Monitoring Hyperproperties with Circuits *

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

[1] ICE-TCS, Department of Computer Science, Reykjavik University, Iceland
[2] Department of Computer Science, University of Malta Msida, Malta
antonios@ru.is, elli19@ru.is, adrian.francalanza@um.edu.mt,

Abstract

We define Hyper-μHML and its semantics over sets of traces. We then propose a monitorable fragment of that logic and we present a monitoring model that can be used to monitor specifications in that fragment in a sound and complete way.

1 Introduction

The field of runtime verification revolves around the effort to provide methods for checking whether a system satisfies its intended specification at runtime. This is done through a computing device called a *monitor* that observes the current run of a system in the form of a trace [3, 10]. Runtime monitoring has recently been extended to the setting of concurrent systems [1, 6, 4, 13] with several attempts to specify properties over sets of traces, and the consequent adaptation of the monitoring setup [5, 9, 2].A centerpiece in this line of work has been the specification logic Hyper-LTL [7]. Intuitively Hyper-LTL allows for existential and universal quantification over a set of traces (which describes the set of observed system runs). The properties over one trace are stated in LTL, with open trace variables, and then made dependent on properties of other traces via the quantification.

We define the specification logic Hyper-μHML, as a counterpart to Hyper-LTL, due to the good behaviors of μHML with respect to monitorability (namely the easiness of monitors to be mapped to it and virse-versa, along with the easiness to define complete fragments of it) [1, 11]. However, just like Hyper-LTL, Hyper-μHML can define dependencies over different traces, which intuitively causes extra delays in the processing of traces as the properties observed on one of them can impact what is expected for another. In existing research, the runtime verification of such properties, is dealt with via a plethora of modifications and assumptions made over the monitoring setup, such as being able to restart an execution or having access to all executions of a system.

In our case we define monitors engineered for a very restricted fragment of the specification language, by utilizing circuit-like structures to combine verdicts over different traces. The fragment of the logic restricts the amount of quantification that can be applied to the single trace properties and thus limits the dependencies between them. This naturally induces circuits with monitors from [1] as input nodes and simple kinds of gates at the higher levels, with the resulting structure belonging to the complexity class AC_0, which is considered a class of problems that can be solved efficiently using parallel computation [12]. This monitor model keeps the processing-at-runtime cost bounded by a constant, while still providing correctness guarantees.

The properties for which we obtain monitors, are properties over sets of traces and thus they are not bound by whether those traces are produced by the same system (something that

*The work reported in this paper is supported by the projects 'Open Problems in the Equational Logic of Processes' (OPEL) (grant 196050-051) and the project "MoVeMnt: Mode(l)s of Verification and Monitorability" (grant no 217987) of the Icelandic Research Fund.

would bring this logic closer to CTL), although such an assumption can add extra layers of semantic meaning to the trace properties.

2 The logic

Syntax: Our logic is defined in the style of Hyper-LTL defined in [7]. The quantification among traces remains the same, but the language in which local trace properties are stated is μHML. We consider the following restriction to a multi-trace sHML logic (the *safety* fragment of μHML [1]), with no alternating quantifiers, called Hyper^1-sHML. We can similarly define the cHML (co-safety) fragment, and the HML fragment.

Definition 1. *Our formulae are constructed by the following grammar:*

$$\varphi \in \mathit{Hyper}^1\text{-sHML} ::= \exists_\pi \psi \quad | \quad \forall_\pi \psi \quad | \quad \varphi \sqcup \varphi \quad | \quad \varphi \sqcap \varphi \ ,$$

and ψ stands for a formula in sHML*. π is a trace variable from an infinite supply of trace variables $\mathcal{V}$. The existential and universal quantification happens over a common, finite set of infinite traces T which is defined over a set of actions* Act*. $\sqcup$ and $\sqcap$ stand for the regular $\vee$ and $\wedge$ boolean connectives, and can only be used at the top level in formulae. Their use allows us to keep the synthesis function below clearer.*

Example 1. *Over the set of actions $\{a, b\}$, the formula $\forall_\pi[a]\mathtt{ff} \sqcap \exists_\pi[b](max\ x.([a]\mathtt{ff} \ \wedge \ [b]x))$ states that none of the traces in T start with a, and at least one trace in T starts with b, but never encounters a.*

Semantics: The semantics of Hyper-μHML is given over a set of traces T. It suffices to define the extension of μHML linear-time semantics ([1]) to the Hyper-μHML semantics which is done in the style of Hyper-LTL. The semantics of Hyper^1-sHML can be inferred thus as a syntactic subset of Hyper-μHML.

Remark 1. *We note here that the more interesting hyperproperties in the literature requite at least two quantifiers to be stated. Some of them might be able to be projected into the Hyper^1-*sHML *fragment, but this procedure is not formally yet defined, or trivial. Thus, our main purpose would be to extend the monitors, and not project formulate case-by-case onto this fragment. The main objective of this fragment and the following synthesis for it is to establish a baseline which we will attempt to in future work, to capture more intricate formulae.*

3 The monitors

Syntax of the Monitors The collection Cmon of circuit monitors is the set of terms generated by the following grammar:

$$\begin{array}{llllll} M \in C\textsc{mon} ::= \bigvee [m]_k & | \ \bigwedge [m]_k & | \ M \vee M & | \ M \wedge M & \\ m ::= \mathit{yes},\ \mathit{no},\ \mathit{end} & | \ a.m & | \ m + n & | \ \mathit{rec}\ x.m & | \ x \end{array}$$

The notation $[m]_k$ corresponds to the parallel dispatch of k identical monitors m, where k is the number of available traces. As this is part of a circuit definition, this operation induces a *family* of circuits that are the monitor. Just like in circuit theory then, the circuit used for a particular instance of the problem is selected based on the amount of available input nodes (in our case traces).

