Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2016-IW-002

2016-IW-002

ACSPChecker: An ASP based CSP Model Checking Tool

Lingyun Situ, Yu Wang, Fengjuan Gao, Linzhang Wang, Lei Bu, Jianhua Zhao, Xuandong Li

Internetware 2016

ACSPChecker: An ASP based CSP Model Checking Tool

Lingyun Situ, Yu Wang, Fengjuan Gao, Linzhang Wang, Lei Bu, Jianhua Zhao, Xuandong Li
State Key Laboratory of Novel Computer Software Technology, Nanjing University, Nanjing 210023, China
Jiangsu Novel Software Technology and Industrialization, Nanjing 210023, China
Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China
{lzwang, bulei, zhaojn, lxd}@nju.edu.cn

ABSTRACT

Existing CSP model checkers are incapable of verifying multiple properties concurrently in one run of a model checker, and when trying to alleviate state space explosion problem, most of reduction work are usually done after rather than before the complete state space was produced. Thus, A new CSP model checking tool named ACSPChecker was developed based on answer set programming, which is a declarative logic programming paradigm for solving combinational search problems with the feature of completely free of sequential dependencies, to verifying multiple properties concurrently in one run of a model checker . Additionally, It integrated an abstraction method, which could be used to alleviate the state space explosion before the complete state space was produced. Furthermore, a preprocessing technique of properties was proposed to improve the verification efficiency by reducing the expense spending on replicated verification of the same sub formulas. The feasibility and efficiency of ACSPChecker are illustrated by the experiments with a classic concurrency problem - dining philosophers problem.

CCS Concepts

•**Software and its engineering** → **Model checking;**

Keywords

Abstraction; Proof; Communicating Sequential Process; Model Checking; Answer Set Programming.

1. INTRODUCTION

Model checking [5][4][2] is one of the most important and powerful formal verification techniques for CSP. Some automated CSP model checking tools have been developed such as FDR [7], PAT[19] , SymFDR [15], ProB [9] [11] [10], ARC [17] [16] and so on. But Existing CSP Model Checkers are incapable of verifying multiple properties concurrently in one run of a model checker, and when trying to alleviate state space explosion problem, most of reduction work are usually done after rather than before the complete state space was produced.

To deal with the shortcoming of inability to verify multiple properties concurrently in one run of existing CSP model checkers, a CSP bounded model checking framework using answer set programming (ASP)[1][3] was established[18] in our previous work , which could be used to achieve the verification of multiple properties concurrently in one run of a model checker, relying on the feature that ASP is completely free of sequential dependencies[6][13]. The framework turns CSP model checking problem into a computation problem of answer sets. A semantic equivalence explanation and proof about the translation from CSP to ASP was given in [20].

ACSPChecker, an ASP based CSP model checking tool was developed based on the framework[18] , an abstraction method was embedded in it, which could be used to alleviate state space explosion before the complete state space was produced. Furthermore, a preprocessing technique of properties was also integrated into it to improve the verification efficiency by reducing the expense spending on replicated verification of the same sub formulas. The feasibility and efficiency of the ASP based CSP model checking tool -ACSPChecker is illustrated by the experiments with dining philosophers problem.

Internetware '16, September 18 2016, Beijing, China

DOI: http://dx.doi.org/10.1145/2993717.2993730

2. PRELIMINARIES

2.1 CSP

Definition 1. A CSP process is defined recursively via the following grammar:

$$P \equiv STOP|SKIP|CHAOS|DIV|X : A \to P(x)|\mu X : A \bullet F(X)|P1 \sqcap P2|P1|P2|P1 \parallel P2|P \setminus A|P[R]$$

$STOP$ represents a deadlocked process, which is not capable of communicating any visible or τ actions. The process $SKIP$ denotes successful termination and is willing to perform $\surd$ at any time. $CHAOS$ is a process that may non-deterministically perform events from A. It may as well refuse to do anything at all. DIV denotes a livelock, a process that is engaged in performing an infinite loop of internal τ actions without ever communicating with the external environment. The prefix process $X : A \to P(x)$ initially offers the environment to perform any event a from A and subsequently behaves like $P(a)$. $\mu X : A \bullet F(X)$ denotes a recursive process.$P1 \sqcap P2$ and $P1|P2$ denote, respectively, external and internal choice of $P1$ and $P2$. Parallel composition of processes $P1$ and $P2$ is written as $P1 \parallel P2$, where

shared events must be synchronized by both processes whose alphabet contains the events. $P \setminus A$ behaves like P except that all the events from A are being hidden. $P[R]$ behaves likes P, except that, whenever P can perform any event a, P[R] can perform any event b, such that aRb.

