Reliable Reconstruction of Fine-Grained Proofs in a Proof Assistant

Hans-Jörg Schurr[1], Mathias Fleury[2,3], and Martin Desharnais[4]

[1] University of Lorraine, CNRS, Inria, and LORIA, Nancy, France
hans-jorg.schurr@inria.fr
[2] Johannes Kepler University Linz, Linz, Austria
mathias.fleury@jku.at
[3] Max-Planck Institute für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
[4] Universität der Bundeswehr München, München, Germany
martin.desharnais@unibw.de

Abstract. We present a fast and reliable reconstruction of proofs generated by the SMT solver veriT in Isabelle. The fine-grained proof format makes the reconstruction simple and efficient. For typical proof steps, such as arithmetic reasoning and skolemization, our reconstruction can avoid expensive search. By skipping proof steps that are irrelevant for Isabelle, the performance of proof checking is improved. Our method increases the success rate of Sledgehammer by halving the failure rate and reduces the checking time by 13%. We provide a detailed evaluation of the reconstruction time for each rule. The runtime is influenced by both simple rules that appear very often and common complex rules.

Keywords: automatic theorem provers · proof assistants · proof verification

1 Introduction

Proof assistants are used in verification and formal mathematics to provide trustworthy, machine-checkable formal proofs of theorems. Proof *automation* reduces the burden of finding proofs and allows proof assistant users to focus on the core of their arguments instead of technical details. A successful approach implemented by "hammers," like Sledgehammer for Isabelle [15], is to heuristically selects facts from the background; use an external automatic theorem prover, such as a satisfiability modulo theories (SMT) solver [12], to filter facts needed to discharge the goal; and to use the filtered facts to find a trusted proof.

Isabelle does not accept proofs that do not go through the assistant's inference kernel. Hence, Sledgehammer attempts to find the fastest internal method that can recreate the proof (*preplay*). This is often a call of the *smt* tactic, which runs an SMT solver, parses the proof, and reconstructs it through the kernel. This reconstruction allows the usage of external provers. The smt tactic was originally developed for the SMT solver Z3 [18, 34].

The SMT solver CVC4 [10] is one of the best solvers on Sledgehammer generated problems [14], but currently does not produce proofs for problems with quantifiers. To reconstruct its proofs, Sledgehammer mostly uses the smt tactic based on Z3. However, since CVC4 uses more elaborate quantifier instantiation techniques, many problems provable for CVC4 are unprovable for Z3. Therefore, Sledgehammer regularly fails to find a trusted proof and the user has to write the proofs manually. veriT [19] (Sect. 2) supports these techniques and we extend the smt tactic to reconstruct its proofs. With the new reconstruction (Sect. 3), more smt calls are successful. Hence, less manual labor is required from users.

The runtime of the smt method depends on the runtime of the reconstruction and the solver. To simplify the reconstruction, we do not treat veriT as a black box anymore, but extend it to produce more detailed proofs that are easier to reconstruct. We use detailed rules for simplifications with a combination of propositional, arithmetic, and quantifier reasoning. Similarly, we add additional information to avoid search, e.g., for linear arithmetic and for term normalization. Our reconstruction method uses the newly provided information, but it also has a *step skipping* mode that combines some steps (Sect. 4).

A very early prototype of the extension was used to validate the fine-grained proof format itself [7, Sect. 6.2, second paragraph]. We also published some details of the reconstruction method and the rules [25] before adapting veriT to ease reconstruction. Here, we focus on the new features.

We optimize the performance further by tuning the search performed by veriT. Multiple options influence the execution time of an SMT solver. To fine-tune veriT's search procedure, we select four different combinations of options, or *strategies*, by generating typical problems and selecting options with complementary performance on these problems. We extend Sledgehammer to compare these four selected strategies and suggest the fastest to the user. We then evaluate the reconstruction with Sledgehammer on a large benchmark set. Our new tactic halves the failure rate. We also study the time required to reconstruct each rule. Many simple rules occur often, showing the importance of step skipping (Sect. 5).

Finally, we discuss related work (Sect. 6). Compared to the prototype [25], the smt tactic is now thoroughly tested. We fixed all issues revealed during development and improved the performance of the reconstruction method. The work presented here is integrated into Isabelle version 2021; i.e., since this version Sledgehammer can also suggest veriT, without user interaction. To simplify future reconstruction efforts, we document the proof format and all rules used by veriT. The resulting reference manual is part of the veriT documentation [40].

2 veriT and Proofs

The SMT solver veriT is an open source solver based on the CDCL($\mathcal{T}$) calculus. In proof-production mode, it supports the theories of uninterpreted functions with equality, linear real and integer arithmetic, and quantifiers. To support quantifiers veriT uses quantifier instantiation and extensive preprocessing.

veriT's proof syntax is an extension of SMT-LIB [11] which uses S-expressions and prefix notation. The proofs are refutation proofs, i.e., proofs of $\bot$. A proof is an indexed list of steps. Each step has a conclusion clause `(cl ..)` and is annotated with a rule, a list of premises, and some rule-dependent arguments. veriT distinguishes 90 rules [40]. Subproofs are the key feature of the proof format. They introduce an additional *context*. Contexts are used to reason about binders, e.g., preprocessing steps like transformation under quantifiers.

The conclusions of rules with contexts are always equalities. The context models a substitution into the free variables of the term on the left-hand side of the equality. Consider the following proof fragment that renames the variable name `x` to `vr`, as done during preprocessing:

```
(assume a0 (exists (x A) (f x))
(anchor :step t3 :args (:= x vr))
(step t1 (cl (= x vr)) :rule refl)
(step t2 (cl (= (f x) (f vr))) :rule cong :premises (t1))
(step t3 (cl (= (exists (x A) (f x))
                (exists (vr A) (f vr))) :rule bind)
```

The `assume` command repeats input assertions or states local assumptions. In this fragment the assumption `a0` is not used. Subproofs start with the `anchor` command that introduces a context. Semantically, the context is a shorthand for a lambda abstraction of the free variable and an application of the substituted term. Here the context is $x \mapsto \text{vr}$ and the step `t1` means $(\lambda x.\ x)\ \text{vr} = \text{vr}$. The step is proven by congruence (rule `cong`). Then congruence is applied again (step `t2`) to prove that $(\lambda x.\ f\ x)\ \text{vr} = f\ \text{vr}$ and step `t3` concludes the renaming.

