Verifying RUP Proofs of Propositional Unsatisfiability

Allen Van Gelder
Computer Science Department
University of California
Santa Cruz, CA, USA

avg@cs.ucsc.edu
http://www.cse.ucsc.edu/~avg/Papers/
Overview — This talk is about propositional logic only

- Review of Conflict Graphs Used in Sat Solvers
- Practicality of Writing Explicit Resolution Refutations
  - A Pitfall to Avoid
  - TVR and PUP Strategies
- Shorter Alternatives to Explicit Proofs
  - The Complexity Issue — Deterministic Log Space vs. P–Complete
  - Proof “Traces” — Much Shorter than Explicit Proofs
  - Reverse Unit Propagation (RUP) Proof Format
  - Converting RUP to Explicit Resolution
- Practicality of Independently Verifying Explicit Resolution Refutations
  - First Experimental Results — Comparison with Goldberg and Novikov
  - SAT 2007 Competition — Certified Unsat Track: 166 gigabyte proof verified
- Current Limits and Capabilities
**Brief Summary of Results**

Reverse Unit Propagation (%RUP format) described and evaluated.
- **TVR** and **PUP** strategies described and briefly compared.
- **%RUP** format is easiest to implement; can be retrofitted to existing solvers.
- Supported and used in **Certified Unsat Track of SAT 2007 Competition**.

First large-scale verifications in which verifier (**checker3**) and solvers (various) were developed by independent researchers.

First large-scale verifications of explicit resolution proofs (**%RES** format)
- Numerous engineering issues
- **checker3** has verified a **%RES** proof of 163 gigabytes.
- C library function **mmap** was essential for implementation simplicity.
- Should scale to nearly 1 terabyte, then chip’s 40-bit address limit reached.
- Compressed formats up to 1000 time shorter.
Conflict Graph: Basic Data Structure in Many Modern SAT Solvers

- Introduced in current form in GRASP by Marques-Silva and Sakallah [96,99]
- Adopted into Chaff by Moskewicz, Madigan, Zhao, L. Zhang, and Malik [01]
- Variations in zChaff by L. Zhang, Madigan, Moskewicz, and Malik [01]
- Theoretical analysis by Beame, Kautz, and Sabharwal [JAIR 2004]

If $x$ is implied by unit-clause propagation, there is some clause $[x, \neg y_1, \neg y_2, \ldots]$ such that $y_1, y_2, \ldots$ were assumed or implied earlier.

Put arrows from vertex $x$ to vertices $y_1, y_2, \ldots$.

$[x, \neg y_1, \neg y_2, \ldots]$ is called the antecedent of $x$. 
Various Cuts in Conflict Graph Yield Different Conflict Clauses

Decision Clause $[p, q, \neg b]$

1UIP Clause $[p, \neg a, t]$

Conflict Antecedent $[x_1, x_2, x_3, y]$

$\bot$ denotes false.

Dashed lines go to vertices at lower (earlier) “decision levels”.

RelSat Clause $[p, \neg a, \neg b]$

FirstNewCut Clause $[x_1, x_2, x_3]$
From Conflict Graph to Resolution Derivation of Conflict Clause

A Pitfall to Avoid

L. Zhang and Malik pseudocode to generate resolution proof of empty clause:

1. \( cl = \text{final\_conflicting\_clause}; \)
2. \( \text{while} (!\text{is\_empty\_clause}(cl)) \{ \)
3. \( \text{lit} = \text{choose\_literal}(cl); \)
4. \( \text{var} = \text{variable\_of\_literal}(\text{lit}); \)
5. \( \text{ante\_cl} = \text{antecedent}(\text{var}); \)
6. \( cl = \text{resolve}(cl, \text{ante\_cl}); \} \)

*Note:* On line 1, `final_conflicting_clause` is the antecedent of \( \bot \), *not* a conflict clause.

