Regular Matching and Inclusion on Compressed Tree Patterns with Context Variables

Iovka Boneva, Joachim Niehren, Momar Sakho

Approximating Certain Query Answering on Hyperstreams*

Iovka Boneva
Université de Lille
France
iovka.boneva@univ-lille.fr

Joachim Niehren
Inria
Lille, France
joachim.niehren@inria.fr

Momar Sakho
Inria
Lille, France
momar.sakho@inria.fr

ABSTRACT

Hyperstreams are multiple streams with references to others. They are naturally modeled as incomplete singleton context-free grammars which are also known as compressed string patterns. The problem of certain query answering on hyperstreams was shown to be PSpace-complete for queries defined by both deterministic finite automata (Dfas) and nondeterministic finite automata (Nfas). In this paper, we present an approximation of certain query answering that can be decided in PTime for queries defined by Nfas. By compilation of path queries to Nfas, this permits to answer navigational path queries on hyperstreams with low latency, while conservatively extending on the best previous approximation for path queries on streams to hyperstreams.

Keywords: Complex event processing, queries, data streams, automata, path queries.

KEYWORDS

Complex event processing, queries, data streams, automata

ACM Reference Format:
Iovka Boneva, Joachim Niehren, and Momar Sakho. 2018. Approximating Certain Query Answering on Hyperstreams. In *Proceedings of Conférence sur la Gestion de Données - Principes, Technologies et Applications (BDA'18).* ACM, New York, NY, USA, Article 4, 13 pages. https://doi.org/10.475/123_4

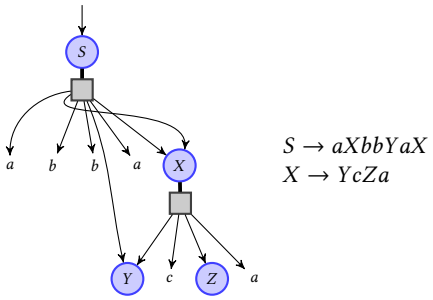

Figure 1: Hyperstream G.

	Dfas	Nfas
Answers	PSpace-c	PSpace-c
Non-answers	PSpace-c	PSpace-c

Figure 2: Certainty.

	Dfas	Nfas
Answers	PTime	PSpace-c
Non-answers	PTime	PTime

Figure 3: Linear certainty.

1 INTRODUCTION

Complex event processing [10, 16, 17, 20] is the problem to process streams of semi-structured data, with the objective to provide low latency, low memory consumption, and very high time efficiency. This way, ever-running streams of complex events can be processed in a reactive manner with low latency (or even in real time), such as streams produced by social networks or trading systems. In this paper, we follow automata based approaches [3, 9, 12, 16] for answering logical queries on streams. For simplicity, we restrict ourselves to streams of words over a finite alphabet, rather than to streams of nested words over an infinite alphabet.

Hyperstreams are collections of streams with references to others [13, 15]. They can be modeled as *incomplete* singleton context-free grammars [18] that are also known as straight-line programs [2] and as Dag compressed string patterns [4]. In Fig. 1 is represented a hyperstream G having the terminal alphabet $\{a, b, c\}$, the set of non-terminal symbols $\{S, X, Y, Z\}$, and the start symbol S. The non-terminals of the grammar are called references of the hyperstream. The right hand side of any grammar rule is a word over the terminal symbols and the references. The hyperstream G contains a rule stating that S refers to $aXbbYaX$ and a second rule stating that X refers to $YcZa$. The incompleteness of the singleton grammar means that grammar rules of some references may be missing. In the example of hyperstream G, rules are missing for the references Y and Z. A hyperstream represents a string pattern in a compressed manner. For instance, G represents the string pattern $aYcZabbYaYcZa$ wherein the non-terminal symbols of G are the variables of the string pattern. Compression is achieved by reusing references for representing common factors, e.g. the reference X that occurs twice in the definition of S. We will identify string patterns with compression-free hyperstreams. The missing rules in a hyperstream can be added incrementally by the hyperstreaming environment until the hyperstream becomes a (complete) singleton grammar. For instance, a possible completion of G can be obtained by adding the rules $Y \to ab$, $Z \to Uc$, and $U \to c$, leading to a compressed representation of the string $aabcccabbabaabccca$. Streams are special string patterns like $aabV$ having at most one occurrence of a reference, which if present must occur at the end of the stream (such as V above).

*Produces the permission block, and copyright information

BDA'18, October 2018, Bucarest, Romania

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

	DFAs	NFAs
Answers	PTIME	?

Figure 4: Strong certainty.

	DFAs	NFAs
Answers	PTIME	PTIME

Figure 5: Pruning strong certainty.

We are interested in answering path queries on hyperstreams that are reminiscent to XPATH queries on XML documents, but restricted to words rather than nested words. For instance consider the Boolean path query P on words with alphabet $\{a, b, c\}$:

$$a(s(s^*(ab(s^*(c))))))$$

Here, s stand for the successor function and s^* for its reflexive transitive closure. This Boolean query accepts all words that start with letter a, succeeded by some letter, then followed eventually by a factor ab which is then followed eventually by a letter c. All instances of the string pattern $aYcZabbYaYcZa$ are accepted by P, independently of how the variables Y and Z – the yet open references of G – will be instantiated. Therefore, the hyperstream G is a certain answer of the path query P. Similar notions of certain query answers are widely used for various kinds of incomplete databases [8].

Gauwin et. al. [6] showed that deciding whether a stream is a certain answer is hard even for tiny fragments of path queries. This justifies the need to approximate the certain query answers. In the case of XPATH queries on XML streams, a first approximation was proposed by Olteanu [17], which is based on a compilation to transducer networks. A slightly better approximation was given in [3], which is based on a compilation to "early" nested word automata ("early" NWAs). These are NWAs [1] with distinguished selection states, in which all well-nested continuations of the current stream will be accepted. Both approximation permit to answer XPATH queries on XML documents with high time efficiency and low latency, but not with lowest latency (see [21] for an experimental comparison).

The objective of the present paper is to find an appropriate approximation of certain query answers for path queries on hyperstreams. The problem is that it is not clear how to lift the approximations based on "early NWAs" or transducer networks from streams to hyperstreams, so a new idea is needed.

Our main contribution to solve this problem is the notion of *pruning strong certainty* for NFA queries on hyperstreams. It

- permits to approximate the certain query answers of NFA queries on hyperstreams,
- conservatively extends on the best previous approximations of certain answers of path queries on streams (based on compilation to "early" NFAs with selection states), and
- can be decided sufficiently efficiently to be promising in practice for query evaluation on hyperstreams with low latency.

In order to develop this notion, we start from the notion of certain query answers and non-answers on hyperstreams developed in a preceding work by the authors [19]. This notion has the advantage to be independent of the query's definition, but the disadvantage that the corresponding decision problem is PSPACE-complete in all possible cases, see Fig. 2.

In the first step we introduce the notion of *linear certainty* on hyperstreams. It approximates the certain (non-)answers on a hyperstream by the certain (non-)answers of the linearizations of its string pattern, obtained by replacing all occurrences of references by fresh references. We then show that linear certainty on hyperstreams can be decided with the same complexity as certainty on streams (see Fig. 3). All decision problems are in PTIME, except for the problem to decide whether a hyperstream is a linearly certain answer of an NFA query, which is PSPACE-complete (as in the case of streams). Unfortunately, determinization is not an option in practice for NWAs obtained by compilation from XPATH queries [21]. The problem is related to the treatment of huge finite alphabets. Indeed, the compiler from XPATH queries produces small descriptors of often huge nondeterministic NWAs; but the determinization would have to be applied to the huge NWAs rather than their small descriptors counterparts. Therefore, linear certainty is not good enough even in the case of path queries on streams, where it coincides with the notion of certainty (given that streams are linear hyperstreams).

In the second step, we introduce the notion of *strongly certain answers* on hyperstreams. The idea is that a hyperstream can be seen as a DAG or circuit and thus evaluated in a bottom-up fashion. For deciding strong certainty for NFA A with state set Q we evaluate hyperstreams over the monoid of functions of type $2^Q \to 2^Q$ while replacing variables by the inverse image of the accessibility relation of the NFA A. It follows immediately from the definitions that any strongly certain answer is also a linearly certain answer. We show that strong certainty:

- for NFAs produced by the compilation from path queries is equivalent to the best previous approximation for path queries on streams based on the compilation to "early NFAs".
- coincides with linear certainty for DFAs (but not for NFAs).
- can be decided in PTIME for hyperstreams without compression (in contrast to linear certainty).

With compression, however, we have to leave the complexity open, see Fig. 4. We believe that adding a visible stack to NFAs as for NWAs (as needed for compiling navigational XPATH queries [3]) would be enough to make the problem hard, but this is out of the scope of the present paper.

In order to deal with general hyperstreams with compression, we finally present the notion of *pruning strong certainty*. It generalizes on the notion of strong certainty, by removing compressed parts of the hyperstream, by using some pruning function. For instance, the pruning function may keep only the n first instances of a shared string factor for some fixed $n \geq 1$. We then apply the notion of strong certainty to the thereby pruned hyperstream. We show for pruning strong certainty that:

- it can be decided in combined linear time depending on the size of the hyperstream and the size of the query (see Fig. 5)

Furthermore, for hyperstreams without compression, pruning strong certainty is equivalent to strong certainty. Therefore, it provides a conservative extension on general hyperstreams of the previous approximation of certain answers for path queries on streams.

Outline. After some preliminaries in Section 2, we recall hyperstreams in Section 3 and certain (non-)answers on hyperstreams in Section 4. We introduce and discuss the notions of linear certainty of (non-)answers in Section 5 and of strong certainty in Section 6. We propose the notion of pruning strong certainty in Section 7. Finally, the power of (pruning) strong certainty for NFAs obtained from path queries compared to previous approximations of certainty for path queries by "early NFAs" is studied in Section 8.

2 PRELIMINARIES

We recall the notions of strings, string patterns, streams, then monoids, automata, and queries on words, and finally certain query answering on streams.

2.1 Words, Strings, String Patterns

Let $\mathbb{N}$ be the set of naturals without zero. A *word* w over a set D is some sequence $w = d_1 \dots d_n \in D^n$ where $n \geq 0$. The length w is the number of its letters $|w| = n$. The set of positions of w is $pos(w) = \{1, \dots, |w|\}$. For any position $j \in pos(w)$ we write $w[j] = d_j$ for the j'th letter of w. The set of all words over D is denoted by D^*, the empty word by ϵ, and the concatenation function on words over D by $\cdot : D^* \times D^* \to D^*$.

We fix an infinite set of variables $\mathcal{Y}$ which elements range over $Y, Y_1, Y_2, \dots$. Let Σ be a finite alphabet with at least 2 elements, ranged over by $a, b, \dots$. A word over Σ is called a *string* and a word over $\Sigma \cup \mathcal{Y}$ a *string pattern*. We will write Pat_Σ for the set of all string patterns. The set of *free variables* of p, that is the variables occurring in p, is denoted by $V(p)$. A string pattern p is *linear* if every variable occurs at most once in p. The *size* $|p|$ of a string pattern p is the length of p.

A (ground) substitution $\sigma = [Y_1/w_1, \dots, Y_n/w_n]$ is a partial function with domain $dom(\sigma) = \{Y_1, \dots, Y_n\} \subseteq \mathcal{Y}$ and $\sigma(Y_i) = w_i \in \Sigma^*$ for all $1 \leq i \leq n$. We write $p\sigma$ for the string pattern obtained by replacing all occurrences of variables Y of $dom(\sigma)$ in p by $\sigma(y)$, so that $w\sigma = w$ for all strings $w \in \Sigma^*$, $y\sigma = \sigma(y)$ for all variables $y \in dom(\sigma)$, $y\sigma = y$ for all variables $y \in \mathcal{Y} \setminus dom(\sigma)$, and $(p \cdot p')\sigma = p\sigma \cdot p'\sigma$ for all patterns $p, p' \in Pat_\Sigma$.

An *instance of a string pattern* p is a string obtained by replacing all variables in p by some string, and concatenating the result. We denote the set of all instances of the pattern p by:

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

A *hyperstream* is a compressed string pattern, whose precise definition will be recalled later on in Section 3.

2.2 Monoids and Transitions

We will base our definitions and algorithms for finite automata on the notions of monoids and transitions.

A *monoid* is an algebra $(M, \cdot, 1)$ such that M is a set, $\cdot : M \to M$ is associative and has $1 \in M$ as neutral element. Most typically, the set of strings can be turned into the monoid $(\Sigma^*, \cdot, \epsilon)$, and similarly, the set of string patterns becomes a monoid $(Pat_\Sigma, \cdot, \epsilon)$.

A *transition on a set* Q is a binary relation in $Q \times Q$. We write T_Q for the set of transitions of Q. The set of transitions defines the monoid $(T_Q, \circ, \{(q, q) \mid q \in Q\})$ where $\circ$ is the composition operator of binary relations.

For any finite set Q we write $|Q|$ for its cardinality. In particular, if $\tau \in T_Q$ is a transition over a finite set Q, then we write $|\tau|$ for the number of pairs in τ. Given a transition $\tau \in T_Q$ and a subset $Q' \subseteq Q$, we define $\tau(Q') = \{q \in Q \mid \exists q' \in Q'.\ (q', q) \in \tau\}$. Note that $\tau(Q')$ can be computed in time $O(|\tau| + |Q'|)$ from τ and Q'. Furthermore, given two transitions $\tau, \tau' \in T_Q$ we can reduce the problem to compute $\tau \circ \tau'$ to the multiplication of two Boolean $|Q| \times |Q|$ matrices.

Let $(M, \cdot, 1)$ and $(M', \cdot', 1')$ be two monoids. A *morphism* between these monoids is a mapping $\mu : M \to M'$ such that $\mu(m_1 \cdot m_2) = \mu(m_1) \cdot' \mu(m_2)$ for all $m_1, m_2 \in M$ and $\mu(1) = 1'$. For any mapping $\mu' : \Sigma \to M$ there exists a unique morphism $\mu : \Sigma^* \to M$ such that $\mu(a) = \mu'(a)$. Furthermore, for any substitution $\sigma : \mathcal{Y} \to M$ and morphism $\mu : \Sigma^* \to M$ there exists a unique morphism $eval_{\sigma,\mu} : Pat_\Sigma \to M$ such that $eval_{\sigma,\mu}(Y) = \sigma(Y)$ for all $Y \in \mathcal{Y}$ and $eval_{\sigma,\mu}(w) = \mu(w)$ for all $w \in \Sigma^*$; such morphism is called *evaluator* in the sequel.

2.3 Automata and Queries

We will use finite automata to define Boolean queries on strings. A Boolean query on strings is nothing else than a string language $L \subseteq \Sigma^*$. In database terminology, we say that the query L selects the empty tuple on a string w if and only if $w \in L$.

A *nondeterministic finite automaton* or equivalently an NFA with alphabet Σ is a tuple $A = (Q, \Sigma, \delta, I, F)$ where Q is a finite set of states, $I, F \subseteq Q$ are respectively sets of initial and final states, and $\delta \subseteq Q \times \Sigma \times Q$ a transition relation. For sets $I' \subseteq Q$ and $F' \subseteq Q$, we denote by $A[I/I', F/F']$ the automaton $(Q, \Sigma, \delta, I', F')$. An NFA is called *deterministic* or equivalently a DFA if δ forms a partial function from $Q \times \Sigma$ to Q and I is a singleton. An NFA is said *complete* if for all $(q, a) \in Q \times \Sigma$, there exists $q' \in Q$ such that $(q, a, q') \in \delta$.

We define the size of A by $|A| = |Q| + |\delta|$. The transition relation can be lifted to a unique morphism $trans_\delta : \Sigma^* \to T_Q$ such that $trans_\delta(a) = \{(q, q') \mid (q, a, q') \in \delta\}$ for all $a \in \Sigma$. The language recognized by A can then be defined as follows:

$$L(A) = \{w \in \Sigma^* \mid trans_\delta(w) \cap (I \times F) \neq \emptyset\}$$

It is well-known that whether $w \in L(A)$ can be decided in time $O(|w||A|)$. For this, we need to avoid to compute $trans_\delta(w)$ that could be done by composing the transitions of the letters of w based on boolean matrix multiplication in time $O(|w||Q|^{2.18})$, but cannot be done in combined linear time (except if A was deterministic). Instead, we can compute $trans_\delta(w)(I)$. If $w = a_1 \dots a_n$ and $\tau_i = trans_\delta(a_i)$ for all $1 \leq i \leq n$ then $trans_\delta(w)(I) = \tau_n(\dots \tau_1(I) \dots)$. This set can indeed be computed in time $O(|w||\delta|)$. Testing whether the intersection of this set with F is nonempty requires time $O(|Q|)$ so the overall time in for membership testing is in $O(|w||A|)$.

For any transition relation $\delta \subseteq Q \times \Sigma \times Q$ and substitution $\sigma : \mathcal{Y} \to T_Q$, we define the evaluator $eval_{\sigma, trans_\delta} : Pat_\Sigma \to T_Q$ as the unique morphism that extends on the morphism $trans_\delta : \Sigma^* \to T_Q$. Note that $eval_{\sigma, trans_\delta}(w) = trans_\delta(w)$ for all strings $w \in \Sigma^*$. Moreover, we define the *accessibility transition* $acc_\delta \in T_Q$ by:

$$acc_\delta = \bigcup_{w \in \Sigma^*} trans_\delta(w)$$

A state $q \in Q$ is called a *selection state of A* if $acc_\delta(\{q\}) \subseteq F$. Note that the sets $acc_\delta(Q')$ and $acc_\delta^{-1}(Q')$ can be computed from δ in time $O(|\delta| + |Q|)$ for all subset $Q' \subseteq Q$, while the full accessibility transition acc_δ can be computed from δ in time $O(Q^3)$.

3 HYPERSTREAMS

We recall that a singleton context-free grammar (sCFGs) is a context-free grammar whose language contains exactly a single word.

Definition 1. *A* hyperstream *G over an alphabet Σ is an acyclic CFG (N, Σ, R, S) where $N \subsetneq \mathcal{Y}$ is a finite set of non-terminals, the* ruling function *R is a partial function from N to words in $(N \cup \Sigma)^*$ and $S \in N$ is the start symbol.*

The acyclicity criterion in the definition of a hyperstream means that the binary relation $>_G = \{(Y, Z) \mid Y \in dom(R) \text{ and } Z \in V(R(Y))\}$ is acyclic.

A *bound variable* in a hyperstream $G = (N, \Sigma, R, S)$ is a non-terminal that has a rule associated to it; thus $dom(R)$ is the set of bound variables of G. A *free variable* of G is a non-terminal that has no associated rule. We denote by $V(h)$ the set of free variables of h, that is $V(h) = N \setminus dom(R)$.

For any hyperstream $G = (N, \Sigma, R, S)$, we define its pattern $pat(G)$ as the only word over $\Sigma \cup V(G)$ in the language of the hyperstream $G' = (N \setminus V(G), \Sigma \cup V(G), R, S)$. The string pattern $pat(G)$ can be obtained by recursively replacing in $R(S)$ each bound variable $Y \in dom(R)$ by $R(Y)$. This definition is well-founded, since hyperstreams are acyclic. Remark that a hyperstream without free variables is a sCFGs and its pattern is a string in Σ^*.

The size $|G|$ of the hyperstream G is given by $|G| = |N| + \sum_{Y \in dom(R)} |R(Y)|$. Thus, a hyperstream may describe an exponentially larger string pattern.

Example 1. For instance, the hyperstream $G_1 = (N_1, \Sigma, R_1, Y_1)$ where $N_1 = \{Y_1, \ldots, Y_n\}$, $R(Y_i) = Y_{i+1}Y_{i+1}$ for all $1 \leq i \leq n-1$ and $R(Y_n) = s \in Pat_\Sigma$, describes the string pattern $pat(G_1) = s^{(2^n)}$.

Notice that string patterns can be seen as special cases of hyperstreams. For any string pattern p, the hyperstream G_p defined here after satisfies $pat(G_p) = p$ and $|G_p| = O(|p|)$. So, $G_p = (V(p) \cup \{Y\}, \Sigma, R, Y)$, where Y is a variable that does not belong to $V(p)$ and $dom(R) = \{Y\}$, and $R(Y) = p$.

Let $G = (N, \Sigma, R, S)$ be a hyperstream. The set of positions of G, denoted by $Pos(G)$, is defined by:

$$Pos(G) = \{(Y, i) \mid Y \in dom(R) \text{ and } i \in pos(R(Y))\}.$$

For any position $(Y, j) \in Pos(G)$ we define $letter_G(Y, j) \in \Sigma \cup \mathcal{Y}$ by $R(Y)[j]$.

We define the set $Inst(G)$ of instances of the hyperstream G as the set of instances of its pattern, i.e., $Inst(G) = Inst(pat(G))$.

We finally introduce linear and compression-free hyperstreams that are restrictions considered in our complexity results.

Definition 2. *A hyperstream h is called* linear *if $pat(h)$ is linear.*

Definition 3. *A hyperstream G with set of non-terminals N is called* compression-free *if it contains no two positions $v \neq v' \in pos(G)$ with the same non-terminal $letter_G(v) = letter_G(v') \in N$.*

4 CERTAIN (NON-)ANSWERS

We recall the notions of certain query answers and non-answers for Boolean queries on hyperstreams from Sakho et al. [19], as well as the complexity of certain query answering.

Definition 4. *Let A be an Nfa. A hyperstream G is called*

- *a* certain query answer *for $\mathbf{Q}(A)$ if all its instances belong to $L(A)$, i.e. $Inst(G) \subseteq L(A)$,*
- *a* certain query non-answer *for $\mathbf{Q}(A)$ if none of its instances belongs to $L(A)$, i.e. $Inst(G) \cap L(A) = \emptyset$.*

Theorem 2 (Sakho et al. [19]). *It is PSpace-complete to decide whether a hyperstream G is a certain query answer (resp. non-answer) for the query $\mathbf{Q}(A)$ of an Nfa A.*

Let's illustrate the PSpace-hardness in the case of certain query non-answers for A being a Dfa. Consider a sequence of Dfas A_1 ... A_n and let # be a new symbol not in Σ. Then $L(A_1) \cap \ldots \cap L(A_n) = \emptyset$ if and only if the string pattern $y\# \ldots \#y$ is a certain query non-answer for the query of the Dfa that recognizes $\{w_1\# \ldots \#w_n \mid w_1 \in L(A_1), \ldots, w_n \in L(A_n)\}$. Therefore, deciding whether a hyperstream is a certain query non-answer for the query of a Dfa can be reduced in PTime to testing the emptiness of the intersection of a sequence of Dfas, which has been shown to be PSpace-complete in [11].

5 LINEARLY CERTAIN (NON-)ANSWERS

As illustrated by the proof of Theorem 2, even for a very simple non-linear hyperstream it is hard to decide whether it is a certain (non-)answer. We therefore propose a weaker notion of certain answers and non-answers in order to make a first step towards an efficient algorithm. The idea is to ignore nonlinearities of hyperstreams (sharing of free variables in its string pattern), while still accounting for compression (sharing of non-terminals). We first define linearly certain (non-)answers and show in Proposition 1 that this is a sound approximation. We next establish in Corollary 1 that testing whether a hyperstream is a linearly certain non-answer is tractable for queries defined by Nfas. Linearly certainty of query answers however remains hard in the case of Nfas (Theorem 3), but becomes tractable for queries defined by Dfas (Proposition 3).

For any pattern p, let $lin(p)$ be a linear pattern obtained from p by replacing all occurrences of variables in p by fresh variables in some fixed order.

