
9-Accept-{7,4}

Solution 2:
0-Accept-{0,1}
1-Accept-{2,0}
2-Reject-{5,4}
3-Accept-{6,7}
4-Reject-{4,4}
5-Accept-{2,1}
6-Accept-{5,0}
7-Reject-{6,4}
```

Figure E.5: Results obtained for L3 using seed 1010 (Deterministic FAs)

Figure E.6 shows a modular example for L15. The modular approach can be applied for evolving deterministic FAs for 3 character languages. The modular approach doesn't improve the nondeterministic FAs so the option is not available when choosing to evolve nondeterministic FAs.

Finite Acceptors

Enter in the GP Parameters

Input GP parameters from file (optional):

Select the Type of FA to generate:

Population Size:

Generations:

Seed:

Mutation Rate:

Crossover Rate:

Reproduction Rate:

Initial Number of Nodes:

Maximum Number of Nodes:

Tournament Size

Non Destructive Operator (Optional)

Crossover Mutation

Number of Iterations:

Alphabet:

Select Fitness File:

Select Test File: (optional)

Maximum Edges (required for NFA2):

Modular (can only be used for Alphabet > 2 chars)

Initial Population Encapsulation

Figure E.6: Example run for deterministic FAs for L15 using the Modular approach

The results for this run is shown in Figure E.7.

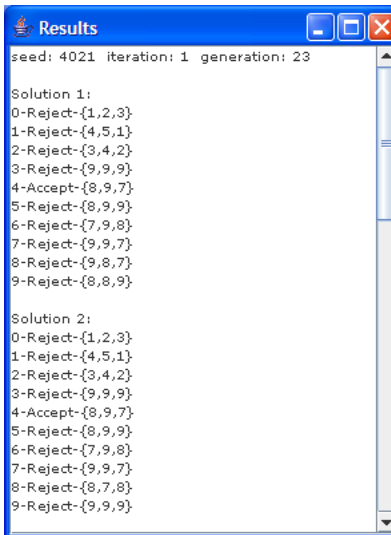


Figure E.7: Results obtained for L15 using seed 4021 (Modular Approach)

Figure E.8 shows an example run for NFA2 for L6. When *Nondeterministic (2)* is chosen for the type of FA to be evolved the *Maximum edges* field becomes available.

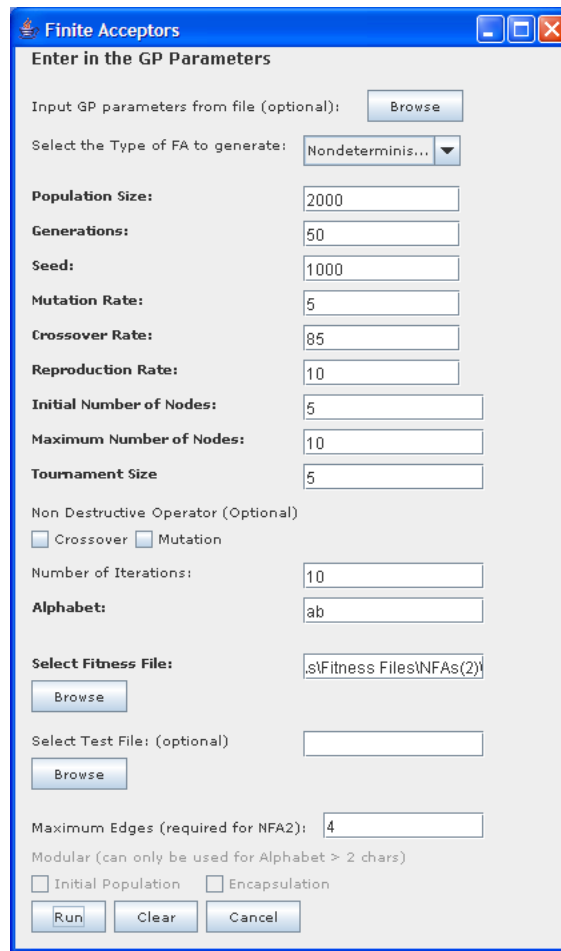


Figure E.8: Example run for deterministic FAs for L6 (NFA2)

The results for this run is shown in Figure E.9.

