Verification of Large Industrial Circuits Using SAT Based Reparameterization and Automated Abstraction-Refinement

Pankaj Chauhan
Ph.D. Thesis Proposal

Computer Science Dept., Carnegie Mellon University
Challenges for a Symbolic Model Checker

- State, transition relation representation
  - Characteristic function based
  - Non-characteristic function based
- Image Computation
- State Space Traversal
- Abstraction/Refinement
State Exploration is the Key

- Bounded model checking (BMC) the first step for finding bugs in the large industrial circuits
- BMC is also a key element of automatic abstraction-refinement algorithms
- For large circuits, even BMC fails [Chauhan et al. FMCAD’02, Kroening et al. VMCAI’03]
Scope of the Thesis

Increase the capacity of state exploration engines by leveraging efficient SAT procedures.

- Image computation algorithms using SAT based enumeration
- Symbolic simulation
- Abstraction-refinement
SAT Based Image Computation
Basic Reachability Algorithm

\[ \text{Img}(S(x)) = \exists x, i. T(x, i, x') \land S(x) \]

**Reachability** \((S_0)\)

\[ S_{\text{reach}} \leftarrow \emptyset \]
\[ i \leftarrow 0 \]

**while** \((S_i \neq \emptyset)\) {
\[ S_{\text{reach}} \leftarrow S_{\text{reach}} \cup S_i \]
\[ S_{i+1} \leftarrow \text{Img}(S_i) \setminus S_{\text{reach}} \]
\[ i \leftarrow i + 1 \]
}

**return** \(S_{\text{reach}}\)
DPLL Style SAT Flowchart

A={}

Empty Clause?

Conflict?

Deduce conflict clause

Branch: add a literal to A

A is total?

UNSAT

SAT
SAT Based Reachability

A SAT Solver checks the validity of $\exists X, Y. f(X, Y)$.

- **Representation:** $S_i$ and $S_{reach}$ as DNF cubes
- **Computing Images:** Enumerate satisfying cubes
  $\Rightarrow$ compute $\exists X. f(X, Y)$.
- **Detection of Fixed Point:** At least one satisfying cube
Enumerate cubes in next state ($x'$) variables to the formula

$$S'_{i-1}(x) \land T(x, i, x') \land \neg S'_{reach}(x')$$

- Convert $S'_{i-1}$ and $T$ to CNF using intermediate variables.
- $S'_{reach}$ in DNF $\Rightarrow \neg S'_{reach}$ in CNF
- Add each satisfying assignment as to $S_i$ and in the SAT as blocking clause
Problems

- **Time complexity**
  - To many cubes to enumerate, each cube represents one state in $S_i$.

- **Space complexity**
  - Storage of cubes expensive and redundant
  - Cubes can be merged
Solving Time Problem

Given an assignment from SAT, reduce the number of assigned literals $\Rightarrow$ **Cube Enlargement**

- Circuit based heuristic to propagate free variables to infer the next state variable assignments that can be safely ignored
- An enlarged cube represents multiple states
- Enlarged cube is added as a blocking clause
Solving Space Problem

\[ x_1 \land x_5 \land x_6 \quad x_1 \land x_5 \land \neg x_6 \]
\[
\begin{array}{c}
\{\text{can be combined to form } x_1 \land x_5} \\
\text{Due to } S_{\text{reach}} \text{ constraint, } x_1 \land x_5 \land x_6 \land x_8 \text{ can never arise.} \\
\text{Therefore, sufficient to check merging with cubes that have the same set of variables}
\end{array}
\]
Solving Space Problem

A hash table based data structures to efficiently add a newly enumerated cube.

- For each added clause, use a hash table to find clauses with the same set of variables
- A smaller hash to find clauses differing only in one literal.

