Formal Verification of Parameterized Multi-agent Systems Using Predicate Diagrams*

Cecilia E. Nugraheni
Informatics Department
Parahyangan Catholic University
Bandung, Indonesia
Email: cheni@unpar.ac.id

***Abstract*—This paper presents a formal diagram-based verification technique for multi-agent systems. A multi-agent system is a collection of intelligent agents that interact with each others and work together to achieve a goal. We view multi-agent systems as parameterized systems which are systems that consist of several similar processes whose number is determined by an input parameter. The motivation of this work is that by treating multi-agent systems as parameterized systems, the specification and verification processes can be done in the same way regardless of the number of agents involved in the multi-agent systems. In this paper, we show how predicate diagrams* can be used to represent the abstractions of parameterized multi-agent systems described by specifications written in TLA*. The verification process is done by integrating deduction verification and algorithmic techniques. The correspondence between the original specification and the diagram is established by non-temporal proof obligations; whereas model checker SPIN is used to verify properties over finite-state abstractions.**

Keywords-multi-agent systems; parameterized systems; verification; predicate diagrams*; TLA*; TLA+.

I. Introduction

A multi-agent system is understood as a collection of intelligent agents, in this case are software or programs that interact with each others and work together to achieve a goal. Sycara [22] said that ”Agent-based systems technology has generated lots of excitement in recent years because of its promise as a new paradigm for conceptualizing, designing, and implementing software systems. This promise is particularly attractive for creating software that operates in environments that are distributed and open, such as the internet.” Because of this promise, many researches on multi-agent systems have been conducted. Most of these researches concentrate on the specification and verification of the agents’ behaviors and the coordination among agents.

In this work, we focus on multi-agent systems which consist of several similar processes. Having this property, a multi-agent system can be viewed as a parameterized system, which is a system that consists of several similar processes whose number is determined by an input parameter. Many interesting systems are of this form. One of them is mutual exclusion algorithms for an arbitrary number of processes wanting to use a common resource. The motivation of this work is that by treating multi-agent systems as parameterized systems, the specification and verification processes can be done more easily. Both processes are expected to be done in the same way regardless of the number of agents involved in the multi-agent systems.

Verification consists of establishing whether a system satisfies some properties, that is, whether all possible behaviors of the system are included in the properties specified. It is common to classify the approaches to formal verification into two groups, which are the deductive approach and the algorithmic approach. The deductive approach is based on theorem proving and typically reduces the proof of a temporal property to a set of proofs of first-order verification conditions. The most popular algorithmic verification method is model checking [6], [7], [21]. Although this method is fully automatic for finite-state systems, it suffers from the so-called state-explosion problem.

The need for a more intuitive approach to verification leads to the use of diagram-based formalisms. Basically, a diagram is a graph whose nodes represent sets of system states and whose edges represent the transition of the systems. Diagram-based approach combines the advantages of deductive and algorithmic approach, which are the process is goal-directed, incremental and can handle infinite-state systems.

In the context of parameterized systems, to provide methods for the uniform verification of such systems is a challenging problem. One solution of this problem is to treat or to represent a family of objects as a single syntactic object. This technique is called parameterization.

In [20], a diagram-based verification for parameterized systems is proposed. The diagrams, which are called predicate diagrams*, are variants of diagrams proposed by Cansell et al. In [5], they presented a class of diagrams called predicate diagrams and showed how to use the diagrams in formal verification. In [20], a little modification of the definition of the original predicate diagrams is made, in particular the definition related to the actions. Instead of actions, the new approach concentrates only on parameterized actions which are actions of the form $A(k)$. This form of actions is usually used in modeling actions of a particular process in the system. TLA* [17] is used to formalize this approach and TLA+ [13] style is used to write specifications.

This paper is structured as follows. Section II explains briefly the formal specification of parameterized systems in TLA*. Section III describes the definition and the use of predicate diagrams* in the verification of parameterized systems. The next section describes the aplication of this approach on a case study which is block world problem. Discussion about the result and some related works are given in Section V. Finally, conclusion and future work will be given in Section VI.

