First-Order Logic in Finite Domains: Where Semantic Evaluation Competes with SMT Solving

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

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 approach of "semantic evaluation" (based on an implementation of the denotational semantics of the RISCAL language) and, second, the later approach of SMT solving (based on satisfiability preserving translations of RISCAL formulas to SMT-LIB formulas as inputs for SMT solvers). 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, we identify 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.

1 Introduction

The aim of the RISCAL system [16] is to support the analysis of theories and algorithms over discrete domains, as they 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 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 which are configurable by model parameters. Therefore, 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. The system has been mainly developed for educational purposes [15] but it has also been applied in research [19].

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 [14, 17]. In this approach, the decision of a RISCAL formula is performed via a translation to a formula in the SMT-LIB language [3] and the application of some external SMT solver (currently, the SMT solvers Boolector, CVC4, Yices,

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

T. Kutsia (Ed.): Symbolic Computation
in Software Science (SCSS'21)
EPTCS 342, 2021, pp. 99–113, doi:10.4204/EPTCS.342.9

and Z3 are supported). Indeed, this has achieved great performance improvements [14] and allowed to constructively work with theories that were out of reach of semantic evaluation.

Actually, SMT solvers have served for a long time as backends of various program verification tools, for real languages such as Java [1] as well as for algorithmic languages such as Dafny [10] where the verification backend Boogie [2] 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 , Event-B, and VDM. Last but not least, they are employed as backends for interactive provers, e.g., in Isabelle's "sledgehammer" component [4] and in Coq's SMTCoq plugin [7].

As for the translation of higher-level specification languages into the languages of satisfiability solvers, [9] describes the techniques used in the Alloy Analyzer to transform formulas from first-order relational logic; [21] discusses improvements implemented in the SAT-based relational model finder Kodkod. On top of Kodkod, the counterexample generator Nitpick [5] generates finite countermodels of Isabelle formulas by a translation to relational logic. In [8], it is briefly sketched how Alloy constraints have been (manually) translated into the language of the SMT solver Yices, which shows drastic speedups when tautologies are decided. [11] sketches the encoding of VDM proof obligations as SMT problems proved with the SMT solver Z3. In somewhat more detail, [6] discusses the implementation of a SMT plugin for the Event-B platform Rodin and experimentally compares this plugin with those 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 [14], 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 [20], it is harder to find comparisons with alternative decision mechanisms such as the one with the Vampire prover presented in [13].

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 and in SMT technology in general. While our insights are clearly limited to the particular strategy applied for translating RISCAL formulas to SMT-LIB, our work shows that SMT is not a panacea in all kinds of reasoning problems but has to be applied with some caveats.

Closest to our work is [12], where the untyped first-order logic of Lamport's language TLA$^+$ 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. As another difference, the TLA$^+$ translation heavily relies on a nonconstructive encoding of non-integer values by uninterpreted sorts and functions with corresponding background axioms. In contrast to this, the RISCAL translation generates 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.

The remainder of this paper is organized as follows: In Section 2, we outline the decision mechanisms applied in RISCAL. In Section 3, we present the artificial benchmarks which we use to compare both mechanisms. 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 RISCAL and SMT technology. Appendix A includes detailed illustrations of the benchmark results. More details can be found in the technical report [18] on which this paper is based. Due to space restrictions, we have to refer the reader to [16] for an overview on the RISCAL language which is elucidated by a tutorial and reference manual and various publications.

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.

Semantic Evaluation The built-in decision mechanism of RISCAL is based on the translation of every syntactic phrase of the RISCAL language into an executable representation of its denotational semantics. 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 [17]. In the case of first-order logic formulas, the most interesting part of the translation is that of a universally quantified formula $\forall x\colon D.\ F$ and that of an existentially quantified formula $\exists x\colon D.\ F$, respectively; 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]] :=
  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]] :=
  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* or a *false existential formula*: here we have to generate all elements, before we can decide that the universal formula is true or the existential formula is false.

Furthermore, RISCAL supports expressions that do not denote unique values, for example the term (choose $x\colon D$ 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 [17] by the computation of a (lazily evaluated) *stream* of such values. 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"). Furthermore, applications of such expressions may emerge 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

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

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

SMT Solving The problem of deciding the validity of a formula F, denoted as $valid[\![F]\!]$, can be reduced to the problem of deciding the satisfiability of the negation of F, denoted as $sat[\![\neg F]\!]$, by applying the equivalence. $valid[\![F]\!] \equiv \neg sat[\![\neg F]\!]$. RISCAL implements a translation to formulas in the SMT-LIB format [3]; thus we can decide the validity of the RISCAL formula F by letting an external SMT solver decide the satisfiability of the SMT-LIB version of $\neg F$. As a background theory we have chosen the theory of *fixed-size bit vectors*: since every RISCAL domain is finite, every element of a domain with n elements can be represented by a vector of $\lceil \log n \rceil$ bits; furthermore the set of bit vector operations 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*. Furthermore, 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. We therefore have to translate a RISCAL formula with quantifiers into a corresponding quantifier-free SMT-LIB formula (since some SMT solvers actually support as a non-standard extension also quantified bit vector formulas with uninterpreted functions, we will in the benchmarks later also experiment with the preservation of quantifiers).

The problem of eliminating quantifiers is addressed by the following equivalences which semi-formally sketch how quantifiers can be removed from RISCAL formulas (the role of function f is explained below):

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

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 decision of $valid[\![\exists x : D.\ F[x]]\!]$ 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 conjunction $\neg F[e_1] \wedge \ldots \wedge \neg F[e_n]$. On the other hand, the decision of $valid[\![\forall x : D.\ F[x]]\!]$ 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 disjunction $\neg F[e_1] \vee \ldots \vee \neg F[e_n]$. However, for this kind of decision 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 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 *equi-satisfiable*, 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 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, problems may arise even with the decision of the validity of a universally quantified formula, which entails deciding the satisfiability of a formula $\neg F[f(x_1,\ldots,x_m)]$ with Skolem function f (please note that $x_1,\ldots,x_m$ represent concrete values, the translated formula does not have any free variables). In the translation to the SMT-LIB theory QF_UFBV, the range of f is not anymore the original RISCAL domain D of the existentially quantified variable, but some bit vector type B whose values encode the values of D. Since not every bit vector in B necessarily represents an element from D, we have to constrain the range of f by a predicate p_D that holds for a bit vector $b \in B$ if and only if b actually represents an element from D. We achieve this by adding to the translation an axiom

$$\bigwedge_{d_1,\ldots,d_m} p_D(f(t(d_1),\ldots,t(d_m)))$$

where $t(d)$ represents an expression that denotes the bit vector associated to the RISCAL value d. The size of this conjunction is proportional to the number of possible combinations of values $d_1,\ldots,d_m$ for the arguments of f; this may blow up the SMT-LIB translation considerably and overcome the benefits of applying Skolemization rather than expansion. Thus the translation can be configured to apply a heuristic: if the number of conjuncts in the Skolemization axiom is significantly larger than the number of conjuncts derived from expanding the original formula, the translation forsakes Skolemization in favor of expansion.

While expressions denoting unique values can be directly encoded by bit vectors operations, an (choose $y\colon D.\ F[x,y]$) with free variable $x\colon D$ gives in the SMT-LIB translation rise to a new function $f:\ D \to D$ with axiom $\forall x\colon D.\ F[x,f(x)]$. 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. 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 above). However, in certain contexts we may replace applications of such axiomatized functions. For instance, take the formula $\forall a.\ (\ldots f(a)\ldots)$ where application $f(a)$ occurs positively (unnegated) in a context $(\ldots\ f(a)\ldots)$ that does not embed $f(a)$ in another quantifier and assume that function $f:\ D \to D$ has been axiomatized as described above. Then above formula can be transformed to the equi-satisfiable formula $\forall a,b.\ (F[a,b] \Rightarrow \ldots b\ldots)$. Thus, we have replaced the application $f(a)$ of axiomatized function f 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 that are actually relevant. 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. 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.

Comparison As discussed above, deciding by SMT solving formulas with quantifiers or choose expressions can become problematic, because the SMT-LIB translation may yield vastly expanded formulas (arising from existential formulas, quantified constraints of Skolem functions, and axioms of uninterpreted functions emerging from choose expressions or modular verification). To which extent this affects the actual performance of the decision process can be investigated only by actual benchmarks.

3 Artificial Benchmarks

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

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

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"; this predicate is valid and its negation is unsatisfiable. On the other side, predicate *cycle4-sat1* is satisfiable but not valid, as is its negation. However, while *cycle4-sat1* has many satisfying assignments, its negation has only few; thus *cycle4-sat1* 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. Here 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. 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 $\exists^i\forall^j$ or $\forall^j\exists^i$, 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.

Experimental Results The four diagrams in Figure 1 plot the decision times for the quantified formulas with (valid) predicate *cycle4-valid* and its (unsatisfiable) negation and for the satisfiable (mostly valid) predicate *cycle4-sat1* and its also satisfiable (but mostly unsatisfiable) negation. The labels of the horizontal axis denote the applied quantification pattern (labels e*i*a*j* and a*i*e*j* denote patterns $\exists^i\forall^j$ and $\forall^i\exists^j$, respectively). 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.

