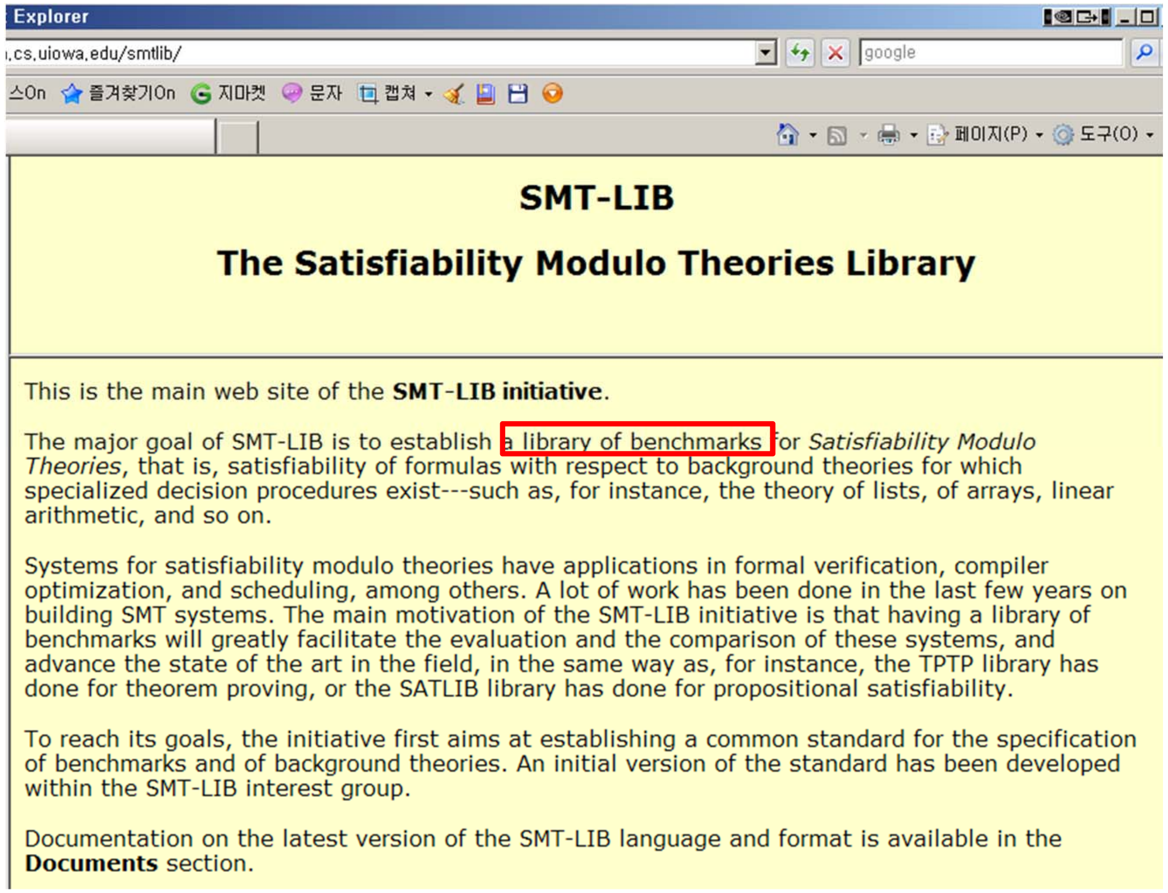
The Satisfiability Modulo Theories Library (SMT-LIB)

Moonzoo Kim

Provable Software Laboratory

CS Dept. KAIST



SMT-LIB

The Satisfiability Modulo Theories Library

This is the main web site of the **SMT-LIB initiative**.

The major goal of SMT-LIB is to establish a library of benchmarks for *Satisfiability Modulo Theories*, that is, satisfiability of formulas with respect to background theories for which specialized decision procedures exist---such as, for instance, the theory of lists, of arrays, linear arithmetic, and so on.

Systems for satisfiability modulo theories have applications in formal verification, compiler optimization, and scheduling, among others. A lot of work has been done in the last few years on building SMT systems. The main motivation of the SMT-LIB initiative is that having a library of benchmarks will greatly facilitate the evaluation and the comparison of these systems, and advance the state of the art in the field, in the same way as, for instance, the TPTP library has done for theorem proving, or the SATLIB library has done for propositional satisfiability.

To reach its goals, the initiative first aims at establishing a common standard for the specification of benchmarks and of background theories. An initial version of the standard has been developed within the SMT-LIB interest group.

Documentation on the latest version of the SMT-LIB language and format is available in the **Documents** section.

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.

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 f unctions

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

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. " )
```

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. "
 )
```

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))))
```

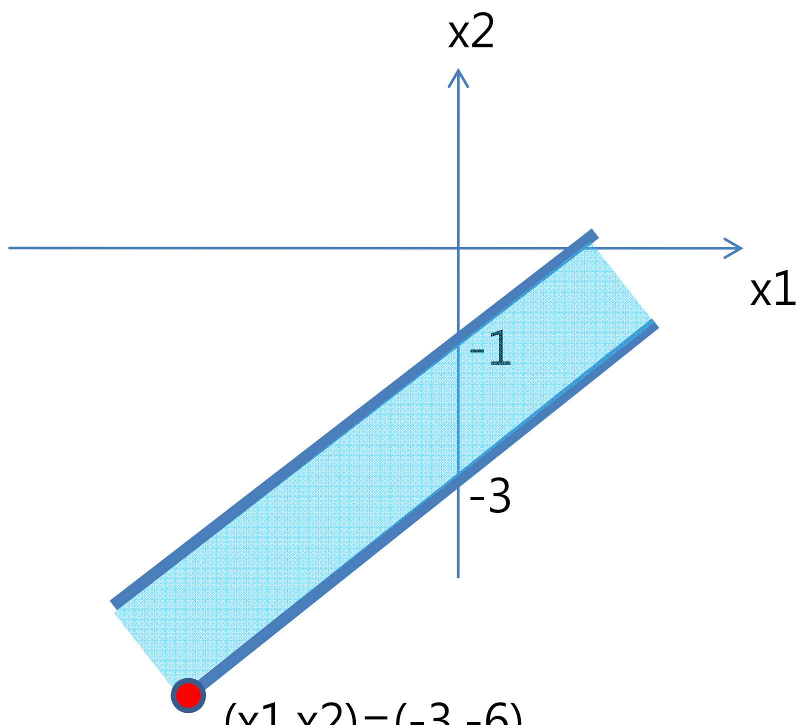
Expected output (optional)

Theory

User defined variables

Comments

Target formula



(x1,x2)=(-3,-6)
(x3,x4,x5)=(-1,-1,-1)

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

<u>A model for the formula</u>

```
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}
```

Another Example of QF_UF Benchmark

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)))))
```