2.2 ASP

Let A be an atom, a literal takes the form A or $\sim A$, where A is a positive literal and $\sim A$ is a negative literal; A and $\sim A$ are called complementary literals[3]. An extended disjunctive logic program is a set of rules, and each rule r is of the form:

$$L_1 \vee ... \vee L_k :- L_{k+1}, ..., L_m, not\ L_{m+1}, ..., not\ L_n$$

where $n \geq m \geq k \geq 0$, each L_i is a literal, and notice the negation as failure (NAF), head(r)=$L_1, ..., L_k$ is the head of r, pos(r)=$L_1, ..., L_m$ and neg(r)=$L_{m+1}, ..., L_n$ are the positive and negative literals present in body respectively. In particular, a rule without head is called a constraint. The computation of answer sets corresponding to ASP logic programs is performed by ASP solvers, such as DLV[8], Smodel[14] and Cmodel[12].

3. ACSPCHECKER

The structure of ASP based CSP model checking tool - ACSPChecker is shown as Fig.1.

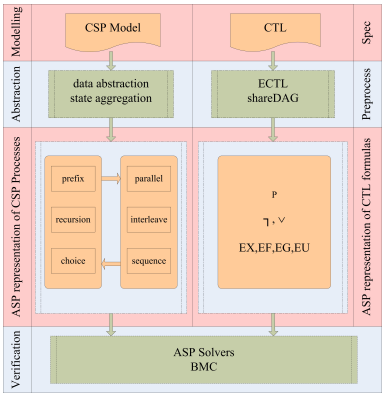

Figure 1: ASP based CSP BMC framework

The verification procedure of ACSPChecker is illustrated as Figure 2.

The input of ACSPChecker is a CSP model and a set of CTL formulas, and the output is a set of answer sets. We decide the properties to be verified is true or not according to the head label of relevant properties is occur in the answer sets or not. If the properties relevant head label is occur, then the property is true, otherwise it is false.

3.1 Abstraction

The fundamental idea of data abstraction is to give a mapping between the actual data values and a small set of abstract data values, which is relying on equivalence class partition. Typically, assume we are interested in whether the parallel process $P1 \parallel P2$ satisfies property $f$or not. αf denotes the events set of occurring in property f. All the events are partitioned into three equivalence classes $C1$,$C2$ and$C3$, where $C1 = (\alpha P1 \cup \alpha P2) \cap \alpha f$, $C2 = (\alpha P1 \cup \alpha P2) - \alpha f - (\alpha P1 \cap \alpha P2)$, $C3 = (\alpha P1 \cap \alpha P2) - \alpha f$.

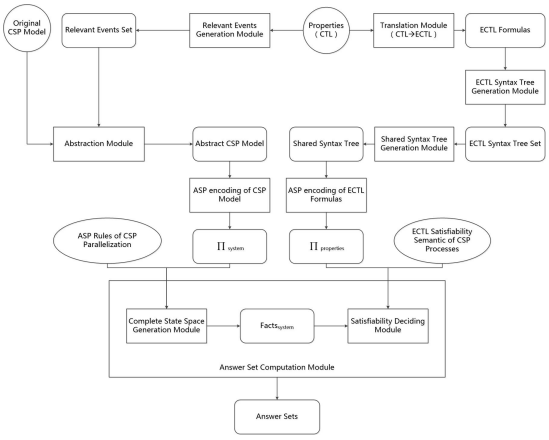

Figure 2: ACSPChecker verificaiton procedure

Now, a process can be seen as a sequence of the events from the three equivalence classes, and the parallel composition of two processes comes down to the synchronization or asynchronization between the three equivalence classes C1,C2, C3 of the two processes.

When the CSP model M is a CSP basic processes P, i.e. a process without parallel operator, then all the events irrelevant to the property φ in process P should be removed.

