EAHyper: Satisfiability, Implication, and Equivalence Checking of Hyperproperties

Bernd Finkbeiner, Christopher Hahn(✉), and Marvin Stenger

Saarland Informatics Campus, Saarland University, Saarbrücken, Germany
{finkbeiner,hahn,stenger}@react.uni-saarland.de

Abstract. We introduce EAHyper, the first tool for the automatic checking of satisfiability, implication, and equivalence of hyperproperties. Hyperproperties are system properties that relate multiple computation traces. A typical example is an information flow policy that compares the observations made by an external observer on execution traces that result from different values of a secret variable. EAHyper analyzes hyperproperties that are specified in HyperLTL, a recently introduced extension of linear-time temporal logic (LTL). HyperLTL uses trace variables and trace quantifiers to refer to multiple execution traces simultaneously. Applications of EAHyper include the automatic detection of specifications that are inconsistent or vacuously true, as well as the comparison of multiple formalizations of the same policy, such as different notions of observational determinism.

1 Introduction

HyperLTL [3] is a recently introduced temporal logic for the specification of hyperproperties [4]. HyperLTL characterizes the secrecy and integrity of a system by comparing two or more execution traces. For example, we might express that the contents of a variable is secret by specifying that an external observer makes the same observations on all execution traces that result from different values of the variable. Such a specification cannot be expressed as a standard trace property, because it refers to multiple traces. The specification can, however, be expressed as a hyperproperty, which is a *set of sets* of traces.

HyperLTL has been used to specify and verify the information flow in communication protocols and web applications, the symmetric access to critical resources in mutex protocols, and Hamming distances between code words in error resistant codes [8,9,13]. The logic is already supported by both model checking [8] and runtime verification [1] tools. In this paper, we present the first tool for HyperLTL *satisfiability*. Our tool, which we call EAHyper, can be used

This work was partially supported by the German Research Foundation (DFG) in the Collaborative Research Center 1223 and by the Graduate School of Computer Science at Saarland University.

R. Majumdar and V. Kunčak (Eds.): CAV 2017, Part II, LNCS 10427, pp. 564–570, 2017.
DOI: 10.1007/978-3-319-63390-9_29

to automatically detect specifications that are inconsistent or vacuously true, and to check implication and equivalence between multiple formalizations of the same requirement.

HyperLTL extends linear-time temporal logic (LTL) with trace variables and trace quantifiers. The requirement that the external observer makes the same observations on all traces is, for example, expressed as the HyperLTL formula $\forall\pi.\forall\pi'.\ \mathsf{G}(O_\pi = O_{\pi'})$, where O is the set of observable outputs. A more general property is *observational determinism* [12,14,17], which requires that a system appears deterministic to an observer who sees inputs I and outputs O. Observational determinism can be formalized as the HyperLTL formula $\forall\pi.\forall\pi'.\ \mathsf{G}(I_\pi = I_{\pi'}) \rightarrow \mathsf{G}(O_\pi = O_{\pi'})$, or, alternatively, as the HyperLTL formula $\forall\pi.\forall\pi'.\ (O_\pi = O_{\pi'})\ \mathcal{W}\ (I_\pi \neq I_{\pi'})$. The first formalization states that on any pair of traces, where the inputs are the same, the outputs must be the same as well; the second formalization states that differences in the observable output may only occur *after* differences in the observable input have occurred. As can be easily checked with EAHyper, the second formalization is the stronger requirement.

EAHyper implements the decision procedure for the $\exists^*\forall^*$ fragment of HyperLTL [7]. The $\exists^*\forall^*$ fragment consists of all HyperLTL formulas with at most one quantifier alternation, where no existential quantifier is in the scope of a universal quantifier. Many practical HyperLTL specifications are in fact alternation-free, i.e., they contain either only universal or only existential quantifiers. The $\exists^*\forall^*$ fragment is the largest decidable fragment. It contains in particular all alternation-free formulas and also all implications and equivalences between alternation-free formulas.

In the remainder of this paper, we give a quick summary of the syntax and semantics of HyperLTL, describe the implementation of EAHyper, and report on experimental results.

