Model Evolution Based Theorem Proving

Peter Baumgartner

***Abstract*—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. In this article I highlight two trends that have received considerable attention over the last ten years. The 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, e.g., software verification applications. I will survey the major paradigms in this space from the perspective of our own developments, mainly the model evolution calculus. It is an ongoing quest for the convergence of automated reasoning methods.**

I. 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. Its popularity is due to its simplicity, its polynomial space requirements, and the fact that, as a search procedure, it is amenable to powerful but also 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, "CDCL solvers" (conflict-driven clause learning solvers). For the purpose of this article it is enough to consider the DPLL core component only, and refer the reader to [3] for more information on CDCL.

Interestingly, the DPLL procedure was actually devised in origin as a proof-procedure for first-order logic. Its treatment of quantifiers is highly inefficient, however, because it is 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 calculus [4] 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 [4] concerns the use of most general unifiers (MGUs). In brief, a *unifier* of two 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 is a substitution γ such that $\sigma\gamma = \tau$. Most general unifiers act in concert with the *resolution inference rule* for reasoning on clauses:

$$\frac{C \lor K \qquad L \lor D}{(C \lor D)\sigma} \quad \text{if } \sigma \text{ is a MGU of } K \text{ and } \overline{L}$$

The notation $\overline{L}$ 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 methods" 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 be uniformly replacing every variable by some (same) constant. If the SAT solver determines unsatisfiability of M^{gr} it follows that the given clause set is unsatisfiable, too, and so the procedure stops.

What distinguishes today's two-level methods from the naive instantiation approach above, among others, 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-Gen [6] method and its main inference rule, which is the following:

$$\frac{C \lor K \qquad L \lor D}{(C \lor K)\sigma \qquad (L \lor D)\sigma} \quad \text{if } \sigma \text{ is a MGU of } K \text{ and } \overline{L}$$

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, do not *integrate* a propositional method, they *generalize* a propositional method for first-order logic [7], [8], [9]. In the following I will focus on the Model Evolution (ME) calculus [9].

ME has been introduced as a lifting of the propositional core of the DPLL procedure to the first-order level. To describe how it works, it is instructive to recapitulate the main idea behind propositional DPLL: given a propositional clause set S, one picks an atom, say A, from a clause in S, and creates by *splitting* two new clause sets $S[A/\top]$ and $S[A/\bot]$ (the clause set $S[A/\bot]$ is the set S with every occurrence of A replaced by $\bot$). The clause sets can be further simplified according to Boolean algebra, e.g. $C \lor \bot \equiv C$ and $C \lor \top \equiv \top$. If a (simplified) clause set contains $\bot$, it is 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 above can be seen as calculus with the following split inference rule:

$$\frac{\Lambda \vdash S \cup \{A \vee C\}}{\Lambda \cup \{A\} \vdash S \cup \{A \vee C\} \qquad \Lambda \cup \{\neg A\} \vdash S \cup \{A \vee C\}}$$

if $A \notin \Lambda$ 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 $\bot$, 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 $\bot \in S$.

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 above. This way, the split rule always "repairs" in its left branch the current interpretation I_Λ towards 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 — a very useful feature.

The ME calculus can be seen as lifting 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 non-ground literals, and the clause set S consists of possibly non-ground clauses. The lifted split rule is as follows:

$$\frac{\Lambda \vdash S \cup \{C\}}{\Lambda \cup \{L\sigma\} \vdash S \cup \{C\} \qquad \Lambda \cup \{\overline{L}\sigma\} \vdash S \cup \{C\}}$$

if $L \in C$, $L\sigma \notin \Lambda$, $\overline{L\sigma} \notin \Lambda$, and σ is a context unifier of C against Λ. A *context unifier* of a clause $L_1 \vee \cdots \vee L_n$, where $n \geq 1$, against Λ is a most general simultaneous unifier of the sets $\{L_i, \overline{K_i}\}$, 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 is 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 above exposition leaves out many details and improvements, which can be found in [9].

In conclusion, 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 does not *include* a propositional method, it *extends* it. For this reason, ME cannot 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 [10] we have shown how to extend ME with clause learning, even in a generalized form. On the other hand, its lifted data structure and induced first-order interpretation enable ME with semantic redundancy criteria that are not 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 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 are not 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 (better said: superposition-based, see Section II) and IBMs. Very roughly speaking, the former tend to perform better for proving validity, the latter for establishing satisfiability.

