Dynamic Abstraction Using SAT-based BMC

Liang Zhang† Mukul R Prasad‡ Michael S Hsiao§ Thomas Sidle‡
†Cadence Design Systems, San Jose, CA
‡Advanced CAD Technology, Fujitsu Laboratories of America, Sunnyvale, CA
§Department of Electrical & Computer Engineering, Virginia Tech, Blacksburg, VA

1. INTRODUCTION

The application of model checking has traditionally been hampered by the commonly known state explosion problem. Abstraction refinement has recently emerged as a promising technology that has the potential to alleviate this problem.

The basic idea of abstraction refinement [7] is to verify the property at hand on a simplified version, or abstraction, of the given design. The abstraction is generated such that whenever a property passes on the abstract model, it is guaranteed to pass on the original design as well. However, when a property fails on the abstract model, the produced counter-examples must be checked to see if they are true counter-examples on the original design. If not, the model checking process is iterated with another abstract model which approximates the original model more closely. The new abstract model can be obtained either by refinement, which embellishes the current abstraction with more details from the original design [2, 5, 12] or by re-generating a more detailed abstract model from the original design [6, 11]. Usually the challenge in abstraction refinement is to construct as small an abstract model as possible so that the model checker can handle it easily. At the same time, the abstract model should retain sufficient details so that the model checker can prove the property. Previous work on abstraction refinement based model checking uses static abstraction in that the abstract model produced by the abstraction step is never modified by the downstream model checker. In this work we make the key observation that when a property is checked on a circuit model, there may be state elements that are partially abstractable, i.e., while a state element is necessary in the proof of the property, it may actually be required only in certain time-frames in the proof. For example, some latches in the design are solely present for initialization purposes.

We propose a new dynamic method of abstraction, which can be applied during successive steps of the model checking algorithm to further reduce the model produced by traditional static abstraction methods. This is facilitated by information gathered from an analysis of the proof of unsatisfiability of SAT-based bounded model checking problems formulated on the original design. The dynamic abstraction effectively allows the model checker to work with smaller abstract models. Experiments on several industrial benchmarks demonstrate that dynamic abstraction can significantly improve both the performance and the capacity of typical abstraction refinement flows.

Categories and Subject Descriptors B.6.2 [Design Aids]: Verification

Keywords: Design, Verification

2. RELATED WORK

Abstraction refinement algorithms can be broadly classified into two categories: 1) counter-example driven and 2) counter-example independent. Counter-example driven methods [1, 4, 5, 10, 12] typically work by iteratively refining the current abstraction so as to block a particular false counter-example encountered in model checking the abstract model. The refinement algorithm could use a combination of structural heuristics and/or functional analysis based on SAT or BDDs. Recent papers [2, 9] enlarged the scope of the refinement by blocking multiple false counter-examples from the abstract model.

Counter-example independent abstraction refinement was introduced in [11] and also independently discovered in [6]. The basic idea is to perform a SAT-based BMC [3] for the property, up to some depth k, on the original design and then generate the abstract model based on an analysis of the proof of unsatisfiability [15] of the BMC problem. Since the abstraction preserves latches and gates that are included in the proof of unsatisfiability of the BMC problem, it guarantees that the abstract model does not have any counter-examples up to depth k. If needed, successive abstract models can be generated by solving BMC problems of increasing depths. The use of BMC to concretize abstract counter-examples was first proposed in [5]. Bjesse et al. [1] proposed an enhancement of the concretization in that the concrete error trace
do not have to be in the same length as the abstract counter-example. Li et al. [8] proposed a new search strategy for the SAT solver so that the proof of unsatisfiability will generate smaller abstract models.

All of the above works share two common features: 1) the abstraction step is algorithmically distinct from the model checking phase, and 2) The abstraction is purely structural and has no temporal component, i.e. the same structural abstraction is used for each image computation step in BDD-based model checking.

Our dynamic abstraction can be distinguished from all previous abstraction algorithms based on these two aspects. Our method first analyzes the temporal behavior of various latches. Then, based on the analysis it dynamically abstracts away a set of latches during the course of the model checking. A key point is that dynamic abstraction can be applied in addition to any traditional abstraction method.

3. PRELIMINARIES

In this paper, we only consider model checking of invariants $AGp$, where $p$ is a boolean expression of the given circuit model. The circuit can be represented as $M = (T(X,Y,W), I(X))$, where $W$ the set of inputs, $X$ the set of present state variables, $Y$ the set of next state variables, $T(X,Y,W)$ the transition relation, and $I(X)$ the set of initial states. $M$ has a set of latches $L = \{l_1,l_2,...,l_m\}$. $x_i$ and $y_i$ are the present state and next state variable corresponding to latch $l_i$. The transition relation $T$ can be represented as:

$$T(X,Y,W) = \bigwedge_{i=1}^{m} T_i(X,y_i,W)$$

$T_i(X,y_i,W) = y_i \rightarrow \Delta_i(X,W)$ is the transition relation of latch $l_i$.

