CELL: A Compositional Verification Framework[*]

Kun Ji[1], Yang Liu[2], Shang-Wei Lin[1], Jun Sun[3], Jin Song Dong[1], and Truong Khanh Nguyen[1]

[1] National University of Singapore
[2] Nanyang Technological University
[3] Singapore University of Technology and Design

Abstract. This paper presents CELL, a comprehensive and extensible framework for compositional verification of concurrent and real-time systems based on commonly used semantic models. For each semantic model, CELL offers three libraries, i.e., compositional verification paradigms, learning algorithms and model checking methods to support various state-of-the-art compositional verification approaches. With well-defined APIs, the framework could be applied to build customized model checkers. In addition, each library could be used independently for verification and program analysis purposes. We have built three model checkers with CELL. The experimental results show that the performance of these model checkers can offer similar or often better performance compared to the state-of-the-art verification tools.

1 Introduction

Compositional verification technique presents a promising way to alleviate *state explosion problem* associated with model checking via the "divide-and-conquer" strategy. In recent years, a number of approaches have been proposed to conduct compositional verification automatically which are categorized as *learning based assume-guarantee reasoning* (LAGR) [4], *symbolic learning based assume-guarantee reasoning* (SLAGR) [3], *assume-guarantee reasoning by abstraction refinement* (AGAR) [5] and *compositional abstraction refinement* (CAR) [2]. Furthermore, different compositional verification paradigms may work with different learning (or abstraction refinement) algorithms and model checking methods (e.g., symbolic model checking, explicit-state model checking). It is thus desirable to build a framework such that different approaches can be systematically experimented, compared or applied.

In this work, we propose a comprehensive and extensible framework named CELL, which contains various state-of-the-art compositional verification approaches for concurrent and real-time systems based on commonly used semantics models (i.e., *labeled transition system* (LTS) for concurrent systems and *timed transition system* (TTS) [6] for real-time systems). For each semantic model, CELL offers three libraries, i.e., compositional verification paradigms, learning algorithms and model checking methods. Various state-of-the-art compositional verification approaches can be constructed by

[*] This project is supported by project 'IDD11100102' from Singapore University of Technology and Design

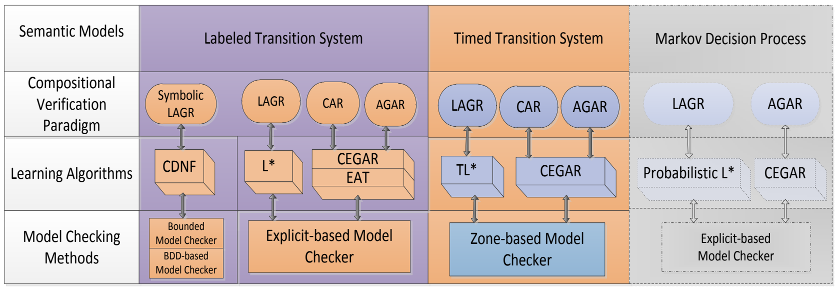

Fig. 1: Design of CELL

combining items from each library. For instance, the compositional verification approach proposed in [4] can be achieved by combining LAGR compositional verification paradigm, the L^* learning algorithm and an explicit-based model checking method respectively from the three libraries designed for models whose semantics are LTSs. Currently, CELL provides seven compositional verification paradigms, seven learning algorithms, four model checking methods and ten ways of combinations to perform automatic compositional verifications. In addition, CELL can be extended in multiple ways, e.g., with new semantic models (e.g., Markov Decision Process), new compositional verification paradigms, learning algorithms or model checking methods. Figure 1 shows the overall architecture of CELL. Notice the light-color part shows how CELL can be (and is being) extended to support probabilistic systems.

To the best of our knowledge, CELL is the only stable and publicly available compositional verification framework. CELL is an open source project under LGPL v3 license in the format of dynamic linked library (DLL) with no GUI. We used PAT [8, 7] framework's GUI to develop the three demonstrating model checkers. It is possible to build new model checkers using CELL to conduct the verification tasks.

2 CELL Architecture

CELL's architecture includes four layers. With the defined APIs from the semantic model layer, domain experts are allowed to easily manufacture model checkers with various compositional verification approaches to alleviate the state explosion problem. Furthermore, the APIs of the lower layers are well defined so that they can be used independently for various purposes.

Semantic Model Layer In this layer, we support commonly used semantic models (i.e., LTS for concurrent systems and TTS for real-time systems). Any modelling language whose semantic model is LTS or TTS can be verified using our framework. In CELL, we assume for systems, which have LTS/TTS semantics, both the system and the property are represented in LTS/TTS[4]. The verification problem is thus reduced to

[4] For real-time system, we assume the property is determinizable.

check the language inclusive of the model and whether the model defines a language which is a subset of that of the property.

Compositional Verification Paradigm Layer This layer contains typical patterns of compositional verification approaches that we have categorized. As shown in the second layer of Figure 1, we provide LAGR, AGAR and CAR for both LTS and TTS semantics models. In addition, we provide SLAGR for LTS model, which may reduce the state space for some models by leveraging the symbolic model checking.

Learning Algorithm Layer To construct the assumptions or model abstractions needed by compositional verification, different learning or abstraction refinement algorithms are supported in this layer. For consistency, we include the abstraction refinement techniques (e.g., CEGAR and EAT [2]) in the set of learning algorithms. The current implementation includes the following: L^* learning algorithm, CDNF Boolean function learning algorithm, CEGAR and EAT techniques for concurrent systems, TL^* learning algorithm and CEGAR for real-time systems. The basic idea of EAT [2] is to use evolutionary algorithm to generate abstractions, which can increase the probability of finding good abstractions.

