# Decision Problems for Nonemptiness of Hyperlanguages: Undecidability and Complexity Results

HYACINTHE HAMON
*Hamon FZCO Research and Development*

***Abstract*—This paper examines different choice issues connected with the nonemptiness of Non-Deterministic Limited Hyperautomata (NFH). I show that while the nonemptiness issue for NFH is for the most part undecidable, it becomes decidable for explicit pieces. I give a decrease from the Post Correspondence Issue (PCP) to demonstrate the undecidability of the nonemptiness issue for NFH, delineating how encoded arrangements of PCP can address legitimate hyperwords. Furthermore, I lay out that the nonemptiness issue for both NFH with existential and widespread evaluation (NFH∃ and NFH∀) is NL-finished. At last, I present a choice methodology for the nonemptiness issue of NFH with blended evaluation (NFH∃∀), demonstrating that it very well may be settled in polynomial space compared with the machine size. Our outcomes add to a more profound comprehension of hyperlanguage choice issues and their computational intricacy***

## I. INTRODUCTION

The study of automata and formal languages has long served as a cornerstone of theoretical computer science, providing a framework for understanding computation and complexity [1]. Among these, the extension to hyperlanguages, which generalize traditional languages to sets of words, introduces a layer of abstraction that poses unique decision problems [2]. Hyperautomata, operating over hyperwords, extend classical automata theory and challenge existing paradigms with their inherent computational intricacies.

Central to the discourse is the nonemptiness problem for hyperlanguages, which interrogates whether any hyperwords can be accepted under specific conditions [3]. This problem, intertwined with undecidability in some contexts and bounded by complexity classes in others, bridges automata theory with decision problems such as the Post Correspondence Problem. Recent advances in hyper automata have also underscored their relevance in fields like data security, where hyperregular ex- pressions formalize intricate properties such as noninterference and observational determinism [4].

This paper delves into the nonemptiness decision problems for non-deterministic finite hyper automata (NFH), unveiling undecidability results, computational bounds, and polynomial- space solvability under specific configurations. By leveraging reductions and canonical constructions, this work contributes to the broader understanding of hyperlanguage decision- making and its practical applications.

## II. PRELIMINARIES

An *alphabet* is a nonempty restricted set $\Sigma$ of *letters*. A *word* more than $\Sigma$ is a restricted progression of letters from $\Sigma$. The *void word* is meant by $\epsilon$, and the game plan of all restricted words is $\Sigma^*$. A *nondeterministic restricted automaton* (NFA) is a tuple:

$A = \Sigma, Q, Q_0, \delta, F$,

where:

$\Sigma$ is the letters all together, $Q$ is a restricted game plan of *states*, $Q_0 \subseteq Q$ is the game plan of *start states*, $F \subseteq Q$ is the game plan of *enduring states*, $\delta \subseteq Q \times \Sigma \times Q$ is the progress relation.

A word $w = \alpha_1\alpha_2 \cdots \alpha_m$ over an alphabet $\Sigma$ induces a *computation trace* of an automaton $A$, which is a sequence of states $s_0s_1 \cdots s_m$ such that $s_0 \in S_0$ (the set of initial states) and $(s_{i-1}, \alpha_i, s_i) \in \delta$ for $1 \leq i \leq m$. A trace is *accepting* if the final state $s_m \in F$, the set of accepting states. The automaton $A$ *accepts* $w$ if there exists an accepting trace corresponding to $w$. The *language* $\mathrm{L}(A)$ of the automaton $A$ is the set of all words $w$ for which there exists an accepting

trace.

An NFA is *deterministic* (DFA) if, for each state $s \in S$ and symbol $\alpha \in \Sigma$, there exists exactly one state $s'$ $\in S$ such that $(s, \alpha, s') \in \delta$. Every NFA can be deterministically converted into an equivalent DFA. Given a word $w = \alpha_1\alpha_2 \cdot \cdot \cdot \alpha_m$, the automaton's transition function can be represented as:

$\delta : S \times \Sigma \longrightarrow S$

where for each $s \in S$ and $\alpha \in \Sigma$, there is a unique state $s'$ such that $(s, \alpha, s') \in \delta$. The language of a DFA is then defined as:

$L(A) = \{w \mid \exists$ an accepting trace for $w\}$

For an NFA, multiple transitions may exist for a given state and symbol, which is the key distinction from DFAs. Nonetheless, any NFA can be converted into a DFA that recognizes the same language, though potentially at the cost of an exponential blow-up in the number of states.

## III. HYPERAUTOMATA

A *hyperword more than* Σ is a lot of words, and a *hyper- language* is a lot of hyperwords.

### A. The General Approach

A hyperautomaton A operates on hyperwords $W$ using *word variables* $Y = \{y_1, y_2, \ldots, y_p\}$. Each variable $y_i \in Y$ is as- signed a component from $W$, forming a mapping $v : Y \rightarrow W$, denoted as a $p$-tuple $(v(y_1), v(y_2), \ldots, v(y_p))$. The order of the tuple is fixed.

