Reachability Modulo Theory Library
(Extended Abstract)

Francesco Alberti
Formal Verification and Security Lab.
University of Lugano, Switzerland
francesco.alberti@usi.ch

Roberto Bruttomesso
Atrenta
Grenoble, France
roberto@atrenta.com

Silvio Ghilardi
Università degli Studi di Milano
Milan, Italy
ghilardi@dsi.unimi.it

Silvio Ranise
FBK - Irst
Trento, Italy
ranise@fbk.eu

Natasha Sharygina
Formal Verification and Security Lab.
University of Lugano, Switzerland
natasha.sharygina@usi.ch

Abstract

Reachability analysis of infinite-state systems plays a central role in many verification tasks. In the last decade, SMT-Solvers have been exploited within many verification tools to discharge proof obligations arising from reachability analysis. Despite this, as of today there is no standard language to deal with transition systems specified in the SMT-LIB format. This paper is a first proposal for a new SMT-based verification language that is suitable for defining transition systems and safety properties.

1 Introduction

Reachability analysis plays a central role in many verification tasks. This kind of analysis addresses the problem of verifying if a given system can reach a particular configuration of interest, representing, e.g., the violation of properties which should be satisfied by every state reachable by the system, a goal that should be always reachable, etc.

While a finite-state representation is well-suited to formally represent hardware systems, it fails to deal with several aspects of software systems, because they give rise to infinite state spaces. Examples of such aspects include parameterization, i.e., systems with an unbounded numbers of components, systems handling variables defined over unbounded domains, dynamic processes creation and memory allocation, etc. The reachability problem for infinite-state systems is, in general, undecidable. Nonetheless it received—and still is receiving—a lot of attention, as many verification tasks of real systems can be reduced to it. For example, think about a protocol ensuring the mutual exclusion in a multi-agent system. The configuration of interest is that in which *at least* two agents are, at the same time, in the critical section.

It is not difficult to see that the class of infinite-state systems is huge, ranging from imperative programs to parameterized protocols, access control policies, timed automata, etc. Each class of systems requires ad hoc techniques, heuristics, algorithms and arrangements to effectively perform a reachability analysis. Since there is no standard for the specification of infinite-state systems, it is extremely difficult to evaluate and compare different tools solving the same problems. Furthermore, there is no collection of benchmarks which would be extremely useful in helping developers in advancing the performance of their tools. Three notable exceptions are (1) the *Intermediate Verification Languages (IVLs)* for program verification such as Boogie [3] and Why [9], (2) the *Numerical Transition Systems (NTSs) Library* (`richmodels.epfl.ch/ntscomp`) whose aim is to provide a common general format and a open library of benchmarks for NTSs, i.e. simple models of computation involving infinite (or very large) data domains, such as integers, rationals or reals, and (3) the *Verification Modulo Theories (VMT) Initiative* (`www.vmt-lib.org`) whose ambitious goal is to provide a common format

and a library of verification problems for transition systems symbolically described in the SMT-LIB standard.

The aim of this paper is to propose a new language for the specification of an important class of *reachability problems* for infinite-state systems, represented in an extension of the SMT-LIB language, so that a collection of benchmarks can be built and verification tools can be compared meaningfully. On the one hand, our aim is more general than IVLs and the NTS Library since we consider a larger class of systems; e.g., not only programs or counter automata but also distributed systems and timed networks. On the other hand, we are more focused than the VMT Initiative since we consider only reachability problems based on our experience with the tools MCMT [13] and SAFARI [1].

For the sake of simplicity and uniformity in presenting our language, we do not consider *ad-hoc* constructs (e.g., abstract reachability trees) for structure preserving abstractions of the control-flow, the recursive or the multi-procedure calls of (imperative) programs that are important ingredients of many software verification techniques for reachability problems. The study of how to combine our proposal with such constructs is left to future work. Some preliminary investigations to represent imperative programs (without recursion and procedures) in a declarative framework similar to the one proposed here have been done in [1].

We hope that this paper can contribute to the debate about the representation of benchmarks for verification problems and advance the state of the art in the field, as happened for, e.g., the SMT-LIB library [18] for SMT-Solvers, TPTP library [20] for theorem proving, or SATLIB library [15] for propositional satisfiability solvers.

