Satisfiability Modulo Theories: The SMTLib2 Input Format
Version 2015.1

Armin Biere (biere@jku.at)
Martina Seidl (martina.seidl@jku.at)
SMT-Lib (www.smtlib.org) is a community portal for people working on and with SMT Solving including

- ... a standard for describing background theories and logics
  6 background theories, > 20 logics

- ... a standard for input/output of SMT solvers

- ... a collection of 95492 benchmark formulas
totalling 59.2 GB in 383 families over 22 logics

- ... a collection of tools

- ... the basis of the annual competition
aim of an SMT solver: check satisfiability of formula $\phi$
- not over all (first-order) interpretations
- but with respect to some background theory

artifacts of an SMT solving system compliant to SMTLib v2:
- based on many-sorted first-order logic with equality
- background theory: taken from catalogue of theories
  - basic theories
  - combined theories
- interface: command language
- input formula
The SMT-Lib Command Language

- communication with the SMT solver
  - textual input channel
  - two textual output channels
    - regular output
    - diagnostic output

- primary design goal: interaction between programs

- types of commands
  - defining sorts and functions
  - managing assertions
  - checking satisfiability
  - setting options
  - getting information
  - exit

- responses: unsupported, success, error (string)
Theories and Logics

A theory
- defines a vocabulary for sorts and functions (signature).
- associates each sort with literals.
- may be infinite.
- has often an informal specification (in natural language).

A logic
- consists of at least one theory.
- restricts the kind of expressions to be used.
- has often an informal specification (in natural language).

SMTLib provides various theories and logics.
### Some Logics without Quantifiers

<table>
<thead>
<tr>
<th>Logic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QF_UF</td>
<td>formulas over uninterpreted functions</td>
</tr>
<tr>
<td>QF_LIA</td>
<td>formulas over linear integer arithmetic</td>
</tr>
<tr>
<td>QF_NIA</td>
<td>formulas over integer arithmetic</td>
</tr>
<tr>
<td>QF_BV</td>
<td>formulas over fixed-size bitvectors</td>
</tr>
<tr>
<td>QF_ABV</td>
<td>formulas over bitvectors and bitvector arrays</td>
</tr>
<tr>
<td>QF_AUFVB</td>
<td>formulas over bitvectors and bitvector arrays with unint. func.</td>
</tr>
<tr>
<td>QF_AUFLIA</td>
<td>linear formulas over integer arrays with uninterpreted functions</td>
</tr>
</tbody>
</table>
Terms, Functions, and Predicates

- Structure of terms and functions:
  - \langle constant \rangle
  - \langle identifier \rangle
  - \text{as (} \langle identifier \rangle \langle sort \rangle \text{)}
  - \langle identifier \rangle \langle term \rangle +
  - \text{as (} \langle identifier \rangle \langle sort \rangle \text{)} \langle term \rangle +
  - quantifier terms with \textit{forall}, \textit{exists}
  - attributed terms !
  - bound terms with \textit{let}

- \textbf{example} \texttt{(or (> p (+ q 2)) (< p (- q 2)))}

- terms are always typed
- no syntactic difference between functions and predicates
Declaring Functions (and Constants)

- **declare-fun** $(\sigma_1 \ldots \sigma_n) \; \sigma$:
  - declaration of new function with $n$ parameters of sorts $\sigma_1 \ldots \sigma_n$
  - return value of sort $\sigma$
- **constants are 0-ary functions**

**Example**
- `(declare-fun x () Bool)`
- `(declare-fun f (Int Int) Bool)`
- `(declare-fun ff ( (Int Int Bool) ) Int)`
Satisfiability Commands

- **(assert ⟨term⟩)**
  - term is of sort Bool
  - solver shall assume that term is true

- **(check-sat)**
  - check consistency of conjunction of assertions
  - response: sat, unsat, unknown

- get a solution with **(get-model)**

---

Example

```
(set-option :model true)
(declare-fun x () Int)
(assert (>= (* 3 x) (+ x x)))
(check-sat)
(get-model)
```
Example: Boolean Expressions

- Boolean expressions are defined in the *Core Theory*
- **sort:** Bool
- **constants:** true, false (*both of sort* Bool)
- **functions:**
  - not
  - or, xor, and, =>
  - =, distinct (*equality, inequality*)
  - **ite** (*if-then-else*)

**Example**

```
(set-logic QF_UF)
(declare-fun x () Bool)
(declare-fun y () Bool)
(assert (and (or x (not y)) (or (not x) y)))
(check-sat)
(exit)
```
Example: Real Expressions

- Real expressions are defined in the *Real Theory*
- sort: Real
- constants: numerals, decimals (all of sort `Real`)
- functions with signature:
  - (- (Real Real) ; negation
  - (- (Real Real) Real ) ; subtraction
  - (+ (Real Real) Real )
  - (* (Real Real) Real )
  - (/ (Real Real) Real )
  - (<= (Real Real) Bool)
  - (< (Real Real) Bool)
  - (>= (Real Real) Bool)
  - (>) (Real Real) Bool)

Example

```
(set-logic QF_LRA)
(declare-fun x () Real)
(declare-fun y () Real)
(assert (and (> ( * 2 x) (+ y 3.2)) (= x y)))
(check-sat)
```
Example: Array Expressions

The theory of Arrays defines functions to read and write elements of arrays.

- **sort**: `Array <sort of index> <sort of elements>`

- **functions**
  
  - `(select (array index) value)` where
    - `array` is of sort `(Array <sort of index> <sort of elements>)`
    - `index` is of sort `<sort of index>`
    - `value` is of sort `<sort of elements>`

  - `(store (array1 index value) array2)` where
    - `array1, array2` are of sort `(Array <sort of index> <sort of elements>)`
    - `index` is of sort `<sort of index>`
    - `value` is of sort `<sort of elements>`

**Example**

```lisp
(declare-fun a () (Array Int Bool))
(declare-fun b () (Array Int Bool))
(assert (= (select a 1) true))
(assert (= (store b 1 false) a))
(check-sat) ; result is unsat
```
Example: Fixed-Sized Bitvectors Expressions

- **sort:** \((\_ \text{BitVec} \ n)\) where \(n\) is the size of the bitvector

- **functions:**
  - \((\text{op1} (\_ \text{BitVec} \ m) (\_ \text{BitVec} \ m))\)
    - with \(\text{op1} \in \{\text{bvnot}, \text{bvneg}\}\)
  - \((\text{op2} (\_ \text{BitVec} \ m) (\_ \text{BitVec} \ m) (\_ \text{BitVec} \ m))\)
    - with \(\text{op2} \in \{\text{bvand}, \text{bvor}, \text{bvadd}, \text{bvmul}, \text{bvudiv}, \text{bvurem}, \text{bvshl}, \text{bvlshr}\}\)
  - \((\text{bvult} (\_ \text{BitVec} \ m) (\_ \text{BitVec} \ m) \text{Bool})\)
    - binary comparison
  - \((\_ \text{extract i j}) (\_ \text{BitVec} \ m) (\_ \text{BitVec} \ n))\)
    - extract contiguous subvector from index \(i\) to index \(j\)
  - \((\text{concat} (\_ \text{BitVec} \ i) (\_ \text{BitVec} \ j) (\_ \text{BitVec} \ m))\)
    - combines two bitvectors
SMTLib2 offers many more language concepts, for example:

- Makros
- User-defined sorts
- Many Options
- Scopes
- ...

More infos: