Epistemic logic for verifying runtime verification communication protocols*

Antonis Achilleos[1], Elli Anastasiadi[1], Adrian Francalanza[2], and Jasmine Xuereb[1,2]

[1] ICE-TCS, Department of Computer Science, Reykjavik University, Iceland
[2] Department of Computer Science, University of Malta, Malta
antonios@ru.is, elli19@ru.is, adrian.francalanza@um.edu.mt, jasmine21@ru.is

Abstract

We build on previous work on monitors for hyperproperties, where we propose a logic for specifying properties over sets of traces that can be monitored with circuit-like non-communicating monitors for violations at runtime. In this paper, we propose an epistemic multi-agent logic framework for proving the correctness of distributed and communicating, runtime verification protocols over hyperproperties. Our protocols use monitors that can communicate and accumulate information. To verify the correctness of such a protocol, we can describe communication with epistemic statements that can be then used to derive a proof in an epistemic logic. We then present an example epistemic proof of correctness for a given communication protocol over a specific property that requires communicating monitors, and therefore is not included in the original fragment. This is a step towards a general epistemic framework for the verification of distributed monitoring systems.

1 Introduction

The field of runtime verification provides methods for checking whether a system satisfies an intended specification at runtime. This runtime analysis is done through a computing device called a *monitor* that observes the current run of a system in the form of a trace and attempts to infer the satisfaction or violation of the specification by the system or its run [1,4,5,7,12,16]. Recent work focuses on monitoring for *hyperproperties*, which are properties of sets of traces, introducing novel monitoring setups that process multiple traces [2,6,11]. A centrepiece in this line of work has been the specification logic Hyper-LTL [8]. Intuitively, Hyper-LTL uses trace variables and allows for quantifying these variables over a set of traces that can represent a set of system runs, or a collection of local executions of different system components. Hyper-LTL can use these trace variables to refer to the satisfaction of propositional variables at specific traces, and thus express relationships between local events.

We use the specification logic Hyper-μHML instead of Hyper-LTL, and we build on previous work on monitorability and monitor synthesis for μHML, which is a reformulation of the μ-calculus, and Hyper-μHML is its extension to hyperproperties [1, 3, 13]. The logic μHML allows for straightforward translations from well-known temporal logics such as LTL, and, at the same time, has an intuitive synthesis for monitors [1, 13]. The current paper extends the work from [3], where the authors give a monitor synthesis from a fragment of Hyper-μHML with good correctness and complexity guarantees. However, just like Hyper-LTL and unlike

*The work reported in this paper is supported by the projects 'Open Problems in the Equational Logic of Processes' (OPEL) (grant 196050-051), 'Mode(l)s of Verification and Monitorability' (MoVeMent) (grant no 217987) of the Icelandic Research Fund, and 'Runtime and Equational Verification of Concurrent Programs' (ReVoCoP) (grant 222021), of the Reykjavik University Research Fund.

the fragment from [3], Hyper-μHML can define dependencies over different traces, which can introduce additional latency when monitoring at runtime. The monitoring framework in [3] kept the processing-at-runtime cost minimal by restricting the type of properties it verified to a fragment of Hyper-μHML that effectively does not allow multiple traces to be referred to in the scope of the same quantifier. Therefore, local monitors do not need to communicate in order to detect violations.

In this work, we consider an extension of the circuit-like monitors from [3] that allows monitors to communicate. We observe that there can be more than one correct way to monitor for a given property, and a monitoring system can be engineered with specific goals, such as to minimise the communication overhead or to preserve certain privacy or robustness properties. Therefore, one needs to consider alternatives to a uniform monitor synthesis, which need to be proven correct. We propose a framework for using epistemic logic to prove the correctness of the communication strategy of distributed monitoring protocols. Then, we give an example of describing communications between monitors with epistemic statements and using these to prove that a monitoring protocol can detect all violations of a specification. Our goal is to extend this framework in future work, so that one can prove the correctness of monitoring systems for more general properties and for more notions of correctness.

