Transition-Based Coverage Estimation for Symbolic Model Checking

Xingwen Xu Shinji Kimura

Graduate School of IPS
Waseda University
2-7 Hibikino, 808-0135 Japan

Kazunari Horikawa Takehiko Tsuchiya

System LSI Design Division
Toshiba Corporation
Sawasaki, 210-8520 Japan

Abstract— Lack of complete formal specification is one of the major obstacles for the deployment of model checking. Coverage estimation addresses this issue by revealing the unverified part of the design according to the specified properties. In this paper we propose a new transition-based coverage metric to evaluate the completeness of properties for symbolic model checking. It is more comprehensive and accurate than the existing coverage metrics for model checking. An efficient symbolic algorithm is presented for computing the transition coverage for a subset of ACTL. Our coverage estimator has been applied to the model checking of a cache coherence protocol. We uncovered several coverage holes including one that eventually led to the discovery of a design bug.

I. Introduction

Model checking [1] proves whether a system satisfies a set of properties under all possible input sequences. However, to model-check a complex design, it is very hard to determine whether sufficient properties have been specified or not [5, 13]. The completeness of formal specification needs to be evaluated for ensuring that there is no unknown behavior in the implementation, assuming the unknown behaviors often contain design bugs. Coverage estimation for model checking compares the given specification with a given implementation and deduces the parts of the implementation that are not covered by the specification. Additional properties then should be specified to close such coverage gap. Assisting the model checking process with coverage estimation can achieve higher degree of confidence in the verification results [10].

One of the most popular coverage methods for model checking is a state-based coverage metric based on state perturbation [2, 3]. Informally, a state is covered by a verified property with respect to an observed signal if changing the value of the signal in that state will cause the property to fail. The accuracy of the coverage computation algorithm is improved in [3]. The state coverage metric has successfully uncovered several meaningful coverage holes in real-word model checking projects [2]. However, one major limitation of the state coverage metric is that it is based on states, not transitions or pathes. A state may be reached via several transitions. Property verification over any of those transitions will cover that state. Consequently the state metric leaves quite a large portion of the design's behaviors unchecked. since design errors usually creep in on the transitions [15], the limitation of the state metric might leave such bugs escaped.

To provide more comprehensive coverage analysis, a transition traversal coverage method is proposed in [4]. Transitions of FSM are covered if they are traversed by verified properties according to the semantics of CTL. The authors provide a novel symbolic approach to compute the transition coverage. However, since the transition traversal dose not consider the signals' values, transition traversal coverage can not precisely reflect the completeness of properties.

Other mutation coverage metrics for model checking are introduced in [6, 7], including metrics based on omitting or replacing transitions (or paths) of the finite state machine. However, the practicality of these metrics has not been full established. Besides perturbations on FSM, the high-level fault model is introduced for generating the perturbed implementation in coverage method of [8]. Farn et. al. also have presented a numerical coverage estimation for the symbolic simulation of real-time systems focusing on safety analysis [9].

In this paper, we propose a new transition-based coverage metric for symbolic model checking based on a novel transition perturbation model. The transition perturbation model is able to pinpoint the transition through which the value of selected observed signal is checked by properties. The transition-based coverage is much more comprehensive than the existing state coverage since the coverage space expands from all states to all transitions of FSM. It avoids the false sense of completeness by the high coverage in the transition traversal method by considering perturbation. We present an efficient symbolic algorithm for computing the transition coverage for a subset of ACTL. The algorithm has the same order of complexity as a model checking algorithm. We have integrated it with the model checking engine of VIS2.0 [14]. The coverage estimator has been applied to the model checking of a cache coherence protocol. We discovered several coverage holes including one that eventually led to the discovery of a design bug. And the computation overhead is less than 20% of the plain model checking in our experiments. We also compare the coverage results and computation time with the state coverage method to demonstrate the advantages of the proposed method.

The rest of the paper is organized as follows. Section II presents the preliminaries including the state coverage

0-7803-9451-8/06/$20.00

metric and the transition representation. The transition coverage metric and symbolic computation algorithm is presented in Section III. In section IV, we discuss our experimental results and Section V concludes this paper.

