Quantifier Inference Rules for SMT proofs

David Déharbe, Pascal Fontaine, Bruno Woltzenlogel Paleo

Quantifier Inference Rules for SMT proofs*

David Deharbe
Universidade Federal do Rio Grande do Norte, Natal, Brazil
deharbe@gmail.com

Pascal Fontaine
University of Nancy and INRIA, Nancy, France
Pascal.Fontaine@loria.fr

Bruno Woltzenlogel Paleo
Technische Universität Wien, Vienna, Austria
bruno@logic.at

Abstract

This paper discusses advantages and disadvantages of some possible alternatives for inference rules that handle quantifiers in the proof format of the SMT-solver veriT. The quantifier-handling modules in veriT being fairly standard, we hope this will motivate the discussion among the PxTP audience around proof production for quantifier handling. This could generate ideas to help us improve our proof production module, and also benefit the SMT community.

1 Introduction

In the typical architecture of an SMT-solver, the core automated reasoner is a propositional SAT-solver, and quantifiers are handled by independent modules [8]. In veriT [4], essentially universal quantifiers are handled by an instantiation module, which heuristically chooses terms to instantiate such quantified variables. The instantiation module is called on-demand as rarely as possible (to reduce the number of generated instances) and only on essentially universally quantified subformulas. Essentially existential quantifiers, on the other hand, are handled by a skolemization module that is called only in a pre-processing phase and replaces all the essentially existentially quantified variables by skolem terms.

Currently, these modules are not proof-producing: if the input problem contains quantifiers that require skolemization, the proof produced by veriT will take as starting point the skolemized formula. If the instantiation module is called, generated instances will be used to deduce unsatisfiability, and the proof produced by veriT will contain holes. This paper discusses advantages and disadvantages of possible inference rules to handle quantifiers in the proof format of the SMT-solver veriT. We believe veriT's instantiation module behaves mostly like those in other solvers that handle quantifiers, e.g. CVC3 [2] or Z3 [5]. We thus believe that the following discussion is relevant in the larger context of SMT solving for quantified formulas.

We aim at developing inference rules for skolemization and instantiation that take into account the following requirements:

- Proof size: the proofs produced by the skolemization and by the instantiation modules should be as short as possible, relative to the size of the formula that needs to be skolemized or instantiated.
- Faithfulness to the inner workings of the quantification modules: the proposed inference rules should reflect what actually happens inside the solver, so that they can also be used for precisely tracing executions; from a tool engineering perspective, this is important for debugging, profiling and maintainability.
- Ease of programming: our solver has a generic framework for proof production; it is desirable that the new inference rules comply with this framework.

Pascal Fontaine, Aaron Stump (eds.); PxTP 2011, pp. 33-39
*This work was supported by the ANR DeCert and the SMT-SAVeS Projects

- Compatibility with the proof format: veriT's input format follows the SMT-LIB standard; its output format also obeys the proposed proof format [3]. Hence the inference rules should be expressible in this format.
- Fine-grainedness and simplicity: the inference rules should describe instantiation and skolemization in steps that are as small and simple as possible.
- Generality: inference rules having a broader range of applicability should be preferred to over-specialized inference rules that can be used only for specific purposes.
- Elegance: the proposed inference rules should fit into the style of rules already existing in the solver.
- Complexity of proof-checking: it should be possible for an external proof checker to efficiently check instances of the proposed inference rules.
- User-friendliness: the proposed inference rules should be suitable for the users of the solver and their applications.

In this paper we discuss some alternative inference rules, focusing on the more objective and more easily measurable criteria mentioned above (e.g. proof size, fine-grainedness and complexity of proof-checking). And we leave the more subjective criteria for future work.

