Invited paper for FLoC'02. Appears in the Proceedings of FME'02, LNCS.
©Springer-Verlag

Little Engines of Proof[⋆]

Natarajan Shankar

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA
shankar@csl.sri.com
URL: http://www.csl.sri.com/˜shankar/
Phone: +1 (650) 859-5272 Fax: +1 (650) 859-2844

Abstract. The automated construction of mathematical proof is a basic activity in computing. Since the dawn of the field of automated reasoning, there have been two divergent schools of thought. One school, best represented by Alan Robinson's resolution method, is based on simple uniform proof search procedures guided by heuristics. The other school, pioneered by Hao Wang, argues for problem-specific combinations of decision and semi-decision procedures. While the former school has been dominant in the past, the latter approach has greater promise. In recent years, several high quality inference engines have been developed, including propositional satisfiability solvers, ground decision procedures for equality and arithmetic, quantifier elimination procedures for integers and reals, and abstraction methods for finitely approximating problems over infinite domains. We describe some of these "little engines of proof" and a few of the ways in which they can be combined. We focus in particular on combining different decision procedures for use in automated verification.

Its great triumph was to prove that the sum of two even numbers is even.

Martin Davis [Dav83] (on his Presburger arithmetic procedure)

The most interesting lesson from these results is perhaps that even in a fairly rich domain, the theorems actually proved are mostly ones which call on a very small portion of the available resources of the domain.

Hao Wang (quoted by Davis [Dav83])

[⋆] Funded by NSF Grants CCR-0082560 and CCR-9712383, DARPA/AFRL Contract F33615-00-C-3043, and NASA Contract NAS1-20334. John Rushby, Sam Owre, Ashish Tiwari, and Tomás Uribe commented on earlier drafts of this paper.

1 Introduction

At a very early point in its development, the field of automated reasoning took an arguably wrong turn. For nearly forty years now, the focus in automated reasoning research has been on *big iron*: general-purpose theorem provers based on uniform proof procedures augmented with heuristics. These efforts have not been entirely fruitless. As success stories, one might list an impressive assortment of open problems that have succumbed to semi-brute-force methods, and spin-off applications such as logic programming. However, there has been very little discernible progress on the problem of automated proof construction in any significant mathematical domain. Proofs in these domains tend to be delicate artifacts whose construction requires a collection of well-crafted instruments, little engines of proof, working in tandem. In other disciplines such as numerical analysis, computer algebra, and combinatorial algorithms, it is quite common to have libraries of useful routines. Such software libraries have not taken root in automated deduction because the scientific and engineering challenges involved are quite significant. We examine some of the successes in building and combining little deduction engines for building proofs and refutations (e.g., counterexamples), and survey some of the challenges that still lie ahead.

The tension between general-purpose proof search and special-purpose decision procedures has been with us from very early on. Automated reasoning had its beginnings in the pioneering Logic Theorist system of Newell, Shaw, and Simon [NSS57]. The theorems they proved were shown by Hao Wang [Wan60b] to fall within simply decidable fragments like propositional logic and the $\forall^*\exists^*$ Bernays-Schönfinkel fragment of first-order logic [BGG97]. Many technical ideas from the Logic Theorist such as subgoaling, substitution, replacement, and forward and backward chaining, have been central to automated reasoning, but the dogma that human-oriented heuristics are the key to effective theorem proving has not been vindicated. Hao Wang [Wan60a] proposed an entirely different approach that he called *inferential analysis* as a parallel to numerical analysis. Central to his approach was the use of domain-specific decision and semi-decision procedures, so that proofs could be constructed by means of reductions to some combination of problems that could each be easily solved. Due to the prevailing bias in artificial intelligence, Wang lost the debate at that point in time, but, as we argue here, his ideas still make plenty of sense. As remarked by Martin Davis [Dav83]:

> *The controversy referred to may be succinctly characterized as being between the two slogans: "Simulate people" and "Use mathematical logic". . . . Thus as early as 1961 Minsky [Min63] remarked*
>
> > . . . it seems clear that a program to solve real mathematical problems will have to combine the mathematical sophistication of Wang with the heuristic sophistication of Newell, Shaw, and Simon.

The debate between human-oriented and logic-oriented approaches is beside the point. The more significant debate in automated reasoning is between two approaches that in analogy with economics can be labelled as *macrological* and *micrological*. The macrological approach takes a language and logic such as first-order logic as given, and attempts to find a uniform (i.e., problem-independent) method for constructing proofs of conjectures stated in the logic. The micrological approach attacks a class of problems and attempts to find the most effective way of validating or refuting conjectures in this problem class. In his writings, Hao Wang was actually espousing a micrological viewpoint. He wrote [Wan60a]

> *In contrast with pure logic, the chief emphasis of inferential analysis is on the efficiency of algorithms, which is usually obtained by paying a great deal of attention to the detailed structure of problems and their solutions, to take advantage of possible systematic short cuts.*

Automated reasoning got off to a running start in the 1950s. Already in 1954, Davis [Dav57] had implemented a decision procedure for Presburger arithmetic [Pre29]. Davis and Putnam [DP60], during 1958–60, devised a decision procedure for CNF satisfiability (SAT) based on inference rules for propagation of unit clauses, ground resolution, deletion of clauses with pure literals, and splitting. The ground resolution rule turned out to be space-inefficient and was discarded in the work of Davis, Logemann, and Loveland [DLL62]. Variants of the latter procedure are still employed in modern SAT solvers. Gilmore [Gil60] and Prawitz [Pra60] examined techniques for first-order validity based on Herbrand's theorem. Many of the techniques from the 1950s still look positively modern.

Robinson's introduction [Rob65] of the resolution principle (during 1963–65) based on unification brought about a qualitative shift in automated theorem proving. From that point on, the field of automated reasoning never looked forward. Resolution provides a simple inference rule for refutational proofs for first-order statements in skolemized, prenex form. It spawned a multitude of strategies, heuristics, and extensions. Nearly forty years later, resolution [BG01] remains extremely popular as a general-purpose proof search method primarily because the basic method can be implemented and extended with surprising ease. Resolution-based methods have had some success in proving open problems in certain domains where general-purpose search can be productive. The impact of resolution on theorem proving in mathematically rich domains has not been all that encouraging.

The popularity of uniform proof methods like resolution stems from the simple dogma that since first-order logic is a generic language for expressing statements, generic first-order proof search methods must also be adequate for finding proofs. This central dogma seems absurd on the face of it. Stating a problem and solving it are two quite separate matters. But the appeal of the dogma is obvious. A simple, generic method for proving theorems basically hits the jackpot by fulfilling Leibniz's dream of a reasoning machine. A more sophisticated version of the dogma is that a uniform proof method can serve as the basic structure for introducing domain-specific automation. There is little empirical evidence that even this dogma has any validity.

