Lazy and Eager Approaches to Bit-Vector Solving

Clark Barrett (NYU) with Kshitij Bansal (NYU), Liana Hadarean (NYU), Dejan Jovanović (NYU), Cesare Tinelli (U Iowa)

Stanford University, CS 357, Lecture 17
Outline

1. Introduction
2. Eager Solver
   - Bit-level Simplification
   - Factoring Isomorphic Sub-circuits
   - Other Improvements
   - Experimental Results
3. Lazy Solver
   - Sub-Solvers
   - Decision Heuristic
   - In-processing
   - Lemmas on demand
4. Conclusion
Outline

1 Introduction

2 Eager Solver
   - Bit-level Simplification
   - Factoring Isomorphic Sub-circuits
   - Other Improvements
   - Experimental Results

3 Lazy Solver
   - Sub-Solvers
   - Decision Heuristic
   - In-processing
   - Lemmas on demand

4 Conclusion
Bit-Vector Theory

Bit-precise reasoning is a key component of hardware and software analysis and verification

Need for bit-precise reasoning

- Reason about circuits
- Reason about binary programs
- Reason about bit-wise operations in software

Bit-Precise Reasoning in SMT

- Provided by bit-vector theory
- Often combined with other theories, especially array theory

Bit-Vector Theory

Sorts

\([n]\) for each \(n \geq 0\),

Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constants</td>
<td>(0 :: [1]), (1 :: [1])</td>
</tr>
<tr>
<td>concat</td>
<td>(_ \circ _ :: [m], [n] \rightarrow [m + n]) for all (m, n \geq 0)</td>
</tr>
<tr>
<td>extract</td>
<td>(_[i:j] :: [m] \rightarrow [i - j + 1]) for all (m &gt; i, j \geq 0) with (i - j \geq -1)</td>
</tr>
<tr>
<td>and</td>
<td>(_ &amp; _ :: [n], [n] \rightarrow [n]) for all (n \geq 0)</td>
</tr>
<tr>
<td>or</td>
<td>(_</td>
</tr>
<tr>
<td>exclusive or</td>
<td>(_ \oplus _ :: [n], [n] \rightarrow [n]) for all (n \geq 0)</td>
</tr>
<tr>
<td>not</td>
<td>(_\sim _ :: [n] \rightarrow [n]) for all (n \geq 0)</td>
</tr>
<tr>
<td>plus</td>
<td>(_+ _ :: [n], [n] \rightarrow [n]) for all (n \geq 0)</td>
</tr>
<tr>
<td>times</td>
<td>(_\cdot _ :: [n], [n] \rightarrow [n]) for all (n \geq 0)</td>
</tr>
<tr>
<td>shift left</td>
<td>(_\ll _ :: [n], [n] \rightarrow [n]) for all (n \geq 0)</td>
</tr>
<tr>
<td>shift right</td>
<td>(_\gg _ :: [n], [n] \rightarrow [n]) for all (n \geq 0)</td>
</tr>
<tr>
<td>equal</td>
<td>(_\approx _ :: [n], [n]) for all (n \geq 0)</td>
</tr>
<tr>
<td>less</td>
<td>(_&lt; _ :: [n], [n]) for all (n \geq 0)</td>
</tr>
</tbody>
</table>
Eager vs Lazy solvers

Eager bit-blasting solvers
- Current state-of-the-art
- Benefit from high-level reasoning only via pre-solve rewriting
- Complexity grows with word size
- Requires monolithic approach

Lazy solver
- Can integrate high-level reasoning during search
- Can focus only on the literals in the current search
Outline

1. Introduction
2. Eager Solver
   - Bit-level Simplification
   - Factoring Isomorphic Sub-circuits
   - Other Improvements
   - Experimental Results
3. Lazy Solver
   - Sub-Solvers
   - Decision Heuristic
   - In-processing
   - Lemmas on demand
4. Conclusion
### Better SAT Solving

- Eager solvers rely on back-end SAT solver
- SMT solvers preprocess at the word level but what about at the bit level?
- Idea: try bit-level simplification

### Integration with abc

- Integrated the eager solver **cvcE** with abc AIG package
- Imposes some overhead on easy problems and structured families
- Dramatic improvement on some families

---

[a] <http://www.eecs.berkeley.edu/~alanmi/abc>
Factoring Isomorphic Sub-circuits

Example

\[
\begin{align*}
    x'_{[32]} & \rightarrow * \\
    y'_{[32]} & \rightarrow + \\
    z'_{[32]} & \rightarrow \text{not} \\
    a'_{[32]} & \\
    x_{[32]} & \rightarrow * \\
    y_{[32]} & \rightarrow + \\
    z_{[32]} & \rightarrow \text{not} \\
    a_{[32]} & \\
\end{align*}
\]

\[
\begin{align*}
    a'_{[32]} & \rightarrow \text{or} \\
\end{align*}
\]
Factoring Isomorphic Sub-circuits

Example

\[
\begin{align*}
&x_{[32]} \rightarrow * \\
&y_{[32]} \rightarrow + \\
&z_{[32]} \rightarrow \text{not} \\
&a_{[32]} \rightarrow = \\
\end{align*}
\]

\[
\begin{align*}
&x'_{[32]} \rightarrow * \\
&y'_{[32]} \rightarrow + \\
&z'_{[32]} \rightarrow \text{not} \\
&a'_{[32]} \rightarrow = \\
\end{align*}
\]

\[
\begin{align*}
&= \rightarrow \text{or} \\
\end{align*}
\]
Factoring Isomorphic Sub-circuits

Example
Factoring Isomorphic Sub-circuits

Example

F(x_1, y_1, z_1, a_1) ∨ ...

