Linear Algebra, Boolean Rings and Resolution?

Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University
Linz, Austria

ACA'08

Applications of Computer Algebra

Symbolic Computation and Deduction in
System Design and Verification

Castle of Hagenberg, Austria

Wednesday, July 30, 2008

- functional-level
  - high-level descriptions of algorithms assume infinite memory
  - for instance infinite tape, integers, functional languages, ...
- word-level
  - 32 or 64 bit systems
  - modular arithmetic instead of integer arithmetic
  - pointer arithmetic
- bit-level
  - HW designs synthesized to bit-level
  - processors all work on the bit-level

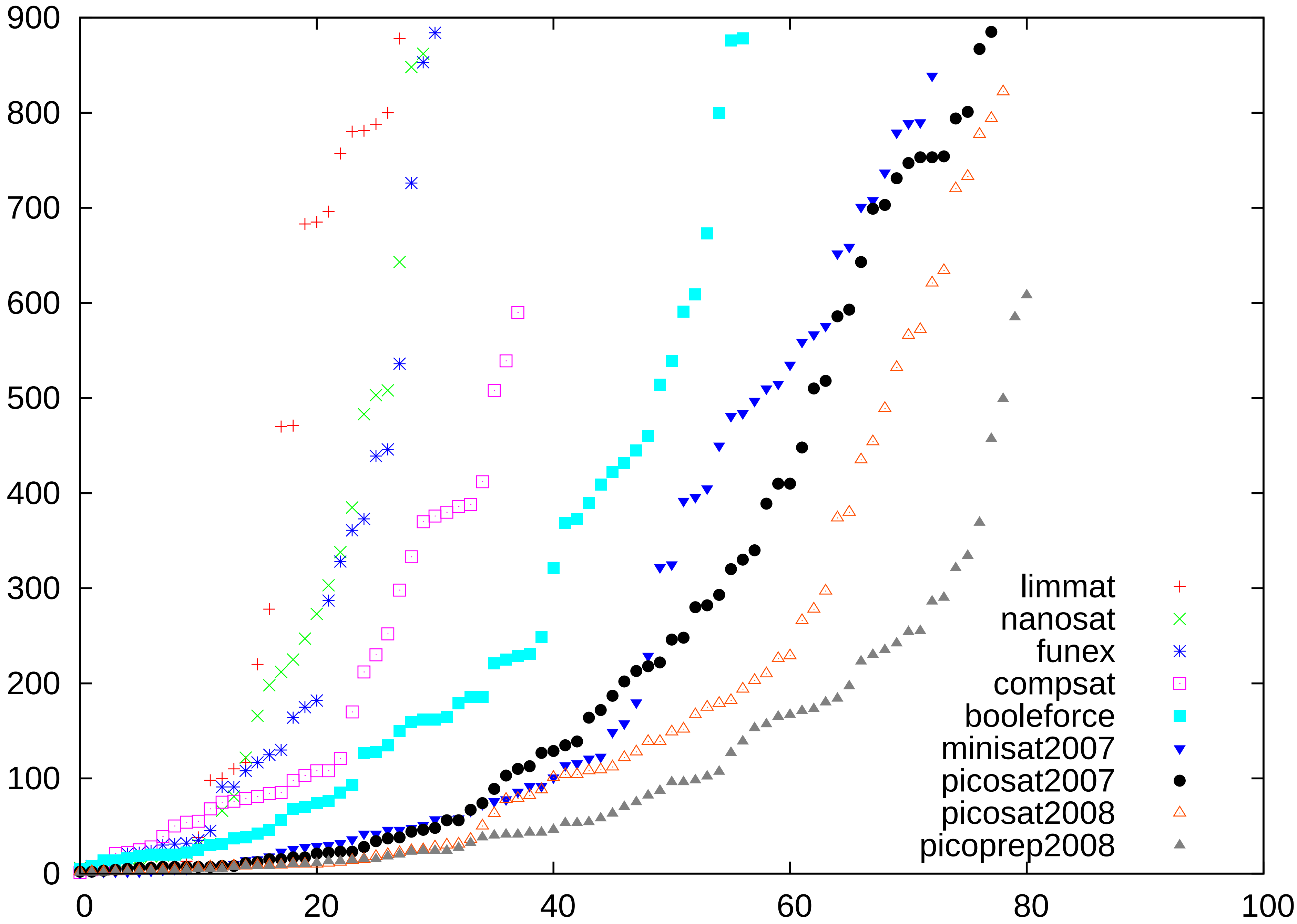
- equivalence checking (HW)
  - heavy (mostly) automatic optimizations on the bit-level
  - comparison with “golden” original implementation
- cryptanalysis (particularly of stream-cyphers)
  - described on the word-level
  - many bit-level operations, e.g. LFSR, AND-gates, XOR-gates
- verifying modular arithmetic in SW

  “Nearly All Binary Searches and Mergesorts are Broken”

  ```
  int l, r, m; ...  m = (l + r)/2; ...
  ```

- DPLL (still!) plus
  - learning: GRASP, RelSAT, SATO
  - VSIDS decision heuristics: Chaff, MiniSAT, PicoSAT, ...
  - ... and many more (important) optimizations:
    restarts, pre-processing, data structures
- driven by yearly SAT competition / SAT race and **many** applications
- extensions to *satisfiability modulo theories* (SMT)
- **the** formal core technology in industry:
  equivalence checking, bounded and unbounded model checking, synthesis
  test case generation, coverage, consistency checking, configuration ...

Cactus Plot for SAT'06 Race Instances



- equivalence checking of arithmetic circuits (on the bit-level) is very difficult
  - for instance associativity: `x * (y * z)` vs `(x * y) * z`
  - needs four 32x32 to 32 bit multipliers after "bit-blasting"
  - again: we need to reason on the bit-level!!

