An Alternative to SAT-based Approaches for Bit-Vectors

Sébastien Bardin, Philippe Herrmann, Florian Perroud

CEA-LIST, Software Safety Labs
(Paris, France)

Motivation

Theory of bit-vectors (BV)

- variables interpreted over fixed-size arrays of bits
- standard low-level operators

BV increasingly popular in software verification

- bounded model checking [Clarke-Kroening-Lerda, TACAS 2004]
- symbolic execution [Cadar-Ganesh-Dill+, CCS 2006]
- extended static checking [Babic-Hu, ICSE 2008]

Why ?

- very precise modelling of low-level constructs
- allows multiplication between variables

The theory of bit-vectors

Variables range over arrays of bits

- a BV variable A has a given size $size(A)$
- $A = a_1 \ldots a_n$ where $a_i \in \{0, 1\}$
- unsigned integer semantics (size n) : $[\![A]\!]_u = \sum_{i=1}^{n} a_i \cdot 2^{i-1}$
- signed integer semantics

Common operations

- bitwise : $\sim, \&, |, xor$
- arithmetic : $\oplus, \ominus, \otimes, \oslash_u, \oslash_s, \%_u, \%_s$
- relations : $=, \neq, \leq_u, <_u, \leq_s, <_s$
- shifts : $\ll, \gg_u, \gg_s$
- extensions : $ext_u(A, k), ext_s(A, k)$
- concatenation : $A :: B$
- extraction : $A[i..j]$

Bit-blasting

Bit-blasting : standard way to solve problems over BV

- encode BV formula into an equisatisfiable boolean formula
- each BV A is encoded into a set of boolean variables $a_1, \ldots, a_n$
- each BV operator is encoded into a logical circuit

Bit-blasting : pros and cons

Very main advantage : rely on the efficiency of SAT solvers

- small effort for good performance
- integration into SMT solvers [Stp,Boolector,MathSat,etc.]

Shortcomings

- formula explosion : too large boolean formulas on some "arithmetic-oriented" BV-formulas
- no more information about the BV-formula structure : may miss high-level simplifications

Our approach : word-level CLP-based BV solving

Goal : outperform SAT on arithmetic-oriented BV formulas

Strategy : word-level approach

- reason on bit-vectors rather than on their separate bits
- BV variables are encoded into bounded integer variables
- BV operators are seen as integer arithmetic operators

Technology : CLP(FD)

- Constraint Logic Programming over Finite Domains
- handle all common arithmetic operators

Restriction : only conjunctive formulas (useful : symbolic execution)

CLP(FD)

Natural extension of DPLL

- each variable ranges over a finite domain

Smart exploration of the tree of partial valuations of the variables

- two steps are interleaved
- propagation : reduce the domain of each variable by removing some inconsistent values
- search : standard “label & backtrack” procedure

Example : constraint $x \leq y$ with $D_x = [50..100]$ and $D_y = [30..70]$

- (propagation) reduce both D_x and D_y to $[50..70]$
- (search) no more propagation, x is arbitrary labelled to 62
- (propagation) D_y is reduced to $[62..70]$
- (search) y is labelled to 68, the procedure returns SAT

Contribution

Difficulty

- word-level CLP-based approaches already tried

  [Diaz-Codognet 01, Ferrandi-Rendine-Sciuto 02]
- performance very far from SAT-based approaches

  [Sülflow-Kühne+ 07]

Existing works rely on standard CLP(FD)

- for small domains and/or linear integer arithmetic
- does not fit the needs of word-level BV solving

Our results

- a new CLP(BV) framework dedicated to BV solving
- fill the gap with the best SAT approaches
- better scaling than SAT approaches w.r.t. BV sizes

Rest of the talk

Why CLP(FD) and direct encoding do not work

Basic ingredients of the CLP(BV) framework

Some experiments

Direct word-level encoding : examples

Each bit-vector A is encoded by its unsigned integer value $[\![A]\!]_u$
Bit-vectors operators are encoded by common integer operators

- (expensive) ext_s(A,k) = R
  - become $R = ite(([\![A]\!]_u < 2^{N-1})?\,[\![A]\!]_u : [\![A]\!]_u + 2^k - 2^{size(A)})$
  - introduce case-split
- (very expensive) A & B = R
  - perform bit-blasting
  - introduce A_is, B_is and R_is in $\{0,1\}$
  - $R_1 = min(A_1, B_1) \wedge \ldots \wedge R_n = min(A_n, B_n)$
    $\wedge \sum A_i \cdot 2^{i-1} = [\![A]\!]_u \wedge \sum B_i \cdot 2^{i-1} = [\![B]\!]_u \wedge \sum R_i \cdot 2^{i-1} = [\![R]\!]_u$