Similar pseudocode for deriving nonempty conflict clause in `cl`. 
DAG Family with Exponential Worst Case for Foregoing Procedure

- Diagram shows $h = 6$ instance. See Van Gelder, SAT 2007 for details.
- Antecedent clauses are shown in brackets; overbars denote negation.
- Grows as $2^h$ for “choose smallest variable number”.
- Grows as $2^h$ for “choose smallest DAG height”.

```
[1]   [3, 1, 2]   [5, 3, 4]   [7, 5, 6]   [9, 7, 8]   [11, 9, 10]
   1   3   5   7   9   11   [11, 12]
  /   /   /   /   /   /   /
 2   4   6   8   10   12
```

```
[2]   [4, 1, 2]   [6, 3, 4]   [8, 5, 6]   [10, 7, 8]   [12, 9, 10]
```

*verify_df* version 2005.11.15 is exponential for reversed variables numbers (thanks to Zhaohui Fu for verifying this experimentally).
Fixed in 2007 version.
What Was the Fix?

Trivial Resolution (TVR) defined by Beame, Kautz, Subharwal
- Linear (must use previous resolvent), Input (must use an input clause)
- No repeated clashing literal
Their Proposition 4: Conflict clause can be derived by a TVR.

Caveat: Appropriate order must be found.

zchaff was already using TVR for all nonempty conflict clauses.

Fix: Do the same to derive the empty clause!

Alternate fix: Pseudo-Unit Propagation (PUP)
Experimental Manifestation of Inefficiency

- Sizes are “mega” literals (“mega” = $2^{20}$).
- “+?” indicates job was killed when size exceeded stated number.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>2004.11.15</th>
<th>with fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>5pipe</td>
<td>153</td>
<td>15</td>
</tr>
<tr>
<td>5pipe_1.ooo</td>
<td>986</td>
<td>92</td>
</tr>
<tr>
<td>5pipe_5.ooo</td>
<td>2320</td>
<td>94</td>
</tr>
<tr>
<td>6pipe</td>
<td>169</td>
<td>97</td>
</tr>
<tr>
<td>6pipe_6.ooo</td>
<td>3072+?</td>
<td>486</td>
</tr>
<tr>
<td>7pipe</td>
<td>358</td>
<td>319</td>
</tr>
<tr>
<td>9vliw_bp_mc</td>
<td>308</td>
<td>58</td>
</tr>
<tr>
<td>barrel7</td>
<td>2754</td>
<td>5</td>
</tr>
<tr>
<td>barrel8</td>
<td>3072+?</td>
<td>73</td>
</tr>
<tr>
<td>barrel9</td>
<td>3072+?</td>
<td>80</td>
</tr>
<tr>
<td>c3540</td>
<td>393</td>
<td>78</td>
</tr>
<tr>
<td>c5315</td>
<td>162</td>
<td>9</td>
</tr>
<tr>
<td>c7552</td>
<td>2650</td>
<td>22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>2004.11.15</th>
<th>with fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>01_SAT_dat.k10</td>
<td>3</td>
<td>0.134</td>
</tr>
<tr>
<td>07_SAT_dat.k30</td>
<td>3</td>
<td>2.582</td>
</tr>
<tr>
<td>07_SAT_dat.k35</td>
<td>3</td>
<td>2.859</td>
</tr>
<tr>
<td>18_SAT_dat.k10</td>
<td>174</td>
<td>1</td>
</tr>
<tr>
<td>18_SAT_dat.k15</td>
<td>102,520+?</td>
<td>14</td>
</tr>
<tr>
<td>1_11_SAT_dat.k10</td>
<td>13</td>
<td>0.338</td>
</tr>
<tr>
<td>1_11_SAT_dat.k15</td>
<td>128</td>
<td>2</td>
</tr>
<tr>
<td>20_SAT_dat.k10</td>
<td>13</td>
<td>0.208</td>
</tr>
<tr>
<td>23_SAT_dat.k10</td>
<td>3</td>
<td>0.062</td>
</tr>
<tr>
<td>23_SAT_dat.k15</td>
<td>15,259+?</td>
<td>4</td>
</tr>
<tr>
<td>26_SAT_dat.k10</td>
<td>0.036</td>
<td>0.036</td>
</tr>
<tr>
<td>2_14_SAT_dat.k10</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>2_14_SAT_dat.k15</td>
<td>3220</td>
<td>7</td>
</tr>
</tbody>
</table>
**TVR and Pseudo-Unit Propagation (PUP) Strategies**

“Pseudo-Unit” because the clause looks like a unit clause to the solver.