On the other hand, certain domain-specific automated theorem provers have been quite effective. The Boyer-Moore line of theorem provers [BM79,KMM00] has had significant success in the area of inductive proofs of recursively defined functions. Various geometry theorem provers [CG01] based on both algebraic and non-algebraic, machine-oriented and human-oriented methods, have been able to automatically prove theorems that would tax human ingenuity. Both of these classes of theorem provers owe their success to domain-specific automation rather than general-purpose theorem proving.

Main Thesis. Automated reasoning has for too long been identified with uniform proof search procedures in first-order logic. This approach shows very little promise. The basic seduction of uniform theorem proving techniques is that phenomenal gains could be achieved with very modest implementation effort. Hao Wang [Wan60b,Wan60a,Wan63] in his early papers on automated reasoning sketched the vision of a field of inferential analysis that would take a deeper look at the problem of automating mathematical reasoning while exploiting domain-specific decision procedures. He wrote [Wan63]

> *That proof procedures for elementary logic can be mechanized is familiar. In practice, however, were we slavishly to follow these procedures without further refinements, we should encounter a prohibitively expansive element. . . . In this way we are led to a closer study of reduction procedures and of decision procedures for special domains, as well as of proof procedures of more complex sorts.*

Woody Bledsoe [Ble77] made a similar point in arguing for semantic theorem proving techniques as opposed to resolution.

Decision procedures [Rab78], and more generally inference procedures, are crucial to the approach advocated here. Few problems are stated in a form that is readily decidable, but proof search strategies, heuristics, and human guidance can be used to decompose these problems into decidable subproblems. Thus, even though not many interesting problems are directly expressible in Presburger arithmetic, a great many of the naturally arising proof obligations and subproblems do fall into this decidable class.

Building a library of automated reasoning routines along the lines of numerical analysis and computer algebra, is not as easy as it looks. A theorem prover has a simple interface in that it is given a conjecture and it returns a proof or a disproof. The lower-level procedures often lack clear interface specifications of this sort. Even if they did, building a theorem prover out of modular components may not be as efficient as a more monolithic system. Boyer and Moore [BM86] indicate how even a simple decision procedure can have a complex interaction with the other components, so that it is not merely a black box that returns *proved* or *disproved*. The construction of modular inference procedures is a challenging research issues in automated reasoning.

Work on little engines of proof has been gathering steam lately Many groups are actively engaged in the construction of little proof engines, while others are putting in place the train tracks on which these engines can run. PVS [ORSvH95] itself can be seen as an attempt to unify many different inference procedures: typechecking, ground decision procedures, simplification, rewriting, MONA [EKM98], model checking [CGP99], abstraction, and static analysis, within a single system with an expressive language for writing mathematics.

2 Propositional Logic

The very first significant metamathematical results were those on the soundness, completeness, and decidability of propositional logic [Pos21]. Since boolean logic has applications in digital circuit design, a lot of attention has been paid to the problem of propositional satisfiability. A propositional formula ϕ is built from propositional atoms p_i by means of negation $\neg\phi$, disjunction $\phi_1 \vee \phi_2$, and conjunction $\phi_1 \wedge \phi_2$. Further propositional connectives can be defined in terms of basic ones like $\neg$ and $\vee$. A propositional formula can be placed in *negation normal form*, where all the negations are applied only to propositional atoms. A literal l is an atom p or its negation $\neg p$. A clause C is a disjunction of literals. By labelling subformulas with atoms and using distributivity, any propositional formula can be efficiently transformed into one that is in conjunctive normal form (CNF) as a conjunction of clauses. A CNF formula can be viewed as a bag Γ of clauses. The Davis–Putnam method (DP) [DP60] consisted of the following rules:

1. Unit propagation: l, Γ is satisfiable if $\Gamma[l \mapsto \top, \neg l \mapsto \bot]$ is satisfiable.
2. Pure literal: Γ is satisfiable if $\Gamma - \Delta$ is satisfiable, for $\neg l \notin [\![\Gamma]\!]$, where $[\![\Gamma]\!]$ is the set of subformulas of Γ, and $l \in C$ for each $C \in \Delta$.
3. Splitting: Γ is satisfiable if either l, Γ or $\neg l, \Gamma$ is satisfiable.
4. Ground resolution: $l \vee C_1, \neg l \vee C_2, \Gamma$ is satisfiable if $C_1 \vee C_2, \Gamma$ is satisfiable.

The Davis–Logemann–Loveland (DLL) variant [DLL62] drops the ground resolution rule since it turned out to be space-inefficient. Several modern SAT solvers such as SATO [Zha97], GRASP [MSS99], and Chaff [MMZ$^+$01], are based on the DLL method. They are capable of solving satisfiability problems with hundreds of thousands of propositional variables and clauses. With this kind of performance, many significant applications become feasible including invariant-checking for systems of bounded size, bounded model checking, i.e., the search for counterexamples of length k for a temporal property, and boolean equivalence checking where two circuits are checked to have the same input/output behavior.

Stålmarck's method [SS00] does not employ a CNF representation. Truth values are propagated from formulas to subformulas through a method known as saturation. There is a splitting rule similar to that of DP, but it can be applied to subformulas and not just propositions. The key component of Stålmarck's method is the dilemma rule which considers the intersection of the two subformula truth assignments derived from splitting. Further splitting is carried out with respect to this intersection.

Binary Decision Diagrams. Reduced Ordered Binary Decision Diagrams (ROBDDs) [Bry86] are a canonical representation for boolean functions, i.e., functions from $[B^n \rightarrow B]$. BDDs are binary branching directed acyclic graphs where the nodes are variables and the outgoing branches correspond to the assignment of $\top$ and $\bot$ to the variable. There is a total ordering of variables that is maintained along any path in the graph. The graph is kept in reduced form so that if there is a node such that both of its branches lead to the same subgraph, then the node is eliminated.

Standard operations like negation, conjunction, disjunction, composition, and boolean quantification, have efficient implementations using BDDs. The BDD data structure has primarily been used for boolean equivalence checking and symbolic model checking. The main advantage of BDDs over other representations is that checking equivalence is easy. Boolean quantification is also handled more readily using BDDs. BDDs can also be used for SAT solving since it is in fact a compact representation for all solutions of a boolean formula. But the strength of BDDs is in representing boolean functions of a low communication complexity, i.e., where it is possible to partition the variables so that there are few dependencies between variables across the partition. BDDs have been popular for symbolic model checking [CGP99] and boolean equivalence checking.

