

Beaver: Engineering an Efficient Bit-Vector SMT Solver

Susmit Jha, Rhishikesh Limaye and Sanjit A. Seshia

UC Berkeley

CAV 2009

July 1, 2009

Satisfiability Modulo Theory (SMT)

- ▶ **SMT** is satisfiability checking beyond propositional logic. Decide the satisfiability of a first order formula with respect to a background theory.

Satisfiability Modulo Theory (SMT)

- ▶ **SMT** is satisfiability checking beyond propositional logic. Decide the satisfiability of a first order formula with respect to a background theory.
 - ▶ **Examples** of relevant theories
 - ▶ uninterpreted functions: $x = y \wedge f(x) \neq f(y)$
 - ▶ difference logic: $x - y < 7$
 - ▶ linear integer arithmetic: $3x + 2y < 12$
 - ▶ arrays: $read(write(M, a0, v0)a1)$
 - ▶ **bit-vectors**
- and their combinations.

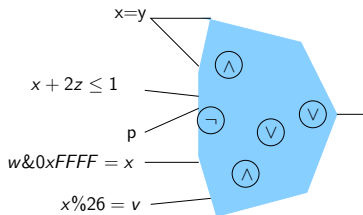
Satisfiability Modulo Theory (SMT)

- ▶ **SMT** is satisfiability checking beyond propositional logic. Decide the satisfiability of a first order formula with respect to a background theory.
 - ▶ **Examples** of relevant theories
 - ▶ uninterpreted functions: $x = y \wedge f(x) \neq f(y)$
 - ▶ difference logic: $x - y < 7$
 - ▶ linear integer arithmetic: $3x + 2y < 12$
 - ▶ arrays: $read(write(M, a0, v0)a1)$
 - ▶ **bit-vectors**
- and their combinations.
- ▶ **Beaver is an open source SMT solver over the theory of bit-vectors (QF_BV)**

Bit-vector Logic

Formula in propositional logic augmented with

- ▶ Bit-vector terms and functions:
 - ▶ constants;
 - ▶ variables;
 - ▶ bit-vector arithmetic: such as `bvadd`, `bvsub`, `bvmul`, `bvmod`, `bvdiv`
 - ▶ bit-wise operations: such as `bvor`, `bvand`, `bvshiftl`
 - ▶ Predicates over bit-vector terms: such as `bvgt`, `bvult`
- ▶ Boolean combination of predicates: `and`, `or`, `not`, `iff`, etc.



Applications

- ▶ Example applications
 - ▶ Verification of implementations (hardware or software) - Model checking, equivalence checking, generation of invariants
 - ▶ Security - exploit generation, generating vulnerability signatures.
 - ▶ Testing - path constraint feasibility check

Bit-vector logic more accurately models the hardware/software implementations.

Related work

- ▶ Very widely investigated in literature with number of efficient tools
 - ▶ BAT, Boolector, CVC3, Spear, UCLID, Yices, Z3, and many others

Only CVC3 is open source among all available bit-vector SMT solvers.

Related work

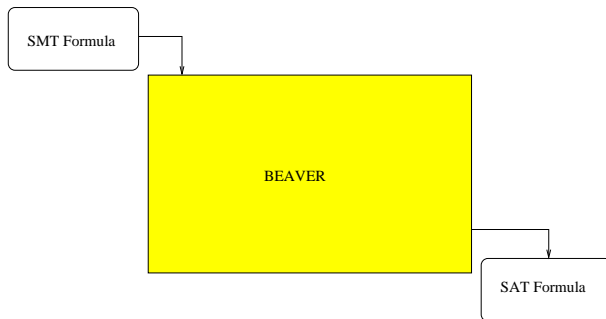
- ▶ Very widely investigated in literature with number of efficient tools
 - ▶ BAT, Boolector, CVC3, Spear, UCLID, Yices, Z3, and many others

Only CVC3 is open source among all available bit-vector SMT solvers.

