Noname manuscript No.
(will be inserted by the editor)

Learning-Oriented Property Decomposition for Automated Generation of Directed Tests

Mingsong Chen · Xiaoke Qin · Prabhat Mishra

Received: date / Accepted: date

Abstract SAT-based Bounded Model Checking (BMC) is promising for automated generation of directed tests. Due to the state space explosion problem, SAT-based BMC is unsuitable to handle complex properties with large SAT instances or large bounds. In this paper, we propose a framework to automatically scale down the SAT falsification complexity by utilizing the decision ordering based learning from decomposed sub-properties. Our framework makes three important contributions: i) it proposes learning-oriented decomposition techniques for complex property falsification, ii) it proposes an efficient approach to accelerate the complex property falsification using the learning from decomposed sub-properties, and iii) it combines the advantages of both property decomposition and property clustering to reduce the overall test generation time. The experimental results using both software and hardware benchmarks demonstrate the effectiveness of our framework.

Keywords Property decomposition · Bounded model checking · SAT · Test generation

1 Introduction

Boolean Satisfiability (SAT) based Bounded Model Checking [3] is promising for automated generation of directed tests. The basic idea is to restrict the search range and transform the test generation problem into a SAT problem. During test generation using SAT-based BMC, the design specification is translated to a formal model (e.g., SMV [12]) and the negation of coverage requirement (e.g., functional fault models [17,23]) is translated to a set of safety properties in the form of temporal logic. Given a formal model M, a safety LTL property p, and a bound k, SAT-based BMC encodes the state search problem by unrolling the model k times using the following propositional Boolean formula.

$$BMC(M,p,k) = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i,s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p(s_i) \quad (1)$$

It consists of three parts: i) $I(s_0)$ indicates the system initial state, ii) $T(s_i,s_{i+1})$ presents the state transition from state s_i to state s_{i+1}, and iii) $\neg p(s_i)$ checks whether property p is violated in the state s_i. This formula is transformed to Conjunctive Norm Form (CNF) and solved by SAT solvers. Semantically, if there is no satisfying assignment for Equation (1), it means that the property holds for the design within bound k, written $M \models_k p$. Otherwise, the property p will not hold in M, written $M \not\models p$. The counterexample (i.e., a satisfying assignment for $BMC(M,p,k)$) can be used as a test to check the functional scenario described by p [11].

In practice, most validation flows need to check various functional scenarios involving a large number of properties. To improve the overall directed test generation performance, property clustering [6] and property learning approaches [9] are proposed. Since learning can be reused across similar properties, many repeated validation efforts can be efficiently avoided. It is important to note that the checking time of the first property significantly impacts (as demonstrated in Section VI) the overall test generation time of a cluster of similar properties. If the checking time for the first property is too long, it will be detrimental for the overall test generation time even if the other properties can efficiently utilize

Mingsong Chen
Shanghai Key Lab of Trustworthy Computing,
East China Normal University, Shanghai, 200062, China
Tel.: +86-21-62224387
Fax: +86-21-62235255
E-mail: mschen@sei.ecnu.edu.cn

Xiaoke Qin and Prabhat Mishra
Department of Computer & Information Science & Engineering,
University of Florida, Gainesville FL 32611, USA
Tel.: +1-352-5051880
Fax: +1-352-3921220
E-mail: {xqin, prabhat}@cise.ufl.edu

the learning from the first property. Therefore, it is crucial to reduce the test generation time of the first property as well as utilize the learning information across similar properties to reduce the overall validation effort.

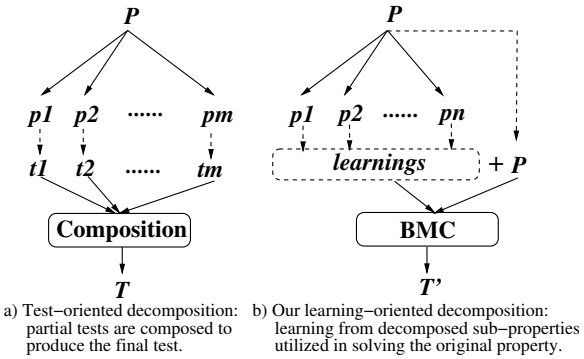

Fig. 1 Two property decomposition techniques

Due to random search in conventional SAT solvers, the test generation time for the base property (first one to be solved in a cluster) can be prohibitively long in the presence of complex design and properties. To address this issue, Koo and Mishra proposed a property decomposition technique [19] as shown in Figure 1a. The basic idea of their method is to decompose a complex property into several simple sub-properties based on design structure. After generating tests for each sub-property, this approach composes the sub tests to form a test for the original property. Since only part of a design is involved in the property falsification, the test generation time of sub-properties can be significantly smaller than that of the original property. However, the composition of sub tests cannot be fully automated when there are complex interactions between components that may lead to conflicting assignments in sub tests. As a result, composition of sub tests may require many iterations and/or manual intervention.

This paper proposes a novel learning-oriented property decomposition approach, which supports both efficient identification of property similarity and automated generation of directed tests. Unlike the test oriented method in [19], our approach (as shown in Figure 1b) uses decision ordering based learning derived from the checking of decomposed sub-properties. Such learning can be used to dramatically accelerate the original property falsification. It is important to note that our approach is based on learning from decomposed sub-properties, thus it can be fully automated. Since our target is to reduce the overall test generation time for a cluster of complex properties, our approach mainly addresses the following three issues.

1. How to effectively decompose a complex property to scale down the property checking complexity as well as achieve profitable learning? This paper proposes profitable spatial, temporal and hybrid decomposition techniques to exploit decision ordering based learning for complex property falsification.
2. How to automatically utilize learning derived from decomposed sub-properties to reduce test generation time of the original complex property? This paper proposes a method to predict the decision ordering for the complex property checking based on the results of sub-properties.
3. How to reduce the overall time (learning derivation time and test generation time) when checking multiple complex properties? This paper proposes two similarity metrics for property clustering to enable knowledge sharing between a cluster of similar properties.

The rest of the paper is organized as follows. Section 2 and Section 3 present related work and background of automated test generation using SAT. Section 4 proposes our learning-oriented property decomposition approach. Section 5 proposes an efficient test generation method based on the decomposition information. Section 6 presents the experimental results. Finally, Section 7 concludes the paper.

2 Related Work

Model checking based property falsification techniques are promising for automated generation of directed tests [11]. However, due to "state space explosion problem", Binary Decision Diagram (BDD) based unbounded model checking approaches [12] cannot handle complex designs. SAT-based BMC [3] is a promising alternative to alleviate the capacity and productivity restrictions over unbounded model checking for many real designs [1]. It has been widely used for faster bug localization during design verification [5].

The complexity of both designs and properties determines the model checking performance. To reduce the checking time, various techniques are proposed. Bjesse and Kukula [4] proposed a method based on the counterexample guided abstraction refinement. The basic idea is to generate *stepping stones* from the abstracted system and to divide the search into a number of short searches. Amla et al. [2] introduced a decompositional algorithm for model checking of timing diagram specifications. Such decompositions can not only promote the model checking performance, but also enable the composition of new regular time graphs hierarchically. To advance the applicability of model checking tools to realistic applications, Meyer et al. [24] proposed an automatic decomposition of duration calculus specification into sub-properties that can be verified quickly and independently. Koo and Mishra [19] proposed a framework that can decompose a complex property into several simple sub-properties. By checking all the sub-properties and combining corresponding sub tests, this method can obtain a counterexample for the original complex property. Although these decomposition techniques are time-efficient, it is difficult to automate their composition procedure in many scenarios.

Sharing learning across properties can improve the overall test generation performance since repeated validation efforts can be avoided. Based on the observation that conflict clauses can be replicated and forwarded as variable assignment constraints, various incremental SAT solvers [29,18] were developed. In the context of Automatic Test Pattern Generation (ATPG), an incremental SAT framework [30,16] which stores learned information was proposed. Here, learned information is used during the search by targeting similar faults that share a cone of influence. However, these approaches rely on structural analysis of the circuit and do not decompose properties. Chen and Mishra [6] noticed that when checking a large set of relevant properties, SAT instances of similar properties have a large overlap of CNF clauses and can be clustered. Conflict clauses generated by the base property can be forwarded to other properties in the same cluster. Decision ordering information can also be used to guide the SAT search [22,14]. Strichman [28] presented a BMC optimization technique based on decision ordering derived from the characteristics of BMC formulas. Wang et al. [31] analyzed the correlation among different SAT instances of a property. They used the *unsatisfiable core* of previously checked SAT instances to derive the variable ordering for the current SAT instance. Zhang et al. [33] investigated the BMC-specific ordering strategies for SAT solvers. They proposed an incremental framework for BMC which uses a clever orchestration approach of variable ordering. Chen and Mishra [9] tuned decision ordering based on the counterexamples of checked properties. Test generation complexity can be reduced by sharing knowledge (i.e. conflict clauses and decision ordering) among properties [8]. However, the clustering-based approaches have one major limitation. The base property (the first property in a cluster) is solved alone without any benefit of learning. Therefore, the solving time for the base property can be too long compared to other properties in the cluster. Consequently, complexity of the base property dominates the overall test generation time.

To the best of our knowledge, our approach is the first attempt to propose efficient property decomposition and learning techniques to significantly reduce the overall test generation time for a large set of properties.

3 Preliminaries

3.1 Test Generation using SAT-Based BMC

Algorithm 1 outlines the test generation procedure using SAT-based BMC [19]. The algorithm produces a test suite from: i) a formal description of *golden reference model*, and ii) a set of false safety properties in the form of $!F(S)$ indicating the negation of the desired functional scenario S. It iterates until all the properties are checked. In each iteration, the bound $bound_i$ of each safety property p_i is determined first. Then SAT-based BMC takes model M, negated property p_i, and bound $bound_i$ as inputs and generates a counterexample (test) to falsify the property p_i.

Algorithm 1: Test Generation using SAT-Based BMC

```
Input: i) Formal Design Model, M and
       ii) A set of false properties P
Output: Testsuite
TestGen(M, P) begin
    TestSuite = ϕ;
    for each property p_i in the set P do
        bound_i = DetermineBound(M, p_i);
        test_i = BMC(M, p_i, bound_i);
        TestSuite = TestSuite ∪ test_i;
    end
    return TestSuite;
end
```

Determination of bound is hard in general. Similar to the work described in [10,11], we assume that the bound of a complex property and decomposed properties can be estimated by exploiting the structure of underlying models of designs (see the example shown in Section 4.5).

3.2 SAT Solving – CDCL Algorithm

Since in SAT-based BMC a test is a satisfying assignment for a SAT instance, the test generation performance is determined by the SAT solving procedure. Several popular SAT solvers such as Chaff [26] and MiniSAT [25] adopt the *Conflict-Driven Clause Learning* (CDCL) algorithm [21].

