

Chapter 1

The Menu, The Map, and the Magic

What is shared by the following three problems, the first an intriguing puzzle, the second an interesting circuit design question, and the third a deep theorem from mathematics?

There are twelve billiard balls, eleven of which are identical in weight. The remaining ball—the odd one—has a different weight. You are not told whether it is heavier or lighter. You have a balance scale for weighing the balls. Can you find which ball is the odd ball in three weighings, and can you also find out whether it is lighter or heavier than the others?

Using as many AND and OR gates as you like, but using only two NOT gates, can you design a circuit according to the following specification? There are three inputs, i_1 , i_2 , and i_3 , and three outputs, o_1 , o_2 , and o_3 . The outputs are related to the inputs in the following simple way:

$$o_1 = \text{not}(i_1) \quad o_2 = \text{not}(i_2) \quad o_3 = \text{not}(i_3).$$

Remember, you can use only two NOT gates!

Do the following three equalities, which characterize a Robbins algebra, provide an axiomatization for Boolean algebra, where $+$ can be interpreted as addition (or, more familiarly in the context of Boolean algebra, as union) and the function n as negation or complement?

- (1) $x + y = y + x$ (commutativity)
- (2) $(x + y) + z = x + (y, z)$ (associativity)
- (3) $n(n(x + y) + n(x + n(y))) = x$ (Robbins axiom)

Axiom 3 can also be written as (3a), where prime denotes complement.

$$(3a) \ ((x + y)' + (x + y')')' = x \text{ (Robbins axiom)}$$

The three problems share the property that each can be solved with logical reasoning, the type used by Mr. Spock of *Star Trek* and by Sherlock Holmes the famous detective or, if you prefer nonfictional people, by the better mathematicians—Hilbert, Gödel, and von Neumann, for example—who are justly revered for the power of their minds. Indeed, what is needed is reasoning that is carefully directed toward the target and that is appropriately restricted to avoid getting lost. Of course, the reasoning must yield conclusions that follow inevitably from the hypotheses.

Far more tantalizing than the solvability by applying logical reasoning, the three problems share a property that not long ago would have been considered *magic*. Specifically, each has been solved by relying on an automated reasoning assistant, a concept fully developed in this book; your assistant can entertain you, but it can also play a vital role in research. Whether or not you are captivated by any game, here you will learn where the fascinating country of automated reasoning is located in the large world of computing. This book provides you with a visit to this country; it will be your guide, to its language, to its customs, to its delights! You, however, decide precisely which route to take, for the book can be read in various ways. You will learn precisely what automated reasoning is, how it works, where it has been applied with astounding results that include answers to questions that had baffled scientists since the mid-1930s, and, most significant, why its use is so effective. You will also learn here that the rules applied by a reasoning assistant for drawing conclusions are strikingly different from those a person uses.

The cited three problems also share a third property: Each is a tough problem to solve. Indeed, the first time (at the age of twenty) I was presented the billiard-ball puzzle, I was unable to find the solution; almost three decades later, I was astounded to learn (thanks to an automated reasoning assistant) that many, many distinct solutions exist. As for the circuit design question, it has been used on Ph.D. exams for engineers; that a circuit of the desired type can be found is, at least to me, counterintuitive.

Although the first two problems are certainly challenging, the difficulty each presents is dwarfed by that of the third, the theorem from mathematics. Mathematics is not the culprit; indeed, many theorems in mathematics are easy to prove, at least by a mathematician. However, the Robbins algebra problem remained a mystery for more than six decades. It was even offered as an open question by the great logician Alfred Tarski in his book (Henkin, Monk, and Tarski 1971). Neither he nor any of his students could say whether the three axioms for a Robbins algebra axiomatized Boolean algebra. The answer to this question eluded one mathematician after another—until one of William McCune's automated reasoning programs provided the answer in October of 1996; for the essentials of the story, see the *New York Times* article by Gina Kolata (1996). The fact that an automated reasoning assistant was able to answer this deep, deep question deservedly received many accolades.

As you will learn in this book, this success in mathematics is by no means isolated. Open questions from other fields yielded their treasure to a reasoning program of the type featured here. Nor is the assistance of a reasoning program *pertinent only to finding proofs*. Indeed, OTTER has been used to complete proofs far more elegant than thought possible; such examples are discussed in Chapters 10 and 11, and some are given in the Appendix. If a reasoning assistant can find more elegant proofs, then it can apply the same techniques to vaguely similar problems, such as finding more efficient circuits and the like.

As you will find, automated reasoning is permeated with magic. In particular, here you will learn how you can magically tell your automated assistant what the problem is about, which choices to make for the type of reasoning to draw new conclusions, and, so intriguing, how to control the reasoning through the use of strategy, thus dramatically increasing the likelihood of success. Is there anyone who avidly plays poker, chess, checkers, basketball, or any other sport who does not find the use of strategy intriguing?

At this point, no doubt, you have numerous questions, of which the following three are most pressing.

- What does this book offer?
- Who will enjoy it?
- How can the book be read?

These three questions are respectively answered in Sections 1.1–1.3.

As the book proceeds, more questions (many of which are answered in this chapter) will be posed and answered:

- Precisely what is *reasoning*?
- What examples exist that illustrate the differences between person-oriented and computer-oriented reasoning?
- Why should you use OTTER?
- How does its approach differ from that taken by other reasoning programs?
- What in general is the Argonne paradigm for the automation of reasoning?
- What obstacles are presented by the noble endeavor of producing a program whose reasoning is so general and so powerful that it is useful for completing assignments that range from solving puzzles to proving deep theorems?
- How does automated reasoning differ from (classical) artificial intelligence?

1.1 The Menu for the Grand Feast

This book is a veritable tapas bar, its menu offering many types of food, in varying portions, for a wide spectrum of readers that includes students, teachers, and researchers (some of whom will have little or no interest in automated reasoning), and the curious who enjoy the frontiers of science. The offerings include a chapter

(Chapter 9) that guides you in the use of OTTER, a manual (included on the CD-ROM) for the program, problems and challenges to be met, and deep questions still unanswered (some in the vignettes chapter, Chapter 10, and many in Chapter 11). Some of the problems (in Chapter 10) are useful to test your understanding and grasp of the concepts, and some of them are useful in honing your skill with the use of the program OTTER. For some of the problems, Section 10.64 is devoted to hints for solving them; for many more of the problems, that section gives answers. To complete the meal is a chapter (Chapter 8) presenting the underlying formalism. For dessert, in a companion book consisting of two volumes, *The Collected Works of Larry Wos*, you will find a library of my published papers, also found on the included CD-ROM.

A full meal virtually demands access to wine—which, in this case, takes the form of vignettes (in Chapter 10). Specifically, for almost all of my published papers, this book offers a companion article. The typical vignette answers questions that focus on genesis, historical perspective, significance, implementation, usefulness, and future research. These short articles are also the home for many problems, challenges, and open questions. As noted, Chapter 10 provides (near its end) hints, so placed to give you a good chance to solve the vignette problems without bias. Chapter 10 (at its close) is also where you can satisfy your craving for answers to many of the problems posed in the vignettes. (Had the answers been placed earlier, the temptation to skip the hints and quickly turn to the answers might have been hard to resist.)

The feast is served in part on the printed page and in part on the included CD-ROM. On the CD-ROM, you will find a copy of OTTER (the versatile and powerful automated reasoning program developed by W. McCune) in three versions: for workstations, for personal computers, and for the Macintosh. OTTER is written in more than 28,000 lines of C. OTTER functions well as a reasoning assistant for mathematics, logic, circuit design, program verification, puzzle solving, conjecture testing, proof improving, proofchecking, and more. To aid you in the use of OTTER, on the CD-ROM are numerous input files that can be used as is or modified appropriately, output files to study for new ideas, proofs to examine with the possibility of finding more elegant ones, and many other treasures (including some unpublished papers) to add to your understanding and enjoyment. The Appendix also offers input files, proofs, and fragments of output files. Finally, on the CD-ROM included in the two-volume companion book, you will find my papers, as PDF files.

