Model Evolution-Based Theorem Proving

Peter Baumgartner, *NICTA*

The area of automated theorem proving is characterized by the development of numerous calculi and proof procedures, from general purpose to rather specialized ones for specific subsets of first-order logic and logical theories. This article highlights two trends that have received considerable attention

over the last 10 years. One is the integration of reasoning methods for propositional and for first-order logic, with a best-of-both-worlds motivation. The other is built-in reasoning support modulo background theories, such as equality and integer arithmetic, which are of pivotal importance for, for example, software verification applications. This survey of the major paradigms in this space comes from the perspective of my own developments, mainly the model evolution calculus. This is an ongoing quest for the convergence of automated reasoning methods.

From Propositional to Instance-Based Methods

In propositional satisfiability, usually called the SAT problem, the DPLL procedure, named after its authors (Davis, Putnam, Logemann, and Loveland^{1,2}) is an important method for building (complete) SAT solvers. This method's popularity is due to its simplicity, its polynomial space requirements, and the fact that, as a search procedure, it's amenable to powerful but relatively inexpensive heuristics for reducing the search space. Thanks to extensions like conflict-driven clause learning,

dynamic weight heuristics, restarts, and carefully engineered data structures, the best SAT solvers today can successfully attack real-world problems with hundreds of thousands of variables and clauses. Indeed, due to these extensions, modern SAT solvers are often subsumed under a new name, conflict-driven clause learning (CDCL) solvers. For the purpose of this article, it's enough to consider the DPLL core component only, and refer the reader to other work³ for more information on CDCL.

Interestingly, the DPLL procedure was actually devised as a proof-procedure for first-order logic. Its treatment of quantifiers is highly inefficient, however, because it's based on enumerating all possible ground instances of an input formula's clause form, and checking the propositional satisfiability of each of these ground instances one at a time. Because of its primitive treatment of quantifiers, the DPLL procedure—which predates Robinson's resolution calculus⁴ by a few years—was quickly overshadowed by resolution as the method of choice for automated first-order reasoning.

One of the key insights in the resolution calculus concerns the use of most general unifiers (MGUs).⁴ In brief, a unifier of two

Two trends are highlighted here, in the area of automated theorem proving: the integration of reasoning methods for propositional and first-order logic, with a best-of-both-worlds motivation; and built-in reasoning support modulo background theories.

literals is a substitution that makes these literals equal (a literal is an atom or a negated atom). A unifier σ is most general if for any unifier τ there's a substitution γ such that $\sigma\gamma = \tau$. Most general unifiers act in concert with the resolution inference rule for reasoning on clauses:

$$\frac{C \vee K \quad L \vee D}{(C \vee D)\sigma}, \text{ if } \sigma \text{ is an MGU of } K \text{ and } \bar{L}.$$

The notation $\bar{L}$ refers to the complement of L , that is, L with the opposite sign.

Starting with the seminal work by Lee and Plaisted in the early 1990s,⁵ researchers began to investigate how to capitalize on both the speed of modern DPLL-based SAT solvers and on successful concepts of first-order theorem proving, such as the use of unification. This led to a family of calculi and proof procedures for first-order logic known as instance-based methods (IBMs).

All IBMs developed so far can be categorized as either one-level or two-level methods. The basic idea behind two-level methods is easy to explain. In an outer loop they maintain a growing set M of instances of a given clause set as determined by the method's inference rules. The set M then is periodically instantiated into a set of ground clauses M^{gr} and passed on to a SAT solver. More precisely, M^{gr} is obtained from M uniformly replacing every variable by some (same) constant. If the SAT solver determines M^{gr} unsatisfiability, it follows that the given clause set is unsatisfiable too, and so the procedure stops.