Can use zDDs or other data structure.
Relative Runtime v/s BDD
Space Savings

![Graph showing storage efficiency for different circuits. The graph plots the percentage space required after cube merging procedure against various circuits, comparing it to a 100% line.](image)
SAT Enumeration Summary

- Algorithm is relatively unaffected by the number of variables to be quantified.
- Important to reduce the number of cubes to enumerate.
- The biggest bottleneck is the clausal representation.
Work in Progress

- Benchmarking on processor designs of McMillan et al. CAV’02, Lahiri et al. CAV’02
- Use of interpolation proofs for non-clausal representation.
SAT Based Reparameterization
Characteristic Functions

\[ V = \{v_1, v_2\} \]
\[ S = \{01, 10\} \]

Characteristic function of \( S \) is

\[ \chi(V) = (v_1 \land \neg v_2) \lor (\neg v_1 \land v_2). \]

\( S \) is given by

\[ S = \{V | \chi(V) = 1\} \]
Parametric Representation[CM’90]

For $S = \{01, 10\}$

\[
\begin{align*}
  f_1(p_1) &= p_1 \\
  f_2(p_1) &= \neg p_1
\end{align*}
\]

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$f_1$</th>
<th>$f_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Parametric Representation [CM’90]

For \( S = \{01, 10\} \)

\[
\begin{align*}
  f_1(p_1) &= p_1 \\
  f_2(p_1) &= \neg p_1
\end{align*}
\]

Or

\[
\begin{align*}
  h_1(p_1, p_2) &= p_1 \lor p_2 \\
  h_2(p_1, p_2) &= \neg p_2 \land \neg p_2
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>( f_1 )</th>
<th>( f_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td>0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>( h_1 )</th>
<th>( h_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
<td>0 1</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td>1 0</td>
</tr>
</tbody>
</table>

Parametric Representation

If $S(V)$ is represented parametrically with a vector of $n$ functions $F(P) = (f_1(P), f_2(P), \ldots, f_n(P))$ over $k$ parameters $P = (p_1, p_2, \ldots, p_k)$, then

$$S(V) = \left\{ V \exists P. \left( \begin{array}{c}
v_1 = f_1(P) \\
v_2 = f_2(P) \\
\vdots \\
v_n = f_n(P)
\end{array} \right) \land \right\}.$$
A parametric representation can be easily converted to a characteristic function by using the following equation.

\[ \chi(V) = \exists P. \left( \begin{array}{c} v_1 \leftrightarrow f_1(P) \land \\ v_2 \leftrightarrow f_2(P) \land \\ \vdots \\ v_n \leftrightarrow f_n(P) \land \\ \end{array} \right) \]
Parametric Representation in Symbolic Simulation

If $I^m = I_0 \cup I_1 \cup \ldots \cup I_m$, then after $m$ steps of symbolic simulation,

$$S_f(V) = \{V|\exists I^m. v_1 = f_1(I^m) \land \ldots \land v_n = f_n(I^m)\}$$
Parametric Representation in Symbolic Simulation

Problems:

- The number of parameters, $|I^m|$, gets larger and larger with the number of simulation steps.
- Functions $f_1, f_2, \ldots$ keep on getting bigger and bigger.
Reparameterization

- Find functions $h_1(P), h_2(P), \ldots, h_n(P)$ in parameters $P$, such that $|P| \ll |I^n|$ and $S_h(V) = S_f(V)$. 
Reparameterization

- Find functions $h_1(P), h_2(P), \ldots, h_n(P)$ in parameters $P$, such that $|P| \ll |Im|$ and $S_h(V) = S_f(V)$.
- It has been shown that a set of vectors in $n$ variables can be represented by parametric functions of $n$ variables.
Reparameterization

- Find functions $h_1(P), h_2(P), \ldots, h_n(P)$ in parameters $P$, such that $|P| \ll |Im|$ and $S_h(V) = S_f(V)$.
- It has been shown that a set of vectors in $n$ variables can be represented by parametric functions of $n$ variables.
- We would like $|P| \leq n$. 
Reparameterization Algorithm

- The algorithm computes functions $h_1(P), h_2(P), \ldots, h_n(P)$ in that order.

We think of each new parameter $p_i$ as a free variable for the $i$th state bit $v_i$. 
Reparameterization Algorithm

- The algorithm computes functions $h_1(P), h_2(P), \ldots, h_n(P)$ in that order.
- $h_i(p_1, \ldots, p_i)$. This means that $h_i$ depends on the first $i$ parameters only. The parantheses denote the arguments to the function in the rest of the talk.
Reparameterization Algorithm

- The algorithm computes functions $h_1(P), h_2(P), \ldots, h_n(P)$ in that order.

