Diagram-based verification of parameterized systems

Cecilia E. Nugraheni

Computer Science Dept., Fac. of Mathematics and Natural Sciences,
Parahyangan Catholic University, Bandung, Indonesia
cheni@home.unpar.ac.id

Abstract. A parameterized system consists of several similar processes whose number is determined by an input parameter. A challenging problem is to provide methods for the uniform verification of such systems, i.e. to show by a single proof that a system is correct for any value of the parameter.
This paper presents a method for verifying parameterized systems using predicate diagrams*. Basically, predicate diagrams* are graphs whose vertices are labelled with first-order formulas, representing sets of system states, and whose edges represent possible system transitions. These diagrams are used to represent the abstractions of parameterized systems described by specifications written in temporal logic.
This presented method integrates deductive verification and algorithmic techniques. Non-temporal proof obligations establish the correspondence between the original specification and the diagram, whereas model checking is used to verify properties over finite-state abstractions.

Keywords: parameterized systems, diagram-based verification, predicate diagrams.

1 Introduction

Reactive systems are systems that are expected to maintain an ongoing interaction with their environment. Verification of reactive systems consists of establishing whether a system satisfies some property, that is, whether all possible behaviors of the system are included in the property specified. There are basically two approaches to formal verification, which are the deductive approach and the algorithmic approach. The deductive approach is based on *verification rules*, which reduce the system validity of a temporal property to the general validity of a set of first-order *verification conditions*. The most popular algorithmic verification method is *model checking*. Although this method is fully automatic for finite-state systems, it suffers from the so-called *state-explosion* problem. The

size of the state space is typically exponential in the number of components, and therefore the class of systems that can be handled by this method is limited.

The need for a more intuitive approach to verification leads to the use of diagram-based formalisms. Usually, these diagrams are graphs whose vertices are labelled with first-order formulas, representing sets of system states, and whose edges represent possible system transitions. This approach combines some of the advantages of deductive and algorithmic verification: the process is goal-directed, incremental and can handle infinite-state systems.

A parameterized system is a class of reactive systems which consists of several similar processes whose number is determined by an input parameter. Many interesting systems are of this form, for example, mutual exclusion algorithms for an arbitrary number of processes wanting to use a common resource. To provide methods for the uniform verification of such systems is a challenging problem. The ability to conduct a uniform verification of a parameterized system is one of the striking advantages of the deductive method for temporal verification over algorithmic techniques such as model-checking techniques. Generally, nothing can be concluded about the property holding for any value of n from the fact that it holds for some finite set of values. In comparison, the deductive method establishes in one fell swoop the validity of the property for any value of n [13, 7].

This paper proposes a diagram-based verification technique for parameterized systems. In [4] CANSELL et.al presented a class of diagrams called predicate diagrams and showed how the diagrams used in formal verification. We investigate the use of these diagrams in verification of parameterized systems [16]. We made a little modification on the definition of the original predicate diagrams, in particular the definition related to the actions. Instead of actions, we concentrate only on parameterized actions which are actions of the form $A(k)$. This form of actions is usually used in modelling actions of a particular process in the system. This new class of diagrams is called predicate diagrams*. We use TLA* [14] to formalize our approach and use TLA+ style [12] for writing specifications.

This paper is structured as follows. We begin with a brief description of the specification of parameterized system in TLA*. Section 3 describes the definition and the use of predicate diagrams in the verification of parameterized systems. As illustration we take the parameterized reader-writer algorithm as case study [10]. This is explained in Section 4. Some related work is given in Section 5. Finally, conclusion and future work will be given in Section 6.

2 Specification of Parameterized Systems

In this work, we restrict on the parameterized systems which are *interleaving* and consist of finitely, but arbitrarily, *discrete* components. Let M denotes a finite and non-empty set of processes running in the system being considered. A parameterized system can be describe as a formula of the form:

$$parSpec \equiv Init \wedge \Box[\exists k \in M : Next(k)]_v \wedge \forall k \in M : L(k) \qquad (1)$$

where

- *Init* is a state predicate that describes the global initial condition,
- $Next(k)$ is an action that characterizes 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 k.

Formulas such as $Next(k)$ and $L(k)$ are called parameterized actions.