Quantified Boolean Formulas and Transition Systems. In a propositional logic formula, all variables are implicitly universally quantified. One obvious extension is the introduction of Boolean existential and universal quantification. The resulting fragment is called quantified boolean formulas (QBF). This kind of quantification can be expressed purely in propositional logic. For example, the formula $(\exists p : Q)$ is equivalent to $(Q[p \mapsto \top] \vee Q[p \mapsto \bot])$. The language of QBF is of course exponentially more succinct than propositional logic. The decision procedure for QBF validity is a PSPACE-complete problem. Many interesting problems that can be cast as interactive games can be mapped to QBF.

Finite-state transition systems can be defined in QBF. A finite state type consists of a finite number of distinct variables over types such as booleans, scalars, subranges, and finite arrays over a finite element type. A finite state type can be encoded in binary form. A transition system over a finite state type that is represented by n boolean variables then consists of an initialization predicate I that is an n-ary boolean function, and a transition relation N that is a $2n$-ary boolean function. The nondeterministic choice between two transition relations N_1 and N_2 is easily expressed as $N_1 \vee N_2$. Internal state can be hidden through boolean quantification. The composition $(N_1; N_2)$ of two transition relations N_1 and N_2 can be captured as $\exists \overline{y} : N_1(\overline{x}, \overline{y}) \wedge N_2(\overline{y}, \overline{x}')$.

Fixpoints and Model Checking. QBF can be further extended through the addition of fixpoint operators that can capture the transitive closure of a transition relation. Given a transition relation N, the reflexive-transitive closure of N can be written as $\mu Q : \overline{x}' = \overline{x} \vee (\exists \overline{y} : N(\overline{x}, \overline{y}) \wedge Q(\overline{y}, \overline{x}'))$. Similarly, the set of states reachable from the initial set of state can be represented as $\mu Q : I(\overline{x}) \vee (\exists \overline{y} : Q(\overline{y}) \wedge N(\overline{y}, \overline{x}))$. The boolean function represented by a fixpoint formula can be computed by unwinding the fixpoint until convergence is reached. For this, the ROBDD representation of the boolean function is especially convenient since it makes it easy to detect convergence through an equivalence test, and to represent boolean quantification [BCM$^+$92,McM93]. The boolean fixpoint calculus can easily represent the temporal operators of the branching-time temporal logic CTL where one can for example assert that a property always (or eventually) holds on all (or some) computation paths leading out of a state. The boolean fixpoint calculus can also represent different fairness constraints on paths. The emptiness problem for Büchi automaton over infinite words can be expressed using fairness constraints. This in turn captures the model checking problem for linear-time temporal logics [VW86,Kur93].

Weak monadic second-order logic of a single successor (WS1S). WS1S has a successor operation for constructing natural numbers, first-order quantification over natural numbers, and second-order quantification over finite sets of natural numbers. WS1S is a natural formalism for many applications, particularly for parametric systems. The logic can be used to capture interesting datatypes such as regular expressions, lists, queues, and arrays. There is a direct mapping between the logic and finite automata. A finite set X of natural numbers can be represented as a bit-string where a 1 in the i'th position indicates that i is a member of X. A formula with free set variables $X_1, \ldots, X_n$ is then a set of strings over B^n. The logical operations have automata theoretic counterparts so that negation is complementation, conjunction is the product of automata, and existential quantification is projection. The MONA library [EKM98] uses an ROBDD representation for the automaton corresponding to the formula.

3 Equality and Inequality

Equality introduces some of the most significant challenges in automated reasoning [HO80]. Many subareas of theorem proving are devoted to equality including rewriting, constraint solving, and unification. In this section we focus on ground decision procedures for equality. Many theorem proving systems are based around decision procedures for equality. The language now includes terms which are built from variables x, and applications $f(a_1, \ldots, a_n)$ of an n-ary function symbol f to n terms $a_1, \ldots, a_n$. The *ground* fragment can be seen as an extension of propositional logic where the propositional atoms are of the form

$a = b$, for terms a and b. The literals are now either equations $a = b$ or disequations $a \neq b$. The variables in a formula are taken to be universally quantified. The validity of a formula ϕ that is a propositional combination of equalities can be decided by first transforming $\neg\phi$ into disjunctive normal form $D_1 \vee \ldots \vee D_n$, and checking that each disjunct D_i, which is a conjunction of literals, is refutable. The refutation of a conjunction D_i of literals can be carried out by partitioning the terms in D_i into equivalence classes of terms with respect to the equalities in D_i. If for some disequation $a \neq b$ in D_i, a and b appear in the same equivalence class, then we have a contradiction and D_i has been refuted. The original claim ϕ is verified if each such disjunct D_i has been refuted.

If the function symbols are all uninterpreted, then congruence closure can be used to construct the equivalence classes corresponding to the conjunction of literals D_i. Let the set of subterms of D_i be $[\![D_i]\!]$. The initial partition P_0 is the set $\{\{c\} \mid c \in [\![D_i]\!]\}$. When an equality of the form $a = b$ from D_i is processed, it results in the merging of the equivalence classes corresponding to a and b. As a result of this merge, other equivalence classes might become mergeable. For example, one equivalence might contain $f(a_1, \ldots, a_n)$ while the other contains $f(b_1, \ldots, b_n)$, and each a_j is in the same equivalence class as the corresponding b_j. The merging of equivalence classes is performed until no further mergeable pairs of equivalence classes remain, and the partition P_1 is constructed. The equalities in D_i are successively processed and the resulting partition is returned as P_m. If for some disequality $a \neq b$, a and b are in the same equivalence class in P_m, then a contradiction is returned. Otherwise, the conjunction D_i is satisfiable.

Linear arithmetic. A large fraction of the subgoals that arise in verification condition generation, typechecking, array-bounds checking, and constraint solving involve linear arithmetic constraints [BW01]. Linear arithmetic equalities in n variables have the form $c_0 + c_1 * x_1 + \ldots + c_n * x_n = 0$, where the coefficients c_i range over the rationals, and the variables x_i range over the rationals or reals. It is easy to isolate a single variable, say x_1, as $x_1 = -c_0/c_1 - (c_2/c_1) * x_2 - \ldots - (c_n/c_1) * x_n$. This solved form for x_1 can then be substituted into the remaining linear equations thus eliminating the variable x_1. Gaussian elimination is based on the same idea where the set of linear equations is represented by $A * X = B$, and the matrix representation of the linear equations is transformed into row echelon form in order to solve for the variables.

Linear inequalities are of the form $c_0 + c_1 * x_1 + \ldots + c_n * x_n \# 0$, where # is either $<$, $\leq$, $>$, or $\geq$. Note that linear inequalities, unlike equalities, are closed under negation. Any linear equality can also be easily transformed into a pair of inequalities. As with linear equalities, linear inequalities can also be transformed into a form where a single variable is isolated. A pair of inequalities, $x \leq a$ and $x \geq b$ can be resolved to obtain $b \leq a$ thus eliminating x. This kind of Fourier-Motzkin elimination [DE73] can be used as a quantifier elimination procedure to decide the first-order theory of linear arithmetic by repeatedly reducing any quantified formula of the form $\exists x : P(x)$ where $P(x)$ is a conjunction of inequalities, into the form P', where x has been eliminated. By eliminating quantifiers in an inside-out order while transforming universal quantification $\forall x : A$ into $\neg\exists x : \neg A$, we arrive at an equivalent variable-free formula that directly evaluates to true or false. Linear programming techniques like Simplex [Nel81] can also be used for solving linear arithmetic inequality constraints. Separation predicates are linear inequalities of the form $x - y \leq c$ or $x - y < c$ for some constant c, and these can be decided with graph-theoretic techniques [Sho81]. This simple class of linear inequalities is useful in model checking timed automata [ACD93].

Presburger arithmetic [Pre29] is the first-order theory of linear arithmetic over the integers. Solving constraints over the integers is harder than over the rationals and reals. Cooper [Coo72,Opp78] gives an efficient quantifier elimination algorithm for Presburger arithmetic. Once again, we need only consider quantifiers of the form $\exists x : P(x)$ where $P(x)$ is a conjunction of inequalities. We add divisibility assertions of the form $k|a$, where k is a positive integer. An inequality of the form $c_0 + c_1 * x_1 + \ldots + c_n * x_n \geq 0$ can be transformed to $c_1 * x_1 \geq -c_0 - c_2 * x_n - \ldots - c_n * x_n$, and similarly for other inequality relations. Since we are dealing with integers, a nonstrict inequality like $a \leq b$ can be transformed to $a < b + 1$. Having isolated all occurrences of x_1, we can compute the least common multiple α_1 of the coefficients corresponding to each occurrence of x_i. Now $P(x_1)$ is of the form $P'(\alpha_1 * x_1)$, and $\exists x_1 : P(x_1)$ can be replaced by $\exists x_1 : P'(x_1) \wedge \alpha_1 | x_1$. Here, $P'(x)$ is a conjunction of formulas of the forms: $x < a$, $x > b$, $k|x+d$, and $j \not| x+e$. Let $A = \{a \mid x < a \in P'(x)\}$, $B = \{b \mid x > b \in P'(x)\}$, $K = \{k \mid (k|x+d) \in P'(x)$, and $J = \{j \mid (j \not| x+e) \in P'(x)\}$. Let G be the least common multiple of $K \cup J$. If A is nonempty, then $\exists x : P'(x)$ can be transformed to $\bigvee_{a \in A} \exists x : a - G \leq x < a \wedge P'(x)$. The bounded existential quantification in the latter formula can easily be eliminated. Essentially, if m satisfies the constraints in $K \cup J$, then so does $m + r * G$ for any integer r. Hence, if $P'(m)$ holds for some m and A is nonempty, then there is an m in the interval $[a - G, a)$ for some $a \in A$ such that $P'(m)$ holds. Similarly, if B is nonempty, $\exists x : P'(x)$ can also be transformed to $\bigvee_{b \in B} \exists x : b < x \leq b + G \wedge P'(x)$. If both A and B are empty, then $\exists x : P'(x)$ is transformed to $\exists x : 0 < x \leq G \wedge P'(x)$. For example, the claim that x is an even integer can be expressed as $\exists u : 2 * u = x$ if we avoid the divisibility predicate. The quantifier elimination transformation above would convert this to $u' > x - 1 \wedge u' < x + 1 \wedge (2|u')$ which eventually yields $(x > x - 1 \wedge x < x + 1 \wedge 2|x) \vee (x + 1 > x - 1 \wedge x + 1 < x + 1 \wedge (2|x + 1))$. The latter formula easily simplifies to $(2|x)$. The claim that the sum of two even numbers is even then has the form $(\forall x : \forall y : 2|x \wedge 2|y \supset 2|(x + y))$. Converting universal quantification to existential quantification yields $\neg\exists x : \exists y : 2|x \wedge 2|y \wedge 2 \not| (x + y)$. Quantifier elimination yields $\neg\exists x : 0 < x \leq 2 \wedge \exists y : 0 < y \leq 2 \wedge (2|x) \wedge (2|y) \wedge (2 \not| x + y)$, which is clearly valid. The decidability of Presburger arithmetic can also be reduced to that of WS1S, and even though the latter theory has nonelementary complexity, this reduction using MONA works quite efficiently in practice [SKR98].

By the unsolvability of Hilbert's tenth problem, even the quantifier-free fragment of nonlinear arithmetic over the integers or rationals is undecidable. However, the first-order theory of nonlinear arithmetic over the reals and the complex numbers is decidable. Tarski [Tar48] gave a decision procedure for this theory. Collins [Col75] gave an improved quantifier elimination procedure that is the basis for a popular package called QEPCAD [CH91]. These procedures have been successfully used in proving theorems in algebraic geometry. Buchberger's Gröbner basis method for testing membership in polynomial ideals has also been successful in computer algebra and geometry theorem proving [CG01,BW01].

Constraint solving and quantifier elimination methods in linear and nonlinear arithmetic over integers, reals, and rationals, are central to a large number of applications of theorem proving that involve numeric constraints.

4 The Combination Problem

The application of decision procedures for individual theories is constrained by the fact that few natural problems fall exactly within a single theory. Many of the proof obligations that arise out of extended typechecking or verification condition generation involve arithmetic equalities and inequalities, tuples, arrays, datatypes, and uninterpreted function symbols. There are two basic techniques for constructing decision procedures for checking the satisfiability of conjunctions of literals in combinations of disjoint theories: the Nelson–Oppen method [NO79,TH96] and the Shostak method [Sho84].

Nelson and Oppen's Method. The Nelson–Oppen method combines decision procedures for disjoint theories by using variable abstraction to purify a formula containing operations from a union of theories, so that the formula can then be partitioned into subgoals that can be handled by the individual decision procedures. Let B represent the formula whose satisfiability is being checked in the union of disjoint theories θ_1 and θ_2. First variable abstraction is used to convert B into $B' \wedge V$, where V contains equalities of the form $x = t$, where x is a fresh variable and t contains function symbols exclusively from θ_1 or from θ_2, and B' contains x renaming t. In particular, if $V[B']$ is the result of replacing each occurrence of x in B' by the corresponding t for each $x = t$ in V, then B must the result of repeatedly applying V to B' and eliminating all the newly introduced variables. Next, $V \wedge B'$ can be partitioned as $B_1 \wedge B_2$, where each B_i only contains function symbols from the theory θ_i. Let X be the free variables that are shared between B_1 and B_2. Guess a partition $X_1, \ldots, X_m$ on the variables in X. Let E be an arrangement corresponding to this partition so that E contains $x = y$ for each pair of distinct variables x, y in some X_j, and $u \neq v$ for each pair of variables u, v, such that $u \in X_j, v \in X_k$ for $j \neq k$. Check if $E \wedge B_1$ is satisfiable in θ_1 and $E \wedge B_2$ is satisfiable in θ_2. If that is the case, then B is satisfiable in $\theta_1 \cup \theta_2$, provided θ_1 and θ_2 are *stably infinite.* A theory θ is stably infinite if whenever a formula is θ-satisfiable (satisfiable in a θ-model), it is θ-satisfiable in an infinite model.

