EPiC Series in Computing

Volume 44, 2017, Pages 61–73

Vampire 2016. Proceedings of the 3rd Vampire Workshop

Global Subsumption Revisited (Briefly)

Giles Reger[1] and Martin Suda[2]

[1] University of Manchester, Manchester, UK
[2] TU Wien, Vienna, Austria

Abstract

Global subsumption is an existing simplification technique for saturation-based first-order theorem provers. The general idea is that we can replace a clause C by its subclause D if D follows from the initial problem as D will subsume C. The effectiveness of the technique comes from a cheap, global approach for (incompletely) checking whether D is a consequence of the initial problem. The idea is to produce and maintain a set S of ground clauses that follow from the input (e.g. grounded versions of all derived clauses) and to check whether a grounding of D follows from this set. As this is now a propositional problem this check can be performed by a SAT solver, making it efficient. In this paper we review the global subsumption technique and pose a number of questions related to the practical implementation of global subsumption and possible variations of the approach. We consider, for example, which groundings to place in S, how to select the subclause(s) D to check, how to integrate this technique with the AVATAR approach and whether it makes sense to replace the SAT solver with an SMT solver. This discussion takes place within the context of the Vampire theorem prover.

1 Introduction

Global Subsumption is a very effective simplification technique based on the notion of *global propositional subsumption* and originally explored by Konstantin Korovin [7]. A more detailed description of the technique followed later [8]. It was first implemented in iProver [6] and later included in Vampire [9] as described in [11]. We describe the technique as *very effective* due to how often it proves useful during proof search. Vampire has a portfolio mode consisting of 596 different strategies, 60% of which make use of global subsumption.

Global Subsumption is a general simplification that can be soundly incorporated into almost any saturation-based calculus as it relies only on some simple properties of the proof search. The term *global subsumption* comes from the fact that the technique is *global* as it relies on global information about the clause search space, and is a form of *subsumption* as a new clause is introduced that subsumes an existing one. The exact process is described in detail below.

Many processes in automated reasoning rely on heuristics and various parameters and the choice for these are often described as *black magic*. This paper is part of our wider effort to analyse and understand techniques implemented in the Vampire theorem prover that behave particularly well with the aim of explaining and improving these techniques further. For example, in [12] we performed a number of experiments to understand parameters surrounding the AVATAR architecture [15] and in [5] we reviewed selection function implemeted within Vampire.

This paper is organised as follows. In Section 2 we recap the theory of Global Subsumption and describe its practical implementation in Vampire. Section 3 aims to further clarify the technique via a number of examples. In Section 4 we discuss a number of research questions relating to the practical configuration of the simplification technique and possible extensions. Finally, Section 5 concludes with some remarks on future work.

2 Recapping Global Subsumption

We review Global Subsumption with the necessary level of detail required to understand the process and the rest of this paper. See previously cited publications for the full formal definitions and proofs.

We assume reasonable knowledge of first-order logic. As usual, a *term* is a constant, variable or a n-ary function applied to n terms, an *atom* is an equality between terms or a predicate applied to terms, a *literal* is an atom or its negation and a *clause* is a disjunction of literals. An expression not containing any variables is *ground*. A substitution is a mapping between variables and terms and can be applied to a term to replace variables by their corresponding terms.

We assume the input problem has been transformed into a set of *clauses* (see e.g. [14]). We will refer to this set of clauses as S. We assume a calculus that can extend S with new clauses S' such that S' follows logically from S. So at any point during proof search the set of generated clauses S' follow from the input clauses.

2.1 The Ground Case

Let S_{gr} be a set of ground clauses implied by S. The most obvious such set S_{gr} would be S where every clause has been grounded by replacing variables by certain fresh constants. Indeed, this is what we do in Vampire and the way in which variables are replaced matters, as we discuss below.

Given a *ground* clause $D \vee D'$ in S such that $S_{gr} \models D$ and D' is not empty, we can replace $D \vee D'$ in S by D. Clearly, D is smaller than $D \vee D'$ and subsumes the larger clause. Therefore, the process can be seen as adding D via global reasoning and removing $D \vee D'$ via the standard subsumption argument. If D is empty, this means that S_{gr} itself is inconsistent and by extension so is S and the initial problem.

As we are only dealing with ground clauses, this entailment can be checked by a SAT solver by consistently encoding the ground atoms as propositional variables.

2.2 The Non-Ground Case

The notion can be lifted to a non-ground clause $C \vee C'$. The intuition is as follows. Let $\overline{x}$ be the variables of $C \vee C'$, i.e., the clause represents the closed formula $\forall\overline{x}.(C \vee C')$. If $S_{gr} \cup \{\neg\forall\overline{x}.(C \vee C')\}$ is inconsistent then $C \vee C'$ follows from S. To perform the consistency check, we thus need to take $\exists\overline{x}.(\neg C \wedge \neg C')$ and skolemize it to produce a set of ground unit clauses representing $\{\neg\forall\overline{x}.(C \vee C')\}$.

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

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

where $S_{gr} \models C\gamma$ for a non-empty C' and an *injective* substitution γ from variables in $C \vee C'$ to a set of fresh (with respect to S) constants Σ_C. Effectively the above skolemization is achieved by γ.

The set Σ_C should be fresh with respect to S, for the same reasons Skolem constants need to be fresh. But it is key to the whole process that Σ_C consists of constants appearing in S_{gr}, otherwise it would not be possible for $S_{gr} \models C\gamma$ to hold for any non-trivial γ. For this reason, the set Σ_C can and should be reused for each non-ground global subsumption check. If we assume all clauses have normalised variables (i.e. they use the same variables $x_1, x_2, \ldots$ from left to right) then γ can be considered as global and static.[1]

As one might expect, 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. The argument for injectivity is the same as for using fresh constants in skolemization.

2.3 Implementation

From the above explanations it should be clear that there are a number of implementation choices. Some of these are the topic of this paper and here we describe the initial implementation in Vampire before the experiments described later. We describe the choices and implementation details below.

Choosing the subclause. One must choose the subclause to be removed/kept when checking for global subsumption. There are an exponential number of such subclauses. The original implementation in Vampire attempted to remove each literal in turn by considering growing subclauses excluding the literal to be removed. This led to some redundancy in checks and skipped some subclause checks but still proved to be highly effective. We note that removing just one literal at a time is a reasonable approach as, in this case, the smaller clause would be consider for further reduction via Global Subsumption within a few steps.

Choosing the injective substitution. We use an injective substitution that maps the i-th variable to fresh constant c_i. This means that the order of literals in the clause affects how a literal is grounded. To maximise the chances of similar clauses being grounded in the same way clauses are reordered before grounding. We use a preference ordering with the following ordered preferences:

1. Prefer fewer variables
2. Prefer lighter literals (i.e. less complex literals)
3. Prefer symbols occurring earlier in the symbol ordering (which is arbitrary)
4. Prefer negative literals
5. Break ties arbitrarily but consistently

[1] This the way to understand the workings of global subsumption pragmatically, as in any reasonable implementation the grounded set S_{gr} is at all times reflected by the content of the SAT solver and only monotonically (incrementally) extended with each new check. However, the simplest way to justify correctness of this approach in theory is perhaps the following. One assumes each new check to start only with the set S. Then the substitution γ introduces globally fresh constants. And only just before the check $S_{gr} \models C\gamma$, the set S_{gr} is formed using the original signature along with Σ_C for the grounding. In practice, we just "happen to" each time construct S_{gr} such that it corresponds to the current content of the SAT solver.

The intuition of the first two preferences is that literals with fewer variables and less complex structure are more likely to occur in other clauses and placing them at the front increases the chances of them being grounded in the same way. Note that these preferences are similar to the quality orderings used in literal selection [5].

Choosing the global grounded clauses S_{gr}. As mentioned above, we produce S_{gr} by grounding clauses in S. In fact, we use the substitutions γ to ground the clauses. Therefore, whenever a clause is considered for global subsumption its grounding with respect to γ is also added to S_{gr}.

Usage of the SAT Solver. As it is important for simplifications to be inexpensive, the SAT solver is run in propagation-only mode.

Due to these implementation decisions, some possible simplifications may be missed as either (i) the necessary groundings do not appear in S_{gr}, or (ii) the SAT solver is not powerful enough in its current mode to detect the entailment. This is clearly a trade-off and the subject of the discussions in Section 4.

3 Some Examples of Global Subsumption

We provide a few examples of global subsumption in action using Vampire 4.1 with the options

```
--saturation_algorithm otter --avatar off --global_subsumption on
```

We have taken only the parts of the proof relevant to global subsumption.

A small example

Let us begin by considering a small example where we have the following set of clauses S, its grounding S_{gr} obtained by applying the substitution $x \mapsto \bot$, and a possible translation to SAT.

$$S = \left\{ \begin{array}{c} p(x) \vee q(a) \\ \neg p(x) \vee q(c) \\ f(a) = a \\ f(f(a)) \neq a \end{array} \right\} \quad S_{gr} = \left\{ \begin{array}{c} p(\bot) \vee q(a) \\ \neg p(\bot) \vee q(c) \\ f(a) = a \\ f(f(a)) \neq a \end{array} \right\} \quad SAT = \left\{ \begin{array}{c} 1 \vee 2 \\ \neg 1 \vee 3 \\ 4 \\ 5 \end{array} \right\}$$

Now consider the newly derived clause

$$q(a) \vee q(b) \vee q(c)$$

which can be translated to SAT using the above naming (which is important for consistency) to

$$2 \vee 6 \vee 3.$$

In an attempt to remove each of the literals we construct the following three SAT problems and check if they are inconsistent:

$$\left\{ \begin{array}{c} 1 \vee 2 \\ \neg 1 \vee 3 \\ 4 \\ 5 \\ \neg 2 \\ \neg 6 \end{array} \right\} \quad \left\{ \begin{array}{c} 1 \vee 2 \\ \neg 1 \vee 3 \\ 4 \\ 5 \\ \neg 2 \\ \neg 3 \end{array} \right\} \quad \left\{ \begin{array}{c} 1 \vee 2 \\ \neg 1 \vee 3 \\ 4 \\ 5 \\ \neg 6 \\ \neg 3 \end{array} \right\}$$

As the middle set is inconsistent we can conclude that the newly derived clause can be replaced by the simpler clause

$$q(a) \vee q(c).$$

At this point we note that S is inconsistent due to the last two clauses. However, Global Subsumption is not aware of these equalities (a possible extension for the future, see later) and would not detect this global inconsistency.

From a proof of NLP001+1.p

The following examples come from a proof of the above TPTP problem.

Consider the following set of derived clauses S_1 taken from a certain point in the proof search (there are lots of others, hence the ...).

$$S_1 = \left\{ \begin{array}{rrr} \mathsf{chevy}(sK9) \vee \neg sP0 & \mathsf{car}(sK9) \vee \neg sP0 & \mathsf{white}(sK9) \vee \neg sP0 \\ \mathsf{dirty}(sK9) \vee \neg sP0 & \mathsf{old}(sK9) \vee \neg sP0 & \mathsf{barrel}(sK7, sK9) \vee \neg sP0 \\ sP0 & \ldots & \end{array} \right\}$$