II. Preliminaries

Let AP be a set of atomic propositions. A Kripke structure over AP is a four tuple $K =< S, S_0, R, L >$, where S is a finite set of states, $S_0 \subseteq S$ is the set of initial states, $R \subseteq S \times S$ is a complete transition relation, $L : S \rightarrow 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state. Any atomic proposition in A is a CTL formula, and if φ and ψ are CTL formula, then so are: $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $AX\varphi$ (φ holds at the next time instant), $A\varphi U\psi$ (φ holds until ψ holds), and $AG\varphi$ (φ holds henceforth). The semantics of other CTL operators like $A\varphi R\psi$ can be found in [1].

The state coverage metric is defined as follows [2]:

Definition 1 : *Given a Kripke structure $K =< S, S_0, R, L >$ over AP, a state $s \in S$, and an atomic proposition $q \in AP$. The perturbed Kripke structure for s with respect to q (observed signal) is $K_s^q =< S, S_0, R, L_s^q >$, where for each state $t \in S$:*

$$L_s^q(t) = \begin{cases} L(t) & \text{if } t \neq s \\ L(s) \setminus \{q\} & \text{if } t = s \text{ and } \{q\} \in L(t) \\ L(s) \cup \{q\} & \text{if } t = s \text{ and } \{q\} \notin L(t) \end{cases}$$

Definition 2 : Given a property f and a Kripke structure K such that $K \models f$, the set of covered states $C \subseteq S$ by f with respect to q satisfies the following condition for any state $s \in S$: $(K_s^q \not\models f) \Leftrightarrow (s \in C)$.

A sequential circuit is usually modeled as a finite state machine (FSM) where input propositions are labeled on the transition edges. We consider the transitions in the circuit FSM when talking about transition coverage. On the other hand, model checking is usually performed on the Kripke structure. Traditionally, a circuit FSM is first translated to a Kripke structure for model checking. For a FSM (Mealy Type) $M=<I, O, S, \delta, \lambda, S_0>$, where S and I are the state space and the input space; δ and λ are the state transition function and the output function, the corresponding Kripke structure can be derived as $K=<S \times I, S_0 \times I, R, L>$, where for any $s, s' \in S$ and any $i, i' \in I$:

- $(< s, i >, < s', i' >) \in R$ iff $\delta(s, i) = s'$,
- $L(< s, i >) = i \cup s \cup \lambda(s, i)$.

For example, the FSM of a simple modulo-3 counter in Fig. 1 is translated to the Kripke structure shown in Fig. 2. From the translation, it is observed that each transition $\delta(s, i) \rightarrow s'$ in the circuit FSM corresponds to a state $< s, i >$ in the Kripke structure one-by-one. In the example, the state $S10$ in the Kripke structure represents the transition from $S1$ to $S2$ in the FSM. As a result, we can evaluate the transition coverage of the circuit FSM based on the states of the Kripke structure.

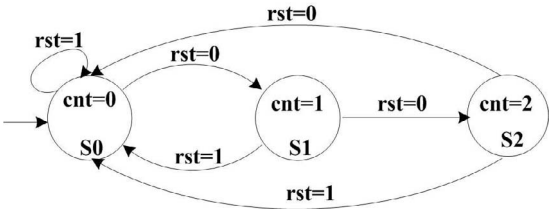

Fig. 1. The FSM of a modulo-3 counter.

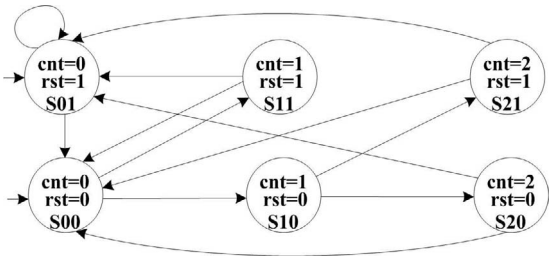

Fig. 2. The Kripke structure model of the module-3 counter.

III. Transition Coverage Estimation

In this section, we first talk about the basic idea of the transition-based coverage method, then we introduce our new transition perturbation model and define the transition coverage metric. We also present a symbolic algorithm to compute the transition coverage for a subset of ACTL.

A. Transition-Based Coverage Method

For a circuit FSM, we can consider two types of coverage metrics: the state coverage and the transition coverage. If the set of states of the FSM is S, then the state coverage space is just S. Deep-hidden design bugs usually creep in on transitions and state-based coverage method is not sufficient to clear such bugs. On the other hand, the transition coverage space is $S \times I$, where I is the set of inputs. Note that by enhancing the coverage space, we can do more precise coverage analysis and higher confidence in the correctness can be achieved [11, 12, 15]. A basic transition-based coverage method for model checking is proposed in [4], where covered transitions are defined as traversed by CTL operators. In this paper, we combine the transition traversal coverage and the state-based one by introducing transition perturbation.

