QuteSat

A Robust Circuit-Based SAT Solver for Complex Circuit Structure

Chung-Yang (Ric) Huang
National Taiwan University

To appear: DATE 2007
02/10/2007
Fact Sheet (Background)

- **Boolean Satisfiability (SAT)**
  - Given a Boolean network
    
    \[ F: B^n \rightarrow B, \]
    
    find an input assignment
    
    \[ A: \{ x_1 = a_1, x_2 = a_2, \ldots, x_n = a_n \mid a_i \in B \} \]
    
    such that \( F = 1. \)
  - First proven NP complete problem (Cook 1971)

- Well researched problems in AI, OR, EDA...
- Widely used in many EDA areas
  - Verification, Testing, Logic optimization, Physical implementation, etc
Yet Another SAT engine?

- Yaya, but with two missions...
  1. Generalized for general constraint satisfiability and optimization problems
  2. Specialized for EDA specific applications
Large vs. Little Engines

- Much of the focus in automated deduction has been on finding **uniform procedures** for large classes of theorems
  - For example, the **resolution method** is a simple, sound and complete inference procedure for first-order logic
  - Resolution-based methods have had significant successes solving open problems in diverse branches of mathematics

- However, big engines are not always predictable enough for serious applications --- they do a poor job of "exploiting domain knowledge"

- **The little engines** ideology is based on composing small theory-specific engines.

“Little Engines of Proof”, Lec Notes, Dr. Shankar, SRI et. al.
Examples of Little Engines

- Propositional satisability solvers
- Binary Decision Diagrams
- Congruence Closure for Equality Propagation
- Real and Integer linear arithmetic solvers
- Decision procedures for lists, arrays, bit-vectors.
- Presburger arithmetic
- Monadic Second Order Logic

“Little Engines of Proof”, Lec Notes, Dr. Shankar, SRI et. al.
Large vs. Little Engines

- **Little Engines**: Tuned for specific problems
- **Large Engines**: Flexibility for new algorithms

Sharing between engines

QuteSat target

Uniformity of data structure and interface
What to expect in this talk...

- Background: Fact sheet, large vs. little engines
- Review: CNF-based SAT
  - 2-literal watch for Boolean Constraint Propagation (BCP)
  - Antecedent clause for efficient conflict-driven learning
- Proposed: QuteSat (Circuit SAT)
  - Generic watch scheme
  - Implicit implication graph
- Experimental results
- Future work
Most modern Boolean SAT solvers are in CNF

- Product of sum (PoS) format
  \[(a+b+c)(a'+b'+c)(a+b+c')(a+b'+c')\]

1. Efficient Boolean Constraint Propagation (BCP)
2. Conflict-driven learning with non-chronological backtracking
3. Decision variable heuristics
4. Clause database reduction

The simpler, the better.
From CNF to Circuit SAT

- However, most of the EDA problems are not naturally in CNF format
  - Operators ($\land \lor \neg \oplus \pm \times \geq \ldots$) $\Rightarrow \land (\neg \lor)$
    - Loss of implicability
  - Structural info (direction, locality,\ldots) $\Rightarrow$ N/A
- Can we implement SAT on circuit structure? (This is not new $\Rightarrow$ e.g. ATPG)
  - Pros: flexibility, structural info
  - Cons: complex implementation, slower BCP
BCP in CNF SAT

- **Boolean implication**
  - e.g. \((a_1 = 1) \land (a_2 = 0) \land \ldots (a_{n-1} = 1) \implies (a_n = 1)\)
  
  \[
  \left(\overline{a_1} + a_2 + \ldots + \overline{a_{n-1}} + a_n\right)
  \]
  
  value: \(x \rightarrow 0 \implies x \rightarrow 1\)

- **BCP algorithms**
  1. **while** \((a_i\) gets a value), check if only one ‘\(x\)’ is left
     \(\implies O(n^2)\)
  2. Count num of ‘\(x\)’
     \(\implies O(n)\), but overhead in maintaining ‘\(x\)’ count
  3. Watch 2 literals in a clause. Only when watched literal changes, then we check the clause
     \(\implies \text{Amortized } O(C)\)
Can circuit SAT do 2-watch?

- Watch 2 fanins?
  - What’s the watch value?
  - How about gate output?
  - How about OR, NOR, NAND,.. gates?
  - How about XOR, MUX, ... complex gates?

omni-directional implications
A Closer Look

Different implications on circuit-based SAT actually map to the same implication on CNF SAT
Direct vs. Indirect Implications

1. Direct implication
   - Corresponding n 2-literal clauses in CNF SAT
   - Single implication source
   ➔ No need to watch
Direct Implication

1. Single source for each implication
2. Only depends on netlist structure; has nothing to do with the proving process (e.g. decisions, etc)
3. Should never encounter “CONFLICT” during the proof process
Direct Implication

Construct a "direct implication graph" in the preprocessing step.