Definition 5. *Let A be an Nfa. We call the hyperstream G a* linearly certain query answer (resp. non-answer) *for $\mathbf{Q}(A)$ if $lin(pat(G))$ is a certain query answer (resp. non-answer) for $\mathbf{Q}(A)$.*

Proposition 1 (Soundness). *If G is a linearly certain query answer (resp. non-answer) for $\mathbf{Q}(A)$, then G is certain query answer (resp. non-answer) for $\mathbf{Q}(A)$.*

Proof. Clearly, $Inst(pat(G)) \subseteq Inst(lin(pat(G)))$. If G is a linearly certain query non-answer, then $Inst(lin(pat(G))) \cap L(A) \neq \emptyset$, so that $Inst(pat(G)) \cap L(A) \neq \emptyset$ and so G is a certain query non-answer. If G is a linearly certain query answer, so $Inst(lin(pat(G))) \subseteq L(A)$, thus $Inst(pat(G)) \subseteq L(A)$, hence G is certain query answer. □

5.1 Tractability for Non-Answers and NFAs

Tractability of linear certainty of non-answers is shown using a characterization in terms of an appropriate evaluator. Let $acc_\delta : \mathcal{Y} \to T_Q$ be the constant substitution function that maps all variables to acc_δ. Then the *non-answers evaluator* $\bar{e}_\delta$: $Pat_\Sigma \to T_Q$ defined by $\bar{e}_\delta = eval_{cacc_\delta, trans_\delta}$ satisfies

PROPOSITION 2. *Let $A = (Q, \Sigma, \delta, I, F)$ be an NFA. A hyperstream G is a linearly certain non-answer for $\boldsymbol{Q}(A)$ if and only if*

$$\bar{e}_\delta\ (pat(G)) \cap (I \times F) = \emptyset.$$

PROOF. Let the transition $lintrans_\delta(p) = \bigcup_{w \in Inst(lin(p))} trans_\delta(w)$. We first show that G is a linearly certain non-answer for $\mathbf{Q}(A)$ if and only if $lintrans_\delta(pat(G)) \cap (I \times F) = \emptyset$:

G is a linearly certain non-answer for $\mathbf{Q}(A)$
$\Leftrightarrow$ $Inst(lin(pat(G))) \cap L(A) = \emptyset$ by Definitions 4 and 5
$\Leftrightarrow$ $\forall w \in Inst(lin(pat(G)))$. $trans_\delta(w) \cap (I \times F) = \emptyset$
$\Leftrightarrow$ $lintrans_\delta(pat(G)) \cap (I \times F) = \emptyset$ (by definition of $lintrans_\delta$)

Now we show that for all $p \in Pat_\Sigma$: $lintrans_\delta(p) = \bar{e}_\delta\ (p)$. By induction on the structure of p, there are 4 cases:

Case $p = \epsilon$. Then $Inst(lin(p)) = \{\epsilon\}$ and

$$lintrans_\delta(p) = trans_\delta(\epsilon) = \bar{e}_\delta\ (\epsilon).$$

Case $p = a$ for some $a \in \Sigma$. In this case, $Inst(lin(p)) = \{a\}$ and

$$lintrans_\delta(p) = trans_\delta(a) = \bar{e}_\delta\ (a).$$

Case $p = y$ for some $y \in \mathcal{Y}$. So $Inst(lin(p)) = \Sigma^*$ and

$$lintrans_\delta(p) = \bigcup_{w \in \Sigma^*} trans_\delta(w) = acc_\delta = \bar{e}_\delta\ (y)$$

Case $p = p_1 p_2$ for smaller patterns $p_1, p_2 \in Pat_\Sigma$. Let $lin(p_1) = p_1'$ and $lin(p_2) = p_2'$. By definition,

$$Inst(lin(p)) = \{u \cdot v \mid u \in Inst(p_1') \text{ and } v \in Inst(p_2')\}.$$

So we have that:

$$lintrans_\delta(p) = \bigcup_{u \in Inst(p_1'), v \in Inst(p_2')} trans_\delta(u \cdot v) = lintrans_\delta(p_1) \circ lintrans_\delta(p_2)$$

By the inductive hypothesis, we assume that $lintrans_\delta(p_1) = \bar{e}_\delta\ (p_1)$ and $lintrans_\delta(p_2) = \bar{e}_\delta\ (p_2)$. Therefore

$$lintrans_\delta(p) = \bar{e}_\delta\ (p_1) \circ \bar{e}_\delta\ (p_2) = \bar{e}_\delta\ (p).$$

Hence $lintrans_\delta(p) = \bar{e}_\delta\ (p)$ for all $p \in Pat_\Sigma$. □

Using the non-answer evaluator we obtain the following tractability result.

COROLLARY 1. *Given a hyperstream $G \in \mathcal{H}$ and a query NFA A over Σ, whether G is a linearly certain non-answer for $Q(A)$ can be decided in PTIME.*

PROOF. By Proposition 2, it is enough to decide whether $\bar{e}_\delta\ (pat(G)) \cap (I \times F) = \emptyset$. Using Boolean square matrices of size $|Q|$ as representations of transitions, we can compute $\bar{e}_\delta\ (pat(G))$ in the required time, since matrix multiplications is in PTIME. For this, we first replace in G any letter $a \in \Sigma$ by its transition $trans_\delta(a)$ and any free variable $Y \in V(G)$ by the accessibility transition acc_δ. Then, let $\leq_G$ be a total order extension of the reverse of the partial order $>_G$, that is for $Y, Y' \in dom(R)$, $Y \leq_G Y'$ if and only if either $Y' >_G^+ Y$, or $(Y', Y) \notin >_G$. So for two bound variables $Y, Y' \in dom(R)$, if $Y \leq_G Y'$ then there is no way that Y' appears in $R(Y)$. Finally, for all $Y \in dom(R)$, we recursively compute $trans_\delta(R(Y))$ in the order defined by $\leq_G$. For some bound variable $Y \in dom(R)$, this is done by multiplying $|R(Y) - 1|$ times the boolean matrices that replacing the letters and non-terminals appearing in $R(Y)$. Note that for any bound variable $Y' \in dom(R)$ appearing in $R(Y)$, we replace the occurrences of Y' in $R(Y)$ by the $trans_\delta(R(Y'))$ transition that is already computed, since $Y' \leq_G Y$. When we finally get to the start symbol S of G, we have computed $trans_\delta(S) = \bar{e}_\delta\ (pat(G))$. The intersection with $I \times F$ is done in time $O(|Q|)$. The overall time of this computation is in PTIME. □

5.2 Intractability for Answers and NFAs

Unfortunately the situation is harder for linearly certain answers of NFA defined queries.

THEOREM 3. *The problem of whether a hyperstream is a linearly certain query answer for a query defined by an NFA is PSPACE-complete. The same holds for string patterns, i.e. for compression-free hyperstreams.*

PROOF. For PSPACE-hardness, note that the string pattern Y is linearly certain answer for the NFA A if and only if A is universal.

Deciding whether G is a linearly certain answer in PSPACE needs some care. To see this, let $G \in \mathcal{H}_\Sigma$ be a hyperstream. The set $Inst(lin(pat(G)))$ is regular, but may not be recognizable by an NFA of polynomial size in $|G|$, so one cannot hope for a PTIME reduction to the inclusion problem of NFAs. Furthermore, adapting the PSPACE algorithm from [19] for deciding whether a hyperstream is a certain query answer would lead to a DEXPTIME algorithm for testing whether G is a linearly certain answer, so this approach fails as well.

The approach that works is to generate an instance of $lin(pat(G))$ nondeterministically and evaluate this instance by A with on-the fly-determinization in a streaming manner. If the evaluation does not yield any final state, so we have found an instance of $lin(pat(h))$ that does not belong to $L(A)$, so G is not a linearly certain answer. This nondeterministic algorithm is in PSPACE. □

5.3 Tractability for Answers and DFAs

In the case of DFAs, the situation is simpler than for NFAs in that linear certainty of answers can be reduced to linear certainty of non-answers.

LEMMA 1. *For any complete DFA $A = (Q, \Sigma, \delta, I, F)$, a hyperstream is a linearly certain query answer for $\boldsymbol{Q}(A)$ if and only if it is a linearly certain query non-answer for $\boldsymbol{Q}(A[F/(Q \setminus F)])$.*

PROOF. Let $A = (Q, \Sigma, \delta, I, F)$ be a complete DFA and $\overline{A}$ its complement, that is $\overline{A} = A[F/(Q \setminus F]$. We then have that $\overline{L(A)} =$

$\Sigma^* \setminus L(A) = L(\overline{A})$. Therefore, we have for any hyperstream $G \in \mathcal{H}_\Sigma$:

G is a linearly certain answer for A	
$\Leftrightarrow Inst(lin(pat(G))) \subseteq L(A)$	by Definition 5
$\Leftrightarrow Inst(lin(pat(G))) \cap \overline{L(A)} = \emptyset$	elementary
$\Leftrightarrow Inst(lin(pat(G))) \cap L(\overline{A}) = \emptyset$	A is complete Dfa
$\Leftrightarrow G$ is a linearly certain non-answer for $\overline{A}$	by Definition 5

□

Proposition 3. *Whether a hyperstream is a linearly certain query answer for a query defined by a Dfa can be decided in PTime.*

Proof. Let $A = (Q, \Sigma, \delta, I, F)$ be a Dfa and G a hyperstream. Let A' be the completion of A. Then G is a certain answer for $\mathbf{Q}(A)$ iff it is a certain answer for $\mathbf{Q}(A')$, which by Lemma 1 is equivalent that G is linearly certain non-answer for $\mathbf{Q}(A'[F/(Q \setminus F)])$. The Dfa $A'[F/(Q \setminus F)]$ can be built in PTime from A and deciding whether G is linearly certain non-answer for $\mathbf{Q}(A'[F/(Q \setminus F)])$ is in PTime by Corollary 1. □

6 STRONGLY CERTAIN ANSWERS

As we saw, linear certainty is an approximation of certain query (non-)answers and it is tractable for certain non-answers and for certain answers for queries defined by Dfas. We now introduce strong certainty as a further approximation of certain answers that is equivalent to linear certainty for Dfas, and is tractable for compression-free hyperstreams for Nfas. The idea behind strong certainty is to test whether for all instances of the linearization of the hyperstream there exists a run of the automaton that ends in a final state. Note that this is a property that depends on the automaton representing the query, which is in contrast with the previous notions of certainty that only depended on the query itself.

6.1 Safety Approximations

A key parameter of the notion of strong certainty will be an approximation of the notion of safe states of an automaton. As introduced in [5, 7], a safe state for a stream wY is a state s.t. whenever reached after reading w it will allow to accept all possible instantiations of Y. For hyperstreams this notion needs to be generalized because variables can occur in the middle of a pattern. The general idea is as follows. Given a pattern pYp', suppose that we know a set of states $Q_{p'}$ s.t. from any state in $Q_{p'}$ we can read (all instantiations of) p' and reach a final state. The question is then to find a set of states $Q_{Yp'}$ s.t. from any state in that set we can read (all instantiations of) Yp' and reach a final state. Because Y is a variable, any q in $Q_{Yp'}$ must be such that from q we can read all words in Σ^* and arrive in a state in $Q_{p'}$. We say then that q is safe for $Q_{p'}$.

For any Nfa $A = (Q, \Sigma, \delta, I, F)$, function $safe_\delta : 2^Q \to 2^Q$ associates with any $Q' \subseteq Q$ the set of states safe for Q':

$$safe_\delta(Q') = \{q \in Q \mid \forall w \in \Sigma^*, trans_\delta(w) \cap (\{q\} \times Q') \neq \emptyset\}.$$

Safety is closely related to universality of automata: for any Nfa $A = (Q, \Sigma, \delta, I, F)$ and any subset of states $Q' \subseteq Q$ it follows from the definitions that $q \in safe_\delta(Q')$ if and only if $L(A[I/\{q\}, F/Q']) = \Sigma^*$. Therefore we cannot hope to compute the set of safe states efficiently for Nfas. This is different in the case of Dfas as already noticed in [5, 7]. For any subset $Q' \subseteq Q$ let:

$$safe\text{-}acc_\delta(Q') = \{q \in Q \mid acc_\delta(\{q\}) \subseteq Q'\}.$$

It is then easy to see that $safe_\delta = safe\text{-}acc_\delta$ for any complete Dfa, and that the set $safe\text{-}acc_\delta(Q')$ can be computed in PTime[1], and even in linear time.

