Towards Lightweight Integration of SMT Solvers

Andrei Lapets
Boston University
Boston, USA
lapets@bu.edu

Saber Mirzaei
Boston University
Boston, USA
smirzaei@bu.edu

1 Introduction

A large variety of SMT techniques and associated solvers (i.e., algorithms and software tools) have been developed by the formal modelling and verification communities. For a particular application domain, each technique has its own unique set of advantages and limitations. Within the context of a particular application domain (characterized by a particular set of possible logical formulas), the fitness of a technique can be characterized along multiple dimensions: expressiveness, soundness, completeness, responsiveness, computational cost, and others. Furthermore, certain application domains may require that multiple techniques be used in concert in order to validate or solve the particular set of formulas that must be supported.

We consider a potential end-user for SMT solvers that may be interested in frequently and quickly verifying a collection of relatively small formal statements within an application domain. We further posit that the end-user does not have the resources, incentive, or interest in: (1) independently becoming familiar with the syntaxes, interfaces, capabilities, and limitations of any existing SMT solver, or (2) understanding how a collection of SMT solvers can be used in concert. Instead, the user wishes to employ a single syntax for formulas that is conventional in their application domain while still employing the verification capabilities of existing SMT solvers. Examples of such end-users are an instructor or student in a university course the topic of which is (or which employs) algebra or logic [Lap12], or a distributed system protocol designer [LSB+12].

In this report we describe an initial exploration of how it might be possible to build a single lightweight, integrated environment that allows end-users to seamlessly employ multiple SMT solvers while intelligently navigating trade-offs. In particular, we focus on defining transformations from a single syntax for logical formulas to four SMT solvers: CVC3 [BT07], Alt-Ergo [CCKL08, BCCL08, Alt], Yices [DdM06, Yic, DM06], and Z3 [DMB08, Z3g]. In anticipation of further work, we also discuss informally a few relevant characteristics of these SMT solvers along relevant dimensions. This work represents an example that will be used to inform further work on lightweight integration of multiple SMT solvers and other formal algorithms, systems, and tools; thus, what is described in this report does not represent a definition or implementation to be deployed and used by actual end-users, but a prototype that can help inform further efforts.

2 Integration of SMT Solvers

Let $\mathcal{V}$ and $\mathcal{P}$ be sets of symbols (variables and predicates), and let $\mathcal{V} \to \mathbb{Z}$ be a set of maps (including all partial finite maps) from variables to integers. A finite map $s \in \mathcal{V} \to \mathbb{Z}$ represents a possible solution to a logical formula with variables that range over the integers. We assume a common *syntax* for logical formulas defined in Table 1.

As illustrated in Figure 1, we have constructed a web-based integrated environment that allows users to submit logical formulas represented using the common syntax in Table 1 for validation. The submitted formula is translated by the web application into four separate formulas, each of which is represented using a syntax that is appropriate for one of the SMT solvers. Each formula is then submitted for processing to an instances of its corresponding SMT solver, being hosted on the cloud. The response times and results are recorded in a database.

The individual translations are defined from this syntax to subsets of the respective syntaxes of the four SMT solvers. The output of each of the SMT solvers is then translated into the logical syntax for outputs in Table 1. Notice that outputs may be incomplete, or may represent actual integer solutions to a formula. We present the target syntax subset for each SMT solver we have considered, including the output syntax, in Tables 2, 3, 4, and 5.

variable	x	$\in$	$\mathcal{V}$
predicate	p	$\in$	$\mathcal{P}$
solution	s	$\in$	$\mathcal{V} \to \mathbb{Z}$
term	t	::=	$\ldots \mid -2 \mid -1 \mid 0 \mid 1 \mid 2 \mid \ldots$
		$\mid$	x
		$\mid$	$-t \mid t_1 + t_2 \mid t_1 - t_2 \mid t_1 \cdot t_2$
formula	f	::=	**true** $\mid$ **false**
		$\mid$	$t_1 = t_2$
		$\mid$	$t_1 \neq t_2 \mid t_1 < t_2 \mid t_1 \leq t_2 \mid t_1 > t_2 \mid t_1 \geq t_2$
		$\mid$	$p\ (\ t_1, \ldots, t_n\ )$
		$\mid$	**not** $f \mid f_1$ **implies** $f_2 \mid f_1$ **and** $f_2 \mid f_1$ **or** f_2
		$\mid$	**for all** $x \in \mathbb{Z}$, f
		$\mid$	**exists** $x \in \mathbb{Z}$, f
		$\mid$	$(\ f\ )$
output	o	::=	**true** $\mid$ **false** $\mid$ **unknown** $\mid$ s

Table 1: Common language for logical formulas.

CVC3. CVC3 provides built-in theories that support working with formulas that govern real numbers and integers, fixed-size bit vectors, arrays, tuples, records, and user-defined recursive data types. Given a formula, the CVC3 output regarding its validity is sound but not necessarily complete (support for formulas with quantifiers and non-linear arithmetic is limited, and the axiom of induction over the natural numbers is not automatically validated). CVC3 cannot return an explicit solution to a formula. This is reflected in the target syntax presented in Table 2.

Alt-Ergo. Alt-Ergo [CCKL08, BCCL08] is a theorem prover capable of recognizing a collection of valid formulas. Built-in theories are provided that support formulas governing real numbers and integers, fixed-size bit vectors, arrays, and user-defined types. While the Alt-Ergo output indicating a formula's validity is sound, for logical formulas that are false Alt-Ergo returns the output message

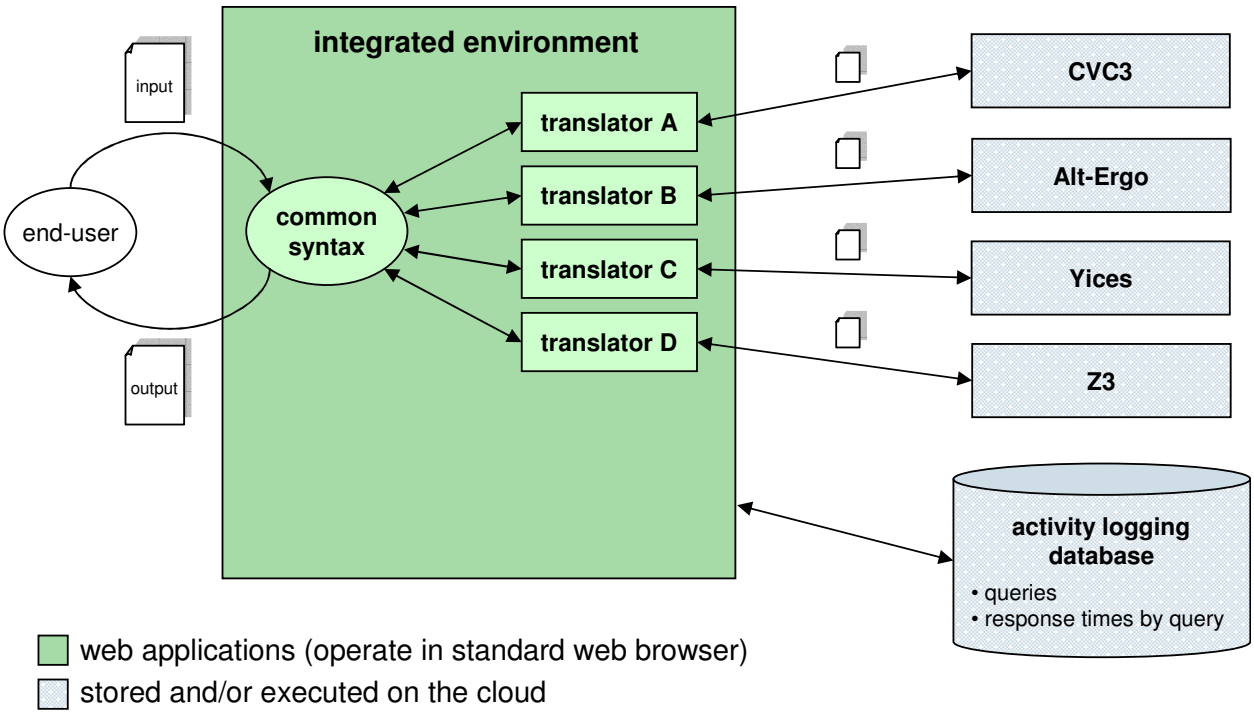

Figure 1: Integrated environment providing end-users a single point from which to employ multiple underlying SMT solvers.

`I don't know`. Thus, the output is not complete and, in fact, cannot confirm that a formula is false directly (although it is possible to negate the formula before invoking Alt-Ergo). Without additional information, the output `I don't know` must be mapped to the output **unknown** in the common logical syntax in Table 1. There is support for non-linear arithmetic and some support for formulas with quantifiers. Furthermore, the axiom of induction over natural numbers is validated automatically because the system is a theorem prover based on CC(X) [CCKL08]. Alt-Ergo cannot return an explicit solution to a formula. This is reflected in the target syntax presented in Table 3.

Yices. The Yices SMT solver is accompanied by built-in theories that support formulas governing real numbers and integers, fixed-size bit vectors, arrays, and user-defined recursive data types. The Yices output indicating a formula's validity is sound but not complete; for example, it cannot successfully validate an axiom of induction over the natural numbers, and support for formulas with quantifiers is limited. There is support for formulas with non-linear arithmetic. Yices cannot return an explicit solution to a formula. This is reflected in the target syntax presented in Table 4.

