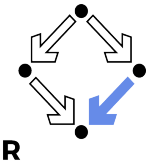
RISC



RESEARCH INSTITUTE FOR
SYMBOLIC COMPUTATION

JKU

JOHANNES KEPLER
UNIVERSITY LINZ

Semantic Evaluation versus SMT Solving in the RISCAL Model Checker

Wolfgang Schreiner, Franz-Xaver Reichl

June 2021

RISC Report Series No. 21-11

Available at https://doi.org/10.35011/risc.21-11

Editors: RISC Faculty
B. Buchberger, R. Hemmecke, T. Jebelean, T. Kutsia, G. Landsmann, P. Paule, V. Pillwein, N. Popov, J. Schicho, C. Schneider, W. Schreiner, W. Windsteiger, F. Winkler.

Supported by: JKU Linz Institute of Technology (LIT) Project LOGTECHEDU, Aktion Österreich- Slowakei Project 2019-10-15-003, Austrian Science Fund (FWF) grant W1255.

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Str. 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Semantic Evaluation versus SMT Solving in the RISCAL Model Checker

Wolfgang Schreiner*
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University Linz, Austria
Wolfgang.Schreiner@risc.jku.at

Franz-Xaver Reichl†
Algorithms and Complexity Group
TU Wien, Austria
freichl@ac.tuwien.ac.at

June 4, 2021

Abstract

In this paper, we compare two alternative mechanisms for deciding the validity of first-order formulas over finite domains supported by the mathematical model checker RISCAL: first, the original built-in approach of "semantic evaluation" (based on an implementation of the denotational semantics of the RISCAL language) and, second, the later implemented approach of SMT solving (based on satisfiability preserving translations of RISCAL formulas to formulas in the SMT-LIB logic QF_UFBV, respectively to quantified SMT-LIB bitvector formulas). After a short presentation of the two approaches and a discussion of their fundamental pros and cons, we quantitatively evaluate them, both by a set of artificial benchmarks and by a set of benchmarks taken from real-life applications of RISCAL; for this, we apply the state-of-the-art SMT solvers Boolector, CVC4, Yices, and Z3. Our benchmarks demonstrate that (while SMT solving generally vastly outperforms semantic evaluation), the various SMT solvers exhibit great performance differences. More important, our investigations also identify some classes of formulas where semantic evaluation is able to compete with (or even outperform) satisfiability solving, outlining some room for improvements in the translation of RISCAL formulas to SMT-LIB formulas as well as in the current SMT technology.

*Supported by the JKU Linz Institute of Technology (LIT) Project LOGTECHEDU and by the Aktion Österreich-Slowakei Project 2019-10-15-003
†Supported by the Austrian Science Fund (FWF) under grant W1255.

Contents

1 Introduction

The aim of the RISCAL system [30, 31] is to support the analysis of theories and algorithms over discrete domains, that arise in computer science, discrete mathematics, logic, and algebra. For this purpose, RISCAL provides an expressive specification language based on a strongly typed variant of first-order logic with a rich set of types and operations in which theories can be formulated and algorithms can be specified; nevertheless the validity of all formulas and the correctness of all algorithms is decidable. This is because all RISCAL types have finite sizes and thus the underlying model is finite; the sizes of the types and thus of the model is configurable by model parameters. Thus, a RISCAL model actually represents an infinite set of finite models; before verifying the validity of a theorem over the infinite model set by a deductive proof in some theorem proving environment, we can check its validity over selected finite instances of the set by model checking in RISCAL. RISCAL thus represents a tool for the automated validation and (more important) *falsification* of conjectures, in order to avoid fruitless attempts to prove invalid statements [33]. The system has been mainly developed for educational purposes [34, 32] but it also has been applied in research [29].

The basic mechanism of RISCAL for deciding the validity of formulas and the correctness of algorithms is "semantic evaluation", which is based on a constructive implementation of the denotational semantics of all kinds of syntactic phrases allowed by the language. However, since 2020, the system also provides an alternative (and potentially much more efficient) decision mechanism based on SMT (satisfiability modulo theories) solving, implemented by the second author [28, 35]. In this approach, the decision of a RISCAL formula is performed via a translation to a formula in the SMT-LIB language [6] and the application of some external SMT solver (currently, the SMT solvers Boolector [25], CVC4 [5], Yices [14], and Z3 [24] are supported). Indeed, this has achieved great performance improvements [28] and allowed to constructively work with theories that were out of reach of semantic evaluation.

Actually, SMT solvers have already served for a long time as the backends of various program verification tools and environments for programming languages, real languages such as Java [11, 12, 2], C# [3] and C [20], as well as algorithmic languages such as Why3 [10] and Dafny [21]. Also Microsoft's verification backend Boogie [4] generates SMT-LIB conditions that are discharged by the SMT solver Z3. Furthermore, SAT/SMT solvers are applied for the analysis of system modeling languages such as Alloy [19], Event-B [1], and VDM [27]. Last but not least, they are employed as automatic proving backends for interactive theorem provers, e.g., in Isabelle's "sledgehammer" component [26, 8] and in Coq's SMTCoq plugin [7, 15].

As for the translation of higher-level specification languages into the languages of satisfiability solvers, [18] describes the techniques used in the Alloy Analyzer to transform formulas from first-order relational logic into SAT problems; [37] discusses improvements of these techniques, which have been implemented in the SAT-based relational model finder Kodkod. On top of Kodkod, the counterexample generator Nitpick [9] generates finite countermodels of Isabelle formulas by a translation to relational logic. In [16], it is briefly sketched how in a case study Alloy constraints have been (manually) translated into the language of the SMT solver Yices, which shows drastic speedups when tautologies are decided. [22] sketches the encoding of VDM proof obligations as SMT problems proved with the SMT solver Z3. In somewhat more detail, [13] discusses the implementation of a SMT plugin for the Event-B platform Rodin and

experimentally compares this plugin with plugins for other provers.

However, as beneficial SMT solving in general is, in our own work of deciding RISCAL formulas we also have regularly encountered cases where the performance of the SMT-based decision is comparatively poor, sometimes even beaten by semantic evaluation. While some potential reasons have already been outlined in [28], a more systematic analysis and evaluation has been lacking so far. Also the work reported in the scientific literature is a bit unsatisfactory in this respect: while the relative merits of SMT solvers are regularly evaluated in the SMT-COMP competition series [36], it is harder to find comparisons of SMT solving with alternative decision mechanisms such as the Vampire prover [17].

Therefore, in this paper we provide a detailed comparison of the built-in decision mechanism of RISCAL by semantic evaluation with the corresponding decisions by SMT solving. In particular, we identify classes of situations, where the performance of SMT-based decisions is relatively low, i.e., where indeed "semantic evaluation competes with SMT solving", as a starting point for potential improvements in the SMT-LIB translation of RISCAL respectively in SMT technology in general. While our insights are clearly limited in the particular strategy applied for translating RISCAL formulas to SMT-LIB (discussed below), our work shows that SMT is not a panacea in all kinds of reasoning problems but has to be applied with certain caveats.

Closest to our work is the presentation given in [23], where the untyped first-order logic of Lamport's TLA$^+$ specification language is translated to SMT-LIB conditions that are discharged by the SMT solvers CVC4 and Z3; however the results have not been experimentally compared with the built-in TLC model checker, but only with the interactive TLPS proving backend. As another difference, the TLA$^+$ translation generates formulas in the SMT-LIB logic AUFLIA (closed formulas over the theory of linear integer arithmetic and integer arrays extended with free sort and function symbols) that heavily relies on a nonconstructive encoding of non-integer values by uninterpreted sorts and functions with corresponding background axioms. In contrast to this, in the RISCAL translation we mainly focus on the generation of formulas in the logic QF_UFBV (unquantified formulas over bit vectors with uninterpreted sort and function symbols), which minimizes the use of uninterpreted functions by a constructive encoding of all types as bit vectors. Indeed, as our experiments will demonstrate, the application of axiomatized functions may significantly hurt the performance of SMT solving. Additionally, to this a quantifier preserving translation is provided.

The remainder of this paper is organized as follows: In Section 2, we outline the principles of the decision mechanisms of semantic evaluation and SMT solving applied in RISCAL. In Section 3, we present a set of artificial benchmarks which we use to compare both mechanisms under variations of several parameters. In Section 4, we extend these investigations to a selected set of benchmarks taken from real-life applications of RISCAL. In Section 5, we present our conclusions derived from these investigations and outline possible strands of further research in the development of RISCAL and in SMT technology.

2 Deciding First-Order Formulas

In the following, we briefly describe the two alternative mechanisms that RISCAL implements for deciding first-order formulas: internal semantic evaluation and the application of external

SMT solvers.

2.1 Semantic Evaluation

The built-in mechanism of RISCAL for deciding first-order formulas (and verifying algorithms) over finite domains is based on the translation of every syntactic phrase of the RISCAL language into an executable representation of its denotational semantics. In more detail, this representation is a Java "lambda expression" that in essence maps an assignment for the free variables of the phrase to the value denoted by its semantics, i.e., the truth value of a formula or the updated variable assignment resulting from the execution of a command [34, 35]. The Java compiler translates this expression to an object of an anonymous class (which corresponds to a "closure" in functional programming) with a corresponding evaluation method that the Java compiler translates into Java byte code. From this, the Java just-in-time (JIT) compiler mechanism generates at runtime, efficient machine code such that, for instance, the execution of a procedure formulated in the RISCAL algorithm language does not take more than one "order of magnitude" (say 10 times) longer than that of a corresponding Java procedure.

In the case of first-order logic formulas, the most interesting part of the translation is that of a universally quantified formula $\forall x{:}D.\ F[x]$ respectively that of an existentially quantified formula $\exists x{:}D.\ F[x]$; these translations are semi-formally sketched below (here $[\![\ F\ ]\!]$ denotes the body of a function whose execution yields the truth value of F):

```
[[ ∀x:D. F[x] ]] :=
  e := enumerate(D)
  loop
    if empty(e) then return true
    x := next(e); e := rest(e)
    if ¬call([[ F[x] ]]) then return false
```

```
[[ ∃x:D. F[x] ]] :=
  e := enumerate(D)
  loop
    if empty(e) then return false
    x := next(e); e := rest(e)
    if call([[ F[x] ]]) then return true
```

The core of the translation is a loop that enumerates every element of the domain D of the quantified variable x and evaluates the body of the quantified formula with x bound to that element, until the truth value of the body determines the overall result. As an optimization, RISCAL actually implements the enumeration of D in a mostly "lazy" fashion such that it is not necessary to simultaneously keep all elements in memory; the generation stops when the first element has been produced that allows to decide the formula. Consequently, the "worst case" is exhibited by a *true universal formula* respectively a *false existential formula*: here we have to generate all elements, before we can decide that the universal formula is true respectively the existential formula is false.