```

Results
seed: 1000 iteration: 1

No Solution Found
Best Of Generation:
0-Accept-{b->1,a->2}
1-Reject-{a->0}
2-Accept-{b->0,a->1}
3-Accept-{b->4,a->5}
4-Reject-{a->3}
5-Accept-{b->3,a->4}
6-Reject-{b->4}
7-Reject-{a->8,b->1}
8-Reject-{}
fitness=197
% = 71.63636363636363
-----

seed: 1000 iteration: 2 generation: 18

Solution 1:
0-Reject-{b->1,a->2}
1-Reject-{b->2,a->3}
2-Reject-{b->3,a->1,b->0}
3-Accept-{a->2,b->1,b->4}
4-Reject-{b->9,a->6}
5-Reject-{b->6,a->8,b->7}
6-Accept-{b->8}
7-Reject-{a->9}
8-Reject-{a->8}
9-Reject-{}

```

Figure E.9: Results obtained for L6 using seed 1000 (NFA2)

E.2.2 FSTs

Double-click on the *FiniteStateTransducers.jar* file found in the *FSTs\Executable* folder. Alternatively, from the command prompt, *cd* to the *FSTs\Executable* and run the command *java -jar FiniteAcceptors.jar*. A screen as shown in Figure E.10 will be displayed.

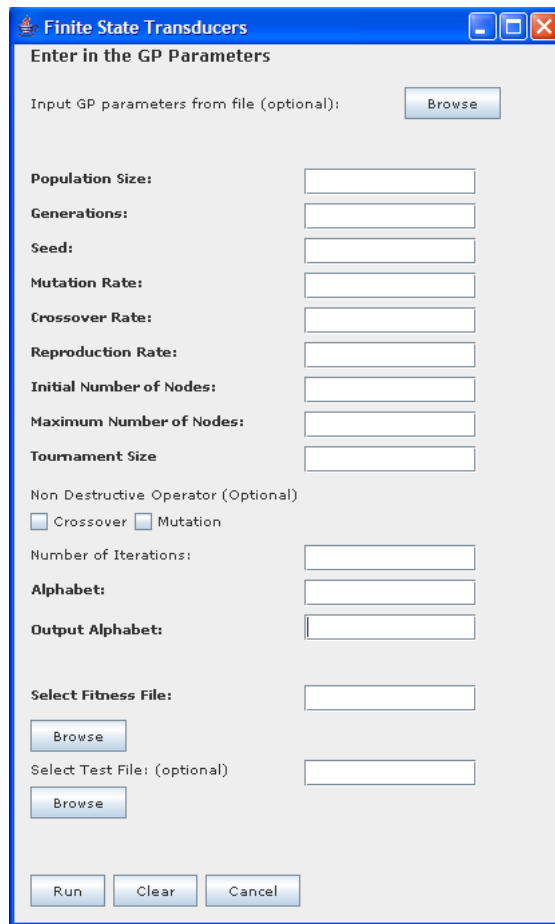


Figure E.10: FSTs Application Screenshot

To execute a run perform the following steps:

- Enter in the parameter values for *Population size*, *Generations*, *Mutation Rate*, *Crossover Rate*, *Reproduction Rate*, *Initial number of Nodes*, *Maximum number of Nodes* and *Tournament size*. These values can be entered in manually or can be entered in from a file. To enter these values from a file click on the *Browse* for the field *Input GP Parameters from file* and select the appropriate file. A file with the GP Parameters used in this thesis can be found in the *FSTs\GP Parameters* folder.
- Enter in the random *Seed* value for the run.
- If the run being performed uses the non-destructive methods, click on the checkbox labelled *Crossover* or/and the checkbox labelled *Mutation*.
- Enter in the alphabet of the language being tested e.g. 'ab'.
- Enter in the output alphabet e.g. '01'.

- Enter in the number of iterations used for each seed value.
- Choose the fitness case file by clicking on the *Browse* button below the label *Choose Fitness File*.
- If you wish to test any solutions generated provide a test file by clicking on the *Browse* button under the label *Choose Test File*.
- Click on the *Run* button to begin the run.

E.2.2.1 An example run for FSTs

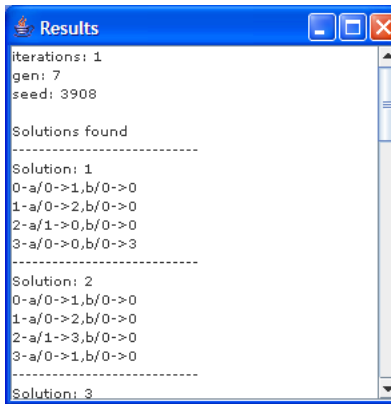
Figure E.11 shows an example with all the necessary fields filled. These parameters are for the L1 and the run is executed using the fitness case file used for this thesis.