Lemma 2. *For any transition relation $\delta \subseteq Q \times \Sigma \times Q$ and set of states $Q' \subseteq Q$, the set $safe\text{-}acc_\delta(Q')$ can be computed in time $O(|\delta| + |Q|)$.*

Proof. This follows from that $safe\text{-}acc_\delta(Q') = Q \setminus acc_\delta^{-1}(Q \setminus Q')$. Therefore, it is sufficient to complement the set of inversely accessible states starting from the complement of Q'. □

In the case of Nfas, we will use various approximations of the function $safe_\delta(Q')$ that can be computed efficiently.

Definition 6 (safety approximation). *Let $\delta \subseteq Q \times \Sigma \times Q$ be a transition relation. A function $s_\delta : 2^Q \to 2^Q$ is called a* safety approximation *if for any set of states $Q' \subseteq Q$, it holds $s_\delta(Q') \subseteq safe_\delta(Q')$.*

As shown by the next lemma, $safe\text{-}acc_\delta$ is a safety approximation for all transition relation δ of complete Nfas.

Lemma 3. *For any complete Nfa $A = (Q, \Sigma, \delta, I, F)$, $safe\text{-}acc_\delta$ is a safety approximation.*

Proof. Let δ be the transition function of the complete Nfa A. We have to show for any $q \in safe\text{-}acc_\delta(Q')$ and for any $w \in \Sigma^*$ that $trans_\delta(w) \cap (\{q\} \times Q') \neq \emptyset$. Given that the A is complete, we have that $trans_\delta(w)(\{q\}) \neq \emptyset$. Since $q \in safe\text{-}acc_\delta(Q')$ it follows that $trans_\delta(w)(\{q\}) \subseteq Q'$. Hence $trans_\delta(w) \cap (\{q\} \times Q') \neq \emptyset$ as required. □

A slightly tighter safety approximation will be introduced in Section 6.4, leading to our final notion of strong certainty.

6.2 Parameterized Strong Certainty

We introduce the notion of s strong certainty, which is parameterized by a function s that maps transition relations δ to safety approximations s_δ. The definition will be based on an appropriate evaluator for hyperstreams. We then show that s strong certainty is sound in that all s strongly certain answers are linearly certain.

Let $\delta \subseteq Q \times \Sigma \times Q$ be a transition relation and s_δ a safety approximation. We define $itrans_\delta : \Sigma^* \to (2^Q \to 2^Q)$ such that

[1] Gauwin et. al. [5, 7] introduced a similar notion of safe states for NWAs, that they called safe states for selection. The set of all such states is denoted by $safe\_sel^A$. They used it to compute the set of certain answers for queries on streams defined by *deterministic* NWAs. Restricted to the case of Nfas $A = (Q, \Sigma, \delta, I, F)$ on words, this set satisfies $safe\_sel^A = safe_\delta(F)$. They also showed that $safe\_sel^A$ can be computed efficiently for deterministic NWAs A based on its accessibility relation. This result restricted to words can be restated in our terminology as $safe_\delta(F) = safe\text{-}acc_\delta(F)$ for all Dfas (but not for all Nfas). Computing the set $safe\_sel^A$ is intractable for automaton with nondeterminism, given that it requires to decide universality problems. Therefore, it was not clear whether this notion could be used to approximate the set of certain answers on streams for NWA queries. In order to circumvent this problem, an alternative approximation was proposed which is restricted to path queries [3] and relies on their compilation to early NWAs. As we will see in Section 8, in the case of words, an equivalent approximation can be obtained by compilation to Nfas by using our notion of strong certainty with respect to $safe\text{-}acc_\delta$, which is sound since $safe_\delta(F) \supseteq safe\text{-}acc_\delta(F)$. Beside of being of equal quality for path queries on streams, this new approximation has the advantage to be sufficiently general, so that it can be applied to Nfa queries on hyperstreams without any restriction.

for all strings $w \in \Sigma^*$, $itrans_\delta(w)$ is the function such that for any state set $Q' \subseteq Q$:

$$itrans_\delta(w)(Q') = trans_\delta(w)^{-1}(Q').$$

In other words, $itrans_\delta(w)(Q')$ contains all states q such that there *exists* a run with transition relation δ that starts in q and ends in Q'. Note that the space of functions $(2^Q \to 2^Q, \circ, \{(Q', Q') \mid Q' \subseteq Q\})$ forms a monoid, where $\circ$ is the composition operator of relations in $2^Q \times 2^Q$ restricted to total functions, so that $(f \circ f')(Q') = f(f'(Q'))$. It is easy to see that $itrans_\delta$ is a homomorphism from the monoid of strings into this monoid of functions.

We define the constant variable assignment $const_{s_\delta} : 2^Q \to 2^Q$ by $const_{s_\delta}(Y) = s_\delta$ for any $Y \in \mathcal{Y}$. Finally, we can define the evaluator on string patterns:

$$e_{s,\delta} = eval_{const_{s_\delta}, itrans_\delta}.$$

Definition 7 (s strongly certain query answer). *Let $A = (Q, \Sigma, \delta, I, F)$ be an NFA and s_δ a safety approximation. We call a hyperstream G a s* strongly certain query answer *for A if A is complete and $e_{s,\delta}(pat(G))(F) \cap I \neq \emptyset$.*

Intuitively, the evaluator $e_{s,\delta}$ propagates the safe states from right to left of the $pat(G)$, starting with the set of final states and aiming to reach at least one initial state. For $s = safe\text{-}acc$ and the hyperstream G from Fig. 1, the transition relation of the NFA $A = (Q, \Sigma, \delta, I, F)$ in Fig. 6, a computation of $e_{s,\delta}(pat(G))(F)$ is depicted in Fig. 7.

Example 4. Let G' be the hyperstream obtained from the hyperstream in Fig. 1 by replacing all the occurrences of the Y free variable and the first occurrence of the Z free variable by the empty word ϵ, and by erasing the trailing a letter. Its string pattern, which is actually a stream, is $pat(G') = acabbacZ$. We consider the NFA $A = (Q, \Sigma, \delta, I, F)$ in Fig. 6 which defines the path query from the introduction. So G' is a *safe-acc* strongly certain answer for A, since

$$\begin{aligned} e_{safe\text{-}acc,\delta}(acabbacZ)(\{q_5\}) &= itrans_\delta(acabbac(safe\text{-}acc_\delta(\{q_5\}))) \\ &= itrans_\delta(acabbac(\{q_5\})) \\ &= \{q_0, q_2, q_4, q_5\} \end{aligned}$$

and $\{q_0, q_2, q_4, q_5\} \cap I = \{q_0\} \neq \emptyset$.

We next notice that nonlinearities are irrelevant for s strongly certainty.

Lemma 4. *Let $\delta \subseteq Q \times \Sigma \times Q$ a transition relation and s_δ be a safety approximation. Then for any pattern $p \in Pat_\Sigma$: $e_{s,\delta}(p) = e_{s,\delta}(lin(p))$.*

Proof. The value of $e_{s,\delta}(p)$ does not depend on which free variables are used in p, since all free variables $Y \in V(p)$ are mapped constantly to s during the evaluation. Therefore, we can freely replace all free variables of p by fresh variables as for converting p into $lin(p)$ without changing the value of $e_{s,\delta}$. Thus $e_{s,\delta}(p) = e_{s,\delta}(lin(p))$. □

Soundness. We now show that *safe-acc* strongly certain query answers are also linearly certain.

Lemma 5. *Let $A = (Q, \Sigma, \delta, I, F)$ be a complete NFA and s_δ a safety approximation. For any string pattern $p \in Pat_\Sigma$ and any $Q', Q'' \subseteq Q$, if $e_{s,\delta}(p)(Q'') \cap Q' \neq \emptyset$, then $Inst(p) \subseteq L(A[I/Q', F/Q''])$.*

Proof. Let $p \in Pat_\Sigma$ and $Q', Q'' \subseteq Q$ s.t. $e_{s,\delta}(p)(Q'') \cap Q' \neq \emptyset$. Let $\mathcal{L} = L(A[I/Q', F/Q''])$, that is, $\mathcal{L} = \{w \in \Sigma^* \mid itrans_\delta(w) \cap (Q'' \times Q') \neq \emptyset)\}$. The proof of the lemma is by induction on the structure of p. There are 4 cases:

- **Case $p = \epsilon$.** We then have $Inst(p) = \{\epsilon\}$ and $e_{s,\delta}(p)(Q'') = itrans_\delta(\epsilon)(Q'') = Q''$. So if $Q'' \cap Q' \neq \emptyset$, then $\epsilon \in \mathcal{L}$, that is $Inst(p) \subseteq \mathcal{L}$.
- **Case $p = a$ for some $a \in \Sigma$.** So $Inst(p) = \{a\}$ and furthermore $e_{s,\delta}(p)(Q'') = itrans_\delta(a)(Q'')$. So if $itrans_\delta(a)(Q'') \cap Q' \neq \emptyset$, then $a \in \mathcal{L}$, that is $Inst(p) \subseteq \mathcal{L}$.
- **Case $p = Y$ for some $Y \in \mathcal{Y}$.** In this case, $Inst(p) = \Sigma^*$ and $e_\delta(p)(Q'') = s_\delta(Q'')$. Let $q \in s_\delta(Q'') \cap Q' \neq \emptyset$, then $L(A[I/\{q\}, F/Q'']) = \Sigma^*$, so $\mathcal{L} = \Sigma^*$ as well. Hence $Inst(p) \subseteq \mathcal{L}$.
- **Case $p = p_1 p_2$ for some smaller patterns $p_1, p_2 \in Pat_\Sigma$.** We have that $Inst(p) = \{uv \in \Sigma^* \mid u \in Inst(p_1)$ and $v \in Inst(p_2)\}$ and $e_{s,\delta}(p)(Q'') = [e_{s,\delta}(p_1) \circ e_{s,\delta}(p_2)](Q'')$ since $e_{s,\delta}$ is a homomorphism. Now, $e_{s,\delta}(p)(Q'') \cap Q' \neq \emptyset$ implies that there exists $P \subseteq Q$ s.t. $e_{s,\delta}(p_2)(Q'') \cap P \neq \emptyset$ and $e_{s,\delta}(p_1)(P) \cap Q' \neq \emptyset$. By the induction hypothesis it follows that $Inst(p_1) \subseteq L(A[I/Q', F/P])$ and $Inst(p_2) \subseteq L(A[I/P, F/Q''])$, thus $Inst(p) \subseteq \mathcal{L}$.

□

Proposition 4 (Soundness). *Let $A = (Q, \Sigma, \delta, I, F)$ be an NFA and s_δ a safety approximation. Then any hyperstream that is a safe-acc strongly certain answer for A is linearly certain for $\mathbf{Q}(A)$.*

Proof. Let $A = (Q, \Sigma, \delta, I, F)$ be a complete NFA and $G \in \mathcal{H}_\Sigma$ a hyperstream. Then G is a *safe-acc* strongly certain answer for A if:

	$e_{s,\delta}(pat(G))(F) \cap I \neq \emptyset$	by Definition 7
$\Leftrightarrow$	$e_{s,\delta}(lin(pat(G)))(F) \cap I \neq \emptyset$	by Lemma 4
$\Rightarrow$	$Inst(lin(pat(G))) \subseteq L(A)$	by Lemma 5
$\Leftrightarrow$	G is a linearly certain answer for $\mathbf{Q}(A)$	by Definition 5

□

6.3 Safe-Acc Strong Certainty

We next study the variant of strong certainty that is parameterized by the safety approximation *safe-acc*.

Completeness for DFAs. We next show that all linearly certain answers of DFA queries are *safe-acc* strongly certain for the DFA, while the generalization to NFAs fails. This shows that the notion of *safe-acc* strongly certain answers is highly dependent on NFA determinization.

Proposition 5 (Completeness). *Let $A = (Q, \Sigma, \delta, I, F)$ be a complete DFA and G a hyperstream. Then G is a safe-acc strongly certain answer for A if and only if G is a linearly certain answer for $\mathbf{Q}(A)$.*

Proof. Let $A = (Q, \Sigma, \delta, I, F)$ be a complete DFA where $I = \{q_0\}$ and G a hyperstream. The forward direction has been already proved by Proposition 4 and holds even for the general case of NFAs.

