HAL
open science

# Certain Query Answering on Compressed String Patterns: From Streams to Hyperstreams

Iovka Boneva, Joachim Niehren, Momar Sakho

## To cite this version:

Iovka Boneva, Joachim Niehren, Momar Sakho. Certain Query Answering on Compressed String Patterns: From Streams to Hyperstreams. RP 2018 - 12th International Conference on Reachability Problems, Sep 2018, Marseille, France. hal-01609498v4

**HAL Id: hal-01609498**

**https://hal.science/hal-01609498v4**

Submitted on 8 Oct 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Certain Query Answering on Compressed String Patterns: From Streams to Hyperstreams

Iovka Boneva[1], Joachim Niehren[2], and Momar Sakho[2]

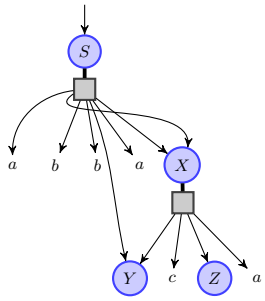
[1] Université de Lille, France
[2] Inria Lille, France

**Abstract.** We study the problem of certain query answering (CQA) on compressed string patterns. These are incomplete singleton context-free grammars, that can model systems of multiple streams with references to others, called hyperstreams more recently. In order to capture regular path queries on strings, we consider nondeterministic finite automata (NFAs) for query definition. It turns out that CQA for Boolean NFA queries is equivalent to regular string pattern inclusion, i.e., whether all strings completing a compressed string pattern belong to a regular language. We prove that CQA on compressed string patterns is PSPACE-complete for NFA queries. The PSPACE-hardness even applies to Boolean queries defined by deterministic finite automata (DFAs) and without compression. We also show that CQA on compressed linear string patterns can be solved in PTIME for DFA queries.

## 1 Introduction

A stream is a sequence of events that arrive incrementally one by one from the left to the right. Most typically, streams are produced by social networks such as Twitter, database systems as for producing financial transactions, information systems, sensor systems, or more generally when communicating semi-structured data over the internet. We are interested in the problem of monitoring streams in a reactive manner [22,16,25,23]. The objective is to select the relevant events of a stream as quickly as possible upon their arrival. This requires to decide whether an event of the stream is a certain answer of the logical query that defines the relevant events of the monitoring task. Lowering the latency of this decision process increases the reactivity of the stream processing system and reduces its memory costs. A limitation to constant memory may seem ideal in theory, but is too restrictive for many monitoring tasks in practice. A less restrictive objective is thus to minimize the latency and thereby to reduce the memory consumption.

In the present paper we study a generalization of streams to multiple streams with references as introduced by Maneth, Pereira and Seidl [21]. The references point to unknown parts in the middle of a stream. The same reference may be used multiple times, allowing to share unknown parts. Streams with similar references were named hyperstreams in own previous work [20]. Here, we propose to formalize hyperstream containing words (rather than linearizations




$$
\begin{aligned}
G_1 &= (\Sigma, N, R, S) \\
&\quad \text{where} \\
&\quad \Sigma = \{a, b, c\} \\
&\quad N = \{S, X, Y, Z\} \\
&\quad R = \{S \to aXbbYaX, \\
&\qquad\quad X \to YcZa \quad \} \\
p_1 &= a\underline{YcZa}bbYa\underline{YcZa}
\end{aligned}
$$

Fig. 1: The hyperstream $G_1$ and its string pattern $p_1$.

$$
\begin{array}{ccccccc}
\text{streams} & \subseteq & \text{linear string patterns} & \subseteq & \text{compressed linear string patterns} & & \\
 & & \cap\!| & & \cap\!| & & \\
 & & \text{string patterns} & \subseteq & \text{compressed string patterns} & = & \text{hyperstreams}
\end{array}
$$

Fig. 2: Landscape from streams to hyperstreams.

of trees or nested words) as *incomplete* versions of singleton context-free grammars [24] (also termed straight line programs [3]), where the rules of some nonterminals may be missing. The hyperstream $G_1 = (\Sigma, N, R, S)$ is illustrated graphically in Fig. 1. It has the terminals in $\Sigma = \{a, b, c\}$, the nonterminals in $N = \{S, X, Y, Z\}$, the set $R$ with the rules $S \to aXbbYaX$ and $X \to YcZa$, and the start symbol $S$. The nonterminals are called the references of the hyperstream. For some of these references there exists a rule in the grammar, and if so, this rule is unique. For any grammar rule, the reference on its left is said to refer to the string pattern on its right. The hyperstream $G_1$ has a rule for $S$ and $X$, while it misses those of $Y$ and $Z$. The missing rules for these references may be added in the future one by one by the hyperstream's environment. Alternatively, hyperstreams can be identified with *compressed string patterns*. The hyperstream $G_1$ for instance represents the string pattern $p_1 = a\underline{YcZa}bbYa\underline{YcZa}$, while sharing the underlined factors substituted for the two occurrences of $X$. Streams are a special case of string patterns that have a unique occurrence of a variable in their last position. The landscape from streams to hyperstreams, over linear string patterns, string patterns, and compressed string patterns is illustrated in Fig. 2.

In this paper, we study the decision problem of certain query answering (CQA) on compressed string patterns, i.e., whether a tuple of positions is a certain answer of a query on a compressed string pattern. Here we consider the positions of the string pattern after decompression rather than the positions of the grammar. Intuitively, a tuple of positions is a certain query answer on a compressed string pattern $G$ if it is an answer to the query on all completions of $G$, up to the offsets raised by the completion of $G$ on its decompression. We will also consider the symmetric problem for certain query non-answers.

Motivated by regular path queries [1], we consider nondeterministic finite automata (NFAs) for defining such queries. For instance, the query $\mathbf{Q}_1$ on strings over $\Sigma = \{a, b, c\}$ that selects all $a$-positions that are followed eventually by $bb$ can be defined by the following regular path in XPATH-like notation:

$$\mathbf{Q}_1 = \texttt{successor}^*\texttt{::}a[\texttt{successor}^*\texttt{::}b/\texttt{successor::}b].$$

It can also be defined by the $x$-pointed regular expression $\Sigma^* a^x \Sigma^* bb \Sigma^*$ where $x$ is a variable for the position that is to be selected. Now, consider the case, where the string is not given explicitly but only described partially by some (compressed) string pattern. On the string pattern $p_1$, for instance, the $a$-positions 1 and 5 are certain query answers for $\mathbf{Q}_1$, while the $a$-positions 9 and 13 are not. The position 13 is even a certain non-answer.

When restricted to Boolean NFA queries, CQA becomes equivalent to the problem of whether all strings described by the completions of a compressed string pattern are accepted by the NFA. For the string pattern $Y$ (for some variable $Y$), this problem clearly generalizes on the universality problem of NFAs, which is well-known to be PSPACE-complete [17]. The following questions, however, are open to the best of our knowledge, even in the case of string patterns without compression: Is CQA on (compressed) string patterns decidable for NFA-defined queries, and if yes, what is the complexity? Does CQA on (compressed) string patterns remain hard for queries defined by deterministic finite automata (DFAs)? For which restrictions of (compressed) string patterns is CQA in PTIME? And what about the symmetric questions concerning certain query non-answers? The objective of the present paper is to answer these questions in all possible cases.

