

Towards A Nanoscopic Model To Derive Macroscopic Behavior Of Swarm Robotic System And Its Reachability Analysis Without State-Explosion

Sungeetha. D and Dr.Vasumathi Narayanan

*Department of Electronics & Commn. Engg..
Research Scholar Sathyabama University Chennai, India
sungeetha5@yahoo.com*

*Department of Information technology
St.Joseph's college of Engineering Chennai,India
vasumathin@yahoo.com*

Abstract

In this paper, we propose a nanoscopic model, which captures the intra-robotic as well as inter-robotic interactions at a level lower than the microscopic level that captures the individual robotic behavior as a whole, which in turn is at a level lower than the macroscopic collective behavior of the swarm of robots. Instead of considering an individual robot as an atomic unit, we represent certain independent, parallel activities that go on inside a robot as a set of multiple, sub-atomic and concurrent agents that communicate with each other as well as with such agents of other robots of the swarm.

Keywords— Swarm robotics, swarm intelligence, nanoscopic intra-robot behavior, microscopic inter-robot behavior, macroscopic collective behavior, synchronization, state-explosion.

1. INTRODUCTION

Swarm robotics is a branch of multi-robot systems that embrace the ideas of biological swarms such as insect colonies, flocks of birds and schools of fish. The term “swarm” is used to refer “*a large group of locally interacting individuals with common goals*” [1,2]. Swarm robotics systems as well as their biological counterparts consist of many individuals exhibiting simple behaviors. While executing these simple behaviors, individuals are capable of producing complex collective behaviors on the swarm level that no individual is able to achieve alone. Ant colony can be

viewed as an example – a single ant has limited sensing capabilities and relies only on local information, but by working together the colony is able to perform rather complex foraging, construction and transportation tasks.

One important additional observation that we make in this research is that in the biological counterparts of swarm robots, say school of fish or ant colony, an *individual* behaviour is constituted by many local behaviours that happen *in parallel*. A fish for example senses information from its environment, processes the sensed information, sends it to its neighbours and also receives some information from its neighbours all of them happening in parallel while it is swimming/in motion using its various organs acting and reacting *concurrently*. Similarly, in an ant colony, depending on the individual ant's role, it does multiple activities along with motion in parallel.

Swarm robots are similarly designed such that each individual robot has to perform multiple activities such as *sensing* information, *processing* it, in addition to *sending* and *receiving* certain information, all happening in *parallel* often simultaneously with *motion*. As noted in [3], swarm robotics systems are characterized by simplicity of individuals, local sensing and communication capabilities, *parallelism in task execution*, scalability, heterogeneousness, flexibility and decentralized control among each other. We can carry this *parallelism one more step forward within an individual robotic unit as well*, with its sub-atomic units doing certain activities simultaneously in parallel.

Swarm robotics has been defined as a novel approach to the coordination of large numbers of robots and as the study of how large numbers of relatively simple, physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among robots and between the robots and the environment. [1]. The design and analysis of individual robotic behavior is considered *microscopic* and those of the collective swarm behaviour is considered *macroscopic*[4]. In this work, we go one step deeper in implementing *parallelism in task execution*, which we claim exists not only among robots (*inter-robot*) in the *microscopic* level but also within each individual robot (*intra-robot*) in what we propose as *nanoscopic/local* level which, as we demonstrate, can be used to mathematically model and derive the *macroscopic* global behavior.

Therefore in this paper, we split each *atomic robot* into multiple *sub-atomic agents* with following characteristics as opposed to those given in [5]:

- A robot has multiple *sub-atomic agents* that are *autonomous*;
- A robot's multiple activities are delegated to these sub-atomic agents that proceed *independently and concurrently in parallel*.
- Robotic agents *do not have access to centralized control* or global-state information of the parent or other robots.
- Robotic agents cooperate to tackle a given *common goal/task*.

1.1 State of the art and Related work:

There are models in the literature which utilize global knowledge [15][16], by having a centralized control. In our work, our model is fully distributed but still derive global

knowledge at synchronization points as opposed to the studies in [17][19]. Our multi-agent model enjoys both being fully distributed as well as deriving vital global knowledge by exchanging with each other at synchronization points /rendezvous.

In the sequel of the paper, we describe a methodology based on CSP model[6] to automatically derive the macroscopic collective behaviour of swarm robots, given the nanoscopic specification of sub-atomic agents of individual robots expressed as a set of CFSMs (Communicating Finite State Machines). Section 2 describes the methodology proposed in [7], [14] and repeated here for clarity and continuity. Section 3 lists the parallel version of the pseudo code of the algorithm, which takes the nanoscopic input with local states and translates it to the macroscopic output with global states of the swarm robotic system. Section 4 describes a case study of a collective transport system of swarm robots dealt with in [8,9], to which our methodology is applied. Section 5 discusses the formal verification of swarm robotic systems Section 6 conclude the paper.

2. THE METHODOLOGY

The multiple sub-atomic robotic agents of an individual robot are specified as a set of CFSMs as mentioned above. Each CFSM performs certain actions from its different states. Some actions are *asynchronous /local* to the agent and some actions are *synchronous/global* in the sense that each synchronous action has a *sending agent* and one or more corresponding *receiving agents* of the same or different robots, participating in the synchronous action. An event is an instance of an action.

We process the specification consisting of a set of n CFSMs representing the n robotic agents into what are called as CMPMs (Communicating Minimal Prefix Machines) that are nothing but the simulated version of the CFSMs in global environment. Every robot has in its environment, other robots and also the common general environment. CFSMs have local knowledge while the CMPMs have global knowledge based on the *state* information.

In particular, we transform the given n CFSM *graphs* into a corresponding set of n CMPM *trees* whose leaves correspond to what are defined as *cyclic/cutoff states* or failed *deadlocked states* or successful *goal states*.

Each state of a CMPM represents not only its corresponding local CFSM state, but also a vector of $(n-1)$ non-local CFSMs in its environment.

Definition:

A CFSM $F_i = (s_{0f_i}, S_{f_i}, A_{f_i}, R_{tf_i}, R_{sync_{f_i}}, R_{sync_{0f_i}}) \forall i \in \{1..n\}$ where[14],

- S_{f_i} is the *finite* set of states of CFSM F_i , $s_{0f_i} \in S_{f_i}$ being the *initial state*.
- A_{f_i} is the *finite set* of *asynchronous* and *synchronous actions* of F_i .
- If $a_{f_i} \in A_{f_i}$ is a *synchronous action*, the list of indices $[j_1, j_2, \dots, j_k], k \leq n$ of the partner CFSMs are also specified in the square brackets along with a_{f_i} .
- R_{tf_i} is a *ternary transition relation* such that: $R_{tf_i} \subseteq S_{f_i} \times A_{f_i} \times S_{f_i}$. In a so-called *i-transition* $(s_{f_i}, a_{f_i}, s'_{f_i}) \in R_{tf_i}, s_{f_i}$ is called the *input state* and s'_{f_i} the *output state*.

An i -transition $(s_{fi}, a_{fi}, s'_{fi}) \in R_{fi}$ is called *synchronous* if $a_{fi}[j_1, j_2, \dots, j_k]$, is a *synchronous action* such that : \exists a set of j -transitions $(s_{fj}, a_{fj}, s'_{fj}) \in R_{fj}$ $\forall j \in \{j_1, j_2, \dots, j_k\}$ $j \neq i$ where $a_{fi} = a_{fj}$ and $(s'_{fi}, s'_{fj}) \in R_{sync_{fi}}$.

- $R_{sync_{fi}} \subseteq S_{fi} \times S_{fj}$, $i \neq j$, $j \in \{1..n\}$, is a binary relation which relates the *output states of synchronous transitions*.
- $R_{sync_{0,fi}} \subseteq R_{sync_{fi}}$ relates the set of pairs of initial states: $R_{sync_{0,fi}} = (s_{0,fi}, s_{0,fj})$, $\forall j \in \{1..n\}$. All the initial states are assumed to be in pair wise synchrony with each other to begin with [14].

Definition:

A CPM $M_i = (S_{0i}, S_i, E_i, R_{fi}, R_{sync_{fi}}, R_{sync_{0,fi}})$, $\forall i \in \{1..n\}$ is a 6-tuple where, [14] 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, $R_{sync_{fi}}$, $R_{sync_{0,fi}}$ can be defined similarly as corresponding instances of CFSM entities $R_{sync_{fi}}$ and $R_{sync_{0,fi}}$. [14].

2.1 Well-founded, Partially-Ordered Causality Generation

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

Definition:

The global, temporal causality order is composed using the binary relations $R_{sync_{fi}}$ and R_{fi} , $i \in \{1..n\}$ as follows [7, 14]:

$$\leq := \sum_{i=1..n} (R_i \cup R_{sync_{fi}})^*$$

The binary successor relation R_i is derived from the ternary relation R_{fi} with its event omitted. The binary relation \leq represents the *partially ordered, well-founded causality relation* among the states of CPMs based on their *points of entry in time*.

The $R_{sync_{fi}}$ relations capture the *equality in time* of the *synchronous output states* they relate.

Using R_i , $R_{sync_{fi}}$ relations, we further define a set of 3 *global relations* of *sequence(seq)*, *conflict/choice(conf)* and *concurrency(co)* among the states of all the CPMs. whose definitions can be found in [7]. Every pair of states of all the CPMs are related by one of the above 3 relations.

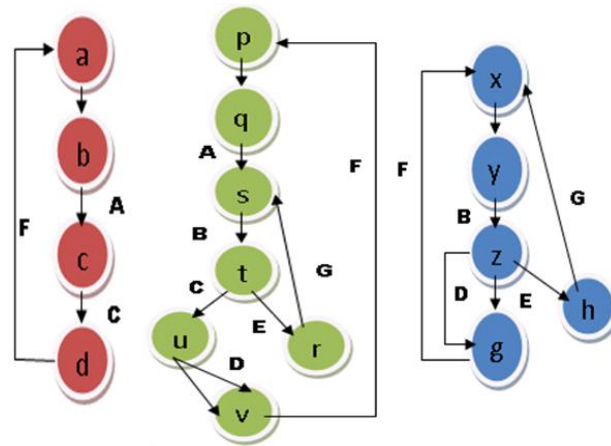


Fig 1. Given set of CFSM graphs representing a set of 3 Nanoscopic agents of a Robot