For the inverse direction, the proof is by induction on the structure of the pattern of h.

Case $pat(G) = \epsilon$. G a is linearly certain answer for $\mathbf{Q}(A) \Rightarrow \epsilon \in L(A) \Rightarrow trans_\delta(\epsilon)(\{q_0\}) \in F \Rightarrow itrans_\delta(\epsilon)(F) \cap \{q_0\} \neq \emptyset \Rightarrow h$ is a *safe-acc* strongly certain answer for $\mathbf{Q}(A)$.

Case $pat(G) = a \in \Sigma$. G is a linearly certain answer for $\mathbf{Q}(A) \Rightarrow a \in L(A) \Rightarrow trans_\delta(a)(\{q_0\}) \in F \Rightarrow itrans_\delta(a)(F) \cap \{q_0\} \neq \emptyset \Rightarrow G$ is a *safe-acc* strongly certain answer for $\mathbf{Q}(A)$.

Case $pat(G) = Y \in \mathcal{Y}$. h is a linearly certain answer for $\mathbf{Q}(A) \Rightarrow \Sigma^* \subseteq L(A)$. Given that A is a complete Dfa, we then have $acc_\delta(\{q_0\}) \subseteq F \Rightarrow safe\text{-}acc_\delta(F) \cap \{q_0\} \neq \emptyset \Rightarrow G$ is a *safe-acc* strongly certain answer for $\mathbf{Q}(A)$.

Case $pat(G) = p_1 p_2$ where p_1 and p_2 are non-empty patterns of some smaller hyperstreams $G_1, G_2 \in \mathcal{H}_\Sigma$. G is a linearly certain answer for $\mathbf{Q}(A) \Rightarrow \{uv \mid u \in Inst(lin(p_1)), v \in Inst(lin(p_2))\} \subseteq L(A)$. So there exists $F' \subseteq Q$ such that h_1 is a linearly certain answer for $\mathbf{Q}(A[F/F'])$ and for all $q' \in F'$, h_2 is a linearly certain answer for $\mathbf{Q}(A[I/\{q'\}])$. Note that since A is a Dfa, so $A[F/F']$ and $\mathbf{Q}(A[I/\{q'\}])$ for all $q' \in F'$ are also Dfas. By the inductive hypothesis, we then have that G_1 (resp. G_2) is a *safe-acc* strongly certain answer for $A[F/F']$ (resp. $\mathbf{Q}(A[I/\{q'\}])$ for all $q' \in F'$). This means that $e_{safe\text{-}acc,\delta}(p_1)(F') \cap \{q_0\} \neq \emptyset$ and for all $q' \in F'$, $e_{safe\text{-}acc,\delta}(p_2)(F) \cap \{q'\} \neq \emptyset$. So $e_{safe\text{-}acc,\delta}(pat(h))(F) \cap \{q_0\} \neq \emptyset \Leftrightarrow h$ is a *safe-acc* strongly certain answer for A.

We can thus conclude that h is a *safe-acc* strongly certain answer for the Dfa A if and only if h is a linearly certain answer for $\mathbf{Q}(A)$. □

Tractable Cases. We show that *safe-acc* strong certainty is tractable for general Dfas queries on hyperstreams and for Nfa queries on compression-free hyperstreams.

Corollary 2. *Whether a hyperstream is a safe-acc strongly certain query answer for a Dfa $A = (Q, \Sigma, \delta, I, F)$ can be decided in PTime.*

Proof. Proposition 5 shows that deciding whether a hyperstream is a *safe-acc* strongly certain answer for a complete Dfa A is equivalent to deciding whether it is a linearly certain answer for $\mathbf{Q}(A)$, which can be decided in PTime by Proposition 3 since A is a Dfa. □

Proposition 6. *Whether a compression-free hyperstream G, i.e., a string pattern, is a safe-acc strongly certain query answer for an Nfa $A = (Q, \Sigma, \delta, I, F)$ can be decided in time $O(|G||A|)$.*

Proof. Starting with the set of final states, we have to apply $O(|G|)$ operations of $itrans_\delta(a)$ or $safe\text{-}acc_\delta$, each of which can be performed in time $O(|A|)$ by Lemma 2. □

Unfortunately, we have not been able to determine the precise complexity of *safe-acc* strong certainty for general hyperstreams with compression.

Incompleteness for NFAs. The notion of *safe-acc* strong certainty is very satisfactory for Dfa queries and also for path queries on streams (see Section 8). However, it remains unsatisfactory for path queries on hyperstreams, as we will illustrate next by example.

Example 5. We reconsider the Nfa $A = (Q, \Sigma, \delta, I, F)$ in Fig. 6 which defines the path query from the introduction. This Nfa is completed with the sink state q_{sink} and the transitions made to it, all colored in gray. In order to test whether the hyperstream G in Fig. 1 is a *safe-acc* strongly certain answer for this Nfa, we have to compute $e_{safe\text{-}acc,\delta}(pat(G))$ as shown in Fig. 7. The result is $\{q_4, q_5\}$ which does not contain the initial state, so the hyperstream G is not a *safe-acc* strongly certain query answer for the Nfa. The

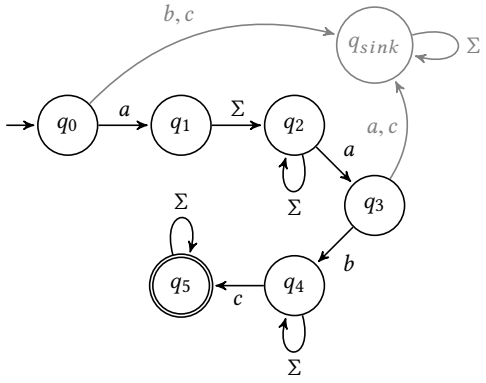

Figure 6: The completed Nfa of the Boolean path query $a(s(s^*(a(b(s^*(c(true)))))))$ with alphabet $\Sigma = \{a, b, c\}$ obtained by the compilation.

$$
\begin{aligned}
& e_{safe\text{-}acc,\delta}(pat(G))(F) \\
&= e_{safe\text{-}acc,\delta}(aYcZabbYaYcZa)(F) \\
&= e_{safe\text{-}acc,\delta}(aYcZabbYaYcZ)(\underbrace{itrans_\delta(a)(\{q_5\})}_{=\{q_5\}}) \\
&= e_{safe\text{-}acc,\delta}(aYcZabbYaYc)(\underbrace{safe\text{-}acc_\delta(\{q_5\})}_{=\{q_5\}}) \\
&= e_{safe\text{-}acc,\delta}(aYcZabbYaY)(\underbrace{itrans_\delta(c)(\{q_5\})}_{=\{q_4,q_5\}}) \\
&= e_{safe\text{-}acc,\delta}(aYcZabbYa)(\underbrace{safe\text{-}acc_\delta(\{q_4,q_5\})}_{=\{q_4,q_5\}}) \\
&= e_{safe\text{-}acc,\delta}(aYcZabbY)(\underbrace{itrans_\delta(a)(\{q_4,q_5\})}_{=\{q_4,q_5\}})) \\
&= e_{safe\text{-}acc,\delta}(aYcZabb)(\underbrace{safe\text{-}acc_\delta(\{q_4,q_5\})}_{=\{q_4,q_5\}}) \\
&= e_{safe\text{-}acc,\delta}(aYcZ)(\underbrace{itrans_\delta(abb)(\{q_4,q_5\})}_{=\{q_2,q_4,q_5\}})) \\
&= e_{safe\text{-}acc,\delta}(aYc)(\underbrace{safe\text{-}acc_\delta(\{q_2,q_4,q_5\})}_{=\{q_4,q_5\}}) \\
&= e_{safe\text{-}acc,\delta}(aY)(\underbrace{itrans_\delta(c)(\{q_4,q_5\})}_{=\{q_4,q_5\}})) \\
&= e_{safe\text{-}acc,\delta}(a)(\underbrace{safe\text{-}acc_\delta(\{q_4,q_5\})}_{=\{q_4,q_5\}}) \\
&= itrans_\delta(a)(\{q_4,q_5\}) \\
&= \{q_4,q_5\}
\end{aligned}
$$

Figure 7: The computation of $e_{safe\text{-}acc,\delta}(pat(G))$ for the hyperstream G in Fig. 1 and the Nfa $A = (Q, \Sigma, \delta, I, F)$ in Fig. 6.

problem is the step where we compute:

$$\textit{safe-acc}_\delta(\{q_2, q_4, q_5\}) = \{q_4, q_5\}$$

Indeed, q_2 can also safely go to $\{q_2, q_4, q_5\}$ for any word, but unfortunately, q_3 is also accessible from q_2, So the safety approximation $\textit{safe-acc}_\delta$ is too rude to capture this natural case and $\textit{safe-acc}_\delta$ strong certainty fails for this certain answer.

6.4 Strong Certainty

Our next objective is to improve the safety approximation, so that we can deal with the example from the introduction.

Let $\delta \subseteq Q \times \Sigma \times Q$ be a transition relation. We say that a state $q \in Q$ has a Σ-*loop* in δ if $(q, a, q) \in \delta$ for all letters q. Define the transition relation $\textit{loop}(\delta)$ by:

$$\textit{loop}(\delta) = \delta \setminus \{(q, a, q') \in \delta \mid q \neq q', q \text{ has a } \Sigma\text{-loop in } \delta\}.$$

For any NFA $A = (Q, \Sigma, \delta, I, F)$, let $\textit{safe-acc-loop}_\delta = \textit{safe-acc}_{\textit{loop}(\delta)}$.

Lemma 6. *For any complete NFA $A = (Q, \Sigma, \delta, I, F)$, $\textit{safe-acc-loop}_\delta$ is a safety approximation.*

Proof. If δ is the transition relation of some complete NFA then the transitions relation $\textit{loop}(\delta)$ is complete, since only outgoing edges of states with Σ-loops are removed from δ (but the Σ-loops are not deleted). Thus $\textit{trans}_\delta(w)(\{q\}) \neq \emptyset$. Furthermore $\textit{trans}_\delta(w)(\{q\}) \subseteq Q'$ since $q \in \textit{safe-acc-loop}_\delta$. Hence $\textit{trans}_\delta(w)(\{q\}) \cap Q' \neq \emptyset$. So $q \in \textit{safe}_\delta(Q')$. □

Definition 8 (strongly certain query answer). *We call a hyperstream a* strongly certain answer *for $A = (Q, \Sigma, \delta, I, F)$ if it is a safe-acc-loop strongly certain answer for A.*

Since $\textit{acc}_{\textit{loop}(\delta)} \subseteq \textit{acc}_\delta$ it follows that $\textit{safe-acc}_\delta \subseteq \textit{safe-acc}_{\textit{loop}(\delta)}$ and thus any *safe-acc* strongly certain answer is strongly certain.

Example 6. We reconsider Example 5 where it was shown that the hyperstream G from Fig. 1 is not a *safe-acc* strongly certain answer for the NFA $A = (Q, \Sigma, \delta, I, F)$ in Fig. 6. We now show that G is a strongly certain answer for A nevertheless. In the first critical step, we now have:

$$\textit{safe-acc}_{\textit{loop}(\delta)}(\{q_2, q_4, q_5\}) = \{q_2, q_4, q_5\}$$

since q_3 is no more accessible from q_2 in $\textit{loop}(\delta)$ while it was in δ. A second critical step follows:

$$\textit{safe-acc}_{\textit{loop}(\delta)}(\{q_1, q_2, q_4, q_5\}) = \{q_1, q_2, q_4, q_5\}$$

where we would lose the states q_1 and q_2 with $\textit{safe-acc}_\delta$ instead of $\textit{safe-acc}_{\textit{loop}(\delta)}$. Overall we have:

$$\begin{aligned}
e_{\textit{safe-acc-loop},\delta}(\textit{pat}(G))(F) &= \ldots \\
&= e_{\textit{safe-acc-loop},\delta}(aYc)(\underbrace{\textit{safe-acc}_{\textit{loop}(\delta)}(\{q_2, q_4, q_5\})}_{=\{q_2, q_4, q_5\}}) \\
&= e_{\textit{safe-acc-loop},\delta}(aY)(\underbrace{\textit{itrans}_\delta(c)(\{q_2, q_4, q_5\})}_{=\{q_1, q_2, q_4, q_5\}}) \\
&= e_{\textit{safe-acc-loop},\delta}(a)(\underbrace{\textit{safe-acc}_{\textit{loop}(\delta)}(\{q_1, q_2, q_4, q_5\})}_{=\{q_1, q_2, q_4, q_5\}}) \\
&= \textit{itrans}_\delta(a)(\{q_1, q_2, q_4, q_5\}) \\
&= \{q_0, q_1, q_2, q_4, q_5\}
\end{aligned}$$

Since $q_0 \in I$, this shows that G is indeed a strongly certain answer for A.

We next show that deciding whether a compression-free hyperstream is a strongly certain answer for an NFA is tractable. This is in sharp contrast to the notion of being a linearly certain answer, which by Theorem 3 is PSPACE-hard even without compression.

Proposition 7. *For any compression-free hyperstream G and any NFA $A = (Q, \Sigma, \delta, I, F)$, deciding whether G is a strongly certain answer for A can be done in time $O(|G||A|)$.*

Proof. By Lemma 2, any set $\textit{safe-acc-loop}_\delta(Q')$ can be computed in time $O(|A|)$. It is then suffices to compute $e_{\textit{safe-acc-loop},\delta}(\textit{pat}(G))(F)$ in a way similar to $e_{\textit{safe-acc},\delta}(\textit{pat}(G))(F)$ in the proof of Proposition 6, and then to intersect it with the set of initial states I. The overall time is still in $O(|G||A|)$. □

7 PRUNING STRONGLY CERTAIN ANSWERS

So far we have identified tractable approximations of certain query (non-)answering for all cases, except for deciding whether a hyperstream with compression is a certain answer of a query defined by an NFA. We propose now a further approximation with which tractability is achieved. Rather than ruling out compression or nondeterminism, we propose to approximate strong certainty in the general case. The idea is to convert hyperstreams with compression into hyperstreams without. But of course, we cannot uncompress hyperstreams, since this may lead to an exponential size explosion. Instead, we will prune shared parts of the hyperstream subject to compression by replacing them by completely unknown fresh streams. We then test strongly certainty on the resulting pruned hyperstream.

Definition 9. *A* pruning function *p for hyperstreams is a function that maps all hyperstreams $G = (N, \Sigma, R, S)$ to subsets of positions of their non-terminals. That is, $p(G) \subseteq \textit{Pos}_N(G)$. We denote by $\mathcal{P}_\Sigma$ the set of all such pruning functions.*

Let $G = (N, \Sigma, R, S)$ be a hyperstream. We next define how to prune G with respect to a given pruning function $p \in \mathcal{P}_\Sigma$. The idea is to replace with fresh free variables all the occurrences of non-terminals in G at positions that are not in $p(G)$. For this we first introduce some notations. Given a subset $L \subseteq \Sigma \cup \mathcal{Y}$, we denote by $\textit{Pos}_L(G) = \textit{letter}_G^{-1}(L)$ the set of positions of G with letters in L. Then we fix a generator of new variables $\text{nvar}_G : \textit{Pos}_N(G) \to \mathcal{Y}$ which is an injection such that $\text{nvar}_h(v) \notin N$ for all $v \in \textit{Pos}_N$. Then for any bound variable $Y \in \textit{dom}(R)$, we define a substitution $\textit{prune}_{p,Y} : \textit{pos}(R(Y)) \to \mathcal{Y}$ such that:

$$\textit{prune}_{p,Y}(j) = \begin{cases} \text{nvar}_G(Y, j) & \text{if } (Y, j) \notin p(G) \\ \textit{letter}_G(Y, j) & \text{otherwise} \end{cases}$$

Thereafter we introduce the function $\textit{prune}_p : \mathcal{H}_\Sigma \to \mathcal{H}_\Sigma$ such that $\textit{prune}_p(G)$ is the hyperstream (N_p, Σ, R_p, S) where

$$N_p = N \cup \{\text{nvar}_G(v) \mid v \in \textit{Pos}_N(G) \setminus p(G)\},$$

$$\textit{dom}(R_p) = \textit{dom}(R) \text{ and for all } Y \in \textit{dom}(R_p) \text{ and for all } j \in \textit{pos}(R(Y)),$$

$$R_p(Y)[j] = \textit{prune}_{p,Y}(j).$$

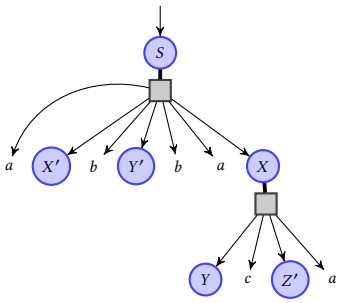

Figure 8: Pruning of the Hyperstream G of Fig. 1

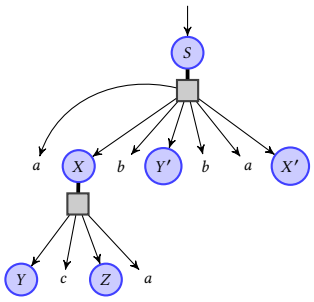

Figure 9: *fo*-pruning of the Hyperstream G of Fig. 1

Example 7. Fig. 8 shows a pruning of the hyperstream G illustrated in Fig. 1. This is done with the pruning function p which is such that $p(G) = \{(S, 7), (X, 1)\}$. In addition to the start symbol S, the occurrence of X (respectively Y) that is at the second position of $R(S)$ (respectively first position of $R(X)$) is left unchanged, where R is the ruling function of G. Contrariwise, the remaining occurrence of X at position $(S, 2)$, the other occurrence of Y at position $(S, 4)$ and the only occurrence of Z are replaced by their images by nvar_G, where $\text{nvar}_G(S, 2) = X'$, $\text{nvar}_G(S, 4) = Y'$ and $\text{nvar}_G(X, 3) = Z'$.

Notice that the pruned hyperstream $prune_p(G)$ has necessarily less data than G, that is $|pat(prune_p(G))| \leq |pat(G)|$ for all $p \in \mathcal{P}_\Sigma$. Thereafter, if p is well chosen, then strong certainty of G_p may be decided efficiently, in contrast to G, as we'll see further.

Definition 10. *Let A be a query NFA over Σ and $p \in \mathcal{P}_\Sigma$ a pruning function. We call the hyperstream h a* strongly certain query answer *by p for A if $prune_p(h)$ is a strongly certain query answer for A.*

We next introduce the pruning function *fo* as an example of a pruning function that ensures tractability of pruning strongly certainty. For any hyperstream G, *fo*-pruning will prune out all but the first occurrences (in the order given by $pat(G)$) of all non-terminals of G.

For any bound variable $Y \in dom(R)$, let its set of addresses be defined in the style of the Dewey notation by:

$$Addr_G(Y) = pos_{\Sigma \cup V(G)}(R(Y)) \cup subAdd_G(Y)$$

where $subAdd_G(Y) = \{i\alpha \mid Y' = letter_G(Y, i) \in dom(R) \text{ and } \alpha \in Addr_G(Y')\}$. We write $Addr(G)$ for $Addr_G(S)$. Note that to any address $\alpha \in Addr(G)$ corresponds a position $pos_G(\alpha)$ of the hyperstream G, defined by:

$$pos_G(\alpha) = \begin{cases} (S, i) & \text{if } \alpha = i \in \mathbb{N} \\ (letter_G(pos_G(\alpha')), i) & \text{if } \alpha = \alpha' i, \alpha' \in Addr(G), i \in \mathbb{N} \end{cases}$$

The letters selector can be naturally extended to addresses, defining $letter_G(\alpha) = letter_G(pos_h(\alpha))$. Then the pruning function *fo* is defined by: for any hyperstream G and any position $\nu \in Pos_N(G)$:

$\nu \in fo(G)$ iff $\nu = pos_G(\min(\{\alpha \in Addr(G) \mid letter_G(\alpha) = letter_G(\nu)\}))$

for a set of addresses U, $\min(U)$ stands for smallest element of U w.r.t. the lexicographic order on addresses, that is, the lexicographic order on words over the set of natural numbers. In other words, for all non-terminal Y of G, $fo(G)$ selects exactly one position ν s.t. $letter_G(\nu) = Y$. Additionally, this position is such that $\nu = pos_G(\alpha)$ where α is the smallest address among all addresses α' with $letter_G(\alpha') = Y$.

Example 8. Let A' be the NFA of the query $P' = a(s(s^*(ab(true))))$, obtained from the automaton in Fig. 6 by removing the state q_5, deleting all the transitions going to it and making the state q_4 final. Thus P' accepts all words that start with letter a, succeeded by some letter, then followed eventually by a factor ab. The hyperstream $prune_{fo}(G)$, illustrated in Fig. 9, is a strongly certain answer for A'. Since, $pat(prune_{fo}(G)) = aYcZabY'baX'$, we have

$$e_{safe\text{-}acc\text{-}loop,\delta}(pat(prune_{fo}(G)))(\{q_4\}) = \{q_0, q_1, q_2, q_4\}$$

which contains the initial state q_0.

Lemma 7. *A hyperstream $G \in \mathcal{H}_\Sigma$ is compression-free if and only if $h = prune_{fo}(h)$.*

Proof. The proof is rather straightforward. For the forward direction, assume G is compression-free, so $h = prune_{fo}(h)$, since for any non-terminal Y there is a unique position in $Addr(G)$ mapping to it, and this position is trivially the first among all the other positions mapping to Y. For the inverse direction, assume $G = prune_{fo}(G)$. By definition, applying $prune_{fo}$ to a hyperstream always yield a compression-free hyperstream, since the newly produced has at most one occurrence of all its non-terminals. And since $prune_{fo}(G) = G$, then G is compression-free. □

Example 9. Let $G = (N, \Sigma, R, S)$ where $N = \{S, Y_1, Y_2, Y_3, Y_4\}$, $\Sigma = \{a, b\}$, $dom(R) = \{S, Y_1, Y_2, Y_3\}$, $R(S) = Y_1Y_1$, $R(Y_1) = Y_2Y_3$, $R(Y_2) = bY_4$ and $R(Y_3) = a$. The compression-free version of G is the hyperstream with ruling function R_p where $R_p(S) = Y_1Y_1'$, $Y_1' = \text{nvar}_G(Y_1)$ and $R_p(y_i) = R(Y_i)$ for $2 \leq i \leq 3$.

Proposition 8. *Let A be an NFA and $G \in \mathcal{H}_\Sigma$ be a hyperstream. Deciding whether G is strongly certain answer by fo for A can be done in time $O(|G||A|)$.*

Proof. Let $G_p = prune_p(G)$. Computing G_p can be done linearly in the size of G. Furthermore, G_p is compression-free by Lemma 7, and therefore deciding strong certainty for G_p is in $O(|G_p||A|)$. Since $|G_p| \leq |G|$, the time complexity for deciding that G is a strongly certain query answer by *fo* for A is then also in $O(|G||A|)$. □

8 PATH QUERIES

We next show for path queries on words, that are closely motivated by forward XPath filters for Xml documents, that *safe-acc* strongly certain answers modulo compilation to Nfas yields the same approximation of certain answers on streams then a compilation to early Nfas as proposed in [3].

We consider a language of boolean path queries for words over Σ that has the following abstract syntax where $a \in \Sigma$:

$$P := true \mid s(P) \mid s^*(P) \mid a(P) \mid P \wedge P \mid P \vee P' \mid \neg P$$

We impose the restriction that any path query $\neg P$ does neither contain the operators s^* nor $\vee$, since as we'll see in the following, the construction of the automata for this queries introduce nondeterminism. Alternatively, we could also rely on determinization of Nfas when compiling path queries $\neg P$ to Nfas, which one might want to avoid for complexity and practical reasons.