II. Pameterized Systems Specification

This work is restricted to a class of parameterized systems that are interleaving and consist of a finitely, but arbitrarily, discrete components. Let M denote a finite and non-empty set of processes running in the system. A parameterized system can be described as a formula of the form:

$$parSpec \equiv \forall k \in M : Init(k) \wedge \Box[Next(k)]_{v[k]} \wedge L(k).$$

where

- $Init$ is a state predicate describing the global initial condition,
- $Next(k)$ is an action characterizing the next-state relation of a process k,
- v is a state function representing the variables of the system and
- $L(k)$ is a formula stating the liveness conditions expected from the process or subsystem k.

III. Predicate Diagrams*

Basically, a predicate diagram* is a finite graph whose nodes are labeled with sets of (possibly negated) predicates, and whose edges are labeled with actions as well as optional annotations. This section gives a brief description of predicate diagrams*. For a detail explanation for predicate diagrams*, the readers may consult [20].

A. Definition

It is assumed that the underlying assertion language contains a finite set $\mathcal{O}$ of binary relation symbols $\prec$ that are interpreted by well-founded orderings. For $\prec \in \mathcal{O}$, its reflexive closure is denoted by $\preceq$. We write $\mathcal{O}^{=}$ to denote the set of relation symbols $\prec$ and $\preceq$ for $\prec \in \mathcal{O}$.

Definition 1. Assume given two finite sets $\mathcal{P}$ and $\mathcal{A}$ of state predicates and parameterized action names. A predicate diagram* $G = (N, I, \delta, o, \zeta)$ over $\mathcal{P}$ and $\mathcal{A}$ consists of :

- a finite set $N \subseteq 2^{\overline{\mathcal{P}}}$ of nodes where $\overline{\mathcal{P}}$ denotes the set of literals formed by the predicates in $\mathcal{P}$,
- a finite set $I \subseteq N$ of initial nodes,
- a family of δ_A where $A \in \mathcal{A}$ of relations $\delta_{A \subseteq N \times N}$; we also denote by δ the union of the relations δ_A, for $A \in \mathcal{A}$ and write $\delta^{=}$ to denote the reflexive closure of the union of these relations,
- an edge labeling o that associates a finite set $\{(t_1, 1), \ldots, (t_k, k)\}$, of terms t_i paired with a relation $\prec_i \in \mathcal{O}^{=}$ with every edge $(n, m) \in \delta$, and
- a mapping $\zeta : \mathcal{A} \rightarrow \{\text{NF,WF,SF}\}$ that associates a fairness condition with every parameterized action in $\mathcal{A}$; the possible values represent no fairness, weak fairness, and strong fairness.

A parameterized action $A \in \mathcal{A}$ can be taken at node $n \in N$ iff $(n, m) \in \delta_A$ holds for some $m \in N$. The set of nodes where A can be taken is denoted by $En(A)$.

A *run* of a predicate diagram* is a sequence of triples, $\rho = (s_0, n_0, A_0)(s_1, n_1, A_1)\ldots$ where s_i is a state, $n_i \in N$ is a node and $A_i \in \mathcal{A} \cup \{\tau\}$ (τ denotes stuttering transition). A *trace* through a predicate diagram*, $\sigma = s_0 s_1 \ldots$, is defined as the set of those behaviors that correspond to fair runs satisfying the node and edge labels.

B. Verification using predicate diagrams*

The verification process using predicate diagrams* is done in two steps. The first step is to find a predicate diagram* that conforms to the system specification. Theorem 1 is used to prove the conformance.

