Finite-Word Hyperlanguages

Borzoo Bonakdarpour[a], Sarai Sheinvald[b]

[a]*Department of Computer Science and Engineering, Michigan State University, USA*
[b]*Department of Software Engineering, ORT Braude College, Israel*

Abstract

Formal languages are in the core of models of computation and their behavior. A rich family of models for many classes of languages have been widely studied. *Hyperproperties* lift conventional trace-based languages from a set of execution traces to a set of sets of executions. Hyperproperties have been shown to be a powerful formalism for expressing and reasoning about information-flow security policies and important properties of cyber-physical systems. Although there is an extensive body of work on formal-language representation of trace properties, we currently lack such a general characterization for hyperproperties.

We introduce *hyperlanguages* over finite words and models for expressing them. Essentially, these models express multiple words by using assignments to quantified *word variables*.

Relying on the standard models for regular languages, we propose *regular hyper-expressions* and *finite-word hyperautomata (NFH)*, for modeling the class of *regular hyperlanguages*. We demonstrate the ability of regular hyperlanguages to express hyperproperties for finite traces. We explore the closure properties and the complexity of the fundamental decision problems such as nonemptiness, universality, membership, and containment for various fragments of NFH.

1. Introduction

Formal languages, along with the models that express them, are in the core of modeling, specification, and verification of computing systems. Execution traces are formally described as words, and various families of automata are used for modeling systems of different types. *Regular languages* are a classic formalism for finite traces and when the traces are infinite, *ω-regular languages* are used.

There are well-known connections between specification logics and formal languages. For example, LTL [40] formulas can be translated to ω-regular expressions, and CTL* [20] formulas can be expressed using tree automata. Accordingly, many verification techniques that exploit these relations have been developed. For instance, in the automata-theoretic approach to verification [44, 45], the model-checking problem

Email addresses: borzoo@msu.edu (Borzoo Bonakdarpour), sarai@braude.ac.il (Sarai Sheinvald)

is reduced to checking the nonemptiness of the product automaton of the model and the negation of the specification. In some cases, the automata used are more expressive than the specification. For example, a finite automaton can express the notion "the signal p holds in all even positions", which LTL cannot express without referring to the odd positions as well.

Hyperproperties [16] generalize the traditional trace properties [4] to *system properties*, i.e., a set of sets of traces. A hyperproperty prescribes how the system should behave in its entirety and not just based on its individual executions. Hyperproperties have been shown to be a powerful tool for expressing and reasoning about information-flow security policies [16] and important properties of cyber-physical systems [46] such as *sensitivity* and *robustness*, as well as consistency conditions in distributed computing such as *linearizability* [11]. While different types of logics have been suggested for expressing hyperproperties, their formal-language counterparts and the models that express them are currently missing.

In this paper, we establish a formal-language theoretical framework for *hyperlanguages*, that are sets of sets of words, which we term *hyperwords*. Our framework is based on an underlying standard automata model for formal languages, augmented with quantified *word variables* that are assigned words from a set of words in the hyperlanguage. This formalism is in line with logics for hyperproperties (e.g., HyperLTL [15] and HyperPCTL [2, 1]). These logics express the behavior of infinite trace systems. However, a basic formal model for expressing general hyperproperties for finite words has not been defined yet.

To begin with the basics, we focus this paper on a regular type of hyperlanguages of sets consisting of finite words, which we call *regular hyperlanguages*. The models we introduce and study, namely, *finite hyperautomata* and *regular hyperexpressions* are based on the standard models for regular languages, namely regular expressions and finite-word automata. While we focus on the basic finite-word model, our framework and results can be naturally extended to handle any type of automata: infinite-word automata, tree automata, etc.

1.1. Motivation and Applications

Hyperlanguages based on finite words have many practical applications. For instance, runtime monitoring typically deals with finite executions. Since runtime monitors are often modeled as finite state machines or finite words in a formal language, it is conceivable that one designs monitors for hyperproperties as an automaton that accepts or rejects sets of executions. Another prominent application is in path planning in multi-agent systems, where the individual and collective objectives of the agents are specified as a set of terminating paths, i.e., an agent terminates its execution when its objective is accomplished. Other examples include learning automata for hyperlanguages, specifying winning strategies to enforce hyperproperties, etc. In Section 3, we provide concrete examples related to information-flow security.

Our finite hyperautomata are used in [34] in order to specify *multi-properties* which express the behaviour of several models that run in parallel. The finite-word regular properties of hyperautomata are then exploited in an $L^\star$-based technique in order to learn small approximations of the models. Then, replacing the concrete mod-

els with their small approximations accelerates the model-checking process for multi-properties.

Let us first explain our proposal for a finite-word hyperlanguage with two examples.

Example 1. Consider the following *regular hyperexpression* (RHE) over the alphabet $\{a\}$.

$$r_1 = \forall x.\exists y.\underbrace{\Big(\{a_x, a_y\}^*\{\#_x, a_y\}^+\Big)}_{\hat{r}_1}$$

The RHE r_1 uses two word variables x and y, which are assigned words from a hyperword. The RHE r_1 contains an underlying regular expression $\hat{r}_1$, whose alphabet is $(\{a, \#\})^{\{x,y\}}$, and whose (regular) language describes different word assignments to x and y, where $\#$ is used for padding the end if the words assigned to x and y are of different lengths. In a word in the language of $\hat{r}_1$, the i'th letter describes both i'th letters in the words assigned to x and y. For example, the word $\{a_x, a_y\}\{a_x, a_y\}\{\#_x, a_y\}$ describes the assignment $x \mapsto aa, y \mapsto aaa$. Notice that the regular expression $\hat{r}_1$ requires that the word assigned to y be longer than the word assigned to x: both variables are assigned a, until at some point, x is assigned $\#$ while y continues to be assigned a. The *quantification condition* $\forall x.\exists y$ of r_1 requires that for every word in a hyperword S in the hyperlanguage of r_1, there exists a longer word in S. This holds iff S contains infinitely many words. Therefore, the hyperlanguage of r_1 is the set of all infinite hyperwords over $\{a\}$. □

Example 2. Path planning objectives for robotic systems often stipulate the *existence* of one or more *finite* paths that stand out from *all* other paths. For example, robotics applications are often concerned with finding the shortest path that reaches a goal g, starting from an initial location i. The shortest path requirement can be expressed by the following RHE over an alphabet Σ:

$$r_2 = \exists x.\forall y.\{i_x, i_y\}\{\bar{g}_x, \bar{g}_y\}^*\Big(\{g_x, \bar{g}_y\} \mid \{g_x, g_y\}\Big)\{\#_x, \$_y\}^*$$

where $\bar{g} \in \Sigma - \{g\}$ and $\$ \in \Sigma$. That is, there exists a path x that is shorter than any other path y in reaching g.

Another interesting application in robotics is in *adversarial* settings, where some robots may interfere (e.g., act as moving obstacles) with a set of controllable robots. In this scenario, given any behavior of the adversarial robots, the controllable robots should be able to achieve their operation objectives. This specification is in general of the following form:

$$r_3 = \underbrace{\forall x_1.\forall x_2 \ldots \forall x_n}_{\text{advarsaries}}.\underbrace{\exists y_1.\exists y_2 \ldots \exists y_m}_{\text{controllable}}.\hat{r}$$

where words $x_1 \cdots x_n$ express the behavior of the adversaries, words $y_1 \cdots y_m$ describe the behavior of the controllable robots and regular expression $\hat{r}$ specifies the control objectives. □

Property	Result	
Closure	Complementation, Union, Intersection (Theorems 1, 3, 2)	
Nonemptiness	$\forall\exists\exists$	Undecidable (Theorem 4)
	$\exists^* \,/\, \forall^* \,/\, \exists^{**} \,/\, \forall^{**}$	NL-complete (Theorems 5, 8)
	$\exists^*\forall^*$	PSPACE-complete (Theorem 6)
	$\exists^*\forall^{**}$	EXPSPACE-complete (Theorem 9)
Bounded Nonemptiness	NFH	PSPACE-complete (Theorem 7)
Universality	$\exists\forall\forall$	Undecidable (Theorem 10)
	$\exists^* \,/\, \forall^*$	PSPACE-complete (Theorem 10)
	$\forall^*\exists^*$	EXPSPACE (Theorem 10)
Finite membership	NFH	PSPACE (Theorem 11)
	$O(\log(k))\ \ \forall$	NP-complete (Theorem 11)
Regular membership	Decidable (Theorem 12)	
Containment	NFH	Undecidable (Theorem 13)
	$\exists^* \subseteq \forall^* \,/\, \forall^* \subseteq \exists^*$	PSPACE-complete (Theorem 14)
	$\exists^*\forall^* \subseteq \forall^*\exists^*$	EXPSPACE (Theorem 14)

Table 1: Summary of results on properties of regular hyperlanguages.

1.2. Contributions

Although there is an ongoing line of research on model-checking hyperproperties [31, 8, 17], the work on finite-trace hyperproperties is limited to [22], where the authors construct a finite-word representation for the class of regular k-safety hyperproperties.

To the best of our knowledge, there is currently no formal automata model that expresses general hyperlanguages. Since automata are more expressive than LTL, our models allow expressing richer hyperproperties than those currently allowed by HyperLTL.

- Introduce regular hyperlanguages and RHEs, and demonstrate the ability of RHEs to express important information-flow security policies such as *noninterference* [33] and *observational determinism* [47].
- Present *nondeterministic finite-word hyperautomata* (NFH), an automata-based model for expressing regular hyperlanguages.
- Conduct a comprehensive study of the properties of regular hyperlanguages (see Table 1):
  - We show that regular hyperlanguages are *closed* under union, intersection, and complementation.
  - We consider the *nonemptiness* problem for NFH:
    - We prove that the nonemptiness problem is in general undecidable for NFH.
    - However, for the alternation-free fragments (which only allow one type of quantifier), as well as for the $\exists\forall$ fragment (in which the quantification condition is limited to a sequence of $\exists$ quantifiers followed by a sequence of $\forall$ quantifiers), nonemptiness is decidable.

    - As another positive result in the area of nonemptiness, we show that the *bounded nonemptiness problem*, in which we decide whether an NFH accepts a hyperword of bounded size, is PSPACE-complete.
    - We consider the construction of RHE and NFH with *wild-card letters*, which allow expressing the assignment to only a subset of the variables, by assigning a wild-card letter to the rest of the variables. We show that adding wild-cards does not alter the complexity of the nonemptiness for the alternation-free fragments, while it does increase the complexity of this problem for the $\exists\forall$ fragment.
    - We describe a semi-algorithm for deciding the nonemptiness of NFH with a $\forall\exists$ quantification condition. The procedure begins with the largest potential hyperword, and iteratively prunes it in a consistent way in case it is not accepted. Since the problem is undecidable, there are inputs for which our semi-algorithm does not halt. However, in case it does halt, it is guaranteed to return a correct answer. Since $\forall\exists$ is a useful fragment, our procedure can be a useful tool.
- We study the *universality*, *membership* and *containment* problems. These results are aligned with the complexity of HyperLTL model checking for tree-shaped and general Kripke structures [8]. This shows that the complexity results in [8] mainly stem from the nature of quantification over finite words and depend on neither the full power of the temporal operators nor the infinite nature of HyperLTL semantics.

1.3. Organization

The rest of the paper is organized as follows. We present preliminary concepts in Section 2. We introduce the notion of RHE and NFH in Sections 3 and 4, and study their properties and complexity results in Sections 5, 6, and 7. We discuss related work in Section 8. Finally, we make concluding remarks and discuss future work in Section 9.

2. Preliminaries

An *alphabet* is a nonempty finite set Σ of *letters*. A *word* over Σ is a finite sequence of letters from Σ. The *empty word* is denoted by ϵ, and the set of all words is denoted by Σ^*. A *language* is a subset of Σ^*. We assume that the reader is familiar with the syntax and semantics of regular expressions (RE). We use the standard notations $\{\cdot, |, *\}$ for concatenation, union, and Kleene star, respectively, and denote the language of an RE r by $\mathcal{L}(r)$. A language L is *regular* if there exists an RE r such that $\mathcal{L}(r) = L$.

Definition 1. A *nondeterministic finite-word automaton* (NFA) is a tuple $A = \langle \Sigma, Q, Q_0, \delta, F \rangle$, where Σ is an alphabet, Q is a nonempty finite set of *states*, $Q_0 \subseteq Q$ is a set of *initial states*, $F \subseteq Q$ is a set of *accepting states*, and $\delta \subseteq Q \times \Sigma \times Q$ is a *transition relation*.

Given a word $w = \sigma_1\sigma_2 \cdots \sigma_n$ over Σ, a *run of A on w* is a sequence of states $(q_0, q_1, \ldots q_n)$, such that $q_0 \in Q_0$, and for every $0 < i \leq n$, it holds that $(q_{i-1}, \sigma_i, q_i) \in$

δ. The run is *accepting* if $q_n \in F$. We say that A *accepts* w if there exists an accepting run of A on w. The *language* of A, denoted $\mathcal{L}(A)$, is the set of all words that A accepts. It is well-known that a language L is regular iff there exists an NFA A such that $\mathcal{L}(A) = L$.

3. Regular Hyperexpressions

Definition 2. A *hyperword over* Σ is a set of words over Σ and a *hyperlanguage* over Σ is a set of hyperwords over Σ.

Before formally defining regular hyperexpressions, we explain the idea behind them. A *regular hyperexpression* (RHE) over Σ uses a set of *word variables* $X = \{x_1, x_2, \ldots, x_k\}$. When expressing a hyperword S, these variables are assigned words from S. An RHE r is composed of a *quantification condition* α over X, and an underlying RE $\hat{r}$, which represents word assignments to X. An RHE r defines a hyperlanguage $\mathfrak{L}(r)$. The condition α defines the assignments that should be in $\mathcal{L}(\hat{r})$. For example, $\alpha = \exists x_1.\forall x_2$ requires that there exists a word $w_1 \in S$ (assigned to x_1), such that for every word $w_2 \in S$ (assigned to x_2), the word that represents the assignment $x_1 \mapsto w_1, x_2 \mapsto w_2$, is in $\mathcal{L}(\hat{r})$. The hyperword S is in $\mathfrak{L}(r)$ iff S meets these conditions.

We represent an assignment $v : X \to S$ as a *word assignment* $\boldsymbol{w}_v$, which is a word over the alphabet $(\Sigma \cup \{\#\})^X$ (that is, assignments from X to $\Sigma \cup \{\#\}$), where the i'th letter of $\boldsymbol{w}_v$ represents the k i'th letters of the words $v(x_1), \ldots, v(x_k)$ (in case that the words are not of equal length, we "pad" the end of the shorter words with $\#$ symbols). We represent these k i'th letters as an assignment denoted $\{\sigma_{1x_1}, \sigma_{2x_2}, \ldots, \sigma_{kx_k}\}$, where x_j is assigned σ_j. For example, the assignment $v(x_1) = aa$ and $v(x_2) = abb$ is represented by the word assignment $\boldsymbol{w}_v = \{a_{x_1}, a_{x_2}\}\{a_{x_1}, b_{x_2}\}\{\#_{x_1}, b_{x_2}\}$.

Definition 3. A *regular hyperexpression* is a tuple $r = \langle X, \Sigma, \alpha, \hat{r}\rangle$, where $\alpha = \mathbb{Q}_1 x_1 \cdots \mathbb{Q}_k x_k$, where $\mathbb{Q}_i \in \{\exists, \forall\}$ for every $i \in [1, k]$, and where $\hat{r}$ is an RE over $\hat{\Sigma} = (\Sigma \cup \{\#\})^X$.

Let S be a hyperword and let $v : X \to S$ be an assignment of the word variables of r to words in S. We denote by $v[x \mapsto w]$ the assignment obtained from v by assigning the word $w \in S$ to $x \in X$. We represent v by $\boldsymbol{w}_v$. We now define the membership condition of a hyperword S in the hyperlanguage of r. We first define a relation $\vdash$ for S, $\hat{r}$, a quantification condition α, and an assignment $v : X \to S$, as follows.

- For $\alpha = \epsilon$, define $S \vdash_v (\alpha, \hat{r})$ if $\boldsymbol{w}_v \in \mathcal{L}(\hat{r})$.
- For $\alpha = \exists x.\alpha'$, define $S \vdash_v (\alpha, \hat{r})$ if there exists $w \in S$ s.t. $S \vdash_{v[x\mapsto w]} (\alpha', \hat{r})$.
- For $\alpha = \forall x.\alpha'$, define $S \vdash_v (\alpha, \hat{r})$ if $S \vdash_{v[x\mapsto w]} (\alpha', \hat{r})$ for every $w \in S$.[1]

[1] In case that α begins with $\forall$, membership holds vacuously with an empty hyperword. We restrict the discussion to nonempty hyperwords.

For example, consider the hyperword $S = \{a, bb\}$, and the RHE $r = \exists x.\forall y.\hat{r}$, where

$$\hat{r} = (\{a_x, b_y\}\{\#_x b_y\}|\{a_x, a_y\}).$$

Then, $S \vdash r$ iff there exists $w_1 \in S$ such that $S \vdash_{[x\mapsto w_1]} \forall y.\hat{r}$, which holds iff for every $w_2 \in S$, it holds that $S \vdash_{[x\mapsto w_1][y\mapsto w_2]} \vdash \hat{r}$. For $w_1 = a$, we have that $S \vdash_{[x\mapsto a][y\mapsto bb]} \vdash \hat{r}$, since $\boldsymbol{w}_{[x\mapsto a][y\mapsto bb]} = \{a_x, b_y\}\{\#_x, b_y\}$, which is in $\mathcal{L}(\hat{r})$, and $S \vdash_{[x\mapsto a][y\mapsto a]} \vdash \hat{r}$, since $\boldsymbol{w}_{[x\mapsto a][y\mapsto a]} = \{a_x, a_y\}$, which is also in $\mathcal{L}(\hat{r})$. Therefore, we have that $S \vdash r$.

Since all variables are under the scope of α, membership is independent of v, and so if $S \vdash (\alpha, \hat{r})$, we denote $S \in \mathfrak{L}(r)$. The hyperlanguage of r is $\mathfrak{L}(r) = \{S \mid S \in \mathfrak{L}(r)\}$.

Definition 4. We call a hyperlanguage $\mathfrak{L}$ a *regular hyperlanguage* if there exists an RHE r such that $\mathfrak{L}(r) = \mathfrak{L}$.

Application of RHE in Information-flow Security

Noninterference [33] requires high-secret commands to be removable without affecting observations of users holding low clearances. More specifically, noninference stipulates that, for all executions, the low-observable behavior must not change when all high inputs are replaced by a dummy input λ, that is, when the high input is removed:

$$\varphi_{\mathsf{ni}} = \forall x.\exists y.\{l_x, l_y^{\lambda}\}^*,$$

where l is a letter in the alphabet that denotes a low state (i.e., the state of only low-security variables) and l^{λ} is a letter in the alphabet that denotes a low state such that all high commands are replaced by a dummy value λ. In other words, l and l^{λ} agree with the value of all low-security variables while all the high commands in l are replaced by λ in l^{λ}.

Observational determinism [47] requires that if two executions of a system start with low-security-equivalent events, they should remain low equivalent. In other words, the system should look deterministic to a low-security user:

$$\varphi_{\mathsf{od}} = \forall x.\forall y.\Big(\{l_x, l_y\}^+ \mid \{\bar{l}_x, \bar{l}_y\}\{\$_x, \$_y\}^* \mid \{l_x, \bar{l}_y\}\{\$_x, \$_y\}^* \mid \{\bar{l}_x, l_y\}\{\$_x, \$_y\}^*\Big)$$

where l denotes the *only* low-security event (for simplicity and without loss of generality[2]), $\bar{l} \in \Sigma \setminus \{l\}$ is any letter in the alphabet but l, and $\$ \in \Sigma$ is any arbitrary letter in the alphabet. We note that similar policies such as *Boudol and Castellani's noninterference* [32] can be formulated in the same fashion. [3]

Declassification [41] relaxes noninterference by allowing leaking information when necessary. Some programs must reveal secret information to fulfill functional requirements. For example, a password checker must reveal whether the entered password is correct or not:

$$\varphi_{\mathsf{dc}} = \forall x.\forall y.\{l_x^i, l_y^i\}\{p_x, p_y\}\{l_x^o, l_y^o\}^+$$

[2]We emphasize that if there exist several low-security events, then φ_{od} can be extended simply by enumerating their combinations.

[3]This policy states that every two executions that start from bisimilar states (in terms of memory low-observability), should remain bisimilarly low-observable.

where letter l^i denotes a low-input state, letter p denotes that the password is correct (not a unique password, but the fact that the password check is successful), and letter l^o denotes a low-output state. We note that for brevity, φ_{dc} does not include behaviors where the first two events are not low or, in the second event, the password is not valid.

4. Nondeterminsitic Finite-Word Hyperautomata

We now present a model for regular hyperlanguages, namely *finite-word hyperautomata*. A hyperautomaton is composed of a set X of word variables, a quantification condition, and an underlying finite-word automaton that accepts representations of assignments to X.

Definition 5. A *nondeterministic finite-word hyperautomaton* (NFH) is a tuple $\mathcal{A} = \langle \Sigma, X, Q, Q_0, F, \delta, \alpha \rangle$, where Σ, X and α are as in Definition 3, and where $\langle \hat{\Sigma}, Q, Q_0, F, \delta \rangle$ forms an underlying NFA over $\hat{\Sigma} = (\Sigma \cup \{\#\})^X$.

The acceptance condition for NFH, as for RHE, is defined with respect to a hyperword S, the NFH $\mathcal{A}$, the quantification condition α, and an assignment $v : X \to S$. For the base case of $\alpha = \epsilon$, we define $S \vdash_v (\alpha, \mathcal{A})$ if $\hat{\mathcal{A}}$ accepts $\boldsymbol{w}_v$. For example, $\{a, ab\} \vdash_{[x \mapsto a][y \mapsto ab]} (\epsilon, \hat{A}_1)$, where $\hat{A}_1$ is the underlying NFA of $\mathcal{A}_1$ of Figure 1. Indeed, $\hat{A}_1$ accepts $\boldsymbol{w}_{[x \mapsto a][y \mapsto ab]} = \{a_x, a_y\}\{\#_x, b_y\}$. The cases where α is of the type $\exists x.\alpha'$ and $\forall x.\alpha'$ are defined similarly as for RHE, and if $S \vdash (\alpha, \mathcal{A})$, we say that $\mathcal{A}$ *accepts* S.

Definition 6. Let $\mathcal{A}$ be an NFH. The *hyperlanguage* of $\mathcal{A}$, denoted $\mathfrak{L}(\mathcal{A})$, is the set of all hyperwords that $\mathcal{A}$ accepts.

Example 3. Consider the NFH $\mathcal{A}_1$ in Figure 1 (left), whose alphabet is $\Sigma = \{a, b\}$, over two word variables x and y. The NFH $\mathcal{A}_1$ contains an underlying standard NFA $\hat{\mathcal{A}}_1$. For two words w_1, w_2 that are assigned to x and y, respectively, $\hat{\mathcal{A}}_1$ requires that (1) w_1, w_2 agree on their a (and, consequently, on their b) positions, and (2) once one of the words has ended (denoted by $\#$), the other must only contain b letters. Since the quantification condition of $\mathcal{A}_1$ is $\forall x_1.\forall x_2$, in a hyperword S that is accepted by $\mathcal{A}_1$, every two words agree on their a positions. As a result, all the words in S must agree on their a positions. The hyperlanguage of $\mathcal{A}_1$ is then all hyperwords in which all words agree on their a positions.

Example 4. The NFH $\mathcal{A}_2$ of Figure 1 (right) depicts the translation of the RHE of Example 1 to an NFH.

Since regular expressions are equivalent to NFA, we can translate the underlying regular expression $\hat{r}$ of an RHE r to an equivalent NFA, and vice versa – translate the underlying NFA $\hat{\mathcal{A}}$ of an NFH $\mathcal{A}$ to a regular expression. It is then easy to see that every RHE has an equivalent NFH over the same set of variables with the same quantification condition.

We consider several fragments of NFH, which limit the structure of the quantification condition α. RHE$_\forall$ is the fragment in which α contains only $\forall$ quantifiers, and similarly, in RHE$_\exists$, α contains only $\exists$ quantifiers. In the fragment RHE$_{\exists\forall}$, α is of the form $\exists x_1 \cdots \exists x_i.\forall x_{i+1} \cdots \forall x_k$.

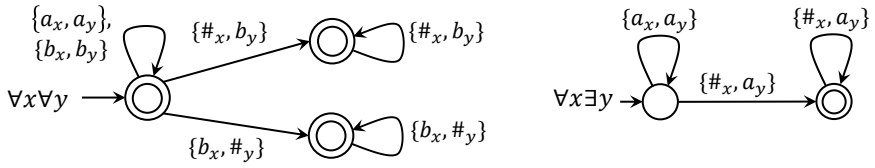

Figure 1: The NFH $\mathcal{A}_1$ (left) and $\mathcal{A}_2$ (right).

4.1. Additional Terms and Notations

We present several terms and notations which we use throughout the paper. Recall that we represent an assignment $v : X \to S$ as a word assignment $\boldsymbol{w}_v$. Conversely, a word $\boldsymbol{w}$ over $(\Sigma \cup \{\#\})^X$ represents an assignment $v_{\boldsymbol{w}} : X \to \Sigma^*$, where $v_{\boldsymbol{w}}(x_i)$ is formed by concatenating the letters of Σ that are assigned to x_i in the letters of $\boldsymbol{w}$. We denote the set of all such words $\{v_{\boldsymbol{w}}(x_1), \ldots, v_{\boldsymbol{w}}(x_k)\}$ by $S(\boldsymbol{w})$. For example, for $\boldsymbol{w} = \{a_x, a_y\}\{\#_x, b_y\}$, we have that $v_{\boldsymbol{w}}(x) = a$, $v_{\boldsymbol{w}}(y) = ab$, and $S(\boldsymbol{w}) = \{a, ab\}$. Since we only allow padding at the end of a word, if a padding occurs in the middle of $\boldsymbol{w}$, then $\boldsymbol{w}$ does not represent a legal assignment. Notice that this occurs iff $\boldsymbol{w}$ contains two consecutive letters $\boldsymbol{w}_i\boldsymbol{w}_{i+1}$ such that $\boldsymbol{w}_i(x) = \#$ and $\boldsymbol{w}_{i+1}(x) \neq \#$ for some $x \in X$. We call $\boldsymbol{w}$ *legal* if $v_{\boldsymbol{w}}$ represents a legal assignment from X to Σ^*.

Consider a function $g : A \to B$ where A, B are some sets. The *range* of g, denoted $\mathsf{range}(g)$ is the set $\{g(a) | a \in A\}$.

A *sequence* of g is a function $g' : A \to B$ such that $\mathsf{range}(g') \subseteq \mathsf{range}(g)$. A *permutation* of g is a function $g' : A \to B$ such that $\mathsf{range}(g') = \mathsf{range}(g)$. We extend the notions of sequences and permutations to word assignments. Let $\boldsymbol{w}$ be a word over $\hat{\Sigma}$. A sequence of $\boldsymbol{w}$ is a word $\boldsymbol{w}'$ such that $S(w') \subseteq S(w)$, and a permutation of $\boldsymbol{w}$ is a word $\boldsymbol{w}'$ such that $S(w') = S(w)$. For example, for $\boldsymbol{w} = \{a_x, a_y\}\{\#_x, b_y\}$, the word $\boldsymbol{w}_1 = \{a_x, a_y\}\{b_x, b_y\}$ is a sequence of $\boldsymbol{w}$, since $S(\boldsymbol{w}_1) = \{ab\} \subseteq \{a, ab\} = S(\boldsymbol{w})$. The word $\boldsymbol{w}_2 = \{a_x, a_y\}\{b_x, \#_y\}$ is a permutation of $\boldsymbol{w}$, since $S(\boldsymbol{w}_2) = \{ab, a\} = \{a, ab\} = S(\boldsymbol{w})$.

Finally, for $n, m \in \mathbb{N}, n < m$, we use $[n, m]$ to denote the set $n, n+1, \ldots m$.

Throughout the paper, when we use a general NFH $\mathcal{A}$, we assume that its ingredients are as in Definition 5.

5. Closure Properties of Regular Hyperlanguages

We now consider closure properties of regular hyperlanguages. We show, via constructions on NFH, that regular hyperlanguages are closed under all the Boolean operations.

Theorem 1. *Regular hyperlanguages are closed under complementation.*

Proof. Let $\mathcal{A}$ be an NFH. The NFA $\hat{\mathcal{A}}$ can be complemented with respect to its language over $\hat{\Sigma}$ to an NFA $\overline{\hat{\mathcal{A}}}$. Then, for every assignment $v : X \to S$, it holds that $\hat{\mathcal{A}}$ accepts $\boldsymbol{w}_v$ iff $\overline{\hat{\mathcal{A}}}$ does not accept $\boldsymbol{w}_v$. Let $\overline{\alpha}$ be the quantification condition obtained from α by replacing every $\exists$ with $\forall$ and vice versa. We can prove by induction on

α that $\overline{\mathcal{A}}$, the NFH whose underlying NFA is $\overline{\hat{\mathcal{A}}}$, and whose quantification condition is $\overline{\alpha}$, accepts $\overline{\mathfrak{L}(\mathcal{A})}$. The size of $\overline{\mathcal{A}}$ is exponential in $|\hat{\mathcal{A}}|$, due to the complementation construction for $\hat{\mathcal{A}}$ and complementing the set of transitions in δ. □

Theorem 2. *Regular hyperlanguages are closed under intersection.*

Proof. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be NFH whose quantification conditions are α_1 and α_2, respectively, and whose variable sets are X and Y, respectively. We construct an NFH $\mathcal{A}_\cap$ whose quantification condition is a concatenation of α_1 and α_2, that is based on the product construction of $\hat{\mathcal{A}}_1$ and $\hat{\mathcal{A}}_2$, in which the letters in every transition are the union of the letters in the respective transitions in $\hat{\mathcal{A}}_1$ and $\hat{\mathcal{A}}_2$. Thus, for a hyperword S and for $v : (X \cup Y) \to S$, we have that $\boldsymbol{w}_v$ is accepted by $\hat{\mathcal{A}}_\cap$ iff $\boldsymbol{w}_{v|_X}$ and $\boldsymbol{w}_{v|_Y}$ are accepted by $\hat{\mathcal{A}}_1$ and $\hat{\mathcal{A}}_2$, respectively. This construction is polynomial in the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$. □

Following De-Morgan's law, we have the following.

Theorem 3. *Regular hyperlanguages are closed under union.*

6. Nonemptiness of NFH.

The *nonemptiness problem* is to decide, given an NFH $\mathcal{A}$, whether $\mathfrak{L}(\mathcal{A}) = \emptyset$. The complexity of the nonemptiness problem affects the complexity of various other decision problems, such as universality and containment. In this section, we extensively study various versions of this problem for various fragments of NFH. First, we show that the problem for general NFH is undecidable. Then, we show that nonemptiness is decidable for various fragments of NFH, with varying complexities.

We then study the *bounded nonemptiness* problem, in which we ask whether an NFH accepts a hyperword of bounded size.

Finally, we study the nonemptiness problem in the presence of *wild-card letters*, which represent free assignments to a variable. Wild-card letters can exponentially decrease the number of transitions of an NFH. We show that for the alternation-free fragments of NFH, wild-card letters do not increase the complexity of the nonemptiness problem, while for the fragment of $\text{NFH}_{\exists\forall}$, the smaller representation comes with an exponential blow-up in complexity.

6.1. General Nonemptiness Results

We begin with the nonemptiness problem for general NFH.

Theorem 4. *The nonemptiness problem for NFH is undecidable.*

Proof. In [23], a reduction from the *Post correspondence problem (PCP)* is used for proving the undecidability of HyperLTL satisfiability. We mimic the proof ideas of [23] to show that the nonemptiness problem for NFH is, in general, undecidable. A PCP instance is a collection C of dominoes of the form:

$$\left\{ \left[\frac{u_1}{v_1}\right], \left[\frac{u_2}{v_2}\right], \ldots, \left[\frac{u_k}{v_k}\right] \right\}$$

where for all $i \in [1, k]$, we have $v_i, u_i \in \{a, b\}^*$. The problem is to decide whether there exists a finite sequence of the dominoes of the form

$$\left[\frac{u_{i_1}}{v_{i_1}}\right]\left[\frac{u_{i_2}}{v_{i_2}}\right] \cdots \left[\frac{u_{i_m}}{v_{i_m}}\right]$$

where each index i_j is in $[1, k]$, such that the upper and lower finite strings of the dominoes are equal, i.e.,

$$u_{i_1} u_{i_2} \cdots u_{i_m} = v_{i_1} v_{i_2} \cdots v_{i_m}.$$

For example, if the set of dominoes is

$$C_{\mathsf{exmp}} = \left\{ \left[\frac{ab}{b}\right], \left[\frac{ba}{a}\right], \left[\frac{a}{aba}\right] \right\}$$

then a possible solution is the following sequence of dominoes from C_{exmp}:

$$\mathsf{sol} = \left[\frac{a}{aba}\right]\left[\frac{ba}{a}\right]\left[\frac{ab}{b}\right].$$

Given an instance C of PCP, we encode a solution as a word w_{sol} over the following alphabet:

$$\Sigma = \left\{ \frac{\sigma}{\sigma'} \mid \sigma, \sigma' \in \{a, b, \dot{a}, \dot{b}, \$\} \right\}.$$

Intuitively, a dotted letter ($\dot{a}$ or $\dot{b}$) marks the beginning of a new domino, and \$ marks the end of a sequence of the upper or lower parts of the dominoes sequence.

We note that w_{sol} encodes a legal solution iff the following conditions are met:

1. For every $\frac{\sigma}{\sigma'}$ that occurs in w_{sol}, it holds that σ, σ' represent the same domino letter (both a or both b, either dotted or undotted).

2. The number of dotted letters in the upper part of w_{sol} is equal to the number of dotted letters in the lower part of w_{sol}.

3. The upper and lower parts of the first letter in w_{sol} are both dotted. Further, for every i, the word u_i between the i'th and $i+1$'th dotted letters in the upper part of w_{sol}, and the word v_i between the corresponding dotted letters in the lower part of w_{sol} are such that $[\frac{u_i}{v_i}] \in C$.

We call a word that represents the removal of the first k dominoes from w_{sol} for some k a *partial solution*, denoted by $w_{sol,k}$. Note that the upper and lower parts of $w_{sol,k}$ are not necessarily of equal lengths (in terms of a and b sequences), since the upper and lower parts of a domino may be of different lengths, and so we use letter \$ to pad the end of the encoding in the shorter of the two parts.

We construct an NFH $\mathcal{A}$, which, intuitively, expresses the following ideas: (1) There exists an encoding w_{sol} of a solution to C, and (2) For every $w_{sol,k} \neq \epsilon$ in a hyperword S accepted by $\mathcal{A}$, the word $w_{sol,k+1}$ is also in S.

$\mathfrak{L}(\mathcal{A})$ is then the set of all hyperwords that contain an encoded solution w_{sol}, as well as all its suffixes obtained by removing a prefix of dominoes from w_{sol}. This ensures that w_{sol} indeed encodes a legal solution. For example, a matching hyperword S (for the solution sol discussed earlier) that is accepted by $\mathcal{A}$ is:

$$S = \{w_{sol} = \frac{\dot{a}}{\dot{a}}\frac{\dot{b}}{b}\frac{a}{a}\frac{\dot{a}}{\dot{a}}\frac{b}{\dot{b}}, w_{sol,1} = \frac{\dot{b}}{\dot{a}}\frac{a}{\dot{b}}\frac{\dot{a}}{\$}\frac{b}{\$}, w_{sol,2} = \frac{\dot{a}}{\dot{b}}\frac{b}{\$}, w_{sol,3} = \epsilon\}.$$

Thus, the quantification condition of $\mathcal{A}$ is $\alpha = \forall x_1.\exists x_2 \exists x_3$, where x_1 is to be assigned a potential partial solution $w_{sol,k}$, and x_2 is to be assigned $w_{sol,k+1}$, and x_3 is to be assigned w_{sol}.

Intuitively, $\mathcal{A}$ makes sure that for every word w in a hyperword S accepted by $\mathcal{A}$, it holds that w starts with a legal tile, and there exists a word w' that is obtained from w by removing the first tile from w. Since w' must also be assigned to x_1 in some accepting run, it too beings with a legal tile, and so on. Hence, every word in S must consist of a legal sequence of tiles. Additionally, α requires that there exists a word w_{sol} in S in which the upper and lower parts are equal, and so w_{sol} is indeed a legal solution of C.

During a run on a hyperword S and an assignment $v : \{x_1, x_2, x_3\} \to S$, the NFH $\mathcal{A}$ checks that the upper and lower letters of w_{sol} all match. That is, the transitions in $\mathcal{A}$ proceed only on letters in which x_3 is assigned letters of the type $\frac{\sigma}{\sigma}$ (where either the upper or lower letters may also be dotted, e.g., $\frac{\dot{b}}{b}$).

In addition, $\mathcal{A}$ checks that the first domino of $v(x_1)$ is indeed in C, and that $v(x_2)$ is obtained from $v(x_1)$ by removing the first tile (that is, that the assignments to x_1 and x_2 can indeed be $w_{sol,k}$ and $w_{sol,k+1}$, respectively).

For the former task, $\mathcal{A}$ proceeds only on words in which $v(x_1)$ starts with a letter whose upper and lower parts are both dotted, and that the sequence of letters c_1 until the second occurrence of a dotted letter in the upper part, and the sequence of letters c_2 until the second occurrence of a dotted letter in the lower part are such that $[\frac{c_1}{c_2}] \in C$. Since C is finite, there are finitely many prefixes of $v(x_1)$ that must be checked by $\mathcal{A}$.

For the latter task, $\mathcal{A}$ checks that the upper and lower parts of $v(x_2)$ are the upper and lower parts of $v(x_1)$ that have been "shifted" back appropriately. That is, if the first tile in $v(x_2)$ is the encoding of $[\frac{w}{v}]$, then $v(x_2)$ is obtained by removing w from the upper part of $v(x_1)$, and removing v from the lower part of $v(x_1)$. To this end, $\mathcal{A}$ remembers, at each point in the run, the last $|w|$ letters of the upper part of $v(x_2)$ and the last $|v|$ letters of the lower part of $v(x_2)$ in a state. $\mathcal{A}$ then verifies that the next letter in $v(x_1)$ matches the matching letter remembered by the state.

For example, consider $w_{sol,1} = \frac{\dot{b}}{\dot{a}}\frac{a}{\dot{b}}\frac{\dot{a}}{\$}\frac{b}{\$}, w_{sol,2} = \frac{\dot{a}}{\dot{b}}\frac{b}{\$}$ from the example above, and consider the beginning of a run in which $v(x_1) = w_{sol,1}$ and $v(x_2) = w_{sol,2}$. Then $\mathcal{A}$ must make sure that the upper part of $v(x_2)$ is the upper part of $v(x_1)$ that has been shifted back two letters, and that the lower part of $v(x_2)$ is the lower part of $v(x_1)$ that has been shifted back one letter. This is achieved by using states to remember, at every point, the previous two letters of the upper part of $v(x_2)$, and the previous letter of the lower part of $v(x_2)$, and comparing them to the current letters in $v(x_1)$. Since C is finite, there are finitely many lengths of $|w_i|, |v_i|$, leading to finitely many states that must be used for this check.

According to this construction, $\mathcal{A}$ is nonempty iff it accepts a hyperword that contains a solution w_{sol} to C, and all its suffixes obtained by removing a prefix of the dominoes. □

Next, we show that for the alternation-free fragments, a simple reachability test suffices to decide nonemptiness.

Theorem 5. *The nonemptiness problem for NFH$_\exists$ and NFH$_\forall$ is NL-complete.*

Proof. The lower bound for both fragments follows from the NL-hardness of the nonemptiness problem for NFA.

We turn to the upper bound, and begin with NFH$_\exists$. Let $\mathcal{A}_\exists$ be an NFH$_\exists$. We claim that $\mathcal{A}_\exists$ is nonempty iff $\hat{\mathcal{A}}_\exists$ accepts some legal word $\boldsymbol{w}$. The first direction is trivial. For the second direction, let $\boldsymbol{w} \in \mathcal{L}(\hat{\mathcal{A}}_\exists)$. By assigning $v(x_i) = v_{\boldsymbol{w}}(x_i)$ for every $x_i \in X$, we get $\boldsymbol{w}_v = \boldsymbol{w}$, and according to the semantics of $\exists$, we have that $\mathcal{A}_\exists$ accepts $S(\boldsymbol{w})$. To check whether $\hat{\mathcal{A}}_\exists$ accepts a legal word, we can run a reachability check on-the-fly, while advancing from a letter σ to the next letter σ' only if σ' assigns $\#$ to all variables for which σ assigns $\#$. While each transition $T = q \xrightarrow{f} p$ in $\hat{\mathcal{A}}$ is of size k, we can encode T as a set of size k of encodings of transitions of type $q \xrightarrow{(x_i,\sigma_i)} p$ with a binary encoding of p, q, σ_i, as well as i, t, where t marks the index of T within the set of transitions of $\hat{\mathcal{A}}$. Therefore, the reachability test can be performed within space that is logarithmic in the size of $\mathcal{A}_\exists$.

Now, let $\mathcal{A}_\forall$ be an NFH$_\forall$ over X. We claim that $\mathcal{A}_\forall$ is nonempty iff $\mathcal{A}_\forall$ accepts a hyperword of size 1. For the first direction, let $S \in \mathfrak{L}(\mathcal{A}_\forall)$. Then, by the semantics of $\forall$, we have that for every assignment $v : X \to S$, it holds that $\boldsymbol{w}_v \in \mathcal{L}(\hat{\mathcal{A}}_\forall)$. Let $u \in S$, and let $v_u(x_i) = u$ for every $x_i \in X$. Then, in particular, $\boldsymbol{w}_{v_u} \in \mathcal{L}(\hat{\mathcal{A}}_\forall)$. Then for every assignment $v : X \to \{u\}$ (which consists of the single assignment v_u), it holds that $\hat{\mathcal{A}}_\forall$ accepts $\boldsymbol{w}_v$, and therefore $\mathcal{A}_\forall$ accepts $\{u\}$. The second direction is trivial.

To check whether $\mathcal{A}_\forall$ accepts a hyperword of size 1, we restrict the reachability test on $\hat{\mathcal{A}}_\forall$ to letters over $\hat{\Sigma}$ that represent fixed functions. □

For NFH$_{\exists\forall}$, we show that the problem is decidable, by checking the nonemptiness of an exponentially larger equi-empty NFA.

Theorem 6. *The nonemptiness problem for NFH$_{\exists\forall}$ is PSPACE-complete.*

Proof. Let $\mathcal{A}$ be an NFH$_{\exists\forall}$ with k quantifiers and m $\exists$-quatifiers. We begin with a PSPACE upper bound.

Let $S \in \mathfrak{L}(\mathcal{A})$. Then, according to the semantics of the quantifiers, there exist $w_1, \ldots w_m \in S$, such that for every assignment $v : X \to S$ in which $v(x_i) = w_i$ for every $1 \leq i \leq m$, it holds that $\hat{\mathcal{A}}$ accepts $\boldsymbol{w}_v$. Let $v : X \to S$ be such an assignment. Then, $\hat{\mathcal{A}}$ accepts $\boldsymbol{w}_{v'}$ for every sequence v' of v that agrees with v on its assignments to $x_1, \ldots, x_m$, and in particular, for such sequences whose range is $\{w_1, \ldots, w_m\}$. Therefore, by the semantics of the quantifiers, we have that $\{w_1, \ldots, w_m\}$ is in $\mathfrak{L}(\mathcal{A})$. The second direction is trivial.

We call $\boldsymbol{w}_{v'}$ as described above a *witness to the nonemptiness of* $\mathcal{A}$. We construct an NFA A based on $\hat{\mathcal{A}}$ that is nonempty iff $\hat{\mathcal{A}}$ accepts a witness to the nonemptiness of $\mathcal{A}$.

Let Γ be the set of all functions of the type $\zeta : [1,k] \to [1,m]$ such that $\zeta(i) = i$ for every $i \in [1,m]$, and such that $\mathsf{range}(\zeta) = [1,m]$. For a letter assignment $f = \{\sigma_{1_{x_1}}, \ldots \sigma_{k_{x_k}}\}$, we denote by f_ζ the letter assignment $\{\sigma_{\zeta(1)_{x_1}}, \ldots, \sigma_{\zeta(k)_{x_k}}\}$.

For every function $\zeta \in \Gamma$, we construct an NFA $A_\zeta = \langle \hat{\Sigma}, Q, Q_0, \delta_\zeta, F \rangle$, where for every $q \xrightarrow{g} q'$ in δ, we have $q \xrightarrow{f} q'$ in δ_ζ, for every f that occurs in $\hat{\mathcal{A}}$ for which $f_\zeta = g$. Intuitively, for every run of A_ζ on a word $\boldsymbol{w}$ there exists a similar run of $\hat{\mathcal{A}}$ on the sequence of $\boldsymbol{w}$ that matches ζ. Therefore, $\hat{\mathcal{A}}$ accepts a witness $\boldsymbol{w}$ to the nonemptiness of $\mathcal{A}$ iff $\boldsymbol{w} \in \mathcal{L}(A_\zeta)$ for every $\zeta \in \Gamma$.

We define $A = \bigcap_{\zeta \in \Gamma} A_\zeta$. Then $\hat{\mathcal{A}}$ accepts a witness to the nonemptiness of $\mathcal{A}$ iff A is nonempty.

Since $|\Gamma| = m^{k-m}$, the state space of A is of size $O(n^{m^{k-m}})$, where $n = |Q|$, and its alphabet is of size $|\hat{\Sigma}|$. Notice that for $\mathcal{A}$ to be nonempty, δ must be of size at least $|(\Sigma \cup \#)|^{(k-m)}$, to account for all the sequences of letters in the words assigned to the variables under $\forall$ quantifiers (otherwise, we can immediately return "empty"). Therefore, $|\hat{\mathcal{A}}|$ is $O(n \cdot |\Sigma|^k)$. We then have that the size of A is $O(|\hat{\mathcal{A}}|^k)$. If the number $k-m$ of $\forall$ quantifiers is fixed, then m^{k-m} is polynomial in k. However, now $|\hat{\mathcal{A}}|$ may be polynomial in n, k, and $|\Sigma|$, and so in this case as well, the size of A is $O(|\hat{\mathcal{A}}|^k)$.

Since the nonemptiness problem for NFA is NL-complete, the problem for $\text{NFH}_{\exists\forall}$ can be decided in space of size that is polynomial in $|\hat{\mathcal{A}}|$.

For the lower bound, we show a reduction from a polynomial version of the *corridor tiling problem*, defined as follows. We are given a finite set T of tiles, two relations $V \subseteq T \times T$ and $H \subseteq T \times T$, an initial tile t_0, a final tile t_f, and a bound $n > 0$. We have to decide whether there is some $m > 0$ and a tiling of a $n \times m$-grid such that (1) The tile t_0 is in the top left corner and the tile t_f is in the bottom right corner, (2) A horizontal condition: every pair of horizontal neighbors is in H, and (3) A vertical condition: every pair of vertical neighbors is in V. When n is given in unary notation, the problem is known to be PSPACE-complete [19]. Given an instance C of the tiling problem, we construct an $\text{NFH}_{\exists\forall}$ $\mathcal{A}$ that is nonempty iff C has a solution. We encode a solution to C as a word $w_{sol} = w_1 \cdot w_2 \cdot w_m\$$ over $\Sigma = T \cup \{0, 1, \ldots n-1, \$\}$, where the word w_i, of the form $0 \cdot t_{0,i} \cdot 1 \cdot t_{1,i}, \ldots n-1 \cdot t_{n-1,i}$, describes the contents of row i.

To check that w_{sol} indeed encodes a solution, we need to make sure that:

1. w_1 begins with t_0 and w_m ends with $t_f\$$.
2. w_i is of the correct form.
3. Within every w_i, it holds that $(t_{j,i}, t_{j+1,i}) \in H$.
4. For w_i, w_{i+1}, it holds that $(t_{j,i}, t_{j,i+1}) \in V$ for every $j \in [0, n-1]$.

Verifying items $1-3$ is easy via an NFA of size $O(n|H|)$. The main obstacle is checking item 4 within polynomial space.

We describe an $\text{NFH}_{\exists\forall}$ $\mathcal{A} = \langle T \cup \{0, 1, \ldots n-1, \$\}, \{y_1, y_2, y_3, x_1, \ldots x_{\log(n)}\}, Q, \{q_0\}, \delta, F, \alpha \rangle$ that is nonempty iff there exists a word that satisfies items $1-4$. The quantification condition α is $\exists y_1 . \exists y_2 . \exists y_3 . \forall x_1 \ldots \forall x_{\log(n)}$.

Intuitively, y_1 is to be assigned w_{sol}, and every x-variable is either assigned w_{sol}, or is assigned a bit value of 0 or 1. When all x-variables are assigned bit values, the variables $x_1, \ldots x_{\log(n)}$ compose a binary encoding of a value between 0 and $n-1$. When this value is i, the run checks that the tiles in the i'th position in every row in w_{sol} match one another according to V. Since the x-variables are under $\forall$, the ultimate requirement is for the vertical rule V to hold for every $i \in [0, n-1]$.

To implement this idea, $\mathcal{A}$ requires the existence of the words $0^{|w_{sol}|}$ and $1^{|w_{sol}|}$ (the 0-word and 1-word, henceforth), which represent bit values, and are to be assigned to y_1 and y_2, respectively.

A run of $\hat{\mathcal{A}}$ on an assignment to $y_1, y_2, y_3, x_1, x_2, \ldots x_{\log(n)}$ performs the following checks:

1. y_1 conforms to the pattern of a legal solution as described in items $1-3$ above.
2. y_2 is assigned 0 throughout the run, and y_3 is assigned 1 throughout the run.
3. Upon reading the first letter, if $x_1, \ldots x_{\log(n)}$ contain only 0 or 1-letters, then $\mathcal{A}$ remembers the value i encoded in $x_1, \ldots x_{\log(n)}$ in a designated state for i. Then, throughout the run, whenever $\mathcal{A}$ reaches a letter that matches position i in a row in w_{sol}, it remembers the current tile in a state, and checks that the current tile matches the i'th tile from the previous row (which is remembered using the state).
4. If one of the x-variables is not assigned 0 or 1, then $\mathcal{A}$ requires that the word assigned to it is equal to the one assigned to y_1. In such cases, $\mathcal{A}$ does not use the x-variables to check the vertical condition. Since w_{sol} must be included in a hyperword accepted by $\mathcal{A}$, this takes care of the $\forall$-requirement for the x-variables, and of all the runs in which w_{sol} is assigned to one of them.

According to the construction of $\mathcal{A}$, if it is nonempty then it accepts the hyperword $\{w_{sol}, 0^{|w_{sol}|}, 1^{|w_{sol}|}\}$, where w_{sol} encodes a legal solution to C. Conversely, if there exists a legal solution to C encoded by w_{sol}, then $\mathcal{A}$ accepts the hyperword $\{w_{sol}, 0^{|w_{sol}|}, 1^{|w_{sol}|}\}$.

We construct a similar reduction for the case that the number of $\forall$-quantifiers is fixed: instead of encoding the position by $\log(n)$ bits, we can directly specify the position by a word of the form $j^{w_{sol}}$, for every $j \in [0, n-1]$. Accordingly, we construct an NFH$_{\exists\forall}$ $\mathcal{B}$ over $\{x, y_1, \ldots y_n, z\}$, with a quantification condition $\alpha = \exists x. \exists y_1 \ldots \exists y_n. \forall z$. The NFA $\hat{\mathcal{B}}$ advances only on letters whose assignments to $y_1, \ldots y_n$ are $0, 1, \ldots n-1$, respectively (thus making sure that a hyperword accepted by $\mathcal{B}$ contains all indices), and checks only words assigned to z that are some $j \in [0, n-1]$. Notice that the fixed assignments to the y-variables lead to a transition relation of polynomial size. In a hyperword accepted by $\mathcal{A}$, the word assigned to x is w_{sol}, and the word assigned to z specifies the index that should be checked for conforming to V in the current run. □

6.2. Bounded nonemptiness

The *bounded nonemptiness problem* is to decide, given an NFH $\mathcal{A}$ and $m \in \mathbb{N}$, whether $\mathcal{A}$ accepts a hyperword of size at most m. Notice that some nonempty NFH

only accept infinite hyperwords (for example, $\mathcal{A}_2$ of Figure 1), and so they do not accept a hyperword of size m, for every $m \in \mathbb{N}$.

We show that the bounded nonemptiness problem is decidable for all of NFH.

Lemma 1. *The bounded nonemptiness problem for NFH is in PSPACE.*

Proof. Let $\mathcal{A}$ be an NFH with a quantification condition α with k quantifiers, and let $m \in \mathbb{N}$. Intuitively, we construct an NFA A in which a single run simultaneously follows all runs of $\hat{\mathcal{A}}$ on the possible assignments of a potential hyperword S of size m to the variables of $\mathcal{A}$. Then, A accepts a set of such legal assignments (represented as a single word) iff $\mathcal{A}$ accepts a hyperword of size at most m.

The *assignment tree* for α and m is defined as follows. The tree T has $k+1$ levels, where the root is at level 0. For $0 < i \leq k$, if $\mathbb{Q}_i = \forall$, then every node in level $i-1$ has m children. If $\mathbb{Q}_i = \exists$, then every node in level $i-1$ has a single child. Every node v in T is associated with an encoding in $\{1, 2, \ldots, m\}^*$ that matches the path from the root to v. For example, if v is in level 2, and α begins with $\exists\forall$, and v is the second child, then the position of v is encoded by $1 \cdot 2$. The leaves of T are then all encoded by elements of $[1, m]^k$.

A *labeling* c of T labels every node (except for the root) by some value in $[1, m]$. For $0 < i \leq k$, if $\mathbb{Q}_i = \forall$, then the m children of every node in level $i-1$ are labeled 1 to m. If $\mathbb{Q}_i = \exists$, then the child of every node in level $i-1$ is labeled by some value in $[1, m]$.

Consider a hyperword $S = \{w_1, w_2, \ldots w_m\}$. Every path p along $c(T)$ matches an assignment of the words in S to the variables in X: the variable x_i is assigned w_j, where j is the labeling of the node in level i in p. Then, $c(T)$ matches a possible set of assignments of the words of S to the variables in X. Given p, we denote this assignment by f_p. According to the semantics of NFH, we have that $\mathcal{A}$ accepts a hyperword of size m iff there is a labeling $c(T)$ such that for every path p of $c(T)$, the underlying NFA $\hat{\mathcal{A}}$ accepts the word assignment for f_p.

We construct A such that a single run of A simultaneously follows every assignment f_p in a labeling $c(T)$, letter by letter.

Let C be the set of all labelings of T, and let L be the set of all indices of leaves of T. We define the NFA A as follows. The alphabet of A is $(\Sigma \cup \{\#\})^m$. The set of states of A is $Q^L \times C$. The set of initial states is $Q_0^L \times C$, and the set of accepting states is $F^L \times C$.

The transition relation of A is as follows. We add a transition labeled $(\sigma_1, \sigma_2, \ldots \sigma_m)$ from $(((q_1, l_1), \ldots (q_{|L|}, l_{|L|})), c)$ to $(((q'_1, l_1), \ldots (q'_{|L|}, l_{|L|})), c')$ if $c = c'$, and for every $1 \leq r \leq |L|$, there is a transition in δ labeled by $\{\sigma'_{1_{x_1}}, \sigma'_{2_{x_2}}, \ldots \sigma'_{k_{x_k}}\}$ from q_r to q'_r, where $\sigma'_{i_{x_i}} = \sigma_j$, where j is the labeling of the node in level i in the path to l_r in c.

For example, consider the labeled assignment tree $c(T)$ of Figure 2 for the quantification condition $\forall x_1.\exists x_2$, and $m = 3$. Then T has three leaves, labeled $1 \cdot 1$, $2 \cdot 1$, and $3 \cdot 1$. The labeling $c(T)$ assigns the nodes of T values in $[1, 3]$ as described in Figure 2. The three transitions in $\mathcal{A}$ from q_1, q_2, q_3 are then translated to the transition from $s = (((q_1, 1 \cdot 1), (q_2, 2 \cdot 1), (q_3, 3 \cdot 1)), c(T))$ labeled (a, b, c), which means that the transition associates label 1 with a, label 2 with b, and label 3 with c, matching the transitions from q_1,q_2, and q_3, when they are associated with the leaves as in s.

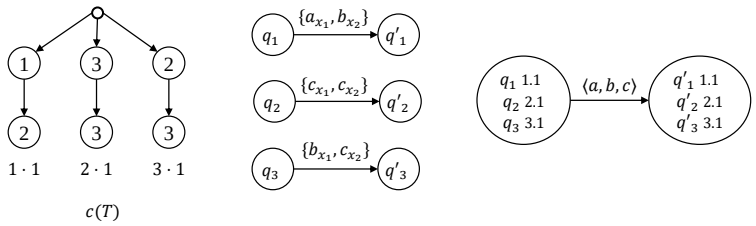

Figure 2: The labeled assignment tree $c(T)$ (left), transitions in $\mathcal{A}$ (middle), and their depiction in A (right).

The size of T (and hence, the size of L) is $O(m^{k'})$, where k' is the number of $\forall$-quantifiers in α. Accordingly, the size of C is $O(m^{m^{k'}})$. Therefore, the state space of A is of size $O(n^{m^{k'}} \cdot m^{m^{k'}})$, where n is the number of states in $\mathcal{A}$.

According to our construction, we have that $\mathcal{A}$ accepts a hyperword of size m iff A is nonempty, when considering only paths that are legal assignments, that is, once a value i is assigned the letter $\#$, it continues to be assigned $\#$. Checking A for such nonemptiness can be done on-the-fly in space that is logarithmic in the size of A. Notice, as mentioned in the proof of theorem 6, that for $m > 1$, the size of the transition relation of $\mathcal{A}$ must be exponential in the size of k', to account for the different assignments to the $\forall$-quantifiers (otherwise, $\mathcal{A}$ is empty and we can return "false"). Therefore, the size of each state of A is polynomial in the size of $\mathcal{A}$, and a PSPACE upper bound follows. □

A PSPACE lower bound for the bounded nonemptiness problem for NFH directly follows from the nonemptiness problem for $\text{NFH}_{\exists\forall}$, since, as we prove in Theorem 6, an $\text{NFH}_{\exists\forall}$ $\mathcal{A}$ with k' $\exists$-quantifiers is nonempty iff it accepts a hyperword of size k'. However, we prove PSPACE-hardness for a $\forall x.\exists y$ quantification condition, showing that this problem is PSPACE-hard even for a fixed number of $\forall$- and $\exists$-quantifiers.

Lemma 2. *The bounded nonemptiness problem for NFH with $\alpha = \forall x.\exists y$ is PSPACE-hard.*

Proof. We reduce from the problem of deciding the nonemptiness of the intersection of k given NFA, which is known to be PSPACE-hard.

Let $A_1, A_2, \ldots A_k$ be NFA, where $A_i = \langle \Sigma, Q_i, Q_0^i, \delta_i, F_i \rangle$. We construct an NFH $\mathcal{A} = \langle \Sigma', \{x, y\}, Q, Q_0, F, \delta, \forall x.\exists y \rangle$ that accepts a hyperword whose size is at most k iff there exists a word w such that $w \in \mathcal{L}(A_i)$ for every $i \in [1, k]$.

The set of states Q of $\mathcal{A}$ is $\bigcup_i (Q_i \times Q_{(i+1) \bmod k})$, and $\Sigma' = \bigcup_i Q_i \times \Sigma \times Q_i$. The set of accepting states is $\bigcup_i (F_i \times F_{(i+1) \bmod k})$, and the set of initial states Q_0 is $\bigcup_i (Q_0^i \times Q_0^{(i+1) \bmod k})$. The transitions are as follows. For every $i \in [1, k]$, every $\sigma \in \Sigma$, and every two transitions $(q, \sigma, q') \in \delta_i, (p, \sigma, p') \in \delta_{(i+1) \bmod k}$, we set $((q, p), \{(q, \sigma, q')_x, (p, \sigma, p')_y\}, (q', p')) \in \delta$. Notice that the size of $\mathcal{A}$ is polynomial in the sizes of $A_1, \ldots A_k$. Every word assignment $\boldsymbol{w}$ that is read along $\hat{\mathcal{A}}$ describes the parallel run of A_i and $A_{(i+1) \bmod k}$ on the same word w. The word assignment $\boldsymbol{w}$ is accepted by $\hat{\mathcal{A}}$ iff w is accepted by both A_i and $A_{(i+1) \bmod k}$.

If there exists a word w that is accepted by all NFA, then the hyperword S that describes all the matching accepting runs on w by the different NFA is accepted by $\mathcal{A}$. Indeed, for the accepting run on w by A_i there is a matching accepting run on w by $A_{(i+1) \bmod k}$.

Conversely, if there exists a hyperword of size (at most) k that is accepted by $\mathcal{A}$, then it contains descriptions of runs of $A_1, \ldots A_k$ on words. By the way we have defined $\mathcal{A}$, if there exists $r \in S$ that describes the accepting run of A_i on a word w, then there must exist $r' \in S$ that describes the accepting run of $A_{(i+1) \bmod k}$ on w. As a result, and combined with the size of S, we have that S must contain an accepting run of every NFA in the set, and these runs must all be on the same word w. Therefore, the intersection of $A_1, \ldots A_k$ is nonempty. □

As a conclusion from Theorems 1 and 2, we have the following.

Theorem 7. *The bounded nonemptiness problem for NFH is PSPACE-complete.*

6.3. NFH with wild-card Letters

When constructing an RHE or an NFH, every letter must include an assignment to all variables. However, an RHE may only need to describe the assignment to a subset of the variables at each step. For example, the RHE

$$\exists x. \exists y. \{a_x\}\{b_y\}$$

describes hyperwords in which there exist two words, where the first word starts with a, and the second word has b in its second position. Since the first letter and the second letter of the second and first words, respectively, do not matter, there is no need to express them. Therefore, we can define a more general and useful notion of RHE in which the letters are *partial* functions from X to Σ.

To translate the notion of partial functions to NFH, we simply add a wild-card letter $\star$ which can stand for every letter assignment to the variables. For example, the letter $\{a_x, \star_y\}$ stands for all the assignments to x, y in which x is assigned a.

The size of the alphabet $\hat{\Sigma}$ of an underlying NFA must be exponential in the size of the number of $\forall$-quantifiers, to account for all the assignments of letters to all the variables under $\forall$-quantifiers. Otherwise, the language of the NFH is empty. Using wild-card letters, such transitions can be replaced by a single transition in which every variable under $\forall$ is assigned $\star$. Thus, using wild-card letters can lead to exponentially smaller NFH.

We define NFH with wild-cards accordingly. An NFH with wild-card letters (NFH*) is a tuple $\mathcal{A} = \langle \Sigma, X, Q, Q_0, F, \delta, \alpha \rangle$ whose underlying NFA $\hat{\mathcal{A}}$ is over the alphabet $\hat{\Sigma} = (\Sigma \cup \{\#, \star\})^X$. The semantics of NFH* is similar to that of NFH. The only difference is that now, $\boldsymbol{w}_v$ contains all possible word assignments in which the letters in Σ may also be replaced with $\star$ in the assignments to the variables.

Obviously, every NFH* can be translated to an NFH with an exponential blow-up in the number of transitions. The constructions for intersection, union, and complementation can all be adjusted to handle the wild-cards. Due to the exponential decrease in size, the complexity of the various decision procedures for NFH* may, in the worst

case, increase exponentially. Since the nonemptiness problem is at the core of most decision procedures, we study its complexity for the various fragments of NFH*.

We begin with NFH$^\star_\exists$ and NFH$^\star_\forall$, and show that for these fragments, adding wild-card letters does not change complexity of the nonemptiness problem.

According to the proof of Theorem 5, a simple reachability test on the underlying NFA suffices to determine nonemptiness for these fragments. We notice that this holds also in the presence of wild-card letters. Indeed, an NFH$^\star_\forall$ is nonempty iff it accepts a hyperword of size 1. The proof of Theorem 5 locates such a word by following an accepting path in the underlying NFH in which all variables are equally assigned at every step. It is easy to see that such a path suffices also when some of the variables are assigned wild-card letters. Similarly, an accepting path in an NFH$_\exists$ induces a finite accepted hyperword, and the same holds also when traversing transitions with wild-card letters. Therefore, we have the following.

Theorem 8. *The nonemptiness problem for NFH$^\star_\exists$ and NFH$^\star_\forall$ is NL-complete.*

We turn to study the fragment of NFH$^\star_{\exists\forall}$. Recall that in the proof for the lower bound of Theorem 6, we argue that the size of the transition relation of a nonempty NFH$_{\exists\forall}$ must be exponential in its number of $\forall$-quantifiers, which affects the space complexity analysis of the size of the NFA that we construct. For an NFH$^\star_{\exists\forall}$ $\mathcal{A}$, this argument no longer holds. While we can construct a similar NFA and check its nonemptiness, its size may now be exponential in that of $\mathcal{A}$, conforming to an EXPSPACE upper bound. We prove a matching lower bound, and conclude that in contrast to the alternation-free fragments, adding wild-card letters hardens the nonemptiness problem for NFH$_{\exists\forall}$.

Theorem 9. *The nonemptiness problem for NFH$^\star_{\exists\forall}$ is EXPSPACE-complete.*

Proof. Let $\mathcal{A}$ be an NFH$_{\exists\forall}$. Consider the NFA A constructed in the proof of Theorem 6. A similar NFA can be constructed to decide the nonemptiness of $\mathcal{A}$. The only difference is the need to consider the intersection of letters which carry wild-card letters. These can be easily computed: the intersection letter of $\{\sigma_{1_{x_1}}, \sigma_{2_{x_2}}, \ldots \sigma_{k_{x_k}}\}$ and $\{\sigma'_{1_{x_1}}, \sigma'_{2_{x_2}}, \ldots \sigma'_{k_{x_k}}\}$ is $\{\gamma_{1_{x_1}}, \gamma_{2_{x_2}}, \ldots \gamma_{k_{x_k}}\}$, where $\gamma_i = \sigma_i$ if $\sigma'_i = \star$, and $\gamma_i = \sigma'_i$ if $\sigma_i = \star$, and otherwise it must hold that $\gamma_i = \sigma_i = \sigma'_i$.

The size of A is, as in the proof of Theorem 6, $O(n^{m^{k-m}})$, where n is the number of states in $\mathcal{A}$, and m is the number of $\exists$-quantifiers in α. Since the nonemptiness problem for NFA is NL-complete, an EXPSPACE upper bound follows.

We turn to the lower bound. As in the proof of Theorem 6, we reduce from the corridor tiling problem: we are given an input C which consists of a finite set T of tiles, two relations $V \subseteq T \times T$ and $H \subseteq T \times T$, an initial tile t_0, a final tile t_f, and a bound $n > 0$. In the exponential version of this problem, we need to decide whether there exists a legal tiling of a $2^n \times m$ for some $m > 0$ (in contrast to the polynomial version which we use for NFH$_{\exists\forall}$). This problem is known to be EXPSPACE-complete.

We use a similar idea as for NFH$_{\exists\forall}$, and encode the legal solution as a word, while using the $0-$ and $1-$words under $\forall$ as memory. However, the exponential length of each row in the tiling poses two main obstacles. First, in a polynomial reduction we can no longer use a state to remember the index in the row that we need to check in

order to verify the vertical condition. Second, we can no longer use numbered letters to mark the index in every row, and using a binary encoding requires verifying that the encoding is correctly ordered. We describe how we overcome these two obstacles by using wild-card letters.

We encode a solution $w_{sol} = \$w_1 \cdot w_2 \cdot w_m\$$ over $\Sigma = T \cup \{0, 1, \$, \&\}$, where the word w_i, of the form $b_0 \cdot t_{0,i} \cdot b_1 \cdot t_{2,i}, \ldots b_{2^n-1} \cdot t_{2^n-1,i}$, describes the contents of row i, where b_j is the n-bit binary encoding of index j. Additionally, we use the 0-word which only consists of 0 letters, and similarly we use the 1-word. Here, we precede the sequence of bits with $\&$.

We construct an $\text{NFH}^{\star}_{\exists\forall}$ $\mathcal{A}$ with a quantification condition $\alpha = \exists s.\exists x_0.\exists x_1.\forall u.\forall y_1 \ldots \forall y_n.\forall z_1 \ldots \forall z_n$ that is nonempty iff C has a solution. Intuitively, as in the proof of Theorem 6, the assignment to s must be w_{sol}, and the assignment to x_0 and x_1 must be the 0- and the 1-words, respectively. The assignment to u must be equal to the assignment of either s, x_0, or x_1. This can be achieved by only using letters in which u is assigned $0, 1$, or the same as s. Notice that since u is under $\forall$, then if $\mathcal{A}$ is nonempty then the only hyperword it can accept is $\{w_{sol}, 0^{|w_{sol}|}, 1^{|w_{sol}|}\}$. Therefore, each of the rest of the variables must always be assigned one of these three words in order for $\mathcal{A}$ to accept.

To check that w_{sol} is legal, $\mathcal{A}$ runs the following checks.

- w_{sol} is of the correct format.
- The indices in every row run from 0 to $2^n - 1$. To this end, we use the y-variables for the binary encoding of a position j, and the z-variables for the binary encoding of position $j + 1$, and check that position $j + 1$ directly follows position j in every row (where 0 follows $2^n - 1$). Since the y- and z-variables are under $\forall$, the requirement holds for every $j \in [0, 2^n - 1]$, throughout w_{sol}.
- Every two consecutive tiles satisfy the horizontal condition.
- For every $j \in [0, 2^n - 1]$, the tiles in position j in every two consecutive rows satisfy the vertical condition. In every run, $\mathcal{A}$ checks this for the position j encoded by the y-variables. Since the y-variables are under $\forall$, the requirement holds for every $j \in [0, 2^n - 1]$, throughout w_{sol}.

We now elaborate on how to handle the encoding of the positions in the y- and z-variables. As we have mentioned, when the assignments to $y_1 \ldots y_n$ are the 0- and 1- words (and not w_{sol}), their binary values are used for encoding a single index j that verifies that every two tiles in position j in two consecutive rows satisfy V. A run in which one of the y- or z-variables is assigned w_{sol} is not used for this check, and in such a case $\mathcal{A}$ accepts. To this end, the transition relation δ of $\hat{\mathcal{A}}$ uses transitions from the initial state labeled by letters in which one of the y- or z-variables is assigned \$, and the rest are assigned $\star$.

To match the encoding of the y-variables with the correct index j in w_{sol}, the transition relation δ of $\mathcal{A}$ describes the n bits of j in cycles of length $n + 1$, where in each cycle, the i'th bit of j is specified in y_i in the i'th step, while the rest of the y-variables are assigned $\star$ in this step. In each cycle, the i'th bit in y_i is compared with the i'th bit in w_{sol}. In cycles in which all n index bits in w_{sol} match those of $y_1 \ldots y_n$,

the tile in the letter that follows the encoding (the $n+1$'th letter in the cycle) is matched with the previous tile, remembered by the state, to verify that they satisfy V.

For example, for $n = 3$, the encoding 101 is as follows.

$$\begin{pmatrix} y_1 & = & \& & 1 & \star & \star & \star & 1 & \star & \star\cdots \\ y_2 & = & \& & \star & 0 & \star & \star & \star & 0 & \star\cdots \\ y_3 & = & \& & \star & \star & 1 & \star & \star & \star & 1\cdots \end{pmatrix}.$$

Notice that (restricting to the y-variables), only $2n + 2$ different letters are needed to describe this encoding: two for every value of the i'th bit (0 or 1), one of all wild-cards, and one for all $\&$. Notice that specifying all bits in a single letter would require exponentially many letters (one for every encoding of a value between 0 and $2^n - 1$), but the wild-card letters allow us to specify the bits one after the other.

We now describe how to verify that the index encoding along w_{sol} is correct. We use the z-variables in a similar way to the y-variables, to encode the successor position of the one encoded in the y-variables. To check that the positions encoded by the y- and z-variables are indeed successors, it suffices to check, within the first cycle, that all bits up to some $1 \leq i < n$ are equal, that $z_i = 1$ and $y_i = 0$, and that $y_{i+1} \dots y_n = 1$ and $z_{i+1} \dots z_n = 0$ (the only exception is for $2^n + 1$ and 0, in which we only need to check that all y bits are 1 and all z bits are 0). Runs of $\hat{\mathcal{A}}$ in which the encoding in the z-variables is not the successor of the encoding of the y-variables, or in which one of the z-variables is assigned w_{sol}, are accepting and are not used for this check. Otherwise, whenever the encoding of the position in w_{sol} is equal to that of the y-variables (we check this bit by bit), we check that the encoding of the position in the next cycle is equal to that of the z-variables.

For example, for checking the successor of 101 (which is 110), the assignments to the y- and z-variables would be as follows.

$$\begin{pmatrix} y_1 & = & \& & 1 & \star & \star & \star & 1 & \star & \star\cdots \\ y_2 & = & \& & \star & 0 & \star & \star & \star & 0 & \star\cdots \\ y_3 & = & \& & \star & \star & 1 & \star & \star & \star & 1\cdots \\ z_1 & = & \& & 1 & \star & \star & \star & 1 & \star & \star\cdots \\ z_2 & = & \& & \star & 1 & \star & \star & \star & 1 & \star\cdots \\ z_3 & = & \& & \star & \star & 0 & \star & \star & \star & 0\cdots \end{pmatrix}.$$

Since the y- and z-variables are under $\forall$, all positions along w_{sol} are checked over all runs of $\hat{\mathcal{A}}$ on the different assignments to the y- and z-variables. It is left to check that the first position in w_{sol} is 0^n, and the last position is 1^n, which can be done during the first and last cycle of states, respectively.

Checking the horizontal condition can be done by comparing every two consecutive tiles in the same row. These tiles are n letters apart, and so this can be done via the states and does not require using the variables as memory. The rest of the checks, i.e, the identity of the first and last tiles, and the correct form of w_{sol}, can also be easily checked by the states.

To summarize, every state in $\hat{\mathcal{A}}$ needs to remember the following information:

- The current bit i, for bits 1 to n.

- The previous tile in the current row, to compare with the following tile for the horizontal condition.
- The tile in the previous row in the position encoded by the y-variables, to compare with the following tile in the same position for the vertical condition.
- Whether or not the bits $1 \dots i$ in w_{sol} match the bits $1 \dots i$ in the y-variables – if this will hold for all current bits $1, \dots n$, then the current position is the one encoded by the y-variables, and the vertical condition should be checked for the following tile in w_{sol}.
- Whether or not the n bits in the previous cycle match the y-variables. If they do, then the run continues only if the current bit i in w_{sol} is equal to the current bit i in z_i. This is checked for all bits $1, \dots n$ in the current cycle. If this does not hold for some bit, then position $j+1$ does not follow position j in w_{sol}, where j is the position encoded by the y-variables, and the run rejects.

Notice that we need polynomially many states to remember the information listed above.

If C has a solution, then it can be encoded by a legal word w_{sol}. In this case, $\mathcal{A}$ accepts the hyperword $\{w_{sol}, 0^{|w_{sol}|}, 1^{|w_{sol}|}\}$. Indeed, the horizontal condition holds in all runs, and in all runs of $\mathcal{A}$ in which the y- and z-variables are assigned positions $j, j+1$, the vertical condition holds for all rows in positions $j, j+1$, and positions $j, j+1$ are encoded legally in w_{sol}, and so all these runs accept. Conversely, if $\mathcal{A}$ is nonempty then since u is under $\forall$, the only hyperword that can be accepted is $\{w_{sol}, 0^{|w_{sol}|}, 1^{|w_{sol}|}\}$. Then the horizontal condition holds in all runs. All runs in which the y- and z-variables are not assigned $j, j+1$ for some $j \in [0, 2^n - 1]$ are accepting. All runs in which the y- and z-variables are assigned $j, j+1$, the vertical condition holds, and the position $j+1$ follows position j in all rows are also accepting. It follows that w_{sol} legally encodes all positions in all rows, and also meets the vertical condition in all positions in all rows. Therefore, C has a solution.

In every letter of $\hat{\mathcal{A}}$ (other than the first in the run, in which all y- and z- variables are assigned &), there are at most six non-wild-card letters: the assignments to s, x_0, x_1 and u, and y_i and z_i for some $1 \leq i \leq n$. Additionally, we have letters in which one of the y- or z- variables is assigned with a word that starts with \$, and the rest are wild-cards. Therefore, the alphabet of $\mathcal{A}$ is polynomial in the size of $|C|$. The number of states needed for the various checks is also polynomial in the size of $|C|$, and so the size of $\mathcal{A}$ is polynomial in $|C|$. □

6.4. A semi-algorithm for deciding the nonemptiness for $\forall\exists$

The nonemptiness problem for NFH is undecidable already for the fragment of $\forall\exists$, as shown in Theorem 4. However, this fragment is of practical use in expressing finite-word properties, as shown in Section 3. We now describe a semi-algorithm for testing the nonemptiness of an NFH with a quantification condition of the type $\forall\exists$. Intuitively, this procedure aims at finding the largest hyperword that is accepted by the NFH.

The procedure first considers the set L_0 of all the words that can be assigned to x_1, and checks whether this set subsumes the matching assignments for the $\exists$-quantifier. If

so, then L_0 is a suitable hyperword. Otherwise, L_0 is pruned to the largest potential hyperword by omitting from L_0 all words that are not assigned to the variable under $\exists$, and the procedure continues to the next round. In case that the procedure does not find an accepted hyperword, or conversely if the procedure does not reach an empty set, it does not halt.

We describe our procedure with more detail. Let $\mathcal{A} = \langle \Sigma, \{x, y\}, Q, Q_0, F, \delta, \forall x.\exists y \rangle$ be an NFH. Let $L^0_\forall = \{u | \exists v : \boldsymbol{w}_{x \mapsto u, y \mapsto v} \in \mathcal{L}(\hat{\mathcal{A}})\}$, and let $L^0_\exists = \{v | \exists u : \boldsymbol{w}_{x \mapsto u, y \mapsto v} \in \mathcal{L}(\hat{\mathcal{A}})\}$. We denote the NFA obtained from $\hat{\mathcal{A}}$ by restricting the transitions to assignments to x by $\hat{\mathcal{A}}_x$, and similarly define $\hat{\mathcal{A}}_y$. It is easy to see that $A^0_\forall = \hat{\mathcal{A}}_x$ is an NFA for $L^0_\forall$, and $A^0_\exists = \hat{\mathcal{A}}_y$ is an NFA for $L^0_\exists$.

If $L^0_\exists \subseteq L^0_\forall$, then by the semantics of NFH, we have that $L^0_\forall$ is accepted by $\mathcal{A}$. If $L^0_\exists \cap L^0_\forall = \emptyset$, then by the semantics of NFH, we have that $\mathcal{A}$ is empty. Otherwise, there exists a word in $L^0_\exists$ that is not in $L^0_\forall$, and vice versa.

We define $L^1_\exists = L^0_\exists \cap L^0_\forall$. Notice that $L^1_\exists$ is regular, and an NFA $A^1_\exists$ for $L^1_\exists$ can be calculated by the intersection construction for $A^0_\forall$ and $A^0_\exists$. Now $L^1_\exists \subseteq L^0_\forall$. However, it may be the case that there exists a word $u \in L^0_\forall$ for which there exists no matching $v \in L^1_\exists$. Therefore, we restrict $L^0_\forall$ to a set $L^1_\forall = \{u | \exists v \in L^1_\exists : \boldsymbol{w}_{x \mapsto u, y \mapsto v} \in \mathcal{L}(\hat{\mathcal{A}})\}$. We calculate an NFA $A^1_\forall$ for L^1, as follows. Let $A^0_\exists = \langle P, \Sigma, p_0, \delta_0, F_0 \rangle$. We define $\hat{A}^1 = \langle Q \times P, (\Sigma \cup \{\#\})^{\{x,y\}}, (q_0, p_0), \delta_1, F_1 \times F_2 \rangle$, where $\delta_1 = \{((q,p), \{\sigma'_x, \sigma_y\}, (q', p') | \sigma, \sigma' \in \Sigma, (q, \{\sigma_x, \sigma'_y\}, q') \in \delta, (p, \sigma', p') \in \delta_0\}$. That is, $\hat{\mathcal{A}}^1$ is roughly the intersection construction of $\hat{\mathcal{A}}$ and $A^0_\exists$, when considering only the letter assignments to y. We denote this construction by $\cap_y$. Finally, we set $A^1_\forall = \hat{\mathcal{A}}^1_x$.

Now, if $\mathcal{L}(A^1_\exists) \subseteq \mathcal{L}(A^1_\forall)$, then $\mathcal{L}(A^1_\forall)$ is accepted by $\mathcal{A}$, and if $\mathcal{L}(A^1_\exists) \cap \mathcal{L}(A^1_\forall) = \emptyset$, then $\mathcal{L}(\mathcal{A}) = \emptyset$. Otherwise, we repeat the process above with respect to $\hat{\mathcal{A}}^1, A^1_\forall, A^1_\exists$.

Algorithm 1 describes the procedure.

Algorithm 1: Nonemptiness test for $\forall\exists$

Input: $\mathcal{A}$.
Output: $\mathfrak{L}(\mathcal{A}) \neq \emptyset$?

$A_\forall = \hat{\mathcal{A}}_x, A_\exists = \hat{\mathcal{A}}_y$
while *true* **do**
  if $\mathcal{L}(A_\exists) \subseteq \mathcal{L}(A_\forall)$ **then**
    return ***tt***
  else if $\mathcal{L}(A_\exists) \cap \mathcal{L}(A_\forall) = \emptyset$ **then**
    return ***ff***
  $A_\exists = A_\exists \cap A_\forall$
  $\hat{\mathcal{A}} = \hat{\mathcal{A}} \cap_y \mathcal{A}_\exists$
  $A_\forall = \hat{\mathcal{A}}_x$
endwhile

7. Additional decision procedures

The *universality problem* is to decide whether a given NFH $\mathcal{A}$ accepts every hyperword over Σ. Notice that $\mathcal{A}$ is universal iff $\overline{\mathcal{A}}$ is empty. Since complementing an NFH involves an exponential blow-up, we conclude the following from the results in Section 6, combined with the PSPACE lower bound for the universality of NFA.

Theorem 10. *The universality problem for*

1. *NFH is undecidable,*
2. *NFH$_\exists$ and NFH$_\forall$ is PSPACE-complete, and*
3. *NFH$_{\forall\exists}$ is in EXPSPACE.*

We turn to study the membership problem for NFH: given an NFH $\mathcal{A}$ and a hyperword S, is $S \in \mathfrak{L}(\mathcal{A})$? When S is finite, so is the set of assignments from X to S, and so the problem is decidable. We call this case the *finite membership problem*.

Theorem 11.
1. *The finite membership problem for NFH is in PSPACE.*
2. *The finite membership problem for a hyperword of size k and an NFH with $O(\log(k))$ $\forall$ quantifiers is NP-complete.*

Proof. Let S be a finite hyperword, and let $\mathcal{A}$ be an NFH with k variables. We can decide the membership of S in $\mathfrak{L}(\mathcal{A})$ by iterating over all relevant assignments from X to S, and for every such assignment v, checking on-the-fly whether $\boldsymbol{w}_v$ is accepted by $\hat{\mathcal{A}}$. This algorithm uses space of size that is polynomial in k and logarithmic in $|\mathcal{A}|$.

In the case that the number of $\forall$ quantifiers is $O(\log k)$, an NP upper bound is met by iterating over all assignments to the variables under $\forall$, while guessing assignments to the variables under $\exists$. For every such assignment v, checking whether $\boldsymbol{w}_v \in \mathcal{L}(\hat{\mathcal{A}})$ can be done on-the-fly.

We show NP-hardness for this case by a reduction from the Hamiltonian cycle problem. Given a graph $G = \langle V, E\rangle$ where $V = \{v_1, v_2, \ldots, v_n\}$ and $|E| = m$, we construct an NFH$_\exists$ $\mathcal{A}$ over $\{0,1\}$ with n states, n variables, δ of size m, and a hyperword S of size n, as follows. $S = \{w_1, \ldots, w_n\}$, where w_i is the word over $\{0,1\}$ in which all letters are 0 except for the i'th. The structure of $\hat{\mathcal{A}}$ is identical to that of G, and we set $Q_0 = F = \{v_1\}$. For the transition relation, for every $(v_i, v_j) \in E$, we have $(v_i, \varphi_i, v_j) \in \delta$, where φ_i assigns 0 to all variables except for x_i. Intuitively, the i'th letter in an accepting run of $\hat{\mathcal{A}}$ marks traversing v_i. Assigning w_j to x_i means that the j'th step of the run traverses v_i. Since the words in w make sure that every $v \in V$ is traversed exactly once, and that the run on them is of length n, we have that $\mathcal{A}$ accepts S iff there exists some ordering of the words in S that matches a Hamiltonian cycle in G.

Remark To account for all the assignments to the $\forall$ variables, δ – and therefore, $\hat{\mathcal{A}}$ – must be of size at least $2^{k'}$ (otherwise, we can return "no"). We then have that if $k = O(k')$, then space of size k is logarithmic in $|\hat{\mathcal{A}}|$, and so the problem in this case can be solved within logarithmic space. A matching NL lower bound follows from the membership problem for NFA. □

When S is infinite, it may still be finitely represented, allowing for algorithmic membership testing. We now address the problem of deciding whether a regular language $\mathcal{L}$ (given as an NFA) is accepted by an NFH. We call this *the regular membership problem for NFH*. We show that this problem is decidable for the entire class of NFH.

Theorem 12. *The regular membership problem for NFH is decidable.*

Proof. Let $A = \langle \Sigma, P, P_0, \rho, F \rangle$ be an NFA, and let $\mathcal{A} = \langle \Sigma, \{x_1, \ldots, x_k\}, Q, Q_0, \delta, \mathcal{F}, \alpha \rangle$ be an NFH.

First, we construct an NFA $A' = \langle \Sigma \cup \{\#\}, P', P'_0, \rho', F' \rangle$ by extending the alphabet of A to $\Sigma \cup \{\#\}$, adding a new and accepting state p_f to P with a self-loop labeled by $\#$, and transitions labeled by $\#$ from every $q \in F$ to p_f. The language of A' is then $\mathcal{L}(A) \cdot \#^*$. We describe a recursive procedure (iterating over α) for deciding whether $\mathcal{L}(A) \in \mathfrak{L}(\mathcal{A})$.

For the case that $k = 1$, if $\alpha = \exists x_1$, then $\mathcal{L}(A) \in \mathfrak{L}(\mathcal{A})$ iff $\mathcal{L}(A) \cap \mathcal{L}(\hat{\mathcal{A}}) \neq \emptyset$. Otherwise, if $\alpha = \forall x_1$, then $\mathcal{L}(A) \in \mathfrak{L}(\mathcal{A})$ iff $\mathcal{L}(A) \notin \mathfrak{L}(\overline{\mathcal{A}})$, where $\overline{\mathcal{A}}$ is the NFH for $\overline{\mathfrak{L}(\mathcal{A})}$. The quantification condition for $\overline{\mathcal{A}}$ is $\exists x_1$, conforming to the base case.

For $k > 1$, we construct a sequence of NFA $A_k, A_{k-1} \ldots, A_1$ as follows. Initially, $A_k = \hat{\mathcal{A}}$. Let $A_i = \langle \Sigma_i, Q_i, Q_i^0, \delta_i, \mathcal{F}_i \rangle$. If $\mathbb{Q}_i = \exists$, then we construct A_{i-1} as follows. The set of states of A_{i-1} is $Q_i \times P$, and the set of initial states is $Q_i^0 \times P_0$. The set of accepting states is $\mathcal{F}_i \times F$. For every $(q \xrightarrow{f} q') \in \delta_i$ and every $(p \xrightarrow{f(x_i)} p') \in \rho$, we have $((q,p) \xrightarrow{f \setminus \{\sigma_{i_{x_i}}\}} (q',p')) \in \delta_{i-1}$. We denote this construction by $A \cap_{x_i} A_i$. Then, A_{i-1} accepts a word assignment $\boldsymbol{w}_v$ iff there exists a word $u \in \mathcal{L}(A)$, such that A_i accepts $\boldsymbol{w}_{v \cup \{x_i \mapsto u\}}$.

If $\mathbb{Q}_i = \forall$, then we set $A_{i-1} = \overline{A \cap_{x_i} \overline{A_i}}$ Notice that A_{i-1} accepts a word assignment $\boldsymbol{w}_v$ iff for every $u \in \mathcal{L}(A)$, it holds that A_i accepts $\boldsymbol{w}_{v \cup \{x_i \mapsto u\}}$.

For $i \in [1,k]$, let $\mathcal{A}_i$ be the NFH whose quantification condition is $\alpha_i = \mathbb{Q}_1 x_1 \cdots \mathbb{Q}_i x_i$, and whose underlying NFA is A_i. Then, according to the construction of A_{i-1}, we have that $\mathcal{L}(A) \in \mathfrak{L}(\mathcal{A}_i)$ iff $\mathcal{L}(A) \in \mathfrak{L}(\mathcal{A}_{i-1})$.

The NFH $\mathcal{A}_1$ has a single variable, and we can now apply the base case.

Every $\forall$ quantifier requires complementation, which is exponential in $|Q|$. Therefore, in the worst case, the complexity of this algorithm is $O(2^{2^{\cdots^{|Q||A|}}})$, where the tower is of height k. If the number of $\forall$ quantifiers is fixed, then the complexity is $O(|Q||A|^k)$. □

The *containment problem* is to decide, given NFH $\mathcal{A}_1$ and $\mathcal{A}_2$, whether $\mathfrak{L}(\mathcal{A}_1) \subseteq \mathfrak{L}(\mathcal{A}_2)$. Since we can reduce the nonemptiness problem to the containment problem, we have the following as a result of Theorem 4.

Theorem 13. *The containment problem for NFH is undecidable.*

However, the containment problem is decidable for various fragments of NFH.

Theorem 14. *The containment problem of $NFH_\exists \subseteq NFH_\forall$ and $NFH_\forall \subseteq NFH_\exists$ is PSPACE-complete. The containment problem of $NFH_{\exists\forall} \subseteq NFH_{\forall\exists}$ is in EXPSPACE*

Proof. A lower bound for all cases follows from the PSPACE-hardness of the containment problem for NFA. For the upper bound, for two NFH $\mathcal{A}_1$ and $\mathcal{A}_2$, we have that $\mathfrak{L}(\mathcal{A}_1) \subseteq \mathfrak{L}(\mathcal{A}_2)$ iff $\mathfrak{L}(\mathcal{A}_1) \cap \overline{\mathfrak{L}(\mathcal{A}_2)} = \emptyset$. We can compute an NFH $\mathcal{A} = \mathcal{A}_1 \cap \overline{\mathcal{A}_2}$ (Theorems 1, 2), and check its nonemptiness. Complementing $\mathcal{A}_2$ is exponential in its number of states, and the intersection construction is polynomial.

If $\mathcal{A}_1 \in \text{NFH}_\exists$ and $\mathcal{A}_2 \in \text{NFH}_\forall$ or vice versa, then $\mathcal{A}$ is an $\text{NFH}_\exists$ or $\text{NFH}_\forall$, respectively, whose nonemptiness can be decided in space that is logarithmic in $|\mathcal{A}|$.

The quantification condition of an NFH for the intersection may be any interleaving of the quantification conditions of the two intersected NFH (Theorem 2). Therefore, for the rest of the fragments, we can construct the intersection such that $\mathcal{A}$ is an $\text{NFH}_{\exists\forall}$. The exponential blow-up in complementing $\mathcal{A}_2$, along with the PSPACE upper bound of Theorem 6 gives an EXPSPACE upper bound for the rest of the cases. □

8. Related Work

It is well-known that classic specification languages like regular expressions and LTL cannot express hyperproperties. The study of specific hyperproperties, such as noninterference, dates back to the seminal work by Goguen and Meseguer [33] in the 1980s. The first systematic study of hyperproperties is due to Clarkson and Schneider [16]. Subsequently, temporal logics HyperLTL and HyperCTL* were introduced [15] to give formal syntax and semantics to hyperproperties. HyperLTL was recently extended to A-HLTL [6] to capture *asynchronous* hyperproperties, where some execution traces can stutter while others advance.

There has been much recent progress in automatically verifying [31, 30, 29, 17, 37, 36, 21, 42] and monitoring [3, 28, 13, 11, 27, 43, 35] HyperLTL specifications. HyperLTL is also supported by a growing set of tools, including the model checkers HyperQube [37], MCHyper [31, 17], the satisfiability checkers EAHyper [26] and MGHyper [24], and the runtime monitoring tool RVHyper [27].