Our first contribution is that CQA on string patterns is PSPACE-complete, both for NFA queries and DFA queries, with and without compression, Boolean or not, see Fig. 3. This upper bound is not fully obvious, as the set of strings defined by a string pattern may be non-regular and even non-context-free. Furthermore, the lower bound may be surprising in that CQA for DFA queries on string patterns is more complex than on streams, where it is in PTIME (Theorem 1 of [13]), and also more complex than string pattern matching, which is NP-complete (Theorem 3.6 of [2]) even with compression (Theorem 4.10 of [11]).

Our second contribution is that CQA for DFA queries can be decided in PTIME on compressed *linear* string patterns, see Fig. 4. The linearity restriction matches with the worst case complexity for streams, even though linear compressed string patterns allow for unknown factors and compression in addition. This result (Corollary 2) is based on a novel algorithm for partial decompression of compressed string patterns that we present (Lemma 6), followed by a test of a reachability property (Theorem 3).

Our third contribution is that the certainty of query non-answers on compressed string patterns is PSPACE-complete, both for Boolean and non-Boolean queries, and independently of whether they are defined by DFAs or NFAs. In the Boolean case, the problem is equivalent to whether a compressed string pattern does *not* match the regular language accepted by the automaton. This problem

| | DFAs | NFAs |
|---|---|---|
| Answers | PSPACE-c | PSPACE-c |
| Non-answers | PSPACE-c | PSPACE-c |

Fig. 3: Query certainty on (compressed) string patterns or hyperstreams.

| | DFAs | NFAs |
|---|---|---|
| Answers | PTIME | PSPACE-c |
| Non-answers | PTIME | PTIME |

Fig. 4: Query certainty on (compressed) linear string patterns or streams.

generalizes on the complement of compressed string pattern matching, and thus is coNP-hard. So while certain query non-answering can be solved in PTIME on streams, the complexity increases to PSPACE on compressed string patterns. Finally, we show that the restriction of the problem to compressed linear string patterns – that is, *regular* compressed linear string pattern matching – can also be solved in PTIME even for queries defined by NFAs.

**Outline**. In Section 2 we start with some preliminaries on finite automata theory. Section 3 recalls the notion of compressed string patterns and in Section 4 we study the problems of regular compressed pattern inclusion and matching. Section 5 recalls how to define non-Boolean queries on strings by automata. In Sections 6 and 7 we generalize the notions of certain query answers and non-answers to (compressed) string patterns and study their complexity. All omitted proofs can be found in the long online version[3].

**Related Work**. The notion of certain query answers for incomplete relational structures is standard in databases research [9]. In the context of stream processing, certain query answers were called answers that are safe for selection and certain query non-answers were called safe for rejection [12]. Certain query non-answers were studied for fast failure [4] and for reducing the memory consumption of streaming systems. The problem of certain query answering and non-answering on streams has been shown to be computationally hard even for queries defined in tiny fragments of first-order logic [12]. Certain query non-answering was shown to be hard in the context of online verification [4,19].

As shown by [12], those classes of queries on strings for which the problem of certain query answering on streams is known to be feasible, are either such that certainty is always determined with 0-delay [22,4,14]) or such that the queries in the class can be compiled to DFAs in PTIME.

Algorithms for processing XML streams or complex event streams raised much interest in the literature [16,25,15] and motivated the work on hyperstreams. XML streams contain nested words [22,18] rather than strings without bracket structure. The best existing algorithms for answering navigational XPATH queries (i.e. first-order logic queries) on XML streams are based on compilation to nested word automata [10,23]. Low but not lowest latency is achieved with high efficiency by approximating certain answers for queries defined by nondeterministic nested word automata.

[3] The long version can be found at the address https://hal.inria.fr/hal-01846016

## 2 Preliminaries

**Words**. The set of natural numbers with 0 is denoted by $\mathbb{N}$. For any set $\Sigma$, a word over $\Sigma$ is a tuple $(a_1, \ldots, a_n) \in \Sigma^n$ where $n \in \mathbb{N}$. We denote such words by $a_1 \ldots a_n$ and by $\varepsilon$ if $n = 0$. We denote the $i^{\text{th}}$ letter of a word $u = a_1 ... a_n$ by $u[i] =_{def} a_i$. The set of all words over $\Sigma$ is denoted by $\Sigma^*$. The concatenation of two words $u_1, u_2 \in \Sigma^*$ is denoted by $u_1 \cdot u_2 \in \Sigma^*$. For instance, if $\Sigma = \{a, b\}$ then $aba \cdot a = abaa$. The set of positions of a word $u = a_1 \ldots a_n$ is $pos(u) = \{1, \ldots, n\}$. For any subset $\Sigma' \subseteq \Sigma$ the set $pos_{\Sigma'}(u)$ is the subset of positions $i$ of $u$ such that $a_i \in \Sigma'$. Given a word $w = a_1 \ldots a_n$ and a second word $u = b_1 \ldots b_n$ of the same length possibly on a different alphabet, we define the zipped word over the product alphabet by $w{*}u = (a_1, b_1) \ldots (a_n, b_n)$. As a convention throughout the paper, we use the term *string* for words over the default alphabet $\Sigma$, as opposed to words over other alphabets such as $\Sigma \cup \mathcal{Y}$ for string patterns, and $\Sigma^{\mathcal{V}}$ for $\mathcal{V}$-annotated strings, that will be introduced later on.

**Monoids**. A monoid is a triple $(M, \cdot^M, 1^M)$ where $M$ is a set, $\cdot^M : M \times M \to M$ is an associative binary operation and $1^M \in M$ is the neutral element, i.e. $1^M \cdot^M m = m \cdot^M 1^M = m$ for all $m \in M$. Given a word $u = m_1 \ldots m_n \in M^*$ we define its evaluation by $u^M = m_1 \cdot^M \ldots \cdot^M m_n$ where $\varepsilon^M = 1^M$.

Most typically, we will consider the monoid of words $(\Sigma^*, \cdot, \varepsilon)$ on some alphabet $\Sigma$, with the concatenation operation $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$, and the empty word $\varepsilon$ as neutral element. Alternatively, given another set $Q$, we will consider the *transition monoid* $(T_Q, \circ, id)$, where $T_Q = 2^{Q \times Q}$ is the set of binary relations over $Q$, $\circ : T_Q \times T_Q \to T_Q$ is the composition operation of binary relations on $Q$, and $id = \{(q, q) \mid q \in Q\}$ is the identity transition.