The tuple $(v(y_1), v(y_2), \ldots, v(y_p))$ is represented as a word $u$, where each letter in $u$ corresponds to a tuple of corresponding letters from $v(y_1), v(y_2), \ldots, v(y_p)$). Words of different lengths are padded with the symbol #. For instance, if $v(y_1) = xy$ and $v(y_2) = xz$, the tuple $(xy, xz)$ is encoded as $(x, x)(y, z)$, where the second tuple is padded if necessary. This encoding process, known as *zipping*, is done without directly referencing the individual components, while maintaining the structure intact.

Given a zipped word w, the $i$-th component of the tuple can be extracted by combining all the $i$-th elements from the tuples in w. This reversibility enables an *unzip* function, which reconstructs the original mapping by separating the components back into their respective positions.

Formally, for a zipped word $w = (w_1, w_2, \ldots, w_l)$, the unzip function unzip(w) generates the original tuple:

$$\text{unzip}(\mathbf{w}) = \left( w_{i_1}, \; w_{i_2}, \ldots, \; w_{i_p} \right)$$

$i_j=1 \qquad i_j=1 \qquad i_j=1$

where $w_{ij}$ represents the $j$-th element from the $i$-th tuple in w

### B. Hyperautomata on Constrained Words

Let $t = (v_1, v_2, \ldots, v_m)$ be a tuple of constrained words over Σ, and let length($t$) denote the length of the longest word in $t$. The function combine($t$) produces a word over $(\Sigma \cup \{\#\})^m$ of length length($t$), where the $i$-th letter of combine($t$) contains the $i$-th letters of $v_1, v_2, \ldots, v_m$, with shorter words padded with #. For example: combine $(xyz, abc, pq) = (x, a, p)(y, b, q)(z, \#, \#)$. Given t, the word formed by the $i$-th letters of t is denoted as t[$i$]. The separate function reconstructs the original tuple:

separate(t) = (t[1], t[2], . . . , t[$m$]).

A nondeterministic restricted automaton over hyperwords (NRH) is a tuple $B = \Lambda, Y, P, P_0, G, \gamma, B\rfloor\sqcup\dashv$, where:

- $\Lambda, P, P_0, G$
- $Y = \{y_1, \ldots, y_m\}$ is a restricted set of *word variables*,
- $\gamma \subseteq P \times (\Lambda \cup \{\#\})^m \times P$ is the transition relation,
- $B\rfloor\sqcup\dashv = \lambda_1 y_1 . \lambda_2 y_2 . \ldots . \lambda_m y_m$ is an *evaluation condi- tion*, where $\lambda_i \in \{\forall, \exists\}$ for each $1 \leq i \leq m$.

The tuple $(\Lambda \cup \{\#\})^m, P, P_0, \gamma, G$ defines the auxiliary NFA $\hat{B}$ of B, with the alphabet $\hat{\Lambda} = (\Lambda \cup \{\#\})^m$.

Let $T$ be a hyperword, and $v : Y \rightarrow T$ be a variable assignment in B. We define $v[y \rightarrow t]$ to denote the assignment where $y$ maps to $t \in T$. The assignment $v$ is represented by the zipped word $(v) = (v(y_1), \ldots, v(y_m))$.

Consider the NFH $B_1$ over $\Lambda = \{b\}$ with variables $z_1, z_2$. The corresponding NFA $\hat{B}_1$ accepts words

for $z_1$ and $z_2$ if the word for $z_2$ is longer than that for $z_1$. With the evaluation condition $\forall z_1 \exists z_2$, $B_1$ requires that for each word in a hyperword $T$, there exists a longer word. This condition holds for infinite hyperwords $T$, so the hyperlanguage of $B_1$ is the set of all infinite hyperwords over $\Lambda = \{b\}$.

Consider NFH $B_3$ over $\Lambda = \{b, c\}$ with variables $z_1$ and $z_2$. In B^3, if the word for $z_2$ contains a $b$ at any position, the word for $z_1$ must also have a $b$ at the same position. The reverse holds in the other directions: if $z_1$ has a $b$, $z_2$ must have a $b$ as well. With the evaluation condition $\forall z_1 \forall z_2$, any hyperword recognized by $B_3$ ensures that the positions of $b$ in one word are a subset of the positions of $b$ in the other word. Thus, the hyperlanguage of $B_3$ includes hyperwords where the positions of $b$ are aligned.

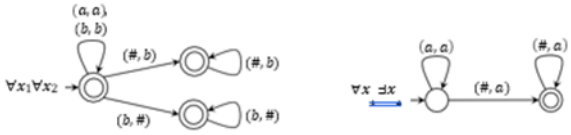


Fig. 1. NFHs $B_1$ (left) and $B_2$ (right)

