Visualizing Game-Based Certificates for Hyperproperty Verification

Raven Beutner, Bernd Finkbeiner, and Angelina Göbl

CISPA Helmholtz Center for Information Security, Germany
{raven.beutner,finkbeiner,angelina.goebl}@cispa.de

Abstract. Hyperproperties relate multiple executions of a system and are commonly used to specify security and information-flow policies. While many verification approaches for hyperproperties exist, providing a convincing *certificate* that the system satisfies a given property is still a major challenge. In this paper, we propose *strategies* as a suitable form of certificate for hyperproperties specified in a fragment of the temporal logic HyperLTL. Concretely, we interpret the verification of a HyperLTL property as a game between universal and existential quantification, allowing us to leverage strategies for the existential quantifiers as certificates. We present `HyGaViz`, a browser-based visualization tool that lets users interactively explore an (automatically synthesized) witness strategy by taking control over universally quantified executions.

1 Introduction

Hyperproperties [17] relate multiple execution traces of a system and occur frequently when reasoning about information flow [38,35], robustness [12,15], independence [3], knowledge [14,10], and causality [19,25]. A popular logic for specifying temporal hyperproperties is HyperLTL [16], an extension of LTL with explicit quantification over execution traces. For example, we can use HyperLTL to express a simple non-interference property as follows:

$$\forall \pi_1. \exists \pi_2. \Box(l_{\pi_1} \leftrightarrow l_{\pi_2}) \land \Box(o_{\pi_1} \leftrightarrow o_{\pi_2}) \land \Box(\neg h_{\pi_2}) \qquad (\varphi_{NI})$$

Informally, this property – called non-inference [33] – requires that any possible observation made via the low-security input (modeled via atomic proposition l) and output (o) is compatible with a fixed "dummy" sequence of high-security inputs (h) [33]. Concretely, φ_{NI} states that for *any* execution π_1, *some* execution π_2 combines the low-security observations of π_1 with fixed dummy values for h; here, we require that h is constantly set to false, i.e., $\Box(\neg h_{\pi_2})$ (cf. [23]).

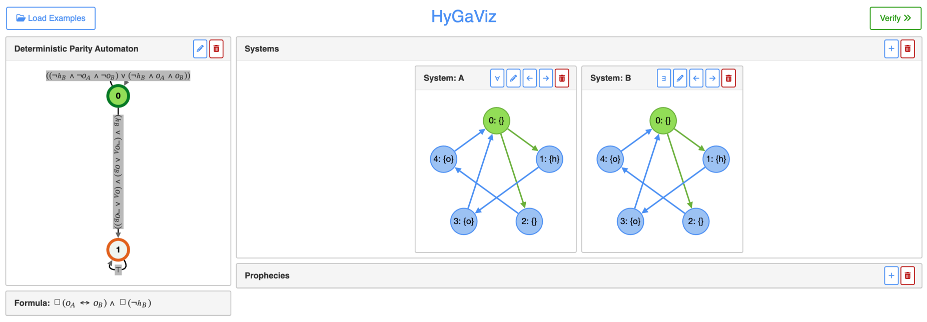
Verification and Certificates. In recent years, many verification techniques for temporal hyperproperties (expressed, e.g., in HyperLTL) have been developed [16,24,8,30,2,9,35]. However, while *checking* if a given system satisfies a HyperLTL property is important, an often equally critical aspect is to convince the user of this satisfaction using explainable *certificates*. For trace properties –

specified, e.g., in LTL – user-understandable certificates for positive and negative verification results have been explored extensively [32,27,4,5,13,28]. Likewise, for alternation-free HyperLTL formulas (i.e., formulas that use a single type of quantifier), known techniques for LTL apply [29]. In contrast, generating explainable certificates for the satisfaction of alternating properties like φ_{NI} is more complex. For example, φ_{NI} states that for any trace π_1, there exists some matching execution π_2. A certificate must thus implicitly define a mapping that, given a concrete choice for π_1, produces a witness trace π_2. Defining and understanding such a mapping can be complex, even for simple systems with few states.