- breaking a stream cypher also needs bit-level reasoning
  - long XOR-chains are bad for standard SAT solvers
  - example: compute parity with two structural different circuits

- Why not use algebraic methods for boolean rings?

- $+$ = XOR $\cdot$ = AND $K = \mathbb{Z}_2 = \{0,1\}$

- SAT usually works on conjunctive normal form (CNF)

  – we can either transform CNF into Ideal

  $(\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)$ satisfiable

  iff

  $1 + a(b+1) = 1, \quad 1 + a(c+1) = 1, \quad 1 + (a+1)bc = 1$ solvable

  iff

  $\langle ab + a, ac + a, abc + b + c \rangle \neq \langle 1 \rangle$

  with

  $\neg a = 1 + a, \quad a \vee b = \neg(\neg a \wedge \neg b) = 1 + (a+1)(b+1) = ab + a + b$

  – or apply similar transformation/encoding of original problem (Tseitin)

- linear algebra
  - Gaussian elimination
  - provides a generalization of various techniques for "equivalence reasoning"
  - can still not be applied blindly (SAT solvers handle million of variables)
  - similar integration as in SMT solvers? DPLL (LA($\mathbb{Z}_2$))
- polynomials
  - computing Gröbner bases with Buchberger's algorithm
  - brute force too expensive (similar problems as DP algorithm)
  - refutational completeness useless in practice
  - useful for preprocessing (?!)

- given two square $n \times n$ matrices A, B over $\mathbb{Z}_2$, then $AB = 1 \Rightarrow BA = 1$

- algebraic bit-level encoding: n^2 polynomials for LHS, n^2 polynomials for RHS
  - compute Gröbner basis for LHS
  - check that each of the RHS polynomials is contained in the generated ideal

- CNF encoding: circuits of size $O(n^3)$ for both LHS and RHS

- benchmark in the crafted category of the SAT solver competition (linvrinv)
  - SAT solvers: $n = 4$: seconds $n = 5$: 800 - 2000 seconds $n = 6$: unsolved
  - Singular: $n = 4$: seconds $n = 5, 6$: unsolved

```
ring r = 2, (
  x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,
  x21,x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,x33,x34,x35,x36,x37,x38,
  x39,x40,x41,x42,x43,x44,x45,x46,x47,x48,x49,x50), dp;

ideal I = [
  x1*x2+x3*x12+x5*x22+x7*x32+x9*x42+1,x1*x4+x3*x14+x5*x24+x7*x34+x9*x44,
  x1*x6+x3*x16+x5*x26+x7*x36+x9*x46,x1*x8+x3*x18+x5*x28+x7*x38+x9*x48,
  x1*x10+x3*x20+x5*x30+x7*x40+x9*x50,x11*x2+x13*x12+x15*x22+x17*x32+x19*x42,
  x11*x4+x13*x14+x15*x24+x17*x34+x19*x44+1,
  x11*x6+x13*x16+x15*x26+x17*x36+x19*x46,
  x11*x8+x13*x18+x15*x28+x17*x38+x19*x48,
  x11*x10+x13*x20+x15*x30+x17*x40+x19*x50,
  x21*x2+x23*x12+x25*x22+x27*x32+x29*x42,x21*x4+x23*x14+x25*x24+x27*x34+x29*x44,
  x21*x6+x23*x16+x25*x26+x27*x36+x29*x46+1,
  x21*x8+x23*x18+x25*x28+x27*x38+x29*x48,
  x21*x10+x23*x20+x25*x30+x27*x40+x29*x50,x31*x2+x33*x12+x35*x22+x37*x32+x39*x42,
  x31*x4+x33*x14+x35*x24+x37*x34+x39*x44,x31*x6+x33*x16+x35*x26+x37*x36+x39*x46,
  x31*x8+x33*x18+x35*x28+x37*x38+x39*x48+1,
  x31*x10+x33*x20+x35*x30+x37*x40+x39*x50,
  x41*x2+x43*x12+x45*x22+x47*x32+x49*x42,x41*x4+x43*x14+x45*x24+x47*x34+x49*x44,
  x41*x6+x43*x16+x45*x26+x47*x36+x49*x46,x41*x8+x43*x18+x45*x28+x47*x38+x49*x48,
  x41*x10+x43*x20+x45*x30+x47*x40+x49*x50+1];

ideal J = groebner (I);
```

- Gaussian Elimination in $\mathbb{Z}_2$ can be simulated by (RO)BDD operations
  - BDD to store a linear equation is linear in the number n of variables
  - XOR operation on BDDs for lin. equations has linear complexity in n
  - in general, BDD operations are in $O(n^2)$
- BDD operations can be simulated by extended resolution [SinzBiere-CSR'06]
  - extension rule: add literal equation $a = b \wedge c$ with fresh a
  - extended resolution is the most powerful bit-level proof system
  - proof linear in the number of recursive BDD computation steps
  - proofs are used in many applications

- same idea does not lift to polynomials:
  - ROBDD size quadratic in the size of the represented polynomial (?)
  - complexity of operations totally unclear
- conjecture:
  - ROBDDs can **not** simulate Buchberger's algorithm linearly
  - unclear whether other BDD variants allow linear simulations
- challenge
  - directly generate (extended) resolution proofs from polynomial reasoning

Summary

- a case for bit-level reasoning ...

- SAT solvers made and are still making tremendous progress

- difficult: arithmetic on the bit-level and cryptanalysis

- Stephen Cook's SAT'04 challenge captures the essence of this problem

- algebraic methods (out of the box) provide no silver bullet

- we need combinations of algebraic methods with SAT on the bit-level

- extensions to word-level (bit-vector) decisions procedures ? $\Rightarrow$ Boolector