VCS: A Verifier for Component-Based Systems*

Fei He[1,2,3], Liangze Yin[1,2,3], Bow-Yaw Wang[4], Lianyi Zhang[1,2,3], Guanyu Mu[1,2,3], and Wenrui Meng[1,2,3]

[1] Tsinghua National Laboratory for Information Science and Technology (TNList)
[2] School of Software, Tsinghua University
[3] Key Laboratory for Information System Security, Ministry of Education, China
[4] Academia Sinica, Taiwan

Abstract. This paper presents the VCS verification tool for the BIP modeling language. The tool admits sophisticated interactions specified in BIP models. Particularly, private variables in components can be updated by user-defined interactions. On the verification back-end, the BIP models are formulated as transition systems. Several efficient algorithms are proposed for verification of transition systems on safety properties. Experimental results show very promising performance of VCS. It runs several magnitudes faster than NuSMV for a variety of examples.

1 Introduction

Component-based design has attracted significant interests from both industry and academy. Recent modeling languages such as AADL [1] and BIP [2] offer mechanisms for specifying sophisticated interactions among components. In the BIP language, for instance, components expose their private variables through ports, and the exposed private variables can be updated during user-specified interactions. The feature allows users to specify intricate interactions among components, but also complicates the semantics of the modeling language. Implementing verification tools for the BIP language can be demanding.

VCS [1] is a verification tool for models specified in the BIP language. In contrast to the existing BIP model checker DFinder [3], VCS allows to specify interactions with data transfer among components. Users are able to fully exploit features of the BIP language in their models. Additionally, the VCS tool verifies properties specified in the Computation Tree Logic (CTL) as well as deadlock freedom on BIP models.

To the best of our knowledge, the VCS tool is the first BIP model checker which admits interactions with data transfer. An efficient SAT-based verification engine is implemented. Experiments show very promising performance of the tool in verification of component-based systems.

* This work was supported by the National 973 Plan (No. 2010CB328003), the NSF of China (No. 61272001, 60903030, 91218302), the Chinese National Key Technology R&D Program (No. SQ2012BAJY4052), the NSC 101-2221-E-001-007, and the Tsinghua University Initiative Scientific Research Program.

[1] `http://code.google.com/p/bip-vcs/`

D. Van Hung and M. Ogawa (Eds.): ATVA 2013, LNCS 8172, pp. 478–481, 2013.

2 Model Representation

The BIP model is defined in a hierarchical way. The model behaviors are described in atomic components. A compound component consists of a collection of (atomic or compound) components and connectors. Each connector can have temporary variables [2] to specify the interactions with data transfer among components. The BIP model is the topmost compound component.

In VCS, each atomic component is encoded as a transition system (X, I, T), where X is a set of variables, I is the initial predicate and T is the transition predicate. Let T^I be the transition predicate related to the internal transitions only. Given a hierarchical BIP model H, the tool first transforms the input model into a flattened BIP model [4]. The flattened BIP model contains only atomic components $A_i = (X_i, I_i, T_i)$ $(1 \leq i \leq N)$ and connectors C_j $(1 \leq j \leq M)$. Let F_j be the symbolic representation of the connector C_j. The hierarchical BIP model H is thus a transition system (X_H, I_H, T_H), where

- $X_H = \bigcup_{i=1}^{N} X_i$;
- $I_H = \bigwedge_{i=1}^{N} I_i$;
- $T_H = \bigvee_{i=1}^{N}(T_i^I \wedge \bigwedge_{k \neq i}(X'_k = X_k)) \vee \bigvee_{j=1}^{M}(F_j \wedge \bigwedge_{k \notin dom(C_j)}(X'_k = X_k))$, where $dom(C_j)$ gives the indices of atomic components in C_j.

3 Verification Algorithms

In traditional settings, the transition systems are interpreted as state machines, and then verified by model checking algorithms (either explicit or symbolic). However, during this interpretation, much useful information implied in the transition system is lost. We propose several efficient techniques to utilize such information to improve the model checking for transition systems.

Macro Step-Based Verification: Given a transition system, we distinguish the set of transitions which may lead the property from true to false, called property-sensitive transitions. Each search step of bounded model checking is extended to a macro step, which consists of exactly one property-sensitive transition and any number of other transitions. Moreover, we employ an algorithm to eliminate all loops among property-sensitive transitions in the model. Then we are able to formulate the model checking problem as a Boolean SAT formula. We call this technique *macro step-based verification*.

Variable Decision Heuristic: We propose in [5] to utilize the structure information hidden in a transition system during model checking. We define a *transition variable* for each transition in the model. During the SAT solving, the transition variable is assigned higher priority than other variables to be chosen as the decision variable. Among the many transition variables, we follow the structure of the transition system to assign their priorities. In such a way, the structure information is utilized to guide the search process of a SAT solver.

[2] The current version of DFinder does not support this feature.

Incremental Verification: The proof for temporal induction [6] consists of two parts: the base case and the induction step. Both parts generate a series of SAT problems. We can exploit the symmetry and similarities in the series for incremental SAT verification. We show that, under certain conditions: (1) conflict clauses can be shared among SAT problems within each sequence; (2) conflict clauses can be shared between these two sequences; (3) after shifting or reversing the time steps, the transformed clauses can also be shared. Compared to existing works, our algorithm explores much bigger degree of clause sharing.