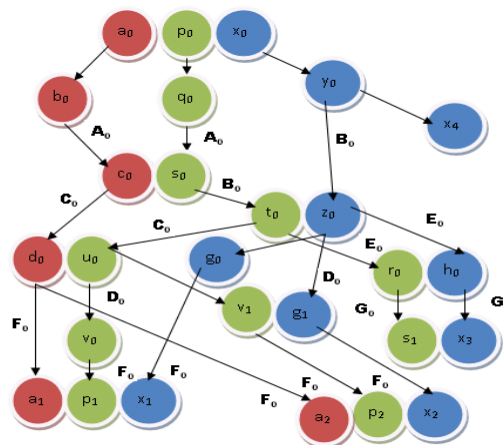


Fig 2. Deduced set of CMPM Trees representing the Macroscopic Swarm-robotic behaviour

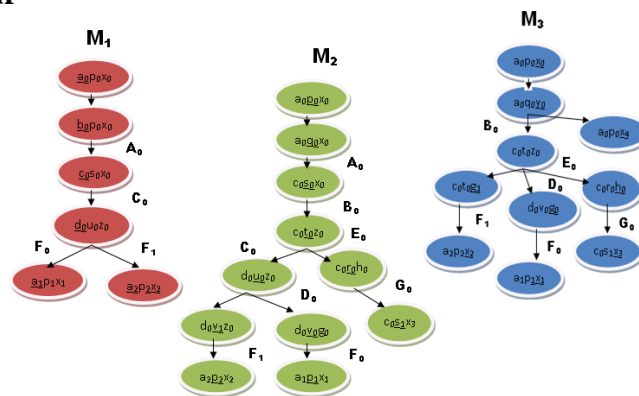


Fig 3. Disjoint CMPM Trees of Fig 2.

2.2 An Example of CFMS-graphs and their Simulation into Finitely terminating CMPM-trees

Fig.1 shows an example [14] of a set of three ($n = 3$) specified CFMSs. The *states* are labeled with *lower case letters* and *synchronous actions* are labeled with *upper case letters*. The *asynchronous actions* of local CFMS transitions are *not labeled* for simplicity. Fig. 2 shows the corresponding set of CMPMs which are simulated CFMSs in global environment. The CMPM states a_0, b_0, c_0 of Fig.2 are instances of the respective CFMS states a,b,c etc. in Fig. 1. Similarly, the CMPM *events* A_0, B_0, C_0 etc., of Fig. 2 & Fig 3 are instances of the corresponding CFMS *actions* A, B, C etc., of Fig. 1. The states that are glued together in Fig. 2 represent the ones related by the $Rsync_i$ relation, the causal equality in time.

The leaves of the three CMPM trees represent the *cut-off* states. The state a_1 in $CMPM_1$ is a *cut-off state* since along with its no-local environment vector (p_1, x_1) , it represents the synchronous vector (a_1, p_1, x_1) which happens to be the descendent of the initial vector (a_0, p_0, x_0) both being the instances of the same FSM-state vector (a, p, x) . Thus a_1 and a_0 are *isomorphic* to each other in the sense that the descendent states and transitions of a_1 will be identical to that of a_0 and hence need not be generated again. Thus a_1 is called the cut-off state of a_0 . Similarly, state a_2 is isomorphic with a_0 and is a cut-off state. The state s_1 in $CMPM_2$ is isomorphic with s_0 since their respective global-state vectors (c_0, s_1, x_3) and (c_0, s_0, x_0) respectively are instances of the same CFMS vector (c, s, x) . The state x_4 in $CMPM_3$ is isomorphic with x_0 by the same token and is a cut-off state.

2.3 The Minimal-Prefix Vector, the Central Notion behind CMPMs that reduces State-explosion

We associate each state s_1 of $CMPM_1$ with a non-local *synchronous vector* (s_2, s_3, \dots, s_n) of size $(n-1)$ belonging to $CMPM_2, CMPM_3, \dots, CMPM_n$ respectively forming the environment of s_1 , referred to as the *environment vector* of state s_1 of $CMPM_1$. Each component of (s_2, s_3, \dots, s_n) is a *synchronous output state* whose entry is necessary to guarantee the entry of some or possibly all of the other components followed by the entry of s_1 in order to satisfy the synchronization requirements. Recursively, each component of the vector (s_2, s_3, \dots, s_n) in turn is associated with its own environment vector. The $(n-1)$ sized environment vector is referred to as the *minimal prefix vector* in the sequel of the paper, since each component, say s_2 of this vector is to be *necessarily/minimally* reached (possibly $CMPM_2$ may transit past the component s_2 to one of its descendent states as well, asynchronously of $CMPM_1$ thus qualifying the necessity/minimality condition of s_2 's entry). Thus the simulated CFMS machine in global environment is called communicating minimal-prefix machine (CMPM) since each state is associated with a minimal-prefix vector as its environment.

2.4 CMPMs and the traditional Product Machine

We believe that the set of CMPM states with their respective minimal-prefix vectors sufficiently comprise the interesting global state vectors of the traditional product machine. The rest of the state vectors of the product machine if needed, can be reached by traversal from one state-vector say (s_i, env_i) of $CMPM_i$ to another vector

(s_j, env_j) of $CMPM_j$, $i \neq j$ where s_j is the j^{th} component of the environment vector env_i of s_i .