Furthermore, RISCAL supports expressions that do not denote unique values, for example the term (choose $x{:}D$ with $F[x]$) that denotes any value x of the domain D that satisfies the formula $F[x]$. RISCAL implements such a term in its "nondeterministic" evaluation mode [35] by the computation of a (lazily evaluated) *stream* of such values:

```
⟦ choose x:D with F[x] ⟧ :=
  e := enumerate(D)
  loop
    if empty(e) then return
    x := next(e); e := rest(e)
    if call(⟦ F[x] ⟧) then yield e
```

Like for quantified formulas, the core of this translation is a loop that enumerates every element of D; the translation yields each element that satisfies the body formula as a value of the expression (i.e., this value is appended to a stream of values denoted by the term). A formula that depends on such terms correspondingly denotes a stream of truth values; the formula is only considered as valid if this stream only consists of instances of truth value "true".

Not necessarily unique choices arise in many mathematical definitions and algorithms ("choose any element e of set S"). In particular, applications of such expressions may arise from the modular verification of user-defined operations; here not the definition of an operation but its contract is considered. For instance, an application $f(a)$ of a function f, specified as

$$\text{fun } f(x{:}D){:}\ D \text{ ensures } F[x, result]$$

(where *result*, is a special variable that denotes the result of the function) can be replaced by the expression (choose $result{:}D$ with $F[a, result]$).

2.2 SMT Solving

The problem of deciding the validity of a formula F, denoted by the predicate $valid[\![F]\!]$, may be reduced to the problem of deciding the satisfiability of the negation of F, denoted by the predicate $sat[\![\neg F]\!]$, by applying the following equivalence:

$$valid[\![F]\!] \equiv \neg sat[\![\neg F]\!]$$

For this reason, RISCAL implements a translation of RISCAL formulas to formulas in the SMT-LIB format [6] which is supported by numerous SMT (satisfiability modulo theories) solvers. Thus RISCAL can decide the validity of the RISCAL formula F by letting an external SMT solver decide the satisfiability of the SMT-LIB version of $\neg F$. However, before doing so, we have to solve various problems:

1. **Encoding:** We have to select an appropriate background theory (respectively a combination of such theories). On the one hand, this theory must be rich enough to encode all the types and associated operations supported by the RISCAL language (integers, tuples/records, sets, arrays with integer indices, maps with arbitrary index domains). On the other hand, this theory must be well supported by various solvers. We chose for this purpose the core theory of *fixed-size bit vectors*. This domain is adequate for our purpose, because every RISCAL domain is finite: every element of a domain with n elements can be (in principle, see the corresponding discussion later) represented by a vector of $\lceil \log n \rceil$ bits (however note that for technical reasons in the actual translation, the translation of a

domain with n elements may require bit vectors longer than $\lceil \log \rceil$; furthermore the set of bit vector operations provided by the theory is expressive enough to allow a proper encoding of the various RISCAL operations. However, bit vectors alone are not enough: the treatment of quantifiers and choose expressions (discussed below) requires functions which are not explicitly characterized by definitions but only implicitly by axioms; therefore we demand from the theory also support for *uninterpreted functions*.

2. **Quantifiers:** The SMT-LIB format also allows to express quantified formulas, and indeed many SMT solvers support quantification. However, the main SMT-LIB logic that provides bit vectors and uninterpreted function is the logic QF_UFBV of "unquantified formulas over bit vectors with uninterpreted sort and function symbols" which is supported, e.g., by the well known SMT solvers Boolector, CVC4, Yices, and Z3. To use this theory, we therefore have to translate a RISCAL formula with quantifiers into a corresponding quantifier-free SMT-LIB formula (since some SMT solvers support, as a non-standard extension, quantified bit vector formulas with uninterpreted functions, we will in the benchmarks later compare the approach of eliminating quantifiers with the approach of preserving them).

3. **Choices:** While expressions denoting unique values can be directly encoded by bit vectors operations, "choose" expressions with their non-unique denotation values have to be especially encoded by uninterpreted functions with an appropriate axiomatizations. However, this translation makes the process of satisfiability solving costly; we thus aim for optimizations that in many cases improve the decision process.

We will now discuss these issues in turn.

Encoding We deal with the problem of encoding in large detail in [28] where we give an appropriate translation of quantifier-free RISCAL formulas to QF_UFBV formulas. These details are not essential for our subsequent discussion, so we only sketch the basic ideas.

The RISCAL numerical domains $\mathbb{N}[N]$ and $\mathbb{Z}[N1, N2]$ are translated into the SMT-LIB domain `(_ BitVec` L`)` of fixed size bit vectors of length L where L is the smallest number such that $N \leq 2^L - 1$ respectively if $N1$ is negative such that $-2^{L-1} \leq N1$ and $N2 \leq 2^{L-1} - 1$. The RISCAL type checker keeps track of the growth of numerical values in arithmetic computations such that for any numerical expression an appropriate bit vector size can be chosen to avoid any overflows; thus the mathematical integers of RISCAL can be faithfully translated to the machine integers of QF_UFBV. Likewise the RISCAL container types (such as `Array`$[N,T]$) are translated to the usual bit vector representations (such as `(_ BitVec` $N \cdot M$`)` where M is the size of the bit vector representation of element type T) with the appropriate encoding of operations for construction, selection, and update.

However, it should be noted that not every bit vector b of the domain B resulting from the translation of a RISCAL domain D necessarily represents an actual value from D, i.e., the mapping $D \rightarrow B$ only denotes an injection, not a bijection. Since this will become relevant in the subsequent treatment of quantifiers, the translation provides for every RISCAL domain D a predicate $p_D(b)$ that determines whether bit vector b of B indeed encodes an element of D.

Quantifiers The problem of dealing with quantifiers is addressed by the following equivalences, which semi-formally sketch how quantifiers can be removed from RISCAL formulas:

$$\begin{aligned} \mathit{valid}[\![\, \forall x{:}D.\ F[x] \,]\!] &\equiv \neg \mathit{sat}[\![\, \neg \forall x{:}D.\ F[x] \,]\!] \\ &\equiv \neg \mathit{sat}[\![\, \exists x{:}D.\ \neg F[x] \,]\!] \equiv \neg \mathit{sat}[\![\, \neg F[f(x_1, \ldots, x_n)] \,]\!] \\ \mathit{valid}[\![\, \exists x{:}D.\ F[x] \,]\!] &\equiv \neg \mathit{sat}[\![\, \neg \exists x{:}D.\ F[x] \,]\!] \\ &\equiv \neg \mathit{sat}[\![\, \forall x{:}D.\ \neg F[x] \,]\!] \equiv \neg \mathit{sat}[\![\, \neg F[e_1] \wedge \ldots \wedge \neg F[e_n] \,]\!] \end{aligned}$$

We assume that before the translation is applied all formulas have been transformed into *negation normal form*, i.e., all applications of the negation symbol have been *pushed inside* down to the level of atomic formulas. Thus, above occurrences of quantified formulas are *positive*, i.e., they do not appear in the context of negation. Then the translation proceeds as follows (we discuss the conceptually simpler second case first):

1. $\mathit{valid}[\![\, \exists x{:}D.\ F[x] \,]\!]$: the decision of this validity boils down to the decision of the satisfiability of the universally quantified formula $\forall x{:}D.\ \neg F[x]$. Now, if D consists of n values denoted by terms $e_1, \ldots, e_n$, we can expand the quantified formula to an equivalent finite conjunction $\neg F[e_1] \wedge \ldots \wedge \neg F[e_n]$.

2. $\mathit{valid}[\![\, \forall x{:}D.\ F[x] \,]\!]$: the decision of this validity boils down to the decision of the satisfiability of the existentially quantified formula $\exists x{:}D.\ \neg F[x]$. Analogously to the previous case, we could in principle also expand this formula, namely to a finite disjunction $\neg F[e_1] \vee \ldots \vee \neg F[e_n]$.

   However, we generally prefer another option that avoids the blow-up of the formula. Let us assume that the existentially quantified formula appears in the context of n universally quantified variables $x_1, \ldots, x_m$, i.e., the problem of deciding the satisfiability of $\exists x{:}D.\ \neg F[x]$ actually occurs in the course of deciding the satisfiability of a global formula of the following shape:

   $$\forall x_1{:}D_1.\ \ldots \forall x_m{:}D_m.\ \ldots \exists x{:}D.\ \neg F[x]$$

   Then we introduce an m-ary function symbol f that does not appear anywhere else in the global formula; the denoted function can therefore have an arbitrary interpretation (we call such a function a *Skolem function*). Finally, we replace $\exists x{:}D.\ \neg F[x]$ by $\neg F[f(x_1, \ldots, x_m)]$. In the special case $m = 0$, i.e., if there is no outer universally quantified variable, f becomes a *Skolem constant* and the formula becomes $\neg F[f]$.

   Although the resulting formula is not logically equivalent to the original one, it is *equisatisfiable*, i.e., it is satisfiable if and only if the original formula is (if we may choose for all values $x_1, \ldots, x_m$ a value for x that makes $F[x]$ true, then from these choices we may construct the Skolem function f and vice versa). Since the translation preserves satisfiability, the equivalence stated above holds.

From the above translation, deciding the validity of a universally quantified formula seems a priori unproblematic. However, deciding the validity of an existentially quantified formula

may blow-up the formula to a size that is exponential in the depth of the nesting of existential quantifiers; this may also increase the complexity of the decision.

Furthermore, also the decision of the validity of a universally quantified formula may have its problems. Remember that this decision boils down to deciding the satisfiability of a formula $\neg F[f(x_1, \ldots, x_m)]$ with Skolem function f. Here we must be aware that in the translation to the SMT-LIB theory QF_UFBV the domain of f is not anymore the original RISCAL domain D of the existentially quantified variable, but some SMT-LIB type B, whose values encode the values of D. Since not every bit vector b in B represents an element from D, we have to add to the generated SMT-LIB formula, whose satisfiability is to be decided, an additional axiom that constrains the domain of the Skolem function f by the predicate p_D that holds for a bit vector $b \in B$ if and only if b actually represents an element from D:

$$\forall d_1{:}D_1, \ldots, d_m{:}D_m.\ p_D(f(t(d_1), \ldots, t(d_m)))$$

Here $t(d_i)$ shall denote the bit vector value that is associated to the RISCAL value d_i. However, the SMT-LIB translation of such a universally quantified formula is expanded to a conjunction with one conjunct for every element of the domain $D_1 \times \ldots \times D_m$, which may blow up the overall result considerably and overcome the benefits of translating the original formula by using Skolemization rather than by expansion. Therefore, the RISCAL translation mechanism can be configured to apply a heuristic: if the number of conjuncts in the expanded Skolemization axiom is significantly larger than the number of conjuncts derived from expanding the original formula, the translation forsakes Skolemization in favor of direct expansion.

Choices Every RISCAL expression (choose $y{:}D$ with $F[x, y]$) with free variable $x{:}D$ gives in the SMT-LIB translation rise to a new function $f\colon D \to D$ with axiom $\forall x{:}D.\ F[x, f(x)]$ (and correspondingly for multiple free variables). Similarly, in modular verification, every RISCAL operation (function, predicate, procedure) specified by a contract gives rise to an SMT-LIB function with a corresponding axiomatization. However, as explained above, such axiomatizations by universally quantified formulas yield large SMT-LIB expansions and potentially costly SMT decisions (in addition to the user-defined axiomatization, such functions have also to be constrained by the type representation axioms explained in the context of Skolemization above).

However, in certain contexts we may replace applications of such axiomatized functions, as demonstrated by the following small example. Consider a formula

$$\forall a.\ (\ldots f(a) \ldots)$$

where application $f(a)$ occurs positively (unnegated) in a context $(\ldots\ f(a) \ldots)\big)$ that does not embed $f(a)$ in another quantifier. Assume that function $f\colon D \to D$ has been axiomatized as described above, i.e., we actually want to decide the validity of

$$\forall f.\ (\forall x{:}D.\ F[x, f(x)]) \Rightarrow \forall a.\ (\ldots f(a) \ldots)$$

Now, this formula is equivalent to

$$\forall a.\ \forall f.\ (\forall x{:}D.\ F[x, f(x)]) \Rightarrow (\ldots f(a) \ldots)$$

which in turn is equivalent to the following formula:

$$\forall a, b.\ (F[a, b] \Rightarrow \ldots b \ldots)$$

Thus, we have replaced the application of axiomatized function $f(a)$ by a fresh universally quantified variable b with assumption $F[a, b]$. This means that the original axiomatization (which applied formula F to arbitrary values x from D) has been specialized to the instances $F[a, b]$ that are actually of interest.

RISCAL optionally implements in the SMT-LIB translation a generalized form of this transformation under the name "eliminate choices", because it is directly applied to choose expressions given by the user and to choose expressions generated from applications of implicitly defined functions. The option may eliminate such expressions that appear in a positive universal (or negative existential) context with no other intervening quantifiers (such that the transformation preserves the semantics of the formula). In combination with the option "inline definitions", also choose expressions indirectly arising from the definitions of operations may be inlined. While this expands the size of the core formula to be decided, it removes general axiomatizations and may thus be beneficial all in all.

2.3 Comparing Both Approaches

The approach of deciding a quantified formula by semantic evaluation may quickly decide a false universal formula or a true existential formula if the value that allows to make the decision appears early in the enumeration of the domain of the quantified variable. Conversely, the worst case is exhibited when a true universal formula or a false existential formula is decided: here all values of the domain have to be enumerated.

The time required for performing the decision by satisfiability solving is not a priori clear; nevertheless, we can analyze the size of the formulas generated. The critical case is that of a formula that has (in its original unnegated form) a positive occurrence of an existential quantifier. If its variable domain has n values, the size of the formula is multiplied by factor n. If we have m nested occurrences of an existential quantifier, the size of the formula is multiplied by factor n^m. However, also a positive occurrence of a universal quantifier may let the formula grow if the domain of the SMT-LIB translation of the Skolem function has to be constrained. If the original formula occurs in the context of m existential quantifiers whose variable domains have size n, also this formula may grow with factor n^m. This expansion can be mitigated by not applying Skolemization but expansion, which lets the formula grow as for an existential quantifier.

Thus, the positive occurrence of existential quantifiers in the original formula may cause multiple problems: on the one side, because by itself it leads to an expanded formula; on the other side because it may cause the expansion of an axiom that constrains the domain of a Skolem function that arises from a universal quantifier that appears in the scope of the existential formula. But how often do we actually have to deal with positive occurrences of existential formulas? Here one should note the following: a formula $F_1 \wedge \ldots \wedge F_n \Rightarrow G$ is logically equivalent to $\neg F_1 \vee \ldots \neg F_n \vee G$. Therefore, if an assumption F_i is universally quantified, the negation normal form of $\neg F_i$ is an existential formula in positive form. Every universally quantified assumption may therefore cause the blowup of the formula.

As for the decision of formulas involving choice terms, the respective advantages/disadvantages of the two approaches are quite unclear. On the one hand, semantic evaluation requires the enumeration of all elements of the domain of the chosen value; on the other side, satisfiability solving in general requires the axiomatization of an uninterpreted function by a formula that is universally quantified over the domain of the free variables of the choose expression; in certain cases, however, an optimization may be applied that only requires a single instance of that axiom.

Summarizing above considerations, there is only one case where satisfiability solving can be a priori expected to beat semantic evaluation: if the validity of a true formula is to be decided that does not contain any positive occurrences of existential formulas, i.e., no occurrences of universal assumptions or existential goals. Here semantic evaluation shows its worst-case behavior and the translation to a satisfiability problem does not require any formula expansion. In all other cases, semantic evaluation may be lucky in the enumeration of values for bound variables and/or the translation to a satisfiability problem may require formula expansion. To which extent this, however, affects the actual performance of the decision process, remains unclear. The remainder of this paper addresses this question.

3 Artificial Benchmarks

In this section, we utilize a number of artificial (micro-)benchmarks to investigate the relative performance of the two kinds of decision mechanisms supported by RISCAL, built-in semantic evaluation and satisfiability solving by external SMT solvers. For this purpose, it does not seem advisable to consider all kinds of formula structures, predicates, functions, and variable types supported by RISCAL; the space of possibilities is overwhelmingly large such that the investigation would be drowned in an unmanageable amount of data. Therefore, we will rather concentrate on a small number of formula patterns that are instantiated with a small number of predicates in the hope of being able to get some initial insights. In Section 4, we will investigate whether and how these insights can be applied to real-life examples.

3.1 Basic Setup

We start by investigating the "base behavior" of the two decision approaches. For this, we use the following two predicates:

$$\begin{aligned} \textit{cycle4-valid} &\equiv \neg(x_1 < x_2 \wedge x_2 < x_3 \wedge x_3 < x_4 \wedge x_4 < x_1) \\ \textit{cycle4-sat1} &\equiv \neg(x_1 < x_2 \wedge x_2 < x_3 \wedge x_3 < x_4 \wedge x_4 < x_1 + 4) \end{aligned}$$

Both predicates have free occurrences of four integer variables x_1, x_2, x_3, x_4. Predicate *cycle4-valid* states that these variables cannot form a "less-than cycle"; due to the transitivity of the less-than relation, this predicate clearly is valid and its negation is unsatisfiable. On the other side, predicate *cycle4-sat* is satisfiable but not valid, as is its negation. However, while *cycle4-sat1* has many satisfying assignments, its negation has only few (those with with $x_2 = 1 + x_1$, $x3 = 1 + x_2 = 2 + x_1$, $x_4 = 1 + x_3 = 3 + x_1$); thus *cycle4-sat* represents a "mostly valid" formula, while its negation denotes a "mostly unsatisfiable" one. Both predicates only depend on the atomic predicate $<$ (the second one, also on the constant addition +4). The predicates do not

require any complex calculations or decisions in order to most clearly exhibit the effect of various forms of quantification structures on the decision process.

In particular, we investigate the eight quantification patterns $\exists^4\forall^0$, $\exists^3\forall^1$,$\exists^2\forall^2$, $\exists^1\forall^3$, $\forall^4\exists^0$, $\forall^3\exists^1$, $\forall^2\exists^2$, $\forall^1\exists^3$ where Q^i represents the i-fold repetition of quantifier Q and the variables are quantified in the order x_1, x_2, x_3, x_4. Thus, e.g., the combination of quantification pattern $\exists^3\forall^1$ with predicate *cycle4-valid* represents the following formula:

$$\exists x_1{:}D, x_2{:}D, x_3{:}D.\ \forall x_4{:}D.\ \neg(x_1 < x_2 \wedge x_2 < x_3 \wedge x_3 < x_4 \wedge x_4 < x_1)$$

Above quantifier patterns consider the cases of purely existential formulas ($\exists^4\forall^0$), purely universal formulas ($\forall^4\exists^0$), as well as existential formulas with universal bodies ($\exists^i\forall^j$) and universal formulas with existential bodies ($\forall^j\exists^i$) where the number of corresponding quantifiers represent different sizes of the respective quantification ranges. Thus, all in all, we consider formulas with at most one alternation of quantifiers (in total eight more quantification patterns with at most three quantifier alternations could be expressed).

As for the domain D of the variables, we focus on $D := \mathbb{N}[2^N - 1]$ for some $N \in \mathbb{N}$, i.e., each of the 4 variables holds some natural number up to maximum $2^N - 1$; the total value space thus consists of 2^{4N} elements. In the following benchmarks, we choose $N := 6$, i.e., a value space of size 2^{24}. For a formula of shape $\exists^i\forall^j$ respectively $\forall^j\exists^i$, this value space is partitioned according to the numbers i and j of existentially and universally quantified variables, respectively. The "existential search space" has size 2^{iN}, which leads in the QF_UFBV translation to the generation of 2^{iN} clauses. The "universal search space" has size 2^{jN}, which leads in QF_UFBV to j Skolem constants (if the universal quantifiers are outermost) or j Skolem functions of arity i (if the universal quantifiers are innermost); the domain of each Skolem constant or function is a bit vector of length N with 2^N possible values.

3.2 Experimental Results

The four diagrams in Figure 1 plot the decision times for the quantified formulas with (valid) predicate *cycle4-valid* respectively its (unsatisfiable) negation and for the satisfiable (mostly valid) predicate *cycle4-sat1* respectively its also satisfiable (but mostly unsatisfiable) negation. The labels of the horizontal axis denote the applied quantification pattern (labels eiaj respectively aiej denote patterns $\exists^i\forall^j$ respectively $\forall^i\exists^j$). The vertical axis denotes the decision time in ms, within the interval $[1, 60000]$ (please note the logarithmic scale). All decision procedures were forcefully terminated after 1 minute; thus, if a plot point is at the top line of the diagram, this actually indicates "timeout" or "no result" (a timeout is also indicated, if the software ran out of memory or produced any other kind of error). All measurements were performed on a virtual GNU/Linux machine with a CPU of type i7-2670QM@2.20GHz using 8 GB RAM. In case of the SMT solvers, only the time for the actual decision (not including the time for translating the RISCAL formula to an SMT-LIB formula) was considered.

The various labeled curves give the times for the decisions mechanisms that we have benchmarked using RISCAL 3.8.5 and the SMT solvers Boolector 3.1.0, CVC4 1.7, Yices 2.6.1, and Z3 4.8.7:

- RISCAL: the built-in semantic evaluation mechanism of RISCAL.