During proof search each module of veriT appends steps onto a list. Once the proof is completed, veriT performs some cleanup before printing the proof. First, a pruning phase removes branches of the proof not connected to the root $\bot$. Second, a merge phase removes duplicated steps. The final pass prepares the data structures for the optional term sharing via name annotations.

3 Overview of the veriT-Powered smt Tactic

Isabelle is a generic proof assistant based on an intuitionistic logic framework, *Pure*, and is almost always only used parameterized with a logic. In this work we use only Isabelle/HOL, the parameterization of Isabelle with higher-order logic with rank-1 (top level) polymorphism. Isabelle adheres to the LCF [26] tradition. Its kernel supports only a small number of inferences. Tactics are programs that prove a goal by using only the kernel for inferences. The LCF tradition also means that external tools, like SMT solvers, are not trusted.

Nevertheless, external tools are successfully used. They provide relevant facts or a detailed proof. The Sledgehammer tool implements the former and passes the filtered facts to trusted tactics during preplay. The smt tactic implements the latter approach. The provided proof is checked by Isabelle. We focus on the smt tactic, but we also extended Sledgehammer to also suggest our new tactic.

The smt tactic translates the current goal to the SMT-LIB format [11], runs an SMT solver, parses the proof, and replays it through Isabelle's kernel. To choose the smt tactic the user applies `(smt (z3))` to use Z3 and `(smt (verit))` to use veriT. We will refer to them as z-smt and v-smt. The proof formats of Z3 and veriT are so different that separate reconstruction modules are needed. The v-smt tactic performs four steps:

1. It negates the proof goal to have a refutation proof and also encodes the goal into first-order logic. The encoding eliminates lambda functions. To do so, it replaces each lambda function with a new function and creates app operators corresponding to function application. Then veriT is called to find a proof.
2. It parses the proof found by veriT (if one is found) and encodes it as a directed acyclic graph with $\bot$ as the only conclusion.
3. It converts the SMT-LIB terms to typed Isabelle terms and also reverses the encoding used to convert higher-order into first-order terms.
4. It traverses the proof graph, checks that all input assertions match their Isabelle counterpart and then reconstructs the proof step by step using the kernel's primitives.

4 Tuning the Reconstruction

To improve the speed of the reconstruction method, we create small and well-defined rules for preprocessing simplifications (Sect. 4.1). Previously, veriT implicitly normalized every step; e.g., repeated literals were immediately deleted. It now produces proofs for this transformation (Sect. 4.2). Finally, the linear-arithmetic steps contain coefficients which allow Isabelle to reconstruct the step without relying on its limited arithmetic automation (Sect. 4.3). On the Isabelle side, the reconstruction module selectively decodes the first-order encoding (Sect. 4.4). To improve the performance of the reconstruction, it skips some steps (Sect. 4.5).

4.1 Preprocessing Rules

During preprocessing SMT solvers perform simplifications on the operator level which are often akin to simple calculations; e.g., $a \times 0 \times f(x)$ is replaced by 0.

To capture such simplifications, we create a list of 17 new rules: one rule per arithmetic operator, one to replace boolean operators such as XOR with their definition, and one to replace n-ary operator applications with binary applications. This is a compromise: having one rule for every possible simplification would create a longer proof. Since preprocessing uses structural recursion, the implementation simply picks the right rule in each leaf case. The example above now produces a `prod_simplify` step with the conclusion $a \times 0 \times f(x) = 0$. Previously, a single step of the `connect_equiv` rule collected all those simplifications and no list of simplifications performed by this rule existed. The reconstruction relied an experimentally created list of tactics to be fast enough.

On the Isabelle side, the reconstruction is fast, because we can direct the search instead of trying automated tactics that can also work on other parts of

the formula. For example, the simplifier handles the numeral manipulations of the `prod_simplify` rule and we restrict it to only use arithmetic lemmas.

Moreover, since we know the performed transformations, we can ignore some parts of the terms by *generalizing*, i.e., replacing them by constants [18]. Because generalized terms are smaller, the search is more directed and we are less likely to hit the search-depth limitation of Isabelle's `auto` tactic as before. Overall, the reconstruction is more robust and easier to debug.

4.2 Implicit Steps

To simplify reconstruction, we avoid any implicit normal form of conclusions. For example, a rule concluding $t \vee P$ for any formula t can be used to prove $P \vee P$. In such cases veriT automatically normalizes the conclusion $P \vee P$ to P. Without a proof of the normalization, the reconstruction has to handle such cases.

We add new proof rules for the normalization and extend veriT to use them. Instead of keeping only the normalized step, both the original and the normalized step appear in the proof. For the example above, we have the step $P \vee P$ and the normalized P. To remove a double negation $\neg\neg t$ we introduce the tautology $\neg\neg\neg t \vee t$ and resolve it with the original clause. Our changes do not affect any other part of veriT. The solver now also prunes steps concluding $\top$.

On the Isabelle side, the reconstruction becomes more regular with fewer special cases and is more reliable. The reconstruction method can directly reconstruct rules. To deal with the normalization, the reconstruction used to first generate the conclusion of the theorem and then ran the simplifier to match the normalized conclusion. This could not deal with tautologies.

We also improve the proof reconstruction of quantifier instantiation steps. One of the instantiation schemes, *conflicting instances* [8,36], only works on clausified terms. We introduce an explicit quantified-clausification rule `qnt_cnf` issued before instantiating. While this rule is not detailed, knowing when clausification is needed improves reconstruction, because it avoids clausifying unconditionally. The clausification is also shared between instantiations of the same term.

4.3 Arithmetic Reasoning

We use a proof witness to handle linear arithmetic. When the propositional model is unsatisfiable in the theory of linear real arithmetic, the solver creates `la_generic` steps. The conclusion is a tautological clause of linear inequalities and equations and the justification of the step is a list of coefficients so that the linear combination is a trivially contradictory inequality after simplification (e.g., $0 \geqslant 1$). Farkas' lemma guarantees the existence of such coefficients for reals. Most SMT solvers, including veriT, use the simplex method [21] to handle linear arithmetic. It calculates the coefficients during normal operation.

The real arithmetic solver also strengthens inequalities on integer variables before adding them to the simplex method. For example, if x is an integer the inequality $2x < 3$ becomes $x \leqslant 1$. The corresponding justification is the rational coefficient $1/2$. The reconstruction must replay this strengthening.