**Finite Automata**. A nondeterministic finite-state automaton (NFA) is a tuple $A = (Q, \Sigma, \delta, I, F)$ where $Q$ and $\Sigma$ are finite sets, $\delta \subseteq Q \times \Sigma \times Q$, and $I, F \subseteq Q$. We call $Q$ the state set, $\Sigma$ the alphabet, $\delta$ the transition relation, $I$ the set of initial states, and $F$ the set of final states of the automaton. An NFA $A$ is deterministic, or a DFA, if it has exactly one initial state and the transition relation $\delta$ forms a partial function from $Q \times \Sigma$ to $Q$. The elements of $T_Q$ are called the *transitions* of $A$. Any transition relation $\delta : Q \times \Sigma \times Q$ can be extended homomorphically to a transition function $\delta : \Sigma^* \to T_Q$ that assigns to any string a transition of $A$. Here we overload the symbol $\delta$ to stand for the transition relation and the transition function. The transition $\delta(a)$ of a letter $a \in \Sigma$ is $\{(q, q') \mid (q, a, q') \in \delta\}$ and the transition $\delta(w)$ of a string $w = a_1 \ldots a_n \in \Sigma^*$ is $\delta(w) = (\delta(a_1) \ldots \delta(a_n))^{T_Q}$. This is the composition of the transitions of all letters of $w$ based on the operations of the transition monoid $T_Q$, and its neutral element for the empty word. Note that if $A$ is a DFA then all transitions $\delta(w)$ are partial functions. A transition $\tau$ is called *$I$,$F$-successful* if $\tau \cap (I \times F) \neq \emptyset$. The language of an NFA $A = (Q, \Sigma, \delta, I, F)$ is the set $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta(w) \text{ is } I,F\text{-successful}\}$. The size of an automaton is $|A| = |Q| + |\delta|$.

*Example 1.* NFAs can be used to define Boolean queries on strings such as query $\mathbf{Q}_2$ which tests whether some position will be selected by query $\mathbf{Q}_1$, i.e., whether

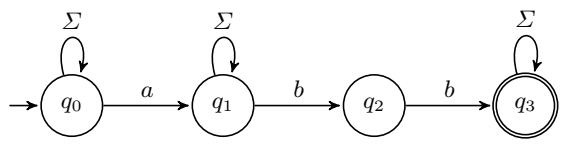



Fig. 5: Automaton $A_2$ defining the Boolean query $\mathbf{Q}_2$ with alphabet $\Sigma = \{a, b, c\}$

a string contains an $a$-letter followed eventually by a factor $bb$. The language of all such strings is defined by the automaton on Fig. 5.

A transition $\tau$ is called *$\delta$-inhabited* if there exists a word $w \in \Sigma^*$ such that $\delta(w) = \tau$. *Transition inhabitation* $\mathrm{Inh}_\Sigma$ is the decision problem that receives as input a finite set $Q$, a transition relation $\delta \subseteq Q \times \Sigma \times Q$ and a transition $\tau \in T_Q$, and outputs whether $\tau$ is $\delta$-inhabited. $\mathrm{Inh}_\Sigma$ is also called the *membership problem of the transition monoid* $\delta(\Sigma^*) \subseteq T_Q$ [7].

**Theorem 1 (Kozen [17]).** *For any set $\Sigma$ with at least 2 elements, the transition inhabitation problem $Inh_\Sigma$ is* PSPACE*-complete.*

The PSPACE hardness proof can be done by reduction from the problem of *non-emptiness of the intersection of sequences of* DFA*s*, which was shown to be PSPACE-complete in [17] too.

## 3 Compressed String Patterns

We fix an infinite set $\mathcal{Y}$ of *string variables* for the rest of the paper. A *string pattern* over a finite alphabet $\Sigma$ is a word in $(\Sigma \cup \mathcal{Y})^*$. The set of all string patterns over $\Sigma$ is denoted by $Pat_\Sigma$. The set of variables that occur in a string pattern $p$ is denoted by $fv(p)$. An instance of a string pattern $p \in Pat_\Sigma$ is a string that can be obtained by substituting the variables of $p$ by strings in $\Sigma^*$. Any substitution $\sigma : \mathcal{Y} \to \Sigma^*$ can be lifted to a substitution on string patterns $\hat{\sigma} : Pat_\Sigma \to \Sigma^*$ such that for all $p, p' \in Pat_\Sigma$, $a \in \Sigma$, and $Y \in \mathcal{Y}$: $\hat{\sigma}(pp') = \hat{\sigma}(p) \cdot \hat{\sigma}(p')$, $\hat{\sigma}(\varepsilon) = \varepsilon$, $\hat{\sigma}(a) = a$, and $\hat{\sigma}(Y) = \sigma(Y)$. We define the set of *instances* of a string pattern $p \in Pat_\Sigma$ as:

$$Inst(p) = \{\hat{\sigma}(p) \mid \sigma : \mathcal{Y} \to \Sigma^*\}.$$

For example, the string $a\underline{cb}c\underline{b}abb\underline{cb}a$ is an instance of the pattern $aYcZabbYa$, obtained with the substitution $[Y/cb, Z/b]$. A string pattern is called *linear*, if all its variables occur at most once. The set of all linear string patterns over $\Sigma$ is denoted $LinPat_\Sigma$.

**Definition 1 (Compressed string pattern).** *A compressed string pattern is an* acyclic *CFG $G = (N, \Sigma, R, S)$ where $N \subseteq \mathcal{Y}$ is a finite set of nonterminals, $\Sigma$ is an alphabet of terminal symbols disjoint from $\mathcal{Y}$, the ruling function $R$ is*

*a partial function that maps some of the nonterminals in $N$ to string patterns in $(N \cup \Sigma)^*$, and $S \in N$ is the start symbol. The set of all compressed string patterns over $\Sigma$ is denoted by $cPat_\Sigma$.*

We recall that a CFG $G$ is acyclic if the binary relation $>_G = \{(Y, Z) \mid Y \in dom(R), Z \in fv(R(Y))\}$ is acyclic. The compressed string pattern from the introduction for instance has the rules $R(S) = aXbbYaX, R(X) = YcZa$. These rules induce the binary relation $\{(S, X), (S, Y), (X, Y), (X, Z)\}$ which is acyclic. The size of $G$ is $|G| = |N| + \sum_{Y \in dom(R)} |R(Y)|$. The set of free variables of $G$ is $fv(G) = N \setminus dom(R)$. A compressed string pattern $G$ is called a *singleton context-free grammar* (sCFG) if it has no free variables, that is $fv(G) = \emptyset$. It is well-known that any sCFG defines a single string in $\Sigma^*$. The object of interest here is the set of strings that can be obtained by completing a compressed string pattern $G$ to a sCFG, or equivalently, the set of instances of the string pattern of $G$ defined as follows. For any compressed string pattern $G = (N, \Sigma, R, S)$, the grammar $G' = (N \setminus fv(G), \Sigma \cup fv(G), R, S)$ is a sCFG. We define the string pattern $pat(G) \in Pat_{\Sigma \cup fv(G)}$ as the unique word in the language of $G'$. Formally, for every $Y \in dom(R)$, let $G_Y$ be the compressed string pattern $G_Y = (N, \Sigma, R, Y)$. If $S \in dom(R)$ then $pat(G) = \hat{\sigma}(R(S))$ where $\sigma(Y) = pat(G_Y)$ for all $Y \in dom(R) \setminus \{S\}$. This recursive definition is well-founded because $G$ is an acyclic CFG. Otherwise, if $S \in fv(G)$, then $pat(G) = S$. For instance, if $G_1$ is the hyperstream from the introduction, then $pat(G_1) = p_1$.

A compressed string pattern $G$ is called a *compressed linear string pattern* if $pat(G)$ is linear. The set of all compressed linear string patterns over $\Sigma$ is denoted $cLinPat_\Sigma$. Finally for any string pattern $p \in Pat_\Sigma$ there exists a compressed string pattern $G_p$ having $p$ as pattern, namely $G_p = (\{S\} \cup fv(p), \Sigma, R, S)$ with $dom(R) = \{S\}$ and $R(S) = p$. Clearly $pat(G_p) = p$. Therefore we will identify $p$ with $G_p$, so that $Pat_\Sigma \subseteq cPat_\Sigma$.

## 4 Regular Pattern Inclusion and Matching

We consider the problems of *regular compressed pattern inclusion*, i.e. testing whether all strings described by a completion of a compressed string pattern to a sCFG are accepted by a finite automaton, and of *regular compressed pattern matching*, whether some string described by a completion of a compressed string pattern is accepted by a finite automaton.

A class of compressed string patterns $\mathcal{G}$ is a function from finite sets $\Sigma$ to subsets $\mathcal{G}_\Sigma \subseteq cPat_\Sigma$ such as for instance $\mathcal{G} \in \{Pat, cPat, LinPat, cLinPat\}$. A class of NFAs $\mathcal{A}$ is a function from finite sets $\Sigma$ to subsets $\mathcal{A}_\Sigma \subseteq \text{NFA}_\Sigma$, where $\text{NFA}_\Sigma$ is the set of NFAs with alphabet $\Sigma$. Most typically, $\mathcal{A} \in \{\text{DFA}, \text{NFA}\}$. For any class $\mathcal{G}$ of compressed string patterns, any class $\mathcal{A}$ of NFAs, and any finite set $\Sigma$ we define the following two decision problems.

**Regular compressed pattern inclusion** $\text{INCL}_\Sigma(\mathcal{G}, \mathcal{A})$. *Input:* A compressed string pattern $G \in \mathcal{G}_\Sigma$ and a finite automaton $A \in \mathcal{A}_\Sigma$.
*Output:* The truth value of whether $Inst(pat(G)) \subseteq \mathcal{L}(A)$.

**Regular compressed pattern matching** $\textsc{Match}_\Sigma(\mathcal{G}, \mathcal{A})$. *Input:* A compressed string pattern $G \in \mathcal{G}_\Sigma$ and a finite automaton $A \in \mathcal{A}_\Sigma$.
*Output:* The truth value of whether $\mathit{Inst}(\mathit{pat}(G)) \cap \mathcal{L}(A) \neq \emptyset$.

The problem $\textsc{coMatch}_\Sigma(\mathcal{G}, \mathcal{A})$ is the complement of the problem $\textsc{Match}_\Sigma(\mathcal{G}, \mathcal{A})$, and thus outputs for a given compressed string pattern $G \in \mathcal{G}_\Sigma$ and a finite automaton $A \in \mathcal{A}_\Sigma$ whether $\mathit{Inst}(\mathit{pat}(G)) \cap \mathcal{L}(A) = \emptyset$.

*Example 2.* Any instance of $\mathit{pat}(G_1) = aYcZabbYaYcZa$ answers the Boolean query $\mathbf{Q}_2 = [\texttt{successor}^*\texttt{::}a/\texttt{successor}^*\texttt{::}b/\texttt{successor::}b]$ from Example 1, i.e., the instance set of $\mathit{pat}(G_1)$ is included in the language of NFA $A_2$ in Fig. 5.

Let $s$DFA be the subclass of DFAs that recognize a singleton language. Note that the well-known problem of string pattern matching is $\textsc{Match}_\Sigma(\mathit{Pat}, s\textsc{Dfa})$, and $\textsc{Match}_\Sigma(\mathit{cPat}, s\textsc{Dfa})$ is its extension with compression. We recall from [11] that string pattern matching with and without compression respectively are NP-complete for all alphabets $\Sigma$ with at least 2 letters, but in PTIME when restricted to linear string patterns even with compression.

Our first main contribution is the following complexity result for regular compressed pattern matching and inclusion (see Fig. 3).

**Theorem 2 (Non-linear patterns).** *For any $\mathcal{G} \in \{\mathit{Pat}, \mathit{cPat}\}$ and $\mathcal{A} \in \{\textsc{Dfa}, \textsc{Nfa}\}$ and for any finite alphabet $\Sigma$ with at least 2 letters, the problems of regular compressed pattern inclusion $\textsc{Incl}_\Sigma(\mathcal{G}, \mathcal{A})$ and matching $\textsc{Match}_\Sigma(\mathcal{G}, \mathcal{A})$ are* PSPACE*-complete.*

This shows that these problems are decidable even though the instance sets of nonlinear patterns like $\mathit{Inst}(YYY)$ are neither regular nor context-free. The theorem also shows that regular pattern matching $\textsc{Match}_\Sigma(\mathit{Pat}, \textsc{Dfa})$ is PSPACE-complete and thus harder than compressed string pattern matching $\textsc{Match}_\Sigma(\mathit{Pat}, s\textsc{Dfa})$ which is NP-complete.

*Proof.* We will present a sequence of PSPACE reductions from Lemma 1 until Lemma 4 that imply the theorem when composed as in Fig. 6.

For two decision problems $A$ and $B$, we write $A =_p B$ when $A$ reduces polynomially to $B$ and $B$ reduces polynomially to $A$.

**Lemma 1.** $\textsc{Incl}_\Sigma(\mathit{LinPat}, \textsc{Nfa})$ *is* PSPACE*-hard if* $|\Sigma| \geq 2$.

The proof is straightforward by reduction from the universality of NFAs.

We now show the PSPACE upper bound for $\textsc{Incl}_\Sigma(\mathit{cPat}, \textsc{Nfa})$. For any transition relation $\delta \subseteq Q \times \Sigma \times Q$ and any substitution into the transition monoid $\sigma : \mathcal{Y} \to T_Q$, we define $\delta^\sigma$ to be the function that takes a string pattern in $\mathit{Pat}_\Sigma$ as input, and returns a transition, such that for all $p, p' \in \mathit{Pat}_\Sigma$, $w \in \Sigma^*$, and $Y \in \mathcal{Y}$:

$$\delta^\sigma(w) = \delta(w), \qquad \delta^\sigma(y) = \sigma(y), \qquad \delta^\sigma(pp') = \delta^\sigma(p) \circ \delta^\sigma(p').$$

