Localizing Errors in Counterexample with Iteratively Witness Searching

ShengYu Shen, Ying Qin, and SiKun Li

Office 607, School of Computer Science, National University of Defense Technology
410073 ChangSha, China
{syshen,qy123,skli}@nudt.edu.cn
http://www.nudt.edu.cn

Abstract. We propose a novel approach to locate errors in complex counterexample of safety property. Our approach measures the distance between two state transition traces with difference of their control flow. With respect to this distance metrics, our approach search for a witness as near as possible to the counterexample. Then we can obtain the set of control flow predicates with difference assignment in witness and counterexample. Run this witness-searching algorithm iteratively, we can then obtain a predicate list with priority. A predicate with higher priority means that this predicate is more likely the actual error. Experiment result shows that our approach is highly accurate.[1]

1 Introduction

Today, model checking is one of the most important formal verification approaches. It is widely employed to verify software and hardware system. One of its major advantages in comparison to such method as theorem proving is the production of a counterexample, which explains how the system violates some assertion.

However, It is a tedious task to understand the complex counterexamples generated by model checking complex hardware system. Therefore, how to automatically extract useful information to aid the understanding of counterexample, is an area of active research.

Many researchers [1,2,4,9] engage in locating errors in counterexample. They first search for a witness as similar as possible to a counterexample. Starting from the difference between them, actual error can then be located by perform a breath-first source code checking.

However, these approaches suffer from a serious problem called "multiple nearest witnesses"(MNW). Under certain circumstance, there are multiple nearest witnesses. Distance between counterexample and these witnesses are of the

[1] Supported by the National Natural Science Foundation of China under Grant No. 90207019; the National High Technology Development 863 Program of China under Grant No. 2002AA1Z1480

F. Wang (Ed.): ATVA 2004, LNCS 3299, pp. 456–469, 2004.
© Springer-Verlag Berlin Heidelberg 2004

same, and only one of them contains the actual error. Then the first nearest witness may be far from the actual error. Thus, they need to perform a very deep breath first code checking and traverse a large fraction of the code, before they found the actual error. In this way, MNW significantly decrease the accuracy of error locating.

At the same time, if they meet with a node with large number of fan-in in breath-first code checking, they will also need to traverse a large number of nodes. This will also significant decrease the accuracy of error locating.

To overcome these problems, we propose a novel error locating approach that based on iteratively witness searching, to improve the accuracy of error locating. We measure the distance between two state transition traces with difference between assignments to their control flow predicates. With this distance metrics, we search for a witness as similar as possible to a counterexample. Then the predicates that take on different assignment can be appended to the tail of a prioritized list. Run this witness-searching algorithm iteratively, we can then obtain a prioritized list of predicates. A predicate with higher priority is more likely the actual error.

The main advantage of our technique is: We use the prioritized predicate list as the result of error locating, no need to perform breath-first code checking. Thus, avoid the impaction of MNW and wide fan-in node.

We implement our algorithm in NuSMV[12]. Moreover, it is straightforward to implement our algorithm for other language such as verilog. The experiment result shows that our approach is much more accurate than that of [2].

The remainder of the paper is organized as follows. Section 2 presents background material. Section 3 describes the impaction of MNW and wide fan-in node. Section 4 presents the algorithm that locates error in counterexample. Section 5 present experiment result of our approach and compare it to that of [2]. Section 6 reviews related works. Section 7 concludes with a note on future work.

2 Preliminaries

2.1 Counterfactual, Distance Metrics, and Causal Dependence

A common intuition to error explanation and error localization is that: successful executions that closely resemble a faulty run can shed considerable light on the cause of the error [1,2,4,9]. David Lewis [6] proposes a theory of causality based on counterfactual, which provides a justification for this intuition. Lewis holds that a cause is something that makes a difference: if the cause c had not been, the effect e would not have been. Lewis equates causality to an evaluation based on distance metrics between possible worlds. We present the definition of counterfactual and causal dependence below:

Definition 1 (Counterfactual). *Assume A and C hold in world W, then counterfactual* $A \mapsto C$ *hold in world W w.r.t distance metrics d iff there exist a W' such that:*

1. A and C hold in world W';
2. For all world W", if A hold and C not hold, then d(W,W') < d(W,W").

Definition 2 (Causal Dependence). *If predicate c and e both hold in world W, then e causally depends on c in world W iff* $\neg c \mapsto \neg e$.

We express "e causal depends on c in world w" as following formula:

$$c(w) \wedge e(w) \wedge \exists w'(\neg c(w') \wedge \neg e(w') \wedge \forall w"((\neg c(w") \wedge e(w")) \Rightarrow (d(w,w') < d(w,w")))) \tag{1}$$

And c(w) means that predicate c is true in world w. e(w) means that predicate e is true in world w. Formula (1) means that e causally depend on c iff an execution that remove both c and e are nearer to origin execution , than any executions that remove c only.

2.2 Pseudo Boolean SAT

Pseudo Boolean SAT(PBS)[14] introduce two types of new constrain into SAT:

1. Pseudo Boolean constraints of the form:$\sum c_i x_i \leq n$, with $n, c_i \in \mathbb{N}$ and $x_i \in \mathbb{B}$;
2. Pseudo Boolean optimization goal of the form $min(\sum c_i x_i)$ and $max(\sum c_i x_i)$;

PBS can efficiently handle these two types of constraints alongside CNF constraints. We use PBS to search for most similar witness.

2.3 Bounded Model Checking, Witness, and Counterexample

Bounded model checking(BMC)[16] is a technique to find bounded-length counterexamples to LTL properties. Recently, BMC has been applied with great success by formulating it as a SAT instance and solving it with efficient SAT solvers such as zchaff[20].

General discussion of BMC is fairly complex. So we refer the reader to A. Biere 's excellent paper[16] for detail of BMC. For simplicity, we only discuss invariant properties here.

Given a system with a boolean formula I representing the initial states, a boolean formula $T_i(X_i, W_i, X_{i+1})$ representing the i-step transition relation, and an invariant with a boolean formula P_k representing the failure states at step k. the length-k BMC problem is posed as a SAT instance in the following manner:

$$F = I \wedge P_k \wedge \bigwedge_{0 \leq i < k} T_i(X_i, W_i, X_{i+1})$$

For state transition path π with length k,if formula f always hold on it, then we denote it by $\pi \models_k f$.

We give our own definition of witness and counterexample below:

Definition 3 (Bounded Witness). *For LTL formula f and bound k, if state transition trace* $\pi \models_k f$*, then we call* π *the bounded witness of f.*

Definition 4 (Counterexample). *For LTL formula f and bound k, if state transition trace* $\pi \models_k \neg f$*, then we call* π *the counterexample of f.*

For Definition 3, when the bound k can be deduced from the context, we omit it and just call it witness.

3 Multiple Nearest Witnesses Effect

Shen[1,2] and A.Groce[4,9] describe error locating approach based on nearest witness searching. For counterexample C, they search for only one nearest witness W. Then starting from the difference Δ between W and C, they perform breath-first code checking to locate the actual error.

However, when we analysis experiment result of [2], we found that nearest witness is not unique. For counterexample C, assume the set of all nearest witness is $\{W_i | 0 \leq i \leq n-1\}$. And difference between W_i and C is Δ_i,distance between W_i and C is $|\Delta_i|$. Assume the actual error e belong to only one arbitrary Δ_i, and the witness obtain by [2] is W_j . In most case, $i \neq j$, that means $e \notin \Delta_i$. So algorithm of [1,2,4,9] need to perform breath-first code checking to locate the actual error.

Under certain circumstance, Δ_i is far from actual error e. Thus, the breath-first code checking must search very deeply into the code.

At the same time, if we meet with a node with large number of fan-in in breath-first code checking, we will also need to traverse a large fraction of the code.

Until now, this section is full of bad news. But we have a good news now. While analysis the result of [2], we found that: although the witness W_j obtain by [2] does not always contain actual error e, but after running witness searching algorithm iteratively for no more than 4 times, we can always find the actual error in Δ_j.

So we propose the iteratively witness searching algorithm in Sect 4. This algorithm performs fairly well in practice, and improves significantly compared to our previous work [1,2].

4 Iteratively Witness Searching Algorithm

We first introduce the overall algorithm flow in Sect 4.1, and then describe every steps of this algorithm in detail.

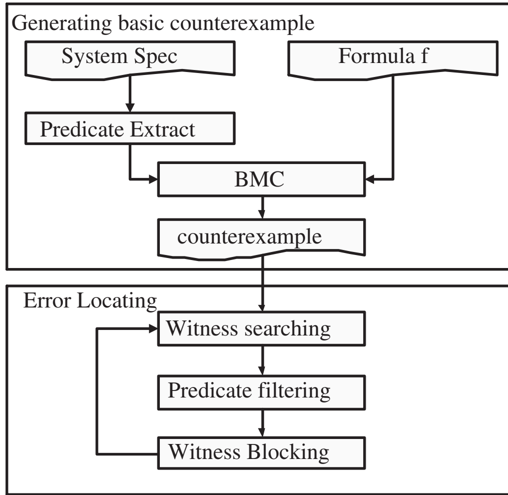

Fig. 1. Overall flow of our error locating algorithm

4.1 Overall Algorithm Flow

As shown in Figure 1, this algorithm contain 2 phase:

1. **Generation of Basic Counterexample**: This phase corresponds to traditional bounded model checking (BMC). Before performing BMC, we need to **extract predicates** first. For every control branch, we generate a predicate. This predicate takes on value 1 at its corresponding control branch only. When BMC generate the basic counterexample, it will also assign arbitrary value to these predicates. With these predicates and their assignment, we can construct the control flow of counterexample.
2. **Error Locating:** In this phase, we run following three steps iteratively:
   a) **Witness Searching:** searching for a witness W_i as similar as possible to basic counterexample, and obtain the set of predicates Δ_i that take on different value in counterexample and witness.
   b) **Predicate Filtering:** Eliminating all predicates irrelevant to violation of LTL formula f from Δ_i.
   c) **Witness Blocking:** Preventing W_i from being generated again by future iterations.

The iteration in the 2nd phase is the major difference between our approach and that of [1,2,4,9].

4.2 Predicate Extraction

In Predicate Extraction, we generate predicates for every control branch. This predicate takes on value 1 at its corresponding control branch only, and value

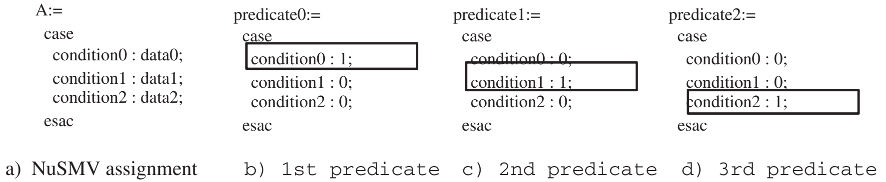

Fig. 2. NuSMV conditional assignment and all extracted predicates

0 at other control branch. Because we implement our algorithm on NuSMV, so we present example with syntax of NuSMV in Figure 2. It is straightforward to extend Predicate Extraction to other language.

The conditional assignment statement of Figure 2a contains three control branches. We insert one predicate for each control branch, as shown in Figure 2b~2d. Every predicate can take on value 1 at corresponding branch only, as shown in rectangles.

4.3 Witness Searching

After BMC generate basic counterexample $\pi 1$, it will assign value to all control flow predicates extracted in last section. With these predicates and their assignment, we can construct the control flow of basic counterexample.

In this section, we will present the algorithm that search for nearest witness. Before that, we must first define Predicate Distance Metrics.

