STATE SPACE GENERATION FRAMEWORK BASED ON BINARY DECISION DIAGRAM FOR DISTRIBUTED EXPLICIT MODEL CHECKING

Nacer Tabib[1], Jean Michel Ilie[2], and Djamel Eddine Saidouni[1]

[1]Misc Laboratory, Constantine 2 University , Algeria
{tabib,saidounid}@misc-umc.org
[2]Lip6 Laboratory, UPMC, France
{jeanmichel.ilie}@upmc.fr

ABSTRACT

This paper proposes a new framework based on Binary Decision Diagrams (BDD) for the graph distribution problem in the context of explicit model checking. The BDD are yet used to represent the state space for a symbolic verification model checking. Thus, we took advantage of high compression ratio of BDD to encode not only the state space, but also the place where each state will be put. So, a fitness function that allows a good balance load of states over the nodes of an homogeneous network is used. Furthermore, a detailed explanation of how to calculate the inter-site edges between different nodes based on the adapted data structure is presented.

KEYWORDS

Graph distribution, Binary Decision Diagram, State space generation, Formal verification, Model Checking.

1. INTRODUCTION

An efficient way to improve applications' performances is to use networks. In fact, many already existent applications have been transformed from their simple versions to distributed ones whether they are not initially implemented in a distributed version in the aim of increasing the storage capacity and driving the computing more quicker.

Let's take the formal verification [1] of systems as an example of such applications. An attractive solution to face the major problem of these applications which focus on the combinatorial states space explosion and computing time is the distribution of the graph (states space)[2].

Despite the large use of graphs [3] in computing science domains, they still meet so serious and heavy difficulties especially when certain thresholds and limits are exceeded. That is why it is useful to split the main graph into a set of distributed sub-graphs.

Jan Zizka et al. (Eds) : CCSIT, SIPP, AISC, CMCA, SEAS, CSITEC, DaKM, PDCTA, NeCoM - 2016
pp. 297–305, 2016. © CS & IT-CSCP 2016 DOI : 10.5121/csit.2016.60125

The workload balancing, minimization of the distributed inter-site communication of an unreliable network represent two important factors that are necessary to take them into account in order to generate an ideal distribution of the graph. Both of them influence the application's performances and because of this reason, taking them into account makes the graph distribution a really hard task.

Using several computers of small capacities all together would give an unlimited capacity in term of speed and memory. However, the main inconvenient of distributed algorithms is on distributing the states space of the graph without taking into account the workload balancing that will affect directly the distributed verification application's performances. Besides considering the workload balancing and the distributed inter-nodes edges separately are not enough to improve the distributed verification performances [4].

Several solutions have been proposed to tackle this problem such as equivalence relations, partial order based relations [5] [6]. Although, these solutions reduce the graph size significantly, the memory capacity remains a problem when dealing with very complex systems.

Nowadays, workstations clusters give more and more hardware resources availability, hence we can represent large graph over the cluster where each workstation can hold a sub-graph [7] [8]. But most works use either the symbolic methods based on BDD [9], [10] or explicit methods [7].
A new approach of distributing system states space is proposed in this paper. This new framework developed is based on a compressed format of data structure called Distribution with Binary Dicision Diagram (DBDD) to keep a local vision of the whole system. The framework exposes throught its API a set of services that can be used by distributed algorithms in order to distribute graphs and perform a distributed verification.

The paper is organized as follows. In Section 2 we introducing fundamental concepts of distributed graphs, BDD and Petri nets, then we move to present our Approach trough different subsections in the same part. After deeply presenting the algorithm in Section 3, we make some experiments on the algorithm to show its performances comparing to other algorithms of graph distributing in Section 4 and Section 5 . Finally, we achieve the paper by Section 6 to conclude.
In the following sections we use interchangeably the terms graph and states space, where we mean by states space a graph generated from a Petri net specification representing its behavioural semantics.

2. BASIC CONCEPTS

The graph to be distributed is generated from a petrinet specification. We briefly recall the definitions of some basic concepts necessary in the following sections.

2.1. Distributed Graph

Let $W = \{W_k\}_{k=1..N}$ be N sites, a distributed graph (noted DiG), is a graph with a function of distribution (partial) f^k.

$$DiG = (G, f^k)\ k=1..N$$

such that :

- $G = (V,E)$: an oriented graph.

- $f^k : G \rightarrow G_k$ is an application of G in G_k, such that $G_k = (V_k, E_k)$

Notation 21 $\{G_k\}_{1 \leq k \leq N}$ *is a set of subsets called fragments* G_k*, such that* $\cup V_k = V$ *and* $\cup E_k \subseteq E$

Definition 1. *a fragment* G_k *is defined by* $G_k = (V_k, E_k)$ *such that :*

- $V_k \subseteq V$: V_k *is a fragment of nodes of V in the site* W_k*.*

- $E_k = E_k^L \cup E_k^R$ *such that* $E_k^L \cap E_k^R = \emptyset$: *the set of intra-site and inter-sites edges with :*

- $E_k^L \subseteq V_k^2$ *is the set of edges between nodes belonged in the same site* W_k *(Local edges).*
- $E_k^R \subseteq V_k \times (V \setminus V_k) = \{(v_k, v_k^{'}) \text{ such that } v_k \in V_k \text{ and } v_k^{'} \notin V_k\}$: *is the set of edges whose the origins are in the local sites and the goals are in the remote sites (Remote edges).*
- α_k *and* β_k *are two applications of* E_k *in V such that for all edges* $e = (v, v^{'}) \in E$:
- $\alpha_k(e) = v \in V_k$: *indicate the origin of the edge e.*
- $\beta_k(e) = v^{'} \in V_k$ *if* $e \in E_k^L$ *and* $\beta_k(e) = v^{'} \in / V_k^{'}$ *else.*

Notation 2.2 *given a set S,* $|S|$ *denotes its cardinality (the number of elements).*

Figure 1 represents a distributed graph over sites (nodes) of a cluster of workstations (workers). We assume that the initial graph is so large that it can't be hold in one machine so distributing it over a different sites while generating it make it possible to take advantage of distributed memory hence we can represent more and more large graphs that correspond to very complex systems.

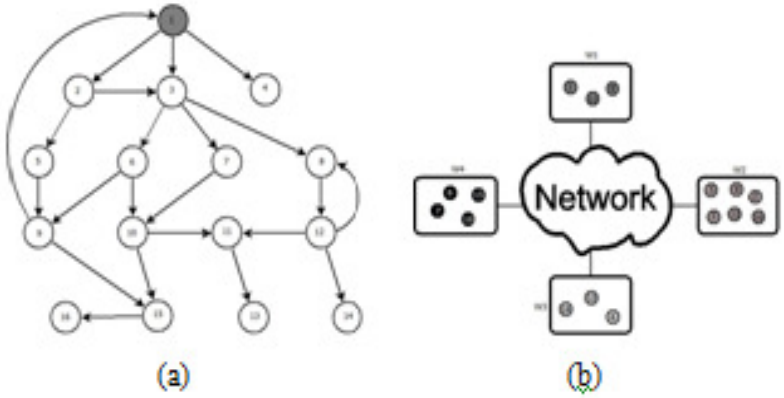


Fig.1. graph before distribution (a) and after (b)

2.2. Petri Net Related Definitions

- A Petri net [11] is a tuple (S,T,W) where S is the set of places, T is the set of transitions such that $S \cap T = \emptyset$, and $W : ((S \times T) \cup (T \times S)) \rightarrow N = \{0,1,2,...\}$ is the weight function. Graphically, transitions of T are represented by rectangles, places of S by circles and weight function by arrows associated with their weights. We suppose that all nets are finite, i.e.$|S \cup T| \in$ N.

- For $x \in S \cup T$, the pre-set ${}^{\bullet}x$ is defined by ${}^{\bullet}x = \{y \in S \cup T | W(y,x) \neq 0\}$ and the post-set $x^{\bullet}$ is defined by $x^{\bullet} = \{y \in S \cup T | W(x,y) \neq 0\}$.

- The *marking* of a Petri net (S,T,W) is defined as a function $M : S \rightarrow \mathrm{N}$. A marking is generally represented graphically by putting tokens in places.

- Safety-Petri net is a Petri net (S,T,W) such that for any s of $S : M(s) \leq 1$

- The transition rule stipulates that a transition t is enabled by M iff $M(s) \geq W(s,t)$ for all$s \in S$. The firing of a transition t will produce a new marking M' defined by$M'(s) = M'(s) - W(s,t) + W(t,s)$ for all$s \in S$. The occurrence of t is denoted by $M[t > M'$.

- Two transitions t_1 and t_2 (not necessarily distinct) are concurrently enabled by a marking M iff $M(s) \geq (s,t_1) + W(s,t_2)$ for all $s \in S$.

- A marked Petri net (S,T,W,M_0) is a Petri net (S,T,W) with an initial marking M_0.

- An alphabet A is a finite set; we suppose that $\tau \in A$(τ will indicate invisible action, or *silent action*).

- The labeling of a Petri net $N = (S,T,W)$ is a function $\lambda : T \rightarrow A \cup \{\tau\}$. If $\lambda(t) \in A$ then t is said to be *observable* or *external*; at the opposite, t is *silent* or *internal*.

- $\Sigma = (S,T,W,M_0,\lambda)$ is a labeled system iff (S,T,W,M_0) is a marked Petri net and λ is a labeling function of (S,T,W)..

2.3 BDD

A Binary Decision Diagram or BDD [10] is data structure used for representation of Boolean functions in the form of rooted directed acyclic graph. A BDD is a rooted directed acyclic graph $G = (V,E)$ with node set V containing two kinds of nodes, *non-terminal* and *terminal* nodes (Figure 2). A non-terminal node v has as tag a variable $index(v) \in \{x_1,x_2,...,x_n\}$ and two children $low(v)$, $high(v) \in V$. The final nodes are called *0-final* and *1-final*. A BDD can be used to compute a Boolean function $f(x_1,x_2,...,x_n)$ in the following way. Each input $a = (a_1,a_2,...,a_n) \in \{0,1\}^n$ defines a computation path through the BDD that starts at the root. If the path reaches a non-terminal node v that is labelled by x_i, it follows the path $low(v)$ if $a_i = 0$, and it follows the path $high(v)$ if $a_i = 1$. The label of the terminal node determines the return value of the BDD on input a. the BDD is called "ordered" if the different variables appear in the same order on all the ways from the root (Figure 2).

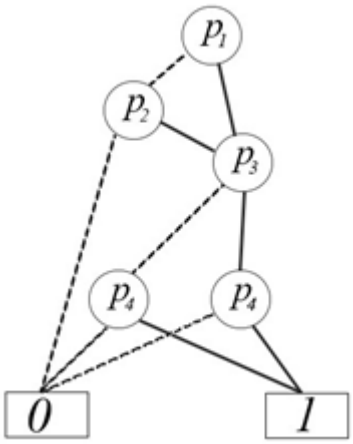
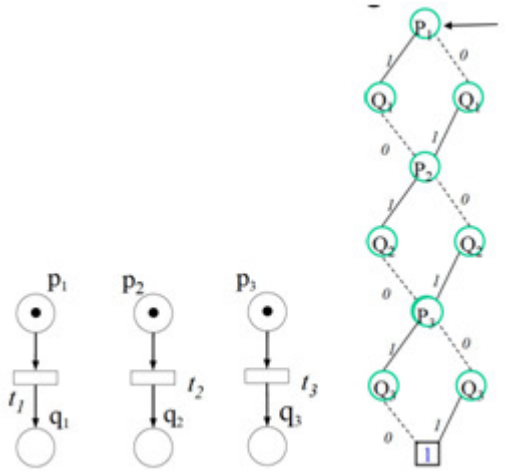


Fig.2. Binary decision diagram

Generating a BDD from a Petri Net BDD: can represent a state space generated from a safe petri Net in an efficient high compressed format. The Figure 3(b) represents a BDD generated from a safe Petri Net 3(a). It uses a set of variables proportional to the number of places in petri net in this example it uses 6 variables to code the different configurations of petri net *p1,p2,p3,q1,q2* and *q3*.



(a) (b)

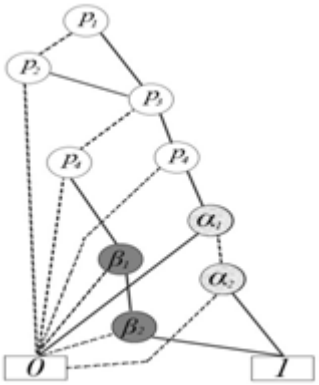
Fig.3. Petri net specification (a) and corresponding BDD (b)

3. PROPOSED APPROACH

Here we are going to present a new framework for graph distribution based on adapted data structure called (DBDD) Distribution with Binary Decision Diagram, the framework provide functions that can be used by parallel and distributed algorithms to generate an explicit state space or to get the location of specific states successors in the distributed graph. Hence the DBDD represent a global state of the system which decrease the communication between several nodes of the network workers and ensure a better fault tolerance.

3.1. Sites Encoding

The DBDD in addition to representing the reachability graph of petri net it encodes the place of each state by injection of a additional game of variables, each variable represent the site where the state is meant to be. Figure 4 represents an example of the encoding of two sites by adding variables which represents these two site ($\alpha 1$, $\alpha 2$) to encode the first site in binary (01). and ($\beta 1$, $\beta 2$) for the second site (10).



ig.4. DBDD represents a graph distributed over two nodes

3.2. DBDD generation

Algorithm 1 below represents the generation of the DBDD, variables are chosen according a binary variable *bddSite*. The fitness function *F* ensures a good load balance.

Algorithm 1: $GenerateDBDD(PNet)$

Constants: $bdd1$: bdd represeting the site0 (00);
$bdd1$: bdd representing site1 (01);
$bdd2$: bdd representing site2 (10)
$bdd3$: bdd representing site3 (11);
s_0 : bdd representing the first marking $M_0$of PNet
T : bdd representing the transition relation

Variables : $Todo = \{s_0\}$: set of states to be processed ;
$Visited = \{\}$: set of processed states;

```
1 while Todo ≠ Visited do
2 |   bddSite ← max(F)({bdd0, bdd1, bdd2, bdd3});
3 |   Visited ← Todo;
4 |   Todo ← Todo ∪ T(Visited) ∪ bddSite
5 end
```

3.3. Fitness function

The site to be chosen for a given set of states is calculated based on the following fitness function:

$$F = \prod_{i=1}^{i=n} |V_i|$$

In an homogeneous network all Sites have the same memory capacity, and a good balance load is when each site hold exactly $\frac{|V|}{n}$ such that $\sum_{i=1}^{i=n} |V_i| = |V|$

4. IMPLEMENTATION AND EVALUATION

The proposed approaches are implemented with JavaBDD [12] (An open source library for manipulating BDD, it is also a wrapper for other libraries such Buddy [13] and Cudd [14]) tested on a network of PC with a 3.0 GHZ processor and 512 MB of memory. We developed a tool that generates distributed graphs associated to petrinets specifications (Figure 5) which is part of FOCOVE framework.

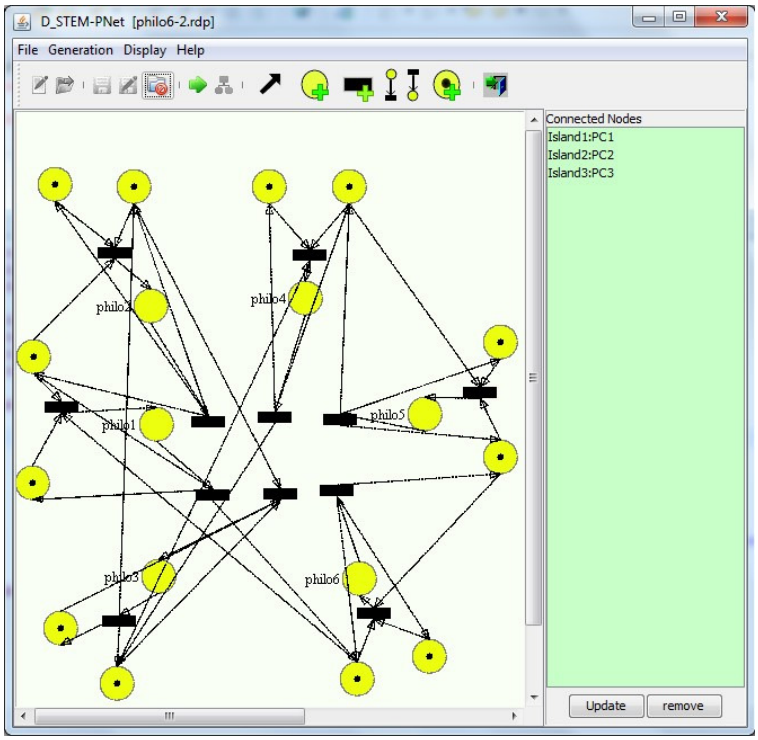


Fig.5. Tool for editing petrinets and generation of state space

5. RESULT AND EXPERIMENTATION

To see the contribution and the advantage of the proposed approach, we compare it to hash function (MD5)[8] based algorithm. Taking examples studied in literature enables us to get more closely to the problem of combinatorial explosion. In the context, we have selected three well known classic case studies in system models. These models include dining philosophers system [15], Peterson solution for mutual exclusion [16] and shared memory system [17].

Table 1. Comparative results of the bdd approach,MD5 based algorithm.

5 sites	**|V |**	**|E|**	σ_v **MD5**	$\sigma_v(\%)$ **MD5**	σ_v **BDD**	$\sigma_v(\%)$ **BDD**
philosophiers	729	3402	21.46	2.9	14.36	1.97
Shared memory	8019	52974	249.61	3.11	96.01	1.19
Peterson	20754	62262	588.67	2.83	607	2.9

The table(1) shows the statistic results according to philosophers, shared memory and Peterson models knowing that the states space has been distributed over 5 sites. The standard deviation of the number of states on each site noted by $\sigma_v(\%)$ is calculated as follows $\sigma_v(\%) = \frac{\sigma_v}{|V|}$. The smaller is the standard deviation σ_v, the better is the distribution over sites, because a tiny σ_v means that the states space is well distributed on the different sites and we see that on table(1). Using the new proposed approach makes it possible to have a fewer σv than the one obtained by using the (MD5) based algorithm except for Peterson and this is due to the replication of some states over the sites.

6. CONCLUSION

In this paper, we have presented a new framework based on binary decision diagrams algorithm to solve the graph distribution problem in context of formal verification. We have used an adapted data structure which ensures a high compression property, the balance load and fault tolerance. We have also compared our work with md5 based algorithm. Results are promising.

To put in practice the result of this work, an optimization algorithm such as evolutionary algorithm or local search may be applied to improve the inter-site communication and tackle also with the variable order problem in BDD. Beside this, different verification algorithms may be applied on the distributed graph generated to verify properties of complex systems.

REFERENCES

[1] Edmund M Clarke, Orna Grumberg, & Doron Peled. Model checking. MIT press,(1999).

[2] Antti Valmari(1998). The state explosion problem , Lectures on Petri nets I: Basic models, pp 429–528. Springer.

[3] Douglas Brent West et al (2001). Introduction to graph theory, volume 2. Prentice hall Upper Saddle River.

[4] Hans Hansson & Bengt Jonsson(1990). A calculus for communicating systems withtime and probabilities, In Real-Time Systems Symposium, 1990. Proceedings., 11th, pp 278–287.

[5] François Vernadat, Pierre Azéma, & François Michel(1996). Covering step graph , Application and theory of Petri nets, pp 516–535. Springer.

[6] Patrice Godefroid, J van Leeuwen, J Hartmanis, G Goos, & PierreWolper. Partialorder(1996) methods for the verification of concurrent systems: an approach to the stateexplosion problem.

[7] Hubert Garavel, Radu Mateescu, & Irina Smarandache(2001). Parallel state space construction for model-checking. , Model Checking Software, pp 217–234. Springer.

[8] Hubert Garavel, Radu Mateescu,Wendelin Serwe(2013), et al. Génération et manipulation d'espaces d'états distribués avec cadp: expériences sur grid'5000, Conférence en Parallélisme, Architecture et Système ComPAS'2013.

[9] Stefan Blom & Simona Orzan(2003). Distributed branching bisimulation reduction of state spaces. Electronic Notes in Theoretical Computer Science,vol.1 n- 89 pp 99–113.

[10] Randal E Bryant.(1992) Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys (CSUR),vol.3 n° 24 pp 293–318.

[11] Eike Best & Harro Wimmel (2013). Structure theory of petri nets, Transactions on Petri Nets and Other Models of Concurrency VII, pp 162–224. Springer.

[12] http://javabdd.sourceforge.net/

[13] http://sourceforge.net/projects/buddy/

[14] http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html.

[15] "NetLogo Models Library: Sample Models/Computer Science Standards" http://ccl.northwestern.edu/netlogo/models/DiningPhilosophers

[16] "Model Checking Contest, "Peterson model" http://sumo.lip6.fr/ Peterson_model.html

[17] "Model Checking Contest, "Shared momory model" http://sumo.lip6.fr/ SharedMemory_model.html