Data Mining Based Decomposition for Assume-Guarantee Reasoning

He Zhu*, Fei He*, William N. N. Hung†, Xiaoyu Song‡ and Ming Gu*
* Tsinghua National Laboratory for Information Science and Technology
Key Laboratory for Information System Security, Ministry of Education
School of Software, Tsinghua University, Beijing, China
Email: hefei@tsinghua.edu.cn
†Synopsys Inc., Mountain View, California, USA
‡Department of ECE, Portland State University, Oregon, USA

***Abstract*—Automated compositional reasoning using assume-guarantee rules plays a key role in large system verification. A vexing problem is to discover fine decomposition of system contributing to appropriate assumptions. We present an automatic decomposition approach in compositional reasoning verification. The method is based on data mining algorithms. An association rule algorithm is harnessed to discover the hidden rules among system variables. A hypergraph partitioning algorithm is proposed to incorporate these rules as weight constraints for system variable clustering. The experiments demonstrate that our strategy leads to order-of-magnitude speedup over previous.**

I. Introduction

Model checking is a popular formal verification technique. However, it suffers from the state explosion problem. Model checking of large scale systems such as microprocessors is known to be extremely difficult, and the problem will only get worse as the complexity scales exponentially with multi-core or distributed systems. To avoid the blow-up, people tend to utilize a divide-and-conquer strategy, i.e., decomposing large problems into multiple pieces and work on them separately. Assume-guarantee reasoning [1], [2] has been proposed as a useful technique to enhance model checking, especially when there are mutual dependencies between components. In this approach, individual component is verified separately according to the assumptions on its environment (i.e. other components), and then this assumption must be discharged by the rest of the system.

Assume-guarantee reasoning has been studied by many researchers for a long time [3], [4]. In this paper, we mainly use the following assume-guarantee rule [4]:

$$\frac{M_1||A \models \varphi \quad \text{(n1)} \qquad M_2 \sqsubseteq A \quad \text{(n2)}}{M_1||M_2 \models \varphi}$$

The rule above says that if M_1 combined with an assumption A satisfies the property φ (in n1 step), and A is further an abstraction of M_2 (in n2 step), then we conclude the system composed of M_1 and M_2 satisfies φ.

This work was supported in part by the Chinese National 973 Plan under grant No. 2004CB719400, the NSF of China under grants No. 60553002, 60635020, 60903030 and 90718039.

Assume-guarantee reasoning requires that correct assumptions be provided, which imposes additional hurdles for this method. [5] proposed a new method based on language learning to automatically learn assumptions from an alphabet over I/O (Input/Output) variables between components. However, this method may be insufficient when the construction of assumption needs to exhaust a large alphabet. In fact, the natural structure of components may be not applicable for system or modular verification. Hence, it is important to explore various decomposition boundaries that are independent of the original modular structure of the system.

Inspired by the work in [6], we present an effective automatic decomposition approach. Given a state transition system, the proposed method decomposes it into n sub-modules. We want to find a partition that converges to smaller assumption construction and early verification termination. In [6], the objective function is defined as partitioning the system to minimize the number of I/O variables between modules. In this paper, we modify the goal not only to reduce the I/O variables but also to enhance each module's cohesion.

If we define state variables' relation as distance which is a common measure concept in data mining, our partitioning goal is to minimize the intra-cluster distances within each module and to maximize the inter-cluster distances between modules. To quantify the distances we propose to use association rule mining to reveal the hidden variable implications and provide qualified information for decomposition. In this way, we find a partition which put state variables frequently communicate to each other together.

To evaluate our method, we incorporate the NuSMV model checker [7] with a synchronous model checker SYMODA [8]. Experimental results show some encouraging improvements of our approach over previous [6].

II. Preliminaries

In this section, we define the concept of decomposition of a state transition system and composition which are used through our paper. Our formalisms use notations similar to [6].

Given any set of state variables X, for each $x \in X$, x is a typed variable defined over a finite domain of values D_x. An assignment $s : X \to V$ maps each x in X to one certain value

978-1-4244-4966-8/09/$25.00 ©2009 IEEE

v in D_x. We write boolean formula $\varphi(s)$ for a property over X and we say $\varphi(s)$ is true if the assignment $s(X)$ satisfies φ. We use $Var(\varphi)$ to denote the set of state variables appearing in φ in our context.