The screenshot shows a window titled "Finite State Transducers" with a sub-header "Enter in the GP Parameters". The window contains the following fields and controls:

- Input GP parameters from file (optional):
- Population Size:
- Generations:
- Seed:
- Mutation Rate:
- Crossover Rate:
- Reproduction Rate:
- Initial Number of Nodes:
- Maximum Number of Nodes:
- Tournament Size:
- Non Destructive Operator (Optional):
 - Crossover
 - Mutation
- Number of Iterations:
- Alphabet:
- Output Alphabet:
- Select Fitness File:
 -
- Select Test File: (optional)
 -
- At the bottom:

Figure E.11: Example run for deterministic FSTs for L1

The results for this run is shown in Figure E.12.



```
Results
-----
iterations: 1
gen: 7
seed: 3908

Solutions found
-----
Solution: 1
0-a/0->1,b/0->0
1-a/0->2,b/0->0
2-a/1->0,b/0->0
3-a/0->0,b/0->3
-----
Solution: 2
0-a/0->1,b/0->0
1-a/0->2,b/0->0
2-a/1->3,b/0->0
3-a/0->1,b/0->0
-----
Solution: 3
```

Figure E.12: Results obtained for L1 using seed 3908 (FSTs)

E.2.3 PDAs

Double-click on the *PushdownAutomata.jar* file found in the *PDAs\Executable* folder. Alternatively, from the command prompt, *cd* to the *PDAs\Executable* and run the command *java -jar PushdownAutomata.jar*. A screen as shown in Figure E.13 will be displayed.

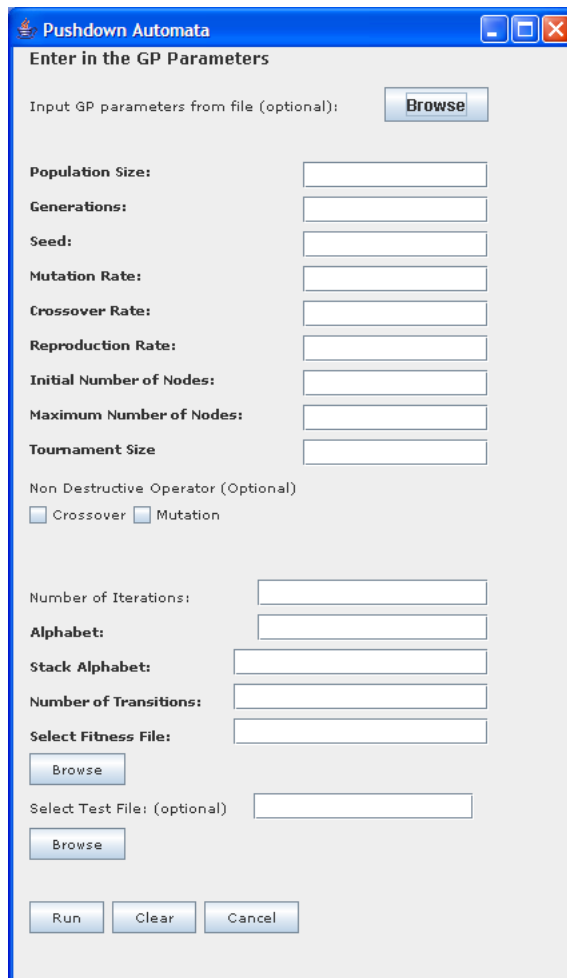


Figure E.13: PDAs Application Screenshot

To execute a run perform the following steps:

- Enter in the parameter values for *Population size*, *Generations*, *Mutation Rate*, *Crossover Rate*, *Reproduction Rate*, *Initial number of Nodes*, *Maximum number of Nodes* and *Tournament size*. These values can be entered in manually or can be entered in from a file. To enter these values from a file click on the *Browse* for the field *Input GP Parameters from file* and select the appropriate file. A file with the GP Parameters used in this thesis can be found in the *PDA's\GP Parameters* folder.
- Enter in the random *Seed* value for the run.
- If the run being performed uses the non-destructive methods, click on the checkbox labelled *Crossover* or/and the checkbox labelled *Mutation*.
- Enter in the alphabet of the language being tested e.g. 'ab'.

- Enter in the stack alphabet e.g. ‘AB’.
- Enter in the number of iterations used for each seed value.
- Enter in number of transitions allowed for each state.
