Simulating Structural Reasoning on the CNF-Level

Symposium on Structure in Hard Combinatorial Problems
TU Vienna, Austria

Armin Biere

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

based on joined work with

Marijn Heule (UT Austin), Matti Järvisalo (Univ. Helsinki)
Gergely Kovásznai (JKU), Andreas Fröhlich (JKU), Norbert Manthey (TU Dresden)

May 18, 2013
new version June 7, 2013
Overview

- What progress has been made since MiniSAT?
- Preprocessing and Inprocessing
- “Unhiding”
- Quantifier-free bit-vector logic (QF_BV)
  - demo: example of Pete Jeavons with QF_BV and solving with Boolector/Lingeling
  - complexity of decision problem for QF_BV, QF_BV_{bw}, QF_BV_{\ll 1}, QF_BV_{\ll c}, ...
Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

- Limmat (2002)
- Zchaff (2002)
- Berkmin (2002)
- Forklift (2003)
- Siege (2003)
- SatELite (2005)
- Minisat 2 (2006)
- Picosat (2007)
- Rsat (2007)
- Minisat 2.1 (2008)
- Precosat (2009)
- Glucose (2009)
- Clasp (2009)
- Cryptominisat (2010)
- Lingeling (2010)
- Minisat 2.2 (2010)
- Glucose 2 (2011)
- Glueeminisat (2011)
- Contrasat (2011)

[Le Berre'11]
Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

- Limmat (2002)
- Zchaff (2002)
- Berkmin (2002)
- Forklift (2003)
- Siege (2003)
- SatELite (2005)
- Minisat 2 (2006)
- Picosat (2007)
- Rsat (2007)
- Minisat 2.1 (2008)
- Precosat (2009)
- Glucose (2009)
- Clasp (2009)
- Cryptominisat (2010)
- Lingeling (2010)
- Minisat 2.2 (2010)
- Glucose 2 (2011)
- Glueminisat (2011)
- Contrasat (2011)
- Lingeling 587f (2011)
SAT Competition 2012 Application Benchmarks

- MiniSAT 2 (2006)
- MiniSAT 2.2.0 (2010)
- Lingeling aqw (2013)
What is Practical SAT Solving?

- Encoding
- Simplifying
- Inprocessing
- Reencoding

[MantheyHeuleBiere’HVC12]

[JarvisaloHeuleBiere’IJCAR12]

Search
Non-working idea existing in the literature (for shrinking learned clauses):

\[(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor e) \land (a \lor b \lor c \lor f)\]

by

\[(x \lor d) \land (x \lor e) \land (x \lor f) \land (\overline{x} \lor a \lor b \lor c)\]

However, bounded variable elimination [DavisPutnam60][EénBiere05] eliminates \(x\) again.

No gain, so this does not work in practice.
Reverse of Davis & Putnam style variable elimination.

Replace

\[
\begin{align*}
(a \lor d) & \quad (a \lor e) \\
(b \lor d) & \quad (b \lor e) \\
(c \lor d) & \quad (c \lor e)
\end{align*}
\]

by

\[
\begin{align*}
(\bar{x} \lor a) & \quad (\bar{x} \lor b) & \quad (\bar{x} \lor c) \\
(x \lor d) & \quad (x \lor e)
\end{align*}
\]

Paper gives an algorithm to find this kind of patterns fast.
At-Most-One Constraints and BVA

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Clauses</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>DE</td>
<td>$\frac{n \cdot (n-1)}{2}$</td>
<td>$n$</td>
</tr>
<tr>
<td>LE</td>
<td>$n \cdot \lfloor \log n \rfloor$</td>
<td>$n + \log n$</td>
</tr>
<tr>
<td>PE</td>
<td>$2n + 4 \cdot \sqrt{n} + O\left(\frac{4}{\sqrt{n}}\right)$</td>
<td>$n + \sqrt{n} + O\left(\frac{4}{\sqrt{n}}\right)$</td>
</tr>
<tr>
<td>SE</td>
<td>$3n - 4$</td>
<td>$2n - 1$</td>
</tr>
<tr>
<td>DE + BVA</td>
<td>$3n - 6$</td>
<td>$\sim 2n$</td>
</tr>
<tr>
<td>LE + BVA</td>
<td>$\sim 3n$</td>
<td>$\sim 1.5n$</td>
</tr>
</tbody>
</table>

