EPiC Series in Computing

Volume 38, 2016, Pages 63–69

Proceedings of the 1st and 2nd Vampire Workshops

The Uses of SAT Solvers in Vampire*

Giles Reger and Martin Suda

University of Manchester, Manchster, UK

Abstract

Reasoning in a saturation-based first-order theorem prover is generally expensive, involving complex term-indexing data structures and inferences such as subsumption resolution whose (worst case) running time is exponential in the length of the clause. In contrast, SAT solvers are very cheap, being able to solve large problems quickly and with relatively little memory overhead. Consequently, utilising this cheap power within Vampire to carry out certain tasks has proven highly successful. We give an overview of the different ways in which SAT solvers are utilised within Vampire and discuss ways in which this usage could be extended.

1 Introduction

Vampire has been shown to be very fast (see the CASC competition at `http://pages.cs.miami.edu/~tptp/CASC`) and part of this success is due to powerful techniques that rely on the power provided by 'cheap' SAT solvers. For example, the AVATAR splitting approach has been shown [8, 9] to prove many new problems in TPTP, and the Global Subsumption reduction technique led to many new problems being solved when it was introduced.

Therefore, the aim of this paper is to give some insight into the ways that SAT solvers are used within Vampire and to discuss how this use could be extended and improved. The paper reviews each Vampire technique utilising SAT solvers before discussing possible future usage.

2 AVATAR

AVATAR [8, 9] (standing for Advanced Vampire Architecture for Theories And Resolution) is an architecture that tightly integrates a SAT solver for making *splitting decisions*.

The notion of splitting is motivated by the observation that the search space of saturation solvers often becomes full of heavy and long clauses that are undesirable for the inferences commonly used in solvers such as Vampire. The idea behind splitting is that given a set of clauses $S \cup \{C_1 \vee C_2\}$ where the clause $C_1 \vee C_2$ consists of two *variable-disjoint* components C_1 and C_2, the set $S \cup \{C_1 \vee C_2\}$ is unsatisifiable if and only if both $S \cup \{C_1\}$ and $S \cup \{C_2\}$ are unsatisfiable. This suggests that the search space can be split into two search spaces containing smaller clauses. In practice [4] this involves splitting each new clause into components (minimal

*This work was supported by EPSRC.

variable-disjoint sub-clauses) then making a *splitting decision* as to which component to include in the search space and organising proof search so that such decisions can be backtracked.

AVATAR achieves this by using a SAT solver to make splitting decisions. First-order clauses are split into components and then *consistently* (i.e. up to variable renaming) mapped to SAT variables to produce a SAT clause. These SAT clauses representing the first-order clause space are fed to a SAT solver which produces a model that must select a component from each clause, therefore the model gives splitting decisions. These selected components are added to the first-order solver, labelled with their assumptions i.e. the SAT variable corresponding to the asserted component. First-order proof search maintains these labels, ensuring that all clauses derived from a conditional clause are appropriately labelled.

At some point the first-order prover may produce a refutation of a labelled clause, demonstrating that the assumptions made in the derivation of this clause are inconsistent. The labels of this refuted clause are used to add a clause to the SAT solver that prunes part of the splitting decision space and a new model is produced. Whenever the SAT solver model changes it is necessary to remove components no longer in the model, as well as adding new components. The mechanisms for this are beyond the scope of this description and we refer to [8, 9].

It seems likely that an important aspect of the SAT solver usage for AVATAR is *incremental* usage of the solver and the ability to produce *similar models* i.e. those with minimal change. This is interesting as these are not common properties that the developers of SAT solvers are interested in - incrementalality has been studied but it does not appear that this notion of similar models has been explored.

3 Global Subsumption

This is a very effective simplification technique based on the notion of *global propositional subsumption* and originally explored in [5, 6]. Let $D \vee D'$ be a ground clause (for non-empty D') in a set of clauses S. Let S_{gr} be a set of ground clauses implied by S. The ground clause $D \vee D'$ can be replaced by D in S if $S_{gr} \models D$ as D follows from S and subsumes $D \vee D'$. This entailment can be checked by a SAT solver. Note that if D is empty then we have established a contradiction i.e. S_{gr} is unsatisfiable and therefore so is S. Furthermore, note that whilst there are an exponential number of subclauses D, one only needs a linear number of calls to the SAT solver to find a minimal D or show that one does not exist.

