CryptoMiniSat — A Rough Guide

Mate Soos

Presentation at SMT/SAT Summer School

14th of June 2011
Story line

1 Vision

2 Architecture

3 Future work
Outline

Vision
  Goals of CryptoMiniSat
  Ideas vs. code
  Expand-contract loop
  The search-conflict duality

Architecture
  Overall architecture
  Implication cache
  Learnt clauses and restarts
  Misc ideas

Future work
Goals of CryptoMiniSat

- SAT solver that excels at cryptography
- General purpose: won SAT Race’10
- Collaborative: GPL, mailing list, regular releases
Ideas vs. code

Ideas first
- Concentrate on stuff that matters: ideas
- Code is there to express them clearly, *not* most efficiently
- The cleaner the code, the easier we can add new ideas

Code second
- An overall 5% speedup of code wins ≈ 1 problem from SAT’09
- A new idea wins ≈ 2 – 3 problems from SAT’09 set
- Speeding up code is useful when there are no other ideas
Expand-contract loop

- **Expand**: Generate new learnt clauses, and clean them when needed
- **Contract**: Apply subs., strength., var-elim, failed lit, etc.
- **Iterate until we solve the problem**: no “pre-processing”
### The search-conflict duality

<table>
<thead>
<tr>
<th>Conflicts</th>
</tr>
</thead>
<tbody>
<tr>
<td>- SAT solvers are glorified resolution engines!</td>
</tr>
<tr>
<td>- Vars: used to compact the proof</td>
</tr>
<tr>
<td>- Conflict clauses: nodes of the conflict tree</td>
</tr>
</tbody>
</table>
# The search-conflict duality

## Conflicts
- SAT solvers are glorified resolution engines!
- Vars: used to compact the proof
- Conflict clauses: nodes of the conflict tree

## Search
- SAT solvers are glorified search engines!
- Vars: store partially computed values
- Conflict clauses: universally applicable search tree compactors (early confl&extra prop)
Example Search Tree

Guess until conflict

Backtrack

First conflict

Start

Solution Found
Outline

Vision
- Goals of CryptoMiniSat
- Ideas vs. code
- Expand-contract loop
- The search-conflict duality

Architecture
- Overall architecture
- Implication cache
- Learnt clauses and restarts
- Misc ideas

Future work
Overall architecture

Classes, communication
- A class for each major element
- Use object composition
- CNF should be the *only* connecting layer

Each class
- Self-contained: internalise complexity
- Build indexes/datastructures/etc., use them, throw them away
- Problem is meant to change (relatively) rapidly, so above makes sense
Get the most out of computed data

### Level 1 propagation
- Failed literal probing
- (inv.) Stalmarck
- Hyper-binary resolution
- (Useless binary clause removal)

### Occurrence lists
- Backward-subsumption
- Strengthening
- Variable elimination
Implication cache

Cache definition

- Enqueue a literal, find what is propagated
- Cache1: Implied by non-learnt binary clauses
- Cache2: Implied by any clause

![Diagram]

Cache1[a]: b, -d
Cache2[a]: b, -d, c, -e
Implication cache usage

- Conflict clause minimisation
- Fast vivification
- Fast, lazy Stalmarck
- Subsumption w/ non-existent non-learnt binary clauses
- Fast literal dependence calculation
- Extended SCC
- (Share data between threads)
Learnt clauses and restarts

Combinations
- Geom. rest. + clause activities: for packed problems
- Glue-based dyn rest. + glues: for large industrial problems
- (Agility-based dyn rest. + glues: for both, but not perfect)

Choose between static and dynamic
- Variable activity stability decides
- Restart 5 times, 100 conflict each, save top 100 vars
- Calculate how many common vars were present in top 100
- Many the same: search tree is static. Otherwise: it’s dynamic
Dynamic vs. Static

**Dynamic**
- Problem will be attacked from many different angles
- Activity of a clause diminishes when search tree is rebuilt
- But its usefulness for this part of tree still stands
- So use glues – they don’t change

**Static**
- The search tree will remain approximately the same
- So the clause should remain active even after restart(s)
- Activity is therefore a good measure, forget glues
Misc ideas

- Randomise *everything*: var-pick, var-polar, failed-lit, etc.
- Burst search: VSISD is nice, but may miss something
- Propagate binary clauses first vs. strengthening with cache
- Unelimination through recursive adding of removed clauses
Outline

Vision
- Goals of CryptoMiniSat
- Ideas vs. code
- Expand-contract loop
- The search-conflict duality

Architecture
- Overall architecture
- Implication cache
- Learnt clauses and restarts
- Misc ideas

Future work
Future work

- CryptoMiniSat is being actively developed
- New multi-threading ideas are experimented on
- Cleaner architecture is a major goal
Thank you for your time

Any questions?