II. 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:

$$x \leq z \vee \neg(x \leq y) \vee \neg(y \leq z) \tag{1}$$

$$x \leq y \vee y \leq x \tag{2}$$

$$\mathsf{select}(\mathsf{store}(a,i,e),i) \approx e \tag{3}$$

$$\mathsf{select}(\mathsf{store}(a,i,e),j) \approx \mathsf{select}(a,j) \vee i \approx j \tag{4}$$

$$\mathsf{select}(\mathsf{a0},i) \leq \mathsf{select}(\mathsf{a0},j) \vee i \approx j \vee \neg(i \leq j) \tag{5}$$

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

The original resolution calculus did not 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 lead to a breakthrough in resolution theory by unifying more or less all resolution variants and improvements until then in a single theoretical framework, yet more elegant, general and powerful. A major outcome of that was the *superposition* calculus, a highly improved generalization of the resolution calculus with inference rules for equality reasoning (see [12]). It 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:

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

where σ is an MGU of l and u, u is not 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 will briefly discuss two such developments.

In [13] we have shown how to integrate a superposition-like inference rule into ME. The resulting calculus, 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 Section I, 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:

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

In contrast to the superposition rule, the left premise is always a unit clause. This is possible because it is 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. See again [13] for a description of the MEE implementation, the E-Darwin prover, and experiments with it.

Consider again the clause set above. When setting up such axiom sets a natural question is whether they are 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, one cannot predict if the prover will ever terminate or not.

Problems like these motivated us to consider the *combination* of MEE and superposition. The rationale is to exploit the benefits of MEE and superposition on clause logic fragments they are best suited for. In the example above, the clauses (1) and (2) fall into the Datalog fragment, which MEE is suitable for, and the clauses (3) - (5) are best treated by superposition: superposition terminates on (3)-(5) alone, but not on (1) and (2); MEE terminates on (1) and (2) alone, but not on (3)-(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:

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

The above rule 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 our 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 an implementation, which we do not yet have.

III. Theory Reasoning

The logic considered in the previous section is predicate logic with equality. Although it is expressive enough in a theoretical sense, many applications benefit from building in knowledge about specific theories, or *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 Section II, a prover that builds-in LIA does not need the clauses (1) and (2) as they are LIA-valid. The axioms (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 ($\leq$, $+$, $-$ etc) their meaning is not fixed a priori.

Unfortunately, theorem proving with first-order formulas over LIA and free symbols is very hard, both theoretically and practically. Computability results do not even permit a semi-decision procedure. That is, in contrast to first-order logic (with or without equality), one cannot 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 [15] we have devised ME(LIA), an extension of ME that builds-in LIA. ME(LIA) supports free predicate symbols and integer-valued symbolic constants from finite domains. With that, for example, reasoning on arrays in a finite index range $1 \ldots \mathsf{a}$ can be expressed, where the constant a is confined to finite intervals, e.g., $5 \ldots 8$. With such restrictions, the calculus is complete, and without them it may 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 its 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 it works as follows: given a quantifier-free (i.e., ground) formula whose satisfiability is to be determined. A trivial example is the clause set $\{\mathsf{select}(\mathsf{a0}, 0) \approx \mathsf{e},\ e > 0,\ \mathsf{select}(\mathsf{a0}, 0) < 0 \lor e > 5\}$. The background theory in question is that of arrays (clauses (3) and (4) above) and LIA. The DPLL(T) procedure starts with a propositional logic abstraction, say, $A,\ B,\ C \lor D\}$ that is 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 in Section I, one obtains a context, undoes the abstraction of the literals in it, and checks their T-satisfiability. In the example, if $\{A, B, C\}$ is that context, undoing the abstraction gives $\{\mathsf{select}(\mathsf{a0}, 0) \approx \mathsf{e},\ e > 0,\ \mathsf{select}(\mathsf{a0}, 0) < 0\}$, which is unsatisfiable wrt. the intended theory. The procedure would at this point go into another splitting branch and conclude with the unabstracted context $\{\mathsf{select}(\mathsf{a0}, 0) \approx \mathsf{e},\ e > 0,\ e > 5\}$, which is satisfiable.

Current SMT-solvers are highly improved versions of the above 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, e.g., integer arithmetic. Despite of 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 $\mathsf{a0}$ is a sorted array, the clause (5) can be added. However, DPLL(T) cannot deal natively with that *quantified* formula. Instead, a DPLL(T) solver will work with finite approximations of (5), i.e., chose (finitely many) ground instances of (5).