One factor that distinguishes today's two-level methods from the naive instantiation approach discussed previously are their inference rules to drive the derivation of the clauses in M in a better, conflict-driven way

based on unification. This can best be seen with the Inst-Gen⁶ method and its main inference rule:

$$\frac{C \vee K \quad L \vee D}{(C \vee K)\sigma \quad (L \vee D)\sigma}, \text{ if } \sigma \text{ is a MGU of } K \text{ and } \bar{L}.$$

The Inst-Gen inference rule differs from the resolution rule by keeping the instantiated premises separate instead of combining them into a new clause. In contrast, resolution may generate new clauses of unbounded length.

One-level methods share with two-level methods the principle of working with instances only of the given clauses. One-level methods, however, don't integrate a propositional method but generalize a propositional method for first-order logic.⁷⁻⁹

Model evolution (ME) calculus⁹ is a lifting of the propositional core of the DPLL procedure to the first-order level. To describe how it works, it's instructive to recapitulate the main idea behind propositional DPLL: given a propositional clause set S , we can pick an atom, A , from a clause in S , and create by splitting two new clause sets $S[A/\top]$ and $S[A/\perp]$ (the clause set $S[A/\perp]$ is the set S with every occurrence of A replaced by $\perp$). The clause sets can be further simplified according to Boolean algebra, for example $C \vee \perp \equiv C$ and $C \vee \top \equiv \top$. If a (simplified) clause set contains $\perp$, it's unsatisfiable. If not, another atom occurring in that clause set is picked for splitting, until all atoms have been exhausted (with the conclusion that S is satisfiable), or all the sets generated are shown to be unsatisfiable, which means that S is unsatisfiable.

In view of ME, DPLL as described here can be seen as calculus with the following split inference rule:

$$\frac{\Lambda \vdash S \cup \{A \vee C\}}{\Lambda \cup \{A\} \vdash S \cup \{A \vee C\} \quad \Lambda \cup \{\neg A\} \vdash S \cup \{A \vee C\}}, \text{ if } A \notin \Lambda \text{ and } \neg A \notin \Lambda.$$

The sequent data structure $\Lambda \vdash S$ represents a current clause set S together with a context Λ . The context Λ represents the guesses made so far whether an atom A is set to $\top$ or $\perp$, corresponding to including the literal A or $\neg A$ in the left or right conclusion of an inference, respectively. There are additional inference rules corresponding to simplification by Boolean algebra, not displayed here. Preferring simplification over splitting leads eventually to a sequent $\Lambda \vdash S$ such that no atom in Λ occurs in S . Unsatisfiability then reduces to testing if $\perp \in S$.

The effect of split can be described semantically: a context Λ represents a partial interpretation that can be turned into a total interpretation I_Λ by assigning false to all atoms not occurring in Λ and otherwise assigns the truth values specified by Λ as described previously. This way, the split rule always repairs in its left branch the current interpretation I_Λ toward an interpretation $I_{\Lambda \cup \{A\}}$ that satisfies the previously falsified clause $A \vee C$. In the satisfiable case, the calculus derives a sequent of the form $\Lambda \vdash \emptyset$, and I_Λ provides a model for the initially given clause set, which is a useful feature.

The ME calculus lifts this model generation process to the first-order level. While the overall layout of the calculus is the same, in a sequent $\Lambda \vdash S$ the context Λ now consists of a set of possibly nonground literals, and the clause set S consists of possibly nonground clauses. The lifted split rule is as follows:

$$\frac{\Lambda \vdash S \cup \{C\}}{\Lambda \cup \{L\sigma\} \vdash S \cup \{C\} \quad \Lambda \cup \{\bar{L}\sigma\} \vdash S \cup \{C\}},$$

if $L \in C$, $L\sigma \notin \Lambda$, $\bar{L}\sigma \notin \Lambda$, and σ is a context unifier of C against Λ . A context unifier of a clause $L_1 \vee \dots \vee L_n$, where $n \geq 1$, against Λ is a most general simultaneous unifier of the sets $\{L_i, \bar{K}_i\}$,

for some (fresh variants of) literals $K_i \in \Lambda$. (For technical reasons we assume every Λ contains the pseudo-literal $\neg x$, whose complement unifies with every positive literal.) Akin to two-level methods, a context $\Lambda \vdash S$ is unsatisfiable if some clause in S can be matched by a context unifier to the ground version of Λ that's obtained by simultaneously replacing all variables in Λ by a (the same) constant. In addition to these rules, the calculus contains a number of simplification rules whose purpose is, again like in DPLL, to simplify the clause set and, as a consequence, to speed up the computation.

