Proceedings of the Estonian Academy of Sciences,
2011, **60**, 1, 48–54
doi: 10.3176/proc.2011.1.05
Available online at www.eap.ee/proceedings

COMPUTER SCIENCE

Model checking of emergent behaviour properties of robot swarms

Silver Juurik* and Jüri Vain

Department of Computer Science, Tallinn University of Technology, Akadeemia tee 21, 12618 Tallinn, Estonia

Received 10 December 2009, revised 28 January 2010, accepted 29 January 2010

Abstract. This paper presents a case study on scalability of explicit state model checking. Three state space reduction methods – state vector compression, bit state hashing, and symmetry reduction – were applied on an exercise with the objective of verifying a distributed coordination algorithm for robot swarms. Based on the analysis results, the feasibility of using explicit state model checking to prove properties of large multi-agent systems is questioned and the limitations faced in verifying a dynamic cleaning algorithm are outlined.

Key words: model checking, emerging behaviour properties, robot swarm, distributed coordination algorithm.

1. INTRODUCTION

Bonabeau et al. described in [4] the emergent behaviour of social insects as “a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components”. Swarm robotics is an approach to the coordination of multi-agent systems that consist of large numbers of physical robots. It is supposed that a desired collective behaviour emerges from the interactions between the robots and interactions of robots with the environment. In addition to simple social structure, the robot swarm can also exhibit intellectual capabilities like collective learning, rule-based reasoning, collective decision making, etc. These capabilities create a huge state space and non-stationary behaviour.

Verification of emergent behaviour properties is imperative in mission critical applications of robot swarms like coordinated actions of space pods [17], early discovery of radioactive leaks in nuclear reactors, intruder detection and capture, etc. The challenge in designing swarm coordination algorithms for such missions is in evaluating the success of the mission, provided that the environment constraints and the swarm configuration are given. The formal methods applicable in multi-agent systems with a low degree of parallelism have clear scalability limits for swarms. Therefore, complete state space exploration has limited uses and current formal approaches have concentrated on simulation-based partial analysis methods and specification techniques.

A natural solution for overcoming the complexity barrier of swarm analysis is to use the compositional verification approach. Let us call the part of the real world to be represented for analysis a “swarm world”. The swarm world consists of swarm robots and the objects in the environment that participate in robot–environment interactions. Ideally, when decomposing a swarm world into smaller parts, the same emergent behaviour has to be exhibited at a smaller scale down to some smallest collaborating part (e.g. a pair of collaborating robots). Homogeneity of swarm behaviour (modulo the smallest collaborating swarm world unit) over all its legal partitionings would then guarantee that proof of a part of the swarm world with inductive argument would allow constructing the proof for the whole swarm.

Unfortunately, the homogeneity assumption is not very common in the context of practical swarm analysis, and finding an abstract invariant interface for swarm world parts in slightly more general cases seems to be a challenging task that needs the existence of a generalizing domain theory relevant to the problem.

In this paper we study explicit state model checking as a potential method for emergent behaviour analysis of swarms. Three state space reduction methods – state vector compression, bit state hashing, and symmetry reduction – were applied on an exercise with

* Corresponding author, silverjuurik@gmail.com

the objective of verifying a distributed coordination algorithm for robot swarms. Due to the lack of an abstract swarm theory that would allow defining rules for swarm composition and structuring for swarm world model partitionings and abstract interfaces, no partitionings of the swarm will be done in the scope of this paper. Our main hypothesis is that as swarm models are highly symmetric because of the symmetric behaviour of agents, symmetry reduction combined with state vector compression and bit state hashing would allow considerable upscaling of models applicable for model checking.

To establish the applicability limits of model checking for analysis of swarm properties, a distributed dynamic cleaning algorithm is studied. Each agent participating in the cleaning is described by the same set of model templates. To illustrate the time-scalability of model checking the parameter “time horizon”, which represents the time during which the observed area must be serviced, has been introduced in the model.

Verification experiments are made for a model described in Subsection 4.2 using the above-mentioned state space reduction methods. The rest of the paper is arranged as follows: Section 2 gives a brief overview of the state of the art of formal methods used for formal analysis of robot swarm emergent behaviour. Section 3 introduces explicit state model checking and the state space reduction techniques used later in the experiments. Section 4 presents a case study and performance analysis of the model checker and state space reduction techniques. Shortcomings and advantages of the explicit state model checking showing up in the experiments are summarized in Section 5.