Apply direct implications whenever a gate is implied to a value.
Direct vs. Indirect Implications

1. Direct implication
   - Corresponding n 2-literal clauses in CNF SAT
   - Single implication source
   - No need to watch

2. Indirect implication
   - Corresponding to the same (n+1)-literal clause
   - Only the last implied pin has different value
   - 2 watches: among all fanins and the gate itself
Indirect Implication
(AND gate)

- Select 2 pins (fanins or the gate itself) in a gate to watch
  - Almost the same as CNF SAT
  - For each gate, a list of watching gates
    - When a gate gets a value, perform direct implication and/or update watch for the gates on the watching list

```
watching-0: { a }
watching-1: { }
```

```
watching-0: { }
watching-1: { a, b }
```

watched pins
Watch Scheme for XOR Gate

- n-input XOR gate
  - $2^n (n+1)$-literal clauses
  - e.g. $(a + b + f) (a + b + f) (a + b + f) (a + b + f)$
- Implication occurs only when n variables become “known”
  - 2-watch; watch-known

watching-0: \{ a \}
watching-1: \{ \}
watching-known: \{ \}

watching-0: \{ \}
watching-1: \{ \}
watching-known: \{ \}

watching-0: \{ \}
watching-1: \{ b \}
watching-known: \{ a \}
Watch Scheme for MUX Gate

### MUX function: \( f = \neg s \land a + s \land b \)

<table>
<thead>
<tr>
<th>Implication</th>
<th>CNF clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg s \rightarrow (f = a) )</td>
<td>( (s + f + \neg a)(s + \neg f + a) )</td>
</tr>
<tr>
<td>( s \rightarrow (f = b) )</td>
<td>( (\neg s + f + \neg b)(\neg s + \neg f + b) )</td>
</tr>
<tr>
<td>( (a = b) \rightarrow (f = a) )</td>
<td>( (\neg f + a + b)(f + \neg a + \neg b) )</td>
</tr>
</tbody>
</table>

- **2-watch; watch-known?**
  - No. If watching \( \{ f, s \} \), when \( \{ a=1, b=1 \} \), we miss the implication \( \{ f=1 \} \)

- **3-watch; watch known!!**
  - Compared: CNF (6 clauses; 12 watch literals)
Watch Scheme for PB Gates

- PB Constraint:
  - $C_0p_0 + C_1p_1 + ... + C_{n-1}p_1 \geq C_n$, where $C_i \in \mathbb{Z}^+$, $p_i \in \mathbb{B}$
  - e.g. $4x_1 + 3x_2 + 2x_3 + x_4 \geq 3$

- How many watches?
  - 2-watch? All-watch?
  - Note:
    min #watches depends on the value assignments
  - Dynamic #watches? Too complicated....

- Our approach
  - $\forall$ combinations of assignments triggering implications
    $\Rightarrow$ max (min (#watches) )
Generic Watch Scheme

1. Watch candidate set
   - Output and input pins of the gate

2. Watched value ‘v’ for each pin
   - If ‘v’ on this pin may eventually lead to an indirect implication on other pin(s), then ‘v’ is the watched value of this pin

3. Find a minimum subset of watched candidates
   - (a) Assigning watched values on all the variables of the subset will produce an indirect implication
   - (b) Removing any of these assignments will void the implication
   - Let ‘k’ be the size of this subset.

4. We will need \((n - k + 1)\) watched pointers.
Generic Watch Scheme

5. Update of the watched pointer

- Called only when there is a watched-value assignment on the watched pin
- Other cases:
  - Whenever there are assignments on the non-watched pins
  - Non-watched value assignments on the watched pins
  - Do nothing
CNF vs. Circuit SAT

2-literal watch scheme

Antecedent ptr + UIP cut

Influence-guided + restart

BCP

Conflict Analysis

Conflict

learned

∅

No Sol.

