Assumption-Based Runtime Verification with Partial Observability and Resets

Alessandro Cimatti, Chun Tian(✉), and Stefano Tonetta

Fondazione Bruno Kessler, Trento, Italy
{cimatti,ctian,tonettas}@fbk.eu

Abstract. We consider Runtime Verification (RV) based on Propositional Linear Temporal Logic (LTL) with both future and past temporal operators. We generalize the framework to monitor partially observable systems using models of the system under scrutiny (SUS) as assumptions for reasoning on the non-observable or future behaviors of the SUS. The observations are general predicates over the SUS, thus both static and dynamic sets of observables are supported. Furthermore, the monitors are *resettable*, i.e. able to evaluate any LTL property at arbitrary positions of the input trace (roughly speaking, $[\![u, i \models \varphi]\!]$ can be evaluated for any u and i with the underlying assumptions taken into account). We present a symbolic monitoring algorithm that can be efficiently implemented using BDD. It is proven correct and the monitor can be double-checked by model checking. As a by-product, we give the first automata-based monitoring algorithm for Past-Time LTL. Beside feasibility and effectiveness of our approach, we also demonstrate that, under certain assumptions the monitors of some properties are predictive.

1 Introduction

Runtime Verification (RV) [15,26] as a lightweight verification technique, aims at checking whether a *run* of a system under scrutiny (SUS) satisfies or violates a given correctness specification (or *monitoring property*). Given any monitoring property, the corresponding runtime monitor takes as input an execution (i.e. finite prefix of a run, or finite word) and outputs a *verdict* for each input letter (or *state*).

The applicability of RV techniques on *black box systems* for which no system model is at hand, is usually considered as an advantage over other verification techniques like model checking. However, as systems are often partially observable, this forces one to specify the monitoring property in terms of the external interface of the SUS and diagnosis condition on its internals must be reflected in input/output sequence with an implicit knowledge about the SUS behavior. For example, the sequence to verify that an embedded system does not fail during the booting phase may involve observing that an activity LED blinks until it becomes steady within a certain amount of time; the booting failure is not

This work has received funding from European Union's *Horizon 2020* research and innovation programme under the Grant Agreement No. 700665 (Project *CITADEL*).

B. Finkbeiner and L. Mariani (Eds.): RV 2019, LNCS 11757, pp. 165–184, 2019.
https://doi.org/10.1007/978-3-030-32079-9_10

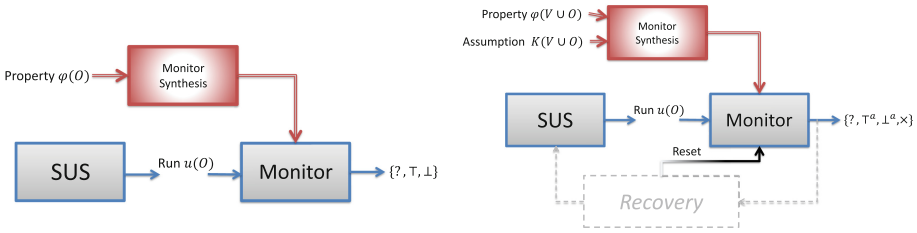

Fig. 1. Traditional RV (left) v.s. ABRV with partial observability & resets (right)

directly observable and the sequence assumes that the LEDs are not broken. In practice, one almost always knows something about the SUS. This information can be derived, for example, from models produced during the system design, or from the interaction with operators (person) of the system. Such information can be leveraged to monitor properties on unobservable parts of the SUS, assuming it behaves the same as specified by its model.

In this paper, we consider the RV problem for Propositional Linear Temporal Logic (PLTL or LTL) with both future and past temporal operators [28]. We extend a traditional RV approach where the monitor synthesis is based on a black-box specification of the system (Fig. 1, on the left) to the practical case where the property to monitor refers to some internal unobservable part of the SUS (Fig. 1, on the right). In order to cope with the partial observability of the SUS, we rely on certain assumption on its behavior, which is expressed in (symbolic) fair transition systems in our framework. Essentially the monitor output in our RV framework can be: the specification is satisfied ($\top^a$) or violated ($\bot^a$) *under* the assumption; the SUS *violates* its assumption ($\times$); or *unknown* (?) otherwise. The output of the monitor depends on the knowledge that can be derived from the partial observations of the system and the semantics of RV is extended to consider all infinite runs of the SUS having the same observed finite execution as prefixes. As for predictive semantics [25,36], by considering the assumption during the synthesis of runtime monitors, the resulting monitors may obtain more precise results: (1) conclusive verdicts could be given on shorter execution prefixes; (2) conclusive verdicts may be obtained from properties that are in general non-monitorable (without assumption).

We also generalize the RV framework to encompass *resettable monitors*. In addition to the observations from SUS, a resettable monitor also takes as input *reset* signals that can change the reference time for the evaluation of the specification without losing the observation history. Consider the case where the monitor is currently evaluating a property φ from the initial position (as done in the traditional case and denoted by $[\![u, 0 \models \varphi]\!]$). Upon a sequence u of observations, receiving as next input a reset, together with a new observation a, the monitor will evaluate φ from the last position. Taking one more observation b but without reset, the monitor will evaluate φ still in the previous position. In general, the monitor can evaluate φ at any position i (denoted by $[\![u, i \models \varphi]\!]$) as

long as a reset is sent to the monitor with the observation at position i in the sequence u. We remark that in this framework if the properties are evaluated under assumptions or contain past operators, the observations before the reset may contribute to the evaluation of the property in the new position.

The motivation for introducing resettable monitors is twofold. First, most monitors based on LTL_3-related semantics are monotonic: once the monitor has reached conclusive true ($\top$) or false ($\bot$), the verdict will remain unchanged for all future inputs, rendering them useless from now on. However, when a condition being monitored occurs (e.g. a fault is detected), and necessary countermeasures (e.g. reconfiguration) have been taken, we want the monitoring process to provide fresh information. Given that the SUS (and maybe also other monitors) is still running, it would be desirable to retain the beliefs of the current system state. The monitor after reset will be evaluating the property at the current reference time, without losing the knowledge of the past. Hence, our reset is different from the simple monitor restart mechanisms found in most RV tools: our monitors keep in track the underlying assumptions and memorize all notable events ever happened in the past, whilst the monitor restart is too coarse in that it wipes out the history, and may thus lose fundamental information. Second, the concept of reset significantly enhances the generality of the formal framework. For example, by issuing the reset signal at every cycle, we capture the semantics of Past-Time LTL, i.e. we monitor $[\![u, |u| - 1 \models \varphi]\!]$ where φ is evaluated with reference to the time point of the most recent observation. As a by-product, this results in the first automata-based monitoring algorithm for Past-Time LTL.

As an example, consider a property $\varphi = \mathbf{G}\,\neg p$, which means that p never occurs, with an assumption K stating that "p occurs at most once." For every sequence u that contains p, the monitor should report a violation of the property (independently of the assumption). After a violation, if the monitor is reset, given the assumption K on the occurrence of p, the monitor should predict that the property is satisfied by any continuation. However, this requires that the reset does not forget that a violation already occurred in the past. Should the SUS produce a trace violating the assumption, where p occurs twice at i and at $j > i$, the assumption-based monitor will output "$\times$" at j.

We propose a new algorithm for assumption-based monitor synthesis with partial observability and resets. It naturally extends the LTL_3 RV approach [4]. Our work is based on a *symbolic* translation from LTL to ω-automata, used also by nuXmv model checker. Using symbolic algorithms, assumptions can be easily supported by (symbolically) composing the ω-automata with a system model representing the assumptions. The algorithm explores the space of beliefs, i.e. the sets of SUS states compatible with the observed signals (traces). The symbolic computation of forward images naturally supports partially observed inputs. Finally, the support of resettable monitors exploits some properties of the symbolic translation from LTL to ω-automata.

The new RV approach has been implemented on top of the nuXmv model checker [8]. We have evaluated our approach on a number of benchmarks showing its feasibility and applicability and the usefulness of assumptions. Beside the

correctness proof, we have also used the nuXmv model checker to verify the correctness and the effectiveness of the synthesized monitors.

The rest of this paper is organized as follows. Preliminaries are presented in Sect. 2. In Sect. 3 our extended RV framework is presented. The symbolic monitoring algorithm and its correctness proof are given in Sect. 4. In Sect. 5 we describe implementation details and the experimental evaluation. Some related work is discussed in Sect. 6. Finally, in Sect. 7, we make conclusions and discuss future directions.

2 Preliminaries

Let Σ be a finite alphabet. A finite word u (or infinite word w) over Σ is a finite (or countably infinite) sequence of letters in Σ, i.e. $u \in \Sigma^*$ and $w \in \Sigma^\omega$. Empty words are denoted by ϵ. u_i denotes the zero-indexed ith letter in u ($i \in \mathbb{N}$ here and after), while u^i denotes the *sub-word* of u starting from u_i. $|u|$ is the length of u. Finally, $u \cdot v$ is the *concatenation* of a finite word u with another finite (or infinite) word v.

Linear Temporal Logic. Let AP be a set of Boolean variables, the set of Propositional Linear Temporal Logic (LTL) [28] formulae, LTL(AP), is inductively defined as

$$\varphi ::= \text{true} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \mathbf{Y}\,\varphi \mid \varphi\,\mathbf{S}\,\varphi$$

with $p \in AP$. Here $\mathbf{X}$ stands for *next*, $\mathbf{U}$ for *until*, $\mathbf{Y}$ for *previous*, and $\mathbf{S}$ for *since*. Other logical constants and operators like false, $\wedge$, $\rightarrow$ and $\leftrightarrow$ are used as syntactic sugars with the standard meaning. The following abbreviations for temporal operators are also used: $\mathbf{F}\,\varphi \doteq \text{true}\,\mathbf{U}\,\varphi$ (*eventually*), $\mathbf{G}\,\varphi \doteq \neg\mathbf{F}\,\neg\varphi$ (*globally*), $\mathbf{O}\,\varphi \doteq \text{true}\,\mathbf{S}\,\varphi$ (*once*), $\mathbf{H}\,\varphi \doteq \neg\mathbf{O}\,\neg\varphi$ (*historically*). Additionally, $\mathbf{X}^n\,p$ denotes a sequence of n nested unary operators: $\mathbf{X}\mathbf{X}\cdots\mathbf{X}\,p$; similar for $\mathbf{Y}^n\,p$.

The semantics of LTL formulae over an infinite word $w \in (2^{AP})^\omega$ is given below:

$$\begin{aligned}
&w, i \models \text{true} \\
&w, i \models p &&\Leftrightarrow && p \in w_i \\
&w, i \models \neg\varphi &&\Leftrightarrow && w, i \not\models \varphi \\
&w, i \models \varphi \vee \psi &&\Leftrightarrow && w, i \models \varphi \ \vee\ w, i \models \psi \\
&w, i \models \mathbf{X}\,\varphi &&\Leftrightarrow && w, i+1 \models \varphi \\
&w, i \models \varphi\,\mathbf{U}\,\psi &&\Leftrightarrow && \exists k.\ i \leqslant k \wedge w, k \models \psi \ \wedge\ \forall j.\ i \leqslant j < k \Rightarrow w, j \models \varphi \\
&w, i \models \mathbf{Y}\,\varphi &&\Leftrightarrow && 0 < i \ \wedge\ w, i-1 \models \varphi \\
&w, i \models \varphi\,\mathbf{S}\,\psi &&\Leftrightarrow && \exists k.\ k \leqslant i \ \wedge\ w, k \models \psi \ \wedge\ \forall j.\ k < j \leqslant i \Rightarrow w, j \models \varphi
\end{aligned}$$

We write $w \models \varphi$ for $w, 0 \models \varphi$ and $\mathcal{L}(\varphi) \doteq \{w \in (2^{AP})^\omega \mid w \models \varphi\}$ for the *language* (or the set of models) of φ. Two formulae ϕ and ψ are equivalent, $\phi \equiv \psi$, iff $\mathcal{L}(\phi) = \mathcal{L}(\psi)$.

Boolean Formulae. Let $\mathbb{B} = \{\top, \bot\}$ denote the type of Boolean values, a set of *Boolean formulae* $\Psi(V)$ over a set of propositional variables $V = \{v_1, \ldots, v_n\}$, is the set of all *well-formed formulae* (wff) [1] built from variables in V, propositional logical operators like $\neg$ and $\wedge$, and parenthesis. Henceforth, as usual in symbolic model checking, any Boolean formula $\psi(V) \in \Psi(V)$ is used to denote the set of truth assignments that make $\psi(V)$ true. More formally, following McMillan [30], a Boolean formula $\psi(V)$ as a set of truth assignments, is the *same* thing as a λ-function of type $\mathbb{B}^{|V|} \to \mathbb{B}$, which takes a vector of these variables and returns a Boolean value, i.e. $\lambda(v_1, \ldots, v_n).\ \psi(v_1, \ldots, v_n)$ or $\lambda V.\ \psi(V)$, assuming a fixed order of variables in V. Thus $\Psi(V)$ itself has the type $(\mathbb{B}^{|V|} \to \mathbb{B}) \to \mathbb{B}$. Whenever V is clear from the context, we omit the whole λ prefix. Therefore, set-theoretic operations such as intersection and union are interchangeable with logical connectives on sets of Boolean formulae.

Fair Kripke Structures. The system models, assumptions and ω-automata used in our RV framework are expressed in a symbolic presentation of Kripke structures called *Fair Kripke Structure* (FKS) [23] (or *Fair Transition System* [29]):

Definition 1. *Let V be a set of Boolean variables, and $V' \doteq \{v' \mid v \in V\}$ be the set of* next state *variables (thus $V \cap V' = \emptyset$). An* FKS *$K = \langle V, \Theta, \rho, \mathcal{J} \rangle$ is given by V, a set of initial states $\Theta(V) \in \Psi(V)$, a transition relation $\rho(V, V') \in \Psi(V \cup V')$, and a set of Boolean formulae $\mathcal{J} = \{J_1(V), \ldots, J_k(V)\} \subseteq \Psi(V)$ called* justice requirements.

Given any FKS $K \doteq \langle V, \Theta, \rho, \mathcal{J} \rangle$, a *state* $s(V)$ of K is an element in 2^V representing a full truth assignment over V, i.e., for every $v \in V$, $v \in s$ if and only if $s(v) = \top$. For example, if $V = \{p, q\}$, a state $\{p\}$ means $p = \top$ and $q = \bot$. Whenever V is clear from the context, we write s instead of $s(V)$. The transition relation $\rho(V, V')$ relates a state $s \in 2^V$ to its successor $s' \in 2^{V'}$. We say that s' is a *successor* of s (and that s is a predecessor of s') iff $s(V) \cup s'(V') \models \rho(V, V')$. For instance, if $\rho(V, V') = (p \leftrightarrow q')$, $s'(V') = \{q'\}$ is a successor of $s(V) = \{p\}$, since $s(V) \cup s'(V') = \{p, q'\}$ and $\{p, q'\} \models (p \leftrightarrow q')$. A path in K is an infinite sequence of states $s_0, s_1, \ldots$ where $s_0(V) \models \Theta$ and, for all $i \in \mathbb{N}$, $s_i(V) \cup s_{i+1}(V') \models \rho(V, V')$. The *forward image* of a set of states $\psi(V)$ on $\rho(V, V')$ is a Boolean formula $\mathrm{fwd}(\psi, \rho)(V) \doteq (\exists V.\ \rho(V, V') \wedge \psi(V))[V/V']$, where $[V/V']$ substitutes all (free) variables from V' to V.

A *fair path* of K is a path $s_0 s_1 \ldots \in \Sigma^\omega$ of K such that, for *all* i we have $s_i \cup s'_{i+1} \models \rho$, and, for all $J \in \mathcal{J}$, for *infinitely many* i, we have that $s_i \models J$. We denote by $\mathrm{FP}^{\rho}_{\mathcal{J}}(\psi)$ the set of fair paths starting from ψ (i.e., such that $s_0 \models \psi$). The language $\mathcal{L}(K)$ is the set of initial fair paths, i.e. $\mathrm{FP}^{\rho}_{\mathcal{J}}(\Theta)$ and $L(K)$ is the set of finite prefixes of paths in $\mathcal{L}(K)$. A state s is *fair* iff it occurs in a fair path. The set of all fair states, denoted by $\mathcal{F}_K$, can be computed by standard algorithms like Emerson-Lei [14]. Finally, let $K_1 = \langle V_1, \Theta_1, \rho_1, \mathcal{J}_1 \rangle$ and $K_2 = \langle V_2, \Theta_2, \rho_2, \mathcal{J}_2 \rangle$, the *synchronous product* of K_1 and K_2 is defined as $K_1 \otimes K_2 \doteq \langle V_1 \cup V_2, \Theta_1 \wedge \Theta_2, \rho_1 \wedge \rho_2, \mathcal{J}_1 \cup \mathcal{J}_2 \rangle$.

Translating LTL to ω-Automata. Our work relies on a linear-time symbolic translation from LTL to ω-automata. The algorithm traces its roots back to [7,10] where only future operators are supported, with additional support of past operators [17]. A set of propositional *elementary variables* of φ, denoted by $\mathrm{el}(\varphi)$, is used for converting any LTL formula into an equivalent propositional formula. It can be defined recursively as follows (where $p \in V$, ϕ and ψ are sub-formulae of φ):

$$\begin{aligned} \mathrm{el}(\mathrm{true}) &= \emptyset, & \mathrm{el}(\mathbf{X}\phi) &= \{\mathrm{X}_\phi\} \cup \mathrm{el}(\phi), \\ \mathrm{el}(p) &= \{p\}, & \mathrm{el}(\phi \mathbf{U} \psi) &= \{\mathrm{X}_{\phi \mathbf{U} \psi}\} \cup \mathrm{el}(\phi) \cup \mathrm{el}(\psi), \\ \mathrm{el}(\neg\phi) &= \mathrm{el}(\phi), & \mathrm{el}(\mathbf{Y}\phi) &= \{\mathrm{Y}_\phi\} \cup \mathrm{el}(\phi), \\ \mathrm{el}(\phi \vee \psi) &= \mathrm{el}(\phi) \cup \mathrm{el}(\psi), & \mathrm{el}(\phi \mathbf{S} \psi) &= \{\mathrm{Y}_{\phi \mathbf{S} \psi}\} \cup \mathrm{el}(\phi) \cup \mathrm{el}(\psi). \end{aligned}$$

For any LTL formula φ, $\mathrm{el}(\varphi) = \mathrm{el}(\neg\varphi)$, and φ can be rewritten into a Boolean formula $\chi(\varphi)$ using only variables in $\mathrm{el}(\varphi)$. Below is the full definition of $\chi(\cdot)$:

$$\chi(\varphi) = \begin{cases} \varphi & \text{for } \varphi \text{ an elementary variable in } \mathrm{el}(\cdot), \\ \neg\chi(\phi) & \text{for } \varphi = \neg\phi, \\ \chi(\phi) \vee \chi(\psi) & \text{for } \varphi = \phi \vee \psi, \\ \mathrm{X}_\phi \ (\text{or } \mathrm{X}_{\phi \mathbf{U} \psi}) & \text{for } \varphi \text{ in forms of } \mathbf{X}\phi (\text{or } \mathbf{X}(\phi \mathbf{U} \psi), \text{ resp.}), \\ \mathrm{Y}_\phi \ (\text{or } \mathrm{Y}_{\phi \mathbf{S} \psi}) & \text{for } \varphi \text{ in forms of } \mathbf{Y}\phi (\text{or } \mathbf{Y}(\phi \mathbf{S} \psi), \text{ resp.}) \ . \end{cases} \tag{1}$$

To apply (1), all sub-formulae of φ leading by $\mathbf{U}$ and $\mathbf{S}$ must be wrapped within $\mathbf{X}$ and $\mathbf{Y}$, respectively. This can be done (if needed) by using the following *Expansion Laws*:

$$\psi \mathbf{U} \phi \equiv \phi \vee (\psi \wedge \mathbf{X}(\psi \mathbf{U} \phi)), \qquad \psi \mathbf{S} \phi \equiv \phi \vee (\psi \wedge \mathbf{Y}(\psi \mathbf{S} \phi)). \tag{2}$$

For instance, $\chi(p \mathbf{U} q) = q \vee (p \wedge \mathrm{X}_{p \mathbf{U} q})$, and $\chi'(p \mathbf{U} q) = q' \vee (p' \wedge \mathrm{X}'_{p \mathbf{U} q})$.

The FKS translated from φ is given by $T_\varphi \doteq \langle V_\varphi, \Theta_\varphi, \rho_\varphi, \mathcal{J}_\varphi \rangle$, where $V_\varphi \doteq \mathrm{el}(\varphi)$.

The initial condition Θ_φ is given by $\Theta_\varphi \doteq \chi(\varphi) \wedge \bigwedge_{\mathrm{Y}_\psi \in \mathrm{el}(\varphi)} \neg \mathrm{Y}_\psi$. Here each $\mathrm{Y}_\psi \in \mathrm{el}(\varphi)$ has an initial false assignment in Θ_φ. This is essentially a consequence of LTL semantics for past operators, i.e. for any word w and formula ψ, $w, 0 \not\models \mathbf{Y}\psi$.

The transition relation ρ_φ (as a formula of variables in $\mathrm{el}(\varphi) \cup \mathrm{el}'(\varphi)$) is given by

$$\rho_\varphi \doteq \bigwedge_{\mathrm{X}_\psi \in \mathrm{el}(\varphi)} \big(\mathrm{X}_\psi \leftrightarrow \chi'(\psi)\big) \ \wedge \bigwedge_{\mathrm{Y}_\psi \in \mathrm{el}(\varphi)} \big(\chi(\psi) \leftrightarrow \mathrm{Y}'_\psi\big). \tag{3}$$

Intuitively, the purpose of ρ_φ is to relate the values of elementary variables to the future/past values: for any $\psi \in \mathrm{el}(\varphi)$, the current value of ψ is *memorized* by the value of Y_ψ in next state; and the next value of ψ is *guessed* by the current value of X_ψ.

The justice set $\mathcal{J}_\varphi$ is given by $\mathcal{J}_\varphi \doteq \{\chi(\psi \mathbf{U} \phi) \rightarrow \chi(\phi) \mid \mathrm{X}_{\psi \mathbf{U} \phi} \in \mathrm{el}(\varphi)\}$. It guarantees that, whenever a sub-formula $\psi \mathbf{U} \phi$ is satisfied, eventually ϕ is

satisfied. Thus an infinite sequence of ψ cannot be accepted by the FKS translated from $\psi \mathbf{U} \phi$.

Notice that T_φ and $T_{\neg\varphi}$ only differ at their initial conditions Θ_φ and $\Theta_{\neg\varphi}$.

3 The Generalized RV Framework

Now we formally present the generalized RV framework which extends the traditional RV with three new features: assumptions, partial observability and resets.

Let $\varphi \in \mathrm{LTL}(AP)$ be a monitoring property[1], $K \doteq \langle V_K, \Theta_K, \rho_K, \mathcal{J}_K \rangle$ be an FKS representing the assumptions under which φ is monitored. Note that K can be a detailed model of the SUS or just a simple constraint over the variables in AP. In general, we do not have any specific assumption on the sets AP and V_K; although it is quite common that $AP \subseteq V_K$, V_K can be even empty if there is no assumption at all. Let $V \doteq V_K \cup AP$.

We say that the SUS is *partially observable* when the monitor can observe only a subset $O \subseteq V$ of variables (O is called the *observables*). Thus, the input trace of the monitor contains only variables from O. However, it is *not* required that all variables in O must be observable in each input state of the input trace. For instance, if $O = \{p, q\}$, it could be imagined that an observation reads the value of p holds but do not know anything about q, or vice versa. It is even possible that an observation does not know anything about p and q, except for knowing that the SUS has moved to its next state. Thus, in general, an observation is a set of assignments to O. If $O = V$ and the observation contains a single assignment to V, then we speak of *full observability*.

As recalled in Sect. 2, this can be represented by a Boolean formula over O. Thus, in our framework, the monitor takes as input a sequence of formulas over O. For example, if the input trace is $\mu = p \cdot q \cdot \top$, then μ represents the following sequence of assignments: $\{\{p\}, \{p, q\}\} \cdot \{\{q\}, \{p, q\}\} \cdot \{\emptyset, \{p\}, \{q\}, \{p, q\}\}$ (recall that, knowing nothing about p and q actually means all 4 possible value assignments are possible, just the monitor does not know which one actually happened in the SUS).

Now we present the *ABRV-LTL* semantics as an extension of Leucker's LTL_3:

Definition 2 *(ABRV-LTL).* *Let $K \doteq \langle V_K, \Theta_K, \rho_K, \mathcal{J}_K \rangle$ be an* FKS*, $\varphi \in \mathrm{LTL}(AP)$, $\mu \in \Psi(O)^*$ be a finite sequence of Boolean formulae over $O \subseteq V_K \cup AP$, and*

$$\mathcal{L}^K(\mu) \doteq \left\{ w \in \mathcal{L}(K) \;\middle|\; \forall i < |\mu|.\, w_i(V_K \cup AP) \models \mu_i(O) \right\} \tag{4}$$

be the set of runs in K which are compatible with μ. The ABRV-LTL semantics of φ over μ under the assumption K, denoted by $[\![\cdot]\!]_4^K \in \mathbb{B}_4 \doteq \{\top^{\mathrm{a}}, \bot^{\mathrm{a}}, ?, \times\}$, is defined as

$$[\![\mu, i \models \varphi]\!]_4^K \doteq \begin{cases} \times, & \text{if } \mathcal{L}^K(\mu) = \emptyset \\ \top^{\mathrm{a}}, & \text{if } \mathcal{L}^K(\mu) \neq \emptyset \wedge \forall w \in \mathcal{L}^K(\mu).\, w, i \models \varphi \\ \bot^{\mathrm{a}}, & \text{if } \mathcal{L}^K(\mu) \neq \emptyset \wedge \forall w \in \mathcal{L}^K(\mu).\, w, i \models \neg\varphi \\ ?, & \text{otherwise.} \end{cases} \tag{5}$$

[1] Here $AP \subseteq V_\varphi$ (the set of variables in T_φ).