The complete linear arithmetic proof step $1 < x \vee 2x < 3$ looks like

```
(step t11 (cl (< 1 x) (< (* 2 x) 3))
    :rule la_generic :args (1 (div 1 2)))
```

The reconstruction of an `la_generic` step in Isabelle starts with the goal $\bigvee_i \neg c_i$ where each c_i is either an equality or an inequality. The reconstruction method first generalizes over the non-arithmetic parts. Then it transforms the lemma into the equivalent formulation $c_1 \Rightarrow \cdots \Rightarrow c_n \Rightarrow \bot$ and removes all negations (e.g., by replacing $\neg a \leqslant b$ with $b > a$).

Next, the reconstruction method multiplies the equation by the corresponding coefficient. For example, for integers, the equation $A < B$, and the coefficient p/q (with $p > 0$ and $q > 0$), it strengthens the equation and multiplies by p to get

$$p \times (A \mathit{div}\, q) + p \times (\text{if } B \mathit{mod}\, q = 0 \text{ then } 1 \text{ else } 0) \leqslant p \times (B \mathit{div}\, q).$$

The if-then-else term (if $B \mathit{mod}\, q = 0$ then 1 else 0) corresponds to the strengthening. If $B \mathit{mod}\, q = 0$, the result is an equation of the form $A' + 1 \leqslant B'$, i.e., $A' < B'$. No strengthening is required for the corresponding theorem over reals.

Finally, we can combine all the equations by summing them while being careful with the equalities that can appear. We simplify the resulting (in)equality using Isabelle's simplifier to derive $\bot$.

To replay linear arithmetic steps, Isabelle can also use the tactic linarith as used for Z3 proofs. It searches the coefficients necessary to verify the lemma. The reconstruction used it previously [25], but the tactic can only find integer coefficients and fails if strengthening is required. Now the rule is a mechanically checkable certificate.

4.4 Selective Decoding of the First-order Encoding

Next, we consider an example of a rule that shows the interplay of the higher-order encoding and the reconstruction. To express function application, the encoding introduces the first-order function app and constants for encoded functions. The proof rule `eq_congruent` expresses congruence on a first-order function: $(t_1 \neq u_1) \vee \ldots \vee (t_n \neq u_n) \vee f(t_1, \ldots, t_n) = f(u_1, \ldots, u_n)$. With the encoding it can conclude $f \neq f' \vee x \neq x' \vee \mathrm{app}(f, x) = \mathrm{app}(f', x')$. If the reconstruction unfolds the entire encoding, it builds the term $f \neq f' \vee x \neq x' \vee fx = f'x'$. It then identifies the functions and the function arguments and uses rewriting to prove that if $f = f'$ and $x = x'$, then $fx = f'x'$.

However, Isabelle β-reduces all terms implicitly, changing the term structure. Assume $f := \lambda x.\ x = a$ and $f' := \lambda x.\ a = x$. After unfolding all constructs that encode higher-order terms and after β-reduction, we get $(\lambda x.\ x = a) \neq (\lambda x.\ a = x') \vee (x \neq x') \vee (x = a) = (a = y')$. The reconstruction method cannot identify the functions and function arguments anymore.

Instead, the reconstruction method does not unfold the encoding including app. This eliminates the need for a special case to detect lambda functions. Such a case was used in the previous prototype, but the code was very involved and hard to test (such steps are rarely used).

4.5 Skipping Steps

The increased number of steps in the fine-grained proof format slows down reconstruction. For example, consider skolemization from $\exists x.\ P\ x$. The proof from Z3 uses *one* step. veriT uses *eight* steps—first renaming it to $(\exists x.\ P\ x) = (\exists v.\ P\ v)$ (with a subproof of at least 2 steps), then concluding the renaming to get $(\exists v.\ P\ v)$ (two steps), then $(\exists v.\ P\ v) = P\ (\epsilon v.\ P\ v)$ (with a subproof of at least 2 steps), and finally $P\ (\epsilon v.\ P\ v)$ (two steps).

To reduce the number of steps, our reconstruction skips two kinds of steps. First, it replaces every usage of the `or` rule by its only premise. Second, it skips the renaming of bound variables. The proof format treats $\forall x.\ P\ x$ and $\forall y.\ P\ y$ as two different terms and requires a detailed proof of the conversion. Isabelle, however, uses De Bruijn indices and variable names are irrelevant. Hence, we replace steps of the form $(\forall x.\ P\ x) \Leftrightarrow (\forall y.\ P\ y)$ by a single application of reflexivity. Since veriT canonizes all variable names, this eliminates many steps.

We can also simplify the idiom "`equiv_pos2`; `th_resolution`". veriT generates it for each skolemization and variable renaming. Step skipping replaces it by a single step which we replay using a specialized theorem.

On proof with quantifiers, step skipping can remove more than half of the steps—only four steps remain in the skolemization example above (where two are simply reflexivity). However, with step skipping the smt method is not an independent checker that confirms the validity of every single step in a proof.

5 Evaluation

During development we routinely tested our proof reconstruction to find bugs. As a side effect, we produced SMT-LIB files corresponding to the calls. We measure the performance of veriT with various options on them and select five different strategies (Sect. 5.1). We also evaluate the repartition of the tactics used by Sledgehammer for preplay (Sect. 5.2), and the impact of the rules (Sect. 5.3).

We performed the strategy selection on a computer with two Intel Xeon Gold 6130 CPUs (32 cores, 64 threads) and 192 GiB of RAM. We performed Isabelle experiments with Isabelle version 2021 on a computer with two AMD EPYC 7702 CPUs (128 cores, 256 threads) and 2 TiB of RAM.

5.1 Strategies

veriT exposes a wide range of options to fine-tune the proof search. In order to find good combinations of options (*strategies*), we generate problems with Sledgehammer and use them to fine-tune veriT's search behavior. Generating problems also makes it possible to test and debug our reconstruction.

We test the reconstruction by using Isabelle's *Mirabelle* tool. It reads theories and automatically runs Sledgehammer [14] on all proof steps. Sledgehammer calls various automatic provers (here the SMT solvers CVC4, veriT, and Z3 and the superposition prover E [38]) to *filter* facts and chooses the fastest tactic that can prove the goal. The tactic smt is used as a last resort.