Strategies as Certificates. In this paper, we propose *strategies* as certificates for the satisfaction of $\forall^*\exists^*$ HyperLTL formulas (i.e., formulas where an arbitrary number of universal quantifiers is followed by an arbitrary number of existential quantifiers; e.g., φ_{NI}). To accomplish this, we take a game-based verification perspective [20,6]. The key idea is to interpret the verification of a $\forall \pi_1.\exists \pi_2.\, \psi$ formula (where ψ is the LTL body) as a game between universal and existential quantification. The $\forall$-player controls the universally quantified trace by moving through the system (thereby producing a trace π_1), and the $\exists$-player reacts with moves in a separate copy of the system (thereby producing a trace π_2). Any strategy for the $\exists$-player that ensures that π_1 and π_2, together, satisfy ψ, implies that the formula is satisfied on the given system. We can think of a winning strategy as a step-wise Skolem function that, for every trace π_1, iteratively constructs a witnessing trace π_2.

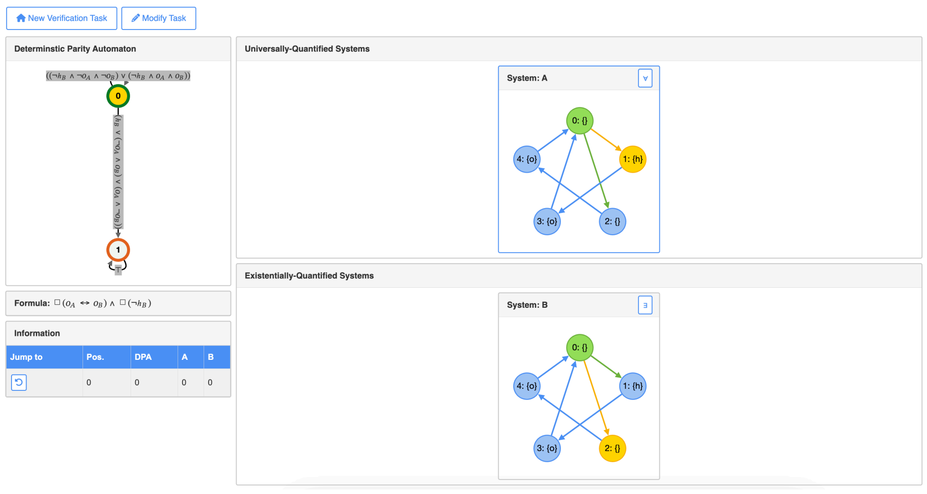
Visualizing Strategies. In this paper, we introduce `HyGaViz`, a verification and visualization tool for strategies in the context of HyperLTL verification. In `HyGaViz`, the user can input (possibly identical) finite-state transition systems and a HyperLTL formula φ. `HyGaViz` then automatically attempts to synthesize a strategy that witnesses the satisfaction of φ. If a strategy exists, `HyGaViz` displays it to the user. Our key insight is that we can let the user explore the strategy interactively by taking control of universally quantified traces. That is, instead of displaying the strategy in its entirety (e.g., as a table or decision diagram), we let the user play a game. In each step of the game, the user decides on a successor state for each universally quantified system (i.e., the user takes the role of the $\forall$-player), and `HyGaViz` automatically updates the states of all existentially quantified systems (i.e., `HyGaViz` plays the role of the $\exists$-player).

Example 1. We consider a simple verification instance in Figure 1. On `HyGaViz`'s initial page (Figure 1a), we create two (in this case, equal) transition systems (labeled A, B) over atomic propositions (APs) o and h, depicted in the top right. In each system, each state is identified by a natural number and lists all APs that hold in the given state. From initial state 0, the system can branch on AP h (states 1 and 2), but, in either case, AP o is set in the next step (states 3 and 4). We want to verify φ_{NI}, which – due to the absence of low-security input l – simplifies to $\forall A.\exists B.\square(o_A \leftrightarrow o_B) \land \square(\neg h_B)$. Note how, in `HyGaViz`, the quantifier prefix is determined implicitly by the order and quantifier type of the


Load Examples
HyGaViz
Verify »
Deterministic Parity Automaton
$((\neg h_B \wedge \neg o_A \wedge \neg o_B) \vee (\neg h_B \wedge o_A \wedge o_B))$
0
$(h_B \vee (\neg o_A \wedge o_B) \vee (o_A \wedge \neg o_B))$
1
$\top$
Systems
System: A
System: B
0: {}
1: {h}
2: {}
3: {o}
4: {o}
Prophecies

Formula: $\square(o_A \leftrightarrow o_B) \wedge \square(\neg h_B)$

(a)


New Verification Task
Modify Task
Deterministic Parity Automaton
$((\neg h_B \wedge \neg o_A \wedge \neg o_B) \vee (\neg h_B \wedge o_A \wedge o_B))$
0
$(h_B \vee (\neg o_A \wedge o_B) \vee (o_A \wedge \neg o_B))$
1
$\top$
Universally-Quantified Systems
System: A
Existentially-Quantified Systems
System: B
0: {}
1: {h}
2: {}
3: {o}
4: {o}

Formula: $\square(o_A \leftrightarrow o_B) \wedge \square(\neg h_B)$

Information

Jump to	Pos.	DPA	A	B
⟲	0	0	0	0

(b)

Fig. 1: Screenshots of `HyGaViz`.

systems, and the LTL body is displayed on the bottom left. The user can change the systems, the quantification type, the name, and the order of the systems using the buttons above each system. Upon entering the LTL formula, `HyGaViz` automatically displays a deterministic automaton for the property (top left). After clicking the *Verify* button (top right), the user is directed to the strategy simulation page (depicted in Figure 1b). During the simulation, `HyGaViz` displays the current state of the automaton and the system state for A and B (in green) and lets the user control the state of (the universally quantified) system A. By hovering over the successor state of system A, `HyGaViz` highlights the next state for system B (in yellow). In this instance, systems A and B are both in state 0. When the user moves system A to state 1, `HyGaViz` reacts by moving system B to state 2 (as it has to ensure $\square(\neg h_B)$). By clicking on a successor state for A, the user locks the choice, and the game progresses to the next round. $\triangle$

Related Work. `HyperVis` [29] is a tool for the visualization of counterexample traces for alternation-free $\forall^k$ formulas. Notably, a counterexample to a $\forall^k$ property is a concrete list of k traces, so visualization is possible by highlighting the relevant parts of the traces, potentially using causality-based techniques [18]. Our visualization for properties involving quantifier alternations is rooted in the game-based verification approach for HyperLTL [20,6], which becomes complete when adding prophecies [6] (see Section 2.2). To the best of our knowledge, we are the first to propose a principled approach to generate and visualize user-understandable certificates for alternating hyperproperties.

2 HyperLTL, Game-Based Verification, and Prophecies

We fix a finite set of atomic propositions AP. A transition system (TS) is a tuple $\mathcal{T} = (S, s_{init}, \kappa, L)$, where S is a finite set of states, $s_{init} \in S$ is an initial state, $\kappa : S \to (2^S \setminus \{\emptyset\})$ is a transition function, and $L : S \to 2^{AP}$ is a state labeling. HyperLTL formulas are generate by the following grammar

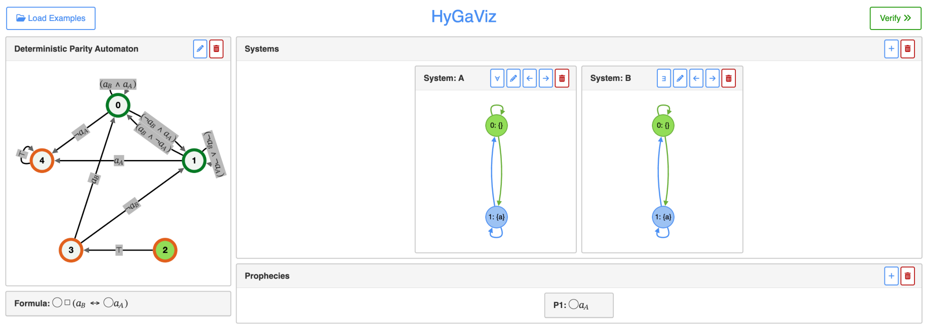
$$\psi := a_\pi \mid \psi \land \psi \mid \neg\psi \mid \bigcirc\psi \mid \psi \,\mathcal{U}\, \psi \qquad\qquad \varphi := \forall \pi.\, \varphi \mid \exists \pi.\, \varphi \mid \psi$$

