Automated Theorem Proving
As the 21st century begins, the power of our magical new tool and partner, the computer, is increasing at an astonishing rate. Computers that perform billions of operations per second are now commonplace. Multiprocessors with thousands of little computers — relatively little! — can now carry out parallel computations and solve problems in seconds that only a few years ago took days or months. Chess-playing programs are on an even footing with the world’s best players. IBM’s Deep Blue defeated world champion Garry Kasparov in a match several years ago. Increasingly computers are expected to be more intelligent, to reason, to be able to draw conclusions from given facts, or abstractly, to prove theorems — the subject of this book.

Specifically, this book is about two theorem-proving programs, THEO and HERBY. The first four chapters contain introductory material about automated theorem proving and the two programs. This includes material on the language used to express theorems, predicate calculus, and the rules of inference. This also includes a description of a third program included with this package, called COMPILE. As described in Chapter 3, COMPILE transforms predicate calculus expressions into clause form as required by HERBY and THEO. Chapter 5 presents the theoretical foundations of semantic tree theorem proving as performed by HERBY. Chapter 6 presents the theoretical foundations of resolution–refutation theorem proving as performed by THEO. Chapters 7 and 8 describe HERBY and how to use it. Chapters 9 and 10 parallel Chapters 7 and 8, but for THEO. Chapter 11 and 12 discuss the source code for the two programs. The final chapter, Chapter 13, briefly examines two other automated theorem-proving programs, Gandalf and Otter.

In the 1970s and 1980s, the author was involved in the design of chess programs. His program OSTRICH competed in five world computer chess championships dating back to 1974, when it narrowly missed defeating the Soviet program KAISSA in the final round of the first World Computer Chess Championship in Stockholm. Many of the lessons of programming
chess carry over to the field of automated theorem proving. In chess, a program searches a large tree of move sequences looking for the best line of play. In theorem proving, a program also searches a large tree of inferences — rather than moves — looking for that special sequence that yields a proof.

At the heart of both problems is the exponential nature of the search tree and the use of various algorithms and heuristics that direct the search toward the more relevant parts of the tree. Chess programs use various algorithms to narrow the search space such as the alpha-beta algorithm, windowing algorithms, algorithms that take advantage of move transpositions, and iteratively deepening depth-first search. Some programs also use various heuristics, best defined as rules of thumb, to further narrow the search, although experience has shown that one must use extreme caution in this case. The heuristics in early chess programs were far too unreliable, and as the former World Champion Mikhail Botvinnik once said, often “threw away the baby with the bath water.” Most heuristics are dangerous because as the level of play goes up, the number of exceptions to any heuristic increases as well. “Beautiful” moves often violate the relatively simpleminded heuristics used in programs.

The search techniques used to prove theorems are very similar to those used in chess programs. The theorem-proving program THEO contained in this package uses iteratively deepening depth-first search, hash tables for reducing the search space, and other algorithms to narrow the search space without sacrificing the ability to find a proof if one exists. Several heuristics are also normally used which do sacrifice this ability, although the user can choose not to use them. The second theorem-proving program HERBY constructs large semantic trees in its effort to prove a theorem using various heuristics to guide the process.

These two programs are meant to familiarize the reader with search techniques used in theorem-proving programs, to permit experiments with two capable theorem-proving programs, and to provide the source code so that the reader can attempt to improve it. In 1989, THEO, then called THE GREAT THEOREM PROVER, or TGTP, first appeared, and over the years it was used as a text-program at several universities and research centers. The latest version, recently renamed THEO, contains a far stronger theorem-proving program, a more extensive text to go along with it, and moreover, this time source code is available. As a theorem-proving program, THEO is quite sophisticated. HERBY is less capable as a theorem-proving program, but its approach is particularly simple and seems to have considerable potential. Both programs have participated in the Conference on Automated Deduction's competitions for such programs, as discussed in Chapter 13.
The package, consisting of software and text, can serve as instructional material for a course on theorem proving at either the undergraduate or graduate level. It can also serve as supplemental material for an introductory course on artificial intelligence. The package includes almost two hundred theorems for the student. Some are very easy and others are very difficult. There are many examples scattered throughout the text, and there are exercises at the end of every chapter. In addition to the theorems included, several thousand theorems that are used by the automated theorem-proving research community can be obtained by ftp, as is explained in Chapter 1.

