SMT-LIB for HOL

Daniel Kroening Philipp Rümmer Georg Weissenbacher

Oxford University Computing Laboratory

ITP Workshop
MSR Cambridge
25 August 2009

The SMT-LIB Standard

SMT $\rightarrow$ **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: at CADE)

Theories in SMT-LIB:

- integer and rational arithmetic (linear)
- uninterpreted functions
- arrays
- finite-width 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 types like sets, lists, maps, 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)

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

Example: Verification Cond. Generated by VDMTools

In VDM-SL notation:

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

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)

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

```
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 $\rightarrow$ SMT-LIB types
- Sets $\rightarrow$ finite sets
- Functions $\rightarrow$ 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
  $\Rightarrow$ Was presented at SMT workshop 2009
- Pre-processor is under development
  $\Rightarrow$ Converter SMT-LIB 2 $\rightarrow$ SMT-LIB 1
- Decidability is being investigated

Proofs vs. Refutations

Refutations: SMT solvers produce satisfying assignments.

What about proofs?

- All SMT solvers use DPLL communicating with theory solvers
- Theory solvers can be made to produce deduction steps

$\Rightarrow$ Proof can be exported, checked by trusted kernel in ITP