Shostak's Method. The Nelson–Oppen combination is a way of combining black box decision procedures. Shostak's method is an optimization of the Nelson–

Oppen combination for a restricted class of equational theories. A theory θ is said to be canonizable if there is a canonizer σ such that the equality $a = b$ is valid in θ iff $\sigma(a) \equiv \sigma(b)$. A theory θ is said to be solvable if there is an operation *solve* such that $\mathit{solve}(a = b)$ returns a set S of equalities $\{x_1 = t_1, \ldots, x_n = t_n\}$ equivalent in some sense to $a = b$, where each x_i occurs in $a = b$ but not in t_j for $1 \leq i, j \leq n$. A Shostak theory is one that is canonizable and solvable. Shostak's combination method can be used to combine one or more Shostak theories with the theory of equality over uninterpreted terms. The method essentially maintains a set S of solutions $S_0, \ldots, S_N$, where each set S_i contains equalities of the form $x = t$ for some term t in θ_i. The theory θ_0 is used for the uninterpreted function symbols. Two variables x and y are said to be merged in S_i if $x = t$ and $y = t$ are both in S_i. It is possible to define a global canonical form $S[\![a]\!]$ for a term a with respect to the solution state S using the individual canonizers σ_i.

Shostak's original algorithm [Sho84] and its proof were both incorrect. The algorithm, as corrected by the author and Harald Ruess [RS01,SR02], checks the validity of a sequent $T \vdash c = d$. It does this by processing each equality $a = b$ into its solved form. If S is the current solution state, then an unprocessed equality $a = b$ in T is processed by first transforming it to $a' = b'$, where $a' = S[\![a]\!]$ and $b' = S[\![b]\!]$. The equality $a' = b'$ is variable abstracted and the variable abstraction equalities $x = t$ are added to the solution S_i, where t is a term in the theory θ_i. The algorithm then repeatedly reconciles the solutions S_i so that whenever two variables x and y are merged in S_i but not in S_j, for $i \neq j$, then they are merged in S_j by solving $t_x = t_y$ in θ_j, for $x = t_x$ and $y = t_y$ in S_j, and composing the solution with S_j to obtain a new solution set S_j. When all the input equalities from T have been processed and we have the resulting solution state S, we check if $S[\![c]\!] = S[\![d]\!]$. A conjunction of literals $\bigwedge_{i=1}^{m} a_i = b_i \wedge \bigwedge_{j=1}^{n} c_j \neq d_j$ is satisfiable iff $S \neq \bot$ and $S[\![c_j]\!] \not\equiv S[\![d_j]\!]$, for each j, $1 \leq j \leq n$, where $S = \mathit{process}(\{a_1 = b_1, \ldots, a_m = b_m\})$.

Ground Satisfiability. The Nelson–Oppen and Shostak decision procedures check the satisfiability of conjunctions of literals drawn from a combination of theories. These procedures can be extended to handle propositional combinations of atomic formulas by transforming these formulas to disjunctive normal form. This method can be inefficient when the propositional case analysis involved is heavy. It is usually more efficient to combine a SAT solver with a ground decision procedure [BDS02,dMRS02]. There are various ways in which such a combination can be executed. Let ϕ be the formula whose satisfiability is being checked. Let L be an injective map from fresh propositional variables to the atomic subformulas of ϕ such that $L^{-1}[\phi]$ is a propositional formula. We can use a SAT solver to check that $L^{-1}[\phi]$ is satisfiable, but the resulting truth assignment, say $l_1 \wedge \ldots \wedge l_n$, might be spurious, that is $L[l_1 \wedge \ldots \wedge l_n]$ might not be ground-satisfiable. If that is the case, we can repeat the search with the added lemma $(\neg l_1 \vee \ldots \vee \neg l_n)$ and invoke the SAT solver on $(\neg l_1 \vee \ldots \vee \neg l_n) \wedge L^{-1}[\phi]$. This ensures that the next satisfying assignment returned is different from the previous assignment that was found to be ground-unsatisfiable. The lemma that is added can be minimized to find the minimal unsatisfiable set of literals l_i.

This means that the lemma that is added is smaller, and the pruning of spurious assignments is more effective. The ground decision procedure can be also be used to precompute a set Λ of lemmas (clauses) of the form $l_1 \vee \ldots \vee l_n$, where $\neg L[l_1] \wedge \ldots \neg L[l_n]$ is unsatisfiable according to the ground decision procedures. The SAT solver can then be reinvoked with $\Lambda \wedge L^{-1}[\phi]$.

A tighter integration of SAT solvers and ground decision procedures would allow the decision procedures to check the consistency of the case analysis during an application of splitting in the SAT solver and avoid cases that are ground-unsatisfiable. Through a tighter integration, it would also be possible to resume the SAT solver with the added conflict information without starting the SAT solving process from scratch. We address the challenge of integrating inference procedures below.

Applications. Ground decision procedures, ground satisfiability, and quantifier elimination have many applications.

Symbolic Execution: Given a transition system, symbolic execution is the process of computing preconditions or postconditions of the transition system with respect to an assertion. For example, the strongest postcondition of an assertion p with respect to a transition N is the assertion $\lambda s : \exists s_0 : p(s_0) \wedge N(s_0, s)$. For certain choices of p and N, this assertion can be computed by means of a quantifier elimination. This is useful in analyzing timed and hybrid systems [ACH+95].

Infinite-State Bounded Model Checking: Bounded model checking checks for the existence of counterexamples of length upto a bound k for a given temporal property. With respect to certain temporal properties, it is possible to reduce the bounded model checking problem for such systems to a ground satisfiability problem [dMRS02].