Outline of the paper. We introduce an overall description of transition systems modulo theories and the related reachability problem in Section 2. Section 3 presents a suitable language for the definition of transition systems and reachability problems as an extension of the SMT-LIB. We conclude in Section 4. An extended version of this report as well as examples written in the proposed language can be found at `http://www.oprover.org/rmt-lib`.

2 SMT-based Transition Systems

We assume the usual first-order syntactic notions of signature, term, formula, and so on. Following the SMT-LIB tradition, a theory is a pair $T = (\Sigma, \mathcal{C})$ where Σ is a signature and $\mathcal{C}$ a set of Σ-structures which are called models of T. Given a theory $T = (\Sigma, \mathcal{C})$ and a set $\mathbf{v}$ of "fresh" constant, function, or predicate symbols (i.e. $\mathbf{v} \cap \Sigma = \emptyset$), a T-formula $\phi(\mathbf{v})$ is a formula built out of the symbols in $\mathbf{v} \cup \Sigma$. The formula ϕ can be seen as a $T^{\mathbf{v}}$-formula where $T^{\mathbf{v}} = (\Sigma', \mathcal{C}')$ with $\Sigma' = \Sigma \cup \mathbf{v}$ and $\mathcal{C}'$ contains all the Σ'-structures whose reduct to Σ is a model of T.[1] We say that a T-formula $\phi(\mathbf{v})$ is T-satisfiable iff ϕ is $T^{\mathbf{v}}$-satisfiable, i.e. if there exists a model $\mathcal{M}$ of $T^{\mathbf{v}}$ such that ϕ is true in $\mathcal{M}$. If $\mathbf{v}$ and $\mathbf{v}'$ are two sets of fresh symbols, then $\mathbf{v}, \mathbf{v}'$ denotes their union $\mathbf{v} \cup \mathbf{v}'$ and $\phi(\mathbf{v}, \mathbf{v}')$ is the formula built out of the symbols in $\mathbf{v} \cup \mathbf{v}' \cup \Sigma$.

In our framework, a transition system over a background theory $T = (\Sigma, \mathcal{C})$ is defined in a completely declarative way as a tuple $\mathcal{S}_T = (\mathbf{v}, I(\mathbf{v}), \tau(\mathbf{v}, \mathbf{v}'))$. The set $\mathbf{v}$ of fresh symbols is called the set of *state variables* of $\mathcal{S}_T$; a formula $\phi(\mathbf{v})$ is called a *state formula*. $I(\mathbf{v})$ is a state formula representing the set of initial states. We assume the availability of the set $\mathbf{v}'$ of primed copies of the state variables in $\mathbf{v}$; a formula $\tau(\mathbf{v}, \mathbf{v}')$ is called a *transition formula*. The transition formula $\tau(\mathbf{v}, \mathbf{v}')$ represents the transition relation, defining an association between

[1] Let Σ and Σ' be two signatures such that $\Sigma \subseteq \Sigma'$. If $\mathcal{M}$ is a Σ'-structure, then its reduct to Σ is obtained from $\mathcal{M}$ by forgetting the interpretations of the symbols in $\Sigma' \setminus \Sigma$.

the state variables $\mathbf{v}$ immediately before the execution of the transition, i.e. in the actual state, and the state variables $\mathbf{v}'$ immediately after the execution of the transition, i.e. in the next state. The nature of T determines if $\mathcal{S}_T$ is a finite- or infinite-state system.

Let $\mathcal{S}_T = (\mathbf{v}, I(\mathbf{v}), \tau(\mathbf{v}, \mathbf{v}'))$ be a transition system over a background theory T and $\mathbf{v}^n$ be obtained from the set $\mathbf{v}$ by renaming a copy of each state variable in $\mathbf{v}$ with n primes. The *(unbounded) reachability problem* for $\mathcal{S}_T$ is defined as a pair $(\mathcal{S}_T, F(\mathbf{v}))$ where $F(\mathbf{v})$ is a state formula describing a (possibly infinite) set of states, called *final*, *error* or *goal* states. The solution of this problem exists, i.e., a set of states represented by $F(\mathbf{v})$ is reachable by $\mathcal{S}_T$ if it exists $n \geq 0$ such that

$$I(\mathbf{v}^{(0)}) \wedge \tau(\mathbf{v}^{(0)}, \mathbf{v}^{(1)}) \wedge \cdots \wedge \tau(\mathbf{v}^{(n-1)}, \mathbf{v}^{(n)}) \wedge F(\mathbf{v}^{(n)}) \tag{1}$$

is T-satisfiable. When, besides the transition system $\mathcal{S}_T$ and the final formula $F(\mathbf{v})$, a bound $\overline{n} \geq 0$ is known, we speak of the *bounded reachability problem* for $\mathcal{S}_T$. A solution to this problem exists if there exists $0 \leq n \leq \overline{n}$ such that (1) is T-satisfiable. In the following, we focus on unbounded reachability problems since their bounded version has received a lot of attention in the SMT literature; e.g., [16] discusses the problem of using the SMT-LIB language to specify transition systems in the context of Bounded Model Checking (BMC) and, more recently, [17] puts forward the need for a common standard for BMC problems. Indeed, our proposal can be easily adapted to bounded reachability problems.

A well-known method (see, e.g., [19]) to solve the reachability problem amounts to repeatedly computing pre-images (post-images) of a final (initial) formula with respect to the transition formula. Formally, given a set $s(\mathbf{v})$ of states and a transition formula $\tau(\mathbf{v}, \mathbf{v}')$, the post-image of $s(\mathbf{v})$ w.r.t. $\tau(\mathbf{v}, \mathbf{v}')$ is the formula $\exists \mathbf{v}'.(s(\mathbf{v}') \wedge \tau(\mathbf{v}', \mathbf{v}))$, while the pre-image is the formula $\exists \mathbf{v}'.(s(\mathbf{v}') \wedge \tau(\mathbf{v}, \mathbf{v}'))$. The reachability procedure terminates in two cases. The former is when a fix-point is reached, i.e. when the set of states described by the pre-image (post-image, resp.) computed at the n-th iteration is a sub-set of the set of states represented by the union of all the pre-images (post-image, resp.) computed in previous iterations (this is called a fix-point test). The latter is when the intersection of the currently computed pre-image (post-image, resp.) with the set of initial (final, resp.) states is non-empty (this is called a safety test). Roughly, the idea is to unwind the transition formula until either all reachable states of $\mathcal{S}_T$ have been explored or an instance of (1) is found to be satisfiable for a certain n.[2] Formally, the intersection between two set $s_1(\mathbf{v})$ and $s_2(\mathbf{v})$ of states is empty iff the state formula $s_1(\mathbf{v}) \wedge s_2(\mathbf{v})$ is T-unsatisfiable and the set $s_1(\mathbf{v})$ is a sub-set of the set $s_2(\mathbf{v})$ iff the state formula $s_1(\mathbf{v}) \wedge \neg s_2(\mathbf{v})$ is T-unsatisfiable. We observe that, when the set $\mathbf{v}$ of state variables contains function and predicate symbols, the formulae defining post- and pre-images are second-order formulae. To avoid T-satisfiability checks involving second-order formulae, we assume that it is possible to compute first-order formulae that are equivalent to the second order post- or pre-images so that safety and fix-point tests can be mechanized by currently available SMT solvers. In many cases of practical interest, this assumption holds as there is a close relationship between second order logic and many-sorted first-order logic (see [8] for details). An alternative to stay in the realm of first-order logic is to encode functions and predicates into the theory of arrays as done, for example, in [12].

A naive implementation of the reachability procedure is based on a client-server architecture. The client repeatedly computes pre- or post-images and generates the proof obligations encoding fix-point and safety checks. The server is an SMT solver capable of discharging the proof obligations generated by the client. In theory, suitable constraints on the background theories and the shape of the formulæ in the SMT-based transition system should be identified to

[2] Other techniques based on overapproximations of reachable states have been recently proposed [5].

guarantee the effectiveness of the reachability procedure, such as decidability of the satisfiability problems encoding safety and fix-point tests, and representability of the fix-point within a class of formulæ for termination. In practice, suitable algorithms and heuristics are needed to scale up to the verification of interesting systems. In these respects, a classification of SMT-based transition systems would be extremely helpful to help tools select the right techniques to efficiently explore the search space of a reachability problem.

Example 1. Consider a simplified variant of the Bakery algorithm in which a finite (but unknown) number of processes should be granted mutual exclusion to a critical section by using tickets that uniquely identify processes. Processes are arranged in an array whose indexes (i.e. tickets) are linearly ordered and each process can be in one of three states: idle, wait, critical. At the beginning, all processes are in the idle state. There are three possible transitions involving a single process with index z (in all transitions, the processes with index different from z remain in the same state): (τ_1) z is in idle, all the processes to its left are idle, and z moves to wait; (τ_2) z is in wait, all the processes to its right are idle, and z moves to critical; and (τ_3) z is in critical and moves to idle. The system should satisfy the following mutual exclusion property: there are no two distinct processes in the critical section at the same time.

