Lessons Learnt – Seven Years of CryptoMiniSat

Presentation at PoS 2016

Mate Soos

Gotham Digital Science
A Stroz Friedberg Company

4th of July 2016

Outline

About Me

- Information security professional
- Working at information security firm, helping firms secure their products&services over the whole life-cycle
- Doing SAT (and some SMT) purely as a hobby, after-hours, weekends
- My employer, Gotham Digital Science, paid for me to come here. Thanks!

Bird's eye view of CryptoMiniSat

- Modern, inprocessing, parallel SAT solver
- First commit 10pm, 10th of Aug, 2009. Already had XOR
- Fully open source, even during competition. 9377 commits
- $\tilde{4}$7kLoC of code (build system > MiniSat code base)
- 300+ bug reports closed, 5 open
- LGPLv2. Can be safely used or linked, but released binaries must come with code
- 180+ citations in Google Scholar to "Extending SAT Solvers to Cryptographic Problems"

SAT Features

- DRAT
- SCC, Eq. literal replacement
- Binary cache, Stamping
- Implicit 2- and 3-long clauses
- Probing, Intree probing
- Hyper-binary resolution, transitive reduction
- Clause distillation
- Subsumption, self-subsuming resolution, BVA, BVE
- Component discovery and solving
- Variable renumbering
- Changeable restart and clause cleaning strategies
- XOR $\rightarrow$ CNF, XOR recovery, Matrix recovery
- On-the-fly Gauss-Jordan elimination

Inprocess Schedule

handle-comps, scc-vrepl, cache-clean, cache-tryboth,
sub-impl, intree-probe, probe,
sub-str-cls-with-bin, distill-cls,
scc-vrepl, sub-impl, str-impl, sub-impl,
occ-backw-sub-str, occ-clean-implicit, occ-bve, occ-bva, occ-xor,
str-impl, cache-clean, sub-str-cls-with-bin, distill-cls,
scc-vrepl, check-cache-size, renumber,
occ-gauss

Other Features

- Python interface
- SQLite, MySQL data dumping
- Web-based data examination
- AWS scripts to run tests
- Extensive fuzz-testing
- Some component-testing
- On-the-fly reconfiguration
- Automatic push-based build, test, fuzz/unit/acceptance-test, static analyse, code coverage for 10+ Linux & Windows configs

Outline

Bogoprops

- We all know the funny story about Fibonacci number func
- Edge cases *will* be exercised. Always.
- SCC is implemented in recursive fashion. Just hit my ankle, "unlimited" stack is 2MB/thread. Crash.
- Everything not $O(n)$ can go haywire. Must measure *everything*
- CPU opcounts would be cheap – but non-reproducible on different builds/compilers
- CMS uses "Bogoprops". Incremented on expensive operations (mostly mem ops). Yes, kinda like Knuth's memops
- Bogoprops is *everywhere*, even SCC, which is $O(e + v)$
- Intree probing needs scc-eqlit until fixedpoint. This can be *expensive*, must measure

Maintenance Overhead is King

- Had a bunch of ideas with changed/strengthened clause queues
- To make complete, it's *really* hard, lots of edge cases
- Randomisation is less performant but much more maintainable
- Maintaining datastructures over a system with complicated ways of changing data (e.g. variable renaming and eqLit) is expensive and bugs will creep in
- Stateless + randomisation is used almost everywhere

Datastructures

- Implicit bin+tri clauses in watchlist and reason
- Occurrence lists are built and thrown away. As is almost every re-computable data
- Class hierarchy: CNF, Propengine, HyperEngine, Searcher, Solver, (SATSolver)
- Helper classes: VarReplacer, DistillerAllWithAll, CompHandler, Gaussian, Prober, Intree, SQL, ..., OccSimplifier
- OccSimplifier: BVA, SubsumeStrenghten, XorFinder
- Common functions: new_var(s), update_vars, mem_used, print_stats

Gauss-Jordan Elimination

- It turns out that having native XOR clauses is painful
- You need to take care about them for *every* simplification
- One needs to reinvent every single 3-letter acronym by Biere, Heule, Jarvisalo for XOR
- Ain't nobody got time for dat
- So inprocessing is bunched together, XORs are recovered at the end, put into matrix
- XOR recovery is the trivial method. I trust inprocessing does the magic

(Re)configuration

- Lots of magic numbers. Manageability is king, all in SolverConf.cpp
- Allows for more through fuzzing (think of cutoffs)
- Obvious idea: dump SQL data, change at predetermined point, run on N orthogonal re-config
- Machine learn SQL-to-config mapping, re-configure during competition
- Demo time!

Parallelism

- Not really important in industry, for academics + hobbyists
- Easy, with locked data structs, yet quite effective
- Launching highly orthogonal solver configs
- Sharing only bin+tri, as orth. config cls don't mix
- Had a MPI system, could run over 50+ machines, passed bin+tri in msgs

Testing – Component Testing

- The most insidious bugs are the ones where there is no SAT→UNSAT issue
- The simplification simply doesn't work all the time, or most of the time
- Only performance test can pick this up, but it can be lost in the noise
- So CMS has 200+ component tests to check subsumption, strengthening, BVE, BVA, etc.
- Big advantage of helper classes is that I can do this
- Demo!

Testing – Fuzzing

- Huge thanks to Armin Biere for this one
- Fuzz test harness about 1kLoC
- Checks: SAT, UNSAT, DRAT, assumptions and conflict for library usage
- Generates: using Beire's and Brummayer's fuzzer + SHA-1 generation by Nossum
- XOR-CNF generator for G-J fuzzing
- CNF concatenator for component fuzzing
- Fuzzes options – Armin told me about this in 2010, paper by Manthey et al. 2016
- Checks for ASAN/MSAN/over-underflow/alignment
- SLOW_DEBUG for more invarant checking during runtime

Testing – Performance

- To be good at SAT Comp, performance-testing is crucial
- Researchers have access to hundreds/thousands of CPUs
- AWS client-server. Small client bids on chunky server(s)
- `./launch_server --cnflist satcomp14 --s3folder new_test --stats --gauss`
- Uses cloud-init, everything is ephemeral, pulled from my GitHub, mails me instructions to download from S3 when done
- Gives me back console output, SQLite data, DRAT stats
- Spent 500+ EUR on this in the past years

Graphical Overview of Solving

- Maybe I could understand what's going on with a GUI
- 1kLOC PHP+JS system based on MySQL dumped data
- Humans are not very good at understanding GBs of data
- Or my visualisation is horrible
- Demo time!

Outline

The Really Though Stuff

- Making sure CMS doesn't hit edge-cases
- Making sure all the algorithms don't have hidden bugs when they skip over things
- Making sure "improvement ideas" don't degrade performance on class(es) of problems
- Navigating SAT Competition where speed on single CNFs that may not be representative is king, versus real-life where library call speed and startup time are kings
- SAT solvers have been tuned to *specific* CNFs in the past years and you need $$$ to do that