ABRV-LTL has four verdicts: *conclusive true* ($\top^{\mathrm{a}}$), *conclusive false* ($\bot^{\mathrm{a}}$), *inconclusive* (?) and *out-of-model* ($\times$). Due to partial observability, the finite trace μ is actually *a set* of finite traces over O, where each u_i of each $u \in \mu$ is a full assignment of truths over O. When $\mathcal{L}^K(\mu) = \emptyset$, K is unable to "follow" the behaviour shown from the SUS, hence the fourth verdict *out-of-model* ($\times$) comes.

The sequence of observations is paired with a sequence of Boolean reset signals. Intuitively, if the monitor receives a reset at cycle i, then it starts to evaluate the truth of φ at i (and does so until the next reset). Formally, the monitor receives inputs in $\Psi(O) \times \mathbb{B}$, the cross-product between formulas over the observables and the reset values. Thus $u = (\mu_0, \mathrm{res}_0), (\mu_1, \mathrm{res}_1), \ldots, (\mu_n, \mathrm{res}_n)$. We denote by $\textsc{res}(u)$ and $\textsc{obs}(u)$ the projection of u respectively on the reset and observation components, i.e. $\textsc{res}(u) = \mathrm{res}_0, \mathrm{res}_1, \ldots, \mathrm{res}_n$ and $\textsc{obs}(u) = \mu_0, \mu_1, \ldots, \mu_n$.

Definition 3 *(ABRV with Partial Observability and Resets). Let K, φ and O have the same meaning as in Definition 2, Let $u \in (\Psi(O) \times \mathbb{B})^*$ be a finite sequence of observations paired with resets. The problem of Assumption-based Runtime Verification (ABRV) w.r.t. K, φ and O is to construct a function $\mathcal{M}^K_\varphi : (\Psi(O) \times \mathbb{B})^* \to \mathbb{B}_4$ such that*

$$\mathcal{M}^K_\varphi(u) = [\![\textsc{obs}(u), \textsc{mrr}(u) \models \varphi]\!]^K_4 \tag{6}$$

where $\textsc{mrr}(u)$ (the most recent reset) is the maximal i such that $\textsc{res}(u_i) = \top$.

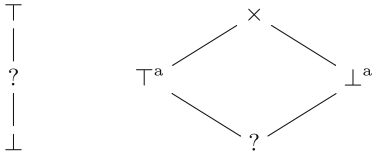

Fig. 2. LTL_3 lattice (left) v.s. ABRV-LTL lattice (right)

Here are some basic properties of the monitor defined in Definition 3. Let $(\mathbb{B}_4, \sqsubseteq)$ be a lattice with the partial order $? \sqsubseteq \top^{\mathrm{a}}/\bot^{\mathrm{a}} \sqsubseteq \times$, shown in Fig. 2 (with a comparison to the LTL_3 lattice). It is not hard to see that, if there is no reset in the inputs, the monitor $\mathcal{M}^K_\varphi$ is always mono-increasing, i.e. $\mathcal{M}^K_\varphi(u) \sqsubseteq \mathcal{M}^K_\varphi(u \cdot (\psi, \bot))$. On the other hand, the monitor is *anti-monotonic* w.r.t. the assumption, i.e. if $\mathcal{L}(K_2) \subseteq \mathcal{L}(K_1)$, then $\mathcal{M}^{K_1}_\varphi(u) \sqsubseteq \mathcal{M}^{K_2}_\varphi(u)$. We omit the proofs of above properties due to page limits, instead the related experiments that use model checkers to prove them on the generated monitors are briefly reported in Sect. 5 with two samples of K_2.

If K_1 is taken as an empty FKS, i.e. $\mathcal{L}(K_1) = (2^O)^\omega$, we say that the assumption K_2 is *valuable* for φ if there exists $u \in (\Psi(O) \times \{\bot\})^*$ such that $\mathcal{M}^{K_1}_\varphi(u) = ?$

and $\mathcal{M}_\varphi^{K_2}(u) = \top^{\mathrm{a}}$ or $\bot^{\mathrm{a}}$. This can happen when the monitor $\mathcal{M}_\varphi^{K_2}$ is *diagnostic*, deducing some non-observable values from the assumption and observations, or when the monitor $\mathcal{M}_\varphi^{K_2}$ is *predictive*, deducing some future facts from the assumption and observations.

Monitoring Past-time LTL. If the monitor is reset on each input state, i.e. $\forall i.\, \textsc{res}(u_i) = \top$, then $\mathcal{M}_\varphi^K(u) = [\![\textsc{obs}(u), |u| - 1]\!]_4^K$. Furthermore, if φ has only past operators ($\mathbf{Y}$ and $\mathbf{S}$), this monitor actually follows the (finite-trace) semantics ($\models_{\mathrm{p}}$) of *Past-Time LTL* [22], where $[\![u \models_{\mathrm{p}} \varphi]\!] \doteq [\![u, |u|-1 \models \varphi]\!]_4^K$ (for $|u| > 0$). The corresponding RV problem (under full observability, without assumptions) is usually handled by rewriting-based approaches or dynamic programming. Using our BDD-based algorithm now it is possible to generate an automaton monitoring Past-Time LTL.

4 The Symbolic Algorithm

Now we present Algorithm 1 for the RV problem given in Definition 3. This algorithm leverages Boolean formulae and can be effectively implemented in Binary Decision Diagrams (BDD) [6]. A monitor is built from an assumption $K \doteq \langle V_K, \Theta_K, \rho_K, \mathcal{J}_K \rangle$ and an LTL property $\varphi \in \mathrm{LTL}(AP)$. Then it can be used to monitor any finite trace $u \in (\Psi(O) \times \mathbb{B})^*$, where $O \subseteq V_K \cup AP$ is the set of observables.

In the monitor building phase (L2–5), the LTL to ω-automata translation algorithm (c.f. Sect. 2) is called on φ and $\neg\varphi$ for the constructions of FKS T_φ and $T_{\neg\varphi}$. The set of fair states of $K \otimes T_\varphi$ and of $K \otimes T_{\neg\varphi}$ are computed as $\mathcal{F}_\varphi^K$ and $\mathcal{F}_{\neg\varphi}^K$. Starting from L6, the purpose is to update two belief states r_φ and $r_{\neg\varphi}$ according to the input trace u. If we imagine $K \otimes T_\varphi$ and $K \otimes T_{\neg\varphi}$ as two NFAs, then r_φ and $r_{\neg\varphi}$ are the sets of current states in them. They are initialized with the initial conditions of $K \otimes T_\varphi$ and $K \otimes T_{\neg\varphi}$ (restricted to fair states). Indeed, their initial values are given by a chain of conjunctions (L6–7). They are then intersected with the first input state u_0 (L9–10). For the remaining inputs (if they exist), when there is no reset (L13–14), the purpose is to walk simultaneously in $K \otimes T_\varphi$ and $K \otimes T_{\neg\varphi}$ by computing the forward images of r_φ and $r_{\neg\varphi}$ with respect to the current input state and the set of fair states.

If any input state comes in with a reset signal, now the monitor needs to be reset (L16–18). Our most important discovery in this paper is that, a simple $r_\varphi \vee r_{\neg\varphi}$ at L16 just did the work. The resulting Boolean formula r actually contains the *history* of the current input trace and the current "position" in the assumption. (c.f. the correctness proof below for more details.) Then the forward image computed in 17–18 is for shifting the current values of all elementary variables by one step into the past, then the conjunction of $\chi(\varphi)$ (or $\chi(\neg\varphi)$, resp.) makes sure that from now on the "new" automata will accept φ (or $\neg\varphi$, resp.) from the beginning, just like in L9–10. We cannot use Θ_φ or $\Theta_{\neg\varphi}$ here, because they contain the initial all-false assignments of the past elementary variables, which may wrongly overwrite the history stored in r, as some of these variables

