Automated Deduction:
Achievements and Future Directions

D.W. Loveland

Automated deduction (AD) is one of the most advanced and technically deep of the
two technologies that constitute computer science. AD software performs tasks
ranging from the fast simple deductions of a type checker and the efficient explo-
ration of models, to fully automated deduction and complex deductive interactions
in high-level languages. The field lies at an interface with mathematics, logic, and
computing theory, as well as with practice. It has contributed fundamental ideas to
these fields as well as drawing heavily from them.

AD has spawned useful tools and systems, and offers great promise for making sub-
stantial progress on some of the most challenging problems in computer science and
engineering—such as how to improve the reliability of systems, how to build secure
software, and how to increase productivity in software production. In the mathematics
world some previously unsolved problems have been solved by or with the assistance of
an AD system.

We use the term AD for automated theorem provers, counterexample (model) gen-
eration, and consequence generators, in both fully automated and interactive systems.
We note that AD systems may establish truth of a statement by means other than a for-
mal proof (a listing of steps that are given or follow from previous steps), for example
by a decision procedure such as computing and comparing terms in an equation.
Likewise, disproofs are frequently counter examples.

This article is based on a report to the National Science Foundation (NSF), Division of
Computer and Computation Research, on the future directions of the field of AD
[4]. The genesis of the report is the Workshop on the Future Directions of AD held in
Chicago on April 20–21, 1996, chaired by the report author and sponsored by the
National Science Foundation. In addition to workshop input from 24 top U.S.
researchers and users of the technology and systems, information was collected from the

Donald W. Loveland (loveland@cs.duke.edu) is a professor of computer science at Duke University.

The Chicago workshop was supported by National Science Foundation grant CCR-9625544. This article reflects viewpoints expressed at
the workshop by the participants and solicited input from other researchers and users, as interpreted and expanded by the author. The source
of this article is a report to NSF; neither the report nor this article represents policies or viewpoints of NSF.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies
bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

© 2000 ACM 0002-0782/00/1100 $5.00
international community by an Internet call for contributions. Since workshop funds limitations inhibited wider representation of the international AD community, we make no claims that the future directions and recommendations we offer here are the most appropriate assessments outside the U.S. For the reader interested in further details, the full report can be found on the Web\(^1\) and the body of the report (modified) appears in *AI Magazine* \(^3\). Many points mentioned here are better understood in the fuller context of the report. Also, the general reader may enjoy the section of the report devoted to the major achievements in the field, which expands on the highlights we include here.

**Achievements**

Early developments within the AD field have had a profound effect on the artificial intelligence (AI) domain, and, indeed, all of computer science. The first automated theorem prover, the Logic Theorist developed by Newell, Shaw, and Simon in the mid-1950s introduced the linked-list construct. With the Resolution proof procedure in the early 1960s came the unification algorithm, which is a pattern-matching procedure with provably maximal representation capability for large classes of patterns. Unification algorithms have been used in programming languages, constraint satisfaction systems, abductive learning systems, inductive generalization systems and other applications outside the theorem-proving domain. Besides algorithm components and implementation devices first used with deductive systems, some early proof procedures have been incorporated into systems from various domains. Resolution procedures of different flavors are used in many ways, from the more obvious, such as AI-based systems (including natural language and planning systems) to the less expected, such as programming languages. The Model Elimination proof procedure (late 1960s) was a catalyst for the inference engine of the programming language Prolog. The Davis-Putnam proof procedure (late 1950s) is a pre-Resolution procedure of continuing importance, with uses such as theoretical studies (including complexity results) and recent state-of-the-art planning systems.

Recent accomplishments in mathematics have given the field new visibility, such as the 1996 computer discovery of a proof for the Robbins algebra conjecture. Mainstream press coverage is rare in areas related to mathematics, but this accomplishment reached the *New York Times* Science section front page \(^1\). The conjecture was established as a theorem by the EQP system developed by William McCune of Argonne National Laboratory in 1996. The proof was obtained with the computer in fully automatic mode. The problem had been investigated by a number of top mathematicians and was a favorite open conjecture of the late mathematician Alfred Tarski, one of the foremost logicians of this century.