The notion can be lifted to a non-ground clause $C \vee C'$ using an *injective* substitution γ from variables in $C \vee C'$ to a set of fresh constants Σ_C. The injectivity of γ is important. Consider the clause $p(x, y) \vee r(x)$ for $S = \{p(x, y) \vee r(x), p(x, x)\}$ and $S_{gr} = \{p(a, a) \vee r(a), p(a, a)\}$, we have $S_{gr} \models p(a, a)$ but $p(x, y)$ does not follow from S.

This leads to the following non-ground global subsumption rule:

$$\frac{C \vee C'}{C}$$

where $S_{gr} \models C\gamma$ for non-empty C' and injective substitution γ. We use an injective substitution that orders variables and maps the ith variable to fresh constant c_i. S_{gr} is constructed using these injective substitutions. We choose not to add further groundings as we want to restrict the size of the SAT clause set. Additionally, as it is important for simplifications to be inexpensive, SAT solvers are run in propagation-only mode. For these two reasons, some possible simplifications may be missed but we believe this to be a necessary trade-off.

As the SAT solver is only used with propagation we do not require a full SAT solver. Further

work may investigate whether it would be more efficient to implement a dedicated propagation-only solver.

4 Instance Generation

This is a saturation-based technique based on the instantiation calculus [3, 5]. The basic idea is as follows. Given a set of first-order clauses S, we produce a ground abstraction $S\bot$ by mapping all variables to a fresh constant $\bot$. If $S\bot$ is unsatisfiable then S is unsatisfiable, note that this is a SAT problem. Otherwise the abstraction may need to be refined by adding clauses to S. As an example of this refinement consider $S = \{p(f(x,a)), \neg p(f(b,y))\}$, giving $S\bot = \{p(f(\bot,a)), \neg p(f(b,\bot))\}$. The set $S\bot$ is SAT satisfiable, whilst the set S is unsatisfiable. The abstraction does not capture all necessary instances of clauses in S. If we add $p(f(b,a))$ and $\neg p(f(b,a))$ to S then $S\bot$ becomes SAT unsatisfiable. This is an instance of the Inst-Gen rule:

$$\frac{C \vee L \qquad D \vee \overline{K}}{(C \vee L)\sigma \qquad (D \vee \overline{K})\sigma}$$

where $\sigma = \texttt{mgu}(L, K)$ and σ is a proper instantiator of L or K (see [3]). Note that in the example the conflicting literals would both be true in the SAT model, this leads to the observation that it is only necessary to apply the Inst-Gen rule to literals that appear in the SAT model, this means that the SAT model can be used as a *selection function* [3, 5].

Instance generation is applied in a saturation loop. All clauses are added to a *passive set* and their groundings are added to a SAT solver; it is sound for the grounding to use a prolific constant from the problem, or a fresh constant, but a single constant should be used. The saturation loop then iterates as follows. The SAT solver is asked for a model, if there is none then the problem is unsatisfiable. Then a clause is activated by selecting its literals using the SAT model, performing all Inst-Gen inferences between it and clauses in the *active set*, and finally adding the clause itself to the *active set*. Generated clauses are added to the *passive set* with their groundings being added to the SAT solver. This is repeated until the *passive set* is empty. Note that if the model changes a clause may need to be moved from the *active set* to the *passive set*. As the instantiation calculus is complete saturation (up to redundancy) implies satisfiability.

The implementation in Vampire is slightly more complex than this as it incorporates the notions of *restarts* and *dismatching constraints*.

As in AVATAR, the SAT solver is being used incrementally. It is not clear whether similar models are desirable as whilst it would reduce the number of clauses 'deselected' it would also reduce the rate at which 'conflicts' with the grounding were found.