This book serves well as a text, as a tutorial for the field of automated reasoning, as a guide to using a most powerful reasoning program, as a source of research problems and open questions, and as a history of my work. For example, in regard to use as a text for courses, Chapters 2 through 7 offer numerous exercises whose answers are given at the end of the respective chapters. You are, in fact, offered here a complete menu that addresses diverse interests and varying appetites.

1.2 The Book's Audience

The book addresses a wide audience: students, teachers, researchers in automated reasoning, researchers in mathematics and logic not necessarily interested in automated reasoning, historians of computer science, and those whose objective is simply to experience excitement and a sense of wonder.

Students, from the novice to the near-expert. Throughout the book (especially in Chapter 10), problems are posed, hints are given, and answers for many (but not all) problems are given. Some of the problems test your understanding of aspects of automated reasoning, but some of them are appropriate for testing understanding of mathematics and logic. Of a different cast, some of the problems are at the level of a Ph.D. thesis—the solution possibly meriting a doctorate. Also, to help familiarize you with the use of OTTER, included are problems to expand your knowledge of that program.

You might wonder how much background is required by this book. The answer is, None. Quite true: For those of you who have absolutely no background in logic or the use of computers, Chapters 2 and 3 take you from zero knowledge to more than a moderate amount; a later chapter, Chapter 9, guides you in making wise choices for effectively using the reasoning program OTTER.

More good news: If you are asked in a course to use OTTER to obtain answers to posed problems, you will find the input files most useful. They will serve as templates, and, in some cases, you will be able to simply download them and make small modifications to achieve your objective. By inspecting the included output files (on the CD-ROM), you will gain much insight into how OTTER works and what to expect.

Teachers, of automated reasoning, of logic, and even of mathematics. Because this book is self-contained, and because it offers many problems of varying difficulty, you will find here what is needed for both graduate and undergraduate courses. For example, although you may have no interest in automated reasoning, you might find Chapter 2 useful by itself, and you might find Chapters 10 and 11 stimulating for the open questions and hard problems they offer. Chapter 2 provides a treatment of logic—including an introduction to the logical connectives **and**, **or**, **not**, and the like—that uses examples from ordinary aspects of life. (You might be encouraged to know that various professors used a chapter of this type, written by me in an earlier book—and the students applauded.)

You might also find this text useful for motivating students to prove various theorems from group theory, ring theory, lattice theory, and Tarskian geometry. Chapter 6 focuses on proving theorems from areas of mathematics; Chapter 7 focuses on proving theorems from areas of logic. You might have the students examine OTTER's output, which shows precisely the axioms, lemmas, and properties of a proposed theorem used for the deduction of each conclusion. Often, from such an examination, students gain a better understanding of what is required for a proof

to be sound and complete. (If further detail is needed, such as which variables are replaced by which expressions, students can obtain that too, simply by requesting what is called a proof object.) Moreover, when a reasoning program such as OTTER finds a proof dramatically different from expected, students may develop an appreciation of the beauty and intrigue of mathematics.

OTTER may also enable some student to discover a new, a far shorter, or a sharply different proof; see Chapter 11 of this volume, and see (Wos 1996a) for additional candidates. The following anecdote is illustrative. Branden Fitelson of the Philosophy Department of the University of Wisconsin was interested in the three-axiom system of Frege in the context of implication. The third axiom was known to be dependent on the first two, the proof being found by Lukasiewicz half a century later. Using OTTER and various strategies, Fitelson produced a proof of essentially the same complexity, one of length 8 and level 7. Fitelson's proof, however, shares but two formulas with that of Lukasiewicz: and, of course, the two proofs must share at least one—the last, which is the goal. Fitelson was unfamiliar with the Lukasiewicz proof at the time; nevertheless, spending but a few hours with OTTER brought him victory. Moreover, whereas Lukasiewicz used lemmas Frege had proved, Fitelson included no lemmas in his input to OTTER.

Fitelson's success may have wide significance. Specifically, various complex proofs exist in the logic literature with the conjecture that their complexity is inescapable, inherent in the discipline. Fitelson strongly believes that OTTER will be useful in refuting this conjecture by finding far more elegant proofs than currently known; my research clearly supports his view (see Chapter 11 and, for one example, the discussion of many-valued sentential calculus in Section 11.4.3). Would it not be a most charming event to have some student in the study of some field of mathematics or logic find a proof substantially more elegant than previously known? OTTER may provide the key.

Researchers, in automated reasoning, in mathematics, in logic, and in other fields. In this book, among the problems that are posed are some whose solution is currently unknown. Some of them ask for a conjecture to be proved or to be refuted; some ask for a proof of a purported theorem; some ask for a more elegant proof than is found in the literature. You may find them intriguing even if your interest in automated reasoning is zero. Use of a program is certainly not required.

On the other hand, if you are conducting research in, for example, lattice theory, OTTER may be of monumental assistance. This program has clearly proved to be a valuable reasoning assistant for the research of mathematicians that include Ken Kunen of the University of Wisconsin Mathematics Department and R. Padmanabhan of the University of Manitoba Mathematics Department.

In the obvious way, far more important than the tireless effort that OTTER can supply is its flawlessness and attention to detail. For example, if you doubt that a proof produced by this program is correct, you can ask for copious detail that permits every aspect to be checked thoroughly. Regarding the soundness of

its reasoning, who wishes to be the subject of a paper entitled “On an Error by MacLane”, as once occurred?

Finally, the input files found throughout this book and on the included CD-ROM will give you a fine start for tackling various areas of mathematics and logic. The output files that are given will provide much insight. The output files you produce from your research will often hold the key to which options should be changed to achieve success. Also, an examination of an output file sometimes reveals an incorrect or undesirable assumption made in presenting the problem to the program.

1.3 The Map

Based on your knowledge, background, interests, and objectives, you map your journey through this book. Indeed, in the country of automated reasoning within the vast world of computing, you may decide to linger awhile in one place, briefly visit others, and completely avoid the rest. To a great extent, the chapters are independent of each other. To aid you in continuing your chosen route or, if such is your wish, in modifying it, both the book and the included CD-ROM contain many signs (pointers) along the way, billboards, and even advertisements. For one example, you might browse in the included output files, which might give you new ideas. Indeed, some of my ideas have come from a reading of an output file; see (Wos, Robinson, Carson, and Shalla 1967) on demodulation.

The uninitiated. If you have the money (measured in time and energy) and the inclination, and if you know essentially nothing about the field, an excellent route to travel begins immediately after this section and includes long stops in Chapters 2 and 3. After you make those two stops, you will have more knowledge and understanding than some who claim to be citizens of automated reasoning. If you complete the grand tour recommended especially for those new to the field, which means reading the chapters on various applications and reading the vignettes, you will know much more than many of the so-called natives. At that point, through the written word, you will be able to communicate with OTTER, give the program advice, and learn from its output about success and, perhaps not inordinately often, about failure. Because you will have read the chapter giving guidelines for the program’s use, browsed among various given input files, and glanced at some of the given output files, you will be well ahead of many of OTTER’s users.

On the other hand, if you are an uninitiated traveler but you prefer a formal itinerary, the recommended route should be modified slightly. Specifically, after you read Chapter 2—but before you read Chapter 3—you should turn to Chapter 8. That chapter gives the formal theory on which automated reasoning is based, in fact, presents it in spades.

