Synchronous and Asynchronous Stream Runtime Verification

César Sánchez
cesar.sanchez@imdea.org
IMDEA Software Institute
Pozuelo de Alarcon, Madrid, Spain

ABSTRACT

We revisit the topic of Stream Runtime Verification (SRV) both for synchronous and asynchronous systems. Stream Runtime Verification is a formalism to express monitors using streams, which results in a simple and expressive specification language. This language is not restricted to describe correctness/failure assertions, but can also collect richer information (including statistical measures) for system profiling and coverage analysis. The monitors generated in SRV are useful for testing, under actual deployment, and to analyze logs.

The steps in many algorithms proposed in runtime verification are based on temporal logics and similar formalisms, which generate Boolean verdicts. The key idea of Stream Runtime Verification is that these algorithms can be generalized to compute richer information if a different data domain is used. Hence, the keystone of SRV is to separate the temporal dependencies in the monitoring algorithm from the concrete operations performed at each step.

Early SRV languages, pioneered by Lola, considered that the observations arrive in a periodic fashion, so the model of time is synchronous sequences like in linear temporal logic. Newer systems, like TeSSLa, RTLola and Striver, have adapted SRV to real-time event streams, where input and output streams can contain events of data at any point.

We will revisit the notions of SRV for synchronous and asynchronous systems. Then, we will justify that synchronous SRV can be modeled by real-time SRV and finally present conditions under which synchronous SRV can simulate real-time SRV.

CCS CONCEPTS

• **Theory of computation → Logic and verification**; **Models of computation**; **Program verification**; • **Software and its engineering → Software verification and validation**.

KEYWORDS

Formal methods, verification, runtime verification

ACM Reference Format:
César Sánchez. 2021. Synchronous and Asynchronous Stream Runtime Verification. In *Proceedings of the 5th ACM International Workshop on Verification and mOnitoring at Runtime EXecution (VORTEX '21), July 12, 2021, Virtual, Denmark*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3464974.3468453

VORTEX '21, July 12, 2021, Virtual, Denmark

ACM ISBN 978-1-4503-8546-6/21/07…$15.00
https://doi.org/10.1145/3464974.3468453

1 INTRODUCTION

Runtime verification (RV) is a dynamic technique for software quality that starts from a formal specification of a property, and studies (1) how to generate a monitor from the specification, and (2) algorithms for the monitors to inspect traces of execution of the system under analysis. Motivated by the counterparts in static verification, many researchers in the early times of runtime verification adapted to RV specification languages based on temporal logics [3, 7, 13], regular expressions [20], timed regular expressions [1], rules [2], or rewriting [18], to name a few. Stream runtime verification, pioneered by Lola [5], defines monitors by declaring the dependencies between output streams of results and input streams of observations. Depending on the model of time considered, different notions of streams arise and in turn, different SRV systems result. We will compare two SRV systems that differ in the model of time assumed (synchronous and real-time event streams) in terms of expressivity and efficiency.

SRV is a more expressive formalism than temporal logics, which are restricted to Boolean observations and verdicts. SRV allows specifying richer properties like collection of statistics counting events, specifying robustness, generating models, quantitative verdicts, etc. For example [21] shows how SRV specifications can describe a Kalman filter that compute predictions of target coordinates and trajectories that a UAV will follow. See [5, 6, 9, 12] for examples illustrating the expressivity of SRV languages.

The most basic early SRV formalisms consider a simple notion of time dictated by the index of the elements in the sequence, like in LTL. Therefore, the data observed in different streams at the same index in their sequences is considered to have occurred at the same time. In this model of time, stream sequences are synchronized and thus we say that formalisms following this paradigm are *synchronous*. Examples of synchronous formalisms in this sense include Lola [5], LTL [16], regular-expressions [14, 20], Mision-time LTL [17], Functional Reactive Programming (FRP) [8] and systems like Copilot [15].

Newer SRV systems consider a model of time streams to be sequences of events formed by data values that are time-stamped with the time at which the data is produced (either observed or generated). In this paradigm, streams can be of different length, and the only condition is that the time-stamps are monotonically increasing. As a result, the same position of different streams are not necessarily time-correlated. In this regard, we can say that stream sequences are *asynchronous*, and thus we say that formalisms following this paradigm are asynchronous SRV formalisms. Examples of asynchronous SRV formalisms include RTLola [10], Striver [12] and TeSSLa [4].