DE  naïve quadratic encoding
PE  [Chen’ModRef11]
LE  [Prestwich’SAT07]
SE  [Sinz’CP05]
Lingeling 587f on SC2009/2011 application instances:

<table>
<thead>
<tr>
<th>Lingeling</th>
<th>2009</th>
<th></th>
<th></th>
<th>2011</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>solved</td>
<td>SAT</td>
<td>UNSAT</td>
<td>time</td>
<td>solved</td>
<td>SAT</td>
</tr>
<tr>
<td>original version 587f</td>
<td>196</td>
<td>79</td>
<td>117</td>
<td>114256</td>
<td>164</td>
<td>78</td>
</tr>
<tr>
<td>only preprocessing</td>
<td>184</td>
<td>72</td>
<td>112</td>
<td>119161</td>
<td>159</td>
<td>77</td>
</tr>
<tr>
<td>no pre- nor inprocessing</td>
<td>170</td>
<td>68</td>
<td>102</td>
<td>138940</td>
<td>156</td>
<td>78</td>
</tr>
</tbody>
</table>
Bounds reached with Blimc Bounded Model Checker:

<table>
<thead>
<tr>
<th></th>
<th>*</th>
<th>bounds</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>with inpr.</td>
<td>158</td>
<td>153975</td>
<td>304158</td>
</tr>
<tr>
<td>no inpr.</td>
<td>115</td>
<td>125436</td>
<td>335393</td>
</tr>
<tr>
<td>non-incr.</td>
<td>67</td>
<td>49915</td>
<td>369104</td>
</tr>
</tbody>
</table>

* = #solved + #(bound 1000 reached)

bounds = ∑ reached bounds

Hardware Model Checking Competition 2011 Instances
Inprocessing Rules

φ irredudant clauses  ρ redundant clauses  σ witness reconstruction stack

ϕ[ρ ∧ C]σ
ϕ ∧ C[ρ]σ

STRENGTHEN

ϕ[ρ]σ
ϕ[ρ ∧ C]σ

LEARN

ϕ[ρ ∧ C]σ
ϕ[ρ]σ

FORGET

ϕ ∧ C[ρ]σ
ϕ[ρ ∧ C]σ, l:C

WEAKEN

L is that φ ∧ ρ and φ ∧ ρ ∧ C are satisfiability-equivalent.

W is that φ and φ ∧ C are satisfiability-equivalent.
Idea: “Resolution Look-Ahead”

Clause $C$ is an *Asymmetric Tautology* (AT) w.r.t. $G$ iff $G \land \neg C$ refuted by BCP.

Given clause $C \in F$, $l \in C$.

Assume all resolvents $R$ of $C$ on $l$ with clauses in $F$ are AT w.r.t. $F \setminus \{C\}$.

Then $C$ is called *Resolution Asymmetric Tautology* (RAT) w.r.t. $F$ on $l$.

In this case $F$ is satisfiability equivalent to $F \setminus \{C\}$.

Inprocessing Rules with RAT simulate all techniques in current SAT solvers
Motivation for Unhiding

- SAT solvers applied to huge formulas
  - million of variables
  - fastest solvers use preprocessing/inprocessing
  - *need cheap and effective inprocessing techniques for millions of variables*

- this talk:
  - **unhiding** redundancy in large formulas
  - almost linear randomized algorithm
  - using the binary implication graph
  - fast enough to be applied to learned clauses

- see our SAT’11 paper for more details
\((\overline{a} \lor c) \land (\overline{a} \lor d) \land (\overline{b} \lor d) \land (\overline{b} \lor e) \land \\
(\overline{c} \lor f) \land (\overline{d} \lor f) \land (\overline{g} \lor f) \land (\overline{f} \lor h) \land \\
(\overline{g} \lor h) \land (\overline{a} \lor \overline{e} \lor h) \land (\overline{b} \lor \overline{c} \lor h) \land (a \lor b \lor c \lor d \lor e \lor f \lor g \lor h)\)
Transitive Reduction TRD

\[(\bar{a} \lor c) \land (\bar{a} \lor d) \land (\bar{b} \lor d) \land (\bar{b} \lor e) \land
(\bar{c} \lor f) \land (\bar{d} \lor f) \land (\bar{g} \lor f) \land (\bar{f} \lor h) \land
(\bar{g} \lor h) \land (\bar{a} \lor \bar{e} \lor h) \land (\bar{b} \lor \bar{c} \lor h) \land (a \lor b \lor c \lor d \lor e \lor f \lor g \lor h)\]