5 Finite Model Building

Whilst there is much of Vampire focussed on *theorem proving* we are also interested in establishing *non-theorems*. A useful tool for this is finite model building where we attempt to construct a finite model of the problem.

Vampire implements a MACE-style finite model builder [7] based heavily on the Paradox prover [2]. This is a recent addition to Vampire and has proved effective in proving non-theorems that other techniques within Vampire are not well suited for.

The idea behind this approach is to introduce a growing set of fresh constants representing a finite domain of the finite model we are searching for. For a finite domain of size i we represent

the constraints on the corresponding finite model in a SAT solver and then check if there is a SAT model. If there is then this gives us the finite model, otherwise we add a new constant to the domain and repeat.

The constraints on the finite model are given by the input clauses S and the signature of the problem. For each clause C in S we ground C with the domain constants $c_1, \ldots, c_i$ in all possible ways. We consistently translate each ground literal into a SAT literal to give a set of SAT clauses. Then, for each function symbol f of arity n in the signature it is necessary to add *functional* and *totality* constraints. In the first case we add a translation of the ground clause $f(d_1, \ldots, d_n) \neq d \vee f(d_1, \ldots, d_n) \neq d'$ for all $d_1, \ldots d_n, d$ and d' in the finite domain for distinct d and d'; this captures the fact that f is a function. In the second case we add a translation of the ground clause $f(d_1, \ldots, d_n) = c_1 \vee \ldots f(d_1, \ldots, d_n) = c_i$ for all $d_1, \ldots d_n$ in the finite domain and the full finite domain $c_1, \ldots c_i$; this captures the fact that f must be total.

To use clauses to constrain the finite model in this way it is necessary to first *flatten* clauses to ensure that there are no nested-terms. Additionally we perform two preprocessing steps aimed at reducing the number of variables in each clause as the number of groundings of a clause is exponential in this value. Prior to flattening we perform *definition introduction* where the clause $p(f(a, b), f(a, b))$ would be transformed into the pair of clauses $p(t_1, t_1)$ and $t_1 = f(a, b)$ for fresh constant t_1. Post flattening we perform *splitting* where a clause $p(x, y) \vee q(x, z)$ can be split into the clauses $p(x, y) \vee s(x)$ and $\neg s(x) \vee q(x, z)$ for fresh split predicate symbol s.

Finally, we have implemented a variation of the *sort inference* and *symmetry breaking* techniques described in [2]. These are necessary when searching for larger domain sizes as they (partially) limit the exponential effects of grounding.

Note that this technique can establish unsatisfiability in certain cases i.e. when the size of the domain has been restricted by the input problem. For this reason it is a complete procedure for effectively propositional (EPR) problems.

The SAT solver is used in a non-incremental setting. This allows us to make use of *variable elimination* techniques in the SAT solver, which prove highly effective and are not available in an incremental setting. As the majority of the time is spent in the SAT solver any improvements in SAT solving on problems of these kind can improve the effectiveness of the technique. Additionally, any methods that reduce the size or complexity of the problem passed to the SAT solver (such as symmetry breaking mentioned above) can have a large impact.

6 Future Usage of SAT Solvers

In this section we consider ways in which the current usage of SAT Solvers could be extended and improved. Note that most of the ideas described in this section are either currently under development or not yet under development.

6.1 Tuning the SAT Solver

This is an area that has already received some attention [8, 1]. It is unlikely that the workloads presented to the SAT solver are the same as those often inspected when 'tuning' the SAT solver. Therefore, it is a good idea to consider how different parameters of the SAT solver could be set to improve performance on these workloads. An example of such a parameter is the default value given to a SAT variable. In addition to altering the SAT solver or (trying different SAT solvers) we have also looked at ways of manipulating a model produced by the SAT solver to make it more useful for the intended purpose. For example, [8] describes model minimisation in the AVATAR setting. This is an area of ongoing work.