The heuristics for choosing such ground instances often work amazingly well in practice. However, the principle problem of incompleteness remains. On the one hand, one may accept the obvious consequence that "some" theorems remain unproved. On the other hand, if an incomplete system terminates without a proof, one must not conclude that a given conjecture formula is disproven. That is, one cannot say that there is 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 our work on first-order theorem proving modulo theories. The ME(LIA) approach above is of that kind, however it does not feature built-in equality and does not support free function symbols. The more recent MEE(T) calculus [18] is more powerful in this regard. The calculus layout is similar to the MEE calculus of Section II. It also works with constrained clauses, however the constraints now include an additional component c which expresses conditions in terms of the background theory. The main inference rule is as follows:

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

As a simple example for how MEE(T) works, consider the clauses $x > 5 \to f(x) \approx g(x)$ 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 is to be determined satisfiable by a LIA-solver coupled in a black-box style.

The MEE(T) calculus is rather complex and not easy to implement. Moreover, although it improves over ME(LIA), it still does not support free background-sorted function symbols, such as read above. 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. In the following I will indicate the core of the problem and partial solutions.

In order to obtain a, say, (refutationally) complete superposition calculus one has to make sure that any clause set that is closed under inference rule applications (modulo redundancy) and that does not contain the empty clause is satisfiable. Without background theories one usually argues with Herband models, which prescribe a fixed, trivial interpretation for function symbols. This does not work with background theories such as LIA and instead requires synthesizing functions ranging into the integers. For example, the singleton clause set $\{x > y \to f(x, y) > f(y, x)\}$ is satisfiable, which can be seen by interpreting f as the projection function of pairs on its first argument.

Unfortunately, the underlying search space is not enumerable in general and takes theorem proving beyond semi-decidability. To somewhat remedy this undesirable situation and to recover semi-decidability one 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 above achieves that. Without it, say, $f(1, 2)$ could be interpreted as a "junk" non-integer domain element a in an extended background domain $\mathbb{Z} \cup \{a\}$.

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

In [19], Bachmair, Ganzinger, and Waldmann introduced the hierarchical superposition calculus as a generalization of the superposition calculus for black-box style theory reasoning. It is complete, under the stated restrictions, for clause sets that are fully abstracted (i.e., where no literal contains both foreground and background theory symbols). Unfortunately, turning a formula into a fully abstract one may destroy sufficient completeness. In [20] we show that this problem can be avoided by using a suitably modified calculus.

In practice, sufficient completeness is a rather restrictive property. While there are application areas where one knows in advance that every input is sufficiently complete, in most cases this does not hold. As a user of an automated theorem prover, one would like to see a best effort behaviour: 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. In [20] we describe several techniques for that, not described here. Just to give an example, it applies to the term $\mathsf{select}(\mathsf{a0}, 0)$ in the example above, which otherwise may be mapped to some junk element. We also report on an implementation and first experiments with it, which demonstrate the benefits of the added power.

IV. Conclusions

The purpose of this article was to emphasize some current trends in automated theorem proving. The format taken was that of a guided tour through their underlying reasoning techniques as indicated by their main inference rules and data structures. I hope that by that the reader got a first impression of how the different calculi relate to each other. The tour had a certain focus on instance based methods, including Model Evolution, which have proven to be a successful alternative to classical, superposition based theorem proving. The developments are far from finished, though. While the convergence of instance based methods, SAT-solving and superposition is already well visible, the same cannot be said for their theory-reasoning versions. A lot needs 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).

a) Acknowledgements: The insightful and critical remarks of the two reviewers helped a lot to improve the paper.

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

b) 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. Before joining NICTA, he worked at the University Koblenz-Landau (Germany) from 1990 until 2003, and at the Max-Planck-Institut for Computer Science in Saarbrücken (Germany) from 2003 to 2005. He holds a Ph.D. in Computer Science and a Habilitation degreee, both from the University Koblenz-Landau. His main interests are in first-order logic theorem proving and its applications to software verification, knowledge representation and dynamic systems analysis.

Email: `Peter.Baumgartner@nicta.com.au`

References

[1] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the ACM*, vol. 7, no. 3, pp. 201–215, Jul. 1960. [Online]. Available: http://www.acm.org/pubs/articles/journals/jacm/1960-7-3/p201-davis/p201-davis.pdf

[2] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, Jul. 1962. [Online]. Available: http://www.acm.org/pubs/articles/journals/cacm/1962-5-7/p394-davis/p394-davis.pdf

[3] J. P. M. Silva, I. Lynce, and S. Malik, "Conflict-driven clause learning sat solvers," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 185, pp. 131–153.

