Runtime Verification for HyperLTL

Borzoo Bonakdarpour[1(✉)] and Bernd Finkbeiner[2]

[1] McMaster University, Hamilton, Canada
borzoo@mcmaster.ca
[2] Saarland University, Saarbrücken, Germany
finkbeiner@cs.uni-saarland.de

Abstract. Information flow security often involves reasoning about multiple execution traces. This subtlety stems from the fact that an intruder may gain knowledge about the system through observing and comparing several executions. The monitoring of such properties of sets of traces, also known as hyperproperties, is a challenge for runtime verification, because most monitoring techniques are limited to the analysis of a single trace. In this tutorial, we discuss this challenge with respect to HyperLTL, a temporal logic for the specification of hyperproperties.

1 Security Policies and Hyperproperties

Runtime verification (RV) is traditionally concerned with the monitoring of *trace properties* such as those expressed in linear-time temporal logic (LTL). Observing a growing prefix of a trace, we determine if the trace belongs to the set of traces that is characterized as correct by the specification.

Information flow security policies usually do not fit this pattern, because they express a relation between multiple traces. Noninterference, for example, requires that two traces that may differ in their high-security inputs, but have the same low-security inputs, must have the same low-security outputs. Such properties are therefore not properties of individual traces, but properties of sets of traces, also known as *hyperproperties*. This is not a matter of linear vs. branching time, as noninterference cannot even be expressed in branching-time temporal logics, such as CTL, CTL* or the modal μ-calculus [2,11]; the challenge, rather, is that information flow properties can be considered as properties on a system that results from the parallel composition of multiple copies of the original system [4,18].

Clarkson and Schneider proposed the notion of *hyperproperties* to account for properties that relate multiple executions of a system [7]. They showed that the class of hyperproperties comprises many of the properties proposed in the literature. A hyperproperty H is defined as a set of sets of executions traces, and a system is defined to satisfy H, if its set of execution traces is an *element* of H. Noninterference between an input h and an output o is, for example, the hyperproperty consisting of all sets of traces, in which all traces that only differ in h have the same output o at all times.

Y. Falcone and C. Sanchez (Eds.): RV 2016, LNCS 10012, pp. 41–45, 2016.
DOI: 10.1007/978-3-319-46982-9_4

2 HyperLTL

Since hyperproperties cannot be expressed in the classic temporal logics like LTL or CTL*, several extensions of the temporal logics have been proposed. Balliu et al. encoded several standard information flow policies in epistemic temporal logics [3], which allows us to specify properties in terms of the knowledge of agents. Another temporal logic that is sufficiently expressive to encode certain information flow policies is SecLTL, which specifies how information flow requirements change over time and in response to events in the system [8]. We focus here on the temporal logic HyperLTL [6,12], which adds explicit and simultaneous quantification over multiple traces to LTL. Compared to previous logical frameworks, HyperLTL significantly extends the range of security policies under consideration, including complex information-flow properties like generalized noninterference, declassification, and quantitative noninterference.

Let AP be a set of *atomic propositions*, and let $\mathcal{V}$ be a set of trace variables. The syntax of HyperLTL is given by the following grammar:

$$\begin{aligned} \psi &::= \exists \pi.\ \psi \mid \forall \pi.\ \psi \mid \varphi \\ \varphi &::= a_\pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi \end{aligned}$$

where $a \in AP$ is an atomic proposition and $\pi \in \mathcal{V}$ is a trace variable. Note that atomic propositions are indexed by trace variables. The quantification over traces makes it possible to express properties like "on all traces ψ must hold", which is expressed by $\forall \pi.\ \psi$. Dually, one can express that "there exists a trace such that ψ holds", which is denoted by $\exists \pi.\ \psi$. We use the usual derived Boolean connectives. The derived temporal operators $\Diamond$, $\Box$, and $\mathcal{W}$ are defined as for LTL: $\Diamond \varphi \equiv \mathit{true} \mathcal{U} \varphi$, $\Box \varphi \equiv \neg \Diamond \neg \varphi$, and $\varphi_1 \, W \varphi_2 \equiv (\varphi_1 \mathcal{U} \varphi_2) \vee \Box \varphi_1$. We call a HyperLTL formula ψ (quantifier) *alternation-free* iff the quantifier prefix only consists of either only universal or only existential quantifiers.

