Communicating Monitors for Hyper-μHML

Antonis Achilleos[1], Elli Anastasiadi[2], Adrian Francalanza[3], and Jana Wagemaker[1]

[1] ICE-TCS, Department of Computer Science, Reykjavik University, Iceland
[2] Department of Information Technology, Uppsala University, Sweden
[3] Department of Computer Science, University of Malta, Msida, Malta
antonios@ru.is, elli.anastasiadi@it.uu.se, adrian.francalanza@um.edu.mt, janaw@ru.is

Abstract

We present a formal definition of distributed monitors that can communicate. This is done with the purpose of instrumenting them on hypertraces and runtime verifying properties of Hyper-μHML which would not be monitorable without the communication.

1 Introduction

Runtime verification (RV) is a verification technique that observes system executions to determine whether some given specification [4] is satisfied or violated. This runtime analysis is usually conducted by a computational entity called a *monitor* [13].

Recently, this verification technique has been extended to parallel setups [5, 8, 14], a large part of the work aims to runtime verify *hyperproperties* [1, 6, 7, 10, 12]. Hyperproperties are properties, or sets, of *hypertraces*, *i.e.* sets of multiple event traces, which encode different local parts of a system execution, or, alternatively, different system executions. The runtime counterpart of this is that several traces are observed against a specification instead of one.

To express hyperproperties, multiple extensions of temporal logics have been defined, interpreted on hypertraces, such as hyper-LTL [9] and a restricted fragment of a variation of the μ-calculus [1]. For instance, in hyperlogics one can express properties such as "If there exists a trace where event 'a' occurs, then there exists another trace where no longer a 'b' occurs", thereby describing dependencies over different traces. It has become of pivotal importance to define logics that express such properties, and provide the corresponding monitors for checking whether an observed hypertrace is in the semantics of a hyperproperty.

In this work we use the logic Hyper-μHML [1, 3] as a specification logic, and we build the foundation for a monitor synthesis procedure for a monitorable fragment [2]. As shown in [1], in order to synthesize monitors for a useful fragment of the logic, it is necessary to extend the monitor capabilities with communication. This is necessary to provide monitors for properties such as the one in the example given above. With this abstract we describe exactly our formalization of this extended communicating monitor setup, which in previous works was only captured informally as protocol descriptions. In this talk we revisit the syntax of Hyper-μHML and discuss part of its semantics over hypertraces to demonstrate our intended monitoring setup. Then we introduce the syntax of communicating monitors, and briefly discuss their semantics. Lastly, we describe the current state of our research and what goals we have in the near and slightly further future.

2 Specification Logic

We define the set of traces $\mathsf{Trc} = \mathsf{Act}^{\omega}$, where Act is a finite set of actions. To define the syntax of our communicating monitors (see Section 3), we need to fix the ID's of the monitors that are communicating. We chose to let this depend on the number of traces that are in the hypertrace we wish to monitor, and as such we study hypertraces of a fixed size. We take index set $I = \{1, \ldots, k\}$ and define hypertraces of a fixed size k as functions from I to Trc. We call the elements of I locations; each location gets assigned a trace.

We consider Hyper-μHML as the logic to specify hyperproperties. The only difference with [1] is the equality and inequality we can express between two traces, which is now supported though our monitor syntax, and thus we chose to include it for the specifications as well. We assume two disjoint, countably infinite sets Π and V of trace variables and recursion variables respectively, and let $\pi, \pi' \ldots \in \Pi$ and $x, y, \ldots \in V$. Formulae $\varphi \in$ Hyper-μHML are constructed as follows:

$$\begin{aligned}
\varphi, \varphi' &::= \exists \pi.\varphi \mid \forall \pi.\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \psi \mid \pi = \pi' \mid \pi \neq \pi' \\
\psi &::= \mathsf{tt} \mid \mathsf{ff} \mid [a_\pi].\psi \mid \langle a_\pi \rangle.\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \max x.\psi \mid \min x.\psi \mid x,
\end{aligned}$$

The semantics is defined over finite sets of sets of traces of size k. We have $\mathsf{HTrc}_I = \{T \mid T \colon I \mapsto \mathsf{Trc}\}$. We require a function $\rho \colon V \mapsto 2^{\mathsf{HTrc}_I}$ assigning sets of hypertraces of size k to recursion variables and an assigment $\sigma \colon \Pi \mapsto I$ assigning a location to trace variables. As each hypertrace is a function assigning traces to locations, σ together with the hypertrace gives us a trace in the specific hypertrace for a trace variable: $T \circ \sigma(\pi) = T(i) = t$ for some $\pi \in \Pi$, $i \in I$, $t \in \mathsf{Trc}$. In this paper we do not present the full semantics, but instead just highlight a few interesting cases.