The path query from the introduction $a(s(s^*(ab(s^*(c)))))$ can be written in this syntax with few syntactic sugar as

$$a(s(s^*(a(b(s^*(c(true))))))).$$

Semantically, any path query P can be seen as a regular expression and thus defines a language of strings $[\![P]\!] \subseteq \Sigma^*$.

This language can be computed as follows by induction on the structure of path queries:

$$\begin{array}{ll} [\![true]\!] = \Sigma^* & \\ [\![s(P)]\!] = \Sigma[\![P]\!] & [\![P \wedge P']\!] = [\![P]\!] \cap [\![P']\!] \\ [\![s^*(P)]\!] = \Sigma^*[\![P]\!] & [\![P \vee P']\!] = [\![P]\!] \cup [\![P']\!] \\ [\![a(P)]\!] = a[\![P]\!] & [\![\neg P]\!] = \Sigma^* \setminus [\![P]\!] \end{array}$$

Definition 11. *An* Nfa with selection states *or an eNfa is a pair $E = (A, S)$ where $A = (Q, \Sigma, \delta, I, F)$, is an Nfa and $S \subseteq F$ a subset of final states of A that is an attractor, whose elements are called selection states, so that $L(A[I/\{q\}]) = \Sigma^*$ for all selection states $q \in S$.*

Given an eNfa $E = (A, S)$, any stream wy that can be evaluated by A into a selection state on S, that is $trans_\delta(w) \cap (I \times S) \neq \emptyset$, is a certain answer by the query defined by A. In this sense, the selection states of eNfas permit to approximate the set of streams that are certain answers of Nfa queries. Lifting this approximation to hyperstreams was one of the motivations of the present paper.

We will next compile path expressions P to eNfas $E_P = (A_P, S_P)$ such that $L(A_P) = [\![P]\!]$. We will name the components of the Nfa such that $A_P = (Q_P, \Sigma, \delta_P, I_P, F_P)$. The compiler is defined by induction on the structure of the path query P, such that for all path queries P' and P'' and all letters $a \in \Sigma$:

E_{true} has a single state q that is initial, final, and has transitions $q \xrightarrow{a} q$ for all $a \in \Sigma$. The state q is also a selection state. Formally, $Q_{true} = I_{true} = F_{true} = S_{true} = \{q\}$ and:

$$\delta_{true} = \{(q, a, q) \mid a \in \Sigma\}.$$

$E_{s(P')}$ is obtained from $E_{P'}$ by adding a new state q that becomes the unique initial state, and transitions $q \xrightarrow{a} q'$ for all $a \in \Sigma$ and initial states $q' \in I_{P'}$. The selection states are the same as for $E_{P'}$. More formally, for $P = s(P')$ we define $Q_P = Q_{P'} \uplus \{q\}$, $I_P = \{q\}$, $F_P = F_{P'}$, $S_P = S_{P'}$ and:

$$\delta_P = \delta_{P'} \cup \{(q, a, q') \mid a \in \Sigma \text{ and } q' \in I_{P'}\}.$$

$E_{s^*(P')}$ is obtained from $E_{P'}$ by adding a new state q that becomes the unique initial state, for all $a \in \Sigma$ adding transitions $q \xrightarrow{a} q$, and a transition $q \xrightarrow{a} q''$ for all transitions $q' \xrightarrow{a} q''$ of $A_{P'}$ that start in an initial state $q' \in I_{P'}$. The selection states are the same as for $E_{P'}$. Formally, let $P = s^*(P')$, $Q_P = Q_{P'} \uplus \{q\}$, $I_P = \{q\}$, $F_P = F_{P'}$, $S_P = S_{P'}$ and:

$$\begin{array}{rcl} \delta_P & = & \delta_{P'} \cup \{(q, a, q) \mid a \in \Sigma\} \\ & \cup & \{(q, a, q'') \mid a \in \Sigma,\ q' \in I_{P'}, (q', a, q'') \in \delta_{P'}\}. \end{array}$$

$E_{a(P')}$ is obtained from $A_{P'}$ by adding a new state q that is the unique initial state, and transitions $q \xrightarrow{a} q'$ for all initial states q' of $A_{P'}$. The selection states are the same as for $E_{P'}$. Formally, let $P = a(P')$, $Q_P = Q_{P'} \uplus \{q\}$, $I_P = \{q\}$, $F_P = F_{P'}$, $S_P = S_{P'}$ and:

$$\delta_P = \delta_{P'} \cup \{(q, a, q') \mid q' \in I_{P'}\}.$$

$E_{P' \wedge P''}$ is the product of sets of states of $A_{P'}$ and $A_{P''}$, which accepts in pairs of final states of $A_{P'}$ and $A_{P''}$. The selection states are the pairs of $S_{P'}$ and $S_{P''}$. Formally, for $P = P' \wedge P''$ we define $Q_P = Q_{P'} \times Q_{P''}$, $I_P = I_{P'} \times I_{P''}$, $F_P = F_{P'} \times F_{P''}$, $S_P = S_{P'} \times S_{P''}$ and:

$$\begin{array}{l} \delta_P = \{((q_1, q_2), a, (q_3, q_4)) \mid \\ \qquad a \in \Sigma, (q_1, a, q_3) \in \delta_{P'} \text{ and } (q_2, a, q_4) \in \delta_{P''}\}. \end{array}$$

$E_{P' \vee P''}$ is made with the union of sets of states of $A_{P'}$ and $A_{P''}$. The set of selection states are the union of $S_{P'}$ and $S_{P''}$. Formally, with $P = P' \vee P''$ let $Q_P = Q_{P'} \cup Q_{P''}$, $I_P = I_{P'} \cup I_{P''}$, $F_P = F_{P'} \cup F_{P''}$, $S_P = S_{P'} \cup S_{P''}$ and $\delta_P = \delta_{P'} \cup \delta_{P''}$.

$E_{\neg P'}$ is obtained from $A_{P'}$ by first adding a sink state q_{sink} and then flipping the final states. The only selection state is the newly added sink. This construction is correct since $A_{P'}$ is deterministic, given that it does neither contain disjunctions $\vee$ nor recursive steps s^* by definition of path queries. Formally, for $P = \neg P'$ we define $Q_P = Q_{P'} \uplus \{q_{sink}\}$, $I_P = I_{P'}$, $F_P = Q_P \setminus F_{P'}$, $S_P = \{q_{sink}\}$ and:

$$\begin{array}{l} \delta_P = \delta_{P'} \cup \{(q, a, q_{sink}) \mid q \in Q_{P'}, a \in \Sigma \\ \qquad \text{and } \nexists q' \in Q_{P'}.(q, a, q') \in \delta_{P'}\}. \end{array}$$

Figure 6 shows the early Nfa obtained by compilation from the path query from the introduction (up to the removal of two unaccessible states). Its only selection state is the final state q_5.

It can be shown in analogy to [3] that the compiler can be executed with only polynomial time in the size of the path query, if the number of conjunctions in the path query is bounded. This is the reason why we do not add more general regular expressions to our language of path queries, such as PP' or P^*. For implementing the construction steps for all operators in constant time, one has to maintain the completeness of the intermediate Nfas in order to avoid the addition of sink states. Furthermore, for proof one needs an invariant stating that the number of outgoing edges from initial states is bounded if the number of conjunctions is bounded.

Lemma 8. *Let P, P' be path queries such that $P = s(P')$ or $P = s^*(P')$, $w \in \Sigma^*$ be a string and $Y \in \mathcal{Y}$ be a variable. So if wY is a safe-acc strongly certain query answer for $A_{P'}$ then wY is a safe-acc strongly certain query answer for $A_P[I_P/I_{P'}]$.*

Proof. Let $P \in \{s(P'), s^*(P')\}$ and q be the newly added state to $A_{P'}$ to build A_P. We have

$$\delta_P = \begin{cases} \delta_{P'} \cup \{(q,a,q') \mid a \in \Sigma \text{ and } q' \in I_{P'}\} & \text{if } P = s(P') \\ \delta_{P'} \cup \{(q,a,q) \mid a \in \Sigma\} \cup \{(q,a,q'') & \\ \mid a \in \Sigma,\ q' \in I_{P'}, (q',a,q'') \in \delta_{P'}\} & \text{if } P = s^*(P') \end{cases}$$

Clearly $A_P[I_P/I_{P'}] = A_{P'}$ in both cases. Thus, for all $w' \in \Sigma^*$, $trans_{\delta_P}(w')(I_{P'}) = trans_{\delta_{P'}}(w')(I_{P'})$. On the other hand side, w is a *safe-acc* strongly certain query answer for $A_{P'}$

$$\begin{array}{ll} \Leftrightarrow & e_{safe\text{-}acc,\delta_{P'}}(wY)(F_{P'}) \cap I_{P'} \neq \emptyset \\ \Leftrightarrow & itrans_{\delta_{P'}}(w)(safe\text{-}acc_{\delta_{P'}}(F_{P'})) \cap I_{P'} \neq \emptyset \\ \Leftrightarrow & trans_{\delta_{P'}}(w)(I_{P'}) \subseteq safe\text{-}acc_{\delta_{P'}}(F_{P'}) \\ \Leftrightarrow & trans_{\delta_P}(w)(I_{P'}) \subseteq safe\text{-}acc_{\delta_{P'}}(F_{P'}) \end{array}.$$

Furthermore, since $F_P = F_{P'}$, $safe\text{-}acc_{\delta_P}(F_P) = \{q' \in Q_P \mid acc_{\delta_P}(q') \subseteq F_P\} = safe\text{-}acc_{\delta_{P'}}(F_{P'}) \cup B$ where

$$B = \begin{cases} \{q\} & \text{if } acc_{\delta_P}(q) \subseteq F_P\} \\ \emptyset & \text{otherwise} \end{cases}$$

So the inclusion $safe\text{-}acc_{\delta_{P'}}(F_{P'}) \subseteq safe\text{-}acc_{\delta_P}(F_P)$ is verified.

We finally have

$$\begin{array}{ll} & trans_{\delta_P}(w)(I_{P'}) \subseteq safe\text{-}acc_{\delta_{P'}}(F_{P'}) \subseteq safe\text{-}acc_{\delta_P}(F_P) \\ \Leftrightarrow & itrans_{\delta_P}(w)(safe\text{-}acc_{\delta_P}(F_P)) \cap I_{P'} \neq \emptyset \\ \Leftrightarrow & e_{safe\text{-}acc,\delta_P}(wY)(F_P) \cap I_{P'} \neq \emptyset. \end{array}$$

Hence wY is a *safe-acc* strongly certain query answer for $A_P[I_P/I_{P'}]$. □

Theorem 10. *Let P be a boolean path query, $w \in \Sigma^*$ a string and $Y \in \mathcal{Y}$ a variable. Then wY is a safe-acc strongly certain query answer for A_P if the eNFA $E_P = (A_P, S_P)$ can evaluate w to its selection state in S_P.*

Proof. We show that for all path queries P and words $w \in \Sigma^*$ that if A_P can evaluate w to some selection state in S_P then the stream wY is a strongly certain answer of A_P. The proof is by induction on the structure of P. We need to consider the 7 different forms that a path query P may have:

Case $P = true$**:** We have $Q_P = I_P = F_P = \{q\}$ and $\delta_P = \{q\} \times \Sigma \times \{q\}$ and therefore $safe\text{-}acc_{\delta_P}(F_P) = \{q\} = Q_P$ and $itrans_{\delta_P}(w)(Q_P) = S_P$. Hence:

$$e_{safe\text{-}acc,\delta_P}(wY)(F_P) = itrans_{\delta_P}(w)(safe\text{-}acc_{\delta_P}(F_P)) = Q_P$$

We then have $e_{safe\text{-}acc,\delta_P}(wY)(F_P) \cap I_P = Q_P \neq \emptyset$, hence wY is a *safe-acc* strongly certain query answer for A_P.