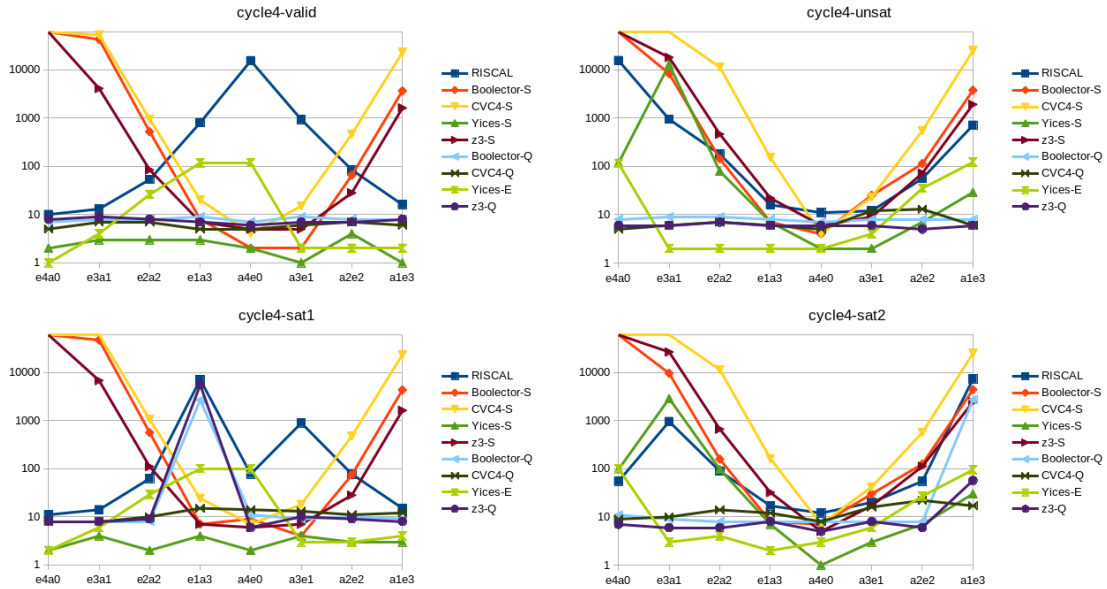


Figure 1: Artificial Benchmarks: Base Behavior

- Boolector-S, CVC4-S, Yices-S, Z3-S: the application of the various SMT solvers to the generated QF_UFBV formula. As described in Section 2, in this formula all quantifiers are removed by Skolemization (in case of the originally universal quantifiers) respectively expansion (in case of the existential quantifiers).

- Boolector-Q, CVC4-Q, Z3-Q: the application of the SMT solvers to a formula in the theory of bit vectors. Here, however, in the generated formula all quantifiers are preserved (Boolector, CVC4, and Z3 also support quantification).

- Yices-E: the application of Yices to deciding the validity of a formula in the SMT-LIB logic QF_UFBV. Here, also the (original) universal quantifiers are removed by expansion rather than by Skolemization.

Thus every SMT solver is benchmarked twice, with two different mechanisms for dealing with quantifiers: by eliminating them as described in Section 2 to yield a formula in the standard logic QF_UFBV of SMT-LIB, or by applying the non-standard quantification support of the various SMT solvers. Only in the case of Yices (which has only a limited support for quantification, namely exists forall problems—which is not sufficient for our purpose), we apply the alternative of expanding also (original) universal quantifiers. Actually, as demonstrated by Figure 1, Yices also yields good performance with this alternative mechanism. However, using this technique, none of the other three solver was able to perform *any* decision in the given time bound; therefore we do not consider this alternative for the other solvers any further.

3.3 Interpretation of the Results

A first rough inspection of the diagrams gives some initial insights:

- When eliminating quantifiers by Skolemization respectively expansion, Yices is mostly the fastest among the benchmarked SMT solvers; the other solvers are able to compete with Yices only if quantifiers are preserved in the formulas.
- For formulas with (mostly) valid base predicates (left diagrams), Yices performs better, when Skolemization is applied. However, for formulas with (mostly) unsatisfiable predicates and outermost existential quantifiers (left boundaries of the right diagrams), Yices performs better when eliminating all quantifiers by expansion.
- The semantic evaluation mechanism is mostly outperformed by Yices and also by the other solvers. However, the other solvers are superior only if the quantifiers are preserved in the formulas, or if we consider the cases in the middle of the left diagrams (few or no existential quantifiers and mainly valid base predicates).
- When quantifiers are eliminated, the semantic evaluation mechanism of RISCAL outperforms Boolector, CVC4, and Yices at the boundaries of the left diagrams (many either innermost or outermost existential quantifiers and mostly valid base predicates); in this case, also in the right diagrams (mostly unsatisfiable base predicates), the performance of semantic evaluation at least matches that of the solvers.

Now let us consider the semantic evaluation mechanism of RISCAL in more detail. In Figure 1, diagram *cycle4-valid* shows that this mechanism exhibits comparatively good performance for the quantification pattern $\exists^i\forall^j$. Since the outermost quantifier is existential, only a single value for its variable has to be found that makes the formula true; since the base predicate is valid, already the first choice is successful. The more existential quantifiers follow, i.e., the bigger i is, the bigger the advantage is. If all quantifiers are existential (case $\exists^4\forall^0$), the first attempted choice for all variables already leads to a decision of the formula. However, if more and more variables get universally quantified, the more and more work has to be performed to validate the existential choice. The worst situation arises, if all variables are universally quantified (case $\forall^4\exists^0$); here the full variable space has to be investigated to determine the validity of the formula.

However, the more of the inner variables get existentially quantified, the quicker the decision for each value of a universally quantified variable becomes. If only the outermost variable is universally quantified (case $\forall^1\exists^3$), only the space of the outermost variable has to be fully investigated. This explains the shape of the RISCAL curve which grows from the fully existentially quantified formula of type $\exists^4\forall^0$ until it reaches a sharp peak at the fully universally quantified formula of type $\forall^4\exists^0$; then the curve goes down again towards the formula pattern $\forall^1\exists^3$.

On the other hand, plot *cycle4-unsat* illustrates the dual behavior for the negated (unsatisfiable) version of the predicate. To show that the fully existentially quantified formula $\exists^4\forall^0$ is false, the whole value space has to be investigated, while for the fully universally quantified formula $\forall^4\exists^0$ the first encountered value combination represents a counterexample to the truth of the formula; for a growing number of inner existential quantifiers, again more value combinations have to be investigated, though.

Finally, the two plots for the mostly satisfiable predicate *cycle4-sat1* and its mostly unsatisfiable negation *cycle4-sat2* are similar to the plots for the valid and unsatisfiable cases, except that there is no more a pronounced "peak" (maximum or minimum) for the fully universally quantified pattern $\forall^4\exists^0$: the investigation of the value space can stop when the first counterexample is found, but not necessarily the first value combination encountered immediately represents such a counterexample.

While thus the behavior of the semantic evaluation mechanism of RISCAL can be well explained, the comparison with the various SMT-based decision mechanisms is more difficult. We have no expertise and insight into the implementations of the various SMT solvers, thus we could only speculate on the reasons of the various performance differences. However, from the benchmarks it becomes clear that for formulas with an existential search space of substantial size, the expansion of the (original) existential formula causes problems for Boolector, CVC4, and Z3; apparently their performance is comparatively poor for formulas that consist of a large number of clauses, such that the semantic evaluation mechanism of RISCAL is here competitive. Yices apparently deals better with such formulas; even more, in the case of mostly invalid base predicates, the performance of Yices even profits from the expansion of (original) universal quantifiers (i.e., the generation of clauses with many literals) more than from the use of Skolem constants respectively functions.

However, it is premature to draw conclusions only from a single type of benchmarks; therefore we are now going to investigate further ones.

3.4 More Valid and Unsatisfiable Predicates

We are now going to consider quantified formulas whose base predicates are a bit more complex than the one considered so far, starting with the following predicates that are all valid (such that their negations are unsatisfiable):

$$\begin{aligned}
\mathit{add4\text{-}valid} &\equiv x_1 + x_2 + x_3 + x_4 = 1 + x_4 + x_3 + x_2 + x_1 + x4 - 1 \\
\mathit{mult4\text{-}valid} &\equiv x_1 \cdot x_2 \cdot x_3 \cdot x_4 = (x_1 + 1) \cdot x_2 \cdot x_3 \cdot x_4 - x_4 \cdot x_3 \cdot x_2 \\
\mathit{max4\text{-}valid} &\equiv x_1 = M \vee x_2 = M \vee \ldots \Leftrightarrow M = \max\, x{:}D \text{ with } (x = x_1 \vee x = x_2 \vee \ldots).\ x \\
\mathit{set4\text{-}valid} &\equiv |\{x_1, x_2, x_3, x_4, M\}| = 1 \Rightarrow x_1 = M \wedge x_2 = M \wedge x_3 = M \wedge x_4 = M \\
\mathit{neq4\text{-}valid} &\equiv x_1 \neq M \wedge x_2 \neq x_1 \wedge x_2 \neq M \wedge \ldots \Rightarrow |\{x_1, x_2, x_3, x_4, M\}| = 5 \\
\mathit{array4\text{-}valid} &\equiv (\mathsf{Array}[4, D](x_1) \text{ with } [1] = x_2 \text{ with } [2] = x_3 \text{ with } [3] = x_4).[0] = x_1
\end{aligned}$$

Predicates *add4-valid* and *mult4-valid* investigate the influence of linear and non-linear arithmetic on the decision process. Predicate *max4-valid* investigates the behavior of the max quantifier (which in the SMT-LIB translation has to be encoded as a nested conditional term). Predicates *set4-valid* and *neq4-valid* investigate the influence of set constructions in combination with cardinality (which in the SMT-LIB translation is encoded by a sum of conditional expressions), once on the left and once on the right side of an implication. Predicate set4-valid investigates the combined effect of array creation, update, and lookup. These predicates thus cover a certain (non-exhaustive) range of types and operations supported by the RISCAL language.

Figure 2 presents the results of the corresponding benchmarks. Various of these diagram pairs resemble those of the top two diagrams of Figure 1; we only highlight the crucial observations.

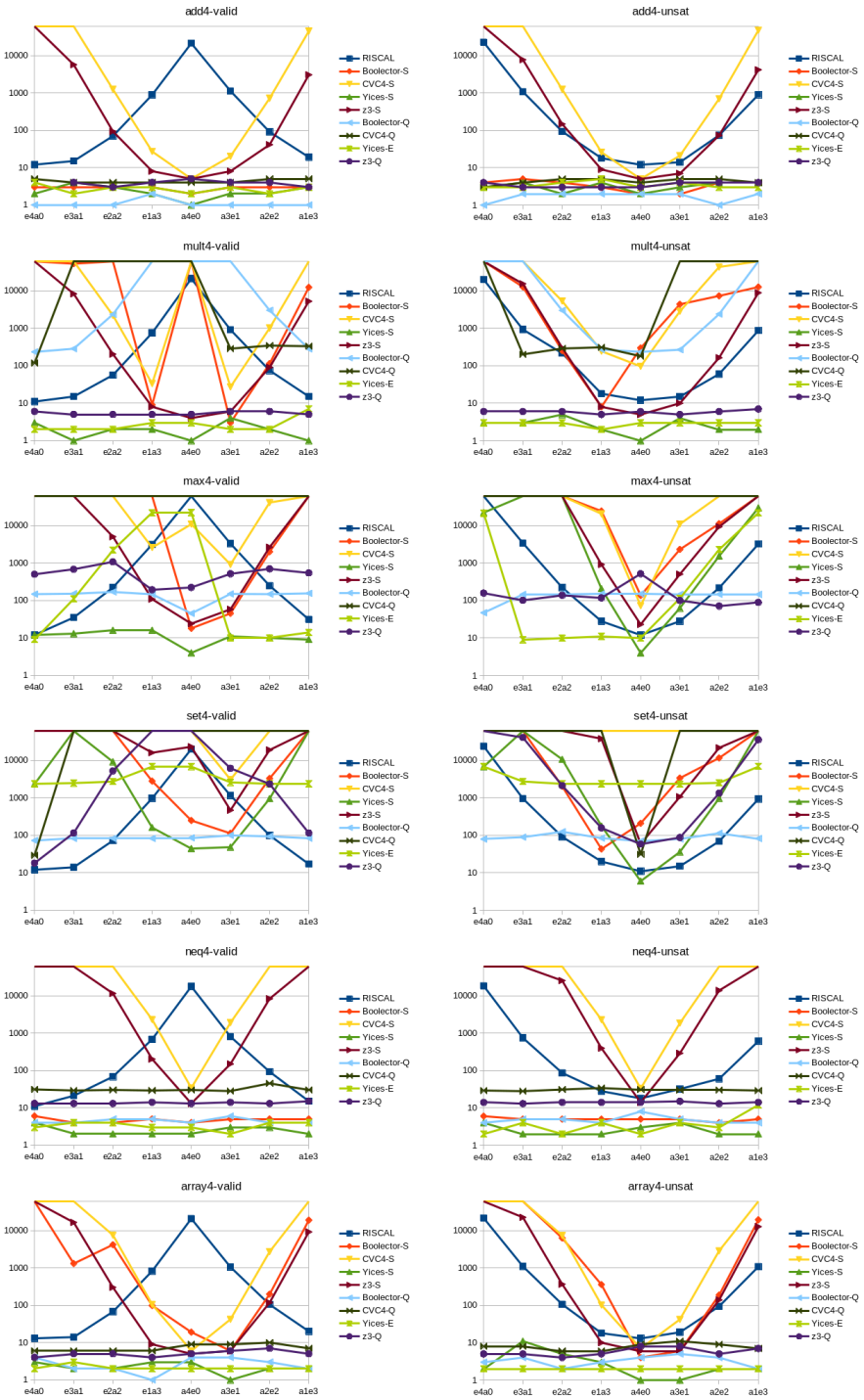


Figure 2: Artificial Benchmarks: Valid versus Unsatisfiable Predicates

- The curves for the RISCAL evaluation mechanism have preserved their shapes; this is not surprising, since the number of formula instances that have to be evaluated for valid/unsatisfiable base predicates does not change. For the SMT-based decision mechanisms, however, various differences are notable.
- For the linear arithmetic benchmarks *add4-valid* and *add4-unsat* and for the set example benchmarks *neq4-valid* and *neq4-sat*, Boolector performs for formulas with removed quantifiers now as well as when quantifiers are preserved. Actually, most SMT-based decision mechanisms perform very well in these cases, the only exceptions are here now CVC4 and Z3, when quantifiers are removed.
- However, in the non-linear arithmetic benchmarks *mult4-valid* and *mult4-unsat*, many SMT-based decision mechanisms perform poorly. Here the semantic evaluation mechanism of RISCAL is generally only outperformed by Yices and by Z3, if quantifiers are preserved.
- In the maximum-oriented benchmarks *max4-valid* and *max4-unsat*, the RISCAL evaluation mechanism performs in the ranges of its minima better than most SMT solvers, except for Yices.
- In the set-oriented benchmarks *set4-valid* and *set4-unsat*, Boolector is generally the fastest solver, if quantifiers are preserved; however also the RISCAL mechanism performs very well in the ranges of its minima, beating here many or even all solvers.
- In the array-oriented benchmarks *array4-valid* and *array4-unsat*, the semantic evaluation mechanism of RISCAL often beats three of the SMT solvers, if quantifiers are removed.

Thus above benchmarks reveal various situations when the semantic evaluation mechanism of RISCAL is able to compete with or even outperform some of the SMT solvers, especially with complex base operations such as non-linear arithmetic or set operations.

3.5 More Satisfiable but Not Valid Predicates

To further extend the domain of our investigations, we modify above predicates such that both the predicates and their negations are satisfiable:

$$
\begin{aligned}
\mathit{add4\text{-}sat1} &\equiv x_1 + x_2 + x_3 + x_4 = 1 + x_4 + x_3 + x_2 + x_1 - x_4 \\
\mathit{mult4\text{-}sat1} &\equiv x_1 \cdot x_2 \cdot x_3 \cdot x_4 = (x_1 + 1) \cdot x_2 \cdot x_3 \cdot x_4 - x_4 \cdot x_3 \cdot x_2 \cdot x_1 \\
\mathit{max4\text{-}sat1} &\equiv x_4 = \max x{:}D \text{ with } x = x_1 \vee x = x_2 \vee \ldots . \, x \\
\mathit{set4\text{-}sat1} &\equiv |\{x_1, x_2, x_3, x_4, M\}| = 4 \\
\mathit{neq4\text{-}sat1} &\equiv |\{x_1, x_2, x_3, x_4, M\}| = 5 \\
\mathit{set4\text{-}sat1} &\equiv (\mathsf{Array}[4, D](x_1) \text{ with } [1] = x_2 \text{ with } [2] = x_3 \text{ with } [3] = x_4).[0] = x_2
\end{aligned}
$$

As the benchmarks in Figure 3 demonstrate, now the clear-cut superiority of the SMT-based decisions over the semantic evaluation mechanism of RISCAL has vanished for most benchmarks; in particular the prominent "spike" exhibited for the quantifier pattern $\forall^4\exists^0$ does not appear

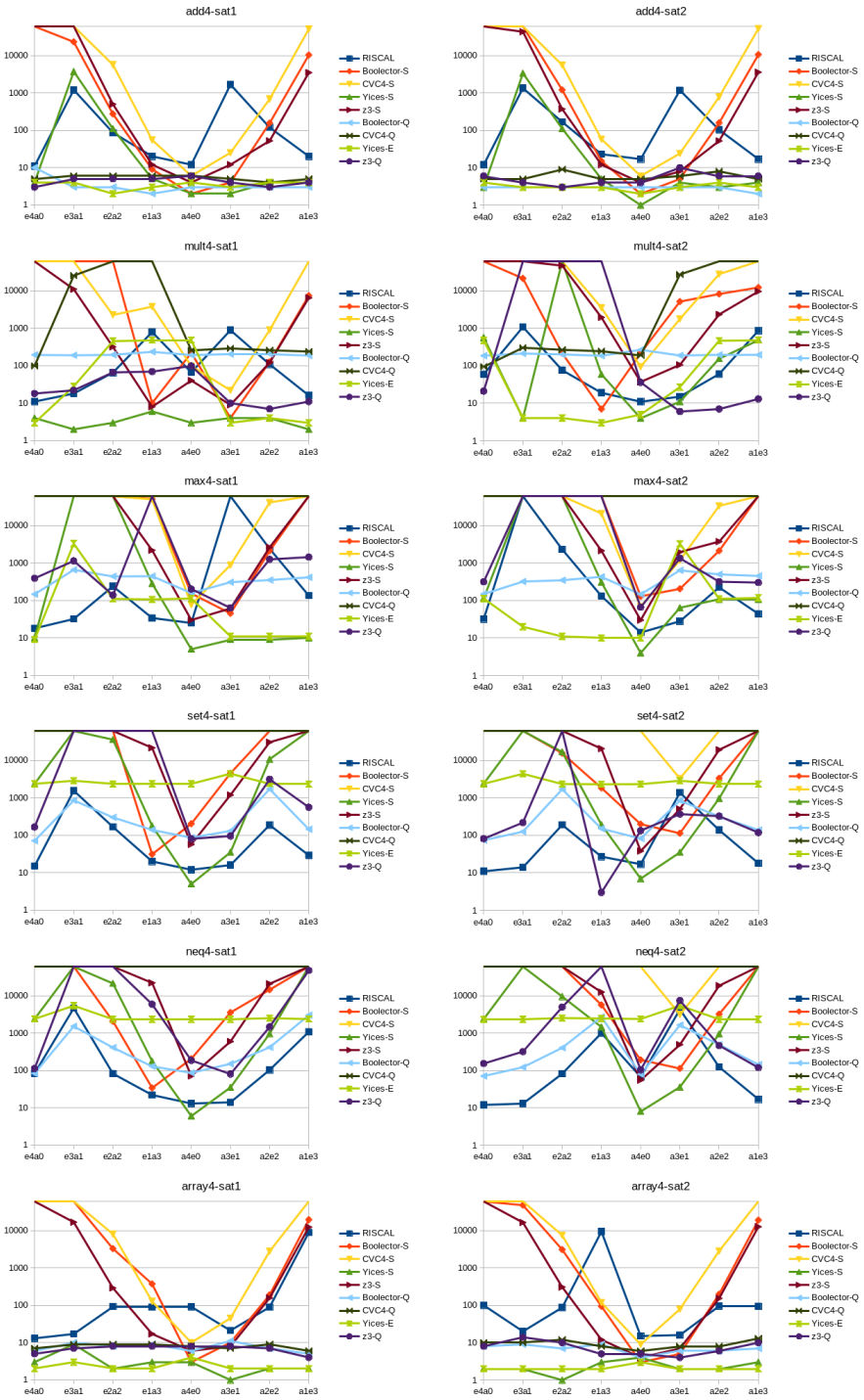


Figure 3: Artificial Benchmarks: Satisfiable Predicates and their Negations

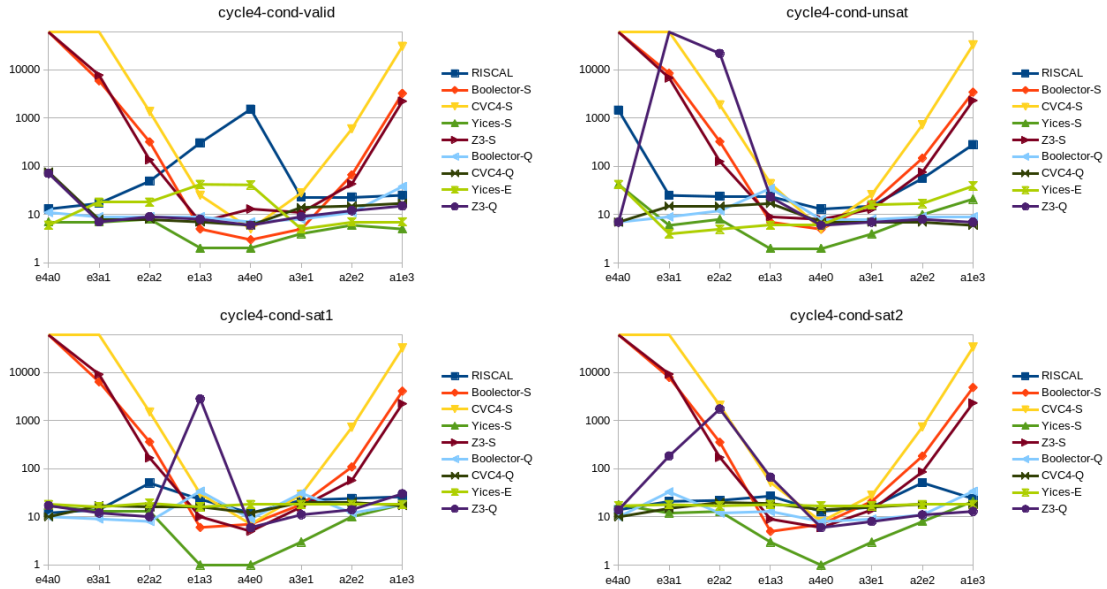


Figure 4: Additional Conditions: Base Behavior

for a mostly (but not fully) valid base predicate. Even more, in the cases of the set-based benchmarks *set4-sat1*, *set4-sat2*, *neq4-sat1* and *neq4-sat2*, the semantic evaluation mechanism is often the most efficient one.