Given a subset of latches $L_{abs}$ that we would like to abstract away from the design, the abstract model can be constructed by opening up the feedback loop of latches $L_{abs}$ at their present state variables $X_{abs}$. The abstract model can then be represented as $\hat{M} = (\hat{T}(\hat{X},\hat{Y},\hat{W}), I(\hat{X}))$, where $\hat{W} = W \cup X_{abs}$, $\hat{X} = X \setminus X_{abs}$, $\hat{Y} = \{y_j : x_j \in \hat{X}\}$.

The basic framework for abstraction refinement in our current implementation is similar to the one developed in [11] and [6]. A simplified version of the algorithm used in [11] is shown in Algorithm 3.1.

### Algorithm 3.1: Abstraction Refinement Using SAT-BMC

1. $k = $ InitValue
2. if SAT-BMC($M, p, k$) is SAT then
3.  return “found error trace”
4. else
5.  Extract proof of unsatisfiability, $\mathcal{P}$ of SAT-BMC
6.  $M' = ABSTRACT(M, \mathcal{P})$
7.  end if
8. if MODELE_CHECK($M'$, $p$) returns PASS then
9.  return “passing property”
10. else
11.  Increase bound $k$
12.  goto Step 2
13. end if

### Algorithm 3.2: Symbolic Invariant Checking

#### Algorithm 3.2: Symbolic Invariant Checking

1: $S_C = \emptyset; S_Y = I$
2: while $S_C \neq S_Y$ do
3:  $S_C = S_Y$
4:  if $B \cap S_C \neq \emptyset$ then
5:    return “found error trace”
6:  end if
7:  $S_Y = S_C \cup IMG(S_C)$
8: end while
9: return “no bad state reachable”

4. DYNAMIC ABSTRACTION

Given an original circuit $M$ and the property $P = AGp$, let us assume that a SAT-BMC problem on $M$ of depth $k$ has been solved and there is no counter-example. Further, suppose that the SAT solver generates a POU $P$ for this problem. Figure 1 is a graphical representation of the POU from a 40-step SAT-BMC problem on a real circuit example. For each latch (plotted for 40 representative latches on the y-axis) the plot shows the time-frames at which the corresponding instantiation of the latch variable involves in the POU of the SAT-BMC problem. The latches have been sorted on the y-axis for better readability of the data. Given a latch variable $l_i \in L$, we can define the redundancy index (RI), $\rho(l)$ of $l$, with respect to the proof $P$, as follows:

#### Definition 4.1 (Redundancy Index).

The redundancy index $\rho(l)$ of latch $l$ with respect to the proof of unsatisfiability $P$ is the smallest time-frame index such that for all time-frames $j$, $\rho(l) \leq j \leq k$, there does not exist a clause with variable $l_j^i$ in $P$. 

![Figure 1: Latch-based Unsatisfiability Analysis](image-url)
For example, in Figure 1 the points marked A(14,15) and B(27,32) show that the latch 15 has a RI of 15, while the latch 32 has a RI of 28. Simply put, the redundancy index is the earliest time-frame after which the given latch stops participating in the POU of the corresponding BMC problem. The situation depicted in Figure 1 is quite typical for a large variety of benchmarks we have experimented with. Large number of latches are not used in all time-frames of the POU. Moreover, some latches are only used in the first few time-frames.

At each step of image computation we can define a candidate set of latches which is essentially a set of latches whose redundancy index is no greater than the index of the current image computation step.

**Definition 4.2 (Candidate Set).** The candidate set of latches for iteration \( j \) of image computation in Algorithm 3.2 is defined as \( C_j = \{ l : l \in L, p(l) \leq j \} \).

For example, in Figure 1 the candidate set at time-frame 15 consists of the first 15 latches, i.e., \( C_{15} = \{ l_1, l_2, \ldots, l_{15} \} \). A modified version of Algorithm 3.2, incorporating dynamic abstraction is given below.

```
InvariantCheck_DynamicAbstract (M(T, I), B)
1: \( S_c = B, S_y = I \);
2: while \( S_c \neq S_y \) do
3: \( L_{abs} = \text{Choose Abstraction Latches}(L) \);
4: \( T = \text{ABSTRACT_TR}(L_{abs}, T) \);
5: \( B = \exists x \text{abs}, B \);
6: \( S_c = S_y \);
7: \( S_c = \exists x \text{abs}, S_c \);
8: if \( B \& S_c \neq \emptyset \) then
9: return "found error trace";
10: end if
11: \( S_y = S_c \cup \text{img}(S_c) \);
12: end while
13: return "no bad state reachable";
```

**Algorithm 4.1: Symbolic Invar. Checking with Dynamic Abstraction**

In Algorithm 4.1 \( X_{abs} \) are the present state variables corresponding to the latches \( L_{abs} \) chosen for abstraction, \( \text{ABSTRACT_TR} \) abstracts the chosen latches from the \( T \) using the approach described in Section 3. The operation \( \text{Choose Abstraction Latches} \) may use different heuristics to choose a subset of the candidate latches \( C_j \) of the current iteration for abstraction. Since the dynamic abstraction is developed from an analysis of the POU of the \( k \)-step SAT-BMC problem, the following proposition about the Algorithm 4.1 is true.

**Theorem 4.1.** Algorithm 4.1 will not find any counter-example to the given property in the first \( k \) steps of image computation.

The proof of of this theorem is along the lines of the main result in [11]. Indeed, the static abstraction can be viewed as a special case of dynamic abstraction, since if we restrict our abstraction to latches with redundancy index ‘0’, the Algorithm 4.1 becomes the conventional static abstraction-based invariant model checking.

### 4.1 Selection Heuristic

A key component of Algorithm 4.1 is the latch selection heuristic \( \text{Choose Abstraction Latches} \). This decides which latches, out of the current candidate set, should be abstracted at a given image computation step. This heuristic can have a significant bearing on the overall performance of the algorithm.

The most aggressive approach would be to perform dynamic abstraction for all latches in the current candidate set and at the earliest possible time as indicated by the redundancy index (RI) of each latch. However, this approach suffers from several drawbacks. The following issues drive the choice of this heuristic:

1. **How often to abstract latches:** Since the dynamic abstraction is implemented via quantification of next-state variables, from the transition relation, the overhead can be significant. Thus, a good heuristic should limit the abstraction to a few image computation steps.
2. **Extrapolating unbounded behavior from \( p(l) \):** In abstraction based on POU of a \( k \)-step SAT-BMC we are trying to extrapolate unbounded behavior of a latch, with respect to the given property, based on its bounded behavior. Intuitively, a latch that was active only in the first few steps of the BMC (i.e. has a small RI), e.g., latch \( l_2 \) in Figure 1, is more likely to be inactive beyond \( k \) time-steps than one which was active up to \( k-1 \) or \( k-2 \) steps (i.e. has a large RI), e.g., latch \( l_{38} \) in Figure 1.
3. **Size and depth of the reachable state space:** Abstraction of latches comes at the cost of enlarging the set of permissible behaviors of the circuit. This can potentially enlarge the reachable state space, result in larger BDDs for the reached state representation and/or increase the depth of the reachability computation. This factor should be considered by the latch selection heuristic.

With the above criteria in mind, we have developed and tested several heuristics for \( \text{Choose Abstraction Latches} \) and found the following two to give a reasonable trade-off between overheads and abstraction power. Several other richer variants of these are possible and could be the subject of future research.

**Heuristic 1:** Dynamically abstract just once at \( \lfloor \delta - k \rfloor \) time-steps (where \( 0 < \delta < 1 \)), and abstract all latches in the candidate set at this point.

The philosophy behind this heuristic is to minimize the overheads of abstraction by doing it only once (issue 1 above) and being aggressive by choosing all candidates for abstraction. \( \delta \) is kept fairly low to increase the likelihood of the latches being redundant for future image computations (in agreement with issue 2 above). Empirically, we used \( \delta = 0.2 \) in our experiments.

**Heuristic 2:** Before the start of model checking analyze the proof \( B \) and gather a set of latches \( S = \{ l : l \in L, p(l) \leq \delta - k \} \). Every \( r \) steps of image computation, compute the set of latches \( N_r \) not in the support of current reached state set \( BDD \). If \( \mid \Delta N_r \mid \geq \gamma \) then abstract all latches in the set \( \Delta N_r \). Repeat every \( r \) image computation steps.

The intuition behind the using of the set \( N_r \) is that the removal of such latches is less likely to cause a blow-up in the current step of image computation. This ties in with issue 3 discussed above. Empirically, we used parameter settings of \( \delta = 0.2, r = 2, \tau = 10 \) for our experiments but the heuristic is not sensitive to these particular settings. Qualitatively, **Heuristic 1** is based on an aggressive one time application of dynamic abstraction whereas **Heuristic 2** is a more conservative and controlled application of dynamic abstraction. This distinction is born out by the experiments discussed in Section 5.

### 4.2 Handling Counter-examples

Our current implementation uses the counter-example independent refinement. Whenever a counter-example is found in the abstract model (line 9 of Algorithm 4.1) SAT-BMC is used to check if it is a true counter-example on the concrete model. If it is not a true counter-example, then we repeat the abstraction process with a deeper unrolling for SAT-BMC as shown in Algorithm 3.1. Otherwise, an error trace is returned to the user. Note that the effectiveness of proposed dynamic abstraction is unaffected by the underlying refinement scheme.

