From Event-B lemmas to SMT-LIB benchmarks

Systerel

May 7, 2009

Abstract

This document provides a specification to translate mathematical lemmas that rely on the Event-B[4] language to the SMT-LIB[5] format.

The considered lemmas are those often encountered in practice, but not yet well supported by proving tools such as those embedded in the *Rodin*[1] platform.

As a consequence, the purpose is not here to entirely define the translation, but rather to give a mapping for this subset of Event-B formulas. These typical mathematical problems are entry points for the DECERT[2] project, and more precisely for the Work Package 1. Using the SMT-LIB standard is among the requirements for this project.

This version of the document mainly focuses on lemmas based on unquantified integer linear arithmetic.

Contents

1 Introduction

The document will first give a high-level specification to establish a link between mathematical lemmas built from proof obligations and SMT-LIB benchmarks to be checked for satisfiability, with respect to background theories. The theories considered here are characterized in a separated document dedicated to the taxonomy[3] of lemmas. These lemmas are essentially issued from the B Book[1] or from the Rodin platform archives, and are identified as being difficult to prove.

Then, the document will expose some rewriting rules to convert lemma hypotheses and goals, which both are logical formulas expressed in the Event-B language, to the SMT-LIB format.

[1] `http://www.event-b.org`
[2] `http://decert.gforge.inria.fr/index.html`

Finally, the document will give some implementation choices to integrate the transformation previously described in the Rodin environment, in a structured manner.

2 High-level specification of the translation

The mathematical lemmas considered in this document are nothing less than proof obligations, where the hypotheses appear as assumptions and the conclusion appears as the proof goal. The hypotheses and proof goals are predicates, as defined in the Event-B language.

A lemma can be related to one or several theories, as explained in the taxonomy[3].

2.1 Terminology

The correspondence between the Event-B terminology and the SMT-LIB terminology is summarized in Table 1.

Event-B	**SMT-LIB**
Predicate	Formula
Expression	Term
Type	Sort

Table 1: Terminology

2.2 Syntax for lemmas

The syntax of the lemmas, in Extended Backus-Naur Form (EBNF), is described below:

$\langle lemma \rangle$ ::= $\langle theories \rangle \langle typenv \rangle \{ \langle hypothesis \rangle \} \langle goal \rangle$

$\langle theories \rangle$ ::= $\langle \{ theory \} \rangle$

$\langle theory \rangle$ ::= *lists*
| arrays
| basic_set
| basic_relation
| full_set_theory
| integer
| linear_order_int
| linear_arith
| nonlinear_arith
| full_arith
| boolean

$\langle typenv \rangle$ ::= $\{ \langle variable \rangle \}$

$\langle variable \rangle$ ::= $\langle name \rangle \langle type \rangle$

$\langle name \rangle$ ::= an identifier, as defined in the §2.2 of the Event-B mathematical language specification[4]

$\langle type \rangle$ ::= a type, as defined in §4.3.1 of[4]

$\langle hypothesis \rangle$ ::= a predicate, as defined in §3.2.4 of[4]

$\langle goal \rangle$ ::= a predicate, as defined in §3.2.4 of[4]

2.3 From lemmas to benchmarks

It is possible to build a mapping between the lemmas and the SMT-LIB benchmarks (see Figure 13 in [5]): an hypothesis of a lemma is to be matched to a benchmark **assumption**; the goal of a lemma is to be matched to a benchmark **formula**; the typing environment is to be matched to benchmarks **extrasorts**, **extrapreds**, **extrafuns** and/or **assumption**.

Quantifier-free formulas are not allowed in SMT-LIB. To circumvent this restriction a first solution would be to close formulas existentially: each formula φ with free variables $\{x_1, ..., x_n\}$ is rewritten as $\exists x_1 : s_1 ... x_n : s_n \varphi$.

Another solution - the solution chosen in this document - is to replace each free term variable by a fresh constant symbol of the appropriate sort. Such symbols are declared with the **extrafuns** attribute. In parallel, each free occurrence of a formula variable is replaced by a fresh predicate symbol of empty arity (i.e. a propositional variable). Such symbols are declared with the **extrapreds** attribute.

More precisely, in a logic where the Int sort is defined (e.g. the logic QF_LIA for unquantified integer linear arithmetic), the following rules apply for the typing environment $\{t \mapsto T\}$, where identifier t has type T:

- If T is the $\mathbb{Z}$ predefined type of the Event-B mathematical language, the typing environment is represented with the : $extrafuns$ $((t\ Int))$ SMT-LIB declaration.
- If T is the BOOL predefined type, it is represented with the : $extrafuns$ $((t\ Bool))$ declaration.

For example, if the goal for a lemma is $0 < n + 1$, under the hypothesis $n \in \mathbb{N}$, in the $\{n \mapsto \mathbb{Z}\}$ typing environment, the associated benchmark is structured as detailed below:

$$
\begin{array}{l}
(benchmark\ example.smt \\
\qquad : status\ sat \\
\qquad : logic\ QF\_LIA \\
\qquad : extrafuns\ ((n\ Int)) \\
\qquad : assumption\ (>=\ n\ 0) \\
\qquad : formula\ (<\ 0\ (+\ n\ 1)))
\end{array}
$$

2.4 Theories

The Ints theory (see Appendix A), that is dedicated to integer numbers, and the Bools theory (see Appendix B), that is dedicated to booleans, are the only applicable theories when considering mathematical lemmas based on the Event-B language .

2.5 Logics

Some sublogics of the main SMT-LIB logic (first-order logic with equality) are listed on the dedicated web page[3].

Thus, QF_LIA is for example the logic for unquantified integer linear arithmetic (i.e. boolean combinations of inequations between linear polynomials over integer variables). It refers to the Ints theory.

The linear_order_int and linear_arith theories described in the taxonomy[3] are to be matched to this logic. More generally, the theories presented in this document are to be translated to SMT-LIB logics.

3 Low-level specification of the translation

The conversion of the logical formulas from the Event-B language syntax to the SMT-LIB syntax is expressed in a left-to-right direction, with the $\rightsquigarrow$ symbol.

For example, the following conversion indicates that the SMT-LIB syntax for the $(x < y)$ formula is $(<\ x\ y)$:

$$(x < y) \rightsquigarrow (<\ x\ y)$$

3.1 From SET to FOL

A many-sorted formula, i.e. a formula containing set symbols (let's call it a SET formula), can be translated into an equivalent one belonging to first-order logic (FOL). It is enough to be fully convinced of such a transformation to think to the characteristic function, which takes the value 1 for elements in the set X and the value 0 for other elements. Thus, for any set E, let's introduce the unary predicate E_x such that E_x is true if and only if x belongs to E.

SET formula	FOL formula
$x \in \{x_1, ..., x_n\}$	$x = x_1 \lor ... \lor x = x_n$
$x \in \varnothing$	$\bot$
$x \in x_1..x_n$	$(x \geqslant x_1) \land (x \leqslant x_n)$
$x \in \mathbb{N}$	$x \geqslant 0$
$x \in \mathbb{N}_1$	$x > 0$

Table 2: From SET to FOL

The translation from a SET formula to a FOL formula is defined in Table 2, where E and F are sets. Both formulas are expressed in Event-B language syntax.

Thus, the SET formulas will not be taken into consideration when specifying the translation from the Event-B syntax to the SMT-LIB syntax.

3.2 From FOL to SMT-LIB

An association between formulas in SMT-LIB format and formulas in first-order logic is given in the §6 Semantics of the SMT-LIB specification[5].

3.3 From Event-B to SMT-LIB

It is possible to derivate a translation from event-B predicates to SMT-LIB formulas from the previous observations, as detailed in Table 3.

Event-B predicate	$\rightsquigarrow$	SMT-LIB formula
TRUE		TRUE
FALSE		FALSE
$P \land Q$		$(and\ P\ Q)$
$P \lor Q$		$(or\ P\ Q)$
$\neg(P)$		$(not\ P)$
$bool(P)$		$(ite\ P$ TRUE FALSE$)$
$P \Rightarrow Q$		$(implies\ P\ Q)$
$P \Leftrightarrow Q$		$(iff\ P\ Q)$
$\forall x.(P)$		$(forall\ (?x\ Int)\ P)$
$\exists x.(P)$		$(exists\ (?x\ Int)\ P)$
$x = y$		$(=\ x\ y)$
$x \neq y$		$(not\ (=\ x\ y))$
$x < y$		$(<\ x\ y)$
$x \leq y$		$(<=\ x\ y)$
$x > y$		$(>\ x\ y)$
$x \geq y$		$(>=\ x\ y)$

Table 3: From Event-B predicates to SMT-LIB formulas

In the same manner, the Event-B expressions can be transformed in SMT-LIB format (see Table 4).

Let F be a formula in Event-B language and $\breve{F}$ be the corresponding SMT-LIB formula.

3.4 Examples

Example 1 How to translate the following formula $0 \in 0..x$, where x is a natural number (i.e. $x >= 0$)?

The first step is to translate it as a FOL formula: $(0 >= 0) \land (0 <= x)$.

In Ints theory, where 0 is defined as being an integer and $>=$ and $<=$ are defined as predicates of arity 2 on integers, it is possible to write the corresponding SMT-LIB formula: $(and\ (>=\ 0\ 0)\ (<=\ 0\ x))$.

[3] `http://combination.cs.uiowa.edu/smtlib/`

Event-B expression	$\rightsquigarrow$	**SMT-LIB formula**
$x + y$		$(+\ x\ y)$
$x - y$		$(-\ x\ y)$
$-x$		$\sim x$
$x * y$		$(*\ x\ y)$
x/y		$(/\ x\ y)$
$x \bmod y$		$(\%\ x\ y)$

Table 4: From Event-B expressions to SMT-LIB formulas

As suggested in the previous examples, it seems pertinent to introduce an additional step before the final translation from FOL to SMT-LIB, which would be a simplification operation on FOL formulas. More precisely, there is no doubt that the $(0 >= 0) \wedge (0 <= x)$ formula can be easily simplified as $0 <= x$.

Such simplifications are not considered in this document, and are only introduced here as topic of interest for further revisions.

4 Implementation of the translation

The purpose of this section is to give a specification to translate Event-B Abstract Syntax Tree (AST) to SMT-LIB AST.

The `org.eventb.core.ast` plug-in for the Rodin platform contains a library to manipulate Event-B mathematical formulas as trees of nodes. In particular, it is possible to instantiate the `ISimpleVisitor` interface to go though a tree.

+ TODO: briefly describe the architecture.

It is possible to use the SMT-LIB parser/checker version 3.0 to validate the format of the produced benchmarks (see `http://combination.cs.uiowa.edu/smtlib/`).

5 Conclusion

TODO

References

[1] J.-R. Abrial. *The B-book: assigning programs to meanings.* Cambridge University Press, New York, NY, USA, 1996.

[2] Clément Hurlin. Reconstruction de preuves pour les formules quantifiées et ensemblistes. `http://www-sop.inria.fr/everest/Clement.Hurlin/publis/mr2.pdf`, June 2006. Mémoire de Master 2.

[3] Minh-Thang Khuu. Taxonomy of lemmas, 2009.

[4] Christophe Métayer and Laurent Voisin. The event-b mathematical language. `http://deploy-eprints.ecs.soton.ac.uk/11/4/kernel_lang.pdf`, 2009.

[5] S. Ranise and C. Tinelli. The SMT-LIB standard: Version 1.2. `http://goedel.cs.uiowa.edu/smtlib/papers/format-v1.2-r06.08.30.pdf`, 2006.

Appendix A: Ints Theory

```
(theory Ints

:written_by {Cesare Tinelli}
:date {08/04/05}
:updated {30/01/09}
:history {
```

```
30/01/09 removed erroneous :assoc attribute for binary minus
28/10/09 Slight changes to the ':notes' attribute
}

:sorts (Int)

:notes
"The (unsupported) annotations of the function/predicate symbols have
the following meaning:

attribute | possible value | meaning
-----------------------------------------------------
:assoc // the symbol is associative
:comm // the symbol is commutative
:unit a constant c c is the symbol's left and right unit
:trans // the symbol is transitive
:refl // the symbol is reflexive
:irref // the symbol is irreflexive
:antisym // the symbol is antisymmetric
"
:funs ((0 Int)
(1 Int)
(~ Int Int) ; unary minus
(- Int Int Int) ; binary minus
(+ Int Int Int :assoc :comm :unit {0})
(* Int Int Int :assoc :comm :unit {1})
)

:preds ((>= Int Int :refl :trans :antisym)
(< Int Int :trans :irref)
(>= Int Int :refl :trans :antisym)
(> Int Int :trans :irref)
)

:definition
"This is the first-order theory of the integers, that is, the set of all
the first-order sentences over the given signature that are true in the
structure of the integer numbers interpreting the signature's symbols
in the obvious way (with ~ denoting the negation and - the subtraction
functions).
"
:notes
"Note that this theory is not (recursively) axiomatizable in a first-order
logic such as SMT-LIB's underlying logic.
That is why the theory is defined semantically.
"
)
```

Appendix B: Booleans Theory

```
(theory Bools

:sorts (Bool)

:funs ((TRUE Bool)
(FALSE Bool)
)
```

```
:definition
"This is the theory of booleans.
It is formally and completely defined by the axioms below.
"
:axioms ((not (= TRUE FALSE))
(forall (?x Bool) (or (= x TRUE) (= x FALSE)))
)
)
```