- $h_i(p_1, \ldots, p_i)$. This means that $h_i$ depends on the first $i$ parameters only. The parantheses denote the arguments to the function in the rest of the talk.

- We think of each new parameter $p_i$ as free variable for the $i^{th}$ state bit $v_i$. 
A Known Decomposition
[CM’90]

\[ h_i(p_1, \ldots, p_i) = h^1_i(p_1, \ldots, p_{i-1}) \lor p_i \cdot h^c_i(p_1, \ldots, p_{i-1}). \]

where,

- \( h^1_i(p_1, \ldots, p_{i-1}) \): Boolean condition under which \( v_i \) is forced to 1
- \( h^c_i(p_1, \ldots, p_{i-1}) \): Boolean condition under which \( v_i \) is free to choose any value
A Known Decomposition
[CM’90]

\[ h_i(p_1, \ldots, p_i) = h^1_i(p_1, \ldots, p_{i-1}) \lor p_i \cdot h^c_i(p_1, \ldots, p_{i-1}). \]

where,

- \( h^1_i(p_1, \ldots, p_{i-1}) \): Boolean condition under which \( v_i \) is forced to 1
- \( h^0_i(p_1, \ldots, p_{i-1}) \): Boolean condition under which \( v_i \) is forced to 0
A Known Decomposition
[CM’90]

\[ h_i(p_1, \ldots, p_i) = h^1_i(p_1, \ldots, p_{i-1}) \lor p_i \cdot h^c_i(p_1, \ldots, p_{i-1}). \]

where,

- \( h^1_i(p_1, \ldots, p_{i-1}) \): Boolean condition under which \( v_i \) is forced to 1
- \( h^0_i(p_1, \ldots, p_{i-1}) \): Boolean condition under which \( v_i \) is forced to 0
- \( h^c_i(p_1, \ldots, p_{i-1}) \): Boolean condition under which \( v_i \) is free to choose any value
A Known Decomposition
[CM’90]

For $S = \{01, 10\}$, suppose $h_1(p_1) = p_1$, then $p_1 = 0 \Rightarrow v_1 = 0 \Rightarrow v_2 = 1$. Thus, $h_{21}(p_1) = \neg p_1$.

Moreover,

$$h_i^c = \neg(h_i^1 \lor h_i^0) = \neg h_i^1 \land \neg h_i^0$$

i.e., $h_i^1$, $h_i^0$ and $h_i^c$ are mutually exclusive and complete.
Computing $h_i^1$ and $h_i^c$

Restriction Function

$$\rho_i(p_1, \ldots, p_{i-1}, I^m) = \bigwedge_{j=1}^{i-1} h_j(p_1, \ldots, p_j) = f_j(I^m)$$

Set of input vectors for which the functions $f_1$ to $f_{i-1}$ compute the same values as those computed by $h_1$ to $h_{i-1}$ for the given parameter assignment $p_1, p_2, \ldots, p_{i-1}$.

Note that $\rho_1 = 1$. 
Computing $h_i^1$ and $h_i^c$

$$h_i^1(p_1, \ldots, p_{i-1}) = \forall I^m. \rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = 1$$

Reparameterization is very expensive with BDDS, even with an altogether different set union based approach.
Computing $h^1_i$ and $h^c_i$

\[
h^1_i(p_1, \ldots, p_{i-1}) = \forall I^m . \rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = 1
\]

\[
h^0_i(p_1, \ldots, p_{i-1}) = \forall I^m . \rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = 0
\]
Computing $h^{1}_{i}$ and $h^{c}_{i}$

$$h^{1}_{i}(p_1, \ldots, p_{i-1}) = \forall I^m. \rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = 1$$

$$h^{0}_{i}(p_1, \ldots, p_{i-1}) = \forall I^m. \rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = 0$$

Reparameterization is very expensive with BDDS, even with an altogether different set union based approach.
High Level Algorithm

\(\text{ORDEREDREPARAM}(\bar{f}(I^m) = (f_1(I^m), \ldots, f_n(I^m)))\)

\(\text{for } i = 1 \text{ to } n \)

\(\rho_i \leftarrow 1\)