As with DPLL, a sequent $\Lambda \vdash S$ induces a Herbrand interpretation I_Λ , again serving as a candidate model for S , which possibly needs to be repaired. From the semantic viewpoint, context unifiers serve to identify clauses in S that are falsified by I_Λ in a conflict-driven way. This is how the ME calculus gets its strength, by avoiding split inferences on clauses that are already satisfied.

The previous exposition leaves out many details and improvements, which can be found in other work.⁹

Like the two-level methods, ME works with instances of clauses identified by most general unification to drive proof search. Unlike the two-level methods, it doesn't include a propositional method, it extends it. For this reason, ME can't use a SAT solver in a black-box style, which is a drawback. Consequently, it requires additional efforts to import successful developments from the CDCL world. A good example for that is clause learning, the inclusion of certain derived clauses for the purpose of cutting off search space. In related work, my colleagues and I showed how to extend ME with clause learning, even in a generalized form.¹⁰ On the other hand, its lifted data structure and induced

first-order interpretation enable ME with semantic redundancy criteria that aren't possible with two-level methods.

All IBMs are decision procedures for the class of clauses resulting from the translation of conjunctions of Bernays-Schönfinkel formulas into clause form. Such clauses contain no function symbols, but no other restrictions apply. That fragment, also known as Datalog, is notoriously difficult to decide by resolution methods, even refined ones. On the other hand, refined versions of resolution decide other fragments of first-order logic that aren't decidable by IBMs.

These theoretical differences are reflected by implementations and problem classes they are good at. Indeed, the annual theorem-proving contest is dominated by resolution-based provers and IBMs. Roughly speaking, the former tend to perform better for proving validity, the latter for establishing satisfiability.

Equality

In many theorem-proving applications, a proper treatment of equational theories or equality is mandatory. Software verification applications, for example, often require reasoning about data structures such as lists, arrays, and records, in combination with (integer) arithmetic. Typically, this requires axioms like the following:

$$x \leq z \vee \neg(x \leq y) \vee \neg(y \leq z), \quad (1)$$

$$x \leq y \vee y \leq x, \quad (2)$$

$$\text{select}(\text{store}(a, i, e), i) \approx e, \quad (3)$$

$$\begin{aligned} \text{select}(\text{store}(a, i, e), j) &\approx \\ \text{select}(a, j) \vee i \approx j, \text{ and} \end{aligned} \quad (4)$$

$$\text{select}(a0, i) \leq \text{select}(a0, j) \vee i \approx j \vee \neg(i \leq j). \quad (5)$$

The clauses in Equations 1 and 2 are properties of total orders, in Equations 3 and 4 axiomatize arrays, and in Equation 5 says that the array $a0$ is sorted. The symbol $\approx$ is the equality symbol.

The original resolution calculus didn't feature inference rules for reasoning with equations. A superficial way to fix this consists in adding the axioms for the equality relation to the initially given clauses and leave the calculus untouched. However, it soon turned out that this approach leads to a too big search space and can be used for toy examples only. It took another 25 years until the development of the modern theory of resolution had begun in the 1990s.¹¹ This led to a breakthrough in resolution theory by unifying more or less all resolution variants and improvements until then in a single theoretical framework that was more elegant, general, and powerful. A major outcome of this development was the superposition calculus, a highly improved generalization of the resolution calculus with inference rules for equality reasoning.¹² Superposition is implemented in the leading theorem provers for equational reasoning today.