<u>A model for the formula</u>

a->v0

b->v1

c->v2

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

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

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

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]&&…);
}
```

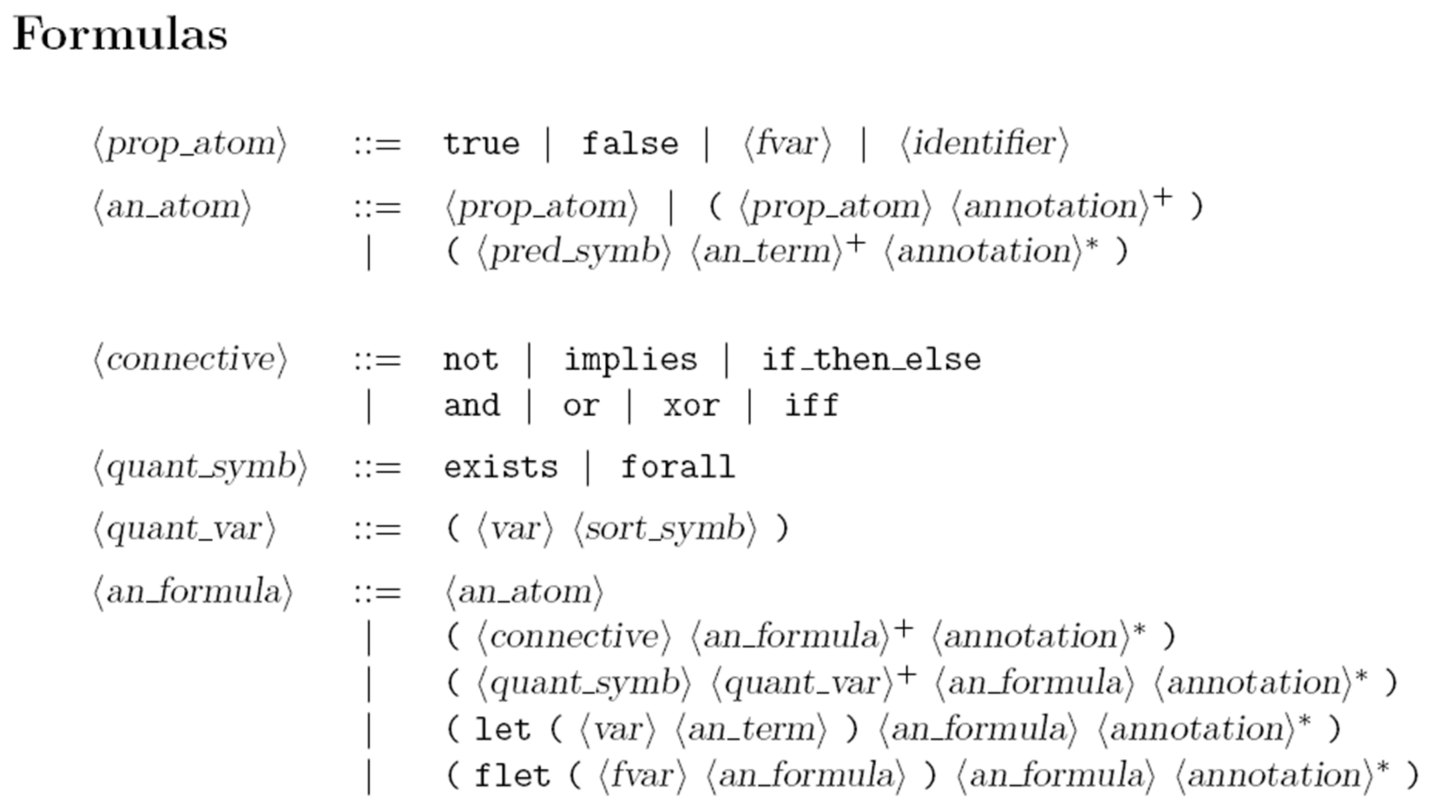
Theory of Fixed_Size_BitVectors[32]

: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

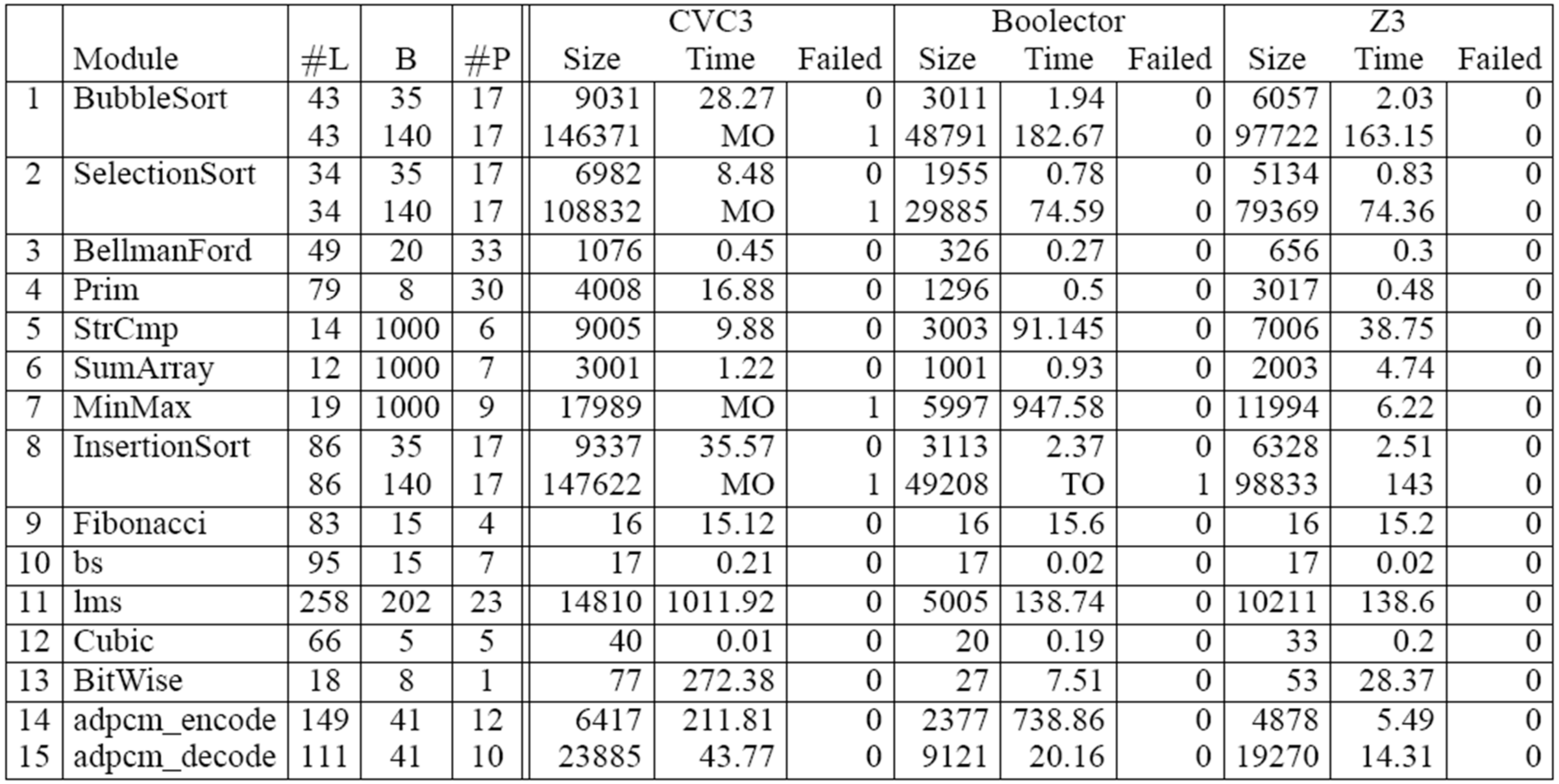


Formulas