where $a \in AP$ is an atomic proposition, and π is a trace variable. In a HyperLTL formula, we can quantify over traces in a system (bound to some trace variable), and then evaluate an LTL formula on the resulting traces. In the LTL body, formula a_π expresses that AP a should hold in the current step on the trace bound to trace variable π. See [23] for details.

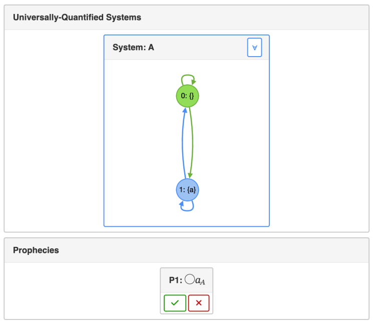
2.1 Game-Based Verification

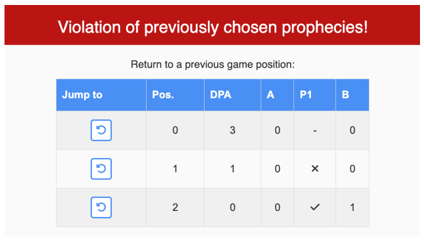
`HyGaViz`'s verification certificates are rooted in a game-based verification method [6]. Given a $\forall^*\exists^*$ HyperLTL formula $\forall \pi_1 \ldots \forall \pi_k.\, \exists \pi_{k+1} \ldots \exists \pi_{k+l}.\, \psi$, we view verification as a game between the $\forall$-player (controlling traces $\pi_1, \ldots, \pi_k$) and the $\exists$-player (controlling traces $\pi_{k+1} \ldots, \pi_{k+l}$). Each state of the game has the form $\langle s_1, \ldots, s_{k+l}, q\rangle$, where $s_1, \ldots, s_{k+l} \in S$ are system states (representing the current state of $\pi_1, \ldots, \pi_{k+l}$, respectively), and q is the state of a deterministic parity automaton (DPA) that tracks the acceptance of the LTL body ψ. When the game is in state $\langle s_1, \ldots, s_{k+l}, q\rangle$, the $\forall$-player first fixes successor states $s'_1, \ldots, s'_k$ for $\pi_1, \ldots, \pi_k$ (such that $s'_i \in \kappa(s_i)$ for all $1 \leq i \leq k$); the $\exists$-player responds by selecting successor states $s'_{k+1}, \ldots, s'_{k+l}$ for $\pi_{k+1}, \ldots, \pi_{k+l}$; and the game repeats from state $\langle s'_1, \ldots, s'_{k+l}, q'\rangle$ (where q' is the updated DPA state).

Visualizing Game-Based Verification. In `HyGaViz`, the user can create a verification scenario by manually creating finite-state transition systems and a HyperLTL formula; see Figure 1a. Note how the quantification prefix is determined implicitly by the order of the systems. In particular, the traces are resolved on individual (potentially different) transition systems. During simulation (cf. the example in Figure 1b), we visualize a game state $\langle s_1, \ldots, s_{k+l}, q\rangle$ by marking the current state of each system – separated into user-controlled (universally



(a)





Violation of previously chosen prophecies!

Return to a previous game position:

Jump to	Pos.	DPA	A	P1	B
⟲	0	3	0	-	0
⟲	1	1	0	×	0
⟲	2	0	0	✓	1

(b) **(c)**

Fig. 2: Screenshots of `HyGaViz` when using prophecies.

quantified) systems (top right) and strategy-controlled (existentially quantified) ones (bottom right) – and display the current state of the DPA (top left). The user takes the role of the ∀-player and, in each step, determines successor states for all universally quantified systems. Once successor states for all universally quantified systems are confirmed, `HyGaViz` automatically updates existentially quantified systems (and the DPA state) based on the internally computed strategy, and the game continues to the next stage. Moreover, `HyGaViz` highlights the next states when the user *hovers* over possible successor states for the universally quantified systems (once successor states for all but one universally quantified system are confirmed). Using the information tab in the bottom left, the user can jump to previous game states and explore the reaction of the strategy to different choices for the universally quantified systems.