Other mathematical achievements include the study of mathematical structures called quasigroups. Using several deduction programs, both the existence and the nonexistence of various classes of quasigroups have been shown. Also, several open problems regarding axioms for equivalential calculi and other logics and algebras have been solved over the past decade or so.

In a slightly different flavor, we note a system (family) that is more approachable by the general computing public. Geometry theorem provers developed by Chou, Gao and

\(^1\)See www.cs.duke.edu/AutoDedFD.
Zhang prove a large class of difficult theorems in Euclidean and non-Euclidean geometries. A version exists for the PC and Macintosh that is user-friendly enough to be suitable for the nonspecialist. Particularly noteworthy are the readable proofs produced.

Recent achievements outside of the mathematics arena include the verification/specification domain, which employs AD systems in almost pure form, usually in the interactive mode. Computers have also been used to verify the specifications of communications networks, by determining that specifications are consistent (that a communications network meets all requirements) and proving that a designed network meets key properties of the specifications. Safety-critical hardware and software have also been checked by AD systems. These systems are too complex to allow unaided human verification. At the high end of the complexity scale are computer chips. Complete commercial microcontrollers, parts of commercial microprocessors, and their key algorithms have been verified mechanically. However, the largest commercial chips are not verifiable using AD systems at present and probably will not be in the near future, partly because leading-edge chips grow in complexity faster than deductive techniques are developed in the AD field.

Program synthesis is an application area of great potential and one where AD technology is key. The prospect of constructing a suitable computer program automatically, based on specifications, was an exciting dream in the early years of AI. Success in this domain would eliminate the need for program verification, since all synthesized programs would be correct by construction. The task proved overwhelming in the early AI years, and research dropped to nearly zero. Recently, however, there have been a few notable achievements, of which we note two. Douglas Smith of the Kestrel Institute developed the KIDS system, an interactive system used for defining various classes of programs, including scheduling programs. One of the scheduling programs devised by the KIDS system outperformed scheduling programs then in use. The KIDS system contains formal specifications of various algorithm schema and uses problem specifications as input. An inference mechanism transforms input into a precise algorithm, or actual program given control and programming language information. This is done using design tactics under human guidance. Another program synthesis system, the AMPHION system of NASA, uses a theorem prover to assemble subroutines from an existing library and produce working code. The programs aid satellite guidance.

We have previously mentioned ties between AD and programming languages. Programming languages such as Prolog, CLP(\(\mathcal{R}\)), and ECLiPSe are built around a deductive inference engine. These languages belong to a class of programming languages called logic programming languages, where computation is equated with deduction. The latter two languages involve constraint satisfaction. Inductive logic programming (ILP) utilizes the concepts of logic programming together with methods of generalization to learn from examples. The products of ILP systems are programs that incorporate features of the positive examples, while avoiding the characteristics of the negative examples. Applications currently exist in areas as diverse as drug design, finite element analysis, and natural language understanding.

Besides these applications there are many diverse uses of AD concepts and systems in research application systems, particularly in artificial intelligence and deductive databases.

**Observations**

While AD has become increasingly sophisticated and diverse, current applications are...
still less complex than we envisioned several decades ago. All applications attempt to find methods to severely limit the proof search space and/or to limit the deductive component to lower-complexity tasks. Lower complexity is a relative and dynamic term, in that the boundary of the practically achievable constantly moves upward.

Although not all work in automating proof discovery centers on attempts to solve open problems, this aspect of mathematics has received a great deal of recent attention. Solutions to open problems have been found both in interactive mode and fully automatic mode, both by proof discovery and by counterexample production. The AD field is currently divided as to whether pursuing open mathematical problems is the optimal way to increase the capability of theorem provers. To date the mathematical areas with approachable open problems have been algebraic or combinatorial in nature, with our capability to automate proof discovery within continuous mathematics well below the open problem realm at present. In addition, some mathematicians envision uses for AD systems that do not involve major open problems. A recurring dream is to build an archive of formally verified mathematical results that will include the entire core of mathematics. AD systems would be needed to provide formal verification for this quantity of work.

When the push to develop hardware and software verification systems, starting in the 1970s, revealed how greatly we underestimated the difficulty of automating verification, U.S. funding in this area fell sharply. In Europe, however, interest is growing in the use of formal methods for the verifictaion of low-complexity networks, such as concurrent systems. In the microprocessor domain, the focus is on model checking. Model checking uses finite state machinery and temporal logic to represent the processor, and the representation need only be consistent (have a model). Interactive theorem provers are used experimentally to verify that microprocessors meet specifications but completing a single chip verification currently requires many hours of work by a highly trained individual. Program synthesis, which was briefly discussed previously, was a major dream of the early AI community. When the initial optimism wore off, the area was viewed as a blue-sky area with little near-term payoff. A few notable projects have shown that applications are possible if carefully selected and given sustained support. Here the key is carefully constrained deduction requirements.

Clearly, mathematics and logic education can benefit from the work of the AD field. Successful educational products and even automated college courses already exist. Much potential remains to be realized, which will require cooperation with educators. Other areas, such as economics and sociology can profit from use of formal methods to express and evaluate theories. Use includes testing consistency of a proposed theory extension with the core knowledge base and generating ramifications of a proposed extension.

It is important to note the interactions of applications and core development of AD. In particular, work on applications, particularly verification, adds to the core capabilities and hence our research base. Verification proofs are “shallow and messy,” that is, more easily discovered than the mathematical proofs currently attempted, but require a larger collection of proof methods. One consequence is that the verification area leads in integration of different proof techniques within single systems. We are beginning to see theorem provers that integrate inductive proof techniques, general rewrite procedures, model checking, propositional provers, linear arithmetic, other decision procedures, enhanced static checking, lemma generation, and other techniques. Attempts to
incorporate algebraic systems are under way, which should significantly enhance the proof capabilities of verifiers. The integration of techniques, in particular the use of computer algebra systems, should also help in mathematics. Most computer algebra systems currently are unsound, and integration cannot be done without modification.

AD research momentum continues to shift to Europe as funding increases in Europe and decreases in the U.S. (The U.S. decrease includes Defense Department funding of areas such as verification systems.) One measure of the shift is the dominance of the International Conference on Automated Deduction (CADE) by European researchers. U.S. contributions to CADE dropped from 61% in 1988 to 19% in 1996, while European contributions increased from 35% in 1988 to 79% in 1996. (These figures were collected by Pierre Lescanne and Christoph Walther. [2]) Research activity in Asia and Australia is growing and has already yielded significant research contributions. Although some of the best AD systems are of U.S. origin, the number of AD research systems is probably larger in Europe, and the best systems in Europe are competitive with those of the U.S.

Recommendations
The purpose of the report underlying this article was to provide recommendations to funding agencies, research centers, universities, and the AD research community on future directions of AD development and how best to support this development. Following are some of the report recommendations; space constraints prevent us from listing the full set.

Funding agencies need to provide steady funding for basic research, which includes both AD system research and formal methods research, in the following areas:

- Deduction engineering, which primarily addresses incremental AD system improvements involving sophisticated advances. Included in this category is everything from a new decision procedure (sometimes for a subarea too context-dependent to have an audience for a major published paper) to a data structure that speeds a key process. Deduction engineering is central to real-world progress in AD and deserves special notice.
- Mathematical proof systems, with limited priority given to systems tackling open problems of interest to the mathematics community. Support for the formalization of the core of mathematics will be important but only when organizational and technical issues are in place for a meaningful attack on the task.
- Formal verification methodology research, which contributes directly to AD core capability, and hence to other applications, as well as to its own growing list of applications. It is important to fund both basic program synthesis research and application projects.
- Processes that will help locate further applications. To promote promising applications, funds should be provided both as venture capital for plausible new projects and as continuation funding to allow further development of current prototype applications. Regarding more specific suggestions of key technical opportunities, the list of technical priorities given at the end of this article should be considered when funding in the AD area.

