VMT-LIB Format Specification

Version 0.1

The VMT format is an extension of the SMT-LIBv2 [1] (SMT2 for short) format to represent symbolic transition systems. This document describes the syntax of a VMT file and its semantics.

1 Theoretical Background and Definitions

Our setting is many-sorted first order logic. We use the standard notions of theory, satisfiability, validity, and logical consequence. We refer to the SMT-LIB specifications [1] for more details. We denote generic theories as $\mathcal{T}$. We write $\varphi \models_{\mathcal{T}} \psi$ to denote that the formula ψ is a logical consequence of φ in the theory $\mathcal{T}$; when clear from context, we omit $\mathcal{T}$ and simply write $\varphi \models \psi$.

We refer to 0-arity predicates as Boolean variables, and to 0-arity uninterpreted functions as (theory) variables.

Given a set of variables X, a signature Σ, a domain M, an interpretation function $\mathcal{I}$ of the symbols in Σ on the domain M, an assignment σ to the variables in X on the domain M, and a σ-formula $\phi(X)$ with free variables in X, the satisfaction relation $\langle M, \mathcal{I} \rangle \models \phi$ is defined in the usual way.

For each variable x, we assume that there exists a corresponding variable x', called the *primed version* of x. If X is a set of variables, X' is the set obtained by replacing each element x with its primed version ($X' = \{x' \mid x \in X\}$). φ' is the formula obtained by replacing each occurrence variable in φ with the corresponding primed.

In the following, the signature Σ and the theory $\mathcal{T}$ are implicitly given. A *transition system (TS)* S is a tuple $\langle X, I(X), T(Y, X, X') \rangle$ where X is a set of *state* variables, $I(X)$ is a formula representing the initial states, and $T(Y, X, X')$ is a formula representing the transitions, where Y is a set of *input* variables.

2 Syntax

VMT exploits the capability offered by the SMT2 language of attaching *annotations* to terms and formulas in order to specify the components of the transition system and the properties to verify. More specifically, the following annotations are used:

`:next name` is used to represent state variables. For each variable x in the model, the VMT file contains a pair of variables, x^c and x^n, representing respectively the current and next version of x. The two variables are linked by annotating x^c with the attribute `:next` x^n. All the variables that are not in relation with another by means of a `:next` attribute are considered inputs.

`:init true` is used to specify the formula for the initial states of the model. This formula should contain neither next-state variables nor input variables. (The "dummy" value `true` in the annotation is needed because the current SMT2 standard requires annotations to always have an associated value.)

`:trans true` is used to specify the formula for the transition relation.

`:invar-property idx` is used to specify invariant properties, i.e. formulas of the form Gp, where p is the formula annotated with `:invar-property`. The non-negative integer *idx* is a unique identifier for the property.

`:live-property idx` is used to specify an LTL property of the form $\mathbf{FG}p$, where p is the formula annotated with `:live-property`. The non-negative integer *idx* is a unique identifier for the property.

In a VMT file, only annotated terms and their sub-terms are meaningful. Any other term is ignored. Moreover, only the following commands are allowed to occur in VMT files: `set-logic`, `set-option`, `declare-sort`, `define-sort`, `declare-fun`, `define-fun`.(For convenience, an additional `(assert true)` command is allowed to appear at the end of the file.)
The following example shows a simple model in the syntax of nuXmv (left) and its corresponding VMT translation (right).

nuXmv	VMT

`-- this is a comment`
`MODULE main`
`VAR x : integer;`
`IVAR b : boolean;`
`INIT x = 1;`
`TRANS`
`next(x) = b ? x + 1 : x;`
`INVARSPEC x > 0;`
`LTLSPEC FG x > 10;`
	`; this is a comment`
	`(declare-const x Int)`
	`(declare-const x.next Int)`
	`(define-fun sv.x () Int (! x :next x.next))`
	`(declare-const b Bool)`
	`(define-fun init () Bool`
	`(! (= x 1) :init true))`
	`(define-fun trans () Bool`
	`(! (= x.next (ite b (+ x 1) x)) :trans true)`
	`(define-fun p1 () Bool`
	`(! (> x 0) :invar-property 1))`
	`(define-fun p2 () Bool`
	`(! (> x 10) :live-property 2))`

Since the SMT2 format (and thus also the VMT one that inherits from SMT2) does not allow to annnotate the declaration of variables, it is a good practice to insert immediately after the declaration of the variables a set of defines to specify the relations among variables. See for instance the define `sv.x` in the example above that introduces the relation between `x` and `x.next`.

3 Semantics

Given a transition system $S \doteq \langle X, I(X), T(Y, X, X') \rangle$ over a background theory T with a signature Σ and an interpretation $\mathcal{I}$, a *state* s of S is an interpretation of the state variables X. A (finite) *path* of S is a finite sequence $\pi \doteq s_0, s_1, \ldots, s_k$ of states, with the same domain and interpretation of symbols in the signature Σ, such that $\mathcal{I}, s_0 \models I(X)$ and for all i, $0 \leq i < k$, $\mathcal{I}, s_i, s'_{i+1} \models \exists Y.T(Y, X, X')$. We say that a state s is reachable in S iff there exists a path of S ending in s.

Invariant Properties

An *invariant property* p is a symbolic representation of a set of states that must be a superset of the reachable states of S. In other words, $S \models p$ iff $\forall s.s$ is reachable in S, $s \models p$. Consequently, a *counterexample* for p is a *finite path* $s_0, \ldots, s_k$ of S such that $s_k \models \neg p$.

Live Properties

A *live property* p represents a set of states that is *eventually invariant*. In LTL syntax, it would be denoted with $\mathbf{FG}p$. More formally, $S \models p$ iff for all paths $s_0, \ldots, s_i, \ldots$, $\exists i. \forall j > i. s_j \models p$. (Notice that finite paths $s_0, \ldots, s_k$ vacuously satisfy a live property, because we can always take $i = k$ to satisfy the previous definition.) Consequently, a *counterexample* for p is an *infinite path* $s_0, \ldots, s_i, \ldots$ of S such that $\forall i. \exists j > i. s_j \models \neg p$.

References

[1] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. `https://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf`