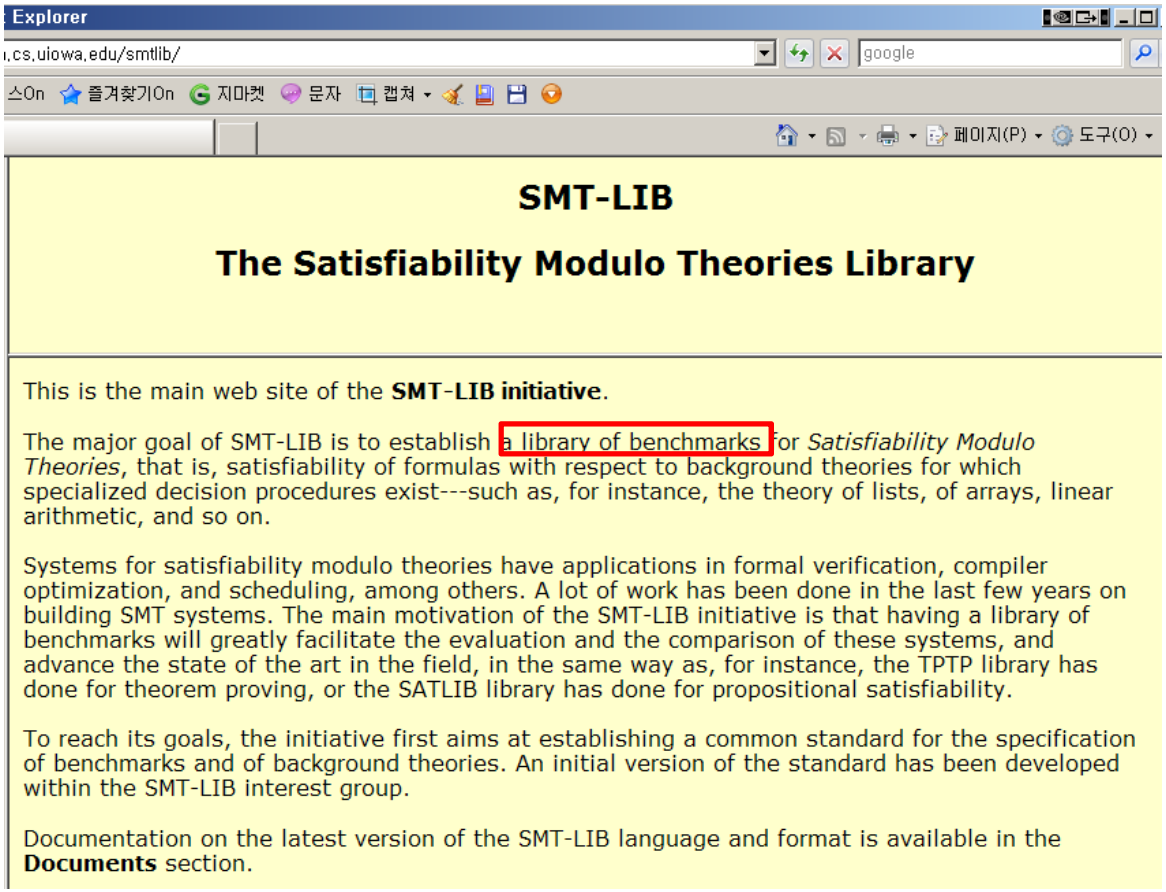
The Satisfiability Modulo Theories Library (SMT-LIB)

Moonzoo Kim

Provable Software Laboratory

CS Dept. KAIST



Supported Theories

- **Arrays**
  - Functional arrays without extensionality.
- **ArraysEx**
  - Functional arrays with extensionality.
- **Fixed_Size_BitVectors[32]**
  - Bit vectors with size up to 32 bits.
- **Fixed_Size_BitVectors**
  - Bit vectors with arbitrary size.
- **BitVector_ArraysEx**
  - Bit vectors with arbitrary size, plus arrays indexed by and containing bit vectors.
- **Empty**
  - Empty theory over the empty signature.
- **Ints**
  - Integer numbers.
- **Int_Arrays**
  - Arrays of integer index and value.
- **Int_ArraysEx**
  - Arrays of integer index and value with extensionality axiom.
- **Int_Int_Real_Array_ArraysEx**
  - Arrays of arrays of integer index and real value, with extensionality axiom.
- **Reals**
  - Real numbers.
- **Reals_Ints**
  - Real and integer numbers.