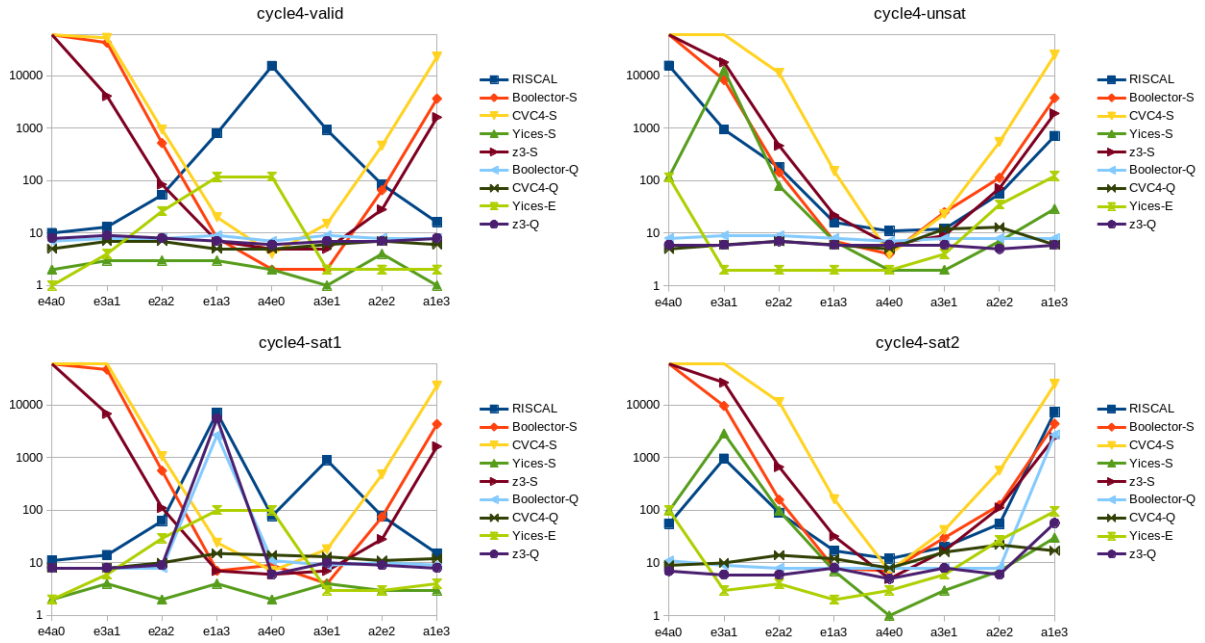

Figure 1: Artificial Benchmarks: Base Behavior

The various labeled curves give the times for the decision 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. Here tag RISCAL represents the built-in semantic evaluation mechanism of RISCAL. Tags Boolector-S, CVC4-S, Yices-S, Z3-S represent the application of the various SMT solvers to the generated QF_UFBV formula where quantifiers are removed by Skolemization (in case of the originally universal quantifiers) or expansion (in case of the existential quantifiers). Tags Boolector-Q, CVC4-Q, Z4-Q represent the application of the SMT solvers where, however, in the generated formula all quantifiers are preserved (Boolector, CVC4, and Z3 also support quantification). Yices-E represents the application of Yices where also the (originally) universal quantifiers have been 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), we apply the alternative of expanding also (originally) universal quantifiers.

An inspection of the diagrams shows that, when eliminating quantifiers by Skolemization or 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. 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; also in the right diagrams (mostly unsatisfiable base predicates), the performance of semantic evaluation at least matches that of the solvers.

Furthermore, the semantic evaluation mechanism of RISCAL exhibits comparatively good performance in diagram *cycle4-valid* 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.

More results are given in Appendix A which illustrates in Figure 4 and Figure 5 corresponding benchmarks with more complex predicates involving operations such as non-linear arithmetic, arithmetic quantification, set and array operations. These benchmarks reveal various situations when the semantic evaluation mechanism of RISCAL is able to compete with or even outperform some of the SMT solvers; for a more detailed interpretation of these and other benchmarks, see [18].

So far, we have only considered formulas with built-in operations (functions and predicates). Now we are going to also consider operations specified by contracts such as the following two functions:

$$\begin{aligned}&\mathsf{fun}\ f(x_1,x_2,x_3,x_4)\ \mathsf{ensures}\\&\quad \mathsf{if}\ x_1<x_2\wedge x_2<x_3\wedge x_3<x_4\wedge x_4<x_1\ \mathsf{then}\ \mathit{result}=0\ \mathsf{else}\ \mathit{result}=1;\\&\mathsf{fun}\ g(x_1,x_2,x_3,x_4)\ \mathsf{ensures}\\&\quad \mathsf{if}\ x_1=x_2\wedge x_3=x_4\ \mathsf{then}\ \mathit{result}=0\ \mathsf{else}\ \mathit{result}=1;\end{aligned}$$

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 four base predicates that test the equality of above functions with values 1 or 0, yielding again one valid, one unsatisfiable, and two satisfiable situations.