The traveler with some experience. A third route through this book is that for the individual who is familiar with logic but unfamiliar with automated reasoning. As you have probably guessed, you bypass Chapter 2. If you are such a traveler, you

can either stop and spend some time in Section 3.1 to review logic at a rapid pace (in contrast to the detailed treatment given in Chapter 2), or you can immediately turn to Section 3.2. Of course, depending on whether or not you prefer very formal education, you can choose over both routes that of turning to Chapter 8.

The seasoned traveler. Next in focus is the person who is quite familiar with automated reasoning and whose objective is the effective use of OTTER. Such a visitor might enjoy the trip that begins with the guidelines chapter (Chapter 9); features input files, proofs, and output files (on the CD-ROM); and offers stops in the applications chapters (Chapters 4 through 7). The CD-ROM plays a key role for this route, providing files for downloading. In addition, various parts of the book offer hard problems and open questions for research and, if enough progress is made, for the focus of a possible publication. In this regard, the vignettes chapter (Chapter 10) is pertinent, and, even more, so is the open questions chapter (Chapter 11).

The hedonistic traveler. Sometimes a person visits a country not with an interest in its language, customs, or sights but with a desire to taste something new or to savor something familiar. For such a hedonist—for example, the mathematician or logician who has no interest in automated reasoning—this book offers an attractive trip or even an intriguing vacation. In particular, the food and drink of interest consists of problems whose solution is unknown and questions whose answer remains elusive; Chapters 10 and 11 are recommended. For these problems and questions, you need not allow OTTER to provide any assistance; just attack them unaided, if you like.

As with most journeys, this book admits other routes to travel. For example, if you are visiting the country of automated reasoning in the role of historian—in contrast to the other discussed visitors—your route through this book might consist mainly of a single stop, in the vignettes chapter (Chapter 10). You might also spend some time on the CD-ROM, specifically with the article on the history of automated reasoning from an Argonne perspective (Wos 1999b) or with some papers of mine (found in the second book consisting of two volumes, entitled *The Collected Works of Larry Wos*; those papers are also found on the CD-ROM included in the second book.

Summing up, *you* make your choice for the reading, whether you are a student, a teacher, a researcher, a historian, or simply a person who enjoys exploring the frontier of science.

1.4 Reasoning in Review

In this book, reasoning refers to *logical reasoning*, to drawing conclusions only if they follow inevitably from the hypotheses in use. (Probabilistic reasoning can be applied by a program such as OTTER, but that type of reasoning is not of concern here.) Mythical people, such as Mr. Spock of *Star Trek* and Sherlock Holmes,

are admired for their careful and flawless reasoning, always obeying the laws of logic. The reasoning of real people, unfortunately, is often less sound. Consider the following examples.

Example 1

Married females are wives.
Nan is a married female.
Therefore Nan is a wife.

Example 2

Married females own hotels.
Nan is a married female.
Therefore Nan owns a hotel.

Spock and Holmes conclude that both examples are logically sound. (In case some doubt exists, statements similar to the first hypothesis in the first example implicitly mean “for all”, in this case for all married females; were such a statement intended to mean “for some”, such a restriction would be given explicitly.) Spock’s friend, however, objects about the second example: she knows several married females who do not own hotels, and she therefore argues that the reasoning is flawed. Is she right?

No, she is not. The reason is that the correctness or soundness of a reasoning step does not depend on whether the hypotheses being used are true. Indeed, the soundness of reasoning is independent of the truth or falsity of the hypotheses in use; the hypotheses are merely accepted as if they are true. The reasoning in the second example is in fact sound, even though one of the two hypotheses is obviously nonsense.

Now consider the following examples of possibly logical reasoning; the verdicts are given after the examples to give you a chance to vote.

Example 3.

Apartment 3809 contains two bedrooms.
Nan is not in the large bedroom.
Therefore Nan is in the small bedroom.

Example 4.

Your girlfriend’s sister has two children.
One of them is a girl.
The gender of the other child is therefore known.

The verdicts are in. The third example is an example of unsound reasoning. Because of the way the items are presented, some people might—in haste—think that Nan must be in one of the two bedrooms and must, therefore, be in the small

bedroom. As has been said so many times, people jump to conclusions. Here, for example, nothing rules out the possibility that Nan might be in the kitchen—nothing.

As for the fourth example, although you most likely did not take a position—at least suspecting a trap—many people would have. But logical reasoning yields *no* position to take. In fact, the unspecified child *could* be a boy, but, equally, that child could be another girl. The unwarranted and not logical taking of a position can easily be explained. Either the person who errs assumes that the utterance, in the case of two girl children, would have demanded something like “both children are”, or the person who errs (in effect) hears the word “only” as part of the hypothesis.

The mistakes in these last two examples would *never* be made by the automated reasoning program OTTER. The explanation rests with the fact that, other than in the context of equality (discussed later), OTTER makes no assumptions and treats no information as being present implicitly. You now have a taste of why a program such as OTTER might be of interest to you and might prove to be a valuable assistant. This assistant is tireless, never impatient, and incredibly fast, and it makes no errors (as far as the evidence shows).

1.5 Reasoning by Computer versus Reasoning by a Person

Having learned that OTTER draws a conclusion only if it follows inevitably from given hypotheses, you might wonder how OTTER’s reasoning compares with that of a person. Simply put, differences abound between the way a person reasons and the way a program of the type featured here reasons. Those differences may in part explain why OTTER has succeeded in answering questions that were unanswered for decades, and also explain why its use has produced proofs far more elegant than previously known. (Even if I knew what was needed, I would not redesign OTTER to function as a mathematician, logician, or any other person does, and not because of a lack of respect for people’s reasoning.) A few examples are in order, to illustrate the differences between computer-oriented reasoning and person-oriented reasoning.

One type of reasoning—an inference rule often used by people but not by the type of reasoning program featured here—is called *instantiation*. As an illustration, by applying instantiation to the statement “all people are female or male”, you can conclude that “Kim is female or male”. Of more substance, from mathematics, the equality $yzyz = e$ (the identity in a group) is obtained by instantiating the equality $xx = e$, by instantiating (replacing) the variable x by the expression yz . (My colleague McCune observes that people can use instantiation well because they can look ahead to see what will be needed for a unification that they can tell will be of interest.) Because instantiation requires the uniform replacement of variables by terms, in the preceding illustration, the conclusion from a strictly technical viewpoint is $(yz)(yz) = e$.

I imagine that you see nothing wrong with the illustrated type of reasoning. In fact, you might be almost certain that the inference rule is logically sound, never yielding a conclusion that fails to follow inevitably from the single hypothesis in use. You are indeed correct: The rule is just fine, and, regarding use, it plays an important role in fields that include mathematics. But—and here you find a sharp difference between the way a person often reasons and the way OTTER, for example, never reasons—the rule is not among this program’s arsenal. Obviously, soundness is not the problem.

The explanation rests with the lack of an effective strategy for controlling the use of instantiation. Indeed, although people—especially mathematicians and logicians—often brilliantly pick just the needed instances of some formula or some equation, to this date nobody to my knowledge has formulated an effective strategy for a program to do so. In particular, no strategy exists that captures the way the skilled individual knows that the instance $yzyz = e$ of the equation $xx = e$ is just what is needed to begin a proof that commutativity is present for groups of exponent 2. Without such a strategy, the reasoning program can get totally lost among the usually infinite set of instances and can wander fruitlessly among unwise choices of replacements for variables in an equation or a formula under examination. Therefore—in contrast to the way people reason—automated reasoning programs do not offer instantiation, nor should they. But be not dismayed: OTTER does offer sound inference rules for drawing conclusions, as you will see shortly (in Section 1.6.2); see also Sections 1.6.12 and 1.7.

Another difference between the reasoning applied by a program such as OTTER and that applied by a person focuses on assumptions. As noted earlier in Examples 3 and 4, people too often make assumptions, whereas (with the exception of properties of equality) the type of program in focus here does not. Its lack of assumption making is an asset and not a liability. You need not check its reasoning for hidden assumptions as you must with a person. Such a program does not even assume that people are female or male. Nor need you check the reasoning of your automated assistant for flaws; there will be none.

Of course, you are right in noting that some assumptions or guesses or conjectures can prove most useful when attacking a problem. In that regard, a person’s reasoning can have an advantage over a program’s. By combining the advantages a person has in the context of reasoning and those a program has, an impressively strong team can be formed. Indeed, one interesting use of OTTER has you make a conjecture and have the program produce a large number of consequences that you then peruse for desirability.

For a simple example of what might occur, imagine that you have told (in the input) your reasoning program that Kim is a female and is Tom’s husband. If the program knows some of the fundamental properties of people and husbands, among its immediate conclusions is that Kim *is not* male. But, from what you said about Kim being a husband, the program also deduces that Kim *is* male. Your assistant

announces that a logical contradiction has been found—but it does not judge you poorly or accuse you of being of unsound mind.

The ability to find logical contradictions is not only useful for establishing an inconsistency in a set of hypotheses, it is often the key to recognizing the completion of an assignment. For example, when the assignment is that of proving a theorem (from mathematics, logic, or some other discipline), you give the program the axioms and lemmas and other data that are pertinent and also give it a statement or statements that correspond to assuming that the theorem is false. The program then begins its attack with the goal of finding a proof by contradiction, a style of proof well known in graduate courses, in textbooks, and in published papers.

At this point, you may yourself be able to suggest additional possible differences between the way a person and the way a program reasons. In particular, you may be thinking of insight, intuition, formulating conjectures, and recognizing patterns of reasoning. All of these occur for people; none of these occur for your reasoning assistant. Indeed, the type of program featured here does not learn; it merely—said with amusement—reasons with astounding speed, accuracy, and energy. But it can take advice from you if you choose to use *weighting* or *hints* or *resonators*, each discussed in this book.

One other key difference between a person's attack and that of an automated reasoning program such as OTTER deserves especial mention. Various mathematicians I have questioned are often unable to explain the strategy they used to attack research problems. OTTER, on the other hand, makes heavy and *explicit* use of strategy. (By the way, I am certain that people do use strategy in research and in other areas, but seldom is the strategy explicit and seldom describable.)

I find strategy to be one of OTTER's finest delicacies; many reasoning programs unfortunately offer little or none. For an appetizer, the first strategy formulated to direct a program's reasoning, the *unit preference strategy* (Wos, Carson, and Robinson 1964), emphasizes the role of information that in a sense is the simplest possible, information that offers a single choice. For example, "Kim is female" and "Judy is not a mathematician" are in this context each simpler than "Kim is female or male"; the last of the three statements offers two choices, the key being the presence within the statement of "or".

Of a sharply different and piquant flavor is the first strategy formulated to restrict a program's reasoning, namely, the *set of support strategy* (Wos, Robinson, and Carlson 1965a). If used in the recommended manner, the strategy prevents the program from drawing a conclusion from hypotheses all of which are among the general background information of the field in focus. For a strategy with an exotic cast, you can employ *resonance* (Wos 1995a), allowing you to supply patterns of information that give preference (for focusing the program's reasoning) to information that matches any of the supplied patterns. Such strategies are indeed crucial when attacking deep questions and hard problems.

Nevertheless, despite these powerful and diverse strategies offered by OTTER,

no claim is made about the existence of a panacea. Indeed, quite the contrary; far more power and ease of use will be offered by the program of the future when and if various obstacles have been overcome. Therefore, the big question focuses, for the past, present, and future, on what are the obstacles to the automation of reasoning.

1.6 Obstacles to the Effective Automation of Reasoning

That logical reasoning has been so successfully automated is miraculous in my view. Had I been asked in 1960, my familiarity with computers and with mathematics would have led me to believe it was essentially impossible. Clearly I would have been wrong.

You might immediately wonder what was in the way, what were the obstacles to an effective automation of reasoning. This section focuses on obstacles, those of the early days when the field was called mechanical theorem proving, and those that still exist. As with the obstacles to any noble endeavor, almost always their conquering is partial. By viewing the difficulties presented by attempting to automate reasoning, you will see why various components are present in the Argonne paradigm featured in this book and discussed in detail later in this chapter (in Section 1.7). You will also be gaining insight into how to use an automated reasoning program, why the paradigm discussed here may be far superior to others, and what more is needed to significantly add to the power of reasoning programs. You may even decide, because of the nature of the various obstacles that are the focus of this section, that the components of the paradigm are indispensable.

The obstacles that beset the automation of reasoning can be quickly summarized.

- 1. Language.** Especially if the goal is access to a single program that is so versatile that it can reason powerfully about design, verification, proof, puzzle solving, and more, a means to present the assignment is the most obvious obstacle. Whether you have written many computer programs or not, you are likely to correctly guess that Fortran, C, IBM assembly, and the like will not provide what is needed for quickly stating the properties, axioms, goal, and such to an automated reasoning assistant. Some other language must be found, sufficiently general to permit discussion of circuits, computer code, areas of mathematics and logic, puzzles, and more.
- 2. Inference Rules.** Another obstacle focuses on the need to draw conclusions that follow inevitably from hypotheses, the need for logically sound rules for reasoning.
- 3. Strategy.** Easily overlooked is the obstacle of at least adequately controlling the reasoning, to avoid drowning in information and to avoid getting lost. A partial solution to the problem focuses on formulating strategy, one type to restrict and one type to direct the use of the inference rules.
- 4. Assignment Completion.** As with any game—and automated reasoning is the finest game in town—or with any computer program, there exists the obstacle

of knowing when to quit; a means must be found of a general nature to enable your reasoning assistant to detect assignment completion.

5. Redundancy. Then the obstacle of retaining identical information but in various equivalent forms is encountered, redundancy, exemplified by “father’s father and paternal grandfather”.

6. General versus Specific Information. An obstacle (pertinent to both the reasoning of a person and that of a computer) concerns the deduction of specific information when general information that captures it is present in the program’s database, exemplified by the fact that $2 + 0 = 2$ and the capturing fact that (for all numbers x) $x + 0 = x$.

In addition to these six obstacles, which are pertinent for people, the following are more pertinent to a computer.

7. Conclusion Retention. Your reasoning assistant can quickly accrue an overwhelming amount of new information.

8. Conclusion Generation. Your assistant can even more quickly deduce an unbelievable amount of information of which much is discardable.

9. Inadequate Focus. A severe obstacle concerns the choice of which information to use to key the program’s attack.

10. Conclusion Repetition. Although your automated assistant avoids duplicating the same path of reasoning, the same item of information can be deduced again and again and again by following radically distinct paths of reasoning.

11. Redundancy-Control Transformations. More than annoying is the obstacle of choosing which rules to use to address redundancy, concerning the equivalent ways of stating an item.

12. Size of Deduction Step. A not-very-obvious obstacle is that focusing on how big of a reasoning step to take; for example, with instantiation, you or your reasoning program usually takes a very small step. On the one hand, deduction steps that are small can lead to a huge amount of information, and on the other hand deduction steps that are large can bypass a key item.

13. Metarules for Program Use. A most formidable obstacle concerns the choices to be made in the problem representation, the type of reasoning, the means for controlling the reasoning, and the like.

14. Indexing. Finally, directly germane to implementation, there exists the CPU-time obstacle focusing on accessing the database to locate the appropriate information.

In the following subsections, each of these obstacles is discussed in detail.

1.6.1 *Language*

Perhaps the most obvious obstacle that existed in the early 1960s to the automation of logical reasoning was language. In the context of the three problems posed at the beginning of this chapter, how could you tell a computer program about billiard

balls, gates and circuits, and Robbins algebra? Indeed, had I been asked, say, in 1960 whether it were possible, despite my experience with various programs I would have been far more than skeptical.