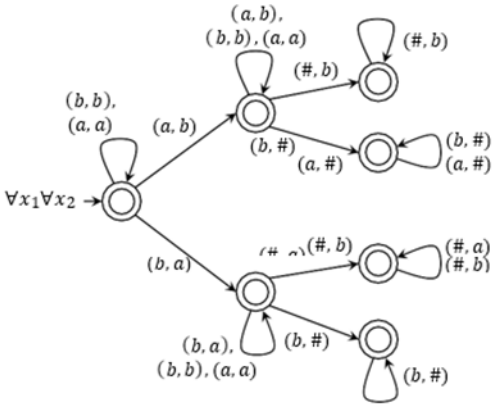


Fig. 2. NFH $B_3$

## IV. HYPER REGULAR EXPRESSIONS AND SECURITY APPLICATIONS

The NFM of a NFH defines a regular expression over hyperwords. This expression, along with the NFH's evaluation, is referred to as a *hyper regular expression* (HRE). For instance, consider NFH $B_1$. Its HRE is:

$$\forall z_1 \forall z_2 \ (b, b) \mid (c, c)^* \ (\#, c)^* \mid (c, \#)^*$$

I now explore the use of Hyper Regular Expressions (HREs) in formalizing security properties of data streams. Noninterference [5] ensures that high-security operations do not influence low-security data:

$$\varphi_{\text{ni}} = \forall z_1 . \exists z_2 . (h, h\lambda)^*$$

where $h$ represents a high-security state and $h\lambda$ signifies a high-security state with additional characteristics for the output.

Observational Determinism [6] ensures that if two executions start in equivalent low-security states, they will remain equivalent:

$$\varphi_{\text{od}} = \forall z_1 . \forall z_2 . (h, h)^+ + (h, h)(\$, \$)^* + (h, h)(\$, \$)^* + (h, h)(\$, \$)^*$$

where $h$ is a high-security state, $\bar{h} \in \Lambda - \{h\}$, and $\$ \in \Lambda$. Similar results to Boudol and Castellani's noninterference [7] can be framed in almost the same way.

Generalized Noninterference (GNI) [8] permits nondeterminism in low-security operations, but ensures that high- security inputs do not affect low-security outputs:

$$\varphi_{\text{gni}} = \forall z_1 . \forall z_2 . \exists z_3 . \ (h, l, hl) + (\bar{h}, l, \bar{h}l) + (h, \bar{l}, h\bar{l}) + (\bar{h}, \bar{l}, \bar{h}\bar{l})^*$$

where $h$ denotes high-security information, $l$ indicates low- security output, $\bar{l} \in \Lambda - \{l\}$, and $\bar{h} \in \Lambda - \{h\}$.

Declassification [9] relaxes certain security policies to allow information leakage when necessary.

$$\varphi_{\text{dc}} = \forall z_1 . \forall z_2 . \ (hi, hi)(pw, pw)(lo, lo)^+$$

where $hi$ represents high-input states, $pw$ denotes a secret key, and $lo$ stands for low-output states.

Termination-Tolerant Noninterference ensures no information leakage from terminal operations in executions that begin from low-security states:

$$\varphi_{\text{tsni}} = \forall z_1 . \forall z_2 . (l, l)(\$, \$) \ (l, l) + (\bar{l}, \bar{l})(\$, \$)^* + (l, \bar{l})(\$, \$)^* + (\bar{l}, l)(\$, \$)^*$$

where $l$ is a low-security state and $\$ \in \Lambda$.

## V. PROPERTIES OF HYPERAUTOMATA AND HYPER REGULAR EXPRESSIONS

This section examines key operations and decision prob- lems for NFH. Let $B = \Lambda, Z, Q, Q_0, \delta, F, D$ where $Z = \{z_1, \ldots, z_m\}$. NFH are closed under

complementation.

*Proof.* Let B be a NFH. The NFA $\hat{B}$ can be extended over its language on $\hat{\Lambda}$ to form $\hat{B}$. For any mapping $w : Y \to T$, $\hat{B}$ recognizes $\lambda(w)$ if and only if $\hat{B}$ does not recognize it.

*Proof.* Given two NFHs $A_1 = \langle \Sigma_1, X_1, Q_1, Q_1^0, \delta_1, F_1, A_1^\alpha \rangle$ and $A_2 = \langle \Sigma_2, X_2, Q_2, Q^0, \delta_2, F_2, A^\alpha \rangle$ we construct a new NFH $A_3 = \langle \Sigma_3, X_3, Q_3, Q_3^0, \delta_3, F_3, A^\alpha \rangle$ that recognizes $L(A_1) \cup L(A_2)$. Define:

- $\Sigma_3 = \Sigma_1 \cup \Sigma_2 \cup \{\#\}$,
- $X_3 = X_1 \cup X_2$,
- $Q_3 = Q_1 \cup Q_2 \cup \{q_{init}\}$,
- $Q_3^0 = q_{init}$,
- $F_3 = F_1 \cup F_2$,
- $A_3^\alpha = A_1^\alpha \cdot A_2^\alpha$.

Let $B_\alpha$ represent the evaluation condition obtained by swapping $\exists$ with $\forall$, and vice versa. By construction, B, a NFH with underlying automaton $\hat{B}$ and evaluation condition $B_\alpha$, recognizes B. The size of B grows exponentially with $n$, based on the complementation of $\hat{B}$. NFH are closed under union.