6.2 AVATAR + Instance Generation

Instance generation can already be combined with resolution. Currently this combination consists of two parts. Firstly, if Global Subsumption is being used the related SAT solver is shared so that clauses can be simplified using a larger set of ground clauses. Secondly, all clauses derived (unconditionally) by the resolution proof attempt are grounded and added to the SAT solver used for Instance Generation. This second step ensures that the model used for Instance Generation also satisfies clauses produced by resolution. Note that clauses conditionally asserted by AVATAR will not be shared in this way so there is no cooperation between AVATAR and Instance Generation .

One approach to combining AVATAR and Instance Generation would be to share the SAT solvers used by each technique. This would allow the models produced in either case to be restricted by information from the other technique. In the case of AVATAR, this could prune areas of the search space shown to be inconsistent by Instance Generation. In the case of Instance Generation the model determines the selection function and captures conflicts, therefore restricting this model using information from AVATAR would avoid deriving additional clauses that represent those conflicts. In both cases this would be sound as only clauses derived from the input clauses are added to the SAT solver.

It is important to note that the SAT literals in the solver used for AVATAR represent *components* whilst the SAT literals in the solver used for Instance Generation represent *ground literals*. Of course, in the AVATAR setting all ground literals are components so there is an overlap in this case *if* these sets of ground literals can overlap. This will only be the case if a constant from the problem is used for grounding in Instance Generation.

In the above approach only derived ground literals from AVATAR contribute to the cooperation. However, we could attempt to relate non-ground components from AVATAR with ground literals from Instance Generation in the following way. First we introduce some terminology: let $D[X]$ be a clause component over variables X and let $[D[X]]$ be the SAT literal representing that component. Whenever, $[D[X]]$ is added to the SAT solver we can also add $[D[X]] \rightarrow [D[C]]$ for the vector C of constants used for grounding in Instance Generation. This will have the effect of (weakly) connecting the models at the non-ground level. If AVATAR shows that the non-ground component $D[X]$ must be true in all models then Instance Generation must choose a literal in $[D[C]]$ to be true in its model. If $[D[C]]$ is shown to be inconsistent then AVATAR cannot select $[D[X]]$ in its model. It is not yet clear if this approach will beneficial and there may be alternative methods for connecting the contents of the SAT solvers in either case.

6.3 AVATAR + Global Subsumption

Currently when Global Subsumption and AVATAR are used together the Global Subsumption method only considers *unconditional* clauses i.e. those with an empty assertion set. This is sound but is weak as it is usually the case that the majority of clauses have non-empty assertion sets with using AVATAR.

A first proposal for combining the techniques is to update the Global Subsumption part only. The idea is that for clauses with a non-empty assertion set we ground the first-order part as usual (giving a SAT clause), but also add SAT literals corresponding to the assertions. Now when a clause is tested for Global Subsumption, we can either assume literals corresponding to its assertions or even corresponding to all components currently active in the first-order solver i.e. true in the AVATAR model. The former corresponds to looking for unconditional reductions and the latter for conditional ones.

The above combination has been implemented but not yet experimented with. This implementation does not look for Global Subsumption inferences which are proper only on the assumption side i.e. it does not attempt to reduce a clauses assertion set. This is because this does not reduce the clause from the perspective of the first-order solver. It is not clear if this would have added benefit.

An extension of this idea would be to allow AVATAR and Global Subsumption to share a SAT solver. However, the benefits to AVATAR are not clear, especially if Global Subsumption checks have the potential to change the AVATAR model. Additionally, as mentioned earlier, Global Subsumption is used in propagation-only mode so sharing the SAT solvers may have no benefit if the current AVATAR model is being asserted.

6.4 All Three Together?

Once we have combined AVATAR with Instance Generation and Global Subsumption separately a natural extension is to combine all three. However, as mentioned above, it may be beneficial to keep a separate SAT solver for Global Subsumption. Experiments are required to explore the potential benefits and costs to the AVATAR or Instance Generation techniques.

6.5 Guiding Literal Selection

In both AVATAR and Instance Generation the model produced by the SAT solver determines how first-order inferences will be performed; in the case of AVATAR it gives the clauses that will be considered and in Instance Generation it gives the selected literals. In both settings there are notions of 'nice' inputs to inferences and it may be possible to guide literal selection in the SAT solver to prefer including these 'nice' components or literals in the SAT model. We consider what this might mean for each setting.