For the type of reasoning program featured here, the obstacle was overcome by adopting the *clause language* (detailed in Chapter 3, and whose choice was one of the most important contributions of J. A. Robinson). (At various points in this book, the term “clause” is used less formally, simply referring to one of the notations acceptable to OTTER.) Although that language could hardly be called rich or natural, from a practical standpoint it more than serves the purpose for presenting many questions and many problems to a reasoning program. From a theoretical viewpoint, the clause language is adequate for all situations, if you are willing to rely on an appropriate set theory and pay the price of the corresponding inefficiency. (If some researcher finds a means to sharply reduce this inefficiency, one of the most significant contributions to the automation of logical reasoning will have been made.) Rather than a disadvantage, the lack of richness of the clause language (in my view) promotes the formulation of effective inference rules for drawing conclusions and powerful strategies for controlling the application of those inference rules. Although disappointing to many, especially those devoted to classical artificial intelligence, natural language (to me) would be a poor choice from the viewpoint of the computer.

Despite the marvelous properties of the clause language—and here you encounter the fact that solutions are usually not total—a linguistic obstacle still exists for the user of a reasoning program. Stated mildly, you will find it somewhat unpleasant to use the clause language, or even to state the assignment in first-order predicate calculus to then be translated by the program into the clause language. Without doubt, most people would prefer to present an assignment in the language appropriate to it, natural language for database queries, gates and the like for circuit design, and some dialect of mathematics for proving theorems and such. That capability is currently unavailable; in other words, such friendliness will not be found yet in automated reasoning. So an obstacle focusing on language still exists, at least for the user. If you formulate a methodology that severely reduces this obstacle, thus permitting the user to use the language appropriate to the problem under study, you will have made a most significant contribution.

1.6.2 *Inference Rules*

The next obstacle in the early days of the field concerned the type of reasoning a computer program could apply to draw conclusions. As discussed in Section 1.5, the inference rule instantiation is indeed *not* the type of reasoning that was needed, even though it is frequently used by people and serves well in mathematics and logic, for example. From experiences in some course, you might justifiably and immediately suggest two familiar-to-some inference rules, syllogism and modus

ponens. Regarding the former, syllogism, have you not seen at some time something like the following?

major premiss: All apples are fruit.
 minor premiss: All McIntoshes are apples.
 conclusion: All McIntoshes are fruit.

(For those who enjoy crossword puzzles or are devoted to precision or wonder whether an error has been made, the spelling of “premiss” is that recommended by the famous logician Alanzo Church; no, I am not following British tradition.) As for modus ponens, which differs only slightly from syllogism, you no doubt have seen something like this.

major premiss: All tiny tigers are a delight.
 minor premiss: Our little girl was a tiny tiger.
 conclusion: Our little girl was a delight.

The distinction is syntactic, with syllogism having the form “P implies Q” and “Q implies R” yields “P implies R”, and modus ponens having the form “P implies Q” and “P yields Q”. You might examine the four examples given earlier in Section 1.4 and classify each as an example of syllogistic reasoning or reasoning by modus ponens.

A breakthrough in computer reasoning (due to J. A. Robinson) occurred in the early 1960s with the formulation of an inference rule called *binary resolution*. As you will learn in Chapter 3, this rule nicely generalizes simultaneously both syllogism and modus ponens and, more important, adapts well to the application by a computer. Whether Robinson had in mind the two classic inference rules of logic will almost certainly remain a mystery; indeed, questioning an originator of an idea thirty-five years after its formulation yields uncertain information. Other types of reasoning soon followed, each well suited to a computer, and another obstacle had been overcome in the main.

But again the solution is only a partial solution. For but one example, although the inference rule *paramodulation* (see Chapter 3) nicely addresses the building in of equality at the reasoning level, still no comparable inference rule exists for set-theoretic reasoning.

1.6.3 *Assignment Completion*

Next came the question of finding an effective test for assignment completion. Brutally put, the computer program has no idea what it is doing, or why. The solution was to use a common style of proof in mathematics: proof by contradiction, deducing two statements that clearly cannot hold simultaneously. For example, the program may deduce the following two statements: “everybody lives in Chicago” and “Kim lives nowhere”. Proof by contradiction remains an excellent solution for

assignment completion. (In Chapter 3, you will learn in detail about the way a reasoning program detects the completion of a proof by contradiction.)

1.6.4 *Strategy*

The next obstacle that arose was the need to control the reasoning. Indeed, simply having a program deduce one conclusion after another with no regard to the path of reasoning or the goal being sought would in the vast majority of cases lead to disaster, to drowning the program in a huge number of conclusions without ever completing the assigned task.

The attack on this obstacle of uncontrolled reasoning marked my entrance into the field. In particular, strategy was deemed (by me) to be a partial solution. The word “partial” is more than apt, for even today’s most powerful program can get lost on its way to an assignment completion or can drown in unwanted drawn conclusions. Section 1.5 briefly focused on the first of a long line of strategies that are now used, namely, *unit preference*; see the *vignettes chapter* (Chapter 10) for more detail concerning various strategies. Of the aspects of automated reasoning, even today (here in 1999) my favorite is still strategy, and I consider my introduction of its use as perhaps my most important contribution. (As an aside for the curious, still in 1999 far too many researchers in automated reasoning underestimate the need for strategy. If you are familiar with some area of mathematics or logic and have experienced its depth and complexity, without any further argument or actual data, you might well share my disbelief at the reluctance to accept the obvious: Strategy is indispensable.)

In the context of a partial solution, far more must be accomplished in the area of strategy. In fact, you can correctly view this area as a mine whose riches will never be exhausted.

1.6.5 *Redundancy*

The next obstacle (in the mid-1960s) focused on an aspect of redundant information. If your friend or your reasoning program deduces that your mother’s mother is 57 years old and then later deduces that your maternal grandmother is 57 years old, redundancy is present; although the words are different, the subjects and content of the deductions are identical. If I or a program draws the conclusion that apples are fruit and later draws the conclusion that *pommes* (French) are fruit, the second conclusion is redundant.

The retention by a program of redundant information may be little more than annoying in many cases. Its presence, however, can spawn an avalanche of unneeded information. Indeed, seldom is there profit in the retention of the facts that $0+a = a$ (for a constant a), $0+0+a = a$, $0+a+0 = a$, ad nauseum. Yet such was occurring in the mid-1960s; and, worse, the redundant information spawned an avalanche of unwanted conclusions, more than 1900 out of 2000 that were retained.

To sharply reduce the severity of this obstacle (and directly prompted by the just-cited item of data), a procedure called *demodulation* was formulated (Wos, Robinson, Carson, and Shalla 1967). That procedure is used to canonicalize and simplify information. But its use can lead to problems for your reasoning assistant. After all, it trusts you almost all of the time. If, for example, you tell your assistant to treat the concept of your father's brother as being synonymous with the concept of uncle, no objection will be raised. Nevertheless, because the first is a proper subset (in the obvious sense) of the second, conclusions that do not correspond to accurate information might be deduced.

1.6.6 *Specific versus General Information*

Of a somewhat related nature is the following. If a program deduces and retains the fact that dogs have four feet and later deduces that the dog Rembrandt has four feet, the latter deduction is of little value. Its lack of value may not be transparent until you are correctly told that reasoning programs of the type in focus here prefer the general to the specific. Further, as you will learn (when inference rules are in focus in Chapter 3), your reasoning assistant's preference for generality pervades its approach to drawing conclusions, which in turn adds greatly to its power.