The following clause C_1 is then newly derived

$$C_1 = \begin{array}{l} \neg\mathsf{dirty}(sK9) \vee \neg\mathsf{car}(X0) \vee \neg\mathsf{white}(X0) \vee \neg\mathsf{old}(sK9) \vee \neg\mathsf{chevy}(X0) \vee \neg\mathsf{barrel}(sK7, X0) \vee \\ \neg\mathsf{old}(X0) \vee \neg\mathsf{car}(sK9) \vee \neg\mathsf{dirty}(X0) \vee \neg\mathsf{chevy}(sK9) \vee \neg\mathsf{white}(sK9) \end{array}$$

and the following subclause

$$C_2 = \neg\mathsf{barrel}(sK7, X0) \vee \neg\mathsf{old}(X0) \vee \neg\mathsf{chevy}(X0) \vee \neg\mathsf{dirty}(X0) \vee \neg\mathsf{white}(X0) \vee \neg\mathsf{car}(X0)$$

is selected to check for Global Subsumption. This means that we check

$$(S_1 \cup \{C_1\})[X0 \mapsto c] \models C_2[X0 \mapsto c.]$$

As the SAT solver is using unit propagation it will immediately propagate the unit $sP0$, which will produce a set of units that imply C_2. Therefore, C_1 can be safely replaced by C_2. Next, some more first-order reasoning is used to extend the set of derived clauses by one more clause (and others we do not care about).

$$S_2 = S_1 \cup \{\neg\mathsf{old}(sK9) \vee \neg\mathsf{chevy}(sK9) \vee \neg\mathsf{dirty}(sK9) \vee \neg\mathsf{white}(sK9) \vee \neg\mathsf{car}(sK9), \ldots\}$$

Now the following holds

$$S_2[X0 \mapsto c] \models \bot$$

as this new clause contains literals that relate to negated literals produced by the earlier unit propagation. Therefore, in this proof Global Subsumption is used to simplify a clause and also demonstrate the final inconsistency of the initial problem.

From a proof of SYN417+1.p

The following examples come from a proof of the above TPTP problem.

At one point in the proof search the set of derived clauses S is as follows

$$S = \left\{ \begin{array}{c} sK1 \neq sK3(sK1) \vee sK0 \neq sK2(sK0) \\ sK1 = sK3(sK1) \\ \ldots \end{array} \right\}$$

this is used to reduce

$$f(g(sK2(sK0))) = sK2(sK0) \vee sK0 = sK2(sK0)$$

to

$$f(g(sK2(sK0))) = sK2(sK0)$$

and

$$g(f(X4)) \neq X4 \vee sK0 = sK2(sK0) \vee sK1 = X4$$

to

$$g(f(X4)) \neq X4 \vee sK1 = X4$$

i.e. by removing the $sK0 = sK2(sK0)$ term as it can be shown not to hold globally[2] In another case the clause

$$sK0 \neq sK2(sK0) \vee g(sK2(sK0)) \neq sK3(g(sK2(sK0))) \tag{1}$$

is replaced by

$$sK0 \neq sK2(sK0).$$

The interesting point here is that the original clause (1) never appears in the proof as the second clause can be derived directly from clauses in S (again this is due to the feature introduced later in Section 4.2).

4 An Initial Study

We introduce a number of questions related to the configuration and extension of Global Subsumption and discuss possible answers to these questions, sometimes with initial experimental results.

4.1 What substitutions are good substitutions?

Recall that Vampire currently uses a single grounding for both queries and groundings. This grounding substitution is built by introducing a set of fresh constants $c_1, c_2, \ldots$ and always replacing the first variable in a clause by c_1 and the second variable by c_2 and so on. We discussed why these substitutions need to be injective previously.

Clearly we want to use the same constants to form the query substitution and the grounding substitution as the goal is to show that the query clause follows from the grounded clauses, which would not be the case if they did not contain the same constants. The next question is, how do we choose those constants and the way in which we organise them in the substitution?

Consider the very small clause set

$$S = \left\{ \begin{array}{c} C_1 = p(x) \vee \neg q(y) \vee r(y) \\ C_2 = \neg p(x) \end{array} \right\}$$

which when grounded using the above scheme would produce the following grounded clause set:

$$S_{gr} = \left\{ \begin{array}{c} C_1\gamma = p(a) \vee \neg q(b) \vee r(b) \\ C_2\gamma = \neg p(a) \end{array} \right\}.$$

[2]This relies on the technique introduced in Section 4.2 that uses SAT solving under assumptions. Here the assumed literal $sK0 \neq sK2(sK0)$ is not needed to show unsatisfiability as it follows from the contents of the SAT solver.

Table 1: Experimental results for varying substitutions.

	Total	Unique for this AVATAR value	Unique Overall
		AVATAR on	
Standard	8873	36	23
Backward	**8882**	**54**	**38**
First	8845	31	25
		AVATAR off	
Standard	8110	26	5
Backward	8099	24	7
First	8029	20	6

Now consider the newly derived clause $\neg q(x) \vee r(x)$. The Global Subsumption check is $S_{gr} \models \neg q(a) \vee r(a)$, which doesn't hold. However, if we had included the grounding

$$p(b) \vee \neg q(a) \vee r(a)$$

in S_{gr} then it would hold. This demonstrates that the choice of grounded clauses we add to S_{gr} matters. Note that the choice of query substitution is a symmetrical concern so we can focus only on the groundings.

4.1.1 Two Implemented Ideas

We implemented two ideas to explore the effects of varying substitutions. These were:

1. Reverse the ordering (backward) to explore the impact this ordering makes (this refers to the preference ordering introduced earlier).
2. Given a clause of length n, produce n substitutions where each literal is put first in the ordering (but keep the rest of the ordering the same). The idea here is to increase the chances of the query clause matching with a ground clause.

Table 1 gives the results of an experiment where we also varied whether AVATAR was on or off (see the discussion of AVATAR in Section 4.4). Surprisingly, for AVATAR on the backward option performed best. Recall that, when AVATAR is in use, clauses tend to be shorter and simpler as they are necessarily unsplittable. Therefore, the criteria used in the preference ordering will have less effect, perhaps highlighting less preferential qualities such as negative literals. This suggests that it could be worthwhile exploring other literal orderings, if other methods for constructing substitutions do not immediately supersede this approach.

The first option never performed best, although had some unique solutions. This may be due to the increased work needed to deal with the additional substitutions. If this is the case (we will check in future work) then this supports the idea that we should attempt to minimise the number of substitutions we make use of. It is likely that a more guided approach to constructing substitutions (as discussed below) would be more appropriate.

4.1.2 Future Ideas

Here we discuss some other ideas for varying substitutions that we have not yet explored.

Special treatment for units. Unit clauses are very useful in the set of grounded clauses, especially as the SAT solver is typically run in propagation-only mode. However, units $p(x)$ and $q(x)$ will only be grounded using the first constant. This means that no contradiction would be detected when checking $\neg p(x) \vee \neg q(y)$ as one of $\neg p(x)$ or $\neg q(y)$ would necessarily be grounded differently from $p(x)$ or $q(x)$. The idea to try would be to ground unit clauses with multiple fresh constants.

Non-injective substitutions. Substitutions only need to be injective for the query clause (for the reasons discussed previously). We could use non-injective substitutions in the groundings. Such grounded clauses still follow from the input clauses as they are instances of such clauses. One example of a non-injective substitution is the single constant substitution (i.e. $\{x_1, \ldots, x_n \mapsto a\}$). This would also solve the above case of where grounding $p(x)$, $q(y)$ and $\neg p(x) \vee \neg q(y)$ does not lead to a contradiction.

Lookahead. Here we refer to the *lookahead* approach to literal selection described in [5]. The idea there was to use information about the current active clauses to *look-ahead* and estimate the effect of selecting a particular literal. The idea here is to *look-ahead* and estimate the effect of selecting a particular grounding.

Intuitively, when we are talking about *good* groundings to add we are trying to guess what groundings are already in the SAT solver. A lookahead approach would maintain an index of the groundings previously used and use this to maximise the chances of grounding a literal in the same way as it was previously grounded.

We note that there is a similar idea in the grounding operation of E-matching [1, 2] (without the **E**quality bit) and want to explore whether there is an overlap in ideas here.

4.2 What subclauses are good subclauses?

This is a question that we think we have a reasonable answer for, i.e., we have a method for finding a minimal subclause that globally subsumes the initial clause.