Algorithm 2: CDCL based SAT-solving procedure

```
while TRUE do
    run_periodic_functions();
    if decide_next_branch() then
        while deduce() == CONFLICT do
            blevel = analyze_conflicts();
            if blevel < 0 then
                return UNSAT;
            end
        end
    else
        return SAT;
    end
end
```

Algorithm 2 shows a general implementation of CDCL based SAT solving procedure. It contains three parts:

- *Periodic functions* update SAT settings periodically, such as updating literal scores after a certain number of backtracks.
- *Boolean Constraint Propagation* (BCP) is implemented in *deduce()*. It determines all possible implications made by *decide_next_branch()*.
- *Conflict analysis* analyzes the reason of conflict and creates a conflict clause to avoid the same conflict in future processing, and then does a non-chronological backtracking up to the closest decision which caused the conflict.

3.3 Decision Ordering

Decision ordering specifies which variable should be selected first and which value (true or false) needs to be first assigned to this variable. Since different decision ordering heuristics can lead to different SAT search trees [22], decision ordering plays an important role in determining the performance of SAT solving. The decision ordering heuristic VSIDS [26] is widely used in modern SAT solvers such as MiniSAT [15] and zChaff [26].

During the SAT solving using MiniSAT, each Boolean variable is associated with a counter which indicates the priority for decision ordering at *decide_next_branch()*. Initially the counter value is only determined by the structural information of the corresponding CNF file. During SAT solving, the update of counter values is triggered by specific periodic events (i.e., a certain number of backtracks). Instead of being divided by a constant factor as implemented in zChaff, in MiniSAT, variable counters are "bumped" with larger and larger floating-point based values. When some variable counter reaches a very large limit value, the value of all variable counters will be scaled down. All the variable counter values are stored in a MinHeap, and the function *decide_next_branch()* will choose the variable with the minimum counter value for decision.

3.4 Property Learning and Clustering Techniques

According to Equation (1), similar properties are expected to have a large overlap between their CNF clauses, because they share both the transition relation $T(s_i, s_{i+1})$ and part of property checking ($p(s_i)$). Since $T(s_i, s_{i+1})$ occupies a major part of CNF clauses, the clause overlap between similar properties is large, which enables the conflict clause sharing between properties for test generation. Alternatively, for similar properties, there exists a large overlap between the variable assignments in corresponding counterexamples. Therefore, the satisfying assignments of checked properties are effective to predict the decision ordering for unchecked properties [7].

To fully exploit the learning potential, [6] identifies four promising similarity metrics for property clustering.

- **Similarity based on structural overlap** can be used when the structure information of the design is provided.
- **Similarity based on textual overlap** is beneficial when the properties are well structured, but the information regarding the design is not available.
- **Similarity based on influence** can be used when the cause-effect relations of design component activations can be inferred.
- **Similarity based on CNF intersection** can be used when only the CNF clauses are provided in the absence of both the design and property information.

Unlike the above clustering methods proposed in [6], the clustering approaches proposed in this paper are based on the decomposed sub-properties. Since we consider both the structure and behavior information of design from the property perspective, the proposed clustering approaches in this paper are more beneficial than the methods in [6]. Although [9] presented a promising approach to improve the base property, it solves the base property using the learning from the SAT instance itself without using any beneficial learning from other external sources. In this paper, we investigate the learning derivation from the decomposed sub-properties, which is more efficient to accelerate the solving of a base property.

4 Learning-Oriented Property Decomposition

During model checking based test generation, falsification of a complex property is very time-consuming. It is promising to exploit the fact that the test of a complex property and the tests of its sub-properties usually have a large overlap in variable assignments. In other words, the results of sub-property checking can be beneficial for the complex property checking.

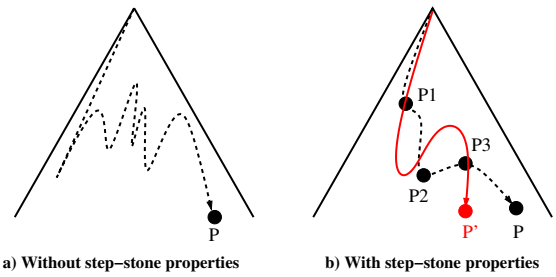

Fig. 2 An example of test generation with learning

Inspired by the ideas presented in [4], several "step stone" properties can be exploited to help the complex property checking. Figure 2 shows the basic idea of our method. In Figure 2a, due to random search in a decision tree and lack of guidance by "step stone" sub-properties, the test generation time is extremely long. In Figure 2b, P_1, P_2 and P_3 represent some sub-functional scenarios of property P, and property P' is in the same cluster as P due to their similarity. Since a sub-functional scenario involves only a small subset of components of the original design, the cost of checking P_1, P_2 and P_3 is typically much lower than checking P. However, the learning derived from such sub-properties can be beneficial for checking P, which can reduce the overall test generation time for P. Furthermore, since P' is similar to P (i.e., they have an overlap on two sub-properties P_1 and P_3), the learning derived from P can also be utilized to accelerate P''s test generation.

Definition 1 A *property series* is a sequence of properties for sharing knowledge. It is in the form $(\{P_1, P_2, \ldots, P_n\} :\xrightarrow{X} P)$, where P is the *target property*; P_i $(1 \leq i \leq n)$ is a *beneficial sub-property* to derive learning; and "$:\xrightarrow{X}$" indicates that the learning is based on the decomposition of type X. ■

In our approach, the goal of property decomposition is to construct a series of simple properties, which can be used to derive *decision ordering based learning* for complex property checking. To construct such a property series, we propose three types of decomposition methods: i) *spatial decomposition* which breaks down a complex system-level property into several component-level sub-properties, ii) *temporal decomposition* which deduces a large-bound property (i.e., property activated at a late stage) from some smaller bound properties (i.e., properties activated at earlier stages), and iii) *hybrid decomposition* which combines the advantages of both temporal and spatial decompositions for further improvement.

4.1 Decision Ordering Based Learning

Generally, the counterexamples of decomposed beneficial sub-properties have a large overlap in variable assignment with the original property. They contain rich information to guide the SAT solving of the base property. Therefore, they can be used as a learning to bias the decision ordering when checking the base property. Moreover, when the base property checking is done, its checking result contains abundant learning information (i.e., variable assignments) to guide the checking of other similar properties in the same cluster.

In SAT search, decision ordering plays an important role to quickly find a satisfying assignment. In our test generation framework, we developed a heuristic to predict the decision ordering based on the statistics collected from the decomposed sub-properties as well as checked complex properties (see details in Section 5.2).

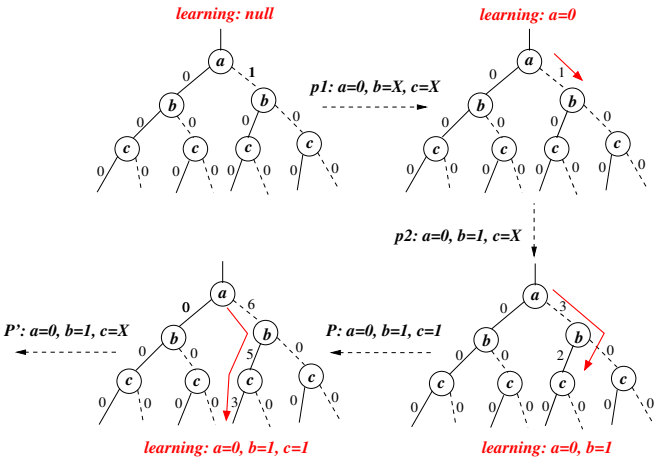

Fig. 3 Decision ordering prediction based on learning information

Figure 3 shows the basic idea of our heuristic. In a decision tree, each node indicates a Boolean variable. The left branch of the node means that the variable is assigned with *true*, while the right branch denotes the *false* assignment[1]. Each edge of the tree is associated with a weight, which is used to predict the decision ordering. Assume that we are checking two similar complex properties P and P', and P is checked first. The property P with a bound of 3 has two beneficial sub-properties p_1 and p_2 with bound 1 and 2, respectively. Assume that we always check the variables in the order of a, b, c. Initially, since the weight on all edges is set to 0, there is no learning when checking p_1. However, after checking p_1, the weighted edges will be updated based on the result of p_1 (i.e., $a = 0$, $b = X$, $c = X$). In this case, the weight of the edge $a = 0$ will be increased by the bound of p_1 (i.e., 1). Then the decision ordering for p_2 can be predicted based on the result of the updated decision tree. As indicated by the solid arrow line, when checking p_2, the assignment of a is more likely to be *false*, and the value of b and c are unknown. Similarly, we can predict the decision ordering for P and P' based on the learning derived from the checked properties.

4.2 Spatial Property Decomposition

A complex property may describe a functional scenario with multiple component interactions. *Spatial property decomposition* tries to partition such a complex functional scenario into several sub-functional scenarios which involve fewer component interactions.

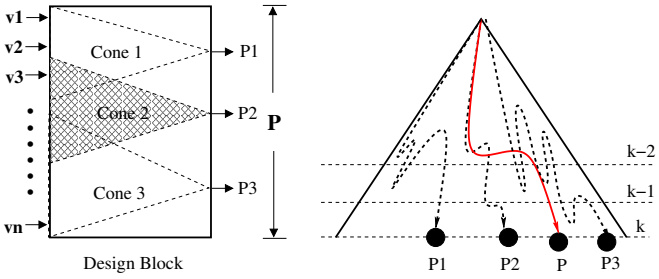

Fig. 4 An illustration of spatial property decomposition

As shown in Figure 4a, assume that property P can be broken into 3 component level sub-properties P_1, P_2 and P_3 with different Cone of Influence (COI)[2], where $COI(P_i) \subset COI(P)$ $(1 \leq i \leq 3)$. As shown in Figure 4b, the spatial decomposition tries to construct a property series $(\{P_1, P_2, P_3\}$

[1] In this paper, $v = 1$, $v = 0$ and $v = X$ indicate that the Boolean variable v equals to *true*, *false* and *UNKNOWN* respectively.

[2] Here, COI indicates the variables involved during the property checking.

$:\xrightarrow{S} P$), where $:\xrightarrow{S}$ indicates that the learning is derived from spatial decomposition. For a complex design, when checking a sub-property such as P_i (with bound K, $1 \leq i \leq 3$) with a smaller COI, it usually needs much less time and memory than P. However, the knowledge learned from P_i can be very useful for test generation of property P.