1UIP Clause

\[ [p, \neg a, t] \]

**PUP:** may begin \( \text{res}(x_2, [\neg x_2, t], [\neg y, x_2]) \) \quad \text{or} \quad \text{res}(x_2, [\neg x_2, t], [x_1, x_2, x_3, y]) \).

**TVR:** may begin \( \text{res}(y, [x_1, x_2, x_3, y], [\neg y, x_2]) \) but \textit{NOT} \( \text{res}(x_2, [\neg x_2, t], [x_1, x_2, x_3, y]) \).
Shorter Alternatives to Explicit Proofs

- *Resolve–Trace* proof format given by L. Zhang and Malik [DATE 03].
  - Implemented in *zchaff* and *zverify*

- *Conflict Clause Proof* proposed by Goldberg and Novikov [DATE 03].
  - Implemented in *berkmin* but not publicly available

- *Resolution Proof Trace* (%RPT) offered in SAT 2007 Certified Unsat Track
  - *binary* — expanded by *expandtrace0*
Reverse Unit Propagation (RUP) Proof Format

Prover claims clause $C$ is implied by prior clauses $\mathcal{F} \cup \mathcal{D}$.

The RUP requirement: $\mathcal{F} \cup \mathcal{D} \cup \neg(C)$ must derive $\bot$ by unit clause propagation.

“Reverse” because units clauses $\neg(C)$ are propagated back into earlier clauses.

Grasp–Chaff–Berkmin conflict clauses are RUP clauses.
- Stated without proof by Goldberg and Novikov [DATE03];
- Proved for a large class of conflict clauses by Beame et al. [JAIR 2004]

Other reasoning methods may be able to use the RUP format.

Compression achieved of as much as 1000 to 1 vs. explicit proofs ($\%\text{RES}$ format).
Converting RUP to Explicit Resolution

rupToRes expands \%RUP to \%RES format.
- Expansion not unique
- rupToRes is not trusted ! !

Implementation based on zchaffSE (after fixes).
- Disable usual selection of a branching literal.
- Assert all the unit clauses of \( \neg(C) \) at one “decision level.”
- If \( \bot \) is derived, output the (TVR) resolution proof of \( C \).
- Otherwise, proof is buggy!

Often a clause stronger than \( C \) is derived!
The Complexity Issue

*Quis custodiet ipsos custodes?* (Who will guard the guards themselves?)
—Juvenal (first century A.D.)

Ultimately, some verifier must be *trusted* by human inspection.

Proposal: Specify the proof format itself to be in a very low complexity class.

- Easy to independently develop your *own* verifier.
- Reasonable to demand that the verifier be simple before you *trust* it.
- No need to *trust* the solver!

**Deterministic Log Space (DLOG) vs. P–Complete**

A decision problem is **P–Complete** if *every* problem that can be solved in deterministic polynomial time can be reduced to this problem.

A decision problem is in **DLOG** if it can be solved by a program that has a *fixed number* of working storage registers (can re-read the input).

- Each register can hold a number of $\log(L)$ bits for inputs of length $L$.
- No other memory, no malloc, no recursion.
Where do Proof Formats Fall in Terms of Complexity Classes?

Unit clause propagation is $\text{P–Complete}$.

- Requires linear space for known algorithms.
- **Corollary:** RUP format is $\text{P–Complete}$.