- ▶ State of the art in bit-vector solving and comparison of different solvers available through the [SMTLIB/SMTCOMP initiative](http://combination.cs.uiowa.edu/smtlib/) hosted at <http://combination.cs.uiowa.edu/smtlib/>

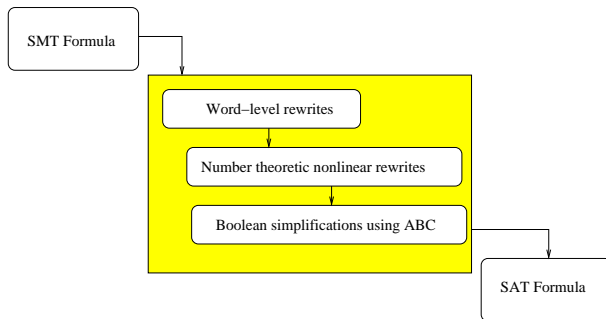
Beaver Approach

Beaver is essentially a compiler from bit-vector SMT to SAT



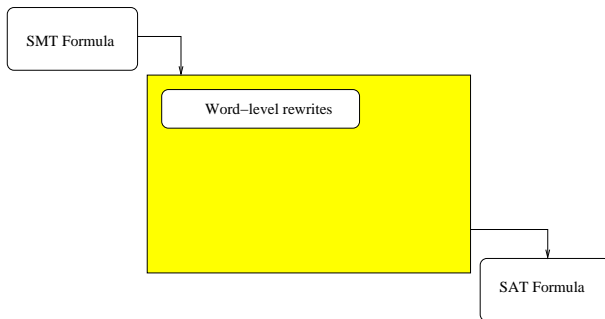
Beaver Approach

Beaver is essentially a compiler from bit-vector SMT to SAT



Beaver Approach

Beaver is essentially a compiler from bit-vector SMT to SAT



Beaver Approach: Word-level rewrites

- ▶ Structural hashing on the fly for some obvious local simplifications.

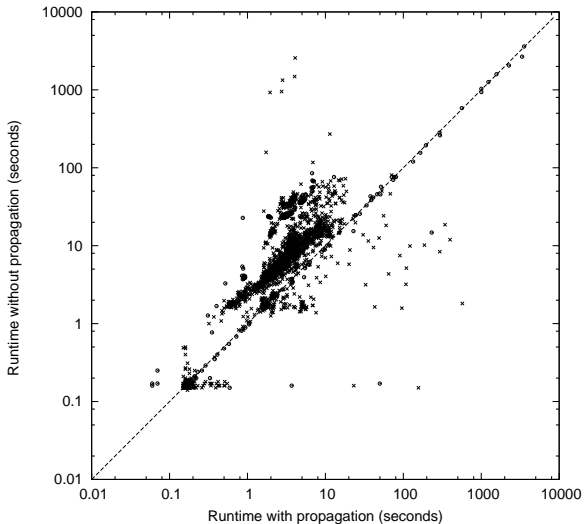
Beaver Approach: Word-level rewrites

- ▶ Structural hashing on the fly for some obvious local simplifications.
- ▶ Forward and backward propagation of constant and equality constraints.

Beaver Approach: Word-level rewrites

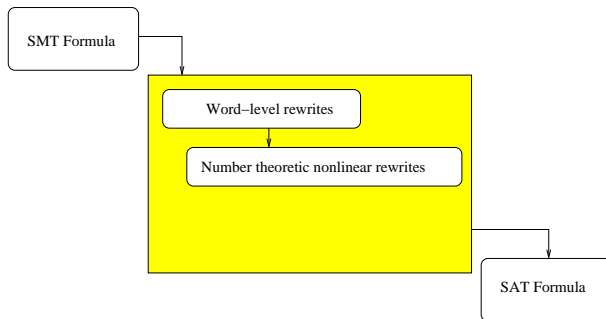
- ▶ Structural hashing on the fly for some obvious local simplifications.
- ▶ Forward and backward propagation of constant and equality constraints.
- ▶ Rewrite rules applied in an online event-driven manner by maintaining a queue for sites where rewrites can be applied.

Propagation of constants and equality



Beaver Approach

Beaver is essentially a compiler from bit-vector SMT to SAT



Beaver Approach: Number theoretic non-linear rewrites

- ▶ Transform division with constants to multiplication with constants by computing inverse modulo 2^k .