A Linear Temporal Logic (LTL [12]) formula consists of temporal operators (G, F, X, U) and Boolean connectives ($\wedge$, $\vee$, $\neg$ and $\rightarrow$). When a property involves complex logic formula syntax, after proper transformation, it can be decomposed into a set of sub-properties. If partial counterexamples generated by the sub-properties can be processed to guide the complex property falsification, the original property is *spatially decomposable*.

Definition 2 A false safety property P is *spatially decomposable* if all of the following conditions hold.

- P can be transformed to the property in the form of $p_1 \wedge p_2 \wedge \ldots \wedge p_n$ or in the form of $p_1 \vee p_2 \vee \ldots \vee p_n$, where the bounds of sub-properties p_i $(1 \leq i \leq n)$ are the same.
- There exists a property series such that $(\{p_i\} :\xrightarrow{S} P)$ $(1 \leq i \leq n)$. ■

It is important to note that, if the original property is in the form $p_1 \wedge p_2 \wedge \ldots \wedge p_n$, then at least one sub-property p_i $(1 \leq i \leq n)$ is required to have a counterexample. The bound of P is the minimum bound of p_i which has a counterexample. If the original property is in the form $p_1 \vee p_2 \vee \ldots \vee p_n$, then every sub-property p_i $(1 \leq i \leq n)$ should be false. The bound of P is the maximum bound of all decomposed sub-properties.

According to Definition 2, the following rules can be used for complex property decomposition.

$$\begin{aligned} &\neg X(p \vee q) \equiv \neg X(p) \wedge \neg X(q) \\ &\neg X(p \wedge q) \equiv \neg X(p) \vee \neg X(q) \\ &\neg F(p \vee q) \equiv \neg F(p) \wedge \neg F(q) \end{aligned} \tag{2}$$

In the context of test generation using some fault models [6], the properties are typically in the form of $\neg F(p \vee q)$, $\neg F(p \rightarrow q)$ and $\neg F(p \wedge q)$. For a complex property in the form of $\neg F(p \vee q)$, it can be decomposed into a conjunctive form $p_1 \wedge p_2 \wedge \ldots \wedge p_n$. In this case, it is not necessary to use the learning information. Since the size of SAT instances derived from model checkers partially reflects the COI and bound information of properties, our framework needs to sort the sub-properties p_i $(1 < i \leq n)$ according to the increasing size of their SAT instances. The counterexample of the first falsified property can be used as a counterexample for the complex property.

It is important to note that the properties in the form of $\neg F(p \wedge q)$ or $\neg F(p \rightarrow q)$ cannot be directly decomposed into conjunctive or disjunctive form. However, by explicitly introducing the notion of a synchronous clock variable *clk* [19], the properties in the form of $\neg F(p \wedge q)$ can be spatially decomposed as shown in the Equation (3).

$$\begin{aligned} &\neg F(p \wedge q) = false \\ &\equiv \neg F(p \wedge q \wedge clk = k) = false \\ &\equiv (\neg F(p \wedge clk = k) \vee \neg F(q \wedge clk = k)) = false \\ &\equiv (\neg F(p \wedge clk = k) = false) \wedge (\neg F(q \wedge clk = k) = false) \\ &\text{where k = bound of } \neg F(p \wedge q). \end{aligned} \tag{3}$$

It implies that the counterexample of $\neg F(p \wedge q)$ can benefit from the counterexamples of $\neg F(p \wedge clk = k)$ and $\neg F(q \wedge clk = k)$, i.e., $(\{\neg F(p \wedge clk = k), \neg F(q \wedge clk = k)\} :\xrightarrow{S} \neg F(p \wedge q))$.

For a property in the form $\neg F(p \rightarrow q)$, it cannot be directly transformed into $\neg F(\neg p \vee q)$. In $\neg F(p \rightarrow q)$, p denotes the pre-condition and q indicates the post-condition. When $G(\neg p)$ holds, $\neg F(p \rightarrow q) = false$ will be vacuously true, and the checking of $\neg F(p \rightarrow q)$ will report a counterexample without satisfying the precondition p. This counterexample cannot match the original testing intention. For the purpose of test generation, in our decomposition approach, the properties in the form of $\neg F(p \rightarrow q \wedge clk = k)$ can be transformed to $\neg F(p \wedge q \wedge clk = k)$ where k equals to the bound of $\neg F(p \wedge q)$. The Equation (3) then can be used to decompose the property $\neg F(p \wedge q \wedge clk = k)$.

Note that when checking a complex property which can be decomposed in disjunctive form, it is not necessary to check all its sub-properties. If the COI of a sub-property is similar to the original property, the complexity of such sub-property will be similar to the complex property. In this case, the learning is not economical. In other words, the sub-properties with smaller COI than the complex property need to be checked.

When a complex property involves too many atomic sub-properties, checking each of them individually may not derive useful learning, since each atomic sub-property only have a narrow view of the system. In this case, the commutative and associative laws can be used to classify atomic sub-properties into several groups. For example, in Equation (4), p_i and p_k are grouped together, and p_j belongs to another group.

$$p_i \vee p_j \vee p_k = (p_i \vee p_k) \vee p_j \tag{4}$$

For each group, we generate a *refined property* which represents all the atomic sub-properties in the group to derive learning. For grouping, the following rules based on modular and functional information work well for most of the time.

– **Modular similarity:** In each group, all the variables in the sub-formulas should come from the same component (e.g., fetch module in a processor design).
– **Functional similarity:** In each group, all the sub-formulas should describe related functional scenarios (e.g., fetching instructions/data in a processor design).

Algorithm 3 outlines our spatial decomposition method to derive a set of refined sub-properties with small COI for deriving learning. The inputs of the algorithm are the design model D and the complex property P in disjunctive form. Step 1 initializes the *SD_props* with an empty set. Step 2 tunes sub-properties' order according to the commutative law and groups sub-properties using the specified *similarity grouping rules*. Step 3 selects the i^{th} group of k sub-properties. If the combined COI of such a group is smaller than $\frac{k}{n}$ of $P's$ COI, step 4 will generate a refined property *newP* for the i^{th} cluster, and step 5 adds *newP* to *SD_props*. Finally this algorithm returns a series of sub-properties for deriving learning (described in Section 5.2). Since the COI of refined properties is small, the test generation complexity for the whole sub-property series is expected to be much smaller than that of the original complex property. It is important to note that Algorithm 3 may return an empty property series if spatial decomposition is not beneficial.

Algorithm 3: Spatial Decomposition

Input: i) The design model, D and
ii) A property P in the form $p_1 \vee p_2 \vee \ldots \vee p_n$
Output: A property series using spatial decomposition
spatial_decompose(D,P) **begin**
  1. $SD\_props = \{\}$;
  2. $(group_1, \ldots, group_m) =$ $grouping(P, modular/functional)$;
  for *i is from 1 to m* **do**
    3. $group_i = \{prop_1, \ldots, prop_k\}$;
    if $COI(group_i) \leq \frac{k}{n} COI(P)$ **then**
      4. $newP = prop_1 \vee \ldots \vee prop_k$;
      5. $SD\_props = SD\_props \bigcup \{newP\}$;
    end
  end
  return $(SD\_props :\xrightarrow{S} P)$;
end

4.3 Temporal Property Decomposition

In event-driven designs, a *transaction* consists of a sequence of correlated *events*, whose order can be used to indicate different stages of a dynamic system behavior. When generating a directed test for activating such a transaction, the corresponding property is specified to validate a single event or a scenario (a sequence of events). If the investigated event has a large delay (i.e., bound), the complexity of the property checking using SAT-based BMC will increase drastically, because it requires increased unrollings of the design.

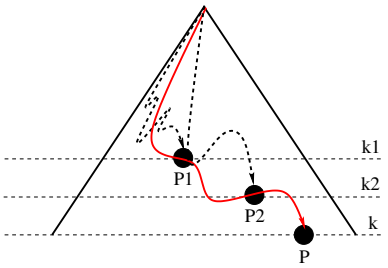

Fig. 5 An illustration of temporal property decomposition

Temporal property decomposition tries to eclipse the effect of bound. The basic idea of temporal decomposition is to deduce the long bound property from a sequence of short bound properties, i.e. to construct a property series $(\{P_1, P_2, \ldots, P_n\} :\xrightarrow{T} P)$ where $bound(P_i) < bound(P_{i+1})$ $(1 \leq i < n)$, and $:\xrightarrow{T}$ indicates that the learning can be achieved from temporally decomposed sub-properties. For example in Figure 5, P_1 and P_2 are sub-properties representing different stages of property P. The bound of them are k_1, k_2, and k respectively and $k_1 < k_2 < k$. Because P_1's counterexample is similar to the prefix of P_2's counterexample, P_1's counterexample contains rich knowledge that can be used for checking P_2. Similarly, during the property checking, P can benefit from both P_1 and P_2. By using such learning, we can quickly obtain a counterexample for P.

Definition 3 Let P be a false safety property of the design, and P is *temporally decomposable* if all of the following conditions are satisfied.

– P can be decomposed into false properties p_1, p_2, …, p_n with increasing bounds and $P = p_n$.
– $(\{\neg p_i\} :\xrightarrow{T} \neg p_{i+1})$ $(1 \leq i \leq n-1)$, which indicates that the counterexample generated from properties p_i can guide the test generation for property p_{i+1}. We use $(\{\neg p_1, \neg p_2, \ldots, \neg p_{n-1}\} :\xrightarrow{T} P)$ to denote $(\{\neg p_i\} :\xrightarrow{T} \neg p_{i+1})$ $(1 \leq i \leq n-2)$, and $(\{\neg p_{n-1}\} :\xrightarrow{T} P)$. ■

In temporal decomposition, finding the implication relation (“$:\xrightarrow{T}$”) between properties is a key process. In our framework, we construct such implication relation by exploring the partial order of events. For example in Figure 6, there are 3 transactions, and each transaction has two events. We classify the relation between these events in two categories. The *cause-effect* relation (marked by $\Rightarrow$) defines the causal relation between inter-transaction events. For example, there are two events e_1 and e_5 in transaction $T1$. $e_1 \Rightarrow e_5$ means that if $e1$ happens, $e5$ should happen in future. The *happened-before* relation (marked by $\prec$) specifies the relation between events. It indicates which events may happen before other events under some condition. For instance, $e_3 \prec e_5$ means that e_3 may happen before e_5.

During test generation, we apply properties in the form $\neg F(e)$ to indicate that the event e cannot be activated. Generally, if the event happens with a large delay, BMC needs to unroll the design many times that can drastically increase the checking complexity. According to Definition 3, the "$\Rightarrow$" relation can be used to derive learning. For example, in Figure 6, instead of checking the property $P_1 = \neg F(e_5)$ directly, we can check $P_2 = \neg F(e_1)$ first. Since $e_1 \Rightarrow e_5$ implies $F(e_1) \rightarrow F(e_5)$ (i.e., $\neg P_2 \rightarrow \neg P_1$), it indicates that P_2's counterexample can be used to guide the generation of P_1's counterexample. This can be written as $(\{P_2\} :\xrightarrow{T} P_1)$. The "$\prec$" relation also can be used to indicate the learning information.