2 Preliminaries

2.1 The Specification Logic

We present Hyper-μHML, the logic that we use to specify hyperproperties. Hyper-μHML extends the linear-time interpretation of μHML [14, 15, 17] by allowing quantification over traces. We assume two disjoint, countably infinite sets: a set Π of trace variables and a set V of recursion variables; and a finite set $\textsc{Act}$ of events or actions. We define $\textsc{Act}_\Pi = \{a_\pi \mid a \in \textsc{Act} \text{ and } \pi \in \Pi\}$. A set $A \subseteq \textsc{Act}_\Pi$ is called *consistent* if for all $a_{\pi_1}, b_{\pi_2} \in A$, $a = b$ or $\pi_1 \neq \pi_2$. Events in a consistent set can occur simultaneously on different traces.

Definition 1. *Formulae $\varphi \in$ Hyper-μHML are constructed by the following grammar:*

$$\begin{aligned}
\varphi &::= \exists_\pi \varphi \quad \mid \forall_\pi \varphi \quad \mid \varphi \wedge \varphi \quad \mid \varphi \vee \varphi \quad \mid \psi \\
\psi &::= \texttt{tt} \quad \mid \texttt{ff} \quad \mid [A]\psi \quad \mid \langle A \rangle \psi \quad \mid \psi \wedge \psi \quad \mid \psi \vee \psi \quad \mid \max x.\psi \quad \mid \min x.\psi \quad \mid x,
\end{aligned}$$

where $\pi \in \Pi$, $x \in V$, and $A \subseteq \textsc{Act}_\Pi$ is consistent. When $A = \{a_\pi\}$, we may simply write $[a_\pi]\psi$ or $\langle a_\pi \rangle \psi$ instead of $[A]\psi$ or $\langle A \rangle \psi$.

Semantics. The semantics of Hyper-μHML is given over a finite set of infinite traces T over a finite set of actions $\textsc{Act}$ and it is a natural extension of the linear-time semantics of μHML. We require an environment ρ that maps recursion variables to sets of traces, and an assignment $\tau : \Pi \to T$ of trace variables to traces in T. Let $T^0_\tau = \{a_\pi \mid \tau(\pi) = at \text{ for some } t \in \textsc{Act}^\omega\}$, $T^X_\tau = \{t \mid \tau(\pi) = at \text{ for some } t \in \textsc{Act}^\omega\}$, and let $\tau^+ : \Pi \to T$ be defined such that $\tau^+(\pi) = t$, where $\tau(\pi) = at$. We only give the case for the universal modality here:

$$T, \tau, \rho \models [A]\psi \text{ iff } A \subseteq T^0_\tau \text{ implies } T^X_\tau, \tau^+, \rho \models \psi.$$

We use the standard notation $T \models \varphi$ to denote that the set of traces T satisfies φ (and similarly for $T \not\models \varphi$). The work in [3] demonstrates how to monitor for the fragment Hyper^1-sHML of this logic, which does not allows nested trace quantification, diamonds, disjunctions, or least-fixpoints, using circuit-like monitors.

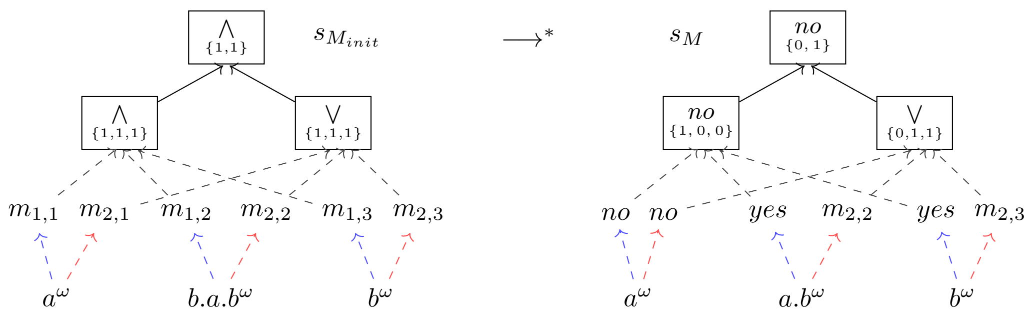
$s_{M_{init}}$
$\bigwedge$ {1,1}
$\bigwedge$ {1,1,1}
$\bigvee$ {1,1,1}
$m_{1,1}$ $m_{2,1}$ $m_{1,2}$ $m_{2,2}$ $m_{1,3}$ $m_{2,3}$
a^ω $b.a.b^\omega$ b^ω
$\longrightarrow^*$
s_M
no {0,1}
no {1,0,0}
$\bigvee$ {0,1,1}
no no yes $m_{2,2}$ yes $m_{2,3}$
a^ω $a.b^\omega$ b^ω