Formally, a state transition system S is a tuple $\langle X, Init_X, T_X\rangle$, where

1) X is a set of state variables of the system.
2) $Init_X = \bigwedge_{x\in X} Init_x(X)$ is an initial predicate over X where $Init_x(X)$ is an initial predicate for variable x.
3) $T_X = \bigwedge_{x\in X} T_x(X, X')$ is a transition predicate over X and X', where X' denotes the next statues of X and $T_x(X, X')$ is the transition predicate for variable x.

A state transition system S may comprise several sub-modules. A sub-module M_i is a tuple $\langle X_i, I_{X_i}, O_{X_i}, Init_{X_i}, T_{X_i}\rangle$, where

- $X_i \subseteq X$ is a set of state variables controlled by M_i.
- I_{X_i} is a set of input variables that are controlled by some other modules and are readable by M_i. Note I_{X_i} is disjoint from X_i.
- $O_{X_i} \subseteq X_i$ is a set of output variables that are controlled by M_i and are accessible by some other modules.
- $Init_{X_i} = \bigwedge_{x\in X_i} Init_x(X)$ is an initial predicate over $X_i \cup I_{X_i}$.
- $T_{X_i} = \bigwedge_{x\in X_i} T_x(X, X')$ is a transition predicate over $X_i \cup X_i' \cup I_{X_i}$.

We use IO_{X_i} to denote the input and output variables of M_i, i.e. $IO_{X_i} = I_{X_i} \cup O_{X_i}$.

The semantic of a state transition system is the set of runs it exhibits. A run of S is a sequence $s_0, s_1, \ldots$ of states where each s_i represents a variable assignment mapping each value in X to its domain, such that $Init_X(s_0)$ holds and for every $j \geq 0$, $T_X(s_j, s_{j+1})$ holds.

Given a state transition system $S(X, Init_X, T_X)$ and an integer n, decomposition problem is to decompose S into n sub-modules $M_i(X_i, I_{X_i}, O_{X_i}, Init_{X_i}, T_{X_i})$, $1 \leq i \leq n$, where $X = \bigcup_{1\leq i\leq n} X_i$ and $\bigwedge_{i,j=1,\ldots,n}^{i\neq j} X_i \cap X_j = \emptyset$.

Given a property φ over $\cup_{1\leq i\leq n} IO_{X_i}$, let $S \models \varphi$ denote φ holds in S, i.e., for each run $s_0, s_1, \ldots$ of S, $\varphi(s_0), \varphi(s_1), \ldots$ holds. According to [6], there is

$$(S \models \varphi) \Leftrightarrow (M_1 || \ldots || M_n \models \varphi).$$

III. Problem Formulation

In this section, we formulate the system decomposition problem into a hypergraph partitioning problem.

A. Hypergraph Partitioning

A hypergraph is a special graph, which can be defined as $G(V, E)$, where

- V is a set of vertices.
- E is a set of hyperedges that connect arbitrary number of vertices. For a hyperedge $e \in E$, let $vertex(e)$ denote the set of vertexes connected by e.

A weighted hypergraph is one that assigns each hyperedge a numerical value. More formally, a weighted hypergraph is a triple $G(V, E, W)$, where V, E are defined identically as in normal hypergraph, and $W : E \to \mathbb{R}$ defines a weight value for each hyperedge.

The K-way hypergraph partitioning problem $P(G, K, \lambda)$ is to partition the original hypergraph G into K parts by clustering the vertexes of G into K disjoint subsets $V_1, \ldots, V_K$, such that $V = \bigcup_{1\leq i\leq K} V_i$. After partitioning, a hyperedge may span different parts. The concept of connectivity λ helps to distinguish whether a hyperedge crosses some parts or lies inside one single part. Connectivity λ_j of a hyperedge $e_j \in E$ is 1 if the number of different parts e_j crosses is greater than 1; otherwise λ_j is 0. A hyperedge e_j is called as a hyperedge-cut if $\lambda_j = 1$.

The K-way hypergraph partitioning algorithm tries to find a partition to minimize $C_p = \sum_{e_j\in E} w_j \cdot \lambda_j$, where w_j is the weight of hyperedge e_j. It also imposes a constraint that the cardinality of each set V_i is bounded by $|V|/(cK) \leq |V_i| \leq |V|(c/K)$ where $|V|$ is the number of vertexes contained in V. The imbalance tolerance c is a parameter outside the partitioning algorithm. A large value of c causes imbalance clusters while a small value of c makes each V_i roughly the same size.

Many researchers have studied hypergraph partitioning intensively and there already exists fast algorithms and tools. Among them we choose hMETIS [9].

B. Decomposition as Hypergraph Partitioning

Given a state transition system $S(X, Init_X, T_X)$, we say there exists variable dependency between x and x', if x' appears in $T_x(X, X')$. Formally, let Y denote the set of variables appearing in $T_x(X, X')$, the variable dependencies of x is the power set of Y.

Given a hypergraph $G(V_X, E)$, we call a vertex v_y is an adjacency of another vertex v_x if it is connected to v_x by some hyperedge in E. The vertex adjacencies of v_x is a set in which each element is a set of vertexes connected by a hyperedge involving v_x i.e., $\bigcup_{e\in E}\{vertex(e)|v_x \in vertex(e)\}$.

Based on above points, if the state variables in X are linked to the vertices in V_X, the power set of variable dependencies of a variable x can be modeled as hyperedges involving v_x. Furthermore, the power sets of variable dependencies certainly have different occurrence in system's state transitions. Intuitively some sets of variables occur more times than the other sets in T_X. If one applies certain precise measure to assign each subset of variable dependencies a numerical value, the hypergraph $G(V_X, E)$ can be upgraded to a weighted hypergraph $G(V_X, E, W)$.

IV. Data Mining Based Decomposition

The flow of our method is shown in Fig. 1. In essence, a state transition system is modeled by a weighted hypergraph then the partition of hypergraph model uniquely determines the system decomposition.

The challenge in our method is how to measure weight values for each hyperedge in the hypergraph model. In this paper, we propose a novel method by applying data mining algorithm to generate these weight values.

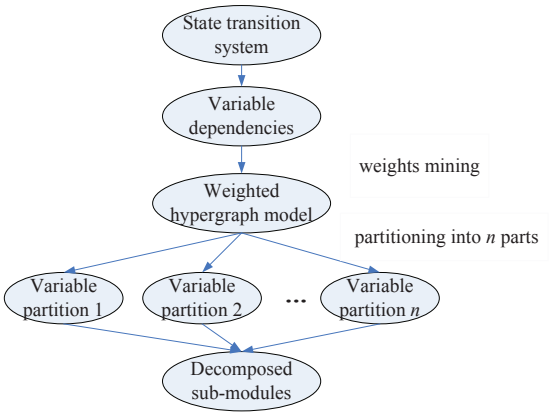

Fig. 1. Our Decompsition Method

A. Association Rule Mining

Association rule mining [10] aims to discover the hidden patterns and correlations from a large dataset. Given an itemset $I = \{I_1, I_2, \ldots, I_n\}$ and a set of transactions T where $t \in T$ is a subset of I. Let X, Y denote two disjoint subsets of I, i.e. $X, Y \subseteq I$, $X \cap Y = \emptyset$. An association rule is an implication in the form of $X \Rightarrow Y$. We define a function $f : 2^I \rightarrow \mathbb{N}$ mapping each subset of I to a natural number which represents the number of transactions (in T) containing this subset.

An association rule is defined by two important notions, the support and confidence of the rule. The support of a rule is defined as $sup(X \cup Y) = f(X \cup Y)/|T|$, i.e. the percentage of transactions that contain $X \cup Y$, where $|T|$ is the total number of transactions contained in T. While the confidence of a rule is defined as $conf(X \cup Y) = f(X \cup Y)/f(X)$, i.e. the ratio of number of transactions that contain $X \cup Y$ to the number of transactions that contain X.

Given a set of transactions T, association rule mining first finds frequent itemsets by generating all combinations of items in I with their supports above user supplied $minsup$ threshold. For a frequent itemset fi, all the association rules with the type $X \Rightarrow fi - X$ $(X \subset fi)$ are outputted if their confidences are above user supplied $minconf$ threshold.

There are many association rule mining algorithms, among which Apriori [10] is a fast and popular algorithm. We use the Apriori algorithm implemented in [11].

B. Weights Mining for Hypergraph Model

Given a state transition system $S(X, Init_X, T_X)$, we convert it into a weighted hypergraph $G(V_X, E, W)$. There are two steps in mining hyperedge weights from S.

1) First step: collect variable dependencies as transactions: In this step, we generate the transactions as the input of association rule mining algorithm by deriving variable dependencies from the system's state transitions.

For each $x \in X$, there is a transaction t_x containing variable x and a variable y belongs to t_x if $y \in Var(Init_x) \cup Var(T_x)$. Intuitively, t_x represents the corresponding variable x and all variables that are read by x. The transactions for S is finally the set of $V_T = \{t_x | x \in X\}$. We apply Apriori algorithm to transactions V_T to generate frequent itemsets and association rules upon X. Suppose the generated frequent itemset is $E_{f_i} = \{f_i | f_i \subseteq t_x \wedge t_x \in V_T \wedge sup(f_i) > minsup\}$.