If you wonder whether this type of redundancy can lead to monumental inefficiency, imagine that your program—instead of purging specific information captured by a general fact it has retained (such as $0 + x = x$)—accrues $0 + 1 = 1$, $0 + 2 = 2$, $0 + 3 = 3$, and so on through the positive integers. Fortunately, as you will learn in Chapter 3, the obstacle in focus was essentially eliminated by the formulation of a procedure called *subsumption*. (thanks to J. A. Robinson).

1.6.7 *Conclusion Retention*

No doubt you have heard the overwhelmingly insightful observation that numbers tell the story. If you believe there is some truth to the observation, perhaps you will experience (as I did) the astonishment when I found that an experiment with OTTER failed to complete the assignment even after retaining more than 300,000 new conclusions. The result is more impressive, or more disappointing, when you learn that many constraints and various strategies were in use at the time. Further, you might then share my exultation at learning that success occurred after the retention of more than 390,000 new conclusions (in roughly 10 CPU-hours on a fast computer). (The task, which was successful, was that of finding a new and shorter proof in many-valued sentential calculus, a proof in the style of Meredith. The proof has length 37, completing with a clause numbered 390,683.)

For an example from a different area of logic, OTTER found a proof upon retention of a conclusion numbered 273,565 after more than 236 CPU-hours. The proof has length 251 and concerns the deduction of a formula in two-valued sentential, or propositional, calculus, derived from Meredith's single axiom (Meredith 1953),

given in Section 11.1.

Without placing limits on the complexity of retained information, without relying on procedures to purge deduced information that is captured by already-retained and more general information, and without using various strategies to curtail the accrual of new conclusions, a quick calculation shows that no integer is large enough to represent what would occur.

That conclusion retention is an obstacle does not rest with memory considerations; rather, the concern is CPU time required to complete assignments. Indeed, in contrast to the 1960s and the 1970s, here in 1999 I often run OTTER in an environment offering 128 megabytes of memory. The presence of many, many retained conclusions costs much CPU time, not because of the simple process of retention, but because of the side effects. Should you formulate an approach that dramatically reduces the number of conclusions retained on the way to assignment completion without paying much of a price measured in CPU time—and, of course, without losing effectiveness or success rate—you will almost certainly have made a significant contribution to the field of automated reasoning and to the power of reasoning programs.

In considering the issue of conclusion retention, you might naturally wonder why new conclusions need be kept at all. Indeed, approaches based on logic programming, for example, do not retain new conclusions. But approaches that retain no new information are fraught with redundant reasoning and, far more significant, are seldom successful when attacking a deep question or a hard problem.

You might appreciate a break from seriousness and enjoy the following amusing anecdote concerning yet another approach (based on connection graphs) that differs sharply from that taken in this book and discussed in Section 1.7. In the early 1980s, Joerg Siekmann was visiting the Argonne group and announced with pride that he and his colleagues could prove one of the then-benchmark theorems with the retention of but 18 new conclusions not in the proof, in contrast to our program's retention of 171. I countered with the announcement that our newer and hidden program could prove this theorem with retention of no unneeded conclusions, causing Joerg to express amazement. He could not resist and asked how we had made such a breakthrough. I said, after pausing for dramatic effect, that, well, our program, upon completing a proof, delayed displaying the result—until all unneeded conclusions were purged. After briefly taking my remarks seriously, Joerg recognized my put-on, my way of maintaining that CPU time is far more important than clause-retention data. (For the perfectionist, no procedure exists that produces proofs always without retention of unneeded conclusions.)

1.6.8 *Conclusion Generation*

The obstacle of rapidly deducing conclusions so many of which are useless in the context of completing the given assignment is indeed formidable. In one of my

experiments that failed, the program in use deduced more than 725,000,000 conclusions, most of which were discarded for a variety of reasons. Among other reasons, a newly generated conclusion may be discarded because of being identical to one already retained, or because of being captured by a conclusion already retained that is more general, or because its complexity exceeds the limit assigned by the researcher, or because it contains too many distinct variables (when compared with the limit assigned by the researcher).

Unfortunately, the program cannot determine before the fact that an application of an inference rule is about to yield a conclusion that will then be discarded for one reason or another. Therefore, your reasoning assistant can squander an incredible amount of CPU time generating conclusions that are of no use. Somewhat redeeming and sharply different from fifteen years ago, OTTER's rate of generating conclusions per CPU-second drops relatively little even after 19 CPU-hours, for which McCune merits substantial acclaim. (Technically, OTTER offers local linearity.) Fortunately, OTTER is extremely fast. Moreover, various strategies have been formulated to reduce the severity of this conclusion-generation obstacle; see the vignettes chapter (Chapter 10) for examples. Many more strategies are needed, and many more (I predict) will be formulated.

1.6.9 *Inadequate Focus*

When you are attacking a problem and know a number of possibly relevant facts, how do you choose wisely which of them to key upon to increase the likelihood of finding a solution and at the same time to sharply decrease the time you spend? If you are deservedly respected for solving puzzles, debugging computer programs, designing circuits, proving theorems, or some such activity, you make your choices based on intuition, experience, guesses, and perhaps some luck. Although an automated reasoning assistant such as OTTER offers you a wide variety of reasoning mechanisms (inference rules), numerous ways to control the reasoning (strategies), procedures to address the obstacle of redundancy and similar obstacles, and far more (as you will learn), the program lacks intuition, does not accrue experience whether it succeeds or fails, cannot guess, and knows nothing of luck.

Because your program must continually select some item of information to drive its reasoning, to initiate each application of an inference rule, it faces the obstacle of choosing where to focus its attention but without access to one of the mechanisms you use. If its choices are in the main poor, the given assignment will almost certainly never be completed. Indeed, luck seldom is much of a lady when attacking some deep question of mathematics, for example. Fortunately, OTTER offers you three means—one called *weighting*, a second called *hints*, and a third called *resonance*—that enable you to impart some of your intuition and experience in the form of advice to assist your assistant. However, in contrast to local linearity, OTTER does not offer *global* linearity; indeed, the number of items it chooses per

CPU-second to drive its reasoning can sharply decrease after a few CPU-hours. Were you able to find a means for significantly reducing the obstacle of lacking global linearity, you would merit great praise, and you would be well advised to submit your breakthrough for publication.

1.6.10 *Conclusion Repetition*

Although arriving at the same exact conclusion a second time, much less for the thousandth time, is more than pointless—CPU time is wasted—your reasoning assistant faces this obstacle continually. If you happen to wonder whether some form of useless looping is occurring, banish the thought. The problem is that many, many diverse paths exist that begin somewhere in the hypotheses (supplied by the user) and end with the identical conclusion. Indeed, in the context of mathematics and logic—from my experiments with an automated reasoning program—I have learned with utter amazement how rich is the space of proofs for a given theorem. I strongly suspect that even the well-respected mathematician or logician would experience the same surprise, given the facts.

Similar to what was observed for the obstacle of conclusion generation, the program cannot tell before the fact that an application of an inference rule will yield a conclusion already present in its database or already drawn and rejected for some reason. For example, if you know that Gail lives in Chicago or Detroit or New York and also know that she does not live in New York nor in Detroit, you deduce that she lives in Chicago. Your reasoning assistant easily comes to the same conclusion. If you know that Gail is a logician or Gail is a millionaire or Gail lives in Chicago, and if you know that in fact she is neither a logician nor a millionaire, again you deduce that she lives in Chicago. Your assistant also draws a conclusion it already has drawn.

Annoying though this example may be, it is not unusual. Indeed, rather than two unrelated paths of reasoning converging as shown, sometimes one hundred or more diverse paths converge.

Of a similar nature is the case that you already have in mind in which the program deduces an item only to find that an existing item in its database captures it trivially. For an example that might not immediately occur to you and that might provide a bit of entertainment, consider the case that you or your automated assistant learns that Gail lives in Chicago. Later you or your program learns that Gail lives in Chicago or lives in New York. That discovery produces little excitement and no useful information. Arriving at that less interesting conclusion took time—and took time to discard it. Although not strictly an example of conclusion repetition, the example can be readily classified as an instance of the same obstacle.