- Choose the fitness case file by clicking on the *Browse* button below the label *Choose Fitness File*.
- If you wish to test any solutions generated provide a test file by clicking on the *Browse* button under the label *Choose Test File*.
- Click on the *Run* button to begin the run.

E.2.3.1 An example run for PDAs

Figure E.14 shows an example with all the necessary fields filled. These parameters are for the L1 and the run is executed using the fitness case file used for this thesis.

The screenshot shows a window titled "Pushdown Automata" with a sub-header "Enter in the GP Parameters". The interface includes the following elements:

- An "Input GP parameters from file (optional):" label with a "Browse" button.
- Fields for: Population Size (2000), Generations (50), Seed (5678), Mutation Rate (30), Crossover Rate (60), Reproduction Rate (10), Initial Number of Nodes (2), Maximum Number of Nodes (5), and Tournament Size (5).
- A "Non Destructive Operator (Optional)" section with checkboxes for "Crossover" and "Mutation".
- Fields for: Number of Iterations (2), Alphabet (ab), Stack Alphabet (AB), and Number of Transitions (4).
- A "Select Fitness File:" field containing the path "stersCode\PDAs\Languages\L1.txt" and a "Browse" button.
- A "Select Test File: (optional)" field with an empty text box and a "Browse" button.
- Buttons for "Run", "Clear", and "Cancel" at the bottom.

Figure E.14: Example run for deterministic PDAs for L1

The results for this run is shown in Figure E.15.

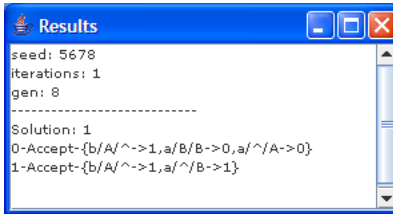


Figure E.15: Results obtained for L1 using seed 5678 (PDAs)

E.2.4 TMAs

Double-click on the *TuringMachineAcceptors.jar* file found in the *TMAs\Executable* folder. Alternatively, from the command prompt, *cd* to the *TMAs\Executable* and run the command *java -jar TuringMachineAcceptors.jar*. A screen as shown in Figure E.16 will be displayed.

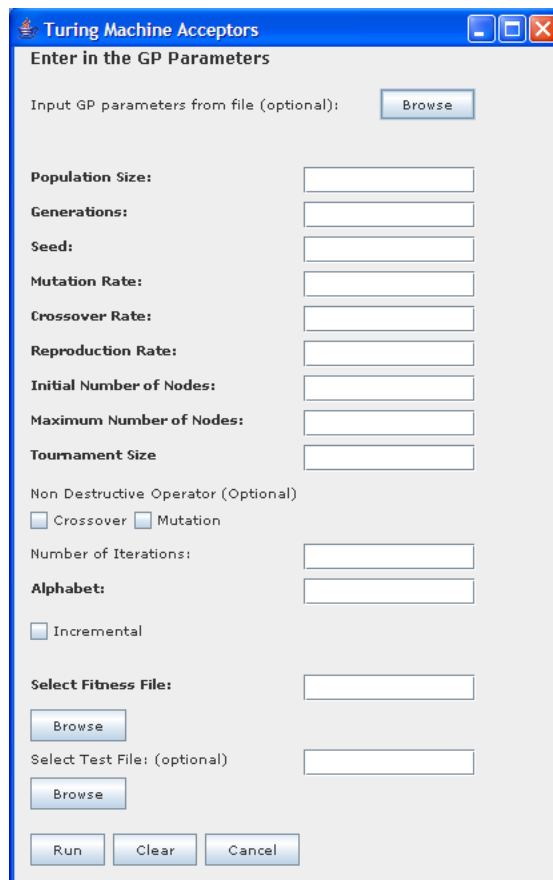


Figure E.16: TMAs Application Screenshot

To execute a run perform the following steps:

- Enter in the parameter values for *Population size*, *Mutation Rate*, *Crossover Rate*, *Reproduction Rate*, *Initial number of Nodes*, *Maximum number of Nodes* and *Tournament size*. These values can be entered in manually or can be entered in from a file. To enter these values from a file click on the *Browse* for the field *Input GP Parameters from file* and select the appropriate file. A file with the GP Parameters used in this thesis can be found in the *TMA's\GP Parameters* folder.
- Enter in the number of generations allowed for the run.
- Enter in the random *Seed* value for the run.