3 Predicate diagrams*

Now we present a class of diagrams that can be used for the verification of parameterized systems. The underlying assertion language, by assumption, 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 $\prec \in \mathcal{O}$.

3.1 Definition of predicate diagrams*

A predicate diagram* is a finite graph whose nodes are labelled with sets of (possibly negated) predicates, and whose edges are labelled with parameterized actions as well as optional annotations that assert certain expressions to decrease with respect to an ordering in $\mathcal{O}^=$. Intuitively, a node of a predicate diagram represents the set of system states that satisfy the formulas contained in the node. An edge (n, m) is labelled with a parameterized action $A(k)$ if $A(k)$ can cause a transition from a state represented by n to a state represented by m. A parameterized action $A(k)$ may have an associated fairness condition; fairness conditions apply to all transitions labelled by the action rather than to individual edges.

Formally, the definition of predicate diagrams* is relative to finite sets $\mathcal{P}$ and $\mathcal{A}$ that contain the state predicates and the parameterized actions of interest; we will later use τ to denote a special stuttering action. We write $\overline{\mathcal{P}}$ to denote the set of literals formed by the predicates in $\mathcal{P}$, that is, the union of $\mathcal{P}$ and the negations of the predicates in $\mathcal{P}$.

Definition 1. *Assume given two finite sets $\mathcal{P}$ and $\mathcal{A}$ of state predicates and parameterized actions. 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,*
- *a finite set $I \subseteq N$ of initial nodes,*
- *a family of $\delta = (\delta_{A(k)})_{A(k) \in \mathcal{A}}$ of relations $\delta_{A(k)} \subseteq N \times N$,*
- *an edge labelling o that associates a finite set $\{(t_1, \prec_1), \ldots, (t_d, \prec_d)\}$, of terms t_i paired with a relation $\prec_i \in \mathcal{O}^=$ with every edge $(n, m) \in \delta$, and*
- *a mapping $\zeta : \mathcal{A} \to \{\mathrm{NF}, \mathrm{WF}, \mathrm{SF}\}$ that associates a fairness condition with every parameterized action in $\mathcal{A}$; the possible values represent no fairness, weak fairness, and strong fairness.*

We say that the parameterized action $A(k)$ can be taken at node $n \in N$ iff $(n,m) \in \mathcal{A}$ holds for some $m \in N$, and denote by $En(A(k)) \subseteq N$ the set of nodes where $A(k)$ can be taken. We say that the parameterized action $A(k)$ can be taken along an edge (n,m) iff $(n,m) \in \delta_{A(k)}$.

We now define runs and traces through a diagram as the set of those behaviors that correspond to fair runs satisfying the node and edge labels. To evaluate the fairness conditions we identify the enabling condition of a parameterized action $A(k)$ with the existence of $A(k)$-labelled edges at a given node.

Definition 2. *Let $G = (N, I, \delta, o, \zeta)$ be a predicate diagram* over sets $\mathcal{P}$ and $\mathcal{A}$. A run of G is an ω-sequence $\sigma(s_0, n_0, A_0)(s_1, n_1, A_1)\ldots$ of triples where s_i is a state, $n_i \in N$ is a node and A_i is a parameterized action such that all of the following conditions hold:*

- *$n_0 \in I$ is an initial node.*
- *$s_i[\![n_i]\!]$ holds for all $i \in N$.*
- *For all $i \in N$, either $A_i = \tau$ and $n_i = n_{i+1}$ or $A_i \in \mathcal{A}$ and $(n_i, n_{i+1}) \in \delta_{A_i}$.*
- *If $A_i \in \mathcal{A}$ and $(t, \prec) \in o(n_i, n_{i+1})$, then $s_{i+1}[\![t]\!] \prec s_i[\![t]\!]$.*
- *If $A_i = \tau$ then $s_{i+1}[\![t]\!] \preceq s_i[\![t]\!]$ holds whenever $(t, \prec) \in o(n_i, m)$ for some $m \in N$.*
- *For every parameterized action $A(k)$ such that $\zeta(A(k)) = \mathrm{WF}$ there are infinitely many $i \in \mathbb{N}$ such that either $A_i = A(k)$ or $n_i \notin En(A(K))$.*
- *For every parameterized action $A(k)$ such that $\zeta(A(k)) = \mathrm{SF}$, either $A_i = A(k)$ holds for infinitely many $i \in \mathbb{N}$ or $n_i \in En(A(k))$ holds for only finitely many $i \in \mathbb{N}$.*

We write $runs(G)$ to denote the set of runs of G. The set $tr(G)$ of traces through G consists of all behaviors $\sigma = s_0 s_1 \ldots$ such that there exists a run $\rho(s_0, n_0, A_0)(s_1, n_1, A_1)\ldots$ of G based on the states in σ.

Informally, $\sigma = s_0 s_1 \ldots$ is a trace through the predicate diagram* G if we can find a sequence of nodes n_i whose associated formulas are true at s_i and that are related by transitions whose edge labels, including the ordering annotations, are satisfied by consecutive states. In addition to the transitions that are explicitly represented by edges of the diagram, we allow stuttering transitions that remain in the source node.

Fairness conditions are used to prevent infinite stuttering. Their interpretation is standard, based on the intuition that the enabledness of actions with non-trivial fairness requirements is reflected in the diagram.

3.2 Verification using predicate diagrams*

The verification process using predicate diagrams is done in two steps [4]. The first step is to find a predicate diagram that can be proven to be the correct representation of the system to be verified, i.e. the diagram conforms to the system specification. For proving whether a diagram conforms to a specification

or not, the so-called conformance theorem is used. Thus the first step is done deductively.

With the current setting, i.e. the using of parameterized actions, some modifications should be done on the conformance theorem. In particular, the conditions related to the fairness conditions should be treated slightly differently from non-parameterized ones. We need to address one important issue that will be used later, which is the issue about fairness. Note that in the specification the fairness condition is represented as a conjunction of formulas of the forms $\forall k \in M : \mathrm{WF}_v(A(k))$ and/or $\forall k \in M : \mathrm{SF}_v(A(k))$, i.e. for every process k in M and for some parameterized action $A(k)$, we associate weak and strong fairness, respectively, with $A(k)$. Let's turn to the definition of predicate diagrams, in particular the definition of ζ. In the context of parameterized systems, $\zeta : \mathcal{A} \rightarrow \{\mathrm{NF}, \mathrm{WF}, \mathrm{SF}\}$ is now a mapping that associates a fairness condition with every parameterized action $A(k)$ in $\mathcal{A}$. For example, for some parameterized action $A(k)$, if $\zeta(A(k)) = \mathrm{WF}$ then we mean $\zeta(\exists k \in M : A(k)) = \mathrm{WF}$.

We say that a predicate diagram* G *conforms* to a parameterized program *parSpec* if every behavior that satisfies *parSpec* is a trace through G.

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

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

Proof. This theorem is a direct consequence of the conformance theorem. Notice that $\exists k \in M : A_1(k) \vee \ldots \vee A_n(k)$ is equivalent to $(\exists k \in M : A_1(k)) \vee \ldots \vee (\exists k \in M : A_n(k))$. Thus, we can use the proof for the conformance theorem which is given in [4] and [14] as reference in order to prove Theorem 1.

Assume that *parSpec* and G are such that all the conditions hold that $\sigma = s_0 s_1 \ldots$ is a behavior that satisfies *parSpec*. We want to show that $\sigma \in tr(G)$ by constructing the corresponding run through G for σ:

1. We inductively define a sequence $n_0, n_1, \ldots$ of nodes $n_i \in N$ and a sequence $\mathcal{A}_0\mathcal{A}_1 \ldots$ of sets of parameterized actions $\emptyset \neq \mathcal{A}_i \subseteq \mathcal{A} \cup \{\tau\}$ such that for all $i \in \mathbb{N}$ the following conditions hold:
   (i) $n_0 \in I$
   (ii) $s_i[\![n_i]\!]$
   (iii) $\mathcal{A}_i = \{A(k) \in \mathcal{A} : (n_i, n_{i+1}) \in \delta_{A(k)}, s_i[\![\langle \exists k \in M : A(k)\rangle_v]\!]s_{i+1}\} \cup \{\tau : n_{i+1} = n_i\}$
   (iv) $s_{i+1}[\![t]\!] \prec s_i[\![t]\!]$ whenever $(t, \prec) \in o(n_i, n_{i+1})$ and
   (v) $s_{i+1}[\![t]\!] \preceq s_i[\![t]\!]$ whenever $n_{i+1} = n_i$ and $(t, \prec) \in o(n_i, m)$ for some $m \in N$.

