

Formal Verification of a Distributed Algorithm for Task Execution

Amar Nath and Rajdeep Niyogi(✉)

Indian Institute of Technology Roorkee, Roorkee 247667, India
anath@cs.iitr.ac.in, rajdpfec@iitr.ac.in

Abstract. In a dynamic environment, the accomplishment of a mission composed of multi-robot tasks is quite a challenging problem. To execute collaborative tasks successfully, robots may need to be present at the location of the task. However, the required robots may not be present at the location of the task. A distributed algorithm has been proposed recently for such task execution. A model checker can analyze the behavior of the algorithm. In this paper, we formally model the algorithm using the PRISM model checker. We identify some interesting safety and liveness properties of the algorithm and perform extensive simulations of the algorithm. The results validate that the behavior of the algorithm is as expected.

Keywords: Task execution · Distributed algorithm · Formal verification · PRISM model checker

1 Introduction

A growing trend in the robotics community is to use a group of robots working in collaboration with each other (called multi-robot system (MRS) [1]) to accomplish tasks instead of using a single robot. The use of MRS provides better scalability, reliability, flexibility, and helps perform the tasks faster and in a cheaper way than single robot systems. MRS can be beneficial in search and surveillance applications, especially in areas that are difficult or impossible for humans to access. Another benefit of MRS is that they have better spatial distribution.

Execution of certain tasks (e.g., moving a heavy object) requires several robots' collective effort since no individual robot possesses all the skills required to achieve the task. Such tasks are to be executed in a dynamic environment, where the location and state of the robots change with time, the robots may enter and exit the environment, the location of static objects change with time as well, and no robot has global knowledge. When a task has to be executed in such an environment, the robots required to perform the task may not be available at the location of the task. So the robots should communicate and coordinate with each other in order to gather the required number of robots

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12253, pp. 120–131, 2020.
https://doi.org/10.1007/978-3-030-58814-4_9

at the location of the task. For this, a distributed algorithm has recently been suggested in [2], and it has been implemented on a multi-robot simulator [14].

The algorithm suggested in [2] forms a coalition on task detection. The process of coalition formation starts on task detection (at runtime), but coalition cannot be formed in every run of the algorithm. The state and location of the robots play an important role in a coalition formation. A multi-robot system deployed in a real-world scenario should behave as expected. Incorrect behavior of the MRS would harm safety aspects and might also cause economic losses. For an MRS supposed to operate in a dynamic environment, it becomes imperative that any probable incorrect behavior is detected before the system is deployed on a real robotics system. Formal verification [3] is a process of checking whether a design satisfies the requirements.

Model-checking is a popular formal verification technique for automatically verifying the correctness properties of finite-state systems. Formal verification technology has been mostly successful and widely used in industry and space applications. Quantitative verification [4,5], allows reasoning in terms of quantitative properties, for instance, can a robot reach the desired location within, say, ten units of time, or what is the probability that a coalition would be formed. PRobabilistIc Symbolic Model checker (PRISM) [4,5], a probabilistic model checker, is a tool for formal modeling and analysis of systems that exhibit random or probabilistic behavior. It is used to analyze systems from many different application domains like communication, randomized distributed algorithms, and many others [4,5].

The rest of the paper is structured as follows. The related work is given in Sect. 2. The distributed algorithm [2] is described in Sect. 3. The formal modeling and verification of the algorithm are given in Sect. 4. The verification results are given in Sect. 5. Conclusions are made in Sect. 6.

2 Related Work

There are many model checking tools (e.g., SPIN [6], UPPAAL [7], MCMAS [8], and PRISM [4]) to formally verify different properties of a system. The work [9] considers the application of formal verification for checking whether the robotic application tasks are schedulable with respect to given hardware. The work [10] describes tools that automatically convert autonomy software into formal models that are then verified using model checking. This approach has been applied for task description language for mobile robotic systems. The work [11] considers some languages suitable for the specification and analysis of MRS. A comparison of these languages is made with respect to their ability to express some features of MRS like foraging and flocking.

PRISM based model checking has been done in [12] for verifying a team formation protocol [13] involving a network of interconnected agents with certain resources. We are not aware of any works that use the PRISM model checker for verifying the distributed algorithm for task execution in a multi-robot setting proposed in [2].