2 The Proof Format

veriT's proof format follows a proposed format [3] in the philosophy of the SMT-LIB standard [1]. Its grammar is partially shown below. Clauses are sets of arbitrary formulas (not only literals), and inference rules have an arbitrary number of clauses as premises and a single clause as conclusion. Optionally, an inference rule may also take terms and attributes as arguments.

```
⟨gen_clause⟩ ::= ⟨clause_id⟩
               | (⟨rule_id⟩
                    (:clauses (⟨gen_clause⟩*) | :all-clauses)?
                    (:terms (⟨term⟩*))?
                    ⟨attribute⟩*
                    (:conclusion ⟨clause⟩)?)
               | (subproof ⟨proofstep⟩* (:conclusion ⟨clause⟩)? )
```

This document describes inference rules abstractly, using a proof-theoretical notation that is independent from any concrete proof format. The translation from this notation to the proof format is easy. An inference with the form

$$\frac{\Gamma_1 \quad \ldots \quad \Gamma_n}{\Gamma} \; \mathit{rule\_id}(\mathit{term}^*; \mathit{attribute}^*)$$

becomes

```
(rule_id
    (:clauses (η(Γ1)...η(Γn)))?
    (:terms (⟨term⟩*))?
    ⟨attribute⟩*
    (:conclusion Γ)?)
```

where $\eta(\Gamma_i)$ is either the clause id of Γ_i or an inference rule instance that derives Γ_i or a subproof that derives Γ_i.

3 Rules for Instantiation of Essentially Universal Quantifiers

Essentially universal quantifiers are universal quantifiers that occur with positive polarity or existential quantifiers that occur with negative polarity. Essentially universally quantified variables can be instantiated by any term of the suitable sort. When a satisfying assignment that does not generate any theory conflict contains an essentially universally quantified formula, the instantiation module generates and returns singleton clauses whose only formulas are instances of an instantiation axiom schema. The instantiation terms are usually chosen by a heuristic based on E-matching [6, 9]. This heuristic selects ground terms that appear in the literals composing the satisfying assignment, sometimes based on annotations called *triggers* (i.e. sets of term patterns). If a quantified formula is of the form $\forall x_1 \ldots . \forall x_n . A[x_1, \ldots, x_n]$ or $\neg\exists x_1 \ldots . \exists x_n . A[x_1, \ldots, x_n]$ — with $A[\_]$ being a formula not starting with an essentially universal quantifier — the instantiation module instantiates not only the first universally quantified variable x_1, but all the variables x_i at once. This instantiation heuristic (based on E-matching) is incomplete, and a simple clause set that is unsatisfiable but irrefutable due to this incompleteness is simple: $\{\forall x.P(x)\ ;\ \forall y.\neg P(y)\}$. To find a refutation, the solver shall instantiate x and y to the same arbitrary term t and resolve the two unit clauses with each other.

Taking these remarks into account, the most straightforward inference rules for the instantiation module would be:

$$\frac{}{\forall \vec{x}.F(\vec{x}) \rightarrow F(\vec{a})}\ \textit{forall\_inst\_axiom}$$

$$\frac{}{F(\vec{a}) \rightarrow \exists \vec{x}.F(\vec{x})}\ \textit{exists\_inst\_axiom}$$

where $\vec{x}$ denotes a sequence of variables $x_1, \ldots, x_n$ and $\vec{a}$ a sequence of terms $a_1, \ldots, a_n$ of suitable sort.

An obvious and easily implementable idea to improve these rules is to combine them with the clause form transformation rule for implication, as shown below. This reduces the size of proofs, since it eliminates the need to always apply the implication rules after the instantiation axioms.

$$\frac{}{\neg\forall \vec{x}.F(\vec{x}), F(\vec{a})}\ \textit{forall\_inst\_cnf\_axiom}$$

$$\frac{}{\neg F(\vec{a}), \exists \vec{x}.F(\vec{x})}\ \textit{exists\_inst\_cnf\_axiom}$$

In first-order resolution proofs, it is usual to follow a convention that considers variables to be implicitly universally quantified. Universal quantifiers then simply do not (need to) appear in the proof. One might wonder if it would be desirable to adopt a similar convention in the presented proof format and rules. The answer is negative: because the SMT-LIB standard does not enforce any naming convention to distinguish identifiers for constants and for free variables, a proof checker would not be able to (easily) tell whether a given identifier (e.g. x) is a constant or a variable. With explicit rules for omiting the quantifiers, a more sophisticated proof-checker could keep track of which identifiers are variables. However, this would imply an undesirable loss of simplicity of the proof format and of the proof checker.