AVATAR. As described above, the SAT model determines the splitting branch being explored. The question is, *are all splitting branches equal?* Perhaps some splitting branches are easier to refute than others. By selecting components that are cheaper to apply inferences to (typically short components with small weight i.e. ground components) we might encourage these 'easier' branches to be explored first. If these contain an unconditional contradiction then this could lead to the problem being solved faster or at all (under limited resources). It is also possible that these 'cheaper' branches can lead to clauses being derived that prune more 'expensive' branches later. There are other ways that literal selection could be used to effect AVATAR usage. Firstly, AVATAR deals with *partial* models produced by *minimising* a model and literal selection could aim to produce a 'small' model to begin with. Secondly, in AVATAR there is a desire for *similar* models. Phase-saving is a current passive technique for producing similar models but more active approaches could be considered i.e. selecting 'older' literals first.

Instance Generation. As described above, the SAT model is used to select literals for applications of the InstGen rule. There are two notions of *niceness* we can consider. Firstly, we could attempt to reduce the number of clauses derived (this is the usual aim of selection). To do this we would prefer ground literals that are groundings of certain kinds of literals; we can use common selection techniques for these literals i.e. fewest variables, deepest variables. Note that this may require us to store some additional information about how these grounded literals were created. Secondly, note that we can apply the InstGen rule between clause $\neg A(x) \vee C(x)$ and both $A(f(y)) \vee D_1$ and $A(f(f(y))) \vee D_2$ but the first application makes the second one

redundant [5]. This is a property taken advantage of by *dismatching constraints*, which prevent the second application after the first has been performed. We could also control literal selection to ensure that the *most general* application is performed first. In combination with dismatching constraints this could significantly reduce the number of applications of the InstGen rule. We note that these two notions of *niceness* are directly in conflict with each other and it is not yet clear if either would be preferable, or perhaps if one is more favourable in certain settings, or if neither has any effect on proof search.

Completeness. Note that both in both AVATAR and Instance Generation it is necessary to consider all models to establish satisfiability. Therefore, the idea behind this preferred literal selection is not to restrict the models considered but change the order in which they are considered with the aim of putting 'nicer' models first.

7 Conclusions

This paper has highlighted and (briefly) described the different techniques that make use of SAT solvers in Vampire. We have also discussed possible extensions to this usage which could increase the impact of these techniques even further. We hope that we will be able to report on the outcome of these ideas in future work.

References

[1] Armin Biere, Ioan Dragan, Laura Kovács, and Andrei Voronkov. Experimenting with SAT solvers in vampire. In *Human-Inspired Computing and Its Applications - 13th Mexican International Conference on Artificial Intelligence, MICAI 2014, Tuxtla Gutiérrez, Mexico, November 16-22, 2014. Proceedings, Part I*, pages 431–442, 2014.

[2] Koen Claessen and Niklas Sörensson. New techniques that improve MACE-style model finding. In *CADE-19 Workshop: Model Computation - Principles, Algorithms and Applications*, 2003.

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

[4] Krytof Hoder and Andrei Voronkov. The 481 ways to split a clause and deal with propositional variables. In Maria Paola Bonacina, editor, *Automated Deduction CADE-24*, volume 7898 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2013.

[5] K. Korovin. Instantiation-based automated reasoning: From theory to practice. In R. A. Schmidt, editor, *22nd International Conference on Automated Deduction CADE-22*, volume 5663 of *Lecture Notes in Computer Science*, pages 163–166. Springer, 2009.

[6] Konstantin Korovin. iProver An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer Berlin Heidelberg, 2008.

[7] William Mccune. A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems. Technical report, Argonne National Laboratory,, 1994.

[8] G Reger, M Suda, and A. Voronkov. Playing with AVATAR. In A. Felty and A. Middeldorp, editors, *25th Internal Conference on Automated Deduction CADE-25*. 2015.

[9] Andrei Voronkov. AVATAR: The architecture for first-order theorem provers. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*, pages 696–710. Springer International Publishing, 2014.