```
⟨prop_atom⟩    ::=  true | false | ⟨fvar⟩ | ⟨identifier⟩
⟨an_atom⟩      ::=  ⟨prop_atom⟩ | ( ⟨prop_atom⟩ ⟨annotation⟩+ )
                |   ( ⟨pred_symb⟩ ⟨an_term⟩+ ⟨annotation⟩* )

⟨connective⟩   ::=  not | implies | if_then_else
                |   and | or | xor | iff
⟨quant_symb⟩   ::=  exists | forall
⟨quant_var⟩    ::=  ( ⟨var⟩ ⟨sort_symb⟩ )
⟨an_formula⟩   ::=  ⟨an_atom⟩
                |   ( ⟨connective⟩ ⟨an_formula⟩+ ⟨annotation⟩* )
                |   ( ⟨quant_symb⟩ ⟨quant_var⟩+ ⟨an_formula⟩ ⟨annotation⟩* )
                |   ( let ( ⟨var⟩ ⟨an_term⟩ ) ⟨an_formula⟩ ⟨annotation⟩* )
                |   ( flet ( ⟨fvar⟩ ⟨an_formula⟩ ) ⟨an_formula⟩ ⟨annotation⟩* )
```

Performance Comparison of SMT Solvers

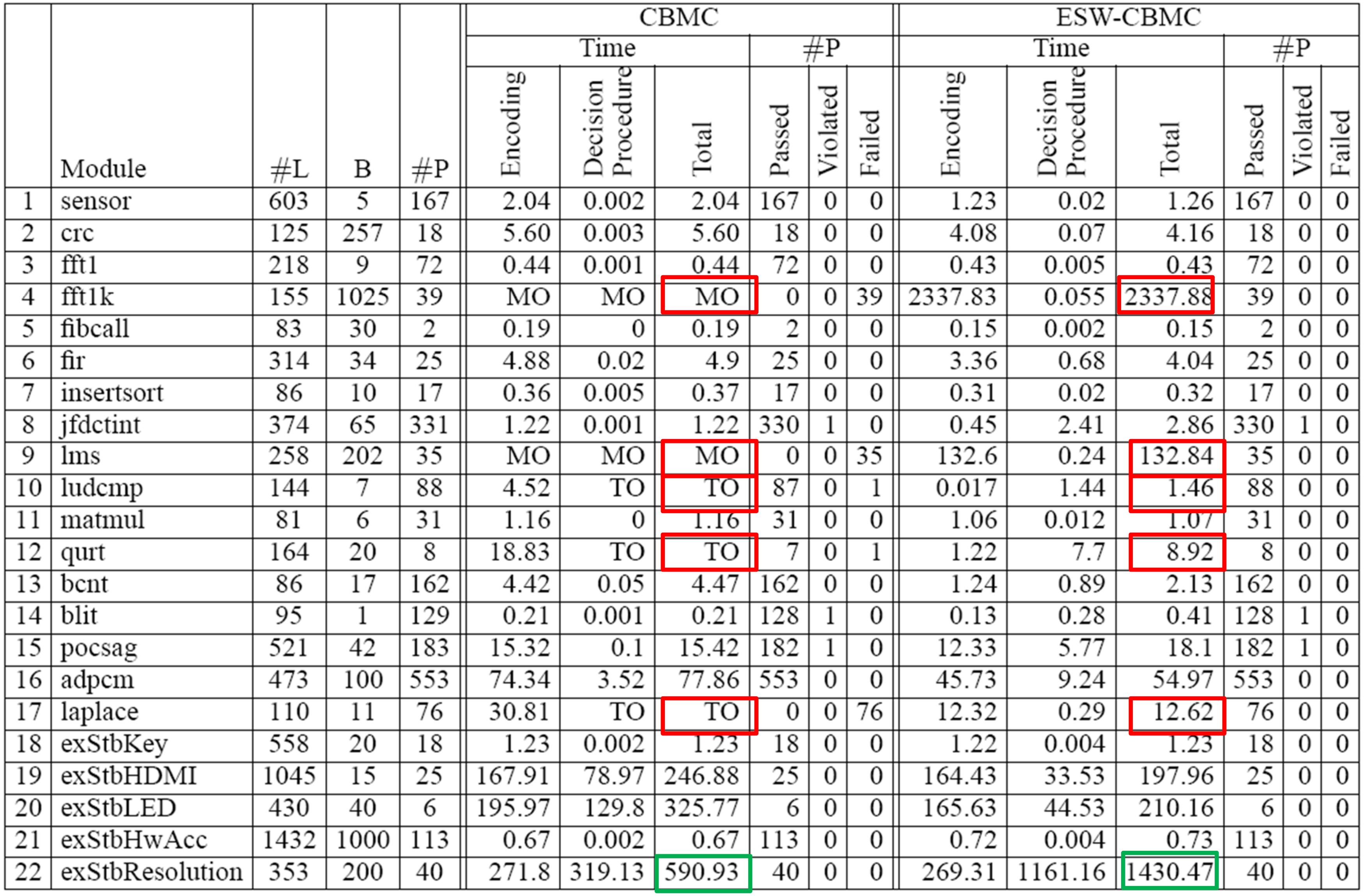


					CVC3			Boolector			Z3		
	Module	#L	B	#P	Size	Time	Failed	Size	Time	Failed	Size	Time	Failed
1	BubbleSort	43	35	17	9031	28.27	0	3011	1.94	0	6057	2.03	0
		43	140	17	146371	MO	1	48791	182.67	0	97722	163.15	0
2	SelectionSort	34	35	17	6982	8.48	0	1955	0.78	0	5134	0.83	0
		34	140	17	108832	MO	1	29885	74.59	0	79369	74.36	0
3	BellmanFord	49	20	33	1076	0.45	0	326	0.27	0	656	0.3	0
4	Prim	79	8	30	4008	16.88	0	1296	0.5	0	3017	0.48	0
5	StrCmp	14	1000	6	9005	9.88	0	3003	91.145	0	7006	38.75	0
6	SumArray	12	1000	7	3001	1.22	0	1001	0.93	0	2003	4.74	0
7	MinMax	19	1000	9	17989	MO	1	5997	947.58	0	11994	6.22	0
8	InsertionSort	86	35	17	9337	35.57	0	3113	2.37	0	6328	2.51	0
		86	140	17	147622	MO	1	49208	TO	1	98833	143	0
9	Fibonacci	83	15	4	16	15.12	0	16	15.6	0	16	15.2	0
10	bs	95	15	7	17	0.21	0	17	0.02	0	17	0.02	0
11	lms	258	202	23	14810	1011.92	0	5005	138.74	0	10211	138.6	0
12	Cubic	66	5	5	40	0.01	0	20	0.19	0	33	0.2	0
13	BitWise	18	8	1	77	272.38	0	27	7.51	0	53	28.37	0
14	adpcm_encode	149	41	12	6417	211.81	0	2377	738.86	0	4878	5.49	0
15	adpcm_decode	111	41	10	23885	43.77	0	9121	20.16	0	19270	14.31	0