F(x_n, y_n, z_n, a_n)

or

F(x', y', z', a')
Factoring Isomorphic Sub-circuits

Example

\[ F(s_1, s_2, s_3, s_4) \land \]
\[
\bigvee_{i=0}^{n} (x_i = s_1 \land y_i = s_2 \land z_i = s_3 \land a_i = s_4)
\]
Factoring Isomorphic Sub-circuits

- Identify patterns in disjunctions
- Compute signature (based on De Bruijn indices):
  \[(\square_1 \ast \square_2) + \square_3 = \square_4\]
- Circuits with the same signature factored out
- Arguments skolemized
Factoring Isomorphic Sub-circuits

Results

- Patterns are common in constraint solving and synthesis
- Either no effect or significant improvement on benchmarks
- For some benchmarks, number of bit-blasted clauses reduced to 16% of original size
Other improvements to eager solver

**SAT solver**
- Use a second SAT solver distinct from the one driving the main DPLL(T) search
- Configure solvers independently
- Allows for more aggressive simplifications and less overhead

**More Preprocessing**
- Lift bit-vector terms with bit-with 1 to Booleans:
  - Discover top-level facts that can be used in preprocessing
- Additional rewriting to target specific performance bottlenecks
Comparison: cvcE+aig+fic vs cvcE
Comparison: cvcE+aig+fic vs cvcE

<table>
<thead>
<tr>
<th>set</th>
<th>solved</th>
<th>time (s)</th>
<th>solved</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS3 (11)</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>bench-ab (285)</td>
<td>285</td>
<td>1.7</td>
<td>285</td>
<td>57.5</td>
</tr>
<tr>
<td>brummayerbiere (206)</td>
<td>114</td>
<td>2606.5</td>
<td>138</td>
<td>3732.3</td>
</tr>
<tr>
<td>calypto (23)</td>
<td>8</td>
<td>19.4</td>
<td>10</td>
<td>9.2</td>
</tr>
<tr>
<td>dwp-formulas (332)</td>
<td>332</td>
<td>3.2</td>
<td>332</td>
<td>68.2</td>
</tr>
<tr>
<td>galois (4)</td>
<td>1</td>
<td>0.2</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>gulwani-pldi08 (6)</td>
<td>6</td>
<td>21.2</td>
<td>6</td>
<td>49.1</td>
</tr>
<tr>
<td>mcm (185)</td>
<td>50</td>
<td>4526.4</td>
<td>64</td>
<td>3937.7</td>
</tr>
<tr>
<td>pipe (1)</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>rubik (7)</td>
<td>5</td>
<td>147.8</td>
<td>5</td>
<td>157.9</td>
</tr>
<tr>
<td>sage (189)</td>
<td>188</td>
<td>170.0</td>
<td>188</td>
<td>205.0</td>
</tr>
<tr>
<td>spear (680)</td>
<td>673</td>
<td>24573.9</td>
<td>675</td>
<td>24057.0</td>
</tr>
<tr>
<td>stp (427)</td>
<td>425</td>
<td>188.0</td>
<td>424</td>
<td>170.3</td>
</tr>
<tr>
<td>tacas07 (5)</td>
<td>3</td>
<td>22.3</td>
<td>3</td>
<td>19.3</td>
</tr>
<tr>
<td>uclid (423)</td>
<td>416</td>
<td>2230.6</td>
<td>414</td>
<td>2651.5</td>
</tr>
<tr>
<td>uum (8)</td>
<td>2</td>
<td>16.0</td>
<td>2</td>
<td>33.9</td>
</tr>
<tr>
<td>wienand-cav2008 (18)</td>
<td>14</td>
<td>27.9</td>
<td>14</td>
<td>32.2</td>
</tr>
<tr>
<td></td>
<td>2522</td>
<td>34555.1</td>
<td>2561</td>
<td>35181.7</td>
</tr>
<tr>
<td>set</td>
<td>solved</td>
<td>time (s)</td>
<td>solved</td>
<td>time (s)</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------</td>
<td>----------</td>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td>VS3 (11)</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>bench-ab (285)</td>
<td>285</td>
<td>57.5</td>
<td>285</td>
<td>0.0</td>
</tr>
<tr>
<td>brummayerbie (206)</td>
<td>138</td>
<td>3732.3</td>
<td>113</td>
<td>1718.1</td>
</tr>
<tr>
<td>calypto (23)</td>
<td>10</td>
<td>9.2</td>
<td>9</td>
<td>6.1</td>
</tr>
<tr>
<td>dwp-formulas (332)</td>
<td>332</td>
<td>68.2</td>
<td>332</td>
<td>0.0</td>
</tr>
<tr>
<td>galois (4)</td>
<td>1</td>
<td>0.4</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>gulwani-pldi08 (6)</td>
<td>6</td>
<td>49.1</td>
<td>6</td>
<td>25.5</td>
</tr>
<tr>
<td>mcm (185)</td>
<td>64</td>
<td>3937.7</td>
<td>54</td>
<td>5308.3</td>
</tr>
<tr>
<td>pipe (1)</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>rubik (7)</td>
<td>5</td>
<td>157.9</td>
<td>5</td>
<td>99.5</td>
</tr>
<tr>
<td>sage (189)</td>
<td>188</td>
<td>205.0</td>
<td>189</td>
<td>9.9</td>
</tr>
<tr>
<td>spear (680)</td>
<td>675</td>
<td>24057.0</td>
<td>680</td>
<td>400.5</td>
</tr>
<tr>
<td>stp (427)</td>
<td>424</td>
<td>170.3</td>
<td>425</td>
<td>5.1</td>
</tr>
<tr>
<td>tacas07 (5)</td>
<td>3</td>
<td>19.3</td>
<td>3</td>
<td>1.5</td>
</tr>
<tr>
<td>uclid (423)</td>
<td>414</td>
<td>2651.5</td>
<td>416</td>
<td>58.6</td>
</tr>
<tr>
<td>uum (8)</td>
<td>2</td>
<td>33.9</td>
<td>2</td>
<td>30.4</td>
</tr>
<tr>
<td>wienand-cav2008 (18)</td>
<td>14</td>
<td>32.2</td>
<td>14</td>
<td>68.6</td>
</tr>
<tr>
<td></td>
<td>2561</td>
<td>35181.7</td>
<td>2534</td>
<td>7732.3</td>
</tr>
</tbody>
</table>
## cvcE Performance