   Induction base: We choose some node $n_0 \in I$ such that $s_0[\![n_0]\!]$ holds. By assumption, $s_0[\![Init]\!]$ holds (Theorem 1 condition 1), therefore some such node exists.
   Induction step: We assume that $n_0, n_1 \ldots n_i$ and $\mathcal{A}_0, \mathcal{A}_1 \ldots \mathcal{A}_{i-1}$ have already been defined such that conditions (i)-(ii) hold for all $j \leq i$ and conditions (iii)-(v) hold for all $j \leq i-1$. In particular, we have $s_i[\![n_i]\!]$. Moreover, the assumption that $\sigma \models parSpec$ implies that $s_i[\![\exists k \in M : [Next(k)]_v]\!]s_{i+1}$, and condition 2 in Theorem 1 ensures that either there exist some parameterized action $A(k) \in \mathcal{A}$ and some $m \in N$ such that $(n_i, m) \in \delta_{A(k)}$ and $s_i[\![\langle \exists k \in M : A(k)\rangle_v \wedge m']\!]s_{i+1}$ holds, or $s_{i+1}[\![n_i]\!]$. In the first case, choose some such node m as n_{i+1}; in the second case, choose $n_{i+1} = n_i$. In either case, define $\mathcal{A}_i$ as described in condition (iii). These choices imply that $s_{i+1}[\![n_{i+1}]\!]$ and that $\mathcal{A}_i \neq \emptyset$. Conditions (iv) and (v) now follow from the choices of n_{i+1} and $\mathcal{A}_i$ with the help of conditions 3a and 3b in Theorem 1.
2. To complete the proof, we pick a sequence $A_0A_1 \ldots$ of parameterized actions such that the last two conditions of definition 2 that concern the fairness annotations in G are satisfied. Choose a sequence such that $A_i = A$ holds for infinitely many $i \in \mathbb{N}$ if $A_i \in \mathcal{A}_i$ holds for infinitely many $i \in \mathbb{N}$. Such a choice is possible by a standard combinatorial argument since the set $\mathcal{A}$ is finite.
   Assume that $A(k) \in \mathcal{A}$ is a parameterized action such that $\zeta(A(k)) = \mathrm{WF}$ and that $n_i \in En(A(k))$ holds for all but finitely many $i \in \mathbb{N}$ (otherwise the condition is trivially satisfied). Because we already know that $s_i[\![n_i]\!]$ holds for all $i \in \mathbb{N}$, condition 4c implies that $s_i[\![\exists k \in M : \mathrm{ENABLED}\langle A(k)\rangle_v]\!]$ holds for all but finitely many $i \in \mathbb{N}$. Moreover, since σ satisfies $parSpec$ and $\models parSpec \rightarrow \mathrm{WF}_v(\exists k \in M : A(k))$ by condition 4a, it follows that $s_i[\![\langle \exists k \in M : A\rangle]\!]s_{i+1}$ holds for infinitely many $i \in \mathbb{N}$. For every such i, we have $A(k) \in \mathcal{A}_i$; otherwise, condition 4d would imply that $s_{i+1}[\![\neg n_{i+1}]\!]$, contradicting assertion (ii) above. But then, the choice of the parameterized action sequence implies that $A_i = A(k)$ holds for infinitely many $i \in \mathbb{N}$, which completes the proof.
3. For parameterized action $A(k) \in \mathcal{A}$ such that $\zeta(A(k)) = \mathrm{SF}$, the proof is analogous, replacing "all but finitely many" by "infinitely many" and using 4b instead of 4a.

■

Condition 1 asserts that every initial state of the system must be covered by some initial node. This ensures that every run of the system can start at

some initial node of the diagram. Condition 2 asserts that from every node, every transition, if it is enabled then it must have a place to go, i.e., there is a successor node which represents the successor state of the transition. It proves that every run of the system can stay in the diagram. Condition 3 is related to the ordering annotations and Condition 4 is related to the fairness conditions.

The second verification step is to prove that all traces through a predicate diagram satisfy some property F. On this case, we view the diagram as a finite transition system that is amenable to model checking. All predicates and actions that appear as labels of nodes or edges are then viewed as atomic propositions.

Regarding predicate diagrams* as finite labelled transition systems, their runs can be encoded in the input language of standard model checkers such as SPIN [11]. Two variables indicate the current node and the last action taken. The predicates in $\mathcal{P}$ are represented by boolean variables, which are updated according to the label of the current node, nondeterministically, if that label contains neither P nor $\overline{P}$. We also add variables $b_{(t,\prec)}$, for every term t and relation $\prec \in \mathcal{O}$ such that $(t, \prec)$ appears in some ordering annotation $o(n, m)$. These variables are set to 2 if the last transition taken is labelled by $(t, \prec)$, to 1 if it is labelled by $(t, \preceq)$ or is stuttering transition and to 0 otherwise. Whereas the fairness conditions associated with the actions of a diagram are easily expressed as LTL (Linear Temporal Logic) assumptions for SPIN.

4 Reader-writer algorithm: a case study

The protocol is expressed in terms of four sets that contain (the identities of) currently active readers, waiting readers, active writers and waiting writers. All the sets are initially empty. Assume there are M processes running in the systems. Every process in the system can perform the following actions:

- $ImmRd(k)$: Process k signals its desire to read. If there are no active or waiting writers, then it is immediately proceed to read the data.
- $ReqRd(k)$: Process k requests reading access, but there are active or waiting writers. It is added to the set of waiting readers.
- $GrRd(k)$: Waiting reader k is allowed to read data when there are no active writers.
- $EndRd(k)$: Active reader k signals that it has finished reading and exists the protocol.
- $ImmWr(k)$: Process k signals its desire to write. If there are neither active or waiting writers nor active or waiting readers, it may immediately proceed to write the data.
- $ReqWr(k)$: Process k signals that it wants to write data, but the preconditions of action $ImmWr(k)$ are not met. It is added to the set of waiting writers.
- $GrWr(k)$: Waiting writer k is allowed to write when there are neither active or waiting readers nor active writers.
- $EndWr(k)$: Active writer k signals that it has finished writing and exits the protocol.

The fairness conditions are chosen such that no process that has entered the system is persistently neglected. In particular, there is a strong fairness condition on the action $DoWr(k)$, for all $k \in M$, presumably implemented using a queue rather than a set of waiting readers.

Our objective is to prove that the algorithm satisfies the following properties:

1. Mutual-exclusion among writers, i.e. every time there is only maximal one writing process.
2. Mutual-exclusion among readers and writers, i.e. every time whenever there are active readers then there are no active writers and vice versa.

The specification of parameterized reader-writer algorithm is given in Figure 1. There are four variables, which are ar, wr, aw, and ww, representing the set of active readers, waiting readers, active writers and waiting writers.