\(\text{for } j = 1 \text{ to } i - 1\)

\(\rho_i \leftarrow \rho_i \land (h_j = f_j)\)

\(h_i^1 \leftarrow \forall I^m. (\rho_i \leftarrow f_i = 1)\)

\(h_i^0 \leftarrow \forall I^m. (\rho_i \leftarrow f_i = 0)\)

\(h_i^c \leftarrow \neg (h_i^1 \lor h_i^0)\)

\(h_i \leftarrow h_i^1 \lor (p_i \land h_i^c)\)

\text{endfor}

\text{return } (h_1(P), h_2(P), \ldots, h_n(P))\)
SAT Based Enumeration

\[ h_i^\alpha(p_1, \ldots, p_{i-1}) = \forall I^m. \rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = \alpha \]

\[ = \neg \exists I^m. (\rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = \alpha) \]

\[ = \neg \exists I^m. \rho_i(p_1, \ldots, p_{i-1}, I^m) \land f_i(I^m) \neq \alpha \]

- SAT based enumeration computes \( \neg h_i^\alpha \) in DNF, thus \( h_i^1 \) and \( h_i^0 \) are in CNF.
- Enumerates on \( \rho_i(p_1, \ldots, p_{i-1}, I^m) \) to compute both \( \neg h_i^1 \) and \( \neg h_i^0 \) in a single call.

Incremental SAT is the key to the performance.
Incremental SAT

Enumeration formula for $h_i^\alpha$ is very similar to the formula for $h_{i-1}^\alpha$.

- Remove blocking clauses and the conflict clauses derived from them from $\rho_{i-1}$.
- Add clauses for $h_{i-1} = f_{i-1}$, repeat
### Experimental Results

<table>
<thead>
<tr>
<th>ckt</th>
<th># regs</th>
<th># inp</th>
<th>bug len</th>
<th>bmc</th>
<th>fmcad bug time</th>
<th>symbol time</th>
<th>symbol max len</th>
<th>symbolic</th>
<th># reps</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2+</td>
<td>94</td>
<td>11</td>
<td>15</td>
<td>18</td>
<td>79</td>
<td>32</td>
<td>221</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>D5+</td>
<td>343</td>
<td>7</td>
<td>32</td>
<td>15</td>
<td>38.2</td>
<td>17</td>
<td>127</td>
<td>1000</td>
<td>13</td>
</tr>
<tr>
<td>D24</td>
<td>223</td>
<td>47</td>
<td>10</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>543</td>
<td>1000</td>
<td>9</td>
</tr>
<tr>
<td>D6</td>
<td>161</td>
<td>16</td>
<td>20</td>
<td>289</td>
<td>833</td>
<td>145</td>
<td>64</td>
<td>1000</td>
<td>5</td>
</tr>
<tr>
<td>D18</td>
<td>498</td>
<td>247</td>
<td>28</td>
<td>6834</td>
<td>9955</td>
<td>1698</td>
<td>56</td>
<td>3000</td>
<td>7</td>
</tr>
<tr>
<td>D20</td>
<td>532</td>
<td>30</td>
<td>14</td>
<td>2349</td>
<td>1947</td>
<td>574</td>
<td>89</td>
<td>3000</td>
<td>9</td>
</tr>
<tr>
<td>IUp1</td>
<td>4494</td>
<td>361</td>
<td>true</td>
<td>3000*</td>
<td>3350</td>
<td>-</td>
<td>183</td>
<td>3000</td>
<td>45</td>
</tr>
<tr>
<td>IUp2</td>
<td>4494</td>
<td>361</td>
<td>true</td>
<td>3000*</td>
<td>712</td>
<td>-</td>
<td>183</td>
<td>3000</td>
<td>45</td>
</tr>
</tbody>
</table>

* BMC could complete only 39 steps of simulation before running out of space

+ BDD based symbolic simulator could do only these two circuits.
Runtime for Bug Finding

![Runtime for Bug Finding](chart.png)
Simulation Depth v/s BMC

![Simulation Depth Diagram](image)
Ongoing and Future Work

- Counterexample generation
- Safety property checking
- Handling of general transition constraints, SMV style \texttt{INVAR, TRANS}
- Effect of variable orders, not as severe as BDDs
Ongoing and Future Research

- Improve enumeration algorithm
- Effects of variable ordering on parameterized representation
- Proving properties true?
Abstraction Refinement
CEGAR Loop

1. Generate an initial abstraction function $h$.

2. Build abstract machine $\hat{M}$ based on $h$. Model check $\hat{M}$. If $\hat{M} \models \varphi$, then $M \models \varphi$. Return TRUE.