- If the run being performed uses the non-destructive methods, click on the checkbox labelled *Crossover* or/and the checkbox labelled *Mutation*.
- Enter in the alphabet of the language being tested e.g. 'abB'.
- Enter in the number of iterations used for each seed value.
- If you wish to employ the incremental method, click on the checkbox labelled *Incremental*.
- Choose the fitness case file by clicking on the *Browse* button below the label *Choose Fitness File*.
- If you wish to test any solutions generated provide a test file by clicking on the *Browse* button under the label *Choose Test File*.
- Click on the *Run* button to begin the run.

E.2.4.1 An example run for TMAs

Figure E.17 shows an example with all the necessary fields filled. These parameters are for the L3 and the run is executed using the fitness case file used for this thesis.

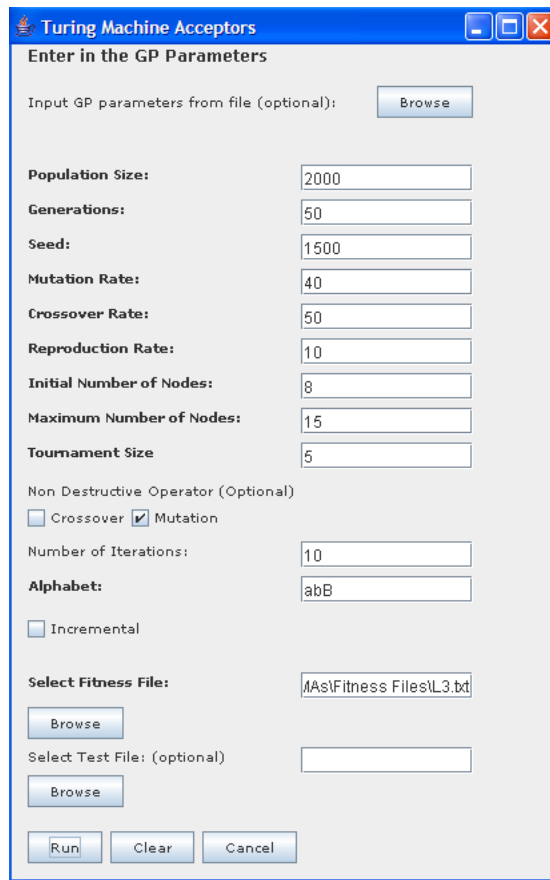


Figure E.17: Example run for deterministic TMAs for L3

The results for this run is shown in Figure E.18.

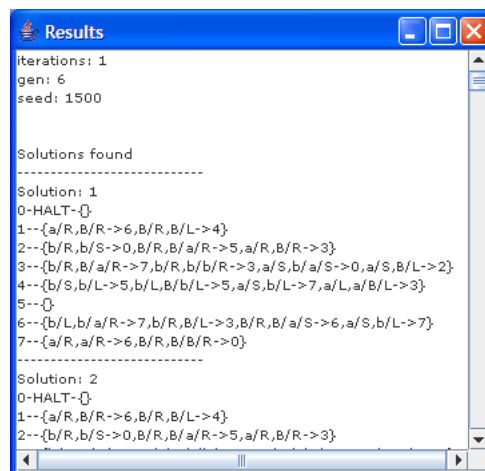


Figure E.18: Results obtained for L3 using seed 1500 (TMAs)

E.2.5 TMTs

Double-click on the *TuringMachineTransducers.jar* file found in the *TMTs\Executable* folder. Alternatively, from the command prompt, *cd* to the *TMTs\Executable* and run the command *java -jar TuringMachineTransducers.jar*. A screen as shown in Figure E.19 will be displayed.

