A Model-checking Algorithm for Formal-verification of Peer-to-peer Fault-tolerant Networks

Sungeetha Dakshinamurthy
Research Scholar/Sathyabama university,Chennai, India
Email: (e-mail:sungeetha5@yahoo.com).

Vasumathi K. Narayanan
St. Joseph's college of Engineering Chennai, India
Email: (e-mail: vasumathin@yahoo.com).

***Abstract*— In this paper, the goal is to perform the verification of fault-tolerant properties of a peer-to-peer (P2P) network consisting of *n* nodes running *n* corresponding parallel processes. The specification of the processes is in the form of *communicating finite state machines* (CFSMs). The work to be reported in this paper follows the prequel work wherein, instead of the traditional approach to construct a single synchronous product machine by composing the given CFSMs, we simulate each of the CFSMs in the non-local environment of other CFSMs and generate a set of what are called Communicating Minimal Prefix Machines(CMPMs). In this paper, we take the CMPMs model and perform the reachability analysis of certain global state vectors without losing the locality of the CFSMs of the given specification. This method cuts down the state space explosion and also opens out the possibility of distributed exploration of the local CFSM states. Fault-tolerance consists of both *safety* and *liveness* properties and our approach provides a sound platform for performing state exploration/model-checking to verify these properties of the given set of application tasks that run in the P2P network.**

***Index Terms*—CFSMs-to-CMPMs model, liveness, model-checking, P2P networks, safety, state-space explosion.**

I. Introduction

A P2P network consisting of *n* distributed nodes running *n* corresponding parallel processes interact with each other through the network to accomplish a common goal. The fault-tolerance properties of a network of *n* distributed nodes can be grouped into two categories viz., *safety* and *liveness* and so proving fault-tolerance amounts to verification of these two sets of properties.
Safety means that bad things (like communication deadlocks) will not happen. Liveness means that good things will happen.

Eventuality properties such as certain global state vectors will be eventually reached come under liveness properties. Traditionally, Petrinet based models [1],[2], [3] are used to verify safety and liveness but in these models, there is no static structure to do the verification.

We assume that the peers of the network communicate with each other by synchronization (rendezvous) according to Hoare's CSP model [4] and Milner's calculus [5]. In our prequel work [6], we propose a computational model called Communicating Minimal Prefix Machines (CMPMs) from a given specification of Communicating Finite State Machines (CFSMs). The CMPMs model is an alternative to the traditional product automaton that incurs state-space explosion of the component CFSMs. CMPMs retain the localities of the partner nodes and at the same time store the synchronous global state vectors without incurring the exponential state complexity of the product machine. Thus it is an ideal model to perform formal verification by way of model-checking.

Model-checking is usually done with the aid of a *temporal logic* such as LTL [7] and CTL [8] which are beyond the scope of this paper. We illustrate the rudiments of model-checking by demonstrating how to perform distributed and parallel searches of the simulated CMPM trees.

We begin by presenting briefly the preliminaries of the computational model of CMPMs [9], [10].

II. The Computational Model of CMPMs

This model is developed from a given specification consisting of a set of *communicating finite state machines (CFSMs)* with inter-process communication similar to the one discussed in [11]. We process this specification consisting of a set of *n* CFSM *graphs* into a corresponding set of *n unfolded trees* whose leaves correspond to what are defined as *cutoff* states. Different CFSMs communicate by synchronous message passing upon synchronous actions. An *event* is an instance of an *action*. An action can be completely asynchronous/local to a CFSM or a synchronous one participated by a set of two or more CFSMs from the given set. Each unfolded CFSM is called a *CMPM (Communicating Minimal Prefix Machine)* for a reason to be explained in the sequel.

Manuscript received January 10, 2013; revised March 15, 2013.

doi: 10.12720/lnit.1.3.128-131

Each CMPM state represents not only its corresponding local CFSM state, but also a vector of non-local CFSM states that are its causal predecessors due to synchronization in the most recent past. This vector forms the *synchronous environment* of the concerned MPM state, unfolded from its corresponding CFSM.