Figure 1: The circuit monitor for the formula in Example 1 over $T = \{a^\omega, b.a.b^\omega, b^\omega\}$.

Example 1. *The Hyper*^{1}-sHML *formula* $\forall_\pi[a_\pi]\mathtt{ff} \wedge \exists_\pi[b_\pi](\max x.([a_\pi]\mathtt{ff} \;\wedge\; [b_\pi]x))$*, over the set of actions* $\{a,b\}$*, states for a set of traces* T*, that no trace starts with* a*, and* $b^\omega \in T$*.*

2.2 The Circuit Monitors Model

In this section, we give the intuition behind the monitor design in [3]. Circuit monitors are composed of a hierarchy of gates, connected in a circuit-like structure and instrumented over a finite set of traces T. Each trace $t \in T$ is assigned a fixed set of regular monitors that correspond to the local properties to be verified and are at the bottom layer of the structure. Monitors assigned to the same trace run in parallel [1] and observe identical events, whereas those assigned to another trace also run in parallel but completely isolated from other traces. When monitors reach a verdict, *yes*, *no* or *end*, they communicate it to the smaller gates connecting them. These then evaluate to some verdict themselves and propagate their evaluation upward through logic gates until the root of the circuit reaches a verdict as well.

Definition 2. *The language* $\textsc{Cmon}_k$ *of* k*-ary monitors, for* $k > 0$*, is given through the following grammar:*

$$\begin{array}{llllll} M \in \textsc{Cmon}_k ::= \bigvee[m]_k & \mid \; \bigwedge[m]_k & \mid \; M \vee M & \mid \; M \wedge M & \\ m ::= \quad yes \mid no \mid end & \mid \; a.m, \;\; a \in \textsc{Act} & \mid \; m + n & \mid \; rec\ x.m & \mid \; x \end{array}$$

$\textsc{Cmon}$ *is the collection of infinite sequences* $(M_i)_{i\in\mathbb{N}}$ *of terms that are generated by substituting* $k = i, \forall i \in \mathbb{N}$*, in a term* M *in* $\textsc{Cmon}_k$*.*

The notation $[m]_k$ corresponds to the parallel dispatch of k identical regular monitors m, where $k = |T|$, with $T = \{t_1, \dots, t_k\}$. The circuit monitor $\bigwedge[m]_k$ evaluates to a *yes* verdict if all sub-monitors evaluate to *yes* verdicts, and a *no* verdict if at least one sub-monitor evaluates to a *no* verdict. Otherwise, if all sub-monitors evaluate to some verdict but none of the previous criteria is met, it evaluates to *end*. The evaluation of $\bigvee[m]_k$ is symmetric, whereas the evaluation of the $\vee$ and $\wedge$ gates over them follows similar rules.

Figure 1 from [3] illustrates the circuit monitor and its evaluation for the formula in Example 1. The notion $m_{i,j}$ signifies that monitor m_i is instrumented with trace j, where monitors m_1 and m_2 respectively monitor for the local properties $\forall_\pi[a_\pi]\mathtt{ff}$ and $\exists_\pi[b_\pi](max\ x.([a_\pi]\mathtt{ff} \wedge [b_\pi]x))$.

Given a formula $\varphi \in$ Hyper^1-sHML and a set of traces T, we can synthesise a circuit monitor M through the recursive function $Syn_T(-)$: Hyper^1-sHML $\to \textsc{Cmon}$ defined in [3].

