A Component-Based Approach to Verification of Formal Software Models to Check Safety Properties of Distributed Systems

Sungeetha Dakshinamurthy and Vasumathi K. Narayanan

Abstract—In this paper, we model a distributed system consisting of \( n \) processes by a respective set of \( n \) Communicating Finite State Machines (CFSMs). The processes run concurrently and communicate with each other to accomplish a common goal. As opposed to the traditional product automaton built from the specified CFSMs, whose state-space explodes, we build a compressed model of what are defined as Communicating Minimal Prefix Machines (CMPMs) by simulating the CFSMs concurrently in parallel. The states of CMPMs form a well-founded, partial order. This model can be used to perform reachability analysis of the given system to check the safety properties such as communication deadlocks. The algorithm to generate the CMPMs model from CFSMs is presented in pseudo-code and its complexity discussed.

Index Terms—CFSMs, CMPMs, distributed-system, state-space explosion, reachability analysis, communication deadlocks.

I. INTRODUCTION

Concurrency, as a basic primitive for software construction, is more relevant today than ever before, primarily due to the multi-core revolution. General-purpose software applications must find ways to exploit concurrency explicitly in order to take advantage of multiple cores. However, experience has shown that explicit parallel programs are difficult to get right.

The traditional method of handling a given set of CFSMs specification is to compose them into one single product automaton. The main disadvantage of the product machine is that it simulates concurrency by non-deterministic interleaving [1] instead of representing concurrency as it is. Because of this, the state space of product automaton also explodes exponentially with respect to the number of component CFSMs. This means that we have to seek an alternative method of modeling the CFSMs.

Among discrete event systems, those exhibiting concurrency are especially challenging, requiring the use of formal methods to deal with them. Petri nets are a well-established such formalism. However Petri nets do not handle conflicts or the system choices very well.

Compared to the event-based Petri net models, our state-based automata theoretic model captures all the three relations viz., sequence, conflict and concurrency truly within a single canonical structure provided by the transformed CMPMs model. The concurrent set of CMPMs generated by simulating a set of CFSMs in global environment models the spatio-temporal dynamics of the global system, in terms of the dynamics of component sub-systems.

In the sequel, we revise the computational model of CMPMs proposed in [2] and refined in [3]. In [2], we presented a sequential algorithm for the generation of CMPMs model whereas in [3] we introduced a parallel algorithm. In this paper we improve the parallel algorithm of [3] in a more efficient way and discuss its complexity. We indicate the verification approach using the local components of the CMPMs model and conclude the paper by indicating the model-checking approach.

II. THE COMPUTATIONAL MODEL OF CMPMS

This model is developed from a given specification consisting of a set of communicating finite state machines (CFSMs). 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. 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:

\[ F_i = (S_i, S_0, A_i, R_i, R_{sync_i}, R_{sync_{off_i}}) \quad \forall i \in \{1..n\} \]

where,

1) \( S_0 \) is the finite set of states of CFSM \( F_i \), \( S_0 \subseteq S_i \) being the initial state.
2) \( A_i \) is the finite set of asynchronous and synchronous actions of \( F_i \).
3) If \( a_i \in A_i \) is a synchronous action, the list of indices

Manuscript received December 13, 2012; revised April 6, 2013.

S. Dakshinamurthy is with St. Joseph’s college of Engineering and Research scholar in Sathyabama University, India (e-mail:sungeetha5@yahoo.com).

V. Narayanan is with St. Joseph’s college of Engineering, India (e-mail: vasumathin@yahoo.com).

DOI: 10.7763/LNSE.2013.V1.42 186
The binary successor relation \( R_i \) is \( R_\emptyset \) with its events omitted. The binary relation \( \preceq \) represents the partially ordered, well-founded causality relation among the states of CMPMs based on their points of entry in time.

The \( \text{Rsync} \) relations capture the equality in time of the synchronous output states they relate (see Fig. 2).

D. Sequence, Conflict and Concurrency

The three fundamental binary relations viz. sequence (seq), conflict (conf) and concurrency (co) are defined using \( R_i \), \( i \in \{1..n\} \) and \( \preceq \) relations, the latter propagating the local sequence and conflict relations globally across all CMPMs.

7) Sequence: \( \forall s_i \in S_i, s'_j \in S_j \) (s_i seq s'_j) iff: \( \exists s''_j \in S_j : (s_i, R_i, s''_j) \land (s''_j \preceq s'_j) \), i, j \( \in \{1..n\} \), possibly, i \( \neq \) j, \( \exists s''_j \in S_j : (s_i \preceq s''_j) \land (s''_j, R_j, s'_j) \), i, j \( \in \{1..n\} \), possibly, i \( \neq \) j.

8) Local Conflict: \( \forall s_i, s'_j \in S_i, (s_i, \text{conf}, s'_j) \) iff: \( (s_i, s'_j) \notin R_i^* \land (s'_j, s_i) \notin R_i^* \), i \( \in \{1..n\} \).

9) Conflict (Global): \( \forall s_i \in S_i, s_j \in S_j \) (s_i conf s_j) iff: \( \exists s''_k \in S_k : (s''_k, \text{conf}, s''_k) \land (s''_k \preceq s_i) \land (s''_k \preceq s_j) \), i, j, k \( \in \{1..n\} \), i \( \neq \) j, possibly k = i or k = j.

10) Concurrency: \( (s_i, \text{co} s_j) \) iff: \( (s_i, s_j) \not\in \text{seq} \land (s_j, s_i) \not\in \text{seq} \land (s_i, s_j) \not\in \text{conf} \land i \neq j \).

III. PARALLEL RECURSIVE ALGORITHM TO GENERATE CMPMs FROM CFSMs

In a prequel paper [2], we have reported a parallel algorithm for generating CMPMs from given CFSMs. In this work we make it more efficient (less complex) to generate the CMPM components containing the interesting synchronous global state vectors whose reachability can be analysed in parallel due to the component-based computational model.

initialize_instance_numbers()
{
Par begin
for i:= 1 to n do in parallel

for all states $s_i$ in $S_n$ of $F_i$ do in parallel

\[\text{ins}(s_i) := 0;\]

for all actions $a_k$ in $A_{i,j}$ do in parallel

\[\text{ins}(a_k) := 0;\]

} /* for loop */

Par end

initialize_states()

{ for $i := 1$ to $n$ do in parallel

\[s_{0_i} := (\text{id}(s_{0_i}), \text{ins}(s_{0_i})++) ;\]

for $i := 1$ to $n$ do in parallel

\[\text{env}(s_{0_i}) := \{ s_{0_j} \text{ for all } j \neq i \};\]

} /* The procedure below initializes the composed flag of outgoing edges to indicate that outgoing edges on all actions are not yet composed. */

initialize_composed()

{ for $i := 1$ to $n$ do in parallel

\[\text{for all (id}(s_{0_i}), a_{k_i}, s'_{0_i}) \text{ in } R_{0_i}\]

\[\text{composed}(s_{0_i}, a_{k_i}) := \text{Null};\]

} /*The procedure below simulates CFSM $F_i$ and possibly other CFSMs $F_j, j \neq i$ (in case of synchronizations ) to generate CMPM $M_j$ in a depth-first, recursive manner. */

simulate_F($s_i$)

{ if $\text{is_cut_off}(s_i) \text{ return; }$

\[s_{0_i} := \text{id}(s_i);\]

for all $(s_{0_i}, a_{k_i}, s'_{0_i}) \text{ in } R_{0_i}$

\[\text{if } a_{k_i} \text{ is local/asynchronous action } \{

\]

\[\text{if } (\text{composed}(s_{0_i}, a_{k_i}) == \text{Null}) \{

\[e_{k_i} := (a_{k_i}, \text{ins}(a_{k_i})++) ;

\]

\[s'_{0_i} := (s'_{0_i}, \text{ins}(s'_{0_i})+++);\]

\[\text{env}(s') := \text{env}(s);\]

\[R_i := R_i \cup \{ t_{0_i} := (s_i, e_{k_i}, s'_{0_i}) \};\]

\[\text{composed}(s_{0_i}, a_{k_i}) := s'_{0_i};\]

\[\text{for all (a'_{k_i} \text{ from id}(s'))}

\[\text{composed}(s'_{0_i}, a'_{k_i}) := \text{Null};\]

} /* already composed and existing */

else $s'_{0_i} := \text{composed}(s'_{0_i}, a_{k_i});$

} /* for loop */

simulate_F($s'_{0_i});$

/*Simulate_F*/

/* This procedure recursively browses the partner CMPM $M_j$ to check if there is a partner state $s_j$ for $s_i$ participating in the synchronous action $a_{k_i}$ */

find_partner_F($a_{k_i}, s'_{0_i}, s_i, s_j$)

{ if $\text{is_cut_off}(s_i) \text{ return(False); }$

for all $(\text{id}(s_i), a_{k_i}, s'_{0_i}) \text{ in } R_{0_i}$

\[\text{if } (a_{k_i} = a_{k_i}) \text{ then } \{

\]

\[\text{if } (\text{composed}(s_i, a_{k_i}) == \text{Null}) \{

\[\text{compose_env_and_states}(s_i, s'_{0_i}, a_{k_i}, s'_{0_i});\]

\[\text{return(True);}\]

\[\text{else return(find_partner_F}(a_{k_i}, s'_{0_i}, s_i, s_j);\}

} /* for loop */

} /* find_partner_F */

/*This routine below composes the states and environments of the partner states from $M_i$ and $M_j$. */

compose_env_and_states($s_{0_i}, s_{0_j}, s_{0_i}, s_{0_j}$)