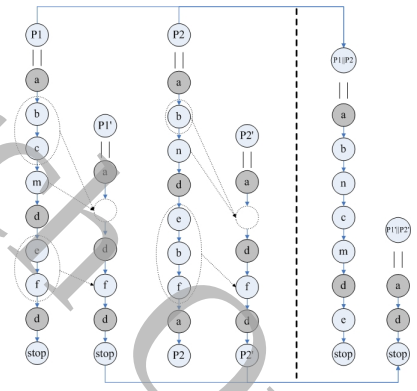

Figure 3: Abstraction example $P1 \parallel P2$

When the CSP model M is a parallel process of the form $P1 \parallel P2$, the events of each component process should be handled differently .The algorithm for dealing with parallel process $P_1 \parallel P_2$ can be presented in Algorithm 1.

Example 1. Let $P1 = a \to b \to c \to m \to d \to e \to f \to d \to STOP$, $P2 = a \to b \to n \to d \to e \to b \to f \to a \to P2$, $M = P1 \parallel P2$ and $Pro = \{a, d\}$, the abstract CSP model of $P1 \parallel P2$ could be obtained as illustrated in Figure 3.

As shown in Figure 3, initially, the process $P1$ has 9 states, and $P2$ is a recursion process with 8 events. After abstraction, the abstract model $P1'$ and $P2'$ contain 5 states, respectively. Further, the states of parallel process $P1' \parallel P2'$ is reduced from 8 to 3.

Furthermore, we proved that :If $M' = abs(M, \varphi)$, then $M' \vDash \varphi \Rightarrow M \vDash \varphi$

Algorithm 1 Abstraction algorithm of parallel process $abs(P_1 \parallel P_2, \varphi Set)$

Require: An original CSP model $P_1 \parallel P_2$, A set of properties φSet
Ensure: An abstract CSP model M'

$Pro \leftarrow \textbf{\textit{REProduce}}(\varphi Set) = \{c | c \in \alpha\varphi \wedge \varphi \in \varphi Set\}$;
$P_1 \leftarrow \textbf{\textit{Hide}}(P_1, \alpha P_1 - Pro - \alpha P_1 \cap \alpha P_2)$;
$P_2 \leftarrow \textbf{\textit{Hide}}(P_2, \alpha P_2 - Pro - \alpha P_1 \cap \alpha P_2)$;
$P_1 \leftarrow \textbf{\textit{Extend}}(P_1)$;
$P_2 \leftarrow \textbf{\textit{Extend}}(P_2)$;
$P_1.MITA \leftarrow \textbf{\textit{MITArea}}(P_1)$;
$P_2.MITA \leftarrow \textbf{\textit{MITArea}}(P_2)$;
$N \leftarrow \textbf{\textit{MinNum}}(P_1.MITA, P_2.MITA)$;
for all i <= N **do**
  if $P_1.MITA[i] \preceq P_2.MITA[i]$ or $P_2.MITA[i] \preceq P_1.MITA[i])$ **then**
    $P_1 \leftarrow \textbf{\textit{Remove}}(P_1, P_1.MITA[i].events)$;
    $P_2 \leftarrow \textbf{\textit{Remove}}(P_2, P_2.MITA[i].events)$;
  else
    $e1 \leftarrow \textbf{\textit{FirstDifferEvent}}(P_1.MITA[i], P_2.MITA[i])$;
    $P_1 \leftarrow \textbf{\textit{Replace}}(P_1.MITA[i], e1)$;
    $e2 \leftarrow \textbf{\textit{FirstDifferEvent}}(P_1.MITA[i], P_2.MITA[i])$;
    $P_2 \leftarrow \textbf{\textit{Replace}}(P_2.MITA[i], e2)$;
  end if
end for
$M' = P_1 \parallel P_2$;
return M'

3.2 Preprocessing

CTL is extended with events for the sake of description of properties based on states and events.

Definition 2. The CTL formulas extended with events are as follows:

$$\varphi ::= e|s|\neg\psi|(\psi_1 \wedge \psi_2)|(\psi_1 \vee \psi_2)|(\psi_1 \rightarrow \psi_2)|$$
$$AX\psi|EX\psi|AF\psi|EF\psi|AG\psi|EG\psi|A[\psi_1 U \psi_2]|E[\psi_1 U \psi_2]$$

where e ranges over $\sum^{\surd}$. s represent the state, A represent all the paths, E represents exist one path at least, X represent the next state, F represent some state in the future, G represent all the state in the following and U represent until.

Given a set U of ECTL formulas, the sharedDAG could be obtained by applying the following rules:

Rule 1(remove duplicate leaf nodes) If a syntax tree has the same leaf nodes $n_1, n_2, ..., n_k$, then remove all the same leaf nodes except n_k, and redirect all arcs into the $n_i (1 \leq i \leq k)$ to n_k.

Rule 2 (remove duplicate internal nodes) Let u and v be internal nodes, the label of u and v are the same, remove the sub tree with the root u and redirect all arcs into u to v if conditions below are satisfied, otherwise, remove v.

(1) Both u and v have two sub-trees, and the sub-trees of u and v are the same.(2) Node u has one sub tree, v has two sub trees, and the sub tree of u is the same as one of the sub trees of v.(3) Both nodes u and v have only one sub tree, and the sub trees of u and v are the same.

Example 2. Consider following properties of dining philosopher problems:

(1)Pro 1: Any fork can not be picked up by two philosophers at the same time.

(2)Pro 2: Any philosopher will eat if he has picked up his forks.

The two properties above are described with extended CTL formulas as follows:

(1) $AG\neg(i.pickFork.i \wedge i \bigoplus 1.pickFork.i)$
(2) $AG(i.pickFork.i \rightarrow AF(i.eat)$

The formulas are input into the preprocessing procedure, and then a sharedDAG is obtained as Fig.4(To be simple, set i=2).

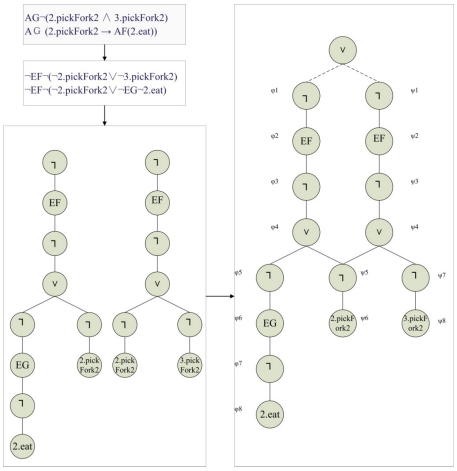

Figure 4: shareDAG

Intuitively, in this example, if we verify the two properties using classical labeling algorithm, $\neg 2.pickfrok2$ will be verified twice, but only once after preprocessing. Our preprocessing procedure will reduce the repeated verification overhead and improve verification efficiency especially when the number of properties is large.

4. EXPERIMENTAL RESULTS

Experimental results with philosophers dining problem are presented as follows, where the configuration of our computer is Intel Core(TM) i3-2100 CPU, 3.10GHz, 4GB(RAM), 64bit window 7.

Table 1: Result concurrently verification

ACSPChecker	CSP Model			
(DLV, N=3)	P 1	P 2	Sum	All
Bounded k=5	0.09/T	0.15/F	0.24	0.11
Bounded k=10	1.32/T	1.38/T	2.70	1.34
Bounded k=15	1.41/T	3.44/T	4.85	3.45
Bounded k=20	6.67/T	6.68/T	13.35	7.39

Table 1 is the situation for the model of 3 philosophers. It is shown that preprocessing technique has great effect on improving the efficiency of verifying properties. The time of multi-properties verification concurrently is smaller than the sum of verification respectively. And the verification of liveness properties is affected by the bounded step k, because the k determine the size of states space of system.

Table 2: Results with increasing bounded k

ACSPChecker	Original CSP model			Abstract CSP model		
DLV, N=3	P1	P2	All	P1	P2	All
Bounded k= 5	0.09/T	0.15/F	0.11	0.02/T	0.03/T	0.03
Bounded k=10	1.32/T	1.38/T	1.34	0.25/T	0.27/T	0.27
Bounded k=15	1.41/T	3.44/T	3.45	0.98/T	0.95/T	1.02
Bounded k=20	6.67/T	6.68/T	7.39	1.35/T	1.63/T	1.97

Table 2 is the situation for a model of 3 philosophers and inner ASP solver of ACSPChecker is DLV[8]. It is shown that the abstraction have great effect on improving the efficiency of verifying the properties. For example, when bounded k steps is 10, the time to verfying P1 in original CSP model is 1.32 second, while 0.25 second in abstract CSP model.

Table 3: Results with increasing philosophers

ACSPChecker (DLV, Bounded k=10)		N=3	N= 5	N=7	N=8	N=10
		Verification Time (s)				
Original model	P1	1.32s	11.42s	37.94s	85.69s	-
	P2	1.38s	11.49s	38.32s	85.25s	-
	All	1.34s	11.55s	38.50s	86.46s	-
Abstract model	P1	0.25s	0.38s	2.29s	3.33s	9.89s
	P2	0.27s	0.41s	2.31s	3.41s	11.01s
	All	0.27s	0.42s	2.33s	3.48s	11.06s

Table 3 collects the experimental results with the dining philosophers problems with increasing numbers of philosophers, where bonded step k is set to 10. We are exciting to find the great improvement in the scale of system model could be verified, as illustrated in Table 3, the maximal numbers of philosophers is only 8 without abstraction in 4GB experimental environment. But the numbers of philosophers could be verified become more than 10, and the time cost is significantly reduced when abstraction technique is applied.

5. CONCLUSION

ACSPChecker is a new CSP model checking tool, developed based on answer set programming, which could verify multiple properties concurrently in one run of a model checker. Additionally, an abstraction method and a preprocessing technique are integrated into ACSPChecker to improve the verification efficiency. The feasibility and efficiency of ACSPChecker are illustrated by the experiments with a classic concurrency problem - dining philosophers problem.

6. ACKNOWLEDGMENT

The paper was partially supported by the National Natural Science Foundation of China (No. 91418204, 61321491, 61472179, 61561146394, 61572249).

7. REFERENCES

[1] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press, 2003.

[2] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and software verification: model-checking techniques and tools*. Springer Science & Business Media, 2013.

[3] P. Bonatti, F. Calimeri, N. Leone, and F. Ricca. Answer set programming. In *A 25-year perspective on logic programming*, pages 159–182. Springer-Verlag, 2010.

[4] E. M. Clarke. The birth of model checking. In *25 Years of Model Checking*, pages 1–26. Springer, 2008.

[5] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT press, 1999.

[6] M. Gebser, M. Maratea, and F. Ricca. WhatâĂŹs hot in the answer set programming competition. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[7] T. Gibson-Robinson, P. Armstrong, A. Boulgakov, and A. W. Roscoe. Fdr3âĂŤa modern refinement checker for csp. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 187–201. Springer, 2014.

[8] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7(3):499–562, 2006.

[9] M. Leuschel and M. Butler. Prob: A model checker for b. In *International Symposium of Formal Methods Europe*, pages 855–874. Springer, 2003.

[10] M. Leuschel, J. Falampin, F. Fritz, and D. Plagge. Automated property verification for large scale b models with prob. *Formal Aspects of Computing*, 23(6):683–709, 2011.

[11] M. Leuschel and M. Fontaine. Probing the depths of csp-m: A new fdr-compliant validation tool. In *International Conference on Formal Engineering Methods*, pages 278–297. Springer, 2008.

[12] Y. Lierler and M. Maratea. Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. In *International Conference on Logic Programming and NonMonotonic Reasoning*, pages 346–350. Springer, 2004.

[13] V. Lifschitz. What is answer set programming?. In *AAAI*, volume 8, pages 1594–1597, 2008.

[14] I. Niemela, P. Simons, and T. Syrjanen. Smodels: a system for answer set programming. *arXiv preprint cs/0003033*, 2000.

[15] H. Palikareva, J. Ouaknine, and A. Roscoe. Sat-solving in csp trace refinement. *Science of Computer Programming*, 77(10):1178–1197, 2012.

[16] A. Parashkevov and J. Yantchev. ArcâĂŤa verification tool for concurrent systems. In *Proceedings of the Third Australasian Parallel and Real-Time Conference*. Citeseer, 1996.

[17] A. N. Parashkevov and J. Yantchev. Arc-a tool for efficient refinement and equivalence checking for csp. In *Algorithms & Architectures for Parallel Processing, 1996. ICAPP 96. 1996 IEEE Second International Conference on*, pages 68–75. IEEE, 1996.

[18] L. Situ and L. Zhao. Csp bounded model checking of preprocessed ctl extended with events using answer set programming. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*, pages 16–23. IEEE, 2015.

[19] J. Sun, Y. Liu, and J. S. Dong. Model checking csp revisited: Introducing a process analysis toolkit. In *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, pages 307–322. Springer, 2008.

[20] L. Zhao, Z. zhongyi, J. Qian, and G. Yunchuang. Model checking csp based on asp and critical-trace model of csp. *Journal of Software*, 26(10), 2015.