Roughly speaking, superposition formalizes the concept of replacing equals by equals in the following inference rule:

$$\frac{l \approx r \vee C' \quad L[u] \vee C}{(L[r] \vee C \vee C')\sigma},$$

where σ is an MGU of l and u , u isn't a variable, and certain ordering restrictions apply. The notation $L[u]$ means that the literal L contains the subterm u . The rule replaces u in L by r and applies the substitution to the resulting clause, which contains the rest-clauses C and C' , similarly as in the resolution rule. The ordering restrictions allow the rule to partner only certain, maximal literals (in a

given ordering on terms) and to not replace u by a larger term.

The natural research question arises whether the successful concepts behind superposition can be used for building in equality into different calculi, such as IBMs. In the following, I'll briefly discuss two such developments.

In earlier work, my colleagues and I showed how to integrate a superposition-like inference rule into ME.¹³ The resulting calculus, Model Evolution with Equality (MEE), relies heavily on notions and techniques originally developed for the superposition calculus. As a result, MEE features powerful redundancy criteria that boost efficiency for clause sets that involve equality.

This integration, however, was non-trivial because of the rather different layout of the two calculi. While superposition maintains clause sets as its main data structure, MEE works with contexts, as explained in the "From Propositional to Instance-Based Methods" section, and a set of clauses, paired into sequents. Moreover, we had to move from clauses C to constrained clauses, pairs of the form $C \cdot \Gamma$. The constraint Γ expresses conditions under which the clause C has been derived. Constraints are needed to get a sound calculus. The MEE calculus has two main inference rules, ME's split rule, and the following adaptation of the superposition rule:

$$\frac{l \approx r \quad L[u] \vee C \cdot \Gamma}{(L[r] \vee C \cdot \Gamma \cup \{l \approx r\})\sigma}.$$

In contrast to the superposition rule, the left premise is always a unit clause. This is possible because it's in conjunction with a current sequent $\Lambda \vdash S$, where the left (right) premise is taken from Λ (S , respectively) and the conclusion goes into S . Similar ordering restrictions apply.

MEE and superposition are conceptually rather different calculi and suitable for different problem domains. A description of the MEE implementation, the E-Darwin prover, and experiments with it are available in other work.¹³

Consider again the clause set shown in Equations 1 through 5. When setting up such axiom sets, a natural question is whether they're consistent (satisfiable), clearly a desirable property. However, when passed to a state-of-the-art IBM (such as E-Darwin) or superposition system, neither will terminate on it. The redundancy criteria available with either of these are just too weak to get termination. In general, we can't predict if the prover will ever terminate.

Problems like these motivated me to consider the combination of MEE and superposition. The rationale is to exploit the benefits of MEE and superposition on clause logic fragments for which they are best suited. In the previous example, the clauses in Equations 1 and 2 fall into the Data-log fragment, which MEE is suitable for, and Equations 3, 4, and 5 are best treated by superposition. Superposition terminates on Equations 3, 4 and 5, but not on Equations 1 and 2; MEE terminates on Equations 1 and 2, but not on Equations 3, 4, and 5.

This problem is fixed in the combined MEE and superposition calculus.¹⁴ The main data structure of that calculus is the same as in MEE, sequents with contexts and constrained clauses. The main inference rules are the split and the superposition rule, again taken from MEE, and the following superposition calculus rule adapted for constrained clauses:

$$\frac{l \approx r \vee C' \cdot \Gamma' \quad L[u] \vee C \cdot \Gamma}{(L[r] \vee C \vee C' \cdot \Gamma \cup \Gamma' \cup \{l \approx r\})\sigma} \quad (6)$$

Equation 6 is applied to constrained clauses from S , where $\Lambda \vdash S$ is the

current sequent, and the conclusion goes into S .