<table>
<thead>
<tr>
<th>set</th>
<th>cvcE solved</th>
<th>cvcE time (s)</th>
<th>yices2 solved</th>
<th>yices2 time (s)</th>
<th>stp2 solved</th>
<th>stp2 time (s)</th>
<th>z3 solved</th>
<th>z3 time (s)</th>
<th>boolector solved</th>
<th>boolector time (s)</th>
<th>mathsat solved</th>
<th>mathsat time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS3 (11)</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>1</td>
<td>270.3</td>
<td>3</td>
<td>341.7</td>
<td>2</td>
<td>258.7</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>bench-ab (285)</td>
<td>285</td>
<td>57.5</td>
<td>285</td>
<td>0.0</td>
<td>285</td>
<td>0.2</td>
<td>285</td>
<td>8.5</td>
<td>285</td>
<td>3.0</td>
<td>285</td>
<td>2.5</td>
</tr>
<tr>
<td>brummayerbierie (206)</td>
<td>138</td>
<td>3732.3</td>
<td>113</td>
<td>1718.1</td>
<td>143</td>
<td>3188.5</td>
<td>115</td>
<td>4005.1</td>
<td>155</td>
<td>4060.8</td>
<td>123</td>
<td>3741.9</td>
</tr>
<tr>
<td>calypto (23)</td>
<td>10</td>
<td>9.2</td>
<td>9</td>
<td>6.1</td>
<td>11</td>
<td>3.5</td>
<td>11</td>
<td>50.8</td>
<td>9</td>
<td>45.0</td>
<td>11</td>
<td>56.2</td>
</tr>
<tr>
<td>dwp-formulas (332)</td>
<td>332</td>
<td>68.2</td>
<td>332</td>
<td>0.0</td>
<td>332</td>
<td>0.9</td>
<td>332</td>
<td>10.0</td>
<td>332</td>
<td>0.0</td>
<td>332</td>
<td>4.2</td>
</tr>
<tr>
<td>galois (4)</td>
<td>1</td>
<td>0.4</td>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>0.2</td>
<td>1</td>
<td>0.3</td>
<td>1</td>
<td>0.6</td>
</tr>
<tr>
<td>gulwani-ldol08 (6)</td>
<td>6</td>
<td>49.1</td>
<td>6</td>
<td>25.5</td>
<td>6</td>
<td>26.7</td>
<td>6</td>
<td>31.2</td>
<td>6</td>
<td>42.1</td>
<td>6</td>
<td>56.5</td>
</tr>
<tr>
<td>mcm (185)</td>
<td>64</td>
<td>3937.7</td>
<td>54</td>
<td>5308.3</td>
<td>44</td>
<td>3616.9</td>
<td>55</td>
<td>4302.8</td>
<td>45</td>
<td>3452.2</td>
<td>42</td>
<td>3429.4</td>
</tr>
<tr>
<td>pipe (1)</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>rubik (7)</td>
<td>5</td>
<td>157.9</td>
<td>5</td>
<td>99.5</td>
<td>7</td>
<td>323.4</td>
<td>6</td>
<td>148.2</td>
<td>7</td>
<td>343.5</td>
<td>6</td>
<td>342.8</td>
</tr>
<tr>
<td>sage (189)</td>
<td>188</td>
<td>205.0</td>
<td>189</td>
<td>9.9</td>
<td>189</td>
<td>35.2</td>
<td>189</td>
<td>49.5</td>
<td>189</td>
<td>706.9</td>
<td>189</td>
<td>49.1</td>
</tr>
<tr>
<td>spear (680)</td>
<td>675</td>
<td>24057.0</td>