SAT Solving under assumptions. Briefly, this is a SAT solving technique where some SAT variables $V = \{v_1, \ldots, v_n\}$ are assumed to have a certain value before solving. If the SAT solver returns an unsatisfiable result then it can also return a (reduced) set of variables $A \subseteq V$ that were used to establish that result. See [3, 4] for further details.

For Global Subsumption we use this technique as follows. For a newly derived clause C and substitution γ (see above for how this should be chosen) we construct the query clause $C\gamma$. Let $L_1 \vee \ldots \vee L_n$ be the corresponding SAT clause (using some appropriate naming). We then assume the complements $\bar{L}_i$ of these literals and check for satisfiability.[3] The set of assumptions A returned by an unsatisfiable result gives us a subclause that globally subsumes the original clause. If all literals are used then there is no global subsumption. If no literals are used then we have shown unsatisfiability of the problem via grounding and propositional reasoning and can report this.

Minimising the assumptions. The set of assumptions returned by solving under assumptions is not necessarily minimal as it depends on how the SAT solver established unsatisfiability.

[3]In the current setup we have just added $C\gamma$ to S_{gr} so the result is necessarily unsatisfiable. But under different variations, if it were satisfiable then it would just mean there is no global subsumption to be performed.

We can, however, attempt to find a (subset-) minimal set of assumptions. This is done by removing one of the assumptions and checking if the unsatisfiability still holds. We implement three different options:

1. No minimisation.
2. In order: try and eliminate each assumption in the order they appear.
3. Randomized order (default): try and eliminate assumptions in a random order.

An experiment. We do not have any experimental results to report here concerning the difference between the solving-under-assumptions based approach and the previous approach. However, we can report that the new approach is an improvement.

The following table reports a small experiment comparing the effects of the different minimisation option. This shows that randomising the order in which we minimise works best.

	Total Solved	Unique Solved
off	8959	16
on	8965	21
randomized	**8981**	**38**

4.3 What should the SAT solver look like?

In the current implementation of Global Subsumption we generally use the MiniSAT solver [3] in *propagation-only* mode (there is also the option of using other solvers but MiniSAT is default). This mode means that the SAT solver does not perform any decision steps; it only propagates units until there are no more unit clauses. The reason for this choice is that we want Global Subsumption (as a frequently applied forward simplification rule) to be very cheap (we discuss this point further below).

However, the assumption that it is necessary to only perform propagations may be flawed. We performed a small experiment where we ran Vampire in default mode with Global Subsumption switched on. We implemented a variation of Global Subsumption where the SAT solver was used fully. The results are as follows (note that we have not corrected for problems solved without using Global Subsumption).

	Total Solved	Unique Solved
Propagation Only	8935	61
Full	8920	46

This hints[4] that (i) full SAT solving has a detrimental effect in general, but (ii) can lead to some problems being solved that were not solved before. Clearly further investigation is required. What is currently unclear is whether the solutions found using full SAT solving would also be found by other methods, e.g., instance based reasoning.[5]

If we assume that we will only perform Global Subsumption using the cheaper propagation-only mode then we no longer require a full SAT solver for this technique. This suggests that it could be worth developing a dedicated propagation-only solver that would not need to keep track of the information required to perform splitting and backtracking.

[4]See [13] for a discussion of why such experiments cannot tell the full picture.
[5]See [11] for a description of this technique implemented in Vampire.

Table 2: Experimental results for combining Global Subsumption with AVATAR.

	Total	Unique
nonsplittable components = known		
off	9030	131
current	6149	6
full	3250	-
nonsplittable components = all		
off	8615	47
current	933	-
full	699	-

	Total	Unique
nonsplittable components = all_dependent		
off	8678	16
current	5915	-
full	3416	-
nonsplittable components = none		
off	8832	43
current	6853	-
full	3586	-

4.4 Can Global Subsumption play nicely with AVATAR?

AVATAR [15, 12, 11] is an architecture that uses a SAT solver to perform clause splitting. To organise splitting decisions, clauses are extended with sets of *assertions*, capturing the splitting context, to become A-clauses. Reductions need to be careful of assertions as clauses taking part in a reduction need to be in the same context or information needs to be stored to backtrack reductions carried out in differing contexts.