Proposition 1 (from [3]). *For a formula $\varphi \in$Hyper^1-sHML and a set of traces T, we have that $Syn_T(\varphi)$ is a violation complete monitor for φ over T, in that $Syn_T(\varphi)$ outputs a verdict no if and only if $T \not\models \varphi$.*

2.3 Epistemic Logic

Definition 3 (Multi-agent modal logic). *For a set of agents $\mathcal{A}$, a formula ϕ in the multi-agent modal logic is defined as:*

$$\phi ::= \top \quad | \quad \bot \quad | \quad p \quad | \quad \neg\phi \quad | \quad \phi \wedge \phi \quad | \quad K_i\phi \quad | \quad C_G\phi$$

where p is an atomic formula, $i \in \mathcal{A}$, and $G \subseteq \mathcal{A}$. Implication and disjunction can be defined from the other operators as usual.

We use the standard multi-agent S5 semantics for epistemic formulas with common knowledge, as seen, for example, in [10]. Later on, we also use a natural deduction proof system for multi-agent $S5$. One would need a more intricate logic to fully analyse monitoring frameworks, but as the following section demonstrates, sometimes the above epistemic logic suffices to prove the correctness of a protocol.

3 Epistemic Analysis of Communication Protocols

The fragment Hyper^1-sHML that was introduced in [3] is quite restricted. This allows for a uniform, correct monitor synthesis that does not require the monitors to communicate. In this section, we consider monitoring systems with a communication protocol, which allows us to monitor for more involved properties. In contrast to [3], instead of giving a monitor synthesis for a larger fragment of Hyper-μHML, we focus on proving the correctness of the communications part of a monitoring framework that might have not been produced by an automated synthesis.

3.1 Two Protocols for Two Quantifiers

We consider the example of the following Hyper-μHML formula, which uses two nested quantifiers:

$$\varphi = \forall\pi\forall\pi' max\ x([p_\pi, \overline{p}_{\pi'}]\mathtt{ff} \wedge [p_\pi, p_{\pi'}]x \wedge [\overline{p}_\pi, \overline{p}_{\pi'}]x)$$

Formula φ states that all traces π and π' must agree on all events p. Said otherwise, if p is observed in some trace, then all traces must have p at that time as well. The setup of circuit monitors from Section 2 cannot handle properties similar to this one. More specifically, all local monitors would only be able to observe the value of their own traces regarding p and produce the verdict *end* when asked for the transition $[p_\pi, \overline{p}_{\pi'}]$. However, the latter cannot happen as the transition specified contains at least one step on a trace that the monitors are not instrumented on.

The following monitoring setup would succeed in detecting the violations of φ. If p occurs in some trace, then there are two possible scenarios: *(i)* either all other traces have p as well or *(ii)* at least one trace does not have p. The key point is that in the case of a violation of the property φ, there must be some traces that do not agree on p. Thus, should we design a naive communication protocol where every monitor instantaneously communicates with all other monitors after each event it observes, to inform them about whether or not it observed p, at least a few of the monitors would indeed observe this violation. Naturally, such a protocol would produce a great number of messages while it runs.

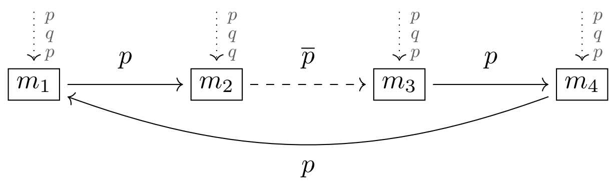

Figure 2: Communication Protocol for formula φ

In order to reduce the significant communication overhead of the above approach, we can use fewer messages and compensate for the lack of information via epistemic reasoning. For instance, consider a protocol where, after observing an event, each monitor communicates with exactly one agent (the one to its right) and receives exactly one message from another agent (the one on its left). Both messages are identical in nature, whereby they inform the receiver whether the sender observed p or not. We refine this further by allowing monitors to communicate only when p is observed since the absence of a message can convey the negation of this statement as shown by the dashed line in Figure 2.