Figure 2 displays the decision times for model parameter $N := 5$ (we now use value 5 rather than 6 to compensate the additional quantifier of the function axiom). Here 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

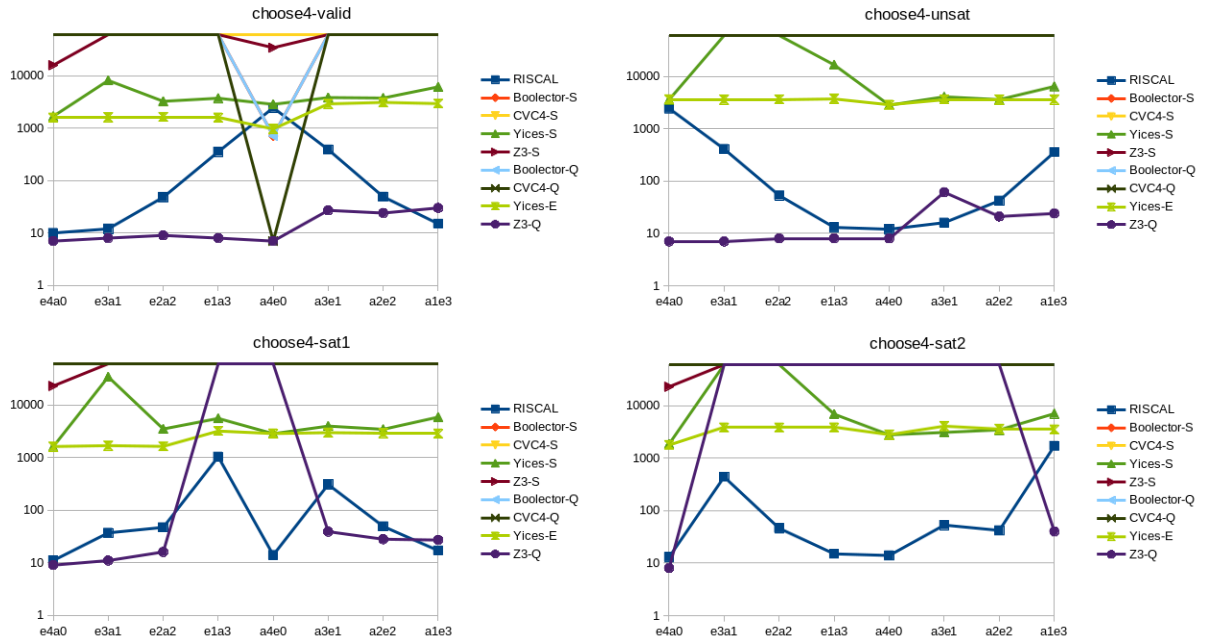

Figure 2: Formulas with Functions Specified by Contracts

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 applied the previously mentioned options "eliminate choices" and "inline definitions" to eliminate certain applications of contract-specified operations by embedding the postconditions into the enclosing formulas; this does not change the satisfiability 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

The artificial benchmarks investigated so far do not demonstrate whether/how often the observed effects indeed emerge in "real-life" examples. To shed some light on this issue, we have also collected from real RISCAL models a selection of formulas whose validity is to be decided. These formulas mainly represent conditions to validate the specifications of problems, verify the correctness of algorithms, or also theorems over the domains of consideration. 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 explicitly selected a sample where this is not the case, i.e., where semantic evaluation is competitive with SMT solving.