*Proof.* Given two NFHs $B = \Lambda, Z, Q, Q_0, \delta, F, B_\alpha$ and $B' = \Lambda', Z', Q', Q'_0, \delta', F', B'_\alpha$, I construct a new NFH $B'' = \Lambda'', Z'', Q'', Q''_0, \delta'', F'', B''_\alpha$ that accepts $B \cup B'$. Define:

- $\Lambda'' = \Lambda \cup \Lambda' \cup \{\#\}$,
- $Z'' = Z \cup Z'$,
- $Q'' = Q \cup Q' \cup \{q_{init}\}$, $\forall q \xrightarrow{(\sigma_1,\ldots,\sigma_k)} q' \in \delta$, add $q \xrightarrow{(\sigma_1,\ldots,\sigma_k,\#,\ldots,\#)} q'$,
- $Q''_0 = q_{init}$, $q_{init} \xrightarrow{(\sigma_1,\ldots,\sigma_k)} q' \in \delta'$ in $B'$, add
- $F'' = F \cup F'$,
- $B''_\alpha = B_\alpha \cdot B'_\alpha$.

For transitions;

This guarantees that $B''$ accepts all hyperwords in B or $B'$. NFH are closed under intersection.

*Proof.* The proof utilizes closure under union and complementation. To construct the intersection of two NFHs, consider $B_1 = \Lambda, Z_1, Q_1, Q^0, \delta_1, F_1, B^\alpha$ and $B_2 = \Lambda, Z_2, Q_2, Q^0, \delta_2, F_2, B^\alpha$, and define: $B_\alpha = B^\alpha \cdot B^\alpha$

$\delta$ includes transitions for all pairs, $(q_1,p)_1 \xrightarrow{(\sigma_1,\ldots,\sigma_k,\sigma,\ldots,\sigma)} (q_2,p)_2$ with additional transitions to handle # when one machine halts. This construction ensures that the resulting NFH recognizes the intersection of the languages of A1 and A2. The nonemptiness problem for NFHs is undecidable.

*Proof.* Let A be an NFH. We show that A recognizes a hyperword $(w, w)$ if and only if both $\hat{A}$ and $\hat{A}$ recognize their respective parts. The construction of such an NFH is polynomial in the size of $A_1$ and $A_2$

The class of NFHs is closed under union

For transitions:

- For each $q \xrightarrow{(\sigma_1,\ldots,\sigma_k)} q' \in \delta_1$, add $q \xrightarrow{(\sigma_1,\ldots,\sigma_k,\#,\ldots,\#)} q'$,
- For each $q \xrightarrow{(\sigma_1,\ldots,\sigma_k)} q' \in \delta_2$ in $A_2$, add $q \xrightarrow{(\#,\ldots,\#,\sigma_1,\ldots,\sigma_k)} q'$.

This guarantees that $A_3$ recognizes all hyperwords in $L(A_1)$ or $L(A_2)$ The class of NFHs is closed under intersection

*Proof.* The proof uses closure under union and complementation. To build the intersection of two NFHs, con- sider

$A_1 = \langle \Sigma, X_1, Q_1, Q^0_1, \delta_1, F_1, A^\alpha_1 \rangle$

and $A_2 = \langle \Sigma, X_2, Q_2, Q^0_2, \delta_2, F_2, A^\alpha_2 \rangle$.

Define:

- $A^\alpha_3 = A^\alpha_1 \cdot A^\alpha_2$,
- The transition relation $\delta_3$ includes transitions for each pair $(q_1, q_2) \xrightarrow{(\sigma_1,\ldots,\sigma_k,\sigma,\ldots,\sigma)} (q', q')$, with $_1 \quad _2$ extra transition to handle the # symbol when one machine halts.

This construction ensures that the resulting NFH recognizes the intersection of the languages of $A_1$ and $A_2$.

The nonemptiness problem for NFHs is undecidable.

*Proof.* We reduce from the Post Correspondence Problem (PCP), which is undecidable. Given a set of dominos $\frac{w}{v}$ where $w_i, v_i \in A^*_{pcp}$, the PCP asks whether there exists a set of dominos such that the upper and lower strings match. This problem can be encoded into an NFH, showing that the nonemptiness problem for NFHs is undecidable.

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

$A_{pcp} = \{\frac{\sigma}{\sigma'} \mid \sigma, \sigma' \in \{a, b, \dot{a}, \dot{b}, \$\}\}$.

For a word $w = \frac{\sigma_1}{\sigma'_1}\frac{\sigma_2}{\sigma'_2}\cdots\frac{\sigma_n}{\sigma'_n}$, the upper part is $\sigma_1 \ldots \sigma_n$ and the lower part is $\sigma'_1 \cdots \sigma'_n$. Dotted letters $\sigma$ mark

the start of a new tile, and \$ marks the end of a sequence. A solution $w_{sol}$ is valid if:

1) Each $\overset{\sigma}{\sigma}$ in $w_{sol}$ has matching domino letters ($a$ or $b$),
2) The number of dotted letters in the upper and lower parts is the same,
3) $w_{sol}$ starts with two dotted letters, and for each pair of dotted letters, the subwords $u_i$ and $v_i$ between the $i$-th and $(i + 1)$-th dotted letters satisfy $\frac{u_i}{v_i} \in C$.

We define a partial solution $w_{sol,k}$ by removing the first $k$ tiles from $w_{sol}$, with \$ used to pad the shorter part.