Table 1. Options corresponding to the different veriT strategies

Name	Options
default	(no option)
del_insts	--index-sorts --index-fresh-sorts --ccfv-breadth --inst-deletion --index-SAT-triggers --inst-deletion-loops --inst-deletion-track-var
ccfv_SIG	--triggers-new --index-SIG --triggers-sel-rm-specific
ccfv_insts	--triggers-new --index-sorts --index-fresh-sorts --triggers-sel-rm-specific --triggers-restrict-combine --inst-deletion-loops --index-SAT-triggers --inst-deletion-track-vars --ccfv-index=100000 --ccfv-index-full=1000 --inst-sorts-threshold=100000 --ematch-exp=10000000 --inst-deletion
best	--triggers-new --index-sorts --index-fresh-sorts --triggers-sel-rm-specific

To generate problems for tuning veriT, we use the theories from HOL-Library (an extended standard library containing various developments) and from the formalizations of Green's theorem [2, 3], the Prime Number Theorem [23], and the KBO ordering [13]. We call Mirabelle with only veriT as a fact filter. This produces SMT files for representative problems Isabelle users want to solve and a series of calls to v-smt. For failing v-smt calls three cases are possible: veriT does not find a proof, reconstruction times out, or reconstruction fails with an error. We solved all reconstruction failures in the test theories.

To find good strategies, we determine which problems are solved by several combination of options within a two second timeout. We then choose the strategy which solves the most benchmarks and three strategies which together solve the most benchmarks. For comparison, we also keep the default strategy.

The strategies are shown in Table 1 and mostly differ in the instantiation schemes. The strategy *del_insts* uses instance deletion [6] and uses a breadth-first algorithm to find conflicting instances. All other strategies rely on extended trigger inference [29]. The strategy *ccfv_SIG* uses a different indexing method for instantiation. It also restricts enumerative instantiation [35], because the options --index-sorts and --index-fresh-sorts are not used. The strategy *ccfv_insts* increases some thresholds. Finally, the strategy *best* uses a subset of the options used by the other strategies. Sledgehammer uses *best* for fact filtering.

We have also considered using a scheduler in Isabelle as used in the SMT competition. The advantage is that we do not need to select the strategy on the Isabelle side. However, it would make v-smt unreliable. A problem solved by only one strategy just before the end of its time slice can become unprovable on slower hardware. Issues with z-smt timeouts have been reported on the Isabelle mailing list, e.g., due to an antivirus delaying the startup [27].

5.2 Improvements of Sledgehammer Results

To measure the performance of the v-smt tactic, we ran Mirabelle on the full HOL-Library, the theory Prime Distribution Elementary (PDE) [22], an executable resolution prover (RP) [37], and the Simplex algorithm [30]. We extended Sledgehammer's proof preplay to try all veriT strategies and added instrumentation for

Table 2. Outcome of Sledgehammer calls showing the total success rate (SR, higher is better) of one-liner proof preplay, the number of suggested v-smt (OL_v) and z-smt (OL_z) one-liners, and the number of preplay failures (PF, lower is better), in percentages of the unique goals.

	HOL-Library (13 562 goals)				PNT (1 715 goals)				RP (1 658 goals)				Simplex (1 982 goals)			
	SR	OL_v	OL_z	PF	SR	OL_v	OL_z	PF	SR	OL_v	OL_z	PF	SR	OL_v	OL_z	PF
Fact-filter prover: CVC4																
z-smt	54.5		2.7	1.5	33.1		3.7	0.8	64.8		1.3	0.8	51.6		1.6	0.9
both	55.5	2.5	1.1	0.5	33.6	3.6	0.6	0.3	65.3	1.4	0.4	0.3	52.1	1.1	1.0	0.4
Fact-filter prover: E																
z-smt	55.5		1.1	1.7	36.0		0.3	1.7	61.7		0.7	1.2	49.8		1.4	0.7
both	56.0	0.8	0.7	1.3	36.4	0.6	0.1	1.3	62.1	0.9	0.2	0.8	49.9	0.3	1.3	0.5
Fact-filter prover: veriT																
z-smt	48.5		1.7	1.2	26.1		1.5	0.5	58.2		0.9	0.7	46.7		0.9	1.0
both	49.4	1.6	0.9	0.4	26.5	1.4	0.4	0.2	58.6	1.1	0.3	0.2	47.4	1.0	0.6	0.3
Fact-filter prover: Z3																
z-smt	50.8		2.5	0.8	27.9		2.7	0.4	60.4		0.8	0.7	48.3		0.9	0.3
both	51.3	1.9	1.1	0.3	28.2	2.5	0.5	0.1	60.9	1.1	0.1	0.2	48.4	0.4	0.6	0.2

the time of all tried tactics. Sledgehammer and automatic provers are mostly nondeterministic programs. To reduce the variance between the different Mirabelle runs, we use the deterministic MePo fact filter [33] instead of the better performing MaSh [28] that uses machine learning (and depends on previous runs) and underuse the hardware to minimize contention. We use the default timeouts of 30 seconds for the fact filtering and one second for the proof preplay. This is similar to the Judgment Day experiments [17]. The raw results are available [1].

Success Rate. Users are not interested in which tactics are used to prove a goal, but in how often Sledgehammer succeeds. There are three possible outcomes: (i) a successfully preplayed proof, (ii) a proof hint that failed to be preplayed (usually because of a timeout), or (iii) no proof. We define the success rate as the proportion of outcome (i) over the total number of Sledgehammer calls.

Table 2 gathers the results of running Sledgehammer on all unique goals and analyzing its outcome using different preplay configurations where only z-smt (the baseline) or both v-smt and z-smt are enabled. Any useful preplay tactic should increase the success rate (SR) by preplaying new proof hints provided by the fact-filter prover, reducing the preplay failure rate (PF).

Let us consider, e.g., the results when using CVC4 as fact-filter prover. The success rate of the baseline on the HOL-Library is 54.5% and its preplay failure rate is 1.5%. This means that CVC4 found a proof for $54.5\% + 1.5\% = 56\%$ of the goals, but that Isabelle's proof methods failed to preplay many of them. In such

cases, Sledgehammer gives a proof hint to the user, which has to manually find a functioning proof. By enabling v-smt, the failure rate decreases by two thirds, from 1.5% to 0.5%, which directly increases the success rate by 1 percentage point: new cases where the burden of the proof is moved from the user to the proof assistant. The failure rate is reduced in similar proportions for PNT (63%), RP (63%), and Simplex (56%). For these formalizations, this improvement translates to a smaller increase of the success rate, because the baseline failure rate was smaller to begin with. This confirms that the instantiation technique *conflicting instances* [8, 36] is important for CVC4.

When using veriT or Z3 as fact-filter prover, a failure rate of zero could be expected, since the same SMT solvers are used for both fact filtering and preplaying. The observed failure rate can partly be explained by the much smaller timeout for preplay (1 second) than for fact filtering (30 seconds).

Overall, these results show that our proof reconstruction enables Sledgehammer to successfully preplay more proofs. With v-smt enabled, the weighted average failure rate decreases as follows: for CVC4, from 1.3% to 0.4%; for E, from 1.5% to 1.2%; for veriT, from 1.0% to 0.3%; and for Z3, from 0.7% to 0.3%. For the user, this means that the availability of v-smt as a proof preplay tactic increases the number of goals that can be fully automatically proved.

Saved time. Table 3 shows a different view on the same results. Instead of the raw success rate, it shows the time that is spent reconstructing proofs. Using the baseline configuration, preplaying all formalizations takes a total of $250.1 + 33.4 + 37.2 + 42.8 = 363.5$ seconds. When enabling v-smt, some calls to z-smt are replaced by faster v-smt calls and the reconstruction time decreases by 13% to $212.6 + 28.4 + 34.4 + 41.6 = 317$ seconds. Note that the per-formalization improvement varies considerably: 15% for HOL-Library, 15% for PNT, 7.5% for RP, and 4.0% for Simplex.

For the user, this means that enabling v-smt as a proof preplay tactic may significantly reduce the verification time of their formalizations.

Impact of the Strategies. We have also studied what happens if we remove a single veriT strategy from Sledgehammer (Table 4). The most important one is *best*, as it solves the highest number of problems. On the contrary, *default* is nearly entirely covered by the other strategies. *ccfv_SIG* and *del_insts* have a similar number where they are faster than Z3, but the latter has more unique goals and therefore, saves more time. Each strategy has some uniquely solved problems that cannot be reconstructed using any other. The results are similar for the other theories used in Table 3.

5.3 Speed of Reconstruction

To better understand what the key rules of our reconstruction are, we recorded the time used to reconstruct each rule and the time required by the solver over all calls attempted by Sledgehammer including the ones not selected. The reconstruction ratio (reconstruction over search time) shows how much slower reconstructing

Table 3. Preplayed proofs (Pr.) and their execution time (s) when using CVC4 as fact-filter prover. Shared proofs are found with and without v-smt and new proofs are found only with v-smt. The proofs and their associated timings are categorized in one-liners using v-smt (OL_v), z-smt (OL_z), or any other Isabelle proof methods (OL_o).

		Total	Shared proofs						New proofs	
			Total	=	OL_v	+	OL_z	+	OL_o	OL_v
		Pr.	Time	=	Time (Pr.)	+	Time (Pr.)	+	Time (Pr.)	Time (Pr.)
HOL-	z-smt	7 409	250.1	=			85.0 (362)	+	165.1 (7 047)	
Library	both	7 545	212.6	=	27.9 (211)	+	19.6 (152)	+	165.1 (7 047)	34.7 (135)
PNT	z-smt	569	33.4	=			14.8 (64)	+	18.5 (505)	
	both	577	28.4	=	7.7 (54)	+	2.1 (10)	+	18.5 (505)	3.4 (8)
RP	z-smt	1 077	37.2	=			8.7 (22)	+	28.5 (1 055)	
	both	1 085	34.4	=	4.5 (16)	+	1.4 (6)	+	28.5 (1 055)	2.2 (8)
Simplex	z-smt	1 024	42.8	=			6.7 (32)	+	36.0 (992)	
	both	1 033	41.6	=	2.4 (13)	+	3.2 (19)	+	36.0 (992)	3.0 (9)

Table 4. Reconstruction time and number of solved goals when removing a single strategy (HOL-Library results only), using CVC4 as fact filter.

	Shared proofs				New proofs	
	OL_v		OL_z		OL_v	
	Time	Proofs	Time	Proofs	Time	Proofs
No *best*	16.5	119	50.6	244	25.9	94
No *ccfv_SIG*	27.0	198	22.6	164	33.5	123
No *ccfv_threshold*	28.3	211	19.6	152	33.9	130
No *del_insts*	27.4	201	21.8	162	32.9	124
No *default*	27.9	207	20.1	156	33.8	134
Baseline	27.9	211	19.6	152	34.7	135

compared to finding a proof is. For the 25% of the proofs, Z3's concise format is better and the reconstruction is faster than proof finding (first quartile: 0.9 for v-smt vs. 0.1 for z-smt). The 99th percentile of the proofs (18.6 vs. 27.2) shows that veriT's detailed proof format reduces the number of slow proofs. The reconstruction is slower than finding proofs on average for both solvers.

Fig. 1 shows the distribution of the time spent on some rules. We remove the slowest and fastest 5% of the applications, because garbage collection can trigger at any moment and even trivial rules can be slow. Fig. 2 gives the sum of all reconstruction times over all proofs. We call `parsing` the time required to parse and convert the veriT proof into Isabelle terms.

Overall, there are two kinds of rules: (1) direct application of a sequence of theorems—e.g., `equiv_pos2` corresponds to the theorem $\neg(a \Leftrightarrow b) \vee \neg a \vee b$—and (2) calls to full-blown tactics—like `qnt_cnf` (Sect. 4.2).

First, direct application of theorems are usually fast, but they occur so often that the cumulative time is substantial. For example, `cong` only needs to unfold

assumptions and apply reflexivity and symmetry of equality. However, it appears so often and sometimes on large terms, that it is an important rule.

Second, rules which require full-blown tactics are the slowest rules. For `qnt_cnf` (CNF under quantifiers, see Sect. 4.2), we have not written a specialized tactic, but rely on Isabelle's tableau-based `blast` tactic. This rule is rather slow, but is rarely used. It is similar to the rule `la_generic`: it is slow on average, but searching the coefficients takes even more time.

We can also see that the time required to check the simplification steps that were formerly combined into the `connect_equiv` rule is not significant anymore.

We have performed the same experiments with the reconstruction of the SMT solver Z3. In contrast to veriT, we do not have the amount of time required for parsing. The results are shown in Figs. 3 and 4. The rule distribution is very different. The `nnf-neg` and `nnf-pos` rules are the slowest rules and take a huge amount of time in the worst case. However, the coarser quantifier instantiation step is on average *faster* than the one produced by veriT. We suspect that reconstruction is faster because the rule, which is only an implication without choice terms, is easier to check (no equality reordering).