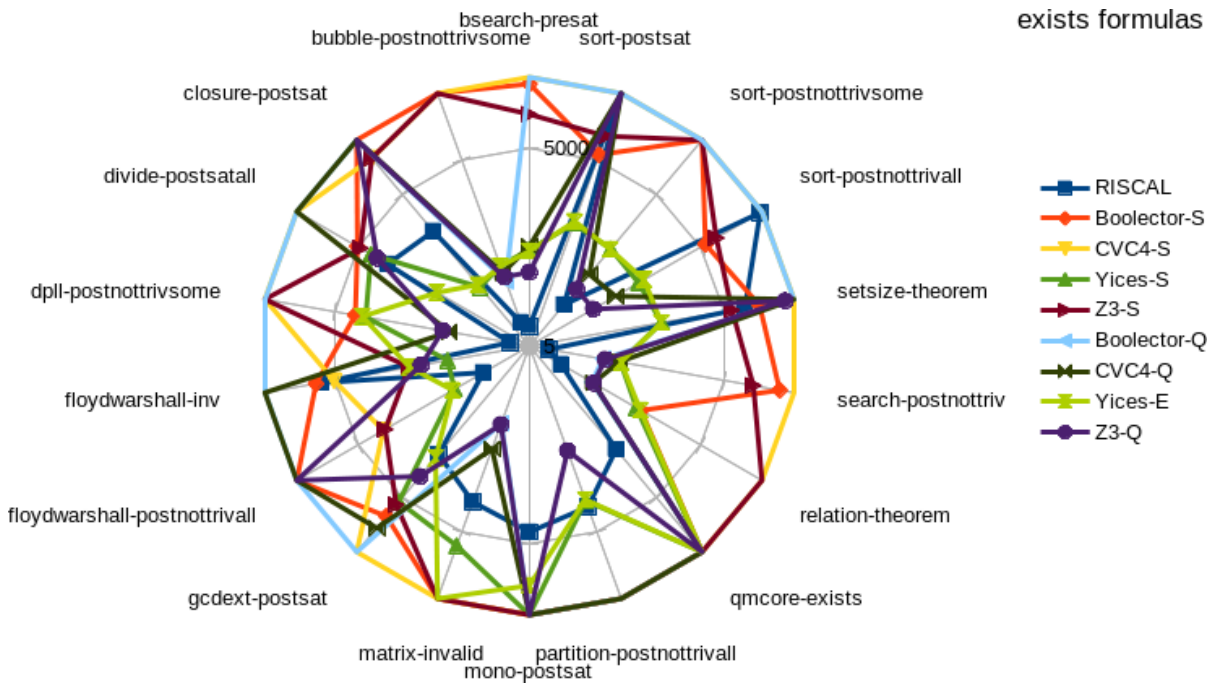

Figure 3: Real-Life Benchmarks

The diagram displayed in Figure 3 presents the results of benchmarking the decision of some of these formulas (see Appendix A or [18] for a link to the sources). The benchmarks are 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 particular, the diagram illustrates benchmarks for formulas that have substantial "existential" content; many stem from validating procedure contracts by checking the satisfiability of the postconditions for all inputs that satisfy the preconditions; here the semantic evaluation 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).

Appendix A lists in Figure 6 more benchmarks where the left column illustrates the execution of the benchmarks with smaller values for the model parameters, while the right column illustrates executions with larger values; all executions were again terminated after 60 seconds. The top row gives the benchmarks already shown in Figure 3, but now also with larger model parameters. The second row illustrates benchmarks for formulas with choose expressions; here in the SMT decisions the option "choose elimination" transformation was *not* applied. The evaluation mechanism of RISCAL again outperforms many of the SMT solvers; among these typically Yices performs best. The third row illustrates benchmarks for the same formulas as in the second row but with the options "choose elimination" and "inline definitions" turned on. This shows a clear improvement in many examples; now various SMT solvers clearly beat the semantic evaluation mechanism of RISCAL. The last row illustrates benchmarks for formulas that do not obviously fall into above categories but where nevertheless the semantic evaluation mechanism is competitive, from the structure of the formulas and/or the complexity of the underlying operations.

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 paper, 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 case are theorems with substantial "existential" content, i.e., positive occurrences of existential quantifiers with large quantification ranges: the resulting formulas may be so big that their decision by SMT solving may be outperformed by semantic evaluation. Another case are theorems with 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 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 denote 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 most real-life examples.

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] Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reinher Hähnle, Peter H. Schmitt & Mattias Ulbrich, editors (2016): *Deductive Software Verification — The KeY Book — From Theory to Practice*. *LNCS* 10001, Springer International Publishing, Cham, doi:10.1007/978-3-319-49812-6.

[2] Mike Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs & K. Rustan M. Leino (2005): *Boogie: A Modular Reusable Verifier for Object-Oriented Programs*. In: *FMCO 2005: Formal Methods for Components and Objects*, *LNCS* 4111, Springer, Berlin, Germany, pp. 364–387, doi:10.1007/11804192_17.

[3] Clark Barrett, Pascal Fontaine & Cesare Tinelli (2016): *The Satisfiability Modulo Theories Library (SMT-LIB)*. Available at `http://www.SMT-LIB.org`.

[4] Jasmin Christian Blanchette, Sascha Böhme & Lawrence C. Paulson (2011): *Extending Sledgehammer with SMT Solvers*. In: *CADE-23: Automated Deduction, 23rd International Conference*, *LNCS* 6803, Springer, Berlin, Germany, pp. 116–130, doi:10.1007/978-3-642-22438-6_11.

[5] Jasmin Christian Blanchette & Tobias Nipkow (2010): *Nitpick: A Counterexample Generator for Higher-Order Logic Based on a Relational Model Finder*. In: *ITP 2010: Interactive Theorem Proving*, *LNCS* 6172, Springer, Berlin, Germany, pp. 131–146, doi:10.1007/978-3-642-14052-5_11.

[6] David Déharbe, Pascal Fontaine, Yoann Guyot & Laurent Voisin (2012): *SMT Solvers for Rodin*. In: *ABZ 2012: Abstract State Machines, Alloy, B, VDM, and Z, Third International Conference, Pisa, Italy, June 18–21, 2012*, *LNCS* 7316, Springer, Berlin, Germany, pp. 194–207, doi:10.1007/978-3-642-30885-7_14.

[7] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz et al. (2017): *SMTCoq: A Plug-In for Integrating SMT Solvers into Coq*. In: *CAV 2017: Computer Aided Verification, Heidelberg, Germany, July 24–28, 2017*, *LNCS* 10427, Springer, Cham, Switzerland, pp. 126–133, doi:10.1007/978-3-319-63390-9_7.

[8] Aboubakr Achraf El Ghazi & Mana Taghdiri (2015): *Analyzing Alloy Formulas using an SMT Solver: A Case Study*. AFM10: Automated Formal Methods, July 14, 2010, Edinburgh, UK. Available at `https://arxiv.org/abs/1505.00672`.