4 Experimental Results

The VCS tool is implemented in C++. All experiments are conducted on a computer with a 2.53GHz Intel Core2 Duo CPU with 2GB memory.

Experimental results for six examples are reported. Three examples are from real systems in industry, including the data processing unit (DPU) used in a space vehicle [7], the gate control system (GCS) used in the stage of LingShan Buddhist Palace in Jiangsu, China [8], and the message transmission protocol (MTP) used in the train communication network. Three examples are originated from public websites or literature, including the ATM system [3], the dining philosophers problem (DPP) [3], and the automatic callback system (ACS) [9].

Note the industrial examples exploit sophisticated interactions among components, they cannot be verified by DFinder [3]. We chose to use NuSMV(version 2.5.3) to perform the comparison. Two state-of-the-art SAT-based algorithms (*Een-sorensson* and *Zigzag* [6]) implemented in NuSMVare tested. For each model, we test both algorithms and report the better one for NuSMV.

Experimental results are listed in Table 1. In the table, *step* give the steps for standard bounded model checking (including NuSMVand *Std*) to find a bug, while $step_m$ give the steps for our macro step-based verification to find a bug. We observe in all cases the value of $step_m$ is much less than that of *step*. This is reasonable since a macro step may involve several transitions in the model. The VCS tool can be configured with different settings, where *Std* stands for the standard bounded model checking, *Mco* stands for the macro step-based verification, Mco^+ stands for *Mco* plus the incremental verification technique, and Mco^{++} stands for Mco^+ plus the variable decision heuristic. All runtimes are reported in seconds. The label "-" indicates the checker cannot get a conclusive answer in 900 seconds.

For all cases, *Mco*, Mco^+ and Mco^{++} run several magnitudes faster than either *Std* or NuSMV, especially when the problems scale up. For the industrial examples DPU, GCS and MTP, which involve sophisticated behaviors and interactions, Mco^{++} runs fastest. For other examples ACS, ATM and DPP, which contain no local variables, the variable decision heuristic is useless, thus Mco^+ runs fastest.

[3] `http://www-verimag.imag.fr/DFinder.html?lang=en`

Table 1. Experimental Results for Macro-Step Verification

Model	Prop	NuSMV		VCS				
		step	time	$step_m$	*Std*	*Mco*	Mco^+	Mco^{++}
DPU	P1	24	2.23	9	1.95	0.52	0.16	**0.1**
DPU	P2	26	2.43	9	2.65	0.7	0.23	**0.08**
DPU	P3	32	5.79	10	7.05	1.07	0.32	**0.17**
GCS	P1	46	7.4	18	28.74	2.05	0.31	**0.22**
GCS	P2	54	28.55	21	127.96	7.63	1.26	**0.42**
MTP	P2	30	-	13	143.92	17.58	13.68	**7.11**
MTP	P3	30	-	13	199.7	18.41	24.37	**4.11**
MTP	P4	33	-	15	199.9	41.03	38.57	**10.32**
MTP	P5	30	-	13	111.19	8.27	4.21	**3.92**
ATM6	P1	13	334.28	6	43.81	0.13	**0.04**	1.83
ATM8	P1	-	-	8	-	1.46	**0.97**	-
ATM10	P1	-	-	10	-	11.13	**11.21**	-
DPP10	P1	10	9.03	10	6.75	1.09	**0.77**	141.85
DPP11	P1	11	61.5	11	29.21	4.11	**2.65**	-
DPP12	P1	12	-	12	152.53	22.6	**18.12**	-
ACS3	P1	24	2.07	6	63.87	0.03	**0.01**	0.04
ACS5	P1	-	-	10	-	0.56	**0.13**	55.85
ACS7	P1	-	-	14	-	4.44	**0.62**	-
ACS9	P1	-	-	18	-	29.9	**9.61**	-

References

1. Feiler, P.H.: The architecture analysis & design language (AADL): An introduction. Technical report, CMU/SEI-2006-TN-011 (2006)
2. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in bip. In: SEFM 2006, pp. 3–12. IEEE (2006)
3. Bensalem, S., Bozga, M., Nguyen, T.-H., Sifakis, J.: D-finder: A tool for compositional deadlock detection and verification. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 614–619. Springer, Heidelberg (2009)
4. Bozga, M., Jaber, M., Sifakis, J.: Source-to-source architecture transformation for performance optimization in bip. IEEE Transactions on Industrial Informatics 5(4), 708–718 (2010)
5. Yin, L., He, F., Gu, M.: Optimizing the sat decision ordering of bounded model checking by structural information. In: Proceedings of the 7th International Symposium on Theoretical Aspects of Software Engineering, TASE (2013)
6. Eén, N., Sorensson, N.: Temporal induction by incremental sat solving. Electronic Notes in Theoretical Computer Science 89(4), 543–560 (2003)
7. Wan, H., Huang, C., Wang, Y., He, F., Gu, M., Chen, R., Marius, B.: Modeling and validation of a data process unit control for space applications. In: Embedded Real Time Software and Systems (2012)
8. Wang, R., Zhou, M., Yin, L., Zhang, L., Gu, M., Sun, J., Bozga, M.: Modeling and validation of plc-controlled system: A case study. Technical report, Tsinghua University (2011)
9. Cha, G., Gu, T.: Formal Analysis and Design of Network Protocols (2003)