CLP(FD) and BV : why it does not work

1- Domain size : finite but huge domains

- CLP(FD) solvers with concrete domains do not scale

2- Inefficient translation

- large scale CLP(FD) solvers tuned for linear arithmetic
- do not perform well on non-linear operations, case-splits, boolean values, etc.
- the direct word-level encoding falls in the worst category

3- Inadequate symbolic domains

- large scale CLP(FD) solvers based on (single) intervals
- does not propagate anything for BV (see after)

CLP(FD) and BV : why it does not work (2)

Unions of intervals are mandatory for BV because of overflows

- $a \oplus 3 = b$ with $N = 8$, $D_a = [251..255]$ and $D_b = [0..255]$
- with Is : D_b can be reduced to $D'_b = [0..2] \cup [254..255]$
- with I : no propagation, $D'_b = [0..255]$

A dedicated CLP(BV) framework

Dedicated propagators for Is/C domain

- no introduction of additional variables
- no introduction of "modulo" operation everywhere
- signed operations handled without any case-split

The new domain BL (bitlist) and its propagators

- no bit-blasting on bitwise operators
- efficient propagation on most "linear bitwise" operations

Framework

- each CLP variable has a Is/C domain and a BL domain
- each BV-constraint has propagators for Is/C and for BL
- propagators to share information between BL and Is/C

Implemented on top of COLIBRI [Marre-Blanc 05]

Dedicated Is/C propagators

Is propagators

- forward and backward propagation of Is
- interleaved until a fixpoint is reached

Signed operators : perform a case-split inside the propagator

For bit-wise operations : very approximated propagation

- A & B = R : propagated like $A \geq R \wedge B \geq R$
- we rely on BL-propagators for these constraints

Other

- congruence propagation
- simplification rules

(preciseness : see the discussion about arc-consistency in the paper)

BL domain

BL (bitlist) : abstract domain designed to be combined with Is/C

The bitlist of A records the known bits of A

- fixed size arrays of values in $\{\perp, 0, 1, \star\}$ (called $\star$-bits)
- $bl_A[k] = 0$ implies that $A[k] = 0$
- $bl_A[k] = 1$ implies that $A[k] = 1$
- $bl_A[k] = \star$ does not imply anything
- $bl_A[k] = \perp$ indicates a contradiction

BL propagators

Propagators : forward and backward propagation of $\star$-bits

Propagators for non-arithmetic operators

- precise and efficient propagation

Propagators for arithmetic operators

- limited form of bit-blasting inside the propagator
- very restricted propagation
- we rely on Is/C propagators for these constraints

Consistency propagators : designed to enforce consistency between the different domains of a same variable

From BL to Is/C

- if $bl_X = \star 1 \star 101$ then $X \in [21..61]$
- if $bl_X = \star 1 \star 101$ then $X \equiv 5 \ mod \ 8$

From Is/C to BL

- (N=6) if $D_x = [0..15]$ then $bl_X = 00 \star\star\star\star$
- (N=6) if $X \equiv 5 \ mod \ 8$ then $bl_X = \star\star\star 101$

Experiments

Implementation : CLP(BV) implemented on top of COLIBRI

Goal : comparison of CLP(BV), CLP(FD) and SAT

Test bench

- 164 problems from the SMTLIB or generated by Osmose
- Mostly 32-bit, up to 1,700 variables and 17,000 operators

Experiment 1 : CLP(BV) vs CLP(FD) vs SAT

Tool	Category	Time	# success
Eclipse/IC	CLP(FD)	1750	79/164
COLIBRI	CLP(FD)	2436	43/164
COL-D	CLP(BV)	893	125 /164
COL-D-BL	CLP(BV)	712	138/164
MathSat	SAT	794	128/164
STP	SAT	618	144/164
Boolector	SAT	291	157/164

Time out = 20s

CLP(BV) vs CLP(FD) vs SAT : results

CLP(BV) vs CLP(FD)

- CLP(BV) outperforms largely CLP(FD) for bit-vectors
- each feature induces a new increase in performance
- results are stable w.r.t. the search heuristics *(see the paper)*

CLP(BV) vs SAT

- CLP(BV) performs roughly like SAT approaches
- however, still behind the very best approaches
- CLP(BV) is better on NLA *(see the paper)*
- CLP(BV) scales better w.r.t. bit-width *(see the paper)*

Conclusion

Word-level CLP-based approach for BV solving

Results

- a new CLP(BV) framework dedicated to BV solving
- largely increase performance compared to direct CLP(FD)
- fill (most of) the gap with the best SAT approaches
- better scaling than SAT approaches w.r.t. BV sizes

Future work

- still room for improvement (search, global constraints)
- handle arbitrary logical connectors
- handle array operations