<td>680</td>
<td>400.5</td>
<td>679</td>
<td>1756.6</td>
<td>675</td>
<td>7546.6</td>
<td>676</td>
<td>5360.9</td>
<td>676</td>
<td>13175.0</td>
</tr>
<tr>
<td>stp (427)</td>
<td>424</td>
<td>170.3</td>
<td>425</td>
<td>5.1</td>
<td>425</td>
<td>41.9</td>
<td>425</td>
<td>58.8</td>
<td>425</td>
<td>22.9</td>
<td>425</td>
<td>47.1</td>
</tr>
<tr>
<td>tacas07 (5)</td>
<td>3</td>
<td>19.3</td>
<td>3</td>
<td>1.5</td>
<td>5</td>
<td>348.4</td>
<td>3</td>
<td>7.2</td>
<td>5</td>
<td>465.6</td>
<td>5</td>
<td>54.9</td>
</tr>
<tr>
<td>uclid (423)</td>
<td>414</td>
<td>2651.5</td>
<td>416</td>
<td>58.6</td>
<td>422</td>
<td>902.0</td>
<td>421</td>
<td>1856.4</td>
<td>422</td>
<td>1368.0</td>
<td>423</td>
<td>1226.6</td>
</tr>
<tr>
<td>uum (8)</td>
<td>2</td>
<td>33.9</td>
<td>2</td>
<td>30.4</td>
<td>2</td>
<td>29.1</td>
<td>2</td>
<td>11.1</td>
<td>2</td>
<td>11.5</td>
<td>2</td>
<td>6.4</td>
</tr>
<tr>
<td>wienand-cav2008 (18)</td>
<td>14</td>
<td>32.2</td>
<td>14</td>
<td>68.6</td>
<td>14</td>
<td>64.6</td>
<td>14</td>
<td>41.4</td>
<td>14</td>
<td>23.3</td>
<td>14</td>
<td>36.6</td>
</tr>
<tr>
<td>TOTAL</td>
<td>2561</td>
<td>35181.7</td>
<td>2534</td>
<td>7732.3</td>
<td>2566</td>
<td>10608.3</td>
<td>2543</td>
<td>18469.6</td>
<td>2575</td>
<td>16164.8</td>
<td>2540</td>
<td>22888.1</td>
</tr>
<tr>
<td>core (672)</td>
<td>132</td>
<td>3208.4</td>
<td>326</td>
<td>5717.9</td>
<td>191</td>
<td>3126.4</td>
<td>672</td>
<td>798.4</td>
<td>656</td>
<td>32176.8</td>
<td>587</td>
<td>21791.1</td>
</tr>
<tr>
<td>lfsr (240)</td>
<td>186</td>
<td>9451.9</td>
<td>181</td>
<td>8394.7</td>
<td>196</td>
<td>8896.3</td>
<td>232</td>
<td>12183.3</td>
<td>213</td>
<td>15939.2</td>
<td>139</td>
<td>7644.1</td>
</tr>
<tr>
<td>simple-processor (64)</td>
<td>33</td>
<td>1566.4</td>
<td>35</td>
<td>824.3</td>
<td>54</td>
<td>1911.1</td>
<td>60</td>
<td>1134.6</td>
<td>60</td>
<td>2377.2</td>
<td>25</td>
<td>1283.3</td>
</tr>
<tr>
<td></td>
<td>351</td>
<td>14226.7</td>
<td>542</td>
<td>14936.9</td>
<td>441</td>
<td>13933.8</td>
<td>964</td>
<td>14116.3</td>
<td>929</td>
<td>50493.3</td>
<td>751</td>
<td>30718.5</td>
</tr>
<tr>
<td>TOTAL</td>
<td>2912</td>
<td>49408.4</td>
<td>3076</td>
<td>22669.2</td>
<td>3007</td>
<td>24542.1</td>
<td>3507</td>
<td>32585.9</td>
<td>3504</td>
<td>66658.1</td>
<td>3291</td>
<td>53006.6</td>
</tr>
</tbody>
</table>
Eager Solver

Challenges

- Competitive on most industrial problems
- Poor performance on some specific families:
  - core: crafted problems with lots of don’t-cares
  - lfsr: linear feedback shift register circuit
  - simple-processor: basic instruction decoding
- Conceptually “simple” properties
- But difficult for SAT solvers: large bitwidth, arithmetic reasoning, use of xor
- These families illustrate weaknesses of the eager approach (hard for all eager solvers except z3 which uses “relevancy” to solve them)
Outline

1. Introduction
2. Eager Solver
   - Bit-level Simplification
   - Factoring Isomorphic Sub-circuits
   - Other Improvements
   - Experimental Results
3. Lazy Solver
   - Sub-Solvers
   - Decision Heuristic
   - In-processing
   - Lemmas on demand
4. Conclusion
Lazy bit-vector solver

Sub-theory solvers
- Core theory
- Inequality theory
- Bit-blasting theory

New techniques
- Leverage word-level structure available *during search*
- Develop heuristics for hard sub-problems:
  - **In-processing**: equational solving and algebraic simplifications per sub-problem
  - **Lemmas on demand**: instantiate lemmas relevant to search context
DPLL(T) Review

- Core
- UF
- Arrays
- Bit-Vectors
- Arithmetic
  - assertions
  - explanations
  - conflicts
  - lemmas
  - propagations
- SAT Solver
  - DPLL
DPLL(T) Review

Theory Solvers
- Decide conjunctions of literals
- Incremental
- Backtrackable
- Conflict Generation
- Theory Propagation
Lazy Bit-vector Architecture

Sub-solvers

- SAT Solver
  - DPLL

- Bit-Vectors
  - Equality+CC
    - complete
      - yes
      - no
  - Inequality
    - complete
      - yes
      - no
  - Bitblaster
    - SAT Solver
      - DPLL
### Core Solver

<table>
<thead>
<tr>
<th>Sub-Solvers</th>
<th>Decision Heuristic</th>
<th>In-processing</th>
<th>Lemmas on demand</th>
</tr>
</thead>
</table>

#### Sub-Solvers

- **Eager Solver**
- **Lazy Solver**

### Core Solver

#### $\Sigma_c$

<table>
<thead>
<tr>
<th>Sub-Solver</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>constants</strong></td>
<td>$0 :: [1], 1 :: [1]$</td>
</tr>
<tr>
<td><strong>equal</strong></td>
<td>$n \approx n :: [n], [n]$ for all $n \geq 0$</td>
</tr>
<tr>
<td><strong>concat</strong></td>
<td>$\circ :: [m], [n] \rightarrow [m + n]$ for all $m, n \geq 0$</td>
</tr>
<tr>
<td><strong>extract</strong></td>
<td>$[i : j] :: [m] \rightarrow [i - j + 1]$ for all $m &gt; i, j \geq 0$ with $i - j \geq -1$</td>
</tr>
</tbody>
</table>
Core Solver

Core solver
- Currently handles only equalities and disequalities
- Ongoing work: extend to concat and extract

Core solver algorithm
1. Until fixed point is reached: propagate all slicings across equations and disequations
2. Split equations along slice points
3. Check if normal forms of two disequalities are in the same equivalence class
Core Solver Example

\[ x \]

\[ a \]

Clark Barrett et al. Lazy and Eager Approaches to Bit-Vector Solving
Core Solver Example

\[
a[7:4] = x
\]
Core Solver Example

\[ a [7 : 4] = x \]
Core Solver Example

\[ a[7:4] = x \]
\[ a[4:1] = x \]
Core Solver Example

\[ a[7:4] = x \]
\[ a[4:1] = x \]
Core Solver Example

\[
a[7:4] = x \\
a[4:1] = x
\]
Core Solver Example

\[ a[7:4] = x \]
\[ a[4:1] = x \]

Clark Barrett et al.
Core Solver Example

\[
\begin{align*}
\text{a} & \quad 7 \quad 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1 \quad 0 \\
\text{x} & \quad \text{3} \quad \text{2} \quad \text{1} \quad \text{0} \\
\text{a} \ [7 : 4] & = x \\
\text{a} \ [4 : 1] & = x
\end{align*}
\]
Core Solver Example

\[
\begin{align*}
    &a[7:4] = x \\
    &a[4:1] = x \\
    &a[7:7] \neq a[1:1]
\end{align*}
\]
Core Solver Example

\[
\begin{align*}
    a[7:4] &= x \\
    a[4:1] &= x \\
    a[7:7] &\neq a[1:1]
\end{align*}
\]
Core Solver Example

\[ a[7:4] = x \]
\[ a[4:1] = x \]
\[ a[7:7] \neq a[1:1] \]
Core Solver Example

\[
\begin{align*}
a[7:4] &= x \\
a[4:1] &= x \\
a[7:7] &\ne a[1:1]
\end{align*}
\]
Core Solver Example

\[ a[7:4] = x \]
\[ a[4:1] = x \]
\[ a[7:7] \neq a[1:1] \]
Core Solver Example

\[
\begin{align*}
a[7 : 4] &= x \\
a[4 : 1] &= x \\
a[7 : 7] &\neq a[1 : 1]
\end{align*}
\]
Core Solver Example

\[
\begin{align*}
a[7:4] &= x \\
a[4:1] &= x \\
a[7:7] &\neq a[1:1]
\end{align*}
\]

Clark Barrett et al. Lazy and Eager Approaches to Bit-Vector Solving 38/70
Inequality Solver

<table>
<thead>
<tr>
<th>$\Sigma_0$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>constants</td>
<td>$0 :: [1], 1 :: [1]$</td>
</tr>
<tr>
<td>equal</td>
<td>$\sim \sim :: [n], [n]$ for all $n \geq 0$</td>
</tr>
<tr>
<td>less</td>
<td>$\sim &lt; :: [n], [n]$ for all $n \geq 0$</td>
</tr>
<tr>
<td>leq</td>
<td>$\sim \lesssim :: [n], [n]$ for all $n \geq 0$</td>
</tr>
</tbody>
</table>

Our solver is complete for constraints including only equalities, disequalities and inequalities.
**Inequality Solver**

**Graph construction**
- Build incremental graph based on constraints
- Edge with weight 1 from $x$ to $y$ if $x < y$
- Edge with weight 0 from $x$ to $y$ if $x \preceq y$

**Model construction**
- Label each root with 0 and each constant with itself
- If unlabeled node, all of whose parents are labeled
- Label with the max of parents plus weight from that parent, and repeat
- If constant node $c$ has parent such that label of parent plus weight from parent is larger than $c$, conflict
Inequality Solver Example

\[ b < c, c < 3, a < c, a < b, 2 < a \]
Inequality Solver Example

\[ b < c, c < 3, a < c, a < b, 2 < a \]
Inequality Solver Example

\[ b < c, \quad c < 3, \quad a < c, \quad a < b, \quad 2 < a \]
Inequality Solver Example

$b < c, c < 3, a < c, a < b, 2 \leq a$
Inequality Solver Example

\[ b < c, \ c < 3, \ a < c, \ a < b, \ 2 \leq a \]
Inequality Solver Example

\[ \begin{align*}
&b < c, \\
&c < 3, \\
&a < c, \\
&a < b, \\
&2 \leq a
\end{align*} \]
Inequality Solver Example

\[ b < c, \ c < 3, \ a < c, \ a < b, \ 2 \leq a \]
Inequality Solver Example

\[ b < c, c < 3, a < c, a < b, 2 \leq a \]
Inequality Solver Example

\[ b < c, c < 3, a < c, a < b, 2 \leq a \]
DPLL(T) Bit-Blasting Solver

**Bit-blasting solver**
- Uses dedicated SAT solver (SAT_{bv}) for bit-vector reasoning
- Uses the *solve with assumptions* SAT solver feature, supported by many SAT solvers

**Incremental SAT**
Given propositional formula $\phi$ and literals $l_1, l_2, \ldots, l_n$ as unit clause assumptions, a call to the SAT_{bv} solver $\text{SolveAssumps}(\phi, l_1 \ldots l_n)$ will decide whether $\phi \land l_1 \land \ldots \land l_n$ holds.
Features of all solvers

- Incremental
- Backtrackable
- Able to produce **conflicts**
- Able to produce **theory propagations**
- Able to produce **explanations** for propagations
Decision Heuristic

Idea

Retain original structure of formula in order to

- Restrict SAT splits to \textit{relevant} literals
- Stop when top formula is \textit{justified} (even if not all literals are assigned)
We wish root to be true, so $a$ and $b$ must be true.

Suppose we set $d$ to true, then:

- $b$ and $d$ are justified
- subtree at $a$ is relevant
- subtree at $e$ (including node $g$) is not relevant
Effect of Decision Heuristic

![Effect of Decision Heuristic Plot](image_url)
Effect of Inequality Solver

![Graph showing the effect of inequality solver](image)
Effect of Inequality Solver on top of Decision Heuristic
Effect of Both
Cactus comparison plot
In-processing

Input: Assertions
while Assertions changed do
    for a ∈ Assertions do
        a ← Simplify(subst(a));
        subst ← subst ∪ Solve(a);
    if false ∈ Assertions then
        return Conflict;
return BvSatSolve(Assertions);
In-processing

Example

\[
\begin{align*}
\text{ite} & \quad \neq \quad 2 \times \text{ite} \\
\times & \quad 2 \\
\text{ite} & \quad \neq \quad 2 \times \text{ite} \\
\times & \quad 1
\end{align*}
\]
Example

\[
\begin{align*}
\text{ite} & \neq 2 \times \text{ite} \\
x_0 = y_0 & \neq x_1 = y_1 \\
x_0 = y_0 & \neq x_1 = y_1 \\
2 & = 2 \times x_1 \\
1 & = 1 \times y_1
\end{align*}
\]
In-processing

Example

Active Assertions

\[ x_0 = y_0 \]
\[ x_1 = y_1 \]
\[ x_0 \times (2 \times x_1) \neq 2 \times (y_0 \times y_1) \]
In-processing

Example

\[
\begin{align*}
&\text{ite} \quad \neq \quad 2 \times \text{ite} \\
&x_0 = y_0 \\
&x_0 \\
&x_1 = y_1 \\
&2 \times x_1 \\
&\text{ite} \\
&2 \\
&\text{ite} \\
&x_0 = y_0 \\
&x_0 \\
&y_0 \\
&x_1 = y_1 \\
&y_1 \\
&\text{ite} \\
&1
\end{align*}
\]
In-processing

Example

```
≠          2 * ite
           x
0
= y
0 * 2
x
0
2
ite
x
1
= y
1
y
1
```

Clark Barrett et al.
Lazy and Eager Approaches to Bit-Vector Solving 60 / 70
In-processing

Example

Active Assertions

\[ x_0 = y_0 \]
\[ x_1 \neq y_1 \]
\[ x_0 \times 2 \neq 2 \times (y_0 \times 1) \]
In-processing

Example

- expensive operators
- trivially false through all ite paths.
Motivating example

\[
\begin{align*}
    r_{[32]} &= a_{[32]} \text{ bvurem } b_{[32]} \\
    r_{[32]} &\geq b_{[32]} \\
    b_{[32]} &\neq 0_{[32]}
    \end{align*}
\]

- none of the solvers we tried can solve it
- recognize pattern and instantiate lemmas relevant in search context

This example occurred as a sub-problem in several Spear benchmarks.
Results
## Lazy vs Eager

<table>
<thead>
<tr>
<th>set</th>
<th>cvcE solved</th>
<th>cvcLz solved</th>
<th>mathsatL solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS3 (11)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bench-ab (285)</td>
<td>285</td>
<td>285</td>
<td>285</td>
</tr>
<tr>
<td>brummayerbiere* (206)</td>
<td>138</td>
<td>112</td>
<td>100</td>
</tr>
<tr>
<td>core (672)</td>
<td>132</td>
<td>672</td>
<td>509</td>
</tr>
<tr>
<td>lfsr (240)</td>
<td>186</td>
<td>240</td>
<td>177</td>
</tr>
<tr>
<td>simple-processor (64)</td>
<td>33</td>
<td>64</td>
<td>18</td>
</tr>
<tr>
<td>calypto (23)</td>
<td>10</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>dwp-formulas (332)</td>
<td>332</td>
<td>332</td>
<td>332</td>
</tr>
<tr>
<td>galois (4)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>gulwani-pldi08 (6)</td>
<td>6</td>
<td>13</td>
<td>18</td>
</tr>
<tr>
<td>mcm (185)</td>
<td>64</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>rubik (7)</td>
<td>5</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>sage (189)</td>
<td>188</td>
<td>188</td>
<td>189</td>
</tr>
<tr>
<td>spear (680)</td>
<td>675</td>
<td>648</td>
<td>478</td>
</tr>
<tr>
<td>stp* (427)</td>
<td>424</td>
<td>424</td>
<td>425</td>
</tr>
<tr>
<td>tacs07 (5)</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>uclid* (423)</td>
<td>414</td>
<td>420</td>
<td>420</td>
</tr>
<tr>
<td>uum (8)</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>wienand-cav2008 (18)</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>unique-solve</td>
<td>4</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>set solved</th>
<th>cvcE time (s)</th>
<th>cvcLz time (s)</th>
<th>mathsatL time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS3 (11)</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>bench-ab (285)</td>
<td>285</td>
<td>57.5</td>
<td>2.4</td>
<td>285</td>
</tr>
<tr>
<td>brummayerbiere* (206)</td>
<td>138</td>
<td>3732.3</td>
<td>112</td>
<td>100</td>
</tr>
<tr>
<td>core (672)</td>
<td>132</td>
<td>3208.4</td>
<td>672</td>
<td>596.4</td>
</tr>
<tr>
<td>lfsr (240)</td>
<td>186</td>
<td>9451.9</td>
<td>240</td>
<td>2286.3</td>
</tr>
<tr>
<td>simple-processor (64)</td>
<td>33</td>
<td>1566.4</td>
<td>64</td>
<td>48.7</td>
</tr>
<tr>
<td>calypto (23)</td>
<td>10</td>
<td>9.2</td>
<td>15</td>
<td>100.7</td>
</tr>
<tr>
<td>dwp-formulas (332)</td>
<td>332</td>
<td>68.2</td>
<td>332</td>
<td>5.5</td>
</tr>
<tr>
<td>galois (4)</td>
<td>1</td>
<td>0.4</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>gulwani-pldi08 (6)</td>
<td>6</td>
<td>49.1</td>
<td>6</td>
<td>63.9</td>
</tr>
<tr>
<td>mcm (185)</td>
<td>64</td>
<td>3937.7</td>
<td>13</td>
<td>392.9</td>
</tr>
<tr>
<td>rubik (7)</td>
<td>5</td>
<td>157.9</td>
<td>2</td>
<td>110.6</td>
</tr>
<tr>
<td>sage (189)</td>
<td>188</td>
<td>205.0</td>
<td>188</td>
<td>174.9</td>
</tr>
<tr>
<td>spear (680)</td>
<td>675</td>
<td>24057.0</td>
<td>648</td>
<td>9347.0</td>
</tr>
<tr>
<td>stp* (427)</td>
<td>424</td>
<td>170.3</td>
<td>424</td>
<td>108.6</td>
</tr>
<tr>
<td>tacs07 (5)</td>
<td>3</td>
<td>19.3</td>
<td>5</td>
<td>294.4</td>
</tr>
<tr>
<td>uclid* (423)</td>
<td>414</td>
<td>2651.5</td>
<td>420</td>
<td>3148.9</td>
</tr>
<tr>
<td>uum (8)</td>
<td>2</td>
<td>33.9</td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>wienand-cav2008 (18)</td>
<td>14</td>
<td>32.2</td>
<td>14</td>
<td>34.7</td>
</tr>
<tr>
<td>unique-solve</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Lazy vs Eager
Discussion: lazy vs eager

Lazy
- \textbf{cvcLz} offers more ways of attacking a hard problem
- Break it down into many small sub-problems
- Ignore irrelevant parts of the problem
- Use sub-problem-specific solvers and simplifications

Eager
- When \textbf{cvcE} works well, it is very hard to beat
- When lazy techniques fail, fall-back is essentially eager plus overhead
- Sometimes it’s much worse (when bit-vector SAT solver and DPLL SAT solver have to communicate a lot)
Portfolio Approach

**Idea**

Use a *portfolio approach*: run lazy and eager in parallel!