6 Related Work

The SMT solvers CVC4 [10], Z3 [34], and veriT [19] produce proofs. CVC4 does not record quantifier reasoning in the proof, and Z3 uses some macro rules. Proofs from SMT solvers have also been used to find unsatisfiability cores [20], and interpolants [32]. They are also useful to debug the solver itself, since unsound steps often point to the origin of bugs. Our work also relates to systems like Dedukti [5] that focuses on translating proof steps, not on replaying them.

Proof reconstruction has been implemented in various systems, including CVC4 proofs in HOL Light [31], Z3 in HOL4 and Isabelle/HOL [18], and veriT [4] and CVC4 [24] in Coq. Only veriT produces detailed proofs for preprocessing and skolemization. SMTCoq [4, 24] currently supports veriT's version 1 of the proof output which has different rules, does not support detailed skolemization rules, and is implemented in the 2016 version of veriT, which has worse performance. SMTCoq also supports bit vectors and arrays.

The reconstruction of Z3 proofs in HOL4 and Isabelle/HOL is one of the most advanced and well tested. It is regularly used by Isabelle users. The Z3 proof reconstruction succeeds in more than 90% of Sledgehammer benchmarks [14, Section 9] and is efficient (an older version of Z3 was used). Performance numbers are reported [16,18] not only for problems generated by proof assistants (including Isabelle), but also for preexisting SMT-LIB files from the SMT-LIB library.

The performance study by Böhme [16, Sect. 3.4] uses version 2.15 of Z3, whereas we use version 4.4.0 which currently ships with Isabelle. Since version 2.15, the proof format changed slightly (e.g., `th-lemma-arith` was introduced), fulfilling some of the wishes expressed by Böhme and Weber [18] to simplify reconstruction. Surprisingly, the `nnf` rules do not appear among the five rules that used the most runtime. Instead, the `th-lemma` and `rewrite` rules were the

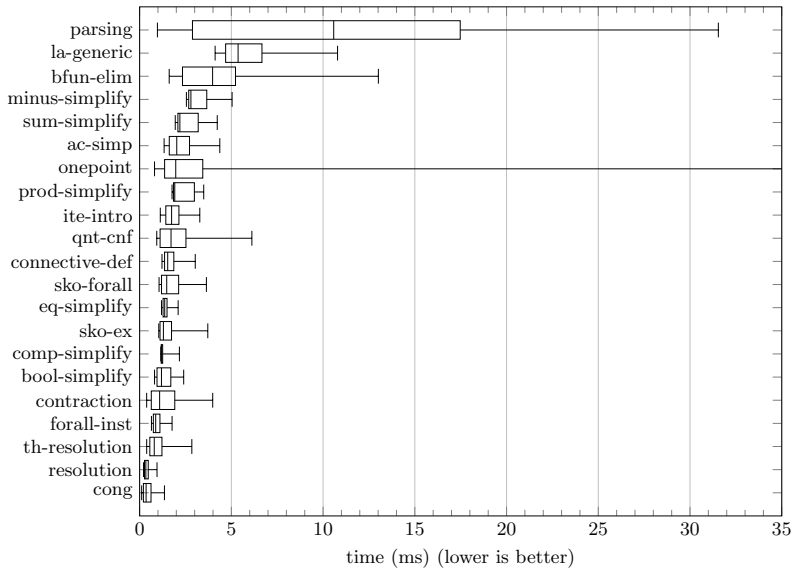

Fig. 1. Timing, sorted by the median, of a subset of veriT's rules. From left to right, the lower whisker marks the 5th percentile, the lower box line the first quartile, the middle of the box the median, the upper box line the third quartile, and the upper whisker the 95th percentile.

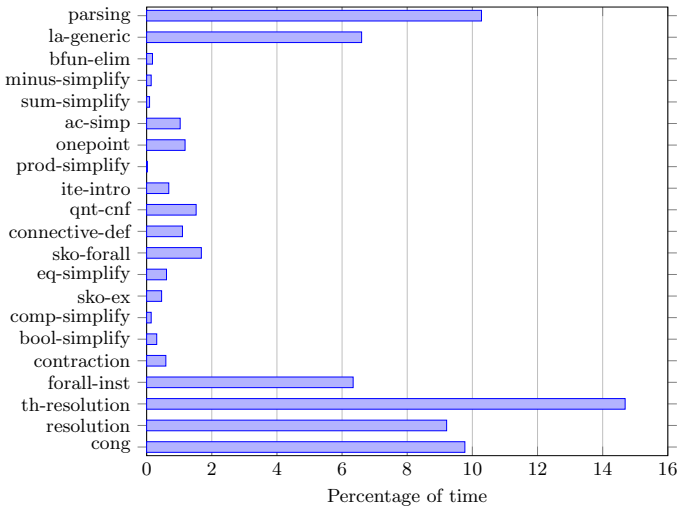

Fig. 2. Total percentage spent on each rule for the SMT solver veriT in the same order as Fig. 1. This graph maps the rules already shown in Fig. 1 to the total amount of time. The slowest rules are `th_resolution` (14.7%), parsing (10.3%), and `cong` (9.77%).

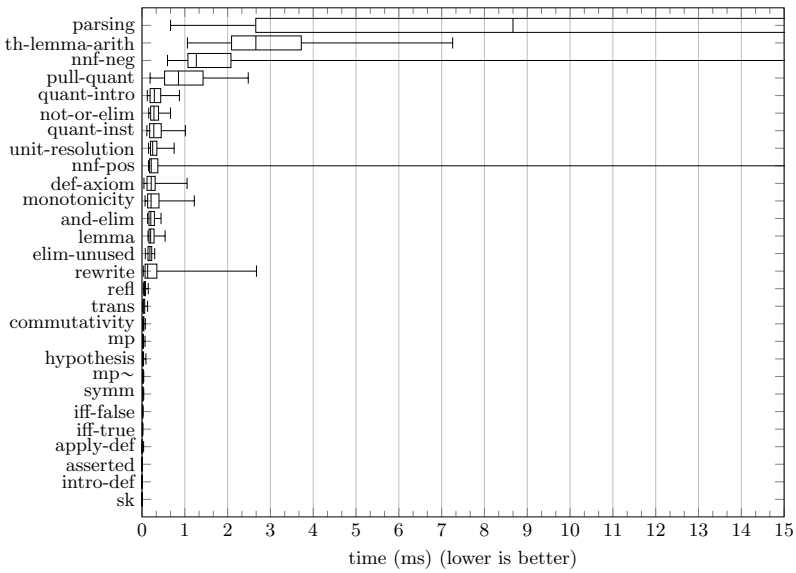

Fig. 3. Timing of some of Z3's rules sorted by median. From left to right, the lower whisker marks the 5th percentile, the lower box line the first quartile, the middle of the box the median, the upper box line the third quartile, and the upper whisker the 95th percentile. nnf-neg's 95th percentile is 87 ms, nnf-pos's is 33 ms, and parsing's is 25 ms.