Algorithm 1: The symbolic (offline) monitor

```
1  function symbolic_monitor(K ≐ ⟨V_K, Θ_K, ρ_K, J_K⟩, φ(AP), u ∈ (Ψ(O) × 𝔹)*)
2      T_φ ≐ ⟨V_φ, Θ_φ, ρ_φ, J_φ⟩ ⟵ ltl_translation(φ);
3      T_¬φ ≐ ⟨V_φ, Θ_¬φ, ρ_φ, J_φ⟩ ⟵ ltl_translation(¬φ);
4      F^K_φ ⟵ fair_states(K ⊗ T_φ);
5      F^K_¬φ ⟵ fair_states(K ⊗ T_¬φ);
6      r_φ ⟵ Θ_K ∧ Θ_φ ∧ F^K_φ;                          /* no observation */
7      r_¬φ ⟵ Θ_K ∧ Θ_¬φ ∧ F^K_¬φ;
8      if |u| > 0 then                                   /* first observation */
9          r_φ ⟵ r_φ ∧ OBS(u_0);
10         r_¬φ ⟵ r_¬φ ∧ OBS(u_0);
11     for 1 ⩽ i < |u| do                                /* more observations */
12         if RES(u_i) = ⊥ then                          /* no reset */
13             r_φ ⟵ fwd(r_φ, ρ_K ∧ ρ_φ)(V_K ∪ V_φ) ∧ OBS(u_i) ∧ F^K_φ;
14             r_¬φ ⟵ fwd(r_¬φ, ρ_K ∧ ρ_φ)(V_K ∪ V_φ) ∧ OBS(u_i) ∧ F^K_¬φ;
15         else                                          /* with reset */
16             r ⟵ r_φ ∨ r_¬φ;
17             r_φ ⟵ fwd(r, ρ_K ∧ ρ_φ)(V_K ∪ V_φ) ∧ χ(φ) ∧ OBS(u_i) ∧ F^K_φ;
18             r_¬φ ⟵ fwd(r, ρ_K ∧ ρ_φ)(V_K ∪ V_φ) ∧ χ(¬φ) ∧ OBS(u_i) ∧ F^K_¬φ;
19     if r_φ = r_¬φ = ⊥ then return ×;
20     else if r_φ = ⊥ then return ⊥^a;
21     else if r_¬φ = ⊥ then return ⊤^a;
22     else return ?;
```

may not be false any more. The whole reset process completes here, then the current input observation $\text{OBS}(u_i)$ is finally considered and the new belief states must be restrict in fair states. Finally (L19–22) the monitor outputs a verdict in $\mathbb{B}_4$, depending on four possible cases on the emptiness of r_φ and $r_{\neg\varphi}$. This is in line with ABRV-LTL given in Definition 2.

Sample Run. Suppose we monitor $\varphi = p\,\mathbf{U}\,q$ (fully observable) assuming $p \neq q$. Here $O = \{p, q\}$, $V_\varphi = \{p, q, x \doteq \mathrm{X}_{p\mathbf{U}q}\}$, $\Theta_\varphi = q \vee (p \wedge x)$, $\Theta_{\neg\varphi} = \neg(q \vee (p \wedge x))$, $\rho_\varphi = x \leftrightarrow (q' \vee (p' \wedge x'))$, and $K = \langle O, \top, p' \neq q', \emptyset\rangle$. ($\mathcal{J}_\varphi$ and $\mathcal{J}_{\neg\varphi}$ can be ignored since all states are fair, i.e. $\mathcal{F}^K_\varphi = \mathcal{F}^K_{\neg\varphi} = \top$.) Let $u = \{p\}\{p\}\cdots\{q\}\{q\}\cdots$ (no reset). Initially (L6–7) $r_\varphi = \Theta_\varphi$, $r_{\neg\varphi} = \Theta_{\neg\varphi}$, taking the initial state $\{p\}$ they become (L9–10) $r_\varphi = \Theta_\varphi \wedge (p \wedge \neg q) \equiv p \wedge \neg q \wedge x$, and $r_{\neg\varphi} = \Theta_{\neg\varphi} \wedge (p \wedge \neg q) \equiv p \wedge \neg q \wedge \neg x$. Since both r_φ and $r_{\neg\varphi}$ are not empty, the monitor outputs ? (if ends here.) If the next state is still $\{p\}$, the values of r_φ and $r_{\neg\varphi}$ actually remain the same, because $\rho_\varphi \wedge (p' \wedge \neg q') \equiv x \leftrightarrow x'$ and L13–14 does not change anything. Thus the monitor still outputs ?, until it received $\{q\}$: in this case $\rho_\varphi \wedge (\neg p' \wedge q') \equiv x \leftrightarrow \top$, and $\text{fwd}(r_{\neg\varphi}, \rho_\varphi)(V_\varphi) \wedge (\neg p' \wedge q')$ (L14) is unsatisfiable, i.e. $r_{\neg\varphi} = \bot$, while r_φ is still not empty, thus the output is $\top^{\mathrm{a}}$. Taking more $\{q\}$ does not change the output, unless the assumption $p \neq q$ is broken (then $r_\varphi = r_{\neg\varphi} = \bot$, the output is $\times$ and remains there, unless the monitor were reset).

Online Monitoring. Algorithm 1 returns a single verdict after processing the entire input trace. This fits into Definition 3. However, runtime monitors are usually required to return verdicts for each input state and "*should* be designed to consider executions in an incremental fashion" [26]. Our algorithm can be easily modified for online monitoring, it outputs one verdict for each input state. It is indeed incremental since r_φ and $r_{\neg\varphi}$ are updated on each input state, and the time complexity of processing one input state is only in terms of the size of K and φ, thus *trace-length independent* [12]. Space complexity is also important, as a monitor may eventually blow up after storing enough inputs. Our algorithm is *trace non-storing* [31] with bounded memory consumption.

Example. Let us consider again the example proposed in Sect. 1: the LTL property $\varphi = \mathbf{G}\,\neg p$ (p never occurs) under the assumption K stating that "p occurs at most once" (expressed in LTL: $\mathbf{G}(p \rightarrow \mathbf{XG}\,\neg p)$). Figure 3 shows the automaton that results from precomputing the states that Algorithm 1 can reach, given φ and K. Each state reports the monitor output (`N` stands for $\bot^{\mathrm{a}}$, `Y` for $\top^{\mathrm{a}}$ and `X` for $\times$), while inputs are represented on the edges (R stands for reset). Starting from state 1, the monitors goes and remains in state 2 with the output ? as long as it reads $\neg p$ independently of the reset; it goes to state 3 with output $\bot$ as soon as it reads p (again independently of the reset); then, either it goes to state 4 with output $\bot$ while still reading $\neg p$ without reset; as soon as a reset is received it goes to state 5 with output $\top$ where it remains while reading $\neg p$; from states 3–5, whenever the monitor receives p (which would be the second occurrence violating the assumption), it goes to the sink state 0 with output $\times$.

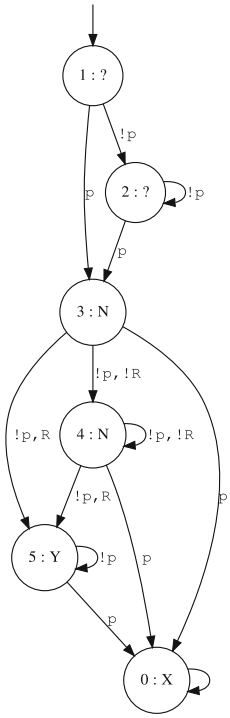

Fig. 3. The monitor of $\mathbf{G}\,\neg p$ under assumption $\mathbf{G}(p \rightarrow \neg\mathbf{XF}\,p)$

Now we show the correctness of Algorithm 1:

Theorem 1. *The function* `symbolic_monitor` *given in Algorithm 1 correctly implements the monitor function* $\mathcal{M}^K_\varphi(\cdot)$ *given in Definition 3.*

Proof (sketch). Fix a trace $u \in (2^O \times \mathbb{B})^*$, we define the following abbreviations:

$$u \lesssim w \Leftrightarrow \forall i.\ i < |u| \Rightarrow w_i(V_k \cup AP) \models \textsc{obs}(u_i)(O), \tag{7}$$

$$\mathcal{L}^K_\varphi(u) \doteq \{w \in \mathcal{L}(K) \mid (w, \textsc{mrr}(u)) \models \varphi) \wedge u \lesssim w\}, \tag{8}$$

$$L^K_\varphi(u) \doteq \{v \mid \exists w.\ v \cdot w \in \mathcal{L}^K_\varphi(u) \wedge |v| = |u|\}. \tag{9}$$

Intuitively, if $u \lesssim w$ holds, w is an (infinite) run of the FKS K *compatible* with the input trace u; $\mathcal{L}^K_\varphi(u)$ is the set of (infinite) u-*compatible* runs of K which satisfies φ w.r.t. the last reset position; And $L^K_\varphi(u)$ is the set of $|u|$-length prefixes from $\mathcal{L}^K_\varphi(u)$.

It is not hard to see that, Definition 3 can be rewritten in terms of $L^K_\varphi(u)$ and $L^K_{\neg\varphi}(u)$:

$$\mathcal{M}^K_\varphi(u) = [\![\text{OBS}(u), \text{MRR}(u) \models \varphi]\!]^K_4 = \begin{cases} \times, & \text{if } L^K_\varphi(u) = \emptyset \wedge L^K_{\neg\varphi}(u) = \emptyset, \\ \top^{\text{a}}, & \text{if } L^K_\varphi(u) \neq \emptyset \wedge L^K_{\neg\varphi}(u) = \emptyset, \\ \bot^{\text{a}}, & \text{if } L^K_\varphi(u) = \emptyset \wedge L^K_{\neg\varphi}(u) \neq \emptyset, \\ ?, & \text{if } L^K_\varphi(u) \neq \emptyset \wedge L^K_{\neg\varphi}(u) \neq \emptyset. \end{cases}$$

Now the proof of Theorem 1 can be reduced to the following sub-goals:

$$L^K_\varphi(u) = \emptyset \Rightarrow r_\varphi(u) = \emptyset \quad \text{and} \quad L^K_{\neg\varphi}(u) = \emptyset \Rightarrow r_{\neg\varphi}(u) = \emptyset. \tag{10}$$

Equation (10) trivially holds when $u = \epsilon$, i.e. $|u| = 0$. Below we assume $|u| > 0$. We first prove the *invariant properties* of r_φ and $r_{\neg\varphi}$: (c.f. L12–18 of Algorithm 1)

$$\begin{aligned} r_\varphi(u) &= \{s \mid \exists w \in \mathcal{L}(K \otimes T_\varphi).\, (w, \text{MRR}(u) \models \varphi) \wedge u \lesssim w \wedge w_{|u|-1} = s\}, \\ r_{\neg\varphi}(u) &= \{s \mid \exists w \in \mathcal{L}(K \otimes T_{\neg\varphi}).\, (w, \text{MRR}(u) \models \neg\varphi) \wedge u \lesssim w \wedge w_{|u|-1} = s\} \end{aligned} \tag{11}$$

Intuitively, $r_\varphi(u)$ is the set of last states of u-compatible runs in $K \otimes T_\varphi$, satisfying φ w.r.t. the last reset position. Now we prove (11) by induction:

If $|u| = 1$, then $r_\varphi = \Theta_K \wedge \Theta_\varphi \wedge \mathcal{F}_{K,\varphi} \wedge \text{OBS}(u_0)$. ($\text{MRR}(u)$ is not used.) Thus, r_φ contains all states s such that $\exists w \in \mathcal{L}(K \otimes T_\varphi), (w, 0 \models \varphi)$, $u_0 \lesssim w_0$ and $w_0 = s$.

If $|u| > 1$ and $\text{RES}(u_n) = \bot$, let $|u| = n + 1$ and $u = v \cdot u_n$ with $|v| > 0$. Here $\text{MRR}(u) = \text{MRR}(v)$. By induction hypothesis, $r_\varphi(v) = \{s \mid \exists w \in \mathcal{L}(K \otimes T_\varphi).\, (w, \text{MRR}(v) \models \varphi) \wedge v \lesssim w \wedge w_{n-1} = s\}$. Thus $r_\varphi(u) = \text{fwd}(r_\varphi(v), \rho_K \wedge \rho_\varphi) \wedge \text{OBS}(u_n) = \{s \mid \exists w \in \mathcal{L}(K \otimes T_\varphi).\, (w, \text{MRR}(v) \models \varphi) \wedge v \cdot u_n \lesssim w \wedge w_n = s\}$. Same arguments for $r_{\neg\varphi}(u)$.

If $|u| > 1$ and $\text{RES}(u_n) = \top$, let $|u| = n + 1$ and $u = v \cdot u_n$ with $|v| > 0$. Here $\text{MRR}(u) = n$. By induction hypothesis, we have

$$\begin{aligned} r_\varphi(v) &= \{s \mid \exists w \in \mathcal{L}(K \otimes T_\varphi).\, (w, \text{MRR}(v) \models \varphi) \wedge v \lesssim w \wedge w_{n-1} = s\}, \\ r_{\neg\varphi}(v) &= \{s \mid \exists w \in \mathcal{L}(K \otimes T_{\neg\varphi}).\, (w, \text{MRR}(v) \models \neg\varphi) \wedge v \lesssim w \wedge w_{n-1} = s\}. \end{aligned}$$

Here, if we take the *union* of $r_\varphi(v)$ and $r_{\neg\varphi}(v)$, the two conjugated terms $(w, \text{MRR}(v) \models \varphi)$ and $(w, \text{MRR}(v) \models \neg\varphi)$ will be just neutralized, i.e., $r_\varphi(v) \vee r_{\neg\varphi}(v) = \{s \mid \exists w \in \mathcal{L}(K \otimes T^0_\varphi).\, v \lesssim w \wedge w_{n-1} = s\}$, where $T^0_\varphi = \langle V_\varphi, \Theta^0_\varphi, \rho_\varphi, \mathcal{J}_\varphi \rangle$ and $\Theta^0_\varphi = \bigwedge_{\text{Y}_p \in \text{el}(\varphi)} \neg \text{Y}_p$. It can be seen that $\forall w \in \mathcal{L}(K \otimes T^0_\varphi), n.\, (w, n \models \varphi) \Leftrightarrow (w^n \models \Theta^0_\varphi)$. Thus $r_\varphi(u) = \text{fwd}(r_\varphi(v) \vee r_{\neg\varphi}(v), \rho_K \wedge \rho_\varphi) \wedge \text{OBS}(u_n) \wedge \chi(\varphi) = \{s \mid \exists w \in$

$\mathcal{L}(K \otimes T_\varphi)$. $(w, n \models \varphi) \wedge (v \cdot u_n \lesssim w) \wedge w_n = s\}$. Same procedure for $r_{\neg\varphi}(u)$, thus (11) is proven.

To finally prove (10), we first unfold (8) into (9) and get $L_\varphi^K(u) = \{v \mid \exists w.\ v \cdot w \in \mathcal{L}(K) \wedge (v \cdot w, \text{MRR}(u) \models \varphi) \wedge u \lesssim v \wedge |v| = |u|\}$. If $L_\varphi^K(u)$ is empty, then by (11) $r_\varphi(u)$ must be also empty, simply because $\mathcal{L}(K \otimes T_\varphi) \subseteq \mathcal{L}(K)$. This proves the first part of (10), the second part follows in the same manner. □

5 Experimental Evaluation

The RV approach presented in this paper has been implemented as an extension of nuXmv [8] in which the BDD library is based on CUDD 2.4.1.1. Besides the offline monitoring in nuXmv, it is also possible to synthesize the symbolic monitors into explicit-state monitors as independent code in various languages as online monitors without dependencies on nuXmv and BDD. The correctness of generated explicit-state monitor code has been extensively tested by comparing the outputs with those from the symbolic monitors, on a large set of LTL properties and random traces.

The comparison of the baseline implementation (no assumption, no reset) with other RV tools is not in the scope of this paper. However, a comparison with the RV-Monitor [27] has been reported in our companion tool paper [9], where our Java-based monitors are shown to be about 200x faster than RV-Monitor at generation-time and 2-5x faster at runtime, besides the capacity of generating monitors from long LTL formulae. As no other tool supports all our extended RV features, here we only focus on experimental evaluations on the usefulness and correctness of our ABRV approach.[2]

Tests on LTL Patterns. To show the feasibility and effectiveness of our RV approach, we have generated monitors from a wide coverage of practical specifications, i.e. Dwyer's LTL patterns [13][3]. To show the impact of assumptions, we generated two groups of monitors, with and without assumption. The chosen assumption says that *the transitions to s-states occur at most 2 times*, which can be expressed in LTL as $((\neg s)\,\mathbf{W}\,(s\,\mathbf{W}\,((\neg s)\,\mathbf{W}\,(s\,\mathbf{W}\,(\mathbf{G}\,\neg s)))))$, where $\mathbf{W}$ denotes *weak until*: $\varphi\,\mathbf{W}\,\psi \doteq (\mathbf{G}\,\varphi) \vee (\varphi\,\mathbf{U}\,\psi) = \varphi\,\mathbf{U}\,(\psi \vee \mathbf{G}\,\varphi)$. Under this assumption we found that, non-monitorable properties like $\mathbf{G}(p \rightarrow \mathbf{F}s)$ now become monitorable, i.e. the monitor may output conclusive verdicts on certain inputs. This is because, if the transitions to s-state have already occurred 2 times, there should be no s any more in the remaining inputs. Thus whenever p occurs, for whatever future inputs it is impossible to satisfy $\mathbf{F}s$, thus the property is violated conclusively. Eight monitors (Pattern 25, 27, 40, 42, 43, 44, 45, 50) are found to be monitorable under this fairness assumption.

[2] All test data, models and other artifacts for reproducing all experiments here are available at https://es.fbk.eu/people/ctian/papers/rv2019/rv2019-data.tar.gz.

[3] The latest version (55 in total) is available at http://patterns.projects.cs.ksu.edu/documentation/patterns/ltl.shtml. We call them Pattern $0, 1, \ldots, 54$ in the same order.

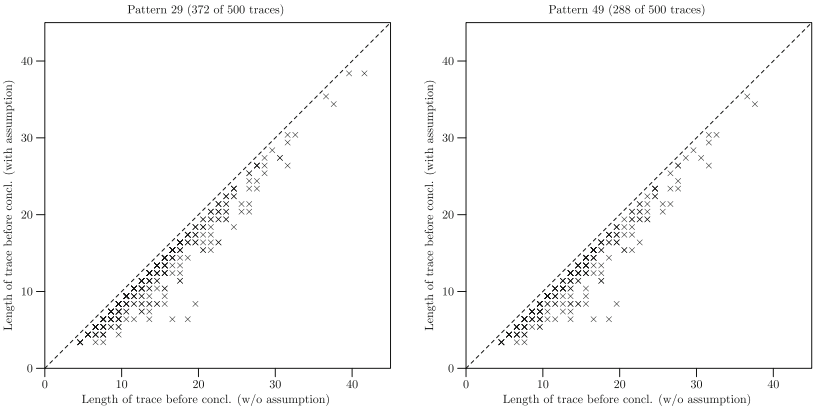

Fig. 4. The number of observations before a conclusive verdict with and w/o assumption

On the other hand, under this assumption some patterns result in predictive monitors, which output conclusive verdicts earlier than those without assumptions. For showing it, we generated 500 random traces (uniformly distributed), each with 50 states, under the assumption (thus the monitor outputs cannot be *out-of-model*). For each pair of monitors (with and without assumption), we record two numbers of states before reaching a conclusive verdict. Whenever the two numbers are the same, the related plot is omitted. In summary, fifteen monitors (Pattern 25, 27, 29, 37, 38, 39, 40, 41, 42, 43, 44, 45, 49, 50, 54) are predictive, and five of them (Pattern 29, 37, 41, 49, 54) have more than 50 traces showing the difference. Figure 4 shows, for example, the tests of Pattern 29 (*s responds to p after q until r*) and 49 (*s, t responds to p after q until r*). The time needed to run the tests on all traces is almost negligible (less than one second) for each pattern.

The *interesting* traces (which show predictive verdicts) can be also obtained by model checking on monitors generated into SMV models. Suppose we have two monitors `M1` (with assumption) and `M2` (w/o assumption), and `AV` := (`M1._concl` $\wedge \neg$ `M2._concl`) (the assumption is valuable iff `M1` has reached conclusive verdicts ($\top^{\mathrm{a}}$, $\bot^{\mathrm{a}}$ or $\times$) while `M2` has not), then the counterexample of model-checking $\neg\mathbf{F}$ `AV` (`AV` *cannot* eventually be true) will be a trace showing that the monitor `M1` is predictive: $\emptyset, \{p, s\}, \emptyset, s, p, \emptyset, \ldots$. Furthermore, it is possible to find a trace such that the distance of conclusive outputs from the two monitors is arbitrary large. For this purpose, we can setup a bounded counter c, whose value only increases when `AV` is true and then verify if c can reach a given maximum value, say, 10. By checking the *invariance* specification $c < 10$, the counterexample will be the desired trace. Similarly, the monotonicity ($\mathbf{G}$ `M._unknown` $\vee$ (`M._unknown` $\mathbf{U}$ `M._concl`)), the correctness (($\mathbf{F}$ `M._true`) $\rightarrow \varphi$ and ($\mathbf{F}$ `M._false`) $\rightarrow \neg\varphi$), and the correctness of resets ($\mathbf{X}^n$(`M._reset` $\wedge \mathbf{X}(\neg$ `M._reset` $\mathbf{U}$ `M._true`)) $\rightarrow \mathbf{X}^n\varphi$) of any monitor `M` generated from φ can also be checked in nuXmv. Details are omitted due to page limits.

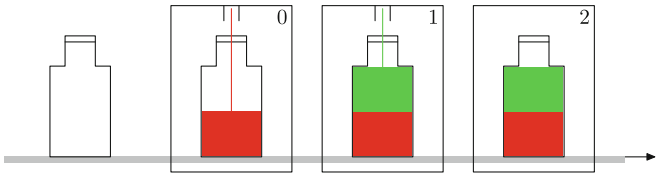

Fig. 5. The factory

Tests on a Factory Model. The assumption used in previous tests may look too artificial, so we present a real-world example taken from [16] and shown in Fig. 5. It models a (simplified) assembly line in a factory, in which some empty bottles need to pass three positions in the assembly line to have two ingredients filled. The red ingredient is filled at position 0, while the green ingredient is filled at position 1. In case of faults, either ingredient may not be correctly filled. The goal is to make sure that all bottles at position 2 have both ingredients filled successfully. There is a belt (the grey bottom line) moving all bottles to their next positions, and the filling operations can only be done when the belt is not moving. All variables in the model are Boolean: `bottle_present[]` (with index 0–2) denotes the existence of a bottle at a position. Similarly, `bottle_ingr1[]` denotes the existence of the red ingredient in the bottle at a position, and `bottle_ingr2[]` for the green ingredient. Besides, `move_belt` denotes if the belt is moving, and `new_bottle` denotes if there is a new bottle coming at position 0 before the belt starts to move. Finally, an unobservable variable `fault` denotes the fault: whenever it happens, the current filling operations (if any) fail and the corresponding ingredients are not filled into the bottle. (The related model files are part of the downloadable artifacts.)

The basic requirement is that all bottles at position 2 have both ingredients filled, if the belt is not moving. It can be expressed by safety property $\mathbf{G}\,((\texttt{bottle\_present[2]} \wedge \neg\,\texttt{move\_belt}) \rightarrow (\texttt{bottle\_ingr1[2]} \wedge \texttt{bottle\_ingr2[2]}))$ (*whenever the belt is not moving and there is a bottle at position 2, both ingredients are filled in that bottle*). We found that, the monitor of the same property, generated with the factory model as assumption, is predictive: it outputs $\bot^{\mathrm{a}}$ *almost immediately* after the first fault happens, *before* the bottle arrived at position 2. To see such a possible trace, again we used model checking. By checking LTL specification $\neg\mathbf{F}\ \texttt{AV}$ where $\texttt{AV} := (\texttt{M1.\_concl} \wedge \neg\,\texttt{M2.\_concl})$ and `M1` (`M2`) are monitors of the above safety property built with (without) assumption, respectively. The counterexample shows one such trace: the fault happens at state 4, and the filling of the red ingredient at position 0 failed at position 1; the monitor with assumption outputs $\bot^{\mathrm{a}}$ at state 6, before the bottle is moved to position 1, while the monitor without assumption can only output $\bot^{\mathrm{a}}$ at state 10, after the bottle is moved to position 2. This is because, any unfilled bottle at position 0 or 1 will remain unfilled at position 2 under the model, thus the monitor with assumption should have known the faults before any unfilled bottle arrived at position 2, even if the fault itself is not directly

observable. In practice, there may be more positions (and more ingredients) in the assembly line, reporting the faults as early as possible may skip the rest of filling operations of the faulty bottle (e.g. the bottle can be removed from the assembly line by a separate recovery process) and potentially reduce the costs.

6 Related Work

The idea of leveraging partial knowledge of a system to improve monitorablity is not altogether new. Leucker [25] considers an LTL_3-based predictive semantics $\mathrm{LTL}_{\mathcal{P}}$, where, given a finite trace u, an LTL formula φ is evaluated on every extension of u that are paths of a model $\hat{\mathcal{P}}$ of the SUS $\mathcal{P}$. Our proposal is a proper conservative extension of this work: in case of full observability, no reset, if the system always satisfies the assumption, i.e. $\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\hat{\mathcal{P}})$, our definition coincides with [25]. As $\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\hat{\mathcal{P}})$ is a strong assumption there, if it is violated, the monitor output will be undefined, while we explicitly take that possibility into account. On the other hand, partial observability is essential for extending traditional RV approaches such that assumptions are really needed to evaluate the property (not only for prediction). In fact, under full observability, if the model $\hat{\mathcal{P}}$ is expressed in LTL, the monitor of [25] coincides with the monitor for $\hat{\mathcal{P}} \rightarrow \phi$ given in [26]. Due to the partial observability, ABRV-LTL monitors cannot be expressed in traditional RV approach (quantifiers over traces would be necessary).

In another three-valued predictive LTL semantics [36], the assumption is based on predictive words. Given a sequence u, a predictive word v of subsequent inputs is computed with static analysis of the monitored program and the monitor output evaluates $[\![u \cdot v \models \varphi]\!]_3$. The assumption used in our framework can be also used to predict the future inputs, but can associate to each u an infinite number of words. Thus our assumption-based RV framework is more general than [36], even without partial observability and resets. On the other side, while our assumptions can be violated by the system execution, the predictive word of [36] is assured by static analysis.

The research of partial observability in Discrete-Event Systems is usually connected with diagnosability [32] and predicability [18,19]. The presence of system models plays a crucial role here, although technically speaking the support of partial observation is orthogonal with the use of system models (or assumptions) in the monitoring algorithm. Given a model of the system which includes faults (eventually leading the system to a failure) and which is partially-observable (observable only with a limited number of events or data variables), diagnosability studies the problem of checking if the faults can be detected within a finite amount of time. On the other hand, if we take an empty specification (true) and use the system model as assumptions, then our monitors will be checking if the system implementation is always consistent with its model—the monitor only outputs $\top^{\mathrm{a}}$ and $\times$ in this case. This is in spirit of Model-based Runtime Verification [2,38], sometimes also combined with extra temporal specifications [34,35,37].

Other work with partial observability appears in decentralised monitoring of distributed systems [3,11], where an LTL formula describing the system's global behavior may be decomposed into a list (or tree) of sub-formulae according to the system components, whose local behaviours are fully observable.

To the best of our knowledge, the concept of resettable monitors was never published before. In general, if we do not consider assumptions or past operators, restarting monitors for LTL is not an issue. For example, in [33], the authors extend a runtime monitor for regular expressions with recovery. Comparing with our work, it is specific to the given pattern and considers neither past operators, nor the system model.

7 Conclusion

In this paper, we proposed an extended RV framework where assumptions, partial observability and resets are considered. We proposed a new four-valued LTL semantics called ABRV-LTL and have shown its necessity in RV monitors under assumptions. As the solution, we gave a simple symbolic LTL monitoring algorithm and demonstrated that, under certain assumptions the resulting monitors are predictive, while some non-monitorable properties becomes monitorable.

Future work includes: (1) analyzing monitorability, fixing the assumption and in the presence of resets; (2) characterizing monitors with partial observability and resets in terms of epistemic operators [21] and forgettable past [24]; (3) synthesizing the minimal assumption and/or the minimal number of observables to make a property monitorable or to detect every violation (this is related to [5,20]).

References

1. Ackermann, W.: Solvable Cases of the Decision Problem. North-Holland Publishing Company (1954). https://doi.org/10.2307/2964059
2. Azzopardi, S., Colombo, C., Pace, G.: A model-based approach to combining static and dynamic verification techniques. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016, Part I. LNCS, vol. 9952, pp. 416–430. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_29
3. Bauer, A., Falcone, Y.: Decentralised LTL monitoring. Formal Methods Syst. Des. **48**(1–2), 46–93 (2016). https://doi.org/10.1007/s10703-016-0253-8
4. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20**(4), 14–64 (2011). https://doi.org/10.1145/2000799.2000800
5. Bittner, B., Bozzano, M., Cimatti, A., Olive, X.: Symbolic synthesis of observability requirements for diagnosability. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, Toronto, Ontario, Canada, 22–26 July 2012. http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5056
6. Bryant, R.E.: Binary decision diagrams. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 191–217. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_7

7. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. **98**(2), 142–170 (1992). https://doi.org/10.1016/0890-5401(92)90017-A
8. Cavada, R., et al.: The nuXmv symbolic model checker. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 334–342. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_22
9. Cimatti, A., Tian, C., Tonetta, S.: NuRV: a nuXmv extension for runtime verification. In: Finkbeiner, B., Mariani, L. (eds.) RV 2019. LNCS, vol. 11757, pp. 382–392. Springer, Cham (2019)
10. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another look at LTL model checking. Formal Methods Syst. Des. **10**(1), 47–71 (1997). https://doi.org/10.1023/A:1008615614281
11. Colombo, C., Falcone, Y.: Organising LTL monitors over distributed systems with a global clock. Formal Methods Syst. Des. **49**(1), 109–158 (2016). https://doi.org/10.1007/s10703-016-0251-x
12. Du, X., Liu, Y., Tiu, A.: Trace-length independent runtime monitoring of quantitative policies in LTL. In: Bjørner, N., de Boer, F. (eds.) FM 2015. LNCS, vol. 9109, pp. 231–247. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19249-9_15
13. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st International Conference on Software Engineering, pp. 411–420. ACM Press, New York (1999). https://doi.org/10.1145/302405.302672
14. Emerson, E.A., Lei, C.-L.: Temporal reasoning under generalized fairness constraints. In: Monien, B., Vidal-Naquet, G. (eds.) STACS 1986. LNCS, vol. 210, pp. 21–36. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-16078-7_62
15. Falcone, Y., Havelund, K., Reger, G.: A tutorial on runtime verification. Eng. Dependable Softw. Syst. **34**, 141–175 (2013). https://doi.org/10.3233/978-1-61499-207-3-141
16. Fauri, D., dos Santos, D.R., Costante, E., den Hartog, J., Etalle, S., Tonetta, S.: From system specification to anomaly detection (and back). In: Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy, pp. 13–24. ACM Press, New York, November 2017. https://doi.org/10.1145/3140241.3140250
17. Fuxman, A.D.: Formal analysis of early requirements specifications. Ph.D. thesis, University of Toronto (2001). http://dit.unitn.it/~ft/papers/afthesis.ps.gz
18. Genc, S., Lafortune, S.: Predictability of event occurrences in partially-observed discrete-event systems. Automatica **45**(2), 301–311 (2009). https://doi.org/10.1016/j.automatica.2008.06.022
19. Genc, S., Lafortune, S.: Predictability in discrete-event systems under partial observation. IFAC Proc. Vol. **39**(13), 1461–1466 (2006). https://doi.org/10.3182/20060829-4-CN-2909.00243
20. Graf, S., Peled, D., Quinton, S.: Monitoring distributed systems using knowledge. In: Bruni, R., Dingel, J. (eds.) FMOODS/FORTE -2011. LNCS, vol. 6722, pp. 183–197. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21461-5_12
21. Halpern, J.Y., Vardi, M.Y.: The complexity of reasoning about knowledge and time. I. Lower bounds. Journal of Computer and System Sciences **38**(1), 195–237 (1989). https://doi.org/10.1016/0022-0000(89)90039-1
22. Havelund, K., Roşu, G.: Synthesizing monitors for safety properties. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 342–356. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46002-0_24

23. Kesten, Y., Pnueli, A., Raviv, L.: Algorithmic verification of linear temporal logic specifications. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 1–16. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055036
24. Laroussinie, F., Markey, N., Schnoebelen, P.: Temporal logic with forgettable past. In: Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS 2002), pp. 383–392. IEEE Comput. Soc., July 2002. https://doi.org/10.1109/LICS.2002.1029846
25. Leucker, M.: Sliding between model checking and runtime verification. In: Qadeer, S., Tasiran, S. (eds.) RV 2012. LNCS, vol. 7687, pp. 82–87. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35632-2_10
26. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Logic Algebraic Program. **78**(5), 293–303 (2009). https://doi.org/10.1016/j.jlap.2008.08.004
27. Luo, Q., et al.: RV-Monitor: efficient parametric runtime verification with simultaneous properties. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 285–300. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_24
28. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, New York (1992). https://doi.org/10.1007/978-1-4612-0931-7
29. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer, New York (1995). https://doi.org/10.1007/978-1-4612-4222-2
30. McMillan, K.L.: Symbolic Model Checking. Springer, Boston (1993). https://doi.org/10.1007/978-1-4615-3190-6
31. Roşu, G., Havelund, K.: Rewriting-based techniques for runtime verification. Autom. Softw. Eng. **12**(2), 151–197 (2005). https://doi.org/10.1007/s10515-005-6205-y
32. Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D.: Diagnosability of discrete-event systems. IEEE Trans. Autom. Control **40**(9), 1555–1575 (1995). https://doi.org/10.1109/9.412626
33. Selyunin, K., et al.: Runtime monitoring with recovery of the SENT communication protocol. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017, Part I. LNCS, vol. 10426, pp. 336–355. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_17
34. Tan, L.: Model-based self-monitoring embedded programs with temporal logic specifications. Autom. Softw. Eng. 380–383 (2005). https://doi.org/10.1145/1101908.1101975
35. Tan, L., Kim, J., Sokolsky, O., Lee, I.: Model-based testing and monitoring for hybrid embedded systems. In: IEEE International Conference on Information Reuse and Integration, pp. 487–492. IEEE, November 2004. https://doi.org/10.1109/IRI.2004.1431508
36. Zhang, X., Leucker, M., Dong, W.: Runtime verification with predictive semantics. In: Goodloe, A.E., Person, S. (eds.) NFM 2012. LNCS, vol. 7226, pp. 418–432. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28891-3_37
37. Zhao, Y., Oberthür, S., Kardos, M., Rammig, F.J.: Model-based runtime verification framework for self-optimizing systems. Electron. Notes Theor. Comput. Sci. **144**(4), 125–145 (2006). https://doi.org/10.1016/j.entcs.2006.02.008
38. Zhao, Y., Rammig, F.: Model-based runtime verification framework. Electron. Notes Theor. Comput. Sci. **253**(1), 179–193 (2009). https://doi.org/10.1016/j.entcs.2009.09.035