3 A Distributed Algorithm for Task Execution in a Dynamic Environment [2]

We consider the formal verification of an existing distributed algorithm for task execution in a dynamic environment [2]. A brief description of the algorithm [2] is given below.

In a dynamic environment $\mathcal{E}$, let a robot i detects a task $\tau = \langle \nu, l, t, k, \Psi \rangle$ at location l, and the robot i has sub-set of skills, required to execute the task, i.e., $\psi_i \supseteq \tau.\Psi$. The skills set for robots and tasks are represented as ψ and Ψ respectively. The elements of the task τ, i.e., ν, l, t, k, Ψ are defined as; ν is the name of a task (e.g., move (carry) box B to location l', lift desk D), $l \in L$ is the location where the task is arrived/detected, t is the time at which the task arrived, k is the number of robots required to execute the task, and Ψ is the skill/capability vector required to execute the task.

The task τ may not be executed by a robot i, and hence a team of robots needs to be formed before starting the execution of a task. However, all the robots may not be present at that location. No robot knows the states, locations, and skills of other robots (i.e., the absence of global knowledge). Thus, the robots should communicate among themselves in order to acquire relevant information for task execution without the intervention of any central authority. This necessitates the design of a distributed approach for task execution in such dynamic environments. In order to form a team for the execution of task τ, the robot i that is called an initiator, start the coalition/team formation process—the rest of the robots besides initiators called non-initiator.

An initiator i broadcasts a Request message and waits for some time, say Δ. It is assumed that a broadcast message would be delivered only to the robots present in the environment at that time. A non-initiator j, who has the necessary skills, will send either a Willing message or an Engaged message if its state is Idle or Promise respectively. Otherwise, it will ignore the Request message. The initiator increases its counter c when it receives a Willing message. After Δ time has elapsed, i checks if there are enough robots available to form a team ($c \geq k-1$). If yes, i select the members for the team and sends *Confirm* messages to them, *Not-Required* message to $(c-(k-1))$ non-initiators, if any. If no, i sends a *Not-Required* message to all c non-initiators who expressed their willingness to help. Depending on its queue status, i will change its state from Ready to Idle or Promise. The selected non-initiators gather at the task location and start the execution of a task. When robots change their state to Busy from either Ready or Promise, it is assumed that robots jointly start the execution of the task.

A non-initiator robot works as follows. The computations are done based on the current state that may be Idle, Promise, Busy, and Ready. Within a state, the type of message is checked, and appropriate actions are taken. For example, when state is Idle, if a Request message is received, it becomes Promise, the identifier of the sender is enqueued, and $flag$ is set to true; all these actions are done atomically (denoted by $\langle \ldots \rangle$). Now, the robot sends a Willing message to the sender (initiator), and $flag$ is set to false. A robot j maintains a local queue Q, which keeps the senders' identifiers based on the incoming Request messages.

Q is used to avoid starvation since more than one initiator may send Request messages at the same instant of time. The boolean variables $flag$ and $flag'$ are used to control the sending of Willing and Engaged messages respectively.

In order to ensure that the algorithm [2] satisfies some properties, formal verification is required. In this paper, the formal verification of the algorithm is done using PRISM.

4 Formal Verification of the Algorithm [2] with PRISM Model Checker

Model-checking is a formal, automatic technique to verify whether or not system design models satisfy given requirements [3]. In other words, it is a problem of establishing whether or not a given formula ϕ is true in a given model $\mathcal{M}$. Model-checking involves three phases:

- Modeling: to convert a design into a formalism so that mathematical computation and logical deduction can be performed.
- Specification: to specify the properties that the model should satisfy.
- Verification: to verify whether the model satisfies the specification [15].

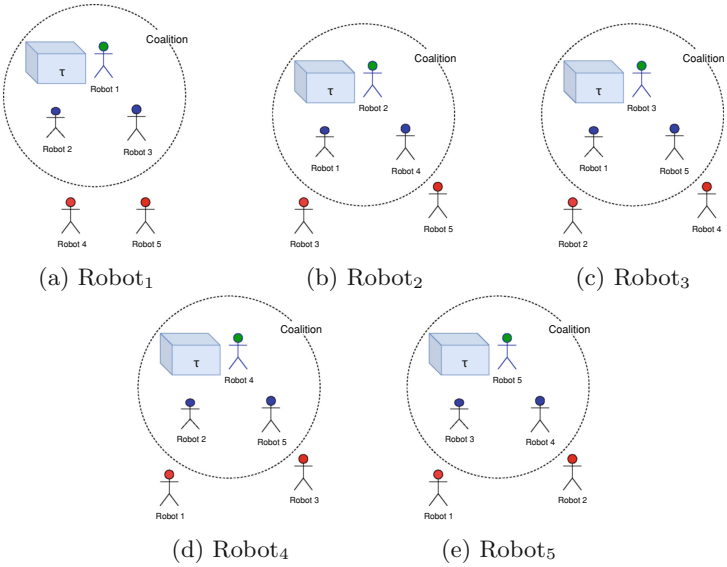
A qualitative model checker gives "yes" or "no" as output. A quantitative (or probabilistic) model checker, on the other hand, gives the result "yes" or "no," or some probability as an output. In probabilistic model checking, the models encode the probability of making a transition between states instead of simply the existence of such a transition.

4.1 Modeling of the Distributed Algorithm in PRISM

The most important activity of the distributed algorithm for task execution in a dynamic environment is coalition formation. In coalition formation, a robot starts the coalition on task detection, if it can not execute the task alone. The task cannot be executed until coalition formation. We need to ascertain whether the algorithm described in Sect. 3 satisfies some properties.

4.2 PRISM Model

We model the distributed algorithm [2] using PRISM model checker [4]. In order to demonstrate the procedure clearly, we assume that at some point in time t, 5 robots ($\text{Robot}_1 \ldots \text{Robot}_5$) are present in the environment. In the environment several tasks ($\tau_1 \ldots \tau_m$) may arrive, and the task may be detected by any robot. The modeling captures the scenarios (given in Fig. 1), where a task τ can be detected by some robot. The model takes care of possibilities of task detection by different robots at different time. In Fig. 1-a, Robot_1 is the initiator robot and it forms the coalition with Robot_2 and Robot_3. Similarly, in Figs. 1-b, 1-c, 1-d and 1-e, Robot_2, Robot_3, Robot_4 and Robot_5 are acting as initiators respectively.



(a) Robot$_1$ (b) Robot$_2$ (c) Robot$_3$

(d) Robot$_4$ (e) Robot$_5$

Fig. 1. Different scenarios of robots detecting tasks and forming their corresponding coalitions

Data Structure: The data structure in the model consists of global variables and local variables. pInt is a global variable to denote the probability with which a robot detects a task. The global variable pOt denotes the probability with which an initiator stops its coalition formation process because it does not receive a sufficient number of willing messages within a predefined period. The representation in PRISM is given below.

```
mdp
const double pInt = .2; // probabaility; a robot detects a task
const double pOt = .1; //probabaility of timeout without team formation
const N = 5; //Total no of robots =5
```

The local variables of each robot are used to control the local aspects of the robots. The variable "state" is an integer which is used to capture the states of a robot during the task execution; 0,1,2,3 denotes Idle, Ready, Promise, and Busy respectively. The variable q1 is for placing the request received by a robot. The value of variable k, represents the total number of robots required for task execution. The variable c is used a count the number of Willing messages received by an initiator.

```
//Robot1 states, i.e., {0=Idle, 1=Ready, 2=Promise, 3=Busy}
state1:[0..3]; //queue
q1:[0..5];//queue
c1:[0..4] init 0; //Variable to records the no. of willing messages
k1:[0..3] init 3;//The required no. of robots required to execute a task
```

The sending of the messages is controlled by Boolean variables as discussed below. All the Boolean variables are initialized to false. The Boolean variable Send_request_12 denotes that it is a Request message which is sent from Robot_1 to Robot_2. Similarly, the other variables are constructed. The variable Send_confirm_12 is used to control the sending of Confirm message, from Robot_1 to Robot_2. Similarly, other message variables are used for sending different messages, i.e., Willing, Not-required and Engaged. The variables wr21, wr31, wr41, and wr51 are used to keep track of robots, who has sent the Willing to the Robot_1.

```
//The messages
Send_request_12 : bool init false; Send_confirm_12 : bool init false;
Send_request_13 : bool init false; Send_confirm_13 : bool init false;
Send_request_14 : bool init false; Send_confirm_14 : bool init false;
Send_request_15 : bool init false; Send_confirm_15 : bool init false;

Send_willing_12 : bool init false; Send_eng_12 : bool init false;
Send_willing_13 : bool init false; Send_eng_13 : bool init false;
Send_willing_14 : bool init false; Send_eng_14 : bool init false;
Send_willing_15 : bool init false; Send_eng_15 : bool init false;

Send_notreq_12 : bool init false; wr21 : bool init false;
Send_notreq_13 : bool init false; wr31 : bool init false;
Send_notreq_14 : bool init false; wr41 : bool init false;
Send_notreq_15 : bool init false; wr51 : bool init false;
```