Previously, Global Subsumption was not used with AVATAR as there is no good mechanism for performing backtracking. We have implemented an approach that does not require backtracking as it always performs reductions with respect to the appropriate splitting context. The idea is to add the additional information about assertions to the set of grounded clauses. In this encoding the assertions are treated as additional literals and encoded in a similar, consistent, way.

Once assertions have been encoded we need to assume enough information about the splitting context for reductions to be performed safely. There are two options for this, leading to the following three options for this combination:

1. off: clauses with assertions are not used with Global Subsumption.
2. current: assertions of the current clause are assumed.
3. full: an encoding of the full current branch is assumed.

The difference between the last two options is that with current we activate fewer clauses to support the Global Subsumption check, but the subsuming clause (if discovered) will unconditionally replace the subsumed on the current branch. With full, stronger reductions (employing more clauses) may be possible, but the reduction may need to be backtracked when the branch later changes. This backtracking is handled by the general AVATAR framework as explained in the previously cited relevant work.

An experiment. Table 2 reports on an initial experiment looking at these three options and how they interact with a key AVATAR option. The AVATAR option we vary is how nonsplittable components are dealt with. This is interesting as this option affects the proportion of clauses that are considered for splitting and, therefore, may become (in-)eligible for Global Subsumption. The main observation is that keeping Global Subsumption and AVATAR separate performs the best. This is most likely due to the fact that Global Subsumption is relatively expensive. Spending extra time on reductions which are only conditional, i.e., relevant only on the current branch, and maintaining the extra information needed to keep track of assertions in the SAT solver does not seem to pay off.

Future Ideas. There are two ideas we have not tried yet:

1. *Reducing assertions.* Given an A-Clause it may be possible to use Global Subsumption to reduce the *assertions* rather than the literals of a clause. This is not a proper reduction but could be useful as it makes the clause applicable to a wider range of splitting contexts.
2. *Using the same SAT solver.* Instead of having to encode the AVATAR information in assumptions for the Global Subsumption solver, we could share the SAT solver. This would mean that Global Subsumption would automatically be aware of the current splitting context but also its justification (in terms of learned clauses etc). It is not yet clear whether there would be any impact on AVATAR from this sharing beyond slightly more work for the SAT solver. This is an idea we also discussed in [11] and still have not explored.

4.5 Can Global Subsumption play with Theories?

Vampire can reason with a number of theories e.g. real arithmetic. Global Subsumption uses a SAT solver to reason globally about reductions. The question is whether this could be extended to theories by replacing the SAT solver with a SMT solver. This is similar to the extension of AVATAR to theories [10]. We consider two approaches but have not yet implemented either.

Uninterpreted Functions. One could include the theory of uninterpreted functions by wrapping the SAT solver in a simple congruence closure procedure (as has been done with AVATAR). This would allow Global Subsumption to perform ground equational reasoning e.g. using $f(a) = b$ and $f(c) = a$ to reduce $p(x) \vee f(f(c)) \neq b$ to $p(x)$.

General SMT. To support theories in general we can replace the SAT solver by an SMT solver for the necessary theories. Appropriately sorted fresh constants should be introduced for the grounding and grounded clauses should then be appropriately translated into the SMT-language, with interpreted operators handled properly. Importantly, if this approach is combined with the AVATAR Modulo Theories work [10] then the assertions from A-clauses must be translated back into the SMT-terms they represent.

Discussion. Both of these extensions go against the idea that Global Subsumption should be cheap and it is not clear whether this extra work will be worthwhile. However, the possible benefits (i.e. simplifying complex theory clauses) suggest that this is worth exploring. Additionally, as previously discussed there may be ways to make use of an expensive approach to Global Subsumption in some cases.

