SMT Solvers

Cristian Cadar

SOFTWARE RELIABILITY GROUP

Department of Computing

Imperial College London

Imperial College London

Software Reliability Course
Autumn 2016

Why SMT?

- SAT solvers operate at the level of Boolean or propositional formulas
- Many application domains generate constraints at a higher level

- SMT supports rich ***theories*** in classical ***first-order logic with equality***

SMT Theories

- A theory consists of a signature S_T and axioms A_T
- S_T consists of 3 types of constants:
  - **Object constants** refer to objects in the universe of discourse, e.g, ***John, Mary***,… for universe of people
  - **Function constants** refer to functions
    - Each function has an associated arity, e.g., ***parent*** has arity 1
    - Object constants can be seen as function constants with arity 0
  - **Predicate constants** refer to relations between objects
    - Each predicate has an associated arity, e.g., ***likes*** has arity 2
- A_T consists of **axioms** which interpret some functions and predicates

Theory of Equality $T_=$

- Also referred to as empty theory
  - $S_{T_=}$: {=, a, b, c, … , f, g, h, …, p, q, r, …}
- f, g, …, p, q, … are ***uninterpreted*** functions and predicates
- "Built-in" predicate **=** is ***interpreted*** by axioms $A_{T_=}$:
  - $\forall x. x = x$ (reflexivity)
  - $\forall x, y. x = y \Rightarrow y = x$ (symmetry)
  - $\forall x, y, z. x = y \land y = z \Rightarrow x = z$ (transitivity)
  - $\forall x_1,\ldots,x_n,y_1,\ldots y_n. \land x_i = y_i \Rightarrow f(x_1, \ldots x_n) = f(y_1,\ldots y_n)$ (function congruence)
  - $\forall x_1,\ldots,x_n,y_1,\ldots y_n. \land x_i = y_i \Rightarrow p(x_1, \ldots x_n) \Leftrightarrow p(y_1,\ldots y_n)$ (predicate congruence)

Theory of Equality $T_{=}$

- Also known as theory of equality with uninterpreted functions
- Uninterpreted functions are useful as an abstraction or over-approximation mechanism
  - Remember static program verification

Theory of Presburger Arithmetic

- Presburger arithmetic: allows only addition over natural numbers
- $S_{\mathbb{N}}$: $\{0, 1, =, +\}$
- $A_{\mathbb{N}}$:
  - $\forall x.\ \neg(x+1 = 0)$ (zero)
  - $\forall x.\ x+0 = 0$ (plus zero)
  - $\forall x, y.\ x+1 = y+1 \Rightarrow x=y$ (successor)
  - $\forall x, y.\ x+(y+1) = (x+y)+1$ (plus successor)
  - $F[0] \wedge (\forall x.\ F[x] \Rightarrow F[x+1]) \Rightarrow \forall x. F[x]$ (induction)

Theory of Fixed-width Bitvectors

- Object constants are fixed-width bitvectors, e.g., 011011, 001
- Functions include extraction, concatenation, bitwise operations, arithmetic operations

Theory of Arrays

- S_A: {a, b, c, …, i, j, k, …v, w, …, =, read, write)
- At a high-level:
  - **read(a, i)** is a binary function that returns the value of array a at index i
  - **write(a, i, v)** is a ternary function that returns an array identical to a except that at index i it has value v
- A_A:
  - $\forall$ a, i, j. i = j => read(a, i) = read(a, j) (array congruence)
  - $\forall$ a, i, j, v. i = j => read(write(a, i, v), j) = v (read-over-write 1)
  - $\forall$ a, i, j, v. ¬(i = j) => read(write(a, i, v), j) = read(a, j) (read-over-write 2)

Solving SMT queries

- **Eager translation** to equisatisfiable SAT formula
  - Some theories are better matches than others
  - Multiple translations possible, SMT solver performs several transformations/optimizations in the process using information available at the theory level
    - E.g., simplifying x –x to 0.
- **DPLL[T]**
  - Adapts DPLL to work at the level of theory T (theory deduction, theory conflicts, etc.)

Combination of Theories

- Given
  - theory T_1 with signature S_{T_1} and axioms A_{T_1}
  - theory T_2 with signature S_{T_2} and axioms A_{T_2}
  - an SMT solver for T_1
  - an SMT solver for T_2
- Can we produce a solver for $T_1 \cup T_2$?
  - $T_1 \cup T_2$ with signature $S_{T_1} \cup S_{T_2}$ and axioms $A_{T_1} \cup A_{T_2}$

Nelson-Oppen Framework

- Framework for deciding combined theories under certain assumptions, e.g, only for quantifier-free theories
- Examples
  - theory of arrays and bitvectors
  - theory of arrays and integers

Nelson-Oppen Framework

- Two phases:
  - **Purification**: transform F into equisatisfiable formula F' = $F_1 \wedge F_2$ such that
    - F_1 belongs only to T_1
    - F_2 belongs only to T_2
  - **Equality propagation**: propagate equalities between theories

STP solver

- SMT solver for the theory of bitvectors and arrays
- Based on *eager* translation to SAT (uses MiniSAT)
- Developed at Stanford by Ganesh and Dill, initially targeted to, and driven by, EXE

Theory of Bitvectors and Arrays

- Can accurately encode the semantics of C programs
  - Model each memory block as an array of 8-bit BVs
  - Bind types to expressions, not bits

```
char buf[N]; // symbolic
struct pkt1 { char x, y, v, w; int z; } *pa = (struct pkt1*) buf;
struct pkt2 { unsigned i, j; } *pb = (struct pkt2*) buf;
if (pa[2].v < 0) { assert(pb[2].i >= 1<<23); }
```

```
buf: ARRAY BITVECTOR(32)OF BITVECTOR(8)
```

```
SBVLT(buf[18], 0x00)
```

```
BVGE(buf[19]@buf[18]@buf[17]@buf[16], 0x00800000)
```

Conversion to SAT

- Each arithmetic operation on bitvectors can be encoded as a circuit / formula
  - E.g., addition translated as a ripple-carry adder
- The main difficulty is removing arrays
- This is done starting from the array axioms

Eliminating Arrays

- Transformation 1: eliminate writes
  - read(write(A, i, v), j) ⇔ ite(i=j, v, read(A, j))
  - a write by itself (not inside a read) is meaningless and can be discarded
- Transformation 2: eliminate reads
  a) replace each syntactically-unique read by a fresh variable
  b) add array axioms: for each pair of indexes, if the indexes are equal, so are the corresponding introduced variables

Eliminating Reads

$$(a[i_1] = e_1) \land (a[i_2] = e_2) \land (a[i_3] = e_3) \land (i_1+i_2+i_3=6)$$

$$(v_1 = e_1) \land (v_2 = e_2) \land (v_3 = e_3) \land (i_1+i_2+i_3=6)$$
$$(i_1 = i_2 \Rightarrow v_1 = v_2) \land (i_1 = i_3 \Rightarrow v_1 = v_3) \land (i_2 = i_3 \Rightarrow v_2 = v_3)$$

STP's read elimination is expensive:

Expands each formula by $n \cdot (n-1)/2$ terms, where n is the number of syntactically distinct indexes

Array-based Refinement in STP

STP's conversion of array terms to SAT is expensive

$$(a[i_1] = e_1) \wedge (a[i_2] = e_2) \wedge (a[i_3] = e_3) \wedge (i_1+i_2+i_3=6)$$

$$(v_1 = e_1) \wedge (v_2 = e_2) \wedge (v_3 = e_3) \wedge (i_1+i_2+i_3=6)$$

~~$(i_1 = i_2 \rightarrow v_1 = v_2) \wedge (i_1 = i_3 \rightarrow v_1 = v_3) \wedge (i_2 = i_3 \rightarrow v_2 = v_3)$~~

Approximation UNSATISFIABLE → Original formula UNSATISFIABLE

Array-based Refinement in STP

STP's conversion of array terms to SAT is expensive

$$(a[i_1] = e_1) \land (a[i_2] = e_2) \land (a[i_3] = e_3) \land (i_1+i_2+i_3=6)$$

$$(v_1 = e_1) \land (v_2 = e_2) \land (v_3 = e_3) \land (i_1+i_2+i_3=6)$$

~~$(i_1 = i_2 \rightarrow v_1 = v_2) \land (i_1 = i_3 \rightarrow v_1 = v_3) \land (i_2 = i_3 \rightarrow v_2 = v_3)$~~

$i_1 = 1$
$i_2 = 2$
$i_3 = 3$
$v_1 = e_1 = 1$
$v_2 = e_2 = 2$
$v_3 = e_3 = 3$

→

$(a[1] = 1) \land (a[2] = 2) \land$
$(a[3] = 3) \land (1+2+3 = 6)$

✓

Array-based Refinement in STP

STP's conversion of array terms to SAT is expensive

$$(a[i_1] = e_1) \land (a[i_2] = e_2) \land (a[i_3] = e_3) \land (i_1+i_2+i_3=6)$$

$$(v_1 = e_1) \land (v_2 = e_2) \land (v_3 = e_3) \land (i_1+i_2+i_3=6)$$

~~$(i_1 = i_2 \rightarrow v_1 = v_2) \land (i_1 = i_3 \rightarrow v_1 = v_3) \land (i_2 = i_3 \rightarrow v_2 = v_3)$~~

$i_1 = 2$
$i_2 = 2$
$i_3 = 2$
$v_1 = e_1 = 1$
$v_2 = e_2 = 2$
$v_3 = e_3 = 3$

$$(a[2] = 1) \land (a[2] = 2) \land (a[2] = 3) \land (2+2+2 = 6)$$

Array-based Refinement in STP

STP's conversion of array terms to SAT is expensive

$$(a[i_1] = e_1) \land (a[i_2] = e_2) \land (a[i_3] = e_3) \land (i_1+i_2+i_3=6)$$

$$(v_1 = e_1) \land (v_2 = e_2) \land (v_3 = e_3) \land (i_1+i_2+i_3=6)$$

$(i_1 = i_2 \Rightarrow v_1 = v_2)$ ~~$\land (i_1 = i_3 \Rightarrow v_1 = v_3) \land (i_2 = i_3 \Rightarrow v_2 = v_3)$~~

$i_1 = 2$
$i_2 = 2$
$i_3 = 2$
$v_1 = e_1 = 1$
$v_2 = e_2 = 2$
$v_3 = e_3 = 3$

$(a[2] = 1) \land (a[2] = 2) \land$
$(a[2] = 3) \land (2+2+2 = 6)$

Array-based Refinement in STP

- When unsuccessful, which axioms to add?
- Different heuristics possible
- STP finds an array index that violates an axiom and adds all axioms involving that index

Evaluation

Solver	Total time (min)	Timeouts
STP (baseline)	56	36
STP (array-based refinement)	10	1

8495 test cases from our symbolic execution benchmarks

- Timeout set at 60s (which are added as penalty), underestimates performance differences

SMT Solvers

- SMT solvers support rich theories in classical first-order logic with equality
  - E.g., theory of Presburger arithmetic, theory of bitvectors and arrays, theory of rationals, etc.
- Approaches for SMT solving include
  - Eager translation to SAT
  - DPLL[T]
  - Nelson-Oppen framework for combining different theories