2.5 The Elimination of State-Explosion due to Non-Deterministic Interleaving in CMPMs

Consider a conventional product machine composed by two component CFSMs. Consider a state vector (s_1, s_2) of the product machine with s_1 representing the state of $CFSM_1$ and s_2 representing that of $CFSM_2$. Consider two local/asynchronous state transitions (s_1, a_1, s'_1) and (s_2, a_2, s'_2) of $CFSM_1$ and $CFSM_2$ respectively. In the traditional construction of the product machine, these two transitions are modeled by choosing either one of the two transitions to happen first followed by the other in sequence. This would lead to the following product machine state transitions as shown in Fig 4 consisting of $2^2 = 4$ transitions representing the 2 independent local/asynchronous transitions of $CFSM_1$ and $CFSM_2$. If we extend this representation of non-deterministic interleaving to n different local/asynchronous transitions of n respective CFSMs, $CFSM_1, CFSM_2 \dots CFSM_n$, we would need 2^n transitions to represent n independent asynchronous transitions of the n CFSMs in the case of product machine. On the other hand, in the case of CMPMs, the n local/asynchronous transitions of the n CFSMs would be modeled by exactly n transitions of the respective CMPMs. This is the main advantage of CMPMs in that the exponential explosion of states due to non-deterministic interleaving is avoided since the locality of each CMPM is maintained.

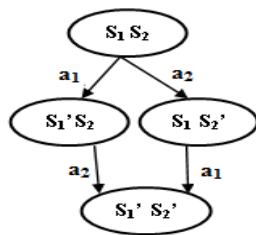


Fig 4a. Concurrency

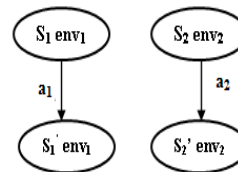


Fig 4b. True concurrency

mimicked by
non-deterministic
interleaving

2.6 Equivalence classes formed by the reachable CMPM State-vectors

Each state of every CMPM M_i , $i \in \{1..n\}$ is associated with a 2-tuple: (s_i, env_i) , with a *primary* state component s_i and a *secondary* environment vector component env_i consisting of $(n-1)$ synchronous output states of the non-local CMPMs M_j , $j \neq i$. Together, (s_i, env_i) forms an instance of a reachable global state vector of the CMPMs. This particular vector can be perceived as a representative of all the other CMPM vectors that can be reached asynchronously to s_i forming the 2-tuple $(s_i, async_env_i)$

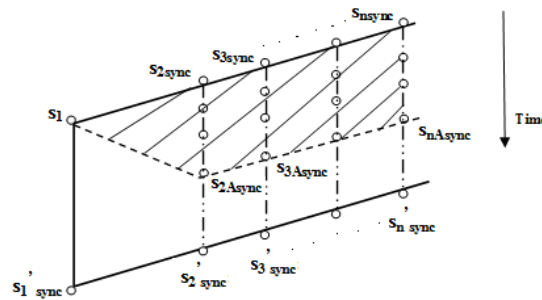
where $async_env_i$ is the set of all the $(n-1)$ -tuple vectors that are reachable asynchronous to s_i by advancing the non-local CMPMs, $M_j, j \in \{1..n\}, j \neq i$ and anchoring the local CMPM M_i at s_i . The only difference between the couple of state vectors (s_i, env_i) and $(s_i, async_env_i)$ is that the former is explicitly stored *statically* in CMPM M_i , whereas the latter is not explicitly stored but can be generated *dynamically* by allowing the non-local CMPMs $M_j, j \neq i$, to make their state transitions asynchronous to M_i , after anchoring the latter at s_i (fig 5).

Definition:

We define a binary *equivalence relation* R_{s_i} that relates every pair of a group/class of reachable CMPM state vectors such that the i^{th} component of both the related vectors is s_i .

That is, $(s R_{s_i} s')$ iff: s_i is the i^{th} component of both s and s' .

It is easy to check that R_{s_i} is *reflexive, transitive* and *symmetric* and thus is an *equivalence relation*. Thus, every CMPM state along with its environment vector is one representative of the whole *equivalence class* of state vectors reachable asynchronously of it. The saving gained by not storing the entire equivalence class of reachable state vectors but only one representative of it is the significant advantage of CMPM construction as opposed to the traditional product machine construction, the latter statically generating all the reachable state vectors thus incurring the interleaved *state-explosion*.



◦ Fig 5.Minimal prefix vector and its equivalence class of states----States

Theorem

The set of states of all n CMPMs along with their respective environment vectors $(s_i, env_i), i \in \{1..n\}$ comprise only a subset of the reachable state vectors of the conventional *product machine* such that the former set consists of only one representative vector per each *equivalence class of state vectors* formed by the latter set.

Proof

The proof follows from the construction of CMPMs from the CFSMs whose pseudo code is given below. The construction method simulates every CFSM in its non-local environment of other $(n-1)$ CFSMs just like the traditional product machine except

that all the n state transitions are generated from a single state vector in the case of product machine while on the other hand the same set of n transitions are distributed among the n different CMPMs in the latter case. Thus all the state transitions of the product machine are generated in the CMPMs *modulo* the *repeated* state transitions of the product machine due to non-deterministic interleaving. This saves an exponential number of states in the CMPMs by avoiding such interleaving by restricting to local transitions alone but at the same time not losing track of the globality, by storing the *synchronous environment vector* of every CMPM state.

3. PSEUDO CODE FOR THE GENERATION OF CMPMS FROM THE GIVEN CFSMS

Input: Set of n CFSMs specification in the form of n *graphs* with local state information representing swarm robotic system in *nanoscopic* level.
 Output: Set of n CMPMs in the form of n *trees*, with global state information representing the swarm robotic system in *microscopic* level.

The algorithm given below is the parallel version of the one presented in [14]. The algorithm unfolds the given CFSMs into CMPMs by recursive simulation of CFSM states and transitions in parallel. Due to their inter-dependencies, every path generation of a given CMPM causally leads to the generation of certain paths of non-local CMPMs as well. Thus the CMPMs are generated as a group concurrently according to synchronization requirements instead of one at a time sequentially. A consistent global-state-vector is composed/generated after every synchronization point. Thus every state generation involves the generation of not only its local predecessors but also certain non-local predecessors from other CMPMs, culminating in a unique environment vector of the generated state.

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

```
generate_transitions_CMPMi(si)
{
  if (iscutoff(si)) return;
  for all FSM-transitions rfi = (sfi, afi, s'fi) from si do
  {
    if (afi is local/asynchronous action)
      gen_state_vector(si, rfi);
    else if (afi is global/synchronous with afj of CFSMj)
    {
      partnerj := find_partner(si, rfi, env_vectorj(si));
      if (partnerj ≠ Null)
        s'i := gen_state_vectors(si, rfi, partnerj);
      else return;
      generate_transitions_CMPMi(s'i);
    }
  }
}
```

```

}/*else*/
}/*for */
}/*generate_transitions_CMPMi ()*/

/* This procedure recursively browses the partner CMPM
Mj to check if there is a partner state sj for si participating in the synchronous action
afi. If so, it generates the next transitions of both partners if not generated already. */

find_partnerj(si, rfi, sj)
{
if (is_cut-off(sj)) return(Null);
for all CFMSMj transitions rfj=(sfj, afj, s'fj) from sj do
{
if (afi(rfi) matches afj) then
return (partnerj= (sj, rfj));
else for rj = (sj, afj, s'j)
find_partnerj(si, rfi, s'j);
}
return (Null);
}/* find_partnerj()*/

/*The routine below composes the states and environments of the partner states from
CMPMs Mi and Mj. */

generate_state_vectors(si, rfi , partnerj)
{
s'i := gen_state_vector(si, rfi);
s'j := gen_state_vector(partnerj);
synchronize_env_vectors(s'i, s'j);
}

gen_state_vector(si, rfi)
{
s'i := (s'fi(rfi) , instance#(si, rfi));
env_vector(s'i) := env_vector(si);
Ri := Ri ∪ (si, afi(rfi), s'i);
}

synchronize_env_vectors(si, sj)
{

for k from 1 to n such that k ≠ i,j do
New_envk := greater_of_Rkplus_k(env_vectork(si),
env_vectork(sj));
New_envi := si;

```

```

New_envj :=sj;
Rsync := Rsync U (si, sj);
}/*sync_env_vectors() */

greater_Rkplus(sk, s'k)
{
if (sk Rk+ s'k) return(s'k); else return(sk);
}

is_cutoff(s'i)
{
s'f := CFSM-vector(s'i) /*state vector without
instance numbers */
if (si Ri+ s'i) such that CFSM_vector (si) = s'f
return(true);
else return(false);
}

Main()
{
for (i:=1 to n) do
Par begin/*in parallel */
generate initial_statess0i := (sf0i,0);/*0th instance*/
generate_transitions_CMPMi(s0i) ;
Par end;
}

```

3.1 Size of CMPMs and the Complexity of their Generation

If N is the maximum number of states per CFSM and if there are n interacting CFSMs the worst case size of the synchronous product machine is N^n , which is exponential in the number of CFSMs. On the other hand, the size of CMPMs is only $d*n$ where d is a constant representing the *degree of synchronization* among CFSMs.

In other words, by the degree of synchronization, we mean and assume that there are at most d instances of each of the CFSM states in a corresponding CMPM and in order to generate the synchronous partner states and environment state vector of each CMPM state, in the worst case, every state of all the n CFSMs will be either generated or visited at most d times. Thus the generation complexity is $d*n*N$.

In order to check the cut-off state status, each state will be visited in the worst case N times. Thus the generation complexity is $d*n*N^2$.

4. CASE STUDY OF A COLLECTIVE TRANSPORT SYSTEM

In this section, we specify the swarm robotic system for collective transport discussed in [8,9]. There are 3 identical robots that work *in parallel* in the *microscopic/atomic*

level to collectively transport an object from a specified location in the arena to a specific common goal area. As explained in the previous sections, we design a set of 7 different activities of each robot that go on *in parallel* at the *nanoscopic/sub-atomic* level. Each activity is represented by a CFSM process/module which models a sub-robotic agent. Thus in total we design 7 different sub-atomic agents represented by 7 different CFSMs that act and interact in parallel, constituting an atomic robot.

Following are the 7 multiple activities/processes/modules of a robot as per the example given in our case study of [8] (fig 6): (1) Light sensor and goal direction setter (Lgd), (2) Distance sensor and distance scanner (DS), (3) Direction arbiter taking care of goal direction (DA,) (4) Direction mediator interacting locally with DA module (DM_l), (5) Direction mediator interacting globally with R&B module (DM_g), (6) Range and bearing module (R&B) and (7) Motion control module (MC). Thus there are 7 agents at the nanoscopic/sub-atomic level, sending and receiving information to each other synchronously in addition to processing information locally/asynchronously, as shown in Fig (1) of our case study[8], describing through a diagram, the various modules which more or less coincide with the above 7 autonomous activities, we have assumed.

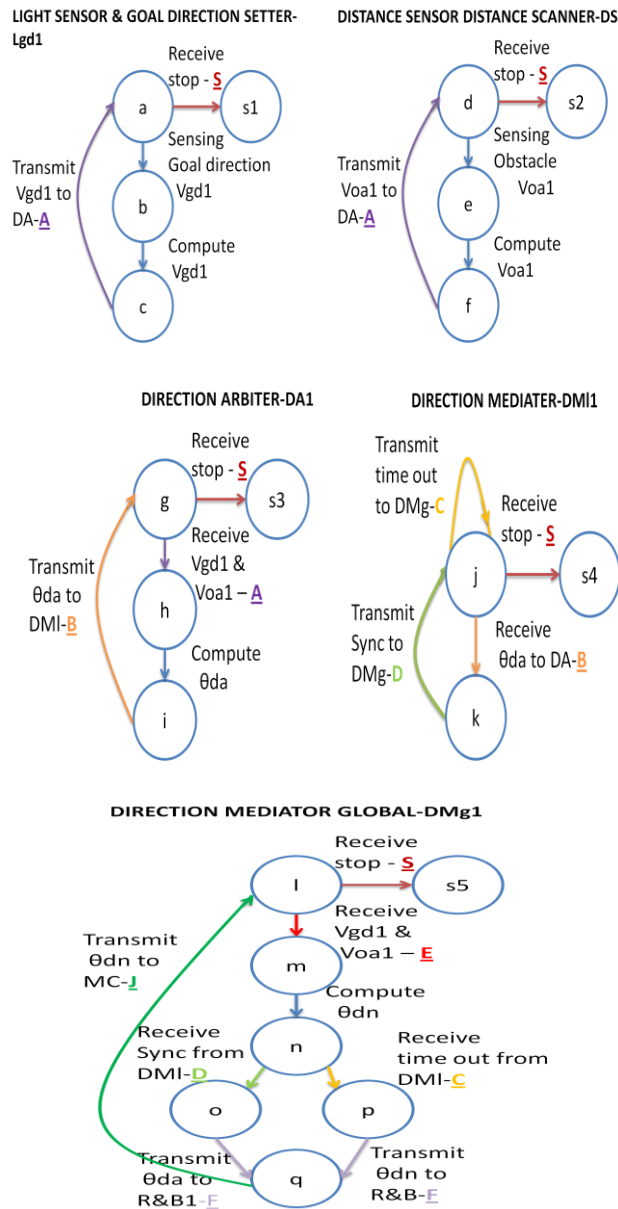
In the CFSM module the robots senses the environment and calculates the desired direction and obstacle. The robot could sense or not the position of the goal and/or the position of obstacles. Depending upon the available information in different step, a robot can be informed, that is, it has a desired direction to follow, or non informed otherwise. If informed robots communicate to other robots their desired direction. Therefore to actuate the wheels, any robot uses the globally mediated direction obtained by averaging the received directions, so all the robot can follow the same direction even if they have different assessment of the environment.

Light Sensor & Goal direction Setter calculates the goal direction and returns the measured direction in form of vector V_{gd} . Distance Sensor distance scanner detects the presence of obstacle and calculates the vector V_{oa} . The Direction Arbiter receives the input V_{gd} and V_{oa} and calculates the θ_{da} , that is the desired direction of the robot before computing the mediated direction. Direction Mediator (local) receives θ_{da} or transmit time out if not received and Direction Mediator (Global) calculates the mediated direction θ_{dn} as the average of direction received from other robots through the range and bearing communication system. The Range and Bearing send a message m to near by robots containing θ_{da} . If robot is informed, and θ_{dn} otherwise. The Motion control receives θ_{dn} and converts it to wheel speed W^r and W^l .

The modules represented in CFSM are cyclic, rooted is simulated in their respective global environments into a corresponding set of CMPMs, each represented by a rooted tree structure. The CMPM (M1, M2, M3, M4, M5, M6, and M7) in which analysis is done independently and thus concurrently. The states in CFSM (1) Light sensor and goal direction setter (Lgd), the states (a,b,c,s1) are denoted as (a0,b0,c0,s1) in CMPM and global action as (A,S) and local action (K,L). The further module are shown in Table 1.

Table 1: Representing Modules & states

S.No	CFSM Module	CFSM-States	CMPM- States	CMPM -Actions	
				Global	Local
1	(Lgd)	(a,b,c,s)	(a0,b0,c0,s1)	(A,S)	(K,L)
2	(DS)	(d,e,f,s)	(d0,e0,f0,s2)	(A,S)	(M,N)
3	(DA)	(g,h,I,s)	(g0,h0,i0,s3)	(A,B,S)	(O)
4	(DM _l)	(j,k,s)	(j0,k0,s4)	(B,C,D,S)	-
5	(DM _g)	(l,m,n,o,p,q,s)	(i0,m0,n0,o0,p0,q0,s5)	(C,D,E,F,J,S)	(P)
6	(R&B)	(r,t,u,s)	(r0,t0,u0,s6)	(E,F,I,S)	-
7	(MC)	(v,w,x,s)	(v0,w0,x0,s7)	(J,S)	(Q, ε)



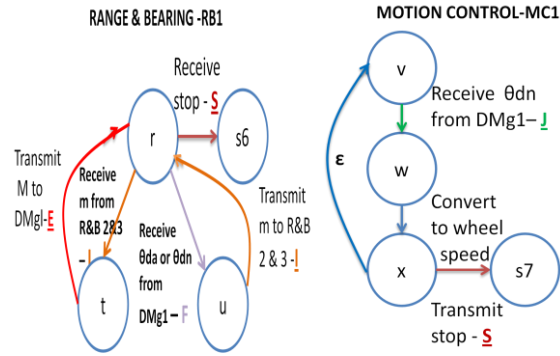
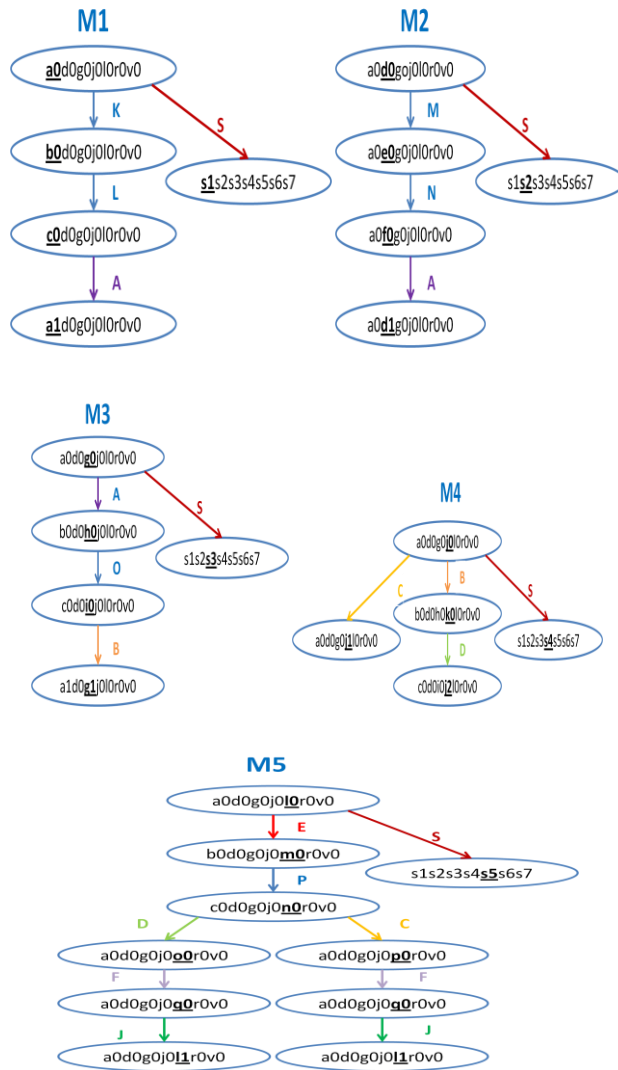
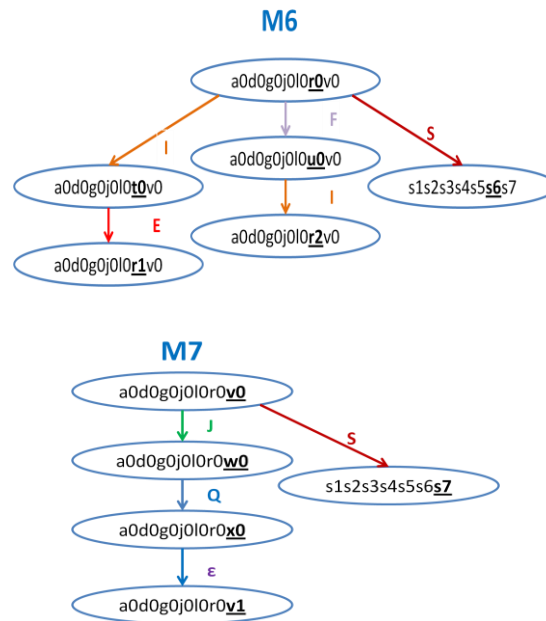


Fig 6: Multiple activities/processes/modules of a robot





5. FORMAL VERIFICATION OF SWARM ROBOTIC SYSTEM

In this section, we introduce the technique of formal verification based on our methodology of CMPMs.