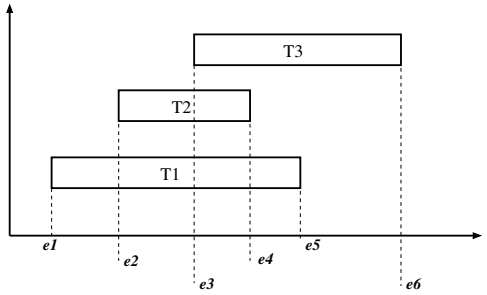

Fig. 6 A sequence of three transactions

In Figure 6, $e_3 \prec e_5$ indicates that the counterexample of $\neg F(e_3)$ is shorter than the counterexample of $\neg F(e_5)$. Although the occurrence of e_3 may not lead to the occurrence of e_5, the counterexample of $\neg F(e_3)$ may still have a large overlap of variable assignments with the counterexample of $\neg F(e_5)$. Therefore the happened-before relation can be used as a weak form of implication defined in Definition 3. It is important to note that the cause-effect relation is a stronger form of happened-before relation.

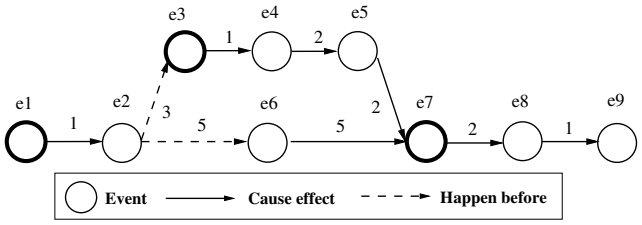

Fig. 7 An example of event relation graph

When checking a property with a large bound, there may be many events along the path to the target events. Checking all of them individually is time-consuming. In fact, only the events that follow branching and merging points along the path from initial events to target events need to be considered since they are important in determining the execution order, while the other points are only used to indicate "$\Rightarrow$" relations. For example, in Figure 7, the relation between events is described using a directed acyclic graph (DAG). Such information can be derived from the graph model automatically by exploring the structure of the given graph models (see the examples in Section 4.5). In a DAG, each node indicates an event; each directed edge indicates the relation of $\Rightarrow$ or $\prec$; and the weight on each edge is the delay between two adjacent events. In this DAG, there are 8 events before $e9$ (i.e., the target event). For the temporal decomposition in this DAG, it is beneficial to consider only the branching and merging events e_1, e_3 and e_7.

Algorithm 4: Temporal Decomposition

Input: i) Formal model of design, D
ii) A property $P = \neg F(dest)$ where $dest$ is a target event
Output: A property series using temporal decomposition
temporal_decompose(D,P) **begin**
1. (D', src) = $event\_graph(D,P)$;
2. $path$ = Dijkstra(D', src, $dest$) to find a shortest path;
3. $TD\_events = \{\neg F(src)\}$;
for *i is from 2 to len (number of events in path)* **do**
4. (e_{i-1}, e_i) = $(i-1)_{th}$ edge of $path$;
if $out\_degree(e_{i-1})$ + $in\_degree(e_i) > 2$ **then**
5. $TD\_events = TD\_events \bigcup \neg F(e_i)$;
end
end
return $(TD\_events :\xrightarrow{T} \neg F(dest))$;
end

Algorithm 4 describes how to obtain a sequence of sub-properties for temporal decomposition. It accepts the formal model of the design and a complex property indicating the target event. Step 1 figures out the corresponding event graph of the design as well as the source event. Since BMC can detect errors within smallest range, we use Dijkstra's algorithm [13] to find a smallest *delay path* in Step 2. Step 3 initializes the set TD_events with the sub-property based on initial event. Steps 4 and 5 select the *branch events* and append corresponding sub-properties to the TD_events. Finally the algorithm reports the sub-property series generated by temporal decomposition. By using this algorithm, $(\{\neg F(e_1), \neg F(e_3), \neg F(e_7)\} :\xrightarrow{T} \neg F(e_9))$ is a property series for temporal decomposition in Figure 7.

4.4 Hybrid Decomposition

Spatial decomposition is promising for the properties in the form of $p_1 \wedge p_2 \ldots \wedge p_n$ or $p_1 \vee p_2 \ldots \vee p_n$. However, when the COI of decomposed sub-properties is large, spatial decomposition may not be economical due to long learning derivation time. As an alternative, temporal decomposition exploits learning from different stages of designs. However, in many scenarios, it is hard to figure out the cause-effect and happened-before relations from designs.

Figure 8 shows a part of a design. Assume that we need to generate a test to exercise three events $e_{1,3}$, $e_{2,2}$, and $e_{3,4}$ at the same time, i.e. $clk = k$, the corresponding false property would be $P = \neg F(e_{1,3} \wedge e_{2,2} \wedge e_{3,4} \wedge clk = k)$. By using spa-

tial decomposition, we can decompose P and achieve a property series ($\{\neg F(e_{1,3} \wedge clk = k), \neg F(e_{2,2} \wedge clk = k), \neg F(e_{3,4} \wedge clk = k)\} :\xrightarrow{S} P$) for test generation. However, since $e_{1,3}$, $e_{2,2}$, and $e_{3,4}$ may have large COI, the test generation will fail because checking SAT instances of sub-properties may be more costly than the original property. Temporal decomposition is promising to handle the properties of large bounds. Since $(e_{1,1} \wedge e_{2,1} \wedge e_{3,1} \wedge clk = k-4) \Rightarrow (e_{1,3} \wedge e_{2,2} \wedge e_{3,4} \wedge clk = k)$, it indicates that ($\{\neg F(e_{1,1} \wedge e_{2,1} \wedge e_{3,1} \wedge clk = k-4)\} :\xrightarrow{T} P$). However, $\neg F(\ e_{1,1}\ \wedge e_{2,1} \wedge e_{3,1} \wedge clk = k-4)$ may still have large SAT instance size, which needs a long checking time.

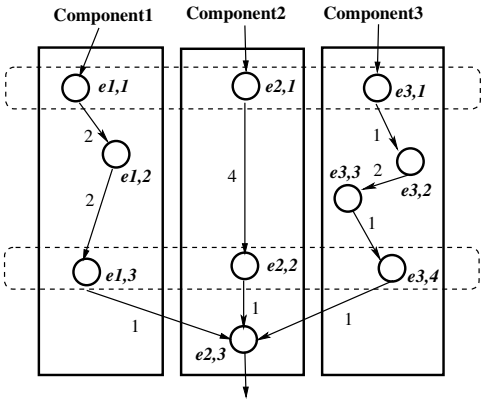

Fig. 8 An example of hybrid decomposition

To address this issue, we can utilize the *hybrid decomposition* combining both spatial decomposition and temporal decomposition. As an example for P, we can get four property series as follows:

1. ($\{\neg F(e_{1,3} \wedge clk = k), \neg F(e_{2,2} \wedge clk = k), \neg F(e_{3,4} \wedge clk = k)\} :\xrightarrow{S} P$)
2. ($\{\neg F(e_{1,1} \wedge clk = k-4)\} :\xrightarrow{T} \neg F(e_{1,3} \wedge clk = k)$)
3. ($\{\neg F(e_{2,1} \wedge clk = k-4)\} :\xrightarrow{T} \neg F(e_{2,2} \wedge clk = k)$)
4. ($\{\neg F(e_{3,1} \wedge clk = k-4)\} :\xrightarrow{T} \neg F(e_{3,4}) \wedge clk = k$)

Although checking $\neg F(e_{1,3})$, $\neg F(e_{2,2})$, $\neg F(e_{3,4})$ individually without learning can be costly, the overall test generation time can be drastically reduced because of the temporal decomposition shown in property series (2) - (4). It is true that we can consider the above property series as a dependence graph. As shown in Figure 9, the property P depends on $\neg F(e_{1,3})$, $\neg F(e_{2,2})$ and $\neg F(e_{3,4})$; $\neg F(e_{1,3})$ depends on $\neg F(e_{1,1})$; $\neg F(e_{2,2})$ depends on $\neg F(e_{2,1})$; and $\neg F(e_{3,4})$ depends on $\neg F(e_{3,1})$.

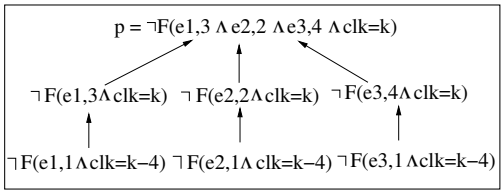

Fig. 9 An example of property dependence

Similarly, when checking the property $P' = \neg F(e_{2,3})$ with a bound k, it is hard to do the spatial decomposition. To achieve learning for test generation, we can use the hybrid decomposition which first temporally decompose the complex property and then spatially decompose the property. It is important to note that only the temporally decomposed sub-property with the smallest bound needs to be spatially decomposed. In the example of decomposition of P', we can get two property series for test generation of P' as follows:

1. ($\{\neg F(e_{1,1} \wedge e_{2,1} \wedge e_{3,1} \wedge clk = k-5), \neg F(e_{1,3} \wedge e_{2,2} \wedge e_{3,4} \wedge clk = k-1)\} :\xrightarrow{T} P'$)
2. ($\{\neg F(e_{1,1} \wedge clk = k-5), \neg F(e_{2,1} \wedge clk = k-5), \neg F(e_{3,1} \wedge clk = k-5)\} :\xrightarrow{S} \neg F(e_{1,1} \wedge e_{2,1} \wedge e_{3,1} \wedge clk = k-5)$)

Algorithm 5 describes our hybrid decomposition method. The inputs of the algorithm are a formal model of the design and a complex property P. Step 1 initializes an empty property series set PS. Step 2 tries to decompose property P spatially and step 3 figures out the corresponding beneficial sub-properties defined in Definition 1. If P cannot be spatially decomposed, step 4 puts P to the property set *props* for the second try of spatial decomposition in steps 7 and 8. Otherwise, step 5 includes the property series into PS. Then the algorithm tries to decompose each beneficial sub-property temporally in step 6. If P cannot be spatially decomposed in step 2, step 7 selects the beneficial sub-property p with the smallest bound, and step 8 applies spatial decomposition on p. Step 9 incorporates the property series derived from temporal decomposition into PS. Finally, the algorithm reports property series PS for test generation.