2. MULTI-AGENT SYSTEMS OF EMERGENT BEHAVIOUR VERIFICATION: STATE OF THE ART

As of 2004, only a few formal approaches existed for analysing the emergent behaviour of swarms [17]. The following approaches were evaluated:

1. Communicating Sequential Processes (CSP) [10]
2. Weighted Synchronous Calculus of Communicating Systems (WSCCS) [18]
3. Unity Logic [5]
4. X-Machines [9].

According to the evaluation, CSP is suitable for specifying the process protocols between the robots in the swarm. Reasoning about a CSP specification can be done to determine race conditions or CSP description can be converted into a model checking language to be run in a model checker.

WSCCS is a process algebra that takes into account the priorities and probabilities of actions performed by robots. Furthermore, it provides syntax and a set of rules for predicting and specifying choices and behaviour as well as a congruence and syntax for determining if two automata modelling the behaviour of different robots are equivalent. Thus, WSCCS can be used to reason about and even predict the behaviour of one or more robots. This affords WSCCS the potential for specifying emergent behaviour in the swarm as used in the ANTS project [7]. However, it lacks the ability to track the goals and model the mission of the robots.

Unity Logic provides a logical syntax equivalent to simple Propositional Logic for reasoning about predicates and the states they imply as well as for defining specific mathematical, statistical, and other simple calculations to be performed. However, it does not appear to be rich enough to allow ease of specification and validation of more abstract concepts such as mission goals.

X-Machines provide a highly executable environment for specifying the behaviour of a single robot. This approach allows for a memory to be kept and for transitions between states to be seen as functions involving inputs and outputs. This enables to track the actions of the robot as well as write any aspect of the goals and the model to the memory. Therefore X-Machines are highly effective for tracking and affecting changes in the goals and robot world model. However, X-Machines do not provide any robust means for reasoning about or predicting behaviours of more than one robot beyond standard propositional logic. This will make specifying emergent behaviour difficult.

A relatively successful attempt of formal modelling is the work of Martinoli et al. [15], which uses a stochastic approach in which an ensemble of probabilistic finite state machines describe the overall structure of the swarm in terms of its microscopic (individual robot) parameters. However, this work concentrates on modelling rather than on formal proofs.

3. EXPLICIT STATE MODEL CHECKING

3.1. Problem and methods of explicit state model checking

Model checking uses an automatic procedure that searches all possible states of a formal model. In each visited state it is checked if the properties of the state conform to some given specification. Violations of the specification are reported to the user [2]. Since systems may contain an infinite number of states, bounds are placed on the parameters of the system to make this number finite and guarantee algorithmic termination. The main obstacle in practical application of model checking is the problem of state space explosion, where the addition of small components to a system results in a combinatorial growth in the number of states to be explored. Thus, verification of large systems becomes intractable. Various methods aim to alleviate this problem, such as partial order reduction [12], abstraction [6], and symmetry reduction [2]. These techniques have been implemented in model checking tools like Spin [12], NuSmV [14], Uppaal [3], etc.

3.2. Techniques of state space reduction

3.2.1. Symmetry reduction

Symmetry reduction works by constructing equivalence classes from the model with components of the model that have equivalent behaviour belonging to the same equivalence class. During the verification only a single representative from each equivalence class is examined. In case the model consists of two identical parts, the reduction in state space can be up to 50%, as will be demonstrated in Section 4.2.

3.2.2. Bit state hashing

Bit state hashing is an abstraction technique that works by computing a hash value of a state using some hash function. The method was originally conceived by Morris [16]. The goal of the method is to construct a bit field that can be used to identify whether the current state has been visited already. However, the use of bit state hashing will reduce the accuracy of the outcome because a state could be mistakenly reported as visited due to a hash collision and is therefore not stored in the hash array. Since some of the states might not be stored, a state that would break the verification conditions may go unnoticed. However, all reported errors that are found are real error conditions.

Because the state vector is tens or hundreds of bytes long, the reduction in memory consumption can be up to 98% when bit state hashing is used [11].

3.2.3. State vector compression and minimized deterministic finite automaton

The third reduction technique used in the experiments presented in this paper is minimized deterministic finite automaton (DFA). In case minimized DFA is used, the model checker SPIN constructs a new, so-called observer or detection automaton during the verification. The constructed automaton is then able to determine if a state has been visited before. As reported by Holzmann and Puri in [13], using DFA in verification can reduce the memory requirements by as much as an order of magnitude. However, the amount of the used memory is reduced at the expense of verification time.

4. CASE STUDY

4.1. Problem description

The dynamic cleaning problem defined in [1] can be described as follows. Let there be a rectangular area A with height h and width l (the cleaning area). Let there be $n = h \cdot l$ RFID tags a on that area, distributed evenly in a grid layout, so that no two tags are closer than one distance unit. The readability range of the RFID tags must be at least $\frac{\sqrt{2}}{2}$ distance units, which ensures that there are no spots within A that are out of the readability range of an RFID tag. A cleaning zone is defined as the zone within area A that is in the range for an RFID tag. Each tag $r \in R$ contains at least three data fields: the timestamp of the latest cleaning, the ID of the agent occupying the associated cleaning zone, and the ID of the agent that has reserved the zone.

The deterioration level at every point within A is considered to be zero initially and it is increased dynamically by the environment. The speed of increasing the deterioration level may be different in various points within the area A; however, the speed in a cleaning zone must remain constant throughout the experiments. For the sake of simplicity, in this paper it is assumed that the deterioration speed is equal throughout the cleaning area.

Let there be a number of agents R that are assigned to clean the area. The agents visit cleaning zones with possibly varying frequency and clean the zones if necessary. Each agent $r \in R$ has a limited range ("visibility range") L in which it can detect RFID tags. In principle, L represents the maximum distance between the agent and a tag that is detectable by the agent. For example, if $L = 1$ then an agent can detect the tag the agent currently occupies and four closest tags in all orthogonal directions. In the following the value $L = \sqrt{2}$ is used, which means nine tags will remain in an agent's visible range – all tags that are detected at $L = 1$ and additionally the four nearest tags in diagonal directions.

Cleaning a zone is defined as reducing the deterioration level in that zone to zero. The area A is considered to be clean iff the deterioration level at all zones within A is not higher than a predefined threshold TR:

$$\forall a \in A : deterioration(a) \leq TR.$$

The goal of the problem is to prove that an algorithm applied to all agents in the swarm will be efficient enough to keep the area A clean during a given time frame.

4.2. The coordination algorithm

The coordination algorithm of the cleaning swarm behaves as follows. The agent examines all cleaning zones within its visible range. In each case it checks if

1. the zone is not occupied,
2. other agents have not decided to move to that zone,
3. the deterioration level of the zone is higher than the highest value currently known to the agent.

If these conditions have been fulfilled, the deterioration level of the zone is set as the current highest value known to the agent. Next, the agent cancels the reservation of the previously reserved zone (if there is any) and the zone with the current highest deterioration level is reserved instead. Reserving the zone is necessary to prevent race conditions between the agents that are within the visible range of the locally most deteriorated zone.

When all zones within the visible range have been examined, there are two actions that the agent can perform. In case the zone selected by the agent is the zone that the agent currently occupies, the agent will start cleaning it. Otherwise, the agent will move to the selected cleaning zone. In the experiments conducted in 4.3, the time for moving from one zone to an adjacent zone and cleaning a zone were chosen as constant values 1 and 5 time units, respectively.

The algorithm has been implemented in Promela, a language used by the model checker Spin. Since Promela does not include the concept of time, it is necessary to emulate the time flow. A simple mutual exclusion protocol is used instead of global clocks: each agent has a counter that is incremented after each action the agent performs; the agent with the highest counter value will be forced to remain idle – meaning that the agent is not allowed to perform any tasks and therefore its counter value cannot increase until the other agents' counters catch up. This ensures that the agents are progressing at an equal pace. To reduce the interleaving between the agents' actions, some of the statements in the Promela model are grouped together using Promela keywords *atomic* and *d_step*.

4.3. Analysis of performance and scalability

It was proved in [19] that the given algorithm is correct when two agents and a relatively short time horizon are considered. In the current section we show that using explicit model checking is not efficient enough to prove the correctness of the algorithm described in 4.2 within an arbitrary time interval. Similar limitation also concerns any other extension of the parameters (presenting scalability) of the algorithm. Therefore, to demonstrate the scalability of the model checking technique under consideration, in all experiments the time period in which the algorithm's properties must be guaranteed is varied. The time period is bounded by the time limits usually set to practical model checking. In this case, the upper bound is further limited by the exponential growth of the state space.

The model used in this case study represents two agents and 16 cleaning zones. The number of cleaning zones was selected based on the performance assumption that a single agent is able to keep clean an area of eight zones. According to the given assumptions the agents are able to clean each zone in the theoretical best case at least once within 47 time units: cleaning each zone takes 5 time units, there are 8 zones per agent, and an agent must move at least 7 times to another zone. In reality the time required to clean the area is longer due to the fact that the initial position of the agents influences the minimal total cleaning time. In our model setting we assume that both agents start from the same position. Therefore, the time period for the cleaning is varied in the scalability experiments from 80 to 90 time units. A further increase, as demonstrated, will lead to infeasibly long verification time and system resource usage. Throughout the rest of the chapter, the following terms are used: by symmetry reduction we mean the usage of TopSpin symmetry reduction algorithm; by bit state hashing we mean compiling the algorithm with the option -DBITSTATE, which applies bit state hashing to the input model; by state vector compression we mean compiling the model with the options -DMA=100, which uses minimized DFA, and -DCOLLAPSE, which applies state vector compression. For performance reasons the following parameters were added when compiling the model: -DSAFETY, which disables cycle detection; -DNOBOUNDCHECK, which disables array bound violations; -DMEMLIM=2800 to ensure that verification will not completely deplete available resources; -DNOFAIR, which disables weak fairness.

Figures 1 and 2 demonstrate the commands executed in verification runs. Additional information on compile time and runtime options is provided in [11] for Spin and in [8] for TopSpin.

4.3.1. Memory usage

Based on the test results, it appears that when bit state hashing is used, Spin allocates memory for hash table using the following formula:

$$M_{\text{total}} = 2^{H-3},$$

where M_{total} represents the total amount of allocated memory (measured in bits) and H represents the size of

```
> java -jar ../topspin/TopSPIN_2.2/TopSPIN_2.2.jar proof2.p
> gcc -o sympan sympan.c group.c -DSAFETY -DNOBOUNDCHECK
-DMEMLIM=2800 -DNOFAIR -DBITSTATE
> ./sympan -c0 -w25 -m10000
```

Fig. 1. Commands used to verify correctness using bit state hashing and symmetry reduction.

```
> spin -a -v proof2.p
> gcc -o pan pan.c -DSAFETY -DNOBOUNDCHECK -DMEMLIM=2800
-DNOFAIR -DCOLLAPSE -DMA=100
> ./pan -c0 -w25 -m10000
```

Fig. 2. Commands used to verify correctness using state vector compression.

Table 1. Memory usage in verifications

Hash table size	Symmetry reduction, MB	No symmetry reduction, MB
25	313	377
26	569	633
27	1081	1145
28	2105	2169

the hash table. The difference between the total amount of consumed memory and the amount of memory allocated for the hash table was less than 1 MB in all verifications where bit state hashing was used. Since the server that was used had only 4 GB of RAM available, some of which was used by the operating system components, the largest hash table that could have been used was of size 34, which is 2^{34-3} B or 2 GB of RAM.

In case of full state space search, the following formula is used to calculate the amount of memory allocated for hash table:

$$M_{\text{total}} = 2^{H+3}.$$

Table 1 shows the total memory consumption in the test runs where bit state hashing was not used. The memory usage of 57 MB for states remained constant throughout the test cases where symmetry reduction was enabled and 121 MB for tests without symmetry reduction. The maximum size of the hash table that could be used was 28, which means 2^{28+3} B or 2 GB would be allocated to the hash table. The amount of consumed memory increases exponentially in both presented cases, due to the memory allocation scheme presented above.

4.3.2. Time

Figure 3 shows that once the hash table reaches a certain level of saturation, the elapsed verification time will vary very little. It is also clear that the saturation level is reached sooner when symmetry reduction is used.

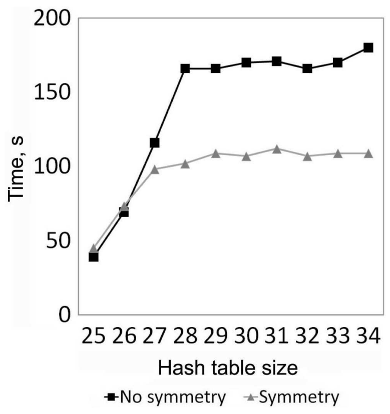

Fig. 3. Elapsed time for the duration of 80 units.

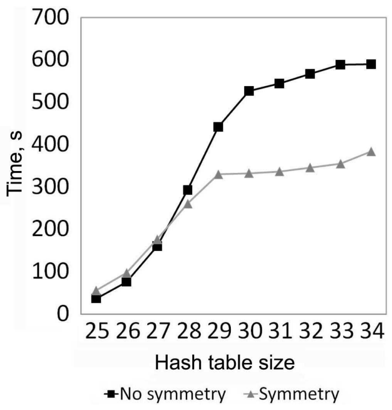

Fig. 4. Elapsed time for the duration of 90 units.

When the cleaning duration parameter was increased to 90, similar results were obtained (see Fig. 4). As the state space was larger, the stabilization point was reached at a later stage but once reached, the increase in the elapsed time almost halted.

4.3.3. State space

In all the cases where bit state hashing was not used, the size of the hash table did not affect the state space in any way. Adding symmetry reduction reduced the state space by approximately 50%, which is the theoretical limit considering the reduction algorithm that was used. Although using bit state hashing reduces the state space, it must be kept in mind that it is an approximate method. Due to hash collisions there may be states that are reported as visited but actually are not explored. The least approximating model checking option used was bit state search with symmetry reduction enabled – the explored state space was exactly two states smaller than the full state space.

4.3.4. Performance evaluation

Two major limitations were met during the verifications. Firstly, the amount of the available memory is easily exhausted. As the memory allocation is exponential, the available memory will be used up very quickly when the used hash table size is increased. Therefore, the scalability of all state space reduction methods tried in the experiments, except the use of minimized DFA, is insufficient in terms of memory usage. Secondly, the amount of the available verification time is limited. The longest verification in this case study took approximately 3 h to complete. Of course, 3 h is a relatively short time but considering the size of the model used in verification – only two agents were used – the elapsed verification time can be expected to be multiple times longer. Although the verification time reaches a stabilization point after which there is no significant increase in the elapsed verification time, verification is still not scalable. Figures 5 and 6 represent the verification times for three different cleaning session durations in case symmetry reduction was not used and in case symmetry reduction was used, respectively. For example, let us look at the verification times for hash table size 32 in case symmetry reduction was not used. It can be seen that the increase in the elapsed time is exponential (166 s for the duration of 80 units; 567 s for the duration of 90 units; 1.56×10^3 s for the duration of 100 units).

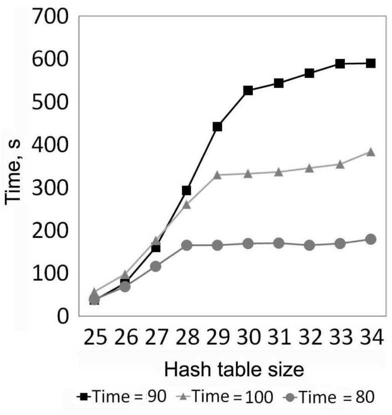

Fig. 5. Comparison of the elapsed time (no symmetry reduction).

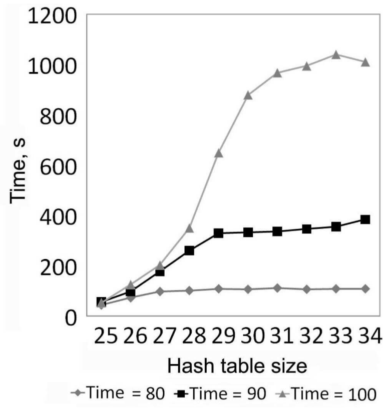

Fig. 6. Comparison of the elapsed time (symmetry reduction was used).

5. CONCLUSIONS AND FUTURE WORK

In this paper a case study of explicit state model checking techniques based on a dynamic cleaning problem for proving properties of emergent behaviour of robot swarms was presented. Performance analysis of the state space reduction methods used for the verification of the distributed coordination algorithm was made. As a result, we concluded that it is not feasible to use explicit state model checking, even with the presence of state space reduction techniques, to prove the properties of multi-agent systems, except in the case of a small number of agents. The results of the experiments imply that explicit state model checking alone is not feasible in industrial-size verification applications. The swarm size that can be used in correctness proofs will inherently depend on the amount of memory and time available.

To overcome the limitations of explicit state model checking, the future challenge is applying combined proof techniques, particularly induction-based proof schemata. If the swarm could be partitioned into tractable segments that maintain the abstract interface specification (the segment's invariant), explicit state model checking could be used to verify the behaviour segment-wise. The verified segment could then be used as a base case for induction argument.

For the induction step, one has to prove that if some composition of segments satisfies the verification conditions, then the condition will remain satisfied when the considered composition is composed with any other segment proved to be correct.

ACKNOWLEDGEMENTS

The research was funded by the Estonian Science Foundation under grant 7667, base funding BF99 of Tallinn Technical University, and targeted research funding SF0140018s08 of the Estonian Ministry of Education and Research.

REFERENCES

1. Altshuler, Y., Bruckelstein, A. M., and Wagner, I. A. Swarm robotics for a dynamic cleaning problem. In *IEEE Swarm Intelligence Symposium*. 2005, 209–216.
2. Baier, C. and Katoen, J.-P. *Principles of Model Checking*. MIT Press, 2008.
3. Behrmann, G., David, A., and Larsen, K. G. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS (Bernardo, M. and Corradini, F., eds). Springer-Verlag, 2004, 200–236.
4. Bonabeau, E., Theraulaz, G., Deneubourg, J.-L., Aron, S., and Camazine, S. Self-organization in social insects. *Trends in Ecology and Evolution*, 1997, **12**, 188–193.

5. Chandy, K. M. and Sanders, B. A. Reasoning about program composition. Preprint. 1998.
6. Clarke, E. M., Grumberg, O., and Long, D. E. Model checking and abstraction. *ACM T. Progr. Lang. Sys.*, 1994, **16**(5), 1512–1542.
7. Curtis, S. A., Rilee, M. L., Clark, P. E., and Marr, G. C. Use of swarm intelligence in spacecraft constellations for the resource exploration of the asteriod belt. In *Proceedings of the Third International Workshop on Satellite Constellations and Formation Flying*. 2003, 24–26.
8. Donaldson, A. TopSPIN 2.2 manual. http://www.allydonaldson.co.uk/topspin/TopSPIN_2.2_manual.pdf. November 2009.
9. Eilenberg, S. *Automata, Languages and Machines*. Vol. A. Academic Press, 1974.
10. Hoare, C. A. R. Communicating sequential processes. *Commun. ACM*, 1978, **21**(8), 666–677.
11. Holzmann, G. J. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.
12. Holzmann, G. J. and Peled, D. Partial order reduction of the state space. In *First SPIN Workshop*. Montreal, Quebec, 1995.
13. Holzmann, G. J. and Puri, A. A minimized automaton representation of reachable states. *Software Tools for Technology Transfer*, 1999, **2**(3), 270–278.
14. Lomuscio, A., Pecheur, C., and Raimondi, F. Automatic verification of knowledge and time with NuSMV. In *Proceedings of 20th International Joint Conference on Artificial Intelligence*. 2007, 1384–1389.
15. Martinoli, A., Easton, K., and Agassounon, W. Modeling swarm robotic systems: a case study in collaborative distributed manipulation. *Int. J. Robot. Res.*, 2004, **23**(4), 415–436.
16. Morris, R. Scatter storage techniques. *Communications of the ACM*, 1968, **11**(1), 38–44.
17. Rouff, C., Vanderbilt, A., Hinchey, M., Truszkowski, W., and Rash, J. Properties of a formal method for prediction of emergent behaviors in swarm-based systems. In *Second International Conference on Software Engineering and Formal Methods (SEFM'04)*. 2004, 24–33.
18. Tofts, C. Describing social insect behaviour using process algebra. *Trans. Social Comput. Simul.*, 1991, 227–283.
19. Vain, J., Tammet, T., Kuusik, A., and Juurik, S. Towards scalable proofs of robot swarm dependability. In *Proceedings of BEC 2008*. 2008, 199–202.

Robotiparvede ilmneva käitumise mudelkontroll

Silver Juurik ja Jüri Vain

On käsitletud ilmutatud olekuruumiga mudelkontrolli meetodi skaleeruvuse probleemi robotiparve hajuskoordinatsiooni algoritmi näitel. On analüüsitud kolme mudelkontrollis kasutatavat olekuruumi kahandamise tehnikat: olekuvektori kokkusurumine, bittvektori paisksalvestus ja sümmeetria reduktsioon. Nimetatud meetodite analüüsi tulemused lubavad väita, et ilmutatud olekuruumiga mudelkontrolli meetod ei sobi üldjuhul suurte multiagent-süsteemide ilmneva käitumise omaduste tõestamiseks, lähtudes üksikagentide käitumismudelitest. Artiklis on välja toodud olekuruumi redutseerimise tehnikate puudused, mis ilmnesid dünaamilise koristusalgoritmi verifitseerimisel.