$$
\begin{array}{ccccccc}
\textsc{PSpace}\text{-hard} & & & & \textsc{PSpace}\text{-complete} & \textsc{PSpace}\text{-hard} & \\
\cup^{\text{Lem. 1}} & & & & \cup^{\text{Lem. 3}} & \cup^{\text{Prop. 3}} & \\
\textsc{Incl}_\Sigma(Pat, \textsc{Nfa}) & \supseteq & \textsc{Incl}_\Sigma(Pat, \textsc{Dfa}) & \overset{\text{Lem. 4}}{=_p} & \textsc{coMatch}_\Sigma(Pat, \textsc{Dfa}) & \textsc{Match}_\Sigma(Pat, \textsc{Dfa}) & \subseteq \textsc{Match}_\Sigma(Pat, \textsc{Nfa}) \\
\cap\!| & & \cap\!| & & \cap\!| & \cap\!| & \cap\!| \\
\textsc{Incl}_\Sigma(cPat, \textsc{Nfa}) & \supseteq & \textsc{Incl}_\Sigma(cPat, \textsc{Dfa}) & \overset{\text{Lem. 4}}{=_p} & \textsc{coMatch}_\Sigma(cPat, \textsc{Dfa}) & \textsc{Match}_\Sigma(cPat, \textsc{Dfa}) & \subseteq \textsc{Match}_\Sigma(cPat, \textsc{Nfa}) \\
\cap^{\text{Prop. 1}} & & & & & & \cap^{\text{Prop. 1}b} \\
\textsc{PSpace} & & & & & & \textsc{PSpace}
\end{array}
$$

Fig. 6: Regular inclusion and matching problems relationship and complexity classes

**Lemma 2.** *Given a transition relation $\delta \subseteq Q \times \Sigma \times Q$ of some* Nfa*, a substitution $\sigma : \mathcal{Y} \to T_Q$ and a compressed string pattern $G \in cPat_\Sigma$, the transition $\delta^\sigma(pat(G)) \in T_Q$ can be computed in time $O(|Q|^3|G|)$.*

**Proposition 1.** $\textsc{Incl}_\Sigma(cPat, \textsc{Nfa})$ *is in* PSpace*.*

*Proof.* Given an Nfa $A$, a compressed string pattern $G$ over $\Sigma$, we have to check whether $Inst(pat(G)) \subseteq \mathcal{L}(A)$. By definition, this is equivalent to checking whether $\delta(\hat{s}(pat(G)))$ is $I$,$F$-successful for every substitution $s : fv(G) \to \Sigma^*$. The latter is equivalent to checking whether $\delta^\sigma(pat(G))$ is $I$,$F$-successful for all substitution $\sigma : fv(G) \to T_Q$ that maps the free variables of $G$ to $\delta$-inhabited transitions. A decision procedure can thus enumerate all substitutions $\sigma : fv(G) \to T_Q$ to $\delta$-inhabited transitions, compute $\delta^\sigma(pat(G))$ and check whether it is $I$,$F$-successful. The number of substitutions $\sigma$ that is to be checked is exponential, but they can be enumerated in PSpace. Whether $\sigma$ maps only to $\delta$-inhabited transition can be tested in PSpace by Theorem 1. Computing $\delta^\sigma(pat(G))$ can be done in PTime by Lemma 2.

So far we have shown that $\textsc{Incl}_\Sigma(cPat, \textsc{Nfa})$ is PSpace-complete. We next consider regular matching. This will permit us to show that $\textsc{Incl}_\Sigma(cPat, \textsc{Dfa})$ is PSpace-hard too.

**Proposition 2.** $\textsc{Match}_\Sigma(cPat, \textsc{Nfa})$ *is in* PSpace*.*

The proof is analogous to that of Proposition 1, except that it is now sufficient to guess some substitution to $\delta$-inhabited transition.

**Proposition 3.** $\textsc{Match}_\Sigma(Pat, \textsc{Dfa})$ *is* PSpace*-hard.*

The proof is by reduction from the non-emptiness problem of the intersection of a sequence of $n$ Dfas $A_1, \ldots, A_n$ over $\Sigma$ by matching the non-linear string pattern $y\# \ldots \#y$ of length $n$ against a Dfa $A$ over $\Sigma \uplus \{\#\}$ recognizing $\mathcal{L}(A) = \mathcal{L}(A_1)\# \ldots \#\mathcal{L}(A_n)$.

**Lemma 3.** *The problem* $\textsc{coMatch}_\Sigma(Pat, \textsc{Dfa})$ *is* PSpace*-complete.*

In order to complete the proof of Theorem 2 it remains to relate regular inclusion and matching in the case of DFAs.

**Lemma 4.** $\text{coMatch}_\Sigma(\mathcal{G}, \text{DFA}) =_p \text{Incl}_\Sigma(\mathcal{G}, \text{DFA})$ *for all* $\mathcal{G}$ *up to* PTIME *red.*

*Proof.* This follows from $Inst(pat(G)) \cap \mathcal{L}(A) = \emptyset \Leftrightarrow Inst(pat(G)) \subseteq \overline{\mathcal{L}(A)}$ and the fact that any DFA $A$ can be complemented in PTIME to some DFA $\overline{A}$ such that $\overline{L(A)} = L(\overline{A})$.

The complexity of inclusion and matching decreases for linear string patterns, with or without compression. We indeed obtain the same complexity as known for streams (see Fig. 4), even though unknown factors and compression are permitted in addition.

**Theorem 3 (Linear patterns).** *Restricted to linear string patterns, regular compressed pattern inclusion and matching have the following complexities:*

1. $\text{Incl}_\Sigma(LinPat, \text{NFA})$ *and* $\text{Incl}_\Sigma(cLinPat, \text{NFA})$ *are* PSPACE*-complete if* $|\Sigma| \geq 2$ *while the problem* $\text{Incl}_\Sigma(cLinPat, \text{DFA})$ *can be solved in* PTIME.
2. $\text{Match}_\Sigma(cLinPat, \text{NFA})$ *is in* PTIME.

For any linear compressed string pattern $G$ there exist a DFA of that recognizes $Inst(pat(G))$, but this DFA may be of exponential size in $|G|$ due to compression. Instead, the PTIME proof for $\text{Incl}_\Sigma(cLinPat, \text{DFA})$ relies on an evaluator of compressed string patterns in the transition monoid, that replaces free variables by the accessibility transition of the DFA, thereby testing a *reachability property.*

## 5 Defining Queries by Automata

We now recall the notion of queries on strings over some alphabet $\Sigma$ with *variables* in some finite set $\mathcal{V}$ and relate them to languages of $\mathcal{V}$-annotated strings called $\mathcal{V}$-structures in [26]. We fix two disjoint finite sets $\Sigma$ and $\mathcal{V}$.

**Definition 2 (Query).** *A* query *with variables in* $\mathcal{V}$ *on strings over* $\Sigma$*, or a* $\Sigma$,$\mathcal{V}$-query *for short, is a function* $\boldsymbol{Q}$ *that maps any string* $w \in \Sigma^*$ *to a set* $\boldsymbol{Q}(w)$ *of total assignments from* $\mathcal{V}$ *to* $pos(w)$*. A Boolean query is a* $\Sigma$,$\emptyset$*-query.*

*Example 3.* Let $\mathcal{V} = \{x, x'\}$. The query $\mathbf{Q}_1$ selects all pairs of letters $(x, x')$ such that position $x$ is labeled by $a$, position $x'$ immediately follows $x$ and is labeled by $b$. This query then satisfies $\mathbf{Q}_1(aa) = \emptyset$, $\mathbf{Q}_1(ab) = \{[x/1, x'/2]\}$, $\mathbf{Q}_1(abab) = \{[x/1, x'/2], [x/3, x'/4]\}$, etc.