Next, we construct an NFH A that recognizes a hyperword $S$, which consists of $w_{sol}$ and all its suffixes formed by removing prefixes of tiles. For each $w_{sol,k}$ in $S$, $w_{sol,k+1}$ must also be in $S$. For example, given the tiles:

$$[\frac{ab}{b}], [\frac{ba}{a}], [\frac{a}{aba}]$$

a solution is

$$[\frac{a}{aba}][\frac{ba}{a}][\frac{ab}{b}],$$

and a matching hyperword $S$ recognized by A is:

$$S = \{w_{sol} = \overset{\cdot\cdot\cdot\cdot\cdot}{a\,b\,a\,a\,b}\ \overset{\cdot\cdot\cdot\cdot}{b\,a\,a\,b}\ \underset{\cdot}{a\,b}\ \underset{\cdot}{a\,b\,a}\ \underset{\cdot}{b}, w_{sol,1} = \underset{\cdot}{a\,b}\,\$\$, w_{sol,2} = \underset{\cdot}{b}\,\$, w_{sol,3} = \epsilon\}$$

The NFH A has a state space $A_\alpha = \forall x_1 \exists x_2 \exists x_3$, where $x_1$ corresponds to a partial solution $w_{sol,k}$, $x_2$ relates to $w_{sol,k+1}$, and $x_3$ corresponds to the full solution $w_{sol}$. During execution, A ensures that the upper and lower parts of $w_{sol}$ match and that $v(x_2)$ is derived from $v(x_1)$ by removing the first tile.

Let A be an NFH with an evaluation condition $A_\alpha = \exists x_1,...,\exists x_m \forall x_{m+1},...,\forall x_k$, where $1 \leq m < k$. Then, A is nonempty if and only if it recognizes a hyperword of size $m$.

*Proof.* Let $S \in L(A)$. By the quantifier semantics, there exist words $w_1,...,w_m \in S$, such that for each assignment $v : X \to S$ with $v(x_i) = w_i$, $\hat{A}$ recognizes $(v)$. For each permutation $\zeta = (1,2,...,m,i_1,...,i_{k-m})$, it holds that $\{v(x_1),...,v(x_m)\} \in L(A)$. The argument follows. □

Using Lemma V, we can formulate a decision procedure for the nonemptiness of NFHs. The nonemptiness problem for an NFH A can be decided in space polynomial in the size of $\hat{A}$.

*Proof.* By Lemma V, A is nonempty if and only if there exists a word $w \in L(\hat{A})$ such that $\hat{A}$ recognizes $w_\zeta$ for all permutations $\zeta = (1,2,...,m,i_1,...,i_{k-m})$, where $1 \leq i_j \leq m$. This means A recognizes $\{w[1],w[2],...,w[m]\}$. □

For the membership problem for NFH, given a NFH B and a hyperword $S$, the question is whether $S \in B$. For bounded $S$, the arrangement of operations from $Z$ to $S$ is constrained, making the problem decidable. This is known as the bounded membership problem.

Let B be a NFH and $S$ be a bounded hyperword over Σ. Then, determining whether $S \in B$ is solvable in space polynomial in $n$, and logarithmic in $|\hat{B}|, |S|$.

*Proof.* As in Theorem V, the size of $\hat{\Sigma}$ increases mainly with the number of ∀ quantifiers in B, making the problem complex in $n$. We can conclude membership by iterating over all functions $u : Z \to S$, constructing $(u)$, and evaluating $\hat{B}$ on-the-fly. □

If the number of ∀ quantifiers is fixed, $^{\hat{\Sigma}}$ may not grow with $n$, improving the problem. For NFH with a fixed number of ∀ quantifiers, the bounded membership problem is NP-complete.

*Proof.* We iterate over all assignments to the ∀ variables, guessing assignments for the ∃ variables. Since the number of ∀ variables is fixed, the number of iterations is polynomial in $n$ and $|S|$. Checking if $(u) \in L(\hat{B})$ is feasible on-the-fly. The problem is in NP because $^{\hat{\Sigma}}$ does not grow significantly.

For NP-hardness, we reduce from the Hamiltonian cycle problem. Given a graph $H = (V,E)$, we construct a NFH B over {0,1} with $n$ states and $n$ variables. The hyperword $S = \{s_1,...,s_n\}$, where each $s_i$ is a word in $\{0,1\}^n$ with only the $i$-th bit set to 1, encodes a Hamiltonian cycle. We show that B accepts $S$ if and only if there is a Hamiltonian cycle in $H$.

For unbounded $S$, we still have a bounded representation. We now address the general membership problem for NFH, which is decidable for all NFHs. The general membership problem for NFH is decidable.

*Proof.* Let $B=\langle \Sigma,Q,q_0,\delta,F \rangle$ be a NFA, and B

=⟨Σ,{$z_1$,...,$z_k$},$Q$,$Q_0$,$\delta$,F,B⌋⊔⊣⟩ be a NFH.

We expand the alphabet in $B$ to Σ ∪ {#}, adding a final state $q_f$ and transitions labeled with #. The language of the modified NFA, $B'$, will be L($B$) · #*. We then recursively determine if L($B$) ∈ B.

For $k$ = 1, if B⌋⊔⊣ = ∃$z_1$, then L($B$) ∈ B if and only if L($B$)∩B⌈ = ∅. If B⌋⊔⊣ = ∀$z_1$, then L($B$) ∈ B if and only if L($B$) ∈/ B, where B is the NFH for B.