Semantics: The semantics of circuit monitors is given over a set of traces T. Each trace $t \in T$ is assigned a set of identical (regular) monitors that correspond to the local properties to be verified. These monitors run in parallel (in the sense of parallel monitors in [1]), and monitors assigned to the same trace observe identical events, while the set of monitors assigned to another trace also run in parallel but completely isolated from other traces. The monitors communicate their verdicts with the appropriate higher level gates of the circuit when those verdicts are reached. The top level of this syntax defines a circuit-like structure, that inputs verdicts from the bottom layer monitors and processes them though logical gates. The processing of verdicts is defined as (we only present the case of large *or* and *and* gates the small ones can be inferred from those):$\bigwedge[m]_k$ is evaluated to *yes*, if all monitors are in a *yes* state, to *no*, if for at least one monitor is evaluate to *no*, and to *end*, if none of the above happens, but all m monitors reach some verdict. The evaluation of $\bigvee[m]_k$ is symmetric, and the evaluation of the $\vee$ and $\wedge$ gates over them is following the same rules.

3.1 Synthesis

Given a formula φ in Hyper^1-sHML, We synthesize a circuit monitor M for it,over the set of traces T, through the following recursive function $Syn(-)$: Hyper^1-sHML $\rightarrow$Cmon.

Definition 2 (Circuit Monitor Synthesis)**.**

$$Syn(\exists_\pi \varphi) = \bigvee [m(\varphi)]_{|T|} \text{ with } \varphi \in \text{sHML} \quad | \quad Syn(\forall_\pi \varphi) = \bigwedge [m(\varphi)]_{|T|} \text{ whith } \varphi \in \text{sHML}$$
$$Syn(\varphi_1 \sqcup \varphi_2) = Syn(\varphi_1) \vee Syn(\varphi_2) \quad | \quad Syn(\varphi_1 \sqcap \varphi_2) = Syn(\varphi_1) \wedge Syn(\varphi_2)$$

Where $m(-)$ is the monitor synthesis function for sHML *defined in [1].*

Soundness: The correctness of the synthesis function can be obtained by straightforward induction on the form of φ. Since sHML formulae always have a violation complete monitor, we can extend this result to:

Proposition 1. *For a formula $\varphi \in$Hyper^1-sHML, we have that $Syn(\varphi)$ is a violation complete monitor for it.*

The above result can also be easily extended to the relevant notions of completeness, for different fragments of Hyper-μHML.

Future work: We expect that the fragment Hyper^1-sHML is maximal towards violation completeness, which means that any monitor in Cmon is monitoring for a formula in Hyper^1-sHML. Towards proving this, we aim to define a sound synthesis, from monitors to formulae in Hyper^1-sHML. However, the ultimate goal of this work, is to extend the amount of monitorable properties, by allowing at least one level of alternating quantifiers. If the maximality result is true for these monitors, which we believe to be the case, then the only way to do so would be by augmenting the monitor syntax. Our approach to this extension would be allowing a notion of synchronization rounds among the monitors (or respectively a round of communication). This would enable more complex processing of sets of traces, as now we would be able to infer some information for properties of a given trace, based on the ones monitored for on a different one. However, the analysis of communications among the monitors is a complicated extension, as their exact content plays a significant role to our understanding of the system, as well as the processing at runtime cost. We plan to implement this therefore by utilizing dynamic epistemic logic [8] in order to perform this extension formally and soundly.

References

[1] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Adventures in monitorability: From branching to linear time and back again. *Proceedings of the ACM on Programming Languages*, 3(POPL):52:1–52:29, 2019.

[2] Shreya Agrawal and Borzoo Bonakdarpour. Runtime verification of k-safety hyperproperties in hyperltl. *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 239–252, 2016.

[3] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification.

[4] 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.

[5] Borzoo Bonakdarpour and Bernd Finkbeiner. The complexity of monitoring hyperproperties. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 162–174, 2018.

[6] Ian Cassar, Adrian Francalanza, Claudio Antares Mezzina, and Emilio Tuosto. Reliability and fault-tolerance by choreographic design. In *PrePost@iFM*, 2017.

[7] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *2008 21st IEEE Computer Security Foundations Symposium*, pages 51–65, 2008.

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

[9] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. pages 190–207, 09 2017.

[10] Adrian Francalanza, Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Ian Cassar, Dario Della Monica, and Anna Ingólfsdóttir. A foundation for runtime monitoring. In Shuvendu K. Lahiri and Giles Reger, editors, *Runtime Verification - 17th International Conference, RV 2017*, volume 10548 of *Lecture Notes in Computer Science*, pages 8–29. Springer, 2017.

[11] Adrian Francalanza, Luca Aceto, and Anna Ingolfsdottir. Monitorability for the hennessy–milner logic with recursion. *Formal Methods in System Design*, 51, 08 2017.

[12] Johan Håstad. *Computational Limitations of Small-Depth Circuits*. MIT Press, Cambridge, MA, USA, 1987.

[13] 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.