{ $s'_{0_i} := (s'_{0_i}, \text{ins}(s'_{0_i})+++);$

\[s'_{0_j} := (s'_{0_j}, \text{ins}(s'_{0_j})+++);\]

\[e_{0_i} := (a_{k_i}, \text{ins}(a_{k_i})+++);\]

\[e_{0_j} := (a_{k_j}, \text{ins}(a_{k_j})+++);\]

\[R_i := R_i \cup \{ (s_i, e_{0_i}, s'_{0_i});\]

\[R_j := R_j \cup \{ (s_j, e_{0_j}, s'_{0_j});\]

\[\text{composed}(s_{0_i}, a_{k_i}) := s'_{0_i};\]

\[\text{composed}(s_{0_j}, a_{k_j}) := s'_{0_j};\]

\[\text{for (k := 1 to n do, k \neq i) } \{

\]

\[\text{env}(s_{0_i}) := \text{max(env}(s_{0_i}), \text{env}(s_{0_j}));\]

\[\text{env}(s'_{0_i}) := s'_{0_i};\]

\[\text{for (k := 1 to n do, k \neq j) } \{

\]

\[\text{env}(s_{0_j}) := \text{max(env}(s_{0_i}), \text{env}(s_{0_j}));\]

\[\text{env}(s'_{0_j}) := s'_{0_j};\]

\[\text{for all (a'_{k_i} \text{ from id}(s'))}

\[\text{composed}(s'_{0_i}, a'_{k_i}) := \text{Null};\]

\[\text{for all (a'_{k_j} \text{ from id}(s'))}

\[\text{composed}(s'_{0_j}, a'_{k_j}) := \text{Null};\]

} /* for loop */

/* The routine below removes the instance numbers to form the CFSM vector from the CMPM vector */

Fsm_vector(s_i)

{ $s_{0_i} := \text{id}(s_i);$

\[\text{for (k := 1 to n, k \neq i) } \{

\]

\[f_{-}\text{env}(s_{0_i}) := \text{id} (\text{env}(s_{0_i}));\]

\[f_{-}\text{vector} := s_{0_i}, f_{-}\text{env}(s_{0_i});\]

\[\text{return}(f_{-}\text{vector});\]

} /* This routine below checks the entire path of predecessors for repetition of the CFSM state vector. */

is_cut_off(s_i)

{ $s_{t} := \text{Fsm_vector}(s_i);$

\[\text{while } (s_i \neq s_{0}) \text{ do } \{

\]

\[s_i := \text{parent}(s_i); \text{ /* trek the ancestors’ path */}

}
if (Fsm_vector(s_i) == s_f) return(true);
}
/* is_cut_off()*/

Main()
{
    initialize_instances();
    initialize_states();
    initialize_composed();
    for (i:=1 to n) do
        Par begin
            Simulate_F(s_0_i);
        Par end;
}

IV. COMPLEXITY ANALYSIS

The complexity of the above algorithm is better than the complexity of the algorithm reported in [2]. This is mainly because in this work, 1) we use certain flags to avoid repetitive state and edge generation of CMPM trees while generating them in parallel and 2) therefore there is no need for simulating the waiting of processes. This cuts down the complexity of the algorithm and also makes it more readable and easier to comprehend.

Assuming there are a maximum of N_f states per CFSM, due to the degree of coupling and synchronization each CMPM will have N = k*N_f states where k is a worst-case constant depending on the degree of synchronization across CFSMs. In order to generate each of the N CMPM states, in the worst case we browse all the (n-1) potential partner CMPMs and each of their N states. Thus the complexity of generating all N states per CMPM is N^*(n-1)*N) which is (n-1)*N^2. This amounts to a total complexity of n*(n-1)*N^2 or in other words, n^2*N^2 which is equal to k^2*n^2*N^2. This is the time complexity of CMPMS generation algorithm which is quadratic (polynomial) in the number of given CFSMs. This must be compared against the exponential time complexity of N_f^n in the case of conventional synchronous product machine. The space complexity is linear in the number of states N mainly due to the storing of instance numbers and composed flags.

V. CONCLUSION

We have proposed a refined parallel algorithm to generate the CMPMs model which is more comprehensive and less complex than the one proposed in [2]. This model can be used to do component-wise formal verification by way of model-checking, as discussed in the sequel paper [5].

REFERENCES


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 Checkpointing - 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 present work is aimed at application of Fault Tolerance in Robotics & Space Systems.

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.