**cvcPll**

- Two approaches are complementary on hard problems
- Portfolio approach: combine the two solvers in cvcPll
  - Take advantage of multi-core architectures
  - Run multiple threads
  - Stop when first thread returns with an answer
## Results

<table>
<thead>
<tr>
<th>set</th>
<th>cvcPll</th>
<th>yices2</th>
<th>stp2</th>
<th>z3</th>
<th>boolector</th>
<th>mathsat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>solved</td>
<td>time (s)</td>
<td>solved</td>
<td>time (s)</td>
<td>solved</td>
<td>time (s)</td>
</tr>
<tr>
<td>VS3 (11)</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>1</td>
<td>270.3</td>
</tr>
<tr>
<td>bench-ab (285)</td>
<td>285</td>
<td>39.1</td>
<td>285</td>
<td>0.0</td>
<td>285</td>
<td>0.2</td>
</tr>
<tr>
<td>brummayerbierie (206)</td>
<td>137</td>
<td>3024.0</td>
<td>113</td>
<td>1718.1</td>
<td>143</td>
<td>3188.5</td>
</tr>
<tr>
<td>core (672)</td>
<td>672</td>
<td>726.6</td>
<td>326</td>
<td>5717.9</td>
<td>191</td>
<td>3126.4</td>
</tr>
<tr>
<td>lfsr (240)</td>
<td>240</td>
<td>2481.3</td>
<td>181</td>
<td>8394.7</td>
<td>196</td>
<td>8896.3</td>
</tr>
<tr>
<td>simple-processor (64)</td>
<td>64</td>
<td>57.8</td>
<td>35</td>
<td>824.3</td>
<td>54</td>
<td>1911.1</td>
</tr>
<tr>
<td>calypto (23)</td>
<td>15</td>
<td>349.1</td>
<td>9</td>
<td>6.1</td>
<td>11</td>
<td>3.5</td>
</tr>
<tr>
<td>dwp-formulas (332)</td>
<td>332</td>
<td>47.4</td>
<td>332</td>
<td>0.0</td>
<td>332</td>
<td>0.9</td>
</tr>
<tr>
<td>galois (4)</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>gulwani-pldi08 (6)</td>
<td>6</td>
<td>44.8</td>
<td>6</td>
<td>25.5</td>
<td>6</td>
<td>26.7</td>
</tr>
<tr>
<td>mcm (185)</td>
<td>63</td>
<td>6152.2</td>
<td>54</td>
<td>5308.3</td>
<td>44</td>
<td>3616.9</td>
</tr>
<tr>
<td>pipe (1)</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>rubik (7)</td>
<td>5</td>
<td>142.7</td>
<td>5</td>
<td>99.5</td>
<td>7</td>
<td>323.4</td>
</tr>
<tr>
<td>sage (189)</td>
<td>188</td>
<td>215.5</td>
<td>189</td>
<td>9.9</td>
<td>189</td>
<td>35.2</td>
</tr>
<tr>
<td>spear (680)</td>
<td>677</td>
<td>11028.4</td>
<td>680</td>
<td>400.5</td>
<td>679</td>
<td>1756.6</td>
</tr>
<tr>
<td>stp (427)</td>
<td>424</td>
<td>168.0</td>
<td>425</td>
<td>5.1</td>
<td>425</td>
<td>41.9</td>
</tr>
<tr>
<td>tasca07 (5)</td>
<td>5</td>
<td>249.8</td>
<td>3</td>
<td>1.5</td>
<td>5</td>
<td>348.4</td>
</tr>
<tr>
<td>uclid (423)</td>
<td>419</td>
<td>3315.6</td>
<td>416</td>
<td>58.6</td>
<td>422</td>
<td>902.0</td>
</tr>
<tr>
<td>uum (8)</td>
<td>2</td>
<td>605.9</td>
<td>2</td>
<td>30.4</td>
<td>2</td>
<td>29.1</td>
</tr>
<tr>
<td>wienand-cav2008 (18)</td>
<td>14</td>
<td>30.8</td>
<td>14</td>
<td>68.6</td>
<td>14</td>
<td>64.6</td>
</tr>
<tr>
<td></td>
<td>3549</td>
<td>28679.7</td>
<td>3076</td>
<td>22669.3</td>
<td>3007</td>
<td>24542.1</td>
</tr>
</tbody>
</table>
Outline

1 Introduction

2 Eager Solver
   - Bit-level Simplification
   - Factoring Isomorphic Sub-circuits
   - Other Improvements
   - Experimental Results

3 Lazy Solver
   - Sub-Solvers
   - Decision Heuristic
   - In-processing
   - Lemmas on demand

4 Conclusion
Summary

**Eager Solvers**
- Current state-of-the-art
- When they work, they work very well
- Still lots of room for improvement

**Lazy Solvers**
- Decompose problem into many smaller problems
- Imposes some overhead
- Provides many new ways to attack hard problems
Summary

Results

- Eager and lazy are surprisingly complementary
- Portfolio results in dramatic increase in solved problems \textit{and} dramatic reduction in solving time
- Can solve a number of benchmarks no other solver can solve (SOTA solver)

What’s next?

- Still a lot of room to improve both solvers
- More sophisticated in-processing
- Additional sub-theory solvers