TRD

g \rightarrow f \rightarrow h
Hidden Tautology Elimination HTE

\[ (\bar{a} \lor c) \land (\bar{a} \lor d) \land (\bar{b} \lor d) \land (\bar{b} \lor e) \land \\
(\bar{c} \lor f) \land (\bar{d} \lor f) \land (\bar{g} \lor f) \land (\bar{f} \lor h) \land \\
(\bar{a} \lor \bar{e} \lor h) \land (\bar{b} \lor \bar{c} \lor h) \land (a \lor b \lor c \lor d \lor e \lor f \lor g \lor h) \]

**HTE**

\[ a \rightarrow d \rightarrow f \rightarrow h \]
Hidden Tautology Elimination HTE

\[(\bar{a} \lor c) \land (\bar{a} \lor d) \land (\bar{b} \lor d) \land (\bar{b} \lor e) \land (\bar{c} \lor f) \land (\bar{d} \lor f) \land (\bar{g} \lor f) \land (\bar{f} \lor h) \land (\bar{b} \lor \bar{c} \lor h) \land (a \lor b \lor c \lor d \lor e \lor f \lor g \lor h)

\text{HTE}
\]
\[c \rightarrow f \rightarrow h\]
Self-Subsuming Resolution SSR

\[ \frac{C \lor l}{D} \frac{D \lor \bar{l}}{C \subseteq D} \]

resolvent \( D \) subsumes second antecedent \( D \lor \bar{l} \)

assume given CNF contains both antecedents

if \( D \) is added to CNF then \( D \lor \bar{l} \) can be removed

which in essence removes \( \bar{l} \) from \( D \lor \bar{l} \)

used in SATeLight preprocessor

now common in many SAT solvers
HTE = Reverse SSR + Tautology Elimination

hidden literal addition (HLA) uses SSR in reverse order

\[
\frac{C \lor l \quad D \lor \bar{l}}{D} \quad C \subseteq D
\]

assume given CNF contains resolvent and first antecedent

\[
\frac{a \lor b \lor l \quad a \lor b \lor c \lor \bar{l}}{a \lor b \lor c}
\]

\[
\ldots (a \lor b \lor l)(a \lor b \lor c) \ldots
\]

\[
\ldots (a \lor b \lor l)(a \lor b \lor c \lor \bar{l}) \ldots
\]

we can replace \( D \) by \( D \lor \bar{l} \)

which in essence adds \( \bar{l} \) to \( D \), repeat HLA until fix-point

keep remaining non-tautological clauses after removing added literals again

HTE = assume \( C \lor l \) is a binary clauses

more general versions in the paper

remove clauses with a literal implied by negation of another literal in the clause

HTE confluent and BCP preserving

modulo equivalent variable renaming
Hidden Literal Elimination HLE

better explained on binary implication graph

remove literal from a clause which implies another literal in the clause

\[\ldots (\bar{a} \lor b)(\bar{b} \lor c)(a \lor c \lor d) \ldots \Rightarrow \ldots (\bar{a} \lor b)(\bar{b} \lor c)(c \lor d) \ldots\]

related work before all uses BCP:

- asymmetric branching
  implemented in MiniSAT but switched off by default

- distillation
  [JinSomenzi’05][HanSomenzi DAC’07]

- vivification
  [PietteHamadiSais ECAI’08]

- caching technique in CryptoMiniSAT

HTE/HLE only uses the binary implication graph!
Hidden Literal Elimination HLE

\((\overline{a} \lor c) \land (\overline{a} \lor d) \land (\overline{b} \lor d) \land (\overline{b} \lor e) \land (\overline{c} \lor f) \land (\overline{d} \lor f) \land (\overline{g} \lor f) \land (\overline{f} \lor h) \land (a \lor b \lor c \lor d \lor e \lor \overline{f} \lor \overline{g} \lor \overline{h})\)