We next show how a $\Sigma$,$\mathcal{V}$-query can be identified with a language of $\mathcal{V}$-annotated strings, i.e., of words over the alphabet $\Sigma^{\mathcal{V}} = \Sigma \times 2^{\mathcal{V}}$. A *(query variable) assignment* $\alpha$ to positions of a string $w \in \Sigma^*$ is a partial function from $\mathcal{V}$ to $pos(w)$. We will identify such variable assignments with words whose letters are sets of variables. For any partial function $\alpha$ from $\mathcal{V}$ to $pos(w)$ where $w \in \Sigma^n$,

we define a corresponding word in $(2^{\mathcal{V}})^n$ by $word(\alpha) = \alpha^{-1}(1)\ldots\alpha^{-1}(n)$. That is, $word(\alpha)[i]$ is the set of variables $x \in dom(\alpha)$ s.t. $\alpha(x) = i$. Furthermore, for any string $w \in \Sigma^*$ and variable assignment $\alpha$ into positions of $w$, we define the $\mathcal{V}$-annotated string $w{*}\alpha$ as a word over $\Sigma^{\mathcal{V}}$ by $w{*}\alpha = w{*}word(\alpha)$. In examples we will write $a^V$ instead of letters $(a, V) \in \Sigma^{\mathcal{V}}$. For instance, $ab{*}[x/1, x'/2]$ is written as $a^{\{x\}}b^{\{x'\}}$.

**Definition 3 ($\mathcal{V}$-structure [26]).** *The set of* $\mathcal{V}$-structures *over $\Sigma$ is the following set of $\mathcal{V}$-annotated strings, i.e., of words over $\Sigma^{\mathcal{V}}$:*

$$Struct^{\mathcal{V}} = \{w{*}\alpha \in (\Sigma^{\mathcal{V}})^* \mid w \in \Sigma^*,\ \alpha : \mathcal{V} \to pos(w)\}.$$

We note that all the assignments $\alpha$ in the definition of $\mathcal{V}$-structures are total functions. For instance for $\mathcal{V} = \{x, x'\}$ and $\Sigma = \{a, b\}$, the words $a^{\emptyset}b^{\{x',x\}}$ and $a^{\{x'\}}b^{\{x\}}$ are $\mathcal{V}$-structures while the words $a^{\emptyset}b^{\{x'\}}$ and $a^{\{x'\}}b^{\{x',x\}}$ are not. Essentially, $\mathcal{V}$-structures represent total variable assignments to the positions of a string without naming the positions.

**Definition 4 (Language of $\mathcal{V}$-structures of a query).** *For any $\Sigma$,$\mathcal{V}$-query $\boldsymbol{Q}$, the* language of $\mathcal{V}$-structures of $\boldsymbol{Q}$ *is* $\mathcal{L}(\boldsymbol{Q}) = \{w{*}\alpha \mid w \in \Sigma^*,\ \alpha \in \boldsymbol{Q}(w)\}$.

We will be interested in queries whose languages are definable by NFAs.

**Definition 5 (Query automata).** *An $\Sigma$,$\mathcal{V}$-query automaton is an* NFA *$A$ such that $\mathcal{L}(A)$ is a language of $\mathcal{V}$-structures over $\Sigma$. The unique $\Sigma$,$\mathcal{V}$-query such that $\mathcal{L}(\boldsymbol{Q}) = \mathcal{L}(A)$ is called the query defined by $A$ and is denoted by $\mathcal{Q}(A) = \boldsymbol{Q}$.*

## 6 Certain Query Answers and Non-Answers

We next formalize the notions of certain query answers and non-answers on string patterns. For streams, these definitions coincide with the notions of earliest query answers from [12] and fast-failure from [4], respectively.

A *$\Sigma$-assignment* for $\mathcal{V}$ on $p \in Pat_{\Sigma}$ is a partial function $\alpha$ from $\mathcal{V}$ to $pos_{\Sigma}(p)$, the $\Sigma$-positions of the pattern $p$. For any $\Sigma$-assignment $\alpha$ on $p$, the word $p{*}\alpha$ is a string pattern over $\Sigma^{\mathcal{V}}$, still with string variables in $\mathcal{Y}$. Therefore, the set $Inst(p{*}\alpha)$ is a well-defined set of words over $\Sigma^{\mathcal{V}}$. Note, however, that some of these $\mathcal{V}$-annotated strings may not be $\mathcal{V}$-structures. For instance, if $x \in \mathcal{V}$, $a \in \Sigma$, and $Y \in \mathcal{Y}$, then $a^{\{x\}}a^{\{x\}} \in Inst(Y{*}[])$ is not a $\mathcal{V}$-structure since $x$ occurs twice (where $[]$ is the empty $\Sigma$-assignment).

**Definition 6 (Certain query answers and non-answers).** *Let $\boldsymbol{Q}$ be a $\Sigma$,$\mathcal{V}$-query, and let $p \in Pat_{\Sigma}$ be a string pattern. A $\Sigma$-assignment $\alpha$ for $\mathcal{V}$ on $p$ is:*

- *a* certain answer *for query $\boldsymbol{Q}$ on $p$ if $\alpha$ is total and $Inst(p{*}\alpha) \cap Struct^{\mathcal{V}} \subseteq \mathcal{L}(\boldsymbol{Q})$,*
- *and a* certain non-answer *for query $\boldsymbol{Q}$ on $p$ if $Inst(p{*}\alpha) \cap \mathcal{L}(\boldsymbol{Q}) = \emptyset$.*

Given an instance $w \in \mathit{Inst}(p)$, each $\Sigma$-assignment $\alpha$ on $p$ defines a set of total $\Sigma$-assignments on $w$, where all variables not in $\mathit{dom}(\alpha)$ must be mapped to some $\Sigma$-positions "created" by the instantiation. More formally:

$$C_{p,w}(\alpha) = \{\alpha' \mid \alpha' \text{ is a total } \Sigma\text{-assignment on } w,\ w{*}\alpha' \in \mathit{Inst}(p{*}\alpha)\}.$$

The offsets of positions of query variables due to the instantiation of pattern variables raise two issues that we illustrate in the following example: 1. even a total $\Sigma$-assignment $\alpha$ of $p$ might have several completions for the same string $w$, and 2. it might be the case that $\alpha \notin C_{p,w}(\alpha)$.