Since the transition coverage method extends the coverage space to $S \times I$, more computation resources might be required. Such drawback should be treated as the tradeoff between completeness and computation efficiency.

B. Coverage Metric Based on Transition Perturbation

A CTL formula usually specifies how particular states should be reached through transitions and the correctness conditions for certain circuit signals on the reached states. The two factors answer the coverage question of what has been verified by a property. In order to catch the verification intent of formal properties on both states and transitions, we propose a novel transition perturbation model. The model is to pinpoint the transition through

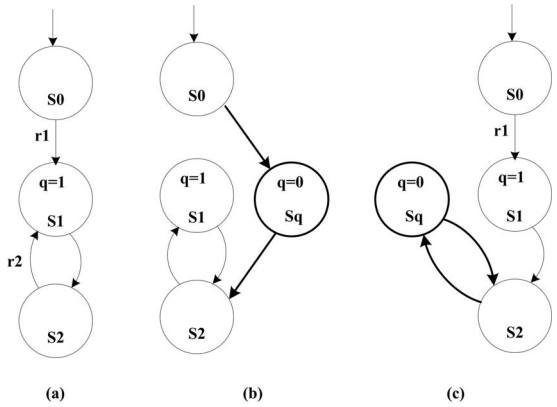

Fig. 3. The transition perturbation model for FSM.

which the correctness condition for the selected observed signal is checked.

We show a simple example to illustrate how a perturbed FSM is constructed. Fig. 3(a) is a simple FSM. To generate a perturbed FSM for transition $r1 = (S0, S1)$ for the selected observed signal q, first a new state Sq is added on which the value of q is different with the state $S1$; then the original transition $r1$ is redirected to the state Sq; finally all transitions starting from the state $S1$ are copied to the state Sq. The perturbed FSM is shown in Fig. 3(b). In the same way, the perturbed FSM for $r2$ is shown in Fig. 3(c).

In the transition perturbation model, there is only one transition reaching at the state where the value of the observed signal q is deliberately changed. On the other hand, all other transition sequences of the original model are maintained. As a result, a property will get failed if and only if the property traverses through the transition to check the value of the observed signal, which provide coverage information for both the state transition and the correctness condition of signals.

Informally, a transition of FSM is covered for the selected observed signal if a proven property gets failed on the perturbed FSM. As in the example of Fig. 3, two properties $AX(q = 1)$ and $AX(AX(AX(q = 1)))$ are satisfied by the original model (Fig. 3(a)). Because $AX(q = 1)$ is no longer satisfied by the model with perturbation on $r1$ (Fig. 3(b)), transition $r1$ is covered by this property with respect to signal q. Similarly, transition $r2$ is covered by $AX(AX(AX(q = 1)))$. According to the state coverage metric, both properties cover state $S1$. With our new transition coverage metric, two transitions reaching at state $S1$ are covered by different properties.

Since model checking is actually performed on the Kripke structure, we give a formal definition of perturbation model and transition coverage metric on the Kripke structure according to the relationship between the FSM and its corresponding Kripke structure (Section II).

Definition 3: *Given a Kripke structure* $K =< S, S_0, R, L >$*, a state* $r_i \in S$ *(it actually represents a transition of the circuit FSM), and an atomic proposition (observed signal)* $q \in AP$*. The perturbed Kripke structure on* r_i *for* q *is* $K_r^q =< S_r^q, S_0, R_r^q, L_r^q >$*, where* $S_r^q = S \cup S^q$*,* S^q *is a new state set and for each state* r_j *with* $(r_i, r_j) \in R$*, we add a new state* r_j^q *in* S^q*.*

For each state $t \in S_r^q$*:*

$$L_r^q(t) = \begin{cases} L(t) & if\ t \notin S^q; \\ L(r_j) \setminus \{q\} & if\ t = r_j^q\ and\ \{q\} \in L(r_j); \\ L(r_j) \cup \{q\} & if\ t = r_j^q\ and\ \{q\} \notin L(r_j). \end{cases}$$