We define a background theory $T_B = (\Sigma_B, \mathcal{C}_B)$ as follows. Σ_B contains *Ind* and *Loc* as sort symbols, three constant symbols i, w, and c of sort *Loc*, and a predicate symbol $<$ (written infix) of arity $Ind \times Ind$. A model in $\mathcal{C}_B$ interprets $<$ as a linear order over the elements in the interpretation of *Ind* and *Loc* is a set of three elements, the interpretations of the constants i, w, and c.

The SMT-based symbolic transition system $(\mathbf{v}_B, I_B, \tau_B)$ is defined as follows: $\mathbf{v}_B$ is a singleton containing the function symbol a of arity $Ind \to Loc$, I_B is the formula $\forall z.a(z) = i$, and τ_B is the disjunction of the following three formulae:

$$\begin{array}{lllllll}
\tau_1(a,a') & := & \exists z.\ (& a(z) = i & \wedge & \forall w.w < z \to a(w) = i & \wedge \quad \forall j.a'(j) = ite(j = z, w, a(j)) \ ) \\
\tau_2(a,a') & := & \exists z.\ (& a(z) = w & \wedge & \forall w.z < w \to a(w) = i & \wedge \quad \forall j.a'(j) = ite(j = z, c, a(j)) \ ) \\
\tau_3(a,a') & := & \exists z.\ (& a(z) = c & & & \wedge \quad \forall j.a'(j) = ite(j = z, i, a(j)) \ ),
\end{array}$$

where z, w, j are variables of sort *Ind*.

Finally, the error formula F can be written as $\exists z_1, z_2.(z_1 \neq z_2 \wedge a(z_1) = c \wedge a(z_2) = c)$, where z_1, z_2 are variables of sort *Ind*.

Notice that $T_B^{\{a\}}$ is the theory obtained by extending T_B with the function symbol a that is to be interpreted as a function mapping elements of the interpretation of *Ind* to elements of the interpretation of *Loc*. Alternative formalization are possible. For example, we could have used the theory of arrays whose indexes are of sort *Ind* and whose elements are of sort *Loc*. The state variable a is simply an array constant and an expression of the form $\forall j.a'(j) = ite(j = z, c, a(j))$ in the τ_j's above could be written as $a' = store(a, j, c)$ for c a constant of sort *Loc*. □

3 Proposal

We describe an extension of the SMT-LIB language that is suitable for specifying reachability problems along the lines of Section 2. Our proposal extends the set of SMT-LIB v.2 script commands in order to define the background theory, the state variables, the initial, transition, and goal formulae that together uniquely identify a reachability problem. As in the SMT-LIB standard, a script is used to communicate with a tool in a read-eval-print loop in such a way that the next command is parsed, executed, and a response message printed until all commands are considered. Possible responses may vary from a single symbol (e.g., `reachable`, `unreachable`,

or `unknown`) to complex expressions like a (finite) sequence of formulae encoding the run of the system leading it from a state satisfying the initial formula to one satisfying the final formula.

In the following, we fix an SMT-based symbolic transition system $\mathcal{S} = (\mathbf{v}, I(\mathbf{v}), \tau(\mathbf{v}, \mathbf{v}'))$ over a background theory $T = (\Sigma, \mathcal{C})$ and a final formula $F(\mathbf{v})$. Below, we let $\langle symbol \rangle$, $\langle numeral \rangle$, $\langle sort \rangle$, $\langle sorted\ var \rangle$, and $\langle term \rangle$ be syntactic categories inherited from the SMT-LIB v.2 standard (see [4] for the definition).

The background theory. The command

`(set-theory` $\langle symbol \rangle$ `)`

allows us to specify the "kernel" of the background theory T identified by $\langle symbol \rangle$ and available among those defined in the SMT-LIB v.2 standard, e.g., `Core`, `Ints`, and `ArraysEx`.

Remark 1. In the SMT-LIB standard, a benchmark problem is associated with a logic that identifies both a class of models and a set of formulae. The main reason is that there may exist sets of formulae of a theory that admits more efficient satisfiability procedures than other sets. For example, the satisfiability of conjunctions of difference logic constraints over the integers in which no negated equalities occur can be checked in polynomial time whereas the satisfiability of arbitrary difference logic constraints becomes NP-complete. Instead, for a reachability problem we specify a background theory (via the command `set-theory`) according to Section 2.

Our motivation is the following: there exist techniques (see, e.g., [2] for an example in a framework which is similar to the one proposed here) to over-approximate the set of reachable states with simple formulae (e.g., containing only existential quantifiers) despite the fact that the formulae in the SMT-based transition system are more complex (e.g., they contain both existential and universal quantifiers as the disjuncts τ_1 and τ_2 of the transition formula in Example 1). If we used such an over-approximation of the fix-point in the safety test and this is negative (i.e. the intersection is empty), we are entitled to conclude that also the safety test with the exact fix-point is negative and the final formula is unreachable. Other techniques (see, e.g., [7, 14]) allows one to transform a complex reachability problem expressed with formulae containing quantifiers into one in which only quantifier-free formulae occur and such that the former admits a solution if the latter does so. For example, the counting abstraction technique in [7] transforms the formulae of cache coherence protocols into quantifier-free formulae of Linear Arithmetic over the integers. The crucial advantage of this kind of techniques is a dramatic simplification of the formulae encoding both the safety and the fix-point tests. As a consequence, the satisfiability problems may become significantly simpler (sometimes, the result is even to transform the problem into an equivalent one falling into a decidable class).

From the viewpoint of the specification of reachability problems, it is difficult (if possible at all) to foresee if the techniques discussed above can be applied and even what kind of transition system they produce. In fact, not only the shape of the formulae may change but also the background theory of the transformed transition system can be different as the example of the counting abstraction shows. Deeper insights into the techniques used by reachability procedures based on SMT techniques must be gained in order to design a more precise specification language. □

The kernel theory can be extended by declaring additional sort and function symbols by using standard SMT-LIB v.2 commands, such as `declare-sort`, `define-sort`, `declare-fun`, and `define-fun` (see [4] for their definition). The declared additional symbols are indeed uninterpreted and to constrain their interpretations, we introduce the command

`(declare-axiom` $\langle term \rangle$ `)`

that defines the axiom encoded by the `Boolean` term *term* and extends the kernel theory. We assume *term* to be a sentence of the background theory of the transition system, i.e. it contains

no state variables.

Example 2. Consider the theory T_B in Example 1. It can be seen as an extension of the theory `Core` with the sort symbols *Ind* and *Loc*, the weak `leq` and strict `le` predicate symbols for the linear order, and four axioms to constrain their interpretation:

```
(set-theory Core)
(declare-sort Ind) (declare-fun leq (Ind Ind) Bool) (declare-fun le (Ind Ind) Bool)
(declare-axiom (forall ((?x Ind)) (leq ?x ?x)))
(declare-axiom (forall ((?x Ind) (?y Ind)) (=> (and (leq ?x ?y) (leq ?y ?x))
                                               (= ?x ?y))))
(declare-axiom (forall ((?x Ind) (?y Ind) (?z Ind)) (=> (and (leq ?x ?y) (leq ?y ?z))
                                                        (leq ?x ?z))))
(declare-axiom (forall ((?x Ind) (?y Ind)) (or (leq ?x ?y) (leq ?y ?x))))
(declare-axiom (forall ((?x Ind) (?y Ind)) (= (le ?x ?z)
                                              (and (leq ?x ?y) (not (= ?x ?y))))))
(declare-sort Loc)
(declare-fun i () Loc) (declare-fun w () Loc) (declare-fun c () Loc)
(declare-axiom (and (not (= i w)) (not (= i c)) (not (= w c))))
(declare-axiom (forall ((?x Loc)) (or (= ?x i) (= ?x w) (= ?x c))))
```

The first three axioms express the fact that `leq` is a linear order and the fourth axiom defines its strict version `le`. The last two axioms above constrain the interpretation of `Loc` to be a set containing exactly the interpretation of the constants `i`, `w`, and `c`. □

Since constraining the interpretation of a sort to be a finite set (as it is the case of *Loc* in the example above) is quite common, we introduce the command

`(define-subrange` ⟨*symbol*⟩ `(`⟨$numeral_1$⟩ ⟨$numeral_2$⟩ `))`

that introduces the sort *symbol* and the numerals in the interval $[numeral_1 \ \ldots \ numeral_2]$ as constants of that sort that are to be interpreted as distinct elements. For instance, we can replace the declarations of the sort *Loc*, the constants `i`, `w`, `c`, and the two axioms in the example above with `(define-subrange Loc (1 3))` and assume that the constant `i` is mapped to the numeral `1`, `w` to `2`, and `c` to `3`. In other words, the command `define-subrange` defines an enumerated datatype over a certain sort whose constants are identified as the numerals in a contiguous sub-set of the naturals. An alternative would be a command to define an enumerated datatype by listing all its elements. In case of large sets of elements, the drawback is that explicit enumerations can be rather tedious. Instead, the corresponding definition by the `define-subrange` command is very compact.