Since few researchers are entering the field in the U.S., funding agencies can help increase the efficiency of new and existing researchers by increasing funding for post-doctoral students, and allowing researchers to take sabbaticals at foreign AD centers.
Universities can help by encouraging the teaching of logic and formal methods to mathematicians and computer scientists so these professionals will understand the opportunities and methodologies associated with automated reasoning. Such opportunities will significantly increase over their working lifetimes. Faculty teaching logic and mathematics should work with practitioners to develop courses that use AD tools, including (semi-)automated theorem provers.

Research centers that house major AD systems should provide access for visiting researchers as well as remote users who desire intensive involvement with the AD system. They should provide short courses on the use of the AD system(s), both for AD researchers who may then use and augment the system and for possible application users. Centers with application systems should provide workshops on the application technology (within proprietary and national security limits). They should also develop working relations with AD research personnel for an in-house work period, and take this opportunity to educate AD researchers on specific technical needs.

Industry should support continued research in formal verification methods and support exploration of AD methods to enhance symbolic testing. It should also offer the opportunities suggested for research centers.

The recommendations for practitioners of AD are broken into two categories: general and technical—we begin with the general requirements:

• Make AD systems accessible to nonspecialists, including input/output in the formalism of the user and output using natural language to report inference connections.
• Develop a set of open problems in mathematics that have a reasonable axiomatic formulation and hence may be approached by AD methods. Maintain a Web site for the listing of such problems.
• Identify a Grand Challenge problem that uses AD technology. For maximum visibility the problem should be economically relevant, although an appropriate open mathematical problem could be of sufficient stature to serve as a Grand Challenge problem.
• Support the competition for AD systems at AD conferences. Enlarge the competition categories so all major classes of AD systems are included. The purpose is more to “improve the breed” than to encourage publicity outside the field.
• Offer short courses on major AD systems. Provide apprenticeships for interested professionals to learn the system operation. Provide network accessibility for remote use of AD systems.

The following list of technical recommendations is incomplete and should be augmented by the recommendations given in [4].

• Make all AD systems easier to use, including increased automation of interactive tasks.
• Expand work on proof presentation, including natural language augmentation.
• Find effective ways to use larger inference steps.
• Develop effective ways of combining interactive and automatic search procedures.
• Continue to integrate proof methods to form more composite systems.
• Incorporate computer algebra systems into AD systems. This effort must include addressing some problems with correctness of computer algebra systems.
• Continue priority attention to equational logic and the techniques, such as rewrite rules, that process the logic.
• Continue priority attention to mathematical induction.
• Accelerate work in proof planning, including proof abstraction and proof by analogy.
• Provide more integration of model generation with proof finding.
• Continue efforts to introduce “semantics” (problem-specific information not easily axiomatized) into AD proof guidance.

The following technical recommendations are assigned a lower priority, although they may become more important in the future.

• Pursue methods for automated proof discovery applicable to the continuous mathematics domain. This problem domain is important because it includes a good portion of mathematics, but the methodology needed must include techniques quite different from those used to date in discrete mathematics. Good work has been done in this area but progress tends to be slow. Interactive mathematical assistant systems should be supported as the best vehicles for progress in this area.
• Pursue AD systems for nonclassical logics. At present, temporal logic is important, and AD systems to better handle this class of logics would be welcome.
• Pursue parallel computation paradigms for deduction. Gains in parallelization will come eventually from outside the AD field, and work within the AD field on parallelization will be particularly effective then. The limited implementation of parallel versions of (portions of) leading AD systems may be appropriate now.
• Give more attention to artificial intelligence, neural net, and general stochastic methodologies for application to proof guidance. In particular, explore the use of learning from past experience. Although abstract pattern recognition and learning capabilities are vital to AD progress, there is currently little idea how to achieve advances here. Funding should be selective but aggressive when real advances become feasible.

References