A Theory of Finite Sets, Lists, and Maps for the SMT-LIB Standard

Daniel Kroening Philipp Rümmer Georg Weissenbacher

Oxford University Computing Laboratory

philr@comlab.ox.ac.uk

Rodin User and Developer Workshop
University of Southampton
15–17 July 2009

Outline

- Overview of SMT-LIB
- Proposal of new theories for SMT-LIB 2
  $\Rightarrow$ Primarily format, no tool
- Application to Event-B, VDM
- Practical and theoretical issues

More information, implementation (soon), paper:
`http://www.philipp.ruemmer.org`

The SMT-LIB Standard

SMT → **S**atisfiability **M**odulo **T**heories

SMT-LIB is . . .

- a standardised input format for SMT-solvers (since 2003)
- a standardised format for exchanging SMT problems
- a library of more than 60 000 SMT benchmarks
- the basis for the annual SMT competition
  (this year: on CADE)

Theories in SMT-LIB:

- integer and rational arithmetic (linear)
- uninterpreted functions
- arrays
- bit-vectors

The SMT-LIB Standard (2)

Some state-of-the-art SMT-solvers:

- Alt-Ergo, Argo-lib, Barcelogic, CVC3, DTP, Fx7, haRVey, MathSAT, Spear, STP, Yices, Z3
- All are completely automatic
- Standard architecture:
  DPLL + small theory engines + quantifier heuristics
- “Good for shallow reasoning”

- Used as back-ends in many verification systems:
  Krakatoa, Caduceus, ESC/Java2, Spec#, VCC, Havoc, CBMC, . . .

Example in SMT-LIB Format

```
(benchmark Ensures_Q_noinfer_2
:source { Boogie/Spec# benchmarks. }
:logic AUFLIA
[...]
:extrapreds (( InRange Int Int ))
:extrafuns (( this  Int ))
:extrafuns (( intAtLeast Int Int Int ))
[...]
:assumption
  (forall (?t Int) (?u Int) (?v Int)
   (implies (and (subtypes ?t ?u) (subtypes ?u ?v)) (subtypes ?t ?v))
   :pat (subtypes ?t ?u) (subtypes ?u ?v))
[...]
:formula
  (not (implies (implies (implies (implies
  (and
   (forall (?o Int) (?F Int)
    (implies (and (= ?o this) (= ?F X)) (= (select2 H ?o ?F) 5)))
   (implies
    (forall (?o Int) (?F Int)
     (implies (and (= ?o this) (= ?F X)) (= (select2 H ?o ?F) 5)))
    (implies true true)))
   (= ReallyLastGeneratedExit_correct Smt.true))
  (= ReallyLastGeneratedExit_correct Smt.true))
 (= start_correct Smt.true))
(= start_correct Smt.true))))
```

The SMT-LIB Format

SMT-LIB is currently quite low-level:

- No high-level datatypes like sets, lists, etc.

Solutions practically used:

- Much can be encoded in arrays + axioms ($+$ prover-specific extensions)
- Some solvers offer algebraic datatypes (not standardised)

$\Rightarrow$ Against the idea of SMT-LIB

The SMT-LIB Format (2)

- Current version of the standard: 1.2
- Version 2 to be finished sometime in 2009

New Features in Version 2

- Type constructors, parametric theories
- Various simplifications
- ...

- New theories! (hopefully)

Proposal for New SMT-LIB Theories

Datatypes inspired by VDM-SL

- Tuples
- (Finite) Lists
- (Finite) Sets
- (Finite) Partial Maps

Our main applications

- Reasoning + test-case generation for UML/OCL
- (Bounded) Model checking with abstract library models
- VDM-SL

Signature of the SMT-LIB Theories

Tuples	**Sets**	**Lists**	**Maps**
`(Tuple` T_1 ... T_n`)`	`(Set T)`	`(List T)`	`(Map S T)`

`tuple` $(x_1, \ldots, x_n)$
`project` x_k
`product` $M_1 \times \cdots \times M_n$
	`emptySet` $\emptyset$
	`insert` $M \cup \{x\}$
	`in` $\in$
	`subset` $\subseteq$
	`union` $\cup$
	`inter` $\cap$
	`setminus` $\setminus$
	`card` $\lvert M \rvert$
		`nil` $[\,]$
		`cons` $x :: L$
		`head`
		`tail`
		`append` $\curvearrowright$
		`length` $\lvert l \rvert$
		`nth` l_k
		`inds` $\{1, \ldots, \lvert l \rvert\}$
		`elems` $\{l_1, \ldots, l_{\lvert l \rvert}\}$
			`emptyMap` $\emptyset$
			`apply` $f(x)$
			`overwrite` $\oplus$
			`domain`
			`range`
			`restrict` $\lhd$
			`subtract` $\unlhd$

Example: Verification Cond. Generated by VDMTools

In VDM-SL notation:

$$\forall l : \mathbb{L}(\mathbb{Z}), i : \mathbb{N}.\ \big(i \in \mathrm{inds}(l) \Rightarrow \forall j \in \mathrm{inds}(l) \setminus \{i\}.\ j \in \mathrm{inds}(l)\big)$$

In SMT-LIB notation:

```
(forall ((l (List Int)) (i Int))
  (implies
     (and (>= i 0) (in i (inds l)))
     (forall (j Int)
       (implies
          (in j (setminus (inds l) (set i)))
          (in j (inds l)))))))
```

Event-B File System Case Study (delete/inv8)

$$\begin{array}{ll} & parent \in objects \setminus \{root\} \rightarrow objects, \\ & obj \in objects \setminus \{root\}, \quad des \subseteq objects, \\ & des = (tcl(parent))^{\sim}[\{obj\}], \quad objs = des \cup \{obj\} \\ \Rightarrow & objs \lhd parent \in (objects \setminus objs) \setminus \{root\} \rightarrow objects \setminus objs \end{array}$$

```
objects, des, objs : (Set OBJECT)
parent : (Map OBJECT OBJECT)
obj : OBJECT
```

```
(implies ... (and
   (= (domain (subtract parent objs))
      (setminus objects
                objs (insert emptySet root)))
   (subset (range (subtract parent objs))
           (setminus objects objs))
))
```

Application to Event-B Verification Conditions (2)

Translation of Event-B proof obligations

- Carrier sets $\longrightarrow$ SMT-LIB types
- Sets $\longrightarrow$ finite sets
- Functions $\longrightarrow$ finite partial maps or arrays

- SMT-LIB is strongly typed $\rightarrow$ type inference necessary
- Potential issue: finiteness of SMT-LIB datatypes

Status of the Proposal

- Syntax + Semantics of theories is formally defined
  $\Rightarrow$ In collaboration with Cesare Tinelli
  $\Rightarrow$ To be discussed at SMT workshop 2009
- Pre-processor is under development
  $\Rightarrow$ Converter SMT-LIB 2 $\rightarrow$ SMT-LIB 1
- Decidability is being investigated
- WANTED: benchmarks
  $\Rightarrow$ Necessary to get theories included in SMT-LIB standard
  $\Rightarrow$ Event-B benchmarks would be awesome!

Identified Sublogics (work in progress)

- Sets with cardinality: non-nested: decidable
  nested + quantifiers: undecidable
  nested, quantifier-free: ???
- Sets + Tuples: undecidable
- Lists with length: word equations with equal-length predicate, known open problem
- Finite Maps: ???
- Combined theories: undecidable

Initial Implementations (in progress)

- Sets with cardinality: arrays + axioms
- Tuples: algebraic datatype, or axioms
- Lists with length: algebraic datatype + axioms
- Finite Maps: arrays + axioms

Conclusion

Trade-off when defining theories:

- Generality $\rightarrow$ good for users
- Implementation complexity
- Decidability $\rightarrow$ good for tool writers

$\Rightarrow$ We hope that we have found a good compromise
$\Rightarrow$ Feedback is welcome!

Thanks for your attention!