Algorithm 5: Hybrid Decomposition

```
Input: i) Formal model of the design, D
       ii) A complex property P
Output: A set of property series, PS
hybrid_decompose(D,P) begin
    1. PS = {};
    2. s_series = spatial_decompose(D,P);
    3. props = beneficial_property(s_series);
    if |props| == 0 then
        4. props = {P};
    else
        5. PS = PS ⋃ s_series;
    end
    foreach property p_i ∈ props do
        6. t_series = temporal_decompose(D,p_i);
        if |beneficial_property(t_series)| > 0 then
            if props == {P} then
                7. p = first_element(t_series);
                8. PS = PS ⋃ spatial_decompose(D,p);
            end
            9. PS = PS ⋃ t_series;
        end
    end
    return PS;
end
```

4.5 An Illustrative Example

This section presents an example to illustrate the use of decomposition methods using a MIPS processor design shown in Figure 10. The figure shows the graph model of a MIPS processor. It consists of five pipeline stages: fetch, decode, execute, memory and writeback. It has four pipeline paths in the execute stage: ALU for integer ALU operation, FADD for floating-point addition operation, MUL for multiply operation and DIV for divide operation. In the figure, solid ovals denote functional units; dashed ovals are storages; solid edges are instruction-transfer (pipeline) edges; and dashed edges are data-transfer edges. Assume that we want to check a complex scenario that the functional units $MUL5$ and $FADD3$ will be active at the same time. We need to generate the property $P = !F(MUL5.active = 1 \& FADD3.active = 1)$ which is the negation of the desired behavior. The remainder of this section describes how to perform property checking using spatial, temporal and hybrid decompositions.

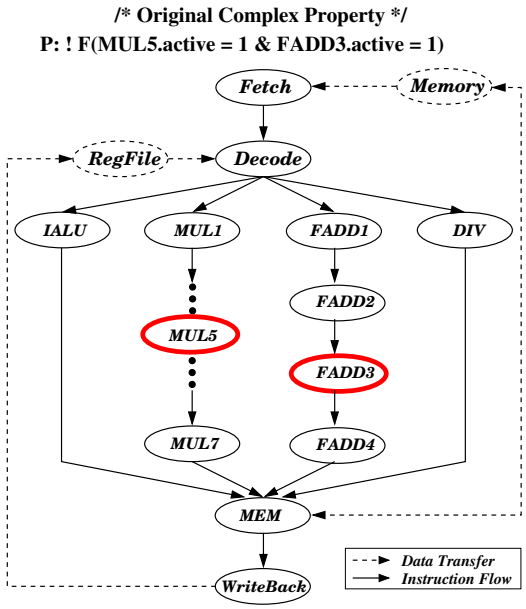

Fig. 10 A VLIW MIPS process example

4.5.1 Example of Spatial Decomposition

In the MIPS design, assume that each functional unit has a delay of one clock cycle. To trigger the functional unit $MUL5.active$, at least 7 clock cycles are required (there are 7 units along the path $Fetch \rightarrow Decode \rightarrow \ldots \rightarrow$ MUL5). Similarly, to trigger the functional unit $FADD3.active$, at least 5 clock cycles are required. To guarantee that both units are active at the same time, at least 8 clock cycles are required to activate this interaction - the first clock cycle is used for instruction initialization; then a MUL instruction is fetched in the second clock cycle, and an $FADD$ instruction is fetched in the fourth clock cycle, and so on. Thus the bound of this property is 8. According to Equation (3) and Algorithm 3, property P can be spatially decomposed into two sub-properties as follows. Assuming that the COI of $P1$ and $P2$ are both smaller than half of COI of P, we can get the property series $(\{P1, P2\} :\xrightarrow{S} P)$. It is important to note that the exact value assigned to clk is based on the bound of the property.

```
/* Spatially decomposed properties*/
P1: !F(MUL5.active=1 & clk=8)
P2: !F(FADD3.active=1 & clk=8)
```

When checking $P1$ and $P2$ individually, we can get the following two counterexamples.

```
Cycles P1's Counterexample  P2's Counterexample
  1      NOP                    NOP
  2      MUL R2, R2, R0         NOP
  3      NOP                    NOP
  4      NOP                    FADD R1, R1, R0
         ...                    ...
```

However, if the test generation of $P2$ is under the guidance of $P1$'s result, the counterexample of $P2$ will reflect $P1$'s partial behavior (see clock cycle 2 below). It can be observed that $P2$'s counterexample has a large overlap with P's counterexample. Thus the counterexamples of both $P1$ and $P2$ can be used to derive decision ordering based learning (see Section 5.2) for P's checking.

```
Cycles   Counterexample for P2 guided by P1
  1          NOP
  2          MUL R2, R2, R0
  3          NOP
  4          FADD R1, R1, R0
             ...
```

4.5.2 Example of Temporal Decomposition

For temporal decomposition, we need to figure out the event implication relation first. Because we want to check the property P, the target event is $MUL5.active = 1 \& FADD3.active = 1$. Figure 11 shows the implication relation for this event. There are 7 events in this graph, and e_7 is the target event.

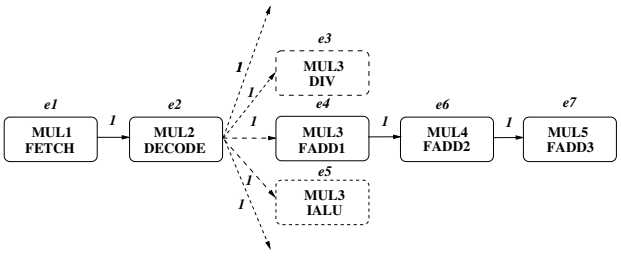

Fig. 11 The event implication graph for property P

Assuming e_1 is the initial event, from e_1 to e_7, there is only one path $e_1 \rightarrow e_2 \rightarrow e_4 \rightarrow e_6 \rightarrow e_7$. Along this path there is a branch node e_4. According to Algorithm 4, we need to check two events e_1 and e_4 using following properties. We can get the property series $(\{P\_e_1, P\_e_4\} :\xrightarrow{T} P)$.

By using our learning technique during the test generation, P_e_1 can benefit P_e_4, and P_e_4 can benefit P. It is important to note, from e_4 to e_7, there are no branch nodes. In other words, if e_4 is triggered, then e_7 should be triggered 2 clock cycles later. That means, the test for P_e_4 is also a test for P.

```
/* Temporally decomposed properties*/
P_e1: !F(FETCH.active=1 & MUL1.active=1)
P_e4: !F(MUL3.active=1 & FADD1.active=1)
```

4.5.3 Example of Hybrid Decomposition

By using our hybrid decomposition method described in Algorithm 5, the complex property P can be first spatially decomposed and then temporally decomposed. In this case, the spatial decomposition is the same as the example shown in Section 4.5.1. Therefore we can get the property series $(\{P1, P2\} :\xrightarrow{S} P)$. Since each of the properties ($P1$ and $P2$) is related to only a single pipeline path, they can be further decomposed temporally. $P1$ can be decomposed into a property series $(\{P1\_1, P1\_2\} :\xrightarrow{T} P1)$, and $P2$ can be decomposed into a property series $(\{P2\_1, P2\_2\} :\xrightarrow{T} P2)$, where $P1\_1$, $P1\_2$, $P2\_1$ and $P2\_2$ are described as follows.

```
/* Properties decomposed by hybrid method*/
P1_1: !F(Fetch.active=1 & clk=2)
P1_2: !F(MUL1.active=1 & clk=4)
P2_1: !F(Fetch.active=1 & clk=4)
P2_2: !F(FADD1.active=1 & clk=6)
```

5 Test Generation using Property Decomposition and Learning Techniques

In our framework, the decomposed sub-properties have two utilities: i) they can be used to efficiently cluster similar complex properties; and ii) they can be used to derive decision ordering based learning to reduce the test generation time. We have developed a test generation framework based on property decomposition and learning techniques.

Figure 12 shows the overview of our test generation framework. Firstly, properties are decomposed based on the structure information of the design. Since this step only analyzes the structure of the design, the decomposition overhead is negligible compared to model checking complexity. Then by comparing the similarity of decomposed sub-properties, complex properties are grouped into several clusters. The accumulative learning (shown using arrow towards "Learning" box) from the decomposed beneficial sub-properties as well as the checked complex properties in a specific property cluster can be utilized to improve the test generation time of unchecked complex properties. It is important to note that, in our approach, only the base property checking directly utilizes the learning from its sub-properties. The non-base property solving is based on accumulative learning from both checked sub-properties and the base property. The following sub-sections will describe our test generation approach in detail.

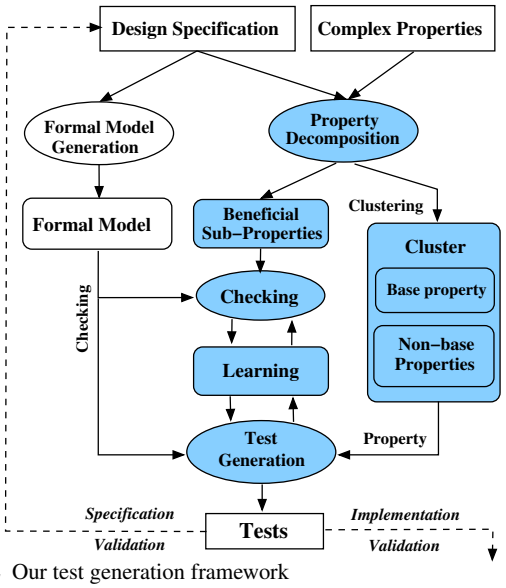

Fig. 12 Our test generation framework

5.1 Clustering using Decomposition Based Similarity

By decomposing a complex property into several sub-properties, spatial and temporal methods unveil different aspects of a system via utilizing structure and behavior information. For spatial decomposition, decomposed sub-properties involve fewer functional components, which in turn indicate partial behavior of the original complex property. For temporal decomposition, decomposed sub-properties infer different execution stages which can be considered as stepping stones of the desired behavior. Based on above facts, decomposed sub-properties can be used as preferable candidates to determine the similarity between complex properties. They can be used to cluster similar properties together for sharing learning.

Definition 4 Assume that two properties P and P' can be spatially decomposed, and their property series are $(\{p_1, p_2, \ldots, p_m\}:\xrightarrow{S}P)$ and $(\{p'_1, p'_2, \ldots, p'_n\}:\xrightarrow{S} P')$ respectively. Let $|A|$ denote the cardinality of the set A. The *spatial similarity* between property P' and P is

$$\frac{|\{p_1, p_2, \ldots, p_m\} \bigcap \{p'_1, p'_2, \ldots, p'_n\}|}{Max(|\{p_1, p_2, \ldots, p_m\}|, |\{p'_1, p'_2, \ldots, p'_n\}|)} \times 100\% \quad \blacksquare$$

To reduce the complexity of a property, spatial decomposition divides a complex property syntactically. The spatial similarity can be simply determined by checking how many sub-properties are same between properties.