Table 1. Results of the comparison between CVC3, Boolector and Z3. Time-outs are represented with TO in the Time column; Examples that exceed available memory are represented with MO in the Time column.

Quoted from "SMT-Based Bounded Model Checking for Embedded ANSI-C Software" by L.Cordeiro, et al ASE 2009

Performance Comparison between CBMC and SMT-CBMC



					CBMC						ESW-CBMC					
					Time			#P			Time			#P		
	Module	#L	B	#P	Encoding	Decision Procedure	Total	Passed	Violated	Failed	Encoding	Decision Procedure	Total	Passed	Violated	Failed
1	sensor	603	5	167	2.04	0.002	2.04	167	0	0	1.23	0.02	1.26	167	0	0
2	crc	125	257	18	5.60	0.003	5.60	18	0	0	4.08	0.07	4.16	18	0	0
3	fft1	218	9	72	0.44	0.001	0.44	72	0	0	0.43	0.005	0.43	72	0	0
4	fft1k	155	1025	39	MO	MO	MO	0	0	39	2337.83	0.055	2337.88	39	0	0
5	fibcall	83	30	2	0.19	0	0.19	2	0	0	0.15	0.002	0.15	2	0	0
6	fir	314	34	25	4.88	0.02	4.9	25	0	0	3.36	0.68	4.04	25	0	0
7	insertsort	86	10	17	0.36	0.005	0.37	17	0	0	0.31	0.02	0.32	17	0	0
8	jfdctint	374	65	331	1.22	0.001	1.22	330	1	0	0.45	2.41	2.86	330	1	0
9	lms	258	202	35	MO	MO	MO	0	0	35	132.6	0.24	132.84	35	0	0
10	ludcmp	144	7	88	4.52	TO	TO	87	0	1	0.017	1.44	1.46	88	0	0
11	matmul	81	6	31	1.16	0	1.16	31	0	0	1.06	0.012	1.07	31	0	0
12	qurt	164	20	8	18.83	TO	TO	7	0	1	1.22	7.7	8.92	8	0	0
13	bcnt	86	17	162	4.42	0.05	4.47	162	0	0	1.24	0.89	2.13	162	0	0
14	blit	95	1	129	0.21	0.001	0.21	128	1	0	0.13	0.28	0.41	128	1	0
15	pocsag	521	42	183	15.32	0.1	15.42	182	1	0	12.33	5.77	18.1	182	1	0
16	adpcm	473	100	553	74.34	3.52	77.86	553	0	0	45.73	9.24	54.97	553	0	0
17	laplace	110	11	76	30.81	TO	TO	0	0	76	12.32	0.29	12.62	76	0	0
18	exStbKey	558	20	18	1.23	0.002	1.23	18	0	0	1.22	0.004	1.23	18	0	0
19	exStbHDMI	1045	15	25	167.91	78.97	246.88	25	0	0	164.43	33.53	197.96	25	0	0
20	exStbLED	430	40	6	195.97	129.8	325.77	6	0	0	165.63	44.53	210.16	6	0	0
21	exStbHwAcc	1432	1000	113	0.67	0.002	0.67	113	0	0	0.72	0.004	0.73	113	0	0
22	exStbResolution	353	200	40	271.8	319.13	590.93	40	0	0	269.31	1161.16	1430.47	40	0	0