### 5. EXPERIMENTAL RESULTS

We have implemented the proposed dynamic abstraction algorithm as well as the static abstraction algorithm of [6, 11] within the VIS framework [13]. Our framework first abstracts a static model using an iterative abstraction loop. The POU extraction is based on the algorithm of [15] and it has been extended to report the redundancy index.
(RJ) for each latch. The downstream model checker has been modified to take RJs of latches as inputs. It then further abstracts the statically abstracted model on the fly using proposed latch selection heuristics. We use CUDD for the BDD-based computation, and ZCHAFF [14] as the SAT solver for BMC. We tested our tool for safety properties on different modules from four real-life industrial designs.

All experiments were run on 1.5 GHz Pentium 4 Linux machines with 1G RAM. The time-out limit is set to 24 hours. The results are reported in the Table 1. The second column shows if the property is a passing property or the length of the shortest counterexample. A question mark was shown for P14, since all methods timed out on it. Column 3 shows the number of latches in the statically abstracted model. Column 7 is the cumulative CPU time, which includes both abstraction and model checking time. Columns 8 and 10 report the number of latches in the final dynamically abstracted model for Heuristics 1 and 2 as well as additional latches abstracted with respect to the static abstraction method. For example, the static abstraction is able to abstract a model with only 224 latches for property P8, for which the concrete model has 5468 latches. Dynamic heuristic 1 is able to further abstract 40 more latches away, while dynamic heuristic 2 is able to abstract 23 more latches. The reported time in column 9 and 11 are also cumulative CPU times. For time-out cases, we also report the number of completed image computation steps. For example, for P14, dynamic heuristic 1 is able to compute 26 steps of images within 24 hours, while static abstraction and heuristic 1 can only finish 13 steps.

We can see that proposed heuristic 1 is extremely powerful in reducing the overall runtime, even though the number of additional abstracted latches may not be very significant. For example, with only 10 additional latches abstracted away for P12, it achieves over an order of magnitude speed-up compared to the pure static abstraction approach. However, as discussed in previous section, aggressive abstraction may potentially slow down the subsequent model checking. In our experiment, dynamic heuristic 1 does experience occasional slow-downs. As explained in the previous section, heuristic 2 is a more conservative and controlled application of the dynamic abstraction. It consistently outperforms the pure static abstraction for all cases significantly.

Table 1: Results: Static Abstraction and Proposed Dynamic Abstraction

<table>
<thead>
<tr>
<th>Problem</th>
<th>Pass/ cex leng.</th>
<th>Concrete Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Pass</td>
<td>330 1158 5155</td>
</tr>
<tr>
<td>P2</td>
<td>Pass</td>
<td>401 1896 8910</td>
</tr>
<tr>
<td>P3</td>
<td>Pass</td>
<td>405 1951 8577</td>
</tr>
<tr>
<td>P4</td>
<td>Pass</td>
<td>671 2735 11381</td>
</tr>
<tr>
<td>P5</td>
<td>Pass</td>
<td>1015 2971 10044</td>
</tr>
<tr>
<td>P6</td>
<td>Pass</td>
<td>1020 3039 10061</td>
</tr>
<tr>
<td>P7</td>
<td>Pass</td>
<td>1981 5407 18193</td>
</tr>
<tr>
<td>P8</td>
<td>Pass</td>
<td>1950 5468 19161</td>
</tr>
<tr>
<td>P9</td>
<td>Pass</td>
<td>1943 5644 19189</td>
</tr>
<tr>
<td>P10</td>
<td>Pass</td>
<td>3490 8998 27297</td>
</tr>
<tr>
<td>P11</td>
<td>60</td>
<td>308 746 3837</td>
</tr>
<tr>
<td>P12</td>
<td>36</td>
<td>289 654 4823</td>
</tr>
<tr>
<td>P13</td>
<td>29</td>
<td>289 654 4826</td>
</tr>
<tr>
<td>P14</td>
<td>?</td>
<td>356 1644 7408</td>
</tr>
<tr>
<td>P15</td>
<td>Pass</td>
<td>82 432 1740</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3185</td>
</tr>
</tbody>
</table>

6. CONCLUSIONS

In this paper we have presented a method for performing dynamic abstraction within a framework for abstraction-refinement based model checking. The dynamic abstraction is applied during successive image computation steps of the model checking algorithm and can be applied in addition to traditional static abstraction methods. It is facilitated by information gathered from an analysis of the proof of unsatisfiability of SAT-based BMC problems formulated on the concrete model. We also proposed two strategies for realizing the dynamic abstraction. Our experiments on several large industrial designs demonstrate that proposed techniques can improve the performance of abstraction refinement based model checking by up to an order of magnitude compared to the state-of-the-art static abstraction refinement methods.

7. REFERENCES