Remark 2. The possibility of defining theories by using finitely many axioms was available in SMT-LIB v.1.2 [18] but is no more so in SMT-LIB v.2 [4]. This is justified by the observation that many theories in the standard require infinitely many axioms [4]. In our experience with the model checker MCMT [13] and its successor SAFARI [1], the flexibility of defining theories by finitely many axioms is crucial to identify classes of the reachability problem that guarantee the decidability of the satisfiability problems encoding safety and fix-point tests or the termination of the reachability procedure. Our experience was driven by the theoretical framework developed in [12] that allowed us to identify sufficient conditions for the mechanization and termination of a class of SMT-based transition systems, called array-based systems. A key ingredient of such conditions is a class of background theories that are (*i*) expressive enough to encode practically interesting transition systems and (*ii*) "simple" enough to guarantee both the decidability of the safety and fix-point tests and sometimes also the termination of (backward) reachability. Roughly, the axioms needed for such "simple" background theories require

just a finite set of predicate symbols and a finite set of universal sentences (i.e. sentences of the form $\forall \underline{x}.\varphi(\underline{x})$ for $\underline{x}$ a finite tuple of variables of appropriate sort and φ a quantifier-free formula). The theory T_B in Example 1 (see also the box in Example 2) is an instance of a "simple" theory in this sense.

From a practical point of view, an obvious question arises: can available SMT solvers cope with such a flexible way of defining theories? The answer is offered by the proposal of several quantifier instantiation procedures (see, e.g., [10] and the references therein) that are quite successful in checking the satisfiability of formulae containing quantifiers. An *ad hoc* technique combining quantifier instantiation and quantifier-free reasoning has been proposed together with some heuristics [11] to discharge the proof obligations arising in the backward reachability of array-based systems. An alternative to quantifier instantiation procedures would be to encode the additional symbols in complex theories for which satisfiability procedures are available. For instance, the standard less-than relation in the theory `Ints` of the standard SMT-LIB v.2 (see page 31 of [18]) can be taken as the linear order $<$ of Example 1; many state-of-the-art SMT solvers provide support for the fragment of `Ints` containing the less-than relation.

To summarize, a careful use of the command `declare-axiom` can have two advantages. First, it is possible to precisely define background theories that allow for proving important properties of classes of reachability problems. Second, reasoning modulo such background theories can be efficiently supported by available SMT solvers when the declared axioms allow the maximal reuse of available procedures, possibly leveraging recent advances in quantifier instantiation procedures. □

The SMT-based transition system. Once the background theory T has been declared, we need to specify the SMT-based transition system $\mathcal{S} = (\mathbf{v}, I(\mathbf{v}), \tau(\mathbf{v}, \mathbf{v}'))$. The command

`(declare-state-var` ⟨*symbol*⟩ `(` ⟨*sort*⟩* `)` ⟨*sort*⟩ `)`

declares the state variable *symbol* together with its arity. This is similar to the behavior of the command `declare-fun` in the SMT-LIB v.2 standard with a key difference: `declare-state-var` also declares a composed symbol

`(primed` ⟨*symbol*⟩`)`

with the same arity of *symbol*. This is the crucial syntactic extension to the SMT-LIB v.2 standard, required to create a relationship between the state variable *symbol* in $\mathbf{v}$ and its copy *symbol*$'$ in $\mathbf{v}'$. The former identifies the value of the state variable immediately before the execution of a transition whereas the latter the value immediately after.

The command

`(declare-initial` ⟨*term*⟩ `)`

defines the state formula encoded by the `Boolean` term *term* characterizing the set of initial states, and the command

`(declare-transition` ⟨*term*⟩ `)`

defines one disjunct of the transition formula encoded by the `Boolean` term *term*. For the command `declare-initial`, the term *term* may contain symbols of the background theory and the state variables; for the command `declare-transition`, the term may also contain the primed version of the state variables.

Example 3. Let us consider the SMT-based transition system in Example 1. The state variable a in the example of Section 1 can be declared as

```
(declare-state-var a (Ind) Loc).
```

As a consequence of the execution of this command, not only the function symbol `a` of arity `(Ind) Loc` will be available but also the (atomic) symbol `(primed a)` with the same arity.

The state formula I_B in Example 1 can be declared as

```
  (declare-initial (forall ((?z Ind)) (= (a ?z) i)))
and
  (declare-transition (exists ((?z Ind)) (and (= (a ?z) c)
   (forall ((?w Ind)) (= ((primed a) ?w) (ite (= ?w ?z) i (a ?w)))))))) .
```

is the disjunct τ_3 of the transition formula τ_B. □

In our experience with MCMT and SAFARI, we have found it useful to describe the behavior of a transition system not only by transition formulae but also by system constraints, i.e. formulae that must be satisfied in every state of a run of the system. Intuitively, constraints usually encode invariants of the transition system that are enforced by the environment in which the system evolves or by the way in which the state variables are updated. For example, it is well-known that a Petri net can be specified by a SMT-based transition system over integer state variables counting the number of tokens in each place; a system constraint is that each state variable must be non-negative. For this reason, we introduce the command

`(declare-system-constraint` ⟨*term*⟩ `)`

that adds the constraint encoded by the `Boolean` term *term* to the specification of the transition system. We assume *term* to be a state formulae.

The final formula. Now that the transition system $\mathcal{S} = (\mathbf{v}, I(\mathbf{v}), \tau(\mathbf{v}, \mathbf{v}'))$ has been specified, we are ready to specify the final formula $F(\mathbf{v})$ to complete the specification of a reachability problem $(\mathcal{S}, F)$. The command

`(declare-goal` ⟨*term*⟩ `)`

sets the state formula encoded by the `Boolean` term *term* as the final (error or goal) formula.

Example 4. The final formula in Example 1 can be given as

```
(declare-goal (exists ((?z1 Ind) (?z2 Ind))
              (and (not (= ?z1 ?z2)) (= (a ?z1) c) (= (a ?z2) c)))).
```

□

When there more than one `declare-goal` command in a script, the goal formula is obtained by taking the disjunction of all their arguments.

Other commands. The command

`(check-reachability)`

actually tells the tool to check if the goal formula is reachable or not. The response to this command can be `reachable`, `unreachable`, or `unknown`. The last symbol can be returned when the tool has used over-approximation techniques.

When there are more than one `declare-goal` command, it is left to the tool to decide to consider each goal separately, all of them together, or some subset. It is known that it is usually better to consider several goals at the same time.

Sometimes, the reachability problem is so complex that it is better to guide the tool towards the fact that a certain goal is unreachable by constructing an appropriate sequence $(\mathcal{S}, F_1), ..., (\mathcal{S}, F_{n-1}), (\mathcal{S}, F_n)$ of reachability problems where F_n encodes the complement of the safety property that $\mathcal{S}$ should satisfy while the complement of $F_1, ..., F_{n-1}$ are invariants of $\mathcal{S}$ that may hopefully contribute to show that F_n is unreachable. For example, if F_1 has been found unreachable, then its negation can be used in the fix-point tests when trying to establish

the unreachability of F_2 by checking the satisfiability of $s_1(\mathbf{v}) \wedge \neg F_1(\mathbf{v}) \wedge \neg s_2(\mathbf{v})$ rather than simply $s_1(\mathbf{v}) \wedge \neg s_2(\mathbf{v})$ for s_1, s_2 state formulae describing the sets of reachable states at two successive iterations of the reachability procedure. Example of this technique can be found in [2, 6].

Indeed, building the right sequence of reachability problems that allow one to prove a certain safety problem is a difficult task that may well depend on the heuristics used by a tool to exploit previously solved reachability problems. To simplify this kind of interaction, we believe it is useful to have commands `push` and `pop` as in the SMT-LIB v.2 standard that allows one to handle a stack of reachability problems. In order to keep those goals whose negation will constitute a useful invariant for the proof of a certain safety property, we provide the command

`(save-verified-goals)`

that permits to save the goals that have been shown unreachable so as to prevent their deletion by some `pop` command.

