Decision Procedures in the Theory of Bit-Vectors

Sukanya Basu
Guided by: Prof. Supratik Chakraborty

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

May 1, 2010

Bit-Vectors

Definition

A bit-vector b is a vector of bits with a given length l (or dimension)

$$b \ : \ \{0, \cdots, l-1\} \rightarrow \{0, 1\}$$

- The set of all 2^l bitvectors of length l is denoted by $bvec_l$. The i-th bit of the bitvector b is denoted by b_i.

Bitvector arithmetic: Syntax

- Domain of bitvectors is finite
- Semantics of operation over unbounded types (integers, natural numbers) need special handling to be represented by bitvectors

Grammar for bitvector arithmetic

$$
\begin{aligned}
\mathit{formula} &: \mathit{formula} \wedge \mathit{formula} \mid \neg \mathit{formula} \mid (\mathit{formula}) \mid \mathit{atom} \\
\mathit{atom} &: \mathit{term}\ \mathit{rel}\ \mathit{term} \mid \mathit{Boolean} - \mathit{Identifier} \mid \mathit{term}\,[\,\mathit{constant}\,] \\
\mathit{rel} &: < \mid = \\
\mathit{term} &: \mathit{term}\ \mathit{op}\ \mathit{term} \mid \mathit{identifier} \mid \sim \mathit{term} \mid \mathit{constant} \mid \\
&\quad \mathit{atom}\,?\,\mathit{term} : \mathit{term} \mid \mathit{term}[\,\mathit{constant} : \mathit{constant}] \mid \mathit{ext}\,(\,\mathit{term}\,) \\
\mathit{op} &: + \mid - \mid \cdot \mid / \mid \ll \mid \gg \mid \& \mid | \mid \oplus \mid \circ
\end{aligned}
$$

Bitwise operators

- The binary bitwise operators take two l-bit bitvectors as arguments and return an l-bit bitvector
- Bitwise OR operator:

$$|_{[l]} : (bvec_l \times bvec_l) \rightarrow bvec_l$$

Example

$$11001000 \mid 01100100 \;=\; 11101100$$

- Bitwise AND operator:

$$\&_{[l]} : (bvec_l \times bvec_l) \rightarrow bvec_l$$

Example

$$11001000 \,\&\, 01100100 \;=\; 01000000$$

Encodings

Numbers are encoded using bitvectors

- Binary encoding
- Two's complement encoding

Binary Encoding

Let x denote a natural number, and b_l a bit vector. b is called a binary encoding of x iff

$$x = \langle b \rangle_U$$

where $\langle b \rangle_U$ is defined as follows:

Definition

$$\langle \cdot \rangle_U : bvec_l \rightarrow \{0, \cdots, 2^l - 1\},$$

$$\langle b \rangle_U = \sum_{i=0}^{l-1} b_i \cdot 2^i.$$

Example

$$\langle 11001000 \rangle_U = 200$$

Two's complement encoding

Let x denote a natural number, and $b \in bvec_l$ a bit vector, b is called a two's complement encoding of x iff

$$x = \langle b \rangle_S$$

where $\langle b \rangle_S$ is defined as follows:

Definition

$$\langle \cdot \rangle_S : bvec_l \rightarrow \{-2^{l-1}, \cdots, 2^{l-1} - 1\},$$

$$\langle b \rangle_S = -2^{l-1} \cdot b_{l-1} + \sum_{i=0}^{l-1} b_i \cdot 2^i.$$

Example

$$\langle 11001000 \rangle_S = -128 + 64 + 8 = -56$$

$$\langle 01100100 \rangle_S = 100$$

Arithmetic operators

Bit-vector arithmetic uses modular arithmetic

Example

$$\begin{aligned} 11001000 &= 200 \\ +01100100 &= 100 \\ =00101100 &= 44 \end{aligned}$$

- Addition

$$a_{[l]} +_U b_{[l]} = c_{[l]} \Longleftrightarrow \langle a \rangle_U + \langle b \rangle_U = \langle c \rangle_U mod 2^l$$

$$a_{[l]} +_S b_{[l]} = c_{[l]} \Longleftrightarrow \langle a \rangle_S + \langle b \rangle_S = \langle c \rangle_S mod 2^l$$

Mixed encoding:

$$a_{[l]U} +_U b_{[l]S} = c_{[l]} \Longleftrightarrow \langle a \rangle_U + \langle b \rangle_S = \langle c \rangle_U mod 2^l$$

Decision Procedures

- A decision procedure is an algorithm that terminates with a correct yes or no answer for a decision problem.

Deciding bitvector arithmetic

Bitvector arithmetic can be decided by

- Flattening or bit-blasting
- Incremental flattening
- Using solvers for linear arithmetic
    - Integer arithmetic
    - Fixed-point arithmetic

Flattening

- Transforms Bit-Vector Logic to Propositional Logic
- Most commonly used decision procedure
- Also called 'bit-blasting'

1. Convert propositional part
2. Add a Boolean variable for each bit of each sub-expression (term)
3. Add constraint for each sub-expression

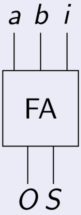
The new Boolean variable for bit i of term t is denoted by $\mu(t)_i$.