Moonzoo Kim
Provable SW Lab
KAIST

Theory of Arrays

```
(theory Arrays
 :written_by {Silvio Ranise and Cesare Tinelli}
 :date {08/04/05}
 :sorts (Index Element Array)
 :funs ((select Array Index Element)
        (store Array Index Element Array))

 :definition
 "This is a theory of functional arrays without extensionality.
  It is formally and completely defined by the axioms below. "

 :axioms  (
  (forall (?a Array) (?i Index) (?e Element)
    (= (select (store ?a ?i ?e) ?i) ?e))

  (forall (?a Array) (?i Index) (?j Index) (?e Element)
    (or (= ?i ?j)
      (= (select (store ?a ?i ?e) ?j)  (select ?a ?j))))
)
```

Predefined data types

Predefined functions

Bounded variables

Prefix operator

```
 :notes
 "It is not difficult to prove that the two
      axioms above are logically equivalent
      to the following \"McCarthy axiom\":

 (forall  (?a Array) (?i Index) (?j Index)
          (?e Element)
        (= (select (store ?a ?i ?e) ?j)
           (ite (= ?i ?j) ?e (select ?a ?j))))

  Such an axiom appeared in the following
      paper:
  Correctness of a Compiler for Arithmetic
      Expressions,
  by John McCarthy and James Painter,
  available at http://www-formal.stanford.
      edu/jmc/mcpain.html.
 "
)
```

If-then-else term construct

Moonzoo Kim
Provable SW Lab
KAIST

Theory of Arrays w/ Extensionability

```
(theory ArraysEx
 :written_by {Silvio Ranise and Cesare Tinelli}
 :date {08/04/05}
 :updated {28/10/05}
 :history {
  Bug fix in the third axiom, pointed out by Robert
  Nieuwenhuis:
  The scope of 'forall (?i Index)' was the whole implication
  instead of just the premise of the implication.
  }
 :sorts (Index Element Array)
 :funs ((select Array Index Element)
        (store Array Index Element Array))
 :definition
 "This is a theory of functional arrays with extensionality.
  It is formally and completely defined by the axioms below.
 "
```

```
 :axioms
 (
  (forall (?a Array) (?i Index) (?e Element)
    (= (select (store ?a ?i ?e) ?i) ?e))
  (forall (?a Array) (?i Index) (?j Index) (?e Element)
    (or (= ?i ?j)
        (= (select (store ?a ?i ?e) ?j)
           (select ?a ?j))))

  (forall (?a Array) (?b Array)
     (implies (forall (?i Index) (= (select ?a ?i) (select ?b ?i)))
              (= ?a ?b)))
 )
 :notes "This theory extends the theory Arrays with
  an axiom stating that any two arrays with the same
  elements are in fact the same array. " )
```