We also envisage other commands for setting the parameters of reachability solver. The command

`(set-option` ⟨*reachability_option*⟩ `)`

may set how the search space should be computed (e.g., depth- or breadth-first) or heuristics for the synthesis of invariants should be turned on. A value for *reachability_option* can be

`:produce-counterexample` ⟨*b_value*⟩

that tells the prover to return (or not according to the fact that *b_value* is true or false) a sequence of transition formulae that leads the system from a state satisfying the inital formula to one satisfying the final formula. Indeed, every tool may define its own set of reachability options. The command

`(set-smt-option` ⟨*option*⟩ `)`

may suggest a set the options to the background SMT solver that should be used to discharge the proof obligations encoding the safety or the fix-point tests where ⟨*option*⟩ is inherited from the SMT-LIB v.2 standard.

Every reachability problem must end with the command

`(exit)`

4 Conclusion

We have presented an extension of the SMT-LIB which is suitable for the description of reachability problems for a large class of heterogeneous transition systems. The problem of finding a common standard is in dire need of solutions for many reasons, most importantly the inability in performing comparison between competitor systems and the lack of a library for collecting benchmarks. Further discussions on this problem are surely needed, and we hope the SMT community will benefit from the ideas presented in this paper.

References

[1] Francesco Alberti, Roberto Bruttomesso, Silvio Ghilardi, Silvio Ranise, and Natasha Sharygina. Lazy abstraction with interpolants for arrays. In *LPAR*, pages 46–61, 2012.

[2] Francesco Alberti, Silvio Ghilardi, Elena Pagani, Silvio Ranise, and Gian Paolo Rossi. Universal guards, relativization of quantifiers, and failure models in model checking modulo theories. *JSAT*, 8(1/2):29–61, 2012.

[3] Michael Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs, and K. Rustan M. Leino. Boogie: A modular reusable verifier for object-oriented programs. In *FMCO*, pages 364–387, 2005.

[4] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. `www.SMT-LIB.org`, 2010.

[5] Aaron R. Bradley and Zohar Manna. Checking safety by inductive generalization of counterexamples to induction. In *FMCAD*, pages 173–180. IEEE Computer Society, 2007.

[6] Roberto Bruttomesso, Alessandro Carioni, Silvio Ghilardi, and Silvio Ranise. Automated analysis of timing based mutual exclusion algorithms. In *NASA FM*, 2012.

[7] Giorgio Delzanno. Automatic Verification of Parameterized Cache Coherence Protocols. In *CAV*, LNCS, 2000.

[8] Herbert B. Enderton. *A mathematical introduction to logic*. Harcourt/Academic Press, 2001.

[9] Jean-Christophe Filliâtre and Claude Marché. The Why/Krakatoa/Caduceus Platform for Deductive Program Verification. In *CAV*, pages 173–177, 2007.

[10] Yeting Ge and Leonardo de Moura. Complete instantiation for quantified SMT formulas. In *Conference on Computer Aided Verification (CAV)*, 2009.

[11] Silvio Ghilardi and Silvio Ranise. Model Checking Modulo Theory at work: the integration of Yices in MCMT. In *AFM*, 2009.

[12] Silvio Ghilardi and Silvio Ranise. Backward Reachability of Array-based Systems by SMT solving: Termination and Invariant Synthesis. *LMCS*, 6(4), 2010.

[13] Silvio Ghilardi and Silvio Ranise. MCMT: A Model Checker Modulo Theories. In *IJCAR*, pages 22–29, 2010.

[14] Sumit Gulwani and Madan Musuvathi. Cover Algorithms and their Combination. In *ESOP*, LNCS, 2008.

[15] Holger H. Hoos and Thomas Stützle. SATLIB: An Online Resource for Research on SAT. In I. P. Gent, H. v. Maaren, and T. Walsh, editors, *SAT*, pages 283–292. IOS Press, 2000.

[16] Tim King and Clark Barrett. Exploring and Categorizing Error Spaces using BMC and SMT. In *Proc. of the 9th Int. Workshop on Satisfiability Modulo Theories (SMT)*, 2011.

[17] Akash Lal, Shaz Qadeer, and Shuvendu K. Lahiri. Corral: A Solver for Reachability Modulo Theories. In *Proc. of Computer-Aided Verification (CAV)*, 2012.

[18] Silvio Ranise and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2006.

[19] Tatiana Rybina and Andrei Voronkov. A logical reconstruction of reachability. In *Ershov Memorial Conf.*, volume 2890 of *LNCS*, pages 222–237. Springer, 2003.

[20] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.