Related to the nonemptiness problem in this paper is the *satisfiability* problem for HyperLTL, which was shown to be decidable for the $\exists^*\forall^*$ fragment, and undecidable for any fragment that includes a $\forall\exists$ quantifier alternation [23]. The hierarchy of hyperlogics beyond HyperLTL has been studied in [18]. Furthermore, our other results are aligned with the complexity of HyperLTL model checking for tree-shaped and general Kripke structures [8], which encode finite traces. In particular, our membership results are in line with the results on the complexity of verification in [8]. This shows that the complexity results in [8] mainly stem from the nature of quantification over finite words and depend on neither the full power of the temporal operators nor the infinite nature of HyperLTL semantics.

The *synthesis* problem has shown to be undecidable in general, and decidable for the $\exists^*$ and $\exists^*\forall$ fragments. While the synthesis problem becomes, in general, undecidable as soon as there are two universal quantifiers, there is a special class of universal specifications, called the linear $\forall^*$-fragment, which is still decidable [25]. The linear $\forall^*$-fragment corresponds to the decidable *distributed synthesis* problems. The *bounded synthesis* problem considers only systems up to a given bound on the number of states. Bounded synthesis from hyperproperties is studied in [25], and has been successfully

applied to small examples such as the dining cryptographers [14]. Program repair and controller synthesis for HyperLTL have been studied in [9, 10]. Our results on bounded nonemptiness complement the known results, as it resembles the complexity of bounded synthesis.

Other efforts in the area of hyperproperties include the *asynchrouns* HyperLTL [6] and other variations proposed in [12]. These temporal logics capture sets of executions that progress with different speeds. Interfaces for information flow has been introduced in [5]. There has also been emerging interest in hyperproperty-preserving secure compilation [39].

In [22], the authors construct a finite-word representation for the class of regular k-safety hyperproperties. There, a finite automaton reads k-tuple words, that are the so-called "bad prefixes" of a k-safety hyperproperty. Then, the regular representation is used in an L^*-based learning algorithm for such hyperproperties.

The notion of running on a tuple of words simulatenously also appears in a different context in automatic structures [7]. There, regular languages map relations in a logical structures to its universe, allowing for algorithmic evaluations and decidability properties for infinite structures.

9. Discussion and Future Work

We have introduced and studied *hyperlanguages* and a framework for their modeling, focusing on the basic class of regular hyperlanguages, modeled by HRE and NFH. We have shown that regular hyperlanguages are closed under set operations and are capable of expressing important hyperproperties for information-flow security policies over finite traces. We have also investigated fundamental decision procedures for various fragments of NFH, concentrating mostly on the important decision problem of nonemptiness. Some gaps, such as the precise lower bound for the universality and containment problems for $\text{NFH}_{\exists\forall}$, are left open.

Since our framework does not limit the type of underlying model, it can be lifted to handle hyperwords consisting of infinite words, with an underlying model designed for such languages, such as *nondeterministic Büchi automata*, which model ω-regular languages. Just as Büchi automata can express LTL, such a model can express the entire logic of HyperLTL [15].

There are several interesting problems regarding finite hyperlanguages, which we plan to study in the future. One such problem is the realizability, or synthesis, problem: given a hyperlanguage $\mathfrak{L}$, can we construct an NFH for $\mathfrak{L}$? We can ask this with or without a given quantification condition. A closely related problem is finding a canonical form for types of NFH, as the one that exists for regular languages.

In word automata, the *shortest witness problem* is to find the shortest representation for a word accepted by a given automaton [38]. This problem is helpful for synthesis. We can lift this problem to hyperautomata: given a hyperautomaton, what is the minimal (or, alternatively, maximal) hyperword that it accepts?

Since hyperautomata run on sets of words rather than words, the natural word operations of concatenation and Kleene star are not naturally lifted to hyperlanguages. However, it would be interesting to define these operations and study their closure properties for hyperautomata.

As further future work, we plan on studying non-regular hyperlanguages (e.g., context-free), and object hyperlanguages (e.g., trees). Another direction is designing learning algorithms for hyperlanguages, by exploiting known canonical forms for the underlying models, and basing on existing learning algorithms for them. The main challenge would be handling learning sets and a mechanism for learning word variables and quantifiers.

References

[1] Ábrahám, E., Bartocci, E., Bonakdarpour, B., Dobe, O.: Probabilistic hyperproperties with nondeterminism. In: Proceedings of the 18th Symposium on Automated Technology for Verification and Analysis (ATVA). pp. 518–534 (2020)

[2] Ábrahám, E., Bonakdarpour, B.: HyperPCTL: A temporal logic for probabilistic hyperproperties. In: QEST. pp. 20–35 (2018)

[3] Agrawal, S., Bonakdarpour, B.: Runtime verification of k-safety hyperproperties in HyperLTL. In: Proceedings of the IEEE 29th Computer Security Foundations (CSF). pp. 239–252 (2016)

[4] Alpern, B., Schneider, F.: Defining liveness. Information Processing Letters pp. 181–185 (1985)

[5] Bartocci, E., Ferrére, T., Henzinger, T.A., Nickovic, D., da Costa, A.O.: A temporal logic for asynchronous hyperproperties. In: Proceedings of the 25th International Conference on Fundamental Approaches to Software Engineering (FASE) (2022), to appear

[6] Baumeister, J., Coenen, N., Bonakdarpour, B., Sánchez, B.F.C.: A temporal logic for asynchronous hyperproperties. In: Proceedings of the 33rd International Conference on Computer-Aided Verification (CAV). pp. 694–717 (2021)

[7] Blumensath, A., Grädel, E.: Automatic structures. In: IN PROC. 15TH IEEE SYMP. ON LOGIC IN COMPUTER SCIENCE. pp. 51–62. IEEE Computer Society Press (1999)

[8] Bonakdarpour, B., Finkbeiner, B.: The complexity of monitoring hyperproperties. In: CSF. pp. 162–174 (2018)

[9] Bonakdarpour, B., Finkbeiner, B.: Program repair for hyperproperties. In: Proceedings of the 17th Symposium on Automated Technology for Verification and Analysis (ATVA). pp. 423–441 (2019)

[10] Bonakdarpour, B., Finkbeiner, B.: Controller synthesis for hyperproperties. In: Proceedings of the IEEE 32th Computer Security Foundations (CSF). pp. 366–379 (2020)

[11] Bonakdarpour, B., Sánchez, C., Schneider, G.: Monitoring hyperproperties by combining static analysis and runtime verification. In: ISoLA. pp. 8–27 (2018)

[12] Bozzelli, L., Peron, A., Sánchez, C.: Asynchronous extensions of HyperLTL. In: Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). pp. 1–13 (2021)

[13] Brett, N., Siddique, U., Bonakdarpour, B.: Rewriting-based runtime verification for alternation-free HyperLTL. In: Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 77–93 (2017)

[14] Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Communications of the ACM **28**(10), 1030–1044 (1985)

[15] Clarkson, M., Finkbeiner, B., Koleini, M., Micinski, K., Rabe, M., Sánchez, C.: Temporal logics for hyperproperties. In: POST. pp. 265–284 (2014)

[16] Clarkson, M., Schneider, F.: Hyperproperties. Journal of Computer Security pp. 1157–1210 (2010)

[17] Coenen, N., Finkbeiner, B., C. Sánchez, C., Tentrup, L.: Verifying hyperliveness. In: Proceedings of the 31st International Conference on Computer Aided Verification (CAV). pp. 121–139 (2019)

[18] Coenen, N., Finkbeiner, B., Hahn, C., Hofmann, J.: The hierarchy of hyperlogics. In: Proceedings 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). pp. 1–13 (2019)

[19] van Emde Boas, P.: The convenience of tiling. In: Sorbi, A. (ed.) Complexity, Logic and Recursion Theory, Lecture Notes in Pure and Applied Mathematics, vol. 187, pp. 331–363. Marcel Dekker Inc. (Feb 1997)

[20] Emerson, E.A., Halpern, J.: "sometimes" and "not never" revisited: on branching versus linear time temporal logic. Journal of the ACM pp. 151–178 (1986)

[21] Farzan, A., Vandikas, A.: Automated hypersafety verification. In: Proceedings of the 31st International Conference on Computer Aided Verification (CAV). pp. 200–218 (2019)

[22] Finkbeiner, B., Haas, L., Torfah, H.: Canonical representations of k-safety hyperproperties. In: CSF 2019. pp. 17–31 (2019)

[23] Finkbeiner, B., Hahn, C.: Deciding hyperproperties. In: The 27th International Conference on Concurrency Theory (CONCUR). pp. 13:1–13:14 (2016)

[24] Finkbeiner, B., Hahn, C., Hans, T.: MGHyper: Checking satisfiability of HyperLTL formulas beyond the $\exists^*\forall^*$ fragment. In: Proceedings of the 16th International Symposium on Automated Technology for Verification and Analysis (ATVA). pp. 521–527 (2018)

[25] Finkbeiner, B., Hahn, C., Lukert, P., Stenger, M., Tentrup, L.: Synthesizing reactive systems from hyperproperties. In: Proceedings of the 30th International Confer ence on Computer Aided Verification (CAV). pp. 289–306 (2018)

[26] Finkbeiner, B., Hahn, C., Stenger, M.: Eahyper: Satisfiability, implication, and equivalence checking of hyperproperties. In: Proceedings of the 29th International Conference on Computer Aided Verification (CAV). pp. 564–570 (2017)

[27] Finkbeiner, B., Hahn, C., Stenger, M., Tentrup, L.: RVHyper: A runtime verification tool for temporal hyperproperties. In: Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 194–200 (2018)

[28] Finkbeiner, B., Hahn, C., Stenger, M., Tentrup, L.: Monitoring hyperproperties. Formal Methods in System Design (FMSD) **54**(3), 336–363 (2019)

[29] Finkbeiner, B., Hahn, C., Torfah, H.: Model checking quantitative hyperproperties. In: Proceedings of the 30th International Conference on Computer Aided Verification (CAV). pp. 144–163 (2018)

[30] Finkbeiner, B., Müller, C., Seidl, H., Zalinescu, E.: Verifying Security Policies in Multi-agent Workflows with Loops. In: Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS) (2017)

[31] Finkbeiner, B., Rabe, M.N., Sánchez, C.: Algorithms for model checking HyperLTL and HyperCTL*. In: Proceedings of the 27th International Conference on Computer Aided Verification (CAV). pp. 30–48 (2015)

[32] G. Boudol, G., Castellani, I.: Noninterference for concurrent programs and thread. In: TCS 2002. pp. 109–130 (2002)

[33] Goguen, J., Meseguer, J.: Security policies and security models. In: IEEE Symp. on Security and Privacy. pp. 11–20 (1982)

[34] Goudsmid, O., Grumberg, O., Sheinvald, S.: Compositional model checking for multi-properties. In: Henglein, F., Shoham, S., Vizel, Y. (eds.) Verification, Model Checking, and Abstract Interpretation - 22nd International Conference (VMCAI). Lecture Notes in Computer Science, vol. 12597, pp. 55–80. Springer (2021)

[35] Hahn, C., Stenger, M., Tentrup, L.: Constraint-based monitoring of hyperproperties. In: Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 115–131 (2019)

[36] Ho, H., Zhou, R., Jones, T.M.: On verifying timed hyperproperties. In: Proceedings of the 26th International Symposium on Temporal Representation and Reasoning (TIME). pp. 20:1–20:18 (2019)

[37] Hsu, T.H., Sánchez, C., Bonakdarpour, B.: Bounded model checking for hyperproperties. In: Proceedings of the 27th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS). pp. 94–112 (2021)

[38] Kupferman, O., Sheinvald-Faragy, S.: Finding shortest witnesses to the nonemptiness of automata on infinite words. In: Baier, C., Hermanns, H. (eds.) The 17th International Conference on Concurrency Theory (CONCUR). Lecture Notes in Computer Science, vol. 4137, pp. 492–508. Springer (2006)

[39] Patrignani, M., Garg, D.: Secure compilation and hyperproperty preservation. In: Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF). pp. 392–404 (2017)

[40] Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57 (1977)

[41] Sabelfeld, A., Sands, D.: Probabilistic noninterference for multi-threaded programs. In: CSFW. pp. 200–214 (2000)

[42] Shemer, R., Gurfinkel, A., Shoham, S., Vizel, Y.: Property directed self composition. In: Proceedings of the 31st International Conference on Computer Aided Verification (CAV). pp. 161–179 (2019)

[43] Stucki, S., Sánchez, C., Schneider, G., Bonakdarpour, B.: Graybox monitoring of hyperproperties. In: Proceedings of the 23rd International Symposium on Formal Methods (FM). pp. 406–424 (2019)

[44] Vardi, M., Wolper, P.: Automata theoretic techniques for modal logic of programs. Journal of Computer and System Sciences pp. 183–221 (1986)

[45] Vardi, M., Wolper, P.: Reasoning about infinite computations. Information and Computation pp. 1–37 (1994)

[46] Wang, Y., Zarei, M., Bonakdarpour, B., Pajic, M.: Statistical verification of hyperproperties for cyber-physical systems. ACM Transactions on Embedded Computing systems (TECS) pp. 92:1–92:23 (2019)

[47] Zdancewic, S., Myers, A.: Observational determinism for concurrent program security. In: CSFW. p. 29 (2003)