ECAI 2025
I. Lynce et al. (Eds.)

doi:10.3233/FAIA250983

Runtime Verification with Rational Multi-Monitors

Davide Catta[a], Angelo Ferrando[b,*] and Vadim Malvone[c]

[a]Université Sorbonne Paris Nord CNRS, LIPN, F-93430 Villetaneuse, France
[b]University of Modena and Reggio Emilia, Modena, Italy
[c]Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France
ORCID (Davide Catta): https://orcid.org/0000-0001-8656-3274, ORCID (Angelo Ferrando): https://orcid.org/0000-0002-8711-4670, ORCID (Vadim Malvone): https://orcid.org/0000-0001-6138-4229

Abstract. Runtime verification (RV) is a lightweight technique for checking system correctness against formal specifications. Traditional RV assumes full system observability, which rarely holds in distributed and component-based systems where monitors only see partial traces. This leads to inconclusive or incorrect verdicts. To address this, recent work has explored monitors that handle imperfect information and reason about visibility. However, these approaches focus on isolated monitors and overlook coordination in distributed settings. We propose a novel framework for runtime verification with *rational multi-monitors*, where each monitor is a resource-bounded agent with a local specification. Monitors strategically decide what information to share or request, balancing verification goals with communication costs. We formalise this interaction using multi-agent system techniques and synthesise cooperative strategies through model checking. We implement our approach and evaluate it in a case study, showing that rational coordination improves monitoring conclusiveness over existing approaches.

1 Introduction

Consider an exploratory robotic rover, such as NASA's *Mars Curiosity Rover*[1], navigating harsh terrain while collecting samples. The robot is equipped with a complex architecture comprising distributed software modules: a navigation stack for autonomous movement, a robotic arm for sample collection, environmental sensors for detecting dust and temperature, and a communication system for reporting back to Earth. These components interact via middleware such as ROS (Robot Operating System) [39], where software modules (nodes) operate independently and communicate by publishing events over topics. Ensuring safe system behaviour during missions is critical: for instance, the rover must never move while the robotic arm is extended, and it must disable sensitive sensors during high-dust conditions.

This kind of architecture – modular, event-driven, and distributed – is not unique to aerospace systems. Similar patterns are increasingly found in domains such as smart homes, industrial automation, and the Internet of Things (IoT), where autonomous components must coordinate their behaviour without centralised control. These environments face comparable challenges: local components observe only part of the system state, communication is constrained, and correctness must be ensured under resource limitations.

To address these challenges, developers can rely on *Runtime Verification* (RV) [35], a lightweight and efficient technique that checks whether an execution trace satisfies a formal specification, typically expressed in a temporal logic such as LTL [38]. RV complements static approaches like model checking [18, 34, 19], offering scalability and flexibility for dynamic and complex systems.

However, in modular and distributed systems like the above-mentioned ones (e.g., Curiosity Rover, smart home, etc.) the traditional assumption of *global observability*, where a single monitor can access the full system trace, is unrealistic. Components are intentionally isolated to support robustness and scalability. Aggregating all local information into a central monitor introduces latency, bandwidth overhead, and architectural violations. More importantly, it contradicts the design principles of frameworks like ROS or typical IoT platforms, which promote autonomy and decentralisation.

In these settings, monitors operate with only partial visibility of the system trace, leading to *imperfect information* [25]. This creates a fundamental challenge for runtime verification: monitors may issue incorrect verdicts (false positives or negatives) or fail to reach any conclusive judgement. For instance, a monitor verifying that "*the rover must not move while the arm is extended*" might detect movement but lack information about the arm's status, leading to spurious violations. Similar limitations apply to a smart home system where local devices (e.g., lights, doors, motion sensors) must be coordinated without a centralised trace.

Recent work in distributed and decentralised runtime verification has addressed the challenges posed by modularity and limited observability [28]. A common approach is to decompose global properties into local sub-properties, monitored independently by components [15, 20]. These local monitors may exchange verdicts, residual formulas [37], or state summaries to collaboratively assess system-wide behaviour. To handle uncertainty, some frameworks employ three-valued or knowledge-based semantics [7, 8], allowing monitors to issue inconclusive verdicts when local traces are insufficient. Coordination is typically handled through asynchronous protocols or timestamped events that help reconstruct a consistent global view.

Yet, despite these advances, fundamental limitations persist. Recent results show that even four-valued logics like RV-LTL fail to guarantee consistent verdicts across asynchronous monitors for all LTL specifications [14]. These issues stem from the logical structure of certain formulas and the difficulty of synchronising distributed views, especially in the presence of crashes. These findings highlight that consistency cannot be ensured by cooperation alone; it may re-

* Corresponding Author. Email: angelo.ferrando@unimore.it.
[1] https://science.nasa.gov/mission/msl-curiosity/

quire enriched semantics or additional coordination mechanisms.

Moreover, most existing methods assume that monitors are fully cooperative: they share information as prescribed by the monitoring protocol, regardless of cost or necessity. The decision of *what* to share and *when* is typically fixed at design time, rather than reasoned about dynamically based on local knowledge, goals, or constraints.

In this work, we propose a novel perspective: *monitors as rational agents*. Inspired by ideas from Multi-Agent Systems (MAS) [31], we model each monitor as an autonomous entity with: a local view of the system (a projection of the trace), a formal LTL property to verify, and limited resources to acquire or share information.

A *rational monitor* reasons about its uncertainty and decides *whether*, *what*, and from *whom* to request information – subject to resource constraints such as bandwidth or computation. Similarly, it evaluates what own information it is willing to share. Monitors engage in strategic interactions, where information exchange is not free but guided by verification goals and operational costs.

A natural question arises: *why not enforce full cooperation among monitors?* In theory, unrestricted sharing could resolve visibility issues. In practice, however, this is rarely feasible. Communication incurs overhead, and full cooperation may violate modular design principles, overload networks, or expose sensitive internal state[2]. Our approach does not reject cooperation – rather, it frames it as a rational, cost-aware choice made dynamically during execution.

We frame the information-sharing problem as a multi-agent system where each monitor is an agent, and the atomic propositions relevant to each agent stem from its property. We formalise this coordination challenge using *Resource Action-based Bounded Alternating-Time Temporal Logic (RAB-ATL)* [16], which enables reasoning about strategic abilities under resource bounds. Information sharing is treated as an agreement game: monitors compute whether coalitions can achieve their goals by forming agreements, and synthesise such strategies using model checking over a resource-bounded concurrent game structure.

Our main contribution is a formal and implemented framework for *runtime verification with rational multi-monitors*. Each monitor dynamically evaluates the cost-benefit trade-offs of communication, and engages in agreement protocols only when doing so improves its monitoring outcome. We demonstrate the feasibility of our approach through a prototype implementation and a realistic distributed case study in a smart home scenario. Experimental results show that strategic information sharing substantially improves monitoring conclusiveness compared to existing approaches.

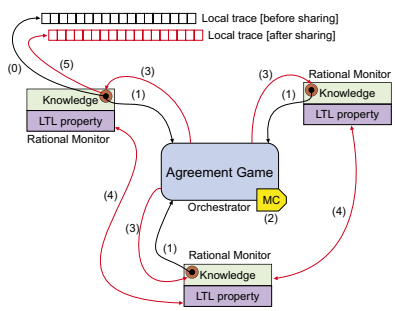

Figure 1: Overview of our approach.

To help the reader visualise our monitoring loop, we refer to Figure 1, which illustrates how local traces, strategic sharing, and monitor verdicts interact. In particular, first the local trace is determined by the agent's knowledge (step 0), then, this knowledge is passed to the agreement game (step 1). At this point, model checking verifies if a solution exists for the agreement game (step 2). In such a case, the agent's knowledge is updated (step 3), the information is exchanged among monitors (step 4), and the local trace is updated based on the agent's new visibility (step 5).

2 Motivating Example

Alice and Bob just bought a smart home [21] equipped with various sensors (temperature, gas, microphone, motion, *etc.*) and actuators (lights, air conditioner, speakers, electric valve, *etc.*). Smart homes can quickly become safety-critical systems. For example, the gas valve should never remain open while the tenants are asleep. To reassure Alice and Bob, the smart home company explains that they use RV [35] to continuously monitor compliance with safety measures.

To perform runtime verification in the smart home, the company deployed monitors directly on specific smart devices equipped with local processing capabilities. In particular, a monitor (M_1) is integrated into an embedded system attached to a video camera installed in the kitchen. This setup enables local video analysis to infer the status of kitchen appliances without streaming raw footage. Another monitor (M_2) is connected to the home's motion and presence sensors through a central smart hub or microcontroller[3]. The third monitor (M_3) is deployed on a similar embedded platform of M_1 co-located with a camera in the laundry area, performing local inference to detect the washing machine's activity and environmental context. Each monitor observes a specific set of events in the system. M_1 has access to the kitchen and can detect whether the coffee machine is making coffee (c), the gas valve is open (g), and the oven is on (o). M_2 monitors motion and presence throughout the home, observing whether the tenants are sleeping (s), in the bathroom (b), or generally present in the house (t). M_3 is responsible for the laundry area and determines whether the washing machine is running (l), is empty (e), and whether it is daytime (d).

Each monitor needs to check two properties. M_1 must check that it is never the case that the gas valve is open while the tenants are sleeping, *i.e.*, $\varphi_1^1 = \mathsf{G}\neg(g \wedge s)$, and that the oven and the washing machine will never be in function at the same time during the day, *i.e.*, $\varphi_2^1 = \mathsf{G}\,(d \rightarrow \neg(l \wedge o))$. M_2 must check that the coffee maker makes coffee as soon as the tenants wake up and go to the bathroom, *i.e.*, $\varphi_1^2 = \mathsf{G}\,((\neg s \wedge b) \rightarrow c)$, and that if the tenants are outside the house, the washing machine will do the laundry, *i.e.*, $\varphi_2^2 = t\,\mathsf{U}\,l$. M_3 must check that when the washing machine is empty, all dirty rags are washed while the oven and the coffee maker are off, *i.e.*, $\varphi_1^3 = \neg e\,\mathsf{U}\,(l \wedge \neg o \wedge \neg c)$, and that the washing machine will never bother the tenants (being in function) while they sleep (in nighttime), *i.e.*, $\varphi_2^3 = \mathsf{G}\,(\neg d \rightarrow \neg(l \wedge s))$.

As the reader can see, each monitor lacks full access to the information needed for verification. For instance, M_3 must request data from M_1 about the oven and coffee maker to verify φ_1^3 and from M_2 about the fact that tenants are sleeping to verify φ_2^3. Furthermore, M_1 must query M_3 for the washing machine's status and the daytime to verify φ_2^1. Similarly, M_2 must query M_3 for the washing machine's status to verify φ_2^2. Finally, M_1 and M_2 must exchange information to verify φ_1^1 and φ_1^2 – M_1 needs to know if the tenants are sleeping, while M_2 requires the coffee maker's status.

In summary, each monitor depends on others to collaborate and share information in order to verify its properties. However, this ex-

[2] Our approach encourages cooperation when it is strategically beneficial under resource constraints.

[3] We abstract the deployment by assuming a single monitor handles all sensor data. In a realistic setting, each sensor would be paired with its own monitor.

change comes with costs: sending messages requires time (for gathering, packaging, and transmission) and consumes network bandwidth. These costs are local to each monitor – unused resources from one cannot be reallocated to another. As a result, monitors must reason carefully about whether, how, and with whom to share information. In other words, they need to behave rationally.

3 Background

If $\rho = x_1, x_2, \ldots$ is a (finite or infinite) sequence, we denote its length as $|\rho|$, and its (j-th) element x_j as ρ_j. For $j \leq |\rho|$, let $\rho_{\geq j}$ be the suffix $\rho_j, \rho_{j+1} \ldots$ of ρ starting at ρ_j and $\rho_{\leq j}$ the prefix $\rho_1, \ldots, \rho_j$ of ρ. The empty sequence will be denoted by ϵ. We use the letters X, Y, Z, etc. to vary over sets, and we denote by $\#(X)$ the cardinality of a given set X. If X is a set of sets $X = \{Y_1, Y_2, \ldots\}$, we write $\bigcup X$ for $Y_1 \cup Y_2 \cup \cdots$. We use $\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$ as variables for tuples of elements of a given set and $\mathbf{a}[i]$ to denote the (i-th) element of such a tuple.

Since our goal is to enhance RV by giving rational power to monitors, we need first to introduce what a monitor is, and how it works. Informally, a monitor is a function that, given a trace of events in input π, returns a verdict which denotes the verification of a formal property φ over the trace. Without loss of generality, if a monitor has multiple objectives, we can express them as a single conjunctive formula. Furthermore, we denote by $[\![\varphi]\!]$ the language of the property, *i.e.*, the set of traces that satisfy φ; namely, $[\![\varphi]\!] = \{\pi \mid \pi \models \varphi\}$.

Definition 1 (Monitor)**.** *Given a set of atomic propositions Ap, a finite trace π over Ap, and an LTL property φ. A monitor for φ is a function $Mon_\varphi : (2^{Ap})^* \to \mathbb{B}_3$, where $\mathbb{B}_3 = \{\top, \bot, ?\}$, such that:*

$$Mon_\varphi(\pi) = \begin{cases} \top & \pi \cdot u \in [\![\varphi]\!],\ \forall u \in (2^{Ap})^\omega \\ \bot & \pi \cdot u \notin [\![\varphi]\!],\ \forall u \in (2^{Ap})^\omega \\ ? & otherwise \end{cases}$$

where $\cdot$ is the standard trace concatenation operator.

Intuitively, a monitor returns $\top$ if all continuations (u) of π satisfy φ; $\bot$ if all possible continuations of π violate φ; ? otherwise.

To handle imperfect information, we consider the same approach presented in [25], where the monitors are built considering the possible lack of information. Specifically, this brings us to a revised definition of monitors, as follows.

Definition 2 (Monitor with imperfect information)**.** *Given a set of atomic propositions Ap, a finite visible[4] trace π_v over Ap, and an LTL property φ. A monitor with imperfect information for φ is a function $Mon^v_\varphi : (2^{Ap})^* \to \mathbb{B}_6$, where $\mathbb{B}_6 = \{\top, \bot, uu, ?_\not\perp, ?_\not\top, ?\}$, as defined in Figure 2.*

Example 1. *Let us consider the motivating example of Section 2 and focus on the M_2 verification of property φ_1^2. Let us assume the global trace produced by the running system is $\pi = \{d, s, e\}\{d, b, c\}\{d, l\}\{d, l, o, c\}$. Thus, the visible trace of events for M_2 is $\pi_v = \{s\}\{b\}\{\}\{\}$. By assuming M_2 is defined as in Definition 1, then it would conclude $\bot$ (i.e., the violation of φ_1^2), since b holds in the second event of π_v, while s and c do not. However, since the trace π_v does not contain c in the second event because of a visibility issue (that is, c is not visible by M_2), it would be wrong to conclude the violation of φ_1^2. Thanks to Definition 2, we may tackle this subtle aspect; indeed, M_2 defined as a monitor with imperfect information would instead return $?_\not\top$ (i.e., we do not know whether we are facing a violation for sure).*

[4] We assume our monitors are applied in distributed systems, they have a local view of the whole system and this affects their visibility over the trace.

Thanks to [16], we have a class of models and formulas that we can use as basis in the multi-monitor setting. We start with the class of models in the following.

Definition 3. *A Resource **Action** Based CGS (RAB-CGS) is a tuple $M = \langle Ap, Ag, S, s_I, \{act_i\}_{i \in Ag}, P, t, L, r, \$\rangle$ s.t.:*

- *Ap is a non-empty set of atomic propositions (denoted by $p, q, r, s \ldots$);*
- *$Ag = \{1, \ldots, n\}$ is a finite set of agents;*
- *S is a non-empty set of states and $s_I \in S$ is the initial state;*
- *for any $i \in Ag$, act_i is a set of actions, $ACT = \Pi_{i \in Ag} act_i$, and $act = \bigcup_{i \in Ag} act_i$;*
- *$P : Ag \times S \to (2^{act} \setminus \emptyset)$ is the protocol function that associates to any agent i and state s a non-empty subset of act_i representing the actions that are available for i at s. We impose that the idle action $\star$ always belongs to $P(i, s)$ for any i;*
- *$t : S \times ACT \to S$ is the transition function, that is given a state s and a tuple of actions $\mathbf{a}$ (where $\forall i$, $\mathbf{a}[i] \in P(i, s)$) such function outputs a state s';*
- *$L : S \to 2^{Ap}$ is the labelling function associating to any state s a set of atomic propositions;*
- *$r \geq 1$ is a natural number (the number of resources types);*
- *$\$: S \times act \times ACT \to \mathbb{N}^r$ is a function mapping state s, action a, and tuple of actions $\mathbf{a} = \langle a_1, \ldots, a_n \rangle$ to a natural number vector of length r. We require a is one of the a_i composing $\mathbf{a}$, and impose that $\$(s, \star, \mathbf{a}) = \mathbf{0}$.*

A **path** ρ is an infinite alternated sequence $s_1, \mathbf{a}_1, s_2, \ldots$ of states and tuples in ACT such that for all $i \geq 1$, $t(s_i, \mathbf{a}_i) = s_{i+1}$. If ρ is a path, we denote by ρ^S the sub-sequence of ρ only containing states. If $h \in S^+$ is a finite sequence of states, we say that h is a **history** iff there is a path ρ such that $h = \rho^S_{\leq i}$ for some $i \in \mathbb{N}$. We use H to denote the set of all histories. A (memoryful) strategy for an agent j is a function $\sigma_j : H \to act_j$, that maps a history to an action $a_j \in act_j$. Let C be a subset of Ag, s a state, and $\mathbf{c}$ be a tuple of actions one for each agent in C. We denote by $Post(s, \mathbf{c})$ the set of states $\{s' \in S \mid t(s, \mathbf{a}) = s' \wedge \mathbf{c}$ is a sub-sequence of $\mathbf{a}\}$. We denote with $P_C(s)$ the set of tuples of actions that are available at s for the coalition C. A path $\rho = s_1, \mathbf{a_1}, s_2 \ldots$ is **compatible** with a strategy σ_j if for every $i \geq 1$ it holds that $\sigma_j(\rho^S_{\leq i}) = \mathbf{a}_i[j]$. A joint strategy σ_C for C is a tuple of strategies: one for each agent in C. A path ρ is compatible with a joint strategy σ_C if it is compatible with any strategy σ_i composing the joint strategy. We denote with $out(s, \sigma_C)$ the set of all σ_C-compatible paths whose first element is s.

In what follows, a bound $\mathbf{b}$ will be any element of $\mathbb{N}^r$. Let s be a state, $\mathbf{a} = \langle a_1, \ldots, a_{\#(Ag)} \rangle \in ACT$ be a tuple of actions, such that for all i, $\mathbf{a}[i] \in P(i, s)$ and $\mathbf{c}$ a sub-sequence of $\mathbf{a}$. The cost of $\mathbf{c}$ in s with respect to $\mathbf{a}$ is given by:

$$cost(s, \mathbf{c}, \mathbf{a}) = \sum_{i=1}^{|\mathbf{c}|} \$(s, \mathbf{c}[i], \mathbf{a})$$

Let σ_C be a strategy for the coalition C, $\rho = s_1, \mathbf{a}_1, s_2, \ldots$ be a path in $out(s, \sigma_C)$, and $\mathbf{b} \in \mathbb{N}^r$. We say ρ is $\mathbf{b}$-consistent when for each natural number $n \geq 1$, we have that:

$$\sum_{k=1}^{n} cost(s_k, \sigma_C(\rho^S_{\leq k}), \mathbf{a}_k) \leq \mathbf{b}$$

$$Mon^{v}_{\varphi}(\pi_v)=\begin{cases}
\top & \pi_v\cdot u\in[\![\varphi]\!]\wedge\pi_v\cdot u\notin[\![\neg\varphi]\!]\wedge\pi_v\cdot u\notin[\![\otimes\varphi]\!],\ \forall u\in(2^{Ap})^{\omega} & \text{[there is no continuation of } \pi_v \text{ which either violates } \varphi \text{ or makes it undefined]}\\
\bot & \pi_v\cdot u\notin[\![\varphi]\!]\wedge\pi_v\cdot u\in[\![\neg\varphi]\!]\wedge\pi_v\cdot u\notin[\![\otimes\varphi]\!],\ \forall u\in(2^{Ap})^{\omega} & \text{[there is no continuation of } \pi_v \text{ which either satisfies } \varphi \text{ or makes it undefined]}\\
uu & \pi_v\cdot u\notin[\![\varphi]\!]\wedge\pi_v\cdot u\notin[\![\neg\varphi]\!]\wedge\pi_v\cdot u\in[\![\otimes\varphi]\!],\ \forall u\in(2^{Ap})^{\omega} & \text{[there is no continuation of } \pi_v \text{ which either satisfies or violates } \varphi]\\
?_{\not\perp} & \pi_v\cdot u'\in[\![\varphi]\!]\wedge\pi_v\cdot u\notin[\![\neg\varphi]\!]\wedge\pi_v\cdot u''\in[\![\otimes\varphi]\!],\ \forall u\in(2^{Ap})^{\omega},\ \exists u',u''\in(2^{Ap})^{\omega} & \text{[every continuation of } \pi_v \text{ either satisfies or makes } \varphi \text{ undefined; none violate it]}\\
?_{\not\top} & \pi_v\cdot u\notin[\![\varphi]\!]\wedge\pi_v\cdot u'\in[\![\neg\varphi]\!]\wedge\pi_v\cdot u''\in[\![\otimes\varphi]\!],\ \forall u\in(2^{Ap})^{\omega},\ \exists u',u''\in(2^{Ap})^{\omega} & \text{[no continuation of } \pi_v \text{ satisfies } \varphi, \text{ but at least one violates or makes } \varphi \text{ undefined]}\\
? & \pi_v\cdot u'\in[\![\varphi]\!]\wedge\pi_v\cdot u''\in[\![\neg\varphi]\!]\wedge\pi_v\cdot u\in[\![\otimes\varphi]\!],\ \exists u,u',u''\in(2^{Ap})^{\omega} & \text{[every continuation of } \pi_v \text{ either satisfies, violates, or makes } \varphi \text{ undefined]}
\end{cases}$$

Figure 2: Monitor with imperfect information, where $[\![\otimes\varphi]\!]$ denotes the traces making φ undefined; *i.e.*, all the traces whose lack of information causes the monitor to not be able to conclude the satisfaction or violation of φ.

A strategy σ_C for a coalition C is $\mathbf{b}$-consistent whenever, for every state s, given any $\rho \in out(s, \sigma_C)$, ρ is $\mathbf{b}$-consistent.

Definition 4. *Formulas of* RAB-ATL *are defined as follows:*

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C^{\mathbf{b}}\rangle\rangle \mathsf{X}\,\phi \mid \langle\langle C^{\mathbf{b}}\rangle\rangle \mathsf{G}\,\phi \mid \langle\langle C^{\mathbf{b}}\rangle\rangle \phi\,\mathsf{U}\,\phi$$

where $p \in Ap$, $C \subseteq Ag$, and $\mathbf{b}$ is any bound. We can derive the boolean connectives $\top$, $\bot$, $\vee$, and $\rightarrow$ as usual, and $\mathsf{F}\phi = \top\,\mathsf{U}\,\phi$. From now on, we use φ, ψ, θ, etc., to denote arbitrary formulas.

Definition 5. *The satisfaction relation of a formula φ from a state s of a RAB-CGS M (notation $M, s \models \varphi$) is defined by induction on φ:*

- $M, s \models p$ *iff* $p \in L(s)$;
- $M, s \models \neg\varphi$ *iff not* $M, s \models \varphi$ *(notation* $M, s \not\models \varphi$*);*
- $M, s \models \varphi \wedge \psi$ *iff* $M, s \models \varphi$ *and* $M, s \models \psi$;
- $M, s \models \langle\langle C^{\mathbf{b}}\rangle\rangle \mathsf{X}\,\varphi$ *iff there is a* $\mathbf{b}$*-consistent strategy* σ_C *for the coalition* C *such that for each path* $\rho \in out(s, \sigma_C)$, $M, \rho^S_2 \models \varphi$;
- $M, s \models \langle\langle C^{\mathbf{b}}\rangle\rangle \mathsf{G}\,\varphi$ *iff there is a* $\mathbf{b}$*-consistent strategy* σ_C *for the coalition* C *such that for each path* $\rho \in out(s, \sigma_C)$*, for all* $i \geq 1$, $M, \rho^S_i \models \varphi$;
- $M, s \models \langle\langle C^{\mathbf{b}}\rangle\rangle \varphi\,\mathsf{U}\,\psi$ *iff there is a* $\mathbf{b}$*-consistent strategy* σ_C *for the coalition* C *such that for each path* $\rho \in out(s, \sigma_C)$*, there is an* $i \geq 1$*, such that* $M, \rho^S_i \models \psi$ *and for all* $1 \leq j < i$ $M, \rho^S_j \models \varphi$.

A formula φ is *true* in a RAB-CGS M, or $M \models \varphi$, iff $M, s_I \models \varphi$. Now, we state the model checking problem and its complexity.

Definition 6. *Given a RAB-CGS M and a* RAB-ATL *formula φ, the model checking problem concerns determining whether $M \models \varphi$.*

Theorem 1 ([16])**.** *Given a RAB-CGS M and a* RAB-ATL *formula φ, determining whether $M \models \varphi$ can be done in polynomial time in the size of M and φ and in exponential time in the number r of resources.*

4 Agreement Games

In a previous work [17], we introduced a simple model of information exchange with n agents, $1, 2, \ldots, n$, where each agent i *privately observes* a set of atomic propositions K_i. Agent i's visibility is determined by K_i, but its size is not constrained to n; rather, each of the n agents has a distinct set K_i. When agents reach an **agreement**, they can share propositions from the above sets via a *communication channel*. For instance, if agent 1 possesses proposition p, she can offer it to agent 2. An exchange occurs if agent 2, in return, offers a proposition q that agent 1 does not possess. The model tracks these exchanges: a state is identified by a set of mutually disjoint agreements, where each agreement is a two-element set containing a proposition privately accessible to one agent and another proposition privately accessible to another. Consider an agent i and the set of atomic propositions K_i to which it has access. To specify which agent in $Ag \setminus \{i\}$ receives a given proposition p from agent i, we index the propositions in K_i by elements of $Ag \setminus \{i\}$, forming the set Σ_i. The intuitive meaning of a proposition $p_j \in \Sigma_i$ is that *agent i offers proposition $p \in K_i$ to agent j.*

Definition 7. *Let $Ag = \{1, \ldots, n\}$ be a finite set of agents and K_i be a non-empty finite set of atomic propositions for any $i \in Ag$. We impose that for all $i, j \in Ag$, $K_i \cap K_j = \emptyset$ whenever $i \neq j$. For any $i \in Ag$ we define:*

$$\Sigma_i = \bigcup_{j\in(Ag\setminus\{i\})} \{p_j \mid p \in K_i\} \tag{1}$$

and we denote with Σ the set $\bigcup_{i\in Ag}\Sigma_i$. We say that a two-elements set $\{x, y\} \subseteq \Sigma$ is a Σ-agreement iff for some j and i in Ag we have that $x \in \Sigma_j$, x is of the form p_i, $y \in \Sigma_i$, y is of the form q_j and $j \neq i$. A set X of Σ-agreements is Σ-balanced iff $X = \emptyset$ or it holds that $Y \cap Z = \emptyset$ for all distinct $Y, Z \in X$. We denote by $\mathcal{A}_\Sigma$ the set of Σ-agreements and by $\mathcal{B}_\Sigma$ the set of balanced sets of Σ-agreements. If X is a set of Σ-agreements, $Z \subseteq \Sigma$, and for all $\{x, y\} \in X$, $x \in Z$, and $y \in Z$, we say that X is a set of Σ-agreements over Z.

The following propositions will be used to assure that the transition function of our agreement games is well-defined.

Proposition 2. *If X and Y are two Σ-balanced sets and either (i) $X \subseteq Y$ or (ii) $Y \subseteq X$ or (iii) for every $U \in X$ and $W \in Y$, $U \cap W = \emptyset$; then $X \cup Y$ is a Σ-balanced set.*

In our agreement games, a game state is identified by a set of mutually disjoint agreements representing the information exchanged so far. Agents transition between states by offering propositions to each other, moving to a state where new exchanges occur. Specifically, an agent i's actions are modelled as propositions in the set Σ_i. To ensure that each action transitions the game from one set of mutually disjoint agreements to another, we have the following conditions.

Proposition 3. *Let $X \in \mathcal{B}_\Sigma$ and $Y \subseteq \Sigma$. Suppose that $Y = \{y_1, \ldots, y_n\}$ and for each $i \leq n$, (i) $y_i \in \Sigma_i$ (each atomic proposition of Y belongs to a different Σ_i) and, (ii) there is no Σ-agreement $Z \in X$ such that $y_i \in Z$; then there is a maximal (with respect to inclusion) Σ-balanced set W whose members are Σ-agreements over Y such that $W \cup X \in \mathcal{B}_\Sigma$.*

Given a set of proposition $P \subseteq \Sigma$, we denote by $P|_i = \{q \mid q \in \Sigma_i\}$. We can now define RAB agreement CGSs.

Definition 8. *Given a RAB-CGS $M = \langle Ap, Ag, S, s_I, \{act_i\}_{i\in AG}, P, t, L, r, \$\rangle$, we say that M is a RAB agreement CGS whenever:*

1. *$Ap = \Sigma$; $Ag = \{1, \ldots, n\}$; $S = \mathcal{B}_\Sigma$ and $s_I \in S$; $act_i = \Sigma_i \cup \{\star\}$;*
2. *$P(i, s) = (\Sigma_i \setminus (\bigcup s)|_i) \cup \{\star\}$. In words: given a state s, an action a_i is either an atomic proposition in Σ_i that does not appear in one of the agreements composing s, or it is the idle action $\star$;*
3. *if $\mathbf{a} = \langle a_1, \ldots, a_n\rangle$ and for all $i \leq n$, $\mathbf{a}[i] \in act_i$, then the transition function is defined by: $t(s, \mathbf{a}) = (s \cup X)$ where X is the maximal (w.r.t. inclusion) balanced set over the set of $\mathbf{a}[i]$ that are propositions such that $s \cup X$ is balanced. Such a set always exists by the fact that the empty-set is Σ-balanced and by Proposition 3. Note that, if $\mathbf{a}$ only contains idle actions, then the definition implies $t(s, \mathbf{a}) = s$;*

4. *$L(s)$ is the identity function on $\bigcup s$;*
5. *given a state $s \in S$, an action $a \in act$, and a tuple of actions $\mathbf{a} \in ACT$, such that $a = \mathbf{a}[j]$ for some j, we have that $\$(s, a, \mathbf{a}) > \mathbf{0}$ iff there is a $k \neq j$ such that $\{a, \mathbf{a}[k]\}$ is a Σ-agreement.*

Condition 5 says that the cost of an action for an agent i in a state s depends upon what other agents do at s. The intuition is that an agent has to pay some amount of resources only if a channel with another agent is open. The opening of this channel depends on the concomitant action of two agents. The agreement game allows autonomous agents to share information only when rewarded. Our approach is general and assumes cooperation only if both agents benefit.

5 Solving the sharing problem for RV

In this section, we present our methodology for handling rational monitors with imperfect information. We first address the information exchange problem using agreement games and then use the results to guide the runtime verification process. The main challenge is linking multi-agent model checking results with the local execution traces verified at runtime by rational monitors. We address this by introducing a time window between agreement games. Specifically, the first game is played before the RV phase, and shared information is retained until the end of the first time window. At this point, the RV phase is performed on the events within the time window. Monitors that achieve their LTL objective terminate, while the others re-enter the agreement game to update their information set. This process repeats for each subsequent time window. Without loss of generality, we assume a fixed sharing cost of 1 per atomic proposition for simplicity. This does not mean each transition has a cost of only 0 or 1; multiple sharing costs can coexist in a single transition. Moreover, generalising to arbitrary costs per proposition does not affect the computational complexity of our approach. Note that the algorithm terminates when no monitors need to share information – either because they have reached a conclusive verdict (*i.e.*, $\top$, $\bot$, or uu) or the trace under evaluation has been fully analysed. Moreover, our approach uses formulas like $\langle\langle A^{\mathbf{b}} \rangle\rangle \psi$ in RAB-ATL, where the bound $\mathbf{b}$ signifies that the coalition A has a strategy to share at most $\mathbf{b}$ atomic propositions in total.

Algorithm 1 RationalMulti-Monitor $\langle M, Ag, \vec{K}, \mathbf{b}, \vec{\varphi}, w \rangle$

1: **create** channels $\vec{in}_{orch}$ **and** $\vec{out}_{orch}$ **of size** $\#(Ag)$
2: **start** $Orchestrator\langle M, Ag, \{\emptyset\}, \vec{K}, \mathbf{b}, \vec{in}_{orch}, \vec{out}_{orch} \rangle$
3: **for** $i \in Ag$ **do**
4: **create** channels $\vec{in}_i$ **and** $\vec{out}_i$ **of size** $\#(Ag)$
5: **start** $RationalMonitor\langle Ag, i, \vec{\varphi}[i], \vec{K}[i], \lfloor \mathbf{b}/\#Ag \rfloor, \vec{in}_i, \vec{out}_i, w \rangle$

We implement our approach using three algorithms: RationalMulti-Monitor (Algorithm 1), RationalMonitor (Algorithm 2), and Orchestrator (Algorithm 3). RationalMulti-Monitor calls each rational monitor using the RationalMonitor (RM) algorithm, while the Orchestrator (OC) algorithm handles their information exchange. Technically, the RationalMulti-Monitor algorithm (Algorithm 1) takes a tuple $\langle M, Ag, \vec{K}, \mathbf{b}, \vec{\varphi}, w \rangle$ as argument, where M is the model of the agreement game[5], Ag is a set of agents (*i.e.*, the rational monitors), $\vec{K}$ is a vector of length $\#(Ag)$ and each component $\vec{K}[j]$ of $\vec{K}$ represents the atomic propositions that are initially known by the agent j, $\mathbf{b}$ denotes a bound, $\vec{\varphi}$ is a vector of length $\#(Ag)$ and each component $\vec{\varphi}[j]$ of $\vec{\varphi}$ represents the runtime LTL objective of the agent j, and w represents the length of the time window (*i.e.*, number of events). In line 1, the Orchestrator's input ($\vec{in}_{orch}$) and output ($\vec{out}_{orch}$) channels are initialised for message exchange with rational monitors. A channel connects a sender and a receiver, with a non-blocking send and a blocking read operation; send takes a target channel and an element – either atomic propositions (Algorithm 2, line 7) or a verdict (line 14), read takes a source channel and the number of elements to retrieve, which can also be atomic propositions (Algorithm 2, line 8) or a verdict (Algorithm 3, line 12). The Orchestrator reads from $\vec{in}_{orch}$ and writes to $\vec{out}_{orch}$. It is launched in line 2. A loop then creates and launches the rational monitors. In line 4, input and output channels are initialised: $\vec{in}_i$ is the input channel for monitor i, and $\vec{out}_i$ is its output channel. In line 5, each monitor i is launched.

[5] M is generated in a setup phase along with the monitors.

Algorithm 2 RationalMonitor $\langle Ag, i, \varphi_i, K_i, \mathbf{b}_i, \vec{in}, \vec{out}, w \rangle$

1: $HoldingAp_i = \emptyset$; $Ap_{\varphi_i} = atom(\varphi_i)$
2: **while true do**
3: $Ap_{G_i} = Ap_{\varphi_i} \setminus \{K_i \cup HoldingAp_i\}$
4: $\vec{exchange} = \emptyset$; $Ap_u = \emptyset$
5: **while** $\vec{exchange} = \emptyset \wedge Ap_u \neq na$ **do**
6: pick a set Ap_u of cardinality $\mathbf{b}_i$ from Ap_{G_i}
7: $send(\vec{out}[Orchestrator], Ap_u)$
8: $\vec{exchange} = read(\vec{in}[Orchestrator], \#(Ag) - 1)$
9: $\pi_i = read(\vec{in}[i], w)$
10: **for** $j \in Ag \setminus \{i\}$ **do** $send(\vec{out}[j], \pi_i \cap \vec{exchange}[j])$
11: **for** $j \in Ag \setminus \{i\}$ **do** $\pi_i = \pi_i \cup read(\vec{in}[j], w)$
12: $Verdict_i = Mon^v_\varphi(\pi_i)$
13: extract $HoldingAp_i$ from $Mon^v_\varphi(\pi_i)$
14: $send(\vec{out}[i], Verdict_i)$; $send(\vec{out}[Orchestrator], Verdict_i)$
15: **if** $Verdict_i \in \{\top, \bot, uu\}$ **or** $empty(\vec{in}[i])$ **then return**

Algorithm 3 Orchestrator $\langle M, Ag, s_I, \vec{K}, \mathbf{b}, \vec{in}, \vec{out} \rangle$

1: **for** $i \in Ag$ **do** $\vec{G}[i] = \emptyset$
2: **while** $Ag \neq \emptyset$ **do**
3: **for** $j \in Ag$ **do** $Ap_u = read(\vec{in}[j], 1)$; $\vec{G}[j] = (\bigwedge_{p \in Ap_u} p)$
4: $Verdict = MC\_\mathsf{RAB} - \mathsf{ATL}(M, s_I, \langle\langle Ag^{\mathbf{b}} \rangle\rangle \mathsf{F}(\bigwedge_{j \in Ag} \vec{G}[j]))$
5: **if** $Verdict == \top$ **then**
6: **for** $j, k \in Ag$ with $j \neq k$ **do** $\vec{exchange}[j][k] = \vec{G}[k] \cap \vec{K}[j]$
7: **else**
8: **for** $j \in Ag$ **do**
9: **for** $k \in Ag$ with $j \neq k$ **do** $\vec{exchange}[j][k] = \emptyset$
10: **for** $j \in Ag$ **do** $send(\vec{out}[j], \vec{exchange}[j])$
11: **for** $j \in Ag$ **do**
12: $\vec{Verdict}[j] = read(\vec{in}[j], 1)$
13: **if** $\vec{Verdict}[j] \in \{\top, \bot, uu\}$ **or** $empty(\vec{in}[j])$ **then** $Ag = Ag \setminus \{j\}$

Rational Monitor. The RationalMonitor algorithm first identifies the atomic propositions needed for verification that are not yet accessible. It then coordinates with the Orchestrator to reach an agreement with other agents. Once an agreement is established, the monitor updates its trace with the new knowledge. Next, it performs the RV step and finally stores the verification result along with any atomic propositions no longer needed for future time windows. The RationalMonitor algorithm, presented in Algorithm 2, takes the tuple $\langle Ag, i, \varphi_i, K_i, \mathbf{b}_i, \vec{in}, \vec{out}, w \rangle$ as input. Here, Ag and w have the same meaning as in the previous algorithm, i is the index of the rational monitor, φ_i is its temporal property of interest, K_i is the set of atomic propositions it can access, and $\mathbf{b}_i$ is the bound on the atomic propositions it can request from other agents. Additionally, $\vec{in}$ is the input channel from which monitor i reads messages, while $\vec{out}$ is the output channel to which it writes messages. Algorithm 2 first extracts the atoms required to verify the property φ_i (line 1), then it starts its iterative execution (loop in lines 2-15). In each iteration, first the rational monitor computes the set of atomic propositions that

are needed to achieve its LTL objective (line 3). Then, in line 4, two elements are initialised; $\vec{exchange}$ is the vector storing the atomic propositions to be exchanged by the monitor in this time window, while Ap_u is the set of atomic propositions selected by the monitor this round according to their payoff in the LTL objective. After that, there is a while loop (lines 5-8), in which it picks a subset of such atoms in accordance with its resource bound $\mathbf{b}_i$. Note that, this set can be picked by using a payoff function for the atomic propositions involved in the LTL objective, similarly to what has been done in [27]. The resulting set is sent to the Orchestrator (line 7) which resolves the agreement game and returns the atomic propositions that the rational monitor has to share in the current time window (line 8). This loop terminates when an agreement amongst the rational monitors is found (*i.e.*, line 8 does not return the empty set), or when no more subsets of atomic propositions can be picked from Ap_{G_i} (*i.e.*, the pick action returns the na symbol, denoting 'not available'). After that, in line 9, the rational monitor reads its local trace. In line 10, rational monitor i transmits the projection of its local trace in accordance to the atomic propositions it consents to share, to each of the other agents. While, in line 11, the rational monitor receives its reward, that is the atomic propositions that it has requested to the Orchestrator for the current time window. These atoms are added to its local trace. Finally, rational monitor performs its verification (line 12). Note that, as in standard RV, the monitor is generated offline before verification; thus, in line 12 the monitor is not synthesised but only applied to the trace. Then, in line 13, the algorithm extracts the atomic propositions that are no longer needed for the current verification of φ_i (these are stored in $HoldingAp_i$). The latter set of atomic propositions is determined at runtime by the monitor verifying the rational monitor's LTL objective. An atomic proposition is removed if it no longer affects the monitor's behaviour. The rational monitor concludes by publishing its verdict (to itself on the left, to the Orchestrator on the right in line 14). The algorithm terminates upon a final verdict ($\top$, $\bot$, or uu) or full trace analysis ($empty(\vec{in}[j])$); otherwise, it starts a new iteration. A rational monitor halts as soon as it completes verification, operating independently with its own objective. Since resources are not shared, it optimally ceases resource use upon completing verification.

Orchestrator. The Orchestrator collects requests, resolves the agreement game, and instructs monitors on which atomic propositions to share. Technically, the Orchestrator algorithm is presented in Algorithm 3 and it takes a tuple $\langle M, Ag, s_I, \vec{K}, \mathbf{b}, \vec{in}, \vec{out}\rangle$ as argument, where M, Ag, $\vec{K}$, and $\mathbf{b}$ have the same meaning as for the RationalMulti-Monitor algorithm, s_I is the initial state of the agreement game and represents a Σ-balanced set, $\vec{in}$ is the input channel from which the Orchestrator reads the messages, and $\vec{out}$ is the output channel to which the Orchestrator writes the messages. In the loop of line 1, the algorithm initialises the goals of the rational monitors. Then, as long as there are rational monitors it iterates in lines 2-13. In this loop, first it gathers the rational monitors' objectives (line 3) which then uses to solve the agreement game by calling the model checking for RAB-ATL (line 4). In case the verification succeeds, the algorithm updates the matrix $\vec{exchange}$ that stores what each rational monitor has to share in the current time window (lines 5-6). If the verification does not succeed, the rational monitors do not have anything to share (lines 8-9). Then, in line 10, the Orchestrator propagates the exchange information to the rational monitors. Finally, it gathers the verdicts produced by the rational monitors and updates the set of active ones by removing the rational monitors that have concluded a final verdict (lines 11-13). Note that, the Orchestrator terminates when there are no more rational monitors in the system.

Now, we provide the complexity of our algorithm.

Theorem 4. *Algorithm 1 is sound. That is, given a global path π, an agent $i \in Ag$, its visible path π_i, and its property φ_i, if $Verdict_i = \mathbf{v}$ then $Mon_{\varphi_i}(\pi) = \mathbf{v}$, where $\mathbf{v} \in \{\top, \bot\}$. Furthermore, it terminates in polynomial time.*

Proof. Suppose that $Verdict_i = \top$. Then $Mon^v_\varphi(\pi_i) = \top$, and we want to prove that $Mon_\varphi(\pi) = \top$. According to Lemma 2 in [25], which establishes the correctness of 6-valued monitors under imperfect information, it suffices to show that π_i is always included in π. This inclusion follows directly from the construction of π_i. Specifically, π_i is initialized during the first iteration of Algorithm 2, at line 9, with the subset of atomic propositions visible to agent i as defined by K_i. In each subsequent iteration, π_i is updated with atomic propositions that are already part of π. Hence, the result follows. Conversely, suppose that $Verdict_i = \bot$. Then $Mon^v_\varphi(\pi_i) = \bot$, and by the same reasoning as above, it follows that $Mon_\varphi(\pi) = \bot$.

The complexity of the RationalMulti-Monitor algorithm depends upon the complexity of the two algorithms that it uses as subroutines. We first examine the complexity of these two algorithms. Concerning Algorithm 2, in line 12 we make LTL verification over a finite trace, that is polynomial in the size of the trace [9]. Notice that the while loop (lines 2-15) terminates, in the worst case, with the end of the trace stored in the input channel. Thus, we can conclude that Algorithm 2 is polynomial in the size of the set of agents and in the length of the trace. The complexity of Algorithm 3 depends upon the complexity of the model-checking algorithm for RAB-CGS. In fact, the loops in lines 1, 3, 6, 8-9, 10, and 11-13 are linear in the cardinality of the set of agents and the while loop (lines 2-13) terminates, in the worst case, with the end of the trace stored in the input channel. In our setting, in which we use r (*i.e.*, the number of resources) equal to 1, the complexity of the model-checking problem for $\mathsf{RAB} - \mathsf{ATL}$ follows by Theorem 1 and it is polynomial in the size of the formula and model under exam. Finally, for Algorithm 1, the worst case scenario is when $window = 1$. In this scenario, we do $|\pi| \times \#(S)$ calls of the model-checking for $\mathsf{RAB} - \mathsf{ATL}$. Since these two algorithms terminates in polynomial time, we conclude that also the RationalMulti-Monitor algorithm terminates in polynomial time. □

Remark 1. *Note that our algorithm is not complete, as it builds upon the six-valued monitor, which has been proven to be incomplete [25].*

Remark 2. *Our approach is synchronous only during the agreement game, where all active monitors and the orchestrator participate in coordinated rounds. Outside this phase, monitors operate independently, observing local traces and performing RV asynchronously. Efficiency is tuned via the time window size, with larger windows for real-time and smaller windows for non-real-time scenarios.*

6 Experiments

We analyse the scenario from Section 2 with experimental results on RV, using VITAMIN [26] for RAB-ATL. From the latter, we infer information sharing and update the local trace accordingly.

Qualitative experiments. Figure 3 compares standard RV, RV with imperfect information [25], and our approach. The global traces analysed, in order, are $\{t, c, e\}\{t, c, s\}\{s, g, l\}$, $\{d, s, e\}\{d, b, c\}\{d, l\}\{d, l, o, c\}$, and $\{s, t\}\{s, l\}\{b\}$. The first trace shows the washing machine running while tenants sleep, violating φ_2^3. The second trace has both the washing machine and oven

Global trace	Local trace M_1	φ_1^1	φ_2^1	Local trace M_2	φ_1^2	φ_2^2	Local trace M_3	φ_1^3	φ_2^3
3-valued monitors									
$\{t,c,e\}\{t,c,s\}\{s,g,l\}$	$\{c\}\{c\}\{g\}$	?	?	$\{t\}\{t,s\}\{s\}$	?	$\bot$	$\{e\}\{\}\{l\}$	$\bot$	?
$\{d,s,e\}\{d,b,c\}\{d,l\}\{d,l,o,c\}$	$\{\}\{c\}\{\}\{o,c\}$	?	?	$\{s\}\{b\}\{\}\{\}$	$\bot$	$\bot$	$\{d,e\}\{d\}\{d,l\}\{d,l\}$	$\bot$	?
$\{s,t\}\{s,l\}\{b\}$	$\{\}\{\}\{\}$	?	?	$\{s,t\}\{s\}\{b\}$	$\bot$	$\bot$	$\{\}\{l\}\{\}$	$\top$	?
6-valued monitors									
$\{t,c,e\}\{t,c,s\}\{s,g,l\}$	$\{c\}\{c\}\{g\}$	$?_{\mathcal{T}}$	?	$\{t\}\{t,s\}\{s\}$	?	uu	$\{e\}\{\}\{l\}$	$\bot$	$?_{\mathcal{T}}$
$\{d,s,e\}\{d,b,c\}\{d,l\}\{d,l,o,c\}$	$\{\}\{c\}\{\}\{o,c\}$	?	$?_{\mathcal{T}}$	$\{s\}\{b\}\{\}\{\}$	$?_{\mathcal{T}}$	uu	$\{d,e\}\{d\}\{d,l\}\{d,l\}$	$\bot$	?
$\{s,t\}\{s,l\}\{b\}$	$\{\}\{\}\{\}$	?	?	$\{s,t\}\{s\}\{b\}$	$?_{\mathcal{T}}$	uu	$\{\}\{l\}\{\}$	?	$?_{\mathcal{T}}$
6-valued monitors with information sharing (*i.e.*, Rational Monitors)									
$\{t,c,e\}\{t,c,s\}\{s,g,l\}$	$\{c\}\{c,s\}\{s,g,l\}$	$\bot$	?	$\{t,c\}\{t,c,s\}\{s,l\}$	?	$\top$	$\{c,e\}\{c,s\}\{s,l\}$	$\bot$	$\bot$
$\{d,s,e\}\{d,b,c\}\{d,l\}\{d,l,o,c\}$	$\{d,s\}\{d,c\}\{d,l\}\{d,l,o,c\}$	?	$\bot$	$\{s\}\{b,c\}\{l\}\{l,c\}$	?	$\bot$	$\{d,s,e\}\{d\}\{d,l\}\{d,l,o,c\}$	$\bot$	?
$\{s,t\}\{s,l\}\{b\}$	$\{s\}\{s,l\}\{\}$	?	?	$\{s,t\}\{s,l\}\{b\}$	$\bot$	$\top$	$\{s\}\{s,l\}\{\}$	$\top$	$\bot$

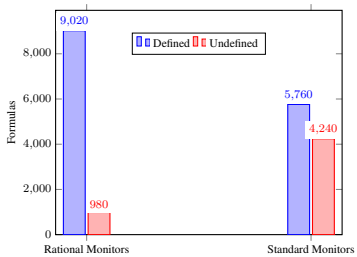

Figure 3: On the left: experiments. On the right: comparison standard and rational monitors.

in use, violating φ_1^3 and φ_2^1. The third trace depicts tenants waking up and going to the bathroom without coffee, violating φ_1^2, and another instance of the washing machine running while they sleep, violating φ_2^3. For each trace, we present its *local trace* based on each monitor's visibility, as described in Section 2. Finally, we include a column for each property, showing the corresponding monitor's verdict. We use three different colours to represent the truth values assigned by each monitor. Specifically, green indicates that the monitor has reached the correct verdict, yellow denotes a less informative but not incorrect verdict, and red signifies that the monitor has reached an incorrect verdict. In the first group, we use standard 3-valued LTL monitors, which introduce errors by treating unknown atomic propositions as false. Consequently, the monitors in Figure 3 misinterpret the local traces. In the second trace, monitor M_2, evaluating φ_1^2, incorrectly concludes $\bot$, assuming c is false, while it was actually true; the correct verdict was ?. Similarly, monitors often lack key information. Monitor M_1 returns ? for φ_2^1 as it cannot observe d and l, though they appear in the global trace – meaning it would have concluded $\bot$. The same issue affects φ_2^3 (first and third trace). We rerun the experiments with 6-valued monitors [25], which account for imperfect information. For instance, in the second trace, φ_1^2, previously misclassified as $\bot$, is now $?_{\mathcal{T}}$, as M_2 acknowledges its lack of information, avoiding the false assumption that c is false. Note that, this case denotes a less informative but not wrong verdict. While 6-valued monitors improve accuracy, they do not actively reduce imperfect information. This motivates our final experiments, where monitors strategically share information. In the third group, 6-valued monitors actively share atomic propositions, unlike the second group, which only recognises imperfect information. Sharing decisions are based on solving the static verification problem, enriching local traces and improving accuracy. In the second trace for φ_1^3, monitor M_3 correctly concludes $\bot$ by obtaining o and c by exchanging d and l with M_1.

Note that, the rational monitors with imperfect information consistently produce the correct verdict, assuming they have sufficient resources to exchange all necessary information. In contrast, the non-rational monitors may fail. Specifically, the 3-valued monitors often arrive at incorrect verdicts, occasionally getting the correct result by chance, while the 6-valued monitors tend to provide less informative verdicts, though they sometimes produce correct ones.

Quantitative experiments. We conducted quantitative tests on 10,000 randomly generated LTL formulas to further evaluate our approach. These random formulas were generated with careful consideration to ensure a diverse distribution of features, including varying levels of operator nesting, coverage of different temporal operators, formula size, the percentage of imperfect information, and more. Figure 3 (right) shows that rational monitors verified 9,020 of 10,000 formulas, with 980 yielding uu. Standard monitors verified 5,760, while 4,240 concluded as uu. Rational monitors verified 90% of LTL formulas, compared to 57% for standard monitors, highlighting the crucial role of information sharing in verification.

7 Related works

Runtime Verification. A survey on decentralised and distributed RV is provided in [28]. The approach in [37] is similar to ours but differs in two key aspects: (i) their monitors verify the same LTL property, whereas ours handle distinct properties; (ii) their monitors fully cooperate and share all information *a priori*. Similarly, [44] examines the events to share, but, unlike our approach, uses a single monitor for verification while distributing the information gathering process. In [3], the verification of distributed systems is approached probabilistically, focussing on communication protocols rather than handling imperfect information. The work in [41] proposes a decentralised verification approach for time-dependent regular expressions. Other relevant contributions include [8] on message failures, [29] using an SMT solver to manage event order uncertainty, and [4] employing Field Calculus. Regarding imperfect information in RV, [33] studies RV with partial system knowledge, encoding specifications as symbolic formulas iteratively refined with new observations. In [24], potential integrations of RV and model checking in MAS are explored, while [25] extends LTL verification to handle imperfect information in a centralised manner. Runtime verification under uncertainty is addressed in [43], where Past-Time LTL is applied to abstract traces sampled over time. Works such as [42, 6, 32, 5, 36] investigate RV on traces with information gaps, proposing methods to fill them. To our knowledge, no existing work treats monitors as agents, assumes non-cooperation, or applies strategic reasoning for information exchange in verification.

Model Checking Multi-Agent Systems. The main development in this field is *Alternating-Time Temporal Logic* (ATL) [2], which allows reasoning about agent strategies with temporal goals. A key aspect in MAS is agents' visibility [40]. ATL model checking becomes undecidable with imperfect information and memoryful strategies [22]. To address this, some works approximate the information [10, 23] or develop new strategy notions [11, 30]. Notably, [12, 13] introduces an information-sharing game where agents share information. Our approach differs in the latter by using perfect information games, with imperfect information present only during runtime verification, and by considering bounded resources using a variant of RB-ATL [1].

8 Conclusions

We introduced rational monitors in a distributed setting, defining formal models where agents cooperate to open communication channels, exchange data, and allocate resources. We used this formalisation to solve information sharing issues among monitors in RV.

Future work includes exploring partial solutions where some agents benefit from the agreement while others do not, enabling agents to unshare variables, and generalising the framework to share information on sub-formulas instead of atomic propositions.

References

[1] N. Alechina and B. Logan. State of the art in logics for verification of resource-bounded multi-agent systems. In *Fields of Logic and Computation III*, volume 12180, pages 9–29, 2020.

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[3] D. Ancona, A. Ferrando, and V. Mascardi. Exploiting probabilistic trace expressions for decentralized runtime verification with gaps. In *CILC*, pages 154–170, 2022.

[4] G. Audrito, F. Damiani, V. Stolz, G. Torta, and M. Viroli. Distributed runtime verification by past-ctl and the field calculus. *J. Syst. Softw.*, 187:111251, 2022.

[5] E. Bartocci and R. Grosu. Monitoring with uncertainty. In *HAS*, volume 124, pages 1–4, 2013.

[6] E. Bartocci, R. Grosu, A. Karmarkar, S. A. Smolka, S. D. Stoller, E. Zadok, and J. Seyster. Adaptive runtime verification. In *RV, Revised Selected Papers*, volume 7687, pages 168–182, 2012.

[7] D. A. Basin, M. Harvan, F. Klaedtke, and E. Zalinescu. Monitoring usage-control policies in distributed systems. In TIME, pages 88–95, 2011.

[8] D. A. Basin, F. Klaedtke, and E. Zalinescu. Failure-aware runtime verification of distributed systems. In *LIPIcs*, pages 590–603, 2015.

[9] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for ltl and tltl. *ACM Trans. Softw. Eng. Methodol.*, 20(4), Sept. 2011.

[10] F. Belardinelli and V. Malvone. A three-valued approach to strategic abilities under imperfect information. In *KR*, pages 89–98, 2020.

[11] F. Belardinelli, A. Lomuscio, and V. Malvone. Approximating perfect recall when model checking strategic abilities. In *KR*, pages 435–444, 2018.

[12] F. Belardinelli, I. Boureanu, C. Dima, and V. Malvone. Verifying strategic abilities in multi-agent systems with private data-sharing. In *AAMAS*, pages 1820–1822, 2019.

[13] F. Belardinelli, I. Boureanu, C. Dima, and V. Malvone. Model-checking Strategic Abilities in Information-sharing Systems. *ACM Trans. Comput. Log.*, 26(1): 5:1–5:45, 2025.

[14] B. Bonakdarpour, P. Fraigniaud, S. Rajsbaum, D. A. Rosenblueth, and C. Travers. Decentralized asynchronous crash-resilient runtime verification. *J. ACM*, 69(5):34:1–34:31, 2022.

[15] I. Cassar and A. Francalanza. On implementing a monitor-oriented programming framework for actor systems. In IFM, pages 176–192, 2016.

[16] D. Catta, A. Ferrando, and V. Malvone. Resource action-based bounded ATL: A new logic for MAS to express a cost over the actions. In *PRIMA*, pages 206–223, 2024.

[17] D. Catta, A. Ferrando, and V. Malvone. Agreement Games in Multi-Agent Systems. In *AAMAS*, pages 2452–2454, 2025

[18] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs, Workshop*, pages 52–71, 1981.

[19] E. M. Clarke, O. Grumberg, and D. A. Peled. Model checking the mit press. *Cambridge, Massachusetts, London, UK*, 988, 1999.

[20] C. Colombo and Y. Falcone. Organising LTL monitors over distributed systems with a global clock. *Formal Methods Syst. Des.*, 49(1-2):109–158, 2016.

[21] J. L. Crowley and J. Coutaz. An ecological view of smart home technologies. In *AmI*, pages 1–16, 2015.

[22] C. Dima and F. L. Tiplea. Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoRR*, abs/1102.4225, 2011.

[23] A. Ferrando and V. Malvone. Towards the verification of strategic properties in multi-agent systems with imperfect information. In *AAMAS*, pages 793–801, 2023.

[24] A. Ferrando and V. Malvone. Give me a hand: How to use model checking for multi-agent systems to help runtime verification and vice versa (short paper). In SPIRIT, 2022.

[25] A. Ferrando and V. Malvone. Runtime verification with imperfect information through indistinguishability relations. In *SEFM*, pages 335–351, 2022.

[26] A. Ferrando and V. Malvone. VITAMIN: A compositional framework for model checking of multi-agent systems. In *ICAART*, pages 648–655, 2025.

[27] A. Ferrando and V. Malvone. Runtime Verification via Rational Monitor with Imperfect Information. *ACM Trans. Softw. Eng. Methodol.*, 2025.

[28] A. Francalanza, J. A. P'erez, and C. S'anchez. Runtime verification for decentralised and distributed systems. In *Lectures on Runtime Verification - Introductory and Advanced Topics*, pages 176–210. Springer, 2018. URL https://doi.org/10.1007/978-3-319-75632-5_6.

[29] R. Ganguly, A. Momtaz, and B. Bonakdarpour. Distributed runtime verification under partial synchrony. In *OPODIS 2020*, pages 20:1–20:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. URL https://doi.org/10.4230/LIPIcs.OPODIS.2020.20.

[30] W. Jamroga, V. Malvone, and A. Murano. Natural strategic ability. *Artif. Intell.*, 277, 2019. URL https://doi.org/10.1016/j.artint.2019.103170.

[31] N. Jennings and M. Wooldridge. *Agent technology: foundations, applications, and markets*. Springer Science & Business Media, 1998.

[32] K. Kalajdzic, E. Bartocci, S. A. Smolka, S. D. Stoller, and R. Grosu. Runtime verification with particle filtering. In *RV 2013*, pages 149–166. Springer. URL https://doi.org/10.1007/978-3-642-40787-1_9.

[33] H. Kallwies, M. Leucker, and C. S'anchez. Symbolic runtime verification for monitoring under uncertainties and assumptions. In *ATVA 2022*, pages 117–134. Springer. URL https://doi.org/10.1007/978-3-031-19992-9_8.

[34] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000. URL https://doi.org/10.1145/333979.333987.

[35] M. Leucker and C. Schallhart. A brief account of runtime verification. *J. Log. Algebraic Methods Program.*, 78(5):293–303, 2009. URL https://doi.org/10.1016/j.jlap.2008.08.004.

[36] M. Leucker, C. Sánchez, T. Scheffel, M. Schmitz, and D. Thoma. Runtime verification for timed event streams with partial information. In *RV 2019*, pages 273–291. Springer. URL https://doi.org/10.1007/978-3-030-32079-9_16.

[37] M. Mostafa and B. Bonakdarpour. Decentralized runtime verification of LTL specifications in distributed systems. In *IPDPS 2015*, pages 494–503. IEEE Computer Society. URL https://doi.org/10.1109/IPDPS.2015.95.

[38] A. Pnueli. The temporal logic of programs. In *SFCS 1977*, pages 46–57. IEEE Computer Society.

[39] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In *(ICRA) Workshop on Open Source Robotics*, 2009.

[40] J. H. Reif. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2):274–301, 1984. URL https://doi.org/10.1016/0022-0000(84)90034-5.

[41] V. Roussanaly and Y. Falcone. Decentralised runtime verification of timed regular expressions. In *TIME 2022*, pages 6:1–6:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL https://doi.org/10.4230/LIPIcs.TIME.2022.6.

[42] S. D. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S. A. Smolka, and E. Zadok. Runtime verification with state estimation. In *RV 2011*, pages 193–207. Springer. URL https://doi.org/10.1007/978-3-642-29860-8_15.

[43] S. Wang, A. Ayoub, O. Sokolsky, and I. Lee. Runtime verification of traces under recording uncertainty. In *RV 2011*, pages 442–456. Springer. URL https://doi.org/10.1007/978-3-642-29860-8_35.

[44] N. Yaseen, B. Arzani, R. Beckett, S. Ciraci, and V. Liu. Aragog: Scalable runtime verification of shardable networked systems. In *USENIX, OSDI 2020*, pages 701–718. USENIX Association. URL https://www.usenix.org/conference/osdi20/presentation/yaseen.