module *Reader-Writer*

VARIABLES ar, wr, aw, ww

$ImmRd(k) \equiv \wedge\ k \notin ar \cup wr \cup aw \cup ww \wedge aw = \emptyset \wedge ww = \emptyset$
$\quad \wedge\ ar' = ar \cup \{k\} \wedge$ UNCHANGED $\langle wr, aw, ww \rangle$
$ReqRd(k) \equiv \wedge\ k \notin ar \cup wr \cup aw \cup ww \wedge \neg(aw = \emptyset \wedge ww = \emptyset)$
$\quad \wedge\ wr' = wr \cup \{k\} \wedge$ UNCHANGED $\langle ar, aw, ww \rangle$
$GrRd(k) \equiv \wedge\ k \in wr \wedge aw = \emptyset$
$\quad \wedge\ ar' = ar \cup \{k\} \wedge wr' = wr \setminus \{k\} \wedge$ UNCHANGED $\langle aw, ww \rangle$
$EndRd(k) \equiv k \in ar \wedge ar' = ar \setminus \{k\} \wedge$ UNCHANGED $\langle wr, aw, ww \rangle$
$ImmWr(k) \equiv \wedge\ k \notin ar \cup wr \cup aw \cup ww \wedge ar \cup wr \cup aw \cup ww = \emptyset$
$\quad \wedge\ aw' = aw \cup \{k\} \wedge$ UNCHANGED $\langle ar, wr, ww \rangle$
$ReqWr(k) \equiv \wedge\ k \notin ar \cup wr \cup aw \cup ww \wedge ar \cup wr \cup aw \cup ww = \emptyset$
$\quad \wedge\ ww' = ww \cup \{k\} \wedge$ UNCHANGED $\langle ar, wr, aw \rangle$
$GrWr(k) \equiv \wedge\ k \in ww \wedge ar \cup wr \cup aw = \emptyset$
$\quad \wedge\ aw' = aw \cup \{k\} \wedge ww' = ww \setminus \{k\} \wedge$ UNCHANGED $\langle ar, wr \rangle$
$EndWr(k) \equiv k \in aw \wedge aw' = aw \setminus \{k\} \wedge$ UNCHANGED $\langle ar, wr, ww \rangle$

$Init \equiv ar = \emptyset \wedge wr = \emptyset \wedge aw = \emptyset \wedge ww = \emptyset$
$Next(k) \equiv \vee\ ImmRd(k) \vee ReqRd(k) \vee GrRd(k) \vee EndRd(k)$
$\quad \vee\ ImmWr(k) \vee ReqWr(k) \vee GrWr(k) \vee EndWr(k)$
$v \equiv \langle ar, wr, aw, ww \rangle$
$L(k) \equiv \mathrm{WF}_v(GrRd(k)) \wedge \mathrm{WF}_v(EndRd(k)) \wedge \mathrm{SF}_v(GrWr(k)) \wedge \mathrm{WF}_v(EndWr(k))$
$ParRW \equiv \forall k \in 1..M : Init \wedge \Box[Next(k)]_v \wedge L(k)$

THEOREM $parRW \rightarrow \Box(|aw| \leq 1)$
THEOREM $parRW \rightarrow \Box(|aw| = 1 \rightarrow |ar| = 0)$
THEOREM $parRW \rightarrow \Box(|ar| > 0 \rightarrow |aw| = 0)$

Fig. 1. Specification of reader-writer algorithm.

Figure 2 shows a suitable predicate diagram* for the parameterized reader-writer algorithm. We set $\mathcal{P}$ to contain the eight predicates, which are: $|ar| = 0$, $|ar| > 0$, $|wr|0$, $|wr| > 0$, $|aw| = 0$, $|aw| = 1$, $|ww| = 0$, and $|ww| > 0$. It is assumed that the following conditions hold: $\neg(|ar| = 0) \leftrightarrow |ar| > 0, \neg(|wr| = 0) \leftrightarrow |wr| > 0, \neg(|aw| = 0) \leftrightarrow |aw| = 1$, and $\neg(|aw| = 0) \leftrightarrow |aw| > 0$. It can be argued that $\neg(|aw| = 0) \leftrightarrow |aw| = 1$ holds, since the number of the active writers is always at most 1. $IR, RR, GR, ER, IW, RW, GW$, and EW stands for $ImmRd(k), ReqRd(k)$, $GrRd(k), EndRd(k), ImmWr(k), ReqWr(k)$, $GrWr(k)$, and $EndWr(k)$, respectively.

Using Theorem 1 it can be shown that the predicate diagram in Figure 2 conform to the specification in Figure 1. For example, we have:

- $Init \rightarrow |ar| = 0 \wedge |wr| = 0 \wedge |aw| = 0|ww| = 0$
- $|ar| = 0 \wedge wr| = 0 \wedge aw| = 0 \wedge |ww| = 0 \wedge [\exists k \in M : Next(k)]_v \rightarrow$
  $|ar| = 0 \wedge |wr| = 0 \wedge |aw| = 0 \wedge ww| = 0 \vee$
  $\langle \exists k \in M : ImmRd(k) \rangle_v \wedge |ar| > 0 \wedge |wr| = 0 \wedge |aw| = 0 \wedge |ww| = 0 \vee$
  $\langle \exists k \in M : ImmWr(k) \rangle_v \wedge |ar| = 0 \wedge |wr| = 0 \wedge |aw| = 1|ww| = 0$

The next step is to encode the predicate diagram* in Promela, the input language of SPIN. To do this, six variables are used which are *action*, *node*, *ar*, *wr*, *aw*, and *ww*. *action* and *node* are used to indicate the last action taken and the current node; whereas ar, wr, aw, and ww are used to represent the predicates that hold on every node, for example, if $ar = 1$ then the predicate $|ar| > 0$ holds and if $ar = 0$ then the predicate $|ar| = 0$ holds. $action = 1$ if $ImmRd(k)$ is taken, $action = 2$ if $ReqRd(k)$ is taken and so on.

The properties to be verified are now can be written as $\Box(aw = 0 \vee aw = 1)$, $\Box(aw = 1 \rightarrow ar = 0)$, and $\Box(ar = 1 \rightarrow aw = 0)$. Last, by using SPIN we model-checked the resulted transition system. As result, we concluded that the algorithm satisfies all the properties we want to prove.

5 Related work

Verification of parameterized systems is often done by hand, or with the guidance of a theorem prover [15, 13, 9]. Several methods have been proposed that, to various degrees, automate this verification process. Methods based on manual construction of a process invariant are proposed in [5, 17]. However, as the general problem is undecidable [1], it is not in general possible to obtain a finite-state process invariant. For classes of parameterized systems obeying certain constrains, for example [8, 6] , there exists algorithms for model checking the parameterized systems.

In this work we have restricted to a class of parameterized systems that are interleaving and consist of a finitely, but arbitrarily, discrete components. The parameterized systems are represented as parameterized TLA* specifications. The verification is done deductively and algorithmically by means of diagrams. The diagrams can be viewed as the abstract representation of parameterized systems, i.e. a family of processes is represented in a single diagram. The same spirit

but using different formalism is the work from BAUKUS et.al. [2]. They proposed a method for the verification of universal properties of parameterized networks based on the transformation of an infinite family of systems into a single WS1S [3] transition system and applying abstraction techniques on this system.

6 Conclusion and future work

We have defined a new class of diagrams for the verification of parameterized systems called predicate diagrams*. These diagrams are a modification of predicate diagrams proposed by CANSELL et.al. [4]. The modification has been done, in particular, on the definition related to the actions. Instead of actions, we concentrate only on parameterized actions.

Using the reader-writer algorithm as case study, it can be shown that predicate diagrams* can be used to verify the properties of parameterized systems.

In this work, we restrict on the parameterized systems which are *interleaving* and consist of finitely, but arbitrarily, *discrete* components. It is planned to investigate the applicability of predicate diagrams* in the verification of parameterized systems which are not interleaving.

References

1. K. Apt and D. kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, Vol. 15, pp. 307-309. 1986.
2. Kai Baukus, Saddek Bensalem, Yassine Lakhnech and Karsten Stahl. Abstracting WS1S Systems to Verify Parameterized Networks. In *Proceeding of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, Volume 1785 of *Lecture Notes in Computer Science*, pages 188-203. Springer, 2000.
3. J.R. Büchi. Weak second-order arithmatic and finite automata. Z. Math. Logik Grundl. Math., 6:66-92, 1960.
4. Dominique Cansell, Dominique Méry and Stephan Merz. Predicate diagrams for the verification of reactive systems. In 2^{nd} *Intl. Conf. on Integrated Formal Methods (IFM 2000)*, vol. 1945 of *Lectures Notes in Computer Science*, Dagstuhl, Germany, November 2000. Springer-Verlag.
5. E.M. Clarke and O. Grumberg. Avoiding the state explosion problem in temporal logic model checking. Proceedings of the 6th annual ACM Symposium on Principles of Distributed Computing, pp. 294-303. Columbia, Canada, August 1987.
6. E.A. Emerson and K.S. Namjoshi. Automatic verification of parameterized synchronous systems. In Proceeding of 8th Conference on Computer Aided Verification. Vol. 1102 of Lecture Notes in Computer Science, pp. 87-98. Springer, 1996.
7. 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.

