Restoring Circuit Structure From SAT Instances

Jarrod A. Roy, Igor L. Markov, and Valeria Bertacco
University of Michigan at Ann Arbor
Outline

- Motivation
- Preliminaries
- A Generic Circuit Detection Algorithm
- AND-OR-NOT Circuit Conversion
- Additional Gate Types
- Empirical Results
- Conclusions and Further Work
Motivation

- SAT solvers have become a staple tool in EDA flows due to recent breakthroughs
- Circuit techniques improve performance on fully circuit derived instances
  - order of magnitude speedup
  - require circuit structure \textit{a priori}
- Literature assumes “structure lost” in CNF
  - We find assumption \textit{not necessarily} true
Motivation

- If we are converting to CNF, don’t we already know the circuit structure?
  - True, no need waste time in this case
- On the other hand, we observe non-trivial structure from several sources
  - property checking: part circuit, part constraints
  - mathematical constructions: DIMACS Pret “encoded 2-colouring forced to be UNSAT”
- Automatically detecting structure and benefiting from it makes solvers more applicable for EDA
Converting Circuits to SAT

- All logic gates have a characteristic function
  - defines compatible assignments of inputs and outputs
- Converting a Circuit to CNF-SAT instance requires one variable per wire and several clauses per gate
- The conversion of gates to clauses is the encoding of each gate’s characteristic function in CNF
  - we call it the CNF-signature of the gate

\[
z = NAND(x_1, \ldots, x_j) \equiv \left[ \prod_{i=1}^{j} (x_i + z) \right] \left( \sum_{i=1}^{j} \overline{x_i} + \overline{z} \right)
\]

\[
z = NOR(x_1, \ldots, x_j) \equiv \left[ \prod_{i=1}^{j} (\overline{x_i} + \overline{z}) \right] \left( \sum_{i=1}^{j} x_i + z \right)
\]
A Generic Circuit Detection Algorithm

- Convert the CNF instance to an undirected graph
- Convert the CNF-signature of the gate to match to an undirected graph
- Use subgraph isomorphism to match instances of the gate

Conversions of the clauses

\[(b+d+c)(c+a+b')(a+c')(d+a')\]
A Generic Circuit Detection Algorithm

- To piece together the circuit, create a maximal independent set (MIS) instance
  - one node per detected gate
  - an edge between nodes if the gates are incompatible (signatures overlap, etc.)

\[(a'+b)(a'+c)(a'+d)(b'+a)(b'+c)\]
\[(a+b'+c')(a+b'+d')(b+a'+c')\]

Encodes (1) \(a=\text{AND}(b,c)\), (2) \(a=\text{AND}(b,d)\), and (3) \(b=\text{AND}(a,c)\)

Only (2) and (3) are compatible.
AND-OR-NOT Circuit Conversion

- Generic alg requires solving NP-hard problems
- Is there a more efficient way, possibly for a slightly more restricted problem?
- Yes: We prove the mapping from AND-OR-NOT circuits to CNF unique, no incompatible gate matches
  - Proof examines each clause of the CNF and shows it must have come from a specific gate
  - Proof suggests efficient linear time algorithm
    - based on pattern-matching of clauses
# Easily Detectable Gate Types

<table>
<thead>
<tr>
<th>Gate type</th>
<th>Difficulty of restoring circuit structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR and AND</td>
<td>Straightforward pattern-matching</td>
</tr>
<tr>
<td>NOR and NAND</td>
<td>Pattern-matching with back-tracking</td>
</tr>
<tr>
<td>NOT, XOR and XNOR</td>
<td>Can be detected by straightforward pattern-matching, but w/o orientation, which can only be determined in the context of other gate types</td>
</tr>
<tr>
<td>MAJ3</td>
<td>More advanced pattern matching with back-tracking</td>
</tr>
</tbody>
</table>

**Table 1:** The relative difficulty of detecting particular types of logic gates in CNF-SAT formulas. Note that this is not an exhaustive listing of detectable gates.
Spotlight on XOR/XNOR

- XOR/XNOR gates are inherently unoriented
  - CNF-signatures are symmetric
  - the 2-input XOR gate
    \[ a = \text{XOR}(b,c) \text{ has CNF signature } (a'+b+c) (a+b'+c) (a+b+c') (a'+b'+c') \]

- Their detection is not all that difficult, but orientation requires proper context
  - With proper context, orientation can be propagated in a BFS like fashion
Spotlight on XOR/XNOR

- What happens without context?
  - Multiple valid interpretations; happens with a chain of XORS

- Could be
- or
Empirical Results

- Implemented detection of AND, OR, NAND, NOR, XOR, XNOR, NOT and MAJ3 gates
- Tested for the presence of structure in DIMACS, SAT2002 and Velev benchmarks
- Results show:
  - Much structure detected
    - sometimes in unexpected places
    - preserved by simplification
  - Technique is fast and scales well
    - small fraction of solving runtime
### Structure in Standard Benchmarks

<table>
<thead>
<tr>
<th>Benchmark series</th>
<th>% variables in simple gates</th>
<th>% clauses in simple gates</th>
<th>% variables in XOR/XNORs</th>
<th>% clauses in XOR/XNORs</th>
<th>Detection runtime (s)</th>
<th># of benchmarks</th>
<th># of variables</th>
<th># of clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bf</td>
<td>54.29%</td>
<td>22.12%</td>
<td>1.18%</td>
<td>0.54%</td>
<td>0.43</td>
<td>4</td>
<td>5793</td>
<td>16566</td>
</tr>
<tr>
<td>Dubois</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td>0.09</td>
<td>13</td>
<td>1275</td>
<td>3400</td>
</tr>
<tr>
<td>Hanoi</td>
<td>43.22%</td>
<td>10.19%</td>
<td>0%</td>
<td>0%</td>
<td>0.37</td>
<td>2</td>
<td>2041</td>
<td>12272</td>
</tr>
<tr>
<td>Parity</td>
<td>33.17%</td>
<td>13.58%</td>
<td>88.35%</td>
<td>68.42%</td>
<td>2.68</td>
<td>30</td>
<td>24267</td>
<td>83330</td>
</tr>
<tr>
<td>Pret</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td>0.09</td>
<td>8</td>
<td>840</td>
<td>2240</td>
</tr>
<tr>
<td>Ssa</td>
<td>47.25%</td>
<td>18.57%</td>
<td>1.45%</td>
<td>0.69%</td>
<td>1.09</td>
<td>8</td>
<td>7228</td>
<td>17669</td>
</tr>
<tr>
<td>XOR-Chain</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>99.55%</td>
<td>0.38</td>
<td>27</td>
<td>4554</td>
<td>12126</td>
</tr>
</tbody>
</table>

Before SAT Preprocessor Hypre

<table>
<thead>
<tr>
<th>Benchmark series</th>
<th>% variables in simple gates</th>
<th>% clauses in simple gates</th>
<th>% variables in XOR/XNORs</th>
<th>% clauses in XOR/XNORs</th>
<th>Detection runtime (s)</th>
<th>Hypre runtime (s)</th>
<th>% variables remaining</th>
<th>% clauses remaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bf</td>
<td>75.31%</td>
<td>46.41%</td>
<td>0.33%</td>
<td>0.12%</td>
<td>0.23</td>
<td>0.49</td>
<td>22.31%</td>
<td>31.17%</td>
</tr>
<tr>
<td>Dubois</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td>0.09</td>
<td>0.05</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Hanoi</td>
<td>52.24%</td>
<td>12.56%</td>
<td>0%</td>
<td>0%</td>
<td>0.24</td>
<td>0.23</td>
<td>64.97%</td>
<td>66.35%</td>
</tr>
<tr>
<td>Parity</td>
<td>30.70%</td>
<td>17.09%</td>
<td>100%</td>
<td>83.24%</td>
<td>2.13</td>
<td>1.47</td>
<td>54.99%</td>
<td>67.01%</td>
</tr>
<tr>
<td>Pret</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td>0.09</td>
<td>0.03</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Ssa</td>
<td>58.21%</td>
<td>29.86%</td>
<td>9.25%</td>
<td>4.74%</td>
<td>0.17</td>
<td>0.34</td>
<td>8.91%</td>
<td>8.33%</td>
</tr>
<tr>
<td>XOR-Chain</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td>0.38</td>
<td>0.17</td>
<td>99.41%</td>
<td>99.55%</td>
</tr>
</tbody>
</table>
Scalability Results

Figure 4: Runtime vs. SAT instance size on Velev’s B25 and B27 series [18] of benchmarks.
## Comparison with Solving Runtime

Circuit Based Technique Runtime = Simulation + (Implicit or Explicit Learning)

<table>
<thead>
<tr>
<th>Circuit</th>
<th>ZChaff</th>
<th>Implicit</th>
<th>Explicit</th>
<th>Simulation</th>
<th>Extraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>9Vliw001</td>
<td>1057</td>
<td>567</td>
<td>793</td>
<td>93</td>
<td>10</td>
</tr>
<tr>
<td>9Vliw004</td>
<td>953</td>
<td>804</td>
<td>1011</td>
<td>92</td>
<td>10</td>
</tr>
<tr>
<td>9Vliw005</td>
<td>3126</td>
<td>740</td>
<td>1314</td>
<td>88</td>
<td>10</td>
</tr>
<tr>
<td>9Vliw007</td>
<td>140</td>
<td>286</td>
<td>855</td>
<td>110</td>
<td>11</td>
</tr>
<tr>
<td>9Vliw008</td>
<td>1450</td>
<td>239</td>
<td>1914</td>
<td>114</td>
<td>12</td>
</tr>
<tr>
<td>9Vliw009</td>
<td>1006</td>
<td>784</td>
<td>829</td>
<td>117</td>
<td>10</td>
</tr>
<tr>
<td>9Vliw010</td>
<td>867</td>
<td>329</td>
<td>1897</td>
<td>96</td>
<td>10</td>
</tr>
<tr>
<td>9Vliw015</td>
<td>2209</td>
<td>985</td>
<td>1270</td>
<td>97</td>
<td>10</td>
</tr>
<tr>
<td>9Vliw017</td>
<td>1007</td>
<td>175</td>
<td>913</td>
<td>109</td>
<td>10</td>
</tr>
<tr>
<td>9Vliw019</td>
<td>2936</td>
<td>849</td>
<td>1448</td>
<td>129</td>
<td>10</td>
</tr>
<tr>
<td>9Vliw021</td>
<td>1666</td>
<td>1069</td>
<td>1345</td>
<td>99</td>
<td>10</td>
</tr>
<tr>
<td>9Vliw024</td>
<td>1375</td>
<td>965</td>
<td>1282</td>
<td>107</td>
<td>9</td>
</tr>
</tbody>
</table>
Conclusions

- Much circuit structure can be extracted efficiently
  - orientation can be difficult
- Tests show structure pops up in many places, possibly unbeknownst to the user
- Circuit-based SAT techniques vastly improve solving when given this structure
- Logical next step: extend general SAT solvers to make use of this structure
Further Work

- How difficult is it to detect other gate types such as AOI/OAI, ITE, etc.?
  - Recent work shows AOI/OAI as difficult as MAJ3

- Examine other methods for orienting inherently unoriented gate types
  - Guess and propagate, …. 

- Is the original orientation of the circuit necessary, or will any valid orientation do?
  - If so, is the original orientation just better than other valid ones?