For $k$ > 1, we construct a sequence of NFHs $B_1$,...,$B_k$, starting with $B_1$ = B. At each step, we update the quantifiers and check for membership in $B_i$ for ∃ and non-membership for ∀, using the complementary NFH. This recursive process ensures that the general membership problem is decidable.

The time complexity is $O(n|Q|^k)$ when the number of ∀ quantifiers is fixed, with a significant factor depending on the number of quantifiers if unbounded. □

The membership problems for NFHE and NFHF, as well as for NFHE in NFHE and NFHF, are PSPACE-complete.

*Proof.* The lower bound follows from the membership problems of NFAs.

For the upper bound, note that taking the complement of a NFH yields another NFH, and similarly for intersection. Given two NFHs $B_1$ and $B_2$, we check if L($B_1$) ⊆ L($B_2$) by testing if L($B_1$) ∩ L($B_2$) = ∅. Using results from Theorems V and V, we construct a NFH B = $B_1$ ∩ $B_2$ and check its non-emptiness. The complementation is exponential in $B_2$'s states and the intersection is polynomial in the sizes of $B_1$ and $B_2$.

If $B_1$ ∈ NFHE and $B_2$ ∈ NFHF, or vice versa, we construct another NFH whose non-emptiness can be decided in logarithmic space. If both $B_1$ and $B_2$ are either NFHE or NFHF, by Theorem V, the result is also a NFH, whose non-emptiness is decidable in PSPACE. Thus, the membership problem is PSPACE-complete. □

## VI. LEARNING NFH

In this section, we present learning algorithms for the classes NFHE and NFHF based on Angluin's L* algorithm [10].

*1)* The L* Algorithm: The L* algorithm consists of two components: a learner, who aims to learn a DFA B for an unknown target language L, and a teacher, who knows L. The learner asks two types of queries: membership queries ("Is w ∈ L?") and equivalence queries ("Is B a DFA for L?").

The learner maintains a table $T$ of truth values, where rows $D$ and columns $E$ are sets of words over Σ. Initially, $D$ = $E$ = {$\epsilon$}, and for each $d$ ∈ $D$ and $e$ ∈ $E$, the entry $T$($d$,$e$) is if $d \cdot e$ ∈ L. The entries are filled via interaction queries. The learner updates the table until it is both *closed* and *consistent*.

A table is closed if for each $d$ ∈ $D$ and $\sigma$ ∈ Σ, there exists $d'$ ∈ $D$ such that ($d'$) = ($d \cdot \sigma$). The table is consistent if for each $d_1$,$d_2$ ∈ $D$ and $\sigma$ ∈ Σ, if ($d_1$) = ($d_2$), then ($d_1 \cdot \sigma$) = ($d_2 \cdot \sigma$).

When the table is closed and consistent, the learner constructs the DFA. If the learner's hypothesis DFA $B$ is incorrect, the teacher provides a counterexample, which the learner adds to $E$ and continues the interaction.

The correctness of L* guarantees that the learner will converge to an equivalent DFA for the target language L.

A. Canonical Constructions for NFHF and NFHE, We describe canonical constructions for the classes NFHF and NFHE, which are essential for applying L*.

*1) Canonical Construction for* NFHF*:* A NFHF is strategy complete if for every word $w$, B̂ accepts $w$ if and only if it accepts all concatenations of $w$. Let B be a NFHF. Then, B has an equivalent strategy complete NFHF.

*Proof.* Foster a progression complete MFJM by overriding the secret NFA with a set union greater than all groupings of (1,2,...$k$). This yields a trivial deterministic gathering complete design. □

*2) Canonical Construction for* MFJD*:* A MFJD is change complete if for each word $w$, B̂ recognizes $w$ iff it recognizes all phases of $w$. Permit B to be a

MFJD. Then, at that point, B has an indistinguishable change complete MFJD.

*Proof.* For the principal heading, in case B̂ recognizes a word,

it perceives all periods of the word by the semantics of the ∀ quantifier. As such, L(B) = L(B′). article amsmath Each MFJD has an indistinguishable change complete MFJD over comparable plan of variables.

*Proof.* I foster B′ from B by describing $B_{\zeta}$ for each change $\zeta$ of (1,2,...,*k*). The MFJD B′ is gained by displacing the major NFA with $^{S}{}_{\zeta\in\Gamma} B_{\zeta}$, where Γ is the plan of phases of (1,2,...,*k*). Permit $B_1$ and $B_2$ to be change completed MFJD over comparative game plan of elements *X*. Then, L(()$B_1$) = L(()$B_2$) iff L(()B̂$_1$) = L(()B̂$_2$).

*Proof.* For the essential heading, accept $w \in$ L(()B̂$_1$). Then, at that point, (*w*) ∈ L(()$B_1$). By the semantics of the ∃ quantifier, there exists a change $w'$ of *w* so much that $w' \in$ L(()B̂$_2$). Since $B_2$ is change gotten done, I have $w \in$ L(()B̂$_2$). A relative conflict holds for the opposite, exhibiting that L(()B̂$_1$) = L(()B̂$_2$). □

I portray a standard design for MFJD as an irrelevant deterministic stage complete MFJD with the most un-number of variables. Each (*k*,*t*)-MFJD B has an indistinguishable stage plan total (*k*,*t*)-MFJD B′.

*Proof.* I assemble B′ by first making B″, the combination of all $B_{\zeta}$ for game plans $\zeta$ = (1,...,*t*,$i_{t+1}$,...,*k*). Then, I structure B′ as the relationship of $B_{\xi}''$ for each stage $\xi$ of ($i_1$,...,$i_t$,*t* + 1,...,*k*). □

Permit $B_1$ and $B_2$ to be change progression complete (*k*,*t*)MFJD. Then, L(()$B_1$) = L(()$B_2$) iff L(()B̂$_1$) = L(()B̂$_2$).

*Proof.* Acknowledge L(()$B_1$) = L(()$B_2$). Let $w \in$ L(()B̂$_1$). Since $B_1$ is stage progression complete, I know that for every plan $\zeta$ of the construction (1,...,*t*,$i_{t+1}$,...,*k*), $w_{\zeta} \in$ L(()$B_1$). By the semantics of the quantifiers, (*w*) ∈ L(()$B_2$). Thus, $w \in$ L(()B̂$_2$). The inverse continues similarly. Thusly, L(()B̂$_1$) = L(()B̂$_2$). □

Permit B to be a (*k*,*t*)-MFJD with *k* being the immaterial number of variables expected to impart L(()B). Then, B̂ recognizes a word *w* where *w*[1],...,*w*[*t*] are specific.

*Proof.* Expect, for coherent irregularity, that for each $w \in$ L(()B̀), two of the first *t* words in *w* are same. By the semantics of the ∃ quantifier, this would conflict with the inconsequentiality of the number of variables. □

Permit $B_1$ and $B_2$ to be stage progression complete MFJD for the identical hyperlanguage L, both with a unimportant course of action of variables. Then the estimation provinces of $B_1$ and $B_2$ are same.

*Proof.* Expect $B_1$ is a (*t*,*k*)-MFJD and $B_2$ is a (*t*,*m*)-MFJD, with *t* < *m*. Expecting $B_1$ recognizes some $S \in$ L(()$B_1$), this recommends that both $B_1$ and $B_2$ have a comparative assessment condition, inciting an irregularity with the exception of on the off chance that the estimation conditions are undefined. □

## VII. LEARNING N FHE AND N FHF

In this section, we describe the learning process for NFHE and NFHF under the framework of the L-star structure. These algorithms aim to learn inconsequential deterministic grouping complete (for NFHF) or stage complete (for NFHE) NFH with a minimal number of variables for a target hyperlanguage L.

In the hyperautomaton setting, the teacher provides membership queries and counterexamples consisting of hyperwords. The teacher responds with minimal counterexamples concerning the size of the hyperword.

The learner constructs an NFH A using the growing number of variables *k*, along with a representation table for Â, over the alphabet $\Sigma = (\hat{\Sigma} \cup \{\#\})^k$, where *k* starts at 1. Once the number of variables increases to $k' > k$, the alphabet of Â expands to $(\Sigma \cup \{\#\})^{k'}$.

To increase the alphabet, define the function $\uparrow_k^{k'}$: $(\Sigma \cup \{\#\})^k \to (\Sigma \cup \{\#\})^{k'}$ that maps each word ($\sigma_1$,...,$\sigma_k$) to ($\sigma_1$,...,$\sigma_k$,$\sigma_k$). This function is extended to words by applying it to each symbol in the word.

When $(d \cdot e) \in$ L(()A), then $(\uparrow^{k}{}_{k'} (d \cdot e)) \in$ L(()A). Therefore, when the number of variables increases, each word in the rows and fragments of the representation table is replaced by $\uparrow^{k}{}_{k'}(w)$.

*1) Learning :* For , if the teacher returns a counterexample *S* with |*S*| > *k*, *S* is positive. Assume for contradiction that *S* is negative. For each *k*-tuple of

words $w_1,...,w_k \in S$, $zip(w_1,...,w_k) \in$ L(()A$^\wedge$), yet $S \not\in$. This implies a word $w = (w_1,...,w_k)$ exists, where $w_i \in S$ for $1 \leq i \leq k$, and $w \not\in$ L(()A$^\wedge$), contradicting the negligibility of $S$.

When $|S| > k$, if $|S| = k+1$, since A recognizes all subsets of size $k$ of $S$, there exists a subset $S' \subseteq S$ of size $k+1$ that should be accepted but isn't, making $S'$ a counterexample.

When a counterexample $S$ of size $k+1$ is returned, the learner increments $k \leftarrow k+1$, updates $T$ by applying $\uparrow^{k}_{k^{+1}}$ to all $w \in D \cup D \cdot \Sigma^\wedge \cup E$, selects a phase $p$ of the words in $S$, and adds ($S$) and its suffixes to $E$. The alphabet $\Sigma^\wedge$ is updated, and missing entries in $D \cdot \Sigma^\wedge$ are filled.

When $|S| \leq k$, if $S$ is positive, the learner finds a phase $p$ such that A does not recognize ($p$) and adds ($p$) and all of its suffixes to $T$. If $S$ is negative, A recognizes all $k$-length tuples of words in $S$, but $S$ should not be recognized. A phase $p$ of $S$ is found such that A recognizes ($p$), but it does not appear in $T$. The learner adds ($p$) and its suffixes to $T$.