For each state pair (t_i, t_j)*,* $t_i \in S_r^q$*,* $t_j \in S_r^q$*:*

$$(t_i, t_j) \in R_r^q \Leftrightarrow \begin{cases} (r_j, t_j) \in R & if\ t_i = r_j^q; \\ true & if\ t_i = r_i\ and\ t_j = r_j^q; \\ false & if\ t_i = r_i\ and\ t_j = r_j; \\ (t_i, t_j) \in R & otherwise. \end{cases}$$

Definition 4: *Given a property* f *and a Kripke structure* K *such that* $K \models f$*. The set of covered transitions* T *by* f *for the selected observed signal* q *is defined as: for any state* $r \in S$*,* $(K_r^q \not\models f) \Leftrightarrow (r \in T)$*.*

$$coverage = \frac{number\ of\ covered\ transitions}{number\ of\ reachable\ transitions}$$

The definition is based on the states of the Kripke structure, but it is different with the state-based coverage method [2]. In the counter example (Fig. 2), consider the property $AG(cnt = 2 \rightarrow AX(cnt = 0))$ and select cnt as observed signal. According to the state coverage method, states {$S01$,$S00$} are covered, which correspond to the state $S0$ in the FSM (Fig. 1). However, there are 4 transitions arriving at $S0$, but the state coverage metric dose not distinguish which transitions are really concerned by the property. Differently, according to the transition coverage metric, states {S21,S20} are covered by the property, which correspond to transitions from $S2$ under input $rst = 1$ and $rst = 0$ in the FSM. The other two transitions arriving at $S0$ in the FSM are not covered. Conceptually, states of the Kripke structure are interpreted as transitions of the circuit FSM in our method, but they are just considered as static states of FSM in the state-based metric.

C. Coverage Computation

The transition coverage metric is general for any property specification language. However, the set of covered transitions may not be easily computed. In this paper, we present a symbolic algorithm which is based on fix point computation and binary decision diagrams (BDDs) [1] to compute the transition coverage for a subset of ACTL.

Definition 5: T*he set of formulae acceptable to our algorithm is defined as: if* b *is a propositional formula, then* b *is acceptable; if* f *and* g *are acceptable, then so are* AXf*,* AGf*,* $A(fUg)$*,*$A(fRg)$*,*$f \wedge g$*,*$b \rightarrow g$*.*

Note that AFf can be represented as $A(trueUf)$, so we do not treat it separately. According to industrial experience, the set of formulae is sufficiently expressive to specify most desirable properties for sequential logic circuits.

Fig. 4 is the main function of our coverage computation algorithm $Cov(\varphi, S)$. The algorithm is performed on the Kripke structure of a circuit design. The input

```
Cov(φ, S){
  if (S == empty) return empty;
  if (φ is propositional) return empty;
  switch (φ)
  case f ∧ g : result = Cov(f, S) ∪ Cov(g, S);
  case b → g : result = Cov(g, S ∩ Sat(b));
  case AGf : result = Cov(f, Rch(S));
  caseAXf :
    cf = Chk(f, Fwd(S));
    r1 = Bwd(cf) ∩ S; r2 = Cov(f, Fwd(S));
    result = r1 ∪ r2;
  casefUg :
    fTrv = empty; gTrv = empty;
    gS = Sat(g); doS = S;
    do{
      gTrv = gTrv ∪ (doS ∩ gS);
      fS = doS \ gS;
      if(fS ≠ empty){
        fTrv = fTrv ∪ fS;
        doS = Fwd(fS) \ (fTrv ∪ gTrv);
      }
    }while(fS ≠ empty & doS ≠ empty));
    c1 = Chk(f, fTrv); c2 = Chk(g, gTrv);
    r1 = fTrv ∩ Bwd(c1); r2 = fTrv ∩ Bwd(c2);
    r3 = Cov(f, fTrv); r4 = Cov(g, gTrv);
    result = r1 ∪ r2 ∪ r3 ∪ r4;
  casefRg : //similar to fUg
  default : Unacceptable formula;
  return result;
}
```

Fig. 4. Coverage computation algorithm: the main function $Cov(\varphi, S)$.

is a property and a set of states; the output is a set of covered states which correspond to the transitions of the circuit FSM. For the top-level run, S is the initial state set, and φ is a verified property. As the algorithm proceeds, sub-formulae and states on the traversing pathes are recursively processed by the algorithm. The algorithm finally returns the covered transitions with respect to the given observed signal q. $Chk(\varphi, S)$ is a sub-function which extracts correctness conditions of each sub-formula and checks their dependency on the observed signal in different states, as shown in Fig. 5. Other predicates used in the algorithm is explained as follows:

$Fwd(S)$: the forward image of a state set S;
$Bwd(S)$: the backward image of a state set S;
$Rch(S)$: all reachable states from a state set S;
$Sat(f)$: the set of states satisfying f;
$\setminus$: the set minus.