2.2 Prophecies

In our game, the ∃-player only observes a finite prefix of the traces produced by the ∀-player (or, equivalently, the user of `HyGaViz`) and is thus missing information about the future. We can counteract this by using *prophecies* [1], which are

LTL formulas over trace variables $\pi_1, \ldots, \pi_k$ [6]. Given an LTL prophecy formula θ, the $\forall$-player (i.e., the user) has to, in each step, decide if its future behavior (on $\pi_1, \ldots, \pi_k$) satisfies θ. If the $\forall$-player decides that θ holds (resp. does not hold), the $\exists$-player can play under the assumption that the *future* behavior of the $\forall$-player satisfies (resp. violates) θ. See [6] for details.

Example 2. We illustrate prophecies with the example in Figure 2. The two systems A and B in Figure 2a generate all traces over AP a, and the HyperLTL formula $\forall A. \exists B. \bigcirc\Box(a_B \leftrightarrow \bigcirc a_A)$ requires that trace B *predicts* the future behavior of A. Without prophecies, the $\exists$-player loses: No matter what successor state the $\exists$-player picks, the $\forall$-player can, in the next step, violate the prediction of the $\exists$-player. `HyGaViz` communicates the absence of a winning strategy if the user pushes the *Verify* button. Instead, the user can add the LTL prophecy $\bigcirc a_A$ (cf. Figure 2a). During simulation, the user (who takes the role of the $\forall$-player) has to, in each step, fix a successor state for system A *and* determine if prophecy $\bigcirc a_A$ holds. We depict an excerpt of the simulation page in Figure 2b. As expected, the strategy for the $\exists$-player (computed automatically by `HyGaViz`) can use the prophecy to win: For example, if the user states that $\bigcirc a_A$ holds (so the $\exists$-player can assume that a hold in the next step in A), `HyGaViz` moves system B to state 1. If the user violates a previous prophecy decision – e.g., by stating that prophecy $\bigcirc a_A$ holds but, in the next step, moving system A to state 0 where AP a does not hold – `HyGaViz` detects this violation and forces the user to restart from an earlier state of the game (Figure 2c). $\triangle$

3 HyGaViz: Tool Overview

`HyGaViz` consists of a backend verification engine written in `F#`. The backend uses `spot` [22] to translate LTL formulas to DPAs and `oink` [21] to synthesize a strategy for the $\exists$-player. We use a stateless `Node.js` [37] backend that communicates with the verification engine via `JSON`. `HyGaViz`'s frontend is written in `JavaScript` and uses `Cytoscape.js` [26] to render transition systems and automata.

4 Conclusion

We have proposed the first method to generate and visualize certificates for the satisfaction of $\forall^*\exists^*$ HyperLTL formulas. Our tool, `HyGaViz`, allows users to *interactively* explore the complex dependencies between multiple traces by challenging a strategy for existentially quantified traces. Ultimately, `HyGaViz` is a first step to foster trust in (and understanding of) verification results for complex alternating hyperproperties, as is needed to, e.g., certify information-flow policies like φ_{NI}. For now, `HyGaViz` can handle (small) finite state systems, which we visualize as directed graphs. The underlying strategy-centered approach also applies to larger (potentially infinite-state) systems represented symbolically [7]. In future work, one could extend `HyGaViz` to such systems by exploring different visualization approaches for larger systems [34,31,36].

Data Availability. `HyGaViz` is available at [11].

Acknowledgments. This work was partially supported by the European Research Council (ERC) Grant HYPER (101055412) and by the German Research Foundation (DFG) as part of TRR 248 (389792660).

References

1. Abadi, M., Lamport, L.: The existence of refinement mappings. Theor. Comput. Sci. **82**(2), 253–284 (1991). https://doi.org/10.1016/0304-3975(91)90224-P
2. Barthe, G., D'Argenio, P.R., Rezk, T.: Secure information flow by self-composition. Math. Struct. Comput. Sci. (2011). https://doi.org/10.1017/S0960129511000193
3. Bartocci, E., Henzinger, T.A., Nickovic, D., da Costa, A.O.: Hypernode automata. In: International Conference on Concurrency Theory, CONCUR 2023 (2023). https://doi.org/10.4230/LIPICS.CONCUR.2023.21
4. Beer, I., Ben-David, S., Chockler, H., Orni, A., Trefler, R.J.: Explaining counterexamples using causality. In: International Conference on Computer Aided Verification, CAV 2009 (2009). https://doi.org/10.1007/978-3-642-02658-4_11
5. Beschastnikh, I., Liu, P., Xing, A., Wang, P., Brun, Y., Ernst, M.D.: Visualizing distributed system executions. ACM Trans. Softw. Eng. Methodol. (2020). https://doi.org/10.1145/3375633
6. Beutner, R., Finkbeiner, B.: Prophecy variables for hyperproperty verification. In: Computer Security Foundations Symposium, CSF 2022 (2022). https://doi.org/10.1109/CSF54842.2022.9919658
7. Beutner, R., Finkbeiner, B.: Software verification of hyperproperties beyond k-safety. In: International Conference on Computer Aided Verification, CAV 2022 (2022). https://doi.org/10.1007/978-3-031-13185-1_17
8. Beutner, R., Finkbeiner, B.: AutoHyper: Explicit-state model checking for HyperLTL. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2023 (2023). https://doi.org/10.1007/978-3-031-30823-9_8
9. Beutner, R., Finkbeiner, B.: Non-deterministic planning for hyperproperty verification. In: International Conference on Automated Planning and Scheduling, ICAPS 2024 (2024). https://doi.org/10.1609/ICAPS.V34I1.31457
10. Beutner, R., Finkbeiner, B., Frenkel, H., Metzger, N.: Second-order hyperproperties. In: International Conference on Computer Aided Verification, CAV 2023 (2023). https://doi.org/10.1007/978-3-031-37703-7_15
11. Beutner, R., Finkbeiner, B., Göbl, A.: HyGaViz: Visualizing Game-Based Certificates for Hyperproperty Verification (2024). https://doi.org/10.5281/zenodo.12206584
12. Biewer, S., Dimitrova, R., Fries, M., Gazda, M., Heinze, T., Hermanns, H., Mousavi, M.R.: Conformance relations and hyperproperties for doping detection in time and space. Log. Methods Comput. Sci. (2022). https://doi.org/10.46298/lmcs-18(1:14)2022
13. Bolton, M.L., Bass, E.J.: Using task analytic models to visualize model checker counterexamples. In: International Conference on Systems, Man and Cybernetics, SMC 2010 (2010). https://doi.org/10.1109/ICSMC.2010.5641711

14. Bozzelli, L., Maubert, B., Pinchinat, S.: Unifying hyper and epistemic temporal logics. In: International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2015 (2015). https://doi.org/10.1007/978-3-662-46678-0_11
15. Chaudhuri, S., Gulwani, S., Lublinerman, R.: Continuity and robustness of programs. Commun. ACM (2012). https://doi.org/10.1145/2240236.2240262
16. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: International Conference om Principles of Security and Trust, POST 2014 (2014). https://doi.org/10.1007/978-3-642-54792-8_15
17. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. (2010). https://doi.org/10.3233/JCS-2009-0393
18. Coenen, N., Dachselt, R., Finkbeiner, B., Frenkel, H., Hahn, C., Horak, T., Metzger, N., Siber, J.: Explaining hyperproperty violations. In: International Conference on Computer Aided Verification, CAV 2022 (2022). https://doi.org/10.1007/978-3-031-13185-1_20
19. Coenen, N., Finkbeiner, B., Frenkel, H., Hahn, C., Metzger, N., Siber, J.: Temporal causality in reactive systems. In: International Symposium on Automated Technology for Verification and Analysis, ATVA 2022 (2022). https://doi.org/10.1007/978-3-031-19992-9_13
20. Coenen, N., Finkbeiner, B., Sánchez, C., Tentrup, L.: Verifying hyperliveness. In: International Conference on Computer Aided Verification, CAV 2019 (2019). https://doi.org/10.1007/978-3-030-25540-4_7
21. van Dijk, T.: Oink: An implementation and evaluation of modern parity game solvers. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2018 (2018). https://doi.org/10.1007/978-3-319-89960-2_16
22. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From spot 2.0 to spot 2.10: What's new? In: International Conference on Computer Aided Verification, CAV 2022 (2022). https://doi.org/10.1007/978-3-031-13188-2_9
23. Finkbeiner, B.: Logics and algorithms for hyperproperties. ACM SIGLOG News (2023). https://doi.org/10.1145/3610392.3610394
24. Finkbeiner, B., Rabe, M.N., Sánchez, C.: Algorithms for model checking HyperLTL and HyperCTL*. In: International Conference on Computer Aided Verification, CAV 2015 (2015). https://doi.org/10.1007/978-3-319-21690-4_3
25. Finkbeiner, B., Siber, J.: Counterfactuals modulo temporal logics. In: International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2023 (2023). https://doi.org/10.29007/QTW7
26. Franz, M., Lopes, C.T., Huck, G., Dong, Y., Sümer, S.O., Bader, G.D.: Cytoscape.js: a graph theory library for visualisation and analysis. Bioinform. (2016). https://doi.org/10.1093/BIOINFORMATICS/BTV557
27. Griggio, A., Roveri, M., Tonetta, S.: Certifying proofs for LTL model checking. In: Formal Methods in Computer Aided Design, FMCAD 2018 (2018). https://doi.org/10.23919/FMCAD.2018.8603022
28. Groce, A., Kroening, D., Lerda, F.: Understanding counterexamples with explain. In: International Conference on Computer Aided Verification, CAV 2004 (2004). https://doi.org/10.1007/978-3-540-27813-9_35
29. Horak, T., Coenen, N., Metzger, N., Hahn, C., Flemisch, T., Méndez, J., Dimov, D., Finkbeiner, B., Dachselt, R.: Visual analysis of hyperproperties for understanding

model checking results. IEEE Trans. Vis. Comput. Graph. (2022). https://doi.org/10.1109/TVCG.2021.3114866

30. Hsu, T., Sánchez, C., Bonakdarpour, B.: Bounded model checking for hyperproperties. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2021 (2021). https://doi.org/10.1007/978-3-030-72016-2_6
31. Jerding, D.F., Stasko, J.T., Ball, T.: Visualizing interactions in program executions. In: International Conference on Software Engineering, ICSE 1997 (1997). https://doi.org/10.1145/253228.253356
32. Kasenberg, D., Thielstrom, R., Scheutz, M.: Generating explanations for temporal logic planner decisions. In: International Conference on Automated Planning and Scheduling, ICAPS 2020 (2020). https://doi.org/10.1609/icaps.v30i1.6740
33. McLean, J.: A general theory of composition for trace sets closed under selective interleaving functions. In: Symposium on Security and Privacy, SP 1994 (1994). https://doi.org/10.1109/RISP.1994.296590
34. Moreno, A., Myller, N., Sutinen, E., Ben-Ari, M.: Visualizing programs with Jeliot 3. In: Conference on Advanced Visual Interfaces, AVI 2004 (2004). https://doi.org/10.1145/989863.989928
35. Rabe, M.N.: A temporal logic approach to information-flow control. Ph.D. thesis, Saarland University (2016)
36. Rajala, T., Laakso, M., Kaila, E., Salakoski, T.: Effectiveness of program visualization: A case study with the ViLLE tool. J. Inf. Technol. Educ. Innov. Pract. (2008)
37. Tilkov, S., Vinoski, S.: Node.js: Using javascript to build high-performance network programs. IEEE Internet Comput. (2010). https://doi.org/10.1109/MIC.2010.145
38. Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: Computer Security Foundations Workshop CSFW 2003 (2003). https://doi.org/10.1109/CSFW.2003.1212703