Abstraction and Model Checking: The early work on abstraction in the context of model checking was on reducing finite-state systems to smaller finite-state systems, i.e., systems with fewer possible states [Kur93,CGL92,LGS+95]. Graf and Saïdi [GS97] were the first to consider the use of a theorem prover for reducing (possibly) infinite-state systems to finite-state (hence, model-checkable) form. Their technique of *predicate abstraction* constructs an abstract counterpart of a concrete transition system where the truth values of certain predicates over the concrete state space are simulated by boolean variables. Data abstraction replaces a variable over an infinite state space by one over a finite domain. Predicate and data abstraction based on theorem proving are widely used [BLO98b,CU98,DDP99,SS99,BBLS00,CDH+00,TK02,HJMS02,FQ02]. The finite-state abstraction can exhibit spurious counterexamples that are not reproducible on the concrete system. Ground decision procedures are also useful here for detecting spurious counterexamples and suggesting refinements to the abstraction predicates [BLO98a,SS99,DD01].

Software Engineering: Ground decision procedures are central to a number of analysis tools for better engineered software including array-bounds checking, extended static checking [DLNS98], typechecking [SO99], and static analysis [BMMR01,Pug92].

5 Challenges

We have enumerated some of the progress in developing, integrating, and deploying various inference procedures. A great many challenges remain. We discuss a few of these below.

The Complexity Challenge. Many decision procedures are of exponential, super-exponential, or non-elementary complexity. However, this complexity often does not manifest itself on practical examples. Modern SAT solvers can solve very large practical problems, but they can also run aground on small instances of simple challenges like the propositional pigeonhole principle. MONA deals with a logic that is known to have a non-elementary lower bound, yet it performs quite well in practice. The challenge here is to understand the ways in which one can overcome complexity bounds on the problems that arise in practice through heuristic or algorithmic means.

The Theory Challenge. Inference procedures are hard to build, extend, and maintain. The past experience has been that good theory leads to simpler decision procedures with greater efficiency. A well-developed theory can also help devise uniform design patterns for entire classes of decision procedures. Such design patterns can contribute to both the efficiency and modularity of these procedures. Methods derived by specializing general-purpose methods like resolution and rewriting can also simplify the construction of decision procedures.

The Modularity Challenge. As we have already noted, inference procedures need rich programmer interfaces (APIs) [BM86,FORS01]. Boyer and Moore [BM86] write:

> ...*the black box nature of the decision procedure is frequently destroyed by the need to integrate it. The integration forces into the theorem prover much knowledge of the inner workings of the procedure and forces into the procedure many features that are unnecessary when the problem is considered in isolation.*

For example, a ground decision procedure can be used in an online manner so that atomic formulas are added to a context incrementally, and claims are tested against the context. The API should include operations for asserting and retracting information, testing claims, and for creating, deleting, and browsing contexts. The decision procedures might need to exchange information with other inference procedures such as a rewriter, typechecker, or an external constraint solver. We already saw how the desired interaction between ground decision procedures and SAT solvers was such that neither of these could be treated as a black box procedure.

The modularity challenge is a significant one. Butler Lampson has argued that software components have always failed at low levels of granularity (see `http://research.microsoft.com/users/blampson/Slides/ReusableComponentsAbstract.htm`). He says that successful software components are those at the level of a database, a compiler, or a theorem prover, but not decision procedures, constraint solvers, or unification procedures. For interoperation between inference components, we also need compatible logics, languages, and term and proof representations.

The Integration Challenge. The availability of good inference components is a prerequisite for integration, but we also need to find effective ways of combining these components in complementary ways. The combination of decision procedures with model checking in predicate and data abstraction is a case where such a complementary integration is remarkably effective. Other such examples include the combination of unification/matching procedures and constraint solving, and typechecking and ground decision procedures.

The Verification Challenge. How do we know that our inference procedures are sound? This question is often asked by those who wish to apply inference procedures in contexts where a high level of manifest assurance is required. This question has been addressed in a number of ways. The LCF approach [GMW79] requires inference procedures to be constructed as tactics that generate a fully expanded proof in terms of low level inferences when applied. Proof objects have also been widely used as a way of validating inference procedures and securing mobile code [Nec97]. Reflection [Wey80,BM81] is a way of reasoning about the metatheory of a theory within the theory itself. The difficult tradeoff with reflection is that the theory has to be simple in order to be reasoned about, but rich enough to reason with. The verification of decision procedures is actually well within the realm of feasible, and recently, there have been several successful attempts in this direction [Thé98,FS02].

6 Conclusions

We have argued for a reappraisal of Hao Wang's programme [Wan60b,Wan60a] of inferential analysis as a paradigm for automated reasoning. The key element of this paradigm is the use of problem-driven combinations of sophisticated and efficient low-level decision procedures. Such an approach runs counter to the traditional thinking in automated reasoning which is centered around uniform proof search procedures. Similar ideas are also central to the automated reasoning schools of Bledsoe [Ble77] and Boyer and Moore [BM79,BM86].

The active use of decision procedures in automated reasoning began with the *west-coast* theorem proving approach pioneered by Boyer and Moore [BM79], Shostak [SSMS82], and Nelson and Oppen [LGvH$^+$79,NO79]. The PVS system is in this tradition [ORS92,Sha01], as are STeP [MT96], SIMPLIFY [DLNS98], and SVC [BDS00].

In recent years there has been a flurry of interest in the development of verification tools that rely quite heavily on sophisticated decision procedures. The quality and efficiency of many of these decision procedures is impressive. The underlying theory is also advancing rapidly [Bjø99,Tiw00]. Such theoretical advances will make it easier to construct correct decision procedures and integrate them more easily with other inference mechanisms. Contrary to the impression that decision procedures are black boxes, they need rich interfaces [BM86,FORS01,GNTV02] in order to be deployed most efficiently. The theory, construction, integration, verification, and deployment of inference procedures is likely to be a fertile source of challenges for automated reasoning in mathematically rich domains.

References

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

[ACH$^+$95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 6 February 1995.

[BBLS00] Kai Baukus, Saddek Bensalem, Yassine Lakhnech, and Karsten Stahl. Abstracting WS1S systems to verify parameterized networks. In Susanne Graf and Michael Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, number 1785 in Lecture Notes in Computer Science, pages 188–203, Berlin, Germany, March 2000. Springer-Verlag.

[BCM$^+$92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[BDS00] Clark W. Barrett, David L. Dill, and Aaron Stump. A framework for cooperating decision procedures. In David McAllester, editor, *Automated Deduction—CADE-17*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 79–98, Pittsburgh, PA, June 2000. Springer-Verlag.

[BDS02] Clark W. Barrett, David L. Dill, and Aaron Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In *Computer-Aided Verification, CAV '02*, Lecture Notes in Computer Science. Springer-Verlag, July 2002.

[BG01] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In Robinson and Voronkov [RV01], pages 19–99.

[BGG97] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer, 1997.

[Bjø99] Nikolaj Bjørner. *Integrating Decision Procedures for Temporal Verification.* PhD thesis, Stanford University, 1999.

[Ble77] W. W. Bledsoe. Non-resolution theorem proving. *Artificial Intelligence*, 9:1–36, 1977.

[BLO98a] Saddek Bensalem, Yassine Lakhnech, and Sam Owre. Computing abstractions of infinite state systems compositionally and automatically. In Hu and Vardi [HV98], pages 319–331.

[BLO98b] Saddek Bensalem, Yassine Lakhnech, and Sam Owre. InVeSt: A tool for the verification of invariants. In Hu and Vardi [HV98], pages 505–510.

[BM79] R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, New York, NY, 1979.

[BM81] R. S. Boyer and J S. Moore. Metafunctions: Proving them correct and using them efficiently as new proof procedures. In R. S. Boyer and J S. Moore, editors, *The Correctness Problem in Computer Science*. Academic Press, London, 1981.

[BM86] R. S. Boyer and J S. Moore. Integrating decision procedures into heuristic theorem provers: A case study with linear arithmetic. In *Machine Intelligence*, volume 11. Oxford University Press, 1986.

[BMMR01] T. Ball, R. Majumdar, T. Millstein, and S. Rajamani. Automatic predicate abstraction of C programs. In *Proceedings of the SIGPLAN '01 Conference on Programming Language Design and Implementation, 2001*, pages 203–313. ACM Press, 2001.

[Bry86] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[BW01] Alexander Bockmayr and Volker Weispfenning. Solving numerical constraints. In Robinson and Voronkov [RV01], pages 751–742.

[CDH$^+$00] James Corbett, Matthew Dwyer, John Hatcliff, Corina Pasareanu, Robby, Shawn Laubach, and Hongjun Zheng. Bandera: Extracting finite-state models from Java source code. In *22nd International Conference on Software Engineering*, pages 439–448, Limerick, Ireland, June 2000. IEEE Computer Society.

[CG01] Shang-Ching Chou and Xiao-Shan Gao. Automated reasoning in geometry. In Robinson and Voronkov [RV01], pages 707–749.

[CGL92] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Nineteenth Annual ACM Symposium on Principles of Programming Languages*, pages 343–354, 1992.

[CGP99] E. M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[CH91] G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 12(3):299–328, 1991.

[Col75] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Second GI Conference on Automata Theory and Formal Languages*, number 33 in Lecture Notes in Computer Science, pages 134–183, Berlin, 1975. Springer-Verlag.

[Coo72] D. C. Cooper. Theorem proving in arithmetic without multiplication. In *Machine Intelligence 7*, pages 91–99. Edinburgh University Press, 1972.

[CU98] M. A. Colón and T. E. Uribe. Generating finite-state abstractions of reactive systems using decidion procedures. In Hu and Vardi [HV98], pages 293–304.

[Dav57] M. Davis. A computer program for Presburger's algorithm. In *Summaries of Talks Presented at the Summer Institute for Symbolic Logic*, 1957. Reprinted in Siekmann and Wrightson [SW83], pages 41–48.

[Dav83] M. Davis. The prehistory and early history of automated deduction. In Siekmann and Wrightson [SW83], pages 1–28.

[DD01] Satyaki Das and David L. Dill. Successive approximation of abstract transition relations. In *Annual IEEE Symposium on Logic in Computer Science01*, pages 51–60. The Institute of Electrical and Electronics Engineers, 2001.

[DDP99] Satyaki Das, David L. Dill, and Seungjoon Park. Experience with predicate abstraction. In Nicolas Halbwachs and Doron Peled, editors, *Computer-Aided Verification, CAV '99*, volume 1633 of *Lecture Notes in Computer Science*, pages 160–171, Trento, Italy, July 1999. Springer-Verlag.

[DE73] George B. Dantzig and B. Curtis Eaves. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory (A)*, 14:288–297, 1973.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962. Reprinted in Siekmann and Wrightson [SW83], pages 267–270, 1983.

[DLNS98] David L. Detlefs, K. Rustan M. Leino, Greg Nelson, and James B. Saxe. Extended static checking. Technical Report 159, COMPAQ Systems Research Center, 1998.

[dMRS02] Leonardo de Moura, Harald Rueß, and Maria Sorea. Lazy theorem proving for bounded model checking over infinite domains. In A. Voronkov, editor, *International Conference on Automated Deduction (CADE'02)*, Lecture Notes in Computer Science, Copenhagen, Denmark, July 2002. Springer-Verlag.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *JACM*, 7(3):201–215, 1960.

[EKM98] Jacob Elgaard, Nils Klarlund, and Anders Möller. Mona 1.x: New techniques for WS1S and WS2S. In Hu and Vardi [HV98], pages 516–520.

[FORS01] J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated Canonization and Solving. In G. Berry, H. Comon, and A. Finkel, editors, *Computer-Aided Verification, CAV '2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 246–249, Paris, France, July 2001. Springer-Verlag.

[FQ02] Cormac Flanagan and Shaz Qadeer. Predicate abstraction for software verification. In *ACM Symposium on Principles of Programming Languages02*, pages 191–202. Association for Computing Machinery, January 2002.

[FS02] Jonathan Ford and Natarajan Shankar. Formal verification of a combination decision procedure. In A. Voronkov, editor, *Proceedings of CADE-19*, Berlin, Germany, 2002. Springer-Verlag.

[Gil60] P. C. Gilmore. A proof method for quantification theory: Its justification and realization. *IBM Journal of Research and Development*, 4:28–35, 1960. Reprinted in Siekmann and Wrightson [SW83], pages 151–161, 1983.

[GMW79] M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[GNTV02] Enrico Giunchiglia, Massimo Narizzano, Armando Tacchella, and Moshe Y. Vardi. Towards an efficient library for SAT: a manifesto. To appear, 2002.

[GS97] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *Conference on Computer Aided Verification CAV'97*, LNCS 1254, Springer Verlag, 1997.

[HJMS02] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Gregoire Sutre. Lazy abstraction. In *ACM Symposium on Principles of Programming Languages02*, pages 58–70. Association for Computing Machinery, January 2002.

[HO80] G. Huet and D. C. Oppen. Equations and rewrite rules: a survey. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic Press, ny, 1980.

[HV98] Alan J. Hu and Moshe Y. Vardi, editors. *Computer-Aided Verification, CAV '98*, volume 1427 of *Lecture Notes in Computer Science*, Vancouver, Canada, June 1998. Springer-Verlag.

[KMM00] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *Computer-Aided Reasoning: An Approach*, volume 3 of *Advances in Formal Methods*. Kluwer, 2000.

[Kur93] R.P. Kurshan. *Automata-Theoretic Verification of Coordinating Processes*. Princeton University Press, Princeton, NJ, 1993.

[LGS$^+$95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:11–44, 1995.

[LGvH$^+$79] D. C. Luckham, S. M. German, F. W. von Henke, R. A. Karp, P. W. Milne, D. C. Oppen, W. Polak, and W. L. Scherlis. Stanford Pascal Verifier user manual. CSD Report STAN-CS-79-731, Stanford University, Stanford, CA, March 1979.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, 1993.

[Min63] Marvin Minsky. Steps toward artificial intelligence. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*. McGraw-Hill Book Company, New York, 1963.

[MMZ$^+$01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, pages 530–535, 2001.

[MSS99] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.

[MT96] Zohar Manna and The STeP Group. STeP: Deductive-algorithmic verification of reactive and real-time systems. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 415–418, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[Nec97] George C. Necula. Proof-carrying code. In *24th ACM Symposium on Principles of Programming Languages*, pages 106–119, Paris, France, January 1997. Association for Computing Machinery.

[Nel81] G. Nelson. Techniques for program verification. Technical Report CSL-81-10, Xerox Palo Alto Research Center, Palo Alto, Ca., 1981.

[NO79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[NSS57] A. Newell, J. C. Shaw, and H. A. Simon. Empirical explorations with the logic theory machine: A case study in heuristics. In *Proc. West. Joint Comp. Conf.*, pages 218–239, 1957. Reprinted in Siekmann and Wrightson [SW83], pages 49–73, 1983.

[Opp78] Derek C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *Journal of Computer and System Sciences*, 16:323–332, 1978.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[ORSvH95] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[Pos21] E. L. Post. Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 43:163–185, 1921. Reprinted in [vH67, pages 264–283].

[Pra60] D. Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960. Reprinted in Siekmann and Wrightson [SW83], pages 162–201, 1983.

[Pre29] M. Presburger. Uber die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. *Compte Rendus du congrés Mathématiciens des Pays Slaves*, pages 92–101, 1929.

[Pug92] W. Pugh. A practical algorithm for exact array dependence analysis. *Communications of the ACM*, 35(8):102–114, 1992.

[Rab78] Michael O. Rabin. Decidable theories. In Jon Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, chapter C8, pages 595–629. North-Holland, Amsterdam, Holland, 1978.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12(1):23–41, 1965. Reprinted in Siekmann and Wrightson [SW83], pages 397–415.

[RS01] Harald Rueß and Natarajan Shankar. Deconstructing Shostak. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 19–28, Boston, MA, July 2001. IEEE Computer Society.

[RV01] A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier Science, 2001.

[Sha01] Natarajan Shankar. Using decision procedures with a higher-order logic. In *Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs 2001*, volume 2152 of *Lecture Notes in Computer Science*, pages 5–26, Edinburgh, Scotland, September 2001. Springer-Verlag. Available at `ftp://ftp.csl.sri.com/pub/users/shankar/tphols2001.ps.gz`.

[Sho81] Robert E. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28(4):769–779, October 1981.

[Sho84] Robert E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, January 1984.

[SKR98] T. R. Shiple, J. H. Kukula, and R. K. Ranjan. A comparison of Presburger engines for EFSM reachability. In Hu and Vardi [HV98], pages 280–292.

[SO99] Natarajan Shankar and Sam Owre. Principles and pragmatics of subtyping in PVS. In D. Bert, C. Choppy, and P. D. Mosses, editors, *Recent Trends in Algebraic Development Techniques, WADT '99*, volume 1827 of *Lecture Notes in Computer Science*, pages 37–52, Toulouse, France, September 1999. Springer-Verlag.

[SR02] N. Shankar and H. Rueß. Combining Shostak theories. In *International Conference on Rewriting Techniques and Applications (RTA '02)*, Lecture Notes in Computer Science. Springer-Verlag, July 2002. Invited Paper.

[SS99] Hassen Saïdi and Natarajan Shankar. Abstract and model check while you prove. In *Computer-Aided Verification, CAV '99*, Trento, Italy, July 1999.

[SS00] Mary Sheeran and Gunnar Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. *Formal Methods in Systems Design*, 16(1):23–58, January 2000.

[SSMS82] R. E. Shostak, R. Schwartz, and P. M. Melliar-Smith. STP: A mechanized logic for specification and verification. In D. Loveland, editor, *6th International Conference on Automated Deduction (CADE)*, volume 138 of *Lecture Notes in Computer Science*, New York, NY, 1982. Springer-Verlag.

[SW83] J. Siekmann and G. Wrightson, editors. *Automation of Reasoning: Classical Papers on Computational Logic, Volumes 1 & 2.* Springer-Verlag, 1983.

[Tar48] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, 1948.

[TH96] Cesare Tinelli and Mehdi Harandi. A new correctness proof of the Nelson-Oppen combination procedure. In Frans Baader and Klaus U. Schulz, editors, *Frontiers of Combining Systems: First International Workshop*, volume 3 of *Applied Logic Series*, pages 103–119, Munich, Germany, March 1996. Kluwer.

[Thé98] Laurent Théry. A certified version of Buchberger's algorithm. In H. Kirchner and C. Kirchner, editors, *Proceedings of CADE-15*, number 1421 in Lecture Notes in Artificial Intelligence, pages 349–364, Berlin, Germany, July 1998. Springer-Verlag.

[Tiw00] Ashish Tiwari. *Decision Procedures in Automated Deduction.* PhD thesis, State University of New York at Stony Brook, 2000.

[TK02] Ashish Tiwari and Gaurav Khanna. Series of abstractions for hybrid automata. In C.J. Tomlin and M.R. Greenstreet, editors, *Hybrid Systems: Computation and Control, 5th International Workshop, HSCC 2002*, volume 2289 of *Lecture Notes in Computer Science*, pages 465–478, Stanford, CA, March 2002. Springer-Verlag.

[vH67] J. van Heijenoort, editor. *From Frege to Gödel: A Sourcebook of Mathematical Logic, 1879–1931.* Harvard University Press, Cambridge, MA, 1967.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings 1st Annual IEEE Symp. on Logic in Computer Science*, pages 332–344. IEEE Computer Society Press, 1986.

[Wan60a] H. Wang. Proving theorems by pattern recognition — I. *Communications of the ACM*, 3(4):220–234, 1960. Reprinted in Siekmann and Wrightson [SW83], pages 229–243, 1983.

[Wan60b] Hao Wang. Toward mechanical mathematics. *IBM Journal*, 4:2–22, 1960.

[Wan63] H. Wang. Mechanical mathematics and inferential analysis. In P. Braffort and D. Hershberg, editors, *Computer Programming and Formal Systems.* North-Holland, 1963.

[Wey80] Richard W. Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, 13(1 and 2):133–170, April 1980.

[Zha97] Hantao Zhang. SATO: An efficient propositional prover. In *Conference on Automated Deduction*, pages 272–275, 1997.