For two of those cases we need notation for a hypertrace taking a step, i.e. of each trace in the hypertrace we keep only the tail: for a trace at in a hypertrace for some $t \in \mathsf{Trc}$ and $a \in \mathsf{Act}$ what remains after a step of the hypertrace is just t. For $t, t' \in \mathsf{Trc}$ we can write $t \xrightarrow{a} t'$ if $t = at'$. We use A to refer to functions from I to Act. For hypertraces T, T', we further define $T \xrightarrow{A} T'$ if and only if for every $i \in I$, $T(i) \xrightarrow{A(i)} T'(i)$.

Now we present the semantics of boxes, diamonds, the existential and the universal quantifier. The semantics of the box expresses that for all hypertraces T that assign to trace variable π a trace that starts with an a, the hypertrace T' that is a result of T after taking a step (where the trace of π takes an a-step), must satisfy ψ. The reasoning for the diamond is dual. For the existential quantifier we take the union over all possible location assignments of π. For the universal quantifier we take the intersection instead of the union.

$$\begin{aligned}
[\![[a_\pi].\psi, \rho, \sigma]\!] &= \{T \in \mathsf{HTrc}_I \mid \forall A \forall T' ((A(\sigma(\pi)) = a \wedge T \xrightarrow{A} T') \text{ implies } T' \in [\![\psi, \rho, \sigma]\!])\} \\
[\![\langle a_\pi \rangle.\psi, \rho, \sigma]\!] &= \{T \in \mathsf{HTrc}_I \mid \exists A \exists T' (A(\sigma(\pi)) = a \wedge T \xrightarrow{A} T' \wedge T' \in [\![\psi, \rho, \sigma]\!])\} \\
[\![\exists \pi.\varphi, \rho, \sigma]\!] &= \bigcup_{i \in I} [\![\varphi, \rho, \sigma[\pi \mapsto i]]\!] \qquad [\![\forall \pi.\varphi, \rho, \sigma]\!] = \bigcap_{i \in I} [\![\varphi, \rho, \sigma[\pi \mapsto i]]\!]
\end{aligned}$$

3 Monitors

We fix a set of monitor names $\mathsf{name}(\mathcal{M}) = \{1, \ldots, l\}$ with $l \geq k$. We fix a surjective function $f \colon \mathsf{name}(\mathcal{M}) \mapsto I$. We define a communication alphabet $\mathsf{Com} ::= (!G, c), \quad \mid (?G, c)\ , G \subseteq \mathsf{name}(\mathcal{M})$ over some finite alphabet Con with $c \in \mathsf{Con}$, of communication constants. We have a communication letter for sending to a group (multicast), and for receiving from a group. Group receive should be understood as: a monitor receives a message from some subset of monitors, but it does not matter which one. This can be relevant if we have multiple monitors monitoring the same trace in a hypertrace. Communication between individual monitors can be represented by taking singleton sets for G. The syntax of communicating monitors is:

$$\begin{aligned}\mathsf{CMon} \ni M, N &::= M \vee N \mid M \wedge N \mid \ [m]_i \\ \mathsf{Mon} \ni m, n &::= \mathit{yes} \mid \ \mathit{no} \mid \ \mathit{end} \mid a.m \mid c.m \mid m + n \mid \mathsf{rec}\ x.m \ ,\end{aligned}$$

with $c \in \mathsf{Con}$, $a \in \mathsf{Act}$, $m \in \mathsf{Mon}$ and $i \in \mathsf{name}(\mathcal{M})$. We require that for our communicating monitors, each name appears at most once. With $[m]_i$ we denote that the monitor with name i in M is in state m. In the talk we will present part of the formalisation of the semantics in terms of transition rules. Here we just give some of the intuition. A communicating monitoring set-up (an element of CMon) is similar to the set-up presented in [1], where a group of monitors is seen as a circuit composed of a hierarchy of gates. Each trace $t \in T$ is assigned a fixed set of monitors in Mon that correspond to the local properties to be verified and are at the bottom layer of the structure. Monitors assigned to the same trace run in parallel and observe identical events. Contrary to [1], monitors assigned to different traces are no longer completely isolated from each other, but can now communicate. Verdict combination is still the same as in the circuit model [1] and is synchronous. We give here an example that demonstrates all concepts presented here working together. Let $\mathsf{Act} = \{a, b\}$, $I = \{i, j\}$ and $\mathsf{name}(\mathcal{M}) = \{1, 2\}$. We monitor the formula

$$\exists\pi\exists\pi'(\min\ x.(\langle a_\pi\rangle.\min\ y.(\langle b_{\pi'}\rangle.\mathsf{tt} \vee \langle a_{\pi'}\rangle.y) \vee \langle b_\pi\rangle.x)) \ .$$

This formula states that some trace exhibits an event "a" and later a trace sees a "b". The communicating monitor set-up for this formula is as follows:

$$\begin{aligned}M ::= &[\mathsf{rec}\ x.(a.\mathsf{rec}\ y.(a.(?\{2\}, b).\mathit{yes} + b.\mathit{yes} + a.y) + b.(!\{2\}, b).x)]_1 \\ &\vee [\mathsf{rec}\ x.(a.\mathsf{rec}\ y.(a.(?\{1\}, b).\mathit{yes} + b.\mathit{yes} + a.y) + b.(!\{1\}, b).x)]_2 \ .\end{aligned}$$

M should be such that M on input hypertrace T of size 2 outputs *yes* only if T has a trace that sees an a and a trace that later sees a b.

4 Future Work

The next step is to determine fragments of the logic for which we can have a synthesis function s such that $s(\varphi)$ is a sound and/or violation-complete monitor for φ (see definitions in [1]). This entails that for any hypertrace T, if $T \in [\![\varphi]\!]$ then $s(\varphi)$ will reject T (violation-completeness), and when $s(\varphi)$ reaches verdict *yes* (*no*) on input T, $T \in [\![\varphi]\!]$ ($T \notin [\![\varphi]\!]$, respectively). Such a synthesis of course might not provide us with *optimal*, or even efficient, monitors. Indeed, as discussed in [3], two or more circuit monitors can be violation- (or satisfaction- etc) complete for the same specification and one of them might require an unrealistic amount of communication to take place at runtime. We would like thus to establish a methodology for proving that an arbitrary monitoring setup is indeed monitoring of a given property so that we can compare between different monitoring setups without jeopardizing the correctness or the monitors themselves and ultimately use this for guaranteeing the optimality of our synthesis.

Such an attempt requires a framework for describing the accumulation of knowledge between different agents (monitors), and thus, as discussed in [3] we aim to utilize dynamic epistemic logic [11] to describe this event driven accumulation of knowledge in our monitors.

Once that is in place, an important next step is to study the extension of our specification logic to allow quantifiers inside the scope of fixpoints. This will allow us to express properties such as "at every step, at least one trace has an a", which could be written as max $x.\exists\pi.(\langle a_\pi\rangle x)$. This is a formula for which our monitor setup is able to detect violations, but that is currently not in the syntax of the specification logic.

References

[1] Luca Aceto, Antonis Achilleos, Elli Anastasiadi, and Adrian Francalanza. Monitoring hyperproperties with circuits. In Mohammad Reza Mousavi and Anna Philippou, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 42nd IFIP WG 6.1 International Conference, FORTE 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings*, volume 13273 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2022.

[2] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Adventures in monitorability: from branching to linear time and back again. *Proc. ACM Program. Lang. POPL*, 3(52):1–29, 2019.

[3] Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza, and Jasmine Xuereb. Epistemic logic for verifying runtime verification communication protocols.

[4] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In Ezio Bartocci and Yliès Falcone, editors, *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*, pages 1–33. Springer, 2018.

[5] Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory*, pages 162–176, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[6] Borzoo Bonakdarpour and Bernd Finkbeiner. Runtime verification for hyperltl. In Yliès Falcone and César Sánchez, editors, *Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*, volume 10012 of *Lecture Notes in Computer Science*, pages 41–45. Springer, 2016.

[7] Borzoo Bonakdarpour and Bernd Finkbeiner. The complexity of monitoring hyperproperties. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 162–174. IEEE Computer Society, 2018.

[8] Ian Cassar, Adrian Francalanza, Claudio Antares Mezzina, and Emilio Tuosto. Reliability and fault-tolerance by choreographic design. In Adrian Francalanza and Gordon J. Pace, editors, *Proceedings Second International Workshop on Pre- and Post-Deployment Verification Techniques, PrePost@iFM 2017, Torino, Italy, 19 September 2017*, volume 254 of *EPTCS*, pages 69–80, 2017.

[9] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014.

[10] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010.

[11] Hans van Ditmarsch, Wiebe van der Hoek, and Barteld Kooi. *Dynamic Epistemic Logic*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[12] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. *Formal Methods Syst. Des.*, 54(3):336–363, 2019.

[13] Adrian Francalanza. A Theory of Monitors. *Inf. Comput.*, 281:104704, 2021.

[14] Claudio Antares Mezzina and Jorge A. Pérez. Causally consistent reversible choreographies: A monitors-as-memories approach. In *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming*, PPDP '17, page 127–138, New York, NY, USA, 2017. Association for Computing Machinery.