The learner's actions are correct, as a positive counterexample guarantees that ($p$) must be in L(()A$^\wedge$) for every superset $S'$ of $S$. If $S$ is negative, ($p$) should not be in L(()A$^\wedge$). Upon the development of an indistinguishable quality order, A becomes an for . However, A may not be fully constructed, as A$^\wedge$ could recognize $w = (w_1,...,w_k)$ but not its transitions. If this occurs, the learner identifies the missing transition and adds it to $T$.

Variables are introduced only when necessary, ensuring that A is an for with the minimal number of variables. The correctness of and the minimization of the counterexamples ensure that no smaller A$^{\wedge\prime}$ for exists, as restricting A$^{\wedge\prime}$ would lead to a smaller automaton for $k'$ variables, leading to a contradiction.

| $T_0$ | $\epsilon$ |
|---|---|
| $\epsilon$ | 1 |
| ($a$) | 1 |
| ($b$) | 1 |
| (#) | 1 |

| $T_1$ | $\epsilon$ | ($a$, $b$) |
|---|---|---|
| $\epsilon$ | 1 | 1 |
| ($a$, $a$) | 1 | 1 |
| ($b$, $b$) | 1 | 1 |
| ($a$, $b$) | 1 | 1 |
| ($a$, #) | 1 | 0 |
| ($b$, $a$) | 1 | 0 |
| ($b$, #) | 1 | 0 |
| (#, $a$) | 1 | 0 |
| (#, $b$) | 1 | 0 |
| (#, #) | 1 | 0 |

Fig. 3. The initial stages of acquiring L(()A$_3$) from Figure 1.

Figure 3 illustrates the initial stages of learning (A$_3$) from Figure 1. The initial table, $T_0$, contains $D = E = \{\epsilon\}$ and $\Sigma =^\wedge \{a,b,\#\}$. Since $\{a\},\{b\},\{\epsilon\}$ are recognized by (A$_3$), the initial NFH A is created over a single variable with a single accepting state.

The smallest positive counterexample the teacher can return is$\{a, b\}$. In $T_1$, $\uparrow^2_1$ is applied, expanding $\Sigma^\wedge$ to $\{a,b,\#\}^2$, and the table is populated with support queries. For example, for $(b,a) \in D \cdot \Sigma^\wedge$ and $(a,b) \in E$, a query for $\{ba,ab\}$ results in a "no" answer from the teacher.

*2) Learning :* The learning process for closely mirrors that of , with a few differences. As with , minimal counterexamples ensure that when $|S| > k$, $S$ is a positive counterexample. If $S$ were negative, the word $(w_1,...,w_k)$ in L(()A$^\wedge$) would imply $S \in$ (A), conflicting with $S \not\in$. When $S$ of size $k' > k$ is returned, the alphabet $\Sigma^\wedge$ is expanded to $(\Sigma \cup \{\#\})^{k'}$ and the table is updated by $\uparrow^{k}_{k'}$, as in . If $|S| \leq k$, $S$ can be positive or negative. If negative, a phase $w$ of $S$ recognized by A$^\wedge$ is added to $E$, as it is not in $T$. If positive, no phase of $S$ is recognized by A$^\wedge$, and the corresponding transition is also added to $E$.

In both cases, equivalence queries do not guarantee that A is fully constructed. If it isn't, the learner identifies words $w$ and $w'$ where $w \in L(()A^{\hat{}})$ and $w' \in L/(()A^{\hat{}})$, and adds $w'$ to $E$, ensuring it was not previously in $T$. The learning process then continues.

REFERENCES


[1] E. John, R. M. Hopcroft, and J. D. Ullman, "Introduction to automata theory languages, and computation [m]," 2004.

[2] C. Li, "From sum of two squares to arithmetic siegel–weil formulas," *Bulletin of the American Mathematical Society*, vol. 60, no. 3, pp. 327–370, 2023.

[3] N. Zhao, H. Zhang, X. Yang, J. Yan, and F. You, "Emerging information and communication technologies for smart energy systems and renewable transition," *Advances in Applied Energy*, vol. 9, p. 100125, 2023.

[4] F. Kruger, Z. Queen, O. Radelva, and N. Lawrence, "Comparative analysis of scientific approaches in computer science: A quantitative study," *International Transactions on Education Technology (ITEE)*, vol. 2, no. 2, pp. 120–128, 2024.

[5] J. A. Goguen and J. Meseguer, "Security policies and security models," in *IEEE Symp. on Security and Privacy*, 1982, pp. 11–20.

[6] S. Zdancewic and A. C. Myers, "Observational determinism for concurrent program security," in *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW)*, 2003, p. 29.

[7] G. Boudol and I. Castellani, "Noninterference for concurrent programs and thread systems," *Theoretical Computer Science (TCS)*, vol. 281, no. 1-2, pp. 109–130, 2002.

[8] D. McCullough, "Noninterference and the composability of security properties," in *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, 1988, pp. 177–186.

[9] A. Sabelfeld and D. Sands, "Probabilistic noninterference for multithreaded programs," in *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW)*, 2000, pp. 200–214.

[10] Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.