Definition 5 (Predicate Distance). *Assume that state transition trace $\pi 1$ and $\pi 2$ contain a common predicate set TAG=$\{tag_0, tag_1, \ldots, tag_n\}$. And assignment to TAG in $\pi 1$ is $\{tag_0 = a_0, tag_1 = a_1, \ldots, tag_n = a_n\}$. Assignment to TAG in $\pi 2$ is $\{tag_0 = b_0, tag_1 = b_1, \ldots, tag_n = b_n\}$. Then define distance between $\pi 1$ and $\pi 2$ as:*

$$d(\pi 1, \pi 2) = \sum_{i=0}^{n} \Delta(i)$$

$$\textit{with } \Delta(i) = \begin{cases} 0 & a_i = b_i \\ 1 & a_i \neq b_i \end{cases}$$

With above definition of distance metrics, we define nearest witness as:

Definition 6 (Nearest Witness). *Assume $\pi 1$ is basic counterexample of LTL formula f, its bound is k. $\pi 2$ is nearest witness of $\pi 1$ iff:*

1. *$\pi 2$ is counterexample of $\pi 2 \models_k \neg f$, this means that $\pi 2$ doesn't violate formula f within bound k;*
2. *$d(\pi 1, \pi 2) \geq 1$;*
3. *For any $\pi 2'$ that satisfy entry 1 and 2, $d(\pi 1, \pi 2) \leq d(\pi 1, \pi 2')$;*
4. *For any counterexample $\pi 3$ of formula f, $d(\pi 2, \pi 3) \geq 1$.*

We will now present our witness-searching algorithm with above definition:

Algorithm 1 Nearest Witness Searching Algorithm

1. Run NuSMV command *gen_ltlspec_bmc_onepb* to generate CNF for assertion ¬f and bound k;
2. According to entry 2 of Definition 6, Encode d(π1,π2)≥1 with PBS inequality;
3. According to entry 3 of Definition 6, Encode minimization of d(π1,π2) with optimization goal of PBS;
4. Solve above three constraints with PBS to obtain π2. This will ensure that π2 is compliant to entry 1, 2 and 3 of Definition 6;
5. Run NuSMV command *gen_ltlspec_bmc_onepb* to generate CNF for assertion f and bound k;
6. According to entry 4 of Definition 6, encode d(π2,π3)≤0 with PBS inequality;
7. Solve above two constrains with PBS, and make sure it is UNSATISFIABLE. This will ensure that π2 is compliant to entry 4 of Definition 6;

With π2 obtain in above algorithm, the set of predicates that take on different value in counterexample π1 and witness π2 is denoted by:

$$Error = \{tag_i | tag_i \in TAG \quad and \quad \Delta(i) \neq 0\} \tag{2}$$

Theorem 1 show that at least one predicate in Error is the cause of ¬f.

Theorem 1. *¬f causally depend on* $\delta(\pi 1) = \bigvee_{tag_i \in Error}(tag_i == a_i)$.

Proof. First,in basic counterexample π1, for all $tag_i \in Error$, $tag_i == a_i$ always hold. so $\delta(\pi 1)$ is true in π1.

Because π1 is a counterexample, so ¬f(π1) is true.

With above conclusion, and replace ∃w' of (1) with π2, we can reduce (1) into following formula:

$$\neg\delta(\pi 2) \wedge f(\pi 2) \wedge \forall w"((\neg\delta(w") \wedge \neg f(w")) \\ \Rightarrow (d(\pi 1, \pi 2) < d(\pi 1, w"))) \tag{3}$$

Because for all $tag_i \in Error$,$\Delta(i) \neq 0$ hold, so obviously $\neg\delta(\pi 2)$ is true.

At the same time, because π2 is a witness, so f(π2) is true.

Then we can reduce (3) into following formula:

$$\forall w"((\neg\delta(w") \wedge \neg f(w")) \Rightarrow (d(\pi 1, \pi 2) < d(\pi 1, w"))) \tag{4}$$

Prove by contradiction, assume (4) is false, then there exist a w" such that the following formula hold:

$$(\neg\delta(w") \wedge \neg f(w")) \wedge (d(\pi 1, \pi 2) \geq d(\pi 1, w")) \tag{5}$$

Because ¬f (w") and f(π2) both hold, according to entry 4 of Definition 6, we can deduce that w" and π2 has different assignment to TAG, discuss in two possible case:

1. There exist a tag∈Error, such that tag(w")≠tag(π2), then tag(w")==tag(π1) must hold, which contradict with $\neg\delta(w")$;
2. There exist a tag∉Error, such that tag(w")≠tag(π2), then tag(w")≠tag(π1) and $\neg\delta(w")$ must both hold, this will lead to d(π1,π2)<d(π1,w"), which contradict with d(π1,π2)≥d(π1,w").

Therefore, Formula (5) doesn't hold, and then ¬f must causally depend on predicate $\delta(\pi 1) = \bigvee_{tag_i \in Error}(tag_i == a_i)$

4.4 Predicate Filtering

In Error given by (2), many predicate are irrelevant to the violation of LTL formula f. They are the byproduct of constructing witness π2. We need to perform a Dynamic Cone of Influence Algorithm to eliminate them from Error.

Let's first present the definition of Dynamic Dependence Set.

Definition 7 (Dynamic Dependence Set). *Assume the conditional assignment statement of variable A is shown in Figure 2a, its i-th condition formula is C_i , and its i-th data formula is D_i , then i-th dynamic dependence set of variable A is*

Dep(A,i)=$\{x|x$ *is a state variable and* x *is sub-formula of* $\bigwedge_{0\leq n\leq i-1}\neg C_n$ *or* C_i *or* $D_i\}$

With above definition, we eliminate irrelevant variable with following algorithm.

Algorithm 2 Dynamic Cone of Influence Algorithm

```
DCOI {
  N1:= {all variables of formula f at time frame k }
  do{
       N1:=DCOIRecur(N1 )
       Node:=Node ∪ N1
  }until N1==φ
  Node is the Dynamic Cone of Influence of formula f
}
DCOIRecur(C ) {
  Node:=φ
  for each v_i∈C {
       if (variable v_i is not a primary input) {
            let the n-th branch predicate of v_i is 1 in π1
            Node:=Node∪Dep(v_i ,n)
       }
  }
  return Node
}
```

Traditional **Static** Cone of Influence algorithm will generate a large node set, which contain all nodes that are connected to formula f with data dependence path.

Our Dynamic Cone of Influence Algorithm generate a much smaller node set, which contain only nodes that ACTUALLY affect f in counterexample π1. As shown in algorithm 2, the bold line state that only **Dep(**v_i **,n)** can be added into Node set.

Now, assume current iteration is the n-th iteration, after Predicate Filtering, the set of predicates that are relevant to violation of formula f is:

$$R_n = \{p|p \in Error, \text{ and } \exists n \in Node, \text{ such that p is a control predicate of variable n}\} \quad (6)$$

After iteratively running the whole algorithm multiple times, we obtain multiple R_n . Then the union of them $\cup R_n$ form a prioritized list. If n<m, $p \in R_n$ and $q \in R_m$, then priority of p is higher than q. A predicate with higher priority means that it is more likely the actual error.

4.5 Witness Blocking

To prevent the witness $\pi 2$ of current iteration from being generated again, we must add $\pi 2$ as a blocking constrain into witness searching algorithm.

We use a blocking set MASK to record $\pi 2$, and modify Algorithm 1 to prevent $\pi 2$ from being generated again. Entry 4 of Algorithm 1' is this new blocking constrain.