The source code provided in this package has evolved over a ten-year period. Every effort has been made to make it readable. Every file lists the functions contained, and every function has a header that lists who calls it, who it calls, its arguments, and what it returns. Students should be able to modify the code as a class project.

The author would like to thank a number of people who have helped to develop THEO and HERBY. In particular, former McGill University students Paul Labutte, Patrice Lapierre, and Mohammed Almulla and current students Choon Kyu Kim and Paul Haroun deserve thanks. In addition, countless McGill students that the author has had in his classes must be thanked for their many suggestions on how to improve the programs. The author had many problems over the years in developing the software; thanks for help is extended to the System Support Group of the School of Computer Science at McGill. Lastly, the author's two daughters, Amy and Molly, must be given a special thanks for tolerating the passion of their father for this esoteric subject.

In the weeks leading up to the publication of this book, the author discussed the manuscript with Gabby Silberman, head of IBM Toronto's Center for Advanced Studies. He offered to add IBM's C compiler for AIX, Version 4.4, to the CD-ROM. IBM's Joe Wigglesworth checked out the theorem-proving software and confirmed that it was compatible with the IBM software. Springer-Verlag and IBM, as you can see, worked out an agreement whereby the compiler is included on the CD-ROM. The author would like to thank IBM and Springer-Verlag for this.

The author wishes the reader many hours of interesting learning and experimentation. Any suggestions for improving this package for the next version can be sent to the author via e-mail at newborn@cs.mcgill.ca.

Montreal, Canada
September 2000

Monty Newborn
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>v</td>
</tr>
<tr>
<td>1 A Brief Introduction to COMPILE, HERBY, and THEO</td>
<td>1</td>
</tr>
<tr>
<td>1.1 COMPILE</td>
<td>2</td>
</tr>
<tr>
<td>1.1.1 Creating an executable version of COMPILE</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2 Running COMPILE</td>
<td>2</td>
</tr>
<tr>
<td>1.2 HERBY</td>
<td>3</td>
</tr>
<tr>
<td>1.2.1 Creating an executable version of HERBY</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2 Running HERBY</td>
<td>3</td>
</tr>
<tr>
<td>1.3 THEO</td>
<td>4</td>
</tr>
<tr>
<td>1.3.1 Creating an executable version of THEO</td>
<td>4</td>
</tr>
<tr>
<td>1.3.2 Running THEO</td>
<td>4</td>
</tr>
<tr>
<td>1.4 The Accompanying Software</td>
<td>5</td>
</tr>
<tr>
<td>Exercises for Chapter 1</td>
<td>6</td>
</tr>
<tr>
<td>2 Predicate Calculus, Well-Formed Formulas and Theorems</td>
<td>7</td>
</tr>
<tr>
<td>2.1 The Syntax of Well-Formed Formulas</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Examples of Well-Formed Formulas</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Creating Well-Formed Formulas from Statements in English</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Interpretations of Well-Formed Formulas</td>
<td>9</td>
</tr>
<tr>
<td>2.5 A Set of Axioms to Prove Theorems in Group Theory</td>
<td>12</td>
</tr>
<tr>
<td>2.6 An Axiom System for Euclidean Geometry</td>
<td>14</td>
</tr>
<tr>
<td>Exercises for Chapter 2</td>
<td>17</td>
</tr>
</tbody>
</table>

ix
3 COMPILE: Transforming Well-Formed Formulas to Clauses 21
  3.1 The Transformation Procedure of COMPILE 21
  3.2 Using COMPILE 24
  3.3 Examples of the Transformation of Wffs to Clauses 25
    Exercises for Chapter 3 25

4 Inference Procedures 29
  4.1 An Informal Introduction to Binary Resolution
    and Binary Factoring 29
  4.2 The Processes of Substitution and Unification 31
  4.3 Subsumption 32
  4.4 The Most General Unifier 33
  4.5 Determining All Binary Resolvents of Two Clauses 33
  4.6 Merge Clauses 37
  4.7 Determining All Binary Factors of a Clause 37
  4.8 A Special Case of Binary Resolution: Modus Ponens 38
  4.9 Clauses and Subsumption 39
  4.10 Logical Soundness 40
  4.11 Base Clauses and Inferred Clauses 40
    Exercises for Chapter 4 40