Reachability in a directed graph is believed *not* to be solvable in $\text{DLOG}$.

- **Corollary:** Resolution proof trace formats (clauses not materialized, as in \texttt{\%RPT} and \texttt{resolve_trace}) are not in $\text{DLOG}$.

An explicit resolution proof (clauses materialized, operands specified, as in \texttt{\%RES}) *is* in $\text{DLOG}$.
Practicality of Independently Verifying Explicit Resolution Refutations

First Experimental Results — Proof Length Comparisons

<table>
<thead>
<tr>
<th>[GN03] Benchmark</th>
<th>Vars $\times 10^3$</th>
<th>Cls $\times 10^3$</th>
<th>Full %RES lits $\times 10^6$</th>
<th>Full %RES cls $\times 10^6$</th>
<th>Resolve-Trace nbrs $\times 10^6$</th>
<th>Resolve-Trace ratio $\times 10^6$</th>
<th>Resolve-Trace cls $\times 10^6$</th>
<th>RUP lits $\times 10^6$</th>
<th>RUP ratio $\times 10^6$</th>
<th>RUP cls $\times 10^6$</th>
<th>rupToRes lits $\times 10^6$</th>
<th>rupToRes ratio $\times 10^6$</th>
<th>rupToRes cls $\times 10^6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>longmult13</td>
<td>7</td>
<td>20</td>
<td>4005</td>
<td>10.3</td>
<td>55</td>
<td>0.014</td>
<td>0.545</td>
<td>138</td>
<td>0.035</td>
<td>0.545</td>
<td>14140</td>
<td>3.53</td>
<td>30.8</td>
</tr>
<tr>
<td>w10_70</td>
<td>33</td>
<td>104</td>
<td>2582</td>
<td>4.3</td>
<td>6</td>
<td>0.002</td>
<td>0.033</td>
<td>8</td>
<td>0.003</td>
<td>0.033</td>
<td>2924</td>
<td>1.13</td>
<td>4.7</td>
</tr>
<tr>
<td>fifo8-400</td>
<td>260</td>
<td>708</td>
<td>9149</td>
<td>5.3</td>
<td>16</td>
<td>0.002</td>
<td>0.201</td>
<td>87</td>
<td>0.010</td>
<td>0.201</td>
<td>11905</td>
<td>1.30</td>
<td>6.4</td>
</tr>
</tbody>
</table>

- Ratios are to length of Full Resolution (%RES).
- Benchmarks (27 in all, see table in proceedings) from:  

Main findings:
- Resolution proofs about 100 times shorter than predicted by above paper.
- Superlinear growth within families not observed.
## First Experimental Results — Times in CPU seconds