3. If $\hat{M} \not\models \varphi$, check the counterexample on the concrete model. If the counterexample is real, $M \not\models \varphi$. Return FALSE.

4. Refine $h$, and go to step 2.
Preservation Theorem

Let $\hat{M}$ be an abstraction of $M$ corresponding to the abstraction function $h$, and $p$ be a propositional formula that respects $h$. Then

$$\hat{M} \models AGp \Rightarrow M \models AGp$$

Diagram:

```
  p   p   p   p   ~p
  |   |   |   |   |
  p   p   p   p   |
  |   |   |   |   |
  p   p   p   ~p   |
```

Refinement
Refinement

- Spurious transition because *deadend* states and *bad* states lie in the same abstract state.
- Refinement: Put deadend and bad states in separate abstract states.
Refinement Using SAT Conflict Analysis

- The SAT formula

\[ \psi_m = \{ (s_1 \ldots s_m) \mid I(s_1) \land \bigwedge_{i=1}^{m-1} R(s_i, s_{i+1}) \land \bigwedge_{i=1}^m h(s_i) = \hat{s}_i \} \]

describes the set of paths corresponding to the abstract counterexample

- We solve \( \psi_m \) with a SAT solver

- For a spurious counterexample, \( \psi_m \) is unsatisfiable
Refinement Using SAT Conflict Analysis

SAT solvers record the important reasons for the unsatisfiability during the SAT check by Boolean constraint propagation and implication graphs.

We proposed [FMCAD’02] two methods to identify important variables by analysing conflicts generated during the SAT check:

- Heuristically score the variables during the SAT check
- Identify important variables by conflict dependency graphs
Experimental Results for Refinement by Conflict Analysis

Apart from the smaller IU circuits shown earlier, these techniques are able to handle large D series circuits and an IU circuit with 5000 latches.

<table>
<thead>
<tr>
<th>circuit</th>
<th># regs</th>
<th>ctrex length</th>
<th>CSMV time</th>
<th>Heuristic Score</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>time</td>
<td>iters</td>
</tr>
<tr>
<td>D2</td>
<td>105</td>
<td>15</td>
<td>152</td>
<td>105</td>
<td>10</td>
</tr>
<tr>
<td>D5</td>
<td>350</td>
<td>32</td>
<td>1,192</td>
<td>29</td>
<td>3</td>
</tr>
<tr>
<td>D6</td>
<td>177</td>
<td>20</td>
<td>45,596</td>
<td>784</td>
<td>24</td>
</tr>
<tr>
<td>D18</td>
<td>745</td>
<td>28</td>
<td>&gt;4 hrs</td>
<td>12,086</td>
<td>69</td>
</tr>
<tr>
<td>D20</td>
<td>562</td>
<td>14</td>
<td>&gt;7 hrs</td>
<td>1,493</td>
<td>56</td>
</tr>
<tr>
<td>D24</td>
<td>270</td>
<td>10</td>
<td>7,850</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>IU-p1</td>
<td>4855</td>
<td>true</td>
<td>-</td>
<td>9,138</td>
<td>22</td>
</tr>
<tr>
<td>IU-p2</td>
<td>4855</td>
<td>true</td>
<td>-</td>
<td>2,820</td>
<td>7</td>
</tr>
</tbody>
</table>
Ongoing and Future Work

• Use symbolic simulation with SAT based reparameterization to simulate abstract counterexamples
• Derive refinement information from the partial simulation
• Extend the method for liveness checking using safety checking as in Biere et al. STTT’03
Expected Thesis Contributions

• SAT based enumeration algorithms for image computation
• Powerful symbolic simulator to simulate deep large circuits with thousands of latches
• Completeness of property checking via automated SAT based abstraction-refinement
Timeline

- Feb’04:
- June’04:
- Sept’04: