

# SMT-LIB in KeY

5<sup>th</sup> KeY Symposium 2006

by Achim Kuwertz

06.06.06

# SMT-LIB

## The Satisfiability Modulo Theories Library

- SMT-LIB initiative <http://combination.cs.uiowa.edu/smtlib/>
- Goal: Produce an extensive on-line library of benchmarks for satisfiability modulo theories

Benchmark = input problem, i.e. logical formula

SMT = Problem of interest is whether a given formula is satisfiable in a given theory

- Motivation: A library will facilitate the evaluation and comparison of SMT checking systems
- Definition of a common standard for the specification of benchmarks and theories

# SMT-LIB

## The Satisfiability Modulo Theories Library

- What does this have to do with KeY?
- Specialized back-end decision procedures are used to check the validity of pure first order logic problems in a *sequent*
- SMT-LIB provides a common standardized input language
- Enables usage of different decision procedures in an easy-to-implement and interchangeable way

# SMT-LIB FORMAT V1.1

- Underlying: First order logic with equality
- Supports different background theories (e.g. integer numbers, real numbers, array, etc.)
- Logic: Combination of a theory and a fragment of  $\text{FOL}^=$  (restriction)
- Our logic of interest: QF\_AUFLIA

# SMT-LIB FORMAT

## Logic QF\_AUFLIA

- Integer numbers and arrays as theory
- Interpreted symbols:
  - integer number
  - $\sim$ ,  $+$ ,  $-$ ,  $*$  (but only linear atoms)
  - *not*, *implies*, *and*, *or*, *iff*, *xor*, *true* and *false*
  - *select*, *store* on arrays
  - *int* and *array* as sort symbols
- Supports free sort, predicate and function symbols
- Convenience symbols (e.g. a *distinct* predicate)

# SMT-LIB FORMAT

## Benchmark

- Benchmark = closed SMT-LIB formula with attached additional information

- Concrete syntax for benchmarks:

```
- <benchmark> ::=  
    (benchmark <bench_name> <bench_attribute>+ )
```

```
- <bench_attribute> ::=  
    :status ( sat | unsat | unknown )  
  | :logic <logic_name>  
  | :extrasorts ( <sort_symb>+ )  
  | :extrafuns ( <fun_symb_decl>+ )  
  | :extrapreds ( <pred_symb_decl>+ )  
  | :formula <an_formula>  
  | :assumption <an_formula>
```

# SMT-LIB

## Example Benchmark

```
(benchmark example1

  :status unknown
  :logic QF_AUFLIA
  :extrafuns (paycard_obj Int)
  :extrafuns (balance_int Int Int)
  :extrapreds (created Int)
  :extrapreds (flag)
  :formula
    (implies
      (and (created paycard_obj) true)
      (or
        (< (balance_int paycard_obj) 0)
        (if_then_else flag (not (= 66 23)) (< 42 17)))
      )
    )
)
```

## SMT-LIB

# Implementation: Structure

- Main class *Benchmark*
- Two abstract superclasses *Term* and *Formula*, containing common methods
- Each of them has several subclasses which implement specific predicates and functions
- Additional class *Signature*

# Implementation: Formula

- Different Formulas:
  - *TruthValueFormula* (*true, false*)
  - *FormulaVariable*
  - *PredicateFormula* (*=, <=, <, >=, >, distinct*)
  - *ConnectiveFormula* (*not, and, or, implies, xor, iff, ifthenelse*)
  - *UninterpretedPredFormula* (free predicates)