Branch on a Variable

successful

all variables assigned

Sol. Found

CNF SAT

Circuit SAT

Topological scheduling

Imp Graph + UIP cut

Influence & structure-guided + restart
CNF vs. Circuit SAT

- 2-literal watch scheme
- Antecedent ptr + UIP cut
- Influence-guided + restart
- BCP
  - conflict
  - learned
  - No Sol.
- Branch on a Variable
  - OK
  - all variables assigned
- Sol. Found

Circuit SAT

- Generic watch scheme
- Imp Graph + UIP cut
- Influence & structure-guided + restart
- Successful
- CNF SAT

02/10/07 Chung-Yang (Ric) Huang ric@cc.ee.ntu.edu.tw +886-2-3366-3644
Conflict-Driven Learning

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

\[
(a + c + d)
\]

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

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

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

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

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

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

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

\[
(a + c)
\]

Learned clause

Implication Graph

Conflict source

Backtrack

Conflict!
Conflict-Driven Learning

Does CNF SAT record the imp graph?

No!!

Clauses:

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

Variables: a b c d

antecedent clause

\(a = 0\)

\(c = 0\)

\(d = 1\)

\((a + c + d)\)
Can circuit SAT do “antecedent gate”?  

<table>
<thead>
<tr>
<th>CNF SAT</th>
<th>Circuit SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Gate</td>
</tr>
<tr>
<td>Literal</td>
<td>Gate + value</td>
</tr>
<tr>
<td>Clause</td>
<td>Gate</td>
</tr>
</tbody>
</table>

Who “is” g’s antecedent gate?
Implicit Implication Graph

- If the implication type is “DIRECT”, then the antecedent pointer is the single implication source.

- If the implication type is “INDIRECT”, then the implication sources are the watched candidates of the antecedent gate that are
  (a) non-watched variables, and
  (b) watched variables with watched values, excluding the implied pin.
Implicit Implication Graph

Example

\[ f \rightarrow g \]

Imp type: DIRECT
Antecedent: \( f \)
\( \Rightarrow \) impSrc: \{ \( f \) \}

\[ g \rightarrow 0 \]

\[ 0 \rightarrow g \]

\[ g \rightarrow f \]

Imp type: INDIRECT
Antecedent: \( f \)
\( \Rightarrow \) impSrc: \{ \( f \), \( a \), \( b \) \}
Implicit Implication Graph

Example

- PB gate C: $4x_1 + 3x_2 + 2x_3 + x_4 \geq 3$
- Implication order: $\{ x_1 = 0, x_2 = 0 \}$
- Watched pins: $\{ x_2, x_3, x_4 \}$
- Indirect implications: "$x_3 = 1", "x_4 = 1"

$x_3$ Imp type: INDIRECT
Antecedent: C
$\Rightarrow$ impSrc: $\{ x_1, x_2 \}$

$x_4$ Imp type: INDIRECT
Antecedent: C
$\Rightarrow$ impSrc: $\{ x_1, x_2 \}$
CNF vs. Circuit SAT

2-literal watch scheme
Antecedent ptr + UIP cut
Influence-guided + restart

BCP
Conflict Analysis
Branch on a Variable

conflict learned

∅
No Sol.

OK

Sol. Found

all variables assigned

Circuit SAT

2-literal watch scheme
Generic watch scheme
Implicit Imp Graph + UIP cut
Influence & structure-guided + restart

CNF SAT
**Experimental Results**

**Table 1. Equivalence checking (EC) experiments**

<table>
<thead>
<tr>
<th>Time: seconds</th>
<th>without circuit info</th>
<th>with circuit info</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QuteSAT</td>
<td>zChaff</td>
</tr>
<tr>
<td>C3540</td>
<td>8.36</td>
<td>7.20</td>
</tr>
<tr>
<td>C7552</td>
<td>3.71</td>
<td>7.55</td>
</tr>
<tr>
<td>S35932</td>
<td>25.4</td>
<td>29.24</td>
</tr>
<tr>
<td>S38417</td>
<td>36.2</td>
<td>85.59</td>
</tr>
<tr>
<td>S38584</td>
<td>29.8</td>
<td>48.46</td>
</tr>
<tr>
<td>B15</td>
<td>116</td>
<td>168.8</td>
</tr>
<tr>
<td>B17</td>
<td>737</td>
<td>&gt;3600</td>
</tr>
<tr>
<td>B20</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
</tr>
<tr>
<td>ave rank</td>
<td>1.69</td>
<td>2.69</td>
</tr>
</tbody>
</table>
Our Contributions

1. A *generic watch scheme* that can seamlessly work on all kinds of circuit gates (simple or complex gates)
2. An *implicit implication graph* that enables efficient conflict-driven learning
3. Careful engineering work to implement most of the advanced SAT algorithm on the circuit data structure
Future Work

- More experiments!! Make sure the robustness of our engine for Boolean SAT
- Mission is not yet completed...
  1. Generalized for general constraint satisfiability and optimization problems
  2. Specialized for EDA specific applications
     (PB, ILP, SMT, …)