5 Proving Theorems by Constructing Closed Semantic Trees 43
  5.1 The Herbrand Universe of a Set of Clauses 43
  5.2 The Herbrand Base of a Set of Clauses 44
  5.3 An Interpretation on the Herbrand Base 45
  5.4 Establishing the Unsatisfiability of a Set of Clauses 45
  5.5 Semantic Trees 47
  5.6 Noncanonical Semantic Trees 50
    Exercises for Chapter 5 52

6 Resolution–Refutation Proofs 53
  6.1 Examples of Resolution–Refutation Proofs 54
  6.2 The Depth and Length of Resolution–Refutation
    Proofs 58
  6.3 Obtaining a Resolution–Refutation Proof from a
    Semantic Tree 58
  6.4 Linear Proofs 64
  6.5 Restrictions on the Form of Linear Proofs 71
  6.6 The Lifting Lemma 79
  6.7 Linear Proofs and Factoring 80
    Exercises for Chapter 6 82
7 HERBY: A Semantic-Tree Theorem Prover 85
  7.1 Heuristics for Selecting Atoms 85
  7.2 Additional Heuristics 91
    7.2.1 List ordering heuristics 91
    7.2.2 Preliminary phase (Phase 0):
      Base clause resolution heuristic (BCRH) 93
    7.2.3 Heuristic limiting the number of literals in a clause 93
    7.2.4 Heuristic limiting the number of terms in a literal 93
    7.2.5 Tree pruning heuristic 94
  7.3 Assigning a Hash Code to a Literal and to a Clause 94
  7.4 The Overall Algorithm 94
  7.5 Obtaining a Resolution–Refutation Proof 95
    Exercises for Chapter 7 95

8 Using HERBY 97
  8.1 Proving Theorems with HERBY: The Input File 97
  8.2 HERBY's Convention on Naming the Output File 98
  8.3 The Options Available to the User 98
    8.3.1 Option to prove a set of theorems 99
    8.3.2 Obtaining help by typing "?" 100
  8.4 User Interaction During the Construction 100
  8.5 Option Examples 100
  8.6 The Printout Produced by HERBY 100
  8.7 A Second Example, the Printout Produced Using the r1 Option 107
    Exercises for Chapter 8 111

9 THEO: A Resolution–Refutation Theorem Prover 113
  9.1 Iteratively Deepening Depth-First Search
    and Linear Proofs 114
  9.2 Searching for a Linear-Merge Proof 116
  9.3 Searching for a Linear-nc Proof 118
  9.4 Searching for a Linear-Merge-nc Proof 119
  9.5 The Extended Search Strategy 120
  9.6 Bounding the Number of Literals in a Clause 121
  9.7 Bounding the Number of Terms in a Literal 121
  9.8 Bounding the Number of Different Variables
    in an Inference 121
  9.9 Ordering Clauses at Each Node 122
  9.10 A Hash Table that Stores Information About Clauses 122
    9.10.1 Assigning a hash code to a literal 122
# Contents

11 A Look at the Source Code of HERBY 161
11.1 Source Files for HERBY 161
11.2 Function Linkage in HERBY 164
11.3 A Brief Description of the Main Functions in HERBY 164
11.4 Machine Code Representation of a Clause in HERBY 165
   11.4.1 The clause header 165
   11.4.2 The literal header 167
   11.4.3 Representing the terms of a literal 167
   11.4.4 An example of the representation 168
11.5 Major Arrays in HERBY 170
   Exercises for Chapter 11 172

12 A Look at the Source Code of THEO 173
12.1 Source Files for THEO 173
12.2 Function Linkage in THEO 174
12.3 A Brief Description of the Main Functions in THEO 174
12.4 Machine Code Representation of a Clause in THEO 178
12.5 Major Arrays in THEO 178
12.6 Functions Related to clause_hash_table 179
12.7 Functions Related to cl_array 180
   Exercises for Chapter 12 180

13 The CADE ATP System Competitions and Other Theorem Provers 181
13.1 Gandalf 182
13.2 Otter 191
   Exercises for Chapter 13 205

Bibliography 207

Appendix A Answers to Selected Exercises 211

Appendix B List of Wffs and Theorems in the Directories WFFS, THEOREMS, GEOMETRY, and THMSMISC 219

Index 225
$P_1: a$ is a animal
$P_2: a$ is a wolf
$P_3: a$ is a fox
$P_4: a$ is a bird
$P_5: a$ is a caterpillar

$(Ax)(P_x \rightarrow P_{ neg x} ) \& (Ax)P_x$

$Q_0: a$ is a plant
$Q_1: a$ is a grain
$R: a$ is a like