3.6 Additional Conditions

Now we investigate the effect of additional conditions on the domains of the various variables. For this, we annotate every quantified formula $(Qx_i{:}D.\ \sqcup)$ as $(Qx_i{:}D \text{ with } x_i > b_i.\ \sqcup)$; we define $b_i := 0$ for $i = 0$ and $b_i := x_{i-1}$ for $i > 0$. In case of quantifier $Q := \exists$, this formula is interpreted as the existentially quantified conjunction $(\exists x_i{:}D.\ x_i > b_i \wedge \sqcup)$; in case of $Q := \forall$, it is interpreted as the universally quantified implication $(\forall x_i{:}D.\ x_i > b_i \Rightarrow \sqcup)$. These annotations thus introduce logical connectives into the formulas that considerably restrict the size of that portion of the value space that is relevant for the truth of the formula; however, formulas with valid base predicates remain true and formulas with unsatisfiable predicates remain false. In case of satisfiable but not valid base predicates, the annotations may change the truth value of the resulting formulas, though.

Figure 4 describes the effect of these additional conditions on the "base behavior" presented in Figure 1. We see that the RISCAL evaluation mechanism considerably profits: for a valid base predicate, the peak of the curve for the quantifier pattern $\forall^4\exists^4$ is now much less pronounced; for an unsatisfiable predicate, the curve is very much flattened. For the two satisfiable but not valid predicates, the curve is almost flat with values that compete with most of the SMT-based decision mechanisms.

Similar improvements can be also observed for the other base predicates, as illustrated in Fig-

ure 5 and Figure 6 (compare with Figure 2 and Figure 3). In many cases, the RISCAL evaluation mechanism is competitive with that of the various SMT solvers, often even with that of Yices and of SMT solvers that are directly applied to quantified formulas; in numerous situations it outperforms many solvers. For the set-based predicates *neq4-sat1* and *set4-sat1*, respectively their negation, it is in general the most efficient mechanism.

3.7 Functions Specified by Contracts

So far, we have only considered formulas with built-in operations (functions and predicates). Indeed, the experimental results remain essentially the same if formulas also contain user-defined operations: since both RISCAL and SMT-LIB directly support explicit function and predicate definitions, this does not pose any challenges for the decision.

The situation, however, fundamentally changes if we also consider operations specified by contracts, i.e., essentially by formulas that constrain the result of the operation. As an example, we will consider the following two functions:

fun $f(x_1, x_2, x_3, x_4)$ ensures
  if $x_1 < x_2 \wedge x_2 < x_3 \wedge x_3 < x_4 \wedge x_4 < x_1$ then $result = 0$ else $result = 1$;
fun $g(x_1, x_2, x_3, x_4)$ ensures
  if $x_1 = x_2 \wedge x_3 = x_4$ then $result = 0$ else $result = 1$;

Here $f(x_1, x_2, x_3, x_4)$ is 1 for all x_1, x_2, x_3, x_4 and $g(x_1, x_2, x_3, x_4)$ may be 1 or 0, depending on the values of x_1, x_2, x_3, x_4. We will consider quantified formulas with the quantification patterns used in the previous sections, using the following base predicates:

$$\begin{aligned} \textit{choose4-valid} &\equiv f(x_1, x_2, x_3, x_4) = 1 \\ \textit{choose4-unsat} &\equiv f(x_1, x_2, x_3, x_4) = 0 \\ \textit{choose4-sat1} &\equiv g(x_1, x_2, x_3, x_4) = 1 \\ \textit{choose4-sat2} &\equiv g(x_1, x_2, x_3, x_4) = 0 \end{aligned}$$

Figure 7 displays the decision times for model parameter $N := 5$ (we now use value 5 rather than 6 to compensate the addition quantifier of the function axiom). The evaluation mechanism of RISCAL is generally faster than the SMT solvers (indeed Boolector and CVC4 do mostly not deliver any answers within the given time bound); the major exception is Z3 if quantifiers are preserved, which is faster than RISCAL for valid and unsatisfiable formulas. Yices also produces results but is mostly much slower than RISCAL. Boolector does not support uninterpreted functions if quantifiers are preserved, thus also the benchmark set "Boolector-Q" expands (as "Boolector-S" does) universal and existential quantifiers to conjunctions and disjunctions, respectively.

The results demonstrate that uninterpreted functions characterized by axioms rather than definitions pose a major problem for most SMT solvers; in the presence of such functions often the nondeterministic evaluation mechanism of RISCAL is superior. In an attempt to mitigate this problem a bit, we have also implemented in RISCAL the options "eliminate choices" and "inline definitions" to eliminate certain applications of contract-specified operations by embedding the

Figure 5: Additional Conditions: Valid versus Unsatisfiable Predicates

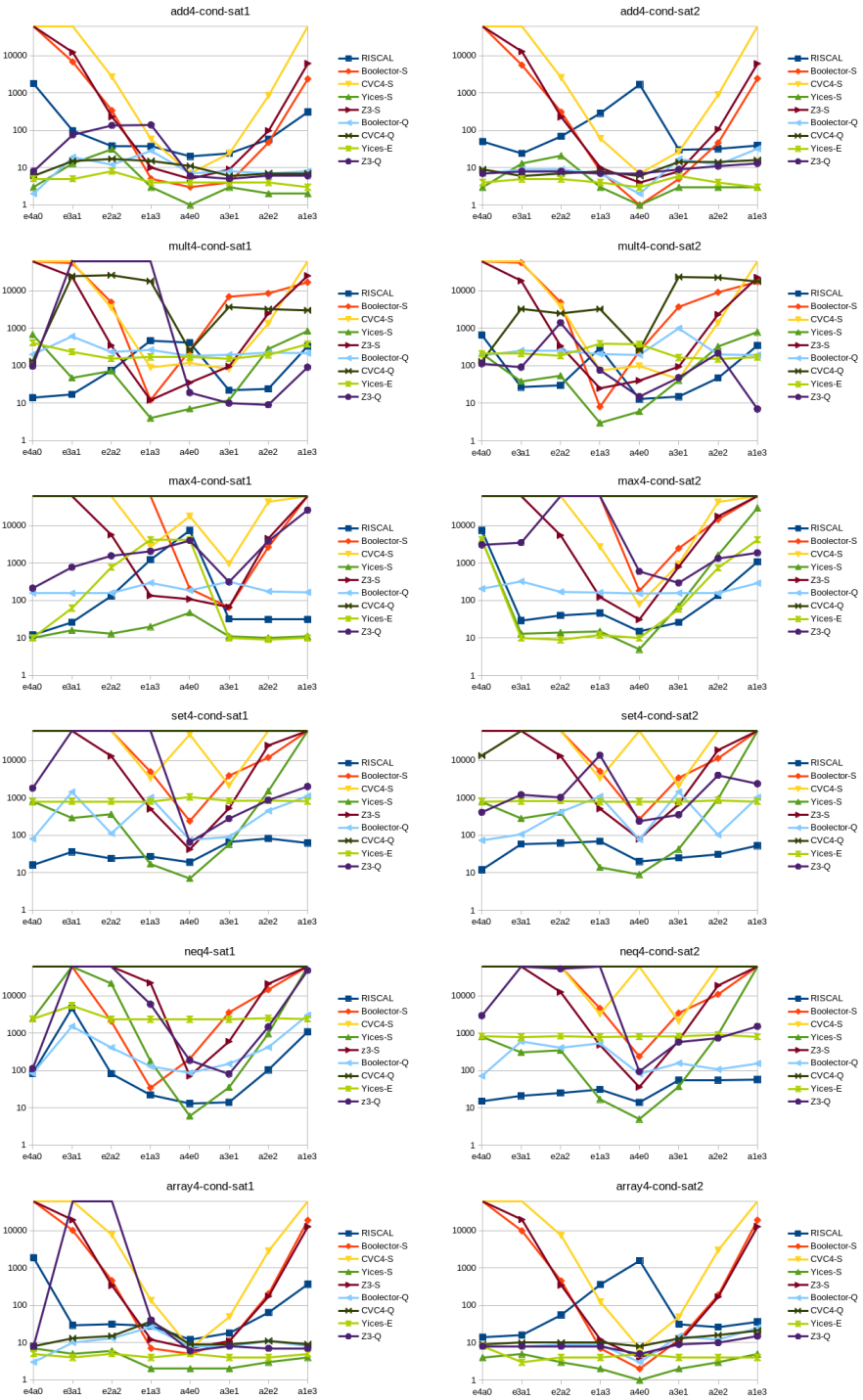


Figure 6: Additional Conditions: Satisfiable Predicates and their Negations

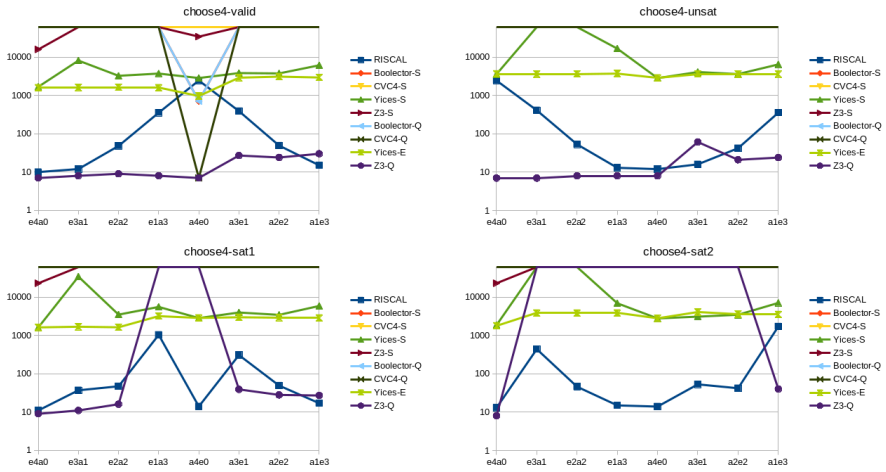


Figure 7: Formulas with Functions Specified by Contracts

postconditions into the enclosing formulas; this does not change the semantics of the formula, if the application occurs in a non-negated purely universally quantified context. In our experiments, this is the case (only) for the quantifier pattern `a4e0` ($\forall^4\exists^0$) which indeed shows substantial speedups (only) with the application of Yices; the experimental results given above have been derived with these options.

4 Real-Life Benchmarks

So far we have investigated artificially generated benchmark examples that were especially designed to exhibit certain effects of the competing decision mechanisms; however, this does not demonstrate whether/how often these effects indeed emerge in "real-life" examples. To shed some light on this issue, we have also collected formulas from a number of RISCAL models. These formulas mainly arise from the specification and verification of algorithms in computer science, set theory, discrete mathematics, algebra, or logic; in particular, they represent conditions to validate the specifications and verify the correctness of the algorithms or also theorems over the domains of consideration. We want to emphasize again that in the overwhelming majority of cases the decision of such conditions by SMT solving vastly outperforms the decision by semantic evaluation. However, for the purpose of this paper, we have selected a sample of those conditions where this is not the case, i.e., where semantic evaluation is competitive with (or even outperforms) SMT solving. Thesese benchmarks are available from the following URL:

> https://www.risc.jku.at/research/formal/software/RISCAL/papers/EvalSMT2021-models.tgz

Figure 8 presents the results of benchmarking the decision of these formulas; the left column illustrates execution of the benchmarks with smaller values for the model parameters (leading to smaller variable domains), while the right column illustrates executions of the same benchmarks

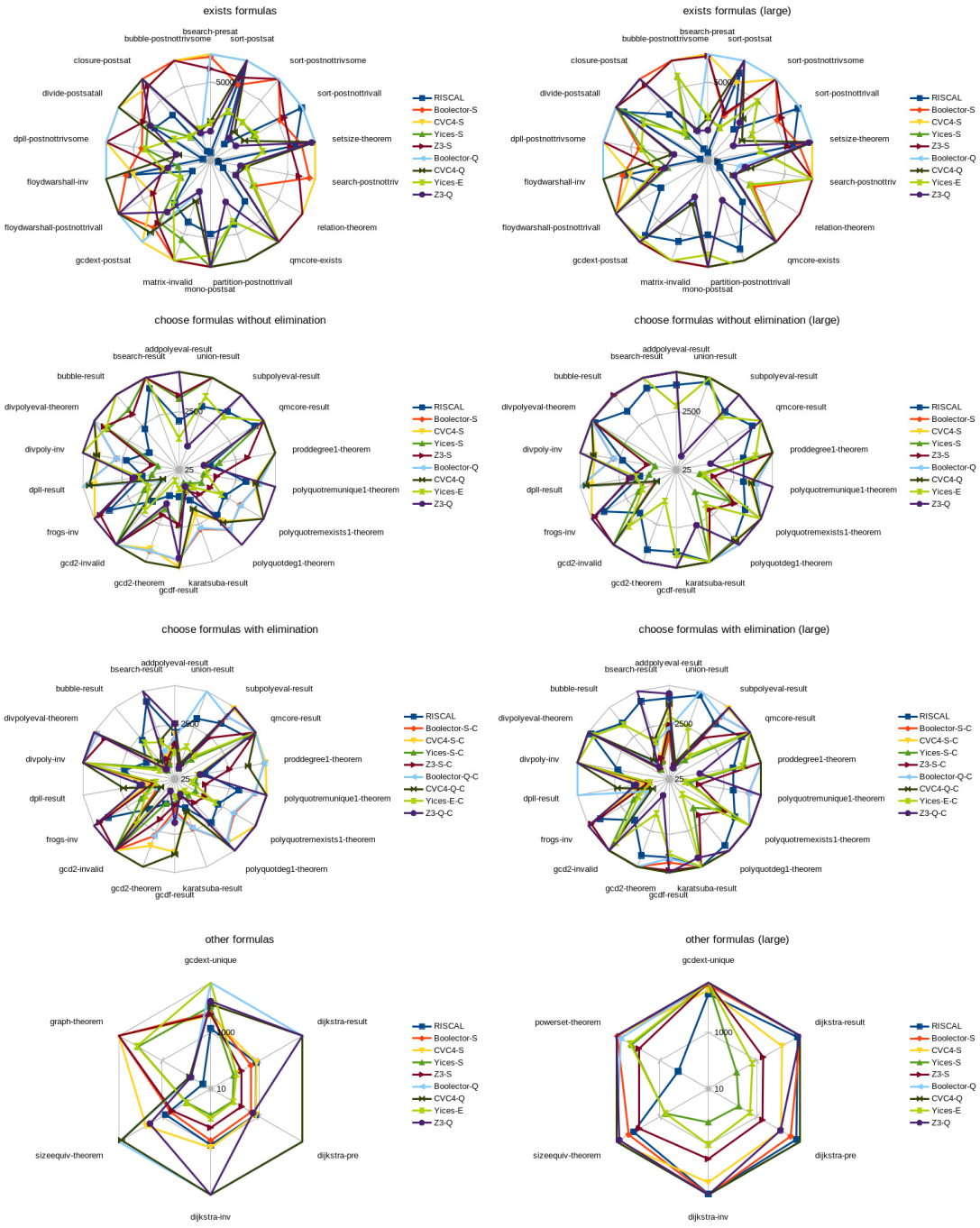


Figure 8: Real-Life Benchmarks

with larger values; all executions, however, were terminated after 60 seconds. Each benchmark is visualized as a circular "net" chart where each radial represents one formula whose validity is to be decided, the center represents some minimal time and the outermost rim of the net represents the 60 seconds timeout limit. Therefore, the closer the lines connecting the benchmark points of a particular decision mechanism are to the center of the net, the faster the decision mechanism is; a line along the outermost rim indicates timeout situations. As in the artificial benchmarks, the values along the radials are plotted in logarithmic magnitudes. In more detail, the results can be interpreted as follows:

- The uppermost row of the figure illustrates benchmarks for formulas that have substantial "existential" content; many (not all) of these formulas stem from validating procedure contracts by checking the satisfiability of the postconditions for all inputs that satisfy the preconditions; here we see that the semantic evaluation mechanism of RISCAL often outperforms the various SMT solvers. The best SMT results are achieved by Z3 when preserving quantifiers (Z3-Q) and Yices with either Skolemization or expansion of the (original) universal quantifiers (Yices-S and Yices-E).
- The second row illustrates benchmarks for formulas that involve choose expressions; here in the SMT decision the options "choose elimination" and "inline definitions" were *not* applied. In the smaller models (left diagram) the evaluation mechanism of RISCAL again outperforms many of the SMT solvers; among these typically Yices performs best. In the larger models (right diagram), RISCAL is more often beaten by Yices but also yields results when most of the other solvers run into timeouts.
- The third row illustrates benchmarks for the same formulas as in the second row but with the options "choose elimination" and "inline definitions" switched on. This shows a clear improvement in many examples, where now various SMT solvers clearly beat the semantic evaluation mechanism of RISCAL; still the benchmarks for the SMT solvers Boolector and CVC4 can be found mostly along the outer "timeout" rim.
- The last row illustrates benchmarks for a couple of formulas that do not clearly fall into above categories but where nevertheless the semantic evaluation mechanism of RISCAL is competitive. These are examples that involve some non-linear arithmetic or set-theoretical notions (cardinality) or deeply nested and hard to analyze conditions arising from the verification of Dijkstra's algorithm.

Generally speaking, among all the SMT solvers Yices (typically with expansion of existential formulas rather than Skolemization) performed best, variously beaten by Z3 when quantifiers were preserved.

5 Conclusions

Generally the decision of first-order formulas over finite domains by external SMT solvers via a translation into the SMT-LIB logic QF_UFBV (unquantified formulas over bit vectors with uninterpreted sort and function symbols) and applying external SMT solvers vastly outperforms

the semantic evaluation mechanism built-in into RISCAL. In this report, however, we have also identified cases where this is not necessarily the case, mainly because the SMT-LIB translation leads to a large number of clauses in the generated conjunctive normal form.

One such case are theorems that have a substantial "existential" content, i.e., positive (unnegated) occurrences of existential quantifiers with large quantification ranges. Apart from the fact, that then the generation of the translation may take some time and the resulting formulas may become huge, their decision by SMT solving may be outperformed by straight-forward semantic evaluation.

Another case are theorems that involve uninterpreted functions that are axiomatized by universally quantified formulas with large quantification ranges; also these axioms lead to the generation of SMT-LIB formulas with a huge number of clauses that slow down the execution of SMT solvers. This problem, however, may be partially mitigated, if applications of such functions occur in a pure universal context; an optimization technique may replace the function application by universally quantified variables that are constrained by an appropriate instance of this axiom.

Furthermore, also for universally quantified theorems the problem arises that the Skolem functions generated from their negated counterparts have to be constrained by axioms that describe which bit vector values indeed describe valid RISCAL values. RISCAL therefore implements an SMT-LIB option that applies a heuristic to decide whether it is cheaper to expand the quantified formula rather than to generate a Skolem function.

Another problem may result from the necessary encoding of various operations on the data types supported by RISCAL (such as non-linear arithmetic, arithmetic quantifiers, or set size computations) where the built-in evaluation mechanisms of RISCAL may perform better than the corresponding RISCAL encodings; however, while this effect can be observed in artificial benchmarks, it seems not to be a major problem in real-life examples.

Among the investigated SMT solvers (Boolector, CVC4, Yices, Z3), often Yices performed best; interestingly the performance of Yices can be generally improved by expanding universal quantifiers rather than generating Skolem functions from their negated counterparts. The application of Z3 often benefits from the preservation of quantifiers in the SMT-LIB formulas, employing the non-standard quantification support of Z3 for the theory QF_UFBV (also Boolector provides such support, but only for the theory QF_BV without uninterpreted functions, which limits the usefulness of this support for our application scenario).

Our work demonstrates that there is still room for improvement in current SMT solvers for the SMT-LIB logic QF_UFBV with respect to deciding theorems with substantial existential content and applications of axiomatized functions. On the other side, we will continue to investigate how the translation of RISCAL formulas to SMT-LIB formulas can be optimized to take the presented findings into account.

References

[1] Jean-Raymond Abrial. *Modeling in Event-B — System and Software Engineering*. Cambridge, UK: Cambridge University Press, May 2010. URL: http://www.cambridge.org/

at/academic/subjects/computer-science/programming-languages-and-applied-logic/modeling-event-b-system-and-software-engineering?format=HB.

[2] Wolfgang Ahrendt et al., eds. *Deductive Software Verification — The KeY Book — From Theory to Practice*. Vol. 10001. Lecture Notes in Computer Science. Cham, Switzerland: Springer International Publishing, 2016. DOI: 10.1007/978-3-319-49812-6.