[9] Daniel Jackson (2000): *Automating First-Order Relational Logic*. In: *SIGSOFT'00/FSE-8 International Symposium, San Diego, California, USA, November 2000*, *SIGSOFT Software Engineering Notes* 25(6), ACM, New York, NY, USA, pp. 130–139, doi:10.1145/355045.355063.

[10] K. Rustan M. Leino (2010): *Dafny: An Automatic Program Verifier for Functional Correctness*. In Edmund M. Clarke & Andrei Voronkov, editors: *LPAR-16: Logic Programming and Automated Reasoning, 16th International Conference, Dakar, Senegal, April 25–May 1, 2010*, *LNCS* 6355, Springer, Berlin, Germany, pp. 348–370, doi:10.1007/978-3-642-17511-4_20.

[11] Hsin-Hung Lin & Bow-Yaw Wang (2017): *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*, ACM, New York, NY, USA, p. 132–135, doi:10.1145/3127041.3127066.

[12] Stephan Merz & Hernán Vanzetto (2016): *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*, *LNCS* 9675, Springer, Cham, Switzerland, pp. 54–69, doi:10.1007/978-3-319-33600-8_3.

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

[14] Franz-Xaver Reichl (2020): *The Integration of SMT Solvers into the RISCAL Model Checker*. Master's thesis, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Austria. Available at `https://www.risc.jku.at/publications/download/risc_6103/Thesis.pdf`.

[15] Wolfgang Schreiner (2018): *Validating Mathematical Theories and Algorithms with RISCAL*. In: *CICM 2018, 11th Conference on Intelligent Computer Mathematics, Hagenberg, Austria, August 13–17*, *LNCS/Lecture Notes in Artificial Intelligence* 11006, Springer, Berlin, pp. 248–254, doi:10.1007/978-3-319-96812-4_21.

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

[17] Wolfgang Schreiner & Franz-Xaver Reichl (2020): *Mathematical Model Checking Based on Semantics and SMT*. *Transactions on Internet Research* 16(2), pp. 4–13. Available at `http://ipsitransactions.org/journals/papers/tir/2020jul/p2.pdf`.

[18] Wolfgang Schreiner & Franz-Xaver Reichl (2021): *Semantic Evaluation versus SMT Solving in the RISCAL Model Checker*. Technical Report 21-11, RISC, Johannes Kepler University, Linz, Austria. Available at `https://www.risc.jku.at/publications/download/risc_6328/21-11.pdf`.

[19] Wolfgang Schreiner, William Steingartner & Valerie Novitzká (2020): *A Novel Categorical Approach to the Semantics of Relational First-Order Logic*. *Symmetry* 12(10), doi:10.3390/sym12101584.

[20] SMT-COMP (2021): *SMT-COMP: The International Satisfiability Modulo Theories (SMT) Competition.* Available at `https://smt-comp.github.io`.

[21] Emina Torlak & Daniel Jackson (2007): *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*, *LNCS* 4424, Springer, Berlin, Germany, pp. 632–647, doi:10.1007/978-3-540-71209-1_49.

A Benchmark Diagrams

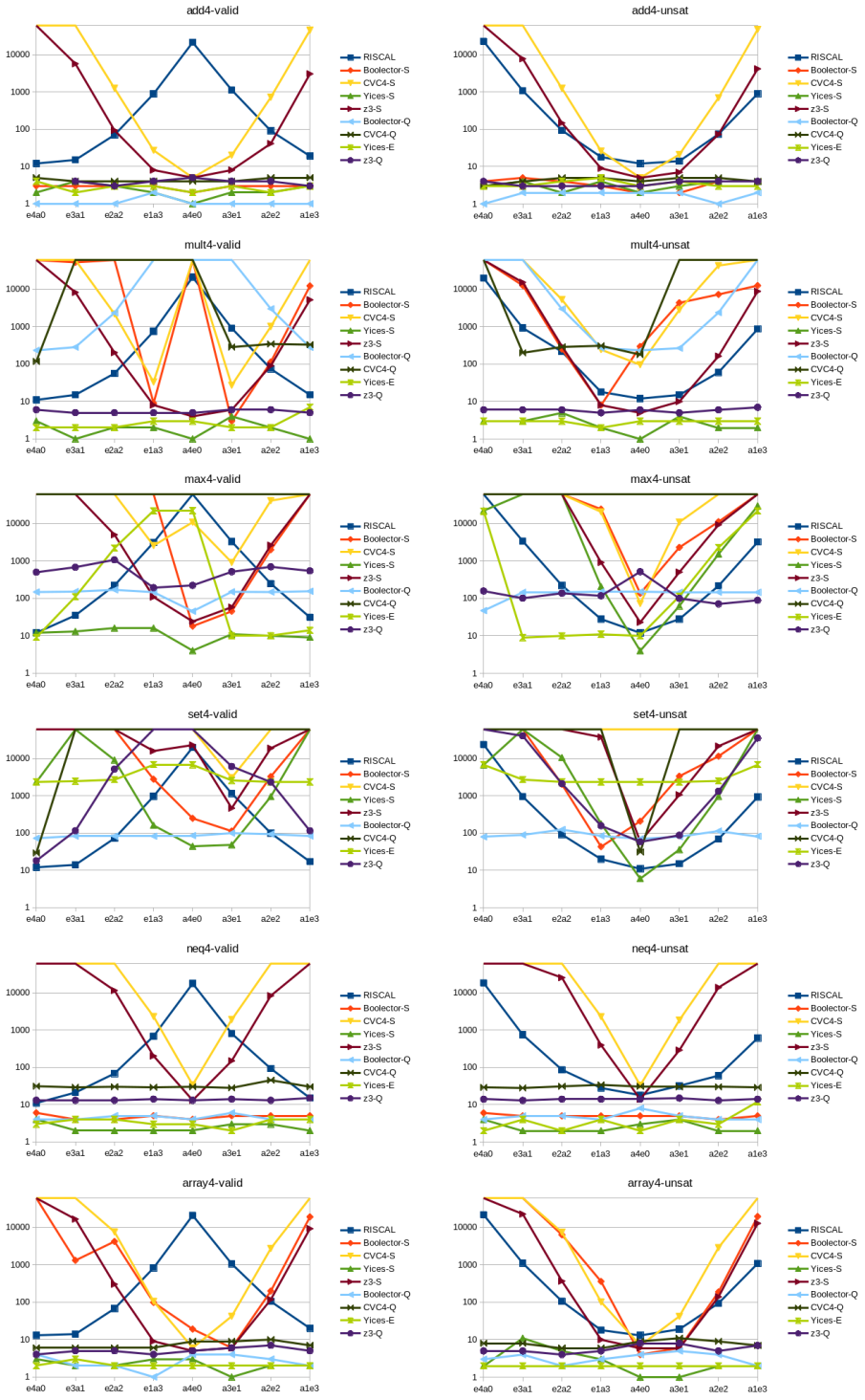

Figure 4: Artificial Benchmarks: Valid versus Unsatisfiable Predicates

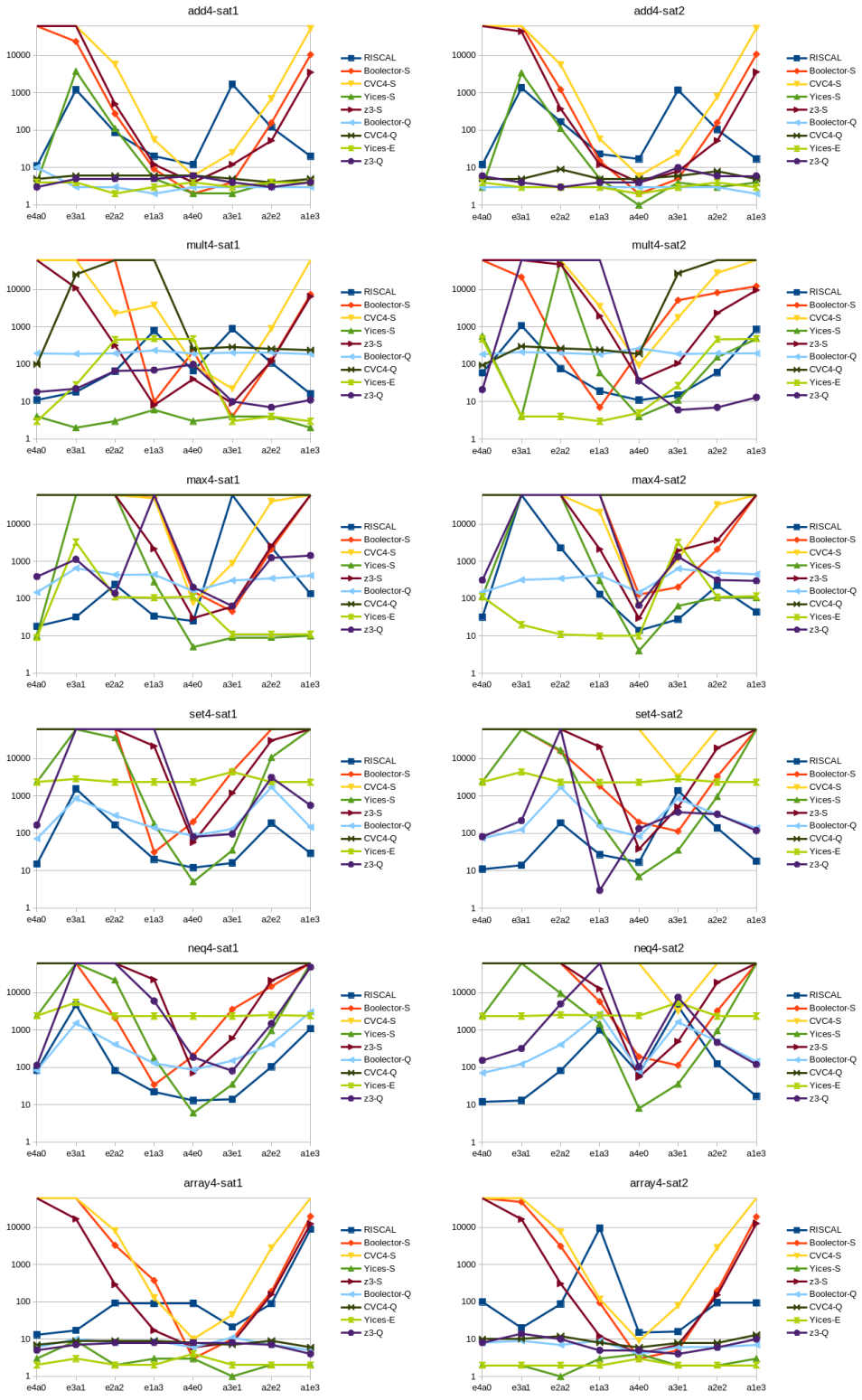

Figure 5: Artificial Benchmarks: Satisfiable Predicates and their Negations

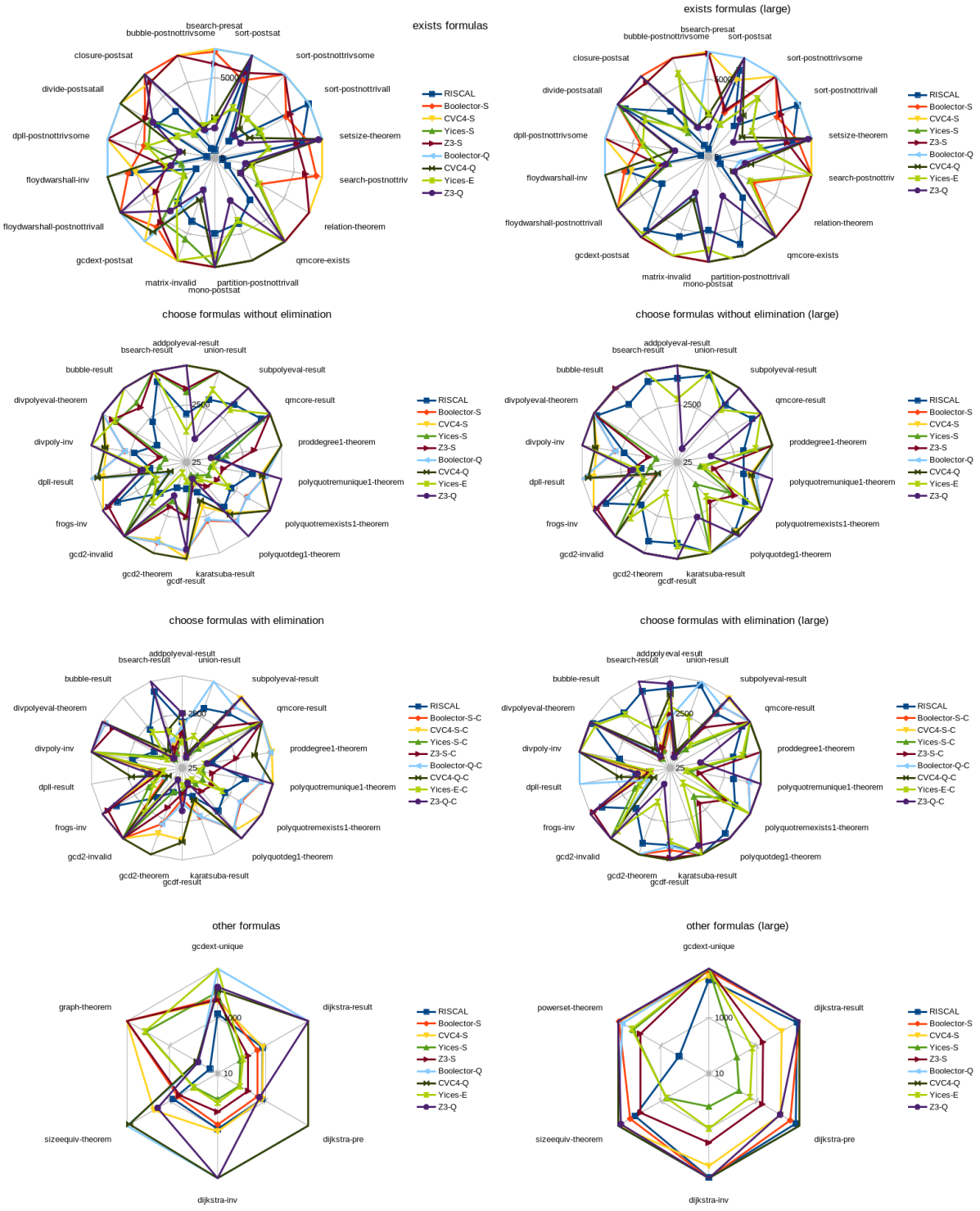

Figure 6: Real-Life Benchmarks (available from `https://www.risc.jku.at/research/formal/software/RISCAL/papers/EvalSMT2021-models.tgz`)