AllenRV: an Extensible Monitor for Multiple Complex Specifications with High Reactivity

Nic Volanschi, Bernard P. Serpette

AllenRV: an Extensible Monitor for Multiple Complex Specifications with High Reactivity

Nic Volanschi and Bernard Serpette

Inria Bordeaux - Sud-Ouest,
(eugene.volanschi,bernard.serpette)@inria.fr

Abstract. AllenRV is a tool for monitoring temporal specifications, designed for ensuring good scalability in terms of size and number of formulae, and high reactivity. Its features reflect this design goal. For ensuring scalability in the number of formulae, it can simultaneously monitor a set of formulae written in past and future, next-free LTL, with some metric extensions; their efficient simultaneous monitoring is supported by a let construct allowing to share computations between formulae. For ensuring scalability in the size of formulae, it allows defining new abstractions as user-defined operators, which take discrete time boolean signals as arguments, but also constant parameters such as delays. For ensuring high reactivity, its monitoring algorithm does not require clock tick events, unlike many other tools. This is achieved by recomputing output signals both upon input signals changes and upon internally generated timeout events relative to such changes. As a consequence, monitoring remains efficient on arbitrarily fine-grained time domains.
AllenRV is implemented by extending the existing Allen language and compiler, initially targeting ubiquitous applications using binary sensors, with temporal logic operators and a comprehensive library of user-defined operators on top of them. The most complex of these operators, including a complete adaptation of Allen-logic relations as selection operators, are proven correct with respect to their defined semantics.
Thus, AllenRV offers an open platform for cooperatively developing increasingly complex libraries of high level, general or domain-specific, temporal operators and abstractions, without compromising correctness.

Keywords: Online monitoring · Allen logic · Linear Time Logic.

1 Introduction

AllenRV is a monitoring tool for detecting temporal conditions about boolean signals over discrete time. Such boolean signals may directly originate from binary sensors, or be abstracted from non-binary sensors based on value thresholds. These signals typically correspond to the monitoring of cyber-physical systems such as smart homes, smart buildings, or other sensor deployments for IoT applications. It is assumed that input signals are piecewise constant, and are represented as timestamped values that are emitted upon significant value switches (for boolean sensors, this means any value switches). Timestamps are

typically labeled in seconds or milliseconds. Based on the input signals and a set of specifications, AllenRV incrementally computes an output boolean signal for each specification. The output signal reports the satisfaction or violation of each monitored condition at each time point.

The AllenRV monitoring algorithm [12] and its language for expressing temporal specifications [11] are designed to satisfy several key requirements for monitoring applications in this domain:

- It supports the efficient monitoring of multiple specifications over a shared sensor infrastructure, by providing a 'let' construct for sharing common computations between different formulae. This avoids repeatedly computing the same sub-formulae many times. Monitoring multiple related specifications is a key need in applications that simultaneously monitor different aspects at different levels, such as low-level concerns (e.g. detecting basic interactions) and application-level concerns (e.g. detecting human activities), where higher-level aspects commonly reuse formulae of lower-level aspects.
- It supports the efficient development of complex specifications by providing the possibility to define new user-defined operators on boolean signals, extending the comprehensive set of predefined operators. This allows to define programming abstractions that can be instantiated for different set of signals and which can also be parameterized with constant values such as delays or dates. The need for developing complex specifications is key when addressing real use cases, where programming abstractions are typically layered on top of each other. As a complement for the 'let' construct, addressing the reuse of common computations, a 'def' construct allows to reuse programming abstractions by instantiating them in different contexts.
- It supports highly reactive applications that rely on the quick detection of conditions being satisfied or violated, by computing changes in the output signals even when they occur between two input events. In contrast, many of the available RV monitoring tools (e.g., [2, 1, 6]) only recompute output signals upon change events, and rely on the introduction of regular clock events for ensuring their reactivity; however, increasing the rate of regular clock events typically hampers efficient monitoring, and therefore is subject to a reactivity *vs.* efficiency tradeoff. AllenRV does not impose such a dilemma, thanks to the self-generation of timeout events relative to value changes, which trigger additional output signals recomputing, without waiting the next input event. The high reactivity requirement is key in many interactive or security-related applications.

While each individual feature is not necessarily novel, this combination of features ensures unprecedented scalability in terms of number and size of the monitored formulae and towards fine-grained reactivity.

For the specification of conditions on boolean signals, AllenRV uses a Next-free subset of past/future LTL, with some metric extensions. This propositional temporal logic, equivalent to a subset of MTL [9], has proven to be sufficient in practice for expressing various real services of ambient assisted living (AAL) in smart homes (SH), as pointed out in Section 3.

AllenRV is implemented by extending an existing tool for expressing context detection logic in ubiquitous applications called Allen [11]. The original tool offered domain-specific operators on boolean signals, extensible with user-defined operators. Domain-specific operators included adaptations of the Allen-logic relations (during, overlaps, etc.) working on boolean signals, hence the name of the tool. AllenRV adds standard LTL operators Since and Until to the set of native operators, and adds a comprehensive library of system-defined operators, (1) re-implementing all the native operators, and (2) adding classical temporal operators such as bounded Historically/Once past operators. The correctness of the most complex operators, namely from the Allen logic, is ensured by formal proofs [10]. The AllenRV implementation is open-source software distributed under the GPL licence.[1]

2 Tool Description

2.1 Foundations

AllenRV monitors temporal formulae over discrete time boolean signals, which are functions $s : \mathbb{N} \to \mathbb{B}$. In practice, signals typically originate from binary sensors, and are given as a non-empty, possible infinite sequence of timestamped value changes $s(t_i)_{i\geq 0}$ where $t_0 = 0$, and $\forall i > 0$, $t_i > t_{i-1} \wedge s(t_i) = \neg s(t_{i-1})$ (repeated values reported by the sensor are dropped). The *current value* of the signal on the interval $[t_i, t_{i+1})$ is $s(t_i)$. Non-binary sensors can also be used as inputs by converting them to boolean signals using a command option that associates a sensor name to a threshold value.

A discrete time boolean signal can also be viewed as a set of *states*, that is, the discrete time intervals where its current value is 1: $\{[t_i, t_{i+1}) \subseteq \mathbb{N} \mid s(t_i)\}$. If the sequence of timestamped values from a sensor is finite $s(t_i)_{0\leq i\leq n}$, the last interval is $[s(t_n), \infty)$. Note that, as the timestamps t_i of a signal are strictly increasing, its states are non-empty, disjoint and non-adjacent.

A *log* is a sequence of timestamped value changes for a finite set of signals $s \in \mathcal{S}$. The definition of signals over the infinite domain of natural numbers allows to use the standard LTL definitions of temporal operators on infinite traces. By putting $\Sigma = 2^{\mathcal{S}}$, each log may be interpreted as an infinite trace in Σ^ω, namely the infinite sequence $(a_t)_{t\in\mathbb{N}}$ where $a_t = \{s \in \mathcal{S} \mid s(t)\}$ is the set of signals whose current value at time t is 1. In particular, a finite log analyzed in offline mode is seen as its infinite constant continuation, that is, the last sensor values reported in the log are prolonged indefinitely.

2.2 Specifications

All operators in AllenRV take a fixed number of signals as input and produce a signal as output. Besides the boolean operators, having the expected pointwise semantics, there are 4 native operators in AllenRV, defined in Figure 1. Since and

[1] https://github.com/NicVolanschi/Allen

$\text{since}(p,q)(t) \leftrightarrow \exists t' \leq t \,.\, q(t') \wedge \forall t'' \in (t',t] \,.\, p(t'')$
$\text{until}(p,q)(t) \leftrightarrow \exists t' \geq t \,.\, q(t') \wedge \forall t'' \in [t,t') \,.\, p(t'')$
$\text{delay}[T](p) = \{[t+T, t'+T) \mid [t,t') \in p\}$
$p >!!\ T = \{[t+T, t') \,|\, [t,t') \in p \wedge t' - t > T\}$

Fig. 1. The native operators in AllenRV.

Until are the standard past/future LTL operators. Operator delay[T] is delaying a signal by a given period $T \in \mathbb{N}^*$, filling the beginning interval $[0, T)$ with 0.[2] Thus, delay[1] is equivalent to a classical Previous operator with a strong sense, i.e. false at t=0. Moreover, delay[T] is equivalent to the Once operator $\text{O}_{[T,T]}$ in the MTL logic, when interpreted synchronously over the discrete domain of natural numbers, seen as timestamps [8]. Finally, the >!! operator selects from a signal the states longer than a given duration T, but dropping their initial period of length T. This operator is similar to the Historically operator $\text{H}_{\leq T}$ in MTL, but has a strong sense, meaning that $true >!!\ T$ is 0 on $[0, T)$,[3] while $\text{H}_{\leq T} true$ is 1 on $[0, T)$. Nevertheless, the MTL operator $\text{H}_{\leq T}$ can be expressed in terms of the Allen operator >!!, as will be shown in Section 2.3. In this sense, we may say that the logic implemented by AllenRV is a subset of MTL, containing unbounded past/future operators, the Previous operator but not the Next operator, and a subset of bounded past operators, including $\text{H}_{[T,T]}$ and $\text{H}_{\leq T}$ (or, equivalently, $\text{O}_{[T,T]}$ and $\text{O}_{\leq T}$).

Among these 4 operators, only delay[T] existed in the original Allen tool. The other three are extensions belonging to AllenRV.

The complete syntax of the specification language is given in Figure 2. A specification may start with a list of `def` constructs, introducing user-defined operators, and a list of global `let` constructs, introducing named expressions common to all the monitored formulae. After this optional prologue, comes the non-empty list of named monitored formulae, also called 'contexts'. The formulae may use boolean operators, duration operators such as >!! and its variations, and named operators, either defined in the system library or user-defined. Atomic formulae may be named expressions introduced via `let` or signals from the log referred by their name as a string. Constant delays and durations are by default in milliseconds, but can be also given in other units such as seconds or minutes. Expressions may also contain local `let` constructs, introducing named sub-expressions local to one formula.

User-defined operators are expanded as macros, by instantiating their definition with the given constant parameters and signal arguments. In contrast, the computation of each `let` expressions is shared by all the containing formulae. The global `let` construct is an extension belonging to AllenRV. Although it adds no expressiveness to the language, this extension allows to greatly improve performance when multiple formulae rely on common sub-formulae.

[2] From its definition, the first state of delay[T] cannot start sooner that time 0+T.
[3] From its definition, the first state of $p >!!\ T$ cannot start sooner that time 0+T.

```
Prog -> Lib LetRules
Lib -> Def*
Def -> "def" id ("[" id+(",") "]")? ("(" id*(",") ")")?
       "=" Context
LetRules -> "let" id "=" Expr "in" LetRules | Rules
Rules -> id ":" Context (";" Rules)?
Context -> "let" id "=" Expr "in" Context | Expr
Expr -> Prod "|" Expr | Prod
Prod -> Comp "&" Prod | Comp
Comp -> Expr1 (">=!"|"<="|">="|">!"|">!!"|"<"|">") Int | Expr1
Expr1 -> true | false | "~" Expr1 | "(" Expr ")" | str
      | id ("[" Int+(",") "]")? ("(" Expr*(",") ")")?
Int -> id | int ("hr" | "min" | "sec")?
```

Fig. 2. The syntax of the specification language for AllenRV.

2.3 The AllenRV Library

Based on the 4 native operators and leveraging the `def` construct, AllenRV offers a comprehensive library of more than 50 system-defined temporal operators, defined in the AllenRV specification language (complete listing in the Appendix). Some of these operators came from needs experienced in practical applications in the SH and AAL domains, but should be useful in other domains, too. Other operators are well-known shorthands in temporal logics. Most, but not all of these operators existed in the original Allen tool, but they were implemented as native operators independently of each other, in ad-hoc ways.

Classic operators include the weak variants Z (weak Since) and W (weak Until), the unbounded past and future logic quantifiers O (Once), H (Historically), F (Finally), G (Globally), defined as expected using Since and Until, and also the bounded versions of past MTL operators $H_{\leq T}$ and $O_{\leq T}$, defined in Figure 3 using the native operators. Other classical operators are the unary operators recognizing the raising edges of a signal (up), its falling edges (dn), or both (sw).

Operators derived from practice include:

- different duration operators (<=, >), which are variants of the >!! native operator. They all select states from a signal, or sub-intervals thereof, by comparing them to a given duration T.

```
def occ(p,q) = since(q, p&q)  # p has already occurred in this state of q
def step[T] = delay[T](true)  # step function, 0 on [0,T) and 1 afterwards
def orig = ~step[1]           # true only at the origin of time, when t=0
def init(s) = occ(orig(), s)  # selects the state of s starting at t=0
def H_le[T](s) = s >!! T | init(s)   # Historically_<=T in MTL
def O_le[T](s) = ~H_le[T](~s)        # Once_<=T in MTL
```

Fig. 3. Defining the bounded past operators $O_{\leq T}$ and $H_{\leq T}$ in AllenRV.

- selection binary operators derived from the 13 relations in the Allen logic. The Allen-logic relations define all the possible positions of two time interval with respect to each other: : during, contains, starts, started, ends, ended, overlaps, overlapped, meets, met, eq, before, after. For each interval relation IRJ, we implemented a binary operator $r(p, q)$ on signals which selects all the states of p which are in relation R with *some* state of q. To improve the practical usefulness, the before and after have been interpreted as immediately before and immediately after, respectively.
- variants of the O (Once) and F (Finally) operators bounded by a signal, instead of a relative delay. This gives binary operators occ(p,q), meaning "p has occurred at least once in the current state of q", and possible(p,q), meaning "p is still possible in the current state of q".
- different forms of a binary flat(p,q) operator, which glue together the different states of p which occur during a same state of q. This operator is frequently used to reconstitute a whole period within a slot out of fragments of it: for instance, reconstituting a presence in a room out of sporadic movements while no other movements are sensed elsewhere.
- other operators such as a binary operator far[T](p,q) selecting states of p that are far away (i.e. more than T away) from any state in q.

2.4 Online Monitoring

The online monitoring algorithm of Allen, first described in [12], is based on detecting informative prefixes. The compiler constructs a graph for the set of monitored formulae, in which let-bound sub-expressions are shared by all the containing expressions, be it in the same or in different monitored formulae. The monitor pushes from bottom up the value changes for each formula from the corresponding signals; other events in the log are dropped. Evaluating certain kinds of operators may generate timer events that are merged in the event input stream. This way, output signals may change either triggered by input value changes or by delays relative to such changes.

For example. the $p >!! T$ operator schedules a timer event at time $t+T$ when signal p raises to 1 at time t, and cancels the scheduled event when signal p falls to 0. If the timer event is encountered, this means that p has been continuously 1 on the interval $[t, t + T)$, so the output signal is switched to 1.

Future time operators are handled by computing three-valued output signals {0,1,?}, like in LTL3 monitoring [3]. However, our monitoring, like any algorithm based on informative prefixes, does not guarantee that a definite signal (0 or 1) is computed the earliest possible for a monitored formula, but rather when sufficient evidence was gathered to evaluate it from bottom to top. However, it is important to note that, when computing an operator over the three-valued domain, our algorithm does not always block on unknown values. Indeed, some operators may return defined values (0 or 1) even when some of the inputs are unknown (equal to '?'). For instance, the value computed for a node since(p,q) is always 1 when the current value of q is 1, independently of the current value of p, and in particular even if p is currently unknown.

Another salient feature of our algorithm, already mentioned in the Introduction, is that the current output of any operator is recomputed both when the current value of its input changes (in the three-valued domain) *and* possibly on timer events, such as the timer events scheduled by the >!! operator mentioned above. An important consequence of this feature is that the monitor may signal the violation or satisfaction of a formula even when no event happens. For instance, when monitoring for a door left open more than time T during a Night slot using the formula $during(Door >!!\, T, Night)$, the satisfaction of the formula is signalled as soon as the delay T has elapsed since the door has been opened, without waiting for a new event to happen. In contrast, many existing monitors [2, 1, 6] wait for a next event, or rely on an artificial clock event to ensure reactivity. The problem in this case is that more the clock event is fine-grained, more the monitor is overloaded by processing these artificial events, decreasing its efficiency. For AllenRV, no such clock events are needed, and the monitor reacts as soon as it processes its timer events. In fact, our timer events may be considered as clock events generated on demand, without bloating the monitor with useless regular events. In practice, this strategy makes possible a reactivity of the order of 1 millisecond, currently unreachable by many other tools.

2.5 Input and Output

AllenRV is constituted by a compiler, called `allenc` and a virtual machine, called `allen`. The compiler takes a program adhering to the syntax in Figure 2, expressing a set of named formulae to be monitored, and produces a compiled module. The virtual machine is designed as a Unix 'filter', that is, takes a log on standard input (which may come from a file or a pipe) containing the value changes of input signals, and produces a log of value changes of the output signals — one for each named formula.

Both the input and output are in CSV format (colon-separated values): each line is a triple containing a timestamp, a signal name, and a value,. In the input log, signals commonly correspond to sensors, and timestamps are ordered in increasing order. Values having the same timestamp must correspond to different signals (recall that for each signal, the timestamps are strictly increasing) and are considered to happen simultaneously. Simultaneity is important in many Allen-logic operators such as meets(p,q), recognizing the situation when signal p falls at the same time when signal q is raising. In the output log, timestamps are not necessarily increasing, as output signals are de-correlated, and each value change is signalled immediately for maximal reactivity. This absence of sorting is similar to other recent monitors [1, 2].

Timestamps may be either opaque integers (number of seconds or milliseconds since "the Epoch") or human-readable standard date-time timestamps.

2.6 Command-Line Options

The `allen` command offers, among others, options for:

- converting number-valued signals to boolean signals, by associating a threshold to a signal name (or a name pattern, to cover several signals)
- specifying symbolic values for a signal (e.g., OPEN/CLOSED) corresponding to 1 and 0
- debugging, e.g., by executing only one named formula in the specification, or by printing information about the computations performed at each event
- loading a library of operators extending the set of pre-defined operators

This last option is used, for instance, to extend the set of 4 native operators with the AllenRV library.

3 Applications

AllenRV has been successfully used for processing logs of real homes produced by different smart home projects. First, it has been used to simultaneously monitor more than 50 real AAL services on logs spanning one year from more than 100 homes of seniors living along, produced by the HomeAssist project [4]. It has also been used to simultaneously monitor 27 reals-size formulae on logs produced by the Orange4Home experiment [5], and to process logs of several weeks produced by the Amiqual4Home experiment [7]. All these examples are available with the AllenRV distribution, and can be reproduced easily by using the makefile in the examples subdirectory.

For instance, the Orange4Home example demonstrates on the corresponding dataset that AllenRV ensures close to real time processing with millisecond-class reactivity when processing the 27 AAL services simultaneously. These services include: infrastructure monitoring such as checking the correct functioning of light switches based on light sensors; recognizers for various activities in the home, such as showering, napping, or cooking; and meta-level rules built on top of the activity recognizers, such as alerting about unusual activity patterns, or about a potential danger when napping while cooking. The 'let' construct is especially useful in this context, when both activity and meta-activity rules are monitored. Without this construct, activity recognition formulae have to be computed many times, which, in this example, multiplies the total size of the monitored formulae by a factor of 4.3, and increases computation time by 63%.

4 Conclusion

Allen RV is a scalable and extensible tool for monitoring multiple complex specifications on discrete time boolean signals, scaling to arbitrarily fine-grain reactivity with no computation overhead. The system library demonstrates its smooth extensibility, but is only a starting point. The open platform constituted by the AllenRV opens the way to experiment with higher-level specifications abstractions in metric propositional temporal logic, driven by concrete practice. Although our previous practice was limited to the SH and AAL domains, we hope that this platform will be useful in other sensor-based applications in the IoT realm, for example.

References

1. Basin, D.A., Klaedtke, F., Zalinescu, E.: The MonPoly monitoring tool. In: RV-CuBES. pp. 19–28 (2017)
2. Basin, D.A., Krstic, S., Traytel, D.: Aerial: Almost event-rate independent algorithms for monitoring metric regular properties. In: RV-CuBES. pp. 29–36 (2017)
3. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology (TOSEM) **20**(4), 14 (2011)
4. Consel, C., Dupuy, L., Sauzéon, H.: HomeAssist: An assisted living platform for aging in place based on an interdisciplinary approach. In: Proceedings of the 8th International Conference on Applied Human Factors and Ergonomics (AHFE 2017). Springer (2017)
5. Cumin, J., Lefebvre, G., Ramparany, F., Crowley, J.L.: A dataset of routine daily activities in an instrumented home. In: International Conference on Ubiquitous Computing and Ambient Intelligence. pp. 413–425. Springer (2017)
6. El-Hokayem, A., Falcone, Y.: Bringing runtime verification home. In: International Conference on Runtime Verification. pp. 222–240. Springer (2018)
7. Lago, P., Lang, F., Roncancio, C., Jiménez-Guarín, C., Mateescu, R., Bonnefond, N.: The ContextAct@A4H real-life dataset of daily-living activities. In: International and Interdisciplinary Conference on Modeling and Using Context. pp. 175–188. Springer (2017)
8. Letier, E., Kramer, J., Magee, J., Uchitel, S.: Fluent temporal logic for discrete-time event-based models. In: ACM SIGSOFT Software Engineering Notes. vol. 30, pp. 70–79. ACM (2005)
9. Thati, P., Roşu, G.: Monitoring algorithms for metric temporal logic specifications. Electronic Notes in Theoretical Computer Science **113**, 145–162 (2005)
10. Volanschi, N., Serpette, B.: Scaling up RV-based activity detection (2019), submitted
11. Volanschi, N., Serpette, B., Carteron, A., Consel, C.: A language for online state processing of binary sensors, applied to ambient assisted living. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. **2**(4), 192:1–192:26 (Dec 2018). https://doi.org/10.1145/3287070
12. Volanschi, N., Serpette, B., Consel, C.: Implementing a semi-causal domain-specific language for context detection over binary sensors. In: Proceedings of the 17th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences. pp. 66–78. ACM (2018). https://doi.org/10.1145/3278122.3278134