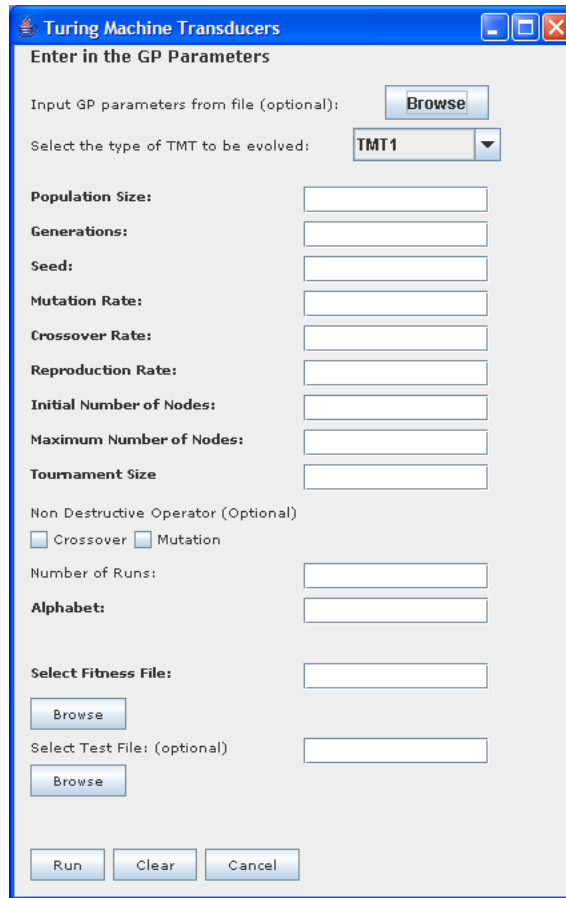


Figure E.19: TMTs Application Screenshot

To execute a run perform the following steps:

- Choose the type of TMT to generate from the drop down list. There are 2 options i.e., *TMT1* and *TMT2*.
- Enter in the parameter values for *Population size*, *Generations*, *Mutation Rate*, *Crossover Rate*, *Reproduction Rate*, *Initial number of Nodes*, *Maximum number of Nodes* and *Tournament size*. These values can be entered in manually or can be entered in from a file. To enter these values from a file click on the *Browse* for the field *Input GP Parameters from file* and select the appropriate file. A file with the GP Parameters used in this thesis can be found in the *TMTs\GP Parameters* folder.

- Enter in the random *Seed* value for the run.
- If the run being performed uses the non-destructive methods, click on the checkbox labelled *Crossover* or/and the checkbox labelled *Mutation*.
- Enter in the alphabet of the language being tested e.g. '01B'.
- Enter in the number of iterations used for each seed value.
- Choose the fitness case file by clicking on the *Browse* button below the label *Choose Fitness File*.
- If you wish to test any solutions generated provide a test file by clicking on the *Browse* button under the label *Choose Test File*.
- Click on the *Run* button to begin the run.

E.2.5.1 An example run for TMTs

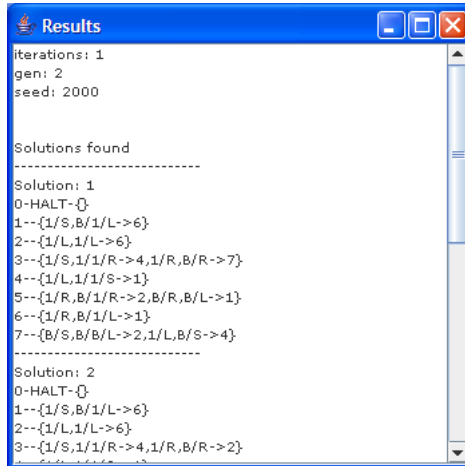
Figure E.20 shows an example with all the necessary fields filled. These parameters are for the L2 and the run is executed using the fitness case file used for this thesis.

The screenshot shows a window titled "Turing Machine Transducers" with a sub-dialog titled "Enter in the GP Parameters". The dialog contains the following fields and controls:

- Input GP parameters from file (optional):
- Select the type of TMT to be evolved: (dropdown menu)
- Population Size:
- Generations:
- Seed:
- Mutation Rate:
- Crossover Rate:
- Reproduction Rate:
- Initial Number of Nodes:
- Maximum Number of Nodes:
- Tournament Size:
- Non Destructive Operator (Optional):
 - Crossover Mutation
- Number of Runs:
- Alphabet:
- Select Fitness File:
- Select Test File: (optional)
- Buttons at the bottom:

Figure E.20: Example run for deterministic TMTs for L3

The results for this run is shown in Figure E.21.



```
Results
iterations: 1
gen: 2
seed: 2000

Solutions found
-----
Solution: 1
0-HALT-[]
1--{1/S,B/1/L->6}
2--{1/L,1/L->6}
3--{1/S,1/1/R->4,1/R,B/R->7}
4--{1/L,1/1/S->1}
5--{1/R,B/1/R->2,B/R,B/L->1}
6--{1/R,B/1/L->1}
7--{B/S,B/B/L->2,1/L,B/S->4}
-----
Solution: 2
0-HALT-[]
1--{1/S,B/1/L->6}
2--{1/L,1/L->6}
3--{1/S,1/1/R->4,1/R,B/R->2}
```

Figure E.21: Results obtained for L2 using seed 2000 (TMTs)