Bitvector Flattening

Example: Bitwise operator

$$a|_{[l]}b : \bigwedge_{i=0}^{l-1}(\mu(t)_i = (a_i \vee b_i))$$

Example: Arithmetic addition $a + b$



$$S \equiv (a + b + i) \mod 2 \equiv a \oplus b \oplus i$$

$$O \equiv (a + b + i) \text{ div } 2 \equiv a \cdot b + a \cdot i + b \cdot i$$

$$(a \vee b \vee \neg o) \wedge (a \vee \neg b \vee i \vee \neg o) \wedge$$
$$(a \vee \neg b \vee \neg i \vee o) \wedge (\neg a \vee b \vee i \vee \neg o) \wedge$$
$$(\neg a \vee b \vee \neg i \vee o) \wedge (\neg a \vee \neg b \vee o)$$

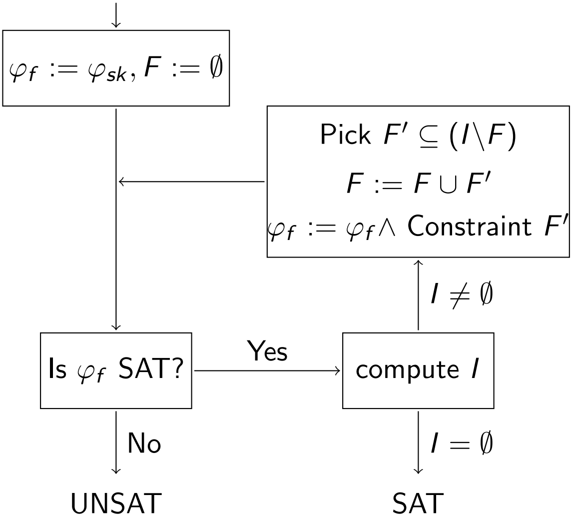
Incremental Bit Flattening

- Start with the propositional skeleton of the formula
- Add constraints for "inexpensive" operators, omit those for "expensive" operators

Example

$$a \cdot b = c \land b \cdot a \neq c \land x < y \land x > y$$

Incremental Flattening



STP

A decision procedure for the satisfiability of quatifier-free first order logic formulas with bitvectors and arrays.

Approach

- Three phases of word-level transformations
- Conversion to a purely Boolean formula and Bit-blasting
- Conversion to propositional CNF
- Solving by a SAT solver

STP: Linear Solver and Variable Elimination

- Efficiently handles linear two's complement arithmetic
- Variable eliminated by substituting in the rest of the formula
- If unable to solve an entire variable, solves for some of the lower bits
- Non-linear or word-level terms treated as bitvector variables

STP: Abstraction Refinement

- Abstract formula obtained by omitting conjunctive constraints from concrete formula
- Checked for satisfiability
  1. Unsatisfiable: Original formula definitely unsatisfiable
  2. Exists satisfying assignment to abstract formula: Converts to a purported concrete model. If original formula evaluates to true, returns without further refinement
  3. Purported model returns false: Refines abstracted formula by choosing additional conjuncts.
- Worst case: Abstracted formula made fully concrete.
- Result guaranteed to be correct because of equisatisfiability

Stanford Validity Checker

- An automatic verification tool developed at Stanford University
- Takes as input a Boolean formula in a quantifier free subset of first order logic
- The framework of SVC is divided into two parts:
  - A canonizer
  - A solver

Canonizer

- To make semantically equivalent terms have a unique representation (canonical form)
- This is complicated because of bitvector arithmetic

Example

$$(x_{[n]} +_{[n+1]} x_{[n]}) \equiv (x_{[n]} \circ 0_{[1]})$$

$$(x_{[1]} +_{[1]} 1_{[1]}) \equiv (NOT x_{[1]})$$

- Converts all expressions to a common form, bitplus expressions

Bitplus expressions

- A modulo 2^n addition expression for some fixed bit-width n of bitvector variables with constant coefficient
- Variables are ordered with duplicates eliminated, and each coefficient reduced to modulo 2^n
- A set of transformation rules are applied

Examples

$$(x_{[n]} \circ 0_{[1]}) \equiv 2^1 \cdot x_{[n]} +_{[n+1]} 0_{[1]}$$

$$(x_{[0]} +_{[m]} \cdots x_s)[i : 0] \equiv (x_{[0]} +_{[i+1]} \cdots x_{[s]})$$

Solver

- A solver for equations involving bit-vector operations
- Requires the equations to be in canonical form
- A total ordering on expressions required for determining complexity
- In case of bit-vectors, longer bit-vectors more complex than shorter ones
- The solver is called for the longest bit-vector in the equation

Solver (contd.)

- The equations that the solver attempts to solve has the general form

$$a_0 \cdot x_0 +_{[n]} \cdots a_p \cdot x_p = b_0 \cdot y_0 +_{[n]} \cdots b_q \cdot y_q$$

- The most complex variable, say $z_{[m]}$, with coefficient c, is isolated on the left-hand side. The resulting equation is of the form

$$c \cdot z_{[m]} = d_0 \cdot w_{0_{[m[0]]}} +_{[n]} \cdots d_j \cdot w_{j_{[m[j]]}}$$

- Coefficient is odd
- Coefficient is even

Integrated Canonizer and Solver

- Decision procedure developed at SRI International
- Quantifier free, first-order theory
- Equality and disequality with both uninterpreted and interpreted function symbols
- Arithmetic, tuples, arrays, sets and bit-vectors
- Core is a congruence closure procedure
- Provides an API, suitable for use in applications with highly dynamic environments

Conclusion

- Notable applications of STP include the EXE project
- Fully exploits the speed of modern SAT solvers
- Primary application for SVC is microprocessor verification
- Has been applied to the TORCH microprocessor
- Is claimed to be complete and automatic
- Sometimes bitplus expressions benefit the core theory of concatenation and extraction
- Currently the more evolved version of SVC is CVC and CVC-lite
- ICS is however deprecated since August 2006 and is no longer supported
- It has been replaced by Yices