Key Procedure: We constructed PRISM modules for every robot, present in the environment, and behaving according to the algorithm described in Sect. 3. A PRISM model for task detection by a Robot_1 is given below. The Robot_1 detects a task with probability with, say, 0.2 (pInt). As soon as it detects a task, it changes its state from Idle to Ready and acts as an initiator. In Ready state, the initiator, start the coalition formation process by broadcasting the Request messages to all other robots.

```
//task detection
[](state1 = 0)&(task1_detected = true) -> pInt:(state1' = 1) &
    (Initiator1'=true)+ (1-pInt):(state1' = 0);
//broadcasting the request
[](state1 = 1) & (Initiator1=true) -> (state1' = 1)
    &(Send_request_12'=true) & (Send_request_13'=true) &
    (Send_request_14' =true) & (Send_request_15' =true);
```

After sending the request to all other robots, the initiator receives either Willing or Engaged messages. When an initiator receives the Willing message from a non-initiator, it increments its counter c. If the sufficient number of

Willing messages are received within a specific period of time, then a coalition is formed successfully, as shown below.

```
//recept of willing message, but condition is not satisfied
[](state1= 1)&(Initiator1=true)& (Send_willing_21=true)& (c1 < k1-1)
    &(wr21=false) -> (state1' = 1)&(c1' = c1 +1)&(wr21'=true);
[](state1= 1)&(Initiator1=true)& (Send_willing_31=true)& (c1 < k1-1)
    &(wr31=false) -> (state1' = 1)&(c1' = c1 +1)&(wr31'=true);
[](state1= 1)&(Initiator1=true)& (Send_willing_41=true)& (c1 < k1-1)
    &(wr41=false) -> (state1' = 1)&(c1' = c1 +1)&(wr41'=true);
[](state1= 1)&(Initiator1=true)& (Send_willing_51=true)& (c1 < k1-1)
    &(wr51=false) -> (state1' = 1)&(c1' = c1 +1)&(wr51'=true);

//receipt of engaged message in any condition, it is just ignored
[](state1 = 1)&(Initiator1=true) & (Send_eng_21=true) -> (state1' = 1);
[](state1 = 1)&(Initiator1=true) & (Send_eng_31=true) -> (state1' = 1);
[](state1 = 1)&(Initiator1=true) & (Send_eng_41=true) -> (state1' = 1);
[](state1 = 1)&(Initiator1=true) & (Send_eng_51=true) -> (state1' = 1);
```

If a sufficient number of willing messages are received, the initiator sends Confirm or Not-required messages depending on the current situation. The selection of the desired robot for a coalition formation is formed at run time. The PRISM code for coalition formation is given below. The sending of either Confirm or Not-required message is also decided at runtime. Similarly, PRISM code for other robots has been constructed.

5 Verification Results

The efficiency of the distributed algorithm designed for task execution in a dynamic environment (given in Sect. 3) depends on the successful coalition formation process. The coalition formation is done via asynchronous message passing. The efficient coordination via communication is the key to success for the algorithm. In a dynamic environment, if coalitions for tasks are formed quickly, tasks may be executed successfully.

The initial conditions produced by the PRISM model checker is shown in Fig. 2. The initial PRISM snapshot illustrates all the labels present in the model. In addition, the built-in labels “init” and “deadlock” are also included. Selecting a label from the list highlights all states in the current path which satisfy it. It clear from Fig. 2, that the model is properly initialized, and no deadlock is present in the model at this moment in time (Fig. 3). The model is ready for verification.