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>[GN03] zchaff Time</th>
<th>zchaffSE Increment</th>
<th>checker3 Time</th>
<th>Res.Trace Increment</th>
<th>zverify Time</th>
<th>rupToRes + checker3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5pipe</td>
<td>11</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>5pipe_1_000</td>
<td>25</td>
<td>12</td>
<td>13</td>
<td>4</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>5pipe_5_000</td>
<td>29</td>
<td>12</td>
<td>12</td>
<td>11</td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>6pipe</td>
<td>80</td>
<td>23</td>
<td>12</td>
<td>31</td>
<td>3</td>
<td>61</td>
</tr>
<tr>
<td>6pipe_6_000</td>
<td>142</td>
<td>58</td>
<td>67</td>
<td>6</td>
<td>4</td>
<td>110</td>
</tr>
<tr>
<td>7pipe</td>
<td>254</td>
<td>54</td>
<td>39</td>
<td>16</td>
<td>5</td>
<td>169</td>
</tr>
<tr>
<td>9vliw_bp_mc</td>
<td>39</td>
<td>14</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>53</td>
</tr>
<tr>
<td>longmult12</td>
<td>1,102</td>
<td>445</td>
<td>57</td>
<td>143</td>
<td>21</td>
<td>3681</td>
</tr>
<tr>
<td>longmult13</td>
<td>1,663</td>
<td>343</td>
<td>89</td>
<td>163</td>
<td>23</td>
<td>4887</td>
</tr>
<tr>
<td>longmult14</td>
<td>974</td>
<td>369</td>
<td>44</td>
<td>153</td>
<td>16</td>
<td>3775</td>
</tr>
<tr>
<td>longmult15</td>
<td>257</td>
<td>72</td>
<td>69</td>
<td>37</td>
<td>5</td>
<td>625</td>
</tr>
<tr>
<td>w10_45</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>w10_60</td>
<td>27</td>
<td>60</td>
<td>73</td>
<td>10</td>
<td>2</td>
<td>88</td>
</tr>
<tr>
<td>w10_70</td>
<td>97</td>
<td>250</td>
<td>391</td>
<td>2</td>
<td>4</td>
<td>394</td>
</tr>
<tr>
<td>fifo8-200</td>
<td>136</td>
<td>27</td>
<td>27</td>
<td>41</td>
<td>3</td>
<td>56</td>
</tr>
<tr>
<td>fifo8-300</td>
<td>370</td>
<td>76</td>
<td>77</td>
<td>66</td>
<td>6</td>
<td>184</td>
</tr>
<tr>
<td>fifo8-400</td>
<td>2,737</td>
<td>1,089</td>
<td>397</td>
<td>75</td>
<td>16</td>
<td>2553</td>
</tr>
</tbody>
</table>

- Easier benchmarks not shown (27 in all, see table in proceedings).
SAT 2007 Competition — Certified Unsat Track

http://www.cse.ucsc.edu/~avg/ProofChecker/cert-poster.pdf

Pigeon-Hole formulas plus 17 industrial benchmarks from SAT 2007 competition

Resources: 1 CPU hour at 2.6 GHz, 8 GB real memory, about 6 GB swap

36 GB (most cases) on a local disk

The Contestants

<table>
<thead>
<tr>
<th>Program</th>
<th>Proof Format</th>
<th>Authors</th>
<th>Program</th>
<th>Proof Format</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>booleforce</td>
<td>%RES</td>
<td>A. Biere</td>
<td>tts</td>
<td>%RPT</td>
<td>I. Spence</td>
</tr>
<tr>
<td>picosat</td>
<td>%RES</td>
<td>A. Biere</td>
<td>zchaffSE07</td>
<td>%RES</td>
<td>Z. Fu, L. Zhang</td>
</tr>
<tr>
<td>picosat</td>
<td>%RUP</td>
<td>A. Biere</td>
<td>zchaff_0</td>
<td>%RES</td>
<td>A. Van Gelder, Z. Fu, L. Zhang</td>
</tr>
</tbody>
</table>

- tts clear winner on pigeon-hole *(asymptotically optimal?)*.
- booleforce edged out zchaff on industrial.
- zchaff had best balance of industrial and pigeon-hole.
- picosat RUP produced most proofs, but we had too little disk to expand many.

**Biggest Verified %RES Proof**: IBM_FV_2004_rb30_Sd.k15

  - booleforce: 163GB
  - picosat RUP: 86GB (from 52MB)
  - zchaff: 36GB.
Conclusions

%RES “language” is recognizable in deterministic log space (DLOG). %RUP format is easiest to implement.

Current Limits and Capabilities

- checker3 has verified a %RES proof of 163 gigabytes.
- C library function mmap was essential for implementation simplicity.
- Should scale to nearly 1 terabyte, then chip’s 40-bit address limit reached.

Summary for GN03 benchmarks (adding up all times):

- Producing resolve_trace and checking with zverify incurs 11% overhead.
- Producing %RES and checking with checker3 incurs 38% overhead.
- Producing %RUP, expanding with rupToRes, and checking with checker3 incurs 200% overhead.
- Wide variations from the average.

Take-Home Message: CPU time is the bottleneck anymore.