Theorem 1. Let $G = (N, I, \delta, o, \zeta)$ be a predicate diagram* over $\mathcal{P}$ and and $\mathcal{A}$, let $parSpec = Init \wedge \Box[\exists k \in M : Next(k)]_v \wedge \forall k \in M : L(k)$ be a parameterized system. If all the following conditions hold then G conforms to $parSpec$:

1) For all $n \in I$, $\models Init \rightarrow n$.
2) $|\!\approx n \wedge [\exists k \in M : Next(k)]_v \rightarrow$
$$n' \vee \bigwedge_{(m,A(k)):(n,m)\in\delta_{A(k)}} \langle \exists k \in M : A(k)\rangle_v \wedge m'$$
3) For all $n, m \in N$ and all $(t, \prec) \in o(n, m)$:
   a) $|\!\approx n \wedge m' \wedge \bigwedge_{A(k):(n,m)\in\delta_{A(k)}} \langle \exists k \in M : A(k)\rangle_v \rightarrow t' \prec t.$
   b) $|\!\approx n \wedge [\exists k \in M : Next(k)]_v \wedge n' \rightarrow t' \preceq t.$
4) For every parameterized action $A(k) \in \mathcal{A}$ such that $\zeta(A(k)) \neq$ NF:
   a) If $\zeta(A(k)) =$ WF then
   $\models parSpec \rightarrow \text{WF}_v(\exists k \in M : A(k)).$
   b) If $\zeta(A(k)) =$ SF then
   $\models parSpec \rightarrow \text{SF}_v(\exists k \in M : A(k)).$
   c) $|\!\approx n \rightarrow$ ENABLED $\langle \exists k \in M : A(k)\rangle_v$ holds whenever $n \in En(A(k))$.
   d) $|\!\approx n \wedge \langle \exists k \in M : A(k)\rangle_v \rightarrow m'$ holds for all $n, m \in N$ such that $(n, m) \notin \delta_{A(k)}$.

The second step of the verification process is to prove that all traces through a predicate diagram* satisfy some property F. In this step, a diagram predicate* is viewed as a finite transition system. As a finite transition system, its

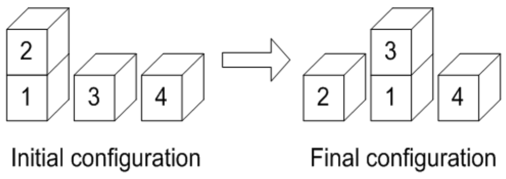

Figure 1. An example of block world problem.

runs can be encoded in the input language of standard model checkers such as SPIN [3].

IV. THE BLOCK WORLD : A CASE STUDY

A. Problem statement

One of the most famous planning domains in artificial intelligence is the block world problem. This problem can be briefly described as follows [23]: given a set of cubes (blocks) on a table and two types of robots whose task is to change the vertical stacks of blocks, from the initial configuration into a new different configuration. It is assumed that only one block may be moved at a time: it may either be placed on the table or placed atop another block. Any block that is, at a given time, under another block cannot be moved. Each robot has different capability: one robot is only capable of 'freeing' one block from another, while the other is only capable of 'moving' one block and putting it on top of another.

As illustration, Figure 1 shows an example of this problem. There are four blocks on the table. Each block is labeled with a number. The left-hand side represents the initial state of the world or initial configuration, whereas the right-hand side represents the goal (the final configuration) to be achieved.

B. Specification

From the problem statement, it is clear that the agents or the robots in the block world problem are not homogeneous. In order to model this problem as a parameterized system as required, one simple solution is taken, which is: (1) every robot is associated with an integer stating its type and (2) the capabilities of every robot type's are stated in a separate action. This is done in order to give an impression that the robots have different capabilities. Only now, an additional condition or precondition is added to each action. The homogeneity requirement is still guaranteed by using the same $Next$ action for each agent.

The specification for the block world problem is given in Figure 2. In this specification we use a set of positive integers, $Blocks$, to represent the collection of the blocks and an array ag_type to identify the agent's type.

To represent the state or configuration of the blocks, we use an array whose elements are pairs of non-negative integers. Each element represents the condition of a block.

$$
\begin{aligned}
isDone \equiv\ & CState = FState \\
move(k) \equiv\ & \land\ \neg isDone \land ag\_type[k] = 1 \\
& \land\ \exists x, y \in Blocks : \\
& \quad \land\ x \neq y \\
& \quad \land CState[x] = \langle 0, 0\rangle \land CState[y][2] = 0 \\
& \quad \land CState' = [CState \text{ EXCEPT }![x][1] = y, \\
& \qquad\qquad\qquad\qquad ![y][2] = x] \\
free(k) \equiv\ & \land\ \neg isDone \land ag\_type[k] = 2 \\
& \land\ \exists x, y \in Blocks : \\
& \quad \land\ x \neq y \\
& \quad \land CState[x][2] = 0 \land CState[y][2] = x \\
& \quad \land CState' = [CState \text{ EXCEPT }![x] = \langle 0, 0\rangle, \\
& \qquad\qquad\qquad\qquad ![y][2] = 0] \\
Init \equiv\ & \land\ CState = IState \land CState \neq FState \\
& \land\ \forall k \in M : ag\_type[k] \in \{1, 2\} \\
Next(k) \equiv\ & Move(k) \lor Free(k) \\
L(k) \equiv\ & \mathrm{WF}_v(Move(k)) \land \mathrm{WF}_v(Free(k)) \\
v \equiv\ & \langle CState \rangle \\
BWorld \equiv\ & \land Init \land \Box[\exists k \in M : Next(k)]_v \\
& \land \forall k \in M : L(k)
\end{aligned}
$$

Figure 2. Specification for Block World Problem.

For example, if the second element of the array is $\langle 3, 1\rangle$ then it means that the block number 3 is under the block number 2 and the block number 1 is on the block number 2. A special number, 0, is used to represent table or nothing. Thus, the initial configuration of Figure 1 is represented by $\langle\langle 0, 2\rangle, \langle\langle 1, 0\rangle, \langle 0, 0\rangle, \langle 0, 0\rangle\rangle$ and the final configuration is represented by $\langle\langle 0, 3\rangle, \langle 0, 0\rangle, \langle 1, 0\rangle, \langle 0, 0\rangle\rangle$.

Three state arrays are used, which are $IState$, $CState$ and $FState$. Each array is used to represent initial, final, and current configuration, respectively. Current configuration records the last configuration of the system. The goal is achieved whenever $isDone$ is true, which means that the current and the final configuration have the same values.

Action $move(k)$ can be taken only by an agent whose capability is to move a block from the table and put it onto another block. Action $free(k)$ can be taken only by an agent whose capability is to free one block on the top of a stack and put it on the table. Formula $CState' = [CState$ EXCEPT$![x][1] = y, ![y][2] = x]$ in $move(k)$ action means that except the values of $CState[x]$ and $CState[y]$, the values of $CState'$ won't change after the action is taken.

The values of $Blocks, ag\_type, IState$, and $FState$ depend on the problem instance at hand. If the number of boxes and agent types change, we simply change $Blocks$ and ag_types accordingly. This holds also for the capabilities of the agents. We need only to add precondition to every action as explained.

C. Verification

We take a problem instance in Figure 1 for the verification purpose. The following formulas are added to our specification in Figure 2:

- $Blocks \equiv \{1, 2, 3, 4\}$
- $IState \equiv \langle\langle 0, 2\rangle, \langle 1, 0\rangle, \langle 0, 0\rangle, \langle 0, 0\rangle\rangle$

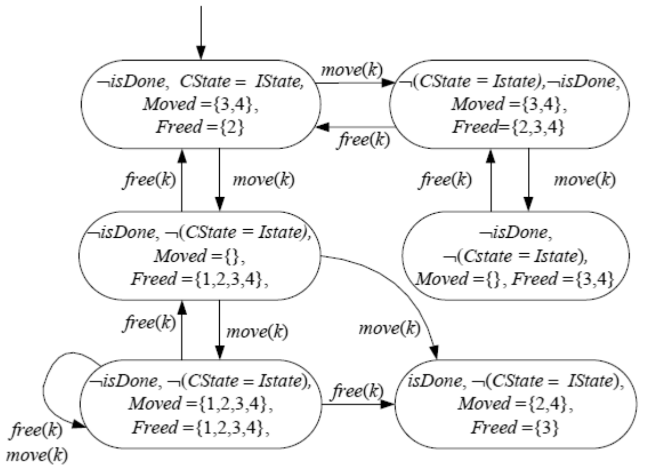
¬isDone, CState = IState, Moved ={3,4}, Freed ={2}
move(k)
free(k)
¬(CState = Istate),¬isDone, Moved ={3,4}, Freed={2,3,4}
free(k)
move(k)
free(k)
move(k)
¬isDone, ¬(CState = Istate), Moved ={}, Freed ={1,2,3,4},
¬isDone, ¬(Cstate = Istate), Moved ={}, Freed ={3,4}
free(k)
move(k)
move(k)
¬isDone, ¬(CState = Istate), Moved ={1,2,3,4}, Freed ={1,2,3,4},
free(k)
isDone, ¬(CState = IState), Moved ={2,4}, Freed ={3}
free(k)
move(k)

Figure 3. Predicate diagrams* for Block World Problem.

- $FState \equiv \langle\langle 0,3\rangle, \langle 0,0\rangle, \langle 1,0\rangle, \langle 0,0\rangle\rangle$

The verification can be stated as to prove the following theorem:

$$BWorld \rightarrow \Box(Init \rightarrow \Diamond isDone).$$

For the first step, we have to find a suitable predicate diagram* for the $BWorld$. Figure 3 depicts one of the possible predicate diagrams* for *BWorld*.

We set $\mathcal{P}$ to contain six predicates. The union of those predicates define the properties of system's states that hold on every node. There are two predicates that are not written explicitly, which are :$k \in \{1,2\}$ and $\forall k \in M : ag\_type \in \{1,2\}$. These predicates hold on every node. We also use two sets $Moved$ and $Freed$ which are sets of integers to indicate the blocks that can be moved or can be freed, respectively. It is assume that the following conditions hold:

- $\forall i \in Blocks : i \in Moved \leftrightarrow (\exists j \in Blocks : i \neq j \wedge CState[i] = \langle 0,0\rangle \wedge CState[j][2] = 0)$
- $\forall i \in Blocks : i \in Freed \leftrightarrow (\exists j \in Blocks : i \neq j \wedge CState[i]\langle j,0\rangle \wedge CState[j][2] = i)$

Using Theorem 1 it can be shown that the predicate diagram* in Figure 3 conforms to the specification in Figure 2. This is done by proving the following formulas:

- $Init \rightarrow \neg isDone \wedge \neg(CState = IState) \wedge Moved = \{3,4\} \wedge Freed = \{2\}$
- $\neg isDone \wedge \neg(CState = IState) \wedge Moved = \{3,4\} \wedge Freed = \{2\} \wedge [\exists \in M : Next(k)]_v \rightarrow$
  $(\neg isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{3,4\} \wedge Freed' = \{2\}) \vee$
  $((\langle \exists k \in M : free(k)\rangle_v \wedge \neg isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{3,4\} \wedge Freed' = \{2,3,4\}) \vee$
  $(\langle \exists k \in M : free(k)\rangle_v \wedge \neg isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{\} \wedge Freed' = \{1,2,3,4\}))$
- $\neg isDone \wedge \neg(CState = IState) \wedge Moved = \{\} \wedge Freed = \{1,2,3,4\} \wedge [\exists \in M : Next(k)]_v \rightarrow$
  $(\neg isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{\} \wedge Freed' = \{1,2,3,4\}) \vee$
  $((\langle \exists k \in M : move(k)\rangle_v \wedge isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{2,4\} \wedge Freed' = \{3\}) \vee$
  $(\langle \exists k \in M : move(k)\rangle_v \wedge \neg isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{1,2,3,4\} \wedge Freed' = \{1,2,3,4\}))$
- $\neg isDone \wedge \neg(CState = IState) \wedge Moved = \{1,2,3,4\} \wedge Freed = \{1,2,3,4\} \wedge [\exists \in M : Next(k)]_v \rightarrow$
  $(\neg isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{1,2,3,4\} \wedge Freed' = \{1,2,3,4\}) \vee$
  $((\langle \exists k \in M : free(k)\rangle_v \wedge (isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{\} \wedge Freed' = \{1,2,3,4\} \vee \neg isDone \wedge \neg(CState = IState) \wedge Moved = \{1,2,3,4\} \wedge Freed = \{1,2,3,4\})) \vee$
  $(\langle \exists k \in M : move(k)\rangle_v \wedge \neg isDone' \wedge (\neg(CState' = IState') \wedge Moved' = \{2,4\} \wedge Freed' = \{3\} \vee \neg isDone \wedge \neg(CState = IState) \wedge Moved = \{1,2,3,4\} \wedge Freed = \{1,2,3,4\}))$
- $(isDone \wedge \neg(CState = IState) \wedge Moved = \{2,4\} \wedge Freed = \{3\}) \wedge [\exists \in M : Next(k)]_v \rightarrow$
  $(isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{2,4\} \wedge Freed' = \{3\})$
- $(\neg isDone \wedge \neg(CState = IState) \wedge Moved = \{3,4\} \wedge Freed = \{2,3,4\}) \wedge [\exists \in M : Next(k)]_v \rightarrow$
  $(\neg isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{3,4\} \wedge Freed' = \{2,3,4\}) \vee$
  $((\langle \exists k \in M : free(k)\rangle_v \wedge (\neg isDone' \wedge CState' = IState' \wedge Moved' = \{3,4\} \wedge Freed' = \{2\})) \vee$
  $(\langle \exists k \in M : move(k)\rangle_v \wedge \neg isDone' \wedge (\neg(CState' = IState') \wedge Moved' = \{\} \wedge Freed' = \{3,4\}))$
- $(\neg isDone \wedge \neg(CState = IState) \wedge Moved = \{\} \wedge Freed = \{3,4\}) \wedge [\exists \in M : Next(k)]_v \rightarrow$
  $(\neg isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{\} \wedge Freed' = \{3,4\}) \vee$
  $((\langle \exists k \in M : free(k)\rangle_v \wedge (\neg isDone' \wedge \neg(CState' = IState') \wedge Moved' = \{3,4\} \wedge Freed' = \{2,3,4\})))$

The next step we encode the diagram in the input language of SPIN. We use 12 variables, which are:

- $action$ and $node$ to indicate the last action taken and the current node,
- $done$ to indicate whether $isDone$ is true or not,
- $cistate$ to indicate whether $CState = IState$ is true or not,
- $m1, m2, m3$, and $m4$ to represent predicate $1 \in Moved$, $2 \in Moved$, $3 \in Moved$ and $4 \in Moved$, respectively, and
- $f1, f2, f3$, and $f4$ to represent predicate $1 \in Freed$, $2 \in Freed$, $3 \in Freed$ and $4 \in Freed$, respectively,

The theorem to be proven is now can be written as $\Box((\neg done \wedge cistate \wedge \neg m1 \wedge \neg m2 \wedge m3 \wedge m4 \wedge \neg f1 \wedge f2 \wedge \neg f3 \wedge \neg f4) \rightarrow \Diamond(done \wedge \neg cistate \wedge \neg m1 \wedge m2 \wedge \neg m3 \wedge m4 \wedge \neg f1 \wedge \neg f2 \wedge f3 \wedge \neg f4))$. Last, by using SPIN we model-checked the resulted transition system. As result, we concluded that the specification satisfies the property we want to prove.

V. Related work

Many works are devoted to the formal specification of multi-agent systems. Most of these works concentrate on the specification of the agents' behaviors and the coordination among agents. Abouaissa et al. [1] presented a formal approach for specification of multi-agent systems. This approach is based on roles and organization notions and high-level Petri nets and is applied on a multi-modal platform associating combined rail-road transportation system. Merayom et al. [16] proposed a formalism called Utility State Machines for specifying e-commerce multi agent systems. Brazier et al. [3] used the DESIRE framework to specify a real-world multi-agent application on a conceptual level. Originally DESIRE is designed for formal specification of complex reasoning systems. Fischer and Wooldridge [9] described first step towards the formal specification and verification of multi-agent systems, through the use of temporal belief logics. This work is closed to the one of Taibi [23]. The similarity is that we use TLA-based to formalize our approaches. However, Taibi's work did not treat the multi -agent systems as parameterized systems.

Besides formal specification, formal verification is also a popular topic in the field of multi-agent systems. Several approaches can be found in [2], [10], [11], [15], [23]. In [11], Giese et al. presented an approach for making complex multi-agent system specifications. Every specification includes a detailed environment model that amenable to verification. The verification process is done by means of simulation and formal verification. Taibi used TLC, the TLA model checker, to verify the specification [23]. Gaud et al. proposed a formal framework based on multi-formalisms language for writing system specification and used abstraction to reduce the state space of the system [10]. Ayed et al. proposed a diagram-based verification by using AUML protocol diagrams for representing multi-agent systems. These diagrams are then translated into event-B language for verification purpose [2]. In [15], Massaci et al. concerned about the use of access control for limiting the agent capability of distributed systems. They presented a prefixed tableau method for the calculus of access control. The calculus was the basis for the development and the verification of an implemented system.

Because diagrams can reflect the intuitive understanding of the systems and their specifications, they are proposed to be used for verification. A diagram can also be seen as an abstraction of the system, where properties of the diagram are guaranteed to hold for the systems as well. In particular, the use of diagrams in verification of distributed systems can be found; for example in [5] the author proposed the use of predicate diagrams, introduced in [4], for analyzing a self-stabilizing algorithm.

VI. Conclusion

We have shown how a multi-agent system can be viewed and thus can be formally specified and verified as a parameterized system. In particular, we define a general form for specification of parameterized multi-agent systems in TLA*. By considering a case study, we have shown that a multi-agent system whose agents are not homogenous still can be specified as a parameterized system.

In this paper, we have successfully write specification and verify the block world problem. This problem is an example of multi-agent system whose agents are not homogeneous. In order to fulfill the homogeneity requirement, we add preconditions to actions in the specification to guarantee that only the appropriate agent may take a particular action. For verification process we use predicate diagram* to represent the abstractions of the systems. The correspondence between the original specification and the diagram is established by non-temporal proof obligations; whereas model checker SPIN is used to verify properties over finite-state abstractions.

In the context of parameterized systems, there are two classes of properties that may be considered, namely the properties related to the whole processes and the ones related to a single process in the system. It is planned to investigate those properties of the block world problem. The verification will be conducted by using a diagram-based verification called parameterized predicate diagrams [18], [19].

References

[1] H. Abouaissa, J.C. Nicolas, A. Benasser, and E. Czesnalowicz. *Formal specification of multi-agent systems: approach based on meta-models and high-level Petri nets - case study of a transportation system*. Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Vol.5, pp. 429-434, 2002.

[2] B. Ayed and F. Siala. *Event-B based Verification of Interaction Properties In Multi-Agent Systems*. Journal of Software, Vol. 4, No. 4, pp. 357-364, June 2009

[3] F. Brazier, B. Dunin-Keplicz, N.R. Jennings, and J. Treur. *Formal Specification of Multi-Agent Systems: a Real World Case*. Proceedings of the First International Conference on Multi-Agent Systems, ICMAS'95, MIT Press, Cambridge, MA, 1995, pp.25-32.

[4] D. Cansell, D. Méry, and S. Merz. *Predicate diagrams for the verification of reactive systems*. In 2nd Intl. Conf. on Integrated Formal Methods IFM 2000, vol. 1945 of Lectures Notes in Computer Science, pp. 380-397, 2000. Springer-Verlag.

[5] D. Cansell, D. Méry, and S. Merz. *Formal analysis of a self-stabilizing algorithm using predicate diagrams*. GI Jahrestagung (1) 2001: 39-45.

[6] E.M. Clarke and E.A. Emerson. *Characterizing correctness properties of parallel programs using fixpoints*. *International Colloquim on Automata, Languages and Programming*. Vol. 85 of Lecture Nodes in Computer Science, pp. 169-181, Springer-Verlag, July, 1980.

[7] E.M. Clarke and E.A. Emerson. *Design and synthesis of synchronization skeletons using branching time temporal logic*. Workshop on Logic of Programs, Yorktown Heights, NY. Vol. 131 of Lecture Nodes in Computer Science, pp. 52-71, Springer-Verlag, 1981.

[8] E. A. Emerson and K. S. Namjoshi. *Verification of a parameterized bus arbitration protocol*. Volume 1427 of Lecture Notes in Computer Science, pp. 452–463. Springer,1998.

[9] M. Fisher and M. Wooldridge. *On the Formal Specification and Verification of Multi-Agent Systems*. Int. J. Cooperative Inf. Syst., 1997: 37-66.

[10] N. Gaud. *A Verification by Abstraction Framework for organizational Multi-Agent Systems*. Jung, Michel, Ricci and Petta (eds.): AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop, May 13, 2008, AAMAS 2008, pp. 67-73, Estoril, Portugal, EU.

[11] H. Giese and F. Klein. *Systematic verification of multi-agent systems based on rigorous executable specifications*. Journal International Journal of Agent-Oriented Software Engineering, Volume 1 Issue 1, pp. 28-62, April 2007.

[12] G. Holzmann. *The SPIN model checker. IEEE Trans*. On software engineering, 16(5):1512-1542. May 1997.

[13] L. Lamport. *The Temporal Logic of Actions. ACM Transactions on Programming Languages and Systems*, 16(3) : 872-923, May 1994.

[14] Z. Manna and A. Pnueli. *Verification of parameterized programs*. In Specification and Validation Methods (E. Borger, ed.), Oxford University Press, pp. 167-230, 1994.

[15] F. Massacci. *Tableau Methods for Formal Verification of Multi-Agent Distributed Systems*. Journal of Logic and Computation, 8(3), 1998.

[16] M.G. Merayo. *Formal specification of multi-agent systems by using EUSMs*. Proceedings of the 2007 international conference on Fundamentals of software engineering, pp. 318-833.

[17] S. Merz. *Logic-based analysis of reactive systems: hiding, composition and abstraction*. Habilitationsschrift. Institut fr Informatik. Ludwig-Maximillians-Universitt, Munich Germany. December 2001.

[18] C.E. Nugraheni. *Predicate diagrams as basis for the verification of reactive systems*. PhD Thesis. Institut fr Informatik. Ludwig-Maximillians-Universitt, Munich Germany. February 2004.

[19] C.E. Nugraheni. *Universal properties verification of parameterized parallel systems*. In Proceeding of the International Confe-rence on Computational Scince and its Applications (ICCSA 2005), Volume 3482 of Lecture Notes in Computer Science, pp. 453-462. Springer, 2005.

[20] C.E. Nugraheni. *Formal Verification of Ring-based Leader Election Protocol using Predicate Diagrams**. IJCSNS Vol. 9. no. 8, pp. 1-8, August 2009.

[21] J.P. Quielle and J. Sifakis. *Specification and verification of concurrent systems in CESAR*. In M. Dezani-Cianzaglini and Ugo Montanari, editors, International Symposium on Programming. Volume 137 of Lecture Notes in Computer Science, pp. 337-350. Springer-Verlag, 1981.

[22] K.P. Sycara. *Multiagent Systems*. AI Magazine, Vol 19, No 2, pp. 79-92, Summer 1998.

[23] T. Taibi. *Formal specification and validation of multi-agent behaviour using TLA+ and TLC model checker*. Int. J. Artificial Intelligence and Soft Computing, Vol. 1, No. 1, pp. 99-113, 2008.