5.1 Detection of Communication deadlocks(Safety Property)

/* Parallel checking of all n CMPM-trees M_i , $i= 1..n$ to check the reachability of *leaf-states* that are *non-cutoff* and *non-final* states. */

deadlock_state_list _{i} : List of deadlocked states from M_i , $i= 1..n$;

find_dead_states _{i} (s_i)

```
{
If  $s_i$  is a (leaf-state  $\wedge$  not (cut_off_state)  $\wedge$  not(final-state) )
```

```
{
add ( $s_i$ , deadlock_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_statesi(s0i);
Par end;
}

```

5.2 *Reachability analysis without state-space explosion (Liveness property)*

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 CPM trees in parallel, checking all the states of each CPM tree at most once. Thus they are linear in the number of total states, N of the biggest sized component CPM.

```

Chk_treei(si, αi)
{
if (si satisfies the predicate αi ∧ i < k)
{
j := i+1;
if (si Rsynci (sj=envj(si)))
return(Chk_treej(sj, αj));
}
else if (si satisfies the predicate αi ∧ i = k)
return(TRUE);
for all next states s'i such that: (si Ri s'i) of Mi do
{
result := Chk_treei(s'i, αi);
if (result == TRUE) return(TRUE);
else continue;
}
return(FALSE);
}/*Chk_treei()*/

```

```

Main()
{
int i,k;
i := 1;
return(chk_treei(s0i, αi));
}

```

6. Conclusions, Future Work

We have proposed in this paper, a nanoscopic model to represent swarm robotic systems using sub-atomic agents specified as a set of CFSMs. Given a set of CFSMs, instead of constructing the conventional total-ordered product machine, we have proposed a partial-ordered CPMs model that cuts down the number of global state vectors stored. Called minimal prefix vectors, we store only the global synchronous vectors using which and their respective equivalence classes, we generate the rest of

the combinatorial states dynamically on need basis depending on the properties to be verified.

We apply this methodology to do the reachability analysis of a swarm robotic system involved in a collective transport system application.

REFERENCES

1. A. Liekna et al., "Towards practical applications of Swarm Robotics: Overview of Swarm Tasks", Proceedings of Engineering for Rural Development May 2014.
2. Barca, J.C., Sekercioglu, Y.A.: Swarm robotics reviewed. *Robotica* 31, pp. 345-359 (2013)
3. Yogeswaran, M., Ponnambalam, S.: *Swarm Robotics: An Extensive Research Review*. Advanced Knowledge Application in Practice, Igor Fuerstner (Ed.) (2010)
4. K.Lerman et al., "A review of Probabilistic Macroscopic models for Swarm Robotic systems", In *Swarm Robotics workshop LNCS 3342*, pp143-152, Springer Verlag, Berlin 2005.
5. M.Brambilla et al, "Swarm Robotics: A review from Swarm engineering perspective", TR/IRIDIA/2012-014, May 2012.
6. C.A.R.Hoare, "Communicating Sequential Processes", Prentice Hall 1984.
7. Vasumathi, K. Narayanan, "A state-oriented, Partial-order model and Logic for Distributed Systems Verification, Ph.D. Thesis, Concordia University, Montreal, 1997.
8. E.Gjondrekaj et al, "Towards a formal verification methodology for collective Robotic Systems" FEM'12, Proc. Of 14th international conference on Formal engineering methods: formal methods and software engineering
9. E. Ferrante et al., "Socially-mediated Negotiation for Obstacle avoidance in collective Transport", *Distributed Autonomous Robotic systems, Springer Tracts in Advanced Robotics Vol 83*, 2013, pp571-583.
10. DeNicola R. et al., "Model-checking Mobile Stochastic logic", *Theoretical computer science* 382(1), 42-70, 2007.
11. Dixon et al., "Towards temporal verification of emerging behavior in swarm-robotic systems" TOROS 2011, LNCS Vol 6856, pp336-347.
12. DeNicola et al, "KLAIM: Kernel Language for Agent Interaction and Mobility", *IEEE transactions on software engineering*, Vol 24, No.5, 1998.
13. A.Martinoli et al,"Modeling Swarm-robotic systems: A case study in collaborative distributed manipulation", *International Journal of Robotics Research*, Vol 23, 2004, pp415-436.
14. Vasumathi K.Narayanan, "Formal verification of robotic and automations tasks" INTERACT-2010, 2010.
15. S.Jeyaraman et al. "Formal techniques for the modeling and validation of a co-operating UAV team that uses Dubins set for path planning" In ACC, Volume 7, pages 4690-4695, IEEE 2005.

16. M.Fisher et al,” On the formal specification and verification of multi-agent systems”, Int. Journal of co-operative information systems, 6(1):37-66,1997.
17. S.Konur et al,”Formal verification of probabilistic swarm behaviours”, In ANTS number 6231 in LNCS, pp572-573, Springer 2010.
18. K.Lerman et al, “A Review of probabilistic macroscopic models for swarm robotic systems” Vol 3342 of LNCS, pp143-152, Springer 2005.
19. A.F.T. Winfield et al., On formal specification of emergent behaviours in Swarm robotic systems”, Int Journal of advanced robotic systems, 2(4):pp363-370,2005.