4.6 When to perform Global Subsumption?

Currently, Global Subsumption is employed as a *forward simplification* in Vampire. This means that it is applied to newly derived clauses and groundings of all newly derived clauses are added to the SAT solver. The advantage of this is that new clauses are eagerly simplified and that the SAT solver has the maximum amount of information. The disadvantage is that Global Subsumption needs to be very cheap (as it is performed very frequently) and, less importantly, the SAT solver may be given too much information, slowing down SAT solving.

We note that this usage also has some odd side-effects when the *discount* saturation strategy is used. In this setup other forward simplifications, such as subsumption, only make use of *active* clauses (not *passive* ones)[6]. This means that Global Subsumption may perform subsumptions. E.g., if $p(x)$ was previously generated and $p(x) \vee q(y)$ is newly derived then the later clause will be replaced by $p(x)$ as this subclause already exists in the set of ground clauses. Note that we would not perform this replacement if the later clause was $q(y) \vee p(x)$ due to the ordering issue discussed earlier.

An alternative is to perform Global Subsumption at *activation*, i.e., to reduce clauses when they are activated using only previously activated clauses. The only advantage of this is that it would be performed far less frequently (with a smaller set of propositional clauses) and therefore more expensive operations could be considered. If the expense of global subsumption checks is found to be an issue during this study then it would make sense to also explore this option.

5 Conclusion

This paper has described initial work on our pragmatic study of the Global Subsumption technique. This involved recapping the idea, with the aim of communicating the intuition rather than the deep theory (which has been given elsewhere). We then discussed a number of research questions and presented some initial, yet inconclusive, results.

Our aim is to complete this study by addressing the research questions in full via extensions to Vampire and appropriate experimentation. We hope to report on the results of the full study in the future.

References

[1] Leonardo Mendonça de Moura and Nikolaj Bjørner. Efficient e-matching for SMT solvers. In *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, pages 183–198, 2007.

[2] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.

[3] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[4] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electr. Notes Theor. Comput. Sci.*, 89(4):543–560, 2003.

[5] Kryštof Hoder, Giles Reger, Martin Suda, and Andrei Voronkov. Selecting the selection. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning: 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 – July 2, 2016, Proceedings*, pages 313–329. Springer International Publishing, 2016.

[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] Konstantin 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.

[8] Konstantin Korovin. *Inst-Gen – A Modular Approach to Instantiation-Based Automated Reasoning*, pages 239–270. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[6]For an explanation of what the terms active and passive mean here see [9].

[9] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35, 2013.

[10] Giles Reger, Nikolaj Bjørner, Martin Suda, and Andrei Voronkov. AVATAR Modulo Theories. In C. Benzmüller, G. Sutcliffe, and R. Rojas, editors, *Proceedings of the 2nd Global Conference on Artificial Intelligence*, EPiC Series in Computing, page To appear. EasyChair Publications, 2016.

[11] Giles Reger and Martin Suda. The uses of sat solvers in vampire. In Laura Kovács and Andrei Voronkov, editors, *Proceedings of the 1st and 2nd Vampire Workshops*, volume 38 of *EPiC Series in Computing*, pages 63–69. EasyChair, 2016.

[12] Giles Reger, Martin Suda, and Andrei Voronkov. Playing with avatar. In P. Amy Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, pages 399–415. Springer International Publishing, 2015.

[13] Giles Reger, Martin Suda, and Andrei Voronkov. The challenges of evaluating a new feature in vampire. In Laura Kov\'acs and Andrei Voronkov, editors, *Proceedings of the 1st and 2nd Vampire Workshops*, volume 38 of *EPiC Series in Computing*, pages 70–74. EasyChair, 2016.

[14] Giles Reger, Martin Suda, and Andrei Voronkov. New Techniques in Clausal Form Generation. In C. Benzmüller, G. Sutcliffe, and R. Rojas, editors, *Proceedings of the 2nd Global Conference on Artificial Intelligence*, EPiC Series in Computing, page To appear. EasyChair Publications, 2016.

[15] 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.