2) Second step: combine association rules with weighted hypergraph model: In this step, we provide a precise measure to assign each hyperedge a numerical value which is explained in section III.

Each variable x in X corresponds to a vertex v_x in V_X such that $V_X = \{v_x | x \in X\}$. For each frequent itemset $f_i \in E_{f_i}$, we make a hyperedge e_{f_i} and a vertex v_x is connected by e_{f_i} if its corresponding variable $x \in f_i$. The weight of the hyperedge e_{f_i} is the average of confidences of all the association rules derived from the frequent itemset f_i, i.e., $W(e_{f_i}) = \forall_{X \subset f_i} \overline{\{conf(X, f_i - X) | conf(X, f_i - X) > minconf\}}$.

We apply hypergraph partitioning algorithm to the generated weighted hypergraph model. After partitioning, we classify variables $X_1, \ldots, X_n$ according to the vertex partitioning result $V_1, \ldots, V_n$. Then we use the results in section II to build each sub-module $M_i(X_i, I_{X_i}, O_{X_i}, Init_{X_i}, T_{X_i})$, where $1 \leq i \leq n$.

Note there are two intrinsic differences between our method and the counterpart of [6] in modeling a hyperedge. First, given a variable x, let x and all the variables that read x compose a set Y; let x and all the variables that are read by x compose Z. The method in [6] models the whole set Y as a hyperedge; while our method models the subsets of Z as corresponding hyperedges. Second, our method incorporates numerical value into hyperedge by using association rules.

Association rule mining is applied to help us find groups of variables with dense dependencies. The generated rules with high confidence indicate the variables referenced by the rules have close dependency between them. While the rules with low confidence indicate the variables referenced by the rules have sparse correlation. Thus in our weighted hypergraph model, heavy weighted hyperedges are more likely to be partitioned in the same cluster while light weighted hyperedges are more likely to be the hyperedge-cuts.

According to the assume-guarantee rule, we need to verify $M_1 || A \models \varphi$ (n1) and $M_2 \sqsubseteq A$ (n2). Suppose we partition the system model into two modules and assign M_1 to be the module which manages almost or all the variables in $Var(\varphi)$. As M_1 has compact state space related to φ, an appropriate assumption A which satisfies $M_1 || A \models \varphi$ (n1) can then be constructed with fewer interventions from M_2. In this case, A avoids to be strengthened too much times to prevent M_1 from getting to the error state and the state space to be visited is reduced. On the other hand, as M_1 and M_2 have relatively few communications, $M_2 \sqsubseteq A$ (n2) can also be verified readily because we have very loose bound on assumption A and A is weak enough in this case.

V. Experiment Results

We have implemented our method and integrated it with the symbolic model checkers NuSMV [7] and SYMODA [8]. We use NuSMV as a front-end to parse NuSMV model file and

TABLE I
EXPERIMENTAL RESULTS

Example	var	Weighted Hypergraph						Unweighted Hypergraph						General
		minsup	c	IO	Mem	Candi	Total	minsup	c	IO	Mem	Candi	Total	Total
s1a	23	0.05	1.0	2	0.03	0.17	0.32	0.05	1.0	2	0.04	0.15	0.31	15.77
s1b	25	0.05	1.0	6	0.14	0.07	0.49	0.05	1.0	6	0.20	0.12	0.60	16.03
msi3	61	0.05	1.8	17	0.90	0.22	2.81	0.05	1.8	19	1.20	0.66	3.53	10.23
msi5	97	0.05	1.8	24	2.31	0.40	5.86	0.05	1.8	32	3.62	1.41	8.81	27.17
msi6	121	0.05	1.8	27	3.42	0.55	9.69	0.05	1.8	33	5.16	1.36	12.11	43.80
syncarb10	74	0.01	1.0	32	26.85	23.18	76.13	0.01	1.0	33	68.94	46.18	129.2	TO
peterson	9	0.01	1.0	7	0.18	0.31	0.65	0.01	1.0	7	19.89	70.34	113.8	27.67
guidance	76	0.05	1.0	37	11.19	4.11	19.93	0.05	1.0	13	1.03	2.00	4.11	18.75

generate sub-modules by our weighted hypergraph model. We use Symoda, a synchronous modular analyzer, to verify the above generated decomposed modules. All experiments were performed on a 1GB memory 3 GHz Intel machine running Ubuntu Linux System.

