WhyMon: A Runtime Monitoring Tool with Explanations as Verdicts

Leonardo Lima, Jonathan Julián Huerta y Munive, and Dmitriy Traytel

Abstract. We present WhyMon, a runtime monitoring tool that produces explanations as verdicts. Receiving as input a metric first-order temporal logic (MFOTL) formula and a stream prefix of time-stamped data-carrying events, WhyMon incrementally outputs explanations that describe *why* each variable assignment satisfies or violates the formula. The tool includes a graphical user interface that facilitates the exploration and understanding of these explanations. Additionally, it incorporates a formally verified checker that can certify the explanations. In this tool paper, we describe WhyMon's architecture and its command line and interactive user interfaces, and demonstrate its usefulness in a case study.

Keywords: runtime monitoring · explanations · metric first-order temporal logic · formal verification · certification

1 Introduction

Formal verification tools like model checkers and runtime monitors play a crucial role in ensuring the correctness of systems. The decision problems these tools solve are computationally challenging, and the tools use complex, optimized algorithms to rise to the challenge. Also, the tools' inputs—the specifications—tend to be complex formulas describing the desired behavior. Given this complexity, correctness is a major concern and puts the burden on the tools and their developers to establish trust in their output [10, 19].

To this end, model checkers output counterexamples to justify specification violations. However, counterexamples, represented by lasso words, often violate the specification for unexpected, vacuous reasons [15]. Vacuity often indicates an issue with the specification and its detection helps model checker users to refine the specification.

In runtime verification (RV), instead of verifying that all traces of a system model satisfy a given specification, one focuses on monitoring: checking the specification on a real system's particular execution trace, i.e., a sequence of data-carrying events. Runtime monitoring is therefore sometimes called "model checking a trace" [4]. While this approach only provides guarantees about the observed trace, it eliminates the need to model the system and supports the specification language's upgrade from propositional temporal logics to first-order temporal logics or similarly expressive data-aware specification languages.

First-order runtime monitors output violating assignments to the specification's free variables, but usually do not justify them. Alas, they require users to trust both the monitor's implementation and to correctly specify the desired properties. We present WhyMon [2, 3]—a runtime monitor that explains its verdicts in great detail.

WhyMon's underlying ideas originate from Basin et al. [5]'s proof-tree-based approach to explaining counterexamples produced by a linear temporal logic (LTL) model checker. Their algorithm, implemented in the Explanator tool, constitutes an offline LTL monitor which processes the counterexample lasso word and outputs a proof tree

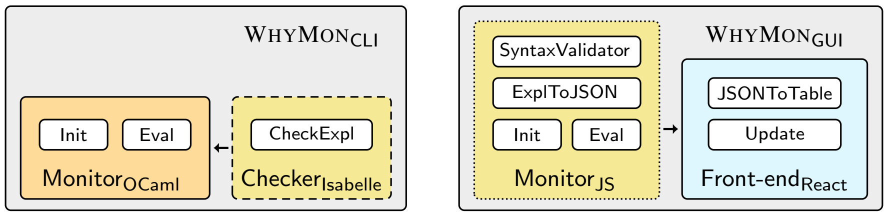


Fig. 1: WHYMON's system architecture.

recursively following the LTL specification's structure and justifying each operator's satisfaction or violation. Lima et al. [16] developed the successor tool EXPLANATOR2 that generalized Basin et al. [5]'s approach to online monitoring and metric temporal logic (MTL) and augmented the proof tree computation with a visualization for interactively exploring these explanations. Our work [17], yielding WHYMON, marks the next evolutionary step by supporting metric first-order temporal logic (MFOTL) [7].

MFOTL monitors like MONPOLY [7, 8] and VERIMON [6, 18] compute and output verdicts in the form of violating assignments of a given specification for every position in the input event log. WHYMON provides a much more detailed verdict (in the form of a proof tree) for every satisfying and violating assignment. As there is an infinite number of assignments (for infinite domains), the main insight of our work [17] is to note that finitely many different proof trees suffice to explain the assignments. WHYMON uses the *partitioned decision tree* (*PDT*) data structure to finitely partition the infinite assignment space into sets of assignments that each have the same proof tree explanations. WHYMON's explanations are PDTs that store proof trees in their leaves.

From the point of view of traditional runtime monitors for first-order temporal logics, WHYMON offers some unique features even without considering explanations. Unlike MONPOLY [7, 8] or VERIMON [6, 18], WHYMON is not restricted to the monitorable fragment of MFOTL. In particular, WHYMON has no restrictions on the usage of negation, universal quantifiers and permits arbitrary free variable patterns in binary operators. (WHYMON's only syntactic restriction is to disallow equalities between variables.) Unlike DEJAVU [11, 12], WHYMON supports (bounded) future temporal operators.

In contrast to our earlier paper [17] describing the theoretical underpinnings of WHYMON, in particular, the proof system, the PDT data structure, and algorithms, this tool paper focuses on WHYMON's architecture (Section 2) and uses the tool to introduce MFOTL without the usual formalities (Section 3). We also describe WHYMON user interfaces: its graphical user interface (GUI) for the interactive exploration of explanations (Section 4), and its command line interface (CLI) for the efficient computation and certification of explanations (Section 5). Finally, we carry out a case study in the area of risk-based authentication (Section 6) to demonstrate WHYMON's usefulness in a realistic setting. We refer to earlier papers [5, 16, 17] for detailed discussions of related work.

2 Architecture

WHYMON's architecture (Figure 1) is subdivided into a CLI (WHYMON$_{\mathsf{CLI}}$) and a GUI (WHYMON$_{\mathsf{GUI}}$). The CLI's main component is Monitor$_{\mathsf{OCaml}}$, which implements WHYMON's monitoring algorithm in OCaml. It has the typical structure of an online monitor, with an Init function that initializes the monitor's state and a step function Eval that updates the monitor's state based on the arrival of new events. The explanations output by the monitor can be checked with the CheckExpl function from the Checker$_{\text{Isabelle}}$ component. This component consists of OCaml code extracted from WhyMon's explanation checker [13] whose correctness we formally verified using the Isabelle proof assistant.

The GUI's main component Front-end$_{\text{React}}$ is a web application [3] written in React [14]. This component interfaces Monitor$_{\text{JS}}$, the JavaScript version of Monitor$_{\text{OCaml}}$ obtained by transpilation using `js_of_ocaml` [20]. Monitor$_{\text{JS}}$ consists of Init and Eval, which remain unchanged, and two additional functions: SyntaxValidator uses WhyMon's parser (also written in OCaml) to validate the GUI's inputs; ExplToJSON converts explanations to a JSON format that is recognized by the front-end. This information is then presented by the Front-end$_{\text{React}}$'s function JSONToTable. Lastly, the Update function maintains the Front-end$_{\text{React}}$'s state updated and calls Monitor$_{\text{JS}}$'s functions according to the interactions coming from the GUI. Our GUI architecture is server-less: the transpilation allows for all the computations to take place in the client's browser.

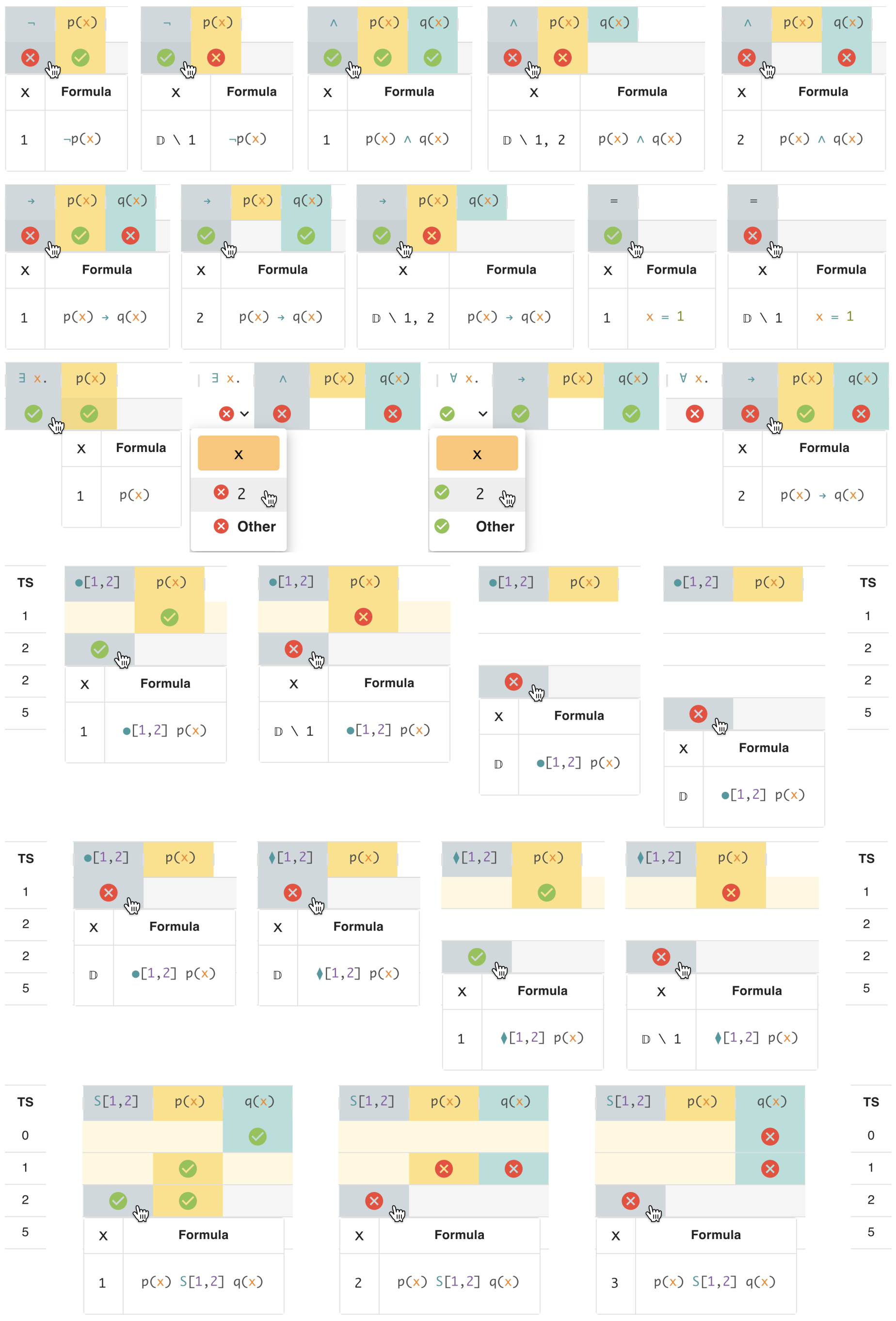
3 Explainable Monitoring

We present the format and meaning of WhyMon's inputs and outputs. We use the GUI to present WhyMon's specification language. Section 4 discusses the GUI in more detail.

An online monitor inputs a trace, i.e., a time-stamped sequence of data-carrying events, and a specification, and outputs whether the specification is satisfied or violated at every position in the sequence. WhyMon's input traces consists of events and their arguments of the form `event_name(arg1,...,argN)`. A *signature* specifies possible events and the types of their arguments using the format `event_name(arg1:type1, ..., argN:typeN)`. WhyMon supports integers (`int`) and strings (`string`) as argument types. A *trace* is a sequence of *time-points* consisting of sets of events and a time-stamp labeled with an "`@`", e.g., `@11 event1(5,some_string), event2(27)`. The time-stamps of a trace may not decrease and must always eventually strictly increase.

WhyMon's specification language is metric first-order temporal logic (MFOTL) [7], an expressive extension of LTL with real-time constraints and first-order quantification over data. The atomic propositions in MFOTL are predicates, i.e., parametrized events consisting of event names applied to a sequence of variables or constants. MFOTL operators, such as Boolean operators (e.g., and, or, not), temporal operators (e.g., since, until), or quantifiers (exists, forall), combine atomic propositions and MFOTL formulas into more complex MFOTL formulas. We visually present the meaning of some MFOTL operators in Figure 2. WhyMon accepts a standard selection of MFOTL operators, including past and future temporal operators and derived operators such as always and eventually. Readers can find a formal description of MFOTL's syntax and semantics elsewhere [17].

For each MFOTL operator in Figure 2, we display WhyMon's verdicts as tables that represent how the operator can be violated or satisfied at a time-point. The input specification's operators and predicates label the columns (at the top) and each operator is followed to its right by its immediate subformulas. Each row represents a trace's time-point and displays the Boolean verdicts (✓ or ✗). Hovering over a Boolean verdict displays the variable assignment for the corresponding subformula. Clicking a Boolean



x	Formula
1	$\neg p(x)$

x	Formula
$\mathbb{D} \setminus 1$	$\neg p(x)$

x	Formula
1	$p(x) \land q(x)$

x	Formula
$\mathbb{D} \setminus 1, 2$	$p(x) \land q(x)$

x	Formula
2	$p(x) \land q(x)$

x	Formula
1	$p(x) \to q(x)$

x	Formula
2	$p(x) \to q(x)$

x	Formula
$\mathbb{D} \setminus 1, 2$	$p(x) \to q(x)$

x	Formula
1	$x = 1$

x	Formula
$\mathbb{D} \setminus 1$	$x = 1$

x	Formula
1	$p(x)$

x	Formula
2	$p(x) \to q(x)$

x	Formula
1	$\bullet[1,2]\ p(x)$

x	Formula
$\mathbb{D} \setminus 1$	$\bullet[1,2]\ p(x)$

x	Formula
$\mathbb{D}$	$\bullet[1,2]\ p(x)$

x	Formula
$\mathbb{D}$	$\bullet[1,2]\ p(x)$

x	Formula
$\mathbb{D}$	$\bullet[1,2]\ p(x)$

x	Formula
$\mathbb{D}$	$\blacklozenge[1,2]\ p(x)$

x	Formula
1	$\blacklozenge[1,2]\ p(x)$

x	Formula
$\mathbb{D} \setminus 1$	$\blacklozenge[1,2]\ p(x)$

x	Formula
1	$p(x)\ S[1,2]\ q(x)$

x	Formula
2	$p(x)\ S[1,2]\ q(x)$

x	Formula
3	$p(x)\ S[1,2]\ q(x)$

Fig. 2: WhyMon's presentation of MFOTL semantics by example.

verdict highlights its justification in yellow (if the operator is unary), or yellow and teal (if it is binary). Moreover, the corresponding formula column and its subformulas' columns are also highlighted in the same colors. Figure 2 uses at most two predicates $\mathsf{p}(x)$ and $\mathsf{q}(x)$ (where x is a variable) to represent each MFOTL operator. The traces used for Boolean operators and quantifiers in Figure 2 have a single entry because their satisfactions or violations depend only on the current time-point. We explain each table in Figure 2 next.

Negation ($\neg$) and equality ($=$): For both of these operators, Figure 2 uses the one-time-point trace "`@0 p(1)`". This should be interpreted as saying that at time $\tau = 0$, the only value that satisfies p is 1. Thus, the negation $\neg\mathsf{p}(x)$ is satisfied (✓) by all other values for x, written $x \in \mathbb{D} \setminus \{1\}$ with $\mathbb{D}$ being x's domain, because the subformula $\mathsf{p}(x)$ does not hold for them. The negation is violated (✗) for $x \in \{1\}$ because the subformula $\mathsf{p}(x)$ satisfies it. Similarly, the formula $x = 1$ is satisfied for $x \in \{1\}$ and violated for $x \in \mathbb{D} \setminus \{1\}$. WhyMon omits set parentheses to improve readability (e.g., $\mathbb{D} \setminus 1$ instead of $\mathbb{D} \setminus \{1\}$).

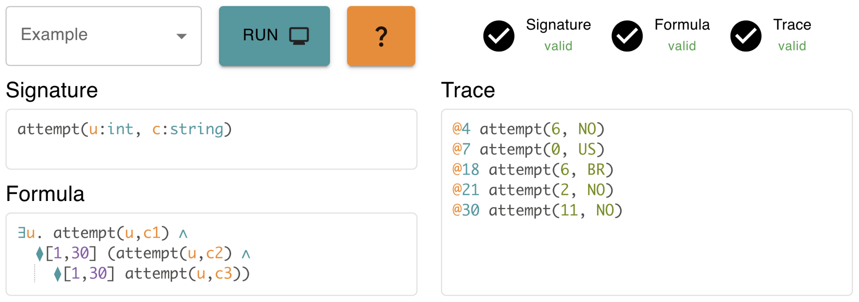
Conjunction ($\wedge$) Figure 2 uses the trace "`@0 p(1), p(2), q(1)`" for $\mathsf{p}(x) \wedge \mathsf{q}(x)$. In this case, the conjunction is satisfied for $x \in \{1\}$ because both conjuncts satisfy it. In contrast, the conjunction is violated in two ways: because the right conjunct violates it when $x \in \{2\}$ and because the left conjunct violates it when $x \in \mathbb{D} \setminus \{1,2\}$.

Implication ($\rightarrow$) The trace "`@0 p(1), p(2), q(2)`" is used for the formula $\mathsf{p}(x) \rightarrow \mathsf{q}(x)$. Here, the implication is violated for $x \in \{1\}$ because the left side is satisfied and the right side is violated. Conversely, it is satisfied either because the right side is satisfied when $x \in \{2\}$, or because the left side is violated when $x \in \mathbb{D} \setminus \{1,2\}$.

Exists ($\exists$) and forall ($\forall$): The assignment of 1 to x witnesses the satisfaction of $\exists x.\ \mathsf{p}(x)$, which stems from the satisfaction of its subformula $\mathsf{p}(x)$ when $x \in \{1\}$, where the trace is "`@0 p(1)`". To violate an existential quantifier, all domain elements should violate its immediate subformula. Accordingly, if we consider the trace "`@0 p(2)`", the formula $\varphi_\exists = \exists x.\ \mathsf{p}(x) \wedge \mathsf{q}(x)$ is violated for $x \in \{2\}$, because the right conjunct is violated while for $x \in \mathbb{D} \setminus \{2\}$ (**Other** in Figure 2), the left conjunct is violated. Thus, the (immediate) subformula is violated for all values of the domain, i.e., for all $x \in \mathbb{D}$. For the universal quantifier, we consider the formula $\varphi_\forall = \forall x.\ \mathsf{p}(x) \rightarrow \mathsf{q}(x)$ and the trace "`@0 p(2), q(2)`". As in the violation for $\varphi_\exists$, the explanation for the satisfaction of $\varphi_\forall$ is split in two, $x \in \{2\}$ and $x \in \mathbb{D} \setminus \{2\}$. Yet, $\varphi_\forall$ overall is satisfied because for all values that x can have ($x \in \mathbb{D}$), its (immediate) subformula is satisfied. In contrast, for the trace "`@0 p(2)`", $\varphi_\forall$ is violated because when $x \in \{2\}$ the subformula is violated. Exploring violations or satisfactions for quantifiers requires interaction in our tool due to the domain partitions.

We now turn to the temporal operators and include the time-stamp column (TS) to track the real-time (metric) constraints. We focus on past operators; the future ones behave dually. The operators include intervals $[a,b]$ restricting the time-range where their immediate subformulas must hold. WhyMon highlights the time-points in these ranges (relative to the considered Boolean verdict) in (light) yellow.

Previous ($\bullet$): The formula $\varphi_\bullet = \bullet_{[1,2]}\mathsf{p}(x)$ with the trace "@1 p(1) @2 @2 @5" is satisfied for $x \in \{1\}$ at the second time-point (second row) because its (immediate) subformula is satisfied at the previous time-point and that time-point lies within 1 to 2 units of time from 2. In contrast, $\varphi_\bullet$ is violated (i) if its subformula is violated at the previous time-point within the interval, e.g., for $x \in \mathbb{D} \setminus \{1\}$; or (ii) if the interval is located either before or after the previous time-point; or (iii) at the trace's first time-point.



```
attempt(u:int, c:string)
```

```
∃u. attempt(u,c1) ∧
  ⧫[1,30] (attempt(u,c2) ∧
    ⧫[1,30] attempt(u,c3))
```

```
@4 attempt(6, NO)
@7 attempt(0, US)
@18 attempt(6, BR)
@21 attempt(2, NO)
@30 attempt(11, NO)
```

Fig. 3: WHYMON's input mode with a signature, a trace and a specification.

Once ($\blacklozenge$): Using the same trace as in the previous case, the formula $\varphi_{\blacklozenge} = \blacklozenge_{[1,2]} \mathsf{p}(x)$ is satisfied if its subformula has been satisfied at some point in the past within the interval, e.g., for $x \in \{1\}$. Yet, $\varphi_{\blacklozenge}$ is violated when: (i) the interval has not yet started; or (ii) the subformula is violated at every time-point within the interval, e.g., when $x \in \mathbb{D} \setminus \{1\}$.
Since ($\mathcal{S}$): The trace "@1 q(1), q(2) @1 p(1), p(3) @2 p(1), p(2), p(3) @5" satisfies the formula $\varphi_{\mathcal{S}} = \mathsf{p}(\mathsf{x})\ \mathcal{S}_{[1,2]}\ \mathsf{q}(x)$ whenever there is a sequence of satisfactions of $p(x)$ up to the current time-point since a satisfaction of $q(x)$ inside the interval. This is the case for $x \in \{1\}$ at time-point 2 in Figure 2. Conversely, $\varphi_{\mathcal{S}}$ is violated when: (i) $p(x)$ is violated at every time-point from (including) the last violation of $q(x)$, e.g., for $x \in \{2\}$; or (ii) $q(x)$ is violated at every time-point inside the interval. Note that a single $p(x)$-violation outside the interval, e.g., between the current time-point and the interval also suffices. The formula $\varphi_{\mathcal{S}}$ is violated (as $\varphi_{\blacklozenge}$) when the interval is before the trace's first time-point.

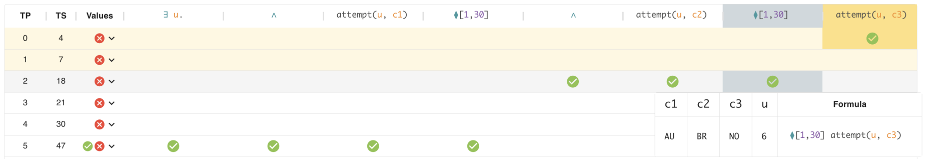
4 Graphical User Interface

We now describe our GUI's key features and outline the usual workflow involved in the interactive exploration of explanations with an example shown in Figures 3 and 4.

Consider an authentication system that records login attempts, including details such as user ID and IP address. Based on the IP address, we can derive the corresponding country. To detect unusual authentication behavior, we use the three_attempts MFOTL policy of Figure 3. It specifies a single user attempting to log in thrice, possibly from different countries, in intervals between 1 and 30 seconds. Figure 3 also includes a signature and a corresponding synthetic trace, where user IDs are integers and country codes are strings.

WHYMON's GUI operates in two modes. Initially, users provide the inputs—the signature, the formula, and the trace (Figure 3)—and the GUI validates them. The GUI includes a live syntax checker and a built-in syntax highlighting. These features help to ensure conformity with WHYMON's syntax and with the signature.

If the inputs are valid, users can switch to the GUI's monitoring mode (Figure 4) using the RUN button. In this mode, users can see a table whose rows corresponds to the log's time-points as seen in Figure 2 but with the first three columns omitted: TP (for time-points), TS (for time-stamps) and Values. The Values column has buttons indicating the presence of Boolean verdicts (✓, ✗, or both) for WHYMON's "explained" time-points (rows). Whenever clicked, these buttons present a (possibly nested) dropdown menu with the possible variable assignments for the formula's free variables.



TP	TS	Values	∃ u.	∧	attempt(u, c1)	⧫[1,30]	∧	attempt(u, c2)	⧫[1,30]	attempt(u, c3)
0	4	✗								✓
1	7	✗								
2	18	✗					✓	✓	✓	
3	21	✗								
4	30	✗								
5	47	✓ ✗	✓	✓	✓	✓				

c1	c2	c3	u	Formula
AU	BR	NO	6	⧫[1,30] attempt(u, c3)

Fig. 4: WHYMON's monitoring mode exploring a satisfaction at time-point 5.

WHYMON is an online monitor. The GUI accounts for that with an input field for new events. Adding the new time-point @47 attempt(6, AU) to the trace extends the GUI's output to indicate that unusual behavior was detected at time-point 5 (Figure 4). In particular, the `three_attempts` policy is satisfied for the variable assignment $(c_1, c_2, c_3) = (\mathtt{AU}, \mathtt{BR}, \mathtt{NO})$. After selecting this variable assignment via the dropdown menu, we explore the associated Boolean verdict in the "$\exists u$" column (corresponding to the topmost operator). Clicking this Boolean verdict makes the GUI uncover and highlight its justification. The satisfaction relies on the first $\wedge$ and we can hover over the Boolean verdict (✓) to see the witness $u = \mathtt{6}$. The first $\wedge$ is satisfied because user `6` attempted to authenticate from Australia (`AU`) at time-point 5 and the right conjunct holds because the first $\blacklozenge_{[1,30]}$ holds. The latter relies on the satisfaction of the second $\wedge$, which occurred at time-point 2. Similarly, the second $\wedge$ is satisfied because user `6` attempted to authenticate from Brazil (`BR`) at time-point 2 and the right conjunct is satisfied. This satisfaction (of the second $\blacklozenge_{[1,30]}$) occurred due to a prior attempt by user `6` from Norway (`NO`) at time-point 0.

5 Command Line Interface

WHYMON includes a CLI that takes as inputs a signature file, a formula file and a trace file (in the same formats as the GUI). If no trace file is given, the CLI reads events from the standard input. There are three execution modes: `unverified` (default), `verified` and `light`. In the `unverified` mode, WHYMON outputs an explanation for each time-point in the trace. In the `verified` mode, the CLI additionally checks the validity of each explanation with its formally verified checker and includes the result of the check in the output (e.g., `Checker output: true`). The `light` mode helps to detect explanations for violations. Specifically, it filters out explanations that only contain satisfactions and for the remaining ones it replaces the proof trees for satisfactions in PDTs with `true` (while keeping the proof trees for violations unchanged). The CLI prints the verdicts to the standard output or to a file, where each explanation is preceded by the corresponding time-stamp:time-point pair (e.g., `3:0`). The CLI executes compiled OCaml, which is orders of magnitude faster than the transpiled JavaScript used by the GUI.

6 Case Study

Risk-based authentication (RBA) monitors users authentication contexts (containing information about, e.g., their geolocation). Upon the detection of unusual behavior, the authentication fails and users are asked to re-authenticate using more secure methods (e.g., two-factor authentication) or their access is blocked. The goal is to mitigate attacks based on stolen or leaked passwords [9, 21]. Traditionally, RBA calculates a risk score based on the login attempt's feature. RBA developers employ various statistical or machine learning methods to compute this score and block a percentage of attackers

$@0\ \mathsf{attempt}(0,\mathtt{US})$
$\vdots$
$@28096\ \mathsf{attempt}(2,\mathtt{AU})$
$@35332\ \mathsf{attempt}(5,\mathtt{NO})$
$\vdots$
$@271368\ \mathsf{attempt}(5,\mathtt{RO})$
$@271794\ \mathsf{attempt}(30,\mathtt{US})$
$\vdots$

$$\begin{aligned}
\mathsf{changed\_to}(u,c) &\overset{def}{=} \mathsf{attempt}(u,c) \\
&\quad \land (\exists c'.\ (\blacklozenge_{[0,t]}\mathsf{travels}(u,c',c)) \land \CIRCLE\,\mathsf{not\_since\_at}(u,c')) \\
\mathsf{travels}(u,c_1,c_2) &\overset{def}{=} \mathsf{attempt}(u,c_1) \land \neg\mathsf{attempt}(u,c_2) \\
&\quad \land \mathsf{at\_country}(u,c_1)\ \mathcal{U}_{[0,t+1]}\ \mathsf{attempt}(u,c_2) \\
\mathsf{at\_country}(u,c) &\overset{def}{=} \forall c'.\ \mathsf{attempt}(u,c') \rightarrow \mathsf{attempt}(u,c) \\
\mathsf{not\_since\_at}(u,c) &\overset{def}{=} (\neg\exists c.\ \mathsf{attempt}(u,c))\ \mathcal{S}_{[0,t]}\ \mathsf{attempt}(u,c')
\end{aligned}$$

Fig. 5: Trace excerpt with an account takeover and the policy definition to find it.

with them. The deployment of these methods reverberates in the login attempts that the rest of the users need to perform. For instance, the RBA algorithm might ask the average legitimate user for re-authentication every second login attempt to block 99.5% of *naive attackers* [9], i.e., attackers that try to sign in from an IP address different from those typically associated with the account. However, determining whether the RBA model has correctly classified a new account takeover attempt can be time-consuming due to obfuscation from multiplied probabilities. Moreover, debugging these models entails accepting the presence of statistical uncertainty. To complement RBA with a more certain analysis, we propose using an MFOTL specification to predict takeovers for the simplest kinds of attackers. We use WHYMON on a subset of an RBA dataset to provide an explanation of an account's behavior that justifies classifying the attempt as a takeover.

We consider a dataset for RBA [1, 21] based on information provided by the Norwegian telecommunications company, Telenor Digital. The dataset is synthetic to protect user anonymity but it preserves the statistical properties of the original one. It includes around 31.3 million login attempts of 3.3 million users spanning over more than a year. Each attempt event contains the time-stamp, the User ID, IP address, and other information (e.g., country). The events also include a Boolean label denoting whether the attempt was deemed an account takeover by the company's security incident response team.

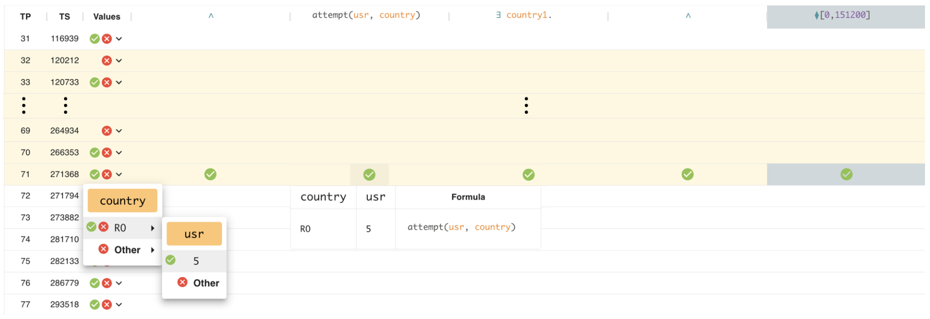
We use the dataset's takeover information and observe that 97% of the attacks are from naive attackers. Hence, we create a trace from the dataset whose events only contain the user ID u and the country c of the attempt. The trace only has one kind of event with the format $@\tau\ \mathsf{attempt}(u,c)$, where τ is the time-stamp in milliseconds. For the sake of presentation, we trim the 1 GB dataset to a subsegment where an account takeover occurs. The subsegment comprises a trace with 150 time-points. We also decrease the trace's time-stamps by the least time-stamp (so that the trace starts with time-stamp 0) and rename user IDs to be smaller integers. An excerpt is shown in Figure 5 where our takeover of interest is greyed out (at time-stamp 271368) with User ID $u = 5$.

Figure 5 shows the definition of the MFOTL formula "$\mathsf{changed\_to}$" that catches naive attackers in general and the takeover at $\tau = 271368$ in particular. Formally, a user u has just *changed to* country c (according to the trace) if currently there is a login-attempt of u from c and if there is a country c' such that the user traveled from c' to c, and the user has not attempted to log in since their last attempt from c'. We model the fact that a user traveled from country c_1 to country c_2 by stating that the user made an attempt from country c_1 at some point in the past and the user remained at country c_1 until they did a login-attempt from country c_2. To distinguish the countries, $\mathsf{travels}$ also asserts that there is no attempt from country c_2 at the same time as another attempt from country c_1. Finally, a user u remains *at a country* c if every login attempt that the user makes is from c. In terms of RBA, we expect that if a legitimate user travels to another country, they will be asked for re-authentication once with our changed_to policy. Afterwards, they would not be asked again until they travel again. The time parameter t represents how far in the past the runtime monitoring algorithm looks for a change of country. That is, setting t to 60 days means that if the user has not attempted to log in for more than two months, any new attempt from a different country would violate changed_to. Approximately 48.3% of users attempted to log in at most one time per month and 22.4% attempted to do so daily [21]. For our subsegment trace, we set t to 151 200 ms (2.52 minutes).

WHYMON successfully finds the takeover attempt against user 5 at time $\tau = 271368$ as seen in Figure 6. Vertical dots represent the 35 omitted rows in the image. It also shows the pop-over after hovering on the attempt from Romania (`RO`) and, in the last column, the greyed out satisfaction of the first conjunct ($\blacklozenge_{[0,t]}\mathsf{travels}(u, c', c)$) under changed_to's existential quantifier. This is a "once" operator whose arguments are shown further in Figure 7. We edit the image to also include the pop-ups when hovering over the first and second conjunct arguments of $\blacklozenge$. These reveal that user 5 was in Norway (`country1` $=$ `NO`) at time-point 32. The until part of the subformula travels is satisfied as (Figure 8). The figure shows that user 5 remained in Norway from time-point 33 to time-point 70, until performing the attempt from Romania (at time-point 71). Finally, Figure 9 depicts the not_since_at part of the formula. It shows that the user did not have any login attempt since its last attempt from Norway until the point just before the attempt from Romania. Therefore, the user suspiciously changed countries in a timespan of less than 2.6 minutes which justifies classifying this attempt as an account takeover.

Reflection WHYMON was useful to refine our specification changed_to as, originally, our candidate for the left subformula of not_since_at was $\neg$at_country. However, all negations in Figure 8 were violated (✖) because the left part of at_country was violated, which made at_country vacuously true. After noticing this issue, we realized that we never meant that the user had to remain at the previous country, but rather that the user did not attempt to log in to the system. WHYMON also helped us with syntactic issues since forgetting the parenthesis around $\blacklozenge_{[0,t]}\mathsf{travels}(u, c', c)$ or the operator $\mathcal{U}$, resulted in unexpected explanations, e.g., ones pointing to the future instead of the past. Moreover, changed_to helped us discover an anomaly in the RBA dataset noticeable in Figure 6: there are many suspicious change of countries from user $u = 2$. We then noticed that this user appeared approximately in 45% of all dataset entries. We suspect that this is either an overflow error in the dataset generation or a legitimate administrative account.

Unfortunately, we could not evaluate WHYMON's performance for larger values of t in changed_to. The GUI struggles with complex specifications on traces with more than 500 time-points. Meanwhile, the CLI processed a 51 minutes prefix of the trace in 48 minutes for $t = 151\,200$ ms on an Apple M1 Chip with 16GB of RAM. Nonetheless, WHYMON proved helpful during the design of our specification when experimenting with short traces. We can use more efficient monitors like MONPOLY to detect violations at scale. We thus expect that to achieve explainable RV we should study combinations of more efficient runtime monitors (e.g. MONPOLY) for analyzing large traces with explainable monitors (e.g. WHYMON) for fragments of these traces.



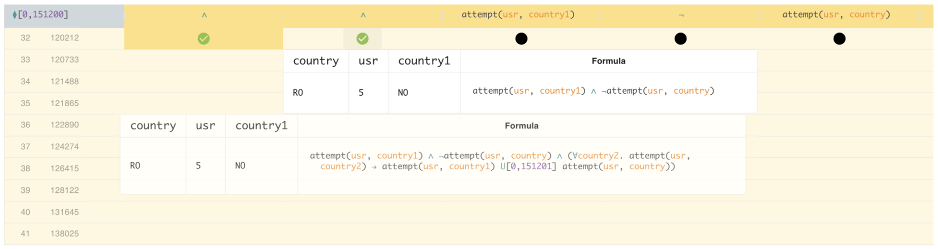
TP	TS	Values	∧	attempt(usr, country)	∃ country1.	∧	⧫[0,151200]
31	116939	✓ ✗					
32	120212	✗					
33	120733	✓ ✗					
⋮	⋮				⋮		
69	264934	✗					
70	266353	✓ ✗					
71	271368	✓ ✗	✓	✓	✓	✓	✓
72	271794						
73	273882						
74	281710						
75	282133						
76	286779	✓ ✗					
77	293518	✓ ✗					

country
✓ ✗ RO
✗ Other

usr
✓ 5
✗ Other

country	usr	Formula
RO	5	attempt(usr, country)

Fig. 6: First part of changed_to(u,c) for user $u = 5$ and country $c = RO$ at $\tau = 271368$.

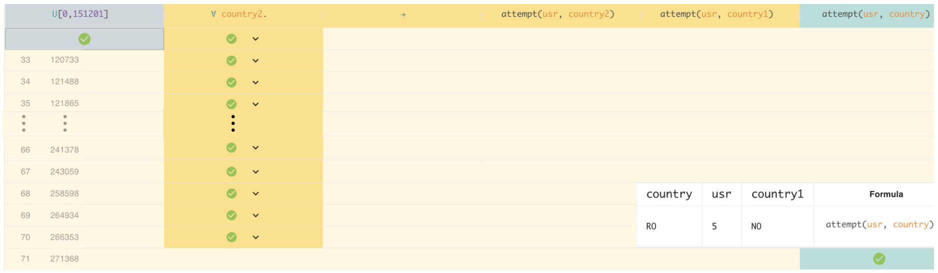


⧫[0,151200]		∧	∧	attempt(usr, country1)	¬	attempt(usr, country)
32	120212	✓	✓	●	●	●
33	120733					
34	121488					
35	121865					
36	122890					
37	124274					
38	126415					
39	128122					
40	131645					
41	138025					

country	usr	country1	Formula
RO	5	NO	attempt(usr, country1) ∧ ¬attempt(usr, country)

country	usr	country1	Formula
RO	5	NO	attempt(usr, country1) ∧ ¬attempt(usr, country) ∧ (∀country2. attempt(usr, country2) → attempt(usr, country1) U[0,151201] attempt(usr, country))

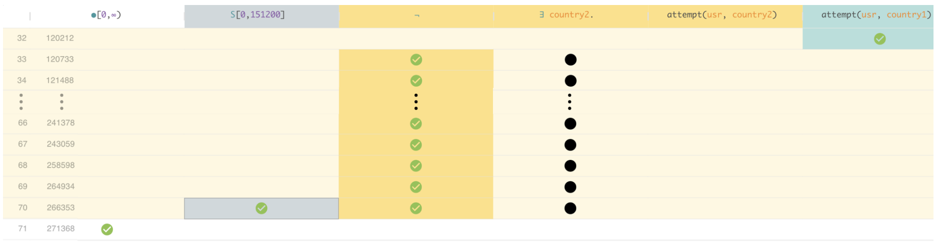
Fig. 7: Once part of changed_to(u,c) for user $u = 5$ and country $c = RO$ at $\tau = 271368$.



U[0,151201]		∀ country2.	→	attempt(usr, country2)	attempt(usr, country1)	attempt(usr, country)
✓		✓				
33	120733	✓				
34	121488	✓				
35	121865	✓				
⋮	⋮	⋮				
66	241378	✓				
67	243059	✓				
68	258598	✓				
69	264934	✓				
70	266353	✓				
71	271368					✓

country	usr	country1	Formula
RO	5	NO	attempt(usr, country)

Fig. 8: Until part of changed_to(u,c) for user $u = 5$ and country $c = RO$ at $\tau = 271368$.



		●[0,∞)	S[0,151200]	¬	∃ country2.	attempt(usr, country2)	attempt(usr, country1)
32	120212						✓
33	120733			✓	●		
34	121488			✓	●		
⋮	⋮			⋮	⋮		
66	241378			✓	●		
67	243059			✓	●		
68	258598			✓	●		
69	264934			✓	●		
70	266353		✓	✓	●		
71	271368	✓					

Fig. 9: Since part of changed_to(u,c) for user $u = 5$ and country $c = RO$ at $\tau = 271368$.

Acknowledgements The research and development of WHYMON are supported by a Novo Nordisk Fonden start package grant (NNF20OC0063462).

References

1. Login Data Set for Risk-Based Authentication (2022), https://www.kaggle.com/datasets/dasgroup/rba-dataset
2. WHYMON's GitHub repository (2024), https://github.com/runtime-monitoring/whymon
3. WHYMON's GUI (2024), https://runtime-monitoring.github.io/whymon
4. Basin, D.: The Cyber Security Body of Knowledge v1.1.0, 2021, chap. Formal Methods for Security. University of Bristol (2021), https://www.cybok.org/, kA Version 1.0.0
5. Basin, D.A., Bhatt, B.N., Traytel, D.: Optimal proofs for linear temporal logic on lasso words. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 37–55. Springer (2018). https://doi.org/10.1007/978-3-030-01090-4_3
6. Basin, D.A., Dardinier, T., Hauser, N., Heimes, L., Huerta y Munive, J.J., Kaletsch, N., Krstić, S., Marsicano, E., Raszyk, M., Schneider, J., Tirore, D.L., Traytel, D., Zingg, S.: VeriMon: A formally verified monitoring tool. In: Seidl, H., Liu, Z., Pasareanu, C.S. (eds.) ICTAC 2022. LNCS, vol. 13572, pp. 1–6. Springer (2022). https://doi.org/10.1007/978-3-031-17715-6_1
7. Basin, D.A., Klaedtke, F., Müller, S., Zalinescu, E.: Monitoring metric first-order temporal properties. J. ACM **62**(2), 15:1–15:45 (2015). https://doi.org/10.1145/2699444
8. Basin, D.A., Klaedtke, F., Zalinescu, E.: The MonPoly monitoring tool. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 19–28. EasyChair (2017). https://doi.org/10.29007/89HS
9. Freeman, D., Jain, S., Dürmuth, M., Biggio, B., Giacinto, G.: Who are you? A statistical approach to measuring user authenticity. In: NDSS 2016. The Internet Society (2016), http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/who-are-you-statistical-approach-measuring-user-authenticity.pdf
10. Goodloe, A.E., Havelund, K.: High-integrity runtime verification. Computer **57**(4), 37–45 (2024). https://doi.org/10.1109/MC.2023.3322902
11. Havelund, K., Peled, D., Ulus, D.: DejaVu: A monitoring tool for first-order temporal logic. In: MT@CPSWeek 2018. pp. 12–13. IEEE (2018). https://doi.org/10.1109/MT-CPS.2018.00013
12. Havelund, K., Peled, D., Ulus, D.: First-order temporal logic monitoring with bdds. Formal Methods Syst. Des. **56**(1), 1–21 (2020). https://doi.org/10.1007/S10703-018-00327-4
13. Herasimau, A., Huerta y Munive, J.J., Lima, L., Raszyk, M., Traytel, D.: A verified proof checker for metric first-order temporal logic. Archive of Formal Proofs (April 2024), https://isa-afp.org/entries/MFOTL_Checker.html, Formal proof development
14. Hunt, P., O'Shannessy, P., Smith, D., Coatta, T.: React: Facebook's functional turn on writing JavaScript. ACM Queue **14**(4), 40 (2016). https://doi.org/10.1145/2984629.2994373
15. Kupferman, O., Vardi, M.Y.: Vacuity detection in temporal model checking. Int. J. Softw. Tools Technol. Transf. **4**(2), 224–233 (2003). https://doi.org/10.1007/S100090100062
16. Lima, L., Herasimau, A., Raszyk, M., Traytel, D., Yuan, S.: Explainable online monitoring of metric temporal logic. In: Sankaranarayanan, S., Sharygina, N. (eds.) TACAS 2023. LNCS, vol. 13994, pp. 473–491. Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_28
17. Lima, L., Huerta y Munive, J.J., Traytel, D.: Explainable online monitoring of metric first-order temporal logic. In: Finkbeiner, B., Kovács, L. (eds.) TACAS 2024. LNCS, vol. 14570, pp. 288–307. Springer (2024). https://doi.org/10.1007/978-3-031-57246-3_16
18. Schneider, J., Basin, D.A., Krstić, S., Traytel, D.: A formally verified monitor for metric first-order temporal logic. In: Finkbeiner, B., Mariani, L. (eds.) RV 2019. LNCS, vol. 11757, pp. 310–328. Springer (2019). https://doi.org/10.1007/978-3-030-32079-9_18

19. Seisenberger, M., ter Beek, M.H., Fan, X., Ferrari, A., Haxthausen, A.E., James, P., Lawrence, A., Luttik, B., van de Pol, J., Wimmer, S.: Safe and secure future ai-driven railway technologies: Challenges for formal methods in railway. In: Margaria, T., Steffen, B. (eds.) ISoLA 2022. LNCS, vol. 13704, pp. 246–268. Springer (2022). https://doi.org/10.1007/978-3-031-19762-8_20
20. Vouillon, J., Balat, V.: From bytecode to JavaScript: the Js_of_ocaml compiler. Softw. Pract. Exp. **44**(8), 951–972 (2014). https://doi.org/10.1002/spe.2187
21. Wiefling, S., Jørgensen, P.R., Thunem, S., Iacono, L.L.: Pump up password security! evaluating and enhancing risk-based authentication on a real-world large-scale online service. ACM Trans. Priv. Secur. **26**(1), 6:1–6:36 (2023). https://doi.org/10.1145/3546069