For example, assume that there are two spatially decomposed property series $(\{a, b, c\}:\xrightarrow{S}P)$ and $(\{b, c, d, e\}:\xrightarrow{S} P')$. The spatial similarity between P and P' is 50%.

Unlike spatial decomposition, temporally decomposed sub-properties denote the potential events during the system execution. The behavior of the original complex property can be considered as an ordered sequence of such events (sub-properties). For two complex properties, the more similarity between two event sequences implies that more learning can be shared between two original complex properties. In our method, the temporal similarity is based on the matching of same segments between two event sequences.

Definition 5 Assume that two properties P and P' can be temporally decomposed, and their property series are ($\{p_1, p_2, \ldots, p_m\}$:$\xrightarrow{T}$$P$=$p_{m+1}$) and ($\{p'_1, p'_2, \ldots, p'_n\}$:$\xrightarrow{T}$ P'=p'_{n+1}) respectively. Let $\sigma(P,i)$ denote the consecutive sub-property pair (p_i, p_{i+1}) of P, and $\delta(P,i)$ = *bound*(p_{i+1}) - *bound*(p_i) denote the bound difference between two elements of the pair. Let $\alpha(\sigma(P,i),P')$ be a Boolean predicate (0 means false, 1 means true) to determine whether $\sigma(P,i)$ is a consecutive sub-property pair of P'. The *temporal similarity* of P' over P is

$$\frac{\Sigma_{i=1}^{m}\ (\delta(P,i) \times \alpha(\sigma(P,i),P'))}{Max(\Sigma_{i=1}^{m}\ \delta(P,i), \Sigma_{j=1}^{n}\ \delta(P',j))} \times 100\% \quad \blacksquare$$

For example, let *P.b* indicate that the bound of property P is b. Assume that there are two temporally decomposed property series ($\{a.1, b.3, c.5\}$:$\xrightarrow{T}$*P.10*) and ($\{b.3, c.5, d.8, e.9\}$:$\xrightarrow{T}$ P'.10). The temporal similarity between P and P' is 20%.

As shown in Algorithm 5, hybrid decomposition approach adopts both spatial and temporal methods. We can consider the hybrid method as a two-phase method (i.e., first spatial decomposition and then temporal decomposition, or vice versa). Since the second phase only decomposes light-weight sub-properties for further improvement, it can be ignored for similarity checking. In other words, the first phase of hybrid method plays a key role in similarity checking. In the case of spatial decomposition followed by temporal decomposition, the sub-properties derived from spatial decomposition already show the similarity between properties. Similarly, in the case of first temporal decomposition and then spatial decomposition, only one sub-property with smallest bound is involved for spatial decomposition. Therefore the temporally decomposed sub-properties are enough to determine the similarity between complex properties in this case.

5.2 Derivation of Decision Ordering Based Learning

Unlike the heuristics proposed in [9], we consider the bound information in our variable assignment statistics (an illustrative example is shown in Figure 3). Let $varStat[sz+1][2]$ (sz is the variable number for a complex property) be a 2-dimensional array to keep literal statistics. Initially, $varStat[i][0] = varStat[i][1] = 0$ ($1 \leq i \leq sz$). Since spatially decomposed sub-properties have the same bounds and are checked first, in our approach, the weight of such sub-properties is set to 1. However for temporal decomposition, the sub-property with larger bound can provide better support to the original property. Therefore, the weight of temporally decomposed sub-properties is equal to its real bound. When the base property checking is done, the weight change on literals also equals to the bound of the checked complex property.

Algorithm 6: Update of Weight Statistics of Literals

```
Input: i) The variable assignment statistics varStat
       ii) A test generated from a checked property P
       iii) Weight_P, which is the weight of P
Output:
Update(varStat, test, Weight_P) begin
    for i is from 1 to sz do
        if test.VarAssign[i] == 0 then
            1. varStat[i][0] += Weight_P;
        end
        if test.VarAssign[i] == 1 then
            2. varStat[i][1] += Weight_P;
        end
    end
end
```

Algorithm 6 describes our approach to achieve the accumulative weight for all literals to predict decision ordering of unchecked properties. After each property checking, *varStat* will be updated. Based on the satisfying variable assignment (i.e., *test*) of P, each element of $varStat[i][\,]$ will be increased by $bound_P$ according to the value of the variable v_i (i.e., $test.VarAssign[i]$). It is important to note that, if variable k is not determined yet when checking P, the value of both $varStat[k][0]$ and $varStat[k][1]$ will remain unchanged. By using Algorithm 6, the decision ordering heuristic of unchecked properties is gradually tuned by the checked properties based on the information saved in *varStat*.

MiniSAT employs a variant of the VSIDS heuristic. However, MiniSAT does not support explicit ordering for literals. It only keeps activity scores for variables and clauses, which cannot be used to predict the variable polarities (i.e., Boolean values of variables). Based on the statistics collected in *varStat*, if a variable has not been determined yet, its polarity can be predicted at the beginning and the restart of the search using the following formula:

$$polarity(v_i) = \begin{cases} true & (varStat[i][1] > varStat[i][2]) \\ false & (varStat[i][1] < varStat[i][2]) \\ skip & (varStat[i][1] = varStat[i][2]) \end{cases}$$

5.3 Test Generation using Our Method

Algorithm 7 outlines our test generation approach illustrated in Figure 12. The inputs of the algorithm are a formal model of the design, a property set P and corresponding satisfying bounds, and the similarity threshold for property clustering. In Algorithm 7, step 1 initializes *DecompProp* and $test_P$. Step 2 decomposes all the complex properties and step 3 clusters them according to decomposition based similarity.

For each property cluster, steps 4-13 utilize both decomposition and learning techniques to reduce the overall test generation time. Step 4 initializes *varStat* and step 5 decomposes p'_0 (p'_0 is the base property, i.e., the property to check first) into a set of properties *props* by applying suitable decomposition methods. Step 6 translates the decomposed sub-properties into a set of CNFs for SAT solving. To handle the simpler sub-properties first, step 7 sorts the encoded $CNFs$ according to their DIMACS file size. Steps 8 and 9 solve each decomposed sub-property individually, and update *varStat* accordingly. Since the complex properties in cluster Clu are assumed to be similar to p'_0, steps 10-13 check the $m+1$ ($m \geq 0$) properties one-by-one based on the statistics collected in *varStat*. Finally, step 14 reports the test set $test_P$ for all properties in P.

Algorithm 7: Our Test Generation Approach

Input: i) Formal model of the design, D
ii) Property set $P = \{p_0, p_1, \ldots, p_n\}$, with satisfying *bounds*
iii) *similarity* to cluster properties
Output: Directed test set $test_P$ for P
OurTestGenApproach$(D, P, similarity)$ **begin**
begin
1. $DecompProp = \{\}$, $test_P = \{\}$;
for *i is from 1 to the n* **do**
2. $DecompProp = DecompProp \bigcup decompose(D, p_i)$;
end
3. $PropClusters = Cluster(P, DecompProp, similarity)$;
foreach *cluster* $Clu = \{p'_0, \ldots, p'_m\} \in PropClusters$ **do**
4. Initialize *varStat*;
5. $(props', bounds') = decompose(D, p'_0)$;
6. $CNFs = BMC(D, props', bounds')$;
7. $(CNF_1, \ldots, CNF_k)$ = sort $CNFs$ using increasing DIMACS size;
for *i is from 1 to the k* **do**
8. $test_i = SAT(CNF_i, varStat)$;
9. $Update(varStat, test_i, bounds'_{CNF_i})$;
end
for *j is from 0 to the m* **do**
10. $CNF_{p'_j} = BMC(D, p'_j, bounds_{p'_j})$;
11. $test_{p'_j} = SAT(CNF_{p'_j}, varStat)$;
12. $test_P = test_P \bigcup test_{p'_j}$;
13. $Update(varStat, test_{p'_j}, bound_{p'_j})$;
end
end
14. return $test_P$
end
end

6 Experiments

In this section, we present two case studies: a MIPS processor design and a stock exchange system. Both of them are generated from high-level graph models [10], which enable automatic analysis of spatial and temporal decompositions. The design is transformed into a formal specification. The testing targets (based on fault modes) are converted into properties. Based on the graph traversal on the graph models using the breadth-first-search approach, the decomposition of each complex property of the two case studies can be conducted within 0.01 second, which is far less than the SAT solving time of the complex property. Therefore, we do not provide the property decomposition time explicitly in the experiment. They are included in either the sub-property or complex property checking time. We used NuSMV [27] to generate SAT instances (in DIMACS format) for automated test generation. Since from graph models we can figure out the bound of each property, the derived SAT instances are all satisfiable. In our approach, we assume that there exists a larger overlap between the variable assignments of original complex property and decomposed sub-properties. According to the observation that decision ordering-based learning are more effective than conflict-based learning under this assumption [7], we only investigate the decision ordering-based learning in our decomposition-based test generation approach. To incorporate our learning-based techniques, we modified the SAT solver MiniSAT-2.2 [25], which supports efficient decision ordering tuning. The experimental results are obtained on a Linux PC using 2.0GHz Core i7 CPU with 3 GB RAM. Since the intra- and inter-property approach [9] outperforms the methods described in [6], we only compare our method to the heuristics proposed in [9].

6.1 A MIPS Processor Design

This section demonstrates the effectiveness of our approaches using a pipelined MIPS processor design illustrated in Section 4.5. Since interactions between functional components [19] are recognized as complex scenarios, in this case study, the directed tests are generated for interaction faults.

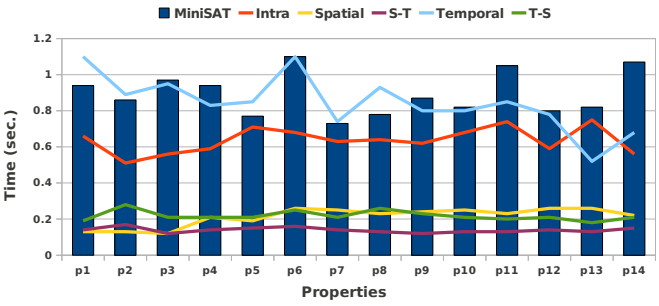

Fig. 13 Test generation results using MiniSAT

We selected 14 complex properties from the MIPS design based on the "2-interaction fault model"[3]. To check each property individually, Figure 13 shows the test generation results using MiniSAT with different learning techniques. When MiniSAT is used for the SAT solving, the

[3] A k-interaction property involves interactions among k components. For example, "*!F(decode.stall~=1 & fetch.stall~=1)*" is a 2-interaction property that involves interactions between *fetch* and *decode* units.

intra-property learning method [9] can get 1.40X overall improvement over MiniSAT. We can find that both of our spatial and temporal decomposition techniques outperform the methods using MiniSAT and intra-property learning approach. In this case, directed test generation using spatial decomposition method (4.20X improvement over MiniSAT) is better than temporal method (1.06X improvement over MiniSAT). This is mainly due to the fact that the temporally decomposed sub-properties still have large bounds, which needs a long solving time. To further reduce the test generation time, we applied hybrid decomposition in two ways: spatial followed by temporal (S-T) and temporal followed by spatial (T-S). The results show that the hybrid approach "S-T" and "T-S" can achieve 6.42X and 4.09X overall improvement over MiniSAT, respectively.

Table 1 Test Generation Results for the MIPS Design

Clustering Methods	Learning Methods	# of Clust.	Time (second)	Speedup
MiniSAT	None	16	23.11	1
Structural Clustering	Intra+Inter	2	3.14	7.36
	Spatial	2	2.49	9.28
	Hybrid (S-T)	2	2.22	**10.41**
	Temporal	2	3.20	7.22
	Hybrid (T-S)	2	2.30	10.05
Textual Clustering	Intra+Inter	3	3.98	5.81
	Spatial	3	2.50	9.24
	Hybrid (S-T)	2	2.37	**9.75**
	Temporal	3	5.19	4.45
	Hybrid (T-S)	3	2.56	9.03
Influence based Clustering	Intra+Inter	2	3.14	7.36
	Spatial	2	2.49	9.28
	Hybrid (S-T)	2	2.22	**10.41**
	Temporal	2	3.20	7.22
	Hybrid (T-S)	2	2.30	10.04
CNF Intersection Clustering	Intra+Inter	16	12.78	1.81
	Spatial	16	4.92	4.70
	Hybrid (S-T)	2	3.55	**6.51**
	Temporal	16	18.48	1.25
	Hybrid (T-S)	16	6.76	3.42
Decomposition based Approach	Intra+Inter	4	4.22	5.48
	Spatial	4	2.48	9.32
	Hybrid (S-T)	2	2.19	10.55
	Temporal	4	5.57	4.15
	Hybrid (T-S)	4	2.07	**11.16**

We also derived 16 complex "3-interaction properties" from the MIPS design, which involve interactions of 3 different functional units of different pipeline paths. Besides our decomposition-based clustering method, Table 1 presents the test generation results using various clustering approaches on these 16 properties. The first four clustering methods are introduced in [6], whereas the last one is our proposed decomposition-based clustering technique. Since structural clustering and influence-based clustering have the same clustering results, their test generation time using the same learning approaches is identical. It is important to note that the clustering strategies are independent with the learning mechanisms. This case study shows that different kinds of learning can be beneficial to the same clustering method. The first column of this table indicates the clustering type. The second column presents the type of the learning strategies. In this column, "None" means that no learning is applied. "Intra+Inter" means that the intra-property learning is applied on the base property and the inter-property learning is shared among all other properties [9]. The test generation schemes of "Spatial", "Temporal" "Hybrid (S-T)" and "Hybrid (T-S)" are similar to the "Intra+Inter" approach. The only difference is that the base property adopts the learning from spatial, temporal and hybrid (S-T or T-S) decomposition approaches, respectively (as described in Section 4). The third column presents the number of clusters using different clustering methods. The fourth column gives the overall test generation time using MiniSAT. Since the overlap between different CNFs of properties is too small, the CNF-intersection clustering fails to cluster the properties. That means each cluster contains only one property. Finally, the last column shows the speedup of the overall test generation time compared to the method using MiniSAT. It can be found that, when using MiniSAT for SAT solving together with decomposition based learning, our decomposition-based clustering approach can achieve the best performance compared to other clustering approaches irrespective of learning categories. The combination of the decomposition-based clustering and hybrid decomposition approaches can achieve the maximum speedup (11.16 times improvement over MiniSAT).

To illustrate the effectiveness of our decomposition-based clustering and learning approaches, Table 2 provides more test generation details. Based on the similarity defined in Section 5.1, we clustered the sixteen properties into four clusters, where each cluster has four properties. The first column shows the interaction types. For example, MUL(2) and FADD(1) indicate that the 3-interaction properties involve 2 multiplication units and 1 floating-point addition unit. The second column shows the property index. The third column provides the similarity between the base property and other properties in the same cluster. For the 3-interaction properties, since the temporally decomposed sub-properties cannot directly reflect the similarity information, this column only presents the spatial similarity. The fourth column presents the test generation time using MiniSAT without any learning. Column 5 shows the test generation time using both the Intra+Inter approach introduced in [9]. In each cluster, we adopted intra-property learning for base property checking, and applied inter-property learning for other properties in the same cluster. Columns 6-9 provide the results using our four decomposition methods. In columns 4-9, each cell gives the test generation time using MiniSAT or our modified MiniSAT. In the first row of each property cluster, we provide the SAT solving time for both sub-properties and the

Table 2 Test Generation Results for 3-Interaction Properties Using Decomposition-based Clustering

Interaction Units	Prop. (Tests)	Similarity Ratio	MiniSAT Time (s)	Intra+Inter Time (s)	Spatial Time (s)	Hybrid (S-T) Time (s)	Temporal Time (s)	Hybrid (T-S) Time (s)	Max Speedup
MUL(2) FADD(1)	p_1 *	-	0.82	0.71	0.19+0.10	0.17+0.07	0.66+0.08	0.24+0.02	3.41
	p_2	66.6%	0.78	0.1	0.09	0.09	0.09	0.10	8.67
	p_3	66.6%	0.90	0.10	0.10	0.10	0.09	0.10	10.00
	p_4	66.6%	0.91	0.11	0.10	0.10	0.09	0.10	10.11
Summary	all	-	3.41	1.02	0.58	**0.53**	1.01	0.56	**6.43**
MUL(3)	p_5 *	-	0.87	0.62	0.15+0.14	0.10+0.08	0.56+0.12	0.13+0.08	4.83
	p_6	66.6%	0.87	0.10	0.08	0.10	0.10	0.10	10.88
	p_7	66.6%	17.99/0.86	0.10	0.10	0.10	0.09	0.11	9.50
	p_8	66.6%	1.01	0.10	0.11	0.10	0.11	0.09	11.22
Summary	all	-	3.61	0.92	0.58	**0.48**	0.98	0.51	**7.52**
MUL(2) FADD(1)	p_9 *	-	1.91	0.83	0.25+0.09	0.13+0.07	1.18+0.09	0.20+0.04	9.55
	p_{10}	66.6%	1.94	0.08	0.08	0.08	0.08	0.08	24.25
	p_{11}	66.6%	1.98	0.09	0.08	0.08	0.08	0.08	24.75
	p_{12}	66.6%	1.92	0.09	0.08	0.08	0.08	0.08	24.00
Summary	all	-	7.75	1.09	0.58	**0.44**	1.53	0.48	**17.61**
MUL(3)	p_{13} *	-	1.64	0.89	0.27+0.16	0.14+0.15	1.55+0.20	0.15+0.08	5.66
	p_{14}	66.6%	2.52	0.14	0.14	0.15	0.14	0.13	19.38
	p_{15}	66.6%	1.98	0.08	0.08	0.09	0.08	0.09	24.75
	p_{16}	66.6%	2.30	0.08	0.09	0.09	0.08	0.09	28.75
Summary	all	-	8.44	1.19	0.74	**0.62**	2.05	0.64	**13.61**

* Base property

original property in the form of a+b, where a indicates the sub-property solving time and b denotes the original property solving time. Finally, column 10 provides the max overall speedup over MiniSAT using the following formula:

$$\frac{column4}{Min(Column5, Column6, Column7, Column8, Column9)}.$$

In the summary row, we provide the overall test generation time and the speedup over MiniSAT for each cluster.

Table 2 shows that our decomposition methods outperform both MiniSAT and the Intra+Inter method [9]. In this case, we can find that the spatial method outperforms temporal method. This is because that: i) we adopted spatial similarity to cluster the properties; and ii) the temporally decomposed sub-properties still have large bounds, which can easily result in a large overhead. To further reduce the test generation time, the temporally/spatially decomposed sub-properties can be further spatially/temporally decomposed. Due to the extra learning derived by hybrid decomposition method (T-S or S-T), the overall test generation time is reduced in all the four clusters. It can be found that the hybrid method outperforms both spatial and temporal decomposition methods. The major reason is that the hybrid decomposition approach is promising in reducing the base property checking time. The smaller base property checking time implies less overall test generation time for a cluster of similar properties. The results indicate that, when the hybrid decomposition is applicable to a design, it can achieve the best test generation time. It shows that, for the overall test generation time of these four clusters, our decomposition methods can achieve up to 17.61 times improvement over MiniSAT. Compared to the Intra+Inter method [9], our decomposition methods can achieve up to 2.27 (1.09/0.48=2.27) times improvement by using MiniSAT.

From the above experimental results, we can find that the hybrid methods (i.e., S-T and T-S) can always achieve best performance for the directed test generation of the MIPS design. As shown in Figure 13, for MiniSAT, the hybrid S-T approach consistently outperforms spatial decomposition method, and hybrid T-S approach outperforms the temporal decomposition method. From Table 1, we also can find that the hybrid approaches can achieve the best results (marked in bold font) when employing different property clustering strategies. Specifically, Table 2 shows the test generation details using the decomposition-based clustering approach. From this table, we can consistently find that the hybrid approaches together with property decomposition-based clustering can achieve the best performance. Although hybrid S-T and hybrid T-S approaches adopt different decomposition order, all the above experiment results demonstrate that the test generation time difference between the two hybrid approaches is quite small compared to the solving time of original complex properties. Therefore, for directed test generation, hybrid decompositions are better choices than both spatial and temporal decomposition methods.

6.2 A Stock Exchange System

The on-line stock exchange system (OSES) [11] is a control intensive design which mainly deals with stock order transactions. All these scenarios are described by a UML activity diagram which contains 27 activities and 29 transitions. We extracted the formal model from the specification and transform it to a NuSMV specification. We gener-

ated 18 properties to check all possible stock transactions. Since each transaction involves operation activities (events) in a path of the UML activity diagram, spatial decomposition is not feasible in this case. Therefore, we only adopted the temporal decomposition to reduce the test generation time. In this example, each transaction is temporally decomposed into several stages which specify branch activities along the transaction flow, and for each stage we created a sub-property. Among the 18 properties, we selected 12 complex ones whose CNF clause number is larger than 300000.