HLE

all but \(e\) imply \(h\)
also \(b\) implies \(e\)
\[
\begin{align*}
(\overline{a} \lor c) \land (\overline{a} \lor d) \land (\overline{b} \lor d) \land (\overline{b} \lor e) \land \\
(\overline{c} \lor f) \land (\overline{d} \lor f) \land (\overline{g} \lor f) \land (\overline{f} \lor h) \land \\
( e \lor h)
\end{align*}
\]
\[(\bar{a} \lor c) \land (\bar{a} \lor d) \land (\bar{b} \lor d) \land (\bar{b} \lor e) \land \\
(\bar{c} \lor f) \land (\bar{d} \lor f) \land (\bar{g} \lor f) \land (\bar{f} \lor h) \land \\
(e \lor h)\]
Failed Literal Elimination FL

actually quite old technique … [Freeman PhdThesis’95] [LeBerre’01] …

\[assume \text{ literal } l, \text{ BCP, if conflict, add unit } \bar{l}\]

rather costly to run until completion in practice

one BCP is linear and also in practice can be quite expensive

need to do it for all variables and restart if new binary clause generated

useful in practice: lift common implied literals for assumption \(l\) and assumption \(\bar{l}\)

\textbf{even on BIG (FL2) conjectured to be quadratic} [VanGelder’05]

\[\ldots (\bar{a} \lor b)(\bar{b} \lor c)(\bar{c} \lor d)(\bar{d} \lor \bar{a}) \ldots \Rightarrow \text{ add unit clause } \bar{a}\]

subsumed by running one HLA until completion

Tree-based Look-Ahead saves some time [HeuleJärvisaloBiere’CPAIOR13]
decompose BIG into strongly connect components (SCCs)

if there is an $l$ with $l$ and $\overline{l}$ in the same component $\Rightarrow$ unsatisfiable

otherwise replace all literals by a “representative”

**linear algorithm** can be applied routinely during garbage collection

but as with failed literal preprocessing may generate new binary clauses

$$\ldots (\overline{a} \lor b)(\overline{b} \lor c)(\overline{c} \lor a)(a \lor b \lor c \lor d) \ldots \Rightarrow \ldots (a \lor d) \ldots$$
DFS tree with discovered and finished times: \([\text{dsc}(l), \text{fin}(l)]\)

\[
\begin{array}{c}
\text{dsc}(a) = 29, \text{fin}(a) = 32 \\
\text{dsc}(b) = 11, \text{fin}(b) = 16 \\
\text{dsc}(c) = 30, \text{fin}(c) = 31 \\
\text{dsc}(d) = 14, \text{fin}(d) = 15 \\
\text{dsc}(e) = 12, \text{fin}(e) = 13 \\
\text{dsc}(f) = 2, \text{fin}(f) = 5 \\
\text{dsc}(g) = 1, \text{fin}(g) = 6 \\
\text{dsc}(h) = 3, \text{fin}(h) = 4 \\
\text{dsc}(\bar{h}) = 17, \text{fin}(\bar{h}) = 28 \\
\text{dsc}(\bar{f}) = 20, \text{fin}(\bar{f}) = 27 \\
\text{dsc}(\bar{c}) = 25, \text{fin}(\bar{c}) = 26 \\
\text{dsc}(\bar{d}) = 21, \text{fin}(\bar{d}) = 24 \\
\text{dsc}(\bar{e}) = 7, \text{fin}(\bar{e}) = 10 \\
\text{dsc}(\bar{a}) = 22, \text{fin}(\bar{a}) = 23 \\
\text{dsc}(\bar{b}) = 8, \text{fin}(\bar{b}) = 9
\end{array}
\]

\textbf{tree edges}

Parenthesis theorem: \(l\) ancestor in DFS tree of \(k\) iff \([\text{dsc}(k), \text{fin}(k)] \subseteq [\text{dsc}(l), \text{fin}(l)]\)

well known

Ancestor relationship gives necessary conditions for (transitive) implication:

if \([\text{dsc}(k), \text{fin}(k)] \subseteq [\text{dsc}(l), \text{fin}(l)]\) then \(l \rightarrow k\)

if \([\text{dsc}(\bar{l}), \text{fin}(\bar{l})] \subseteq [\text{dsc}(\bar{k}), \text{fin}(\bar{k})]\) then \(l \rightarrow k\)
• time stamping in previous example does not cover $b \rightarrow h$
  