The calculus gets its power from allowing the user to tag clauses or their literals for treatment by the MEE or superposition rules, respectively. If desired, the tags can be chosen in a way that the pure version of either calculus results. With suitably chosen tags, the combined calculus terminates on my example. In general, suitably chosen tags can result in significant search space reduction. How well this translates into practice will have to be seen with a future implementation.

Theory Reasoning

The logic considered in the previous section is predicate logic with equality. Although it's expressive enough in a theoretical sense, many applications benefit from building in knowledge about specific (background) theories into a theorem prover by dedicated inference rules or plug-in decision procedures. A prime example is that of linear integer arithmetic (LIA). The satisfiability problem for arbitrary LIA formulas is decidable (by quantifier elimination methods), and theorem provers can greatly benefit from using a LIA decision procedure as a black-box reasoner.

In the example in the "Equality" section, a prover that builds in LIA doesn't need the clauses in Equations 1 and 2 as they're LIA-valid. The axioms in Equations 3 and 4 are still needed, of course, to axiomatize arrays. Technically, the symbols `store` and `select` are said to be free symbols; unlike the symbols of LIA ($\leq$, $+$, $-$, and so on) their meaning is not fixed a priori.

Unfortunately, theorem proving with first-order formulas over LIA and free symbols is difficult both theoretically and practically. Computability results don't even permit a semidecision procedure. That is, in

contrast to first-order logic (with or without equality), we can't devise a theorem prover that, resource limits aside, will prove every theorem.

One way to fix that problem is to restrict to fragments that are computationally more friendly. For example, in related work, my colleagues and I devised ME(LIA), an extension of ME that builds-in LIA.¹⁵ ME(LIA) supports free predicate symbols and integervalued symbolic constants from finite domains. With that, for example, reasoning on arrays in a finite index range 1, ..., a can be expressed, where the constant a is confined to finite intervals, for example, 5, ..., 8. With such restrictions, the calculus is complete, and without them it might still find a proof in some cases. Further restricting the variables to finite intervals, too, makes the logic decidable, and the calculus terminating.

The currently dominating approach to theorem-proving modulo theories, however, is a family of proof procedures subsumed under the name satisfiability modulo theories (SMT). In one of this method's main approaches, DPLL(T), a DPLL-style SAT-solver is combined with a decision procedure for the quantifier-free fragment of the background theory T .¹⁶ The background theory T can itself be a combination of theories, such as lists, arrays, and LIA, provided certain reasonable assumptions are met. Interestingly, superposition provers are well-suited for integration as decision for theories like lists, records, and arrays into a DPLL(T)-solver.¹⁷

Essentially, DPLL(T) lifts these decision procedures to one for arbitrary Boolean combinations of literals over the signature of T . In its simplest form, DPLL(T) works as follows: we start with a quantifier-free (that is, ground) formula whose satisfiability

is to be determined. A trivial example is the clause set $\{\text{select}(a0, 0) \approx e, e > 0, \text{select}(a0, 0) < 0 \vee e > 5\}$. The background theory in question is that of arrays (Equations 3 and 4) and LIA. The DPLL(T) procedure starts with a propositional logic abstraction, say, $\{A, B, C \vee D\}$ that's in one-to-one correspondence to the given clauses. The abstraction is passed to a DPLL-solver. If it returns unsatisfiable, the given clause set is unsatisfiable and the procedure stops. Otherwise, taking the ME view of DPLL discussed previously, we obtain a context, undo the abstraction of the literals in it, and check their T -satisfiability. In the example, if $\{A, B, C\}$ is that context, undoing the abstraction gives $\{\text{select}(a0, 0) \approx e, e > 0, \text{select}(a0, 0) < 0\}$, which is unsatisfiable with regard to the intended theory. The procedure would at this point go into another splitting branch and conclude with the unabstracted context $\{\text{select}(a0, 0) \approx e, e > 0, e > 5\}$, which is satisfiable.