[3] Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. "The Spec# Programming System: An Overview". In: *CASSIS 2004: Construction and Analysis of Safe, Secure, and Interoperable Smart Devices, International Workshop, Marseille, France, March 10–14, 2004*. Ed. by Gilles Barthe et al. Vol. 3362. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2005, pp. 49–69. DOI: 10.1007/978-3-540-30569-9_3.

[4] Mike Barnett et al. "Boogie: A Modular Reusable Verifier for Object-Oriented Programs". In: *FMCO 2005: Formal Methods for Components and Objects, 4th International Symposium, November 1–4, 2005*. Ed. by Frank S. de Boer et al. Vol. 4111. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2005, pp. 364–387. DOI: 10.1007/11804192_17.

[5] Clark W. Barrett et al. "CVC4". In: *CAV 2011: Computer Aided Verification, 23rd International Conference, 2011, Snowbird, UT, USA, July 14–20, 2011*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Vol. 6806. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2011, pp. 171–177. DOI: 10.1007/978-3-642-22110-1_14.

[6] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB)*. 2016. URL: http://www.SMT-LIB.org.

[7] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development — Coq'Art: The Calculus of Inductive Constructions*. Berlin, Germany: Springer, 2016. URL: https://doi.org/10.1007/978-3-662-07964-5.

[8] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. "Extending Sledgehammer with SMT Solvers". In: *CADE-23: Automated Deduction, 23rd International Conference, Wrocław, Poland, July 31–August 5, 2011*. Ed. by Nikolaj Bjørner and Viorica Sofronie-Stokkermans. Vol. 6803. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2011, pp. 116–130. DOI: 10.1007/978-3-642-22438-6_11.

[9] Jasmin Christian Blanchette and Tobias Nipkow. "Nitpick: A Counterexample Generator for Higher-Order Logic Based on a Relational Model Finder". In: *ITP 2010: Interactive Theorem Proving, First International Conference, Edinburgh, UK, July 11-14, 2010*. Ed. by Matt Kaufmann and Lawrence C. Paulson. Vol. 6172. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2010, pp. 131–146. DOI: 10.1007/978-3-642-14052-5_11.

[10] François Bobot et al. "Why3: Shepherd Your Herd of Provers". In: *Boogie 2011: First International Workshop on Intermediate Verification Languages*. Wrocław, Poland, Aug. 2011, pp. 53–64. URL: http://proval.lri.fr/publications/boogie11final.pdf.

[11] Patrice Chalin et al. “Beyond Assertions: Advanced Specification and Verification with JML and ESC/Java2”. In: *Formal Methods for Components and Objects 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1–4, 2005*. Ed. by Frank S. de Boer et al. Vol. 4111. Lecture Notes in Computer Science. Springer, Berlin, Germany, 2006, 342–363. DOI: 10.1007/11804192_16.

[12] David R. Cok. “OpenJML: JML for Java 7 by Extending OpenJDK”. In: *NFM 2011: NASA Formal Methods, Third International Symposium, Pasadena, CA, USA, April 18–20, 2011*. Ed. by Mihaela BobaruKlaus HavelundGerard J. HolzmannRajeev Joshi. Vol. 6617. Lecture Notes in Computer Science. Springer, Berlin, Germany, 2011, 472–479. DOI: 10.1007/978-3-642-20398-5_35.

[13] David Déharbe et al. “SMT Solvers for Rodin”. In: *ABZ 2012: Abstract State Machines, Alloy, B, VDM, and Z, Third International Conference, Pisa, Italy, June 18–21, 2012*. Ed. by John Derrick et al. Vol. 7316. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2012, pp. 194–207. DOI: 10.1007/978-3-642-30885-7_14.

[14] Bruno Dutertre. “Yices 2.2”. In: *CAV 2014: Computer-Aided Verification, Vienna, Austria, July 18–22, 2014*. Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2014, pp. 737–744. DOI: 10.1007/978-3-319-08867-9_49.

[15] Burak Ekici et al. “SMTCoq: A Plug-In for Integrating SMT Solvers into Coq”. In: *CAV 2017: Computer Aided Verification, Part II, 29th International Conference, Heidelberg, Germany, July 24–28, 2017*. Ed. by Rupak Majumdar and Viktor Kunčak. Vol. 10427. Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2017, pp. 126–133. DOI: 10.1007/978-3-319-63390-9_7.

[16] Aboubakr Achraf El Ghazi and Mana Taghdiri. “Analyzing Alloy Formulas using an SMT Solver: A Case Study”. In: *AFM10: Automated Formal Methods, July 14, 2010, Edinburgh, UK*. 2015. URL: https://arxiv.org/abs/1505.00672.

[17] Andrei Voronkov Giles Reger Martin Suda. “Instantiation and Pretending to be an SMT Solver with Vampire”. In: *SMT 2017 Workshop, Heidelberg, Germany, July 22–23, 2017*. Vol. 1889. CEUR Workshop Proceedings. 2017, pp. 63–75. URL: http://ceur-ws.org/Vol-1889/paper6.pdf.

[18] Daniel Jackson. “Automating First-Order Relational Logic”. In: *SIGSOFT ’00/FSE-8: Foundations of Software Engineering: Twenty-First Century Applications, 8th ACM SIGSOFT International Symposium, San Diego, California, USA, November 2000*. Vol. 25. SIGSOFT Software Engineering Notes 6. New York, NY, USA: ACM, 2000, pp. 130–139. DOI: 10.1145/355045.355063.

[19] Daniel Jackson. *Software Abstractions — Logic, Language, and Analysis*. Revised. Cambridge, MA, USA: MIT Press, Nov. 2011. URL: https://mitpress.mit.edu/books/software-abstractions.

[20] Florent Kirchner et al. “Frama-C: A Software Analysis Perspective”. In: *Formal Aspects of Computing* 27 (2015), pp. 573–609. DOI: 10.1007/s00165-014-0326-7.

[21] K. Rustan M. Leino. “Dafny: An Automatic Program Verifier for Functional Correctness”. In: *LPAR-16: Logic Programming and Automated Reasoning, 16th International Conference, Dakar, Senegal, April 25–May 1, 2010*. Ed. by Edmund M. Clarke and Andrei Voronkov. Vol. 6355. Lecture Notes in Computer Science. Springer, Berlin, Germany, 2010, 348–370. DOI: 10.1007/978-3-642-17511-4_20.

[22] Hsin-Hung Lin and Bow-Yaw Wang. “Releasing VDM Proof Obligations with SMT Solvers”. In: *MEMOCODE ’17: 15th ACM-IEEE International Conference on Formal Methods and Models for System Design, Vienna, Austria, September 29–October 2, 2017*. New York, NY, USA: ACM, 2017, pp. 132–135. DOI: 10.1145/3127041.3127066.

[23] Stephan Merz and Hernán Vanzetto. “Encoding TLA$^+$ into Many-Sorted First-Order Logic”. In: *ABZ 2016: Abstract State Machines, Alloy, B, TLA, VDM, and Z: 5th International Conference, Linz, Austria, May 23–27, 2016*. Ed. by Michael Butler et al. Vol. 9675. Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2016, pp. 54–69. DOI: 10.1007/978-3-319-33600-8_3.

[24] Leonardo de Moura and Nikolaj Bjørner. “Z3: An Efficient SMT Solver”. In: *TACAS 2008: Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, Budapest, Hungary, March 29–April 6, 2008*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Vol. 4963. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

[25] Aina Niemetz, Mathias Preiner, and Armin Biere. “Boolector 2.0”. In: *Journal on Satisfiability, Boolean Modeling and Computation* 9.1 (2014), pp. 53–58. DOI: 10.3233/sat190101.

[26] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Berlin, Germany: Springer, Dec. 2016. URL: https://isabelle.in.tum.de/doc/tutorial.pdf.

[27] *Overture Tool — Formal Modelling in VDM*. Nov. 2020. URL: http://overturetool.org.

[28] Franz-Xaver Reichl. “The Integration of SMT Solvers into the RISCAL Model Checker”. MA thesis. Johannes Kepler University Linz, Austria: Research Institute for Symbolic Computation (RISC), Apr. 2020. URL: https://www.risc.jku.at/publications/download/risc_6103/Thesis.pdf.

[29] Wolfgang Schreiner. “Logic and Semantic Technologies for Computer Science Education”. In: *Informatics’2019, 2019 IEEE 15th International Scientific Conference on Informatics*. Poprad, Slovakia, November 20–22: IEEE, 2019, pp. 415–420. DOI: 10.1109/Informatics47936.2019.9119285.

[30] Wolfgang Schreiner. *The RISC Algorithm Language (RISCAL)*. Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria. 2019. URL: https://www.risc.jku.at/research/formal/software/RISCAL.

[31] Wolfgang Schreiner. *The RISC Algorithm Language (RISCAL) — Tutorial and Reference Manual (Version 1.0)*. Technical Report. Available at [30]. Johannes Kepler University, Linz, Austria: RISC, Mar. 2017.

[32] Wolfgang Schreiner. "Theorem and Algorithm Checking for Courses on Logic and Formal Methods." In: *Post-Proceedings ThEdu'18, Theorem proving components for Educational software, Oxford, United Kingdom, July 18, 2018*. Ed. by Pedro Quaresma and Walther Neuper. Vol. 290. EPTCS. 2019, pp. 56–75. DOI: 10.4204/EPTCS.290.5.

[33] Wolfgang Schreiner. "Validating Mathematical Theories and Algorithms with RISCAL". In: *CICM 2018, 11th Conference on Intelligent Computer Mathematics, Hagenberg, Austria, August 13–17*. Ed. by F. Rabe et al. Vol. 11006. Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence. Springer, Berlin, 2018, pp. 248–254. DOI: 10.1007/978-3-319-96812-4_21.

[34] Wolfgang Schreiner, Alexander Brunhuemer, and Christoph Fürst. "Teaching the Formalization of Mathematical Theories and Algorithms via the Automatic Checking of Finite Models". In: *Post-Proceedings ThEdu'17, Theorem proving components for Educatio nal software, Gothenburg, Sweden, August 6, 2017*. Ed. by Pedro Quaresma and Walther Neuper. Vol. 267. EPTCS. 2018, pp. 120–139. DOI: 10.4204/EPTCS.267.8.

[35] Wolfgang Schreiner and Franz-Xaver Reichl. "Mathematical Model Checking Based on Semantics and SMT". In: *Transactions on Internet Research* 16.2 (July 2020), pp. 4–13. URL: http://ipsitransactions.org/journals/papers/tir/2020jul/p2.pdf.

[36] *SMT-COMP: The International Satisfiability Modulo Theories (SMT) Competition.* May 2021. URL: https://smt-comp.github.io.

[37] Emina Torlak and Daniel Jackson. "Kodkod: A Relational Model Finder". In: *TACAS 2007: Tools and Algorithms for the Construction and Analysis of Systems, 3th International Conference, Braga, Portugal, March 24–April 1, 2007*. Ed. by Orna Grumberg and Michael Huth. Vol. 4424. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2007, pp. 632–647. DOI: 10.1007/978-3-540-71209-1_49.