A. The CFSMs Specification

The CFSM specification is based on Hoare's CSP model [4]. We assume a set of n communicating and non-terminating FSMs. Each CFSM is defined as a 6-tuple as shown in Fig. 1:

A CFSM $F_i = (s_{0fi}, S_{fi}, A_{fi,} R_{tfi,} Rsync_{fi,,} Rsync_{0fi,})$ $\forall i \in \{1..n\}$ *where,*

- S_{fi} is the *finite* set of states of CFSM F_i, $s_{0fi} \in S_{fi}$ being the *initial state*.
- A_{fi} is the *finite set* of *asynchronous* and *synchronous actions of* F_i.
- If $a_{fi} \in A_{fi}$ is a *synchronous action*, the list of indices $[j_1,j_2,...j_k]$, $k \leq n$ of the partner CFSMs are also specified in the square brackets along with a_{fi}
- R_{tfi} is a *ternary transition relation* such that*:* $R_{tfi} \subseteq S_{fi} \times A_{fi} \times S_{fi}$. In a so-called i-transition $(s_{fi}, a_{fi}, s'_{fi}) \in R_{tfi}$, s_{fi} is called the *input state* and s'_{fi} the *output* state. An i-transition $(s_{fi}, a_{fi}, s'_{fi}) \in R_{tfi}$ is called *synchronous* if $a_{fi}[j_1, j_2,...,j_k]$, is a *synchronous action* such that : $\exists$ a set of *j-transitions* $(s_{fj}, a_{fj}, s'_{fj}) \in R_{tfj}$, $\forall j \in \{ j_1,j_2,...j_k\}$ $j \neq i$ where $a_{fi} = a_{fj}$ and $(s'_{fi}, s'_{fj}) \in Rsync_{fi}$
- $Rsync_{fi} \subseteq S_{fi} \times S_{fj}$, $i \neq j$, $j \in \{1..n\}$, is a binary relation which relates the *output states* of *synchronous transitions*.
- $Rsync_{0fi} \subseteq Rsync_{fi}$ relates the set of pairs of initial states: $Rsync_{0fi} = \{(s_{0fi}, s_{0fj}), \forall j \in \{1..n\}, i \neq j\}$. All the initial states are assumed to be in pairwise synchrony with each other to begin with.

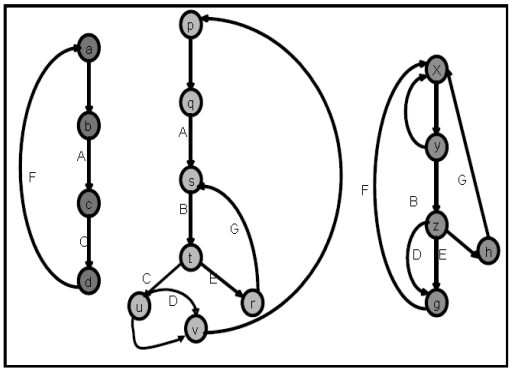

Figure 1. Given set of CFSM graphs representing a P2P network

B. The Simulation of Non-terminating CFSMs into Finitely Terminating CMPM s

The given set of CFSMs represented as cyclic, rooted, directed graphs is simulated in their respective global environments into a corresponding set of CMPMs, each represented by a directed, rooted tree structure as shown in Fig. 2.

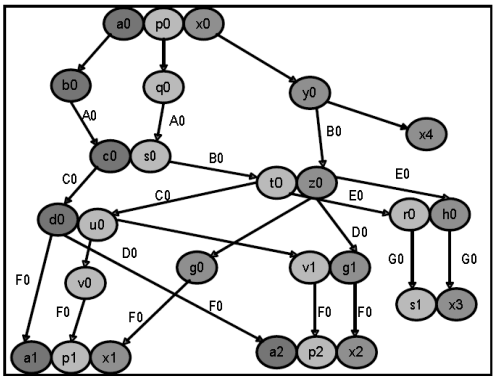

Figure 2. Refined set of Terminating CMPM Trees from the Simulated Non-terminating CFSM graphs

A CMPM $M_i = (s_{0i}, S_i, E_{i,} R_{ti,} Rsync_{i,} Rsync_{0i})$, $\forall i \in \{1..n\}$ *where,*

the countably *infinite* sets of states S_i and events E_i are generated as *instances* of corresponding *finite* sets S_{fi} and A_{fi} respectively of CFSM F_i, $i \in \{1..n\}$.

$S_i \subseteq S_{fi} \times Nat$, $E_i \subseteq A_{fi} \times Nat$ *such that:*

f_{si}*:* $S_i \rightarrow S_{fi}$, $\forall i \in \{1..n\}$ are a set of *n many-to-one* functions, *mapping the infinite domain into finite range.*

where *Nat* is the set of *natural numbers* with $s_{0i} = (s_{f0i}, 0)$, $\forall i \in \{1..n\}$. The entities $Rsync_{i,,}$ $Rsync_{0i}$ can be defined similarly as corresponding instances of CFSM entities $Rsync_{fi}$ and $Rsync_{0fi.}$

C. Well -founded, Partially-Ordered Causality Generation

We unwind the CFSM graphs in their mutual global environment into CMPM *trees* by simulating each of the former in their respective non-local environments.

The global, temporal causality order is composed using the *binary relations* $Rsync_i$ and R_i, where $i \in \{1..n\}$ as follows*:*

$\leq ::= (R_i \cup Rsync_i)^*$

The binary *successor relation* R_i is R_{ti} with its events omitted. The binary relation $\leq$ represents the *partially ordered, well-founded* causality relation among the states of CMPMs based on their *points of entry in time*.

The $Rsync_i$ relations capture the *equality in time* of the *synchronous output* states they relate.

D. Cut- off States of Simulation

The given CFSMs specification contains non-terminating states due to cycles. We unwind these cycles during simulation and terminate them into *cut-off states*. The cut-off state has the property that its corresponding global state vector of CFSMs is identical with one of its predecessors/ancestors in the tree structure generated.

Definition 1: A *path* P_i of the Minimal prefix machine is defined as a sequence of local states starting at the *initial state* s_{0i} and ending at an arbitrary state s''_i such that $P_i = <s_{oi} R_i s_i R_i s'_i R_i ... R_i s''_i>$. The state s''_i is said to be the *maximal state* of P_i.

Definition 2: A *cut-off state is one satisfying the following condition:*

If s_i , s'_i are two states of a given path P_i of M_i such that: $s = (s_1,s_2,...s_n)$, $s' = (s'_1,s'_2,...,s'_n)$ and if the corresponding CFSM-vectors are equal i.e., $(f_{s1}(s_1), f_{s2}(s_2),..., f_{sn}(s_n)) = (f_{s1}(s'_1), f_{s2}(s'_2),..., f_{sn}(s'_n))$, then we say that s'_i which is the descendent of s_i is *isomorphic* with s'_i and is called a *cut-off state* of the CMPM M_i.

III. Distributed Algorithm for Model Checking

The simulated CMPM trees represent a globally interacting set of *heterogeneous automata* without incurring the state-space explosion of the conventional one single *homogeneous synchronous product automaton.* The algorithm to generate the CMPM trees by recursive simulation of given CFSM graphs are reported in [9],[10] and [12]. This enables us to perform the state exploration in a distributed, parallel fashion. Since the locality of each CMPM component is maintained, it is enough to search the corresponding component trees depending on the property to be verified.

A. Assumptions Made in Model-checking

Our model-checking consists of *reachability analysis* of the state vectors. The CMPMs model has distributed the synchronous global-state vectors into *n* interactive components.

The fault-tolerance properties consist of *safety* and *liveness* properties. Safety properties mainly consist of guarantee of absence of *communication deadlocks.* A *deadlocked state* is a state such that there is no outgoing transition is possible. Liveness properties consist of *eventuality* guarantee which means that eventually certain global state vectors are *reachable*. A *reachable state* is a state such that there exists a path from the initial state to the state in question. We assume that all the interesting global-state vectors of scrutiny are synchronous global-state vectors whose reachability can be analysed in a parallel fashion by making depth-first analysis of the CMPM trees independently and thus concurrently.

B. The Model –checking Algorithm

1) Detection of Communication deadlocks

```
/* Parallel checking of all n CMPM-trees Mi, i= 1..n to
check the reachability of leaf-states that are non-cutoff
states. */
deadlock_state_list_i : List of deadlocked states from Mi,
i= 1..n;
find_dead_states_i(s_i)
{
  if s_i is a (leaf-state ∧ not (cut_off_state))
  {
        add (s_i, dead_state_list_i);
   return;
  }
 else if s_i is a leaf-state
  return;
 for all next_states s'_i of s_i
  return(find_dead_states_i(s'_i));
}
Main()
{
 Deadlock_state_list_i:= Null, for all i=1..n;
 Par begin
  For i=1..n do find_dead_states_i(s_0i);
 Par end;
}
```

2) Detection of Liveness Property

```
/* This involves checking all the k CMPM trees Mi,
i=1..k
For the reachability of the given synchronous state
vector.*/
 Chk_tree_i(s_i, s_f)
 {
 if (id(s_i) = s_fi ∧ id(env_j(s_i)) = s_fj for all j = 1..k, k≠ i)
  return(true);
    else if s_i is a leaf state
      return(false) ;
          else for all next states s'_i
       such that: (s_i R_i s'_i) is a transition do
    {
      success_i:= Chk_tree_i(s'_i, s_f);
                          if (success_i) return(true);
                          else continue;
    }
 }/*Chk_tree_i() */

 Main()
 {
    Par begin
      for i = 1..k do
            success_i := Chk_tree_i(s_0i, s_f);
    Par end;
  }
```

C. Complexity of the Model –Checking Algorithms

Since the procedure of distributed model-checking is *recursive*, the proof of correctness can be done by *induction*.

The time complexities of both the algorithms involve *depth-first* recursive *search* of at most all *n* CMPM trees in parallel, checking all the states of each CMPM tree at most once. Thus they are linear in the number of total states, N of all the component CMPMs.

IV. Conclusions, Future Work

We have proposed a couple of model-checking algorithms based on CMPMs model to verify the fault-tolerant properties consisting of *safety* and *liveness* properties. Safety involves detection of communication deadlocked states. Liveness property involves eventual occurrence of certain required synchronous global state vectors. In the future, we plan to extend the model-checking algorithms by proposing a *branching-time temporal logic* whose formulae can be checked using our CMPMs model. Compared to the methods reported in [8] and [13], our method is distributed. Also, compared to the approach reported in [13] and [14], our method is more efficient and easier as there is a static model to perform the verification in our case.

References

[1] W. Reisig, "Towards a temporal logic for causality and choice in Distributed systems," Lecture Notes in Computer Science, 1989, vol. 354, pp. 603-627.
[2] K. L. McMillan, "Using unfolding to avoid the state space explosion problem in the verification of asynchronous circuits," in Proc. 4th Workshop on Computer Aided Verification, 1992.
[3] J. Esparza, S. Romer, and W. Vogler, "An improvement of Mc. Millan's unfolding algorithm," Formal Methods in System Design, May 2002, vol. 20, no. 3, pp. 285-310.
[4] C. A. R. Hoare, "Communicating Sequential Processes," Prentice Hall 1984.
[5] R. Milner, "Calculi for synchrony and asynchrony," Theoretical Computer Science, vol. 25, no. 2, pp. 267-310, 1983.
[6] S. Dakshinamurthy and V. Narayanan, "A parallel algorithm for model-transformation of interactive state machine specification," *International Journal of Wisdom Based Computing*, vol. 2 no. 1, pp. 52-57, Apr 2012.
[7] E. M. Clarke, "Model-checking–My 27 year quest to overcome the state explosion problem," *Lecture Notes in Artificial Intelligence*, 2008, vol. 5330.
[8] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite state concurrent systems using temporal logic specification: A practical approach," in Proc. 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 1983, pp. 117-126.
[9] S. Dakshinamurthy and V. Narayanan, "A fully-distributed checkpointing-protocol for fault-tolerance in real-time distributed systems," in National IETE Conf., 2012.
[10] Vasumathi K. Narayanan, "A state-oriented, partial-order model and logic for distributed systems verification," Ph.D. Thesis, Concordia University, Montreal, 1997.
[11] L. Lamport, "On interprocess communication, Part I: Basic formalisms; Part II: Algorithms," *Distributed Computing*, vol. 1, pp. 77-101, 1986.
[12] S. Dakshinamoorthy and V. Narayanan, "A component-based approach to verification of formal software models to check safety properties of distributed systems," *Submitted to the Conference*, 2013.
[13] J.Esparza and S. Romer, "An Unfolding Algorithm for synchronous products of Transition Systems," in *Proc. International Conference on Concurrency Theory*, 2002.
[14] Valmari, "A stubborn attack on state explosion," *Lecture Notes in Computer Science*, 1991, vol. 531, pp. 156-165.

Dr. Vasumathi Narayanan has completed her B.E. from Anna University, Chennai, M.E. from Indian Institute of Science, Bangalore and Ph D from Concordia University, Montreal Canada in 1997. Her areas of specialization are in formal methods, concurrency theory, model-checking of distributed systems and temporal logics. She has about five years of industrial experience and about fifteen years of research experience. She has co-authored more than ten research publications. Presently she is working as a professor in St.Joseph's college of engineering in Chennai, India.

Sungeetha Dakshinamurthy has completed M.Tech in VLSI at Sathyabama University, Chennai, India. She is an Associate Professor at St. Joseph's college of Engineering, Chennai, India. She also coauthored A Parallel Algorithm for Model-Transformation of Interactive State machine Specification (Wisdom based computing, 2012), A fully - distributed Check pointing - protocol for fault Tolerance in Real –Time Distributed systems (IETE ,2012).She is currently undergoing Ph.D., in Fault tolerance for Dynamic Reconfigurable System .Her experience includes work in the area of VLSI Communication. Her present work is aimed at application of Fault Tolerance in Robotics & Space Systems.