2 HyperLTL

HyperLTL Syntax. HyperLTL extends LTL with trace variables and trace quantifiers. Let $\mathcal{V}$ be an infinite supply of trace variables, AP the set of atomic propositions, and TR the set of infinite traces over AP. The syntax of HyperLTL is given by the following grammar:

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

where $a \in AP$ is an atomic proposition and $\pi \in \mathcal{V}$ is a trace variable. The derived temporal operators F, G, and W are defined as for LTL. Logical connectives, i.e., $\wedge$, $\rightarrow$, and $\leftrightarrow$ are derived in the usual way. We also use syntactic sugar like $O_\pi = O_{\pi'}$, which abbreviates $\bigwedge_{a\in O} a_\pi \leftrightarrow a_{\pi'}$ for a set O of atomic propositions.

The $\exists^*$ fragment of HyperLTL consists of all formulas that only contain existential quantifiers. The $\forall^*$ fragment of HyperLTL consists of all formulas

that only contain universal quantifiers. The union of the two fragments is the *alternation-free fragment*. The $\exists^*\forall^*$ *fragment* consists of all formulas with at most one quantifier alternation, where no existential quantifier is in the scope of a universal quantifier.

HyperLTL Semantics. A HyperLTL formula defines a *hyperproperty*, which is a set of sets of traces. A set T of traces satisfies the hyperproperty if it is an element of this set of sets. Formally, the semantics of HyperLTL formulas is given with respect to *trace assignment* Π from $\mathcal{V}$ to TR, i.e., a partial function mapping trace variables to actual traces. $\Pi[\pi \mapsto t]$ denotes that π is mapped to t, with everything else mapped according to Π. $\Pi[i, \infty]$ denotes the trace assignment that is equal to $\Pi(\pi)[i, \infty]$ for all π.

$$
\begin{array}{lll}
\Pi \models_T \exists \pi.\psi & \text{iff} & \text{there exists } t \in T \ : \ \Pi[\pi \mapsto t] \models_T \psi \\
\Pi \models_T \forall \pi.\psi & \text{iff} & \text{for all } t \in T \ : \ \Pi[\pi \mapsto t] \models_T \psi \\
\Pi \models_T a_\pi & \text{iff} & a \in \Pi(\pi)[0] \\
\Pi \models_T \neg\psi & \text{iff} & \Pi \not\models_T \psi \\
\Pi \models_T \psi_1 \vee \psi_2 & \text{iff} & \Pi \models_T \psi_1 \text{ or } \Pi \models_T \psi_2 \\
\Pi \models_T \mathsf{X}\psi & \text{iff} & \Pi[1, \infty] \models_T \psi \\
\Pi \models_T \psi_1 \,\mathsf{U}\, \psi_2 & \text{iff} & \text{there exists } i \geq 0 : \Pi[i, \infty] \models_T \psi_2 \\
 & & \text{and for all } 0 \leq j < i \text{ we have } \Pi[j, \infty] \models_T \psi_1
\end{array}
$$

A HyperLTL formula φ is *satisfiable* if and only if there exists a non-empty trace set T, such that $\Pi \models_T \psi$, where Π is the empty trace assignment. The formula φ is *valid* if and only if for all non-empty trace sets T it holds that $\Pi \models_T \psi$.

3 EAHyper

The input of EAHyper is either a HyperLTL formula in the $\exists^*\forall^*$ fragment, or an implication between two alternation-free formulas. For $\exists^*\forall^*$ formulas, EAHyper reports satisfiability; for implications between alternation-free formulas, validity. EAHyper proceeds in three steps:

1. *Translation into the* $\exists^*\forall^*$ *fragment:* If the input is an implication between two alternation-free formulas, we construct a formula in the $\exists^*\forall^*$ fragment that represents the *negation* of the implication. For example, for the implication of $\forall\pi_1 \ldots \forall\pi_n.\ \psi$ and $\forall\pi'_1 \ldots \forall\pi'_m.\ \varphi$, we construct the $\exists^*\forall^*$ formula $\exists\pi'_1 \ldots \exists\pi'_m \forall\pi_1 \ldots \forall\pi_n.\ \psi \wedge \neg\varphi$. The implication is valid if and only if the resulting $\exists^*\forall^*$ formula is unsatisfiable.
2. *Reduction to LTL satisfiability:* EAHyper implements the decision procedure for the $\exists^*\forall^*$ fragment of HyperLTL [7]. The satisfiability of the HyperLTL formula is reduced to the satisfiability of an LTL formula:
   - Formulas in the $\forall^*$ fragment are translated to LTL formulas by discarding the quantifier prefix and all trace variables. For example, $\forall\pi_1.\forall\pi_2.\ \mathsf{G}b_{\pi_1} \wedge \mathsf{G}\neg b_{\pi_2}$ is translated to the equisatisfiable LTL formula $\mathsf{G}b \wedge \mathsf{G}\neg b$.

- Formulas in the $\exists^*$ fragment are translated to LTL formulas by introducing a fresh atomic proposition a_i for every atomic proposition a and every trace variable π_i. For example, $\exists\pi_1.\exists\pi_2.\ a_{\pi_1} \wedge \mathsf{G}\neg b_{\pi_1} \wedge \mathsf{G}b_{\pi_2}$ is translated to the equisatisfiable LTL formula $a_1 \wedge \mathsf{G}\neg b_1 \wedge \mathsf{G}b_2$.
- Formulas in the $\exists^*\forall^*$ fragment are translated into the $\exists^*$ fragment (and then on into LTL) by unrolling the universal quantifiers. For example, $\exists\pi_1.\exists\pi_2.\forall\pi_1'.\forall\pi_2'.\ \mathsf{G}a_{\pi_1'} \wedge \mathsf{G}b_{\pi_2'} \wedge \mathsf{G}c_{\pi_1} \wedge \mathsf{G}d_{\pi_2}$ is translated to the equisatisfiable $\exists^*$ formula $\exists\pi_1.\exists\pi_2.\ (\mathsf{G}a_{\pi_1} \wedge \mathsf{G}b_{\pi_1} \wedge \mathsf{G}c_{\pi_1} \wedge \mathsf{G}d_{\pi_2}) \wedge (\mathsf{G}a_{\pi_2} \wedge \mathsf{G}b_{\pi_1} \wedge \mathsf{G}c_{\pi_1} \wedge \mathsf{G}d_{\pi_2}) \wedge (\mathsf{G}a_{\pi_1} \wedge \mathsf{G}b_{\pi_2} \wedge \mathsf{G}c_{\pi_1} \wedge \mathsf{G}d_{\pi_2}) \wedge (\mathsf{G}a_{\pi_2} \wedge \mathsf{G}b_{\pi_2} \wedge \mathsf{G}c_{\pi_1} \wedge \mathsf{G}d_{\pi_2})$.

3. *LTL satisfiability:* The satisfiability of the resulting LTL formula is checked through an external tool. Currently, EAHyper is linked to two LTL satisfiability checkers, pltl and Aalta.
   - Pltl [15] is a one-pass tableaux-based decision procedure for LTL, which not necessarily explores the full tableaux.
   - Aalta_2.0 [11] is a decision procedure for LTL based on a reduction to the Boolean satisfiability problem, which is in turn solved by minisat [6]. Aalta's on-the-fly approach is based on so-called obligation sets and outperforms model-checking-based LTL satisfiability solvers.

EAHyper is implemented in OCaml and supports UNIX-based operating systems. Batch-processing of HyperLTL formulas is provided. Options such as the choice of the LTL satisfiability checker are provided via a command-line interface.

4 Experimental Results

We report on the performance of EAHyper on a range of benchmarks, including observational determinism, symmetry, error resistant code, as well as randomly generated formulas. The experiments were carried out in a virtual machine running Ubuntu 14.04 LTS on an Intel Core i5-2500K CPU with 3.3 GHZ and 2 GB RAM. We chose to run EAHyper in a virtual machine to make our results easily reproducible; running EAHyper natively results in (even) better performance.[1]

- *Observational Determinism* [12,14,17]. Our first benchmark compares the following formalizations of observational determinism, with $|I| = |O| = 1$: $(OD1) : \forall\pi_1.\forall\pi_1'.\ \mathsf{G}(I_{\pi_1} = I_{\pi_1'}) \rightarrow \mathsf{G}(O_{\pi_1} = O_{\pi_1'})$, $(OD2) : \forall\pi_2.\forall\pi_2'.\ (I_{\pi_2} = I_{\pi_2'}) \rightarrow \mathsf{G}(O_{\pi_2} = O_{\pi_2'})$, and $(OD3) : \forall\pi_3.\forall\pi_3'.\ (O_{\pi_3} = O_{\pi_3'})\,\mathsf{W}\,(I_{\pi_3} \neq I_{\pi_3'})$. EAHyper needs less then a second to order the formalizations with respect to implication: $OD2 \rightarrow OD1$, $OD2 \rightarrow OD3$, and $OD3 \rightarrow OD1$.
- *Quantitative Noninterference* [2]. The bounding problem of quantitative noninterference asks whether the amount of information leaked by a system is bounded by a constant c. This is expressed in HyperLTL as the requirement that there are no $c+1$ distinguishable traces for a *low-security* observer [16].

$$QN(c) \coloneqq \forall\pi_0 \ldots \forall\pi_c.\ \neg((\bigwedge_i I_{\pi_i} = I_{\pi_0}) \wedge \bigwedge_{i \neq j} O_{\pi_i} \neq O_{\pi_j})$$

[1] EAHyper is available online at https://react.uni-saarland.de/tools/eahyper/.

Table 1. Quantitative noninterference benchmark: wall clock time in seconds for checking whether QN(row) implies QN(column). "–" denotes that the instance was not solved in 120 s.

(a) Aalta

QN	1	2	3	4	5
1	0.04	0.04	0.54	–	–
2	0.03	0.09	1.58	–	–
3	0.03	0.05	0.68	–	–
4	0.03	0.11	0.34	8.68	–
5	0.06	0.34	–	–	–

(b) pltl

QN	1	2	3	4	5
1	0.05	0.05	0.08	0.13	0.23
2	0.05	0.11	0.25	0.39	0.79
3	0.07	0.25	0.77	2.02	5.12
4	0.16	0.73	3.12	17.73	43.26
5	0.26	2.57	15.67	71.82	–

In the benchmark, we check implications between different bounds. The performance of EAHyper is shown in Table 1. Using Aalta as the LTL satisfiability checker generally produces faster results, but pltl scales to larger bounds.

– *Symmetry* [8]. A violation of symmetry in a mutual exclusion protocol indicates that some concurrent process has an unfair advantage in accessing a critical section. The benchmark is derived from a model checking case study, in which various symmetry claims were verified and falsified for the Bakery protocol. EAHyper checks the implications between the four main symmetry properties from the case study in 13.86 s. Exactly one of the implications turns out to be true.
– *Error resistant code* [8]. Error resistant codes enable the transmission of data over noisy channels. A typical model of errors bounds the number of flipped bits that may happen for a given code word length. Then, error correction coding schemes must guarantee that all code words have a minimal Hamming distance. The following HyperLTL formula specifies that all code words $o \in O$ produced by an encoder have a minimal Hamming distance [10] of d: $\forall\pi.\ \forall\pi'.\ F(\bigvee_{i\in I} \neg(i_\pi \leftrightarrow i_{\pi'})) \rightarrow \neg \mathit{Ham}_O(d-1,\pi,\pi')$. Ham_O is recursively defined as $\mathit{Ham}_O(-1,\pi,\pi') = \mathit{false}$ and $\mathit{Ham}_O(d,\pi,\pi') = (\bigwedge_{o\in O} o_\pi \leftrightarrow o_{\pi'})\,\mathsf{W}\,(\bigvee_{o\in O} \neg(o_\pi \leftrightarrow o_{\pi'}) \wedge \mathsf{X}\ \mathit{Ham}_O(d-1,\pi,\pi'))$. The benchmark checks implications between the HyperLTL formulas for different minimal Hamming distances. The performance of EAHyper is shown in Table 2.
– *Random formulas.* In the last benchmark, we randomly generated sets of 250 HyperLTL formulas containing five atomic propositions, using randltl [5] and assigning trace variables randomly to atomic propositions. As shown in Table 3, EAHyper reaches its limits, by running out of memory, after approximately five existential and five universal quantifiers.

5 Discussion

EAHyper is the first implementation of the decision procedure for the $\exists^*\forall^*$ fragment of HyperLTL [7]. For formulas with up to approximately five universal quantifiers, EAHyper performs reliably well on our broad range of benchmarks, which represent different types of hyperproperties studied in the literature as well as randomly generated formulas.

Table 2. Error resistant codes benchmark: wall clock time in seconds for checking whether Ham(row) implies Ham(column).

Ham	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0.03	0.02	0.03	0.02	0.02	0.02	0.03	0.03	0.04	0.08	0.10	0.18	0.25	0.46	0.74	1.35	2.62
1	0.03	0.02	0.03	0.03	0.04	0.03	0.05	0.04	0.06	0.08	0.13	0.21	0.40	0.49	0.82	1.50	2.99
2	0.01	0.03	0.03	0.03	0.04	0.02	0.03	0.04	0.04	0.07	0.12	0.21	0.36	0.55	0.88	1.59	3.09
3	0.03	0.04	0.04	0.05	0.04	0.04	0.03	0.04	0.05	0.07	0.12	0.23	0.36	0.52	0.87	1.56	3.12
4	0.04	0.04	0.04	0.06	0.10	0.02	0.03	0.05	0.08	0.08	0.16	0.21	0.36	0.52	0.86	1.66	3.05
5	0.03	0.03	0.05	0.07	0.07	0.19	0.14	0.17	0.05	0.08	0.14	0.22	0.30	0.52	0.92	1.55	2.99
6	0.03	0.04	0.05	0.06	0.09	0.22	0.35	0.21	0.25	0.11	0.25	0.26	0.36	0.53	0.87	1.57	3.00
7	0.04	0.05	0.05	0.05	0.14	0.24	0.32	0.37	0.38	0.42	0.14	0.20	0.37	0.52	0.89	1.65	3.05
8	0.05	0.05	0.07	0.10	0.17	0.23	0.26	0.36	0.50	0.56	0.47	0.40	0.53	0.53	1.13	1.61	3.18
9	0.07	0.08	0.08	0.10	0.16	0.19	0.21	0.43	0.70	0.64	0.48	0.52	0.90	0.65	1.03	1.71	3.08
10	0.09	0.13	0.15	0.15	0.21	0.20	0.34	0.43	0.54	0.76	1.38	1.55	0.61	0.89	1.03	1.78	3.22
11	0.16	0.23	0.22	0.24	0.24	0.26	0.41	0.53	0.62	0.81	1.30	1.29	1.81	1.05	1.86	2.33	3.17
12	0.27	0.30	0.36	0.30	0.32	0.41	0.45	0.46	0.85	0.91	1.69	1.28	2.81	2.82	1.14	3.91	4.49
13	0.38	0.46	0.51	0.47	0.57	0.52	0.57	0.86	1.03	1.27	1.47	2.16	3.19	8.22	5.48	8.64	7.08
14	0.69	0.87	0.91	0.84	0.84	0.98	0.94	1.02	1.46	1.30	2.01	3.82	3.96	6.35	7.50	9.06	11.11
15	1.22	1.52	1.58	1.70	1.69	1.65	1.67	1.74	1.87	2.73	3.02	3.08	5.87	7.25	13.04	34.17	12.26
16	2.26	3.04	2.97	3.00	3.10	3.11	3.35	3.29	3.57	4.17	3.76	5.78	7.45	17.31	17.75	31.51	48.09

Table 3. Random formulas benchmark: instances solved in 120 s and average wall clock time in seconds for 250 random formulas. Size denotes the tree-size argument for randltl.

size	40	60	40	60	40	60	40	60	40	60	40	60	40	60	40	60	40	60
	$\exists^0\forall^0$		$\exists^1\forall^0$		$\exists^2\forall^0$		$\exists^3\forall^0$		$\exists^4\forall^0$		$\exists^5\forall^0$		$\exists^6\forall^0$		$\exists^7\forall^0$		$\exists^8\forall^0$	
solved			250	250	250	250	250	250	250	250	250	250	250	250	250	250	250	250
avgt			0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	$\exists^0\forall^1$		$\exists^1\forall^1$		$\exists^2\forall^1$		$\exists^3\forall^1$		$\exists^4\forall^1$		$\exists^5\forall^1$		$\exists^6\forall^1$		$\exists^7\forall^1$		$\exists^8\forall^1$	
solved	250	250	250	250	250	250	250	249	250	250	249	247	250	248	249	247	247	248
avgt	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.05	0.02	0.06	0.02	0.01	0.02	0.01	0.13	0.02	0.04	0.08
	$\exists^0\forall^2$		$\exists^1\forall^2$		$\exists^2\forall^2$		$\exists^3\forall^2$		$\exists^4\forall^2$		$\exists^5\forall^2$		$\exists^6\forall^2$		$\exists^7\forall^2$		$\exists^8\forall^2$	
solved	250	250	250	250	248	249	249	247	247	247	248	246	246	246	244	246	244	247
avgt	0.01	0.01	0.01	0.01	0.03	0.12	0.03	0.01	0.26	0.02	0.32	0.02	0.09	0.02	0.02	0.02	0.05	0.03
	$\exists^0\forall^3$		$\exists^1\forall^3$		$\exists^2\forall^3$		$\exists^3\forall^3$		$\exists^4\forall^3$		$\exists^5\forall^3$		$\exists^6\forall^3$		$\exists^7\forall^3$		$\exists^8\forall^3$	
solved	250	250	250	250	249	247	248	246	247	245	245	246	245	246	244	247	243	246
avgt	0.01	0.01	0.01	0.01	0.03	0.02	0.07	0.02	0.06	0.03	0.14	0.05	0.17	0.08	0.23	0.16	0.45	0.25
	$\exists^0\forall^4$		$\exists^1\forall^4$		$\exists^2\forall^4$		$\exists^3\forall^4$		$\exists^4\forall^4$		$\exists^5\forall^4$		$\exists^6\forall^4$		$\exists^7\forall^4$		$\exists^8\forall^4$	
solved	250	250	250	250	250	246	247	246	245	246	244	247	245	247	244	245	0	0
avgt	0.01	0.1	0.01	0.01	0.02	0.01	0.21	0.03	0.35	0.09	0.23	0.28	0.46	1.01	0.98	2.41	–	–
	$\exists^0\forall^5$		$\exists^1\forall^5$		$\exists^2\forall^5$		$\exists^3\forall^5$		$\exists^4\forall^5$		$\exists^5\forall^5$		$\exists^6\forall^5$		$\exists^7\forall^5$		$\exists^8\forall^5$	
solved	250	250	250	250	249	247	248	247	243	245	245	246	0	0	0	0	0	0
avgt	0.01	0.01	0.01	0.01	0.26	0.02	0.18	0.07	0.27	0.37	0.51	2.81	–	–	–	–	–	–

References

1. Bonakdarpour, B., Finkbeiner, B.: Runtime verification for HyperLTL. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 41–45. Springer, Cham (2016)

2. Clark, D., Hunt, S., Malacaria, P.: Quantified interference for a while language. Electron. Notes Theoret. Comput. Sci. **112**, 149–166 (2005)
3. 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
4. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**(6), 1157–1210 (2010)
5. Duret-Lutz, A.: Manipulating LTL formulas using spot 1.0. In: Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 442–445. Springer, Cham (2013). doi:10.1007/978-3-319-02444-8_31
6. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24605-3_37
7. Finkbeiner, B., Hahn, C.: Deciding hyperproperties. In: Proceedings of the 27th International Conference on Concurrency Theory, CONCUR 2016, pp. 13:1–13:14 (2016)
8. Finkbeiner, B., Rabe, M.N., Sánchez, C.: Algorithms for model checking HyperLTL and HyperCTL*. In: Kroening, D., Păsăreanu, C. (eds.) Computer Aided Verification. LNCS, vol. 9206, pp. 30–48. Springer, Cham (2015)
9. Finkbeiner, B., Seidl, H., Müller, C.: Specifying and verifying secrecy in workflows with arbitrarily many agents. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 157–173. Springer, Cham (2016). doi:10.1007/978-3-319-46520-3_11
10. Hamming, R.W.: Error detecting and error correcting codes. Bell Labs Tech. J. **29**(2), 147–160 (1950)
11. Li, J., Zhang, L., Pu, G., Vardi, M.Y., He, J.: LTL satisfiability checking revisited. In: 2013 20th International Symposium on Temporal Representation and Reasoning, TIME 2013, pp. 91–98 (2013)
12. McLean, J.: Proving noninterference and functional correctness using traces. J. Comput. Secur. **1**(1), 37–58 (1992)
13. Rabe, M.N.: A Temporal Logic Approach to Information-flow Control. Ph.D. thesis, Saarland University (2016)
14. Roscoe, A.W.: CSP and determinism in security modelling. In: Proceedings of the 1995 IEEE Symposium on Security and Privacy, pp. 114–127 (1995)
15. Schwendimann, S.: A new one-pass tableau calculus for **PLTL**. In: Swart, H. (ed.) TABLEAUX 1998. LNCS (LNAI), vol. 1397, pp. 277–291. Springer, Heidelberg (1998). doi:10.1007/3-540-69778-0_28
16. Smith, G.: On the foundations of quantitative information flow. In: Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2009, pp. 288–302 (2009)
17. Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: 16th IEEE Computer Security Foundations Workshop CSFW-16 2003, p. 29 (2003)