Model Checking Method Layer In this layer, we provide various model checking methods. We provide *explicit-state model checking* and *symbolic model checking* for LTS, and *zone-based model checking* for TTS. For symbolic model checking, we provide both SAT-based bounded model checking and BDD-based model checking.

Under each semantic model, compositional verification paradigms, learning algorithms and model checking methods can be mix-and-match to construct compositional verification approaches. Notice that not every combination is effective. The arrows in Fig. 1 show the relationship. Currently, CELL supports seven different verification approaches for LTS and three for TTS. All these combinations and their features are summarized in our website [1]. A technical report that explains more details about each component in CELL can be also found there.

3 Implementation and Evaluation

CELL is implemented on Microsoft .NET framework via $C\#$ language. Starting from 2011, the latest version 0.3 of CELL has 54K LOC. CELL is a stand-alone library in the format of DLL and can be used by calling its APIs.

To prove the capability of CELL framework, we developed three compositional model checkers adopting the GUI from PAT framework [8]. The model checkers include *CLTS* that is used to verify concurrent systems modelled by finite state machines, *CERA* to verify real-time systems modelled by *event-recording automata* (ERAs) and *CTA* to verify real-time systems modelled by *timed automata* (TAs). It is non-trivial to measure how easy to use CELL. However, we have built those model checkers within one month, which shows that our design is promising. The CELL DLL binary file together with the source code, complete APIs description document, user manual and three aforementioned model checkers are available in [1].

With CLTS and CEAR, we modelled a bunch of concurrent and real-time systems which include the *AIP manufacturing system*, *Dinner Philosopher problems* (DP) and

Table 1: Running time (in seconds), the number of highest visited locations in all the verification rounds $\mid L \mid$, $\mid P \mid$ means number of processes and ROM means running out of memory.

LTS			Monolithic		AGAR		LAGR		CAR		SLAGR
Case	$\mid P \mid$	Valid?	$\mid L \mid$	Time	$\mid L \mid$	Time	$\mid L \mid$	Time	$\mid L \mid$	Time	Time
AIP	10	Yes	104,650	7.86	2,745	0.44	2,745	**0.29**	2,878	0.98	0.90
DP	30	Yes	ROM	ROM	20,824	11.95	20,824	7.03	1,500	**3.32**	11.19
FMS-3	11	Yes	312,064	12.77	1,920	0.11	1,260	**0.08**	20	0.17	0.12
FMS-4	14	Yes	ROM	ROM	24,744	6.93	26,320	2.61	530	0.22	**0.14**
TTS			Monolithic		AGAR		LAGR		CAR		
FMS-1	6	Yes	212	0.13	36	0.02	36	**0.01**	36	0.02	
FMS-2	10	Yes	97,136	7.49	1,260	0.29	1,260	0.13	1,260	**0.02**	
FMS-3	11	Yes	312,064	23.39	1,920	0.35	1,528	**0.19**	3,936	1.42	
FMS-4	14	Yes	ROM	ROM	24,744	30.93	26,320	**5.13**	24,744	12.81	

various versions of *flexible manufacturing systems* (FMSs) that differ by complexities for both concurrent and real-time versions (FMS-4 is the most complex one). We did not compare with other model checkers such as NuSMV or Uppaal because of the different modelling languages and supported properties. In addition, it is unfair to compare with these monolithic model checkers since CELL adopts compositional technique, and NuSMV and Uppaal may have advanced reduction techniques that are not available in CELL. Table 1 shows the verification results. For the concurrent systems, due to the limited space, we show results collected from subset of the verification approaches, which are CEGAR-based AGAR, L^*-based LAGR, EAT-based CAR, CDNF-based (with BDD) SLAGR. It can be obversed that all the compositional verification approaches outperform the monolithic approach. CDNF-based SLAGR has better performance since it takes advantages of symbolic model checking. EAT-based CAR outperforms CEGAR-based CAR as EAT can find better abstractions [2]. For the real-time experiments, we show results from all the three approaches, which respectively are CEGAR-based AGAR, TL^*-based LAGR and CEGAR-based CAR. Observe that all the compositional verification approaches outperform the monolithic one. More detailed results are available with our technical report [1].

References

1. CELL website, http://www.comp.nus.edu.sg/~pat/cell/.
2. EAT: Evolutionary abstraction technique, https://sites.google.com/site/shangweilin/eat.
3. Y. Chen, E. Clarke, A. Farzan, M. Tsai, Y. Tsay, and B. Wang. Automated assume-guarantee reasoning through implicit learning. In *CAV*, pages 511–526, 2010.
4. J. Cobleigh, D. Giannakopoulou, and C. Păsăreanu. Learning assumptions for compositional verification. In *TACAS*, pages 331–346, 2003.
5. M. Gheorghiu Bobaru, C. Păsăreanu, and D. Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In *CAV*, pages 135–148, 2008.
6. T. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In *Real-Time: Theory in Practice*, pages 226–251, 1992.
7. Y. Liu, J. Sun, and J. S. Dong. Pat 3: An extensible architecture for building multi-domain model checkers. In *ISSRE*, pages 190–199, 2011.

8. J. Sun, Y. Liu, J. Dong, and J. Pang. PAT: Towards flexible verification under fairness. In *CAV*, pages 709–714, 2009.