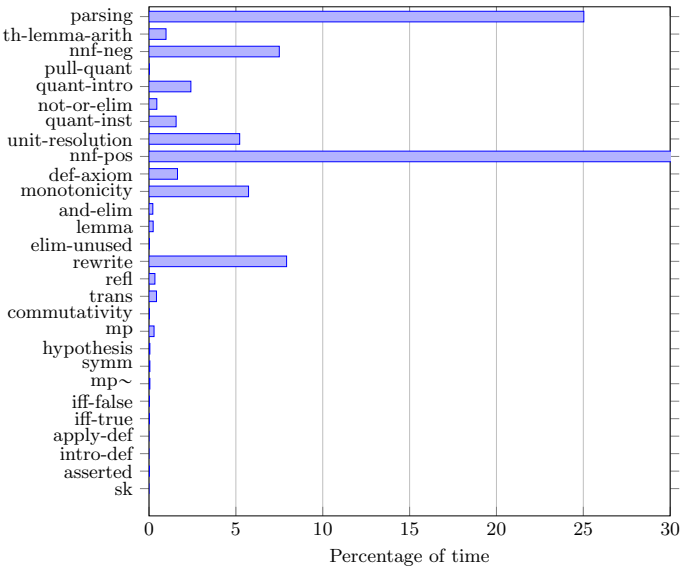

Fig. 4. Total amount of time per rule for the SMT solver Z3. nnf-neg takes 39% of the reconstruction time.

slowest. Similarly to veriT, the `cong` rule was among the most used (without accounting for the most time), but it does not appear in our Z3 tests.

CVC4 follows a different philosophy compared to veriT and Z3: it produces proofs in a logical framework with side conditions [39]. The output can contain programs to check certain rules. The proof format is flexible in some aspects and restrictive in others. Currently CVC4 does not generate proofs for quantifiers.

7 Conclusion

We presented an efficient reconstruction of proofs generated by a modern SMT solver in an interactive theorem prover. Our improvements address reconstruction challenges for proof steps of typical inferences performed by SMT solvers.

By studying the time required to replay each rule, we were able to compare the reconstruction for two different proof formats with different design directions. The very detailed proof format of veriT makes the reconstruction easier to implement and allows for more specialization of the tactics. On slow proofs, the ratio of time to reconstruct and time to find a proof is better for our more detailed format. Integrating our reconstruction in Isabelle halves the number of failures from Sledgehammer and nicely completes the existing reconstruction method with Z3.

Our work is integrated into Isabelle version 2021. Sledgehammer suggests the veriT-based reconstruction if it is the fastest tactic that finds the proof; so users profit without action required on their side. We plan to improve the reconstruction of the slowest rules and remove inconsistencies in the proof format. The developers of the SMT solver CVC4 are currently rewriting the proof generation and plan to support a similar proof format. We hope to be able to reuse the current reconstruction code by only adding support for CVC4-specific rules. Generating and reconstructing proofs from the veriT version with higher-order logic [9] could also improve the usefulness of veriT on Isabelle problems. The current proof rules [40] should accommodate the more expressive logic.

Acknowledgment We would like to thank Haniel Barbosa for his support with the implementation in veriT. We also thank Haniel Barbosa, Jasmin Blanchette, Pascal Fontaine, Daniela Kaufmann, Petar Vukmirović, and the anonymous reviewers for many fruitful discussions and suggesting many textual improvements. The first and third authors have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreements No. 713999, Matryoshka, and No. 830927, Concordia). The second author is supported by the LIT AI Lab funded by the State of Upper Austria. The training presented in this paper was carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

References

1. Reliable Reconstruction of Fine-Grained Proofs in a Proof Assistant. Zenodo (Apr 2021). https://doi.org/10.5281/zenodo.4727349
2. Abdulaziz, M., Paulson, L.C.: An Isabelle/HOL formalisation of Green's theorem. Archive of Formal Proofs (Jan 2018), `https://isa-afp.org/entries/Green.html`, formal proof development
3. Abdulaziz, M., Paulson, L.C.: An Isabelle/HOL formalisation of Green's theorem. Journal of Automated Reasoning **63**(3), 763–786 (Nov 2019). https://doi.org/10.1007/s10817-018-9495-z
4. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A modular integration of SAT/SMT solvers to Coq through proof witnesses. In: Jouannaud, J.P., Shao, Z. (eds.) CPP 2011. LNCS, vol. 7086, pp. 135–150. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-25379-9_12
5. Assaf, A., Burel, G., Cauderlier, R., Delahaye, D., Dowek, G., Dubois, C., Gilbert, F., Halmagrand, P., Hermant, O., Saillard, R.: Expressing theories in the $\lambda\pi$-calculus modulo theory and in the Dedukti system. In: TYPES: Types for Proofs and Programs. Novi SAd, Serbia (May 2016)
6. Barbosa, H.: Efficient instantiation techniques in SMT (work in progress). vol. 1635, pp. 1–10. CEUR-WS.org (Jul 2016), `http://ceur-ws.org/Vol-1635/#paper-01`
7. Barbosa, H., Blanchette, J.C., Fleury, M., Fontaine, P.: Scalable fine-grained proofs for formula processing. Journal of Automated Reasoning (Jan 2019). https://doi.org/10.1007/s10817-018-09502-y
8. Barbosa, H., Fontaine, P., Reynolds, A.: Congruence closure with free variables. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 214–230. Springer Berlin Heidelberg (2017). https://doi.org/10.1007/978-3-662-54580-5_13
9. Barbosa, H., Reynolds, A., Ouraoui, D.E., Tinelli, C., Barrett, C.W.: Extending SMT solvers to higher-order logic. In: Fontaine, P. (ed.) CADE 27. LNCS, vol. 11716, pp. 35–54. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-29436-6_3
10. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_14
11. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Tech. rep., Department of Computer Science, The University of Iowa (2017), available at `www.SMT-LIB.org`
12. Barrett, C.W., Tinelli, C.: Satisfiability modulo theories. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 305–343. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_11
13. Becker, H., Blanchette, J.C., Waldmann, U., Wand, D.: Formalization of Knuth–Bendix orders for lambda-free higher-order terms. Archive of Formal Proofs (Nov 2016), `https://isa-afp.org/entries/Lambda_Free_KBOs.html`, formal proof development
14. Blanchette, J.C., Böhme, S., Fleury, M., Smolka, S.J., Steckermeier, A.: Semi-intelligible Isar proofs from machine-generated proofs. Journal of Automated Reasoning **56**(2), 155–200 (2016). https://doi.org/10.1007/s10817-015-9335-3

15. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending Sledgehammer with smt solvers. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 23. LNCS, vol. 6803, pp. 116–130. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-22438-6_11
16. Böhme, S.: Proving Theorems of Higher-Order Logic with SMT Solvers. Ph.D. thesis, Technische Universität München (2012), http://mediatum.ub.tum.de/node?id=1084525
17. Böhme, S., Nipkow, T.: Sledgehammer: Judgement day. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. pp. 107–121. Springer Berlin Heidelberg (2010). https://doi.org/10.1007/978-3-642-14203-1_9
18. Böhme, S., Weber, T.: Fast LCF-style proof reconstruction for Z3. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 179–194. Springer Berlin Heidelberg (2010). https://doi.org/10.1007/978-3-642-14052-5_14
19. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: veriT: An open, trustable and efficient SMT-solver. In: Schmidt, R.A. (ed.) CADE 22. LNCS, vol. 5663, pp. 151–156. Springer Berlin Heidelberg (2009). https://doi.org/10.1007/978-3-642-02959-2_12
20. Déharbe, D., Fontaine, P., Guyot, Y., Voisin, L.: SMT solvers for Rodin. In: Derrick, J., Fitzgerald, J.A., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E. (eds.) ABZ 2012. LNCS, vol. 7316, pp. 194–207. Springer Berlin Heidelberg (Jun 2012). https://doi.org/10.1007/978-3-642-30885-7_14
21. Dutertre, B., de Moura, L.: Integrating simplex with DPLL(T). Tech. rep., SRI International (May 2006), http://www.csl.sri.com/users/bruno/publis/sri-csl-06-01.pdf
22. Eberl, M.: Elementary facts about the distribution of primes. Archive of Formal Proofs (Feb 2019), https://isa-afp.org/entries/Prime_Distribution_Elementary.html, formal proof development
23. Eberl, M., Paulson, L.C.: The prime number theorem. Archive of Formal Proofs (Sep 2018), https://isa-afp.org/entries/Prime_Number_Theorem.html, formal proof development
24. Ekici, B., Mebsout, A., Tinelli, C., Keller, C., Katz, G., Reynolds, A., Barrett, C.W.: SMTCoq: A plug-in for integrating SMT solvers into Coq. In: Majumdar, R., Kuncak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 126–133. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-63390-9_7
25. Fleury, M., Schurr, H.: Reconstructing veriT proofs in Isabelle/HOL. In: Reis, G., Barbosa, H. (eds.) PxTP 2019. EPTCS, vol. 301, pp. 36–50 (2019). https://doi.org/10.4204/EPTCS.301.6
26. Gordon, M.J.C., Milner, R., Wadsworth, C.P.: Edinburgh LCF: A Mechanised Logic of Computation, LNCS, vol. 78. Springer Berlin Heidelberg (1979). https://doi.org/10.1007/3-540-09724-4
27. Immler, F.: Re: [isabelle] Isabelle2019-RC2 sporadic smt failures. Email (May 2019), https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2019-May/msg00130.html
28. Kühlwein, D., Blanchette, J.C., Kaliszyk, C., Urban, J.: Mash: Machine learning for Sledgehammer. In: ITP. LNCS, vol. 7998, pp. 35–50. Springer (2013)
29. Leino, K.R.M., Pit-Claudel, C.: Trigger selection strategies to stabilize program verifiers. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 361–381. Springer International Publishing (2016). https://doi.org/10.1007/978-3-319-41528-4_20

30. Marić, F., Spasić, M., Thiemann, R.: An incremental simplex algorithm with unsatisfiable core generation. Archive of Formal Proofs (Aug 2018), https://isa-afp.org/entries/Simplex.html, formal proof development
31. McLaughlin, S., Barrett, C., Ge, Y.: Cooperating theorem provers: A case study combining HOL-Light and CVC Lite. Electronic Notes in Theoretical Computer Science **144**(2), 43–51 (2006). https://doi.org/10.1016/j.entcs.2005.12.005
32. McMillan, K.L.: Interpolants from Z3 proofs. In: FMCAD 2011. pp. 19–27. FMCAD Inc, Austin, Texas (2011)
33. Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. J. Appl. Log. **7**(1), 41–57 (2009)
34. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer Berlin Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
35. Reynolds, A., Barbosa, H., Fontaine, P.: Revisiting enumerative instantiation. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10806, pp. 112–131. Springer International Publishing (2018). https://doi.org/10.1007/978-3-319-89963-3_7
36. Reynolds, A., Tinelli, C., de Moura, L.: Finding conflicting instances of quantified formulas in SMT. In: FMCAD 2014. pp. 195–202. IEEE (2014). https://doi.org/10.1109/FMCAD.2014.6987613
37. Schlichtkrull, A., Blanchette, J.C., Traytel, D., Waldmann, U.: Formalization of Bachmair and Ganzinger's ordered resolution prover. Archive of Formal Proofs (Jan 2018), https://isa-afp.org/entries/Ordered_Resolution_Prover.html, formal proof development
38. Schulz, S.: E - a brainiac theorem prover. AI Communications **15**(2-3), 111–126 (2002), http://content.iospress.com/articles/ai-communications/aic260
39. Stump, A., Oe, D., Reynolds, A., Hadarean, L., Tinelli, C.: SMT proof checking using a logical framework. Formal Methods in System Design **42**(1), 91–118 (Feb 2013). https://doi.org/10.1007/s10703-012-0163-3
40. The veriT Team and Contributors: Proofonomicon: A reference of the veriT proof format. Software Documentation (2021), https://www.verit-solver.org/documentation/proofonomicon.pdf, last Accessed: April 2021