Current SMT-solvers are highly improved versions of this basic procedure. Some of the best-known systems are Yices, CVC4, and Z3, all professionally engineered. The latter is a commercial product developed by Microsoft. SMT-solvers are going to replace SAT-solvers in many applications that require, for example, integer arithmetic. Despite their success, the practical usefulness of SMT-solvers is sometimes rather limited. DPLL(T) is essentially limited to the ground case and resorts to incomplete or inefficient heuristics to deal with quantified formulas. In the example, if, say, the problem at hand prescribes that $a0$ is a sorted array, Equation 5 can be added. However, DPLL(T) can't deal natively with that quantified formula. Instead, a DPLL(T) solver will work with finite approximations of Equation 5—that is,

it will choose (finitely many) ground instances of that equation.

The heuristics for choosing such ground instances often work amazingly well in practice. However, the principle problem of incompleteness remains. On the one hand, we might accept the obvious consequence that some theorems remain unproved. On the other hand, if an incomplete system terminates without a proof, we must not conclude that a given conjecture formula is disproven. That is, we can't say that there's a counterexample that falsifies it—a situation that often occurs during program development or specification design.

Addressing this intrinsic limitation of DPLL(T) for reasoning with quantified formulas is one of the main motivations for some of my work on first-order theorem proving modulo theories. The ME(LIA) approach discussed earlier is of that type, however, it doesn't feature built-in equality and doesn't support free function symbols. The more recent MEE(T) calculus is more powerful in this regard.¹⁸ The calculus layout is similar to the MEE calculus of the "Equality" section. It also works with constrained clauses, however, the constraints now include an additional component c that expresses conditions in terms of the background theory. The main inference rule is as follows:

$$\frac{l \approx r \quad L[u] \vee C \cdot \Gamma \cdot c}{(L[r] \vee C \cdot \Gamma \cup \{l \approx r\} \cdot c) \sigma}.$$

As a simple example for how MEE(T) works, consider the clauses $x > 5 \rightarrow f(x) \approx g(x)$ and $\neg(f(y + y) \approx g(8))$. These clauses will be refuted, essentially, by deriving with the inference rules the set $\{v_1 = v_2 + v_2, v_1 > 5, v_1 = 8\}$ of background theory constraints, which an LIA-solver coupled in a black-box style will determine as satisfiable.

The MEE(T) calculus is rather complex and not easy to implement. Moreover, although it improves over

ME(LIA), it still doesn't support free background-sorted function symbols, such as `read` given previously. Indeed, developing first-order theorem provers, or SMT-solvers for that matter, that provide reasonably complete reasoning support in the presence of such symbols is a major unsolved research challenge. However, investigations into that problem are clearly worth pursuing, with the goal of developing systems that can be used more reliably for both proving theorems and finding counterexamples.

To obtain a, say, (refutationally) complete superposition calculus, we must make sure that any clause set that's closed under inference rule applications (modulo redundancy) and that doesn't contain the empty clause is satisfiable. Without background theories we would usually argue with Herbrand models, which prescribe a fixed, trivial interpretation for function symbols. This doesn't work with background theories such as LIA and instead requires synthesizing functions ranging into the integers. For example, the singleton clause set $\{x > y \rightarrow f(x, y) > f(y, x)\}$ is satisfiable, which can be seen by interpreting f as the projection function of pairs on its first argument.

Unfortunately, the underlying search space isn't enumerable in general and takes theorem proving beyond semidecidability. To somewhat remedy this undesirable situation and to recover semidecidability, we can impose certain a priori restrictions: first, the given clause set has to be sufficiently complete. Intuitively, this means that already the input clause set constrains the interpretations for free function symbols with a background result sort to functions ranging into that background sort. For example, adding the clause $f(x, y) \approx x$ to the clause set (Equations 1 through 5) achieves that. Without it, say, $f(1, 2)$ could be interpreted as a junk noninteger

domain element a in an extended background domain $\mathbb{Z} \cup \{a\}$.