Beaver Approach: Number theoretic non-linear rewrites

- ▶ Transform division with constants to multiplication with constants by computing inverse modulo 2^k .
- ▶ Decompose multiplication constraint of large bit-width to conjunction of k multiplications of smaller bit-widths modulo primes $2, 3, \dots, p_k$.
 - ▶ *Theorem:* We need at most $4w/\log(2w)$ primes if $w > 29$ (for smaller w , we can find out k individually) to replace multiplication with equi-satisfiable set of smaller bitwidth (that is, $\log p_i$) multiplications.
 - ▶ 16 bit multiplication requires first 10 primes, max bitwidth of decomposed multiplication is 5 bit ($p_{10} = 29$) !

Beaver Approach: Number theoretic non-linear rewrites

- ▶ Transform division with constants to multiplication with constants by computing inverse modulo 2^k .
- ▶ Decompose multiplication constraint of large bit-width to conjunction of k multiplications of smaller bit-widths modulo primes $2, 3, \dots, p_k$.
 - ▶ *Theorem:* We need at most $4w/\log(2w)$ primes if $w > 29$ (for smaller w , we can find out k individually) to replace multiplication with equi-satisfiable set of smaller bitwidth (that is, $\log p_i$) multiplications.
 - ▶ 16 bit multiplication requires first 10 primes, max bitwidth of decomposed multiplication is 5 bit ($p_{10} = 29$) !
- ▶ Residue constraints of large bit-width bitvectors with respect to constants simplified using linear constraints over ITE terms.
 - ▶ Similar to grade school method to calculate remainders in base 10 - for 3, we could calculate sum of digits and then sum of

Beaver Approach: Number theoretic non-linear rewrites

- ▶ Magic numbers: If there is k bitwidth division with constant c , one can calculate inverse of c modulo 2^k as convert division to multiplication.
 - ▶ Modulo multiplicative inverses (for non power of 2) are used to convert division by a constant into an ordinary multiplication modulo 2^k .
 - ▶ Adapted from <http://www.hackersdelight.org/magic.htm>

Beaver Approach: Number theoretic non-linear rewrites

Decomposing multiplication into smaller bitwidth multiplications using Chinese remainder theorem.

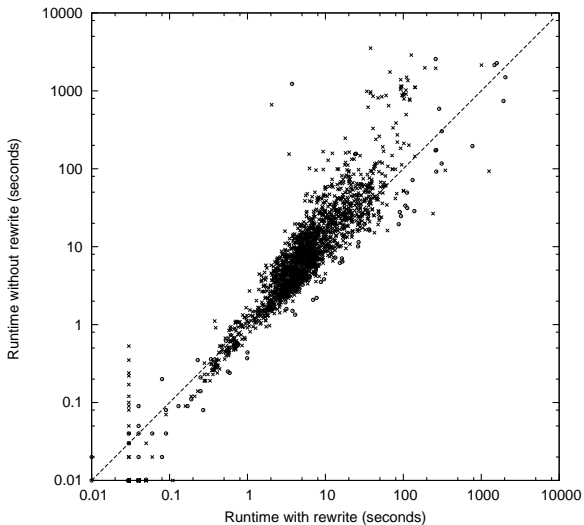
- ▶ $\phi \rightarrow \phi[t'/x * y] \wedge t' = \text{extract}[w - 1 : 0]t \wedge t = [x *_{\text{nonverflow}} y]$
where x, y, t' are of bit-width w and t is of bit-width $2w$.
- ▶ Rewrite $t = [x *_{\text{nonverflow}} y]$ as a conjunction of smaller bitwidth multiplications $t \bmod p_i = (x \bmod p_i * y \bmod p_i) \bmod p_i$ where p_i is a prime $2, 3, 5 \dots, p_k$.
- ▶ **Theorem:** We need at most $4w/\log(2w)$ primes if $w > 29$ (for smaller w , we can find out k individually) to replace multiplication with equi-satisfiable set of smaller bitwidth (that is, $\log p_i$) multiplications.
- ▶ 16 bit multiplication requires first 10 primes, max bitwidth of decomposed multiplication is 5 bit ($p_{10} = 29$) !

Beaver Approach: Number theoretic non-linear rewrites

Rewriting modulo constant (say, c) constraints using a table-lookup of $2^k \bmod c$.

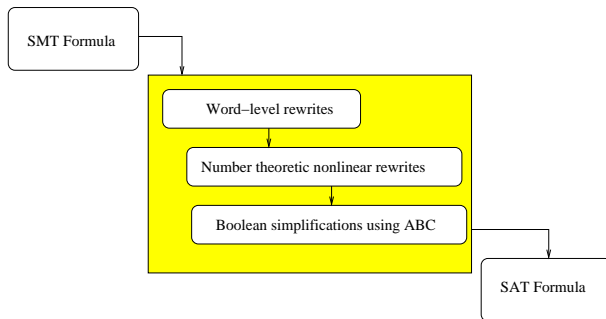
- ▶ Let us consider a constraint $y = b_3b_2b_1b_0 \bmod 3$. Consider, $y^1 =$
 $(if(b_3)then(2^3 \bmod 3)else0) + (if(b_2)then(2^2 \bmod 3)else0) +$
 $(if(b_1)then(2^1 \bmod 3)else0) + (if(b_0)then(2^0 \bmod 3)else0),$
that is, $y_1 = 2b_3 + b_2 + 2b_1 + b_0 \leq 6$, Thus, y^1 would be of three bits and we can again repeat the above and further reduce the bitwidth or directly bitblast $y = y^1 \bmod 3$.
- ▶ Similar to grade school method to calculate remainders in base 10 - for 3, we could calculate sum of digits and then sum of the sum, and so on, till we get a value less than 10.

Rewrite rules for nonlinear arithmetic



Beaver Approach

Beaver is essentially a compiler from bit-vector SMT to SAT



Beaver Approach : Bitblast and Boolean formula

- ▶ Use a presynthesized and optimized circuit templates for the bitvector operators to bitblast the formula.

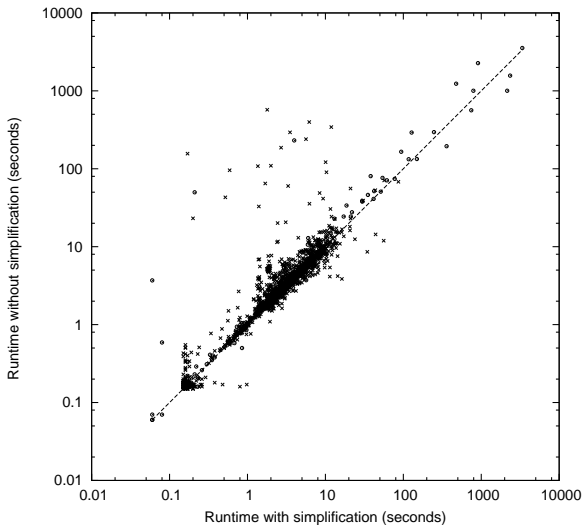
Beaver Approach : Bitblast and Boolean formula

- ▶ Use a presynthesized and optimized circuit templates for the bitvector operators to bitblast the formula.
- ▶ Use logic synthesis tool ABC over the Boolean formula to reduce SAT runtime.

Beaver Approach : Bitblast and Boolean formula

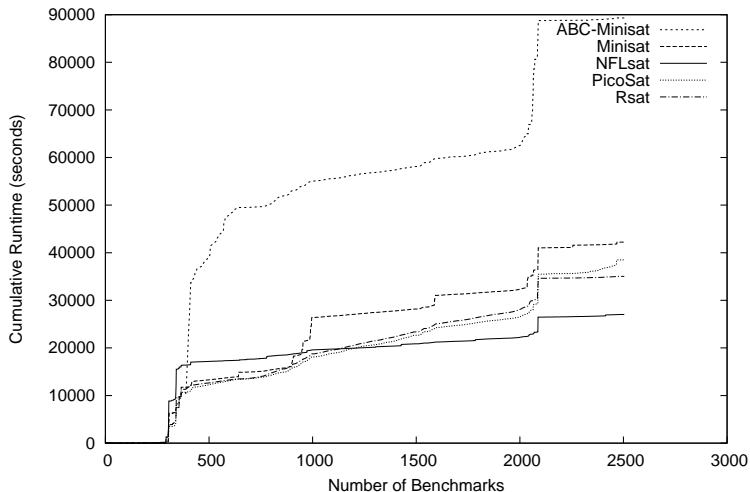
- ▶ Use a presynthesized and optimized circuit templates for the bitvector operators to bitblast the formula.
- ▶ Use logic synthesis tool ABC over the Boolean formula to reduce SAT runtime.
- ▶ We tried a number of clausal SAT solvers and the only publicly available non-clausal SAT solver - NFLSat.
 - ▶ We found that Beaver's sensitivity to different clausal SAT solvers is low.
 - ▶ The circuit SAT solver NFLSat has synergy with Beaver and gives the best performance.