Synchronous SRV formalisms are best suited for cases when data is periodically gathered for every input stream at the same time from the system under analysis. Asynchronous formalisms are best suited for situations when data on the input streams can be received at unpredictable moments—when something of interest happens—and results can be calculated at any time, not only when an event is observed. By these characteristics, we say that synchronous SRV formalisms are *sample based*, while asynchronous SRV formalisms are *event based*.

We will use here Lola and Striver to show how the semantics of a synchronous SRV formalism can be mimicked by an asynchronous SRV formalism and vice versa. As a corollary, the languages subsumed by each formalism can be automatically translated to the other under the conditions of our results. We also compare the efficiency of each approach, and the different alternatives to deal with the loss in performance.

2 LOLA AND STRIVER

We use sequences to refer to typed synchronous streams (formally a map from $\mathbb{N}$ to the type of the stream), and we use $z(n)$ for the value at the n-th position in a sequence z. On the other hand, an *event stream* is a succession of events (t, d) where d is a value from a value domain (as in sequences) and t is a time-stamp from a temporal domain (e.g. $\mathbb{R}, \mathbb{Q}, \mathbb{Z}$). Time values are totally ordered and serve to compare the relative time between two events.

Streams and sequences are typed. A type has a collection of function symbols used to construct expressions, together with an interpretation of these symbols to compute values from values of the arguments. Lola specifications describe monitors by declaratively specifying the relation between output sequences (verdicts) and input sequences (observations). Similarly, Striver specifications describe the relation between output event-streams and input event-streams. Both input and output streams are identified by (typed) stream variables. In Lola, output stream variables are assigned a defining expression that can built using (1) offsets $v[k, d]$ where v is a stream variable of type D, k is an integer number and d a value from D, and (2) function applications $f(t_1, \ldots, t_n)$ using constructors f from the theories to previously defined terms. The intended meaning of expression $v[-1, \mathit{false}]$ is the value of sequence v in the previous position of the trace (or *false* if there is no such previous position, that is, at the beginning). An offset to the current time $v[\mathit{now}]$ does not require a default value. For example, the specification "*the mean level of CO2 in the air in the last 3 instants*", can be expressed as follows:

```
input  num co2
output num steps := steps[-1|0]+1
output num evs   := min(3,steps[now])
output num sum   := co2[-2|0]+co2[-1|0]+co2[now]
output num mean  := sum[now]/evs[now]
```

The semantics of Lola specification is one sequence for each output stream, given a sequence for each input stream. This is guaranteed to be unique if the dependencies given by the offsets do not contain cycles of weight 0. Moreover, efficient online monitors exist if no cycles of positive weight exist. See [5, 19] for details.

The language Striver attempts to extend Lola to real-time event-streams. An output stream variable y is now associated with two expressions: a ticking expression that captures the set of instants at which the y may contain events, and a value expression that describes the value if there is an event. Expressions in Striver are:

- *ticking expressions* are formed by constant instants, union of the ticking instants of other streams, with possible shifts and delays. Shifts and delays may be constants or values of type time for example obtained from streams of type time.
- *offset expressions*, which are the analogous to offsets in Lola, allowing to refer to the time instants at which other streams contain values. The expression `t` represents the current instant, The expression $x\texttt{«}\tau$ is used to refer to the previous instant at which x ticked in the past of τ ($x\texttt{<~}\tau$ is similar but also considers the present as a possibility instant). Analogously, $x\texttt{»}\tau$ refers to the next instant strictly in the future of τ at which x ticks ($x\texttt{>~}\tau$ also considers the present as a candidate)
- *Value Expressions*, which compute values using the constructor symbols from the data theories and offset expressions to fetch events from other streams.

We use `x(<t,d)` and `x(~t,d)` as syntactic sugar (mimicking `x[-1|d]` and `x[now]` from Lola) easily definable in terms of offset expressions.

A Striver specification relates every output stream variable y with a ticking expression T_y and a value expression V_y (of the same type as y). The intended meaning of a specification is that for every collection of real-time event streams for input stream variables, there is a unique collection real-time event streams (one per output variables) satisfying the specification. Again, simple condition on non-circularity of offsets guarantee well-defined semantics, and the absence of positive cycles guarantee the existence of efficient monitors. See [12] for details.

For example, consider the property "*count for how long has the tv been on*", can be expressed as follows, where stream variable `tv_on` computes the result.

```
input TV_Status tv
ticks tv_on := tv.ticks
define int tv_on := if tv(<t,off) == on
                    then tv_on(<t,0) + t - tv<<t else 0
```

3 FROM SYNCHRONOUS TO ASYNCHRONOUS AND BACK

3.1 From Lola to Striver

Simulating synchronous SRV monitoring with an asynchronous SRV system is easy. One just needs to translate a $v[-1, d]$ offset with the next event access `v(<t,d)`, a $v[+1, d]$ offset with the next event access `v(>t,d)` and a $v[\mathit{now}]$ access with a `v(~<t,d)` access. The only technical problem is the following. Circular dependencies that combine future and past accesses which do not add to a zero path are not a problem in Lola. However this becomes problematic in Striver, as combining future and past references in asynchronous systems is forbidden because absence of cycles at runtime is not guaranteed. This is fixed by first transforming the synchronous specification into a specification in which every cycle in the specification only contain edges of one kind (either the cycle contains only future edges or only past edges). Different cycles can contain different kinds of edges, but each cycle is only of one kind. This transformation is always possible (see [11]). The resulting specification is linear in

the size of the original specification, and if the input fed into the asynchronous monitor happens to be synchronized (in terms of the time-stamps which are the indices), so will be the output. In other words the output of the Striver monitor is equivalent to the output produced by the starting Lola monitor. There is empirical evidence [11] that indicates that the Striver monitor is virtually as efficient as the Lola monitor.

3.2 From Striver to Lola

In order to simulate asynchronous real-time stream with a synchronous language like Lola, we introduce two new concepts. The first concept is that of temporal *backbone*. A temporal backbone is a set of time instants at which events can happen in any stream; in other words, events can only happen at instants covered in the backbone. The second concept is that of an *empty* event, which allows to encode the absence of an event as an actual element in a sequence. Note that an event stream extended with additional "empty" events encodes the same event stream.

One way to obtain a backbone is under the assumption of a finite universe of time instants, for example because the quantum of time is the millisecond. In this manner, every event stream can be transformed into a sequence by assuming that there is an element in every millisecond obtained by filling an empty millisecond an empty event whenever there is no actual event in the instant considered. The resulting system can be much less efficient than the original Striver specification, particularly if the occurrence of events is infrequent.

Another way to obtain a backbone is to assume that events only exist in output streams whenever there are events in the input streams as well. That is, is the system is purely *event driven*. Even if the times at which events can happen is in principle unpredictable statically, this assumption allows Lola to simulate asynchronous monitoring. The reason is that Lola is only activated upon the reception of an event, filling with empty events the streams that do not have an event at the time. In this case, the resulting simulation is as efficient as the original Striver monitor.

The second assumption can be extended even further with backbones that combine events from inputs and periodic events that can be predicted statically (timeouts). This third assumption results again into an efficient simulation. This explains the reported high efficiency of RTLola [10], which makes the assumption of asynchronous arrivals and periodic summary calculations.

The general case occurs when events can happen:

- as a result of events in the input streams,
- as periodic events, or
- at instants calculated dynamically from value delays.

This case seems to increase the expressive of the format as no known translation exists into a synchronous engine.

ACKNOWLEDGMENTS

This work was funded in part by Madrid Regional Government under project "S2018/TCS-4339 (BLOQUES-CM)" and by Spanish National Project "BOSCO (PGC2018-102210-B-100)".

REFERENCES

[1] Eugene Asarin, Paul Caspi, and Oded Maler. 2002. Timed regular expressions. *J. ACM* 49, 2 (2002), 172–206. https://doi.org/10.1145/506147.506151

[2] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. 2004. Rule-Based Runtime Verification. In *Proc. of the 5th Int'l Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI'04) (LNCS, Vol. 2937)*. Springer, 44–57. https://doi.org/10.1007/978-3-540-24622-0_5

[3] Andreas Bauer, Martin Leucker, and Chrisitan Schallhart. 2011. Runtime Verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology* 20, 4 (2011), 14:1–14:64. https://doi.org/10.1145/2000799.2000800

[4] Lukas Convent, Sebastian Hungerecker, Martin Leucker, Torben Scheffel, Malte Schmitz, and Daniel Thoma. 2018. TeSSLa: Temporal Stream-based Specification Language. In *Proc. of SBMF'18 (LNCS, Vol. 11254)*. Springer. https://doi.org/10.1007/978-3-030-03044-5_10

[5] Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. 2005. LOLA: Runtime Monitoring of Synchronous Systems. In *Proc. of the 12th Int'l Symp. of Temporal Representation and Reasoning (TIME'05)*. IEEE CS Press, 166–174. https://doi.org/10.1109/TIME.2005.26

[6] Luis Miguel Danielsson and César Sánchez. 2019. Decentralized Stream Runtime Verification. In *Proc. of the 19th Int'l Conf. on Runtime Verification (RV'19) (LNCS, Vol. 11757)*. Springer, 185–201. https://doi.org/10.1007/978-3-030-32079-9_11

[7] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. 2003. Reasoning with Temporal Logic on Truncated Paths. In *Proc. of the 15th Int'l Conf. on Computer Aided Verification (CAV'03) (LNCS, Vol. 2725)*. Springer, 27–39. https://doi.org/10.1007/978-3-540-45069-6_3

[8] Conal Eliot and Paul Hudak. 1997. Functional Reactive Animation. In *Proc. of ICFP'07*. ACM, 163–173. https://doi.org/10.1145/258948.258973

[9] Peter Faymonville, Bernd Finkbeiner, Sebastian Schirmer, and Hazem Torfah. 2016. A Stream-Based Specification Language for Network Monitoring. In *Proc. of the 16th Int'l Conf. on Runtime Verification (RV'16) (LNCS, Vol. 10012)*. Springer, 152–168. https://doi.org/10.1007/978-3-319-46982-9_10

[10] Peter Faymonville, Bernd Finkbeiner, Maximilian Schwenger, and Hazem Torfah. 2017. Real-time Stream-based Monitoring. *CoRR* abs/1711.03829 (2017). arXiv:1711.03829 http://arxiv.org/abs/1711.03829

[11] Felipe Gorostiaga, Luis Miguel Danielsson, and César Sánchez. 2020. Unifying the Time-Event Spectrum for Stream Runtime Verification. In *Proc. of the 20th Int'l Conf on Runtime Verification (RV'20) (LNCS, Vol. 12399)*. Springer, 462–481. https://doi.org/10.1007/978-3-030-60508-7_26

[12] Felipe Gorostiaga and César Sánchez. 2018. Striver: Stream Runtime Verification for Real-Time Event-Streams. In *Proc. of the 18th Int'l Conf. on Runtime Verification (RV'18) (LNCS, Vol. 11237)*. Springer, 282–298. https://doi.org/10.1007/978-3-030-03769-7_16

[13] Klaus Havelund and Grigore Roşu. 2002. Synthesizing Monitors for Safety Properties. In *Proc. of the 8th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02) (LNCS, Vol. 2280)*. Springer-Verlag, 342–356. https://doi.org/10.1007/3-540-46002-0_24

[14] Stephen C. Kleene. 1956. Representation of Events in Nerve Nets and Finite Automata. In *Automata Studies*, Claude E. Shannon and John McCarthy (Eds.). Vol. 34. Princeton University Press, Princeton, New Jersey, 3–41. https://doi.org/10.1515/9781400882618-002

[15] Lee Pike, Alwyn Goodloe, Robin Morisset, and Sebastian Niller. 2010. Copilot: A Hard Real-Time Runtime Monitor. In *Proc. of the 1st Int'l Conf. on Runtime Verification (RV'10) (LNCS, Vol. 6418)*. Springer, 345–359. https://doi.org/10.1007/978-3-642-16612-9_26

[16] Amir Pnueli. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS'77),*. IEEE CS Press, 46–67. https://doi.org/10.1109/SFCS.1977.32

[17] Thomas Reinbacher, Kristin Y. Rozier, and Johann Schumann. 2014. Temporal-logic based runtime observer pairs for system health management of real-time systems. In *Proc. 20th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14) (LNCS, Vol. 8413)*. Springer, 357–372. https://doi.org/10.1007/978-3-642-54862-8_24

[18] Grigore Roşu and Klaus Havelund. 2005. Rewriting-Based Techniques for Runtime Verification. *Automated Software Engineering* 12, 2 (2005), 151–197. https://doi.org/10.1007/s10515-005-6205-y

[19] César Sánchez. 2018. Online and Offline Stream Runtime Verification of Synchronous Systems. In *Proc. of the 18th Int'l Conf. on Runtime Verification (RV'18) (LNCS, Vol. 11237)*. Springer, 138–163. https://doi.org/10.1007/978-3-030-03769-7_9

[20] Koushik Sen and Grigore Roşu. 2003. Generating Optimal Monitors for Extended Regular Expressions. In *Electronic Notes in Theoretical Computer Science*, Oleg Sokolsky and Mahesh Viswanathan (Eds.), Vol. 89. Elsevier. Issue 2. https://doi.org/10.1016/S1571-0661(04)81051-X

[21] Sebastián Zudaire, Felipe Gorostiaga, César Sánchez, Gerardo Schneider, and Sebastián Uchitel. 2021. Assumption Monitoring Using Runtime Verification for UAV Temporal Task Plan Executions. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA'21)*. IEEE.