  $[11, 16] = [\text{dsc}(b), \text{fin}(b)] \not\subseteq [\text{dsc}(h), \text{fin}(h)] = [3, 4]$

  $[17, 28] = [\text{dsc}(\bar{h}), \text{fin}(\bar{h})] \not\subseteq [\text{dsc}(\bar{b}), \text{fin}(\bar{b})] = [8, 9]$

• in example still both HTE “unhidden”, HLE works too (since $b \rightarrow e$)

• “coverage” heavily depends on DFS order

• as solution we propose multiple randomized DFS rounds/phases

• so we approximate a quadratic problem (reachability) randomly by a linear algorithm

• if BIG is a tree one time stamping covers everything
Unhiding through Time Stamping

Unhiding (formula $F$)

1  $\text{stamp} := 0$
2  $\text{foreach} \text{ literal } l \text{ in } \text{BIG}(F) \text{ do}$
3   $\text{dsc}(l) := 0; \text{fin}(l) := 0$
4   $\text{prt}(l) := l; \text{root}(l) := l$
5  $\text{foreach } r \in \text{RTS}(F) \text{ do}$
6     $\text{stamp} := \text{Stamp}(r, \text{stamp})$
7  $\text{foreach} \text{ literal } l \text{ in } \text{BIG}(F) \text{ do}$
8     $\text{if } \text{dsc}(l) = 0 \text{ then}$
9       $\text{stamp} := \text{Stamp}(l, \text{stamp})$
10  $\text{return Simplify}(F)$

Stamp (literal $l$, integer $\text{stamp}$)

1  $\text{stamp} := \text{stamp} + 1$
2  $\text{dsc}(l) := \text{stamp}$
3  $\text{foreach } (\bar{l} \lor l') \in F_2 \text{ do}$
4     $\text{if } \text{dsc}(l') = 0 \text{ then}$
5       $\text{prt}(l') := l$
6       $\text{root}(l') := \text{root}(l)$
7       $\text{stamp} := \text{Stamp}(l', \text{stamp})$
8     $\text{stamp} := \text{stamp} + 1$
9     $\text{fin}(l) := \text{stamp}$
10    $\text{return } \text{stamp}$

Simplify (formula $F$)

1  $\text{foreach } C \in F$
2     $F := F \setminus \{C\}$
3     $\text{if } \text{UHTE}(C) \text{ then continue}$
4  $F := F \cup \{\text{UHLE}(C)\}$
5  $\text{return } F$
**UHTE (clause C)**

1. \( l_{pos} := \) first element in \( S^+(C) \)
2. \( l_{neg} := \) first element in \( S^-(C) \)
3. **while** true
4.   **if** \( \text{dsc}(l_{neg}) > \text{dsc}(l_{pos}) \) **then**
5.     **if** \( l_{pos} \) is last element in \( S^+(C) \) **then** return false
6.     \( l_{pos} := \) next element in \( S^+(C) \)
7.   **else if** \( \text{fin}(l_{neg}) < \text{fin}(l_{pos}) \) **or** \( |C| = 2 \) and \( (l_{pos} = \overline{l_{neg}} \) or \( \text{prt}(l_{pos}) = l_{neg}) \) **then**
8.     **if** \( l_{neg} \) is last element in \( S^-(C) \) **then** return false
9.     \( l_{neg} := \) next element in \( S^-(C) \)
10. **else** return true

\( S^+(C) \) sequence of literals in \( C \) ordered by \( \text{dsc}() \)

\( S^-(C) \) sequence of negations of literals in \( C \) ordered by \( \text{dsc}() \)

\( O(|C| \log |C|) \)
**UHLE** (clause \( C \))