*Example 4.* Consider the string pattern $p = ay_1ay_2$, the string $w = aaba$ in $\mathit{Inst}(p)$, $\mathcal{V} = \{x\}$, and the total $\Sigma$-assignment $\alpha = [x/3]$ on $p$. In order to make $p$ match $w$, the second $a$-letter of $p$ matches either the second or the fourth position in $w$. Therefore, there are exactly two substitutions that make $p$ match $w$, which are $\sigma_1 = [y_1/\varepsilon, y_2/ba]$ and $\sigma_2 = [y_1/ab, y_2/\varepsilon]$. Now, $\hat{\sigma_1}(p{*}\alpha) = \hat{\sigma_1}(ay_1a^{\{x\}}y_2) = aa^{\{x\}}ba = w{*}[x/2]$, thus $[x/2] \in C_{p,w}(\alpha)$. Also, $\hat{\sigma_2}(p{*}\alpha) = \hat{\sigma_1}(ay_1a^{\{x\}}y_2) = aaba^{\{x\}} = w{*}[x/4]$, thus $[x/4] \in C_{p,w}(\alpha)$. Given that there is no further way to match $p$ with $w$, there is no further completion. That is, $C_{p,w}(\alpha) = \{[x/2], [x/4]\}$, and $\alpha \notin C_{p,w}(\alpha)$.

The next proposition relates certain query answers (resp. non-answers) on a pattern $p$ to query answers (resp. non-answers) on its instances.

**Proposition 4.** *Let $\alpha$ be a $\Sigma$-assignment on string pattern $p$ and $\boldsymbol{Q}$ be a $\Sigma,\mathcal{V}$-query. It then holds for all instances $w \in \mathit{Inst}(p)$:*

- *If $\alpha$ is a certain answer for query $\boldsymbol{Q}$ on $p$ then $C_{p,w}(\alpha) \subseteq \boldsymbol{Q}(w)$.*
- *If $\alpha$ is a certain non-answer for query $\boldsymbol{Q}$ on $p$ then $C_{p,w}(\alpha) \cap \boldsymbol{Q}(w) = \emptyset$.*

## 7 Certain Query Answering and Non-Answering

We introduce the problems of certain query answering and non-answering for classes of compressed string patterns $\mathcal{G}$ and of query NFAs $\mathcal{A}$:

**Certain query answering** $\textsc{Cert}^{ans}_{\Sigma,\mathcal{V}}(\mathcal{G},\mathcal{A})$. *Input:* a compressed string pattern $G \in \mathcal{G}_\Sigma$, a $\Sigma$-assignment $\alpha$ for $\mathcal{V}$ on $\mathit{pat}(G)$, and a query NFA $A \in \mathcal{A}_{\Sigma^\mathcal{V}}$. *Output:* whether $\alpha$ is a certain answer for query $\mathcal{Q}(A)$ on $\mathit{pat}(G)$.

**Certain query non-answering** $\textsc{Cert}^{\neg ans}_{\Sigma,\mathcal{V}}(\mathcal{G},\mathcal{A})$. *Input:* as above *Output:* whether $\alpha$ is a certain non-answer for query $\mathcal{Q}(A)$ on $\mathit{pat}(G)$.

Note that $\alpha$ is an assignment to positions of $\mathit{pat}(G)$ and not to positions of $G$. This is necessary because due to compression, a position of $G$ might correspond to several positions of the underlying word which need to be distinguished. Indeed, considering $G$ to be the compressed string pattern in the introduction, the $a$-letter in the rule for $X$ corresponds to positions 5 and 13 in the decompressed pattern $a\,YcZabb\,Ya\,YcZa$, and position 5 is a certain answer for query $\mathbf{Q}_1$, while position 13 is a certain non-answer.

The following straightforward lemma relates certain query answering to regular inclusion and certain query non-answering to regular matching.

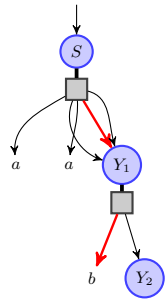



Fig. 7: $G$ with red thick path to the shared position 5.

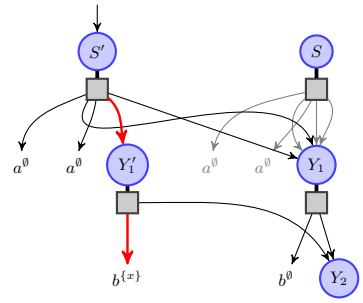



Fig. 8: $G'$ on $\Sigma^{\mathcal{V}}$ with $pat(G') = pat(G)*[x/5]$.

**Lemma 5 (Boolean Queries).** *For any classes $\mathcal{G}$ and $\mathcal{A}$,* $\textsc{Incl}_{\Sigma}(\mathcal{G},\mathcal{A}) =_p \textsc{Cert}^{ans}_{\Sigma,\emptyset}(\mathcal{G},\mathcal{A})$ *and* $\textsc{coMatch}_{\Sigma}(\mathcal{G},\mathcal{A}) =_p \textsc{Cert}^{\neg ans}_{\Sigma,\emptyset}(\mathcal{G},\mathcal{A})$.

Since PSpace-complete problems are closed by complement, Lemma 5 implies together with Theorem 2 that $\textsc{Cert}^{ans}_{\Sigma,\emptyset}(Pat, \textsc{Dfa})$ and $\textsc{Cert}^{\neg ans}_{\Sigma,\emptyset}(Pat, \textsc{Dfa})$ are PSpace-complete, even though these problems are restricted to Dfas, without compression, and for Boolean queries. Therefore, all certainty problems $\textsc{Cert}^{\mathcal{B}}_{\Sigma,\emptyset}(\mathcal{G},\mathcal{A})$ where $\mathcal{B} \in \{\neg ans, ans\}$, $\mathcal{G} \in \{Pat, cPat\}$, and $\mathcal{A} \in \{\textsc{Dfa}, \textsc{Nfa}\}$ are PSpace-hard. In the sequel we show that all these problems can be solved in PSpace for arbitrary finite sets $\Sigma$ and $\mathcal{V}$. Basically, these results will be corollaries of Theorem 2, Lemma 5 on Boolean queries, and the following partial decompression lemma. This result is new to the best of our knowledge, even though its proof relies on similar techniques as used for instance in [5] for computing in PTime the letter at the $n$-th position of $pat(G)$ for a singleton grammar $G$.

**Lemma 6 (Partial Decompression).** *For any $G \in cPat_{\Sigma}$ and any $\Sigma$-assignment $\alpha$ for $\mathcal{V}$ on $pat(G)$, we can compute in* PTime *some $G' \in cPat_{\Sigma^{\mathcal{V}}}$ such that $pat(G)*\alpha = pat(G')$. In particular, if $pat(G)$ was linear then $pat(G')$ is linear.*

For illustration, let $\mathcal{V} = \{x\}$, $\alpha = [x/5]$. Fig. 8 shows a compressed string pattern $G'$ obtained by partially decompressing the compressed string pattern $G$ in Fig. 7 at the position 5 of its pattern, such that $pat(G)*\alpha = pat(G')$.