[4] J. Robinson, "A machine-oriented logic based on the resolution principle," *JACM*, vol. 12, no. 1, pp. 23–41, January 1965.

[5] S.-J. Lee and D. Plaisted, "Eliminating Duplicates with the Hyper-Linking Strategy," *Journal of Automated Reasoning*, vol. 9, pp. 25–42, 1992.

[6] H. Ganzinger and K. Korovin, "New directions in instantiation-based theorem proving," in *Proc. 18th IEEE Symposium on Logic in Computer Science,(LICS'03)*. IEEE Computer Society Press, 2003, pp. 55–64.

[7] R. Letz and G. Stenz, "Proof and Model Generation with Disconnection Tableaux," in *LPAR*, ser. Lecture Notes in Computer Science, R. Nieuwenhuis and A. Voronkov, Eds., vol. 2250. Springer, 2001.

[8] P. Baumgartner, "FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure," in *CADE-17 – The 17th International Conference on Automated Deduction*, ser. Lecture Notes in Artificial Intelligence, D. McAllester, Ed., vol. 1831. Springer, 2000, pp. 200–219. [Online]. Available: FDPLL-CADE-17.pdf

[9] P. Baumgartner and C. Tinelli, "The Model Evolution Calculus as a First-Order DPLL Method," *Artificial Intelligence*, vol. 172, no. 4-5, pp. 591–632, 2008. [Online]. Available: ME-AIJ-preprint.pdf

[10] P. Baumgartner, A. Fuchs, and C. Tinelli, "Lemma learning in the model evolution calculus," in *Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, ser. LNAI, M. Hermann and A. Voronkov, Eds., vol. 4246. Springer, 2006, pp. 572–586. [Online]. Available: ME-lemma-learning.pdf

[11] L. Bachmair and H. Ganzinger, "On Restrictions of Ordered Paramodulation with Simplification," in *10th International Conference on Automated Deduction*, ser. LNAI 449, M. E. Stickel, Ed. Kaiserslautern, FRG: Springer-Verlag, Jul. 24–27, 1990, pp. 427–441.

[12] R. Nieuwenhuis and A. Rubio, "Paramodulation-based theorem proving." in *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," *Journal of Symbolic Computation*, vol. 47, no. 9, pp. 1011–1045, September 2012. [Online]. Available: MEE-revised-final.pdf

[14] P. Baumgartner and U. Waldmann, "A combined superposition and model evolution calculus," *Journal of Automated Reasoning*, vol. 47, no. 2, pp. 191–227, August 2011. [Online]. Available: MESUP-long.pdf

[15] P. Baumgartner, A. Fuchs, and C. Tinelli, "ME(LIA) – Model Evolution With Linear Integer Arithmetic Constraints," in *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'08)*, ser. Lecture Notes in Artificial

Intelligence, I. Cervesato, H. Veith, and A. Voronkov, Eds., vol. 5330. Springer, November 2008, pp. 258–273. [Online]. Available: MELIA.pdf

[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)," *Journal of the ACM*, vol. 53, no. 6, pp. 937–977, Nov. 2006.

[17] A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz, "New results on rewrite-based satisfiability procedures," *ACM Trans. Comput. Log.*, vol. 10, no. 1, 2009.

[18] P. Baumgartner and C. Tinelli, "Model evolution with equality modulo built-in theories," in *CADE-23 – The 23nd International Conference on Automated Deduction*, ser. Lecture Notes in Artificial Intelligence, N. Bjoerner and V. Sofronie-Stokkermans, Eds., vol. 6803. Springer, 2011, pp. 85–100. [Online]. Available: MEET-draft.pdf

[19] L. Bachmair, H. Ganzinger, and U. Waldmann, "Refutational theorem proving for hierachic first-order theories," *Appl. Algebra Eng. Commun. Comput*, vol. 5, pp. 193–212, 1994.

[20] P. Baumgartner and U. Waldmann, "Hierarchic superposition with weak abstraction," in *CADE-24 – The 24th International Conference on Automated Deduction*, ser. Lecture Notes in Artificial Intelligence, M. P. Bonacina, Ed., vol. 7898. Springer, 2013, pp. 39–57. [Online]. Available: MPI-I-2013-RG1-002.pdf