Table 3 Test Generation Results for the OSES Example

Clustering Methods	Learning Methods	# of Clust.	Time (second)	Speedup
MiniSAT	None	12	16.51	1.00
Structural Clustering	Inter+Intra	4	4.06	4.07
	Temporal	4	2.58	**6.40**
Textual Clustering	Inter+Intra	4	4.06	4.07
	Temporal	4	2.58	**6.40**
Influence-based Clustering	Inter+Intra	3	5.59	**2.95**
	Temporal	3	7.41	2.23
CNF Intersection Clustering	Inter+Intra	5	3.33	4.96
	Temporal	5	3.14	**5.26**
Decomposition based Approach	Inter+Intra	4	7.85	2.10
	Temporal	4	2.56	**6.45**

Table 3 shows the clustering and test generation results for the 12 properties. It can be found that the combination of the decomposition based clustering and temporal decomposition based learning can achieve the best performance (i.e, 6.45 times improvement over MiniSAT). This is consistent with the results achieved in Section 6.1.

To illustrate the efficiency of our approaches, Table 4 shows the test generation details using our temporal decomposition technique. By using the decomposition based clustering, the 12 properties are clustered into 4 groups, and each group has 3 properties. The first column indicates the transaction type of properties. There are four transaction types in this example: *market buy*, *market sell*, *limit buy* and *limit sale*. The second column indicates the property index. The third column provides the bound information for each property. Columns 4-5 give the variable and clause statistics of the generated CNF files for each property. For each property, we performed the temporal decomposition. Column 6 shows the number of decomposed sub-properties (including the base property). The seventh column presents the test generation time using MiniSAT without any decomposition and learning techniques. Columns 8-9 present the results using the Intra+Inter approach proposed in [9], including the overall test generation time and corresponding improvement over MiniSAT. The final three columns show the test generation time using our temporal decomposition method. Column 10 gives the temporal similarity between the base property and the other properties in a cluster, and columns 11-12 give the test generation time as well as its improvement over MiniSAT. Our approach can achieve 4.37-7.42 times improvement over MiniSAT, and 1.07-5.99 times improvement over the Intra+Inter approach.

7 Conclusions

Since fewer tests can achieve required coverage, directed tests are promising to reduce overall validation effort. However, most automated directed test generation methods, especially for SAT-based BMC, suffer from the state space explosion problem. Although learning techniques are promising to check a cluster of similar properties, the base property cannot benefit from any external learning because it is checked first. To enable efficient test generation for a large number of properties using SAT-based BMC, this paper presented a novel approach that utilizes both property decomposition and learning techniques to reduce the overall test generation time. The case study using both hardware and software designs demonstrated the effectiveness of our method.

Acknowledgments

This work was partially supported by NSF of China 61202103 and 91118007, Innovation Program of Shanghai Municipal Education Commission 14ZZ047, Open Project of SW/HW Co-design Engineering Research Center of MoE 2013001, and Shanghai Knowledge Service Platform Project ZF1213. This work was also partially supported by NSF grants CNS-0746261 and CCF-1218629. A preliminary version [7] of this paper appeared in the proceedings of Design, Automation and Test in Europe (DATE) 2011.

References

1. N. Amla, X. Du, A. Kuehlmann, R. Kurshan, and K. McMillan, "An analysis of SAT-based model checking techniques in an industrial environment," *Proceedings of Correct Hardware Design and Verification Methods (CHARME)*, 2005, pp. 254–268.
2. N. Amla, E. A. Emerson and K. S. Namjoshi, "Efficient decompositional model checking for regular timing diagrams," *Proceedings of Correct Hardware Design and Verification Methods (CHARME)*, 1999, 67–81.
3. A. Biere, A. Cimatti, E. M. Clarke and Y. Zhu, "Symbolic model checking without BDDs," *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 1999, pp. 193–207.
4. P. Bjesse and J. H. Kukula, "Using counter example guided abstraction refinement to find complex bugs," *Proceedings of Design, Automation, and Test in Europe (DATE)*, 2004, 156–161.
5. K. Chang, V. Bertacco and I. L. Markov, "Simulation-based bug trace minimization with BMC-based refinement," *IEEE Transactions on CAD of Integrated Circuits and Systems (TCAD)*, vol. 26, no. 1, pp. 152–165, 2007.
6. M. Chen and P. Mishra, "Functional test generation using efficient property clustering and learning techniques," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 3, pp. 396–404, 2010.
7. M. Chen and P. Mishra, "Decision ordering based property decomposition for functional test generation," *Proceedings of Design, Automation and Test in Europe (DATE)*, 2011, pp. 167-172.

Table 4 Test Generation Results for OSES Design Using Decomposition-based Clustering

Transac. Type	Prop. (Tests)	Bound	CNF Size		Decom. #	MiniSAT	Intra + Inter		Temporal Decomposition		
			Vars.	Cls.		Time (s)	Time (s)	Speedup	Similarity	Time (s)	Speedup
Market Sale	p_1 *	10	119098	362519	4	1.69	0.36	4.69	90%	0.16	10.56
	p_2	10	146024	362500	4	0.16	0.13	1.23	90%	0.06	2.67
	p_3	10	118988	362519	4	2.13	0.31	6.87	90%	0.39	5.46
Summary	all	-	-	-	-	3.98	0.80	4.98	-	0.61	**6.52**
Market Buy	p_4 *	10	119371	326326	4	0.64	0.34	1.88	90%	0.28	2.29
	p_5	10	147159	398680	4	5.43	5.17	1.05	90%	0.42	3.91
	p_6	10	119191	326326	4	1.05	0.24	4.38	90%	0.26	4.38
Summary	all	-	-	-	-	7.12	5.75	1.24	-	0.96	**7.42**
Limited Sale	p_7 *	11	131414	398741	5	0.18	0.26	0.69	91%	0.18	1.00
	p_8	11	131546	398741	5	2.20	0.21	10.48	91%	0.11	20.00
	p_9	11	131678	398741	5	0.67	0.25	2.68	91%	0.16	4.19
Summary	all	-	-	-	-	3.05	0.72	4.24	-	0.45	**6.78**
Limited Buy	p_{10} *	11	156144	398743	5	0.68	0.26	2.62	91%	0.14	4.85
	p_{11}	11	156276	398743	5	0.26	0.18	1.44	91%	0.19	1.37
	p_{12}	11	156012	398743	5	1.42	0.14	10.14	91%	0.21	6.76
Summary	all	-	-	-	-	2.36	0.58	4.07	-	0.54	**4.37**

* Base property

8. A. Li and M. Chen, "Efficient self-learning techniques for SAT-based test generation," *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2012, pp. 197-206.
9. M. Chen and P. Mishra, "Property learning techniques for efficient generation of directed tests," *IEEE Transactions on Computers*, vol. 60, no. 6, pp. 852–864, 2011.
10. P. Mishra and N. Dutt, "Graph-based functional test program generation for pipelined processors" *Proceedings of Design, Automation and Test in Europe (DATE)*, 2004, pp. 182–187.
11. M. Chen, X. Qin, H. Koo and P. Mishra, *System-Level Validation: High-Level Modeling and Directed Test Generation Techniques*. Springer, 2012.
12. E. Clarke, O. Grumberg and D. Peled, *Model Checking*. MIT Press, 1999.
13. E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.
14. V. Durairaj and P. Kalla, "Variable ordering for efficient SAT search by analyzing constraint-variable dependencies," *Proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2005, pp. 415–422.
15. N. Eén and N. Sörensson, "An extensible SAT-solver," *Proceedings of Theory and Application of Satisfiability (SAT)*, 2003, pp. 502–518.
16. S. Eggersglüß and R. Drechsler, "Efficient data structures and methodologies for SAT-Based ATPG providing high fault coverage in industrial application," *IEEE Transactions on CAD of Integrated Circuits and Systems (TCAD)*, vol. 30, no. 9, pp. 1411–1415, 2011.
17. I. G. Harris, "Fault models and test generation for hardware-software covalidation," *IEEE Design and Test*, vol. 20, no. 4, pp. 40–47, 2003.
18. H. Jin and F. Somenzi, "An incremental algorithm to check satisfiability for bounded model checking," *Proceedings of International Workshop on Bounded Model Checking*, 2004, pp. 51–65.
19. H. Koo and P. Mishra, "Functional test generation using design and property decomposition techniques," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 8, no. 4, 2009.
20. P. Mishra, A. Shrivastava and N. Dutt, "Architecture description language (ADL)-driven software toolkit generation for architectural exploration of programmable SOCs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)* , vol. 11, no. 3, 626-658, 2006.
21. J. Marques-Silva and K. Sakallah, "Grasp: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999.
22. J.P. Marques-Silva and K.A. Sakallah, "The impact of branching heuristics in propositional satisfiability" *Proceedings of the 9th Portuguese Conference on Artificial Intelligence*, 1999, pp. 62–74.
23. D. A. Mathaikutty, S. K. Shukla, S. V. Kodakara, D. Lilja and A. Dingankar, "Design fault directed test generation for microprocessor validation," *Proceedings of Design, Automation and Test in Europe (DATE)*, 2007, pp. 761–766.
24. R. Meyer, J. Faber, J. Hoenicke and A. Rybalchenko, "Model checking duration calculus: a practical approach," *Formal Aspects of Computing*, vol. 20, pp. 481–505, 2008.
25. *MiniSAT*. http://minisat.se/.
26. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik, "Chaff: Engineering an efficient SAT solver," *Proceeding of Design Automation Conference (DAC)*, 2001, pp. 530–535.
27. *NuSMV*. http://nusmv.irst.itc.it/.
28. O. Shtrichman, "Tuning SAT checkers for bounded model checking," *Proceedings of Computer Aided Verification (CAV)*, 2000, pp. 480–494.
29. O. Strichman. "Pruning techniques for the sat-based bounded model checking problem," *Proceedings of Correct Hardware Design and Verification Methods (CHARME)*, 2001, pp. 58–70.
30. D. Tille, S. Eggersglüß and R. Drechsler, "Incremental solving techniques for SAT-based ATPG," *IEEE Transactions on CAD of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 7, pp. 1125–1130, 2010.
31. C. Wang, H. Jin, G. D. Hachtel, and F. Somenzi, "Refining the SAT decision ordering for bounded model checking," *Proceedings of Design Automation Conference (DAC)*, 2004, pp. 535–538.
32. *zChaff*. http://www.princeton.edu/∼chaff/zchaff.html.
33. L. Zhang, M. R. Prasad and M. S. Hsiao, "Incremental deductive & inductive reasoning for SAT-based bounded model checking," *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, 2004, pp. 502–509.