The idea of our algorithm is to perform transition traversal for a property, and while traversing a transition, we extract the correctness conditions from the property on its destination state, the transition is identified as covered if the correctness condition depends on the value of the observed signal.

```
Chk(φ, S){
  if (S == empty) return empty;
  if (φ is propositional)
    result = S ∩ Sat(¬φ|_{q→¬q});
    return result;
  switch (φ)
  case f ∧ g : result = Chk(f, S) ∪ Chk(g, S);
  case b → g : result = Chk(g, S ∩ Sat(b));
  case AGf : result = Chk(f, S);
  caseAXf : result = empty;
  casefUg :
    r1 = Chk(g, S ∩ Sat(g));
    r2 = Chk(f, S \ Sat(g));
    result = r1 ∪ r2;
  casefRg : //similar to fUg
  default : Unacceptable formula;
  return result;
}
```

Fig. 5. Coverage computation algorithm: the sub function $Chk(\varphi, S)$.

In our algorithm, we care more about the verification intent of a property formula. As for formula AGf, the set of reachable states are assumed and there are no verification intent for transitions from the initial state to all other states. Thus, in our algorithm, we do not perform traversing for AG but directly compute the coverage of its sub-formula f with all reachable states.

Compared with the state coverage estimation algorithm [2], the main extra computation in our algorithm comes from the backward image computation. This will not increase much computation cost since it is only performed on certain state sets. The algorithm has the same order of complexity as a model checking algorithm, which is exponential in the worst case. The actually computation overhead is quite small in our experiments when we integrate the coverage estimation with model checking itself.

IV. Experimental Results

We have implemented the transition coverage estimator by extending the state-of-the-art model checker VIS2.0 [14]. We integrate the coverage estimation with model checking to save computation cost. The Kripke structure constructed for model checking is also used for coverage estimation. The model checking results for sub-formulas can be used in the coverage estimation. For comparison purpose, we have also implemented the state coverage method [2]. We have applied the coverage method to the model checking of a cache coherence protocol.

One of the most successful application domains for model checking has been multiprocessor cache coherence protocols. This application is commercially very important since almost all high-end servers are now cache-coherence multiprocessors. In our work, we apply coverage metrics to assist the model checking process for a full-

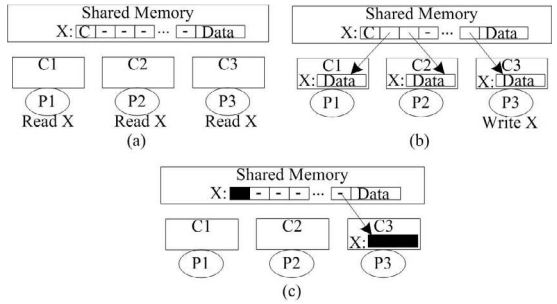

Fig. 6. The three different states in a full-map directory.

map directory-based cache coherence protocol [16]. The protocol uses directory entries with one bit per processor and dirty bit. Each bit represents the status of the block in the corresponding processors's cache (present or absent). If the dirty bit is set, then one and only one processor's bit is set, and that processor has permission to write into the block. Each cache block may be in one of the three states: INVALID, SHARED, or EXCLUSIVE. Fig. 6 illustrates three different states of a full-map directory. In (a), location X is missing in all of the caches; (b) results from three caches requesting copies of location X; (c) results from cache C3 requesting writing to location X. Note the dirty-bit is set to clean in (a) and (b), and to dirty in (c). To simply the complexity, we first configure the protocol model with two processors and two memory entries. The cache for each processor only contains one data block.

The cache protocol should keep the block states in the memory directory and those in the caches consistent. We use invariant properties to check the state consistency. For example, the following two properties check the SHARED and EXCLUSIVE states for cache C1 with address 0:

$$AG(c1\_sta = SHA * c1\_add = 0 \leftrightarrow dirty[0] = 0 * share\_c1[0] = 1); \quad (1)$$

$$AG(c1\_sta = EXC * c1\_add = 0 \leftrightarrow dirty[0] = 1 * share\_c1[0] = 1); \quad (2)$$

Since invariant properties do not contribute transition coverage, we estimate state coverage by selecting the dirty bit $dirty[0]$ as the observed signal. The coverage results show certain states with clean dirty-bit are not covered. After analysis, we found a design error in our initial model. When a cache is full, a write-back operation should be performed when the cache gets a read or write miss. After fixing the bug, additional properties are added to cover all states. For example, dirty bit for address 0 should be clean when cache C1 is in INVALID state and cache C2 is not for address 0:

$$AG(c1\_sta = INV * c2\_add \neq 0 \rightarrow dirty[0] = 0) \quad (3)$$

At this point, the state coverage metric is of great value since we are mainly concerned for the state consistency. However, we do not consider the state transition behaviors of the protocol.The cache protocol should ensure sequential consistency. Intuitively, each read should return the value of the most recent write to the same memory location.

TABLE I
State and transition coverage results.

Observed Signal	Number of Properties	State Coverage	Transition Coverage
ack1	9	100%	100%
c1_o	7	100%	99.72%
dirty[0]	3	100%	95.49%

We select the data output of cache C1 $c1\_o$ as the observed signal and seven properties are specified. For instance, if any processor reads value 0 from memory address 0, then all read at address 0 should return 0 unless there is a write to address 0 with value 1. This property can be expressed by CTL formula like:

$$AG(r\_a0\_v0\_ack \rightarrow A((w\_a0\_v1)R (c1\_r\_a0\_ack \rightarrow c1\_o = 0))); \quad (4)$$

Three properties are specified for the observed signal $dirty[0]$. For example, the dirty-bit should remain clean until there is a write for its address. And if cache C1 is in EXCLUSIVE state, the dirty-bit for the same address as cache C1 should remain dirty until cache C1 is requested to write back or invalidate its data.

$$AG(dirty[0] = 0 \rightarrow A(wri\_add0)R(dirty[0] = 0)); \quad (5)$$

$$AG(c1\_sta = EXC * c1\_add = 0 \rightarrow A((wri\_bak1 = 1 + inva1 = 1)R(dirty[0] = 1))); \quad (6)$$

Another 9 properties are specified when taking the acknowledge for processor P1 $ack1$ as the observed signal.

Tab. I compares the state and transition coverage results. All states are covered by these properties but the transition coverage results discover certain conditions which are not considered by these properties.

When a write operation by a processor begins, the read operation by another processor may have not finished. And the outcome of such a read is not verified. The state coverage metric is unable to discover this coverage hole because this state in which the read operation finishes is covered through other transitions. Another corner case is that, when processor P1 begins to write, cache C2 is asked to invalidate its content, then cache C1 writes back its content to the memory because it is full. After that the new value is written to cache C1. The uncovered transition is illustrated in Fig. 7 (the bold edge). The state metric is unable to detect this coverage hole because a read operation by processor P2 will also trigger P1 to write back. The invalidation–write-back sequence is actually a design error since it will degrade performance. A superior sequence is write-back–invalidation for such a condition.

We also compare the computation time of transition and state coverage estimation. We add two other differ-

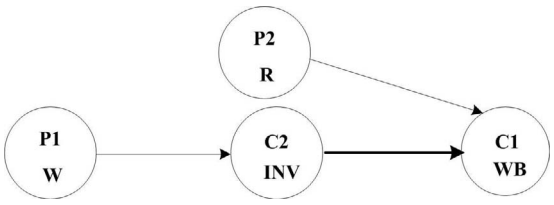

Fig. 7. One uncovered transition in the cache protocol.

TABLE II
Comparison of computation time.

Protocol Configure	T1 (Seconds)	T2 (Seconds)	T3 (Seconds)
2p2m	51	65	66
3p2m	3956	4065	4071
2p4m	9938	10444	12592

ent types of configuration as two processor with 4 memory entries and 3 processors with 2 memory entries. The computation time for the 7 properties for observed signal c1_o is shown in Tab. II, where T1 is the time of plain model checking; T2 and T3 are respectively the times of model checking along with state and transition coverage estimation. The CPU times are measured on a IBM IntelliStation Z-Pro with 3.0GHZ CPU and 2.3GB RAM. The results confirm our discussion in section III that the computation costs are similar for the transition and state coverage estimation. The computation overhead of coverage estimation compared to plain model checking is also shown in Tab. II, which is less than 20% of the plain model checking.

