Parametric Computation Tree Logic with Knowledge

A. V. Jones[1], A. Lomuscio[1], M. Knapik[2,3], and W. Penczek[2,4]

[1] Department of Computing, Imperial College London, UK
{andrewj,alessio}@ic.ac.uk
[2] Institute of Computer Science, PAS, Ordona 21, 01-237 Warszawa, Poland
{Michal.Knapik,penczek}@ipipan.waw.pl
[3] Systems Research Institute, PAS, Newelska 6, 01-447 Warszawa, Poland
International PhD Projects in Intelligent Computing
[4] University of Natural Sciences and Humanities,
ICS, 3 Maja 54, 08-110 Siedlce, Poland

Abstract. We present a certain extension of temporal-epistemic logic CTLK, allowing for specification of parametric group knowledge properties. These properties allow for free variables, and we introduce a technique for synthesizing all the possible variable substitutions which make the given formula hold in a model. We present the relevant theory, and demonstrate the efficiency and practical usefulness of our approach by performing the parametric analysis of several properties of the IEEE 802.5 token ring LAN protocol with some faults injected.

1 Introduction

Multi-Agent systems (MAS) are distributed systems in which the components, or agents, interact with one another trying to reach private or common goals. One of the recent topics of interest in this area is the issue of verification and validation of MAS, i.e., how to ascertain whether a given MAS satisfies certain specifications of interest. In this context a number of model checkers [7,10,9] have been developed to verify logics for MAS, including epistemic logics, deontic logics, and ATL.

Of particular interest to the community is work on automated model checking tailored to temporal-epistemic specifications. In this line specifications of MAS are defined on temporal languages augmented with modalities to reason about the knowledge of the agents in the system. For example, in a MAS context one may be interested to check whether a particular agent will always know the position of the other agents in the system. A key interest is being able to verify that particular epistemic properties pertaining to groups of agents are valid. As an example of this, most coordination protocols require common knowledge to be obtained within the group of agents before the protocol can be executed by the MAS [5]. Common knowledge (and other group modalities such as distributed knowledge [6]) are expressed in temporal-epistemic logic by using indexes representing the groups they refer to; e.g., $C_\Gamma\phi$ represents the fact that all agents in the group Γ

CONCURRENCY, SPECIFICATION AND PROGRAMMING
M. Szczuka et al. (eds.): Proceedings of the international workshop CS&P 2011
September 28-30, Pułtusk, Poland, pp. 286-300

have common knowledge of ϕ. Model checkers such as MCMAS already support specifications with group operators including common knowledge and can be used to check this kind of properties in a system.

There are scenarios, however, when checking common knowledge in a run is not sufficient. For example, in a MAS that implements distributed diagnosis we, as specifiers, would actually like to know which group in the system obtains distributed knowledge of a particular fault in the system. Moreover, even if we have an intuition as to whether a particular group reaches common or distributed knowledge of a particular property, it is of interest to ascertain whether there is a maximal or minimal group that obtains this so that, for instance, we can minimise the number of agents involved in a coordination protocol.

If the set of agents is finite, we can solve this problem by repeatedly querying a model checker with all the possible instantiations of the specification of interest for all possible groups. However, if this set is large, its power set will not be of a trivial size, resulting in many checks to be carried out. If the specification of interest involves several, not necessarily equal groups, the number of formulae to be checked grows rapidly. In symbolic model checking the bottleneck is normally the computation of the set of reachable states, but if the number of formulae to be checked is sufficiently high, the labelling for the formulae can become the most time consuming operation.

The aim of the paper is to explore alternative, potentially more efficient techniques to identify (or *synthesise*) the groups of agents for which a given temporal-epistemic specification holds. We call this *parametric model checking for temporal-epistemic logic* due to its clear correspondence to parametric model checking of temporal specifications [1,12] where, for instance, temporal intervals are synthesised. In a nutshell, in our approach groups on which the synthesis is carried out are treated as variables ranging on the subsets of the set of agents. The model checking algorithm we put forward returns which subsets validate a given formula.

The rest of the paper is as follows. Syntax and semantics of PCTLK logic is presented in the next section. Section 3 presents labelling algorithms for verification of parametric formulae, and Section 4 presents an evaluation of experimental results on diagnosability properties for a commonly used network protocol.

2 Parametric CTLK with Knowledge

The logic introduced in this section allows for specification of parametric properties – i.e., formulae with free variables. The validity of a given property is decided not only with respect to a model, but also with respect to a valuation of these variables. The aim of parametric model checking is to synthesize the set of all the valuations which make the given formula hold in the initial state of a fixed model.

Definition 1 (PCTLK **syntax).** *Let $\mathcal{PV}$ be a set of propositions,* Groups *be a finite set of variables, and* Agents $= \{1, \ldots, n\}$ *be a finite set of agents. The set*

*of well-formed Parametric Computation Tree Logic with Knowledge (*PCTLK*) formulae is defined as the smallest set satisfying the following conditions:*

- $\mathcal{PV} \subseteq$ PCTLK,
- *if* ϕ, ψ *are formulae of* PCTLK *then:*
  - $\neg\phi$, $\phi \vee \psi$,
  - $EX\phi$, $EG\phi$, $E\phi U\psi$,
  - $K_i\phi$ *for all* $i \in$ Agents,
  - $E_\Gamma\phi$, $D_\Gamma\phi$, $C_\Gamma\phi$, *where* $\Gamma \subseteq$ Agents, $\Gamma \neq \emptyset$, *and*
  - $K_Y\phi$, $E_Y\phi$, $D_Y\phi$, $C_Y\phi$ *for all* $Y \in$ Groups

  are formulae of PCTLK *as well.*

The elements of Groups are called *group variables*, and the formulae containing group variables are called *parametric*.

Recall that EX, EG, EU are temporal modalities, where E stands for "there is a path", and X, G, U mean "in the next state", "for all the states", and "until", respectively. The modalities K_i, E_Γ, D_Γ, C_Γ have epistemic meaning: K_i stands for "agent i knows", E_Γ stands for "everyone in group Γ knows", and D_Γ (C_Γ) stands for "group Γ has distributed (common, resp.) knowledge". The set of the nonparametric formulae of PCTLK is called Computation Tree Logic with Knowledge (CTLK).

Intuitively, PCTLK extends the set of the formulae of CTLK logic (i.e., the nonparametric formulae) by allowing free variables in epistemic modalities. As to give an example, consider the formula $D_Y(AG(sensorsOnline))$ of PCTLK, where Y is a group variable. Our aim is to characterize all the possible substitutions of Y which make the formula true in a given model. This may be perceived as finding an answer to the question "what groups Y of agents have a distributed knowledge that sensors are online in each run of the system?".

We interpret the formulae of PCTLK in Kripke structures, extended with the set of indistinguishability relations over the states.

Definition 2 (Kripke models for knowledge).
Let S denote a set of states, $s^0 \in S$, let $Agents = \{1, \ldots, n\}$ be a set of agents, and $T \subseteq S \times S$ be a temporal transition relation (assumed to be total). For each $i \in Agents$ let $\sim_i \subseteq S \times S$ denote an equivalence relation over S. Let $\mathcal{PV}$ be a set of propositions, and $\mathcal{L} : S \to 2^{\mathcal{PV}} \cup \{true\}$ be such a function that $true \in \mathcal{L}(s)$ for all $s \in S$. The tuple $M = (S, s^0, T, Agents, \{\sim_i\}_{i \in Agents}, \mathcal{L})$ is called a model, *and s^0 its* initial state.

If $s \sim_i s'$ for some states s, s' and agent i, we say that i does not distinguish between s and s'. In addition to this type of knowledge, we define knowledge with respect to a group of agents [5] as follows.

Definition 3 (Three types of group knowledge).
Let $M = (S, s^0, T, Agents, \{\sim_i\}_{i \in Agents}, \mathcal{L})$ be a model and $\Gamma \subseteq Agents$, where $\Gamma \neq \emptyset$. We define three distinct types of group knowledge with respect to Γ by means of the following relations:

1. *everybody knows:* $\sim_\Gamma^E = \bigcup_{i\in\Gamma} \sim_i$,
2. *distributed knowledge:* $\sim_\Gamma^D = \bigcap_{i\in\Gamma} \sim_i$,
3. *common knowledge:* $\sim_\Gamma^C = \left(\bigcup_{i\in\Gamma} \sim_i\right)^+$,

where $^+$ *denotes the transitive closure.*

In order to give the meaning to the temporal modalities, we introduce the notion of a *path*, i.e., a sequence $\pi = (s_0, s_1, \ldots)$ such that $s_i \in S$ and $(s_i, s_{i+1}) \in T$ for all $i \geq 0$. We denote $\pi(j) = s_j$ for all $j \geq 0$, and if for some $s \in S$ there exists a path π such that $\pi(0) = s^0$ and $\pi(j) = s$ for some $j \geq 0$ then we say that s is *reachable*.

We interpret parametric modalities with respect to the valuation of group variables, i.e., such a function υ : Groups $\to 2^{\text{Agents}}$ that $\upsilon(Y) \neq \emptyset$ for each $Y \in$ Groups. The set of all the valuations of the group variables is denoted by GroupVals.

Now we are in the position to introduce the semantics of our logic. If ϕ is a formula of PCTLK, then by $M, s \models_\upsilon \phi$ we denote that ϕ holds in the state s of the model M given the valuation υ. We omit the model symbol where it does not lead to ambiguity.

Definition 4 (PCTLK **semantics).**
Let $M = (S, s^0, T, \text{Agents}, \{\sim_i\}_{i\in\text{Agents}}, \mathcal{L})$ *be such a model that all the states of* S *are reachable from* s^0*, and* υ *be a valuation of the group variables. The relation* $\models_\upsilon$ *is defined recursively as follows:*

- $s \models_\upsilon p$ *iff* $p \in \mathcal{L}(s)$ *for* $p \in \mathcal{PV}$,
- $s \models_\upsilon \neg\phi$ *iff* $s \not\models_\upsilon \phi$,
- $s \models_\upsilon \phi \vee \psi$ *iff* $s \models_\upsilon \phi$ *or* $s \models_\upsilon \psi$,
- $s \models_\upsilon EX\phi$ *iff there exists a path* π *starting at* s*, such that* $\pi(1) \models_\upsilon \phi$,
- $s \models_\upsilon EG\phi$ *iff there exists a path* π *starting at* s*, such that* $\pi(i) \models_\upsilon \phi$ *for all* $i \geq 0$,
- $s \models_\upsilon E\psi U\phi$ *iff there exists a path* π *starting at* s*, such that* $\pi(j) \models_\upsilon \phi$ *for some* $j \geq 0$*, and* $\pi(i) \models_\upsilon \psi$ *for all* $0 \leq i < j$,
- $s \models_\upsilon K_i\phi$ *iff for all* $s' \in S$ *if* $s \sim_i s'$*, then* $s' \models_\upsilon \phi$,
- $s \models_\upsilon Z_\Gamma\phi$ *iff for all* $s' \in S$ *if* $s \sim_\Gamma^Z s'$ *then* $s' \models_\upsilon \phi$*, where* $Z \in \{E, D, C\}$,
- $s \models_\upsilon K_Y\phi$ *iff* $|\upsilon(Y)| = 1$*, and* $s \models_\upsilon K_{\upsilon(Y)}\phi$,
- $s \models_\upsilon Z_Y\phi$ *iff* $s \models_\upsilon Z_{\upsilon(Y)}\phi$*, where* $Z \in \{E, D, C\}$.

We say that ϕ *holds in a model* M *under valuation* υ *(*$M \models_\upsilon \phi$*) if* ϕ *holds in the initial state* s^0.

Note that if ϕ is a nonparametric formula (i.e., a CTLK formula), then $M, s \models_\upsilon \phi$ does not depend on the choice of υ.

3 Model Checking Parametric CTLK

Throughout this section $M = (S, s^0, T, \mathit{Agents}, \{\sim_i\}_{i\in\mathit{Agents}}, \mathcal{L})$ is assumed to be a fixed model and therefore we omit M from the signatures of the presented algorithms.

We formulate the task of parametric verification as follows.

For a given model M and a formula $\phi \in$ PCTLK find all the valuations v of the group variables such that $M \models_v \phi$.

Let ϕ be a formula of PCTLK. By $f_\phi : S \to 2^{\mathrm{GroupVals}}$ we denote a function such that for all $s \in S$ we have $s \models_v \phi$ iff $v \in f_\phi(s)$. The problem of the parametric verification is equivalent to the problem of an efficient computation of f_ϕ for each formula ϕ of PCTLK.

The output of each of the algorithms we introduce is a function from S to $2^{\mathrm{GroupVals}}$. To be more concise, we use the name of a given algorithm to represent the returned function. For example, by $Synth_C(f_\phi)(s)$ we mean a value assigned to s by the function returned by Algorithm 6 for an input f_ϕ.

Let us present the main algorithm for parameter synthesis.

Algorithm 1 $ParameterSynth(\eta)$

Input: $\eta \in$ PCTLK
Output: $f_\eta \in 2^{\mathrm{GroupVals}^S}$

if $\eta \in \mathcal{PV}$ **then**
  return f_η
end if
if $\eta = \neg\phi$ **then**
  return $Complement(ParameterSynth(\phi))$
end if
if $\eta = \phi \vee \psi$ **then**
  for all $s \in S$ **do**
    $sum(s) := ParameterSynth(\phi)(s) \cup ParameterSynth(\psi)(s)$
  end for
  return sum
end if
if $\eta = Oper\phi$ and $Oper \in \{EX, EG\}$ **then**
  return $SAT_{Oper}(ParameterSynth(\phi))$
end if
if $\eta = E\phi U\psi$ **then**
  return $SAT_{EU}(ParameterSynth(\phi), ParameterSynth(\psi))$
end if
if $\eta = Oper_\Gamma\phi$ and $Oper \in \{E, D, C, K\}$ **then**
  return $SAT_{Oper}(ParameterSynth(\phi), \Gamma)$
end if
if $\eta = Oper_Y\phi$ and $Oper \in \{E, D, C, K\}$ **then**
  return $Synth_{Oper}(ParameterSynth(\phi), Y)$
end if

The purpose of the above algorithm is described by the following theorem.

Theorem 1. *Let η be a formula of* PCTLK. *Then,*

$$s \models_v \eta \text{ iff } v \in ParameterSynth(\eta)(s)$$

for each state $s \in S$.

We postpone the proof of the theorem until the end of this section, after we have presented the details of all the subroutines introduced in Algorithm 1.

3.1 Boolean Operations and Nonparametric Modalities

It is straightforward to see that for each $p \in \mathcal{PV}$ the function f_p satisfies:

$$f_p(s) = \begin{cases} \text{GroupVals} & \text{if } p \in \mathcal{L}(s), \\ \emptyset & \text{otherwise.} \end{cases}$$

Basically, $s \models_\upsilon p$ iff the state s is labelled with p, and in this case any group valuation υ is proper.

In order to compute $f_{\neg\phi}$ when f_ϕ is known we introduce the following algorithm.

Algorithm 2 *Complement*(f_ϕ)

Input: $f_\phi \in 2^{\text{GroupVals}^S}$
Output: $f_{\neg\phi} \in 2^{\text{GroupVals}^S}$
1: **for all** $s \in S$ **do**
2: $\quad g(s) := \text{GroupVals} \setminus f_\phi(s)$
3: **end for**
4: **return** g

Obviously, $s \models_\upsilon \neg\phi$ iff $s \not\models_\upsilon \phi$, which is equivalent to $\upsilon \notin f_\phi(s)$, which in turn is equivalent to $\upsilon \in \text{GroupVals} \setminus f_\phi(s)$.

For each $\phi, \psi \in \text{PCTLK}$ we have that $f_{\phi\vee\psi}(s) = f_\phi(s) \cup f_\psi(s)$ for each $s \in S$. To see this, notice that $s \models_\upsilon \phi \vee \psi$ iff $s \models_\upsilon \phi$ or $s \models_\upsilon \psi$, which is equivalent to $\upsilon \in f_\phi(s)$ or $\upsilon \in f_\psi(s)$.

The function $f_{EX\phi}(s) = \bigcup_{s':(s,s')\in T} f_\phi(s')$ for each $s \in S$. As we have that $f_{EG\phi}(s) = f_\phi(s) \cap f_{EXEG\phi}(s)$ for each $s \in S$, we can compute $f_{EG\phi}$ using the standard fixpoint algorithms for EG computation (see [3,8]). A similar observation that $f_{E\phi U\psi}(s) = f_\psi(s) \cup (f_\phi(s) \cap f_{EXE\phi U\psi}(s))$ allows us to use fixpoint algorithms for EU.

In order to compute $f_{K_i\phi}$, $f_{C_\Gamma\phi}$, $f_{E_\Gamma\phi}$, and $f_{D_\Gamma\phi}$ the methods from [11] can be used with slight changes only.

We can therefore focus our presentation on new, parametric modalities.

3.2 Synthesis of K

In order to find out which individual agents have the knowledge of a given property, we simply select the single-agent groups from the sets of groups returned by $Synth_E$ (described in the next subsection). To this end we define a notion of *single agent selector* $SinAg$ with respect to group variable Y as

$$SinAg_Y = \{\upsilon \in \text{GroupVals} \mid \ |\upsilon(Y)| = 1\}.$$

The algorithm is as follows.

Algorithm 3 $Synth_K(f_\phi, Y)$

Input: $f_\phi \in 2^{\text{GroupVals}^S}$
Output: $f_{K_Y\phi} \in 2^{\text{GroupVals}^S}$
1: $f := Synth_E(f_\phi, Y)$
2: **for all** $s \in S$ **do**
3: $\quad g(s) := f(s) \cap SinAg_Y$
4: **end for**
5: **return** g

3.3 Synthesis of Everybody Knows

Let us move to the case of $f_{E_Y\phi}$. For any two states $s, s' \in S$, and $Y \in$ Groups let us define

$$Link^{\exists}_Y(s, s') = \{\upsilon \in \text{GroupVals} \mid s \sim_i s' \text{ for some } i \in \upsilon(Y)\}.$$

Intuitively, $Link^{\exists}_Y(s, s')$ consists of all such valuations of the group variables that assign to Y a group of agents which contains an agent which perceives s and s' as indistinguishable.

Algorithm 4 $Synth_E(f_\phi, Y)$

Input: $f_\phi \in 2^{\text{GroupVals}^S}$
Output: $f_{E_Y\phi} \in 2^{\text{GroupVals}^S}$
1: g := Complement(f_ϕ)
2: $h := g$
3: **for all** $s \in S$ **do**
4: $\quad g(s) := g(s) \cup \big(f_\phi(s) \cap \bigcup_{s' \in S}(Link^{\exists}_Y(s, s') \cap h(s'))\big)$
5: **end for**
6: **return** Complement(g)

In the above algorithm we first evaluate g as $f_{\neg\phi}$, then (in loop 3–5) as $f_{\neg\phi}(s) \cup \big(f_\phi(s) \cap \bigcup_{s' \in S}(Link^{\exists}_Y(s, s') \cap f_{\neg\phi}(s'))\big)$ for each $s \in S$. Notice that at the end of the 3–5 loop $\upsilon \in g(s)$ iff $\upsilon \in f_{\neg\phi}(s)$ or $\upsilon \in f_\phi(s)$ and there exist $s' \in S$ and $i \in \upsilon(Y)$ such that $s \sim_i s'$ and $\upsilon \in f_{\neg\phi}(s')$. This means that either $s \not\models_\upsilon \phi$ or $s' \not\models_\upsilon \phi$ and $s \sim^E_{\upsilon(Y)} s'$, thus $s \not\models_\upsilon E_Y\phi$. Therefore g is finally evaluated as $f_{\neg E_Y\phi}$, and its complement (i.e., $f_{E_Y\phi}$) is returned.

3.4 Synthesis of Distributed Knowledge

For any two states $s, s' \in S$, and $Y \in$ Groups let us define

$$Link^{\forall}_Y(s, s') = \{\upsilon \in \text{GroupVals} \mid s \sim_i s' \text{ for all } i \in \upsilon(Y)\}.$$

Similarly to the previous case $Link^{\forall}_Y(s, s')$ consists of all such valuations of the group variables that assign to Y a group of agents which contains *only* agents which perceive s and s' as indistinguishable.

Using the above notion we can present the algorithm for computing $f_{D_Y\phi}$ in a form similar to Algorithm 4.

Algorithm 5 $Synth_D(f_\phi, Y)$

Input: $f_\phi \in 2^{\text{GroupVals}^S}$
Output: $f_{D_Y\phi} \in 2^{\text{GroupVals}^S}$
1: g := Complement(f_ϕ)
2: $h := g$
3: **for all** $s \in S$ **do**
4: $\quad g(s) := g(s) \cup \big(f_\phi(s) \cap \bigcup_{s' \in S}(Link_Y^{\forall}(s, s') \cap h(s'))\big)$
5: **end for**
6: **return** Complement(g)

The correctness of the above algorithm can be established by the reasoning analogous to the one below Algorithm 4, with $E_Y\phi$ replaced by $D_Y\phi$, and $s \sim^E_{v(Y)} s'$ substituted by $s \sim^D_{v(Y)} s'$.

3.5 Synthesis of Common Knowledge

The algorithm for synthesis of common knowledge is based on finding the fixpoint, which mimicks the nonparametric case. The algorithm is as follows.

Algorithm 6 $Synth_C(f_\phi, Y)$

Input: $f_\phi \in 2^{\text{GroupVals}^S}$
Output: $f_{C_Y\phi} \in 2^{\text{GroupVals}^S}$
1: g := Complement(f_ϕ)
2: **repeat**
3: $\quad h := g$
4: $\quad$ **for all** $s \in S$ **do**
5: $\quad\quad g(s) := g(s) \cup \big(f_\phi(s) \cap \bigcup_{s' \in S}(Link_Y^{\exists}(s, s') \cap h(s'))\big)$
6: $\quad$ **end for**
7: **until** $(h = g)$
8: **return** Complement(g)

Let us denote $E_Y^0\phi = \phi$, and $E_Y^{i+1}\phi = E_Y(E_Y^i\phi)$ for all $i \geq 0$. In the first line of the above algorithm the function g evaluates as $f_{\neg\phi}$. Then, in the i–th turn of the loop 2–7 the g function evaluates as $f_{\bigvee_{j=0}^i \neg E_Y^j\phi}$ until it reaches the fixpoint in line 7. Recall (see [5]) that $lim_{i\to\infty} \bigwedge_{j=0}^i E_Y^j\phi = C_Y^j\phi$, therefore at this stage g is equal to $f_{\neg C_Y\phi}$, and its complement is returned.

3.6 The proof of Theorem 1

The proof is by the induction on the structure of η. Assume that η is a proposition, and notice that $s \models_\upsilon \eta$ iff $\eta \in \mathcal{L}(s)$ iff $f_\eta(s) = \text{GroupVals}$. Thus in view of $ParameterSynth(\eta) = f_\eta$ we have that $s \models_\upsilon \eta$ implies $\upsilon \in$

$ParameterSynth(\eta)(s)$. If η is a proposition and $\upsilon \in ParameterSynth(\eta)(s)$, then $f_\eta(s) = ParameterSynth(\eta)(s) = \text{GroupVals}$. Therefore by the definition of f_η we have that $\eta \in \mathcal{L}(s)$, thus $s \models_\upsilon \eta$.

If $\eta = \phi \vee \psi$, then by the inductive assumption we have that $s \models_\upsilon \phi \vee \psi$ iff ($s \models_\upsilon \phi$ or $s \models_\upsilon \psi$) iff ($\upsilon \in ParameterSynth(\phi)(s)$ or $\upsilon \in ParameterSynth(\psi)(s)$). The latter is equivalent to $\upsilon \in ParameterSynth(\phi \vee \psi)(s)$.

For the treatment of the nonparametric temporal modalities we refer to [3,8], and for the nonparametric epistemic properties – to [11].

Let us move to the case of $\eta = C_Y\phi$. Additionally, let g_0 denote the evaluation of the function g in the first line of Algorithm 6, and let g_i denote the evaluation of g at the end of the i–th turn of the loop 2–7 of the same algorithm. We need to prove that

$$s \models_\upsilon \bigvee_{j=0}^{i} \neg E_Y^j \phi \text{ iff } \upsilon \in g_i(s) \qquad (\star)$$

for all $s \in S$, and $i \geq 0$.

The proof is by the induction on i. In the base case of $i = 0$ we immediately obtain from the definitions of *Complement* and f_ϕ that $s \models_\upsilon \neg\phi$ iff $\upsilon \in g_0(s)$ for all $s \in S$. For the inductive step, let us fix υ and s. Notice that the 5–th line of the algorithm admits the following equality

$$g_i(s) = g_{i-1}(s) \cup \big(f_\phi(s) \cap \bigcup_{s' \in S} (Link_Y^\exists(s, s') \cap g_{i-1}(s'))\big).$$

If $s \models_\upsilon \neg E_Y^j \phi$ for some $0 \leq j < i$ then we obtain $\upsilon \in g_i(s)$ immediately from the inductive assumption and fact that $g_j(s) \subseteq g_i(s)$. As a special case, notice that if $s \models_\upsilon \neg\phi$ then $\upsilon \in g_i(s)$. We can therefore assume that $s \models_\upsilon E_Y^j \phi$ for all $0 \leq j < i$ (from which $\upsilon \in f_\phi(s)$ follows), and $s \models_\upsilon \neg E_Y^i \phi$. Thus there exist $s' \in S$ and $i \in \upsilon(Y)$ such that $s \sim_i s'$, and $s' \models_\upsilon \neg E_Y^{i-1}\phi$. This means that $\upsilon \in Link_Y^\exists(s, s')$, and $\upsilon \in g_{i-1}(s')$, therefore $\upsilon \in \big(f_\phi(s) \cap Link_Y^\exists(s, s') \cap g_{i-1}(s')\big)$, thus $\upsilon \in g_i(s)$. On the other hand, let $\upsilon \in g_i(s)$. If $\upsilon \in g_j(s)$ for some $0 \leq j < i$, then we obtain $s \models_\upsilon \bigvee_{j=0}^{i} \neg E_Y^j \phi$ immediately from the inductive assumption. We can therefore assume that $\upsilon \notin g_j(s)$ for all $0 \leq j < i$. In particular this means that $\upsilon \notin g_{i-1}(s)$, therefore there exists $s' \in S$ such that $\upsilon \in \big(f_\phi(s) \cap Link_Y^\exists(s, s') \cap g_{i-1}(s')\big)$. This in turn means that $s \models_\upsilon \phi$, $s \sim_{\upsilon(Y)}^E s'$, and $s' \models_\upsilon \neg E_Y^{i-1}\phi$, therefore $s \models_\upsilon \neg E_Y^i \phi$.

The case of $\eta = E_Y\phi$ is now easy to derive from the observation that Algorithm 4 is essentially Algorithm 6 with the loop condition removed. Thus ($\star$) applies here with $i = 1$, and takes form $s \models_\upsilon (\neg\phi \vee \neg E_Y\phi)$ iff $\upsilon \in g(s)$. This equivalence is valid in line 5 of Algorithm 4, therefore (as the algorithm returns the complement of g) the equivalence $s \models_\upsilon (\neg\phi \vee \neg E_Y\phi)$ iff $\upsilon \notin Synth_E(f_\phi, Y)(s)$ holds. From the latter $s \models_\upsilon E_Y\phi$ iff $\upsilon \in Synth_E(f_\phi, Y)(s)$ follows, which concludes the case.