Moonzoo Kim
Provable SW Lab
KAIST

Theory of Integer

```
(theory Ints
:sorts (Int)
:notes
 "The (unsupported) annotations of the function/pre
      dicate symbols have
  the following meaning:
   attribute | possible value | meaning
  ----------------------------------------------------
  :assoc          //         the symbol is associative
  :comm           //         the symbol is commutative
  :unit       a constant
  :trans          //         the symbol is transitive
  :refl           //         the symbol is reflexive
  :irref          //         the symbol is irreflexive
  :antisym        //         the symbol is antisymmetric
"
 :funs ((0 Int)
       (1 Int)
       (~ Int Int)     ; unary minus
       (- Int Int Int) ; binary minus
       (+ Int Int Int :assoc :comm :unit {0})
       (* Int Int Int :assoc :comm :unit {1})   )
 :preds ((<= Int Int :refl :trans :antisym)
         (< Int Int :trans :irref)
         (>= Int Int :refl :trans :antisym)
         (> Int Int :trans :irref)
        )
 :definition
 "This is the first-order theory of the integers, that is
, the set of all the first-order sentences over the given
signature that are true in the structure of the integer
numbers interpreting the signature's symbols in the
obvious way (with ~ denoting the negation and - the
subtraction functions). "

 :notes
 "Note that this theory is not (recursively) axiomatizable
in a first-order logic such as SMT-LIB's underlying logic.
That is why the theory is defined semantically. "
 )
```

Moonzoo Kim
Provable SW Lab
KAIST

Example of QF_LIA Benchmark

```
(benchmark example
:status sat
:logic QF_LIA
:extrafuns ((x1 Int)
(x2 Int) (x3 Int) (x4 Int) (x5 Int))
;human readable form
; x1-1 => x2 /\
; x1-3 <= x2 /\
; x1 = 2 x3+x5 /\
; x3 = x5   /\
; x2 = 6 x4
:formula (and
        (>= (- x1 x2) 1)
        (<= (- x1 x2) 3)
        (= x1 (+ (* 2 x3) x5))
        (= x3 x5)
        (= x2 (* 6 x4))))
```

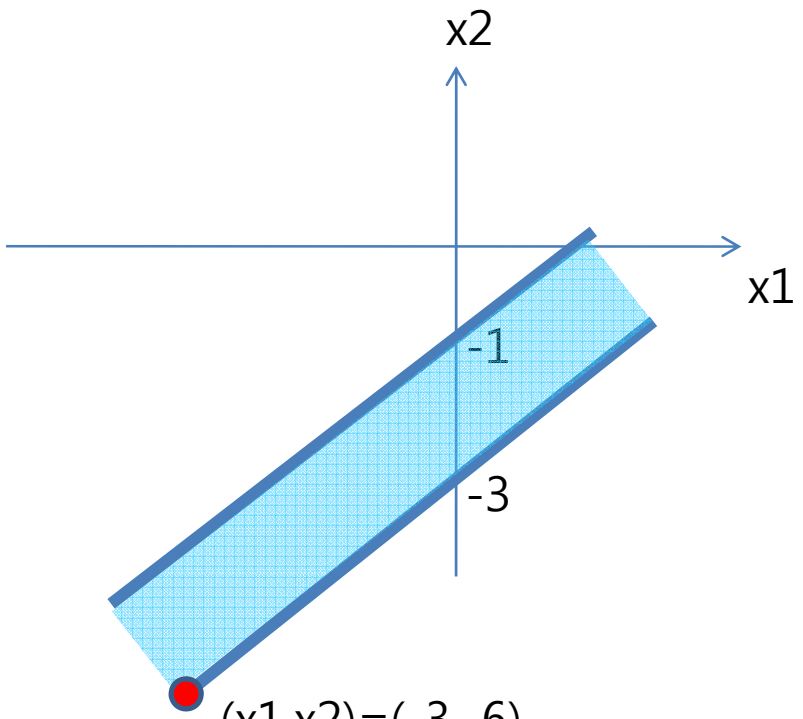
Expected output (optional)

Theory

User defined variables

Comments

Target formula



Moonzoo Kim
Provable SW Lab
KAIST

Example of QF_UF Benchmark

```
(benchmark example2-1
:logic QF_UF
:extrasorts (A B C D)
:extrafuns  ((x A)(y B)(w A)(z C)(u D))
:extrafuns  ((f A A B)
(g A B B) (h1 B A B) (h2 B B B))

;human readable form
;   g(x,y) = h1(y,x) /\
;   f(x,x) = h2(y,y) /\
;   f(x,x) != f(x,w)
:assumption((= (g x y) (h1 y x)))
:assumption((= (f x x) (h2 y y))
:assumption((not (= (f x x) (f x w))))
:formula true
)
```

User defined data types

User defined functions

A model for the formula

```
x-> v0
y->v1
w->v4
g->{ (v1,v0)->v2,
        else-> v2}
f->{(v0,v0)->v3,
      (v0,v4)->v5,
      else->v5}
h2->{(v1,v1)->v3,
        else -> v3}
```


KAIST

Another Example of QF_UF Benchmark

Nov 10th CS453 QUIZ #2

3. Prove that

F : a=b ^ b=c -> g(f(a), b) = g(f (c), a)

is T_{EUF}–**satisfiable**

4. Prove

F : a=b ^ b=c -> g(f(a), b) = g(f (c), a)

is T_{EUF}–**valid** through proof tree

```
(benchmark Quiz2
:logic QF_UF
:extrasorts (A B C)
:extrafuns  ((a A) (b A) (c A))
:extrafuns  ((f A B) (g B A C))
;human readable form
; a=b/\ b=c ->  g(f(a),b) = g(f(c),a)
:formula (implies (and (= a b) (= b c))
                  (= (g ( f a) b) (g (f c) a)))))
```

Then, how to check T_{EUF}–validity of F by using a SMT solver???

A model for the formula

a->v0

b->v1

c->v2

f->{v0->v3,v2->v5,else->v5}

g->{(v3,v1)->v4,(v5,v0)->v6,else->v6}

Example of QF_AUFLIA

```
(benchmark sort
:logic QF_AUFLIA
:extrafuns ((data_7 Array)) ; initial data[] declaration
:extrafuns ((tmp_0 Int))
:extrafuns ((i_9 Int))
 :assumption (= i_9 0) ; i=0;
:extrafuns ((j_1 Int))
 :assumption (= j_1 1) ; j=1;
:extrafuns ((tmp_1 Int))
 :assumption (= tmp_1 (select data_7 0))   ; tmp = data[0]
:extrafuns ((data_8 Array))
 :assumption (= data_8 (store data_7 0 (select data_7 1))); data[0]=data[1];
:extrafuns ((data_9 Array))
 :assumption (= data_9 (store data_8 1 tmp_1)) ; data[1] =  tmp;
:extrafuns ((data_10 Array))
 ; if (data[0] > data[1])  { tmp=data[0]; data[0]=data[1]; data[1]=tmp}
 :assumption (= data_10 (if_then_else (> (select data_7 0) (select data_7 1)
      ) data_9 data_7))
…
:formula (not
(and (<= (select data_70 0) (select data_70 1))
   (<= (select data_70 1) (select data_70 2))
    …)
```

```
#define N 7
int main(){
  int data[N], i, j, tmp;
  for (i=0; i<N-1; i++)
    for (j=i+1; j<N; j++)
       if(data[i]>data[j]){
          tmp = data[i];
          data[i] = data[j];
          data[j] = tmp;
       }
  assert(data[0]<=data[1]&&…);
}
```


Moonzoo Kim
Provable SW Lab
KAIST

Theory of Fixed_Size_BitVectors[32]

:sorts_description
"All sort symbols of the form BitVec[i],
where i is a numeral between 1 and 32, inclusive.“
:funs_description
"All function symbols with arity of the form
(concat BitVec[i] BitVec[j] BitVec[m]) where
- i,j,m are numerals
- i,j > 0
- i + j = m <= 32 “
:funs_description
"All function symbols with arity of the form
(extract[i:j] BitVec[m] BitVec[n]) where
- i,j,m,n are numerals
- 32 >= m > i >= j >= 0,
- n = i-j+1. “
:funs_description
"All function symbols with arity of the form
(op1 BitVec[m] BitVec[m]) or
(op2 BitVec[m] BitVec[m] BitVec[m]) where
- op1 is from {bvnot, bvneg}
- op2 is from {bvand,bvor,bvxor,bvsub,bvadd,bvmul}
- m is a numeral
- 0 < m <= 32 “
:preds_description
"All predicate symbols with arity of the form
(pred BitVec[m] BitVec[m]) where
- pred is from {bvlt, bvleq, bvgeq, bvgt}
- m is a numeral
- 0 < m <= 32 "