The second restriction requires that the background theory enjoys compactness, that is, satisfiability of all finite subsets of a set S of background formulas entails satisfiability of S .

Others have introduced the hierarchical superposition calculus as a generalization of the superposition calculus for black-box style theory reasoning.¹⁹ This calculus is complete, under the stated restrictions, for clause sets that are fully abstracted (that is, where no literal contains both foreground and background theory symbols). Unfortunately, turning a formula into a fully abstract one might destroy sufficient completeness. Elsewhere, a colleague and I show that this problem can be avoided by using a suitably modified calculus.²⁰

In practice, sufficient completeness is a rather restrictive property. Although there are application areas where we know in advance that every input is sufficiently complete, in most cases this doesn't hold. As a user of an automated theorem prover, we would like to see a best-effort behavior: the prover might, for instance, try to make the input sufficiently complete by adding further theory axioms or forced mapping of terms that could be interpreted as junk to domain elements. Several techniques for that are described in other work.²⁰ For example, this approach applies to the term `select(a0, 0)` in the previous example, which otherwise might be mapped to some junk element. This work also reports on an implementation and first experiments with it, which demonstrate the benefits of the added power.²⁰

The purpose of this article was to emphasize some current trends in automated theorem proving by offering something of a guided tour through underlying reasoning techniques, as indicated by their main inference rules

and data structures. Hopefully, you'll now have a first impression of how the different calculi relate to each other. The tour had a certain focus on instance-based methods, including ME, which have proven to be a successful alternative to classical, superposition-based theorem proving. The developments are far from finished, though. Although the convergence of instance-based methods, SAT-solving, and superposition is already visible, the same can't be said for their theory-reasoning versions. Much remains to be done, for instance, to integrate DPLL(T), instance-based, and superposition calculi for theory reasoning in one theoretical framework (let alone in an efficient implementation). ■

Acknowledgments

I thank the two reviewers who helped improve this article. NICTA (Australia's Information and Communications Technology Research Centre of Excellence) is funded by the Australian government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

1. M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *J. ACM*, vol. 7, no. 3, 1960, pp. 201–215.
2. M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem Proving," *Comm. ACM*, vol. 5, no. 7, 1962, pp. 394–397.
3. J.P.M. Silva, I. Lynce, and S. Malik, "Conflict-Driven Clause Learning SAT Solvers," *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications series, vol. 185, A. Biere et al., eds., IOS Press, 2009, pp. 131–153.
4. J. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *J. ACM*, vol. 12, no. 1, 1965, pp. 23–41.
5. S.-J. Lee and D. Plaisted, "Eliminating Duplicates with the Hyper-Linking

IEEE Computer Society Publications Office

10662 Los Vaqueros Circle, PO Box 3014
Los Alamitos, CA 90720-1314

Lead Editor
Brian Kirk
bkirk@computer.org

Editorial Management
Tammi Titsworth

Manager, Editorial Services
Jenny Stout

Publications Coordinator
isystems@computer.org

Director, Products & Services
Evan Butterfield

Senior Manager, Editorial Services
Robin Baldwin

Digital Library Marketing Manager
Georgann Carter

Senior Business Development Manager
Sandra Brown

Senior Advertising Coordinator
Marian Anderson
manderson@computer.org

Submissions: For detailed instructions and formatting, see the author guidelines at www.computer.org/intelligent/author.htm or log onto *IEEE Intelligent Systems* author center at Manuscript Central (www.computer.org/mc/intelligent/author.htm). Visit www.computer.org/intelligent for editorial guidelines.

Editorial: Unless otherwise stated, bylined articles, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *IEEE Intelligent Systems* does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society. All submissions are subject to editing for style, clarity, and length.

REPRESENTATION AND REASONING

THE AUTHOR

Peter Baumgartner is a principal researcher and research leader with NICTA, Australia's Center of Excellence in Information and Communication Technology. He also is an adjunct associate professor at the Australian National University in Canberra. His research interests include first-order logic theorem proving and its applications to software verification, knowledge representation, and dynamic systems analysis. Baumgartner has a PhD in computer science and a habilitation degree, both from the University of Koblenz-Landau. Contact him at Peter.Baumgartner@nicta.com.au.

- Strategy," *J. Automated Reasoning*, vol. 9, no. 1, 1992, pp. 25–42.
6. H. Ganzinger and K. Korovin, "New Directions in Instantiation-Based Theorem Proving," *Proc. 18th IEEE Symp. Logic in Computer Science*, 2003, pp. 55–64.
7. R. Letz and G. Stenz, "Proof and Model Generation with Disconnection Tableaux," LNCS 2250, R. Nieuwenhuis and A. Voronkov, eds., Springer, 2001.
8. P. Baumgartner, "FDPLL—A First-Order Davis-Putnam-Logeman-Loveland Procedure," *Proc. 17th Int'l Conf. Automated Deduction*, LNAI 1831, D. McAllester, ed., 2000, Springer, pp. 200–219.
9. P. Baumgartner and C. Tinelli, "The Model Evolution Calculus as a First-Order DPLL Method," *Artificial Intelligence*, vol. 172, nos. 4–5, 2008, pp. 591–632.
10. P. Baumgartner, A. Fuchs, and C. Tinelli, "Lemma Learning in the Model Evolution Calculus," *Logic for Programming, Artificial Intelligence, and Reasoning*, LNAI 4246, M. Hermann and A. Voronkov, eds., 2006, Springer, pp. 572–586.
11. L. Bachmair and H. Ganzinger, "On Restrictions of Ordered Paramodulation with Simplification," *Proc. 10th Int'l Conf. Automated Deduction*, LNAI 449, M.E. Stickel, ed., Springer, 1990, pp. 427–441.
12. R. Nieuwenhuis and A. Rubio, "Paramodulation-Based Theorem Proving," *Handbook of Automated Reasoning*, J.A. Robinson and A. Voronkov, eds., Elsevier and MIT Press, 2001, pp. 371–443.
13. P. Baumgartner, B. Pelzer, and C. Tinelli, "Model Evolution with Equality—Revised and Implemented," *J. Symbolic Computation*, vol. 47, no. 9, 2012, pp. 1011–1045.
14. P. Baumgartner and U. Waldmann, "A Combined Superposition and Model Evolution Calculus," *J. Automated Reasoning*, vol. 47, no. 2, 2011, pp. 191–227.
15. P. Baumgartner, A. Fuchs, and C. Tinelli, "ME(LIA)—Model Evolution with Linear Integer Arithmetic Constraints," *Proc. 15th Int'l Conf. Logic for Programming, Artificial Intelligence and Reasoning*, LNAI 5330, I. Cervesato, H. Veith, and A. Voronkov, eds., 2008, Springer, pp. 258–273.
16. R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)," *J. ACM*, vol. 53, no. 6, 2006, pp. 937–977.
17. A. Armando et al., "New Results on Rewrite-Based Satisfiability Procedures," *ACM Trans. Computational Logic*, vol. 10, no. 1, 2009.
18. P. Baumgartner and C. Tinelli, "Model Evolution with Equality Modulo Built-In Theories," *Proc. 23rd Int'l Conf. Automated Deduction*, LNAI 6803, N. Bjoerner and V. Sofronie-Stokkermans, eds., 2011, Springer, pp. 85–100.
19. L. Bachmair, H. Ganzinger, and U. Waldmann, "Refutational Theorem Proving for Hierarchic First-Order Theories," *Applicable Algebra in Engineering, Communication and Computing*, vol. 5, nos. 3–4, 1994, pp. 193–212.
20. P. Baumgartner and U. Waldmann, "Hierarchic Superposition with Weak Abstraction," *Proc. 24th Int'l Conf. Automated Deduction*, LNAI 7898, M.P. Bonacina, ed., 2013, Springer, pp. 39–57.