1. \( \text{finished} := \text{finish time of first element in } S_{\text{rev}}^+ (C) \)
2. \( \text{foreach } l \in S_{\text{rev}}^+ (C) \text{ starting at second element} \)
3. \( \text{if } \text{fin}(l) > \text{finished} \text{ then } C := C \setminus \{l\} \)
4. \( \text{else } \text{finished} := \text{fin}(l) \)
5. \( \text{finished} := \text{finish time of first element in } S^- (C) \)
6. \( \text{foreach } \bar{l} \in S^- (C) \text{ starting at second element} \)
7. \( \text{if } \text{fin}(\bar{l}) < \text{finished} \text{ then } C := C \setminus \{l\} \)
8. \( \text{else } \text{finished} := \text{fin}(\bar{l}) \)
9. \( \text{return } C \)

\[ S_{\text{rev}}^+ (C) \text{ reverse of } S^+ (C) \]

\[ O(|C|\log|C|) \]
Stamp (literal \( l \), integer \( \text{stamp} \))

1. BSC \( \text{stamp} := \text{stamp} + 1 \)
2. BSC \( \text{dsc}(l) := \text{stamp}; \ \text{obs}(l) := \text{stamp} \)
3. ELS \( \text{flag} := \text{true} \) // \( l \) represents a SCC
4. ELS S.push(\( l \)) // push \( l \) on SCC stack
5. BSC \textbf{for each} \( (\bar{l} \lor l') \in F_2 \)
6. TRD \textbf{if } \text{dsc}(l) < \text{obs}(l') \textbf{ then } F := F \setminus \{ (\bar{l} \lor l') \}; \textbf{continue}
7. FLE \textbf{if } \text{dsc} \text{root}(l)) \leq \text{obs}(\bar{l}) \textbf{ then}
8. FLE \( l_{\text{failed}} := l \)
9. FLE \textbf{while } \text{dsc}(l_{\text{failed}}) > \text{obs}(\bar{l}) \textbf{ do } l_{\text{failed}} := \text{prt}(l_{\text{failed}})
10. FLE \( F := F \cup \{ (\bar{l}_{\text{failed}}) \} \)
11. FLE \textbf{if } \text{dsc}(\bar{l}) \neq 0 \textbf{ and } \text{fin}(\bar{l}) = 0 \textbf{ then } \textbf{continue}
12. BSC \textbf{if } \text{dsc}(l') = 0 \textbf{ then}
13. BSC \( \text{prt}(l') := l \)
14. BSC \( \text{root}(l') := \text{root}(l) \)
15. BSC \( \text{stamp} := \text{Stamp}(l', \text{stamp}) \)
16. ELS \textbf{if } \text{fin}(l') = 0 \textbf{ and } \text{dsc}(l') < \text{dsc}(l) \textbf{ then}
17. ELS \( \text{dsc}(l) := \text{dsc}(l'); \ \text{flag} := \text{false} \) // \( l \) is equivalent to \( l' \)
18. OBS \( \text{obs}(l') := \text{stamp} \) // set last observed time attribute
19. ELS \textbf{if } \text{flag} = \text{true} \textbf{ then} // if \( l \) represents a SCC
20. BSC \( \text{stamp} := \text{stamp} + 1 \)
21. ELS \textbf{do}
22. ELS \( l' := S.\text{pop()} \) // get equivalent literal
23. ELS \( \text{dsc}(l') := \text{dsc}(l) \) // assign equal discovered time
24. BSC \( \text{fin}(l') := \text{stamp} \) // assign equal finished time
25. ELS \textbf{while } l' \neq l
26. BSC \textbf{return } \text{stamp}
• implemented as one inprocessing phase in our SAT solver Lingeling
  beside variable elimination, distillation, blocked clause elimination, probing, …

• bursts of randomized DFS rounds and sweeping over the whole formula

• fast enough to be applicable to large learned clauses as well
  unhiding is particularly effective for learned clauses

• beside UHTE and UHLE we also have added hyper binary resolution UHBR
  not useful in practice
### Lingeling 571 on SAT’09 Competition Application Benchmarks

#### Similar results for crafted and SAT’10 Race instances

**Table 1**: Configuration Results for SAT’09 Competition Application Benchmarks

<table>
<thead>
<tr>
<th>Configuration</th>
<th>sol</th>
<th>sat</th>
<th>uns</th>
<th>unhd</th>
<th>simp</th>
<th>elim</th>
</tr>
</thead>
<tbody>
<tr>
<td>adv.stamp (no uhbr)</td>
<td>188</td>
<td>78</td>
<td>110</td>
<td>7.1%</td>
<td>33.0%</td>
<td>16.1%</td>
</tr>
<tr>
<td>adv.stamp (w/uhbr)</td>
<td>184</td>
<td>75</td>
<td>109</td>
<td>7.6%</td>
<td>32.8%</td>
<td>15.8%</td>
</tr>
<tr>
<td>basic stamp (no uhbr)</td>
<td>183</td>
<td>73</td>
<td>110</td>
<td>6.8%</td>
<td>32.3%</td>
<td>15.8%</td>
</tr>
<tr>
<td>basic stamp (w/uhbr)</td>
<td>183</td>
<td>73</td>
<td>110</td>
<td>7.4%</td>
<td>32.8%</td>
<td>15.8%</td>
</tr>
<tr>
<td>no unhiding</td>
<td>180</td>
<td>74</td>
<td>106</td>
<td>0.0%</td>
<td>28.6%</td>
<td>17.6%</td>
</tr>
</tbody>
</table>

**Table 2**: Configuration Results for SAT’10 Race Benchmarks

<table>
<thead>
<tr>
<th>Configuration</th>
<th>hte</th>
<th>stamp</th>
<th>redundant</th>
<th>hle</th>
<th>redundant</th>
<th>units</th>
<th>stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>adv.stamp (no uhbr)</td>
<td>22</td>
<td>64%</td>
<td>59%</td>
<td>291</td>
<td>77.6%</td>
<td>935</td>
<td>57%</td>
</tr>
<tr>
<td>adv.stamp (w/uhbr)</td>
<td>26</td>
<td>67%</td>
<td>70%</td>
<td>278</td>
<td>77.9%</td>
<td>941</td>
<td>58%</td>
</tr>
<tr>
<td>basic stamp (no uhbr)</td>
<td>6</td>
<td>0%</td>
<td>52%</td>
<td>290</td>
<td>78.0%</td>
<td>273</td>
<td>0%</td>
</tr>
<tr>
<td>basic stamp (w/uhbr)</td>
<td>7</td>
<td>0%</td>
<td>66%</td>
<td>288</td>
<td>76.7%</td>
<td>308</td>
<td>0%</td>
</tr>
<tr>
<td>no unhiding</td>
<td>0</td>
<td>0%</td>
<td>0%</td>
<td>0</td>
<td>0.0%</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>
Demo: Solving Pete Jeavons CSP Example with Boolector

Boolector:

http://fmv.jku.at/boolector

Generator for example in QF_BV logic of SMTLIB 2 format:

http://fmv.jku.at/biere/talks/pjex.c
Theorem  Deciding QF_BV is NEXPTIME complete.

Proof (A) QF_BV ∈ NEXPTIME: exponential (!) bit-blasting, then call SAT solver.

(set-logic QF_BV)
(declare-fun x () (_ BitVec 1000000))
(declare-fun y () (_ BitVec 1000000))
(assert (distinct (bvadd x y) (bvadd y x)))

(B) hardness: reduce DQBF to QF_BV

DQBF is NEXPTIME complete  [AzharPetersonReif’01]

DQBF is QBF with explicit dependencies:  ∀x, y, z ∃a(x, y), b(y, z)....

Idea: encode variables as bit-vectors of size 2^m, with m number universals (polynomial!)
encode independence from universal variables (using cofactors + binary magic numbers)
Complexity of Sub-classes of Bit-Vector Logics

\[ \text{QF}_\text{BV}_{bw} \] bit-wise operation, equality, and inequality

\[ \text{QF}_\text{BV} \ll 1 \] additionally left-shift by one

\[ \text{QF}_\text{BV} \ll c \] additionally left-shift by arbitrary constant

**Theorem** Deciding \( \text{QF}_\text{BV}_{bw} \) is NP-complete

**Theorem** Deciding \( \text{QF}_\text{BV} \ll 1 \) is PSPACE-complete

**Theorem** Deciding \( \text{QF}_\text{BV} \ll c \) is NEXPTIME-complete

Proofs similar to or based on ideas/results by [Johannsen02][SpielmannKuncak'IJCAR12].

addition, comparison, etc. can be expressed polynomially in \( \text{QF}_\text{BV} \ll 1 \).