**Proposition 5.** *For all $\mathcal{B}$ in $\{ans, \neg ans\}$, all $\mathcal{G}$ in $\{Pat, cPat, LinPat, cLinPat\}$, and all $\mathcal{A}$ in $\{\textsc{Dfa}, \textsc{Nfa}\}$, there is a* PTime *reduction from $\textsc{Cert}^{\mathcal{B}}_{\Sigma,\mathcal{V}}(\mathcal{G},\mathcal{A})$ to $\textsc{Cert}^{\mathcal{B}}_{\Sigma^{\mathcal{V}},\emptyset}(\mathcal{G},\mathcal{A})$.*

**Corollary 1 (Non-linear patterns).** *For all $\mathcal{B} \in \{ans, \neg ans\}$, $\mathcal{G} \in \{Pat, cPat\}$ and $\mathcal{A} \in \{\textsc{Dfa}, \textsc{Nfa}\}$, the problem $\textsc{Cert}^{\mathcal{B}}_{\Sigma,\mathcal{V}}(\mathcal{G},\mathcal{A})$ is* PSpace*-complete.*

**Corollary 2 (Linear patterns).** *For all $\mathcal{A} \in \{\textsc{Dfa}, \textsc{Nfa}\}$, the certainty problems $\textsc{Cert}^{\neg ans}_{\Sigma,\mathcal{V}}(cLinPat, \mathcal{A})$ (1) and $\textsc{Cert}^{ans}_{\Sigma,\mathcal{V}}(cLinPat, \textsc{Dfa})$ (2) can be solved in* PTime*. The problem $\textsc{Cert}^{ans}_{\Sigma,\mathcal{V}}(\mathcal{G}, \textsc{Nfa})$ is* PSpace*-complete for all $\mathcal{G} \in \{LinPat, cLinPat\}$ (3).*

## 8 Conclusion

There exist highly efficient streaming algorithms for answering queries defined by NFAs [10] on complex event streams with low latency, but not with lowest latency, since they approximate the sets of certain query answers at any event. The positive results presented here yield good hope that similar algorithms could be developed for hyperstreams when approximated by compressed linear string patterns. As shown by the authors in a follow-up paper, the linearity restriction is not sufficient. But with a further restriction it is possible to approximate certain query answers efficiently and with decent precision [8]. However, this still requires more research. First, one needs to understand how such algorithms may deal with unknown factors incrementally, without requiring cubic time per step such as previous algorithms on incremental evaluation of queries defined by NFAs [6]. Second, one has to understand how to deal with nested word automata rather than NFAs for dealing with regular path queries on complex event streams. Another point is to develop streaming algorithms for hyperstreams with data values from an infinite signature. Finally, the feasibility of hyperstreaming algorithms needs to be proven in practice.

## References

1. R. Angles, M. Arenas, P. Barceló, A. Hogan, J. L. Reutter, and D. Vrgoc. Foundations of modern graph query languages. *CoRR*, abs/1610.06264, 2016.
2. D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.
3. L. Babai and E. Szemeredi. On the complexity of matrix group problems i. In *Proceedings of the 25th Annual Symposium onFoundations of Computer Science, 1984*, SFCS '84, pages 229–240, Washington, DC, USA, 1984. IEEE Computer Society.
4. M. Benedikt, A. Jeffrey, and R. Ley-Wild. Stream Firewalling of XML Constraints. In *ACM SIGMOD International Conference on Management of Data*, pages 487–498. ACM-Press, 2008.
5. P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann. Random access to grammar-compressed strings and trees. *SIAM J. Comput.*, 44(3):513–539, 2015.
6. H. Björklund, W. Gelade, and W. Martens. Incremental XPath evaluation. *ACM Trans. Database Syst.*, 35(4):29, 2010.
7. M. Blondin, A. Krebs, and P. McKenzie. The complexity of intersecting finite automata having few final states. *computational complexity*, 25(4):775–814, Dec 2016.
8. I. Boneva, J. Niehren, and M. Sakho. Approximating Certain Query Answering on Hyperstreams, June 2018. technical report.
9. C. David, L. Libkin, and F. Murlak. Certain answers for XML queries. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 191–202. ACM, 2010.

10. D. Debarbieux, O. Gauwin, J. Niehren, T. Sebastian, and M. Zergaoui. Early nested word automata for xpath query answering on XML streams. *Theor. Comput. Sci.*, 578:100–125, 2015.
11. A. Gascón, G. Godoy, and M. Schmidt-Schauß. Context matching for compressed terms. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 93–102. IEEE Computer Society, 2008.
12. O. Gauwin and J. Niehren. Streamable fragments of forward XPath. In B. B. Markhoff, P. Caron, J. M. Champarnaud, and D. Maurel, editors, *International Conference on Implementation and Application of Automata*, volume 6807 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2011.
13. O. Gauwin, J. Niehren, and S. Tison. Earliest Query Answering for Deterministic Nested Word Automata. In *17th International Symposium on Fundamentals of Computer Theory*, volume 5699 of *LNCS*, pages 121–132, Wraclaw, Poland, Sept. 2009. Springer Verlag.
14. O. Gauwin, J. Niehren, and S. Tison. Queries on XML streams with bounded delay and concurrency. *Information and Computation*, 209:409–442, 2011.
15. T. J. Green, A. Gupta, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata and stream indexes. *ACM Trans. Database Syst.*, 29(4):752–788, Dec. 2004.
16. M. Kay. A streaming XSLT processor. In *Balisage: The Markup Conference 2010. Balisage Series on Markup Technologies*, volume 5, 2010.
17. D. Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 254–266. IEEE Computer Society, 1977.
18. V. Kumar, P. Madhusudan, and M. Viswanathan. Visibly pushdown automata for streaming XML. In *16th international conference on World Wide Web*, pages 1053–1062. ACM-Press, 2007.
19. O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.
20. P. Labath and J. Niehren. A functional language for hyperstreaming XSLT. Technical report, INRIA Lille, 2013.
21. S. Maneth, A. O. Pereira, and H. Seidl. Transforming XML streams with references. In C. S. Iliopoulos, S. J. Puglisi, and E. Yilmaz, editors, *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings*, volume 9309 of *Lecture Notes in Computer Science*, pages 33–45. Springer, 2015.
22. B. Mozafari, K. Zeng, and C. Zaniolo. High-performance complex event processing over XML streams. In K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, A. Fuxman, K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, and A. Fuxman, editors, *SIGMOD Conference*, pages 253–264. ACM, 2012.
23. D. Olteanu. SPEX: Streamed and progressive evaluation of XPath. *IEEE Trans. on Know. Data Eng.*, 19(7):934–949, 2007.
24. W. Plandowski. *The complexity of the morphism equivalence problem for context-free languages.* PhD thesis, Warsaw University. Department of Informatics, Mathematics, and Mechanics, 1995.
25. M. Schmidt, S. Scherzinger, and C. Koch. Combined static and dynamic analysis for effective buffer minimization in streaming XQuery evaluation. In *23rd IEEE International Conference on Data Engineering*, pages 236–245, 2007.
26. H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Progress in Computer Science and Applied Series. Birkhäuser, 1994.

**Acknowledgments**. We are thankful to C. Paperman, who saw the PSpace-hardness of regular string pattern matching in a discussion on the topic. We thank S. Salvati and S. Tison for discussions on regular string pattern matching. It is a pleasure to thank all the anonymous reviewers for their extraordinary helpful feedback.