Using logic synthesis



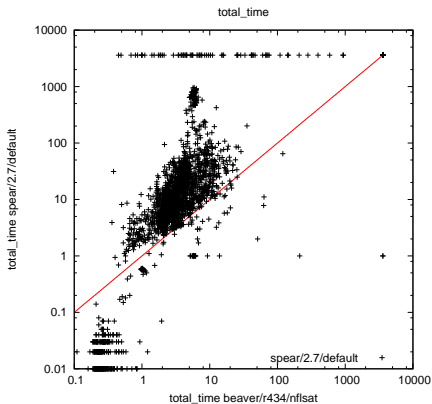
Comparison with different SAT solvers

Comparing different SAT solvers used with Beaver



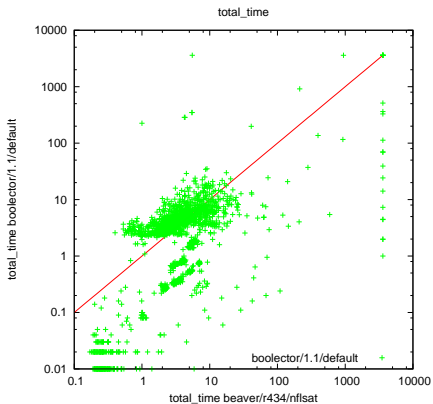
Comparison with Spear

Spear won the SMTCOMP07 year before last.



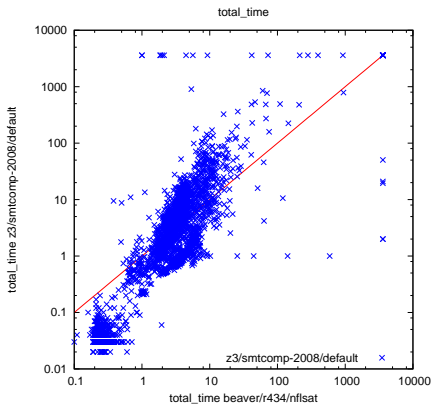
Comparison with Boolector

Boolector was ranked 1 in SMTCOMP08 and Beaver was ranked 3



Comparison with Z3

Z3 was ranked 2 in SMTCOMP08 and Beaver was ranked 3



Conclusion: How does Beaver make any difference to you ?

- ▶ **As a bit-vector SMT solver user:** Several optimizations can be switched on and off as a parameter to Beaver. One can find the best set of simplification rules for a particular application. SAT solver to be used as back-engine is also accepted as a parameter. Current users are in industry (such as Intel, NuSym) as well as academia.

Conclusion: How does Beaver make any difference to you ?

- ▶ **As a bit-vector SMT solver user:** Several optimizations can be switched on and off as a parameter to Beaver. One can find the best set of simplification rules for a particular application. SAT solver to be used as back-engine is also accepted as a parameter. Current users are in industry (such as Intel, NuSym) as well as academia.
- ▶ **As an SMT solver developer:** If you have a cool idea for SMT solving, you can just implement it over Beaver and have no overhead in catching up with the crowd as Beaver is competitive to best solvers. As of the last SMTCOMP, Beaver and CVC3 are the only **open source** SMT solver.

Conclusion

- ▶ Beaver has
 - ▶ OCaml code base
 - ▶ Modular design which is easily extensible
 - ▶ Documentation prepared using Ocamldoc
 - ▶ Released under BSD
- ▶ Tool webpage: uclid.eecs.berkeley.edu/Beaver
 - ▶ Source code and documentation
 - ▶ Executable (linux-32/64 and cygwin 32 binary)
 - ▶ Documentation of used word-level simplification rules
 - ▶ Results of experiments
- ▶ Google group:
<http://groups.google.com/group/beaver-bitvector-smt>