- Variables
If v is a variable of sort BitVec[m] with 0 < m <= 32, then
[[v]] is some element of [{0,...,m-1} -> {0,1}], the set of
total functions from {0,...,m-1} to {0,1}.
- Constant symbols bv0 and bv1 of sort BitVec[32]
[[bv0]] := \lambda x : [0...32). 0
[[bv1]] := \lambda x : [0...32). if x = 0 then 1 else 0
- Function symbols for concatenation
[[(concat s t)]] := \lambda x : [0...n+m).
if (x<m) then [[t]](x) else [[s]](x-m) where
s and t are terms of sort BitVec[n] and BitVec[m],
respectively, 0 < n <= 32, 0 < m <= 32, and n+m <= 32.
- Function symbols for extraction
[[(extract[i:j] s)]] := \lambda x : [0...i-j+1). [[s]](j+x)
where s is of sort BitVec[l], 0 <= j <= i < l <= 32.
- Function symbols for arithmetic operations
To define the semantics of the bitvector operators bvadd
, bvsub, bvneg, and bvmul, it is helpful to use these
ancillary functions:
o bv2nat which takes a bitvector b: [0...m) --> {0,1}
with 0 < m <= 32, and returns an integer in the range
[0...2^m), and is defined as follows:
bv2nat(b) := b(m-1)*2^{m-1} + b(m-2)*2^{m-2} + ... + b(0)*2^0
o nat2bv[m], with 0 < m <= 32, which takes a non-negative
integer n and returns the (unique) bitvector b: [0,...,m) -> {0,1}
such that b(m-1)*2^{m-1} + ... + b(0)*2^0 = n MOD 2^m
where MOD is usual modulo operation.
[[(bvadd s t)]] := nat2bv[m](bv2nat(s) + bv2nat(t))

SMTLIB Benchmark Syntax

- Reserved keywords
  - =, and, benchmark, distinct, exists, false, flet, forall, if then el se, iff, implies,ite, let, logic, not, or, sat, theory, true, unknow n, unsat, xor

Formulas

$\langle prop\_atom \rangle$ ::= `true` | `false` | $\langle fvar \rangle$ | $\langle identifier \rangle$

$\langle an\_atom \rangle$ ::= $\langle prop\_atom \rangle$ | `(` $\langle prop\_atom \rangle$ $\langle annotation \rangle^+$ `)`
| `(` $\langle pred\_symb \rangle$ $\langle an\_term \rangle^+$ $\langle annotation \rangle^*$ `)`

$\langle connective \rangle$ ::= `not` | `implies` | `if_then_else`
| `and` | `or` | `xor` | `iff`

$\langle quant\_symb \rangle$ ::= `exists` | `forall`

$\langle quant\_var \rangle$ ::= `(` $\langle var \rangle$ $\langle sort\_symb \rangle$ `)`

$\langle an\_formula \rangle$ ::= $\langle an\_atom \rangle$
| `(` $\langle connective \rangle$ $\langle an\_formula \rangle^+$ $\langle annotation \rangle^*$ `)`
| `(` $\langle quant\_symb \rangle$ $\langle quant\_var \rangle^+$ $\langle an\_formula \rangle$ $\langle annotation \rangle^*$ `)`
| `(` `let` `(` $\langle var \rangle$ $\langle an\_term \rangle$ `)` $\langle an\_formula \rangle$ $\langle annotation \rangle^*$ `)`
| `(` `flet` `(` $\langle fvar \rangle$ $\langle an\_formula \rangle$ `)` $\langle an\_formula \rangle$ $\langle annotation \rangle^*$ `)`

KAIST