# SMT-LIB

## Implementation: Term

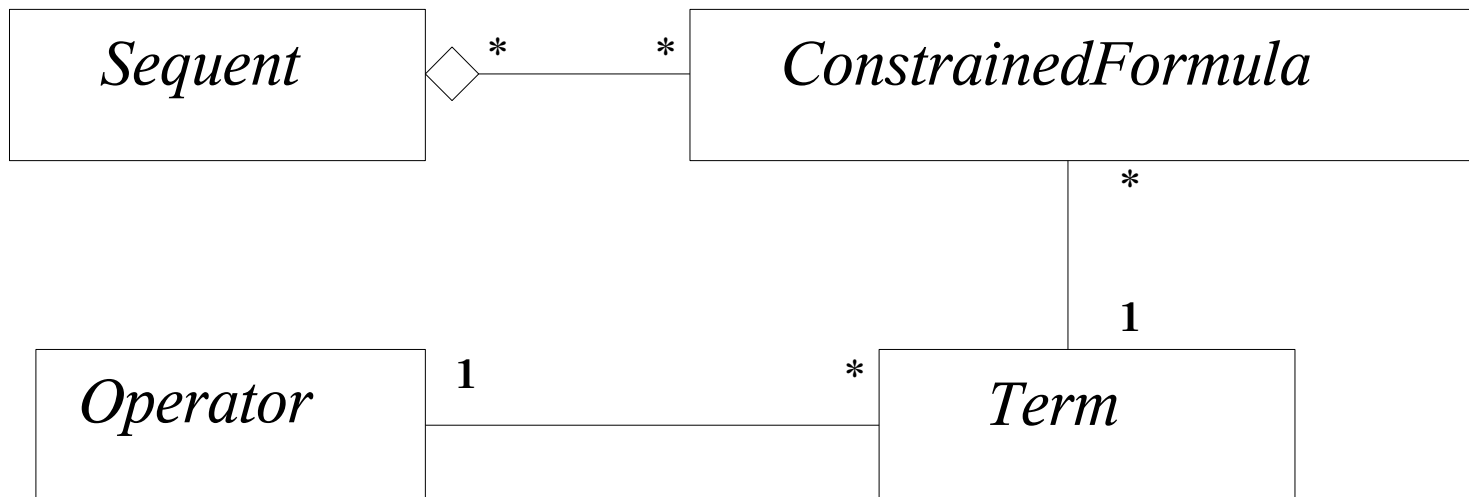
- Different Terms:
  - *NumberConstantTerm* (non negative integers)
  - *TermVariable*
  - *InterpretedFuncTerm* (  $\sim$ , + , - , \* , *select*, *store* )
  - *UninterpretedFuncTerm* (free functions)

## Implementation: Benchmark

- Result of translation process from a KeY sequent
- Input for back-end decision procedures
- Takes a *Formula* instance and assembles an SMT benchmark:
  - Generates **:extrasorts**, **:extrapreds** and **:extrafuns** attributes out of the free sorts, predicates and functions in the *Formula* object
  - Assembles all attributes to SMT compliant benchmark and prints it

# KeY - SMT Translation

- Input: *Sequent*
- Output: *Benchmark*
- What is a *Sequent* ?



# KeY - SMT Translation

- Interesting case: *Term* and *Operator* translation
- QF\_AUFLIA: No quantifiers, no free variables  
=> Terms containing *Quantifiers* and *Metavariables*  
are currently ignored for translation
- *Modalities*, *UpdateOperators* and *ProgramMethods*  
should be handled by KeY => ignore them
- All remaining operators are translated

# KeY - SMT Translation

- *Term* translation
  - Done by a visitor
  - Walks through the syntax tree of the term in post order
  - Operators are translated into SMT-LIB equivalents
  - Translation results are assembled on the fly
- *Operator* translation
  - Simple interpreted operators are translated directly
  - Complex operators are decomposed and translated
  - For every uninterpreted, translatable operator a unique predicate or function is generated

# KeY - SMT Translation

- *Sequent* translation
  - Logically connect *Term* translations
  - Negate resulting formula
  - Embed in *Benchmark*
- Check benchmark for satisfiability with a back-end decision procedure supporting SMT-LIB format
- **[Demo]**

# SMT-LIB Competition 05

## Tools and results for QF\_AUFLIA

<b>Solver</b>	<b>Score</b>	<b>Time</b>	<b>Unsat</b>	<b>Sat</b>	<b>Unknown</b>	<b>Wrong</b>
Yices	49	46.8	35	14	3	0
CVC	34	243	34	0	18	0
CVC Lite	34	769.3	28	6	18	0
SVC	30	84.3	30	0	22	0
HTP	21	132	25	1	26	1
Sammy	-38	344.6	10	3	39	7

=> Evaluating usage of Yices, CVC Lite and SVC

Thanks for your attention!

Any questions?