1.6.11 Redundancy-Control Transformations

With a transformation that automatically replaces, for example, “mother’s mother” by “maternal grandmother”, a program can avoid one type of redundancy. Applying such a transformation apparently presents no overhead to speak of and no negative side effects.

Not nearly so obvious is the choice between left associating all expressions and right associating them, specifically, between always storing $(a + b) + c$ in preference to storing $a + (b + c)$. Depending on various other factors, the former or the latter may be the good choice. Because one transformation (for canonicalization or simplification) can interfere with another and because side effects may be possible, the choice of which transformations to use presents a severe obstacle. In fact, an entire field, called *complete sets of reductions*, is devoted to various puzzling questions in the context of transformations.

1.6.12 Size of Deduction Step

If you examine the reasoning that occurs in a typical mathematics paper, you find that much is left to the imagination. Not only is the justification of a step of a proof omitted, but many steps are also absent. You could view this latter phenomenon as an instance of taking very large deduction steps, leaving out intermediate conclusions. Both omissions can materially add to the difficulty of checking the proof that is given. Fortunately, a program such as OTTER explicitly provides the justification including the parents. But, in the sense under discussion, steps can be omitted.

For an example that illustrates the obstacle of choosing the appropriate size for deduction steps and that also illustrates some of the offerings of OTTER, imagine that you are trying to solve a puzzle or a mystery and are asked to add to the store of information about Kim. At one point in your search, you are told that three possibilities exist: Kim is not married, or Kim has no sisters, or Kim lives in Chicago. As you learn more, you can take a small step of reasoning that focuses on eliminating one of the three possibilities, or you can take a larger step of reasoning and draw a conclusion only when two of the possibilities have been eliminated. If the latter, your learning that Kim has no sisters leads you to draw no conclusion; if the former, you draw a conclusion that leaves two possibilities. With a reasoning assistant such as OTTER, you are offered such choices. The inference rule that permits the program to take the cited small deduction step is called *binary resolution*, and for the larger step, the rule is *UR-resolution*; see Chapter 3.

If the choice is that of taking very small steps of reasoning, the program can drown in information. If the choice is large steps, then the program can bypass a useful conclusion. In the cited example, with the choice of large reasoning steps, the program would not retain the two-possibility conclusion, but it might later find that such a conclusion could be combined with another two-possibility conclusion

to then lead to a solution to the puzzle or mystery. You thus see why an obstacle of the type under consideration merits attention. (Note that instantiation yields even smaller deduction steps than does binary resolution, yielding conclusions that are captured by the item being instantiated.)

1.6.13 *Metarules for Program Use*

Choosing which options to use and which values to assign to various parameters can be annoying for the user. You may decide for your assistant which types of reasoning will be used (inference rules), how the reasoning will be controlled (strategies), which transformations to use (demodulators), and the like. A virtual disaster can occur if you make odd choices, for example, choose a representation for the problem that emphasizes the use of an equality-oriented notation and at the same time choose an inference rule that is not equality oriented. Fortunately, OTTER has an autonomous mode that permits you to make no choices other than that for presenting the problem or question—and it works quite well.

However, and certainly not unexpected in view of the depth of mathematics and logic and other sciences, the autonomous mode is by no means a panacea. Indeed, when using an automated reasoning assistant for finding a proof of a conjectured theorem or finding a more elegant proof or synthesizing a more efficient circuit, for example, even the expert in the use of a reasoning program often expends substantial time and effort identifying an effective combination of choices.

1.6.14 *Indexing*

You might be astounded to learn how much CPU time can be required (for your reasoning assistant) to locate the appropriate information, such as a needed transformation for simplification, or an existing item that captures a newly deduced conclusion making it discardable. Occasionally confusion still exists regarding the importance of memory compared with CPU time: Here in 1999, memory is astoundingly inexpensive; CPU time is the key ingredient. The indexing scheme employed by a reasoning program can (in too many cases) make or break it.

That chosen by McCune for OTTER more than makes it. Indeed, even after more than 19 CPU-hours on a fast machine, the rate of conclusion drawing is cut only in half.

1.7 **Paradigms for Reasoning and for Research**

At this point, you again are asked to make a decision regarding your route through this book. You can choose to avoid this stop, for the material here is indeed somewhat technical. Instead, if you wish to know now what paradigms are the basis for this book's treatment of automated reasoning and for research, then you are more

than welcome to spend some time here. By stopping here for a while—rather than waiting for your visit to other chapters—you will immediately learn of the approach taken at Argonne from 1963 to the present, an approach that has produced reasoning programs that have at each stage been the most versatile and powerful and (at the same time) an approach that has yielded numerous successes in the contexts of mathematics and logic.

Far more than any other paradigm for the automation of reasoning, the Argonne paradigm emphasizes the role of strategy: some to restrict the reasoning and some to direct it. Compared with other approaches or paradigms, the Argonne paradigm (featured throughout this book and the basis for OTTER) demands access to diverse types of reasoning (inference rules). Rather than natural language or a natural-deduction scheme, the clause language is used to present questions or problems. You can also use first-order predicate calculus and have OTTER translate the presentation into clauses. The Argonne paradigm also rests heavily on the use of procedures that canonicalize and simplify and that purge less general information, in contrast to other paradigms such as that based on logic programming. Finally, a key difference between this paradigm and many others concerns the retention of (often a vast number of) conclusions, sometimes far more than 390,000.

Although crucial to the paradigm is the test for assignment completion, namely, the use of proof by contradiction—so commonly occurring in mathematics, logic, and other paradigms—no attempt is made to emulate the reasoning of a person or the approach taken by people. Among other comparisons, the Argonne approach to automated reasoning differs sharply from that taken in classical artificial intelligence, where emulation of person-oriented reasoning is key. Although I personally support the right of a researcher to pursue the study of approaches taken by people, I must say (for the curious) that programs that reason with great power will all be based on logic, at least for many, many decades. If you wonder why I take such a strong position, note that I have never been told precisely how anybody actually solves problems. The mystery of the mind will remain essentially that for decades, a mystery.

You might now wish an overview of how the paradigm has been programmed, in particular, in OTTER. The following list gives a brief description of the flow of the actions of OTTER 3.0.4 (the version found on the CD-ROM), of course modified by the choice of various options; earlier versions of the program also adhere to the given flow. Note that, although this powerful and versatile automated reasoning program offers not much in the way of interactive features and does not offer induction, the program will serve you well in many areas of study and research. Without its use, clearly (to me) many of the open questions that have been answered in the preceding seven years would still be open.

- Read in four lists of clauses: the usable list, the set of support list, the passive list, and a list of demodulators.
- While the program has not yet completed a proof and clauses are still in

list(sos), perform the following steps.

- Select a clause from list(sos) and call it the *focal clause* (called the *given clause* in the output and in the users manual). A number of methods may be used for selecting the next focal (or given) clause. However, the technique of assigning “weights” to clauses and then selecting the “lightest” clause is one effective method.
- Move the focal (or given) clause to list(usable), and, using some specified set of inference rules, draw all conclusions that can be obtained. Each deduced clause must have the focal clause as one parent, and all other parent clauses must be selected from the usable list. For each deduced clause, perform the following steps.
 - (1) Simplify the clause, using the available demodulators.
 - (2) Weigh the clause, using the available weight templates in weight_list (pick_and_purge) and in weight_list(purge_gen), and discard the clause if its weight exceeds max_weight.
 - (3) Test the clause to see whether it is subsumed by an existing clause in list(usable), list(sos), or list(passive). If not, add the new clause to list(sos).
 - (4) If the clause is a unit clause, test for unit conflict with clauses in list(usable), list(sos), and list(passive).
 - (5) If the new clause subsumes any existing clauses, delete them from whatever lists contain them.

I strongly conjecture that the array of inference rules for drawing conclusions and the variety of strategies for controlling their application (offered by OTTER) are by themselves sufficient to provide far more than entertainment for the user of this program. If you are eager to see how OTTER can reason in a manner that differs sharply from even a mathematician, and if you are ready for a rather complex example, consider the following use of its inference rule that builds in equality. Be warned that the example is not natural for a person, is fine for a computer to apply, and is not by any means transparent.

Paramodulation *from* the equation $x + (-x) = 0$ *into* the equation $y + (-y + z) = z$ yields in a single step the conclusion $y + 0 = -(-y)$. If you see how the conclusion is obtained, you are indeed well ahead of the game. The following three clauses capture the example.

```

from EQUAL(sum(x,minus(x)),0).
into EQUAL(sum(y,sum(minus(y),z)),z).
conclude EQUAL(sum(y,0),minus(minus(y))).

```

Summarizing, a successful use of paramodulation combines in a single step the process of finding the (in an obvious sense) most general common domain for which

both the *from clause* and the *into clauses* are relevant and applying standard equality substitution to that common domain. The complexity of this inference rule rests in part with (1) its unnaturalness (if viewed from the type of reasoning people employ), in part with (2) the fact that the rule is permitted to apply nontrivial variable replacement to both of the statements under consideration, and in part with (3) the fact that different occurrences of the same expression can be transformed differently. In contrast to being literal oriented, paramodulation is a term-oriented inference rule that generalizes equality substitution.

The Argonne paradigm for the automation of reasoning meshes well with the Argonne paradigm for research in automated reasoning. The latter heavily emphasizes experimentation, thus requiring access to a powerful and versatile program. The experiments frequently focus on a theorem whose proof is already in hand but out of the reach of the best program available (usually one designed and implemented at Argonne). With some proof (not necessarily that in hand) as the target, the research paradigm asks for the eventual formulation of a new inference rule, new strategy, or new technique that succeeds. To be viewed as succeeding, far more than a proof is required: the new item must apply to other theorems besides the target theorem.

In addition to focusing on already-proven theorems, the Argonne paradigm for research in automated reasoning encourages attacking questions whose answer has eluded various attempts. The latest triumph of note in this regard was McCune's use of his program EQP to answer the Robbins question in the affirmative, the question posed at the beginning of this chapter. This book will provide you with numerous other examples and will also feature successes concerned with finding proofs more elegant than thought possible.

1.8 The Future of Automated Reasoning

How different my predictions about the future of the field of automated reasoning are now (in 1999) compared with what they would have been but seven years ago. The cause for the widely different outlook rests with the unexpected successes focusing on answering open questions and finding elegant proofs. Indeed, in various areas of mathematics and logic, OTTER has answered many questions by producing the first-ever known proof or by producing a new proof more elegant than was known; regarding the former, see the McCune and Padmanabhan monograph for example (McCune and Padmanabhan 1996), and for the latter see (Wos 1998). Almost as impressive, and contributing to the optimism that I am about to express, is the use of a program such as OTTER by researchers whose expertise lies outside of automated reasoning. The final peg on which my optimism rests concerns the power and versatility of OTTER and the introduction in the recent years of new strategies; see Sections 10.53, 10.56, 10.58, and 10.59, for example. (You might enjoy the epilogue given in Chapter 12, if you find the preceding remarks stirring.)

Although I am saddened at the number of researchers in automated reasoning who are *not* heavily experimenting with a program, I nevertheless predict that the power of such programs will continue to grow and at a rate far exceeding that of the years prior to 1990. Every week, many copies of OTTER are taken by electronic means. Among the takers, some are using this marvelous program as the basis for further development of the field, extending and modifying it. More strategies will result from attempts to offer increased power. Eventually, a breakthrough will occur in the context of set-theoretic reasoning. Even now, researchers are extending the work of Quaife; see, for example, Belinfante 1999a; Belinfante 1999b. Also active in the study of set theory is the Mizar group; see the Web site www.mizar.org/. (On a grand scale, and of great use, Sutcliffe and associates have organized and maintained a fine database of test problems; see the Web site www.cs.jcu.edu.au/~tptp/.) Linked inference rules will begin to play a key role, as Robert Veroff and others attack the pertinent issues.

The most exciting and gratifying (to me) prediction concerns the users of an automated reasoning program. Specifically, I conjecture that many mathematicians and logicians will routinely use an automated reasoning assistant in the next decade. (As I noted, some do now, even though they are far from expert in the use of such a program.)

As for convenience, for numerous fields and areas, there will exist collections of axioms, lemmas, properties, and definitions. A researcher or user of an automated reasoning assistant will simply draw upon the appropriate collection needed for attacking a question or problem, whether it be from mathematics, from logic, from physics, or from everyday activity. For example, if a person wishes to use such an assistant for improving some design, say in architecture, the appropriate background axioms and properties will be already available. This book provides a small beginning, with the inclusion of various input files for studying group theory, quasilattices, various logic calculi, and more.

The future program will offer a variety of modes, each being automatic in the sense of removing the need to make option and parameter choices. OTTER's autonomous mode is the start. What will be new are modes of varying sophistication, just as you can access chess programs that play at different levels of expertise.

Rather than being restricted to the clause language or first-order predicate calculus, you will be able to converse with your automated assistant in the language appropriate to the study. Algebraists will most likely use a language well known to them, whereas database designers will use a sharply different and familiar language.

The future reasoning program will be self-analytical, a property that has interested me since the mid-1980s. Indeed, given permission by you, your reasoning assistant will modify during the run your choice of inference rules and strategies and the like, based on its estimate of effectiveness. Thus, unwise choices will be quickly recognized and replaced with wiser ones by your program. The future program will also examine its results and present to you, if so instructed, those that merit

particular attention by you.

Nor will the program of the future confine itself to proving theorems that some person had in mind. Indeed, although accurate in the mid-1960s when my friend and colleague George Robinson (now gone) noted that researchers would most likely use an automated reasoning program only to study theorems with which they were familiar and thus miss proving good theorems, beginning now in 1999, such is no longer the case, as the following suggests. When I was recently studying many-valued sentential calculus with the focus on proving two associative laws for the logical connective **or**, two laws studied by Hilbert and Ackermann (1950), OTTER eventually proved a generalization of one of them; see Section 11.4.3. Because Hilbert and Ackermann were focusing on **or**, perhaps they never would have considered the theorem corresponding to the cited generalization. If so, a *good theorem* would have been missed, one that was in fact found by OTTER even though I was not aware of its existence and only seeking a proof of the theorem of which I was aware.

Just as equality-oriented reasoning is built in with the use of paramodulation, set-theoretic reasoning will be built in. This latter class of reasoning is vital to numerous precise arguments, many of which have little or nothing to do with mathematics or logic. Perhaps a new class of linked inference rules will supply what is needed. Certainly, there will exist such rules that address the combining of equality-oriented reasoning with reasoning in which equality plays no role.

Many new strategies will have been formulated and implemented, some to restrict reasoning well, and some to direct it effectively. But I sincerely doubt that very many of the strategies will emulate an approach that is conjectured to be taken by people. Perhaps one or more of the new strategies will reduce essentially to zero the obstacle of focusing, as the CPU-seconds mount, on sharply fewer and fewer items to initiate applications of the chosen inference rules. In other words, global linearity will be offered by the reasoning assistant of the future.

Will this marvelous future program obviate the need for researchers and the like? Ridiculous! No substitute for the mind of the talented will ever exist; only an excellent assistant is possible. But, automated reasoning *will* offer challenges forever—in the realms of theory, of implementation, and of application.