A logician could easily recognise that the above protocol detects a violation of the property described above through the following basic epistemic reasoning. Assume that there are two traces π, π' that do not both have p. If all agents are assigned some order in which they will perform the described communication protocol, there will be two consecutive agents whose values of p do not match, and both of them will be able to deduce that the property is violated. For instance, monitors m_2 and m_3 in Figure 2 detect a violation of φ since the former received p but it observed q, whereas the latter observed p but didn't receive anything, from which it can infer that m_2 didn't observe p.

Remark 1. *In the worst case, two monitors will be able to infer the no verdict, while all the others produce the end verdict. However, this is sufficient for the gate on the higher level to produce the no verdict as well, giving us violation completeness for the specific property φ.*

In what follows, we demonstrate how to prove the correctness of the protocol that we discussed, using epistemic logic.

3.2 Proving Correctness

A first attempt at presenting a correctness proof for the communication protocol discussed above is modelling each monitor as an agent $r \in \mathcal{A}$. The protocol is modelled thought sentences in epistemic logic that are obeyed in each round, where a round is the time during which an event is observed by all agents.

As is described, a monitor (agent r) can observe the occurrence of the event p on trace r (denoted p_r), in which case it has to inform the monitor assigned to the same property on the trace "on its right" about this occurrence. We denote the "next" agent as $r + 1$ *mod* k, where k is the total number of traces. Thus, in a round i we first prove that with the protocol we mentioned it is always the case that a monitor $r + 1$ knows whether p_1 or $\overline{p_r}$. The natural deduction proof for this can be seen in Table 1, given in the appendix due to lack of space.

that a violation of the property φ occurs the behaviour of the monitors will be in accordance with the guarantees provided by the epistemic natural deduction proof given in Table 2. Note that since a violation has occurred it means that there exists traces (and thus monitors r_0, r_1 such that p_{r_0} and $\overline{p_{r_1}}$.

Having established this inference, we use it to prove that when a violation of the property occurs in round i, (i.e., $\exists r_0 \exists r_1 p_{r_0} \wedge \overline{p_{r_1}}$) then there is some agent j that detects the violation. The proof of this inference can be seen in Table 2 in the appendix.

We remark that one can use a similar approach to prove the correctness of monitoring setups for more interesting properties. For example, propositional variables in the epistemic syntax can be used to encode the violations of arbitrary monitorable formulas; the (eventual) detection of such a violation encoded by p by the monitor on trace i can be written as $K_i p$, and its monitorability as $p \rightarrow K_i p$. Then, we can proceed as above.

4 Conclusion and Future Work

In this work we present an initial attempt to incorporate multi-agent epistemic reasoning to the analysis of distributed runtime verification protocols. The key aspect of our approach is to first design a protocol for sharing information and then prove formally that it provides correctness guarantees.

Besides the verification of distributed monitoring setups, one of our aims is to eventually produce a sound *synthesis* algorithm for communication protocols such as the one given in [3] (Proposition 1). However, there are several obstacles that remain to be incorporated into this reasoning framework before reaching this goal. The first shortcoming is the static way in which epistemic logic has been incorporated, which constrains the proofs to be done in a round-by-round fashion as they are currently. We aim to model the exact content of a communication into an *epistemic action* that occurs and has an outcome of the models of a formula. Our approach here would be to incorporate Dynamic epistemic logic [9] so that the temporal aspect of a proof is not introduced externally.

Moreover we have not yet formally extended the monitoring setup to include monitors that can synchronise and produce these communications, and we have not assigned any sort of formal semantics to such a syntactical modification. Thus, to fully perform the upgrade we need also to adapt the implementation to mach the capabilities of the theory.

Finally in order to automate the synthesis of a communicating monitor setup, after having performed the above steps, we want be able to extract from a proof for a certain epistemic theorem into a communication protocol. For example a theorem we would like to test for Hyper-μHML formulae could be formulated as $\neg\varphi \rightarrow \exists i K_i \neg\varphi$. In such a scenario one would want to synthesise a valid monitoring setup from potential tableau proof of this statement. Of course, the semantics of the epistemic operator in the above formula would need to be defined as part a more complex language that will be able to refer to past and future situations.