It has been shown that many hyperproperties of interest can be expressed in HyperLTL [6,12,16]. For many properties, it in fact suffices to use the alternation-free fragment of HyperLTL. The following are two typical examples:

- *Observational determinism* [19] requires that every pair of traces with the same initial low observation remain indistinguishable for low users. That is, the program appears to be deterministic to low-security users. Observational determinism can be expressed in HyperLTL as follows:

$$\forall \pi . \forall \pi' .\ (\mathit{lowIn}_\pi \Leftrightarrow \mathit{lowIn}_{\pi'}) \ \Rightarrow \ \Box(\mathit{lowOut}_\pi \Leftrightarrow \mathit{lowOut}_{\pi'}),$$

  where *lowIn* and *LowOut* are atomic propositions representing the low-security inputs and outputs, respectively.
- *Shamir's secret sharing scheme* [17] is the following policy: A system stores a secret by splitting it into k shares. The requirement is that not all of the k shares are revealed:

$$\forall \pi_1 \ \ldots \ . \forall \pi_k .\ (\Box(\neg sr^1_{\pi_1} \wedge \cdots \wedge \neg sr^1_{\pi_k}) \ \vee \ \ldots \ \vee \ \Box(\neg sr^k_{\pi_1} \wedge \cdots \wedge \neg sr^k_{\pi_k})),$$

where the atomic proposition sr^i, $i \in [1, k]$, means that share i of the secret has been revealed.

The *satisfiability problem* of HyperLTL formulas is in general undecidable, but decidable for the fragment without quantifier alternations and for the $\exists^*\forall^*$-fragment. Since, in practice, many HyperLTL specifications only contain universal quantifiers, this means that the satisfiability, implication, and equivalence of such specifications can be checked automatically [10]. The *model checking problem* of HyperLTL formulas over finite-state Kripke structures is decidable for the full logic, and has, in fact, the same complexity (PSPACE-complete) as standard LTL model checking for the alternation-free fragment. MCHyper is an efficient tool implementation for hardware model checking against alternation-free HyperLTL formulas [12]. Beyond finite-state systems, it was recently shown that a first-order extension of HyperLTL can be checked automatically over workflows with arbitrarily many agents [13].

3 Runtime Verification for HyperLTL

For *runtime verification*, it is necessary to define finite-trace semantics for HyperLTL. Analogously to the three-valued semantics of LTL [5], such a finite-trace semantics for HyperLTL can be defined based on the truth values $\mathbb{B}_3 = \{\top, \bot, ?\}$. In this semantics, "?" means that for the given formula φ and the current set M of finite execution traces at run time, it is not possible to tell whether M satisfies or violates φ; i.e., both cases are possible in this or future extensions and/or executions.

Let M be a finite set of finite traces. The truth value of a closed HyperLTL formula φ with respect to M, denoted by $[M \models \varphi]$, is an element of the set $\mathbb{B}_3 = \{\top, \bot, ?\}$, and is defined as follows:

$$[M \models \varphi] = \begin{cases} \top & \text{if } \forall \text{ sets } T \text{ of infinite traces with } M \leq T,\ T \text{ satisfies } \varphi \\ \bot & \text{if } \forall \text{ sets } T \text{ of infinite traces with } M \leq T,\ T \text{ does not satisfy } \varphi \\ ? & \text{otherwise,} \end{cases}$$

where $\leq$ is a prefix relation on sets of traces defined as follows. Let u be a finite trace and v be a finite or infinite trace. We denote the concatenation of u and v by $\sigma = uv$. Also, $u \leq \sigma$ denotes the fact that u is a prefix of σ. If U is a set of finite traces and V is a finite or infinite set of traces, then $U \leq V$ is defined as $U \leq V \ \equiv \ \forall u \in U.\ (\exists v \in V.\ u \leq v)$. Note that V may contain traces that have no prefix in U.

Pnueli and Zaks [15] characterize an LTL formula φ as *monitorable* for a finite trace u, if u can be extended to one that can be evaluated with respect to φ at run time. For example, the LTL formula $\Box\Diamond p$ is not monitorable, since there is no way to tell at run time whether or not p will be visited infinitely often in the future. By contrast, safety (e.g., $\Box p$) and co-safety (e.g., $\Diamond p$) LTL formulas are monitorable. We can extend the concept of LTL-monitorability to HyperLTL

by requiring that every finite set U of finite traces can be extended to a finite set V of finite traces such that every trace in U is the prefix of some trace in V and that V evaluates to $\top$ or $\bot$. It is easy to see that an alternation-free HyperLTL formula with monitorable inner LTL subformula is also monitorable. For example, observational determinism and Shamir's secret sharing scheme are both monitorable. Note, however, that only violations of such formulas can be detected at run time (detecting their satisfaction requires examining all traces of the system under inspection, which is a model checking problem).

A monitor for a HyperLTL formula must match the observed traces with the quantifiers of the HyperLTL formula and ensure that the inner LTL subformula is satisfied on the combined trace. For alternation-free HyperLTL formulas, this can be done by creating a monitor automaton for the LTL subformulas that are inter-trace independent, then *progressing* inter-trace dependent subformulas for each observed trace, and finally building a monitor automaton for each progressed formula [1]. This approach has proven successful on complex data sets, such as the GPS location data of 21 users taken over a period of eight weeks in the region of Seattle, USA. However, there is clear potential for further optimization, for example, by analyzing the observed execution trace in relation to an abstract model of the system at run time (cf. [9]). Another important line of work is the extension of the approach to a distributed monitoring framework (cf. [14]).

Acknowledgment. This work was partially supported by the German Research Foundation (DFG) in the Collaborative Research Center 1223 and by Canada NSERC Discovery Grant 418396-2012 and NSERC Strategic Grants 430575-2012 and 463324-2014.

References

1. Agrawal, S., Bonakdarpour, B.: Runtime verification of k-safety hyperproperties in hyperltl. In: Proceedings of the 29th IEEE Computer Security Foundations Symposium (CSF) (2016, to appear)
2. Alur, R., Černý, P., Zdancewic, S.: Preserving secrecy under refinement. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 107–118. Springer, Heidelberg (2006). doi:10.1007/11787006_10
3. Balliu, M., Dam, M., Guernic, G.L.: Epistemic temporal logic for information flow security. In: Proceedings of the 2011 Workshop on Programming Languages and Analysis for Security, PLAS 2011, San Jose, CA, p. 6, June 2011
4. Barthe, G., D'Argenio, P.R., Rezk, T.: Secure information flow by self-composition. In: CSFW, pp. 100–114 (2004)
5. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20**(4), 14 (2011)
6. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: Abadi, M., Kremer, S. (eds.) POST 2014. LNCS, vol. 8414, pp. 265–284. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54792-8_15
7. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**(6), 1157–1210 (2010)

8. Dimitrova, R., Finkbeiner, B., Kovács, M., Rabe, M.N., Seidl, H.: Model checking information flow in reactive systems. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 169–185. Springer, Heidelberg (2012). doi:10.1007/978-3-642-27940-9_12
9. Dimitrova, R., Finkbeiner, B., Rabe, M.N.: Monitoring temporal information flow. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012. LNCS, vol. 7609, pp. 342–357. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34026-0_26
10. Finkbeiner, B., Hahn, C.: Deciding hyperproperties. In: Proceedings of the CONCUR 2016 (2016)
11. Finkbeiner, B., Rabe, M.N.: The linear-hyper-branching spectrum of temporal logics. IT Inform. Technol. **56**(6), 273–279 (2014)
12. Finkbeiner, B., Rabe, M.N., Sanchez, C.: Algorithms for model checking HyperLTL and HyperCTL*. In: Proceedings CAV 2015 (2015)
13. Finkbeiner, B., Seidl, H., Müller, C.: Specifying and verifying secrecy in workflows with arbitrarily many agents. In: Proceedings of the ATVA 2016 (2016)
14. Mostafa, M., Bonakdarpour, B.: Decentralized runtime verification of LTL specifications in distributed systems. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 494–503 (2015)
15. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 573–586. Springer, Heidelberg (2006). doi:10.1007/11813040_38
16. Rabe, M.N.: A Temporal Logic Approach to Information-flow Control. Ph.D. thesis, Saarland University (2016)
17. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
18. Terauchi, T., Aiken, A.: Secure information flow as a safety problem. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, pp. 352–367. Springer, Heidelberg (2005). doi:10.1007/11547662_24
19. Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: Proceedings IEEE Computer Security Foundations Workshop, pp. 29–43, June 2003