All benchamrks are obtained form the public places [6], [12], [13]. Table I compares the decomposition results by verification time for our weighted hypergraph model aided, unweighted hypergraph model [6] aided verification and general model checking without decomposition. All examples are partitioned into two components.

In Table I, $minsup$ is a parameter for Apriori. The other threshold $minconf$ is always assigned to its default value (0.8). c is a parameter for hMETIS. We try five different values (i.e. 1.0, 1.2, ..., 1.8) and pick one that is suitable for each benchmark. var represents the number of variables. IO gives the input/output variables between sub-modules. Note we count up the number of typed variables instead of boolean variables. Except $guidance$, all benchmarks have more than (or about) one hundred boolean variables. Mem and $Candi$ are the total membership and candidate query time respectively. $Total$ gives the overall verification time. All times are in seconds. TO means verification doesn't terminate in 1200s.

It is obvious that assume-guarantee reasoning with appropriate decomposition performs much better than the general modular method without decomposition as the decomposition either reduced verification time or converted infeasible problem into feasible one. In most cases, our solution that incorporates association rule mining and weighted hypergraph model obtains significantly better result than the approach in [6] which used unweighted hypergraph partitioning method solely. In $peterson$ and $syncarb10$ examples, our solution achieves dramatic improvements.

However our method obtained negative result as shown in $guidance$ example. It is probably because the transition relations between the variables in original $guidance$ model are so sparse, i.e., a large portion of variables in this example only interact with no more than two variables directly. In this case, Apriori algorithm can't mine meaningful rules but leads to inappropriate decomposition.

VI. CONCLUSION

In this paper, we presented an automatic method to decompose system model into simpler sub-modules. The use of association rule mining improves the partitioned modules' inner cohesion and the confidence of association rules restricts to put related variables together. As a result, inter-connections between modules are reduced and each module is compact. The experiments demonstrate our method not only leads to good decomposition but also improves verification scalability in the context of assume-guarantee reasoning.

We used non-circular assume-guarantee rule in this paper. In future, we plan to extend our framework to make it scalable to circular assume-guarantee rules [14] too. Furthermore, data mining remains an active subject; we also intend to apply some other classification algorithms to find even better system decompositions contributing to simple assumptions.

REFERENCES

[1] A. Pnueli, "In transition from global to modular temporal reasoning about programs," *Logics and models of concurrent systems*, 1985.

[2] C. B. Jones., "Specification and design of (parallel) programs," in *Proceedings of the IFIP 9th World Congress*, 1983, p. 321332.

[3] T. Henzinger, S. Qadeer, and S. Rajamani, "You assume, we guarantee: Methodology and case studies," in *Proc. Computer Aided Verification (CAV)*, 1998, pp. 521–525.

[4] K. S. Namjoshi and R. J. Trefler, "On the completeness of compositional reasoning," in *Proc. Computer Aided Verification (CAV)*, 2000, pp. 139–153.

[5] J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu, "Learning assumptions for compositional verification," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2003.

[6] W. Nam and R. Alur, "Learning-based symbolic assume-guarantee reasoning with automatic decomposition," in *Automated Technology for Verification and Analysis (ATVA)*, 2006, pp. 170–185.

[7] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking," in *Proc. Computer Aided Verification (CAV)*, 2002, pp. 359–364.

[8] N. Sinha and E. Clarke, "Sat-based composition verification using lazy learning," in *Proc. Computer Aided Verification (CAV)*, 2007, pp. 3–5.

[9] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain," *IEEE Trans. VLSI Systems*, vol. 7, no. 1, pp. 69–79, 1999.

[10] R. Agrawal, T. Imielienski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," in *Proc. Conf. Management of Data*, 1993, pp. 207–216.

[11] C. Borgelt, "Efficient implementations of apriori and eclat," in *Proc. ICDM workshop on frequent itemset mining implementations*, 2003.

[12] NuSMV examples: the collection. [Online]. Available: http://nusmv.irst.itc.it/examples/examples.html

[13] VIS verification benchmarks. [Online]. Available: ftp://vlsi.colorado.edu/pub/vis/vis-verilog-models-1.0.tar.gz

[14] H. Barringer, D. Giannakopoulou, and C. S. Pasareanu, "Proof rules for automated compositional verification," in *2nd Workshop on Specification and Verification of Component-Based Systems (ESEC/FSE)*, 2003.