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
  - 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 → Satisfiability Modulo Theories

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
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)

[...] :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))))
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)

⇒ Against the idea of SMT-LIB
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
<table>
<thead>
<tr>
<th>Tuples</th>
<th>Sets</th>
<th>Lists</th>
<th>Maps</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Tuple $T_1 \ldots T_n$)</td>
<td>(Set $T$)</td>
<td>(List $T$)</td>
<td>(Map $S \times T$)</td>
</tr>
<tr>
<td>tuple $(x_1, \ldots, x_n)$</td>
<td>emptySet $\emptyset$</td>
<td>nil $[]$</td>
<td>emptyMap $\emptyset$</td>
</tr>
<tr>
<td>project $x_k$</td>
<td>insert $M \cup {x}$</td>
<td>cons $x :: L$</td>
<td>apply $f(x)$</td>
</tr>
<tr>
<td>product $M_1 \times \ldots \times M_n$</td>
<td>in $\in M$</td>
<td>head</td>
<td>overwrite $\leftarrow$</td>
</tr>
<tr>
<td></td>
<td>subset $\subseteq M$</td>
<td>tail</td>
<td>domain</td>
</tr>
<tr>
<td></td>
<td>union $\cup M$</td>
<td>append $\leftarrow M$</td>
<td>range</td>
</tr>
<tr>
<td></td>
<td>intersect $\cap M$</td>
<td>length $</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>setminus $\setminus M$</td>
<td>nth $l_k$</td>
<td>subtract $\triangle$</td>
</tr>
<tr>
<td></td>
<td>card $</td>
<td>M</td>
<td>$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>elems ${l_1, \ldots, l_{</td>
<td>M</td>
</tr>
</tbody>
</table>
In VDM-SL notation:

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

In SMT-LIB notation:

$$(\forall l \ (\text{List} \ \text{Int}) \ (\ i \ \text{Int}))$$

$$(\implies$$

$$(\ \implies$$

$$(\ and \ (\geq \ i \ 0) \ (\ \text{in} \ i \ (\text{inds} \ l)))$$

$$(\forall j \ \text{Int})$$

$$(\implies$$

$$(\implies$$

$$(\ \text{in} \ j \ (\text{setminus} \ (\text{inds} \ l) \ (\text{set} \ i)))$$

$$(\text{in} \ j \ (\text{inds} \ l))))))))$$
The Event-B File System Case Study (delete/inv8) involves:

- `parent ∈ objects \ {root} → objects`,
- `obj ∈ objects \ {root}`,  `des ⊆ objects`,
- `des = (tcl(parent)) ∼ [{obj}]`,  `objs = des ∪ {obj}`

From above, it follows that:

- `objs ⊆ parent ∈ (objects \ objs) \ {root} → objects \ objs`.

**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))
))
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
Syntax + Semantics of theories is formally defined
⇒ In collaboration with Cesare Tinelli
⇒ To be discussed at SMT workshop 2009

Pre-processor is under development
⇒ Converter SMT-LIB 2 → SMT-LIB 1

Decidability is being investigated

WANTED: benchmarks
⇒ Necessary to get theories included in SMT-LIB standard
⇒ Event-B benchmarks would be awesome!
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 → good for users
- Implementation complexity → good for tool writers
- Decidability

⇒ We hope that we have found a good compromise
⇒ Feedback is welcome!
Thanks for your attention!