Let us move to the case of $\eta = D_Y\phi$, and notice that $\upsilon \in Synth_D(f_\phi, Y)(s)$ iff $\upsilon \notin g(s)$ – where g is as evaluated at the end of 3-5 loop of Algorithm 5. From the inductive assumption it follows that $g(s)$ consists of all those group valuations υ for which either $s \not\models_\upsilon \phi$ or such that $s \models_\upsilon \phi$ and there exists $s' \in S$ such that $s \sim_i s'$ for all $i \in \upsilon(Y)$, and $s' \not\models_\upsilon \phi$. These are exactly all those group valuations

for which $s \models_{\upsilon} D_Y \phi$ does not hold. In order to complete the proof for this case it suffices to notice that Algorithm 5 returns the complement of g.

The last case of $\eta = K_Y \phi$ is rather straightforward, and it is based on the observation that group knowledge of a single agent is this agent's knowledge.

4 Evaluation

In this section we evaluate the effectiveness of the presented parametric technique and its implementation on the top of the temporal-epistemic module of the MCMAS model checker against a naïve, brute force approach to the same problem. In comparison to the parametric technique presented so far, the naïve approach iteratively checks each possible group assignment for satisfiability. For n agents and m group variables, there exists $(2^n - 1)^m$ unique group assignments.

Along with the parametric extension to MCMAS, we implemented the naïve approach as its own separate branch of MCMAS. Importantly, both implementations use the same parser and syntax, thus allowing for a direct comparison between both tools.

To perform a full comparison between the two approaches, we evaluate the techniques using the industry standard IEEE token ring bus network. In the comparison that follows, we automated the injection of faults into the model, following the approach of [4]. Such faults allow for the automatic analysis of a variant of diagnosability properties. We briefly summarise the network below; for a full description of the model we refer the reader to [4].

The IEEE token ring protocol connects n nodes in a ring topology; data circulates between nodes on the network in a clockwise fashion. Access is granted to nodes on the network in form a token, which is passed from node to node.

An important feature of the token ring network is its ability to handle fault tolerance: should a fault occur on an individual node, that node can be physically disconnected from the ring network, thus restoring the ability for data to flow through the network.

Tokens are issued onto the network from an "active monitor" node. To detect faults, tokens contain a counter field, starting at the maximum time that a token would take to circulate the whole network and counting down to zero. Should a token fail to circulate back to the active monitor within the given time-frame, it can be deduced that a fault has occurred on the network.

We consider an instantiation of the network, where the first node wishes to transmit a data token to the n^{th} node. This means that the data token needs to pass through every single intermediate node on the system.

Using the ISPL models and fault injector of [4], we insert two types of faults: even nodes stop sending tokens and odd nodes stop receiving tokens. Such faults allow for the analysis of diagnosability style faults. The properties verified are shown in Table 1.

To exemplify the properties, φ_2 states, for example: "for which two groups does there exist a future such that the first group possesses common knowledge

that the second group does not have common knowledge that faults have not been injected".

Table 1. Diagnosability Properties for the Token Ring Protocol

Formula	Specification
φ_1	$EF\ C_{Y_1}\ \neg\ (\bigvee_{fault \in Faults} fault\_injected)$
φ_2	$EF\ C_{Y_1}\ \neg\ C_{Y_2}\ \neg\ (\bigvee_{fault \in Faults} fault\_injected)$
φ_3	$E[(C_{Y_1}\ \neg(\bigvee_{fault \in Faults} fault\_injected))\ U$ $(EF\ E_{Y_2}\ D_{Y_3} \neg(\bigvee_{fault \in Faults}(fault\_injected \vee fault\_stopped)))]$

Table 2 shows the empirical comparison between the parametric and naïve approach. The values were collected over three runs, with a timeout of one hour on each run.

Table 2. Comparing MCMAS-PARAMETRIC and MCMAS-NAÏVE

Model		Formula	Time (s)		Memory (KiB)		Group Valuations	
Nodes	States		PARAMETRIC	NAÏVE	PARAMETRIC	NAÏVE	ALL	SAT
4	1,116	φ_1	0.594	0.544	14,016	12,072	15	5
		φ_2	0.752	0.761	25,388	12,300	225	171
		φ_3	0.776	3.117	15,888	12,160	3,375	3,255
6	21,578	φ_1	1.434	1.209	23,684	23,680	63	7
		φ_2	2.321	7.285	55,588	23,680	3,969	3,571
		φ_3	11.368	408.478	40,752	23,680	250,047	232,407
8	336,632	φ_1	3.369	3.107	58,708	39,960	255	9
		φ_2	7.497	193.948	63,496	48,200	65,025	62,803
		φ_3	1,127.170	TIMEOUT†	62,316	TIMEOUT†	16,581,375	15,253.335
10	$7.769 \cdot 10^6$	φ_1	6.236	17.052	58,444	53,756	1,023	11
		φ_2	65.875	TIMEOUT‡	61,788	TIMEOUT‡	1,046,529	1,035,387
		φ_3*	TIMEOUT*				1,070,599,167	TIMEOUT*
12	$1.157 \cdot 10^8$	φ_1	15.556	204.853	66,224	109,280$^\bullet$	4,095	13
		φ_2	1,423.12	TIMEOUT*	71,160	TIMEOUT*	16,769,025	16,715,947
		φ_3*	TIMEOUT*				68,669,157,375	TIMEOUT*

† Calculated 8.35% of all the satisfiable groups within the timeout period.
‡ Calculated 73.92% of all the satisfiable groups within the timeout period.
$^\bullet$ Represents an anomalous result with MCMAS-NAÏVE/CUDD. The memory usage of MCMAS-NAÏVE should, under a correctly functioning system, always be lower than that of MCMAS-PARAMETRIC.
* Calculated 1.82% of all the satisfiable groups within the timeout period.
* Both MCMAS-PARAMETRIC and MCMAS-NAÏVE failed to halt-and-succeed within the one-hour time limit.

Additionally, the table also shows the number of satisfiable group valuations (SAT) versus the total number of all possible group assignments. The ratio between the number of satisfiable and total groups is desirable as no formula presents an easy challenge for either technique. Few (or no) satisfiable groups favours the parametric technique, while if all groups are satisfiable, this would favour the naïve technique.

These results demonstrate that the parametric technique can complete synthesis within the timeout period while the naïve approach does not. It can easily be

seen that the results corroborate the expected: the parametric technique sacrifices memory efficiency[5] for tractability. It should be noted that while the current results present very favourable results for the parametric technique, there may exists models where the naïve technique could complete parametric synthesis but the parametric approach would fail. The reason for this is the memory overhead that parametric verification pays: it may well be possible that, where the parametric technique would exhaust all the available memory, the naïve approach to may still be able to brute-force all the groups as it does not pay this overhead. Such a run-time may still be prohibitive though.

References

1. Véronique Bruyère, Emmanuel Dall'olio, and Jean-François Raskin. Durations and parametric model-checking in timed automata. *ACM Trans. Comput. Logic*, 9:12:1–12:23, April 2008.
2. Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.
3. Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.
4. Jonathan Ezekiel and Alessio Lomuscio. A methodology for automatic diagnosability analysis. In *Proceedings of the 12th international conference on Formal engineering methods and software engineering*, ICFEM'10, pages 549–564, Berlin, Heidelberg, 2010. Springer-Verlag.
5. Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.
6. Ronald Fagin, Joseph Y. Halpern, and Moshe Y. Vardi. What can machines know?: On the properties of knowledge in distributed systems. *J. ACM*, 39:328–376, April 1992.
7. Peter Gammie and Ron van der Meyden. Mck: Model checking the logic of knowledge. In Rajeev Alur and Doron Peled, editors, *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 479–483. Springer, 2004.
8. Michael Huth and Mark Ryan. *Logic in Computer Science: modelling and reasoning about systems (second edition)*. Cambridge University Press, 2004.
9. Magdalena Kacprzak, Wojciech Nabialek, Artur Niewiadomski, Wojciech Penczek, Agata Pólrola, Maciej Szreter, Bozena Wozna, and Andrzej Zbrzezny. Verics 2007 - a model checker for knowledge and real-time. *Fundam. Inform.*, 85(1-4):313–328, 2008.
10. Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. Mcmas: A model checker for the verification of multi-agent systems. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 682–688. Springer, 2009.
11. Franco Raimondi and Alessio Lomuscio. Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. *J. Applied Logic*, 5(2):235–251, 2007.
12. Farn Wang. Timing behavior analysis for real-time systems. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pages 112–, Washington, DC, USA, 1995. IEEE Computer Society.

[5] with the except of one anomalous result

5 Appendix: the Details of Implementation

The implementation of the introduced algorithms is carried out by means of operations on Reduced Ordered Binary Decision Diagrams (ROBDDs). ROBDDs (see [2]) are basically special data structures allowing for an efficient representation and operations on propositional formulae.

Again, we assume that $M = (S, s^0, T, Agents, \{\sim_i\}_{i \in Agents}, \mathcal{L})$ is a fixed model, such that all the states of S are reachable from s_0. Additionally we assume that $Agents = \{1, \ldots, n\}$, and Groups $= \{Y_1, \ldots, Y_k\}$ is a finite set of group variables.

5.1 Group Selectors

For each $Y_i \in$ Groups we define $agent_1^{Y_i}, \ldots, agent_n^{Y_i}$ as propositional variables (called *agent variables*). The propositional formulae built from them are called *group selectors*.

Let $\bar{v}$ be a valuation of all the variables from $\bigcup_{Y_i \in \text{Groups}} \{agent_1^{Y_i}, \ldots, agent_n^{Y_i}\}$ set. We say that $\bar{v}$ *encodes* the group valuation $gval(\bar{v})$ defined as follows:

$$gval(\bar{v})(Y_i) = \{j \in Agents \mid \bar{v}(agent_j^{Y_i}) = true\} \text{ for each } 0 \leq i \leq n.$$

For each group selector α we define $gvals(\alpha) = \{gval(\bar{v}) \mid \bar{v} \models \alpha\}$.

As to give an example, consider a group selector $\alpha = (agent_1^{Y_1} \vee agent_2^{Y_1}) \wedge (agent_1^{Y_2} \vee \neg agent_2^{Y_2})$ over the set of $Agents = \{1, 2\}$. It is straightforward to verify that $gvals(\alpha)$ contains all the group valuations υ such that $\upsilon(Y_2) = \{1\}$, and $\upsilon(Y_1) \subseteq \{1, 2\}$, $\upsilon(Y_1) \neq \emptyset$.

Additionally, it is easy to see that we can define a function $gsel$ which assigns to each $B \subseteq$ GroupVals a group selector $gsel(B)$ which encodes B.

5.2 Representations, Parametric Indistinguishability and Preimage

Each the state of M can be uniquely represented using the conjunction of $k = \lceil log|S| \rceil$ propositional variables or their negations. We actually need two disjoint sets of such variables – primed $X = \{x_1, \ldots, x_k\}$, and nonprimed $X' = \{x'_1, \ldots, x'_k\}$. Let s be a state of M, then by $enc(s)$, $enc'(s)$ we denote such representations using primed or nonprimed variables, respectively. We assume that $enc(s_1) \wedge enc(s_2) = False$ and $enc'(s_1) \wedge enc'(s_2) = False$ for all $s_1, s_2 \in S$ such that $s_1 \neq s_2$. We can easily represent subsets of S using disjunction of encodings of its elements.

With that in mind, recall that our task is to effectively compute the $f_\phi : S \to 2^{\text{GroupVals}}$ function which assigns to each state s such a set of group valuations $f_\phi(s)$ that $s \models_\upsilon \phi$ iff $\upsilon \in f_\phi(s)$. This function can be encoded as:

$$[f_\phi] = \bigvee_{s \in S} enc'(s) \wedge gsel(f_\phi(s)).$$

We encode $Link^{\exists}_{Y_i}$ and $Link^{\forall}_{Y_i}$ for each $Y_i \in$ Groups as follows:

$$\mathrm{IR}^{\exists}_{Y_i} = \bigvee_{s,s' \in S} enc(s) \wedge gsel(Link^{\exists}_{Y_i}(s,s')) \wedge enc'(s'),$$

$$\mathrm{IR}^{\forall}_{Y_i} = \bigvee_{s,s' \in S} enc(s) \wedge gsel(Link^{\forall}_{Y_i}(s,s')) \wedge enc'(s').$$

Intuitively, in $\mathrm{IR}^{\exists}_{Y_i}$ and $\mathrm{IR}^{\forall}_{Y_i}$ we augment each pair of states with a special group selector. In case of $\mathrm{IR}^{\exists}_{Y_i}$ this group selector describes group valuations assigning to the current group variable a set of agents containing at least one which perceives s and s' as indescernible. In case of $\mathrm{IR}^{\forall}_{Y_i}$, all the agents in assigned set of agents see s and s' as indistinguishable.

Similarly as in nonparametric methods, we extensively use the operation of existential preimage, denoted by $pre_{\exists}$. Let $2^{X'}$ denote the set of all Boolean functions with domain X'. The existential preimage of $[f_\phi]$ with respect to binary relation R (i.e. $\mathrm{IR}^{\exists}_{Y_i}$ or $\mathrm{IR}^{\forall}_{Y_i}$) is defined as:

$$pre_{\exists}([f_\phi], R) = \bigvee_{bf \in 2^{X'}} (R \wedge [f_\phi])[x'_1 \leftarrow bf(x'_1), \ldots, x'_k \leftarrow bf(x'_k)].$$

In practice, we perform the above operation by means of the standard *exist* operation on ROBDDs (see [2]). Additionally, we assume that after each $pre_{\exists}$ application, all the nonprimed variables are replaced with their primed counterparts.

5.3 The Implementation of Complement

We start with the presentation of ROBDD implementation of Algorithm 2, i.e. the operation of complement.

Algorithm 7 $BDDComplement([f_\phi])$

1: **return** $\neg[f_\phi] \wedge \bigvee_{bf \in 2^{X'}} [f_\phi][x'_1 \leftarrow bf(x'_1), \ldots, x'_k \leftarrow bf(x'_k)]$

In order to obtain the fact that $[f_{\neg\phi}] = BDDComplement([f_\phi])$ apply the following lemma to $F = [f_\phi]$, where f_i are substituted with state encodings, and g_i are substituted with associated group selectors.

Lemma 1. *Let $F = \bigvee_{i \in A} f_i \wedge g_i$, where A is a finite set of indices, and f_i, g_i are propositional formulae such that:*

- *sets of propositional variables present in f_i, g_i are disjoint,*
- *for all $i, j \in A$ such that $i \neq j$ we have that $\not\models f_i \wedge f_j$.*

Let $F' = \bigvee_{i \in A} f_i$, then $\neg F \wedge F' = \bigvee_{i \in A} f_i \wedge \neg g_i$.

Proof. Let $\upsilon \models \neg F \wedge F'$, then there exists exactly one $i \in A$ such that $\upsilon \models f_i$. On the other hand $\upsilon \not\models f_i \wedge g_i$, thus $\upsilon \models f_i \wedge \neg(f_i \wedge g_i)$. Now it suffices to notice that $f_i \wedge \neg(f_i \wedge g_i) \equiv f_i \wedge \neg g_i$.

Now let $\upsilon \models \bigvee_{i \in A} f_i \wedge \neg g_i$. As previously, there exists exactly one $i \in A$ such that $\upsilon \models f_i \wedge \neg g_i$, and obviously $\upsilon \models F'$. Now it suffices to notice that if we had $\upsilon \models F$ then $\upsilon \models f_i \wedge g_i$ would hold. This is not possible, therefore $\upsilon \models \neg F$, and $\upsilon \models \neg F \wedge F'$. □

5.4 Group and Agent Knowledge Synthesis Implementation

Using the introduced notions, we can present the ROBDD implementation of algorithms 6, 4, and 5 in a concise way as follows.

Algorithm 8 $BDDSynth_C([f_\phi], Y_i)$

1: $g := BDDComplement([f_\phi])$
2: **repeat**
3: $\quad h := g$
4: $\quad g := g \vee pre_\exists(g, \mathrm{IR}^{\exists}_{Y_i})$
5: **until** $(h = g)$
6: **return** $BDDComplement(g)$

Algorithm 9 $BDDSynth_E([f_\phi], Y_i)$

1: $g := BDDComplement([f_\phi])$
2: $g := g \vee pre_\exists(g, \mathrm{IR}^{\exists}_{Y_i})$
3: **return** $BDDComplement(g)$

Algorithm 10 $BDDSynth_D([f_\phi], Y_i)$

1: $g := BDDComplement([f_\phi])$
2: $g := g \vee pre_\exists(g, \mathrm{IR}^{\forall}_{Y_i})$
3: **return** $BDDComplement(g)$

For each group variable Y_i define $SinAg_{Y_i} = \bigvee_{j=1}^{n} \left(agent_j^{Y_i} \wedge \bigwedge_{l \neq j} \neg agent_l^{Y_i} \right)$. Algorithm 3 is then implemented as follows.

Algorithm 11 $BDDSynth_K([f_\phi], Y_i)$

1: **return** $BDDSynth_E([f_\phi], Y_i) \wedge SinAg_{Y_i}$