Event-Based Characterisation of Temporal Properties over System States

Christian Colombo[1], Gordon J. Pace[1], and Justine Scicluna[1]

University of Malta

The design of runtime verification [3] (or monitoring) systems presents a myriad of options — how to instrument properties, in which logic to specify properties, what algorithms to use to implement the property checking, etc. One crucial issue is what elements of the system one is interested in observing, and what points-of-interest one must capture to be able to perform this monitoring. Many runtime verification tools base their properties on the control flow of the system (e.g. [2]): method calls, object creation, exception raising, etc. Especially in the domain of distributed systems, one also finds communication-centric runtime monitoring, in which one focuses on the communication taking place between nodes (e.g. see [1]). Finally, a minority of tools take a data-centric approach, in which one can write properties about the values stored in the system state. The choice of approach has a major influence on how monitoring code can be instrumented in the system. Typically, control-centric approaches use aspect-oriented programming (or similar) technologies to insert additional code identifying the events of interest in the system. On the other hand, to monitor communication in a distributed, message-passing system, one may create communication proxies (actual or local virtual ones) which capture and analyse the messages, i.e. the temporal points-of-interest in such a system. In a data-centric approach, one typically captures points of discontinuity in the values of variables — when they are assigned a value — to be able to capture properties which talk about how the values of the system state changes over time[1].

In this brief presentation, we will present ongoing experiments on adding data-oriented monitoring to the runtime verification tool LARVA [2], and in particular looking at temporal values which change continuously over time, such as the average of the value of a variable over time.

Real-Time System Behvaiour. The behaviour of a system can be characterised by the values stored in variables, possibly changing over time. Let $\mathcal{I}$ be all possible interpretations of variables *Var* ranging over values *Val* for a single instant of time ($\mathcal{I} = \mathit{Var} \rightarrow \mathit{Val}$). The temporal behaviour of such a system is characterised by a total function $\mathcal{I}_t \in \mathbb{T} \rightarrow \mathcal{I}$ — where $\mathbb{T}$ is the time-domain, typically $\mathbb{R}^+$.

Practically Monitorable Points. In practice, monitors cannot keep track of the variables at each point on the real-time line, but typically, one would be able to monitor (effectively) points in time when a variable has been assigned to a new value.

[1] In the literature, one also finds work on the monitoring of analogue systems e.g. [4], in which variables may change values in a continuous way over time. However, in this presentation, we will focus on digital system which exhibit memory values which discretely change (in a discontinuous manner) over time.

This provides a discrete way of characterising the value of a variable over time in a set of sequential time-stamped values identifying (all) points in time when the variable changed its value. Thus, for example, two variables x and y both with initial value 12 and whose value increasing by 7 and 9 (respectively) every 1.2 and 1.8 seconds (respectively) would be represented as a prefix of the infinite time-stamped trace: $\langle(0, \{x \mapsto 12, y \mapsto 12\},\ (1.2, \{x \mapsto 19, y \mapsto 12\}),\ (1.8, \{x \mapsto 19, y \mapsto 21\}), \ldots\rangle$. The set of finite timed traces *Tr* is thus equivalent to $(\mathbb{T} \times \mathcal{I})^*$. We will assume that our variables do not exhibit Zeno-like behaviour (there can only be a finite number of value changes over any finite time interval), which makes the trace models equally expressive as the modelling of variables as a function over time.

Data-Oriented Monitoring. The time-stamped trace identifies the points in time which one can easily capture at runtime. To identify which of these are of interest to our properties, we add two operators $\overleftarrow{e}$ and $\overrightarrow{e}$, which respectively give the value of expression e just before and just after the moment of evaluation. For example, $\overleftarrow{x} \neq \overrightarrow{x}$ identifies the points in time when the value of x has changed. We can then design monitors based on these chosen events using any temporal logic to describe prohibited system traces. For instance, one may use LTL to write the property that the number of downloads may only increase until it is reset[2]: $\Box(\overleftarrow{\mathit{downloads}} \leq \overrightarrow{\mathit{downloads}})\ U\ \mathit{reset}$.

We have explored this approach of data-oriented point-of-interest identification into the runtime verification tool LARVA [2], and applied it to a number of case studies, including data modification intensive red-black tree implementation and an online shopping system SoftSlate[3].

New Points-of-Interest. Sometimes, we need to consider properties over values which change in a continuous manner along time. A typical example of this is the average value of a variable over time e.g. the event characterised when the average value of variable x exceeds 20. Now, if x has held value 10 from time $t = 0s$ till $t = 100s$ when it is assigned to 30, and if no other changes on x occurs, the average will exceed 20 at time $t = 200s$, despite the fact that the system would not have otherwise been interrrupted at that point in time.

To enable identification of such temporal points of interest, we characterise them using an aritmetic expression over the integral of system variables (written $\int v$), going above an upper limit (written $@ \uparrow_n (e)$) or below a lower limit (written $@ \downarrow_n (e)$).

For instance, the event which triggers when the average of x exceeds 20 would be written as $@ \uparrow_{20} (\int x / \int 1)$. Since these integrals increase linearly over time, we can calculate the implicit events when to interrupt the system with every system event. For instance, in the average example, after the system has performed $\langle(0s, \{x \mapsto 10\}), (100s, \{x \mapsto 30\})\rangle$, we can add a timer to cause an event at time 200s, which would be cancelled (or rescheduled) if x changes its value again in the meantime.

Formally, we define a function *earliest*, which given a property (using the integral over time of variables) and a prefix behaviour, will return the earliest time (if any)

[2] Although different approaches are possible, we assume that the inequality is checked upon any system event, including an assignment to any variable.

[3] An open source system available from `http://www.softslate.com`.

when the property would be violated if no other events happen from the system side. It turns out, that for linear and quadratic uses of the integral operator, this can be readily computed.

Discussion and Future Work. We are currently finalising the formalisation of these notions, and looking into extending the expressivity of the integral operator, by adding two special cases: (i) $[\int x]_n$ which takes the integral of x over the past n time units; and (ii) $[\int x]_e$ takes the integral of x since the last occurence of event e. It would be interesting to investigate how these added operators abstract away complex calculations and timer handling in our specification language.

References

1. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, M. Montali, S. Storari, and P. Torroni. Computational logic for run-time verification of web services choreographies: Exploiting the *SOCS-SI* tool. In *Web Services and Formal Methods, Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006, Proceedings*, pages 58–72, 2006.
2. C. Colombo, G. J. Pace, and G. Schneider. Safe runtime verification of real-time properties. In *Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings*, pages 103–117, 2009.
3. M. Leucker and C. Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.
4. O. Maler, D. Nickovic, and A. Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, pages 475–505, 2008.