Algorithm 1' Modified Nearest Witness Searching Algorithm

1. Run NuSMV command *gen_ltlspec_bmc_onepb* to generate CNF for assertion ¬f and bound k;
2. According to entry 2 of Definition 6, Encode d($\pi 1$,$\pi 2$)≥1 with PBS inequality;
3. According to entry 3 of Definition 6, Encode minimization of d($\pi 1$,$\pi 2$) with optimization goal of PBS;
4. **For all witness $\pi \in MASK$, encoding d(π, $\pi 2$)>0 with PBS inequality, such that they will not be generated by current iteration;**
5. Solve above four constraints with PBS to obtain $\pi 2$. This will ensure that $\pi 2$ is compliant to entry 1, 2 and 3 of Definition 6;
6. Run NuSMV command *gen_ltlspec_bmc_onepb* to generate CNF for assertion f and bound k;
7. According to entry 4 of Definition 6, encode d($\pi 2$,$\pi 3$)≤0 with PBS inequality;
8. Solve above two constrains with PBS, and make sure it is UNSATISFIABLE. This will ensure that $\pi 2$ is compliant to entry 4 of Definition 6 ;

5 Experiment Result and Analysis

First, we briefly introduce two different score functions for evaluating error localization techniques in Sect 5.1. One for algorithm of [2] , another for algorithm of this paper. Next, we present the experiment results and analysis in Sect 5.2 and 5.3.

5.1 Score Functions for Evaluating Error Localization Techniques

To compare different error locating technique, quantifiable metrics must be built. We call these metrics as score function. Because our error locating result is different from that of [1,2,4,9], so our score function is also different from that of [1,2,4,9]. However, both functions are of the same meaning:

> After locate the actual error under guidance of error locating result, how much percentage of program statements have not being check. Obviously, a higher score means more accurately error locating.

We first introduce score function of [1,2,4,9]:

> Consider a breath-first search of the program dependence graph(PDG) starting from the set of nodes in the potential error report R. Call R a layer, BFS_0 . Then define BFS_{n+1} as a set containing BFS_n and all nodes reachable in one direct step in the PDG from BFS_n .let BFS_{k1} be the smallest layer containing at least one error node. Then the score for algorithm of [2] is $1 - \frac{|BFS_{k1}|}{|PDG|}$.

We then describe score function of this paper:

> Starting from the head of prioritized predicate list, we check each predicate to determine if it is the actual error. Assume that the actual error is contained in the result of the n-th iteration R_n ,then $\cup_{1 \le i \le n} R_i$ is the minimal set of checked predicates that contain the actual error. Then the score of our algorithm is $1 - \frac{|\cup_{1 \le i \le n} R_i|}{|PDG|}$.

With above two score functions, we can then compare the result of this paper and that of our previous work [2].

5.2 Experiment Result

The origin gigamax cache coherence protocol [17] is distributed with NuSMV[12]. We convert all its CTL assertions into LTL equivalent version such that we can check it with BMC package of NuSMV. The property used to detect errors is: G !(p0.writeable & p1.writeable). This means that it is not possible to make two caches writeable at the same time.

We insert 10 errors into it. Five of them are data flow errors. The other five are control flow errors.

The NuSMV source code contains 189 lines. After flatten there are 458 lines and 41 conditional assignments.

All experiments are performed on Pentium 3 1GHz.

As shown in Table 1, we compare result of this paper and that of [2]. The third column is the bound of basic counterexample N, so the total number of conditional assignment statement is 41*N. The 4-th column is the size of smallest layer containing at least one error node. The 5-th column is score of algorithm of [2]. the 6-th column is the run time of [2].

The number of iterations of our algorithm is shown in the 7-th column. Size of the minimal set of checked predicates that contain the actual error, is shown in the 8-th column. Score of our algorithm is shown in the 9-th column. the run time of our algorithm is shown in 10-th column.