V. Conclusions

We have proposed a novel transition perturbation model, and based on this model, we have defined a new transition-based coverage metric for model checking to evaluate the completeness of properties. An efficient symbolic algorithm has been presented to compute the transition coverage for a subset of ACTL. The implemented coverage estimator has been applied to the model checking of a cache coherence protocol. The experiment results have confirmed that the proposed method can identify critical coverage holes which may escape the state-based coverage estimation, while the computation overhead is minor compared with plain model checking. Our future work is to develop automatic techniques for analyzing the coverage results so as to fill coverage holes efficiently.

References

[1] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[2] Y. Hoskote, T. Kam, Pei-Hsin Ho, and Xudong Zhao. "Coverage estimation for symbolic model checking". In *Proceedings of the 36th Design Automation Conference (DAC)*, page 300-305, 1999.

[3] N. Jayakumar, M. Purandare, and F. Somenzi. "Do's and don'ts of CTL state coverage estimation". In *Proceedings of the 40th Design Automation Conference (DAC)*, page 292-295, 2003.

[4] X. Xu, S. Kimura, K. Horikawa, and T. Tsuchiya. "Transition Traversal Coverage Estimation for Symbolic Model Checking". In *Proceedings of the 3rd ACM/IEEE internatinoal Confernece on Formal Methods and Models for Co-Design (MEMOCODE)*, page 259-260, 2005.

[5] S. Katz, O. Grumberg, and D. Geist. "Have I written enough properties? - A method of comparison between specification and implementation". In *Proceedings of the 10th ACM Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARM)*, page 280-297, 1999.

[6] H. Chockler, O. Kupferman, and M.Y. Vardi. "Coverage Metrics for Formal Verification". In *Proceedings of 12th ACM Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARM)*, page 111-125, 2003.

[7] H. Chockler, O. Kupferman, R. Kurshan and M.Y. Vardi. "A Practical Approach to Coverage in Model Checking". In *Proceedings of the 2001 International Conference on Computer Aided Verification (CAV)*, page 66-78, 2001.

[8] F. Fummi, G. Pravadelli, A. Fedeli, U. Rossi, and F. Toto. "On the use of a high-level fault model to check properties incompleteness". In *Proceedings of the 1st ACM/IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE)*, pages 145-152, 2003.

[9] F. Wang, G.-D. Huang, F. Yu. "Numerical Coverage Estimation for the symbolic simulation of real-time systems". FORTE 2003, LNCS 2767, pages 160-176, 2003.

[10] S. Tasiran, and K. Keutzer. "Coverage metrics for functional validation of hardware designs". In *IEEE Journals of Design & Test of Computers*, Volume 18(4), pages 36-45, 2001.

[11] Y. Hoskote, D. Moundanos, and J.A. Abraham. "Automatic extraction of the control flow machine and application to evaluating coverage of verification vectors". In *Proceedings of the 1995 IEEE Internatinal Conference on Computer Design VLSI in Computers and Processors (ICCD)*, page 532-537, 1995.

[12] R.C. Ho, C. Han Yang, M.A. Horowitz, and D.L. Dill. "Architecture validation for processors". In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, page 404-413, 1995.

[13] S. Das, et al.. "Formal Verification Coverage: Computing the Coverage Gap between Temoral Specifications". In *Proceedings of the IEEE/ACM International Conference on Compute Aided Design (ICCAD)*, page 198-203, 2004.

[14] R.K. Brayton, et al.. "VIS : A System for Verification and Synthesis". In *Proceedings of the 1996 International Conference on Computer Aided Verification (CAV)*, page 428-432, 1996.

[15] A. Gupta, S. Malik, and P. Ashar. "Toward Formalizing A Validation Methodology Using Simulation Coverage". In *Proceedings of the 34th Design Automation Conference (DAC)*, page 740-745, 1997.

[16] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal. "Directory-based cache coherence in large-scale multiprocessors Computer". In *Computer*, IEEE Computer Society, Volume 23, Issue 6, page 49-58,1999.