4 Rules for Skolemization of Essentially Existential Quantifiers

Essentially existential quantifiers are existential quantifiers that occur with positive polarity or universal quantifiers that occur with negative polarity. veriT eliminates them by skolemization during a pre-processing phase. The simplest solution would be to disregard this kind of pre-processing in the proof

production. However, this would go against the style of veriT's proof format, since veriT does produce proofs for other pre-processing tasks such as clause form transformation. Another simple solution consists of having a single macro inference rule that skolemizes all essentially existential quantifiers:

$$\frac{F}{sk(F)}\ \mathit{skolemize\_all}$$

where $sk(F)$ is any skolemization of F.

The rule *skolemize_all* is simple to implement in veriT and simple to check by an independent proof checker. The proof checker just needs to traverse F and $sk(F)$ once, checking that each essentially existential quantifier is eliminated and the quantified variables it binds are replaced by skolem terms headed by skolem symbols that do not occur anywhere else in the proof and whose arguments are (a subset, depending on the skolemization algorithm used, of) variables bound by essentially universal quantifiers having scope over the eliminated quantifier. This rule is also convenient from the point of view of size, since it is clearly linear in the size $|sk(F)|$ of $sk(F)$. $|sk(F)|$, however, is in the worst case $\Theta(|F|^2)$, if F is a tree-formula. To see that $|sk(F)|$ is $O(|F|^2)$, just note that the number of essentially existential quantifiers in $|F|$ is $O(|F|)$ and each of these quantifiers is replaced by a skolem-term of size $O(|F|)$. To see that in the worst case $|sk(F)|$ is $\Omega(|F|^2)$, just consider the following example sequence:

$$F_n = \forall x_1 \ldots \forall x_n \exists y_1 \ldots \exists y_n . P(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

If F is a dag-formula, skolemization may require it to be transformed to an exponentially bigger tree-formula first. In this case, the worst-case size of $|sk(F)|$ is $\Theta(2^{|F|})$. An example where this happens is available in [7].

However, the rule *skolemize_all* has the disadvantage of being very coarse-grained, since it skolemizes the whole formula at once. In trying to develop more fine-grained inference rules, it would be desirable to have something analogous to the rules *forall_inst_axiom* and *exists_inst_axiom*. This could be attempted with rules such as the following:

$$\frac{}{\exists x.F(x) \rightarrow F(f_{new}(x_1, \ldots, x_n))}\ \mathit{exists\_skolem\_axiom}$$

$$\frac{}{F(f_{new}(x_1, \ldots, x_n)) \rightarrow \forall x.F(x)}\ \mathit{forall\_skolem\_axiom}$$

$$\frac{}{\neg\exists x.F(x), F(f_{new}(x_1, \ldots, x_n))}\ \mathit{exists\_skolem\_cnf\_axiom}$$

$$\frac{}{\neg F(f_{new}(x_1, \ldots, x_n)), \forall x.F(x)}\ \mathit{forall\_skolem\_cnf\_axiom}$$

where $x_1, \ldots, x_n$ are the free variables occurring in $F(x)$, f_{new} is a fresh new skolem symbol, not occurring anywhere else in the proof. The rules above are validity-preserving only if we just consider models in which $f_{new}(x_1, \ldots, x_n)$ has a fixed interpretation as the witness of the essentially existentially quantified variable it replaces (if such a witness exists). Otherwise, these rules are merely satisfiability-preserving.

In the case of instantiation of essentially universal quantifiers, when a quantified formula g (e.g. $\forall x.F(x)$) needs to be instantiated, g is one of the formulas in a clause c (e.g. $\Gamma, \forall x.F(x)$). So, after stating an instantiation axiom clause c' (e.g. $\neg\forall x.F(x), F(a)$), we can do the actual instantiation of g in c simply by resolving c with c'. This replaces g by its instance in c. However, in the case of skolemization, a quantified formula g (e.g. $\exists x.F(x)$) may often occur not shallowly as a direct formula of a clause c but more deeply as a subformula of a formula in c (e.g. $\Gamma, \forall y.\exists x.F(x)$). Therefore, replacing g by its instance in c cannot be done simply by resolution with a skolemization axiom. To overcome this problem, a deep version of resolution is proposed, so that one of the resolved formulas can occur arbitrarily deep inside

another formula. This rule may be used in the more general case of the replacement of a deep occurrence of a subformula by another:

$$\frac{\Gamma, \neg F_1, F_2 \quad \Delta, F^+(F_1)}{\Gamma, \Delta, F^+(F_2)}\ \textit{deep\_resolution}+$$

$$\frac{\Gamma, \neg F_1, F_2 \quad \Delta, F^-(F_2)}{\Gamma, \Delta, F^-(F_1)}\ \textit{deep\_resolution}-$$

where the signs $+$ and $-$ indicate the polarity of the annotated subformula.