Case $P = s(P')$**:** By assumption w can be evaluated by A_P to some selection state in $S_P = S_{P'}$. Since ϵ can only be evaluated to the newly added initial state $q \in I_P$ and $q \notin S_P$ it follows that $w \neq \epsilon$. Hence there exist $a \in \Sigma$ and $w' \in \Sigma^*$ such that $w = aw'$. Furthermore, since $trans_{\delta_P}(a) = \{q\} \times I_{P'}$ it follows that the suffix w' can be evaluated with $A_{P'}$ to some selection state of $S_{P'}$. By the inductive hypothesis, we have that $w'Y$ is a *safe-acc* strongly certain query answer for $A_{P'}$. According to Lemma 8, this implies that $w'y$ is a *safe-acc* strongly certain query answer for $A_P[I_P/I_{P'}]$, i.e. $K \cap I_{P'} \neq \emptyset$, where $K = e_{safe\text{-}acc,\delta_P}(w'Y)(F_P)$. We then have

$$e_{safe\text{-}acc,\delta_P}(wY)(F_P) = itrans_{\delta_P}(a)(K).$$

Given that $K \cap I_{P'} \neq \emptyset$, it follows that $itrans_{\delta_P}(a)(K) \cap I_P \neq \emptyset$ by the construction rules of A_P, so that wY is a *safe-acc* strongly certain query answer for A_P.

Case $P = s^*(P')$**:** By the construction rules of $E_{s^*(P')}$, $w = w'w''$ where $w', w'' \in \Sigma^*$ such that there are initial states of $A_{P'}$ among the states reached by A_P after having read w', i.e. $I_{P'} \subseteq \delta_P(w')(I_P)$. This way, $E_{P'}$ necessarily evaluates w'' to $S_{P'}$, since E_P evaluates w to S_P and $S_P = S_{P'}$ by the construction rules of $E_{s^*(P')}$. Furthermore, we have:

$$\begin{array}{l} e_{safe\text{-}acc,\delta_P}(wY)(F_{s^*(P')}) = \\ \quad itrans_{\delta_P}(w')(e_{safe\text{-}acc,\delta_P}(w''Y)(F_{s^*(P')})). \end{array}$$

By the inductive hypothesis, $w''Y$ is a *safe-acc* strongly certain query answer for $A_{P'}$. By Lemma 8, this implies that $w''Y$ is also a *safe-acc* strongly certain query answer for $A_P[I_P/I_{P'}]$, that is $K \cap I_{P'} \neq \emptyset$, where $K = e_{safe\text{-}acc,\delta_P}(w''Y)(F_P)$. Thus:

$$e_{safe\text{-}acc,\delta_P}(wY)(F_P) = itrans_{\delta_P}(w')(K)$$

and given that $K \cap I_{P'} \neq \emptyset$ and $I_{P'} \subseteq \delta_P(w')(I_P)$ as stated earlier, $itrans_{\delta_P}(w')(K) \cap I_P \neq \emptyset$, which means that wY is a *safe-acc* strongly certain query answer for A_P.

Case $P = aP'$**:** the proof for this part is almost the same than for $P = s(P')$, and we will thus not develop it.

Case $P = P' \wedge P''$**:** This implies that $w \in [\![P']\!] \cap [\![P'']\!]$, and that $E_{P'}$ evaluates w to $S_{P'}$ while $E_{P''}$ evaluates w to $S_{P''}$ by the construction rules of A_P. By the inductive hypothesis, we thus have that wY is a *safe-acc* strongly certain query answer for $A_{P'}$ and for $A_{P''}$, that is $e_{safe\text{-}acc,\delta_{P'}}(wY)(F_{P'}) \cap I_{P'} \neq \emptyset$ and $e_{safe\text{-}acc,\delta_{P''}}(wY)(F_{P''}) \cap I_{P''} \neq \emptyset$. Since $F_P = F_{P'} \times F_{P''}$ and $I_P = I_{P'} \times I_{P''}$ by the construction rules of A_P, we finally obtain $e_{safe\text{-}acc,\delta_P}(wY)(F_P) \cap I_P \neq \emptyset$, hence wY is a *safe-acc* strongly certain query answer for A_P.

Case $P = P' \vee P''$**:** This implies that $w \in [\![P']\!] \cup [\![P'']\!]$, and that $E_{P'}$ evaluates w to $S_{P'}$ or $E_{P''}$ evaluates w to $S_{P''}$, by the construction rules of $A_{P' \vee P''}$. By the inductive hypothesis, we thus have that wY is a *safe-acc* strongly certain query answer for $A_{P'}$ or for $A_{P''}$, that is $e_{safe\text{-}acc,\delta_{P'}}(wY)(F_{P'}) \cap I_{P'} \neq \emptyset$ or $e_{safe\text{-}acc,\delta_{P''}}(wY)(F_{P''}) \cap I_{P''} \neq \emptyset$. Since $F_P = F_{P''} \cup F_{P''}$ and $I_{P'} = I_{P'} \cup I_{P''}$ by the construction rules of $A_{P' \vee P''}$, we finally obtain $e_{safe\text{-}acc,\delta_P}(wY)(F_P) \cap I_{P' \vee P''} \neq \emptyset$, hence wY is a *safe-acc* strongly certain query answer for A_P.

Case $P = \neg P'$**:** By the construction rules of $E_{\neg P'}$, the E_P evaluates w to q_{sink} since the only selection state of E_P is the sink state added to $A_{P'}$ to make it complete. So $safe\text{-}acc_{\delta_P}(F_P) = q_{sink}$. Now,

$$\begin{array}{rl} e_{safe\text{-}acc,\delta_P}(wY)(F_P) & = itrans_{\delta_P}(w)(safe\text{-}acc_{\delta_P}(F_P)) \\ & = itrans_{\delta_P}(w)(q_{sink}) \end{array}$$

and we have that $itrans_{\delta_P}(w)(q_{sink}) \cap I_P \neq \emptyset$. Thus $e_\delta(wY)(F_P) \cap I_P \neq \emptyset$ hence wY is a *safe-acc* strongly certain query answer for A_P.

□

9 CONCLUSION

We studied the problem to find approximations of certain query (non-)answers for path queries on hyperstreams that can be computed efficiently.

For compression-free hyperstreams we proposed the notion of strong certainty for NFAs, which approximates the universal states of the NFA based on the accessibility relation of the automaton modulo self-loops. We conjecture that strong certainty yields an exact approximation for path queries without disjunction and negation. This was illustrated by Example 6 but still needs verification in the general case. Restricted to the case of streams, the conjecture may follow from Theorem 10 of the present paper and an adaptation of Theorem 2 of [14] from NWAs to NFAs.

For hyperstreams with compression, we do not know whether strong certainty can be decided in PTIME. Therefore, we proposed the notion of pruning strong certainty which removes compression by pruning the hyperstream before applying the notion of strong certainty. This yields a PTIME algorithm that is promising in practice, but at the cost of a coarser approximation.

The next natural steps in this line of research are first, to lift the notion of pruning strong certainty to path queries on nested words, and thereby to path queries on trees or graphs. Second, we would like to develop efficient algorithms that test incrementally evolving hyperstreams for pruning strong certainty dynamically. One question here is which information to maintain to obtain high time efficiency without spoiling neither the space efficiency nor the latency.

REFERENCES

[1] Rajeev Alur and P. Madhusudan. 2009. Adding nesting structure to words. *J. ACM* 56, 3 (2009), 1–43. http://doi.acm.org/10.1145/1516512.1516518

[2] L. Babai and E. Szemeredi. 1984. On The Complexity Of Matrix Group Problems I. In *Proceedings of the 25th Annual Symposium onFoundations of Computer Science, 1984 (SFCS '84)*. IEEE Computer Society, Washington, DC, USA, 229–240. https://doi.org/10.1109/SFCS.1984.715919

[3] Denis Debarbieux, Olivier Gauwin, Joachim Niehren, Tom Sebastian, and Mohamed Zergaoui. 2015. Early nested word automata for XPath query answering on XML streams. *Theor. Comput. Sci.* 578 (2015), 100–125. https://doi.org/10.1016/j.tcs.2015.01.017

[4] Adria Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. 2008. 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*. IEEE Computer Society, 93–102. https://doi.org/10.1109/LICS.2008.17

[5] Olivier Gauwin. 2009. *Streaming Tree Automata and XPath.* Ph.D. Dissertation. Université Lille 1. http://tel.archives-ouvertes.fr/tel-00421911/

[6] Olivier Gauwin and Joachim Niehren. 2011. Streamable Fragments of Forward XPath. In *16th International Conference on Implementation and Application of Automata*. Blois, France, 3–15. https://hal.inria.fr/inria-00442250 Long version: http://www.grappa.univ-lille3.fr/ niehren/Papers/streamability/0.pdf.

[7] Olivier Gauwin, Joachim Niehren, and Sophie Tison. 2009. Bounded Delay and Concurrency for Earliest Query Answering. In *3rd International Conference on Language and Automata Theory and Applications (Lecture Notes in Computer Science)*, Adrian Horia Dediu, Armand Mihai Ionescu, and Carlos Martin-Vide (Eds.), Vol. 5457. Springer, Tarragona, Spain, 350–361. https://doi.org/10.1007/978-3-642-00982-2

[8] John Grant. 1977. Null Values in a Relational Data Base. *Inf. Process. Lett.* 6, 5 (1977), 156–157. https://doi.org/10.1016/0020-0190(77)90013-8

[9] Todd J. Green, Ashish Gupta, Gerome Miklau, Makoto Onizuka, and Dan Suciu. 2004. Processing XML streams with deterministic automata and stream indexes. *ACM Trans. Database Syst.* 29, 4 (Dec. 2004), 752–788. https://doi.org/10.1145/1042046.1042051

[10] Michael Kay. 2010. A Streaming XSLT Processor. In *Balisage: The Markup Conference 2010. Balisage Series on Markup Technologies*, Vol. 5.

[11] Dexter Kozen. 1977. Lower Bounds for Natural Proof Systems. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*. IEEE Computer Society, 254–266. https://doi.org/10.1109/SFCS.1977.16

[12] Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. 2007. Visibly pushdown automata for streaming XML. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy (Eds.). ACM, 1053–1062. https://doi.org/10.1145/1242572.1242714

[13] Pavel Labath and Joachim Niehren. 2013. *A Functional Language for Hyperstreaming XSLT.* Technical Report. INRIA Lille.

[14] Anthony Lick and Joachim Niehren. 2013. *Early = Earliest?* Technical Report. INRIA Lille. http://hal.inria.fr/hal-00873742.

[15] Sebastian Maneth, Alberto Ordóñez Pereira, and Helmut Seidl. 2015. Transforming XML Streams with References. In *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings (Lecture Notes in Computer Science)*, Costas S. Iliopoulos, Simon J. Puglisi, and Emine Yilmaz (Eds.), Vol. 9309. Springer, 33–45. https://doi.org/10.1007/978-3-319-23826-5_4

[16] Barzan Mozafari, Kai Zeng, and Carlo Zaniolo. 2012. High-performance complex event processing over XML streams. In *SIGMOD Conference*, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, Ariel Fuxman, and Candan (Eds.). ACM, 253–264. https://doi.org/10.1145/2213836.2213866

[17] Dan Olteanu. 2007. SPEX: Streamed and Progressive Evaluation of XPath. *IEEE Trans. on Know. Data Eng.* 19, 7 (2007), 934–949. https://doi.org/10.1109/TKDE.2007.1063

[18] Wojciech Plandowski. 1995. *The complexity of the morphism equivalence problem for context-free languages.* Ph.D. Dissertation. Warsaw University. Department of Informatics, Mathematics, and Mechanics.

[19] Momar Sakho, Iovka Boneva, and Joachim Niehren. 2017. Complexity of Certain Query Answering on Hyperstreams. In *BDA 2017 - 33ème conférence sur la " Gestion de Données - Principes, Technologies et Applications "*. Nancy, France. https://hal.archives-ouvertes.fr/hal-01609498

[20] Michael Schmidt, Stefanie Scherzinger, and Christoph Koch. 2007. Combined Static and Dynamic Analysis for Effective Buffer Minimization in Streaming XQuery Evaluation.. In *23rd IEEE International Conference on Data Engineering*. 236–245.

[21] Tom Sebastian. 2016. *Evaluation of XPath Queries on XML Streams with Networks of Early Nested Word Automata.* Theses. Universite Lille 1. https://hal.inria.fr/tel-01342511