5.1 Modeling Using PRISM

In order to analyze a PRISM model, it is necessary to specify one or more properties. PRISM’s property specification is based on Computational Tree Logic (CTL) [3].

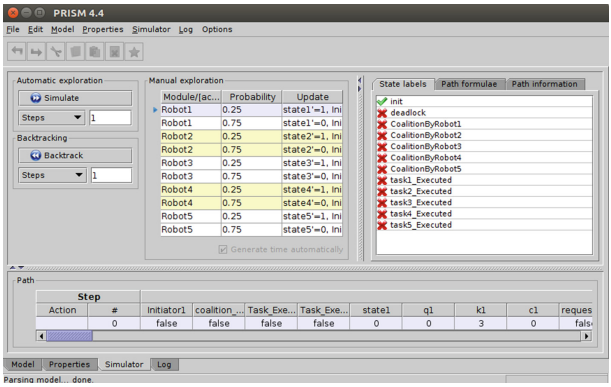


Fig. 2. Initial conditions of the model in PRISM model checker

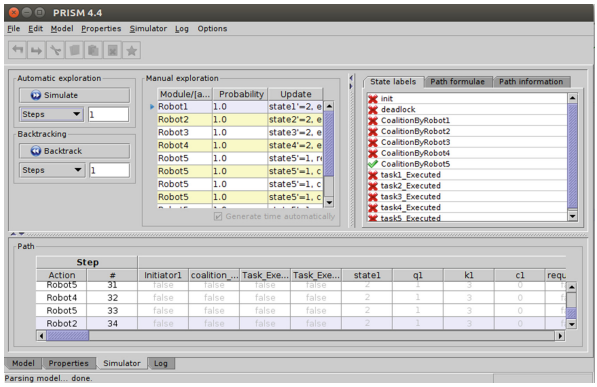


Fig. 3. A coalition is formed by Robot_5

5.1.1 Computation Tree Logic (CTL)

CTL has a two-stage syntax where formula in CTL are classified into *state* and *path* formulas. CTL *state formula* over the set AP of an atomic proposition are formed according to the following grammar [3].

$$\Phi ::= true \mid p \in AP \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid E\varphi \mid A\varphi$$

where, AP is the set of atomic propositions, E and A are the path quantifiers– "for some path" and "for all path" respectively and φ is a *path formula*. CTL *path formula* are formed according to the following grammar:

$$\varphi ::= X\Phi \mid F\Phi_1 \mid G\Phi_1 \mid \Phi_1 \bigcup \Phi_2;$$

where Φ, Φ_1 and Φ_2 are the state formula and X, F, G, U are the temporal operators to denote "next", "some time in the future", "always in the future",

and "until" operator respectively. $F\Phi$ and $G\Phi$ are dual operators where, $F\Phi = true\ \mathrm{U}\ \Phi$ and $G\Phi = \neg F \neg \Phi$.

Intuitively, *state formula* express a property of a state, while a *path formula* express a property of a computation path where a computation path is an infinite sequence of states. For the semantics of CTL we refer to [3].

5.1.2 Probabilistic Computation Tree Logic (PCTL)

Probabilistic Computation Tree Logic (PCTL) is an extension of CTL with a probability operator *(P)* [3]. PCTL is a useful logic for stating soft deadline properties. The *state formula* of PCTL over the set AP of atomic propositions are defined by the following grammar.

$$\Phi ::= true \mid p \in AP \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid P_{\bowtie k}\ (\varphi)$$

where, φ is a *path formula* and $\bowtie \in \{<, \leq, >, \geq\}$ and $k \subseteq [0, 1]$ is an interval with rational bounds. PCTL *path formulae* are formed according to the following grammar:

$$\varphi ::= X\Phi \mid \Phi_1 \bigcup \Phi_2 \mid \Phi_1 \bigcup^{\leq n} \Phi_2$$

Where Φ, Φ_1, and Φ_2 are state formula and $\bigcup^{\leq n}$ is a "bounded until" operator where $n \in \mathbb{N}$ is a positive integer number reflecting the maximum number of transitions needed to reach a certain state. For the semantics of PCTL we refer to [3].

Some safety and liveness properties for the algorithm, specified in PCTL are given below.

- **Safety property:** Safety property has the form:
  $\mathrm{P}_{\bowtie k}$ G $(\Phi_1 \rightarrow \Phi_2)$ which is equivalent to $\mathrm{P}_{\bowtie k}$ G $((\neg\Phi_1) \vee \Phi_2)$
  We identified the following safety property for the distributed algorithm designed for task execution in a dynamic environment.
  1. Robot$_1$ sends Confirm messages in Ready state, after receiving sufficient number of Willing messages. This property can be expressed in PCTL as:
     S_1 : $P_{>0}$ `G (!((state1=1)&((Send_confirm_12=true)|(Send_confirm_13=true)|(Send_confirm_14=true)|(Send_confirm_15=true)))|(c1>=(k1-1)))`
  2. If Robot$_1$ is in Promise state, and it receives a Confirm message, then its next state will be Busy.
     S_2 : $P_{>0}$ `G (!((state1 = 2)&((Send_confirm_21=true)|(Send_confirm_31=true)|(Send_confirm_41=true)|(Send_confirm_51=true)))|` ($P_{>0}$ `X(state1=3))`
- **Liveness property:**
  Liveness property has the form:
  $\mathrm{P}_{\bowtie k}$ G $((\Phi_1 \rightarrow P_{\bowtie k} F\Phi_2)$ which is equivalent to $\mathrm{P}_{\bowtie k}$ G $((\neg\Phi_1) \vee P_{\bowtie k} F\Phi_2))$

1. If Robot_1 broadcasts a request message then eventually it must receive either Willing or Engaged messages.

   L_1 : $P_{>0}$ G (!((Send_request_12=true)|(Send_request_13=true)|(Send_request_14=true)|(Send_request_15=true))|$P_{>0}$ F(((Send_willing_21=true)|(Send_willing_31=true)|(Send_willing_41=true)|(Send_willing_51=true))|((Send_eng_21=true)|(Send_eng_31=true)|(Send_eng_41=true)|(Send_eng_51=true))))

2. If Robot_1 detects a task, eventually it forms the coalition successfully with probability greater than zero. The coalition may not form every time as at some time, other robots might be busy doing other activities, and hence, condition $(c \geq (k-1))$ might not be satisfied.

   L_2 : $P_{>0}$[true U (coalition_Formation1 = true)]

3. Every task in the environment is detected by a robot and eventually it is executed by a team of robots with some probability greater than 0. The task τ_1 is detected by Robot_1 and eventually Robot_1 goes to Busy state to execute the task. After finishing the task, the label "Task_ExecutionFinish1=true" becomes true.

   $P_{>0}$[true U "task1_Executed"] can be rewritten as:

   L_3 : $P_{>0}$[true U ((Initiator1=true)& (Task_ExecutionFinish1=true))]

The verification of a system that is complex and has a large number of variables is quite challenging. Due to the large state space exploration, the PRISM model checker demands a huge amount of memory to store the explored states. In the case of low memory, the discrete-event simulator built into PRISM can be used to generate approximate results for PRISM properties, a technique called statistical model checking. To verify some of the identified properties (e.g., S_1, L_1 and L_2), we have used this technique for verification. Essentially, statistical model checking is achieved by sampling: generating a large number of random paths through the model, evaluating the result of the given properties on each run, and using this information to generate an approximately correct result. This approach is particularly useful on very large models when normal model checking is infeasible. This is because the discrete-event simulation is performed using the PRISM language model description, without explicitly constructing the corresponding probabilistic model.

PRISM supports four different methods for performing statistical model checking: CI (Confidence Interval), ACI (Asymptotic Confidence Interval)-for large sample size, APMC (Approximate Probabilistic Model Checking), SPRT (Sequential Probability Ratio Test).

We use ACI sampling methodology for verification; the ACI method uses the Normal distribution. This method is appropriate when the number of samples is significant (because we can get a reliable estimation of the variance from the samples) but maybe less accurate for small samples (Figs. 4 and 5).

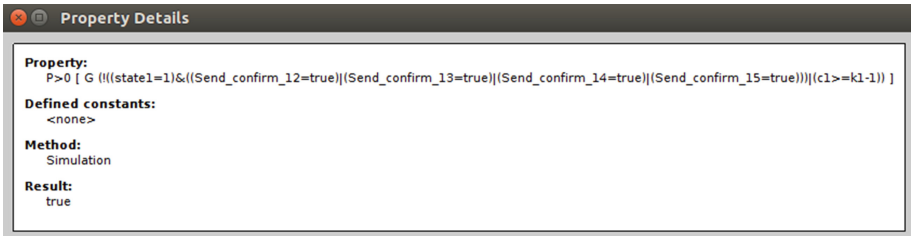
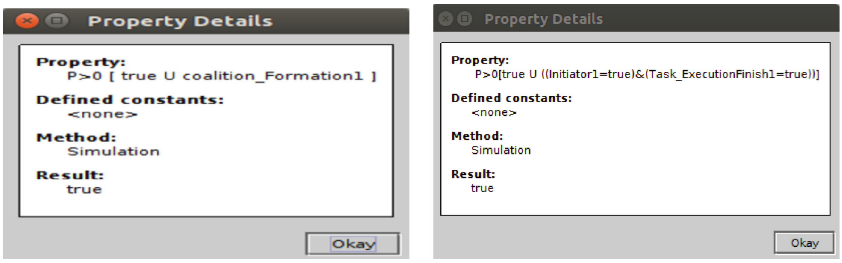


Fig. 4. Result for the safety property S_1



(a) Result for L_2 (b) Result for L_3

Fig. 5. Results for the liveness properties

6 Conclusion

In this paper, we considered the problem of verifying a distributed algorithm designed for task execution in a dynamic environment. The correctness of the algorithm is guaranteed by formally verifying the algorithm with probabilistic model checking. We identified some important safety and liveness properties of the algorithm. We constructed the algorithm's model in the PRISM model checker and verified the algorithm's properties. Extensive experiments were performed with a varying number of agents. The results are quite encouraging, and it confirms the expected execution of the algorithm.

Acknowledgements. The authors thank the anonymous reviewers of ICCSA 2020 for their valuable suggestions. The second author was in part supported by a research grant from Google.

References

1. Yan, Z., Jouandeau, N., Cherif, A.A.: A survey and analysis of multi-robot coordination. Int. J. Adv. Rob. Syst. **10**(12), 399 (2013)
2. Nath, A., Arun, A.R., Niyogi, R.: An approach for task execution in dynamic multirobot environment. In: Mitrovic, T., Xue, B., Li, X. (eds.) AI 2018. LNCS (LNAI), vol. 11320, pp. 71–76. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03991-2_7

3. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)
4. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: probabilistic symbolic model checker. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) TOOLS 2002. LNCS, vol. 2324, pp. 200–204. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46029-2_13
5. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
6. Holzmann, G.: Spin Model Checker, the: Primer and Reference Manual. Addison-Wesley Professional, Boston (2003)
7. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**(1–2), 134–152 (1997)
8. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: a model checker for the verification of multi-agent systems. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 682–688. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_55
9. Foughali, M., Berthomieu, B., Dal Zilio, S., Hladik, P.-E., Ingrand, F., Mallet, A.: Formal verification of complex robotic systems on resource-constrained platforms. In: 2018 IEEE/ACM 6th International FME Workshop on Formal Methods in Software Engineering (FormaliSE), pp. 2–9. IEEE (2018)
10. Simmons, R., Pecheur, C., Srinivasan, G.: Towards automatic verification of autonomous systems. In: Proceedings 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No. 00CH37113), vol. 2, pp. 1410–1415. IEEE (2000)
11. De Nicola, R., Di Stefano, L., Inverso, O.: Towards formal models and languages for verifiable multi-robot systems, arXiv preprint arXiv:1804.08091 (2018)
12. Chen, T., Kwiatkowska, M., Parker, D., Simaitis, A.: Verifying team formation protocols with probabilistic model checking. In: Leite, J., Torroni, P., Ågotnes, T., Boella, G., van der Torre, L. (eds.) CLIMA 2011. LNCS (LNAI), vol. 6814, pp. 190–207. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22359-4_14
13. Gaston, M.E., desJardins, M.: Agent-organized networks for dynamic team formation. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 230–237. ACM (2005)
14. Pinciroli, C., et al.: Argos: a modular, multi-engine simulator for heterogeneous swarm robotics. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5027–5034. IEEE (2011)
15. Sultan, K.I.: Modeling and verifying probabilistic social commitments in multi-agent systems. Ph.D. thesis, Concordia University (2015)