Note that *deep_resolution*$+$ and *deep_resolution*$-$ are analogous to deep applications of modus ponens and modus tollens, but fit better in the style of veriT's proof format, which is based on resolution.

$$\frac{\Gamma, F_1 \to F_2 \quad \Delta, F^+[F_1]}{\Gamma, \Delta, F^+[F_2]}\ \textit{deep\_modus\_ponens}$$

$$\frac{\Gamma, F_1 \to F_2 \quad \Delta, F^-[F_2]}{\Gamma, \Delta, F^-[F_1]}\ \textit{deep\_modus\_tollens}$$

$$\frac{\Gamma, F_1 \leftrightarrow F_2 \quad \Delta, F[F_1]}{\Gamma, \Delta, F[F_2]}\ \textit{deep\_replacement}$$

This approach with skolemization axioms and deep resolution has many problems, though. Firstly, there is a significant increase in the size of proofs: if m quantifiers need to be skolemized and for the sake of fine-grainedness a deep replacement is performed separately for each of the quantifiers, then there will be $\Theta(m)$ inferences, whose conclusions are of size $O(|sk(F)|)$. Consequently, there is also a significant increase in the proof-checking time. Anti-prenexing the quantifiers as much as possible could reduce this problem in the average case.

Secondly and perhaps more seriously, proof-checking the skolemization axiom rules and the deep resolution rules depends on being capable of distinguishing identifiers of free variables and constants, and the SMT-LIB standard does not enforce any distinction. To understand this issue, consider the following formulas:

$$F_1 := \forall x.\exists y.P(x,c,y) \qquad F_2 := \forall c.\forall x.\exists y.P(x,c,y)$$

and consider the following *exists_skolem_axiom*:

$$\exists y\, P(x,c,y) \to P(x,c,f_{new}(x))$$

Proof-checking this axiom depends on being able to tell whether c is a variable or a constant, for if it were a variable, then c should also have been listed as an argument of f_{new}. Moreover, both F_1 and F_2 might occur in a proof, and then c occurs both as a constant identifier and as a variable identifier. In such cases, the proof-checker should be able to accept this skolem axiom, it should be able to accept a deep resolution (or modus ponens) with F_1 (concluding $\forall x.P(x,c,f_{new}(x))$), but it should also be able to reject an incorrect deep resolution (or modus ponens) with F_2 (unsoundly concluding $\forall x.P(x,c,f_{new}(x))$ instead of $\forall x.\forall c.P(x,c,f_{new}(x,c))$). This simple example shows that the combination of skolem axioms, deep replacement rules and no distinction between identifiers for constants and for variables leads to unsoundness. This could be fixed by requiring non-local side conditions in the deep replacement rules, so that a deep replacement is only allowed if any skolem function symbol occuring in the replacing formula has as arguments all the identifiers that occur free in the replacing formula and that become bound after the replacement. Although this is technically feasible, it is questionable whether the increase

in the complexity of proof-checking is a reasonable price to pay for the fine-grainedness and elegance provided by the deep replacement rules and the skolem axioms.

This means that the proof checker would not be able the check the correctness of a skolemization axiom inference locally; it would not be able to verify whether the list of arguments of the skolem function, which ought to contain all the free variables (but not the constants) of $F(x)$, is correct. The proof checker would only be able to tell whether the arguments of the skolem function are variables or constants when the deep resolutions are performed. Proof-checking would not be just local anymore, since the proof checker would need to keep track of some global correctness conditions. One way to keep proof-checking local, while still keeping fine-grainedness would be to combine the skolemization axiom rule and the deep resolution rule into a single unary inference rule, as follows:

$$\frac{\Gamma, G^+[\exists x.F(x)]}{\Gamma, G^+[F(f_{new}(x_1,\ldots,x_n))]}\ \textit{exists\_skolem}$$

$$\frac{\Gamma, G^-[\forall x.F(x)]}{\Gamma, G^-[F(f_{new}(x_1,\ldots,x_n))]}\ \textit{forall\_skolem}$$

where $x_1,\ldots,x_n$ are the free variables of $F(x)$ that are bound in $G(Qx.F(x))$.