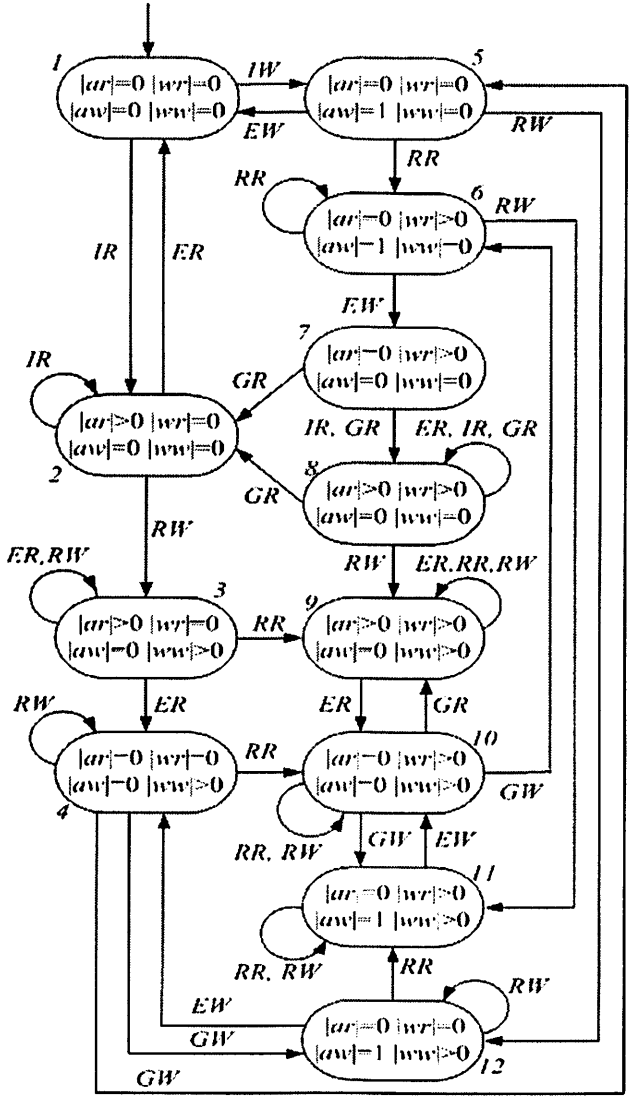

Fig. 2. predicate diagram for the reader-writer algorithm.

8. S. German and A.P. Sistla. Reasoning about systems with many processes. Journal of the ACM, Vol. 39, Number 3, July 1992.
9. K. Havelund and N. Shankar. Experiments in theorem proving and model checking for protocol verification. FME. Vol. 1051 of Lecture Notes in Computer Science, pp. 662-681. Springer, 1996.
10. Hermann Hellwagner. Scalable readers/writers synchronization on shared-memory machines. Technical report, Siemens AG, ZFE ST SN 2, 1993.
11. G. Holzmann. The SPIN model checker. *IEEE Trans. on software engineering*, 16(5):1512-1542. May 1997.
12. Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3) : 872-923, May 1994.
13. Zohar Manna and Amir Pnueli. Verification of parameterized programs. In *Specification and Validation Methods (E. Borger, ed.)*, Oxford University Press, pp. 167-230, 1994.
14. Stephan Merz. Logic-based analysis of reactive systems: hiding, composition and abstraction. Habilitationsschrift. Institut für Informatik. Ludwig-Maximillians-Universität, Munich Germany. December 2001.
15. J. Misra and K.M. Chandy. Parallel program design: a foundation. Addison-Wesley Publishers, 1988.
16. Cecilia E. Nugraheni. Predicate diagrams as basis for the verification of reactive systems. PhD Thesis. Institut für Informatik. Ludwig-Maximillians-Universität, Munich Germany. February 2004.
17. P. Wolper and V. Lovinfose. Verifying properties of large sets of processed with network invariants. In J. Sifakis (ed), Automatic Verification Methods for Finite State Systems. Vol. 47 of Lecture Notes in Computer Science, pp. 68-80. Springer-Verlag, 1990.