Table 1. Experiment result of this paper and that of [2]

	error	Bound of Cex	Result of [2]			Result of this paper			
			$\|BFS_{k1}\|$	Score(%)	Time	Num of Iter	$\| \cup_{1\leq i\leq n} R_i \|$	Score(%)	Time
Data flow error	D1	6	18	92.7	19.56	3	22	91	100
	D2	4	6	96.3	16.72	2	6	96.3	
	D3	5	2	99	25.13	1	2	99	
	D4	5	14	93.1	21.92	3	11	94.6	
	D5	5	7	96.5	14.25	2	3	98.5	
Control flow error	R1	6	8	96.7	19.22	3	6	97.5	
	R2	5	24	88.3	22.86	3	17	91.7	
	R3	6	24	90.2	23.37	2	11	95.5	
	R4	2	18	78	7.12	2	5	94	
	R5	5	26	87.3	17.71	2	8	96	

5.3 Result Analysis

As shown by Table 1, all score of this paper is higher than 90%. This is significantly higher than that of [2]. From Table 1, we can conclude that:

1. All errors that have been accurately located in [2] are also accurately located in this paper.
2. All errors that are poorly located in [2], such as R2, R3, R4 and R5, are all accurately located in this paper.
3. All actual error can be located within 3 iterations.

Our new algorithm improve the accuracy of error locating in two aspect:

1. We do not need to perform breath-first code checking any more, so we avoid the impact of multiple nearest witness, which is describe in Sect 3 in detail;
2. We also avoid the impact of wide fan-in node.

6 Related Work

It is a tedious task to understand the complex counterexamples generated by model checking complex hardware system. Therefore, how to automatically extract useful information to aid the understanding of counterexample, is an area of active research.

Research works in this field can be divided into 3 categories:

6.1 Counterexample Compaction

These works focus on making the counterexample more succinct.

T.Ball[5] search for all state graph transition edges that only belong to the counterexample.

Jin,Ravi and Somenzi[3] propose a game-like explanation in which an adversary try to force the system into error. They try to partition a counterexample into two types of fragments: fate fragment that are unavoidable path leading to violation of assertion, and free will fragment that can avoid the violation of assertion.

K.Ravi[3] formulate the extraction of a succinct counterexample as the problem of finding a minimal assignment that, together with the boolean formula describing the model, implies the violation of LTL formula.

6.2 Error Locating

Error Locating approach is a much more aggressive form of Counterexample Compaction. They drop the completeness requirment, and try to find even more succinct error locating results.

In [1], Shen propose the control predicate distance metrics for the first time, and present a nearest witness-searching algorithm with this metrics.

In [2] , Shen integer predicate filtering into the framework of [1].

A.Groce[8] generate multiple similar successful and failing versions of a counterexample, and analysis their difference.

A.Groce [4,9] define "data flow distance" between two paths, and search for a witness most similar to a counterexample, then analysis their difference to locate actual error.

G.Fey[11] analysis the counterexample of equivalence checking, and generate multiple similar counterexample, then locate the actual error by analysis commonness of these counterexample.

6.3 Annotate Counterexample with Proof

Several researchers also try to explain non-linear counterexample and witness with annotated proof steps.

M.Chechik[13] generate proof for non-linear counterexample of ACTL, and then extend their approach to deal with fairness condition.

D. Peled[15] generate proof for witness of LTL formula.

K. Namjoshi[19] concentrate on generating a proof of validity for a run of global μ-calculus model checker.

Tan [18] extend Namjoshi's work to local model checking.

7 Conclusions

It is a tedious task to manually analysis counterexample of complex hardware system. We have shown how to locate the bug in counterexample generated by bounded model checker. Experiment result show that our approach is highly accurate.

Our current implementation is based on NuSMV. However, our techniques are quite general and we are porting it to Verilog.

To locate bug more accurate, we are considering using multiple assertions to locate the bug instead of "one assertion" debugging approach of this paper.

We are also considering locating bug in loop-like counterexample of liveness property.

Finally, due to the computation complexity of PBS,it is infeasible to directly locate error in large-scale concrete model, so we are considering impose abstract/refine approach into our witness searching algorithm.

Acknowledgements. We would like to thank the anonymous referees for their careful reading of the submitted version of the paper and for their comments.

References

1. ShengYu Shen , Ying Qin and SiKun Li., "Bug Localization of Hardware System with Control Flow Distance Minimization". 13th IEEE International Workshop on Logic and Synthesis (IWLS 2004), Temecula, California, USA. June 2-4, 2004 . accepted.
2. ShengYu Shen , Ying Qin and SiKun Li., "Debugging Complex Counterexample of Hardware System using Control Flow Distance Metrics".47rd IEEE Midwest Symposium on Circuits and Systems(MWSCAS 2004), Hiroshima, Japan. July 25-28, 2004 .accpeted.
3. Kavita Ravi and Fabio Somenzi. Minimal Assignments for Bounded Model Checking. Tools and Algorithms for the Construction and Analysis of Systems (TACAS), March-April 2004.
4. Alex Groce, Daniel Kroening, and Flavio Lerda. "Understanding Counterexamples with explain". Proceeding of 16th International Conference on Computer Aided Verification(CAV), July 13 – 17 , 2004
5. Thomas Ball, Mayur Naik, Sriram Rajamani."From Symptom to Cause: Localizing Errors in Counterexample Traces". In Proceedings of ACM Symposium on Principles of Programming Languages (POPL '03), New Orleans, LA, January 2003.
6. D.Lewis ."Causation". Journal of Philosophy, 70:556-567,1973.
7. M.Renieris and S .Reiss . "fault localization with nearest neighbor queries". In Automatic Software Engineering,2003
8. Alex Groce and Willem Visser. "What Went Wrong: Explaining Counterexamples." In SPIN Workshop on Model Checking of Software, pages 121–135, May 2003.
9. Alex Groce. "Error Explanation with Distance Metrics." Tools and Algorithms for the Construction and Analysis of Systems (TACAS), March-April 2004.

10. H.Jin, K.Ravi,and F.Somenzi. "Fate and free will in error traces". In Tools and Algorithms for the Construction and Analysis of Systems, page 445-458,2002.
11. Gorschwin Fey ,Rolf Drechsler. Finding Good Counter-Examples to Aid Design Verification. Proceedings of the First ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE'03),2003
12. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella. "NuSMV 2: An OpenSource Tool for Symbolic Model Checking" . In Proceeding of CAV 2002. Copenhagen, Denmark, July 27-31, 2002
13. M.Chechik and A.Gurfinkel. "Proof-like counterexamples". In Tools and Algorithms for the Construction and Analysis of Systems,page160-175,2003.
14. F.Aloul, A.Ramani, I.Markov, and K.Sakallah ."PBS: A Backtrack-Search Pseudo-Boolean Solver and Optimizer".Symposium on the Theory and Applications of Satisfiability Testing (SAT),Ohio,pp.346-353, 2002
15. D. Peled, A. Pnueli, and L. Zuck. From falsification to verification. In FST&TCS, volume 2245 of LNCS. Springer Verlag, 2001.
16. A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, Y. Zhu . "Symbolic Model Checking using SAT procedures instead of BDDs",Proceedings of Design Automation Conference (DAC'99)
17. K. L. McMillan and J. Schwalbe, "Formal verification of the Encore Gigamax cache consistency protocols", Int. Symp. on Shared Memory Multiprocessors, Tokyo, Japan, 2-4 April 1991, pp. 242-51
18. L. Tan and R. Cleaveland. Evidence-Based Model Checking. In Proceedings of 14th Conference on Computer-Aided Verification (CAV'02), volume 2404 of LNCS, pages 455–470, Copenhagen, Denmark, July 2002. Springer-Verlag.
19. K. Namjoshi. Certifying Model Checkers. In Proceedings of 13th International Conference on Computer Aided Verification (CAV'01), volume 2102 of LNCS. Springer Verlag, 2001.
20. M. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In Proceedings of the Design Automation Conference, pages 530-535,Las Vegas, NV, June 2001.