Another approach would be to give up using skolem terms and use Hilbert's epsilon terms instead. A problem with this approach is that the size of the transformed formula $epsilon(F)$ for a first-order formula F is in the worst-case $\Omega(2^{|F|})$ for tree-like F. This lower bound can be easily proved by considering the sequence of linearly growing formulas $F_n := \exists x_1 \ldots \exists x_n.P(x_1,\ldots,x_n)$ and checking that the sequence $epsilon(F_n)$ grows indeed in $\Omega(2^n)$. For dag-like F, another exponential blow up is possible, since the formula may need to be transformed to a tree, and hence the size is $\Omega(2^{2^{|F|}})$.

$$\frac{}{\exists \vec{x}.F(\vec{x}) \rightarrow F(\varepsilon \vec{x}.F(\vec{x}))}\ \textit{epsilon\_axiom\_1}$$

$$\frac{}{F(\varepsilon \vec{x}.\neg F(\vec{x})) \rightarrow \forall \vec{x}.F(\vec{x})}\ \textit{epsilon\_axiom\_2}$$

Yet another alternative worth considering would be to stop doing skolemization as a pre-processing step altogether, and do it only on demand, when an essentially existentially quantified formula occurs shallowly as a direct formula of a clause. In this case, no deep replacement would be necessary, and skolem terms would be always just skolem constants. Equivalently, strong quantifier rules or axioms that instantiate the essentially existentially quantified variables by eigen-variables could be used.

5 Conclusions

In this paper we have presented a few alternative inference rules for handling quantifiers in the proof format of veriT. We showed that each alternative has advantages but also disadvantages with respect to the requirements mentioned in the introduction. Therefore, we found none of the alternatives completely satisfactory.

Since it seems to be difficult to find the right balance to satisfy most if not all the requirements simultaneously, we have implemented some of these alternative rules in veriT selecting those that seemed to fit better within the existing proof style of that tool. The quantifier instantiation module produces instances of the rules *forall_inst_axiom* and *exists_inst_axiom* to justify lemmas that are added to the Boolean satisfiability solver. The clauses they introduce are then combined with existing rules for CNF transformation and resolution. Skolemization is applied to the input formula only on essentially existentially quantified

variables occuring at the outermost level, and thus only produce skolem constants. Also, since the formulas generated by the quantifier instantiation module might reveal essentially existential quantifiers at the outermost level, skolemization is also applied to instances. In both cases, *exists_skolem_cnf_axiom* and *forall_skolem_cnf_axiom* are used in the proof, and the resulting clauses are further combined using *deep_resolution* rules.

References

[1] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB standard : Version 2.0, March 2010. First official release of Version 2.0 of the SMT-LIB standard.

[2] Clark Barrett and Cesare Tinelli. CVC3. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer Berlin / Heidelberg, 2007.

[3] Frédéric Besson, Pascal Fontaine, and Laurent Théry. A flexible proof format for SMT: a proposal, 2011. Workshop on Proof eXchange for Theorem Proving (PxTP).

[4] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. veriT: an open, trustable and efficient SMT-solver. In Renate Schmidt, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Computer Science*, pages 151–156, Montreal, Canada, 2009. Springer.

[5] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer Berlin / Heidelberg, 2008.

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

[7] Pascal Fontaine. *Techniques for Verification of Concurrent Systems with Invariants*. PhD thesis, Université de Liège, Belgium, September 2004.

[8] Yeting Ge, Clark Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. In Frank Pfenning, editor, *Automated Deduction — CADE-21*, volume 4603 of *Lecture Notes in Computer Science*, pages 167–182. Springer Berlin / Heidelberg, 2007.

[9] M. Moskal, J. Lopuszanski, and J.R. Kiniry. E-matching for fun and profit. *Electronic Notes in Theoretical Computer Science*, 198(2):19–35, 2008. Proceedings of the 5th Int'l Workshop on Satisfiability Modulo Theories (SMT 2007).