Finally, our contribution, even though we only present one specific protocol-property pair, can also provide privacy guarantees. Specifically, instead of our protocol, all of the trace observation could be done in a central fashion, or in a distributed one but where every event is fully broadcast to all agents. However, it is easy to see that both such scenario allow for also security breaches, as all information is gathered in one place which can be compromised. Our alternative allows not only for less communication- and thus smaller overhead at runtime- but also minimises the amount of information exchange, which ensures that for example a compromised node will only gather partial information about the system. Thus our effort is a step towards enabling the formal verification of concurrent systems though faster and more secure distributed monitoring mechanisms.

References

[1] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Adventures in monitorability: From branching to linear time and back again. *Proceedings of the ACM on Programming Languages*, 3(POPL):52:1–52:29, 2019.

[2] Shreya Agrawal and Borzoo Bonakdarpour. Runtime Verification of k-Safety Hyperproperties in HyperLTL. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 239–252, 2016.

[3] Elli Anastasiadi, Luca Aceto, Antonis Achilleos, and Adrian Francalanza. Monitoring Hyperproperties with Circuits. In *42nd International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE 2022)*, volume short paper, to appear, 2022.

[4] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. volume 10457 of *Lecture Notes in Computer Science*, pages 1–33. Springer, 2018.

[5] Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory*, pages 162–176, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[6] Borzoo Bonakdarpour and Bernd Finkbeiner. The complexity of monitoring hyperproperties. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 162–174, 2018.

[7] Ian Cassar, Adrian Francalanza, Claudio Antares Mezzina, and Emilio Tuosto. Reliability and fault-tolerance by choreographic design. In *PrePost@iFM*, 2017.

[8] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *2008 21st IEEE Computer Security Foundations Symposium*, pages 51–65, 2008.

[9] Hans van Ditmarsch, Wiebe van der Hoek, and Barteld Kooi. *Dynamic Epistemic Logic*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[10] Ronald Fagin, Joseph Y Halpern, Yoram Moses, and Moshe Vardi. *Reasoning about knowledge*. MIT press, 2004.

[11] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. *Formal Methods Syst. Des.*, 54(3):336–363, 2019.

[12] Adrian Francalanza, Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Ian Cassar, Dario Della Monica, and Anna Ingolfsdottir. A foundation for runtime monitoring. volume 10548, pages 8–29, 09 2017.

[13] Adrian Francalanza, Luca Aceto, and Anna Ingolfsdottir. Monitorability for the Hennessy–Milner logic with recursion. *Formal Methods in System Design*, 51, 08 2017.

[14] Dexter C. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[15] Kim G. Larsen. Proof Systems for Satisfiability in Hennessy-Milner Logic with recursion. *Theoretical Computer Science*, 72(2):265 – 288, 1990.

[16] Claudio Antares Mezzina and Jorge A. Pérez. Causally consistent reversible choreographies: A monitors-as-memories approach. In *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming*, PPDP '17, page 127–138, New York, NY, USA, 2017. Association for Computing Machinery.

[17] Moshe Y. Vardi. A Temporal Fixpoint Calculus. In *POPL*, pages 250–259, New York, NY, USA, 1988. ACM.

Appendix 1 - Natural Deduction Proofs

1	$\forall r : C_{\mathcal{A}}(p_r \leftrightarrow K_r p_r)$	
2	$\forall r : C_{\mathcal{A}}(K_r p_r \rightarrow K_{r+1} p_r)$	
3	$\overline{p_c}$	
4	$K_{c+1} p_c \rightarrow p_c$	$S5$ axiom.
5	$\overline{p_c} \rightarrow \neg K_{c+1} p_c$	prop Thm (4)
6	$\neg K_{c+1} p_c$	$\rightarrow_e$, (3,5)
7	$K_{c+1} \neg K_{c+1} p_c$	$S5$ axiom (6)
8	$\boxed{K_{c+1}}$	
9	$p_c \rightarrow K_c p_c$	C (1)
10	$K_c p_c \rightarrow k_{c+1} p_c$	C (2)
11	$\neg K_{c+1} p_c$	$K_e e$ (7)
12	p_c	assumption.
13	$K_c p_c$	$\rightarrow_e$, (9,12)
14	$K_{c+1} p_c$	$\rightarrow_e$, (10,13)
15	$\perp$	$\neg_e$ (11,14)
16	$\overline{p_c}$	$\neg_i$ (12,15)
17	$K_{c+1}(\overline{p_c})$	$K_c i$ (8,16)

Table 1: Deduction of the non-occurrence of p_c from agent $c+1$

In Tables 1 and 2, lines 1 and 2 are using the quantification not as part of the syntax, but over the number of agents to indicate the existence of k many *real* premises corresponding to the relative line - one for each agent. There we model the communications taking place as part of the protocol though epistemic premises. Line 1 describes that all agents are operating on a distributed monitoring scenario where all agents can know whether p or $\neg p$ on each round, and are aware that this is the protocol applied ot all of them. Line 2 encapsulates the communication of the observance of p in a similar fashion. Assuming on round i we have a violation, we have that $\exists \pi$, and $\exists \pi'$ where due to the argument given above, $\pi' = \pi + 1 \ mod\ k$ such that $p_\pi \wedge \overline{p}_{\pi'}$. Thus we show that there exists some trace c, where the relative monitor (agent) will observe the appropriate events that enable it to deduce $[K_c(p_c \wedge \overline{p_c})]$.

1	$\forall r : C_{\mathcal{A}}(p_r \leftrightarrow K_r p_r)$	
2	$\forall r : C_{\mathcal{A}}(K_r p_r \rightarrow K_{r+1} p_r)$	
3	p_c	
4	$\overline{p_d}$	
5	$K_c p_c$	$\rightarrow_e$ (1,3)
6	$K_{c+1} p_c$	$\rightarrow_e$ (2,4)
7	$K_{c+1} K_{c+1} p_c$	$S5$ axiom (6)
8	$\boxed{c+1}$	
9	$(p_{r_0+1}) \vee \overline{p_{r_0+1}}$	taut.
10	$\overline{p_{c+1}}$	assumption
11	$K_{c+1}(\overline{p_{c+1}})$	$\rightarrow_e$ (1)
12	$K_{c+1} p_c$	$S5$ axiom (6)
13	p_c	$S5$ axiom (12)
14	$p_c \wedge \overline{p_{c+1}}$	$\wedge_i$ (10,13)
15	$K_{c+1}(p_c \wedge \overline{p_{c+1}})$	$S5$ axiom (14)
16	$[K_{c+1}(p_c \wedge \overline{p_{c+1}})] \vee p_{c+1}$	$\vee_i$ (15)
17	p_{c+1}	assumption
18	$[K_{c+1}(p_c \wedge \overline{p_{c+1}})] \vee p_{c+1}$	$\vee_i$ (17)
19	$[K_{c+1}(p_c \wedge \overline{p_{c+1}})] \vee p_{c+1}$	$\vee_e$ (16,18)
20	$\boxed{c+2}$	
21	...	(8-19)
22	$[\bigwedge_{r \in \mathcal{A}} p_r] \vee [\bigvee_{r \in \mathcal{A}} [K_r(p_{r-1} \wedge \overline{p_r})]]$	
23	$[\bigwedge_{r \in \mathcal{A}} p_r]$	assumption
24	p_d	$\wedge_e$ (23)
25	$\bot$	$\neg_e$ (4,24)
26	$\neg([\bigwedge_{r \in \mathcal{A}} p_r])$	$\neg_i$ (23,25)
27	$\bigvee_{r \in \mathcal{A}} [K_r(p_{r-1} \wedge \overline{p_r})]$	prop. Thm (22,26)

Table 2: Epistemic guarantees of correctness of round i