Z3. Z3 provides built-in theories that support working with formulas that govern real numbers and integers, fixed-size bit vectors, arrays, and user-defined recursive data types. Given a formula, the Z3 output regarding its validity is sound but not necessarily complete (the system may time out for some formulas with quantifiers, and the axiom of induction over the natural numbers is not automatically validated). Formulas with non-linear arithmetic are supported. Finally, Z3 is distinguished by its ability to return an explicit solution to a formula. This is reflected in the target syntax presented in Table 5.

variable	x	$\in$	$\mathcal{V}$
predicate	p	$\in$	$\mathcal{P}$
solution	s	$\in$	$\mathcal{V} \to \mathbb{Z}$
term	t	::=	... | `-2` | `-1` | `0` | `1` | `2` | ...
		|	x
		|	`-` t | t_1 `+` t_2 | t_1 `-` t_2 | t_1 `*` t_2
formula	f	::=	`TRUE` | `FALSE`
		|	t_1 `=` t_2
		|	t_1 `/=` t_2 | t_1 `<` t_2 | t_1 `<=` t_2 | t_1 `>` t_2 | t_1 `>=` t_2
		|	p `(` $t_1, \ldots, t_n$ `)`
		|	`NOT` f | f_1 `=>` f_2 | f_1 `AND` f_2 | f_1 `OR` f_2
		|	`FORALL (` x `:` `INT ):` f
		|	`EXISTS (` x `:` `INT ):` f
		|	`(` f `)`
output	o	::=	`Valid.` | `Invalid.` | `Unknown.`

Table 2: Target subset of CVC3 syntax.

variable	x	$\in$	$\mathcal{V}$
predicate	p	$\in$	$\mathcal{P}$
solution	s	$\in$	$\mathcal{V} \to \mathbb{Z}$
term	t	::=	... | `-2` | `-1` | `0` | `1` | `2` | ...
		|	x
		|	`-` t | t_1 `+` t_2 | t_1 `-` t_2 | t_1 `*` t_2
formula	f	::=	`true` | `false`
		|	t_1 `=` t_2
		|	t_1 `<` t_2 | t_1 `<=` t_2 | t_1 `>` t_2 | t_1 `>=` t_2
		|	p `(` $t_1, \ldots, t_n$ `)`
		|	`not` f | f_1 `->` f_2 | f_1 `and` f_2 | f_1 `or` f_2
		|	`forall (` x `:` `int ).` f
		|	`exists (` x `:` `int ).` f
		|	`(` f `)`
output	o	::=	`Valid` | `I don't know`

Table 3: Target subset of Alt-Ergo syntax.

variable	x	$\in$	$\mathcal{V}$
predicate	p	$\in$	$\mathcal{P}$
solution	s	$\in$	$\mathcal{V} \to \mathbb{Z}$
term	t	::=	`...` | `-2` | `-1` | `0` | `1` | `2` | `...`
		|	x
		|	`-` t | `+` t_1 t_2 | `-` t_1 t_2 | `*` t_1 t_2
formula	f	::=	`true` | `false`
		|	`=` t_1 t_2
		|	`/=` t_1 t_2 | `<` t_1 t_2 | `<=` t_1 t_2 | `>` t_1 t_2 | `>=` t_1 t_2
		|	p t_1 ... t_n
		|	`not` f | `=>` f_1 f_2 | `and` f_1 f_2 | `or` f_1 f_2
		|	`forall (` x `::` `int )` f
		|	`exists (` x `::` `int )` f
		|	`(` f `)`
output	o	::=	`sat` | `unsat` | `unknown`

Table 4: Target subset of Yices syntax.

variable	x	$\in$	$\mathcal{V}$
predicate	p	$\in$	$\mathcal{P}$
solution	s	$\in$	$\mathcal{V} \to \mathbb{Z}$
term	t	::=	`...` | `-2` | `-1` | `0` | `1` | `2` | `...`
		|	x
		|	`-` t
		|	`+` t_1 t_2 | `-` t_1 t_2 | `*` t_1 t_2
formula	f	::=	`true` | `false`
		|	`=` t_1 t_2
		|	`/=` t_1 t_2 | `<` t_1 t_2 | `<=` t_1 t_2 | `>` t_1 t_2 | `>=` t_1 t_2
		|	p t_1 ... t_n
		|	`not` f | `=>` f_1 f_2 | `and` f_1 f_2 | `or` f_1 f_2
		|	`forall (` x `Int )` f
		|	`exists (` x `Int )` f
		|	`(` f `)`
output	o	::=	`sat` | `unsat` | `time-out` | s

Table 5: Target subset of Z3 syntax.

3 Conclusion and Future Work

This report describes a prototype attempt to integrate four SMT solvers by identifying and defining bidirectional mappings from a common syntax for logical formulas and system outputs to the input and output syntaxes of the four SMT solvers. These mappings provide a rough characterization of the solvers along the dimensions of expressiveness, soundness, and completeness. All the underlying solvers return sound outputs given an input formula, but they are not all complete over the set of all input formulas.

Even within the restricted context of the common logical syntax in Table 1, unique capabilities have been identified for two of the systems that distinguish them from the others: Alt-Ergo can verify an axiom of induction, and Z3 can return as output an explicit solution to a formula over the integers. Deploying even these two systems behind a single integrated interface already provides an end-user with an environment the capabilities of which are beyond those of any of the individual tools being considered.

The presented collection of mappings and the integrated environment implementation into which the SMT solvers are incorporated provides a starting point from which it will be possible to further extend and refine the characterization of the solvers along the dimensions of expressiveness, completeness, and response time. This will likely lead to further distinctions between the systems that it may be possible for users to exploit (automatically or manually).

For example, for each SMT solver, we can define much more precisely which of the following characteristics it possesses (as subsets of the common logical syntax in Table 1), in order to better delineate the domain of logical formulas over which it is complete:

- support for formulas with quantifiers;
- support for formulas with non-linear arithmetic;
- support for formulas that involve induction over natural numbers;
- support for returning explicit solutions to arithmetic formulas.

Precisely defining the logical syntax subset over which an SMT solver is known to be complete makes it possible to intelligently choose an SMT solver based on the user's needs (e.g., the quantifiers and operations that appear in the formula, and whether the user requires completeness in the output). Notably, the integrated environment implementation prototyped in this work provides an opportunity to automate this process. Given a library of various subsets of the common syntax (possibly corresponding to some of the categories above), it might be possible to automate the process of determining gradually over which of these subsets an underlying solver is complete. This can be accomplished by recording particular information for every query: the submitted logical formula, the subsets of the common syntax to which the formula belongs, and the output of each tool. Initially, the solver is assumed to be complete over all these subsets; as counterexamples are encountered, this assumption is adjusted.

It will also be of interest to explore ways to predict and compare the response time of each of the integrated SMT solvers on various classes of formulas. These classes of formulas may correspond to the manually defined categories discussed above, or to classes defined in terms of characterization and metric functions (e.g., based on their syntactic structure and depth, the number of variables and quantifiers, and so on [AWD12]) and recording for each submitted formula both its characteristics and the response time. This data can then be presented to users so that they can make intelligent trade-offs, or it can be used to automatically choose the most suitable SMT solver based on the user's priorities (e.g., completeness or response time) by incorporating machine learning techniques [AWD12].

References

[Alt] Alt-Ergo/examples. `http://alt-ergo.lri.fr/examples/`.

[AWD12] Mohammad Abdul Aziz, Amr Wassal, and Nevine Darwish. A Machine Learning Technique for Hardness Estimation of QFBV SMT Problems (Work in progress). In *Proceedings of the 10th International Workshop on Satisfiability Modulo Theories*, pages 56–65, Manchester, UK, 2012.

[BCCL08] François Bobot, Sylvain Conchon, Evelyne Contejean, and Stéphane Lescuyer. Implementing Polymorphism in SMT solvers. In Clark Barrett and Leonardo de Moura, editors, *SMT 2008: 6th International Workshop on Satisfiability Modulo*, 2008.

[BT07] Clark Barrett and Cesare Tinelli. CVC3. In Werner Damm and Holger Hermanns, editors, *Proceedings of the* 19^{th} *International Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, July 2007. Berlin, Germany.

[CCKL08] Sylvain Conchon, Evelyne Contejean, Johannes Kanig, and Stéphane Lescuyer. CC(X): Semantic Combination of Congruence Closure with Solvable Theories. *Electronic Notes in Theoretical Computer Science*, 198(2):51–69, May 2008.

[DdM06] B. Dutertre and L. de Moura. The Yices SMT solver. Tool paper at http://yices.csl.sri.com/tool-paper.pdf, August 2006.

[DM06] B. Dutertre and L. Moura. System description: Yices 1.0. In *SMT-COMP'06*, 2006.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient smt solver. In *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[Lap12] Andrei Lapets. Accessible Integrated Formal Reasoning Environments in Classroom Instruction of Mathematics. Technical Report BUCS-TR-2012-015, CS Dept., Boston University, December 2012.

[LSB+12] Andrei Lapets, Richard Skowyra, Christine Bassem, Sanaz Bahargam, Assaf Kfoury, and Azer Bestavros. Towards Accessible Integrated Formal Reasoning Environments for Protocol Design. Technical Report BUCS-TR-2012-016, CS Dept., Boston University, December 2012.

[Yic] Yices/Examples. `http://yices.csl.sri.com/examples.shtml`.

[Z3g] Z3 - guide. `http://rise4fun.com/z3/tutorialcontent/guide`.